

Microprocessors and Microcontrollers

**Architecture, Programming
and Interfacing using
8085, 8086, 8051**

ABOUT THE AUTHOR

Soumitra Kumar Mandal holds a BE (Electrical Engineering) from Bengal Engineering College, Shibpur, Calcutta University, and an MTech (Electrical Engineering) with specialization in Power Electronics from the Institute of Technology, Banaras Hindu University, Varanasi. He obtained a PhD from Panjab University, Chandigarh. Prof. Mandal started his career as a lecturer of Electrical Engineering at SSGM College of Engineering, Shegaon. Thereafter, he joined as a lecturer at Panjab Engineering College, Chandigarh, and served there from March 1999 to January 2004. Then he joined National Institute of Technical Teachers' Training and Research, Kolkata, as Assistant Professor in Electrical Engineering in February 2004 and is presently working as Associate Professor.

Prof. Mandal is a life member of ISTE and a member of IE. Throughout his academic career, he has published about twenty-five research papers in national and international journals and presented many papers in national and international conferences. His research interests are in computer-controlled drives, microprocessor- and microcontroller-based system design, embedded system design and neuro-fuzzy computing.

The author can be contacted at his email id: *mandal_soumitra@yahoo.com*.

Microprocessors and Microcontrollers

Architecture, Programming
and Interfacing using
8085, 8086, 8051

Soumitra Kumar Mandal

Associate Professor

Department of Electrical Engineering

National Institute of Technical Teachers' Training and Research

Kolkata



Tata McGraw Hill Education Private Limited
NEW DELHI

McGraw-Hill Offices

New Delhi New York St Louis San Francisco Auckland Bogotá Caracas
Kuala Lumpur Lisbon London Madrid Mexico City Milan Montreal
San Juan Santiago Singapore Sydney Tokyo Toronto



Tata McGraw-Hill

Published by Tata McGraw Hill Education Private Limited,
7 West Patel Nagar, New Delhi 110 008

Microprocessors and Microcontrollers

Copyright © 2011, Tata McGraw Hill Publishing Company Limited No part of this publication can be reproduced or distributed in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise or stored in a database or retrieval system without the prior written permission of the publishers. The program listings (if any) may be entered, stored and executed in a computer system, but they may not be reproduced for publication.

This edition can be exported from India only by the publishers,
Tata McGraw Hill Education Private Limited

ISBN (13 digits) : 978-0-07-1329200

ISBN (10 digits) : 0-071-32920X

Vice President and Managing Director—McGraw-Hill Education—Asia Pacific Region: *Ajay Shukla*
Head—Higher Education Publishing and Marketing: *Vibha Mahajan*

Publishing Manager—(SEM & Tech. Ed.): *Shalini Jha*

Editorial Executive: *Smruti Snigdha*

Editorial Researcher: *Noaman Khan*

Executive—Editorial Services: *Sohini Mukherjee*

Sr Production Manager: *Satinder S Baveja*

Production Executive: *Anuj K Shrivastava*

Marketing Manager—Higher Education: *Vijay S Jagannathan*

Senior Product Specialist (SEM & Tech Ed.): *John Mathews*

General Manager—Production: *Rajender P Ghansela*

Assistant General Manager—Production: *B L Dogra*

Information contained in this work has been obtained by Tata McGraw-Hill, from sources believed to be reliable. However, neither Tata McGraw-Hill nor its authors guarantee the accuracy or completeness of any information published herein, and neither Tata McGraw-Hill nor its authors shall be responsible for any errors, omissions, or damages arising out of use of this information. This work is published with the understanding that Tata McGraw-Hill and its authors are supplying information but are not attempting to render engineering or other professional services. If such services are required, the assistance of an appropriate professional should be sought.

Typeset at Print-O-World, 2579, Mandir Lane, Shadipur, New Delhi 110 008, and printed at Adarsh Printers, C-50-51, Mohan Park, Naveen Shahdara, Delhi 110 032

Cover : A. P. OFFSET

RAXYYDAGDCQYA

The McGraw Hill Companies

Dedication

*To the memory of
My youngest son, Late Gajanan Mandal
And to
My parents, Smt. Arati Mandal and Shri Prokash Mandal
My wife, Malvika, and my children, Om and Puja*

Soumitra Kumar Mandal

CONTENTS

Preface

xiii

1. Introduction to Microprocessors and Microcontrollers	1
1.1 Introduction	1
1.2 Microprocessor	3
1.3 Microcomputer	4
1.4 Architecture of Microprocessors	6
1.5 History of Microprocessors	8
1.6 Evolution of Microprocessors	9
1.7 Microprocessor Applications	12
1.8 Evolution of Microcontrollers	13
1.9 Applications of Microcontrollers	15
<i>Summary</i>	<i>16</i>
<i>Multiple-Choice Questions</i>	<i>16</i>
<i>Short-Answer-Type Questions</i>	<i>17</i>
<i>Review Questions</i>	<i>17</i>
<i>Answers to Multiple-Choice Questions</i>	<i>17</i>
2. Architecture of the 8085 Microprocessor	18
2.1 Introduction	18
2.2 Block Diagram of the 8085 Microprocessor	18
2.3 Pin Diagram of 8085 Microprocessor	28
<i>Summary</i>	<i>33</i>
<i>Multiple-Choice Questions</i>	<i>34</i>
<i>Short-Answer-Type Questions</i>	<i>35</i>
<i>Review Questions</i>	<i>35</i>
<i>Answers to Multiple-Choice Questions</i>	<i>36</i>
3. Instruction Set and Addressing Modes of 8085 Microprocessor	37
3.1 Introduction	37
3.2 Addressing Modes	37
3.3 Instruction Set	40
3.4 Instruction and Data Formats	43
3.4 Symbols and Abbreviations	45
3.5 8085 Instructions	46
3.6 Instruction Timing Diagram	63
3.7 Timing Diagram	66

<i>Summary</i>	77
<i>Multiple-Choice Questions</i>	77
<i>Short-Answer-Type Questions</i>	79
<i>Review Questions</i>	79
<i>Answers to Multiple-Choice Questions</i>	80

4. Assembly – Language Programs of the 8085 Microprocessor 81

4.1 Introduction	81
4.2 Machine Language	82
4.3 Assembly Language	82
4.4 High-Level Language	83
4.5 Stack	87
4.6 Subroutines	89
4.7 Time-Delay Loops	91
4.8 Modular Programming	94
4.9 Macro	95
4.10 Instruction Format	96
4.11 Assembly-Language Programs	97
<i>Summary</i>	142
<i>Multiple-Choice Questions</i>	142
<i>Short-Answer-Type Questions</i>	144
<i>Review Questions</i>	145
<i>Answers to Multiple-Choice Questions</i>	146

5. Architecture of 8086 and 8088 Microprocessors 147

5.1 Introduction	147
5.2 Architecture of 8086	149
5.3 Registers	152
5.4 Logical And Physical Address	156
5.5 Address Bus, Data Bus, Control Bus	158
5.6 Memory Segmentation	158
5.7 8086 Memory Addressing	160
5.8 Pin Description of 8086	164
5.9 Memory Read and Write Bus Cycle of 8086	170
5.10 Intel 8088 Processor	176
5.11 Demultiplexing of the System Bus in 8086 and 8088 Microprocessors	180
5.12 Some Important ICs: 8284A, 8286/8287, 8282/8283, and 8288	183
<i>Summary</i>	189
<i>Multiple-Choice Questions</i>	190
<i>Short-Answer-Type Questions</i>	190
<i>Review Questions</i>	191
<i>Answers to Multiple-Choice Questions</i>	191

6. Instruction Set and Addressing Modes of the 8086 Microprocessor	192
6.1 Introduction	192
6.2 Addressing Modes	192
6.3 8086 Instruction Set	203
<i>Summary</i>	239
<i>Multiple-Choice Questions</i>	240
<i>Short-Answer-Type Questions</i>	241
<i>Review Questions</i>	241
<i>Answers to Multiple-Choice Questions</i>	242
7. Assembly-Language Programs of the 8086 Microprocessor and 8087, 80287 and 80387 Numeric Data Processors	243
7.1 Introduction	243
7.2 Assembly-Language Commands	246
7.3 Assembly Language Programs	255
7.4 8087, 80287 And 80387 Numeric Data Processors	289
7.5 8087 Numeric Data Processor	289
7.6 80287 Numeric Data Processor	303
7.7 80387 Numeric Data Processor	307
<i>Summary</i>	308
<i>Multiple-Choice Questions</i>	308
<i>Short-Answer-Type Questions</i>	310
<i>Review Questions</i>	310
<i>Answers to Multiple-Choice Questions</i>	312
8. I/O and Memory Interfacing Using 8085/8086	313
8.1 Introduction	313
8.2 Memory Interfacing	313
8.3 Interrupts of the 8085 Microprocessor	335
8.4 Interrupts of 8086/8088 Microprocessor	348
8.5 8259A Programmable Interrupt Controller	355
8.6 Programmable Peripheral Interface, 8255	368
8.7 8253 Programmable Counter/Interval Timer	382
<i>Summary</i>	397
<i>Multiple-Choice Questions</i>	398
<i>Short-Answer Type Questions</i>	399
<i>Review Questions</i>	400
<i>Answers to Multiple-Choice Questions</i>	403
9. Communication and Bus Interfacing with the 8085/8086 Microprocessor	404
9.1 Introduction	404
9.2 Serial Communication Interface 8251	404
9.3 Direct Memory Access (Dma) Controller 8257	417
9.4 8279—Programmable Keyboard And Display I/O Interface	430

9.5	8275 Crt Controller	441
9.6	Analog-to-Digital Converter Interfacing	445
9.7	Digital-to-Analog Converter Interfacing	458
9.8	Bus Interface	467
9.9	8250 Uart	480
9.10	16550 Uart	485
9.11	8089 I/O Processor	487
	<i>Summary</i>	493
	<i>Multiple Choice Questions</i>	493
	<i>Short-Answer Type Questions</i>	495
	<i>Review Questions</i>	496
	<i>Answers to Multiple-Choice Questions</i>	498

10. Applications of 8085/8086 Microprocessors

499

10.1	Introduction	499
10.2	Seven-Segment Display	500
10.3	Measurement of Electrical Quantities	504
10.4	Measurement of Physical Quantities	528
10.5	Microprocessor-Based Protection	549
10.6	Microprocessor-Based Traffic Control	552
10.7	Microprocessor-Based Firing Circuit of A Thyristor	558
10.8	Speed Control of DC Motor	562
10.9	Stepper Motor	569
	<i>Summary</i>	575
	<i>Multiple-Choice Questions</i>	575
	<i>Short Answer Questions</i>	576
	<i>Review Questions</i>	576
	<i>Answers to Multiple-Choice Questions</i>	578

11. 80186, 80286, 80386 and 80486 Microprocessors

579

11.1	Introduction	579
11.2	80186 Microprocessor Architecture	579
11.3	Pin Description of 80186	594
11.4	Addressing Modes of 80186	598
11.5	Data Types of 80186	598
11.6	Instruction Set of 80186	599
11.7	Comparison Between 8086 and 80186	601
11.8	Introduction to 80286	601
11.9	Architecture of 80286	602
11.10	Pin Diagram of 80286	606
11.11	Addressing Modes of 80286	609
11.12	Data Types of 80286	610
11.13	80286 Instruction Set	610

11.14	80286 Addressing Mode	614
11.15	Comparison Between 8086 and 80286	620
11.16	Comparison Between 80186 and 80286	621
11.17	Introduction to 80386	622
11.18	Architecture of 80386	623
11.19	Registers of 80386	624
11.20	Pin Functions of 80386	628
11.21	Addressing Modes of 80386	630
11.22	Data Types of 80386	631
11.23	Operating Mode of 80386	631
11.24	Instruction Set	638
11.25	Comparison Between 80286 And 80386	639
11.26	Introduction to 80486	640
11.27	Architecture of 80486	640
11.28	Pin Descriptions of 80486	643
11.29	Comparison Between 80386 And 80486	648
	<i>Summary</i>	648
	<i>Multiple-Choice Questions</i>	649
	<i>Short-Answer Questions</i>	650
	<i>Review Questions</i>	650
	<i>Answers to Multiple-Choice Questions</i>	652

12. Pentium and RISC Processors

653

12.1	Introduction	653
12.2	Pentium Internal Architecture	653
12.3	Pentium Operating Modes	662
12.4	Segmentation	665
12.5	Physical, Linear and Logical Address	666
12.6	Virtual 8086 Mode	670
12.7	Pin Description of Pentium Processor	673
12.8	Addressing Modes of The Pentium Processor	676
12.9	Pentium Bus Interfacing	677
12.10	System Management Mode (SMM) of the Pentium Processor	684
12.11	Cache Memories	686
12.12	Pentium MMX	697
12.13	Pentium Pro, Pentium II, and Pentium III: P6 Family Processors	698
12.14	Comparison of Pentium and Pentium-Pro Processor	704
12.15	Pentium 4 Processor	705
12.15	Comparison of Pentium III and Pentium 4 Processors	713
12.16	Risc Processors	713
12.17	Core Processor	717
	<i>Summary</i>	718

Multiple-Choice Questions 719
Short-Answer Questions 720
Review Questions 720
Answers to Multiple-Choice Questions 721

13. Introduction to 8051 Microcontroller **722**

13.1 Introduction 722
13.2 Architecture of 8051 Microcontroller 725
13.3 Memory Organization 730
13.4 Pin Diagram of 8051 Microcontroller 735
13.5 Timers/Counters 744
13.6 Serial Communication 749
13.7 Interrupts 756

Summary 760

Multiple-Choice Questions 760

Short-Answer Questions 761

Review Questions 761

Answers to Multiple-Choice Questions 762

14. Instruction Set and Programming of the 8051 Microcontroller **763**

14.1 Introduction 763
14.2 Addressing Modes 763
14.3 8051 Instruction Set 767
14.4 Simple Examples in Assembly-Language Programs of 8051 Microcontroller 792
14.5 Assembly-Language Programs 795
14.6 Applications of Microcontrollers 807

Summary 827

Multiple-Choice Questions 827

Short-Answer/Viva-Voce Questions 828

Review Questions 828

Answers to Multiple-Choice Questions 829

Appendix A - OPCODE of the 8085 Instruction Set 830

Appendix B - Some Important Tables for 8051 833

Appendix C - Some Important Tables for 8085 838

Appendix D - Some Important Tables for 8086 842

Model Question Paper - 1 856

Model Question Paper - 2 860

Model Question Paper - 3 863

Index 867

PREFACE

Overview

Though progress and advancement in microprocessor technology has been very fast, the study of the basic principles, e.g. the digital building blocks of 8085 and 8086 microprocessors and 8051 microcontrollers are continuing. Nowadays, this subject forms a part of undergraduate courses, namely Electrical, Instrumentation, Electronics, Electronics and Communication, Computer Science and Engineering, and Information Technology. The present book, thus, aims to be of use to students of EE, CS & E, ECE, IN, IT and engineers working in automation industries.

Aim

Although a large number of books on microprocessors are available in the market, most cover either the 8085 microprocessor or its interfacing or advanced microprocessors like the 8086 to Pentium processors or the 8051 microcontroller. Consequently, there is no book that covers all the theory starting from the 8085 and 8086 microprocessors, 80186, 80286, 80386 and 80486 to Pentium processors and the 8051 microcontroller. Since almost none of the reference books have syllabus compatibility and right pedagogy, many students find it difficult to conceptualize the subject. As per feedback from students and teachers, there is need for a single book that can cover all the topics as per university curricula. This book is an outcome of my decade-long teaching experience of Microprocessors and Microcontrollers at SSGM College of Engineering, Shegaon; Panjab Engineering College, Chandigarh; and NITTTR, Kolkata. The content of this book covers the syllabi of microprocessors and microcontrollers of major Indian universities like WBUT, UPTU, PTU, RGTU, Mumbai University, Pune University, Anna University, JNTU, VTU, and many more.

I have written this book attempting to cover all the important topics of 8085 and 8086 microprocessors, 80186 to 80486 microprocessors, Pentium processors and the 8051 microcontroller. The examples of assembly-language programs and a variety of theoretical and multiple-choice questions at the end of each chapter give students a chance to check and enhance their conceptual understanding. Though this book is written to help students develop basic concepts of microprocessor and microcontroller architecture, programming and their applications, this course will also mitigate a definite percentage of every competitive examination of engineering professionals, namely IES, UPSC, GATE, etc.

Salient Features

Some salient features of this book are

- ✦ Complete coverage of the syllabi on Microprocessors and Microcontrollers of major Indian universities
- ✦ Large number of assembly-language programs incorporated from examination papers of different universities and competitive examinations like IES, UPSC, and GATE
- ✦ Architecture, Programming, Interfacing of Microprocessors and Microcontrollers explained in lucid language
- ✦ Detailed coverage of Advanced Microprocessors
- ✦ Hands-on approach through applications such as Traffic Control, Keyboard Interfacing, Stepper Motor Control, Seven Segment Display, Control of Firing Circuit of a Thyristor

Feature	Description	Benefit
Brief Introduction	A brief description of the chapter topics is given.	Student gets a subjective overview of the contents of the chapter.
Diagrams	Over 550 diagrams are given in the text.	Diagrams are an important tool in the presentation of text material in a clear and lucid manner. These enable the reader to effectively understand the various microprocessor concepts discussed in a chapter.
Definitions	Wherever appropriate, useful definitions related to the topic being described have been inserted.	These will help the students to quickly revise the definitions before an exam.
Solved Examples/ Programs	Close to 100 Solved Examples/ Programs are present in the text.	A stepwise approach for solving problems is used throughout the book, thereby making it easier for the reader to apply the learnt concepts.
Summary	A concise Summary is provided at the end of each chapter.	It underlines the important concepts learnt in the chapter.
Multiple-Choice Questions	More than 250 Multiple-Choice Questions are present in the text.	These help readers have a quick revision of the concepts discussed in the chapter.
Review Questions	Almost 400 Review Questions are given in the text to test the theoretical grasp of the students.	These are set to develop confidence in the principles discussed and check the understanding of the student.
Short-Answer/ Viva-Voce Questions	130 Short-Answer/Viva-Voce Questions are present as chapter-end pedagogy.	These will be helpful in revising the concepts and preparing for competitive exams.
Model Question Papers	3 Model Question Papers are provided at the end of the text which focus on university patterned questions.	Essentially, these Model Question Papers make the reader realize the pattern of questions in university examinations.

Chapter Organization

This book has 14 chapters. **Chapter 1** covers the evolution of microprocessors and their applications in detail. **Chapters 2 to 4** are devoted to the 8085 microprocessor architecture, addressing modes, instruction set and assembly-language programs; whereas **chapters 5 to 7** cover architecture of 8086 and 8088 microprocessors, addressing modes, instruction set, assembly-language programs and 8087, 80287 and 80387 numeric data processors. **Chapters 8 and 9** incorporate the interfacing devices of 8085 and 8086 microprocessors. **Chapter 10** deals with the applications of 8085 and 8086 microprocessors. **Chapters 11 and 12** cover advanced microprocessors. Finally, **chapters 13 and 14** discuss the architecture, addressing modes, instruction set, assembly-language programs and applications of 8051 microcontroller.

Given below is a detailed chapter synopsis.

Chapter 1 presents the basic concept of microprocessors and microcontrollers, history of microprocessors, evolution of microprocessors and its applications, evolution of microcontrollers and their applications.

Chapter 2 deals with the architecture of the 8085 microprocessor in a generalized way. This chapter covers the block diagram of the 8085 microprocessor and its operating principles. It also elaborately explains the pin diagrams of 8085 microprocessors and functions of pins.

Chapter 3 describes the different addressing modes and instruction sets of the 8085 microprocessor. There is detailed discussion on instruction timing diagrams of 8085 μ P, fetch cycle, execute cycle and machine cycles, timing diagram of memory read, memory write, I/O read and write operations.

Chapter 4 deals with machine-level languages, assembly-level languages, and high-level languages with detailed coverage of operation of stacks, subroutines and time delay loops. This chapter also incorporates modular programming, macro, instruction formats and assembly-language programs.

Chapter 5 introduces the architecture of 8086 and 8088 microprocessors. It also covers minimum-mode and maximum-mode configurations, memory addressing, pin descriptions of 8086 and 8088, and other supporting ICs such as 8284A, 8286/8287, 8282/8283 and 8288.

Chapter 6 covers the different addressing modes and instruction set of the 8086 in detail.

Chapter 7 introduces assembly-language commands and assembly-language programs of 8086. This chapter also includes the architecture of the 8087 numeric data processor, pin descriptions, interfacing with 8086, instruction set and assembly-language programs of 8087 with explanation on the architectures of numeric data processors 80287 and 80387.

Programmable peripheral chips are equipment that support any interfacing devices with microprocessors. There are many such devices available in the market. **Chapter 8** covers memory ICs and their interfacing, interrupts of 8085, 8086 and 8088 microprocessors, 8259 Programmable Interrupt Controller, 8255A Programmable Peripheral Interface, and the 8253 Programmable timer/counters.

Chapter 9 presents the 8251 Serial Communication Interface, 8279 Programmable Keyboard and Display Interface, 8257 Direct Memory Access (DMA) Controller, 8275 CRT controller, ADC as well as DAC ICs and their interfacing, Bus interface, RS232C, IEEE-488, Parallel printer interface, 8250 UART, 16550 UART and 8089 I/O processor.

In industries, there are varieties of microprocessor applications such as instrumentation, industrial automation and aerospace, etc. The applications of microprocessors in display systems; measurement of electrical quantities such as voltage, current, frequency and phase angle; measurement physical quantities like displacement, strain, force, temperature, water level and speed; microprocessor-based protection, traffic light control, and speed control of motors are incorporated in **Chapter 10**.

Chapter 11 describes the 80186 microprocessor architecture, pin description, addressing modes, data types and instruction sets. It also covers the architecture, pin description, addressing modes (real addressing modes and protected virtual address mode), data types and instruction sets of the 80286 microprocessor, and the operations of 808386 and 80486.

Chapter 12 presents Pentium architecture, Pentium operating modes, segmentation, physical, linear and logical address, virtual 8086 mode, pin descriptions of Pentium processors, cache memories, Pentium MMX processor, Pentium Pro, Pentium II, Pentium III and Pentium 4 processors, RISC and CISC processors in detail.

Chapter 13 describes the architecture of the 8051 microcontroller. This chapter also describes the special function registers and memory organization, timer/counters, interrupts and serial communication of the 8051 microcontroller.

Chapter 14 deals with the addressing modes and instruction set of the 8051 microcontroller with elaborate emphasis on assembly-language programs for the 8051 microcontroller, and applications of microcontrollers for keyboard interfacing, A/D converter interfacing, traffic light control, stepper motor control and washing machine control.

Appendix A covers opcodes of the 8085 instruction set.

Appendix B provides the instruction-set summary for 8051.

Appendix C contains the instruction-set summary for 8085

Appendix D includes the instruction-set summary for 8086.

Finally, there are Model Question Papers at the end of the book.

Web Supplements

This book also has an exhaustive Online Learning Centre, which can be accessed at <https://www.mhhe.com/mandal/> designed to provide valuable resources for instructors and students.

✦ For Instructors

- Powerpoint Slides (chapter-wise)
- Solution Manual For Selected Questions
- Tutorial Sheets

✦ For Students

- Instruction Set of 8085 Microprocessor
- Instruction Set of 8086 Microprocessor
- Instruction Set of 8051 Microcontroller
- Lab Experiments Manual
- Answers for selected Review Questions
- Viva-voce Questions with Answers
- References/Suggested Further Reading

Acknowledgements

I have received cooperation and inspiration for this book from Dr Gurnam Singh, PEC, Chandigarh; Dr S Chatterjee, NITTTR, Chandigarh; Dr S K Bhattachariya, Former Director, NITTTR, Kolkata; Prof. Amitabha Sinha, Director School of IT, WBUT; Dr C K Chanda and Dr P Shyam, Bengal Engineering College, Shibpur; Dr P Sarkar, Professor and Head Electrical, Dr S Chattopadhyay, Associate Professor, and Dr S Pal, Asst. Professor, NITTTR, Kolkata. I am also thankful to other staff of Electrical Engineering—Mr A K Das, Mr N K Sarkar, Mr S Roy Choudhury and Mr Surojit Mallick—who helped me complete the manuscript of this book.

At this juncture, I would also like to express my gratitude to the numerous reviewers who took out time to review the manuscript. Their names are given below.

Madan Mohan Agarwal	Birla Institute of Technology, Jaipur, Rajasthan
P K Mukherjee	Institute of Technology—Banaras Hindu University (IT—BHU), Varanasi, Uttar Pradesh
Siddharth Chauhan	National Institute of Technology (NIT), Hamirpur, Himachal Pradesh
Vishal Nimbork	Ajay Kumar Garg College of Engineering, Ghaziabad, Uttar Pradesh
Sampath Kumar V	Jagadguru Sri Shivarathreshwara (JSS) Academy of Technical Education, Noida, Uttar Pradesh

Agamani Chakraborty	Asansol Engineering College, Asansol, West Bengal
Pinaki Ranjan Ghosh	ADAMAS Institute of Technology, Kolkata, West Bengal
Jaydip Nath	Future Institute of Engineering and Management (FIEM), Kolkata, West Bengal
A G Keskar	Visvesvaraya National Institute of Technology, Nagpur, Maharashtra
C A Ghuge	PE Society's Modern College of Engineering, Pune, Maharashtra
Sunil N Kore	Walchand College of Engineering, Sangli, Maharashtra
Vikas J Dongre	Government Polytechnic, Nagpur, Maharashtra
Lyla B Das	National Institute of Technology (NIT), Calicut, Kerala
S Solai Manohar	College of Engineering, Anna University, Chennai, Tamil Nadu
Jayakumar Vijayaraghavan	Rajalakshmi Engineering College, Chennai, Tamil Nadu
S R Malathi	Sri Venkateshwara College of Engineering, Chennai, Tamil Nadu
S Rathinavel	Sri Subramanya College of Engineering and Technology, Udumalpet, Tamil Nadu
M Sreelatha	Kakatiya Institute of Technology and Science, Warangal, Andhra Pradesh
M Shailaja	JNTU College of Engineering, Kakinada, Andhra Pradesh
M Sampath Kumar	College of Engineering, (Andhra University), Visakhapatnam, Andhra Pradesh

Feedback

Any suggestions for improving the contents of the text are always welcome. Please give suggestions or feedback to the publisher's email id mentioned below.

Soumitra Kumar Mandal

Publisher's Note

Do you have a feature request? A suggestion? We are always open to new ideas (the best ideas come from you!). You may send your comments to tmh.csefeedback@gmail.com (kindly mention the title and author name in the subject line).

Piracy-related issues may also be reported.

Chapter 1

Introduction to Microprocessors and Microcontrollers

1.1 INTRODUCTION

The computer is a machine that processes data to generate information with speed and accuracy. Electronic and electromechanical devices, and software make this programmable machine. The basic block diagram of a computer is shown in Fig. 1.1. The computer comprises four basic units, namely, input (I/P), memory, output (O/P), and central processing unit.

1.1.1 Input Devices

An input device accepts data from the environment, converts it into digital form and sends it to the memory of the computer for storing. Commonly used input devices are punched cards, paper tapes, magnetic tapes, floppy disks, and magnetic disks.

Card readers, paper tape readers, magnetic tape readers, disk drives read data transmitted by input devices. A keyboard terminal can be used as input to the computer. Optical mark readers and optical character readers, are input devices that are scanned by an array of photocells. The input is then converted into machine code and transmitted into the memory of the computer for processing. On identical principles, bar-code readers read the information prepared in bar code for application by computers. In magnetic ink readers, information written or printed in magnetic ink is read and transmitted directly to the memory for processing. Electronic mouse, touchscreens, and light pens are also used as input devices. Figure 1.2 shows the different input devices.

1.1.2 Memory

A computer system also has storage areas, often referred to as memory. The memory unit stores the information to be processed by the CPU. This information consists of the program as well as data. The memory can receive data, hold them and deliver them when instructed to do so. The

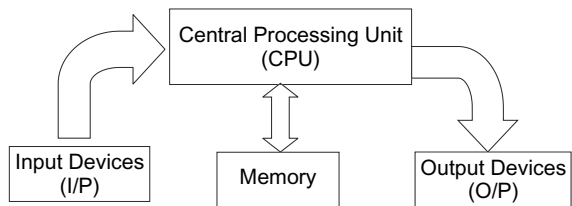


Fig. 1.1 Basic block diagram of a computer

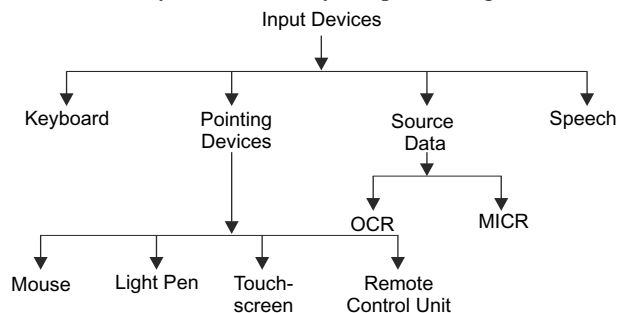


Fig. 1.2 Input devices

storage available in the memory is also known as *main storage* or *primary storage* as in Fig. 1.3. The data can be processed only when it is available in the main memory, which is finite. It may be increased by adding auxiliary or *secondary storage*, such as magnetic tapes or magnetic disks as shown in Fig. 1.4. The information stored in the auxiliary storage can be transferred to the main memory for processing at a high speed.

1.1.3 Output Devices

When a program is executed in the computer, the result will be computed and readily available for display. The computer needs output devices to display the information to the user. The most commonly used output devices are monitor screens, printers, graphics plotters, speech and microfilm as depicted in Fig. 1.5.

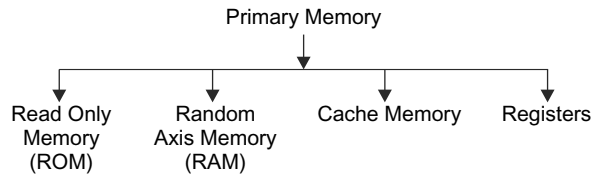


Fig. 1.3 Primary memory

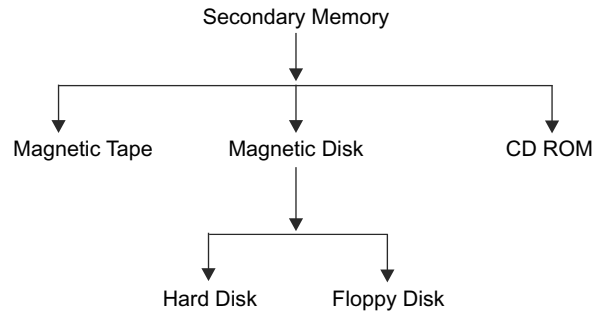


Fig. 1.4 Secondary memory

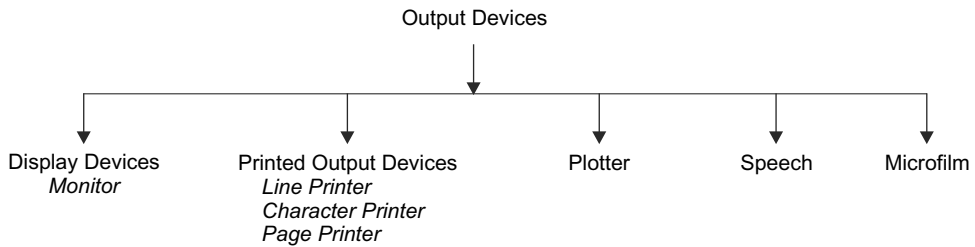


Fig. 1.5 Output devices

1.1.4 Central Processing Unit

The central processing unit is the brain of the computer. It executes the programmer’s software and controls the memory, input and output devices. Programs are stored in the memory. The CPU fetches instructions of a program sequentially from the memory. It fetches one instruction at a time, decodes it and then executes it. After decoding an instruction, the CPU comes to know what operations are to be performed. It also comes to know whether the data to be processed are in the memory, general-purpose registers of the microprocessor or at input/output ports. If data are in the general-purpose registers, the CPU executes the program. The CPU controls memory, input and output devices to receive, store and send data/result of the program under execution. Under its control programs, data and results are displayed on the CRT, stored in the memory or printed by the printer. The major components of a CPU are ALU, timing and control unit and registers as depicted in Fig. 1.6.

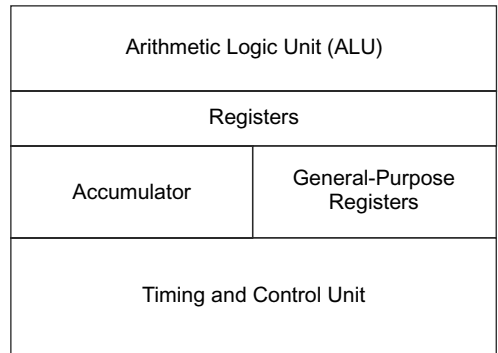


Fig. 1.6 Block diagram of Central Processing Unit (CPU)

Arithmetic Logic Unit (ALU)

The ALU performs the actual processing of data including addition, subtraction, multiplication and also division. This unit also performs certain logical operations such as comparing two numbers to see if one is larger than the other or if they are equal. Arithmetic or logic operations are performed by bringing the required operands into the ALU. Suppose two numbers located in the main memory are to be added. They are brought into the arithmetic unit and temporarily stored in registers or in accumulators associated with this unit where the actual addition is carried out. The result is placed in one of the registers and subsequently transferred to the memory.

Control Unit

The control unit directs and coordinates all activities of the computer system including the following:

- ✦ Control of input and output devices
- ✦ Entry and retrieval of information from storage
- ✦ Routing of information between storage and arithmetic logic unit
- ✦ Direction of arithmetic and logical operations

Although the control section does not process data, it acts as a central nervous system for the other data manipulating components of the computer. At the beginning of the processing, the first program instruction is selected and fed into the control section from the program storage area. Thus it is interpreted, and from there signals are sent to other components to execute the necessary action.

The central processing unit built on a single IC is called a *microprocessor*. In a microcomputer, the microprocessor acts as the central processing unit. Figure 1.7 shows the block diagram of microcomputer. Architecture of microprocessors is explained in this chapter.

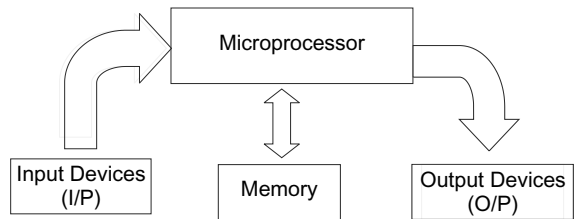


Fig. 1.7 Basic block diagram of a microcomputer

1.2 MICROPROCESSOR

The microprocessor is a multipurpose, programmable, and clock-driven integrated circuit. This IC can read binary instructions from any storage device called memory. It also accepts binary data as input, processes data according to instructions, and provides results as output.

The microprocessor is the Central Processing Unit (CPU) of digital computers and it is constructed with IC technology. Figure 1.8 shows the block diagram of a microprocessor. The microprocessor has a digital circuit for data handling and computation under program control. The microprocessor is a data-processing unit. Data processing includes both computation and data handling. Computation is performed by logic circuits called the Arithmetic Logic Unit (ALU). The ALU is used to perform Add, Subtract, AND, OR, XOR, Compare, Increment, and Decrement functions. The ALU cannot perform any functions without control signals. In order to process data, the microprocessor must have control logic which instructs the microprocessor how to decode and execute the program. A program is a set of instructions required by a computer to perform any task.

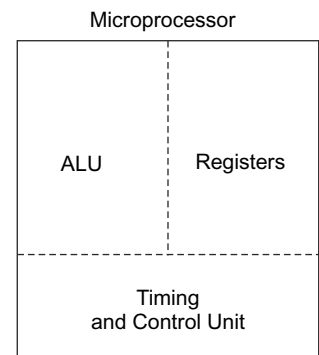


Fig. 1.8 Architecture of a microprocessor

The control logic sends signals to the microprocessor and instructs how to operate the stored instructions in memory. Figure 1.9 shows the operation technique of a microprocessor. There are four steps of operation. In Step 1, the microprocessor fetches an instruction and in the next step, the control logic decodes what the instruction has to do. Then decoding is done in the third step and in the last step, the microprocessor executes the instruction.

The microprocessor always operates in binary digits 0 and 1, known as bits. Bit is an abbreviation for ‘binary digit’ which can be represented in terms of voltages. The microprocessor recognizes and processes a group of bits called *word*. Microprocessors are classified according to their word length such as 8-bit, 16-bit, 32-bit and 64-bit microprocessors. Microprocessor ICs are programmable so that instructions can be executed by a microprocessor to perform given tasks within its capability. The instructions are stored in a storage device which is called the memory, and the microprocessor can read instructions from the memory.

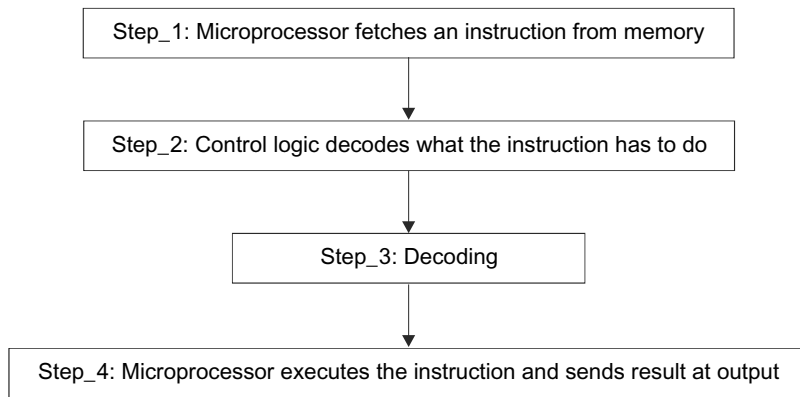


Fig. 1.9 Operation technique of a microprocessor

1.3 MICROCOMPUTER

Generally, the words ‘microprocessor’ and ‘micro-computer’ are used to correspond to the same thing, but in fact these words have different meanings. The microprocessor is an Integrated Circuit (IC) developed on LSI or VLSI technology. It is the core of any computer system, but a microprocessor by itself is completely useless, until external peripheral devices are connected with it to enable it to interact with the outside world. The microcomputer is a complete computing system and it is built with a microprocessor, input/output devices and memory (RAM and ROM). The schematic block diagram of a microcomputer is shown in Fig. 1.10. The detailed architecture of a microcomputer is illustrated in Fig. 1.11.

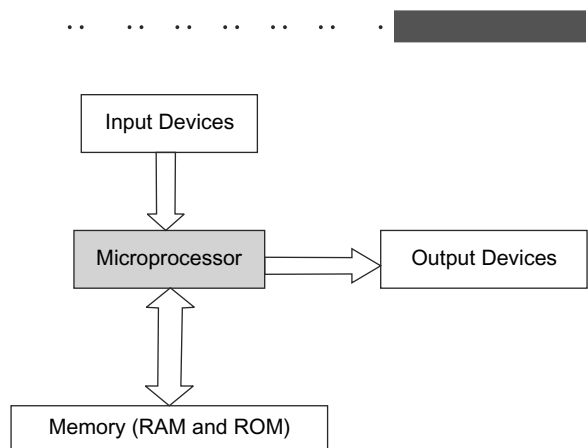


Fig. 1.10 Schematic block diagram of a microcomputer

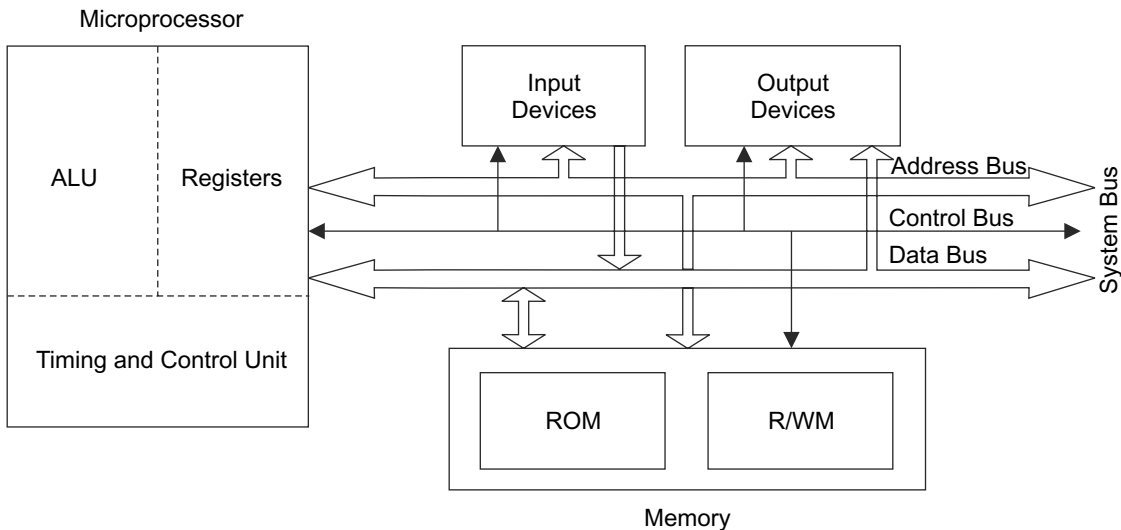


Fig. 1.11 Architecture of a microcomputer

Arithmetic/Logic Unit (ALU) The ALU performs arithmetic operations such as addition, subtraction, multiplication and division, and logic operations, namely, AND, OR, XOR, Complement, Rotate and Shift. After the operations, results must be stored either in a specified register or in the memory.

Register The microprocessor has various general-purpose registers such as B, C, D, E, H, L, and the Accumulator (A). These registers are used to store data and addresses temporarily during the execution of a program.

Timing and Control Unit The timing and control unit provides the necessary timing and control signals to perform any operation in the microcomputer. Actually, it controls the flow of data between the microprocessor and memory/peripheral devices.

Input Devices The input devices transfer data in binary from the outside world to the microprocessor. The most commonly used input devices are the keyboard, switches, mouse, scanner, and analog-to-digital converter.

Output Devices The output devices transfer data from the microprocessor to the outside world, e.g. printers, plotters, monitors, and magnetic tapes.

Memory The memory unit stores the binary information such as instructions and data, and provides that information to the microprocessor for processing. To execute any instruction, the microprocessor reads instructions and data from memory. After the computational operations in the ALU, microprocessor again stores results in memory for further use.

System Bus: Address Bus, Data Bus and Control Bus The microprocessor always communicates with input/output devices and memory via some path called the *system bus*. The system bus consists of address bus, data bus and control bus. *Address bus* is used to locate any input/output devices and memory. *Data bus* is used to transfer data in binary form between the microprocessor and peripherals. The microprocessor communicates with only one peripheral at a time. The timing signals are provided by the control bus of the microprocessor.

1.4 ARCHITECTURE OF MICROPROCESSORS

Usually, the function of microprocessors is to process or manipulate data. Except data manipulations, the processor is used to read data and instructions from the memory, read and write data to the memory, read data from input devices and write data into output devices. To perform these operations, the processor communicates with the memory and I/O devices through the address bus, data bus and control bus.

The *address bus* carries the address information from the processor to locate the memory as well as I/O devices. This bus is unidirectional.

The *data bus* carries the data between the processor and peripheral devices. This bus is bi-directional.

The *control bus* is used to carry control/status information. This bus is bi-directional. Figure 1.12 shows the interaction between processors and memory and I/O devices using address bus, data bus and control bus.

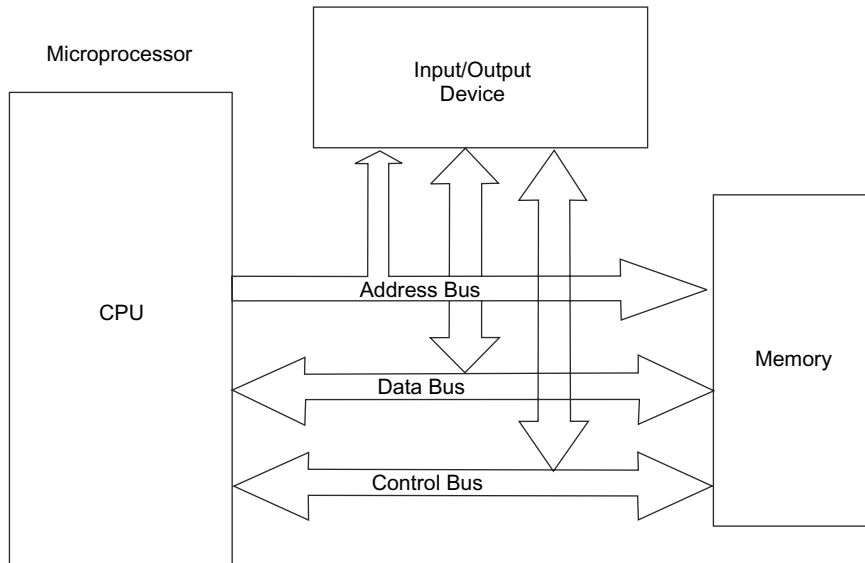


Fig. 1.12 Communication between CPU and memory and input/output devices

Depending upon the number of data buses and memory, there are three types of processor architecture such as

- (i) Von Neumann architecture
- (ii) Harvard architecture
- (iii) Super Harvard architecture

1.4.1 Von Neumann Architecture

Figure 1.13 shows the Von Neumann architecture of processors and this architecture is most commonly used in processors. In this architecture, one memory chip is used to store both instructions and data. The processor interacts with the memory through address and data buses to fetch instructions as well as data.

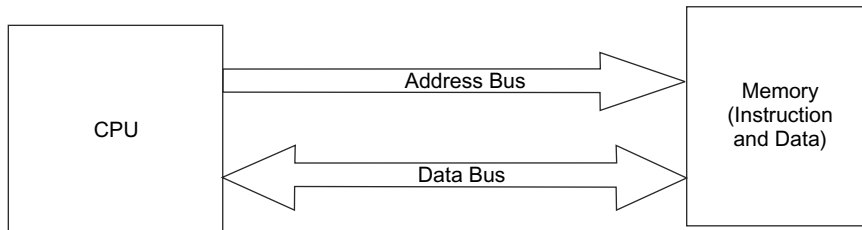


Fig. 1.13 Von Neumann architecture of processors

1.4.2 Harvard Architecture

Figure 1.14 shows the Harvard architecture of a processor. In this processor architecture, two separate memory blocks, namely, program memory and data memory are used. The program memory is used to store only instructions and data memory is used to store data. The program memory address bus is used to locate the program memory and through program memory data bus, the processor can write/read instructions to/from memory. Similarly, the data memory address bus is used to locate data memory and the data memory data bus can be used to access the data memory. Consequently, this architecture is efficient than Von Neumann architecture as the instructions and data will be accessed very fast.

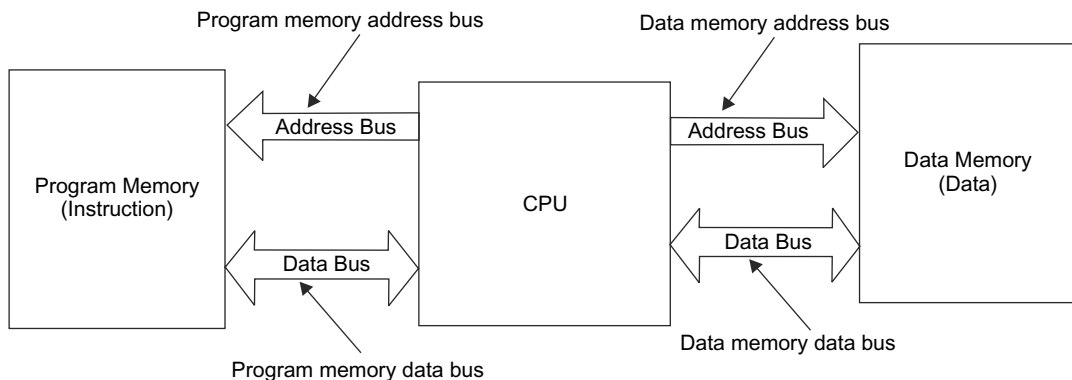


Fig. 1.14 Harvard architecture of a processor

1.4.3 Super Harvard Architecture

Figure 1.15 shows the super Harvard architecture which is the modified Harvard Architecture. Generally, the data memory is accessed more frequently than the program memory in Harvard architecture. In the super Harvard architecture, the program memory can store secondary data to balance the load on both program memory and data memory. The instruction cache is in-built within the processor. This architecture is most commonly used in Digital Signal Processing (DSP).

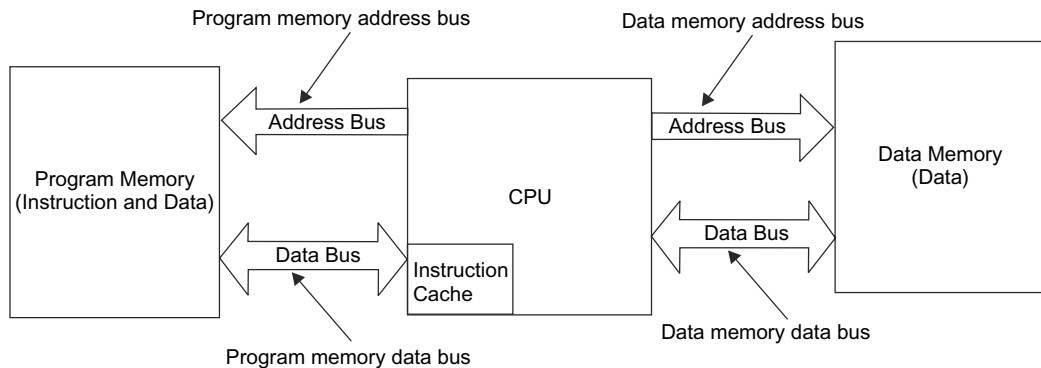


Fig. 1.15 Super Harvard architecture

1.5 HISTORY OF MICROPROCESSORS

The history of computation began with the abacus. The abacus is a manual device. In an abacus, numerical information can be represented in physical form, and this information can also be manipulated in physical form to produce the necessary output. Even a thousand years before Christ, the abacus was very well-known and extensively used for arithmetical calculations. Consequently, it was known as the first actual machine which was used to perform addition, subtraction, division and multiplication.

In 1643, Blaise Pascal, a French mathematician and philosopher, invented the first mechanical calculator to perform addition as well as subtraction. In the 17th century, the multiplication and division facilities were added by the German mathematician Gottfried Leibniz.

In 1832, the Difference Engine was developed by Charles Babbage, a professor of mathematics at Cambridge University. This machine could add, subtract, multiply, divide and perform a sequence of steps automatically. In 1887, Herman Hollerith invented a device for automatic census tabulation. The first large-scale electronic digital computer was designed and constructed at the Moore school of Electrical Engineering of the University of Pennsylvania. In 1943, JW Mauchly and J Presper Eckert prepared a proposal for the US army to build an Electronic Numerical Integrator and Computer (ENIAC), and subsequently they started construction of the ENIAC. In 1944, the ENIAC team members began work on stored program computers. Then ENIAC was finally ready in 1946. It occupied a room approximately of 12 m × 6 m. It contained nearly 18000 vacuum tubes and its power consumption was about 150 kW. It operated on numbers with ten decimal digits. Addition could be carried out at the rate of 5000 calculations per second, multiplication at 350 per second and division at 166 per second. It was able to store upto 20 different numbers and recall them immediately whenever required. After that, John Von Neumann developed an improved version of ENIAC with the help of all ENIAC team members.

In 1949, the Electronic Delay Storage Automatic Calculator (EDSAC) was developed by Maurice Wilkes at University Mathematical Laboratory, Cambridge University. In 1951, the Universal Automatic Computer was built. In 1952, the Electronic Discrete Variable Automatic Computer (EDVAC) was developed by JW Mauchly and J Presper Eckert. This was the first electronic machine to use binary arithmetic. It operated on binary numbers of 43 digits and could store over 1000 numbers for immediate recall. This was also the first machine to use an external store using magnetic recording.

After World War II, scientists made great achievements in solid-state technology development and invented the transistor, i.e., a solidstate device, in 1948, at Bell Laboratories. Initially, germanium was the chief material for making the early semiconductor devices such as transistors. The use of silicon lowered

costs, because silicon is much more plentiful than germanium. The mass-production methods made transistors common and inexpensive. Then computer designers started to work on how to use transistors place of vacuum tubes in the late 1950s.

In the early 1960s, the American firm International Business Machines (IBM) manufactured huge solid-state scientific computers named IBM 7090 using solidstate technology. These systems required air-conditioned rooms. Usually, these systems were used for commercial and scientific applications to process large amounts of data. The cost of these computers was very high. So, the work on building small computers had been started. In the 1960s, the semiconductor industry developed a way to integrate a number of transistors on one silicon wafer. The transistors were connected together with small metal traces. When the transistors were connected together, they became a circuit which performed different functions such as gate, flip-flop, register, counter or adder. This new technology created the basic semiconductor building blocks. The building blocks or circuit modules made this way were known as an Integrated Circuit (IC). From then on, integrated circuits became feasible and the integration has been developed with time. There are three different stage of development of ICs from the period 1961 to 1972, namely, Small Scale Integration (SSI), Medium Scale Integration (MSI) and Large Scale Integration (LSI). In general, an SSI chip has dozens of transistors with their associated circuit components, but an MSI chip has hundreds of transistors and an LSI chip has thousands of transistors.

Due to development of SSI, MSI and LSI ICs, desktop computers were built at the end of the 1960s. These desktop computers were called *minicomputers* which were used in scientific applications. In the late 1960s and early 1970, Large Scale Integration (LSI) became common. Large-scale integration was making it possible to produce more and more digital circuits on a single IC. In 1965, Gordon More noted that the number of transistors on a chip doubled every 18 to 24 months. He made a prediction that semiconductor technology will double its effectiveness every 18 months. After that the next stage of development was started by the active research and development effort on solid-state technology. This stage of development was called Very Large Scale Integration (VLSI). By the 1980s, VLSI gave us ICs with over 100,000 transistors.

The microprocessor is an integrated circuit and it is the combination of solid-state technology development and the advancing computer technologies. It was developed in the early 1970s using LSI. It performs both control and processing functions having the low cost of a device and the flexibility of a computer.

1.6 EVOLUTION OF MICROPROCESSORS

In 1971, the Intel Corporation introduced the first 4-bit microprocessor 4004 which was developed using LSI technology. In 1972, the 8-bit microprocessor 8008 was produced by Intel. These microprocessors were not able to survive as general-purpose microprocessors due to their limitations and low performances. The first general-purpose 8-bit microprocessor 8080 was developed in 1974 by Intel. The microprocessor 8085 followed 8080 with some additional features. The limitations of 8-bit microprocessors were low operating speed, limited memory-addressing capability, less number of general-purpose registers and less number of instructions. To overcome all limitations of the 8085 microprocessor, computer scientists and designers worked to develop more powerful processors in terms of architecture, operating speed, memory and instruction set. As a result, a 16-bit microprocessor 8086 was developed in 1978.

Thereafter, the 80186 processor was designed with few more instructions and additional on-chip circuits such as clock generators, timers, DMA controllers and interrupt controllers, but the addressing capability was same as the 8086 microprocessor in 1982. But due to need of large memory in advance applications, processor designers put effort to design advanced microprocessors. The 80286 microprocessor is the first advanced microprocessor with proper memory management and protection abilities. It was developed by Intel in 1982 and it has an address capability of 16 Mbyte and its operating frequency is 12.5 HMz. The semiconductor technology could support the fabrication of a CPU with a 32-bit word size and higher operating frequency.

Hence, the 32-bit processor 80386 was developed. The first 32-bit processor was 80386. The numerical processor 80387 is compatible with 80386. In 1989, the 80486 was developed by Intel which combines all the features of 80386 after incorporating the math processor 80387 inside processor.

After the 80486 microprocessor, the Pentium family of processors was developed. The name Pentium was derived from the Greek *pente*, meaning ‘five’, and the Latin ending *-ium*. The term ‘Pentium processor’ refers to a family of microprocessors that share a common architecture and instruction set. The original Pentium processor was a 32-bit microprocessor produced by Intel. The first Pentium processors, P5, were developed in 1993. The P5 processor operated at a clock frequency of either 60 MHz or 66 MHz. This processor had 3.1 million transistors. The next version of the Pentium processor family, the P54C processor, was introduced in 1994.

In 1996, the Pentium MMX was introduced with the same basic micro-architecture with MMX instructions, and larger caches. The P55C (or 80503) Pentium MMX was introduced by Intel in October 1996 and it was based on the P5 core. It featured a new set of 57 MMX instructions intended to improve performance on multimedia tasks.

The Pentium Pro is a sixth-generation x86 microprocessor developed and introduced by Intel in November 1995. It was based on the P6 micro-architecture. While the Pentium and Pentium MMX had 3.1 and 4.5 million transistors, respectively, the Pentium Pro contained 5.5 million transistors.

The Pentium II processors refer to Intel’s sixth-generation micro-architecture called ‘Intel P6’, introduced in May 1997. This processor consisted of 7.5 million transistors. The Pentium II was an improved version of the first P6-generation core of the Pentium Pro CPUs, which contained about 5.5 million transistors. In early 1999, the Pentium II was superseded by the Pentium III.

The Pentium III processors were based on the sixth-generation Intel P6 microarchitecture introduced in February 1999. These processors were very similar to the earlier Pentium II microprocessors with the addition of the SSE instruction set to accelerate floating point and parallel calculations. The first Pentium III variant was the *Katmai*, Intel 80525. It was first released at speeds of 450 and 500 MHz. Two more versions were released: 550 MHz in May 1999 and 600 MHz in August 1999. It was built on a 0.18 μm process. Pentium III Coppermines running at 500 to 733 MHz were first released in October 1999. From December 1999 to May 2000, Intel released Pentium IIIs running at speeds of 750, to 1000 MHz (1 GHz). The third revision, Tualatin (80530), was a trial for Intel’s new 0.13 μm process. Pentium III Tualatins were introduced during 2001 and these processors could operate at speeds of 1.0 to 1.4 GHz. Tualatin performed quite well, especially in variations which had 512 KB L2 cache.

The Pentium III was eventually superseded by the Pentium 4. The Pentium 4 brand refers to Intel’s line of single-core mainstream desktop and laptop central processing units developed in November 2000. This processor had the 7th-generation micro-architecture, called NetBurst. The original Pentium 4, codenamed ‘Willamette’, ran at 1.4 and 1.5 GHz and was released in November 2000 on the Socket 423 platform. In 2004, the initial 32-bit x86 instruction set of the Pentium 4 microprocessors was extended by the 64-bit x86-64 set. Pentium 4 CPUs introduced the SSE2 and, in later versions, SSE3 instruction sets were released to accelerate calculations, transactions, media processing, 3D graphics, and games. In 2005, the Pentium 4 was complemented by the Pentium D and Pentium Extreme Edition dual-core CPUs.

A dual-core processor is a CPU with two separate cores on the same die, each with its own cache. It is the equivalent of getting two microprocessors in one. The dual core processor is the first double core technology from Intel. It is a better performer than all previous processors in the Pentium series. A maximum of 2.33 GHz is available for model no. T2700 with a 2 MB L2 cache and a maximum of 667 MHz speed. The AMD Athlon 64 X2 Dual-Core Processor was developed in 2007. This processor can support SSE, SSE2, SSE3, MMX™, 3D technology and legacy x86 instructions.

Intel Core 2 Extreme Quad-Core Processor QX6000 was introduced by Intel in 2007. This processor is designed to deliver performance across applications and usages in Internet, image processing, video content

creation, 3D, CAD, games, speech, multimedia and multitasking user environments. Intel 64 architecture enables the processor to execute operating systems and applications written to take advantage of the Intel 64 architecture. Quad-core processors are available in the FC-LGA6 package with a 2x4 MB L2 cache.

The Intel Core 2 Duo processor uses architecture to create two cores on a single die or in other words, there are two chips. It has better performance than dual-core processors in almost all benchmarking tests. They can be easily overclocked up to 4.0 GHz with suitable coolers. The Intel Core 2 Duo processor E8000 and E7000 series are 64-bit processors that maintain compatibility with IA-32 software and are based on the Enhanced Intel Core micro-architecture. These processors use Flip-Chip Land Grid Array (FC-LGA8) package technology, and plug into a 775-land surface mount, Land Grid Array (LGA) socket. These processors are based on 45 nm process technology. The Intel Core 2 Duo processor E8000 series features a 1333 MHz Front Side Bus (FSB) and 6 MB of L2 cache. The Intel Core™ 2 Duo processor E8300 and E7200 were released in April 2008. The Intel Core 2 Duo processor E7600 was developed in June 2009. These processors are used in Internet audio and streaming video, image processing, multimedia, and multitasking user environments. The differences between microprocessors are word length, size of the memory and speed at which the microprocessor can execute instructions. The comparison between different microprocessors is shown in Table 1.1. Figure 1.16 shows the evolution of processors with respect to the year of development and number of transistors in the processor.

Table 1.1 Comparative study of different microprocessors: 4004, 8085, 8086, 80186-80486, Pentium and Core 2 Duo

Microprocessor	No. of Transistors length	Data bus/ Word	Address bus	Memory address range	Clock frequency	Pin	Year of development
4004	2300	4-bit	10-bit	640 B/1 KB	75 kHz	16	1971
8008	3500	8-bit	14-bit	16 KB	0.5–8 MHz	18	1972
8080	6000	8-bit	16-bit	64 KB	2 MHz	40	1974
8085	6500	8-bit	16-bit	64 KB	3–6 MHz	40	1976
8088	29 K	8-bit/16-bit	20-bit	1 MB	5–10 MHz	40	1980
8086	29 K	16-bit	20-bit	1 MB	5–10 MHz	40	1978
80186	29 K	16-bit	20-bit	1 MB	5–16 MHz	68	1982
80286	134 K	16-bit	24-bit	16 MB real 4 GB virtual	6–12.5 MHz	68	1982
80386	275 K	32-bit	24/32-bit	4 GB real 64 TB virtual	20–33 MHz	132	1985
80486	3200 K	32-bit	32-bit	4 GB real 64 TB virtual	25–100 MHz	168	1989
Pentium	3200 K	32-bit	32-bit	4 GB real	60–200 MHz	264	1993
Pentium Pro	5500 K	32-bit	36-bit	64 GB	150–200 MHz	387	1995
Pentium II	7500 K	32-bit	36-bit	64 GB	233–400 MHz	387	1997
Pentium III	9500 K	32-bit	36-bit	64 GB	600–1000 MHz	387	1999
Pentium 4	55000 K	32-bit	36-bit	64 GB	1.3–2 GHz	478	2001
Dual-Core Processor (Athlon)	1.72 billion	64-bit	40-bit	1 TB	2.93 GHz		2007
Core 2 Duo processor E8500	410 million transistors	64-bit	40-bits	1 TB	3.16 GHz	775	2008

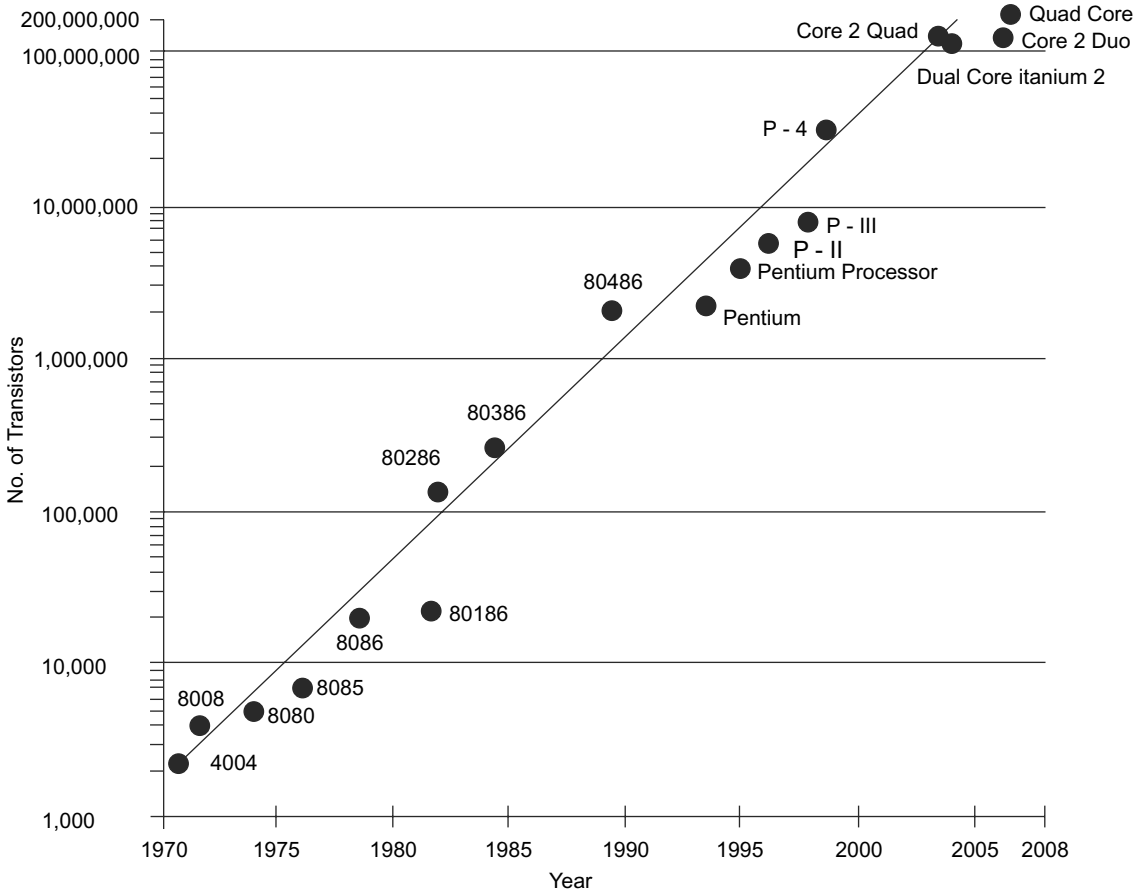


Fig. 1.16 Evolution of microprocessors

1.7 MICROPROCESSOR APPLICATIONS

The microprocessor started as a 4-bit device. It has progressed to an 8-bit, a 16-bit, a 32-bit and now a 64-bit device. A microprocessor with a longer word length will solve more problems at a faster rate. Therefore, a longer word length should give a better and faster solution to all problems. However, the consideration of product cost is important and it has been increased by number of data bits. The applications of 4-bit, 8-bit and other microprocessors are given below:

Applications of 4-bit Microprocessors

The 4-bit microprocessors are very suitable for simple applications, namely, children's toys, calculators, microwave ovens, telephone diallers, etc. The 4-bit microprocessor based system cost is low. Generally, we will find the 4-bit microprocessor in

- ✦ **Toys:** Robots, remote-controlled cars, handheld games
- ✦ **Calculators:** Financial, scientific, database
- ✦ **Power Tool Controllers:** Speed controls, sequencers, measurement devices

- ✦ **Computer Peripherals:** Keyboard scanners, simple printers, clocks
- ✦ **Other Simple Applications:** Microwave ovens, telephone diallers, smart thermostats, shortwave scanners, TV remote controls

Applications of 8-bit Microprocessors

The 8-bit microprocessor-based system is more costly than a 4-bit microprocessor-based system. It is used where memory requirement is large and fast operation is also required. The examples of 8-bit microprocessor applications are given below:

- ✦ **Toys:** Video games, programmable robots
- ✦ **Complex Intelligent Product Controllers:** VCR control and programming, security systems, and lighting system controllers
- ✦ **Computer Peripherals:** Video display, higher-speed printers, modems, plotters, and communication controllers
- ✦ **Industrial Controllers:** Robotics, processing control, sequence control and machine tool control
- ✦ **Instruments:** Logic analyzers, communication analyzers, disk-drive testers, digital oscilloscopes, and smart voltmeters

Applications of 16-bit, 32-bit and 64-bit Microprocessors

The 16-bit, 32-bit and 64-bit microprocessors are used in graphic oriented CAD and CAM systems. The 32-bit and 64-bit microprocessors feature very fast operation, extreme computing power, and megabytes, or even gigabytes, of main memory-addressing space. The list of applications where microprocessors have been already used are illustrated below:

- ✦ **Communications:** Data, voice, mobile, electronic switching, routing
- ✦ **Intelligent Instruments:** CRT terminals, digital multimeters, synthesizers, oscilloscopes and counters. Microcontrollers are used in military equipment, radars, tanks, etc.
- ✦ **Automatic Test Equipment:** Automatic test equipment at all levels from development, fabrication, component testing assembly, PCB, module and system testing
- ✦ **Electrical Power System:** Data acquisition, logging, protection, metering, control and processing, automatic control of generators, voltage and fuel control of furnaces in a power plant
- ✦ **Industrial Process Control:** Instrumentation, monitoring and control, data acquisition, logging and processing
- ✦ **Traffic Control:** Traffic light control for road crossing
- ✦ **Electronic Games:** Various games possible, quizzes and self-teaching
- ✦ **Household Appliances:** Cooking ovens, washing machines and other appliances have started using microprocessors since they replace much electronic hardware at lower cost
- ✦ **Medical Electronics:** Quick patient check-up, diagnosis, blood analysis, ECG, etc.
- ✦ **Database Management:** Computers are used for word processing, database management, storing information, to connect any institution/organisation to other institutions/organisations through Internet

1.8 EVOLUTION OF MICROCONTROLLERS

A microcontroller is a small computer on a single Integrated Circuit (IC) containing a processor, memory, and programmable input/output ports. The program memory, in the form of flash or ROM, is also incorporated on

a chip and a small amount of RAM is also included on a single chip. Microcontrollers are specially designed for embedded applications.

After the innovation of 8080 microprocessors in 1975, Intel Corporation started research on developing an IC which could be used as a microprocessor and should have on-chip data storage. Consequently, Intel developed the first dedicated microcontroller (MCU) chip 8048 IC in 1976. The 8048 IC, was known as MCS-48 microcontroller and it had only 1-byte instructions.

In 1980, Intel had developed an 8-bit microcontroller named the 8051 microcontroller. It had 128 bytes of RAM, 4 K bytes of on-chip ROM, two timers, four parallel ports with each port 8-bits wide and a serial port. This microcontroller had 2-byte instructions. Thereafter, the 8052 microcontroller was developed. This microcontroller had all the standard features of 8051 with an extra 128 bytes of RAM, 4K bytes of ROM and an extra timer. Therefore 8052 had 256 bytes of RAM, 8 K bytes of ROM and three timers. The 8031 microcontroller is also a member of the 8051 family. This microcontroller has all features of 8051 microcontroller except 0 K bytes on-chip ROM. Table 1.2 shows the salient features of 8051, 8052, 8031 and 8032 microcontrollers.

Table 1.2 *Comparative studies of salient features of 8051, 8052, 8031 and 8032 microcontrollers*

Microcontroller IC	ROM (on-chip program memory)	RAM (on chip data memory)	No. of timers	No. of pins in DIP	No. of I/O pins	No. of vector interrupts	Full duplex serial I/O port
8051	4K bytes	128 bytes	2	40	32	5	1
8052	8K bytes	256 bytes	3	40	32	6	1
8031	0K bytes	128 bytes	2	40	32	5	1
8032	0K bytes	256 bytes	3	40	32	6	1

The 8051 microcontroller was developed by incorporating different types of memory such as UV-EPROM, NV-RAM and flash. The UV-EPROM version of the 8051 microcontroller is called the 8751 microcontroller family. The NV-RAM version of the 8051 microcontroller was called DS500 and it was manufactured by Dalas semiconductor. The flash ROM version is known as AT89C51 family microcontroller, and it is manufactured by Atmel corporation. This microcontroller is called Amtel family microcontroller. The Atmel microcontroller family such as AT89CXX, AT89CXX51 are most widely used in industry. The AT 89C51 has 4K bytes of flash ROM and it is extensively used in development of small projects. The most popular Atmel microcontrollers are AT89C51, AT89C52, AT89C1051, AT89C2051, AT89C4051, and AT89LV52 and their features are given in Table 1.3.

Table 1.3 *Comparative studies of salient features of AT89C51, AT89C52, AT89C1051, AT89C2051, AT89C4051, and AT89LV52 microcontrollers*

Microcontroller IC	Flash (on-chip program memory)	RAM (on-chip data memory)	No. of timers DIP	No. of pins in pins	No. of I/O	No. of interrupts	Full duplex serial I/O port	VCC
AT89C51	4K bytes	128 bytes	2	40	32	6	1	5 V
AT89C52	8K bytes	256 bytes	3	40	32	6	1	5 V
AT89C1051	1K bytes	64 bytes	1	20	15	3	1	3 V
AT89C2051	2K bytes	128 bytes	2	20	15	6	1	3 V
AT89C4051	4K bytes	128 bytes	2	20	15	6	1	3 V
AT89LV52	8K bytes	256 bytes	3	40	32	8	1	3 V

The Peripheral Interface Controller (PIC) family of microcontrollers was developed by Microchip in 1985. PIC microcontrollers are based on Harvard architecture and Reduced Instruction Set (RISC). During 1997, 8-bit microcontrollers were introduced by Atmel, based on reduced instruction set. The 8-bit PIC microcontrollers such as PIC16CXX, PIC17CXX were also developed by microchip and manufactured using CMOS Technology. These microcontrollers are extensively used in industry due to their very good performance, low cost and small size. The most commonly used PIC microcontrollers are PIC16C54, PIC16C55, PIC16C56, PIC16C57, PIC16C71, PIC17C42A, PIC17C43, PIC17C44 and PIC17C752 and their features are illustrated in Table 1.4.

Table 1.4 Comparative studies of salient features of PIC16C54, PIC16C55, PIC16C56, PIC16C57, PIC16C71, PIC17C42A, PIC17C43, PIC17C44 and PIC17C752 microcontrollers

Microcontroller IC	EPROM (on-chip program memory)	RAM (on-chip data memory)	No. of instructions DIP	No. of pins in	No. of I/O pins	No. of timers	No. of ADC channels
PIC16C54	512 bytes	25 bytes	33 single-word instructions	18	12	1+ Watchdog timer(WDT)	-
PIC16C55	512 bytes	24 bytes	33 single-word instructions	28	20	1+ Watchdog timer(WDT)	-
PIC16C56	1K bytes	25 bytes	33 single-word instructions	18	12	1+ Watchdog timer(WDT)	-
PIC16C57	2K bytes	72 bytes	33 single-word instructions	28	20	1+ Watchdog timer(WDT)	-
PIC16C71	1K×14 bytes	36 bytes	35 single-word instructions	18	13	1+ Watchdog timer(WDT)	4 channels 8-bit ADC
PIC17C42A	2K bytes	232 bytes	58 single-word instructions	40	33	4	-
PIC17C43	4K bytes	454 bytes	58 single-word instructions	40	33	4	-
PIC17C44	8K bytes	454 bytes	58 single-word instructions	40	33	4	-
PIC17C752	8K×16 bytes	678 bytes	58 single-word instructions	40	33	4	12-channels 10-bit ADC

1.9 APPLICATIONS OF MICROCONTROLLERS

Nowadays microcontrollers are most commonly used in industrial and household applications. The major areas of applications are as follows:

- ✦ Measurement of any physical quantity such as pressure, force, velocity, acceleration, displacement, force, stress, strain, water level
- ✦ Microcontroller-based laboratory instruments to measure voltage, current, phase angle, power factor, frequency, resistance, power, and energy, etc.
- ✦ Robot-arm position control
- ✦ Angular speed measurement

- ✦ Temperature measurement
- ✦ dc motor and stepper motor control
- ✦ Induction motor control
- ✦ Traffic light control system
- ✦ Automobile applications
- ✦ Household appliances such as washing machine, light control, camera, TV, VCR and video games, etc.
- ✦ Office equipments such as photocopying machines, telephones, fax machines, printers, and security system, etc.

SUMMARY

- In this chapter, the microprocessor and microcontroller are properly defined.
- The basic microprocessor architecture and operation of its components are discussed.
- The genesis and evolution of microprocessors from 4004 processor to Pentium and Core 2 Duo processor E8500 are explained elaborately. The comparative study of different microprocessors: 4004, 8085, 8086, 80186-80486, Pentium and Core 2 Duo have been presented in tabular form.
- The evolution of microcontrollers are explained briefly and comparative studies of salient features of 8051 and its derivatives AT89C51, AT89C52, AT89C1051, AT89C2051, AT89C4051, and AT89LV52 microcontrollers and PIC family microcontrollers are presented in tabular form.
- The applications of microprocessors and microcontrollers are also discussed in this chapter.

MULTIPLE-CHOICE QUESTIONS

- | | |
|---|--|
| <p>1.1 The first microprocessor was</p> <p>(a) 4001 (b) 4002
(c) 4003 (d) 4004</p> <p>1.2 The 64-bit processor is</p> <p>(a) Pentium (b) Pentium II
(c) Pentium III (d) Pentium 4</p> <p>1.3 The memory capacity of 8085 microprocessor is</p> <p>(a) 64K (b) 1 MB
(c) 16 MB (d) 640 B</p> <p>1.4 The address bus of 80186 microprocessor is</p> <p>(a) 16-bit (b) 20-bit
(c) 24-bit (d) 32-bit</p> | <p>1.5 The operating frequency of 8086 microprocessor is about</p> <p>(a) 750 KHz (b) 3–6 MHz
(c) 5–10 MHz (d) 3-6 GHz</p> <p>1.6 The data bus of Pentium II and Pentium III processors is</p> <p>(a) 16-bit (b) 20-bit
(c) 24-bit (d) 32-bit</p> <p>1.7 The first electronic computer was</p> <p>(a) ENIAC (b) EDVAC
(c) EDSAC (d) Difference Engine</p> <p>1.8 More than ten thousand transistors are exist in</p> <p>(a) LSI ICs (b) MSI ICs
(c) SSI ICs (d) VLSI ICs</p> |
|---|--|

- 1.9 The memory capacity of a Pentium Pro microprocessor is
 (a) 64 KB (b) 64 MB
 (c) 64 GB (d) 640 B
- 1.10 Which of the following processors has an in built math processor?
 (a) 8086 (b) Pentium-4
 (c) 8085 (d) 8088
- 1.11 A microcontroller has
 (a) ROM (b) RAM
 (c) I/O ports (d) all of these
- 1.12 A general-purpose microprocessor requires which of the following devices to operate properly?
 (a) ROM (b) RAM
 (c) I/O ports (d) All of these
- 1.13 The 8051 microcontroller is a ____ bit processor
 (a) 4-bit (b) 8-bit
 (c) 16-bit (d) 32-bit
- 1.14 The 8052 microcontroller has
 (a) 20 pins for I/O (b) 32 pins for I/O
 (c) 35 pins for I/O (d) 40 pins for I/O

SHORT-ANSWER-TYPE QUESTIONS

- 1.1 What is the first microprocessor? Which company built that microprocessor?
- 1.2 Define SSI, MSI, LSI and VLSI.
- 1.3 Define microprocessor, microcontroller and microcomputer.
- 1.4 Write the features of 8051 and 8052 microcontrollers.
- 1.5 List the components of a microprocessor and microcomputer.

REVIEW QUESTIONS

- 1.1 Write the difference between microprocessor and microcomputer.
- 1.2 Explain briefly the genesis of microprocessors.
- 1.3 What is ALU? Explain the following terms: Registers, Control unit, and Input and Output devices.
- 1.4 Draw the architecture of a microcomputer and explain briefly.
- 1.5 Write the comparison between the following processors
 (i) 8085 and 8086 (ii) 80286 and 80486 (iii) Pentium II and Pentium 4
- 1.6 Give a list of applications of the following processors
 (i) 4-bit processors (ii) 8-bit processors (iii) 16-bit processors
- 1.7 Discuss the evaluation of microprocessors and microcontrollers.
- 1.8 What is the major difference between 8051 and 8052 microcontrollers?
- 1.9 List the applications of microcontrollers.

Answers to Multiple-Choice Questions

- 1.1 (d) 1.2 (d) 1.3 (a) 1.4 (b) 1.5 (c) 1.6 (d) 1.7 (a) 1.8 (a) 1.9 (c)
 1.10 (b) 1.11 (d) 1.12 (d) 1.13 (b) 1.14 (b)

Chapter 2

Architecture of the 8085 Microprocessor

2.1 INTRODUCTION

The Intel 8085/8085AH is a microprocessor, i.e., an 8-bit parallel central processing unit implemented in silicon gate NMOS/HMOS/C-MOS technology. It is available in a 40-pin IC package fabricated on a single LSI chip. It is designed with higher processing speed, ranging from 3 MHz to 5 MHz. Lower power consumption and power-down mode is provided, thereby offering a high level of system integration. This processor uses a multiplexed address/data bus. The address bus is split between the 8-bit address bus and the 8-bit data bus. The on-chip address latch allows a direct interface with the processor. The features of 8085 microprocessors are given below:

Features

- ◆ Power-down mode (HALT-HOLD)
- ◆ Low power dissipation: about 50 mW
- ◆ Single +3 to +6 V power supply
- ◆ Operating temperature from -40 to $+85^{\circ}\text{C}$
- ◆ On-chip clock generator incorporating external crystal oscillators
- ◆ On-chip system controller
- ◆ Four-vectored interrupt including one non-maskable
- ◆ Serial input/Serial output port
- ◆ Addressing capability to 64K bytes of memory
- ◆ TTL compatible
- ◆ Available in 40-pin plastic DIP package

2.2 BLOCK DIAGRAM OF THE 8085 MICROPROCESSOR

The functional block diagram of Intel 8085 is depicted in Fig. 2.1. It consists of three main sections: an arithmetic and logic unit, timing and control unit and a set of registers. These important sections are described in the subsequent pages.

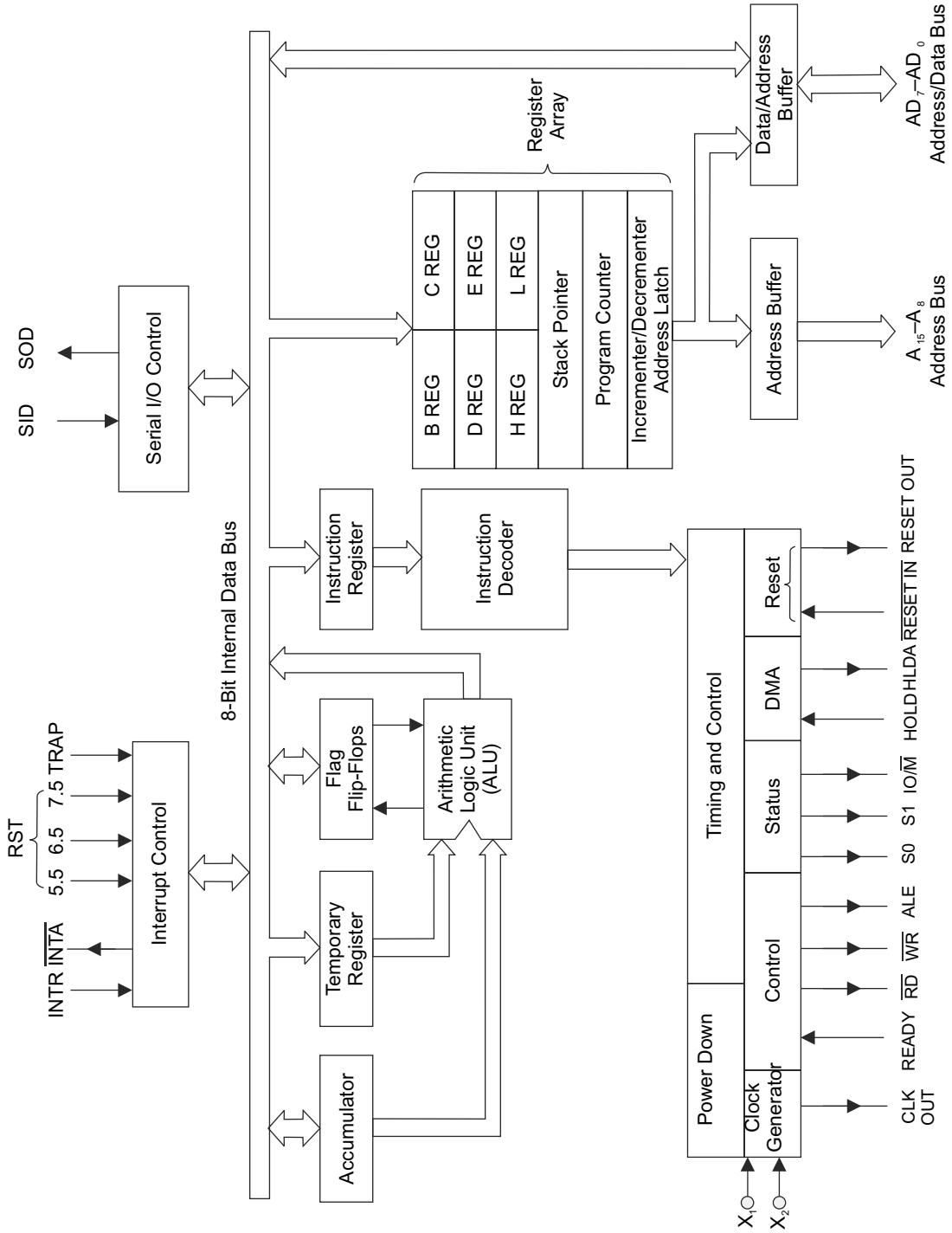


Fig. 2.1 Architecture of 8085 Microprocessor

2.2.1 Operation of 8085 Microprocessor

Generally, a microprocessor performs four different operations: memory read, memory write, input/output read and input/output write. In the memory read operation, data will be read from memory and in the memory write operation, data will be written in the memory. Data input from input devices are I/O read and data output to output devices are I/O write operations.

The memory read/write and Input/Output read and write operations are performed as part of communication between the microprocessor and memory or Input/Output devices. Microprocessors communicate with the memory, and I/O devices through address bus, data bus and control bus as depicted in Fig. 2.2. For this communication, firstly the microprocessor identifies the peripheral devices by proper addressing. Then it sends data and provides control signal for synchronization.

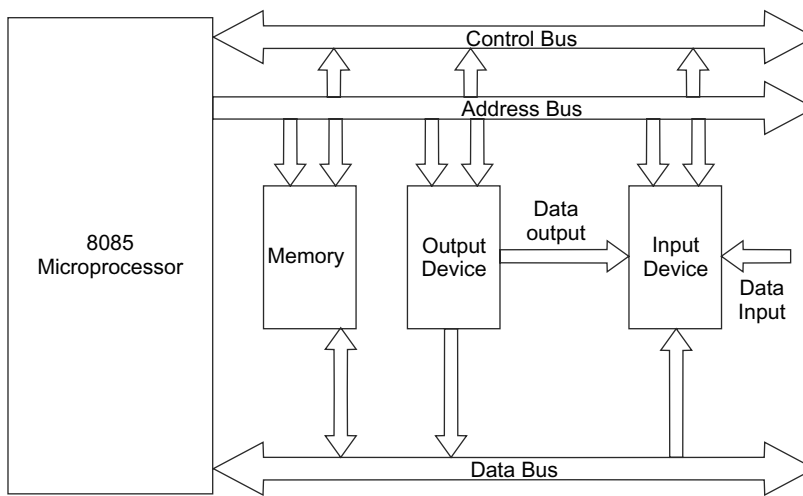


Fig. 2.2 Bus structure of 8085 microprocessor

Figure 2.3 shows the memory read operation. Initially, the microprocessor places a 16-bit address on the address bus. Then the external decoder logic circuit decodes the 16-bit address on the address bus and the memory location is identified. Thereafter, the microprocessor sends \overline{MEMR} control signal which enables the memory IC. After that, the content of the memory location is placed on the data bus and also sent to the microprocessor. Figure 2.4 shows the data flow diagram for data transfer from the memory to microprocessor. The step-by-step procedure of data flow is given below:

- ◆ The 16-bit memory address is stored in the program counter. Therefore, the program counter sends the 16-bit address on the address bus. The memory address decoder is decoded and identifies the specified memory location.
- ◆ The control unit sends the control signal \overline{RD} in the next clock cycle and the memory IC is enabled. \overline{RD} is active for two clock periods.
- ◆ When the memory IC is enabled, the byte from the memory location is placed on the data bus AD_7-AD_0 . After that data is transferred to the microprocessor.

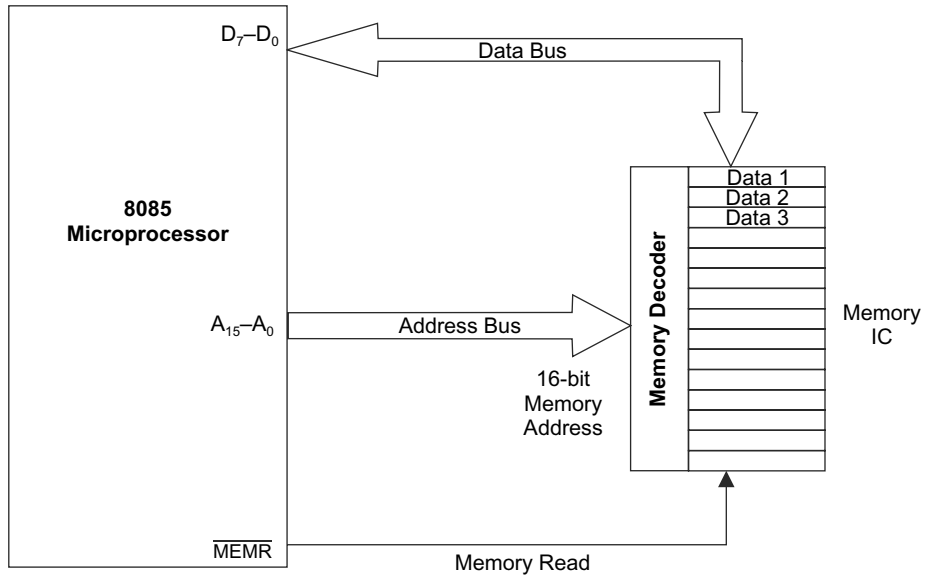


Fig. 2.3 Memory read operation

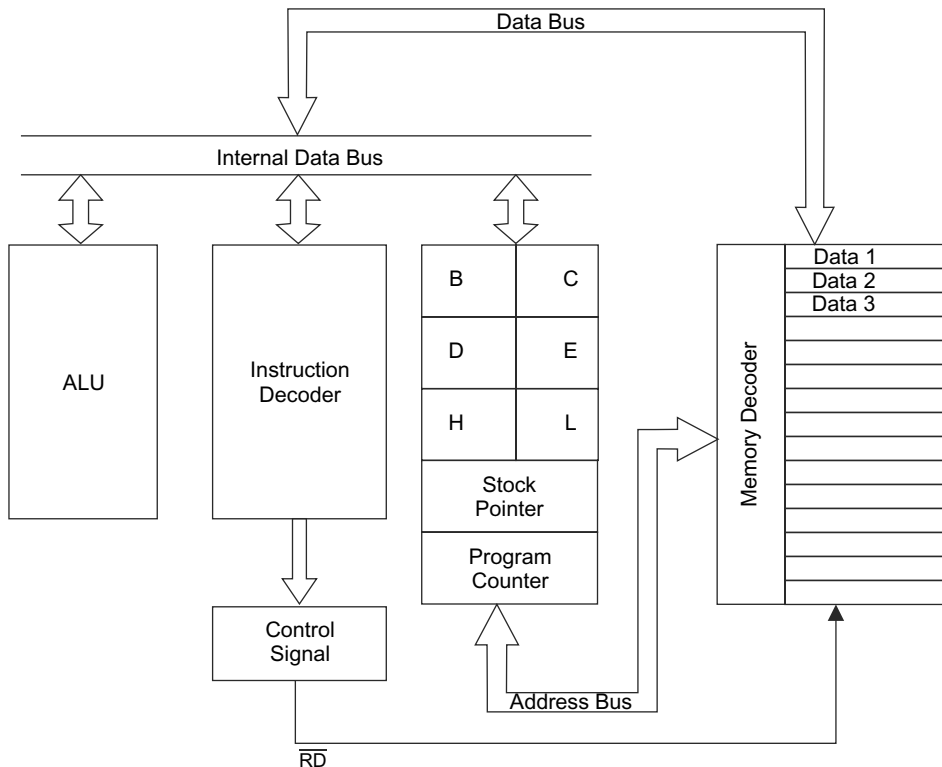


Fig. 2.4 Data flow from memory to microprocessor

2.2.2 Arithmetic Logic Unit (ALU)

All arithmetic and logical operations are performed in the Arithmetic Logic Unit (ALU). The functioning of the ALU is given in Fig. 2.5. The ALU functioning consists of Accumulator (A), Temporary Register (TR), Flag Register (FR) and arithmetic logic unit. The temporary register is not accessible to the user. Therefore, the user cannot read the content of TR. Actually, this register is used to store or load the operand during

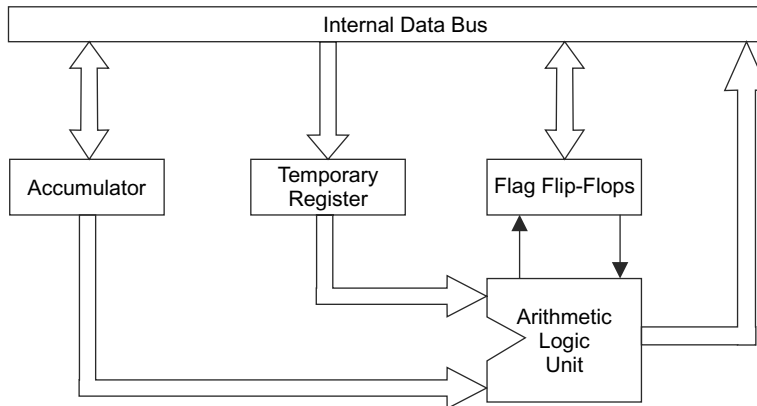


Fig. 2.5 ALU functioning

arithmetic and logical operations. Accumulator, TR and flag register are explained in Section 2.2.5 in detail. The ALU always operates with one or two operands. Generally, operands are available in general-purpose registers or memory locations. The results after arithmetic and logical operations are stored in the accumulator. The sequence of operations in ALU are given below:

- (i) One operand is in the A register.
- (ii) The other operand may be in the general-purpose register or memory location, which will be transferred to the temporary register.
- (iii) Then contents of the accumulator and temporary registers are considered as inputs of ALU and the specified operation is carried out in the ALU.
- (iv) The result of ALU operation is transferred in the A register through internal data bus.
- (v) The content of the flag register will be changed depending on the result.

The arithmetic logic unit (ALU) performs the following operations:

- ◆ Addition
- ◆ Subtraction
- ◆ Logical AND
- ◆ Logical OR
- ◆ Logical EXCLUSIVE OR
- ◆ Complement
- ◆ Increment by 1
- ◆ Decrement by 1
- ◆ Rotate Left, Rotate Right
- ◆ Clear

2.2.3 Timing and Control Unit

The control unit controls the operations of different units while the CPU generates timing sequence signals for the execution of instructions. This unit controls the data flow between CPU and memory and CPU and peripheral devices. This unit provides control, status, DMA and reset signals to perform any memory and input–output related operations. Actually, it controls the entire operation of microprocessors. Therefore, the timing and control unit acts as the brain of the microprocessor.

2.2.4 Registers

The Intel 8085 has six general-purpose registers to store 8-bit data and these registers are identified as B, C, D, E, H and L. When two registers are combined, 16-bit data can be stored in a register pair. The only possible combinations of register pairs are BC, DE and HL. These register pairs are used to perform 16-bit operations. There is an accumulator register and one flag register. The accumulator is an 8-bit register. Arithmetic and logical operations are performed in the accumulator and after operation, the result will be stored in the accumulator. In addition with the above registers, there are two 16-bit registers, namely, the Stack Pointer (SP) and Program Counter (PC). The following registers of Intel 8085 microprocessor have been depicted in Fig. 2.1:

- ◆ One 8-bit accumulator (ACC) known as register A
- ◆ Six 8-bit general-purpose registers: B, C, D, E, H and L
- ◆ One 16-bit Stack Pointer (SP)
- ◆ One 16-bit Program Counter (PC)
- ◆ Instruction register
- ◆ Temporary register
- ◆ Program Status Word (PSW) Register

Accumulator The accumulator is an 8-bit register, which is part of the Arithmetic Logic Unit (ALU). This is identified as register A or ACC. It is used to store 8-bit data and to perform arithmetic as well as logic operations. The final result of an operation performed in the ALU is also stored in the accumulator.

General-Purpose Registers The general-purpose registers of the 8085 microprocessor are B, C, D, E, H and L registers as shown in Fig. 2.6. These registers are used to store 8-bit operands. To hold a 16-bit data or 16-bit memory address location, two 8-bit registers can be combined. The combination of two 8-bit registers is known as a *register pair*. The only possible combination register pairs of the 8085 microprocessor are B-C, D-E and H-L. The programmer cannot form a register pair by selecting any two registers of his choice. The H-L register pair can be used as the address of memory location whereas B-C and D-E register pairs are used to store 16-bit data. During the execution of the program, all general-purpose registers can be accessed by program instructions and also used for data manipulation.

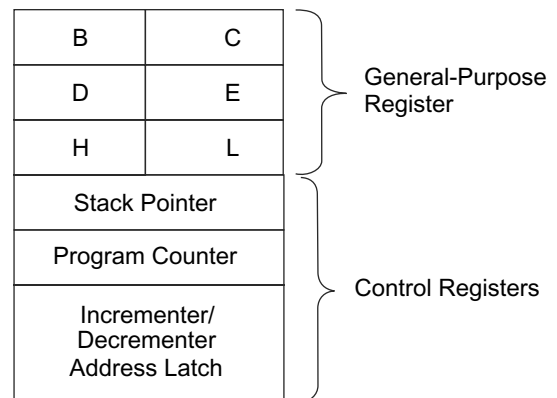


Fig. 2.6 Registers of 8085 microprocessor

Special-Purpose Registers

In addition to the above general-purpose registers, the 8085 microprocessor has special-purpose registers, namely, Program Counter (PC), Stack Pointer (SP), Flags/Status Registers (SR), Instruction Register (IR), Memory Address Register (MAR), Temporary Register (TR), and Memory Buffer Register (MBR).

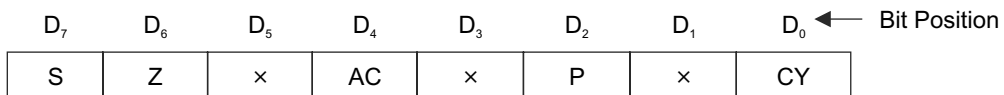
✓ **Program Counter (PC)** The program counter is a 16-bit special-purpose register. This is used to hold the memory address of the next instruction which will be executed. Actually, this register keeps track of memory locations of the instructions during execution of program. The microprocessor uses this register to execute instructions in sequence. For this, the microprocessor increments the content of the program counter.

✓ **Stack Pointer (SP)** The stack pointer is a 16-bit register, which is used to point the memory location called the stack. The stack is a sequence of memory locations in the R/W memory. The starting of the stack is defined by loading a 16-bit address into the stack pointer. Generally, the programmers use this register to store and retrieve the contents of the accumulator, flags, program counter as well as general-purpose registers during the execution of a program. The organization and application of stacks are incorporated in Chapter 4.

✓ **Flags/Status Registers (SR)** The Arithmetic Logic Unit (ALU) includes five flip-flops, which are set or reset after an ALU operation according to data conditions of the result in the accumulator and other general-purpose registers. The status of each flip-flop is known as a flag. Therefore, there are five flags, namely, Carry flag (CY), Parity flag (P), Auxiliary Carry flag (AC), Zero flag (Z), and Sign (S) flags. The most commonly used flags are Carry(CY), Zero(Z) and Sign(S). Generally, the microprocessor uses these flags to test data conditions.

For example, after addition of two 8-bit numbers, if the sum in the accumulator is larger than eight bits, the flip-flop, which is used to indicate a carry, is set to one. So the Carry flag (CY) is set to 1. If the result is zero after any arithmetic operation, the Zero (Z) flag is set to one.

Figure 2.7 shows an 8-bit register, which indicates bit positions of different flags. This register is known as *flag register* and it is adjacent to the accumulator. Though it is an eight-bit register, only five bit positions out of eight are used to store the outputs of the five flip-flops. The flags are stored in the 8-bit register so that the programmer can check these flags through an instruction. These flags are used in the decision-making process of the microprocessor.



S — Sign Flag

Z — Zero Flag

AC — Auxiliary Carry Flag

P — Parity Flag

CS — Carry Flag

Fig. 2.7 Flag register

(i) **Carry Flag (CY)** The arithmetic operation generates a carry in case of addition or a borrow in case of subtraction after execution of an arithmetic instruction and the carry flag is set to 1. When the two 8-bit

numbers are added and the sum is larger than 8 bits, a carry is produced and the carry flag is set to 1. During subtraction, if borrow is generated, the carry flag is also set to 1. The position of carry flag is D_0 as depicted in Fig. 2.7.

(ii) **Parity Flag (P)** After an arithmetic or logical operation, if the number of 1s in the result is even (even parity), this parity status flag (P) is set, and if the number of 1s is odd (odd parity), this flag is reset. For example, if the data byte is 1 1 1 1 1 1 1, the number of 1s in the data byte is eight (even parity) and the parity flag (P) is set to 1. The position of the parity flag is D_2 as shown in Fig. 2.7.

(iii) **Auxiliary Carry Flag (AC)** In arithmetic operations of numbers, if a carry is generated by bit D_3 and passed on to D_4 , the auxiliary carry flag (AC) is set. Actually this flag is used for internally Binary Coded Decimal (BCD) operations and this is not available for the programmer to change the sequence of operations through jump instructions. The position of auxiliary carry flag is D_4 as given in Fig. 2.7.

(iv) **Zero Flag (Z)** When an 8-bit ALU operation results in zero, the Zero (Z) flag is set; otherwise it is reset. This flag is affected by the results of the accumulator and general-purpose registers.

(v) **Sign Flag (S)** The sign flag has its importance only when a signed arithmetic operation is performed. In arithmetic operations of signed numbers where the bit D_7 is used to indicate a sign, this flag is set to indicate the sign of a number.

The most significant bit of an 8-bit data is the sign bit. When a number is negative, the sign bit is 1. If the number is positive, the sign bit is 0. For an 8-bit signed operation, the remaining 7 bits are used to represent the magnitude of a number. After execution of a signed arithmetic operation, the MSB of the result also represents its sign. The position of the sign flag is D_7 as depicted in Fig. 2.7.

(vi) **PSW** In a flag register five bits ($D_7 D_6 D_4 D_2 D_0$) indicate the five status flags and three bits $D_5 D_3$ and D_1 are undefined. The combination of these 8 bits is known as Program Status Word (PSW). The PSW and the accumulator can be used as a 16-bit unit for stack operation.

Example 2.1

Determine the status of different flags after addition of 07H and CFH.

Solution

When 07H and CFH are added, the result is non zero. The Z flag is set to 0. There is a carry from 3rd bit to 4th bit. Therefore, the auxiliary carry (AC) flag is set to 1. As the MSB of the sum is 1, the S flag is set to 1. Since there are five numbers of 1s in the result, the parity flag (P) is set to 0. Figure 2.8 shows the status of different flags after addition of 07H and CFH.

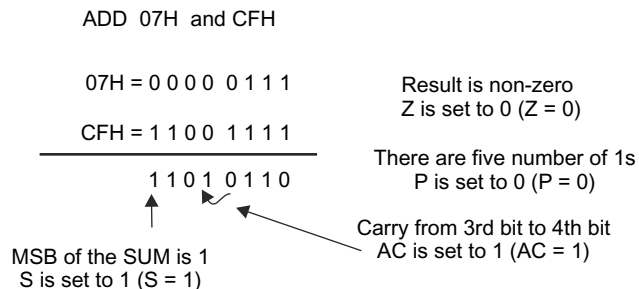


Fig. 2.8 Status of different flags after addition of 07H and CFH

Example 2.2

Find out the status of different flags after addition of CEH and 9BH

Solution

If CEH and 9BH are added, the result is non zero. Hence, the Z flag is set to 0. There is a carry from 3rd bit to 4th bit. As a result, the auxiliary carry (AC) flag is set to 1. Since the MSB of the sum is 0, the S flag is set to 0. As there are four numbers of 1s in the result, the parity flag (P) is set to 1. The carry is generated after addition of CEH and 9BH as the sum is greater than 8 bits. Therefore, CS is set to 1. Figure 2.9 shows the status of different flags after addition of CEH and 9BH.

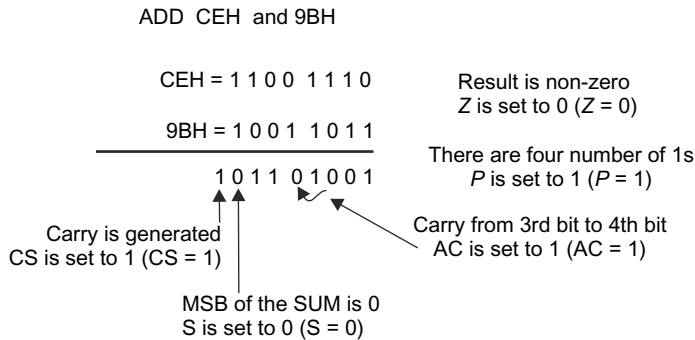


Fig. 2.9 Status of different flags after addition of CEH and 9BH

- ✓ **Instruction Register (IR)** The instruction register holds the operation code (opcode) of the current instruction of a program during an arithmetic/logical operation. The instruction is fetched from the memory prior to execution. The decoder takes the instruction and decodes it. After that, the decoded instruction is passed to the next stage for execution.
- ✓ **Memory Address Register (MAR)** The Memory Address Register holds the address of the next program instruction. Then MAR feeds the address bus with addresses of the memory location of the program instruction which will be executed.
- ✓ **Temporary Register (TR)** This is an 8-bit register, which is associated with ALU. This register holds data during arithmetic and logical operation. This register can be used by the microprocessor but is not accessible to the programmer.

2.2.5 System Bus: Address Bus, Data and Control Bus

The system bus is collection of wires which are used to transfer binary numbers, one bit per wire. The 8085 microprocessor communicates with memory and input and output devices using three buses, namely, address bus, data bus and control bus, as depicted in Fig. 2.10.

Address Bus

Each memory location has a unique address. The address bus consists of 16 wires, therefore address bus has 16 bits. Its 'width' is 16 bits. A 16-bit binary number allows 2^{16} different

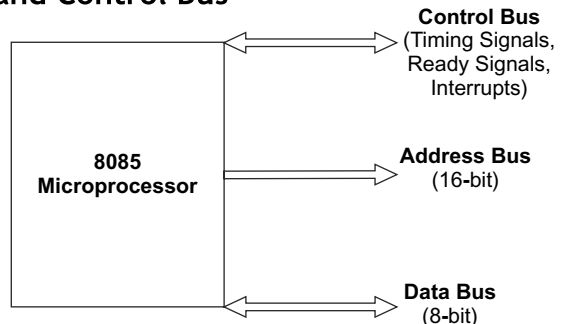


Fig. 2.10 Microprocessor and its buses

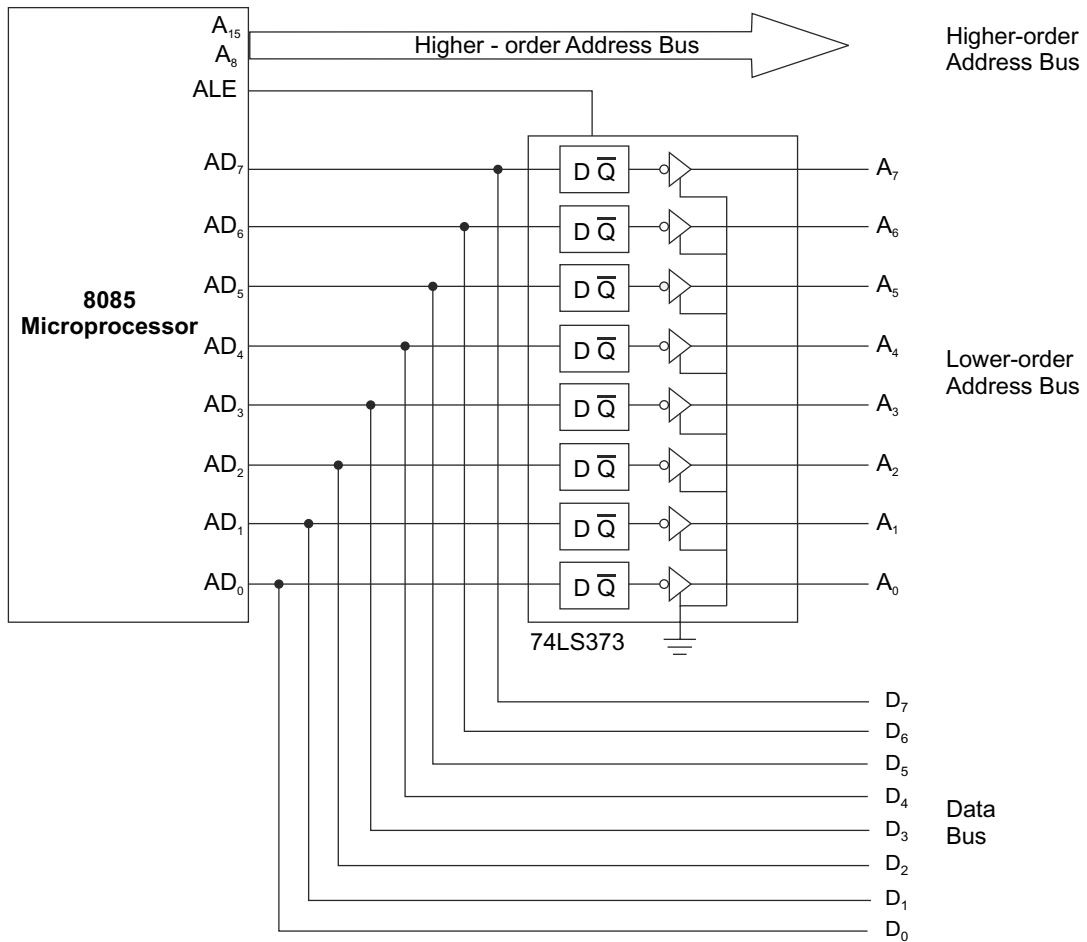


Fig. 2.11 Multiplexing of lower-order address bus

numbers, or 65536 different numbers, i.e., 0000 0000 0000 0000 up to 1111 1111 1111 1111. Therefore, the Intel 8085 microprocessor has $65536 = 64K$, where $1K = 1024$) memory for locations and each memory location contains 1 byte of data. The address bus is unidirectional. That means numbers can only be sent from microprocessor to memory, and not the other way. The 16-bit address bus consists of the 8 most significant bits of the address A_{15} – A_8 and the 8 least significant bits of the address/data AD_7 – AD_0 . A_7 – A_0 is multiplexed with the data lines D_0 – D_7 . During the first clock period of the machine cycle, the microprocessor sends 8 MSBs of the address on the A bus and 8 LSBs of the address on the AD bus. If the data is sent during the first clock period, it will be latched. After the first clock pulse, AD lines as data bus is shown in Fig. 2.11 and the timing diagram is depicted in Fig. 2.12.

Data Bus

Data bus as 8-bit data is stored in each memory location. The data bus is used to move or transfer data in binary form. The data is transferred between the microprocessor and external devices. In the 8085 microprocessor, the data size is 8 bits. Consequently, the data bus typically consists of 8 wires. Therefore, 2^8 combinations of binary digits are possible. Data bus is used to transmit 'data', i.e., information,

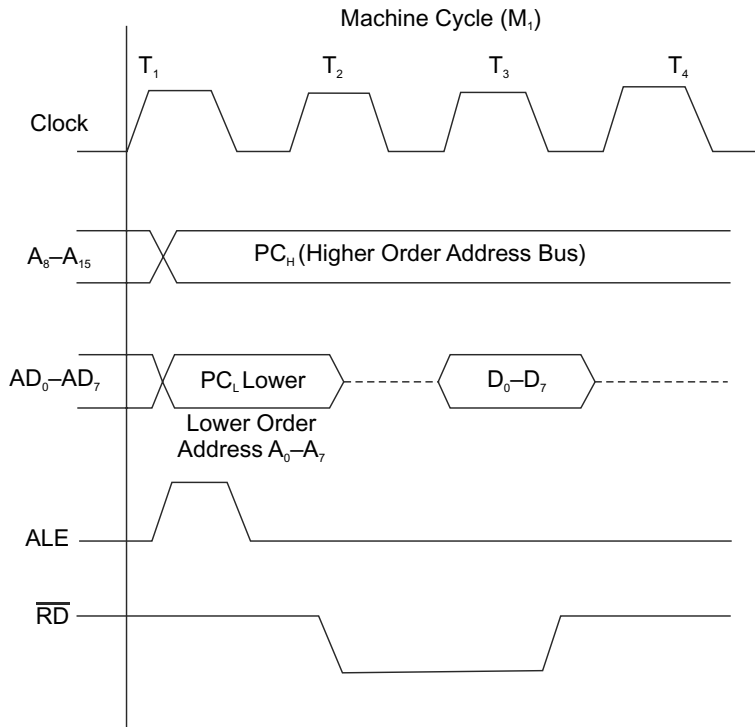


Fig. 2.12 Time multiplexing of Address Bus

results of arithmetic, etc, between the memory and the microprocessor. This bus is bi-directional. Size of the data bus determines what arithmetic can be done. As a data bus is 8 bits wide, the largest number is 11111111 (255 in decimal).

The address/data bus sends data and addresses at different instant of time. Therefore, it transmits either data or an address at a particular moment. The AD-bus always operates in the time-shared mode.

Control Bus

The control bus has various lines which have specific functions for coordinating and controlling microprocessor operations. For example, the RD/ \overline{WR} line is a control signal and this is also a single binary digit. This signal can differentiate the read and write operations. When RD is logically '1', memory and other input output devices are read. If \overline{WR} is logically '0', data can be written in memory and any other devices. Various other control signals are used to control and coordinate the operation of the system. Typically, 8085 microprocessor has 11 control lines, namely, S₀, S₁, IO/ \overline{M} , RD, \overline{WR} , ALE, READY, HOLD, HLDA, $\overline{RESET\ IN}$ and RESET OUT. The microprocessor cannot function correctly without these vital control signals. The control bus carries control signals, partly unidirectional, partly bi-directional.

2.3 PIN DIAGRAM OF 8085 MICROPROCESSOR

Figure 2.13 shows the schematic diagram of Intel 8085. The PIN diagram of the 8085 microprocessor is illustrated in Fig. 2.14. The descriptions of various PINS are as follows:

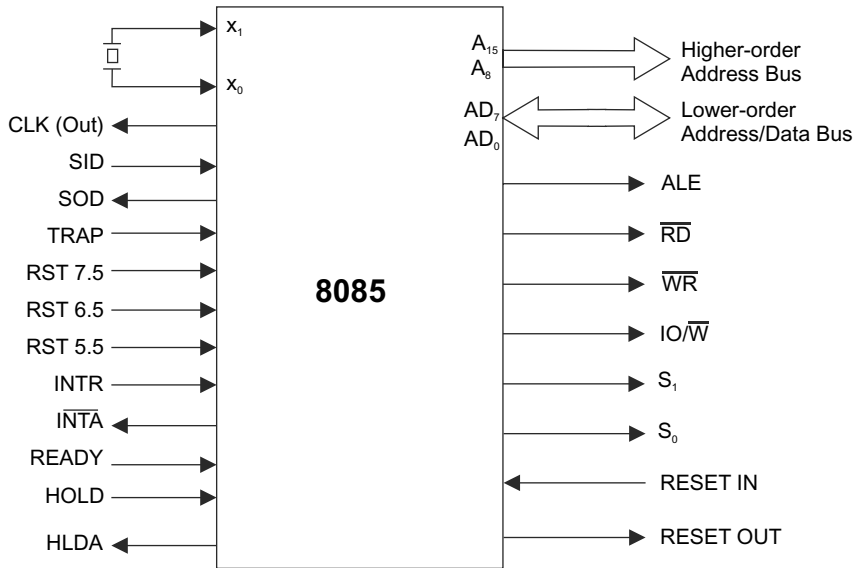


Fig. 2.13 The schematic diagram of Intel 8085

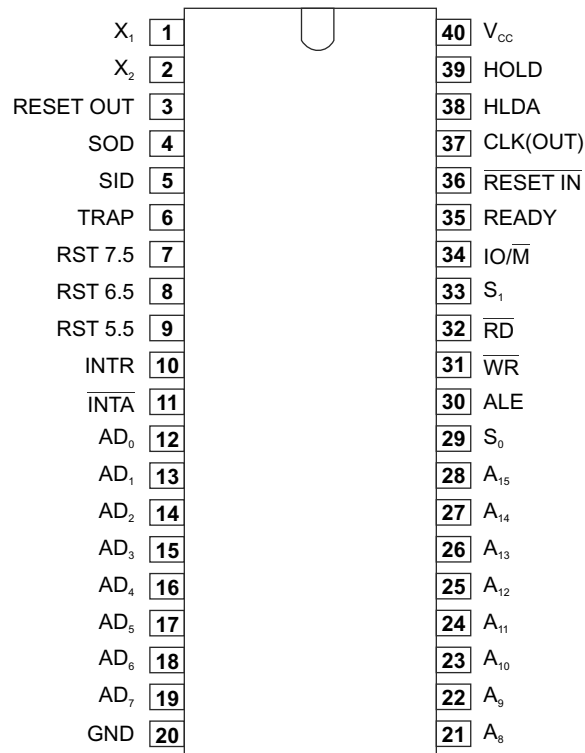


Fig. 2.14 PIN diagram of Intel 8085

Table 2.1 (a) Status codes and states of 8085

Machine cycle status			States
IO/ \overline{M}	S_1	S_0	
0	0	1	Memory write
0	1	0	Memory read
1	0	1	I/O write
1	1	0	I/O read
0	1	1	Opcode fetch

Table 2.1 (b) Status codes and states of 8085

Machine cycle status			States
IO/ \overline{M}	S_1	S_0	
1	1	1	Interrupt Acknowledge
x	0	0	Halt
x	x	x	Hold
x	x	x	Reset

A_{15} – A_8 (Output, 3-state) These are address buses. These are used for the most significant 8 bits of the memory address or 8 bits of I/O address, 3-stated during Hold and Halt modes and during RESET.

AD_7 – AD_0 (Input/Output, 3-state) These are multiplexed address/data buses. These lines serve a dual purpose. These are used for the least significant 8 bits of the memory address or I/O during the first clock cycle (T state) of a machine cycle. After that it becomes the data bus during the second and third clock cycles.

The address on the higher-order bus remains on the bus for the entire machine cycle. But the lower-order address is changed after the first clock cycle. Actually, this address is latched and used for identifying the memory address. The AD_7 – AD_0 is used to identify the memory location. Figure 2.9 shows the address bus A_{15} to A_0 after latching operation. When ALE signal is low, the data is latched till the next ALE signal. The output of the latch represents the lower-order address bus A_7 – A_0 . If ALE is high, the latch is transparent. This means that the output changes according to input data.

ALE (output) Address Latch Enable ALE stands for Address Latch Enable. During the first clock state of a machine cycle, it becomes high and enables the address to get latched either into the memory or external latch. The falling edge of ALE is set to guarantee set-up, can hold times for the address information. The falling edge ALE can also be used to strobe the status information.

S_0 , S_1 , IO/ \overline{M} (Output) These are machine-cycle status signals sent by the microprocessor to distinguish the various types of operations given in Table 2.1. IO/ \overline{M} , S_0 and S_1 become valid at the beginning of a machine cycle and remain stable throughout the cycle. IO/ \overline{M} signals differentiate whether the address is for memory or input–output devices. When IO/ \overline{M} becomes high, I/O operation is performed. It is low for memory operations. When this signal is combined with \overline{RD} and \overline{WR} , this signal transfers the CPU data into I/O or memory devices.

\overline{RD} (Output, 3-state) Read Memory or I/O Devices It is a READ control signal. When RD is low level, the selected memory or I/O device to be read is available in the data bus for the data transfer. It has 3-stated during Hold and Halt modes and during RESET.

\overline{WR} (Output, 3-state) Write Memory or I/O Devices The \overline{WR} signal is used for WRITE control operation. The low level on \overline{WR} indicates the data on the data bus to be written into the selected memory or I/O location. It has 3-stated during Hold and Halt modes and during RESET.

Figure 2.15 shows the generation of four different control signals by combining \overline{RD} , \overline{WR} and IO/ \overline{M} signals. The signal IO/ \overline{M} is low for any memory-related operation. The IO/ \overline{M} is logically ANDed with

\overline{RD} , \overline{WR} signals and generates memory read \overline{MEMR} and memory write \overline{MEMW} control signals. If IO/\overline{M} becomes high, then Input/Output peripheral operations. It is depicted in Fig. 2.15 that the signal is ANDed with \overline{RD} , \overline{WR} signals and generate I/O read \overline{IOR} and I/O write \overline{IOW} control signals for any I/O related operation.

READY (Input)

When READY is high during a read or write operation, it indicates that the memory or I/O device is ready to send or receive data. When READY is low, the CPU will wait for the number of clock cycles until READY becomes high.

HOLD (Input)

HOLD indicates that another master is requesting the use of the address and data buses. After receiving the hold request, the CPU will relinquish the use of the bus as soon as the completion of the current bus cycle.

The processor can regain the bus only after the HOLD is removed. When the HOLD is acknowledged, the address, data, RD, WR, and IO/M lines are 3-stated.

HLDA (Output)

HLDA stands for HOLD ACKNOWLEDGE. This signal indicates that the CPU has received the Hold request and that it will relinquish the bus in the next clock cycle. When the Hold request is removed, HLDA goes low. The CPU takes the bus one half-clock cycle after HLDA goes low.

INTR (Input)

INTR is the INTERRUPT REQUEST signal. It is used as general-purpose interrupt. Among interrupts, it has the lowest priority. It is sampled only during the next to the last clock cycle of an instruction. If it is active, the Program Counter (PC) will be inhibited from incrementing and an Interrupt Acknowledge (INTA) signal will be issued. The microprocessor suspends its normal sequence of instructions. During this cycle, a RESTART or CALL instruction can be inserted to jump to the interrupt service routine. The INTR can be enabled and disabled by using software. It is disabled by RESET and immediately after an interrupt is accepted. Generally, the interrupt signal is used by I/O devices to transfer data to the microprocessor without wasting its time.

\overline{INTA} (Output)

It is an interrupt acknowledge signal. This is used instead of \overline{RD} during the instruction cycle after an INTR is accepted. This signal is sent by the microprocessor after INTR is received. It can be used to activate the 8259 interrupt IC.

RST 5.5, RST 6.5, and RST 7.5

RST 5.5, RST 6.5 and RST 7.5 are the restart interrupts. These three inputs have the same timing as INTR except they cause an internal restart to be automatically inserted. The priority order of these interrupts is given in Table 2.2. These interrupts have a higher priority than INTR. These are vectored interrupts and during execution, transfer the program to the specified memory location.

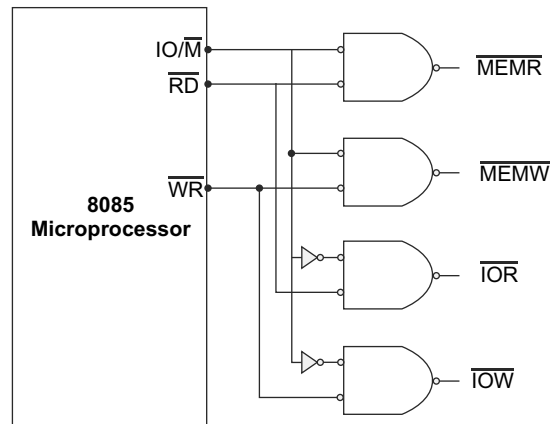


Fig. 2.15 Generation of Memory and I/O Read/Write control signals

Table 2.2 Interrupt priorities and restart address

Name	Priority	Address branched to memory location when interrupt occurs
TRAP	1	0024H
RST 7.5	2	003CH
RST 6.5	3	0034H
RST 5.5	4	002CH
INTR	5	The address branched depending on the instruction provided to the CPU when the interrupt is acknowledged

TRAP (Inputs)

Trap interrupt is a nonmaskable restart interrupt. It has the highest priority of any interrupt as depicted in Table 2.2. It is recognized at the same timing as INTR or RST 5.5 or RST 6.5 or RST 7.5. It is unaffected by any mask or Interrupt enable.

 $\overline{RESETIN}$ (Input)

The $\overline{RESETIN}$ signal resets the program counter to zero and it also resets the Interrupt Enable and HLDA flip-flops. It does not affect any other flag or register except the instruction register. The data and address buses and the control lines are 3-stated during $\overline{RESETIN}$ and because of the asynchronous nature of $\overline{RESETIN}$, the processor's internal registers and flags may be altered by RESET with unpredictable results. $\overline{RESETIN}$ is a Schmitt-triggered input, allowing connection to an RC network for power-on RESET delay. The CPU is held in the reset condition as long as $\overline{RESETIN}$ is applied.

 $\overline{RESETOUT}$ (Output)

$\overline{RESETOUT}$ indicates that the CPU is in \overline{RESET} condition. This can be used as a system reset. This signal is also synchronized to the processor clock and lasts an integral number of clock periods.

 X_1, X_2 (Input)

The X_1 and X_2 terminals are connected to a crystal or RC network or LC network to drive the internal clock generator. X_1 may be an external clock input from a logic gate. The input frequency is divided by 2 to give the processor's internal operating frequency. When 6 MHz clock frequency of a crystal or RC network or LC network is applied to the processor, the microprocessor operates in 3 MHz.

CLK (Output)

The Clock output signal can be used as a system clock. The time period of CLK is twice the X_1, X_2 input time period.

SID (Input)

SID stands for Serial Input Data line. The data on this line is loaded into the accumulator bit 7 whenever a RIM instruction is executed.

SOD (Output)

SOD stands for Serial Output Data line. The output SOD is set or reset as specified by the SIM instruction.

VCC + 5 Volt Supply

The 8085 microprocessor operates on a 5 V supply, which is connected with VCC terminal at PIN number 40.

GND Ground Reference.

The power supply ground is connected to GND at PIN number 20.

Table 2.3 shows the comparisons between 8085 and 8080A based on power supply, frequency and chip count. The 8085 is much simpler than 8080A for generating status information and control signals. The 8085 include all 72 instructions of 8080A, but it has two more instructions such as serial I/O and additional interrupt lines.

Table 2.3 Comparison of 8085 and 8080A

Parameters	8085	8080A
Power supply	+5V	+5V, -5V and +12V
Functional microprocessor	One 8085 IC with latch and gates	One 8080A, one 8224 and one 8228
Clock pulse	One ϕ	Two ϕ_1 ϕ_2
Clock frequency	3 MHz	2 MHz
Address bus	16-bit address lines Lower-order address bus is multiplexed with data bus	16-bit address lines
Data bus	8-bit data lines	8-bit data lines Data and status information are multiplexed
Interrupt	Five lines	One line
Extra features	Serial I/O lines	
Status	The lines S_0 , S_1 and IO/\bar{M} indicates operation status	Complex procedure to generate status information
Instruction set	74 instructions	72 instructions

SUMMARY

- In this chapter the architecture of 8085 microprocessor has been explained. The microprocessor communicates with memory, input and output peripheral devices through an address bus, data bus and control bus. An address bus is a group of lines that are used to locate a memory address or external device.
- The 8085 microprocessor has 16-bit address lines A_{15} to A_0 . This bus is unidirectional. Data bus is a group of bi-directional lines. These lines are used to transfer data between microprocessor and memory or input and output peripheral devices. The 8085 microprocessor has 8-bit data lines D_7 to D_0 , which is time multiplexing of lower order address/data bus AD_7 - AD_0 .
- Generally, a microprocessor performs four operations: memory read, memory write, I/O read and I/O write. The generation of read/write control signal \overline{MEMR} , \overline{MEMW} , \overline{IOR} , \overline{IOW} are incorporated in this section.
- The PIN diagrams of 8085 and their functions are also explained in this chapter.

MULTIPLE-CHOICE QUESTIONS

- 2.1 The microprocessor was introduced in the year
- (a) 1940 (b) 1971
(c) 1973 (d) 1980
- 2.2 The first company to bring out the microprocessor was
- (a) Intel (b) IBM
(c) HP (d) Texas
- 2.3 Which semiconductor technology is used for fabrication of 8085 microprocessor?
- (a) ECL (b) NMOS
(c) NMOS and HMOS (d) NMOS and CMOS
- 2.4 Which of the following microprocessors is a 4-bit microprocessor?
- (a) 4004 (b) 8080
(c) 8085 (d) Z80
- 2.5 Which of the following microprocessors is an 8-bit microprocessor?
- (a) 4004 (b) 8080
(c) 8085 (d) Z80
- 2.6 Which of the following microprocessors has an 8-bit data bus ?
- (a) 4004 (b) 8080
(c) 8085 (d) Z80
- 2.7 Which of the following microprocessors has a 16-bit address bus ?
- (a) 4004 (b) 8088
(c) 8085 (d) 8086
- 2.8 An 8-bit microprocessor has an
- (a) 8-bit data bus (b) 8-bit address bus
(c) 8-bit control bus (d) 8 interrupt lines
- 2.9 If a microprocessor is capable of addressing 64K bytes of memory, its address-bus width is
- (a) 16 bits (b) 20 bits
(c) 8 bits (d) none of these
- 2.10 If a microprocessor is capable of addressing 1 MB memory, its address-bus width is
- (a) 16 bits (b) 20 bits
(c) 8 bits (d) none of these
- 2.11 Flip-flops are used in a microprocessor to indicate
- (a) shift register (b) latches
(c) counters (d) flags
- 2.12 A microprocessor performs as a
- (a) CPU of a computer
(b) memory of a computer
(c) output device of a computer
(d) input device of a computer
- 2.13 A microprocessor consists of
- (a) ALU
(b) address bus
(c) data bus
(d) ALU, address bus and data bus
- 2.14 A microprocessor is a/an
- (a) SSI device (b) MSI device
(c) LSI device (d) VLSI device
- 2.15 The program counter in a microprocessor
- (a) keeps the address of the next instruction to be fetched
(b) counts the number of instructions being executed on the microprocessor
(c) counts the number of programs being executed on the microprocessor
(d) counts the number of interrupts handled by the microprocessor
- 2.16 For using a microprocessor-based system,
- (a) a program is required
(b) the program must be stored in memory before the system can be used
(c) the program need not be stored in memory
(d) the program is stored in the internal resistors of the microprocessor
- 2.17 A microcomputer consists of
- (a) microprocessor (b) memory
(c) output device (d) input device
- 2.18 A microprocessor is
- (a) an analog device
(b) a digital device

- (c) an analog and digital device
(d) none of these
- 2.19 The number of flags of the 8085 microprocessor is
(a) 6 (b) 5
(c) 4 (c) 3
- 2.20 The word size of the 8085 microprocessor is
(a) 8 bits (b) 16 bits
(c) 20 bits (d) 4 bits
- 2.21 The 8085 microprocessor is a
(a) 40-PIN IC (b) 32-PIN IC
- (c) 28-PIN IC (d) 24-PIN IC
- 2.22 The address bus of a microprocessor is
(a) unidirectional
(b) bi-directional
(c) unidirectional as well as bi-directional
(d) none of these
- 2.23 The data bus of a microprocessor is
(a) unidirectional
(b) bi-directional
(c) unidirectional as well as bi-directional
(d) none of these

SHORT-ANSWER-TYPE QUESTIONS

- 2.1 What are the different registers of 8085 microprocessor?
- 2.2 Mention the purpose of SID and SOD lines.
- 2.3 Write the difference between 8085 and 8080 microprocessor.
- 2.4 What is the crystal frequency if 8085 microprocessor operates in 3.5 MHz frequency?
- 2.5 Discuss the function of the following signals of the 8085 microprocessor:
RD, WR, ALE, S_0 and S_1
- 2.6 Explain the functions of following interrupt signal lines of 8085A:
TRAP, RST 7.5, RST 6.5, RST 5.5
- 2.7 What are the different buses of the 8085 microprocessor?
- 2.8 Explain the need to de-multiplex the bus AD_0-AD_7 .

REVIEW QUESTIONS

- 2.1 Draw the schematic diagram of all the functional blocks of the 8085 microprocessor along with their necessary interconnection. Explain in brief this architecture.
- 2.2 Draw and explain the functional block diagram of Intel 8085.
- 2.3 List the various registers of 8085 and explain their function.
- 2.4 Discuss the function of the ALU of 8085 with functional diagram.
- 2.5 What are the flags in 8085? Discuss the flag register with some example.
- 2.6 Explain the need of a program counter, stack pointer and status flags in the architecture of the Intel 8085 microprocessor.
- 2.7 What is the clock frequency of 8085 microprocessor if the crystal frequency is 5 MHz?
- 2.8 An 8085 program adds the hex numbers 2FH and 32H and places the result in its accumulator. What would be the status of the 8085 flags *CY*, *P*, *AC*, *Z*, *S* on completion of this addition?

- 2.9 An 8085 program subtracts the hex number 23H from FFH and places the result in its accumulator. What would be the status of the 8085 flags *CY, P, AC, Z, S* on completion of this subtraction?
- 2.10 Draw the schematic diagram to generate Read/Write control signals (\overline{MEMR} , \overline{MEMW} , \overline{IOR} , \overline{IOW}) for memory and I/O devices in the 8085 microprocessor.
- 2.11 Explain the control and status signals of the microprocessor in memory read and write operations.
- 2.13 Discuss the function of the following signals of the 8085 microprocessor:
INTR, INTA, HOLD, HLDA and READY.
- 2.14 Draw and explain the time multiplexing of AD_0-AD_7 .
- 2.15 What is the difference between a microprocessor and microcontroller ? Describe how data can flow between the microprocessor, memory and I/O devices.
- 2.16 Findout the status of different flags after addition of CBH and E8H.
- 2.17 Determine the status of differ flags after subtraction 25 H from FFH.
- 2.18 What would be the status of the 8085 flags after addition of 69 H and 72 H?

Answers to Multiple-Choice Questions

-
- | | | | | | | | |
|---------------------------|----------|----------|----------|-----------------|-----------------|----------|----------|
| 2.1 (b) | 2.2 (a) | 2.3 (b) | 2.4 (a) | 2.5 (b) and (c) | 2.6 (b) and (c) | 2.7 (c) | 2.8 (a) |
| 2.9 (a) | 2.10 (b) | 2.11 (d) | 2.12 (a) | 2.13 (d) | 2.14 (c) | 2.15 (a) | 2.16 (b) |
| 2.17 (a),(b), (c) and (d) | 2.18 (b) | 2.19 (b) | 2.20 (a) | 2.21 (a) | 2.22 (a) | 2.23 (b) | |

Chapter 3

Instruction Set and Addressing Modes of 8085 Microprocessor

3.1 INTRODUCTION

An instruction is any specified binary pattern which is placed inside the microprocessor to perform a specific operation. The instructions of the 8085 microprocessor are classified into five different groups, namely, data transfer group, arithmetic group, logical group, branch control group, I/O and machine control group. In this chapter all types of instruction groups are explained. The *instruction set* is the collection of all groups of instructions. Each instruction has two parts: the first part is the task to be performed. This is known as *operation code (opcode)*. The second part is data to be operated on, called *operands*. There are various techniques to specify operand of instructions. These techniques are known as *addressing modes*. All types of addressing modes are enlightened in this chapter. Generally, instructions are stored in the memory devices. Before execution of any instruction, the microprocessor locates the memory location and fetches the operational code through a data bus. Then the decoder decodes the instruction and performs the specified function. Therefore, the opcode fetch and its execution are performed in sequence. The sequencing is done by the control unit of the microprocessor and synchronized with the clock. The timing diagrams of read and write operations of memory and other peripheral devices are incorporated in this chapter.

3.2 ADDRESSING MODES

The instructions are used to copy or transfer data from a source into a destination. The source may be a register, memory, an input port, or an 8-bit number (00H to FFH). In the same way, the destination may also be a register, memory or an output port. The sources and destinations of data are known as operands. There are various formats to specify operands for instructions. The different techniques of specifying data are called the addressing modes. Generally, the following addressing modes are used in the 8085 microprocessor :

1. Immediate addressing
2. Register addressing
3. Direct addressing
4. Indirect addressing

3.2.1 Direct Addressing

In this addressing mode, the address of the operand always exists within the instruction. This mode can be used to read data from output devices and store it in the accumulator or write the data, content of the accumulator to the output devices. Examples of direct addressing are illustrated in Table 3.1.

Table 3.1 *Direct addressing*

<i>Instruction</i>	<i>Task</i>
IN 00H	Read data from Port 00H
OUT 01H	Write data in Port 01H
LDA 8000H	Load the content of the memory location 8000H in the accumulator
STA 9000H	Store the content of the accumulator in the memory location 9000H

In the instruction IN 00H, the address of an I/O port is 00H where the data is available. From this location, data is to be read and stored in the accumulator. Similarly, the content of the accumulator can be sent to the output port address 01H using OUT 01H instruction.

In LDA 8000H instruction, 8000H is the memory location from where data is to be copied. Therefore, the instruction itself specifies the source of data. After reading data from 8000H, it will be stored in the accumulator.

3.2.2 Register Addressing

In the register addressing mode, one of the registers A, B, C, D, E, H and L can be used as source of operands. Consequently, data is provided through the registers. In this mode, the accumulator is implied as a second operand. For example, the instruction ADD C says that the contents of the C register will be added with the contents of the accumulator. Most of the instructions using register-addressing mode have 8-bit data, though, some instructions deal with 16-bit register pairs, for example, PCHL instruction. Examples of register addressing are given in Table 3.2.

Table 3.2 *Register addressing*

<i>Instruction</i>	<i>Task</i>
MOV A,B	Move the content of B register to the accumulator
ADD C	The content of C register is added with the content of the accumulator
SUB B	Subtract the content of B register from the accumulator
PCHL	Exchanges the contents of the program counter with the contents of the H and L registers

3.2.3 Register Indirect Addressing

In the register indirect mode, the contents of specified registers are used to specify the address of the operand. Therefore, in register indirect instructions, the address is not explicitly specified. For example, the instruction MOV A, M means that move the contents of the content of the memory location whose address is stored in H and L register pair, in the accumulator. The instruction LDAX B is also another example. In this instruction, load the accumulator with the byte of data that is specified by the address in the B and C register pair. The instruction ADD M is also an example of register indirect addressing. Table 3.3 shows the examples of register indirect addressing.

Table 3.3 Register indirect addressing

<i>Instruction</i>	<i>Task</i>
LDAX B	Load the accumulator from address in register pair B-C
MOV A,M	Move the content of the memory location whose address is given in H and L registers in the accumulator
MOV M,B	Move the content of the accumulator in the memory location whose address is given in H and L registers
ADD M	Addition of the content of the memory location whose address is given in H and L registers and the content of the accumulator

3.2.4 Immediate Addressing

In the immediate addressing mode, the operand or data is present within the instruction. Load the immediate data to the destination, which is given in the instruction. Examples of direct addressing are depicted in Table 3.4.

Table 3.4 Immediate addressing

<i>Instruction</i>	<i>Task</i>
MVI D,FFH	Move FFH in Register D
LXI H,8050H	Load H and L register pair with 8050H
ADI 44H	Add 44H with the content of the accumulator
CPI B	Compare the contents of the accumulator with the content of Register B

The immediate instructions use the accumulator as an implied operand. The MVI (move immediate) instruction can move its immediate data to any of the working registers. For example, the instruction MVI D, FFH moves the hexadecimal data FFH to the D register.

The LXI instruction (load register pair immediate) uses 16-bit immediate data. This instruction is generally used to load addresses into a register pair. In LXI H, 8050H load H and L register pair with 16-bit immediate data 8050H.

3.2.5 Implicit Addressing

The addressing mode of certain instructions can be implied by the instruction's function. Actually, these instructions work on the content of the accumulator and there is no need of the address of the operand. Examples of implicit addressing are given below:

Table 3.5 Implicit addressing

<i>Instruction</i>	<i>Task</i>
DAA	Decimal adjust accumulator
CMA	Complement the content of accumulator
STC	Set carry flag
RAL	Rotate accumulator left

3.3 INSTRUCTION SET

An instruction is a command applied to the microprocessor to perform a specific function. The instruction set of a microprocessor means the entire group of instructions. Generally, instructions have been classified into the following five functional groups.

- ◆ Data transfer group
- ◆ Arithmetic group
- ◆ Logical group
- ◆ Branch control group
- ◆ I/O and machine control group
- ◆ Data-Transfer Group

3.3.1 Data Transfer Group

The data-transfer instructions copy data from a source to a destination without modifying the contents of the source. The term *data transfer* has been used for copying data. The data transfer can be possible between registers or between memories or between memory and registers or between I/O ports and the accumulator. The various types of data transfer are shown in Table 3.6.

Table 3.6 Types of data transfer

Types	Examples
Between registers.	Copy the contents of the register B into the register A.
Load a specific data byte to a register or a memory location.	Load the register B with the specific data byte FFH.
Between a memory location and a register.	Move data from a memory location 9000H to the register B.
Between an I/O device and the accumulator.	Move data from an input port to the accumulator.
Between registers pairs	Exchange the content of two register pairs.

EXAMPLES

MOV Move
MVI Move immediate
LDA Load accumulator directly from memory
STA Store accumulator directly in memory
LHLD Load H and L registers directly from memory
SHLD Store H and L registers directly in memory

An 'X' in the name of a data-transfer instruction means that the data transfer operation is performed with a register pair.

LXI Load register pair with 16-bit immediate data
LDAX Load accumulator from memory whose address is in a register pair
STAX Store the content of the accumulator in memory whose address is in the register pair
XCHG Exchange H and L with D and E
XTHL Exchange top of stack with H and L

3.3.2 Arithmetic Group

The arithmetic instructions perform arithmetic operations such as addition, subtraction, increment, and decrement data in registers or memory.

- ✓ **Addition** The contents of a register or the contents of a memory location or any 8-bit number can be added to the contents of the accumulator. After addition, the sum is stored in the accumulator.
- ✓ **Subtraction** An 8-bit number or the contents of a register or the contents of a memory location can be subtracted from the contents of the accumulator. After subtraction, the result will be stored in the accumulator.
- ✓ **Increment/Decrement** The content of a register or a memory location, 8-bit data can be incremented or decremented by 1.
 - ◆ In the same way, the contents of a register pair H-L or B-C or D-E (16-bit data) can be incremented or decremented by 1.
 - ◆ The increment and decrement operations can also be performed in a memory location.

EXAMPLES

ADD	Add to accumulator
ADI	Add immediate 8-bit data to accumulator
ADC	Add to accumulator using carry flag
ACI	Add immediate data to accumulator with carry
SUB	Subtract from accumulator
SUI	Subtract immediate data from accumulator
SBB	Subtract from accumulator with borrow (carry) flag
SBI	Subtract immediate from accumulator with borrow (carry) flag
INR	Increment specified 8-bit data or byte by one
DCR	Decrement specified 8-bit data or byte by one
INX	Increment register pair by one
DCX	Decrement register pair by one
DAD	Double register addition: Add content of register pair to H-L register pair

3.3.3 Logical Group

This group of instructions performs various logical operations such as AND, OR, Exclusive-OR, Rotate, Compare, and Complement with the contents of the accumulator.

- ✓ **AND, OR, Exclusive-OR** The content of a register or content of a memory location or content any 8-bit data can be logically ANDed, Ored, or Exclusive-ORed with the contents of the accumulator. Then results must be stored in the accumulator.
- ✓ **Rotate** Each bit of the accumulator can be shifted either left or right by one bit.
- ✓ **Compare** An 8-bit number or the content of a register or content of a memory location be compared with the contents of the accumulator to check greater than or equal or less than.
- ✓ **Complement** The contents of the accumulator can be complemented. Therefore, all 0's are replaced by 1's and all 1's are replaced by 0's.

EXAMPLES

ANA	Logical AND with accumulator
ANI	Logical AND with accumulator using immediate data
ORA	Logical OR with accumulator
OR	Logical OR with accumulator using immediate data
XRA	Exclusive logical OR with accumulator
XRI	Exclusive OR using immediate data

The *Compare* instructions compare the content of a register, or content of a memory location or an 8-bit data with the contents of the accumulator;

CMP	Compare
CPI	Compare using immediate data

The *Rotate* instructions shift the contents of the accumulator one bit in the left or right:

RLC	Rotate accumulator left
RRC	Rotate accumulator right
RAL	Rotate left through carry
RAR	Rotate right through carry

Complement and *Carry flag* instructions are

CMA	Complement accumulator
CMC	Complement carry flag
STC	Set carry flag

3.3.4 Branch Control Group

This group includes the instructions that change the sequence of program execution using conditional and unconditional jumps, subroutine call and return, and restart.

✓ **Jump** Jump instructions are generally conditional jump and unconditional jump types. Conditional jump instructions always test certain conditions such as *Zero* or *Carry flag* and then change the program execution sequence once the condition arises. On the other hand, when conditions are not used in the instruction set, the instruction is called unconditional jump.

✓ **Call, Return, and Restart** These instructions can also change the sequence of a program execution by calling a subroutine or returning from a subroutine. Like Jump instructions, Call instructions are conditional call and unconditional call. Conditional call instructions test all condition flags.

The unconditional branch control instructions are as follows:

JMP	Jump
CALL	Call
RET	Return

Conditional branching instructions always check the status of any one of the four condition flags to decide the sequence of a program execution. The following conditions may be specified

NZ	Not Zero (Z = 0)
Z	Zero (Z = 1)
NC	No Carry (C = 0)
C	Carry (C = 1)
PO	Parity Odd (P = 0)
PE	Parity Even (P = 1)
P	Plus (S = 0)
M	Minus (S = 1)

Thus, the conditional branching instructions are specified as follows:

Jumps	Calls	Returns	
C	CC	RC	(Carry)
JNC	CNC	RNC	(No Carry)
JZ	CZ	RZ	(Zero)
JNZ	CNZ	RNZ	(Not Zero)
JP	CP	RP	(Plus)
JM	CM	RM	(Minus)
JPE	CPE	RPE	(Parity Even)
JPO	CPO	RPO	(Parity Odd)

3.3.5 Stack, I/O and Machine Control Group

These instructions can perform various functions related to stack and input/output ports and machine control.

The following instructions are related with the Stack and/or Stack Pointer:

PUSH	Push two bytes of data onto the stack
POP	Pop two Bytes of data off the stack
XTHL	Exchange top of stack with H and L
SPHL	Move content of H and L to stack pointer

The I/O instructions are given below:

IN	Initiate input operation
OUT	Initiate output operation

The Machine Control instructions are as follows:

EI	Enable interrupt system
DI	Disable interrupt system
HLT	Halt
NOP	No operation

3.4 INSTRUCTION AND DATA FORMATS

In the Intel 8085 microprocessor, instructions are used to perform specified functions. Each instruction consists of two parts, namely, operation code (opcode) and operand. The opcode states the operation which will be performed. Each operation is always performed with some data. These data are known as operand.

Instructions are performed operations with 8-bit data and 16-bit data. 8-bit data can be obtained from a register or a memory location or input port. Similarly, 16-bit data may be available from a register pair or two consequent memory locations. Therefore binary codes of instructions are different. Due to different ways of specifying data for instructions, the machine or binary codes of all instructions are of different lengths. The Intel 8085 instructions are classified into the following three groups as given below:

- ◆ One-byte or 1-word instructions
- ◆ Two-byte or 2-word instructions
- ◆ Three-byte or 3-word instructions

3.4.1 One-Byte Instructions

A one-byte instruction consists of the opcode and operand in the same byte. Operand(s) are internal register and are coded into the instruction. For example,

<i>Op Code</i>	<i>Operand</i>	<i>Binary code</i>	<i>Hex code</i>	<i>Operations</i>
MOV	B,A	0100 0111	47H	Copy the contents of the accumulator in the register B
ADD	C	1000 0001	81H	Add the contents of the register C to the contents of the accumulator
SUB	B	1001 0000	90H	Subtract the contents of the register B to the contents of the accumulator
CMA		0010 1111	2FH	Compliment each bit in the accumulator
RAR		0001 1111	1FH	Rotate accumulator right

The above instructions are 1-byte instructions. In the first instruction, MOV B, A both operand registers are specified in A and B registers. In the second instruction ADD C, one operand is specified in the C register and the other operand is in the accumulator, which is assumed. In CMA instruction, the accumulator is assumed to be the implicit operand. These instructions are one-byte long and each instruction requires only one memory location.

3.4.2 Two-Byte Instructions

In a two-byte instruction, the first byte specifies the operation code and the second byte states the operand. The source operand is an 8-bit data immediately subsequently the opcode. For example,

<i>Op Code</i>	<i>Operand</i>	<i>Binary code</i>	<i>Hex code</i>	<i>Operations</i>
MVI	B,55H	0100 1111 0101 0101	4FH First-byte operation code 55H Second-byte data	Load an 8-bit data byte (55H) in the register B.
ADI	85H	1100 0110 1000 0101	C6H First-byte operation code 85H Second-byte data	Add 85H to the contents of the accumulator
IN	01H	1101 1011 0000 0001	DBH First-byte operation code 01H Second-byte data	Copy data to accumulator from input port address 01H
OUT	02H	1101 0011 0000 0010	D3H First-byte operation code 02H Second-byte data	The content of accumulator transfer to port address 02H

The above instructions are 2-byte instructions. This instruction would require two consecutive memory locations to store in memory.

3.4.3 Three-Byte Instructions

In a three-byte instruction, the first byte specifies the operation code (opcode), and the following two bytes stand for the 16-bit address. It may be noted that the second byte will be the lower-order address and the third byte will be the higher-order address. These instructions are three-byte instructions which consist of one opcode and two data bytes. Therefore, this instruction would require three memory locations to store in memory.

Op Code	Operand	Binary code	Hex code	Operations
LXI	H,8000H	0010 0001 0000 0000 1000 0000	21H First-byte operation code 00H Second-byte data 80H Third byte data	Load immediately 8000H in the HL registers pair
JMP	8085H	1100 0011 1000 0101 1000 0000	C3H First-byte operation code 85H Second-byte data 80H Third-byte data	Jump to the memory location 8085H
LDA	8000H	0011 0100 0000 0000 1000 0000	34H First-byte operation code 00H Second-byte data 80H Third-byte data	The content of memory location 8000H is copied to the accumulator

3.5 SYMBOLS AND ABBREVIATIONS

The symbols and abbreviations which have been used while explaining Intel 8085 instructions, are as follows:

Table 3.7 Symbols and abbreviations for Intel 8085

Symbol/Abbreviations	Meaning
16-bit address	16-bit address of the memory location
data	8-bit data
16-bit data	16-bit data
R, R _d , R _s	One of the registers A, B, C, D, E, H, L
A, B, C, D, H, L	8-bit register
A	Accumulator
H-L	Register pair H-L
B-C	Register pair B-C
D-E	Register pair D-E
RP	One of the register pairs. The representation of a register pair is given below: B represents B-C pair, B is higher-order register and C lower-order register D represents D-E pair, D is higher-order register and E is lower-order register H represents H-L pair, H is higher-order register and L lower-order register
Rh	The higher-order register of a register pair
RI	The lower-order register of a register pair
SP, SPH, SPL	SP represents 16-bit stack pointer. SPH is higher-order 8 bits and SPL lower-order 8 bits of register SP

(Contd.)

(Contd.)

PC, PCH, PCL	16-bit program counter. PCH is higher-order 8 bits and PCL lower-order 8 bits of register PC
PSW	Program Status Word
M	Memory whose address is in H-L pair
CS	Carry status
[]	The content of the memory location
←	Move data in the direction of arrow
↔	Exchange contents
∧	Logical AND operation
∨	Logical OR operation
⊕	Logical EXCLUSIVE OR
–	One's complement

3.6 8085 INSTRUCTIONS

The 8085 instructions are classified into the following main types as given below:

- ◆ Data-transfer instructions
- ◆ Arithmetic instructions
- ◆ Logical instructions
- ◆ Branch instructions

Some of Intel 8085 instructions are frequently, some occasionally and some seldom used by the programmer. It is not necessary that one should learn all the instructions to understand simple programs. The beginner can learn about 15 to 20 important instructions such as MOV, MVI, LXI, LDA, LHLD, STA, SHLD, ADD, ADC, SUB, JMPJC, JNC, JZ, INZ, INX, DCR, CMP, etc., and start to understand simple programs given in Chapter 4. While learning programs, the beginner can understand new instructions. The explanations of the instructions are given below:

3.6.1 Data-Transfer Instructions

Data-transfer instructions are used to transfer data between registers, register pairs, memory and registers, etc. In this section, all data-transfer instructions are described.

MOV Rd, Rs (Move the content of source register to destination register)

Rd ←Rs, Machine cycles: 1, States: 4, Flags: none, Register addressing mode, one-byte instruction. This instruction copies the contents of the source register into the destination register but the contents of the source register are not altered. For example, the instruction MOV B, C moves the content of Register C to Register B. To execute this instruction, 4 clock periods are required and flags are not affected.

MOV M, Rs (Move the content of source register to memory)

[M]←Rs, Machine cycles: 2, States: 7, Flags: none, Register indirect addressing, one-byte instruction. The content of the source register moves to the memory location, its location is specified by the contents of the HL registers. For example, the instruction MOV M, B will move the content of Register B to the memory location 8000H if the content of HL register pair is 8000H.

MOV Rd, M (Move the content of memory to destination register)

Rd ← [M], Machine cycles: 2, States: 7, Flags: none, Register indirect addressing, one-byte instruction. The content of memory location moves to a register. For example, the instruction MOV C, M will move the content of the memory location 8000H to Register C when the content of HL register pair is 8000H.

MVI Rd, data (Move immediate 8-bit data to register)

Rd ← data, Machine cycles: 2, States: 7, Flags: none, Immediate addressing modes, two-byte instruction
The 8-bit data is stored in the destination register. For example, the instruction MVI B, FFH moves FF to Register B. The code of this instruction is 3E, FF. The first byte of the instruction is opcode, and the Second byte of the instruction is operand, FFH that is to be moved to Register B.

MVI M, data (Move immediate data to memory)

[M] ← data, Machine cycles: 3, States: 10, Flags: none, Immediate addressing, two-byte instruction. The data will be stored in the memory location which is specified by the contents of the H L register pair.
Example: MVI M, 22H.

In this instruction, MVI M, 22H, 22H data moves to memory location 8500H as the content of H L register pair is 8500H. The opcode for MVI is 36 and 22H is the data. Therefore, instruction code is 36, 22.

LDA 16-bit address (Load Accumulator Direct)

A ← [16-bit address], Machine cycles: 4, States: 13, Flags: none, Direct addressing register, three-byte instructions. The content of a memory location, specified by a 16-bit address in the operand, is copied to the accumulator and the contents of the source are not altered.

Example: LDA 9000H.

This instruction can load the content of the memory location 9000H into the accumulator. The instruction code is 3A, 00, 90.

LDAX B/D Register pair (Load Accumulator Indirect)

A ← [BC] or A ← [DE], Machine cycles: 2, States: 7, Flags; none. Register indirect addressing, three-byte instructions.

The contents of the selected register pair locate a memory location. This instruction copies the contents of that memory location into the accumulator. The contents of either the register pair or the memory location are not altered. This instruction is only applicable for B–C and D–E register pairs.

Example: LDAX D.

The instruction LDAX D will load the content of the memory location whose address is in the DE register pair.

LXI Register pair, 16-bit data (Load register pair immediate)

Register pair ← 16 bits data, Rh ← 8 MSBs, Rl ← 8 LSBs of data.

Machine cycles: 2, States: 10, Flags none, Immediate addressing, three-byte instructions

This instruction loads 16-bit immediate data in the register pair designated in the operand. For example, LXI H, 9050H (16-bit data). In the code form it can be written as 3A, 50, 90. H ← 90H MSBs, L ← 50H LSBs of data

LHLD 16-bit address (Load H and L registers direct)

$L \leftarrow [\text{address}]$, $H \leftarrow [\text{address}+1]$, Machine cycles: 5, States: 16, Flags: none, Direct addressing, three-byte instructions.

The instruction copies the contents of the memory location located by the 16-bit address into Register L and also copies the content of the next memory location into Register H. The contents of source memory locations are not changed. For example, the instruction LHLD 80F0H will load the content of the memory location 80F0 H into Register L and the content of the memory location 80F 1 H is loaded into Register H.

STA 16-bit address (Store accumulator direct)

16-bit address $\leftarrow A$, Machine cycles: 4, States: 13, Flags: none, Addressing direct, three-byte instructions. The content of the accumulator is stored into the memory location specified by the operand. This is a 3-byte instruction. The second byte specifies the lower-order address and the third byte specifies the higher-order address.

Example: STA 9050H.

This instruction stores the content of the accumulator in the memory location 9050H.

SHLD 16-bit address (Store H and L pair registers direct)

$[\text{address}] \leftarrow L$, $[\text{address} + 1] \leftarrow [H]$. Machine cycles: 5, States: 16, Flags: . None, Direct addressing, three-byte instructions.

The content of Register L is stored into the memory location specified by the 16-bit address in the operand and the content of Register H is also stored into the next memory location by increasing the operand. The contents of Registers HL will not be changed. This is a 3-byte instruction. The second byte specifies the lower-order address and the third byte specifies the higher-order address.

Example: SHLD 8000H

This stores the content of Register L in the memory location 8000H. The content of Register H is stored in the memory location 8001H.

XCHG (Exchange the contents of H and L with D and E)

$H \leftrightarrow D$, $L \leftrightarrow E$, Machine cycles: 1, States: 4, Flags: none, Register addressing.

The content of Register H is exchanged with the content of Register D, and the content of Register L will be exchanged with the content of Register E.

Example: XCHG

3.6.2 Arithmetic Instructions

The arithmetic instructions are used to perform arithmetic operations. In this section, all arithmetic instructions are explained.

ADD R (Add register to accumulator)

$A \leftarrow A + R$, Machine cycles: 1, States: 4, Flags: all, Register addressing, one-byte instruction.

The contents of the operand (register) are added to the contents of the accumulator and the result is stored in the accumulator.

Example: ADD B

ADD M (Add memory to accumulator)

$A \leftarrow A + [M]$, Machine cycles: 2, States: 4, Flags: all, register indirect addressing, one-byte instruction.

The contents of the memory location specified by the contents of the HL registers are added with the accumulator. All flags are modified to reflect the result of the addition. For example, the instruction is ADD M.

ADC R (Add register to accumulator with carry)

$A \leftarrow A + R + CS$, Machine cycles: 1, States: 7. Flags: all. Register addressing, one-byte instruction.

The contents of the register and the carry flag are added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition.

Example: ADC C

ADC M (Add register to accumulator with carry)

$A \leftarrow A + M + CS$, Machine cycles: 2, States: 7, Flags: all, Register indirect addressing, one-byte instruction.

The content of the memory location, which is specified by the contents of the H-L register pair and the carry flag are added to the contents of the accumulator. The result is stored in the accumulator. All flags are effected to reflect the result of the addition.

Example: ADC M

ADI 8-bit data (Add immediate 8-bit data to accumulator)

$A \leftarrow A + \text{data}$, Machine cycles: 2, States: 7, Flags: all, Immediate addressing, two-byte instruction.

The 8-bit immediate data is added to the content of the accumulator and the result is stored in the accumulator. All flags will be modified to reflect the result of addition. For example, the instruction is ADI 78H. Here data 78H can be added with the content of accumulator.

DAD Register pair (Add register pair to H and L registers)

$H-L \leftarrow H-L + \text{Register pair}$, Machine cycles: 3, States: 10, Flags, CS, Register addressing, one-byte instruction.

The 16-bit contents of the specified register pair can be added to the contents of the HL register pair and the sum is stored in the H and L registers. The contents of the source register pair cannot be modified. When the result is greater than 16 bits, the CY flag will be set and no other flags are affected.

Example: DAD B.

The instruction DAD B states that the contents of the BC register pair will be added with the contents of the HL register pair.

SUB R (Subtract register from accumulator)

$A \leftarrow A - R$, Machine cycles: 1, States: 4, Flags: all, Register addressing, one-byte instruction.

The content of Register R is subtracted from the content of the accumulator, and the result is stored in the accumulator.

Example SUB C

SUB M (Subtract memory from accumulator)

$A \leftarrow A - [M]$, Machine cycles: 2, States: 7, Flags: all Register indirect addressing, one-byte instruction.

The contents of the memory are subtracted from the contents of the accumulator and the result is placed in

the accumulator. The memory location is specified by the contents of the H L register pair. All flags will be modified to reflect the result.

Example: SBB M

SBB R (Subtract register from accumulator with borrow)

$A \leftarrow A - R - CS$, Machine cycles: 2, States: 4, Flags: all, Register addressing, one-byte instruction.

The contents of the register and the borrow flag are subtracted from the contents of the accumulator and the result is placed in the accumulator. All flags are modified to reflect the result of the subtraction.

Example: SBB B

SBB M (Subtract memory and borrow from accumulator)

$A \leftarrow A - [M] - CS$, Machine cycles: 2, States: 7, Flags: all, Register indirect addressing, one-byte instruction.

The content of the memory location specified by HL pair and borrow flag are subtracted from the content of the accumulator. The result is placed in the accumulator.

Example: SBB M

SUI 8-bit data (Subtract immediate 8-bit data from accumulator)

$A \leftarrow A - 8\text{-bit data}$, Machine cycles: 2, States: 7, Flags: all, Immediate addressing, two-byte instructions.

The 8-bit data is subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are affected to reflect the result of the subtraction.

Example: SUI 34H.

This instruction will subtract 34H from the content of the accumulator and store the result in the accumulator.

SBI 8-bit data (Subtract immediate 8 bit data from accumulator with borrow)

$A \leftarrow A - 8\text{-bit data} - CS$, Machine cycles: 2, States: 7, Flags: all, Immediate addressing, two-byte instructions.

The 8-bit data and the borrow flag is subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are affected to reflect the result of the subtraction.

Example: SBI 34H.

This instruction will subtract 34H and the borrow flag from the content of the accumulator and store the result in the accumulator.

INR R (Increment register by 1)

$R \leftarrow R + 1$, Machine cycles: 1, States: 4, Flags: all except carry flag, Register addressing, one-byte instruction.

The contents of the selected register are incremented by 1 and the result is stored in the same register. All flags except CS are affected.

Example: INR D

INR M (Increment memory by 1)

$[M] \leftarrow [M] + 1$, Machine cycles: 3, States: 10, Flags: all except carry flag, Register indirect addressing, one-byte instruction.

The content of the memory location addressing by H and L registers is incremented by one. In this instruction all flags except CS are affected.

Example: INR M

INX RP (Increment register pair)

$RP \leftarrow RP + 1$, Machine cycles: 1, States: 6, Flags are not effected, Register addressing, one-byte instruction. The content of the specified register pair is incremented by one, and the result will be stored in the same register pair.

Example, INX H

DCR R (Decrement register by 1)

$R \leftarrow R - 1$, Machine cycles: 1, States: 4. Flags, all flags except carry flag, Register addressing, one-byte instruction.

The contents of the selected register are decremented by 1 and the result is stored in the same place. All flags except CS are affected.

Example: DCR C

DCR M (Decrement memory by 1)

$[M] \leftarrow [M] - 1$, Machine cycles: 3, States:10, Flags: all except carry flag, Register indirect addressing, one-byte instruction.

The content of the memory location addressed by HL pair is decremented by one.

Example: DCR M

DCX RP (Decrement register pair by 1)

$RP \leftarrow RP - 1$, Machine cycles: 1, States: 6, Flags: none, Register addressing, one-byte instruction.

The contents of the specified register pair are decremented by 1 and the result is stored in the same place.

Example: DCX D

DAA (Decimal adjust accumulator)

Machine cycles: 1, States: 4, Flags: all, one-byte instruction.

The contents of the accumulator are transferred from a binary code to two 4-bit Binary Coded Decimal (BCD) digits. This is the only instruction, which uses the auxiliary flag to perform the binary to BCD conversion. The conversion procedure as follows:

When the value of the lower-order 4 bits in the accumulator is greater than 9 or AC flag is set, the instruction adds 6 to the lower-order four bits. If the value of the higher-order 4 bits in the accumulator is greater than 9 or the carry flag is set, the instruction adds 6 to the higher-order four bits. In this instruction S, Z, AC, P, CY flags are altered to reflect the results of the operation. *Example*: DAA

3.6.3 Logical Instructions

The logical instructions perform AND, OR, EX-OR, operations; compare, rotate or complement of data in register or memory. All logical instructions are discussed in this section.

CMP R (Compare register with accumulator)

$A - R$, Machine cycles: 1, States: 4, Flags: all, Register addressing, one-byte instruction.

The contents of the register are compared with the contents of the accumulator. Both contents are conserved.

The result of the comparison can be reflected by setting the flags in PSW. When $A < \text{register}$, carry flag is set. If $A = \text{register}$, zero flag is set. While $A > \text{register}$, carry and zero flags are reset.

Example: CMP B

CMP M (Compare memory with accumulator)

$A - [M]$, Machine cycles: 2, States: 7, Flags: all, Register indirect addressing, One-byte instruction.

The content of the memory location specified by HL pair is compared with the content of the accumulator, and status flags are set according to the result of the comparison. The content of the accumulator remains unchanged.

Example: CMP M

CPI 8-bit data (Compare immediate 8-bit data with accumulator)

A 8-bit data, Machine cycles: 2, States: 7, Flags: all, Immediate addressing, two-byte instructions.

The second byte of the instruction or 8-bit data is compared with the contents of the accumulator. The values being compared remain unchanged. The result of the comparison will be reflected by setting the flags of the PSW. If $A < \text{8-bit data}$: carry flag is set. If $A = \text{data}$: zero flag is set. When $A > \text{8-bit data}$: carry and zero flags are reset.

Example: the instruction is CPI 46H

ANA R (Logical AND register with accumulator)

$A \leftarrow A \wedge R$, Machine cycles: 1, States: 4. Flags: all, Register indirect addressing, one-byte instruction.

The contents of the accumulator are logically ANDed with the contents of the register. The result is stored in the accumulator. All flags are modified to reflect the result of the operation. CY is reset and AC is set.

Example: ANA B

ANA M (Logical AND memory with accumulator)

$A \leftarrow A \wedge [M]$, Machine cycles: 2, States: 7, Flags: all, Register indirect addressing, one-byte instruction.

The content of the memory location whose address is specified by the contents of HL registers is AND with the accumulator. The result is placed in the accumulator. All status flags are affected.

Example: ANA M.

ANI 8-bit data (Logical AND immediate 8-bit data with accumulator)

$A \leftarrow A \wedge \text{8-bit data}$, Machine cycles: 2, States: 7, Flags: all, Register indirect addressing, one-byte instruction.

The contents of the accumulator are logically ANDed with the 8-bit data. After ANDing the result is stored in the accumulator. All flags are modified to reflect the result of the operation. The CY flag is reset and AC flag is set.

Example: ANI 24H.

ORA R (Logical OR register with accumulator)

$A \leftarrow A \vee R$, Machine cycles: 1, States: 4, Flags: all, Register addressing, one-byte instruction.

The content of Register R is logically ORed with the content of the accumulator. The result is stored in the

accumulator. All flags are modified to reflect the result of the operation. Carry flag CY and auxiliary carry AC are reset.

ORA M (Logical OR memory with accumulator)

$A \leftarrow A \vee [M]$, Machine cycles: 2, States: 7, Flags: all, Register addressing, one-byte instruction.

The contents of the accumulator are logically ORed with the contents of the memory location whose address is specified by the contents of H and L registers and the result is placed in the accumulator. All flags are modified to reflect the result of the operation. CY and AC are reset.

Example: ORA M

ORI 8-bit data (Logical OR immediate 8-bit data with accumulator)

$A \leftarrow A \vee$ 8-bit data. Machine cycles: 2, States: 7, Flags: all, Immediate addressing, two-byte instructions.

The second byte of the instruction is 8-bit data. This 8-bit data is ORed with the content of the accumulator and the result is placed in the accumulator. All flags are modified to reflect the result of operation. The CY and AC flags are reset.

Example: ORI FFH

XRA R (EXCLUSIVE-OR register with accumulator)

$A \leftarrow A \oplus R$, Machine cycles: 1, States: 4. Flags: all, Register addressing, one-byte instruction.

The contents of the accumulator are Exclusive ORed with the contents of the register and the result is placed in the accumulator. All status flags are affected. The CY and AC flags are reset.

Example: XRA B

XRA M (EXCLUSIVE-OR memory with accumulator)

$A \leftarrow A \oplus M$, Machine cycles: 2, States: 4. Flags: all, Register addressing, one-byte instruction.

The contents of the accumulator are Exclusive ORed with the contents of the memory location, which is specified by H L, register pair and the result is placed in the accumulator. All status flags are affected. The CY and AC flags are reset.

Example: XRA M

XRI 8-bit data (EXCLUSIVE-OR immediate 8-bit data with accumulator)

$A \leftarrow A \oplus$ data, Machine cycles: 2, States: 7, Flags: all immediate addressing, one-byte instruction.

The contents of the accumulator are Exclusive ORed with the 8-bit data. The result is stored in the accumulator. Flags are modified to reflect the result of the operation. The CY and AC flag become 1.

Example: XRI 86H

RLC (Rotate accumulator left)

$A_{n+1} \leftarrow A_n$, $A_0 \leftarrow A_7$, $CS \leftarrow A_7$, Machine cycles: 1, States: 4. Flags: CS, Addressing: implicit, one-byte instruction.

Each bit of the accumulator is rotated left by one bit. The seventh bit (D_7) of the accumulator is placed in the position of D_0 as well as in the carry flag. Only CY flag is modified depending on D_7 bit as shown in Fig. 3.1. S, Z, P and AC are not affected.

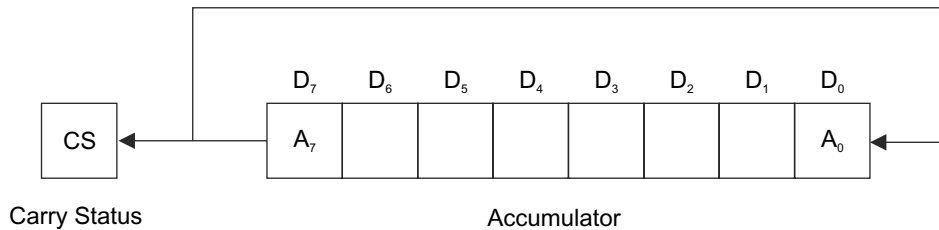


Fig. 3.1 Diagram for RLC

RRC (Rotate accumulator right)

$A_7 \leftarrow A_0$, $CS \leftarrow A_0$, $A_n \leftarrow A_{n+1}$, Machine cycles: 1, States: 4, Flags: CS, Implicit addressing, one-byte instruction.

Each binary bit of the accumulator is shifted right by one position. Bit D_0 is placed in the position of D_7 as well as in the carry flag. Therefore, CY is modified accordingly as depicted in Fig. 3.2. S, Z, P, AC are not affected.

Example: RRC

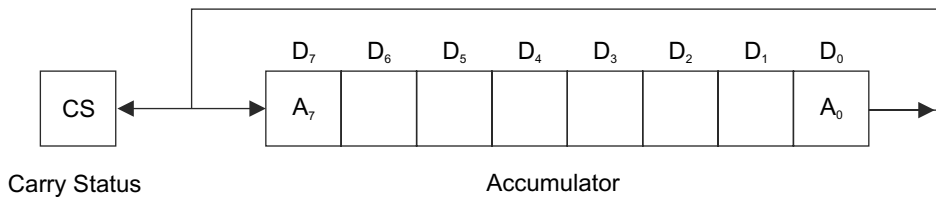


Fig. 3.2 Diagram for RRC

RAL (Rotate accumulator left through carry)

$A_{n+1} \leftarrow A_n$, $CS \leftarrow A_7$, $A_0 \leftarrow CS$, Machine cycles: 1, States: 4. Flags: CS, Implicit addressing, one-byte instruction.

The content of the accumulator is rotated left one bit through carry flag. The seventh bit of the accumulator, D_7 is placed in the carry flag, and the carry flag is placed to the least significant bit of the accumulator, D_0 . S, Z, P and AC are not affected but only carry flag is affected as shown in Fig. 3.3.

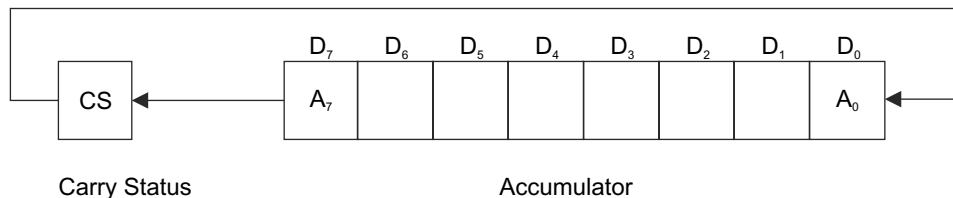


Fig. 3.3 Diagram for RAL

RAR (Rotate accumulator right through carry)

$A_n \leftarrow A_{n+1}$, $CS \leftarrow A_0$, $A_7 \leftarrow CS$, Machine cycles: 1, States: 4, Flags: CS, Addressing implicit, one-byte instruction.

The content of the accumulator is rotated right one bit through carry flag. The least significant bit of the

accumulator, D_0 is placed in the carry flag. The carry flag is placed in the most significant position of the accumulator D_7 . The carry flag is modified according to the bit D_0 . In this instruction CS flag is affected as depicted in Fig. 3.4.

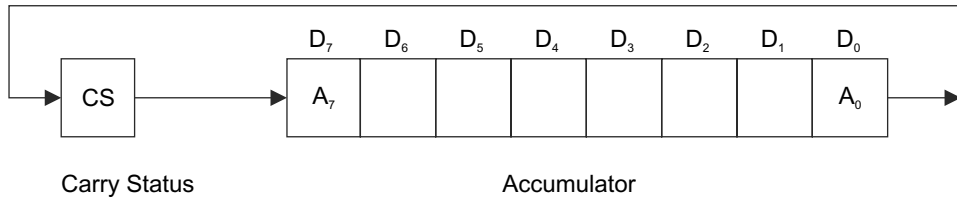


Fig. 3.4 Diagram for RAR

CMA (Complement the accumulator)

$A \leftarrow \bar{A}$, Machine cycles: 1, States: 4, Flags: none, Implicit addressing, one-byte instruction.

The content of the accumulator is complemented, and the result is placed in the accumulator. No flags are affected. For example, CMA determines one's complement of 0000 1100 is 1111 0011.

CMC (Complement the carry)

$CS \leftarrow \overline{CS}$, Machine cycles: 1, States: 4, Flags: CS, one-byte instruction.

The carry flag CS is complemented. No other flags are affected.

Example: CMC

STC (Set the carry)

$CS \leftarrow 1$, Machine cycles: 1, States: 4, Flags: CS, one-byte instruction.

The carry flag is set to 1. No other flags are affected.

Example: STC

3.6.4 Branch Group

The branch group instructions are generally used to change the sequence of the program execution. There are two types of branch instructions, namely, conditional and unconditional. The conditional branch instructions transfer the program to the specified address when the condition is satisfied. The unconditional branch instructions transfer the program to the specified address unconditionally. All conditional and unconditional branch instructions are explained in this section.

JMP 16-bit address (Jump unconditionally)

$PC \leftarrow \text{Label}$ (16-bit address), Machine cycles: 3, States: 10, Flags: none, Immediate addressing, three-byte instructions.

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand. For example, JMP 8000H, the program jumps to the instruction specified by the address location 8000H unconditionally.

Conditional Jump 16-bit address (Jump unconditionally)

In the conditional jump instruction, the program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW. All conditional jump instructions are given below:

<i>Opcode</i>	<i>Description</i>	<i>Flag Status</i>
JC	Jump on carry	CY = 1
JNC	Jump on no carry	CY = 0
JP	Jump on positive	S = 0
JM	Jump on minus	S = 1
JZ	Jump on zero	Z = 1
JNZ	Jump on no zero	Z = 0
JPE	Jump on parity even	P = 1
JPO	Jump on parity odd	P = 0

JC 16-bit address (Jump on carry)

PC ← 16-bit address, jump if CY = 0.

Machine cycles: 2/3, States: 7/10, Flags: none, Immediate addressing, three-byte instructions.

The program jumps to the memory location specified by the 16-bit address when the carry flag CY = 1.

Example: JC 9000H

JNC 16-bit address (Jump on no carry)

PC ← 16-bit address, jump if CY = 0.

Machine cycles: 2/3, States: 7/10, Flags: none, Immediate addressing, three-byte instructions.

The program is transferred to the memory location specified by the 16-bit address when there is no carry.

Example: JNC 9000H

JP 16-bit address (Jump on positive)

PC ← 16-bit address, jump if S = 0.

Machine cycles: 2/3, States: 7/10, Flags: none, Immediate addressing, three-byte instructions.

The program jumps to the memory location specified by the 16-bit address if the result is plus or positive.

Example: JP 8000H

JM 16-bit address (Jump on minus)

PC ← 16-bit address, jump if S = 1.

Machine cycles: 2/3, States: 7/10, Flags: none, Immediate addressing, three-byte instructions.

When the result is minus, the program jumps to the memory location specified by the 16-bit address.

Example: JM 9060H

JZ 16-bit address (Jump on zero)

PC ← 16-bit address, jump if Z = 1.

Machine cycles: 2/3, States: 7/10, Flags: none, Immediate addressing

The program jumps to the memory location specified by the 16-bit address while the result is zero or the zero flag is set.

Example: JZ 9500H

JNZ 16-bit address (Jump on no zero)

PC ← 16-bit address jump if CS = 1.

Machine cycles: 2/3, States: 7/10, Flags: none, Immediate addressing, three-byte instructions.

The program jumps to the memory location specified by the 16 -bits when the result is not zero or the zero flag is reset.

Example: JNZ 9500H

JPE 16-bit address (Jump on even parity)

PC ← 16-bit address (jump if even parity)

Machine cycles: 2/3, States: 7/10, Flags: none, Immediate addressing, three-byte instructions.

When the result contains an even number of 1s, the program jumps to the memory location specified by the 16-bit address.

Example: JPE 8500H

JPO 16-bit address (Jump on odd parity)

PC ← 16-bit address; the parity status P = 0,

Machine cycles: 2/3, States: 7/10, Flags: none, Immediate addressing, three-byte instructions.

The program jumps to the memory location specified by the 16-bit address when the result contains an odd number of 1s.

Example: JPO 8000H.

CALL 16-bit address (Unconditional subroutine CALL)

$([SP] - 1) \leftarrow PCH, ([SP]-2) \leftarrow PCL, [SP] \leftarrow [SP]-2, PC \leftarrow 16 \text{ bit address}$

Machine cycles: 5, States: 9/18, Flags: none, Immediate/register

The program sequence is transferred to the memory location specified by the 16-bit address given in the instruction. Before the transfer, the contents of the program counter (the address of the next instruction after CALL) are pushed onto the stack.

Example: CALL 8700H

CALL 16-bit address (CALL Conditionally)

$([SP] - 1) \leftarrow PCH, ([SP]-2) \leftarrow PCL, [SP] \leftarrow [SP]-2, PC \leftarrow 16 \text{ bit address}$

Machine cycles: 2/5, States: 9/18, Flags: none, Immediate/register, three-byte instructions.

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand of the instruction based on the specified flag of the PSW. Before the transfer, the address of the next instruction after the call or the contents of the program counter is pushed to the stack. All conditional CALL instructions are given below.

<i>Opcode</i>	<i>Description</i>	<i>Flag Status</i>
CC	Call on carry	CY = 1
CNC	Call on no carry	CY = 0
CP	Call on positive	S = 0
CM	Call on minus	S = 1
CZ	Call on zero	Z = 1
CNZ	Call on no zero	Z = 0
CPE	Call on parity even	P = 1
CPO	Call on parity odd	P = 0

RET (Return from subroutine unconditionally)

PCL ← [SP]

PCH ← [SP]+1

[SP] ← [SP]+2

Machine cycles: 3, States: 10. Flags: none, Register indirect addressing

The program sequence is transferred from the subroutine to the calling program. Therefore, RET is used at the end of a subroutine. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address.

Example: RET

Conditional Return (Return from subroutine unconditionally)

PCL ← [SP]

PCH ← [SP]+1

[SP] ← [SP]+2

Machine cycles: 1/3, States: 10, Flags: none, Register indirect addressing

The program sequence is transferred from the subroutine to the calling program based on the specified flag of the PSW. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address. All conditional call instructions are below:

<i>Opcode</i>	<i>Description</i>	<i>Flag Status</i>
RC	Return on carry	CY = 1
RNC	Return on no carry	CY = 0
RP	Return on positive	S = 0
RM	Return on minus	S = 1
RZ	Return on zero	Z = 1
RNZ	Return on no zero	Z = 0
RPE	Return on parity even	P = 1
RPO	Return on parity odd	P = 0

PCHL (Load program counter with H-L contents)

PC ← H-L, PCH ← H, PCL ← L

Machine cycles: 1, States: 6 Flags: none, Register addressing, one-byte instruction.

The contents of H and L registers are transferred into the program counter. The contents of Register H are placed as the higher-order byte of PC register and the contents of Register L as the lower-order byte.

Example: PCHL.

RST 0-7 (Restart)

$[SP] - 1 \leftarrow PCH, [SP] - 2 \leftarrow PCL$

$[SP] \leftarrow [SP] - 2, [PC] \leftarrow 8 \text{ times } n$

Machine cycles: 3, States: 12, Flags: none, Register indirect addressing, one-byte instruction.

The RST(Restart) is a one-byte CALL instruction to one of the eight memory locations depending upon the number. These instructions are generally used in conjunctions with interrupts and inserted using external hardware. These instructions can also be used as software instructions in a program to transfer program execution to any one of the eight locations. The address of the restart instructions are given below:

<i>Instruction</i>	<i>Restart Address</i>
RST 0	0000H
RST 1	0008H
RST 2	0010H
RST 3	0018H
RST 4	0020H
RST 5	0028H
RST 6	0030H
RST 7	0038H

Interrupt Instructions

The 8085 microprocessor has four additional interrupts. The interrupt instructions and their restart addresses are given below:

<i>Interrupt</i>	<i>Restart Address</i>
TRAP	0024H
RST 5.5	002CH
RST 6.5	0034H
RST 7.5	003CH

These interrupts generate RST instructions internally and thus do not require any external hardware.

3.6.5 Stack/PUSH and POP Instructions

These instructions are used to manipulate stack-related operations. All stack instructions are discussed as follows:

PUSH B (Push the content of register pair B and C to stack)

$[SP] - 1 \leftarrow B$

$[SP] - 2 \leftarrow C,$

$[SP] \leftarrow [SP] - 2$

Machine cycles: 3, States: 12. Flags: none, Register indirect addressing, one-byte instruction.

The contents of the register pair designated in the operand are copied onto the stack in the following sequence. The stack pointer register is decremented and the contents of the higher-order Register B are copied into that location. Then stack pointer register is decremented again and the contents of the lower-order Register C are copied to that location.

PUSH D (Push the content of register pair D and E to stack)

$[SP] - 1 \leftarrow D$
 $[SP] - 2 \leftarrow E,$
 $[SP] \leftarrow [SP] - 2$

Machine cycles: 3, States: 12. Flags: none, Register indirect addressing, one-byte instruction.

The contents of the register pair DE are pushed into the stack in the following sequence. The stack pointer register is decremented by 1 and the contents of Register D are copied into that location. After that, the stack pointer register is decremented and the contents of Register E are copied to that location.

PUSH H (Push the content of register pair H and L to stack)

$[SP] - 1 \leftarrow H$
 $[SP] - 2 \leftarrow L$
 $[SP] \leftarrow [SP] - 2$

Machine cycles: 3, States: 12. Flags: none, Register indirect addressing, one-byte instruction.

The contents of the register pair H and L to stack are copied onto the stack in the sequence given above.

PUSH PSW (PUSH accumulator content and flags on stack)

$[SP] - 1 \leftarrow A$
 $[SP] - 2 \leftarrow \text{PSW (Program Status Word)}$
 $[SP] \leftarrow [SP] - 2$

Machine cycles: 3, Status: 12, Flags: none, Register indirect addressing, one-byte instruction.

The stack pointer register is decremented by 1 and the content of the accumulator is pushed into the stack. Again, the stack pointer register is decremented by 1 and the contents of status flags are also pushed into the stack. Then content of the register SP is decremented by 2 to indicate new stack.

POP B (Pop off stack to register pair B and C)

$C \leftarrow [SP]$
 $B \leftarrow [SP] + 1$
 $[SP] \leftarrow [SP] + 2$

Machine cycles: 3, States: 10. Flags: none, Register indirect addressing, one-byte instruction.

The contents of the memory location pointed out by the stack pointer register are copied to the lower-order Register C. The stack pointer is incremented by 1 and the contents of that memory location are copied to the higher-order register B. The stack pointer register is again incremented by 1.

POP D (Pop off stack to register pair D and E)

$E \leftarrow [SP]$
 $D \leftarrow [SP] + 1$
 $[SP] \leftarrow [SP] + 2$

Machine cycles: 3, States: 10, Flags: none, Register indirect addressing, one-byte instruction.

The contents of the memory location pointed out by the stack pointer register are copied to Register E. Then stack pointer is incremented by 1 and the contents of that memory location are copied to Register D. After that the stack pointer register is incremented by 1.

POP H (Pop off stack to register pair H and L)

$L \leftarrow [SP]$
 $H \leftarrow [SP] + 1$
 $[SP] \leftarrow [SP] + 2$

Machine cycles: 3, States: 10, Flags: none, Register indirect addressing, one-byte instruction.

The contents of the memory location specified by the stack pointer register are copied to the lower-order Register L. The stack pointer is incremented by 1 and the contents of that memory location are copied to the higher-order Register H. Then stack pointer register is incremented by 1.

POP PSW (Pop off stack to accumulator and flags)

$PSW \leftarrow [SP]$
 $A \leftarrow [SP] + 1$
 $[SP] \leftarrow [SP] + 2$

Machine cycles: 3, States 10, Flags: none, Register indirect addressing, one-byte instruction.

The process status word, which was saved during the execution of the POP PSW, can move from the stack to PSW. The stack pointer is incremented by 1 and the contents of that memory location are copied to accumulator. Then stack pointer register is incremented by 1.

XTHL (Exchange H and L with top of stack)

$L \leftrightarrow [SP]$
 $H \leftrightarrow [SP] + 1$

Machine cycles: 5, States: 16, Flags: none, Register indirect addressing, one-byte instruction.

The contents of the L register are exchanged with the stack location pointed out by the contents of the stack pointer register. Then contents of the H register are exchanged with the next stack location (SP+1); but the contents of the stack pointer register are not altered.

SPHL (Copy the contents of H-L register pair to the stack pointer)

$[H-L] \rightarrow [SP]$.

Machine cycles: 1, States: 6, Flags: none, Addressing: register, one-byte instruction.

This instruction copied the contents of the H and L registers into the stack pointer register. The contents of the H register provide the higher-order address and the contents of the L register also provide the lower-order address. The contents of the H and L registers are not altered.

3.6.6 I/O and Machine Control Instructions

These instructions are used to perform the input/output operations and machine control operations. In this section, the I/O and machine control instructions are explained below:

EI (Enable interrupts)

Machine cycle: 1, States: 4, Flags: none, one-byte instruction.

When this instruction is executed, the interrupt enable flip-flop is set and all interrupts are enabled. No flags are affected. After the acknowledgement of an interrupt, the interrupt enable flip-flop is reset, thus disabling the interrupts.

Example: EI

DI (Disable Interrupts)

Machine cycle: 1, States: 4, Flags: none, one-byte instruction.

When this instruction is executed, the interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled. No flags are affected.

Example: DI

NOP (No operations)

Machine cycle: 1, States: 4, Flags: none, one-byte instruction.

No operation is performed. The instruction is fetched and decoded. However no operation is executed. Therefore, the registers and flags are not affected.

Example: NOP

HLT (Halt and enter wait state)

Machine cycle: 1, States: 5, Flags: none, one-byte instruction.

When the instruction HLT is executed, the microprocessor finishes executing the current instruction and halts any further execution. An interrupt or reset is necessary to exit from the halt state. No registers and status flags are affected.

Example: HLT

SIM (Set Interrupt Mask)

Machine cycle: 1, States: 4, Flags: none, one-byte instruction.

This instruction is used to read the status of interrupts 7.5, 6.5, 5.5 and read serial data input bit. The instruction loads eight bits in the accumulator with the following interpretations.

Example: RIM

RIM (Read Interrupt Mask)

Machine cycle: 1, States: 4, Flags: none, one-byte instruction.

This instruction is used to implement the interrupts 7.5, 6.5, 5.5 and read serial data output. When this instruction is executed, the accumulator is loaded with eight bits with the following interpretations.

Example: RIM

IN 8-bit port-address (Input data to accumulator from an I/O port with 8-bit address)

$A \leftarrow [\text{Port}]$, Machine cycles: 3, States: 10, Flags: none, Direct addressing, two-byte instructions.

The contents of the input port whose address is specified by 8-bit port address are read and loaded into the accumulator. For example, IN 00H. This instruction states that the data available on the port address 00H is moved to the accumulator.

OUT 8-bit port-address (Output data from accumulator to a I/O port with 8-bit address)

$[\text{Port}] \leftarrow A$, Machine cycles: 3, States 10, Flags: none, Direct addressing, two-byte instructions.

The contents of the accumulator are copied into the I/O port specified by the 8-bit address. For example, OUT 01H. This instruction states that the content of the accumulator is moved to the port address 01H.

3.7 INSTRUCTION TIMING DIAGRAM

Instructions are stored in the memory of a microcomputer to enable it perform a specified operation on given data. To perform a particular task, a programmer should write a sequence of instructions, called a program. To execute an instruction, the microprocessor fetches one instruction code from the memory via a data bus at a time. Then it decodes the instruction code in the instruction register and performs the specified function. In the same way, all other instructions of a program are executed one by one to produce the final result.

In a 1-byte instruction, only the operation code is fetched from memory and executed it. But in 2-byte instructions and 3-byte instructions, the subsequent codes are fetched, decoded and executed in the same way as a 1-byte instruction. All read operations such as *opcode fetch* from memory and *operand read* from memory are performed within a given time period. The system clock provides the timing of instruction and this operation is controlled by the control unit of the microprocessor. In this section, the timing diagram of instructions are explained in detail. To understand the timing diagram, the following terms must be well known:

T state, Instruction cycle, Fetch cycle, Execute cycle, and machine cycle

✓ **T state** This is defined as one subdivision of the operation performed in one clock period. Each subdivision is considered as internal state of operation, which is synchronized with the system clock.

✓ **Instruction Cycle (IC)** This is defined as the total time required executing an instruction completely. The instruction cycles consist of a fetch cycle and execute cycle as depicted in Fig. 3.5. In a Fetch Cycle (FC), the microprocessor fetches the opcode from the memory. After the opcode fetch operation, the necessary steps are carried out to get the operand, if required, from the memory and then perform the specific operation specified in an instruction. This operation is called Execute Cycle (EC). The time period of Fetch Cycle (FC) is a fixed but the time required to execute an instruction or time period of Execute Cycle (EC) is variable which depends on the type of instructions. The total time required to execute an instruction is the summation of the time period of fetch cycle and execute cycle. This can also be written as

$$IC = FC + EC$$

3.7.1 Fetch Operation

The first byte of an instruction is known as its *opcode*. An instruction may be one-byte or two-byte or three-byte long. When an instruction is more than one byte, the other bytes are data or operand. In the Program Counter (PC), the memory address of the next instruction to be executed is stored. At the starting of a fetch

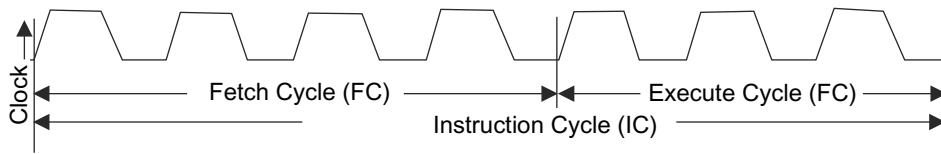


Fig. 3.5 Instruction cycle

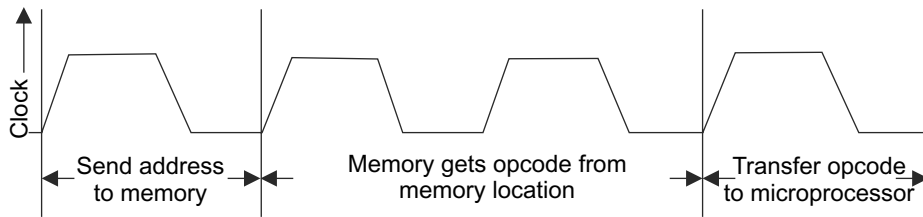


Fig. 3.6 Fetch Cycle

cycle, the content of the program counter, which is the address of the memory location where the opcode is stored, will be sent to the memory. Then the memory gets the opcode from memory location on the data bus. To perform this operation, two consecutive clock pulses are required. In the next clock cycle, data will be transferred to the microprocessor. To complete the entire fetching operation, four clock cycles are required as depicted in Fig. 3.6. In slow memory, the microprocessor has to wait till the memory sends the opcode. The clock cycle for which the CPU waits is known as *wait cycle*. Therefore, more than four clock cycles are required for opcode fetch operation in case of a slow memory system.

3.7.2 Execute Operation

In the fetch cycle, the opcode is fetched from the memory and moves to the data register (DR) and then moves to instruction register (IR). After that it moves to the decoder circuit which decodes the instruction. After decoding the instruction, execution starts. When the operand is in the general-purpose registers, the execution is immediately performed. The time taken in decoding and execution is one clock cycle. When the instruction contains the operand, which is in the memory, the microprocessor has to perform read operations to get the desired operand. After receiving the operand from memory, the microprocessor performs the execute operation. A read cycle is similar to a fetch cycle. In some instructions, a write operation is performed. In case of a write cycle, data are sent from the microprocessor to the memory or an output device. Therefore, an execute cycle consists of one or more read or write cycles or both.

3.7.3 Machine Cycle

The machine cycle is the sequence of operations required to complete one of the following functions:

Opcode fetch, memory read, memory write, I/O read and I/O write operations.

In other words, the operation of accessing either memory or I/O device is called a *machine cycle*. In the 8085 microprocessor, the machine cycle consists of three to six clock cycles. An instruction cycle consists of several machine cycles. In the first machine cycle of an instruction cycle, the opcode of an instruction is fetched. The 1-byte instructions require only one machine cycle to fetch the opcode and execute the instruction. Two-byte and three-byte instructions require more than one machine cycle. The additional machine cycles are required to read data from memory or I/O devices or to write data into the memory or I/O devices. For example, instruction cycle for MVI B, data is depicted in Fig. 3.7. This instruction has two machine

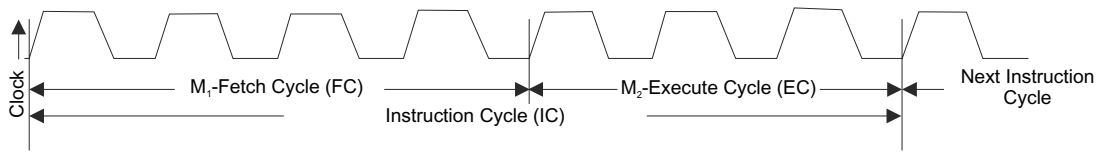


Fig. 3.7 Instruction cycle

cycles. The first machine cycle (M_1) is for fetching opcode, and the other machine cycle (M_2) is for reading data from the memory and executing the instruction.

3.7.4 Flow of Opcode and Data for Instruction

At the starting of the Fetch Cycle (FC), the content of the Program Counter (PC) is loaded into a Memory Address Register (MAR). After that the content of MAR is transferred to the memory through the address bus. Subsequently, the microprocessor sends control signals to the memory and it indicates that microprocessor wants to read the content of the memory. Then the decoder circuit in the memory is activated and the memory can understand what will happen. Consequently, the memory sends an opcode to the microprocessor through the data bus. Initially, the opcode is placed in the memory data bus and then it is placed from the memory data bus to Memory Data Register (MDR). Subsequently, the opcode is placed in the Instruction Register (IR). Then, the instruction decoder decodes the instruction and the instruction is executed by the microprocessor. After that the content of the Program Counter (PC) is incremented by one. Figure 3.8 shows the flow of instruction.

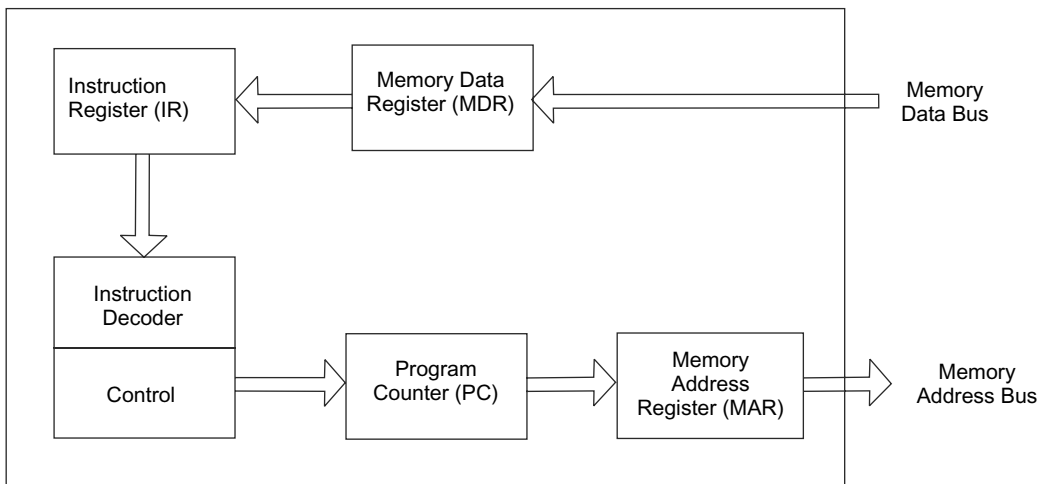


Fig. 3.8 Flow of instruction

Most of the instructions are of two bytes. Hence, the execution of an instruction requires opcode as well as operand/data. The flow of data is depicted in Fig. 3.9. The data can be received from memory or any input device. Actually, the data is input to the processor through the data bus and then it is placed either in the accumulator or in the general-purpose registers as per instruction. After execution of an instruction, the data is placed in a register or a memory location. After completion of execution of a program, the result is placed in the memory or any output device. Whenever data is written in the memory, it must be stored in the memory data register until the write operation has been completed.

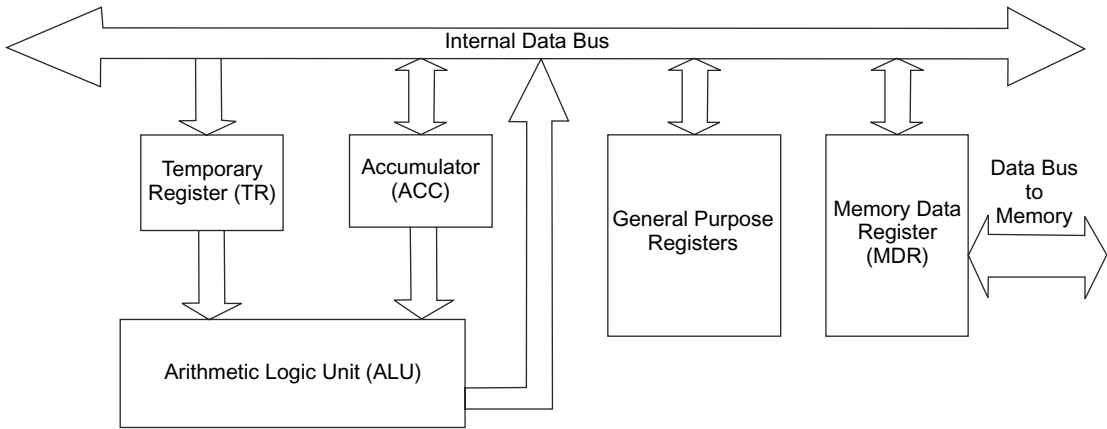


Fig. 3.9 Flow of data

3.8 TIMING DIAGRAM

The graphical representation of all steps which are performed in a machine cycle is known as the timing diagram. In this section, the timing diagrams for opcode fetch, memory read, memory write, I/O read and I/O write operations are explained briefly.

3.8.1 Timing Diagram of Fetch Cycle

In the fetch cycle, the microprocessor fetches the opcode of an instruction from the memory. Figure 3.10(a) shows the timing diagram for an opcode fetch cycle of an instruction MOV A, B. Assume that the opcode of instruction MOV A, B is stored in 8000H and the content of Register B is 4FH.

Memory location	Opcode	Mnemonics
8000H	78H	MOV A, B

To execute this instruction, four consecutive clock cycles T_1 , T_2 , T_3 and T_4 are required. The sequence of operations are given below:

First Clock Cycle

- ◆ In the first clock cycle T_1 , the microprocessor places the content of the program counter, address of the memory location 8000H, where the opcode is available on the 16-bit address bus. The 8 MSBs of the memory address (80H) are placed on the higher-order address bus, $A_{15}-A_8$ and 8 LSBs of the memory address (00H) are placed on the lower-order address bus, AD_7-AD_0 . Since the AD bus is needed to transfer data during subsequent clock cycles, it is used in a time-multiplexed mode.
- ◆ The microprocessor sends an Address Latch Enable (ALE) signal to go high and latch the 8 LSBs of the memory address. Therefore, lower-order address bus is demultiplexed and the complete 16-bit memory address available in the subsequent clock cycles to get the opcode from the specified memory address, 8000H.
- ◆ The microprocessor sends the status signals $IO/\overline{M} = 0$, $S_0 = 1$ and $S_1 = 1$ to indicate opcode fetch operation.

Second Clock Cycle

- ◆ During T_2 , lower-order bus AD_7-AD_0 is ready to carry data from the memory location. The

microprocessor sends the control signal $\overline{RD} = 0$ to enable memory and the program counter is incremented by 1 to 8001H. Now the opcode from the specified memory location 8000H is placed on the data bus.

Third Clock Cycle

- ◆ During T_3 , the microprocessor reads the opcode and places it in the instruction register IR. The memory is disabled when \overline{RD} goes high during T_3 . The fetch cycle is completed by T_3 .

Fourth Clock Cycle

- ◆ The microprocessor decodes the instruction *opcode* in T_4 . It also places the content of the B register in the temporary register. After that it transfers it to the accumulator.

3.8.2 Timing Diagram of Memory Read

When an instruction is one-byte long, only one machine cycle is required as depicted in Fig. 3.10(a). For

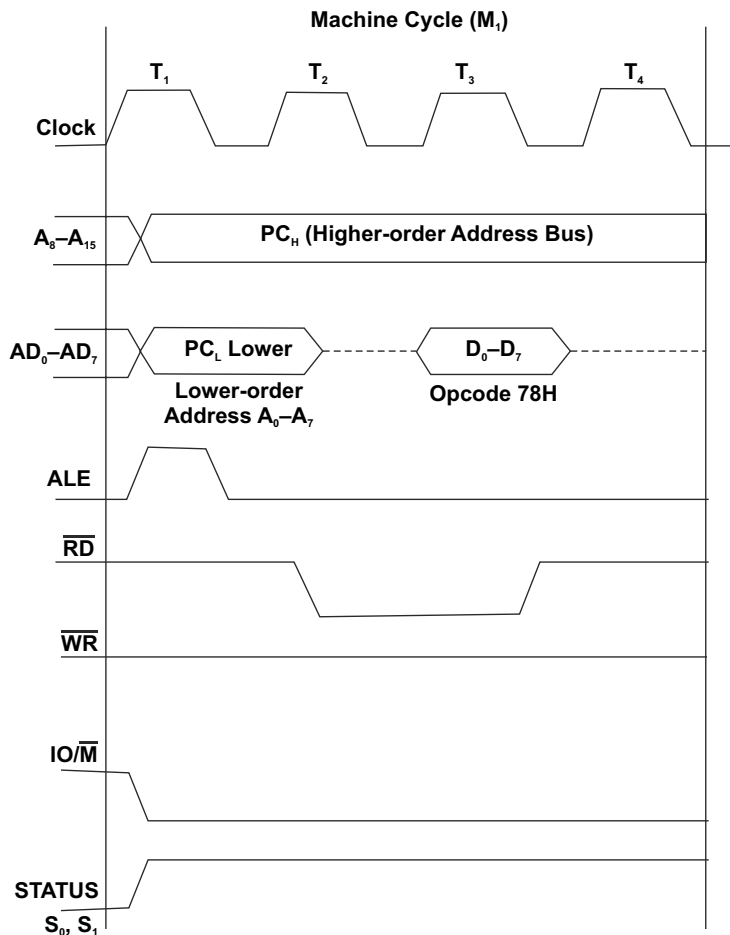


Fig. 3.10(a) Timing diagram of opcode fetch operation (MOV A,B)

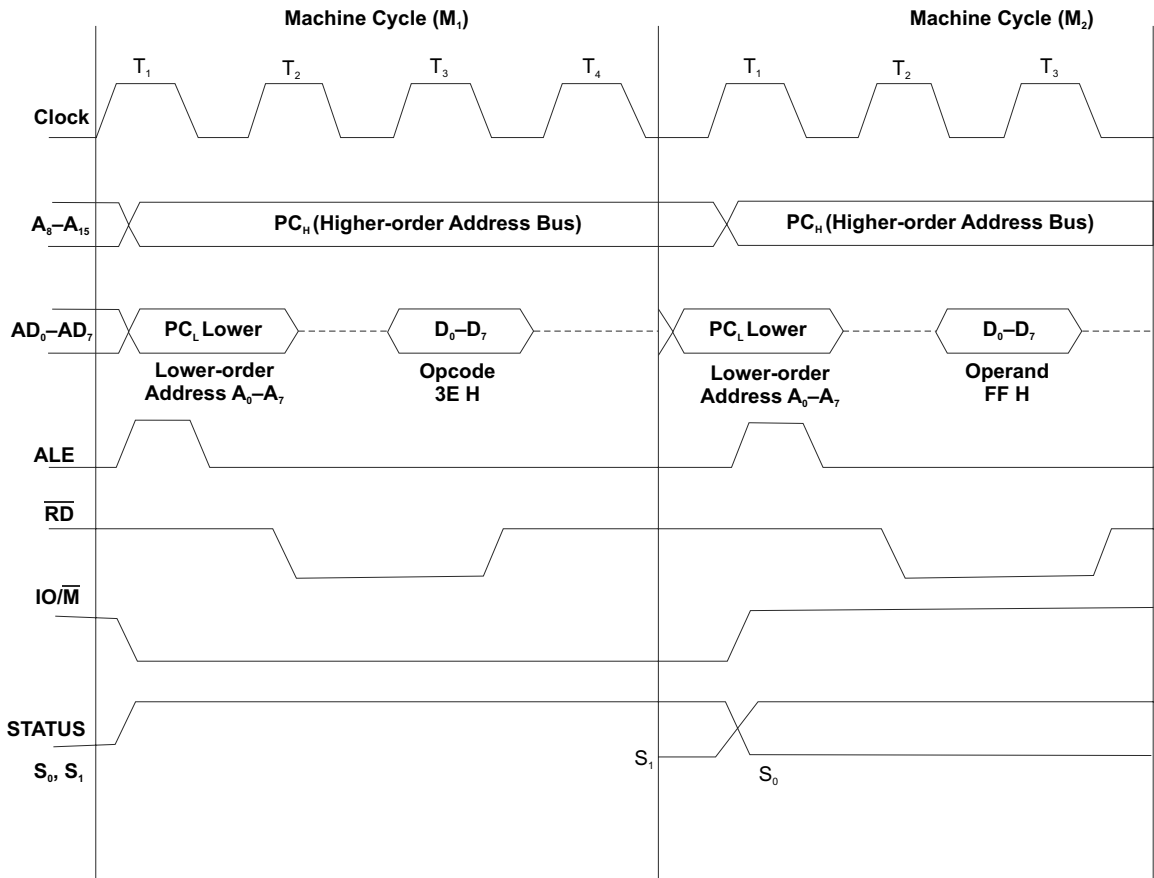


Fig. 3.10(b) Timing diagram of *MVI R, data* (*MVI C, FFH*)

example, MOV, SUB, ADD, RAL are one-byte long instructions. The operands are in the general-purpose registers; the decoding of the operation code and its execution takes only one clock cycle, T_4 . When an instruction is two or three bytes long, more than one machine cycle is required. In the first machine cycle, M_1 the opcode, is fetched from the memory. The subsequent machine cycles M_2, M_3 are required to read operands from the memory or I/O devices or to write data into the memory or I/O devices. The timing diagram for a two-byte instruction *MVI C, data* is illustrated in Fig. 3.10(b).

The *MVI C, data* is a two-byte instruction. In the coded form, it is written as $3E, FF$ where $3E$ is the opcode for *MVI C* instruction and FF is data. This instruction is stored in two consecutive memory locations, $8000H$ and $8001H$.

Memory location	Opcode	Mnemonics
$8000H$	$3E H$	MOV C,FF H
$8001H$	$FF H$	

This instruction requires two machine cycles, M_1 and M_2 . The first machine cycle M_1 is known as the fetch cycle to fetch operation code $3E$ from the memory. The timing diagram for opcode fetch operation has already been explained in Section 3.7.1. The second machine cycle M_2 is used to read the operand

(FFH) from the memory. Actually this is a memory read cycle. Figure 3.11 shows the machine cycle M_2 and its operation is explained below:

First Clock Cycle of M_2 In the first clock cycle (T_1) the microprocessor places the content of the program counter, 8001H, which is the address of operand on the 16-bit address bus. The 8 MSBs of the memory address 80H are placed on the higher-order address bus, $A_{15}-A_8$ and 8 LSBs of the memory address, 01H are placed on the lower-order address bus, AD_7-AD_0 .

The microprocessor sends an Address Latch Enable (ALE) signal to go high and latch the 8 LSBs of the memory address. Then a lower-order address bus is demultiplexed and the complete 16-bit memory address is available in the subsequent clock cycles to get the operand from memory location 8001H.

The status signals $IO/\overline{M} = 0$, $S_0 = 0$ and $S_1 = 1$ to identify the memory read operation.

Second Clock Cycle of M_2 The lower-order bus AD_7-AD_0 is ready to accept operand from memory. The microprocessor sends the control signal $\overline{RD} = 0$ to enable memory and the program counter is incremented by 1 to 8002H. After that the operand from the memory location 8001H is placed on the data bus.

Third Clock Cycle of M_2 During T_3 , the microprocessor reads the operand. \overline{RD} becomes high during T_3 and the memory is disabled.

The microprocessor also places the operand in the C register.

3.8.3 Timing Diagram of I/O Read

In an I/O read operation, the microprocessor reads the data from a specified input port or an input device. The I/O read operation is similar to memory read cycle except the control signal IO/\overline{M} . In a memory read cycle, IO/\overline{M} is low but IO/\overline{M} is high in case of I/O read cycle operation because the signal IO/\overline{M} goes high in case of I/O read.

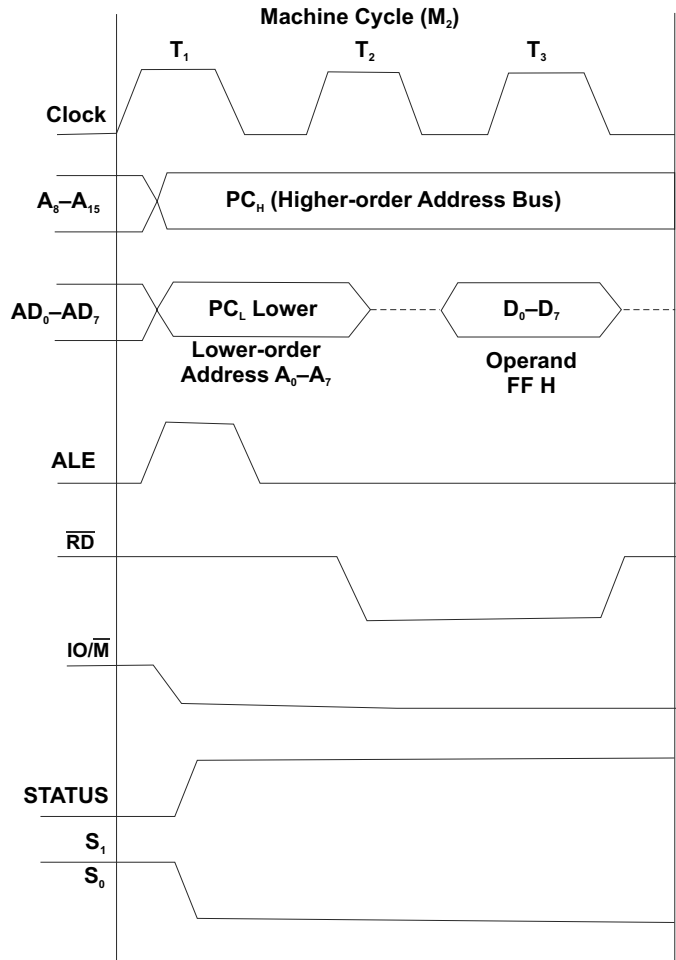


Fig. 3.11 Timing diagram of memory read operation

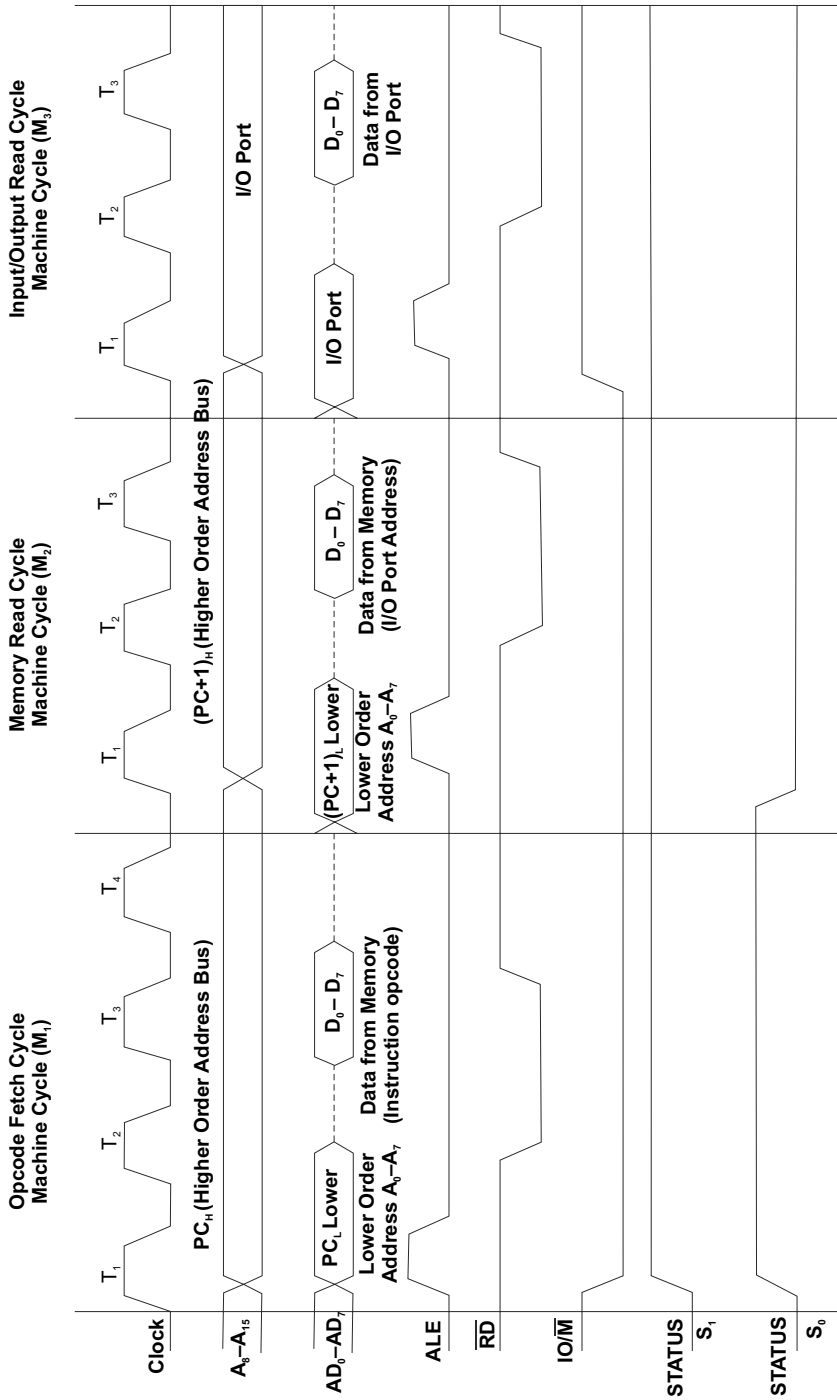


Fig. 3.12 Timing diagram of IN port address (IN 00H)

The timing diagram of I/O read operation is shown in Fig. 3.12. In this case, the address on the A-bus is for an input device. As I/O device or I/O port address is only 8 bits long, the address of I/O device or I/O port is duplicated on both higher-order address bus A_8-A_{15} and low order address bus AD_0-AD_7 .

For I/O read operation, the IN instruction is used. One example is IN 00. This is a two-byte instruction. The code of this instruction is DB, 00 where DB is for IN and 00 is the input port address.

Memory location	Opcode	Mnemonics
8000H	DB H	IN 00 H
8001H	00 H	

This instruction requires three machine cycles for execution. The first machine cycle is opcode fetch cycle, and the second machine cycle is a memory read cycle to read the address of input device or input port. In the third machine cycle, the I/O read operation is performed, it means that the data to be read from the input device or input port. After execution of this instruction, the data is placed in the accumulator. The opcode fetch cycle and memory read cycle are exactly similar to MVI C, FF H instruction. Figure 3.13 shows the machine cycle M_3 of I/O read operation and it is explained below:

T_1 State of M_3

- ◆ CPU places the address of I/O port or input-output peripheral devices.
- ◆ ALE signal is high.
- ◆ IO/\overline{M} becomes high to perform I/O operation.

T_2 State of M_3

- ◆ \overline{RD} is low for read operation.

T_3 State of M_3

- ◆ CPU reads data from I/O devices and places in Register A through a data bus.
- ◆ \overline{RD} signal becomes high as I/O read operation has been completely performed.

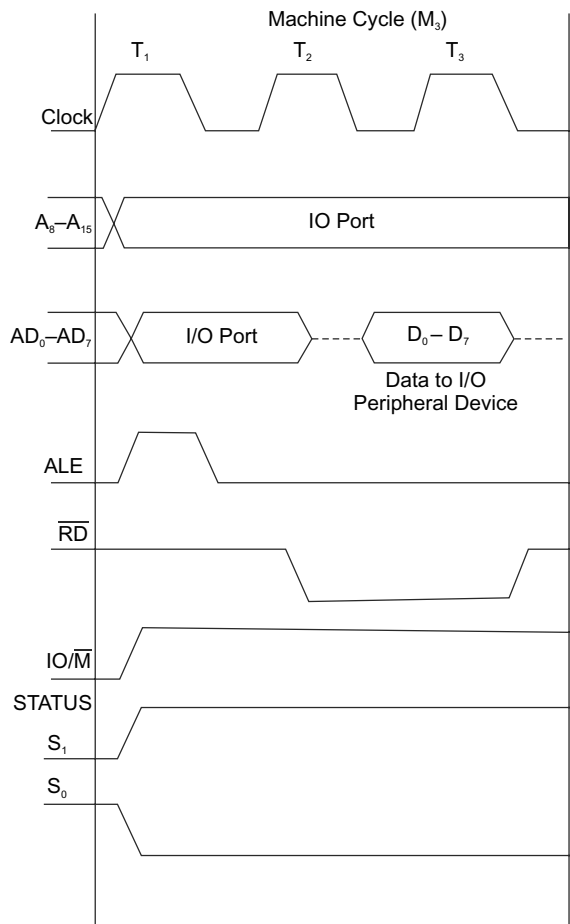


Fig. 3.13 Timing diagram of I/O read in machine cycle M_3

3.8.4 Timing Diagram of Memory Write

In a memory write operation, the microprocessor sends data from the accumulator or any general-purpose register to the memory. The timing diagrams of a memory write cycle is depicted in Fig. 3.14. The memory write cycle is similar to the memory read cycle, but there are differences on status signals. The status signals

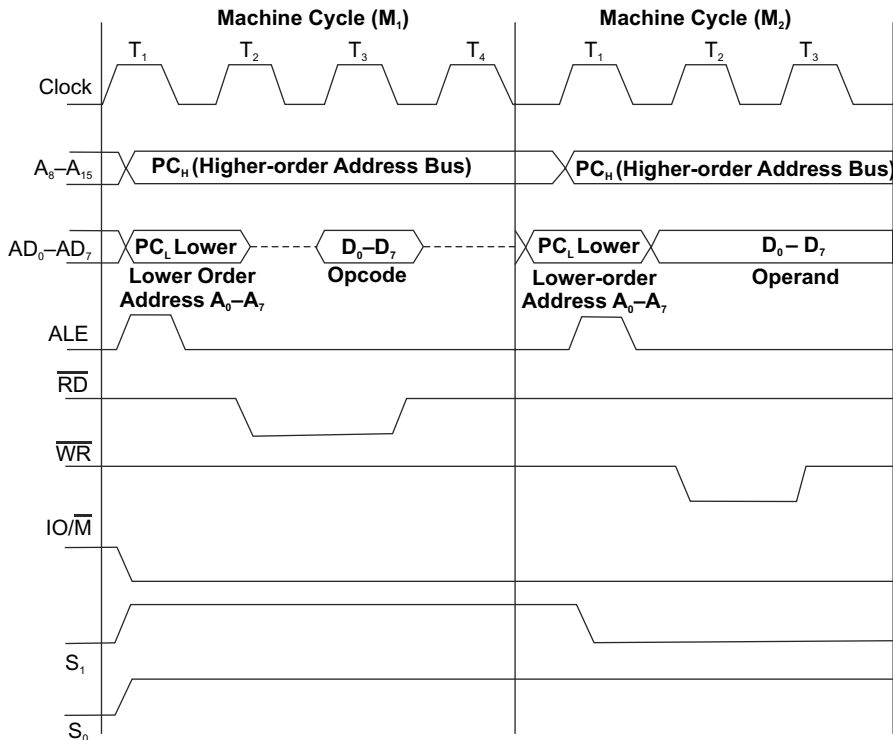


Fig. 3.14 Timing diagram for memory write operation

$S_0=1$ and $S_1=0$ and write \overline{RW} is low during T_2 of machine cycle M_2 which indicates that the memory write operation is to be performed.

During T_2 of the machine cycle M_2 , the lower-order address bus AD_0-AD_7 is not disabled as the data to be sent out to the memory, which is placed on the lower-order, address bus. When \overline{WR} becomes high in T_3 of machine cycle M_2 , the memory write operation will be terminated. The following instructions use the memory write cycle: MOV M, B; MOV M, A and STA 8000 H, etc. The timing diagram of STA 8000 H is given in Example 3.1.

3.8.5 Timing Diagram of I/O Write

The microprocessor sends the content of the accumulator to an I/O port or I/O device in an I/O write cycle. The operation of an I/O write cycle is similar to a memory write cycle. But the difference between a memory write cycle and an I/O write cycle is that IO/\overline{M} becomes high in case of I/O write cycle. When IO/\overline{M} is high, the microprocessor locates the address of any output device or an output port. The address of an output device or an output port is duplicated on both higher-order address bus A_8-A_{15} and lower-order address bus AD_0-AD_7 .

The OUT instruction is used for I/O write operation. This is a two-byte instruction and it requires three machine cycles as depicted in Fig. 3.15. The first machine cycle is for opcode fetch operation and the second machine cycle is a memory read cycle for reading the address output device or output port from the memory. In the next step, the third machine cycle data will be written in the output device or output port. In other words, data to be send to the I/O device. The third machine cycle is explained below:

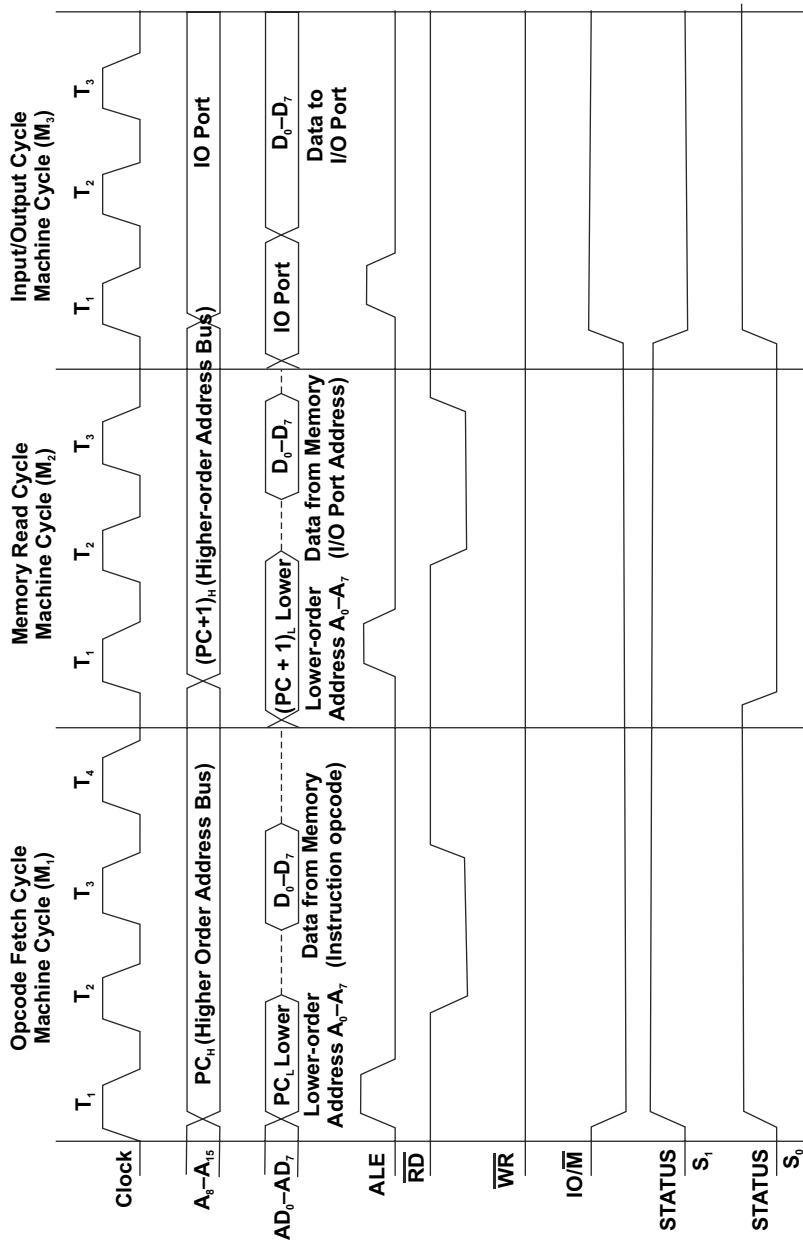


Fig. 3.15 Timing diagram of OUT Port Address

T_1 State of M_3

- ◆ CPU places the address of I/O port or input-output peripheral devices.
- ◆ ALE signal is high.
- ◆ $\overline{IO/\overline{M}}$ signal is also high in order to perform I/O operation.

T_2 State of M_3

- ◆ \overline{WR} becomes low for write operation.

 T_3 State of M_3

- ◆ CPU places the content of Register A in a data bus.
- ◆ Then it writes data to I/O port.
- ◆ \overline{WR} Signal becomes high as I/O read operation has been completed.

Example 3.1

Draw and explain the timing diagram for the instruction STA 9400H

Solution

Consider that instruction STA 9400H is stored at 9000H, 9001H and 9002H memory locations as given below:

Memory location	Opcode	Mnemonics
9000 H	32 H	STA 9400H
9001 H	00 H	
9002 H	94 H	

STA 9400H is a three-byte instruction. Figure 3.16 shows the timing diagram of STA 9400H. This instruction requires four machine cycles as depicted in Fig. 3.16. The first machine cycle M_1 is opcode fetch cycle to read the opcode from 9000H memory location. The memory read cycle machine cycle M_2 is used to read the lower-order address from memory location 9001H. The machine cycle M_3 is also a memory read cycle to read the higher-order address from 9002H. The last machine cycle M_4 can store the contents of Register A at the specified memory location 9400H. Therefore, machine cycle M_4 is memory write cycle.

The opcode fetch cycle and first memory read cycle is same as timing diagram of MVI A, FFH. In STA 9400H instruction, the second memory read cycle is used to read the higher-order address. In this section, the operation of M_4 has been explained below:

 T_1 State of M_4

- ◆ CPU places the content of program counter on the address bus during T_1 of machine cycle M_4 . Here, PCL content 00H and PCH content is 94H.
- ◆ ALE signal is high.
- ◆ IO/\overline{M} signal is low so that program counter content locates the memory location.

 T_2 State of M_4

- ◆ CPU places the content of Register A on the data bus and \overline{WR} becomes low so that a write operation will be performed.

 T_3 State of M_4

- ◆ The content of data bus will be written in the specified memory location 9400H.
- ◆ Then \overline{WR} signal will be changed from low to high as I/O read operation has been completed.

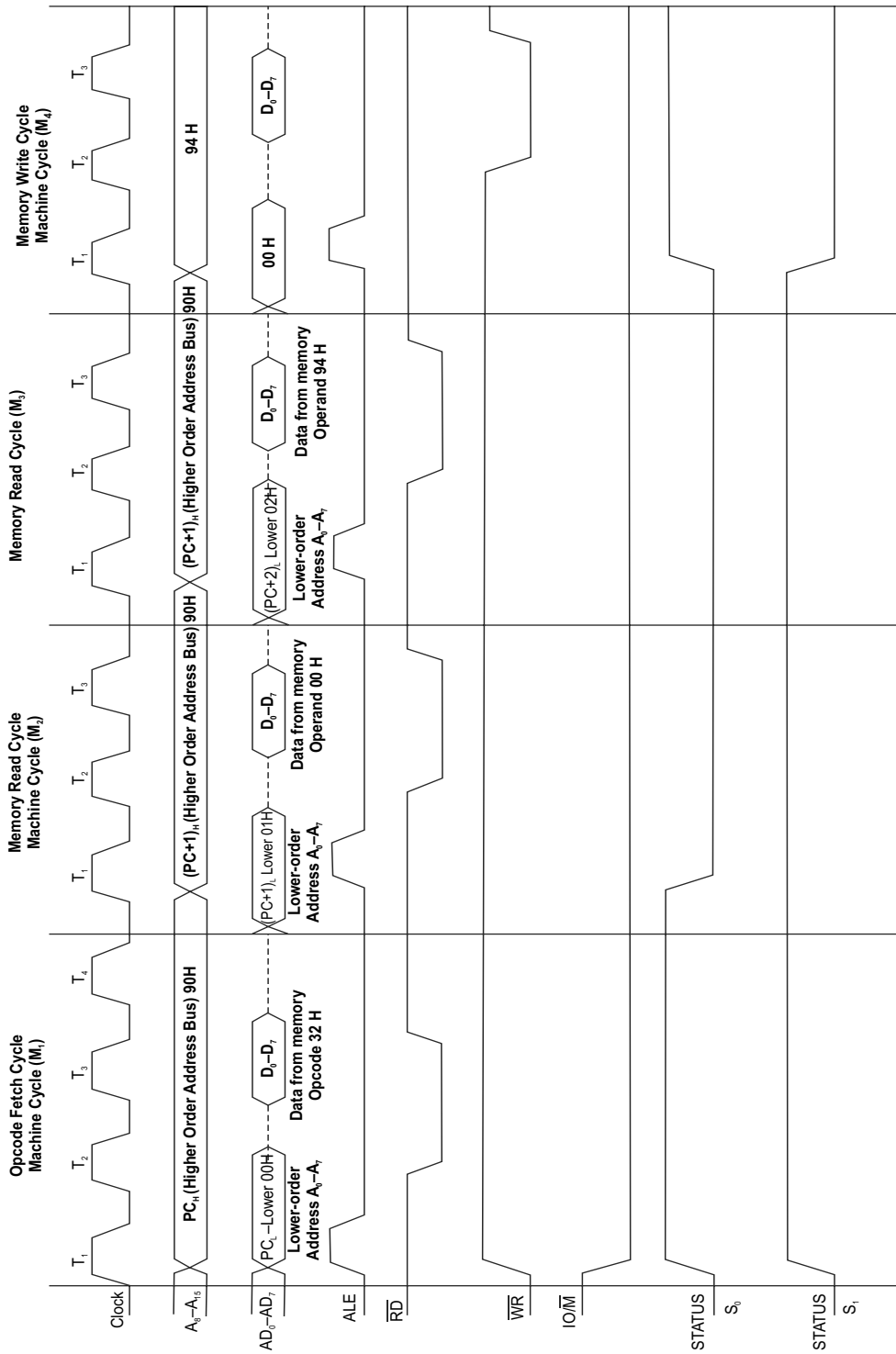


Fig. 3.16 Timing diagram of STA 9400H

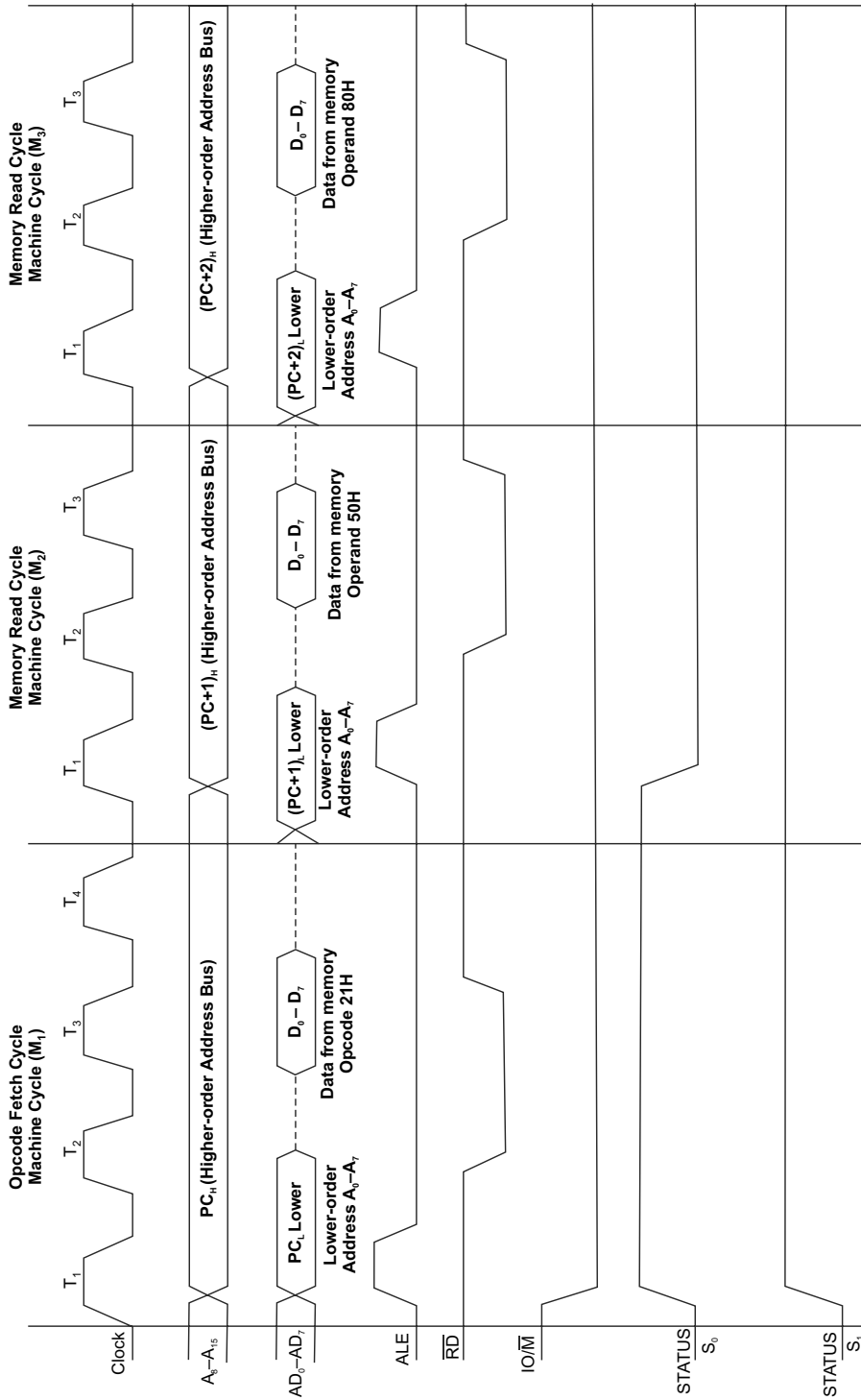


Fig. 3.17 Timing diagram of LXI H, 8050H

Example 3.2

Draw and explain the timing diagram for the instruction LXI H, 8050H.

Solution

The LXI H, 8050H is a three-byte long instruction and is stored at the memory location 8000H, 8001H and 8002H. The opcode for LXI H is 21H and the address is 8050H. The timing diagram of this instruction is depicted in Fig. 3.17. It is clear from Fig. 3.17 that this instruction requires the three-machine cycle. The first machine cycle M_1 is a fetch cycle and other two consecutive machine cycles, M_2 and M_3 are memory read cycles.

In the fetch cycle, the opcode for LXI H, 21H is fetched from the memory location 8000H and program counter is incremented by 1 to 8001H. In the second machine cycle M_2 , 8 LSBs of the 16-bit data (8050H) are read. Similarly in the third machine cycle M_3 , 8 MSBs of the 16-bit data (8050) are read. After execution of this instruction, 50H is stored in Register L and 80H is also stored in Register H.

Memory location	Opcode	Mnemonics
8000 H	21 H	LXI H, 8050H
8001 H	50 H	
8002 H	80 H	

SUMMARY

- In this chapter, different types of 8085 microprocessor-addressing modes have been explained.
- The 8085 microprocessor’s instruction set can be divided into five different groups such as data transfer, arithmetic, logical, branching, stack I/O and machine-control operations.
- Instructions of all categories have been described in detail in this chapter. During execution of an instruction, the CPU should perform the following operations: opcode fetch from memory through data bus, read operand/data from memory and perform the specified operation with data.
- All operations are synchronized with the system clock. The timing diagram of any instruction shows the sequence of different operations with respect to the clock. Instruction cycle is the time required to complete the execution of an instruction. In this chapter, instruction cycle of one byte, two bytes and three bytes have been incorporated with timing diagrams.

MULTIPLE-CHOICE QUESTIONS

- 3.1 The instructions of 8085 microprocessor has been classified into
- 3.2 XCHG is a/an
- (a) four groups of instructions
 - (b) five groups of instructions
 - (c) six groups of instructions
 - (d) seven groups of instructions
 - (a) data transfer instruction
 - (b) arithmetic instruction
 - (c) logical instruction
 - (d) I/O and stack pointer instruction

- 3.3 DAD is a/an
 (a) data-transfer instruction
 (b) arithmetic instruction
 (c) logical instruction
 (d) I/O and stack pointer instruction
- 3.4 When PUSH B instruction is executed,
 (a) the content of Register B and C register is copied in the stack
 (b) the content of Register B and Register C is stored in the stack and the registers are cleared
 (c) Register B and Register C are cleared by data transfer instructions
- 3.5 ORA is a/an
 (a) data transfer instruction
 (b) arithmetic instruction
 (c) logical instruction
 (d) I/O and stack pointer instructions
- 3.6 CALL and RET are used in
 (a) data transfer instructions
 (b) arithmetic instructions
 (c) logical instructions
 (d) Branch control instructions
- 3.7 HLT is a/an
 (a) data transfer instruction
 (b) arithmetic instruction
 (c) logical instruction
 (d) Machine control instruction
- 3.8 A one-byte instruction has
 (a) opcode and an operand
 (b) opcode only
 (c) opcode and two operand
 (d) operand only
- 3.9 A two-byte instruction consists of
 (a) opcode and an operand
 (b) opcode only
 (c) opcode and two operand
 (d) operand only
- 3.10 A three-byte instruction should have
 (a) opcode and an operand
 (b) opcode only
 (c) opcode and two operand
 (d) operand only
- 3.11 IN 00H is an instruction of
 (a) direct addressing mode
 (b) indirect addressing mode
 (c) register addressing mode
 (d) immediate addressing mode
- 3.12 STA 9000H is an instruction of
 (a) one byte
 (b) two bytes
 (c) three bytes
 (d) four bytes
- 3.13 MOV is an instruction of
 (a) direct addressing mode
 (b) indirect addressing mode
 (c) register addressing mode
 (d) immediate addressing mode
- 3.14 CMA is an instruction of
 (a) direct addressing mode
 (b) indirect addressing mode
 (c) register addressing mode
 (d) immediate addressing mode
- 3.15 SUB A instruction in the 8085 microprocessor
 (a) resets the zero flag
 (b) sets the zero flag
 (c) sets the carry flag
 (d) resets the auxiliary carry flag
- 3.16 LXI H, 2500H is an instruction of
 (a) direct addressing mode
 (b) indirect addressing mode
 (c) register addressing mode
 (d) immediate addressing mode
- 3.17 CALL 8000H is an instruction of
 (a) direct addressing mode
 (b) indirect addressing mode
 (c) register addressing mode
 (d) immediate addressing mode
- 3.18 MOV A, C is executed by
 (a) one machine cycle
 (b) two machine cycles
 (c) three machine cycles
 (d) four machine cycles

- 3.19 MOV B, C is executed by
 (a) one machine cycle
 (b) two machine cycles
 (c) three machine cycles
 (d) four machine cycles
- 3.20 SUB B instruction in 8085 microprocessor
 (a) resets the carry and sign flag
 (b) sets the zero and parity flag
 (c) sets the zero and carry flag
 (d) can modify all flags according to result
- 3.21 LDA 9500H is executed by
 (a) one machine cycle
 (b) two machine cycles
 (c) three machine cycles
 (d) four machine cycles
- 3.22 STA 8000H is executed by
 (a) one machine cycle
 (b) two machine cycles
 (c) three machine cycles
 (d) four machine cycles
- 3.23 OUT 02H is executed by
 (a) one machine cycle
 (b) two machine cycles
 (c) three machine cycles
 (d) four machine cycles

SHORT-ANSWER-TYPE QUESTIONS

- 3.1 What are the various types of data formats for 8085 instructions? Give a list of examples for each type of data format.
- 3.2 Write the addressing modes of the following instructions:
 (a) MOV A, B (b) MVI C, FFH (c) LXI H, 2500H (d) RAL
- 3.3 Define opcode and operand, and specify the opcode and the operand in the instruction MVI C, 45H.
- 3.4 Classify 8085 instructions in various groups. Give a list of examples of instructions for each group.
- 3.5 Write the differences between
 (a) RAL and RLC (b) RAC and RAR (c) PUSH and POP
- 3.6 Define: instruction cycle, machine cycle, and T-state.
- 3.7 Write instructions for the following operations:
 (a) Clear accumulator
 (b) Exchange the content of DE and HL register pair
 (c) Logical AND memory with accumulator
 (d) Decrement DE register pair by 1
- 3.8 Distinguish between JMP and CALL instructions

REVIEW QUESTIONS

- 3.1 What are the types of addressing modes of Intel 8085? Explain any one addressing mode with suitable examples.
- 3.2 Explain the operation of the following instructions when they are executed
 (a) LXI rp, data (b) STA addr (c) DAD rp (d) DAA
 (e) CMP M (f) RAR (g) PUSH rp (h) POP rp

- 3.3 Find the machine code for the instruction MOV H, A if the opcode = 01, the register code for H = (100)₂, and the register code for A = 111.
- 3.4 Draw and explain the timing diagram for opcode fetch operation.
- 3.5 Draw and explain the timing diagram for memory read operation.
- 3.6 Draw and explain the timing diagram for memory write operation.
- 3.7 Draw and explain the timing diagram for I/O read operation.
- 3.8 Draw and explain the timing diagram for I/O write operation.
- 3.9 Draw the timing diagram for the execution of the following instructions:
 (a) MVI A, FFH (b) SUB C (c) LXI H, 5000H (d) MOV A, B
- 3.10 Draw I/O read and write machine cycles. Compare the two machine cycles.
- 3.11 Compare the following instructions:
 (a) MOV A, M and LDAX D (b) CMP B and SUB B (c) XCHG and XTHL (d) DAD and DAA
- 3.12 Determine the time required to execute the following two instructions if the microprocessor clock frequency is 1 MHz.
 MOV A, B 5 T-states
 MOV B, C 5 T-states

Answers to Multiple-Choice Questions

-
- 3.1 (b) 3.2 (a) 3.3 (b) 3.4 (b) 3.5 (c) 3.6 (d) 3.7 (d) 3.8 (b) 3.9 (a)
 - 3.10 (c) 3.11 (a) 3.12 (c) 3.13 (c) 3.14 (b) 3.15 (b) 3.16 (d) 3.17 (d) 3.18 (a)
 - 3.19 (a) 3.20 (d) 3.21 (d) 3.22 (d) 3.23 (c)

Chapter 4

Assembly–Language Programs of the 8085 Microprocessor

4.1 INTRODUCTION

A program is a sequence of instructions which operate with operands or data. The program may be written in any one of the available languages to achieve the objective of the user. When a programmer writes a program for a particular problem, the following five steps are followed:

Step 1 *Define the problem*

Before starting, it is required to understand the problem completely and assume all inimical conditions.

Step 2 *Plan the solution*

Break the problem into modular form and determine how the modules are logically linked.

Step 3 *Code the program*

Translate the logical solution of each module in assembly or any programming language which the microcomputer can understand.

Step 4 *Test the program*

After writing the program, implement/test the program in a microcomputer system.

Step 5 *Documentation*

All related matters must be documented, as we may not always remember the most important steps that we did during development of the program.

The development of the program depends on the skill of the programmer as well as the complexity of the problem. Generally, the program is fed into the microcomputer through input devices such as the keyboard and is stored in the memory of the microcomputer. The 8085 microprocessor is able to understand instructions which are written in 0s and 1s. When a program is written using 0s and 1s, the program is known as a machine-language program. But it is very difficult for a programmer to write a program in machine language. The other way of writing a program is using mnemonic operation codes in hexadecimal, octal or binary notations, which is known as assembly language. In assembly language, when a program is executed, instructions are converted/translated into machine code. The translator which translates/converts an assembly-language program into machine language is known as an *assembler*. In this chapter, assembly-language programs are discussed in detail.

4.2 MACHINE LANGUAGE

Programmers write instructions in various programming languages. Some programs are directly understandable by the computer and other programs require intermediate translation steps. Nowadays hundreds of computer languages are available for use in solving different problems. These programming languages are classified into three general types as given below:

- ◆ Machine languages
- ◆ Assembly languages
- ◆ High-level languages

Machine language is the ‘natural language’ of computers. Machine-language programs are usually written in binary code. Therefore, 0s and 1s are used in a machine-language program. Machine languages are machine-dependent, that is, a particular machine language can be used on only one type of computer. In this language, a specific binary code is used for each instruction. For example, to copy data from Register A to B, the binary code 0100 0111 is used. Similarly, different binary codes are available in the 8085 microprocessor for different operations such as addition, subtraction, increment, decrement, rotate, and compare. But it is very difficult to write machine-level programs. The program can be simplified by converting binary code to hexadecimal.

Machine language has the following advantages:

1. This is suitable for small and simple programs.
2. Program execution is very fast and requires less computation time.
3. Generally, this language is suitable for prototype applications as the final product.

The disadvantages of machine-language programs are the following:

1. A program written in machine code is a set of binary numbers. Therefore, program writing is difficult and time consumable.
2. It is also very difficult to understand a program which is written in machine language or hexadecimal form.
3. Since a program is always written in 0s and 1s, each bit has to be entered individually. Thus, time taken for data entry becomes very slow and tiresome.
4. There is always some possibility of errors in writing programs. Even a single bit error in any instruction can generate unsatisfactory results.
5. Such programs tend to be very long.

4.3 ASSEMBLY LANGUAGE

To overcome the limitations of machine languages, assembly language was developed. In this language, machine-level instructions are replaced by mnemonics. For example, ADD represents addition, SUB represents subtraction, INC for increment, RAL for rotate left, and CMP for compare. These instructions are known as *mnemonics*. A program written in mnemonics is called an assembly-language program. It is easier for a programmer to write programs in assembly language compared to those in machine language. It is also easier to understand an assembly-language program. Such programs are microprocessor-specific.

Assembly-language programs are translated to machine-level programs using a translator program known as assembler, as shown in Fig. 4.1.

Assembly language has the following advantages:

1. It is easy to write.
2. It is easy to understand.

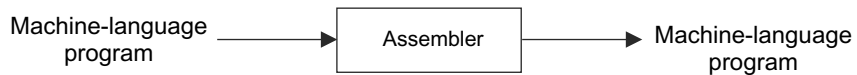


Fig. 4.1 Translation of assembly language programs to machine language programs

3. Assembly-language programs produce faster results.
4. It is suitable for real-time control and industrial applications.
5. It requires less computation time.

The disadvantages of assembly language compared to high-level languages are given below:

1. An assembly language is microprocessor specific. Detailed knowledge of the particular microprocessor is required to write a program. The programmer should know all about registers and instructions of the microprocessor.
2. An assembly language program is not portable, as a program written for one microprocessor may not be used in other processors.
3. Assembly-language program writing is difficult and time consuming compared to that of high-level languages.

4.4 HIGH-LEVEL LANGUAGE

The demerits of assembly languages are overcome by using high-level languages. High-level languages can improve the readability by using English words which make it easier to understand the code and to sort out any faults in the program. In addition, the high-level languages relieve the programmer of any need to understand the internal architecture of the microprocessor. Ideally, the programmer need not even know what processor is being used. For programs written in high-level languages, any type of computer can be used easily. Therefore, the program should be totally portable. The programs written in high-level languages are very easy to write and fast to execute but a compiler is required to translate high-level language into machine codes, as the microprocessor can understand only machine code, 1 or 0.

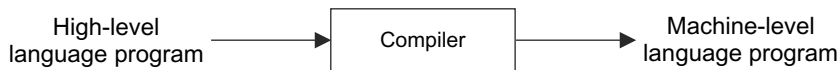


Fig. 4.2 Translation of high-level language to machine-level language

Translators

Translators translate high-level programming language to binary codes and make the program understandable for the computer. There are two general types of translators, namely, compiler and interpreter. The *compiler* translates an entire program at one time and then executes it. The interpreter also translates one program line at a time while executing. The differences between compilers and interpreters are given below:

Compiler	Interpreter
Compiled programs execute much faster.	Interpreted programs are slower because translation takes times.
Compilation is usually a multi-step process.	Interpretation translates in one step.
Compilers do not require space in memory when programs run.	Interpreters must be in memory while a program is running.
It is more costly than an interpreter, and suitable for a larger system	It is cheaper and suitable for a smaller system.

Instructions written in high-level languages are called *statements*. High-level languages allow programmers to write instructions that look almost like everyday English and contain commonly used mathematical notations.

High-level languages are much more desirable from the programmer's point of view. Translator programs called compilers convert high-level language programs into machine language. FORTRAN, COBOL, BASIC, PASCAL, ALGOL, PL/M, C/C++ and Java are among the most powerful and most widely used high-level languages. The features of some such high-level languages are discussed in this section.

Advantages of High-Level Languages

High-level languages have the following advantages:

1. In high-level languages the programs are written using instructions and each instruction is very clear about performing a specified operation.
2. Writing programs in high-level languages is very easy and fast. These languages are suitable for large programs and for developing large projects.
3. Programs are portable in high-level languages and can be executed in any standard.
4. Complex mathematical computation is possible in these languages.
5. Report writing and documentation are simple in high-level languages.
6. The program is independent of the internal architecture of the microprocessor structure. The programs are problem-oriented and can run in any standard computer.

Disadvantages of High-Level Languages

High-level languages have the following disadvantages:

1. Each high-level language has a standard syntax and specified rules to write programs.
2. Each statement of a high-level language is equivalent to many instructions in machine language. Therefore, the execution time of programs written in high-level languages is more and to reproducing results, also takes more time. So the high-level language speed is slow compared to that in assembly language.
3. Hardware and software supports are required.
4. Large volume of data needs to be processed in high-level language and programs in high-level languages require large memory. Hence, memory utilization is less.
5. To translate a high-level language program into a machine-language program, a compiler is required. Most compilers are costly.

4.4.1 Fortran

FORTRAN stands for FORmula TRANslator. IBM introduced this language in the mid-1950s' primarily for scientific and engineering applications. This was the first high-level language to gain widespread acceptance. FORTRAN has built instructions to handle most scientific formulas such as sine, cos, tan, etc., which are extremely difficult to write in assembly language.

A FORTRAN program must first be converted into the machine code understood by the microprocessor. If the FORTRAN program can be converted to machine code, then it follows that the program could have been written in machine code in the first case. It is just a matter of saving an enormous amount of work.

In 1958, a re-defined language called FORTRAN 2 was developed by small additions of new 'dialects' and alterations. After that, FORTRAN 3 and 4 were developed. In the early 60s, FORTRAN 4 was very popular. In 1966, FORTRAN 66 was designed with new dialects. In 1977, the American National Standards Institute (ANSI) published a standard for FORTRAN. This standard form of the language is known as FORTRAN 77 is industry and education institutions.

The absolutely final and totally definitive FORTRAN is FORTRAN 90. Presently, FORTRAN 95 versions are also available. Each new version included extra features rather than changes. This ensured that all previous versions were incorporated in the next versions, and even a system designed for FORTRAN 90 can still use the original FORTRAN.

4.4.2 Basic

BASIC stands for Beginners' All-purpose Symbolic Instruction Code. In the early days, the emphasis was given on 'easy to learn' and 'use of minimum memory'. These two attributes were very useful in colleges but not in the real world. In 1960, a simplified language was developed in Dartmouth College, USA. This language was known as BASIC (Beginners' All purpose Symbolic Instruction Code). It was based on FORTRAN and was designed as a simpler language and easier to learn.

The most popular versions are Q-BASIC, GW BASIC and Quick Basic. Q-Basic is a menu-oriented language. It provides a Graphical User Interface (GUI). In 1991, a new version called Visual Basic appeared which has features to make the generation of Windows programs much easier.

To save memory, BASIC was designed as an interpreted language. An interpreter rather than a compiler carried out the conversion of the source code to the object code. The compiler converts the whole program into object code, stores it, and then runs the program. The interpreter takes a different approach. The first instruction in the program is converted to source code and it is then executed. The next item of the source code is then converted to object code and then run, and so on, right through the program. The interpreter never stores the whole of the machine-code program. It just generates it, a line at a time, as needed.

4.4.3 Cobol

COBOL stands for COmmon Business-Oriented Language. FORTRAN and BASIC did not make enormous steps towards employing normal English-language phrases. In 1959, COBOL was introduced by the US Defense Department to make easy-to-read business programs. Its purpose was information handling. It proved to be successful at this and spread from the US Navy where it kept records of stocks and supplies, to the business world. COBOL was designed, more in hope than reality, to be easily read by non-programmers.

Large businesses handle enormous amounts of information every day. Just imagine the amount of information involved in a few everyday activities like handling bank accounts, credit cards, stock information, prices, dates, our card numbers, and names. All such data are transmitted to the national card centre and our accounts are amended. None of these transactions involve particularly complicated mathematics. The calculations are basically addition and subtraction of totals.

COBOL has survived by meeting a specific need and has had a series of upgraded standard versions. They refer to the date of adoption: COBOL 60, COBOL 74, COBOL 85 and COBOL 97.

4.4.4 Pascal

PASCAL was first designed by Nicklaus Wirth in Switzerland in 1971. This is mostly an academic language but was also suitable for scientific applications. During learning other languages, a short course of Pascal is often employed as an introduction. Pascal is used because it is very useful for beginners. Pascal is a structured language. A structured program consists of a series of separate, self-contained units, each having a single starting point and a single exit point. The program layout looks like a simple block diagram with all the blocks arranged one under the other. Every unit can be isolated from the others.

This is a compact language and its compiler is quite suitable for a smaller system. Program design and debugging in Pascal is simpler. This language produces a very efficient machine-code program when it is

compiled. Pascal is several times faster than BASIC or FORTRAN. This is more versatile than BASIC and more modular than FORTRAN. Pascal can handle numbers, vectors, matrices, strings of characters, sets, records, files and lists.

4.4.5 C

The language C was invented a year after Pascal and allowed all the good practice programming methods of Pascal with a few extras. The main difference between C and Pascal is that C is a lower-level language than Pascal. It is a powerful, flexible and very efficient language used in writing operating-system code and software package. Its advantage is that it can control low-level features like memory loading as in assembly language. It has many high-level features and low-level facilities when we require them and can produce very compact, and therefore fast, code.

4.4.6 C++ and Object-Oriented Programming (OOP)

The new version of C which incorporates all features of the C language and adds a new feature, is known as Object-Oriented Programming (OOP). This version of C is known as C++. Object-oriented programming is a different approach to programming. In these languages, a number of objects may be of any form of data such as a diagram on a monitor screen, a block of text or a complete program. Once objects are defined, some storage areas are allocated for an entire object at the same time.

For example, suppose you have a square/circle on the monitor screen and you wish to move it. There are different ways to do this. For this, take each point on the screen and shift its position, and hence rebuild the square/circle in a different position. The object-oriented approach must define the shape as an object, and then instruct the object to move. We use a mouse to take hold of an object, say a menu, and simply drag it to a new position. The menu is being treated as a single lump, which is an example of an object.

4.4.7 LISP

LISP was designed in the early 1960s by an American, John McCarthy. LISP (LISt Processing) involves the manipulation of data, which are entered by the keyboard. This language is suitable for artificial intelligence, searching, handling and sorting long strings. LISP is a function-oriented language. This means that functions such as add, subtract or more complex combinations can be easily handled through this.

A sample of LISP is a list consisting of a series of 'members' separated by spaces and enclosed in brackets. A simple function defined as (PLUS 6 4) would return 10 by adding the two numbers. Since LISP is an interpreted language, the program is executed one step at a time, and so inputted values are used they are entered.

4.4.8 APL

The APL stands for 'A Programming Language'. This is an interpreted language developed by IBM around 1962. This is only used for handling numerical data. Actually, it is a curious mixture of LISP and FORTRAN. It combines the function orientation of LISP and the procedural mathematics of FORTRAN. This language allows users to define functions and has a large library of solutions to common problems. This requires a special keyboard and terminal as special symbols are used for sin, cos and tan, etc.

4.4.9 ADA

This language was named after Lady Augusta Ada Lovelace. CII-Honeywell-Bull of France introduced it in 1980. It is a multipurpose language suitable for both scientific and business applications. This language combines the best features of Pascal, ALGOL and PL. It also has the feature of real-time multitasking required in control applications. It has complex subroutines, which are used as software components while writing programs. Actually, this is a 'do-everything language'.

4.4.10 PROLOG

PROLOG stands for PROgramming by LOGic. It is a ‘declarative’ language. It was first developed at the University of Marseillers, France, in 1972. Other versions were developed, such as DEC10, and IC PROLOG in the UK and the US. PROLOG is a nonprocedural language. This is capable of handling large databases and can be used in the development of artificial intelligence and expert systems.

4.5 STACK

The stack is a group of memory locations in Read/Write (R/W) memory of any microcomputer and is used to store the contents of the register, operand and memory address. The starting location of the stack is defined by loading a 16-bit address into the stack pointer, a reserved space usually at the top of memory map. Theoretically, the size of the stack is unlimited, but it is restricted only by the available R/W memory in a microcomputer system. The stack can be initialized anywhere in the user-memory map, but the stack is initialized at the highest user-memory location so that there will not be any interface with the program.

In 8085 microprocessor systems, the beginning of the stack is defined in the program by using the instruction LXI SP, 16-bit. The LXI SP is a 16-bit state that loads the 16-bit address into the stack pointer register. Then contents of register pairs (BC, HL, etc.) can be stored in two consecutive stack memory locations by using the instruction PUSH and can be retrieved from the stack into register pairs by using the POP instruction. The microprocessor keeps track of the next available stack memory location by incrementing or decrementing the address in the stack pointer. The address in the stack pointer (register) always points to the top of the stack and indicates that the next memory location (SP-1) is available to store information.

This method of information storage looks like the process of stacking books one above another. Therefore, data is always retrieved from the top of the stack. So data are stored in the stack on Last-In-First-Out (LIFO) principle. The syntax of stack instructions to store data on and retrieve data from the stack are given below:

<i>Opcode</i>	<i>Operand</i>	<i>Description</i>
LXI	SP, 16-bit	Load 16-bit address into the stack pointer register. This is a load instruction, similar to other 16-bit load instructions discussed previously.
PUSH PUSH	RP R	This is a 1- or 2-byte instruction and copies the contents of the specified register pair or index register onto the stack as described below. Instructions for four register pairs and index registers are listed here.
PUSH PUSH PUSH PUSH	PSW BC DE HL	The instruction first decrements the stack pointer (register) and copies the higher-order byte of the register pair or the index register on the stack location SP – 1. Then it again decrements the stack pointer and copies the lower-order byte of the register pair or the index register onto the stack location SP – 2.
POP POP	RP R	This is a 1- or 2-byte instruction and copies the contents of the top two locations of the stack into the specified register pair or the index register.
POP POP POP POP	PSW BC DE HL	First, the instruction copies the contents of the stack indicated by SP into the lower-order register (for example, register C of the BC pair) or as a lower-order byte into the index register and then increments the stack pointer to SP + 1. It copies the contents of the SP + 1 location into the high order register (for example, Register B of the BC pair) or as a higher-order byte into the index register and increments the stack pointer to SP + 2.

Figure 4.3 shows a stack and stack top location. The SP register holds the address of the stack top location, i e , 8004 H.

For example, a program is stored in memory locations starting from 7000H as given below, and the stack is initialized at the location 8004H.

Program

```

7000      LXI SP, 8004H
7003      PUSH DE
7004      POP DE
7005      HALT
    
```

The position of a stack before PUSH operation is depicted in Fig. 4.3. When the program is executed, the contents of the register pair HL must be pushed to the stack. After the PUSH operation, the stack position is changed to 8001H. In the same way, a POP operation is used to transfer the contents from the stack to the register. The stack position before and after the PUSH operation has been given in Fig. 4.4 (a) and Fig. 4.4 (b) respectively. Figures 4.5 (a) and 4.5 (b) show the stack position before and after the POP operation correspondingly.

From the above example, the following points can be summarized:

1. During the execution of a program, a 16-bit address (8004H) is stored in the stack pointer register. The stack space grows upward in the numerically decreasing order of memory addresses. The contents of HL register pairs can be stored beginning from the next location (SP-1).

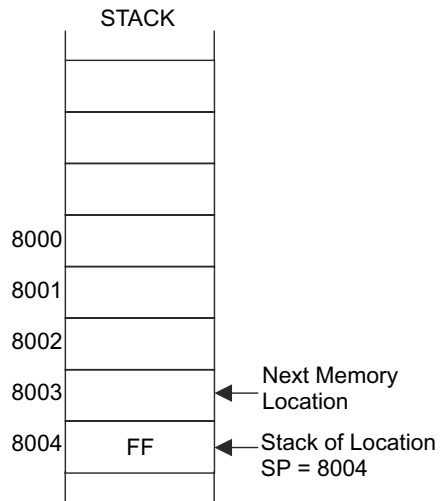


Fig. 4.3 Stack and stack top location

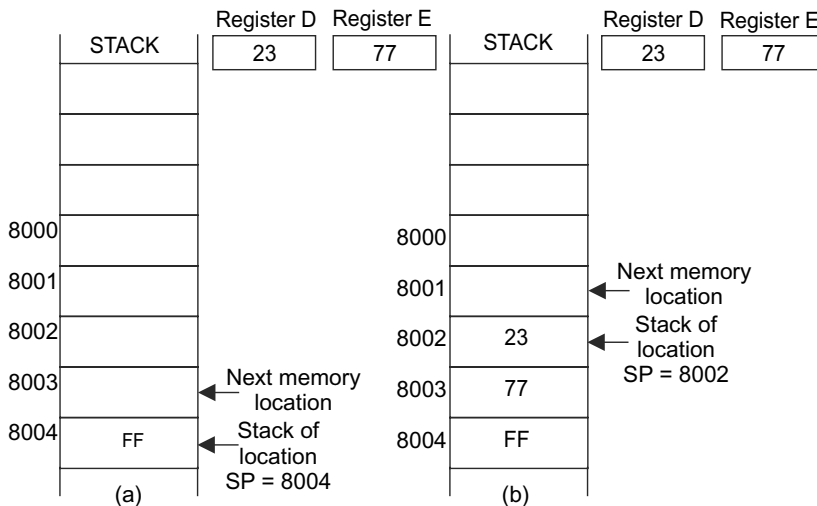


Fig. 4.4 (a) Stack before PUSH operation (b) Stack after PUSH operation

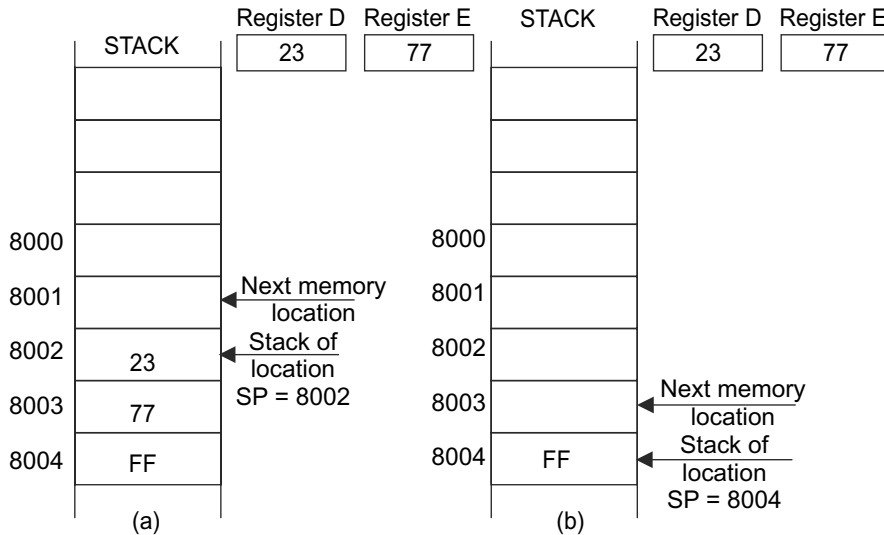


Fig. 4.5 (a) Stack before POP operation (b) Stack after POP operation

2. The PUSH instructions are used to store contents of register pairs on the stack, and the POP instructions are used to retrieve the information from the stack. The address in the stack pointer register always points to the top of the stack, and the address is decremented as information is stored or retrieved, respectively.
3. The storage and retrieval of the content of registers on the stack should follow the LIFO (Last-In-First-Out) sequence.
4. Information in the stack locations may not be destroyed until new information is stored in that memory location.

4.6 SUBROUTINES

Some operations/functions are repeatedly performed in a main program like multiplication, division, and time delay between two operations, etc., Groups of instructions are written to perform these operations and these groups of instructions are known as *subroutines*, which are called by the main program whenever required. When a main program calls a subroutine, the program execution is transferred to the subroutine and after the completion of the subroutine, the program execution returns to the main program. The microprocessor uses the stack to store the return address of the subroutine. For example, generally, subroutine are written for sine, cosine, logarithms, square root, time delay, and multiplication functions in 8085 microprocessors.

A subroutine is implemented with two associated instructions, namely, *Call* and *Return*. *Call* is used to call a subroutine and the *Call* instruction is written in the main program. *Return* is used to return from the subroutine. and the *Return* instruction is written in the subroutine to return to the main program. When a subroutine is called, the contents of the program counter are stored on the stack, and the program execution is transferred to the subroutine address. When the *Return* instruction is executed at the end of the subroutine, the memory address stored in the stack is retrieved and the sequence of execution is resumed in the main

program. All types of CALL and RET instructions are explained in this chapter. The syntax of CALL and RET are given below:

Opcode	Operand	Description
CALL	16-bit	Call subroutine conditionally located at the memory address specified by 16-bit operand. This instruction places the address of the next instruction on the stack and transfers the program execution to the subroutine address.
RET		Return unconditionally from the subroutine. This instruction locates the return address on the top of the stack and transfers the program execution back to the calling program.

The general characteristics of CALL and RET instructions are given below:

1. The *Call* instructions are 3-byte instructions; the second byte specifies the lower-order byte, and the third byte specifies the higher-order byte of the subroutine address.
2. The *Return* instructions are 1-byte instructions.
3. A *Call* instruction must be used in conjunction with a *Return* instruction (conditional or unconditional) in the subroutine.

The following types of subroutines are generally used in microprocessors:

1. Multiple CALL subroutines
2. Nested subroutines
3. Multiple ending subroutines

4.6.1 Multiple CALL Subroutines

Figure. 4.6 shows the basic concept of multiple CALL subroutines. This is a subroutine called from many locations in the main program. For example, the DELAY routine is a multiple CALL subroutine. These types of routines are easy to trace and need minimal stack space. Initially, stack pointer content is XX55H so that the return address can be stored on the stack. When the CALL instruction starts to execute, the subroutine is called from the 8050 memory location. The return address is stored on the stack and the stack pointer is decrement by two locations to XX53H.

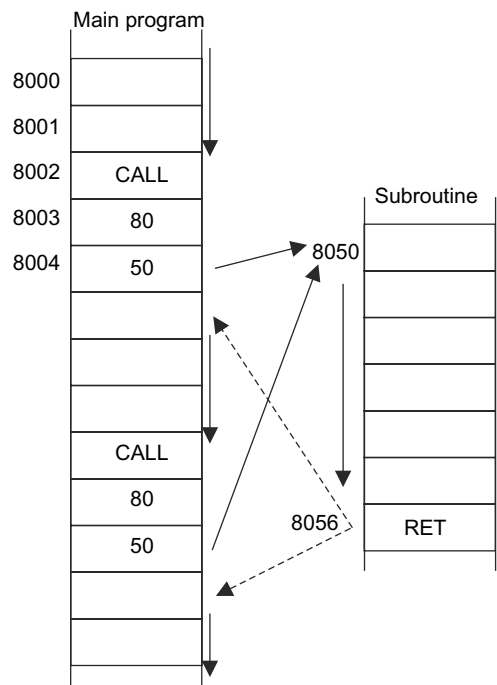


Fig. 4.6 Multiple CALL Subroutines

4.6.2 Nested Subroutine

When the subroutine is called by another subroutine, it is called a *nested subroutine*. When a subroutine calls another subroutine, all return addresses are stored on the stack. Therefore, only the number of available stack locations limits the extent of nesting. The structure of a nested subroutine is depicted in Fig. 4.7.

The main program calls Subroutine I from location 8050H. The address of the next instruction, 8053H, is placed on the stack, and the program is transferred to Subroutine I at 8150H. During the execution of Subroutine I, it calls Subroutine II from location 8190H. The address 8193H is placed on the stack, and the program is transferred to Subroutine II. The sequence of execution returns to the main program as shown in Fig. 4.7.

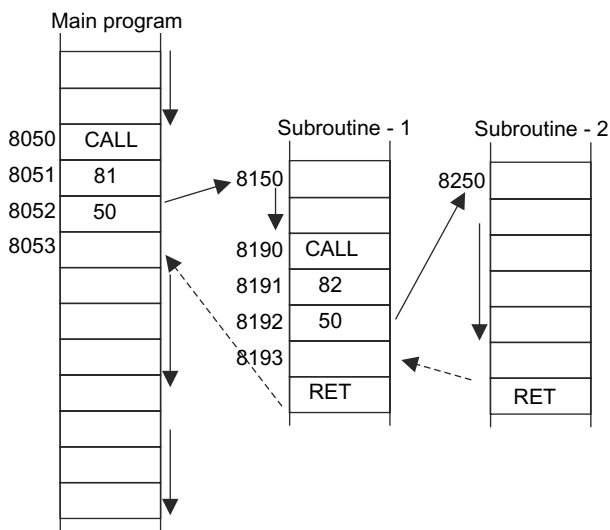


Fig. 4.7 Nested subroutines

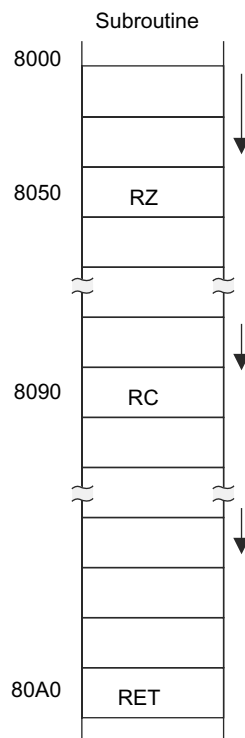


Fig. 4.8 Multiple ending subroutine

4.6.3 Multiple Ending Subroutines

When a subroutine can be terminated at more than one place, it is called a *multiple ending subroutine*, as illustrated in Fig. 4.8. The subroutine has conditional returns such as RET Z (RZ) – Return On Zero and RET C (RC) — Return on Carry. This subroutine has an unconditional return (RET). While the Z flag is set, the subroutine returns from location 8050H, and if the CY flag is set, it returns from location 8090H. If neither flag is set, the subroutine returns from location 80A0H.

4.7 TIME-DELAY LOOPS

Microprocessors perform different operations in sequence and one operation at a time. To complete an operation, some time is required. When some time delay is required between two operations, a time-delay loop is used to provide it.

Time delay can be generated using a register or a register pair. Initially, a register is loaded with an operand or number and then the number is decremented until it reaches zero. So a conditional jump instruction is used in a delay loop to come out from the loop. The time delay depends on the number which is loaded in the register. Figure 4.9 shows the flowchart of time delay-loop using one register.

4.7.1 Calculation of Time Delay Using One Register

The typical instructions of a time-delay loop are given below:

PROGRAM 4.1

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments	T state
8000	06, 80		MVI	B, 80	Initialise Register B	7
8002	05	LOOP	DCR	B	Decrement Register B	5
8003	C2, 03, 80		JNZ	LOOP	Jump not zero to LOOP	10

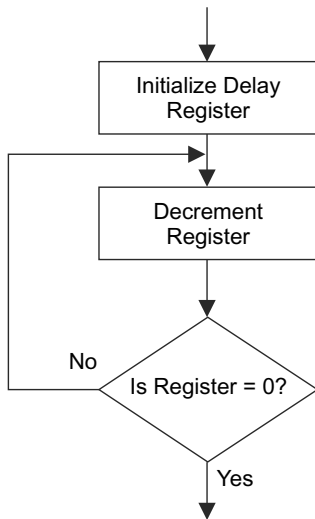


Fig. 4.9 Flowchart for time delay using a register

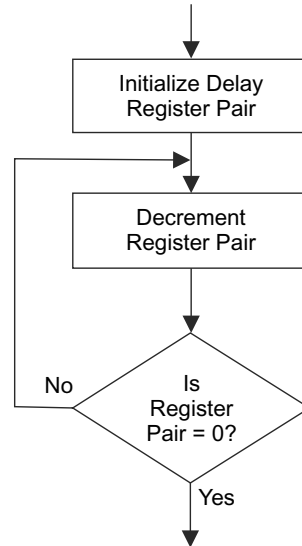


Fig. 4.10 Flowchart for time delay using register pair

It is clear from the above instructions that MVI B, 80 requires seven clock cycles, DCR B requires 5 clock pulse and JNZ also requires 10 clock pulses during execution. When these instructions are executed, MVI B,80 instruction is executed once and the next two instructions are executed 128 times.

The number of T states for execution of LOOP is

$$\begin{aligned}
 &= (T \text{ states for DCR B} + T \text{ states for JNZ}) \times \text{Number of times LOOP is executed} \\
 &= (7 + 5) \times 128 \text{ } T \text{ states}
 \end{aligned}$$

The delay time to execute the LOOP instruction is $T_L = T \times$ number of T states for execution of LOOP, where T is the system clock period. When the microprocessor operates in 5 MHz clock frequency,

$$T_L = \frac{1}{5} \times 10^{-6} (5 + 10) \times 128 \text{ s} = 384 \text{ } \mu\text{s}.$$

The total time delay T_D is calculated from the summation of time to execute instruction of outside LOOP, T_{OL} and time to execute LOOP instruction, T_L .

$$T_D = T_{OL} + T_L = \frac{1}{5} \times 10^{-6} \times 7 + \frac{1}{5} \times 10^{-6} \times (5 + 10) \times 128 \text{ } \mu\text{s} = 385.4 \text{ } \mu\text{s}$$

Using only one register in a delay loop, a limited time delay is generated. If very high time delay is required, a register pair will be used in place of a register. Figure 4.10 shows the flowchart for time-delay generation using a register pair. For example, a 16-bit operand is loaded in the DE register pair. Then the DE register pair is decremented by one using DCX D instruction. The DCX instruction does not set the zero flag. Therefore, additional testing will be done using some extra instructions as the JNZ instruction is executed only when the zero flag is set.

The typical instructions of a time-delay loop using a register pair are given below:

PROGRAM 4.2

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments	T state
8100	11, 00, 80		LXI	D, 8000	Initialise the DE register pair	10
8103	1B	LOOP	DCX	D	Decrement the DE register pari	5
8104	7B		MOV	A,E	Copy content of Register E in the accumulator	5
8105	B2		ORA	D	OR D with accumulator	4
8106	C2, 06, 81		JNZ	LOOP	Jump not zero to LOOP	10

4.7.2 Calculation of Time-Delay Using Register Pair

In the above instructions, LXI D, 8000 is executed once and the other instructions (DCX D, MOV A, E, ORA D and JNZ) are executed for 8000H (32768_D) times.

The number of T states for execution of LOOP is
 = (T states for DCX D + T states for MOV A, E + T states for ORA D + T states for JNZ) × Number of times LOOP is executed
 = (5 + 5 + 4 + 10) × 32768 T states

If the microprocessor clock frequency is 5 MHz, time delay in LOOP is equal to T_L . $T_L = T \times \text{number of T states for execution of LOOP} = \frac{1}{5} \times 10^{-6} \times (5 + 5 + 4 + 10) \times 32768 \mu s = 157.268 \mu s$ (approx).

$$T_D = T_{OL} + T_L = \frac{1}{5} \times 10^{-6} \times 10 + \frac{1}{5} \times 10^{-6} \times (5 + 5 + 4 + 10) \times 32768 \mu s = 157.288 \mu s$$

(approx)

4.7.3 Time Delay Using Two LOOPS

The time delay can also be generated by using two loops as depicted in Fig. 4.11. The C register is used in inner loop and the B register is used in external loop. Here, both B and C registers are loaded with numbers. Then Register C is decremented until it becomes zero. When the content of Register C is zero, decrement Register B. If the content of Register B is not zero, load the Register C with initial value and repeat the process.

The example of time delay using two loops is given below:

PROGRAM 4.3

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments	T state
8200	06, 80		MVI	B, 80	Initialise the Register B	7
8202	0E,FF	LOOP-II	MVI	C, FF	Initialise the Register B	7
8203	OD	LOOP-I	DCR	C	Decrement Register B	5
8204	C2, 03, 82		JNZ	LOOP-I	Jump not zero to LOOP	10
8207	05		DCR	B	Decrement Register B	5
8208	C2, 02, 82		JNZ	LOOP-II	Jump not zero to LOOP	10

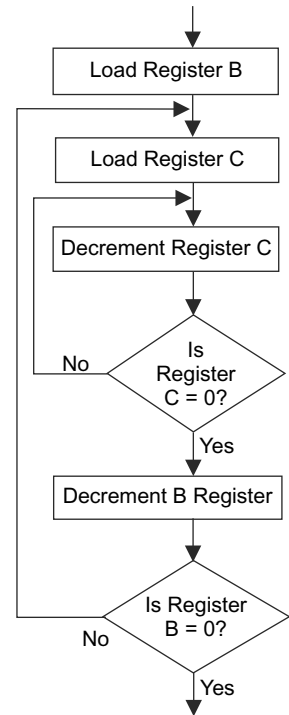


Fig. 4.11 Flowchart for time delay using two loops

Time delay will be calculated based on time delays for LOOP-I and LOOP-II.

Time delay for LOOP-I is

$$T_{\text{LOOP-I}} = T \times (T \text{ states for DCR B} + T \text{ states for JNZ}) \times \text{Number of times LOOP is executed}$$

$$= \frac{1}{5} \times 10^{-6} \times (5 + 10) \times 256 \mu\text{s}$$

Time delay for LOOP-II is

$$T_{\text{LOOP-II}} = \{T_{\text{LOOP-I}} + T (T \text{ states for MVI C, FF} + T \text{ states for DCR B} + T \text{ states for JNZ})\} \times \text{Number of times LOOP-II is executed}$$

$$= \left\{ \frac{1}{5} \times 10^{-6} \times (5 + 10) \times 256 + \frac{1}{5} \times 10^{-6} \times (7 + 5 + 10) \right\} \times 128 \mu\text{s}$$

Total delay time (T_D) = Time to execute instruction of outside LOOP (T_{OL}) + Time to execute LOOP-II

$$(T_L T_{\text{LOOP-II}}) = \frac{1}{5} \times 10^{-6} \times (7) + \left\{ \frac{1}{5} \times 10^{-6} \times (5 + 10) \times 256 + \frac{1}{5} \times 10^{-6} \times (7 + 5 + 10) \right\} \times 128 \mu\text{s} = 98.868 \text{ ms (approx.)}$$

4.8 MODULAR PROGRAMMING

Generally, industry-programming projects consist of thousands of lines of instructions or operation code. Such huge monolithic programs would be unmanageable and incomprehensible. Therefore, it is difficult to design, write, debug, and test such a project. Hence, the complete project is divided into sub-problems or small modules. Each independent modules are separately named and are individually invokeable program elements. The sizes of modules are reduced to a humanly comprehensible and manageable level. This approach is known as *modular programming*. The divide-and-conquer approach is used in programming.

Modules are designed, written, tested and debugged by individuals or small teams to allow for multiple programmers to work in parallel. Modules are integrated to become a software system that satisfies the problem requirements. To integrate successfully, the original decision must be good and interfaces between modules must be correct.

Each module will be different, depending on the specific problem being solved. In very simple problems only one module exists, but complex problems have many hundreds of modules. Modules are written in such a way that everybody understands the program very easily. Generally, a top-down design is used in modular programming. In this programming, high-level instructions are broken down into smaller sets of instructions and again into smaller sets until we get the smallest module. The characteristics of modules are given below:

1. Each module is independent of other modules.
2. Each module has one input and one output.
3. A module is small in size.
4. Programming a single function per module is a goal.

4.8.1 Advantages of Modular Programming

The advantages of modular programming are the following:

1. It is easy to write, test and debug a module.
2. Generally modules of common nature are prepared, which can be used at many places.
3. The programmer can divide tasks and use the previously written programs.
4. If a change is to be made, it is made in the particular module; the entire program is not affected.
5. Pieces can be debugged independently.
6. Work can be divided between multiple programmers.
7. Code can be reused.

8. Problems can be reduced to smaller, simpler, humanly comprehensible levels.
9. Modules can be assigned to different teams/programmers. This enables parallel work, reduces program-development time, and facilitates programming, debugging, testing and maintenance.
10. Individual modules are portable; so they can be modified to run on other platforms.
11. Modules can be re-used within a program and across programs.

4.8.2 Disadvantages of Modular Programming

The disadvantages of modular programming are the following:

1. The combining of modules together is a difficult task.
2. It needs careful documentation as it may affect the other parts of the program.
3. While testing modules it may be found that the module under test may require data from other modules or its results may be used by other modules. To solve such problems, special programs called *drivers* are to be developed to produce the desired data for the testing of modules. The development of drivers requires extra effort and time.
4. Modular programming requires extra time and memory.
5. The modular programming was originally developed for writing long programs but this technique can also be used for shorter programs written for microcomputers. Modules are divided on functional lines and hence, they can form a library of programs. Modules of 20 to 50 lines should be developed. They are very useful. There is unnecessary wastage of time in preparing shorter modules. Longer modules are not converted to general nature. The modules should be developed for common tasks. They should be of general form.

4.9 MACRO

Although 246 instructions are available in the 8085 microprocessor, some new instructions can be developed using a sequence of known instructions. These new instructions are always assigned a name and known as MACRO, used in assembly-language programming. Examples are DELAY, LARGE, SMALL, MUL, and DIV, etc. Most of the assemblers have macro facility. The general form of a macro is

```
Name      MACRO arg
           Statement-1
           Statement-2
           ENDM
```

where, *Name* is the name of the macro, *arg* represents the arguments of the macro, *statements* are instructions, and ENDM is used to end the macro.

The example of a DELAY macro is

```
DELAY      MACRO 8000H
           LXI B, 8000H
LOOP       DCX B
           MOV A, C
           ORA D
           JNZ LOOP
           ENDM
```

In the above example DELAY is the name of the macro to generate a time delay. In the assembly-language program if we write DELAY 8000H, the assembler replaces the macro by the above instructions.

When a sequence of instructions is written and the macro name is assigned to it, the macro name can be used repeatedly the main program and this makes the program easy to understand.

Another example of macro is ADDER as given below:

```

ADDER    MACRO ADDRESS (8000H)
          LXI H, 8000H
          MOV A, M
          INX H
          ADD M
          ENDM
    
```

In this example, ADDER is the name of the macro. Here, ADDRESS is a parameter and ENDM is used at the end of the macro. In an assembly-language program, if ADDRESS is 8000H, the macro replaces the above instructions.

Macros and subroutines are similar. A subroutine requires CALL and RETURN instructions whereas macros do not. Macros execute faster than subroutines. Macros are used for short sequences of instructions, whereas subroutines for longer ones, generally more than 10 instructions and more. Like subroutines, a macro can be written in nested form. One macro can be called by another macro. The differences between macro and subroutines are given below.

<i>Macros</i>	<i>Subroutine</i>
It is used to perform specified operations.	Subroutines are also used in specified operations like macros.
In macros, only name of the macro is used and at the end of each macro, ENDM is used.	In a subroutine, CALL and RET are used.
Macros are faster than subroutines.	Subroutines are slower than macros.
Macros are used for very few instructions, approximately. 10 instructions.	More than ten instructions are used in a subroutine.

4.10 INSTRUCTION FORMAT

Each statement in an assembly-language program consists of the following fields: Memory address, Machine Codes, Labels, Mnemonics, Operands and Comments. The commonly used format of an instruction in assembly language is given below:

Memory Address	Machine Codes	Labels	Mnemonics	Operands	Comments
----------------	---------------	--------	-----------	----------	----------

- ✓ **Memory Address** This is the address of the memory location in which a program or a series of instructions are stored.
- ✓ **Machine Codes** Every instruction has a unique one-byte code called operation code. Instructions are operated using data. Data may be of one byte or two bytes. Machines codes are the hexadecimal representation of operation codes and codes.
- ✓ **Labels** It is assigned in the instruction in which it appears. The presence of a label in an instruction is optional. When a label is present, it provides a symbolic name that can be used in branch instructions to branch to the instruction. If there is no label then the colon must not be entered. A label may be of any length, from 1 to 35 characters. A label appears in a program to identify the name of a memory location for storing data and other purposes. This is used for conditional/unconditional jumping.
- ✓ **Mnemonics** Each instruction has a specific mnemonic. The mnemonic states the operation which will be executed.

✓ **Operands** Operands depend on the type of instruction. In a one-byte instruction, there is no operand. Only one operand exists in two-byte instructions and a three-byte instruction has two operands which are separated by a comma.

✓ **Comments** In this field, general comments about the instructions are always incorporated to understand the program easily. It is optional. The comment field contains any combination of characters. A comment may appear on a line and the first character of the line must be a semicolon.

4.11 ASSEMBLY-LANGUAGE PROGRAMS

4.11.1 Simple Examples of Assembly-Language Programs

Example 4.1

Transfer data from accumulator to Register B respectively.

<i>Mnemonics</i>	<i>Opcode</i>	<i>Comments</i>
MOV B,A	47	Copy the content of accumulator to Register B

Example 4.2

Load FFH in Register C.

<i>Mnemonics</i>	<i>Opcode</i>	<i>Comments</i>
MVI C, FFH	0E, FF	Load FFH in Register C immediately

Example 4.3

Load 22H and 67H in registers B and C respectively.

<i>Mnemonics</i>	<i>Opcode</i>	<i>Comments</i>
LXI B 22 67	01, 67, 22	Load Register C with 67H and Register B with 22H

Example 4.4

Load HL register pair by the data 8150H.

<i>Mnemonics</i>	<i>Opcode</i>	<i>Comments</i>
LXI H,8150H	21, 50, 81	Load HL register pair with 8150H

Example 4.5

Load the content of memory location 8100H in the accumulator.

<i>Mnemonics</i>	<i>Opcode</i>	<i>Comments</i>
LXI H,8100H	21, 00, 81	Load memory location address 8100H in HL register pair
MOV A, M	7E	Copy content of memory location in the accumulator

Example 4.6

Store the content of accumulator in 8001H location.

<i>Mnemonics</i>	<i>Opcode</i>	<i>Comments</i>
STA 8001H	32, 01, 80	Content of accumulator is stored in 8001H location

Example 4.7

Transfer data stored in memory location 9950H to the accumulator.

<i>Mnemonics</i>	<i>Opcode</i>	<i>Comments</i>
LDA 9950H	3A 50 99	Move data to accumulator from memory location 9950H

Example 4.8

Load 45H data in the memory location 8500H. Increment the content of memory location.

<i>Mnemonics</i>	<i>Opcode</i>	<i>Comments</i>
LXI H,8500H	21, 00, 85	Load memory location address 8500H in HL register pair
MVI M,45H	36, 45	45H is stored in 8500H location
INR M	34	Content of memory location incremented by one

Example 4.9

Transfer the contents of 8101H and 8100H to registers H and L respectively. Then store the HL content to memory location 9301H and 9300H respectively.

<i>Mnemonics</i>	<i>Opcode</i>	<i>Comments</i>
LHLD 8100 H	2A, 00, 81	Load H and L from memory location 8101H and 8100H
SHLD 9300H	22, 00, 93	Store data from H and L memory location 9301H and 9300H respectively.

4.11.2 Addition of Two 8-bit Numbers with 8-bit Sum

Consider the first number 26H is stored in the memory location 8000H and the second number 62H is stored in the memory location 8001H. The result after addition of two numbers is to be stored in the memory location 8003H. Assume the program starts from the memory location 8500H. The program flow chart is shown in Fig. 4.12.

Algorithm

1. Initialize the memory location of the first number in the HL register pair.
2. Move first number/data into the accumulator.
3. Increment the content of the HL register pair to initialize the memory location of second data.
4. Add the second data with the accumulator.
5. Store the result in the memory location 8003H.

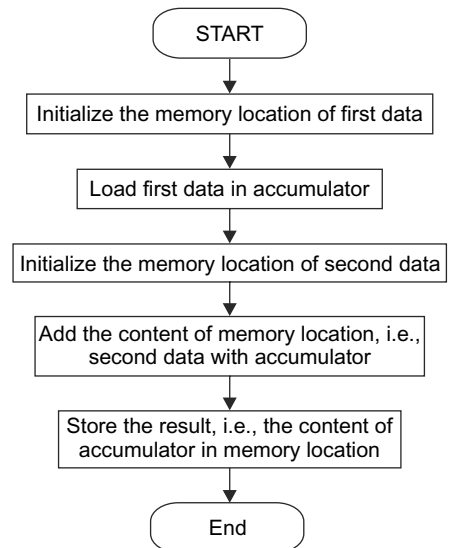


Fig. 4.12 Flow chart for addition of two 8-bit numbers with 8-bit sum

PROGRAM 4.4

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8500	21, 00, 80		LXI	H, 8000 H	Address of first number in HL register pair
8503	7E		MOV	A,M	Transfer first number in accumulator
8504	23		INX	H	Increment content of HL register pair
8505	66		ADD	M	Add first number and second number
8506	32, 03, 80		STA	8003H	Store sum in 8003 H
8509	76		HLT		Halt

Example 4.10

<i>Data</i>		<i>Result</i>	
Memory location	Data	Memory location	Data
8000	26H	8003	88H
8001	62H		

4.11.3 Addition of two 8-bit Numbers with 16-bit Sum

The first number F2H is stored in the memory location 8501H and the second number 2FH is stored in the memory location 8502H. The result after addition will be stored in the memory locations 8503H and 8504H. Consider the program is written from the memory location 8000H. The program flow chart is depicted in Fig. 4.13.

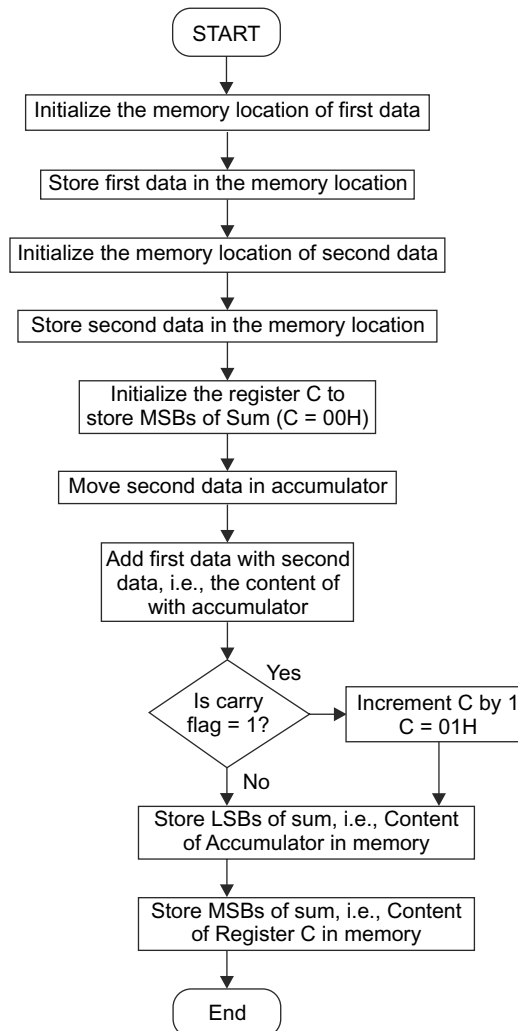


Fig. 4.13 Flow chart for addition of two 8-bit numbers with 16-bit sum

Algorithm

1. Initialize the memory location of first data in the HL register pair.
2. Store first data in the memory location.
3. Increment the content of the HL register pair for entering next data in the next memory location.
4. Store second data in the memory location.
5. Move the second number in accumulator.
6. Decrease the content of the HL register pair.
7. Add the content of memory (first data) with the accumulator.
8. Store the results in memory locations 8503H and 8504H.

PROGRAM 4.5

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	21, 01, 85		LXI	H, 8501 H	Address of 1st number in HL register pair
8003	36, F2		MVI	M, F2H	Store 1st number in memory location represented by HL register pair
8005	23		INX	H	Increment content of HL register pair
8006	36, 2F		MVI	M, 2FH	Store 2nd number in memory location represented by HL register pair
8008	7E		MOV	A, M	2nd number in accumulator
8009	0E, 00		MVI	C, 00H	Initialize Register C with 00H to store MSBs of sum
800B	2B		DCX	H	Address of 1st number 8501 in HL pair
800C	66		ADD	M	Addition of 1st number and 2nd number
800D	D2, 11, 85		JNC	LEVEL_1	If carry does not generate, jump to LEVEL 1
8010	0C		INR	C	When carry is generated, increment Register C
8011	32, 03, 85	LEVEL 1	STA	8503H	Store LSBs of sum in memory location 8503H
8014	79		MOV	A,C	Move MSBs of sum in accumulator
8015	32, 04, 85		STA	8504 H	Store MSBs of sum in memory location 8504H
8018	76		HLT		Halt

Example 4.11

<i>Data</i>		<i>Result</i>	
Memory location	Data	Memory location	Data
8501	F2H	8503	82H LSBs of sum
8502	2FH	8504	01H MSBs of sum

4.11.4 Addition of N 8-bit Numbers

Write a program for addition of a series of 8-bit numbers with carry. The 'N' number of hexadecimal numbers are stored from F101H onwards. F100H has the number of hexadecimal bytes to be added. The result is stored at F200H and F201H memory locations. Assume the program is written in the memory location F000H. Figure 4.14 shows the flowchart for addition of 'N' 8-bit numbers.

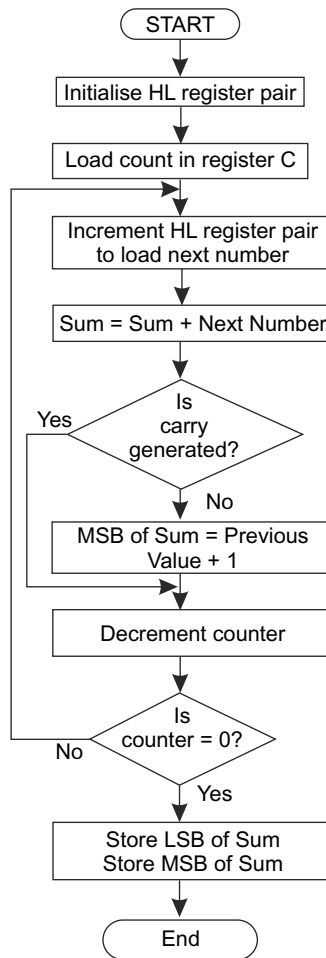


Fig. 4.14 Flowchart for addition of N 8-bit numbers

Algorithm

1. Load the number of bytes to be added in the F100H memory location.
2. Initialize accumulator, as LSBs of the result will be stored in accumulator.
3. Register B is also initialized to store MSBs of sum.
4. Let the memory point the number of the bytes to be added and stored in Register C.
5. Move next memory location for data and data with accumulator.
6. If carry is generated, Register B will be incremented by one.
7. Decrement the counter having number of bytes.
8. Check if zero—no repetition from point 5.
9. Store the result at F200H and F201H locations.

PROGRAM 4.6

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
F000	21, 00, F1		LXI	H, F100H	Address of number of bytes in HL register pair
F003	4E		MOV	C, M	Transfer number of bytes from memory location to Register, C
F004	AF		XRA	A	Clear accumulator register
F005	06, 00		MVI	B, 00	Initialize Register B with 00H to store MSBs of sum.
F007	23	LOOP	INX	H	Address of 1st number in HL pair.
F008	66		ADD	M	Add memory to accumulator
F009	D2, 0D, F0		JNC	LEVEL_1	If carry does not generate, jump to LEVEL 1
F00C	04		INR	B	If carry is generated, increment Register, B
F00D	0D	LEVEL 1	DCR	C	Decrement count by one
F00E	C2, 07, F0		JNZ	LOOP	Test to check whether addition of all numbers are done
F011	32, 00, F2		STA	F200	Store LSBs of sum in memory location F200H
F014	78		MOV	A, B	Copy content of B in accumulator
F015	32, 01, F2		STA	F201	Store MSBs of sum in memory location F201H.
F018	76		HLT		Stop

Example 4.12

Consider five (N = 05H) data are stored from the location F101 onwards as given below. The result is stored in locations F200H and F201.

<i>Data</i>		<i>Result</i>	
Memory location	Data	Memory location	Data
F100	05H	F200	0FH LSBs of sum
F101	01H	F201	00H MSBs of sum
F102	02H		
F103	03H		
F104	04H		
F105	05H		

4.11.5 Addition of Two 16-Bit Numbers with 16-bit Sum

The first 16-bit number is stored in 8501H and 8502H memory locations. The second 16-bit number is stored in 8503H and 8504H memory locations. After addition, the result will be stored from 8505H and 8506H memory locations. Assume the program starts from the memory location 8000H. The program flow chart for addition of two 16-bit numbers with 16-bit sum is depicted in Fig. 4.15.

Algorithm

1. Store first 16-bit number in HL pair.
2. Exchange the contents of DE pair and HL pair to store the first number in DE register pair.

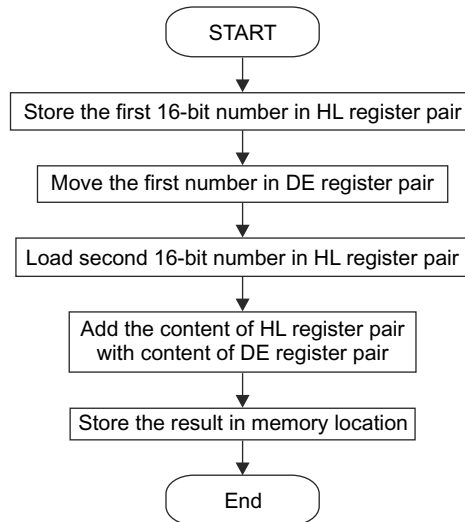


Fig. 4.15 Flow chart for addition of two 16-bit numbers with 16-bit sum

3. Store second 16-bit number in HL register pair.
4. Addition of first and second numbers.
5. Store result in 8505H and 8506H locations.

PROGRAM 4.7

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	2A, 01, 85		LHLD	8501 H	Load the content of 8501H location in Register L and Register H is loaded with the content of 8502H location
8003	EB		XCHG`		The contents of HL register pair are exchanged with DE register pair so that first data is stored in DE register pair
8004	2A, 03, 85		LHLD	8503H	Load second 16-bit number (data-2) in HL pair
8007	19		DAD	D	The contents of DE pair are added with the contents of HL pair and result is stored in HL pair
8008	22, 05, 85		SHLD	8505H	Store LSBs of sum in 8505H and MSBs of sum 8506 H
800B	76		HLT		Halt

Example 4.13

<i>Data</i>		<i>Result</i>	
Memory location	Data	Memory location	Data
8501	05H LSBs of data-1	8505	0FH LSBs of sum
8502	01H MSBs of data-1	8506	00H MSBs of sum
8503	02H LSBs of data-2		
8504	03H MSBs of data-2		

4.11.6 Decimal Addition of Two 8-Bit Numbers with 16-bit Sum

Two decimal numbers 52H and 85H are stored in F050H and F051H locations respectively. The result is to be stored in F052H and F053H locations. Assume program is written from memory location F100H. The program flow chart for decimal addition of two 8-bit numbers with 16-bit sum is shown in Fig. 4.16.

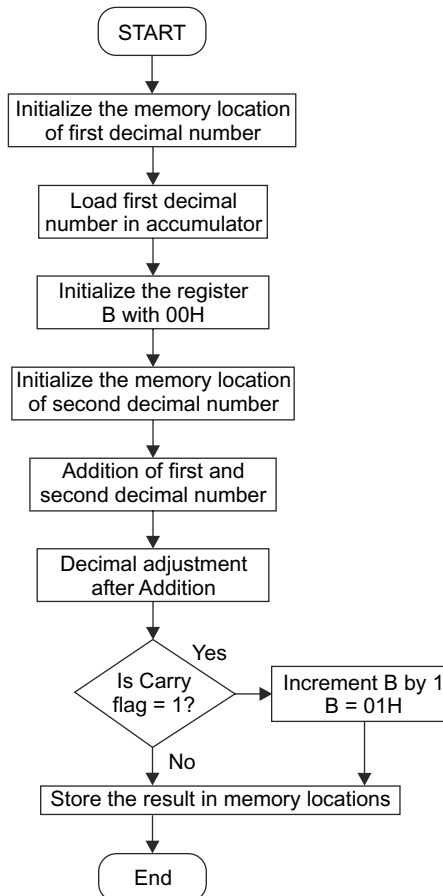


Fig. 4.16 Flow chart for decimal addition of two 8-bit numbers with 16-bit sum

Algorithm

1. Load address of the first number in HL register pair.
2. Load the first number in accumulator and store 00H in Register B.
3. Increment the content of HL register pair to initialize the memory location of second data.
4. Addition of the content of second memory location with first data.
5. Decimal adjustment of result.
6. If carry is generated, Register B is incremented by one.
7. Store the result in F052H and F053H locations.

PROGRAM 4.8

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
F100	21, 50, F0		LXI	H,F050	Address of first number in HL register pair
F103	7E		MOV	A,M	Move first number in accumulator
F104	06, 00		MVI	B,00H	Load 00H in Register B
F105	23		INX	H	Increment HL register pair to locate second number.
F106	86		ADD	M	Addition of 1st and 2nd number
F107	27		DAA		Decimal adjust
F108	D2, 00, F1		JNC	LEVEL_1	If carry does not generated, jump to LEVEL_1
F10B	04		INR	B	Increment Register B
F10C	32, 52, F0	LEVEL_1	STA	F052H	Store LSDs of result in F052H location
F10F	78		MOV	A,B	Move MSDs from B to A
F110	32, 53, F0		STA	F053	Store MSDs of result in F053H location
F103	76		HLT		Halt

Example 4.14

<i>Data</i>		<i>Result</i>	
Memory location	Data	Memory location	Data
F050	52H	F052	89H LSDs of sum
F051	85H	F053	01H MSDs of sum

4.11.7 Addition of N 8-bit Decimal Numbers

Consider a number of data in 8000H and data are stored in 8001H onwards. After addition, the result will be stored in 8100H and 8101H. Assume that the program starts from memory location 8500H. The program flow chart for addition of N 8-bit decimal numbers is illustrated in Fig. 4.17.

Algorithm

1. A number of data is loaded in Register C.
2. Load 00H in Register B.
3. Load address of first number in HL register pair.
4. Load the first number in accumulator.
5. Increment the content of HL register pair to initialize the memory location of next data.
6. Addition of the content of next memory location with first data.
7. Decimal adjustment of result.
8. If carry is generated, Register B is incremented by one.
9. Decrement Register C.
10. Test to check whether additions of all numbers are done. If C is not equal to zero, repeat steps 5 to 10.
11. Store the result in 8100H and 8101H locations.

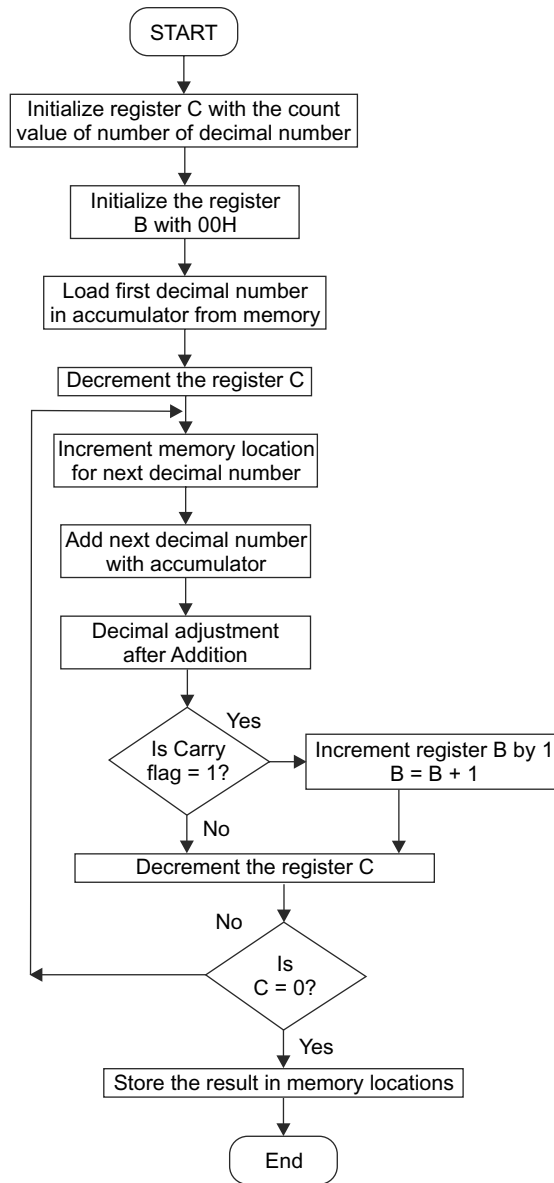


Fig. 4.17 Flow chart for decimal addition of N 8-bit numbers with 16-bit sum

PROGRAM 4.9

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8500	21, 00, 80		LXI	H,8000H	Address of number of data in HL register pair
8503	4E		MOV	C,M	Move number of data in accumulator
8504	06, 00		MVI	B,00H	Load 00H in Register B

(Contd.)

(Contd.)

8506	23		INX	H	Increase HL register pair to locate first number
8507	7E		MOV	A,M	Move first number into accumulator
8508	0D		DCR	C	Decrease Register C
8509	23	LOOP	INX	H	Increase HL register pair to locate next number
850A	86		ADD	M	Addition of content of accumulator and next number
850B	27		DAA		Decimal adjust
850C	D2, 10, 85		JNC	LEVEL_1	If carry does not generated, jump to LEVEL_1
850F	04		INR	B	Increase Register B
8510	0D	LEVEL_1	DCR	C	Decrease Register C
8511	C2, 09, 85		JNZ	LOOP	Jump not zero to LOOP
8514	32, 00, 81		STA	8100H	Store LSDs of result in 8100H location
8517	78		MOV	A,B	Move MSDs from B to A
8518	32, 01, 81		STA	8101H	Store MSDs of result in 8101H location
851B	76		HLT		Stop

Example 4.15

Data		Result	
Memory location	Data	Memory location	Data
8000	05H	8100	0FH LSBs of sum
8001	01H	8101	00H MSBs of sum
8002	02H		
8003	03H		
8004	04H		
8005	05H		

4.11.8 Subtraction of Two 8-bit Numbers

Data are stored in memory locations 8000H and 8001H. The result after subtraction will be stored in 8002H. Consider the program is written from memory location 8500H. The program flow chart for subtraction of two 8-bit numbers is shown in Fig. 4.18.

Algorithm

1. Load address of first number in HL register pair.
2. Move first data into accumulator.
3. Increase the content of HL register pair.
4. Subtract the second data from accumulator.
5. Store the result in memory location 8002H.

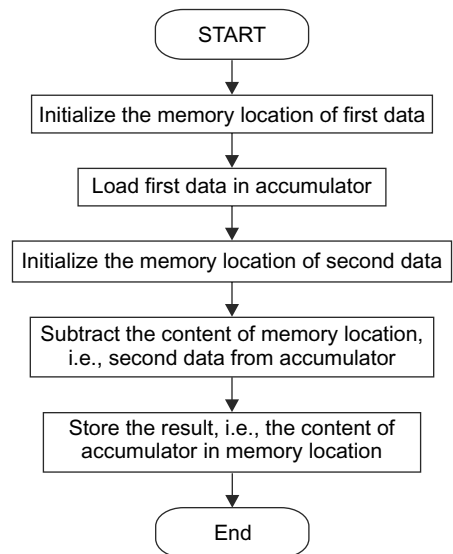


Fig. 4.18 Flow chart for subtraction of two 8-bit numbers

PROGRAM 4.10

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8500	21, 00, 80		LXI	H,8000	Address of first number in HL register pair
8503	7E		MOV	A,M	Transfer first number in accumulator
8504	23		INX	H	Increment content of HL pair
8505	96		SUB	M	Subtract second number from first number
8506	23		INX	H	Increment content of HL pair
8507	77		MOV	M,A	Store result
8508	76		HLT		Halt

Example 4.16

Data		Result	
Memory location	Data	Memory location	Data
8000	89H	8002	47H
8001	42H		

4.11.9 Subtraction of Two 16-bit Numbers

Consider first number in DE register pair and the second number is in BC register pair. After subtraction, result will be stored in 9000H onwards. Assume MSBs of first number is greater than second number. Assume the program starts from 8500H memory location. The program flow chart for subtraction of two 16-bit numbers is depicted in Fig. 4.19.

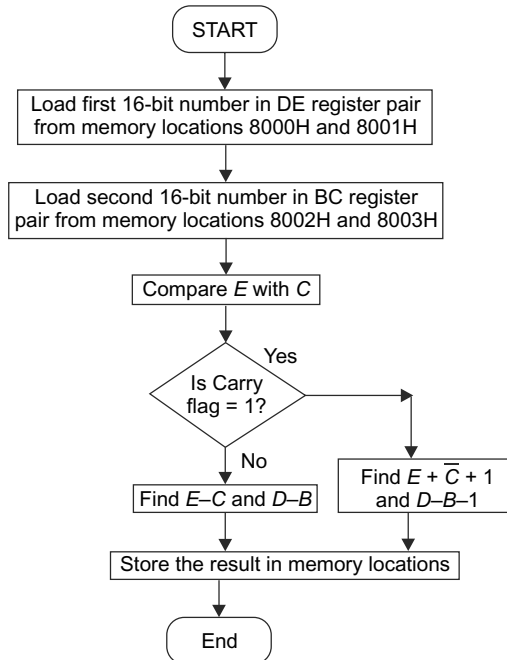


Fig. 4.19 Flow chart for subtraction of two 16-bit numbers

Algorithm

1. Load first number in DE register pair from memory locations 8000H and 8001H.
2. Load second number in BC register pair from memory locations 8002H and 8003H.
3. Compare LSBs of two numbers, E and C . If $E \geq C$, find $D-B$ and $E-C$. When $E < C$, find $D-B-1$ and $E + \bar{C} + 1$.
4. Then store results in memory locations 9000H and 9001H.

PROGRAM 4.11

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8500	21, 00, 80		LXI	H,8000	Load address of first number in HL register pair
8503	5E		MOV	E,M	Load first two byte numbers in DE register pair LSBs in Register E and MSBs in Register D
8504	23		INX	H	Increment content of HL pair for address of second
8505	56		MOV	D, M	number
8506	23		INX	H	
8507	4E		MOV	C,M	Load second two byte number in BC register pair,
8508	23		INX	H	LSBs in Register C and MSBs in Register B
8509	46		MOV	B,M	
850A	7B		MOV	A,E	Transfer LSBs of first number in accumulator.
850B	B9		CMP	C	Compare between LSBs of second number and LSBs of first number
850C	DA, 00, 85		JC	LEVEL 1	If carry is generated, jump to LEVEL_1
850F	7B		MOV	A,E	Transfer LSBs of first number in accumulator
8510	91		SUB	C	Find $E - C$
8511	32, 00, 90		STA	9000	Store LSBs, the result of $(E - C)$ in 9000H location
8514	7A		MOV	A,D	MSBs of second number in accumulator
8515	98		SUB	B	Find $D-B$
8516	32, 01, 90		STA	9001	Store MSBs, $D-B$ in 9001 location
8519	C3, 2B, 85		JMP	852B	Jump to 852B
851C	79	LEVEL 1	MOV	A,C	Transfer LSBs of first number in accumulator.
851D	2F		CMA		Get the complement of $C = \bar{C}$
851E	83		ADD	E	
851F	C6, 01		ADI	01H	Determine $E + \bar{C} + 1$ pair
851I	32, 00, 90		STA	9000	Store LSBs the result of $E + \bar{C} + 1$ in 9000H location
8524	7A		MOV	A,D	Transfer MSBs of first number in accumulator
8525	90		SUB	B	Subtract B from accumulator
8526	D6, 01		SUI	01H	Subtract 01H from accumulator. Find $D-B-1$
8528	32, 01, 90		STA	9001	Store MSBs, the result of DB1 in 9001H location
852B	76		HLT		

Example 4.17

Data		Result	
Memory location	Data	Memory location	Data
8001	F0H LSBs of data-1	8005	FFH LSBs of sum
8002	F0H MSBs of data-1	8006	0FH LSBs of sum
8003	0FH LSBs of data-2	8007	01H MSBs of sum
8004	1FH MSBs of data-2		

4.11.10 8-Bit Decimal Subtraction

Two 8-bit decimal numbers are stored in memory locations 8000H and 8001H. The result after subtraction will be stored in 8002. Assume program is written from memory location 8500H. The program flow chart for 8-bit decimal subtraction is given in Fig. 4.20.

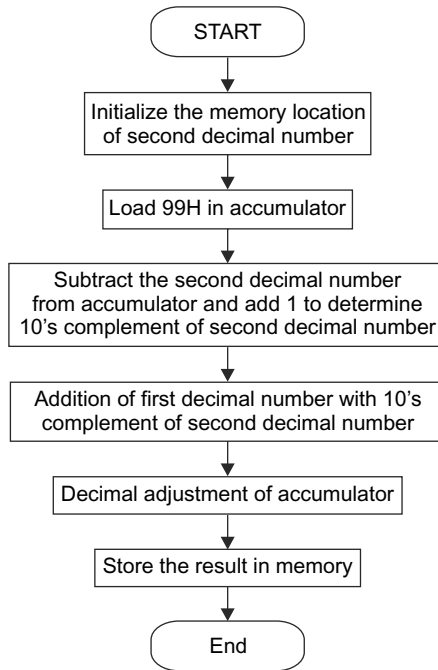


Fig. 4.20 Flow chart for decimal subtraction of two 8-bit numbers

Algorithm

1. Find 9's complement of second number = 99H –second Number.
2. Determine 10's complement of second number = 9's complement of number +1.
3. Add second number with 10's complement of second number.
4. Addition of 1st decimal number with 10's complement of 2nd decimal number.
5. Decimal adjustment of accumulator.
6. Store the result in memory location 8002H.

PROGRAM 4.12

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8500	21, 01, 80		LXI	H,8001	Load the address of 2nd number in HL register pair
8503	3E, 99		MVI	A,99H	Get 9's complement of 2nd number = 99H – 2nd Number
8505	96		SUB	M	
8506	C6, 01		ADI	01H	10's complement of 2nd number = 9's complement of number +1
8508	2B		DCX	H	Decrement HL register pair
8509	8E		ADD	M	Addition of 1st number and 10's complement of 2nd number
850A	27		DAA		Decimal adjustment
850B	32, 02, 80		STA	8002	Store result in 8002H location
850E	76		HLT		

In the above program, the instruction INR A may be used in place of ADI 01H.

Example 4.18

Data		Result	
Memory location	Data	Memory location	Data
8000	89H	8002	47H
8001	42H		

4.11.11 One's Complement of an 8-bit Number

The number is stored in the memory location 8050H and one's complement of the number will be stored in the location 8051H. Assume the program memory starts from 8000H. The program flow chart for one's complement of an 8-bit number is shown in Fig. 4.21.

Algorithm

1. Load memory location of data 8050H in HL register pair.
2. Move data into accumulator.
3. Complement accumulator.
4. Store the result in the memory location 8051H.

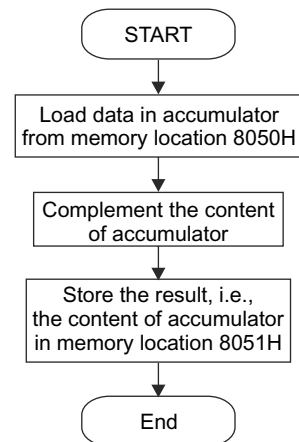


Fig. 4.21 Flow chart for 1's complement of an 8-bit number

PROGRAM 4.13

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	21, 50, 80		LXI	H,8050H	Load address of number in HL register pair
8003	7E		MOV	A,M	Move the number into accumulator
8004	3F		CMA		Complement accumulator
8005	32, 51, 80		STA	8051H	Store the result in 8051H
8008	76		HLT		Stop

In the above program, the first two instructions LXI H,8050H and MOV A,M can be replaced by directly loading the content of location 8050H in accumulator. For this LDA 8050H can be used.

Example 4.19

Data		Result	
Memory location	Data	Memory location	Data
8050	F0H	8051	0FH

4.11.12 Two's Complement of an 8-bit Number

The number is stored in the memory location 8500H. The two's complement will be stored in 8501H. The program is written from the memory location 8510H. The program flow chart for 2's complement of an 8-bit number is depicted in Fig. 4.22.

Algorithm

1. Transfer the content of the memory location 8500H to accumulator.
2. Complement the content of the accumulator.
3. Add 01H with accumulator to get two's complement of a number.
4. Store the result in the memory location 8501H.

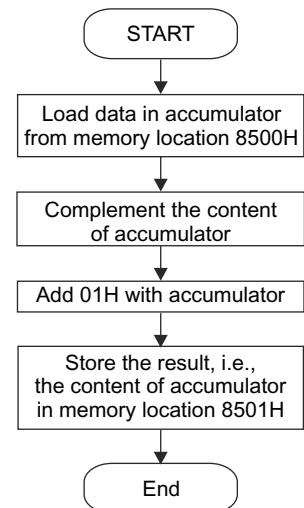


Fig. 4.22 Flow chart for 2's complement of an 8-bit number

PROGRAM 4.14

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8510	3A, 00, 85		LDA	8500H	Load the content of memory location 8500H in accumulator
8513	2F		CMA		Complement accumulator
8514	C6, 01		ADI	01H	Add 01H with accumulator to find two's complement of the number
8516	32, 01, 85		STA	8501H	Store result in 8501H location
8519	76		HLT		Stop

Example 4.20

Data		Result	
Memory location	Data	Memory location	Data
8500	F0H	8501	10H

4.11.13 Shift an 8-bit Number Left by One Bit

The number is stored in 8000H and its one bit left shift will be stored in 8001H. Assume the program is written from memory location 8010H. The program flow chart for shifting an 8-bit number left by one bit is depicted in Fig. 4.23.

Algorithm

1. Load memory location of data 8000H in HL register pair.
2. Move data from memory to accumulator.
3. Content of accumulator rotate left by one bit.
4. Store the result in the memory location 8001H.

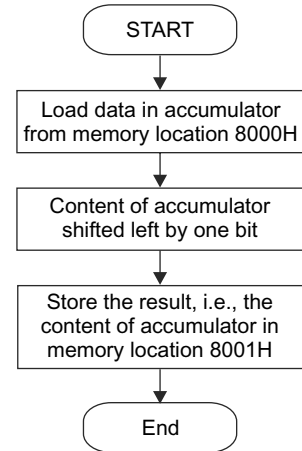


Fig. 4.23 Flow chart for shifting an 8-bit number left by one bit

PROGRAM 4.15

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8010	21, 00, 80		LXI	H,8000	Load memory address of data 8000H in HL pair
8013	7E		MOV	A, M	Move data in accumulator
8014	07		RLC		Content of accumulator rotate left by one bit
8015	23		INX	H	Increment HL register pair
8016	77		MOV	M, A	Store the result in memory location 8001H
8017	76		HLT		Halt

A number will be shifted by one bit when the same number is added with itself. Actually, the number will be doubled. For example, if the number is 02H, after shifting one bit left the number becomes 04 H. Therefore RLC instruction can be replaced by ADD A.

Example 4.21

Data		Result	
Memory location	Data	Memory location	Data
8050	04H	8051	08H

4.11.14 Shift a 16-bit Number Left by One Bit

Assume a 16-bit number is stored in 8050H and 8051H locations. After shifting one bit, the result will be stored in 8053 and 8054 locations. The program memory starts from 8100H. The program flow chart for shift a 16-bit number left by one bit is shown in Fig. 4.24.

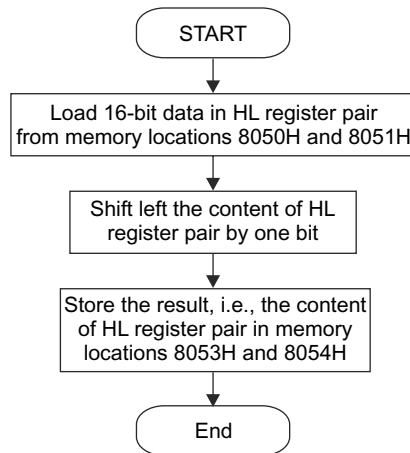


Fig. 4.24 Flow chart for shift a 16-bit number left by one bit

Algorithm

1. Load 16-bit or 2-byte data from memory location to HL register pair.
2. Contents of HL register pair are added to itself once and the result stored in HL pair for shift left by one bit
3. Store the result in 8053H and 8053H locations.

PROGRAM 4.16

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8100	2A, 50, 80		LHLD	8050 H	Load data from memory location 8050H and 8051 to HL register pair
8103	29		DAD	H	Shift 16-bit number by one bit
8104	22, 53, 80		SHLD	8053 H	Store result in 8053H and 8054H memory locations
8107	76		HLT		

Example 4.22

Data		Result	
Memory location	Data	Memory location	Data
8050	52H	8053	A4H LSBs of result
8051	85H	8054	0AH MSBs of result

4.11.15 Find out the Largest of Two Numbers

The first number is stored in the memory location 8050H and the second number is placed in 8051H location. The result will be stored in the memory location 8052. Assume the program memory starts from 8100H. The program flow chart is shown in Fig. 4.25.

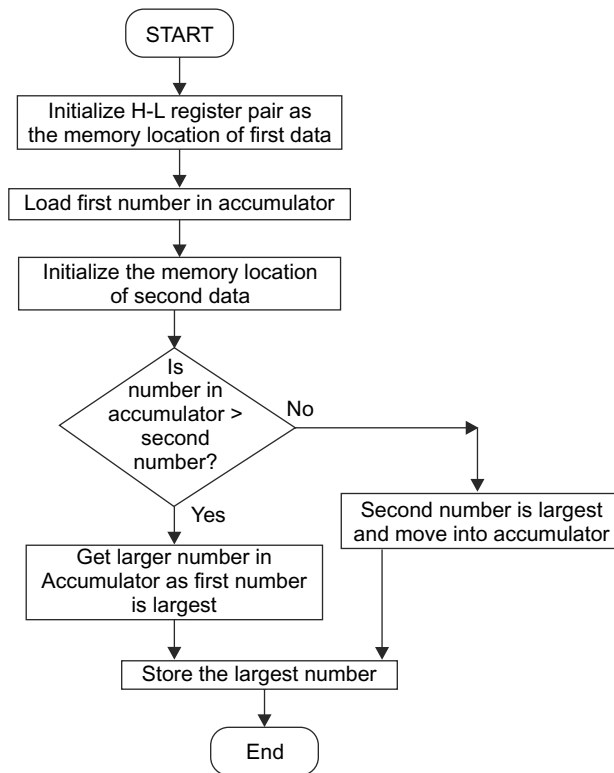


Fig. 4.25 Flow chart to find out the largest of two numbers

Algorithm

1. Load the first number in accumulator from the memory location 8050H.
2. Compare second number with first number.
3. If second number is greater than first number, copy Second number in the accumulator from memory.
4. Store the result in 8052H location.

PROGRAM 4.17

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8100	21, 50, 80		LXI	H,8050	Load the 1st number in accumulator from memory location 8050H
8103	7E		MOV	A,M	
8104	23		INX	H	Address of 2nd number in HL pair
8105	BE		CMP	M	Compare between 2nd number and 1st number
8106	D2, 0A, 81		JNC	LEVEL	If borrow (carry) is not generated, jump LEVEL
8109	7E		MOV	A,M	Move 2nd number in accumulator
810A	32	LEVEL	STA	8052	Store largest number in 8052H
810B	76		HLT		Halt

Example 4.23

Data		Result	
Memory location	Data	Memory location	Data
8050	78H	8052	FFH
8051	FFH		

4.11.16 Find out the Largest Number from an Array of Numbers

The count value of numbers 05H is stored in Register C directly and the numbers are stored in the memory locations from 9001H to 9005H. The largest number will be stored in 9006H location. Assume the program memory starts from 9100H. The flowchart to find out the largest number from an array is depicted in Fig. 4.26.

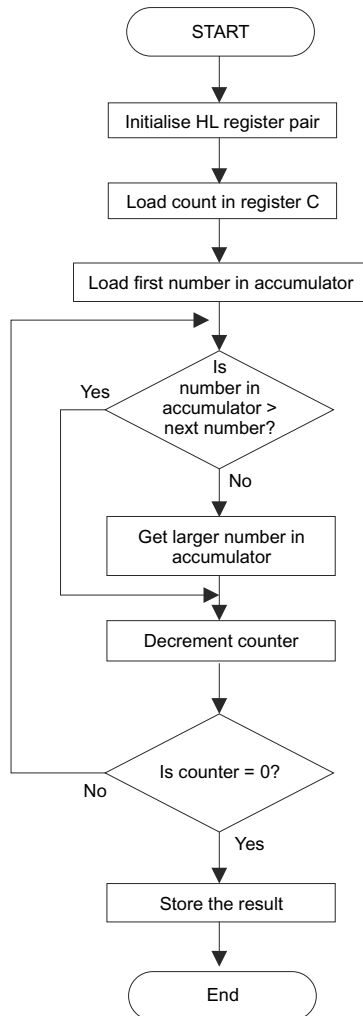


Fig. 4.26 Flowchart to find out the largest number from an array

Algorithm

1. Load count value of numbers 05H in Register C immediately.
2. Load the first number in accumulator from the memory location 9001H.
3. Move the first number in the accumulator.
4. Decrement the count value by one.
5. Move to the next memory location for next data.
6. Compare the content of memory with content of accumulator.
7. If carry is generated, copy content of memory in accumulator.
8. Decrement the count value by one.
9. If count value does not equal to zero, repeat steps 5 to 8.
10. Store the result in 9006H location.

PROGRAM 4.18

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
9100	0E, 05		MVI	C, 05	Load cout value in Register C
9102	21, 01, 90		LXI	H, 9001	Load address of first data in HL register pair
9105	7E		MOV	A, M	Copy 1st data in accumulator
9106	0D		DCR	C	Decrement Register C
9107	23	LOOP	INX	H	Increment HL register for address of next data
9108	BE		CMP	M	Compare next data with the content of accumulator
9109	D2,0D,91		JNC	LEVEL	If carry is not generated, jump to LEVEL
910C	7E		MOV	A, M	Copy large number in accumulator from memory
910D	0D	LEVEL	DCR	C	Decrement Register C
910E	C2, 07, 91		JNZ	LOOP	Jump not zero to LOOP
9111	32, 06, 90		STA	9006	Store largest number in 9006H location
9114	76		HLT		

Example 4.24

<i>Data</i>		<i>Result</i>	
Memory location	Data	Memory location	Data
9001	23H	9006	FFH
9002	FFH		
9003	47H		
9004	92H		
9005	10H		

4.11.17 Find out the Smallest of Two Numbers

The first number is placed in the memory location 9050H and the second number is placed in the 9051H location. The smallest number is to be stored in the memory location 9052. Consider the program memory starts from 9100H. The program flow chart to find out the smallest of two numbers is depicted in Fig. 4.27.

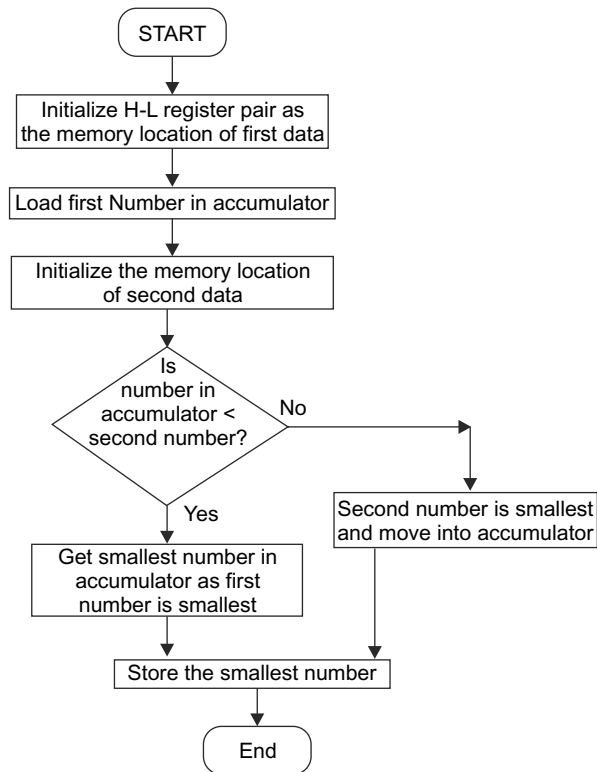


Fig. 4.27 Flow chart to find out the smallest of two numbers

Algorithm

1. Address of the first number is in HL register pair.
2. Move first number into accumulator.
3. Increment HL register pair for addressing second number.
4. Compare second number with first number.
5. When the first number is less than second number, the content of accumulator is the smallest number.
If second number is less than first number, copy second number in accumulator from memory.
6. Store the result in 9052H location.

PROGRAM 4.19

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
9100	21, 50, 90		LXI	H, 9050H	Load the 1st number in accumulator from memory location 9050H
9103	7E		MOV	A,M	
9104	23		INX	H	Address of 2nd number in HL pair
9105	BE		CMP	M	Compare 2nd number and 1st number
9106	DA,--		JC	LEVEL	If borrow (carry) is generated, jump LEVEL
9109	7E		MOV	A,M	Move 2nd number in accumulator
910A	32, 52, 90	LEVEL	STA	9052H	Store smallest number in 9052H
910D	76		HLT		Halt

Example 4.25

Data		Result	
Memory location	Data	Memory location	Data
9050	78H	9052	78H
9051	FFH		

4.11.18 Find out the Smallest Number from an Array of Numbers

A series of five numbers: (01H, FFH, 27H, 44H, 65H) are stored in memory locations from 8001H to 8005H. The largest number will be stored in the 8006H location. Assume the program memory starts from 8100H. The program flow chart to find out the smallest number from an array of numbers is shown in Fig. 4.28.

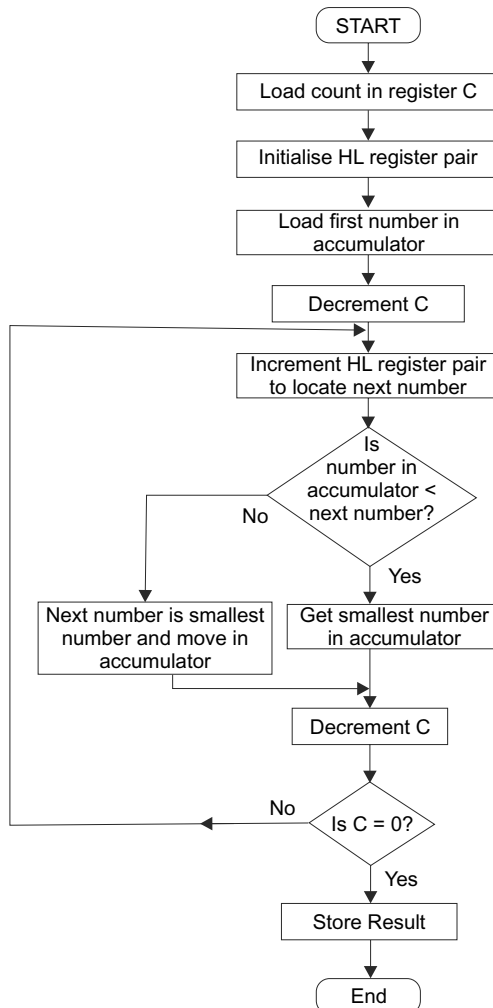


Fig. 4.28 Flow chart to find out the smallest from an array of numbers

Algorithm

1. Store count of numbers 05H in Register C immediately.
2. Load the first number in accumulator from memory location 8001H.
3. Decrement the count value by one.
4. Move to next memory location for next data.
5. Compare the content of memory with content of accumulator.
6. If carry is not generated, copy the content of memory in accumulator.
7. Decrement the count value by one.
8. If count value does not equal to zero, repeat steps 4 to 8.
9. Store result in 8006H location.

PROGRAM 4.20

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8100	0E, 05		MVI	C,05	Load cout value in Register C
8102	21, 01, 80		LXI	H,8001	Load address of first number in HL register pair
8105	7E		MOV	A,M	Copy first number in accumulator
8106	0D		DCR	C	Decrement Register C
8107	23	LOOP	INX	H	Increment HL register for address of next number
8108	BE		CMP	M	Compare next number with the content of accumulator
8109	DA, 0D, 81		JC	LEVEL	If carry is generated, jump to LEVEL
810C	7E		MOV	A,M	Copy large number in accumulator from memory
810D	0D	LEVEL	DCR	C	Decrement Register C
810E	C2, 07, 81		JNZ	LOOP	Jump not zero to LOOP
8111	32, 06, 80		STA	8006	Store smallest number in 9006H location
8104	76		HLT		

Example 4.26

<i>Data</i>		<i>Result</i>	
Memory location	Data	Memory location	Data
8001	01H	8006	01H
8002	FFH		
8003	27H		
8004	44H		
8005	65H		

4.11.19 Arrange a Series of Numbers in Descending Order

A series of five numbers (11H, 05H, 46H, 23H, 65H) are stored in memory locations from 9001H to 9005H. Arrange the above numbers in descending order and to be stored in 9001H to 9005H locations. Assume the program memory starts from 9100H.

Algorithm

1. Store 05H, number of data to be arranged in Register C from memory and store number of comparisons in Register D.
2. Initialize the memory location 9001H of first data.
3. Load the first data in accumulator from memory.
4. Increment HL register pair for addressing next data.
5. Load the next data in Register B from memory.
6. Compare next data with accumulator. Store the smallest number in accumulator and largest number in memory.
7. Then next number is compared with accumulator and store the largest number in memory and smallest number in accumulator.

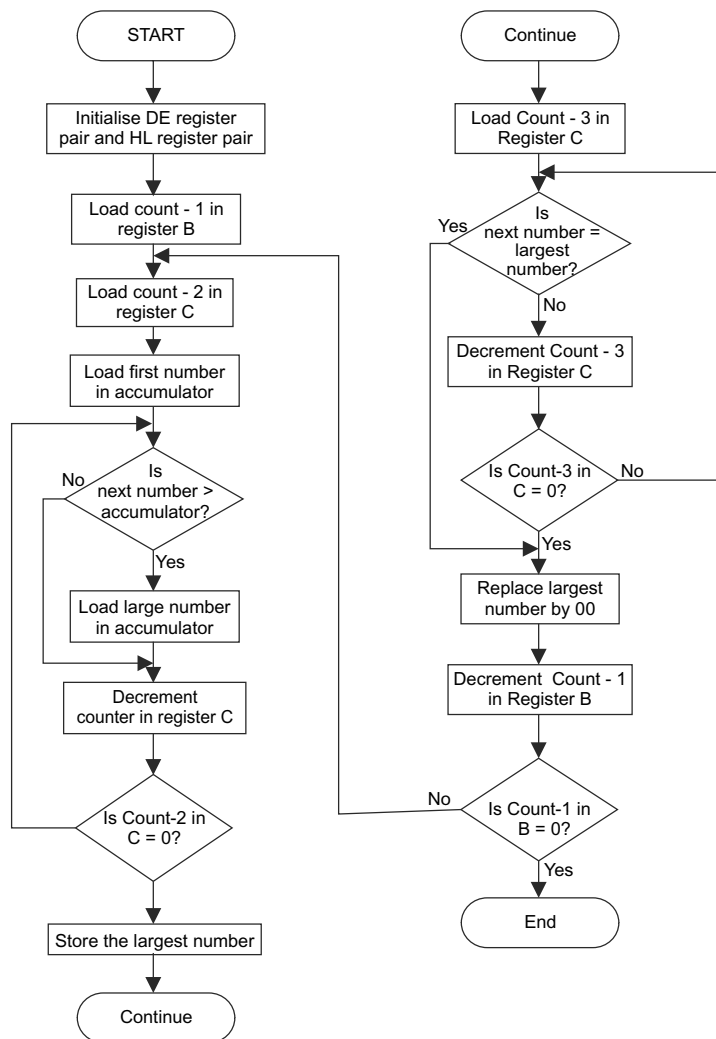


Fig. 4.29 Flow chart for arranging a series of numbers in descending order

8. This process will continue till comparison of all numbers have been completed. After completion of comparison of all numbers, the smallest number in accumulator and store it in memory. In this way, the first process will be completed.
9. At the starting of second process, Register C is decremented and store number of comparisons in Register D Then repeat steps 2 to 8. After completion of this process, the smallest number is in 9005H and the second smallest number is in 9004H.
10. Register C is decremented and the next process starts if the content of Register C is not zero.

The flowchart for arranging a series of numbers in descending order is depicted in Fig. 4.29.

PROGRAM 4.21

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
9100	0E, 05		MVI	C, 05H	Load count value of number of data in Register C
9102	16, 05	START	MVI	D, 05	Load count for number of comparisons in Register D
9104	21, 01, 90		LXI	H, 9001H	Load memory location of 1st number
9107	7E		MOV	A, M	1st number in accumulator
9108	23	LOOP	INX	H	Increment HL register pair for addressing next number
9109	46		MOV	B, M	Copy next number in Register B from memory
910A	B8		CMP	B	Compare next number with accumulator
910B	DA, 17, 91		JC	LEVEL 1	If the content of accumulator > next number, jump to LEVEL 1
910E	2B		DCX	H	Increment HL register pair to locate the addressing for storing largest number
910F	77		MOV	M,A	Store largest of the two numbers in memory
9110	78		MOV	A,B	Move smallest of the two numbers in accumulator from Register B
9111	C3, 19, 91		JMP	LEVEL_2	Jump to LEVEL_2
9114	2B		DCX	H	
9115	77		MOV	M,A	Place smaller of the two numbers in accumulator
9116	C3		JMP		Jump to LEVEL_2
9117	2B	LEVEL 1	DCX	H	Store largest of the two numbers in memory
9118	70		MOV	M,B	
9119	23	LEVEL 2	INX	H	
911A	15		DCR	D	Decrement Register D to count for number of comparisons
911B	C2, 08, 91		JNZ	LOOP	Jump zero to
911E	77		MOV	M,A	Place smallest number in memory
911F	0D		DCR	C	Decrement count value
9120	C2, 02, 91		JNZ	START	Jump not zero to START
9123	76		HLT		Halt

Example 4.27

Data		Result				
Memory location	Data	Memory location	After 1st process	After 2nd process	After 3rd process	After 4th process
9001	11H	9001	11H	46H	46H	65H
9002	05H	9002	46H	23H	65H	46H
9003	46H	9003	23H	65H	23H	23H
9004	23H	9004	65H	11H	11H	11H
9005	65H	9005	05H	05H	05H	05H

4.11.20 Arrange a Series of Numbers in Ascending Order using a Subroutine

A series of five numbers are stored in memory locations in 8001H to 8005H. These numbers are 56H, F4H, 22H, 9AH and A1H.

Arrange the above numbers in ascending order in 9001H to 9005H locations using a subroutine. Assume the program memory starts from 9100H. The program flow chart to arrange a series of numbers in ascending order using subroutine is depicted in Fig. 4.30.

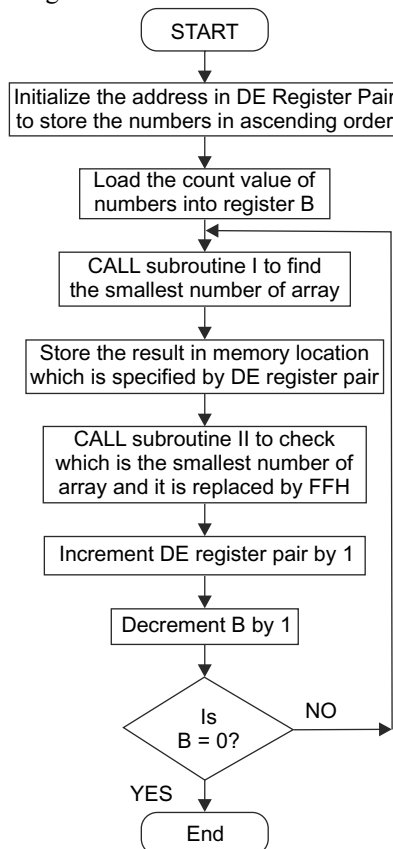


Fig. 4.30 Flow chart to arrange a series of numbers in ascending order using a subroutine

Algorithm

1. Find the smallest number from the given series of numbers and store it in the memory location 9001H.
2. Then detect the memory location of the smallest number from the given series and store FFH in that memory location. As the content of the memory location is replaced by FFH, initial series of numbers will be modified and a new series will be developed.
3. Decrement count value by one.
4. Again find the smallest number from the modified series of numbers and store it in the next memory location.
5. Then detect the memory location of the smallest number from the given series and place FFH in that memory location. As the content of the memory location is replaced by FFH, the array is again modified.
6. Decrement count value by one.
7. Repeat steps 4 to 6, till the counter value becomes zero and all the numbers of the given series are arranged in descending order.

The program is divided into three subparts, namely, main program, Subroutine-I and Subroutine-II. The main program is used to call Subroutine-I and Subroutine-II. Application of Subroutine-I is to find the smallest number of a series of numbers and Subroutine-II is used to detect the location of the smallest number and replace it by FFH.

MAIN PROGRAM 4.22

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
9100	11, 01, 90		LXI	D, 9001H	Load memory locations in DE register pair to store the result
9103	21, 01, 80		LXI	H, 8001H	Load count address in HL register pair
9106	46		MOV	B, M	Copy count values in Register B to check whether all numbers have been arranged in ascending order.
9107	CD, 00, 85	START	CALL	8500H	Call Subroutine-I to find smallest number of array
910A	12		STAX	D	Store the result in memory location whose address is in DE register pair
910B	CD, 00, 86		CALL	8600H	Call Subroutine-II to check which number is smallest
910E	13		INX	D	Increment in DE register pair to store next smallest number
910F	05		DCR	B	Decrement Register B
9120	C2, 07, 91		JNZ	START	If the content of B is not equal to zero, repeat process
9123	76		HLT		Stop

PROGRAM for SUBROUTINE-I

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8500	21, 00, 80		LXI	H,8000H	Load address of number of data in HL register pair
8503	4E		MOV	C,M	Load count value in Register C
8504	23		INX	H	Increment HL register pair to locate first data

Contd.

Contd.

8505	7E		MOV	A,M	Copy 1st data in accumulator
8506	0D		DCR	C	Decrement Register Cc
8507	23	LOOP_1	INX	H	Increment HL register for address of next data
8508	BE		CMP	M	Compare next data with the content of accumulator
8509	DA, 0D, 85		JC	LEVEL_1	If carry is not generated, jump to LEVEL_1
850C	7E		MOV	A,M	Copy smallest number in accumulator from memory
850D	0D	LEVEL_1	DCR	C	Decrement Register C
850E	C2, 07, 85		JNZ	LOOP_1	Jump not zero to LOOP
8511	C9		RET		

PROGRAM for SUBROUTINE-II

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8600	21, 00, 80		LXI	H, 8000 H	Copy the count value from memory location 8000H to Register C
8603	4E		MOV	C, M	
8604	23	LOOP_2	INX	H	Increment HL register for address of next number
8605	BE		CMP	M	Compare the next number with smallest number in accumulator
8606	CA, 06, 86		JZ	LEVEL 2	If the present number is smallest number, jump to LEVEL 2
8609	0D		DCR	C	Decrement C register
860A	C2, 04, 86		JNZ	LOOP_2	If C is not zero, jump to take up next number
860D	3E, FF	LEVEL_2	MVI	A, FF	
860F	77		MOV	M, A	Replace smallest number by FFH
8610	C9		RET		

Example 4.28

Data		Modified Array					
Memory location	Data	Memory location	After 1st process	After 2nd process	After 3rd process	After 4th process	After 5th process
8001	56H	8001	56H	FFH	FFH	FFH	FFH
8002	F4H	8002	F4H	F4H	F4H	F4H	FFH
8003	22H	8003	FFH	FFH	FFH	FFH	FFH
8004	9AH	8004	9AH	9AH	FFH	FFH	FFH
8005	A1H	8005	A1H	A1H	A1H	FFH	FFH

Result	
Memory location	Data
9001	22FH
9002	56H
9003	9AH
9004	A1H
9005	F4H

4.11.21 Find out Square Root of 0, 1, 4, 9, 16, 25, 36, 49, 64 and 81 using a Look-Up Table

Load the number in the memory location 9000H and the square root of the number will be stored in the memory location 9001H. The square roots of numbers 0, 1, 4, 9, 16, 25, 36, 49, 64 and 81 are stored in 8500H, 8501H, 8504H, 8509H, 8516H, 8525H, 8536H, 8549H, 8564H and 8581H locations respectively as given in tabular form. Assume the program is written from memory location 9100H. The program flow chart to find out square root a decimal number using look up table is given in Fig. 4.31.

Algorithm

1. Store the number in the accumulator from the memory location 9000H.
2. Move the content of accumulator in Register L and store 85H in Register H.
3. When the number is 16, the content of H and L registers are 85H and 16H respectively. Then the HL register pair represents the 8516H memory location.
4. Copy the square root of the number in the accumulator from the memory location which is represented by HL register pair.
5. Store the result in 9001H.

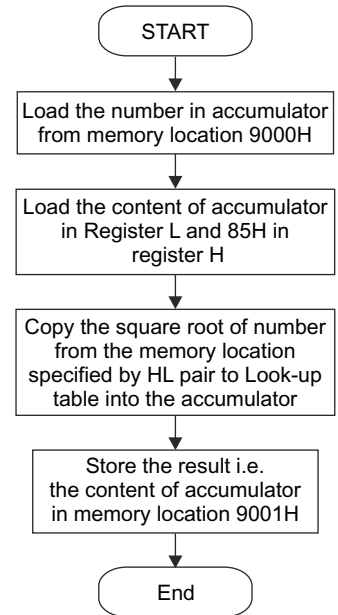


Fig. 4.31 Flow chart to find out the square root of a number using look-up table

PROGRAM 4.23

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
9100	3A, 00, 90		LDA	9000H	Load the number in accumulator from memory location 9000H
9103	6F		MOV	L, A	Copy the content of Accumulator in Register L
9104	26, 85		MVI	H, 85	Load 85H in Register H
9106	7E		MOV	A, M	Move square root of decimal number in accumulator from memory
9107	32, 01, 90		STA	9001H	Store square root value in 9001H
910A	76		HLT		

Look-up Table

ADDRESS	SQUARE ROOT
8500	00
8501	01
8504	02
8509	03
8516	04

(Contd.)

(Contd.)

8525	05
8536	06
8549	07
8564	08
8581	09

Example 4.29

ADDRESS	SQUARE	ADDRESS	Result
9000 H	16	9001 H	04

4.11.22 Multiplication of Two 8-bit Numbers

Two eight-bit data are stored in 8000H and 8001H memory locations. After multiplication, the result will be stored in 9000H and 9001H memory locations. Assume the program is written from memory location 9100H.

Algorithm for Repetitive Addition

1. Store multiplicand in Register B and multiplier in Register E.
2. Clear Register D and clear HL register pair.
3. Content of DE registers will be added with content of HL registers.
4. Decrement Register B.
5. If content of Register B is not equal to zero, repeat steps 3 to 5.
6. When Register B is equal to zero, the content of HL will be stored in memory locations 9000H and 9001H.

PROGRAM 4.24

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
9100	21, 00, 80		LXI	H,8000H	Address of multiplier in HL pair
9103	4E		MOV	C,M	Store multiplier in Register C from memory
9104	24		INX	H	Address of multiplicand in HL pair
9105	5E		MOV	E,M	Multiplicand in Register E
9106	16, 00		MVI	D,00H	Load 00H in Register D
9108	21, 00, 00		LXI	H, 0000	Initial value of product = 00H in HL pair
910B	19	LOOP	DAD	D	Add content of DE with content of HL
910C	0D		DCR	C	Decrement Register C
910D	C2, 0B, 91		JNZ	LOOP	If not zero, jump to LOOP
9110	EB		XCHG		The content of DE register pair and HL register pair exchanged; result in DE register
9111	21, 00, 90		LXI	H, 9000H	Load 9000H in HL pair
9114	73		MOV	M,E	Store content of Register E in 9000H location
9115	23		INX	H	Address of next memory location in HL pair
9116	72		MOV	M,D	Store content of Register E in 9001H location
9117	76		HLT		Stop

Example 4.30

Address	Data	Address	Result
8000 H	45H Multiplicand	9000 H	05H
8001 H	13H Multiplier	9001 H	FFH

Binary Multiplication

The other method of multiplication is binary multiplication. If the multiplicand is multiplied by 1, the product will be equal to the multiplicand. If the multiplicand is multiplied by zero, the product is zero.

For binary multiplication, the following procedure is followed :

Example 4.31

45 H Multiplicand
 × 13 H Multiplier
 05 1FH Product

```

      0 1 0 0 0 1 0 1
    × 0 0 0 1 0 0 1 1
    -----
      0 1 0 0 0 1 0 1
     0 1 0 0 0 1 0 1 x
    0 0 0 0 0 0 0 0 x x
    0 0 0 0 0 0 0 0 x x x
    0 1 0 0 0 1 0 1 x x x x
    -----
    0 1 0 1 0 0 0 1 1 1 1 1 (05 1F)H
    
```

Step 1 The multiplicand is multiplied by the LSB of the multiplier and the partial product is stored. Then multiplicand is shifted right.

Step 2 Again the shifted multiplicand is multiplied by the second bit and then added with the previous result. Then the shifted multiplicand is shifted right. If the bit is a 0 bit, nothing will be added with the partial product but the multiplicand is simply shifted right by one bit.

Step 3 The step will be repeated till the completion of multiplication of all bits of the multiplier.

In binary multiplication, the multiplicand is shifted right and shift multiplier left to check the LSB bit whether it is 1 or 0. The flowchart for multiplication of two numbers is shown in Fig. 4.32.

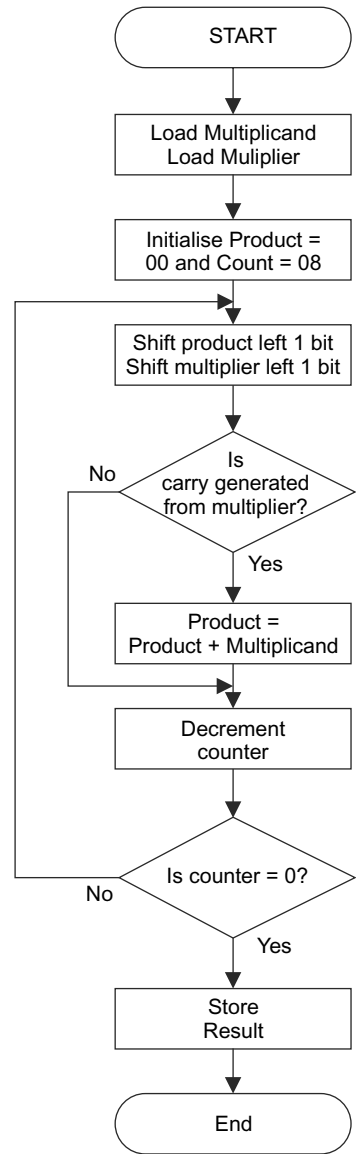


Fig. 4.32 Flow chart for multiplication of two numbers

Algorithm

1. Load the multiplicand and multiplier.
2. Initialize product value = 0.
3. Load number of bits of multiplier in Register C.
4. Shift multiplier right by one bit.
5. If carry flag is set, multiplicand adds with initial value 0000H + multiplicand. Then product is equal to 0000H+ multiplicand. This result is also called as partial product. Then partial product is shifted left by one bit.
6. DCR counts value.
7. If the content of Register C is not zero, modified multiplier again shifted one bit right.
8. If carry flag is set, shifted multiplicand adds with partial product. Then once again shifts the modified multiplicand left.
9. Repeat steps 6, 7 and 8 till the content of Register C becomes zero.
10. Store the result in 9000H and 9001H.

PROGRAM 4.25

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
9100	21, 00, 80		LXI	H,8000H	Address of multiplicand in HL pair
9103	5E		MOV	E,M	Store multiplicand in Register E from memory
9104	23		INX	H	Address of multiplicand in HL pair
9105	56		MOV	D,M	Multiplicand in Register D
9106	0E, 08		MVI	C,08H	Load 08H in Register C
9108	3A, 02, 80		LDA	8002H	Load multiplier in accumulator
910B	21, 00, 00		LXI	H, 0000H	Initial value of product = 00 in HL pair
910E	0F	LOOP	RRC		Rotate accumulation left
910F	D2, 13, 91		JNC	LEVEL	If there is no carry, jump to level
9112	19		DAD	D	Add content of DE with content of HL
9113	EB	LEVEL	XCHG		The content of DE register pair and HL register pair exchanged, result in DE register
9114	19		DAD	H	Multiplicand shifted one bit right
9115	EB		XCHG		The content of DE register pair and HL register pair exchanged, result in DE register
9116	37		STC		Clear the carry flag using set carry status and then complement the carry status
9117	3F		CMC		
9118	0D		DCR	C	Decrement Register C
9119 LOOP	C2, 0E, 91		JNZ	LOOP	If content of Register C is not zero, jump to LOOP
911C	22, 00, 90		SHLD	9000H	Store the content of HL register pair in 9000H and 9001 memory location
911F	76		HLT		Stop

Example 4.32

Address	Data	Address	Result
8000 H	45H Multiplicand	9000 H	05H
8001 H	13H Multiplier	9001 H	FFH

4.11.23 Division of Two 8-bit Numbers

Repetitive Subtractions The division can be performed by repetitive subtractions. The divisor is subtracted from the dividend. When there is no borrow, the quotient is incremented by one. If there is borrow, the quotient and remainder are stored in specified memory location. Assume dividend and divisor are nonzero quantities. Assume the program starts from memory location 9100H.

Algorithm

1. Store the dividend in the memory location 8000H and the divisor in the memory location 8001H.
2. Clear Register C by storing 00H within it.
3. Move dividend in accumulator and copy it in Register D.
4. Subtract divisor from dividend.
5. If carry is not generated, increment Register C. Repeat steps 3 to 5.
6. When carry is generated, quotient, content of Register C, and remainder, content of Register D are stored in memory location.
7. If zero flag is set, Register C is incremented by one. Then quotient, content of Register C and remainder, content of Register D are stored in memory location.

The flowchart for division of two numbers is illustrated in Fig. 4.33.

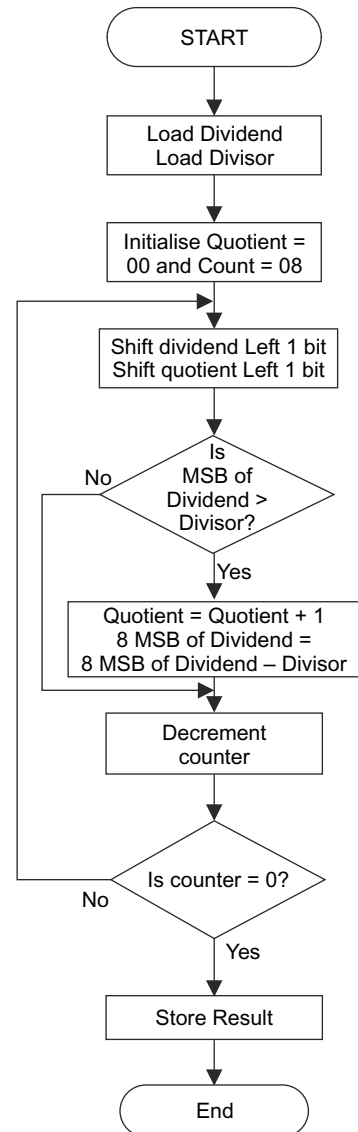


Fig. 4.33 Flow chart for division of two numbers

PROGRAM 4.26

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
9100	21, 00, 80		LXI	H,8000H	Address of dividend in HL pair.
9103	36, 22 (Dividend)		MVI	M, Dividend	Store dividend in memory location
9105	23		INX	H	Address of divisor in HL pair
9106	36, 21 (Divisor)		MVI	M, Divisor	Store divisor in memory location
9108	0E, 00		MVI	C,00H	Load 00H in Register C for initial value of quotient
910A	2B		DCX	H	Decrement HL register pair
910B	7E	LOOP	MOV	A,M	Load dividend in accumulator from memory
910C	57		MOV	D,A	Copy dividend in Register D
910D	23		INX	H	Increment HL register pair
910E	96		SUB	M	Subtract divisor from dividend
910F	DA, 1B, 91		JC	LEVEL_1	If there is carry, jump to LEVEL_1
9112	CA, 20, 91		JZ	LEVEL_2	If there is zero, jump to LEVEL_2
9115	2B		DCX	H	Decrement HL register pair
9116	77		MOV	M,A	Store modified dividend in memory location from accumulator
9117	0C		INR	C	Increment Register C
9118	C3, 0B, 91		JMP	LOOP	
911B	37	LEVEL 1	STC		Clear the carry flag using set carry status and then complement the carry status
911C	3F		CMC		
911D	C3, 21, 91		JMP	LEVEL_3	Jump to LEVEL_3
9120	0C	LEVEL_2	INR	C	
9121	21, 00, 90	LEVEL_3	LXI	H, 9000H	
9124	71		MOV	M,C	Store quotient in 9000H from Register C
9125	23		INX	H	Increment HL register pair
9126	72		MOV	M,D	Store remainder in 9000H from Register C
9127	76		HLT		Stop

Binary Division

Binary division is also performed by trial subtractions. The divisor is subtracted from the 8 most significant bits of the dividend. When there is no borrow, the bit of the quotient is set to 1; otherwise, 0. Then the dividend and quotient are shifted left by one bit before the next subtraction. The dividend and quotient can use a 16-bit register. As dividend is shifted, one bit of the register falls vacant in each step and the quotient is stored in unoccupied bit positions.

The dividend is a 16-bit number and divisor, an 8-bit number. When the dividend is an 8-bit number, place 00H in MSBs positions. The dividend is stored in the memory locations 8000H and 8001H. The divisor

is placed in the memory location 8002H. The results will be stored in the memory locations 8003H and 8004H.

PROGRAM 4.27

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
9100	2A, 00, 80		LHLD	8000 H	Store dividend in HL pair
9103	3A, 02, 80		LDA	8002H	Store divisor in accumulator from memory location 8002H
9106	47		MOV	B,A	Copy the content of accumulator in Register B
9107	0E, 08		MVI	C, 08	Load 08H in Register C
9109	29	LOOP	DAD	H	Shift dividend and quotient right by one bit
910A	7C		MOV	A,H	Move MSBs of dividend in accumulator
910B	90		SUB	B	Subtract divisor from dividend
910C	DA, 11, 91		JC	LEVEL_1	If there is carry, jump to LEVEL_1
910F	67		MOV	H,A	After subtraction store dividend in Register H from accumulator
9110	2C		INR	L	Increment L register
9111	0D	LEVEL_1	DCR	C	Decrement C register
9112	C2, 12, 91		JNZ	LOOP	If there is no zero, jump to LOOP
9115	22, 03, 80		SHLD	8003H	Store results in 8003 and 8004 H
9118	76		HLT		Stop

Example 4.33

<i>Address</i>	<i>Data</i>	<i>Address</i>	<i>Result</i>
8000 H	9A H LSBs of dividend	9003H	F2 Quotient
8001 H	48 H MSBs of dividend	9004 H	06 Remainder
8002 H	1A H Divisor		

4.11.24 Convert an 8-bit Hexadecimal Number to Binary Number

Store an 8-bit data in 8000H location and load 08H in Register C. Transfer data from memory to accumulator. The content of accumulator is shifted right with carry and store carry bit in the 8058H location. After that clear carry. Again content of the accumulator is shifted right with carry, and the content of carry will be stored in the previous memory location. In this way the operation will be repeated till Register C becomes zero.

Assume the program starts from the memory location 8100H. The program flow chart to convert a 8-bit Hexadecimal number to binary number is illustrated in Fig. 4.34.

Algorithm

1. Initialize memory location 8000H. Load an 8-bit hexadecimal number in the memory.
2. Initialize the memory location to store result.
3. Load 8-bit data in accumulator.

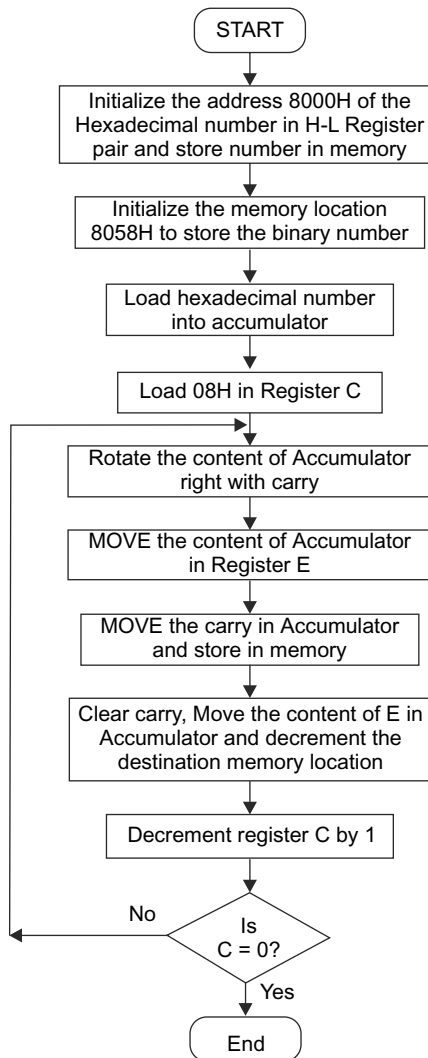


Fig. 4.34 Flow chart to convert hexadecimal number to binary number

4. Load 08H in Register C.
5. Rotate accumulator right with carry.
6. Copy content of accumulator in Register E.
7. Save carry in accumulator.
8. Store in memory.
9. Transfer Register E to accumulator.
10. Clear carry.
11. Decrement HL pair.
12. Decrement Register C.
13. Repeat steps 6 to 12 till the content of Register C becomes zero.

PROGRAM 4.28

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8100	21, 00, 80		LXI	H, 8000H	Initialize the memory location 8000 H through HL register pair
8103	36, DATA		MVI	M, DATA	Load DATA in the memory
8105	21, 58, 80		LXI	H, 8058H	Initialize the memory Location 8058H to store binary
8108	3A, 00, 80		LDA	8000 H	Load data in accumulator from the memory location 8000H
810B	0E, 08		MVI	C, 08H	Store 08H in Register C
810D	1F	LOOP	RAR		Rotate the content of accumulator right with carry
810E	5F		MOV	E, A	Copy the content of accumulator in Register E
810F	3E, 00		MVI	A, 00H	Store 00H in accumulator
8111	8F		ADC	A	Add the contents of accumulator and carry
8112	77		MOV	M,A	Move the content of accumulator in memory
8113	7B		MOV	A, E	Transfer the content of accumulator in Register E
8114	B7		ORA	A	Clear carry
8115	2B		DCX	H	Decrement HL register pair
8116	0D		DCR	C	Decrement Register C
8117	C2, 0D, 81		JNZ	LOOP	If content of C is not zero, Jump to LOOP
811A	76		HLT		Halt

Example 4.34

<i>8-Bit DATA</i>	<i>Binary number in Memory Location</i>							
	<i>8051H</i>	<i>8052H</i>	<i>8053H</i>	<i>8054H</i>	<i>8055H</i>	<i>8056H</i>	<i>8057H</i>	<i>8058H</i>
27	0	0	1	0	0	1	1	1
9F	1	0	0	1	1	1	1	1

4.11.25 Transfer a Block of Data from One Section of Memory to the Other Section of Memory

A block of data is available starting from 9051H. Transfer the block so that it can be stored from 9100H. The number of bytes in the block is stored in 9050H. Assume the program starts from 8000H. The program flow chart is depicted in Fig. 4.35.

Algorithm

1. Store the address of the number of data in HL register pair.
2. Load number of data in Register B from memory.
3. Store the starting address of destination in DE register pair.

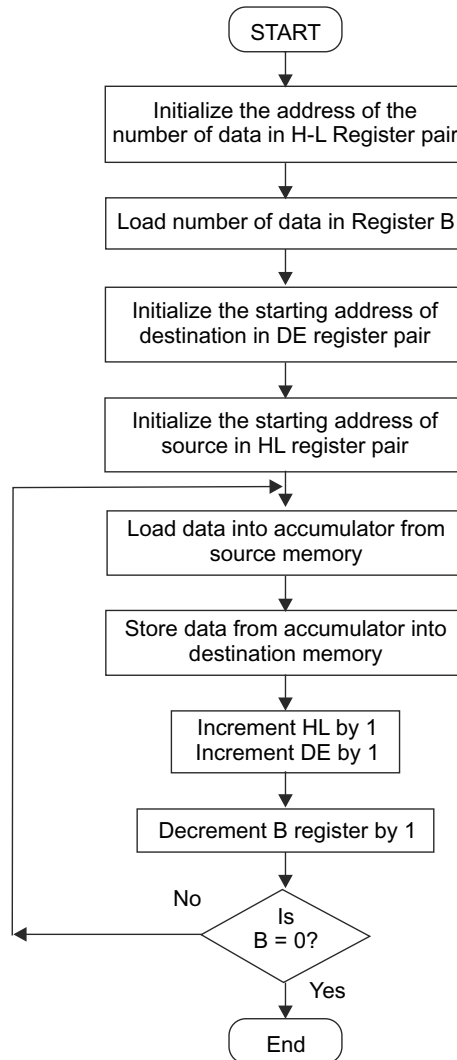


Fig. 4.35 Flow chart to transfer a block of data from one section of memory to the other section of memory

4. Increment HL register pair to get data from source.
5. Copy data from source to accumulator.
6. Exchange HL and DE register pair, store the content of accumulator in destination address.
7. Exchange HL and DE register pair.
8. Increment HL and DE register.
9. Decrement Register B.
10. If Register C is not zero, repeat steps 5 to 9.

PROGRAM 4.29

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	21, 50, 90		LXI	H,9050	Store the address of number of data, 9050H in HL register pair
8003	46		MOV	B,M	Load number of data in Register B from memory
8004	21, 00, 91		LXI	D,9100	Store the destination address in DE register pair
8007	23		INX	H	Increment HL register pair
8008	7E	LOOP	MOV	A,M	Move data from source to accumulator
8009	EB		XCHG		Exchange the content of HL and DE
800A	77		MOV	M,A	Store the content of accumulator, data in destination address
800B	EB		XCHG		Exchange the content of HL and DE
800C	23		INX	H	Increment source address
800D	13		INX	D	Increment destination address
800E	05		DCR	B	Decrement Register B.
800F	C2, 08, 80		JNZ	LOOP	If B is not zero, Jump to LOOP
8012	76		HLT		

Example 4.35

<i>Input</i>		<i>Result</i>	
<i>ADDRESS</i>	<i>DATA</i>	<i>ADDRESS</i>	<i>DATA</i>
9050	05 H	9100	05 H
9051	48 H	9101	48 H
9052	1A H	9102	1AH
9053	F2 H	9103	F2H
9054	06 H	9104	06H
9055	33H	9105	33H

4.11.26 Find out Square of Decimal Number Using Look Up Table

Load the decimal number in memory location F000H and the square of the decimal number will be stored in the memory location F001H. The square values of decimal numbers from 0 to 9 are stored in F110 to F119H in tabular form as depicted in the Look-up table. Assume the program is written from memory location F150H. The program flow chart to find out the square of a number using a look-up table is shown in Fig.4.36.

Algorithm

1. Store the decimal number in the accumulator from memory location F000H.
2. Move the content of the accumulator in L register and Load F1H in H register.
3. If the decimal number is 05, the content of H and L registers are F1 and 05H respectively. Then the memory location F105H will be denoted by H-L register pair.
4. Move square of decimal number in the accumulator from memory location represented by H-L register pair.
5. Store the result, square value in F001H.

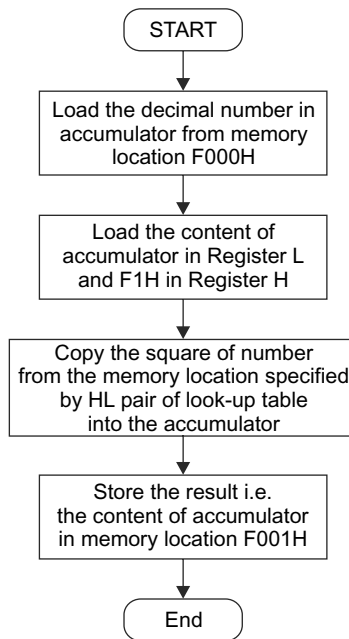


Fig. 4.36 Flow chart to find out the square of a number using look-up table

PROGRAM 4.30

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
F150	3A, 00, F0		LDA	F000H	Load the decimal number in accumulator from memory location F000H
F153	6F		MOV	L,A	Copy the content of accumulator in L register
F154	26, F1		MVI	H,F1	Load F1H in H register
F156	7E		MOV	A,M	Move square of decimal number in accumulator from memory
F157	32, 01, F0		STA	F001H	Store square value in F001H
F15A	76		HLT		Halt

Look-up Table

<i>ADDRESS</i>	<i>SQUARE</i>
F100 H	00
F101 H	01
F102 H	04
F103 H	09
F104 H	16
F105 H	25
F106 H	36
F107 H	49
F108 H	64
F109 H	81

Example 4.36

<i>Address</i>	<i>Square (Decimal)</i>	<i>Address</i>	<i>Result (Decimal)</i>
F000 H	05	F001 H	25

4.11.27 Convert a BCD Number to a Binary Number

Load the BCD number in memory location 8000H and the binary equivalent of BCD number will be stored in the memory location 8001H. Assume the program is written from memory location 8100H. The program flow chart to convert BCD number to binary number is depicted in Fig. 4.37.

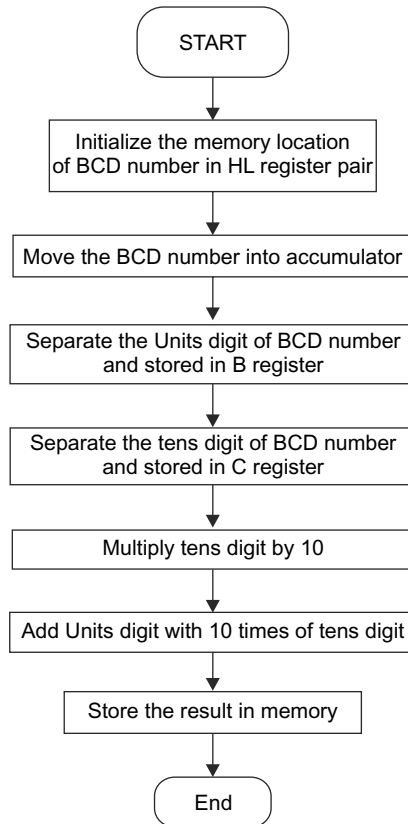


Fig. 4.37 Flow chart to convert BCD number to binary number

Algorithm

1. Store the BCD number in accumulator from memory location 8000H.
2. Separate the units digit of the BCD number and stored in B register.
3. Separate the tens digit of the BCD number and stored in C register.
4. Multiply tens digit by 10.
5. Add units digit with 10 times of tens digit.
6. Store the result, i.e., binary equivalent in 8001H.

PROGRAM 4.31

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8100	21, 00, 80		LXI	H,8000H	Load the address of a BCD number in HL register pair
8103	7E		MOV	A,M	Move data from memory to accumulator
8104	E6, 0F		ANI	0FH	Perform logical AND operation between 0FH and Accumulator content, i.e., mask units digit
8106	47		MOV	B,A	Store units digit in register B
8107	7E		MOV	A,M	Move data from memory to Accumulator
8108	E6, F0		ANI	F0H	Perform logical AND operation between 0FH and Accumulator content, i.e., mask tens digit
810A	0F		RRC		Rotate accumulator right four times, so that the tens digit is placed in units digit position
810B	0F		RRC		
810C	0F		RRC		
810D	0F		RRC		
810E	87		ADD	A	The content of A is added with A, i.e., tens digit becomes 2 times
810F	4F		MOV	C,A	Store 2 time of tens digit in C register
8110	87		ADD	A	tens digit becomes 4 times
8111	87		ADD	A	tens digit becomes 8 times
8112	81		ADD	C	Get 10 times of tens digit
8113	80		ADD	B	Add units digit
8114	23		INX	H	Increase HL pair by 1
8115	77		MOV	M,A	Store result in memory
8116	76		HLT		Halt

Example 4.37

<i>Address</i>	<i>Decimal Number</i>	<i>Address</i>	<i>Result (Binary)</i>
8000 H	68	8001 H	1010110

4.11.28 Convert an 8-bit Binary Number to Decimal Number

An 8-bit binary number is stored in memory location 8000H. Find out the decimal equivalent of binary number and decimal number will be stored in the memory locations 8001H and 8002H. Assume the program is written from memory location 8100H. The program flow chart to convert a 8-bit binary number to decimal number is shown in Fig. 4.38.

Algorithm

1. Initialize the register B = 00H as Hundreds counter and initialize the register C = 00H as tens counter.
2. Move the binary number into accumulator from memory location 8000H.

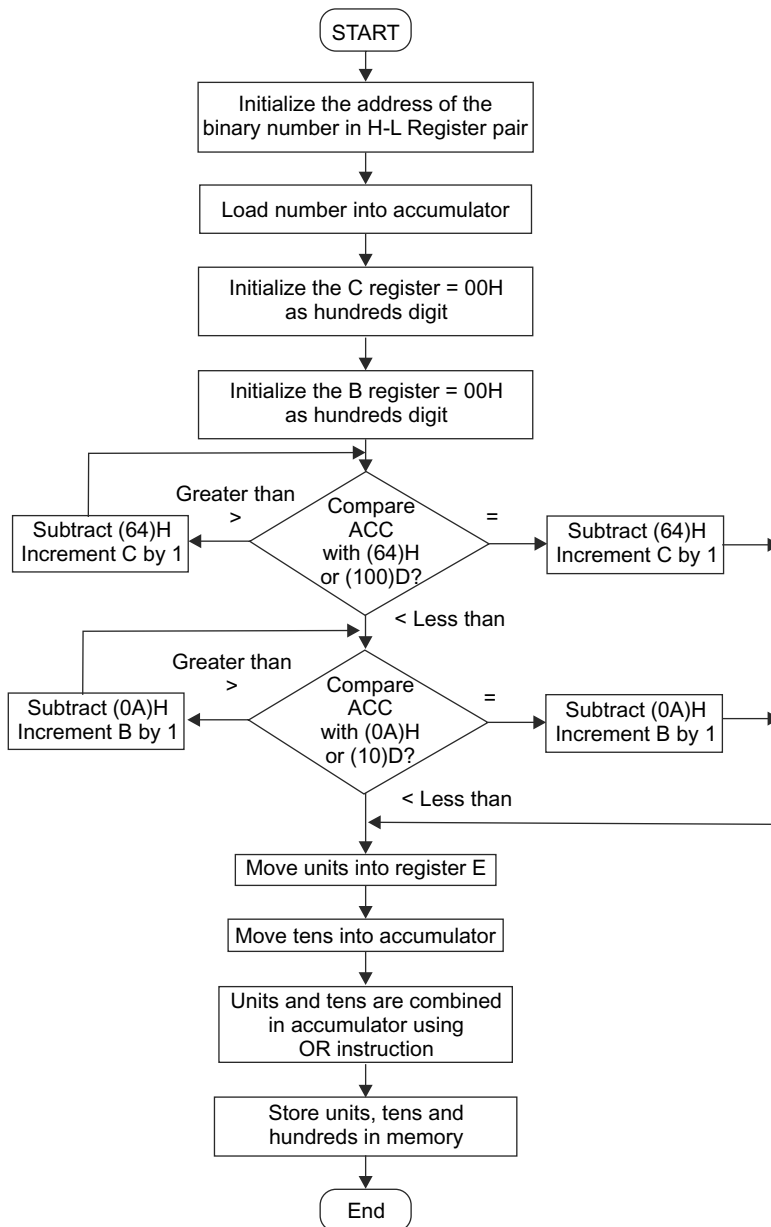


Fig. 4.38 Flow chart to convert binary number to decimal number

3. To separate the hundreds digit of the decimal number, subtract $(100)_D$ or $(64)_H$ from the number until the remainder becomes less than 100 and the Hundreds digit must be stored in C register.
4. To separate the tens digit of the decimal number, i.e., remainder after subtracting $(100)_D$ or $(64)_H$ repeatedly, subtract $(10)_D$ or $(0A)_H$ from the remainder, until the next remainder becomes less than $(10)_D$ and the tens digit must be stored in B register.

5. The next remainder is the units digit of the decimal number and stored in E register.
6. Load the content of register B in accumulator and rotate left four times.
7. Perform OR operation between the content of accumulator and register E and combined the tens digit and units digit and stored in 8001H.
8. Store the hundreds digit in 8002H.

PROGRAM 4.32

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8100	21, 00, 80		LXI	H,8000H	Load the address of a 8-bit binary number in HL register pair
8103	7E		MOV	A,M	Move data from memory to accumulator
8104	0E,00		MOV	C,00	Initialize hundreds digit in register C
8106	06,00		MOV	B,00	Initialize tens digit in register B
8108	FE,64	Start	CPI	64H	Compare accumulator with 64H or 100D
810A	CA, 16,81		JZ	Level-1	When the content of accumulator is (64)H or (100)D, jump to Level-1
810D	DA,1C,81		JC	Level-2	When the content of accumulator is less than (64)H or (100)D, jump to Level-2
8110	DE, 64		SBI	64H	Subtract (64)H or (100)D from accumulator
8112	0C		INR	C	Increment Hundreds digit count in register C
8113	C3,08, 81		JMP	Start	Jump to Start
8116	0C	Level-1	INR	C	Increment Hundreds digit count in register C
8117	DE, 64		SBI	64	Subtract 64H or 100D from accumulator
8119	C3,2D,81		JMP	Level-4	Jump to Level-4
811C	FE,0A	Level-2	CPI	0A	Compare accumulator with (0A)H or (10)D
811E	CA,2A,81		JZ	Level-3	When accumulator content is equal to (0A)H or (10)D, Jump to Level-3
8121	DA,00,81		JC	Level-4	When accumulator content is less than (0A)H or (10)D, jump to Level-4
8124	DE,0A		SBI	0AH	Subtract (0A)H or (10)D from the accumulator
8126	04		INR	B	Increment tens digit count in register B
8127	C3,1C,81		JMP	Level-2	Jump to Level-2
812A	04	Level-3	INR	B	Increment tens digit count in register B
812B	DE,0A		SBI	0A	Subtract (0A)H or (10)D from the accumulator
812D	5F	Level-4	MOV	E,A	Store units digit in register E
812E	78		MOV	A,B	Move tens digit count into Accumulator
812F	07		RAL		Rotate accumulator left four times, so that the tens digit is placed in tens digit position
8130	07		RAL		
8131	07		RAL		
8132	07		RAL		
8133	B3		ORA	E	The content of E is ORed with accumulator so that tens digit and units digit is available in accumulator

(Contd.)

(Contd.)

8134	23	INX	H	Increase HL pair by 1
8135	77	MOV	M,A	Store tens digit and units digit in memory
8136	79	MOV	A,C	Move hundreds digit into accumulator
8137	23	INX	H	Increase HL pair by 1
8138	77	MOV	M,A	Store result in memory
8139	76	HLT		Halt

Example 4.38

Address	Binary Number	Address	Result (Decimal)
8000 H	(FD) _H	8001 H	(53) _D
		8002 H	(02) _D

SUMMARY

- This chapter has given a brief introduction to machine-language, assembly language and high-level languages with their advantages and disadvantages.
- Time-delay loops, modular programming, and macro are also discussed. Time delay can be developed using registers and register pairs, and delay time can be determined by the number of T states in a delay loop, system frequency and the number of times the loop is repeated. The calculation of time delay has been incorporated in this chapter. Time-delay loops are very important in digital clocks, process control, traffic signals, etc.
- In this chapter, programming techniques such as looping, counting using data transfer instructions, arithmetic and logical instructions are explained. The application of CALL, RET, PUSH and POP instructions in some programs are discussed.
- Assembly-language programs on arithmetic addition, subtraction, multiplication, division, and conversion from BCD to binary are illustrated. Assembly-language programs to arrange an array in ascending order and descending order, find the maximum and minimum number of an array, and transfer a block of data from one section of memory to another section of memory are also incorporated.

MULTIPLE-CHOICE QUESTIONS

- 4.1 The assembler is
- (a) program that translates mnemonics into binary code
 - (b) program that translates mnemonics into octal code
 - (c) an operating system which manages all the programs in the system
 - (d) a compiler that translates statements from assembly language into machine language
- 4.2 The compiler is
- (a) faster than an interpreter
 - (b) slower than an interpreter
 - (c) an interpreter
 - (d) a single-step process
- 4.3 When a subroutine is called by CALL instruction, the microprocessor stores the 16-bit address of the instruction next to CALL on the

- (a) stack pointer (b) accumulator
(c) program counter (d) stack
- 4.4 When a CALL instruction is executed, the stack pointer register is
(a) decremented by two
(b) incremented by two
(c) decremented by one
(d) incremented by one
- 4.5 When the RET instruction is executed at the end of a subroutine,
(a) the memory address of the RET instruction is transferred to the program counter
(b) two data bytes stored in the top locations of the stack are transferred to the stack pointer
(c) the data where the stack is initialized is transferred to the stack pointer
(d) two data bytes stored in the top two locations of the stack are transferred to the program counter
- 4.6 Whenever the PUSH H instruction is executed,
(a) data bytes in the HL pair are stored on the stack
(b) two data bytes at the top of the stack are transferred to the HL register pair
(c) two data bytes at the top of the stack are transferred to the program counter
(d) two data bytes from the HL register that were previously stored on the stack are transferred back to the HL register
- 4.7 Whenever the POP H instruction is executed,
(a) two data bytes from the HL register which were previously stored on the stack, are transferred back to the HL register
(b) data bytes in the HL pair are stored on the stack
(c) two data bytes at the top of the stack are transferred to the program counter
(d) data bytes in the HL pair are stored on the program counter
- 4.8 What will be the content of the stack pointer after execution of the following instructions?
MVI C, 05

```
LXI H, 8000H; SPHL
(a) 8000H (b) 0005H
(b) 0080H (d) 0500H
LOOP MVI C, FFH
      DCR C
      JNZ LOOP
      HLT
```

- 4.9 In the above instructions, how many times is DCR C instruction executed?
(a) 256 (b) 255
(b) 155 (d) 156
- 4.10 How many times does the program jump to LOOP?
(a) 256 (b) 255
(b) 155 (d) 156
- 4.11 Match the following:
- | <i>Column I</i> | <i>Column II</i> |
|-----------------|--|
| (a) POP | (i) to save data in the stack |
| (b) PUSH | (ii) to read data from the stack |
| (c) Stack | (iii) a portion of memory reserved for return address and data |
| (d) Mask | (iv) a byte used with an ANI instruction to blank out certain bits |
- (A) a – i, b – ii, c – iii, d – iv
(B) a – iv, b – iii, c – ii, d – i
(C) a – ii, b – i, c – iii, d – iv
(D) a – iii, b – i, c – ii, d – iv
- 4.12 Opcode is
(a) the part of the instruction which tells the computer what operation to perform
(b) an auxiliary register that stores the data to be added or subtracted from the accumulator
(c) the register that receives the constructions from memory
(d) The data which will be used in data manipulation of instruction

Register A contains 5FH, B contains 4FH, C contains 26H, H contains 80H, L contains FFH and the memory locations 80EE and 80FF contains 2D and 4E respectively. The following program begins at the memory location 8110H.


```
ADD C
MOV B,M
MOV M, A
DAD B
```

- 4.13 What will A contain after the program?
 (a) 85 H (b) 5FH
 (c) FFH (d) 4AH
- 4.14 What will B contain after the program?
 (a) 4E H (b) AA H
 (c) DD H (d) 4A H
- 4.15 What will H contain after the program?
 (a) CF H (b) AB H
 (c) DA H (d) 4AH
- 4.16 What will L contain after the program?
 (a) 25 H (b) AA H
 (c) DD H (d) 4AH
- 4.17 What will be the content of the memory location 80EEH after the program?
 (a) 2D H (b) AA H
 (c) DD H (d) 4AH
- 4.18 What will be the content of the memory location 80FF after the program?
 (a) 85H (b) AD H
 (c) D0H (d) 4FH
- 4.19 The PC contains 8452H and SP contains 88D6H. What will be the content of PC and SP following a CALL to subroutine at location 82AFH?
 (a) 82AF, 88D4 (b) 82AF, 8450
 (c) 8450, 88D4 (d) 82AF, 8452
- 4.20 What will be the delay generated by the following instructions?

```
MVI C, FF 10T
LOOP DCR C 5T
JNZ LOOP 10/7 T
```

 (a) 3832 T states (b) 3850 T states
 (c) 3857 T states (d) 3835 T states

SHORT-ANSWER-TYPE QUESTIONS

- 4.1 Explain the following terms:
 (a) Assembly language (b) Machine language (c) High-level language (d) Compiler
- 4.2 What the disadvantages of machine languages? What the advantages of assembly languages?
- 4.3 Write the disadvantages of high-level languages.
- 4.4 What are translators? Write the differences between compilers and interpreters.
- 4.5 Define stack. Explain the function of PUSH and POP instructions.
- 4.6 List the high-level languages. What are the advantages and disadvantages of high-level languages?
- 4.7 Explain the features of the following languages:
 (a) FORTRAN (b) C (c) ADA
 (d) PASCAL (e) PROLOG (f) BASIC
- 4.8 Define macro and explain the operation of a macro with examples. What is the difference between macro and subroutine ?
- 4.9 What is modular programming? Write the advantages and disadvantages of modular programming.

REVIEW QUESTIONS

- 4.1 Explain time-delay loop using register and register pair. Write some applications of time-delay loop. Calculate the time required to execute the following two instructions if the system-clock frequency is 1 MHz.
- | | | |
|-------|----------|-------------|
| LOOP: | MOVA,B | 5 T-states |
| | JMP LOOP | 10 T-states |
- 4.2 Calculate the time delay to execute the following instructions:
- | | | |
|-------|-----------|-------------|
| LEVEL | MOV C,B | 5 T-states |
| | NOP | 5 T-states |
| | NOP | 5 T-states |
| | JMP LEVEL | 10 T-states |
- 4.3 Write an assembly-language program to load memory locations 8100H and 8200H with 58H and 42H. Add the contents and store it in the memory location 8300H.
- 4.4 Data byte 28H is stored in Register B and data byte 97H is stored in the accumulator. Show the contents of registers B, C, and the accumulator after the execution of the following instructions:
- (a) MOV C, A (b) MOV A, B (c) ADD B
- 4.5 Two numbers A and B are stored in successive memory locations 8500H and 8501H respectively. Write a program to determine and store the results of the following operations starting from 8520H.
- (a) A + B, (b) A–B, (c) A NOR B, (d) A NAND B,
 (e) A XOR B, (f) A AND B (g) A OR B
- 4.6 Write an assembly-language program for addition of 00H and 80H and save the result in the memory address 8000H.
- 4.7 Write an assembly-language program for addition of two 8-bit decimal numbers and save the result in the memory address 8000H.
- 4.8 Find the sum of 10 numbers stored in successive memory locations starting from 2000H and store the result in two bytes, 8100H and 8101H.
- 4.9 Data stored in locations 8000H and 8001H are 07 and 40H respectively. Assemble them to 47 and store in location 8002H.
- 4.10 A series of sixteen bytes of data are stored in memory locations from 9000 H to 900F H. Write an assembly-language program to transfer the entire block of data bytes to new memory locations starting from 9100H.
- 4.11 Write an assembly-language program to multiply two 8-bit numbers.
- 4.12 Assume that six data bytes are stored in the memory locations starting from 8100H. Write an assembly-language program to transfer the data to the locations 8200H to 8205H in the reverse order.
- 4.13 A string of ten data bytes is stored starting from the memory location 9000H. The string includes some bytes with FF value. Write a program to eliminate the FF from the string.
- 4.14 A string of ten data bytes is stored starting from the memory location 9000H. The string includes some bytes with zero values. Write a program to eliminate the blank from the string.
- 4.15 Write a program to subtract two bytes at a time and store the result in memory locations.
- 4.16 Calculate the 1's and 2's complement of the contents of two successive memory locations 8500H and 8501H and store them in four consecutive memory locations starting from 8600H.

- 4.17 N numbers are stored in consecutive memory locations starting from 8001H and the value of N is available in memory location 8000H. Find the maximum of N numbers and store in 9001.
- 4.18 N numbers are stored in consecutive memory locations starting from 8101H and the value of N is available in memory location 8100H. Find the minimum of N numbers and store in 9001.
- 4.19 N numbers are stored in consecutive memory locations starting from 8001 and the value of N is available in memory location 8000H. Sort the numbers in ascending order and store in the memory location starting from 9001.
- 4.20 N numbers are stored in consecutive memory locations starting from 8001H and the value of N is available in memory location 8000H. Sort the numbers in descending order and store in the memory location starting from 9001.
- 4.21 Subtract the content of the memory location 9501H from the content of the memory location 9500H and place the result in the memory location 9502H. The contents are in signed magnitude format.
- 4.22 Find the 1's and 2's complement of a 16-bit number stored in two consecutive memory locations and store the desired result in the next two consecutive memory locations.
- 4.23 Calculate the square of the contents of the memory location 9100H using a look-up table and place the result in the memory location 9501.
- 4.24 Write an assembly-language program to divide two 8-bit numbers.
- 4.25 A block of 32 bytes of data is stored at the memory location starting from 8000H. Move this block to the memory location starting from 9000H.
- 4.26 Write an assembly-language program to detect a even and odd numbers.
- 4.27 Write an assembly-language program for addition of two 16-bit numbers whose sum is more than 16 bits.
- 4.28 Write an assembly-language program for decimal addition of two 8-bit numbers whose sum is 8 bits.
- 4.29 Write an assembly-language program for 16-bit decimal subtraction.
- 4.30 Write an assembly-language program for one's complement of a 16-bit number.
- 4.31 Write an assembly-language program for two's complement of an 16-bit number.
- 4.32 Write an assembly-language program for shifting an 8-bit number left by two bit.
- 4.33 Write an assembly-language program for shifting an 16-bit number left by two bit.
- 4.34 Write an assembly-language program to arrange a series of numbers in descending order using a subroutine.
- 4.35 Write an assembly-language program to find the square of a number using look-up table.
- 4.36 Write an assembly-language program for converting temperature from F to C degree.
- 4.37 Write an assembly-language program for converting hexadecimal number into its ASCII number.
- 4.38 Write an assembly-language program for signed arithmetic operation.
- 4.39 Write an assembly-language program to display digits 123456789ABCDE or 1 to F on the screen.
- 4.40 Write an assembly-language program for rolling display.

Answers to Multiple-Choice Questions

- 4.1 (d) 4.2 (a) 4.3 (d) 4.4 (a) 4.5 (d) 4.6 (a) 4.7 (a) 4.8 (a) 4.9 (b)
- 4.10 (b) 4.11 (c) 4.12 (a) 4.13 (a) 4.14 (a) 4.15 (a) 4.16 (a) 4.17 (a) 4.18 (b)
- 4.19 (a) 4.20 (c)

Chapter 5

Architecture of 8086 and 8088 Microprocessors

5.1 INTRODUCTION

The Intel 8086 is a high-performance 16-bit, *N*-channel, HMOS microprocessor which is available in three clock rates: 5, 8, and 10 MHz. The term *HMOS* stands for 'High-Speed MOS'. The 8086 is Intel's first 16-bit microprocessor. This processor was introduced in 1978, due to the demand for more powerful and high-speed computers. This processor has a more powerful instruction set and more programming flexibility, and its speed is more than the 8085 microprocessor. The CPU of the 8086 processor is implemented in *N*-channel, depletion load, and silicon-gate technology. This processor has the following features:

- ◆ The CPU has a direct addressing capability of 1 MB memory.
- ◆ Bit, byte, word and block operations are available.
- ◆ 8-bit and 16-bit signed and unsigned arithmetic in binary and decimal operations are performed.
- ◆ It is available in 40-pin lead CERDIP and plastic DIP package (Dual In-Line Package).
- ◆ It has architectures designed for assembly language as well as high-level language.

The 8086 is manufactured for the standard temperature range (32°F to 180°F) and extended temperature range (40°F to + 225°F). It contains an electronic circuitry of 29000 transistors. The 8086 has 20 address lines and 16 data lines. This CPU can directly address up to $2^{20} = 1$ Mbytes of memory. The 16-bit data word can be divided into a low-order byte and a high-order byte. The 20-bit address lines are time multiplexed to select lines of low-order byte and high-order byte data separately. The 8088 is an 8-bit processor designed around the 8086 architecture. The internal functions of 8088 are same as the 8086 processor functions. The 8088 processor has a 20-bit address bus and an 8-bit data bus. The comparison between 8085, 8086 and 8088 microprocessors are illustrated in Table 5.1 and Table 5.2. In this chapter, the architectures of 8086 and 8088 are discussed in detail.

Table 5.1 Comparison between 8085 and 8086 microprocessors

8085 Microprocessor	8086 Microprocessor
8085 is an 8-bit processor created in 1977 and it has an 8-bit data bus.	8086 is a 16-bit processor developed in 1978 and it has a 16-bit data bus.
8085 is manufactured using NMOS technology and this processor IC consists of about 6200 transistors.	8086 is fabricated on HMOS technology and the processor IC consists of approximately 29000 transistors.
8085 has a 16-bit address bus and is able to access $2^{16} = 64$ KB memory locations.	8086 has a 20-bit address bus and is able to access $2^{20} = 1$ MB memory locations.
Number of flags are 5.	Number of flags are 9.
Pipelining concept is not used in 8085.	8086 uses pipelining.
Instruction queue does not exist in 8085 and it sequentially executes instructions.	8086 has a 6-byte instruction queue in BIU (Bus Interface Unit).
No segment registers exist in 8085.	There are four segment registers, CS, DE, ES, SS, in 8086.
Only four types of addressing modes are available.	Eight types of addressing modes are available.
8085 has less instructions than 8086. Direct multiplication, divide, string byte block movement and loop instructions are not available in 8085.	8086 has more instructions than 8085. Direct multiplication, divide, string byte block movement and loop instructions are available in 8085.

Table 5.2 Comparison between 8086 and 8088

8086 Microprocessor	8088 Microprocessor
8086 is a 16-bit processor developed in 1978 and it has a 16-bit data bus.	8088 is an 8-bit processor developed in 1979 and it has an 8-bit data bus.
8086 has a 6-byte instruction queue in BIU.	8086 has a 4-byte instruction queue in BIU.
The 8086 BIU fills the queue when its queue is having an empty space of 2 bytes.	The 8088 BIU fetches a new instruction byte to load into the queue whenever there is one byte hole in the queue.
As 8086 has a 16-bit data bus, and 8-bit or 16-bit memory read/write operation is possible in a single operation.	As 8088 has an 8-bit data bus, it can read 8 bits of data from memory or I/O devices and write 8-bit data to memory or I/O devices. To read 16-bit data, the 8088 requires two memory read operations.
$AD_{15} - AD_8$ pins are used as time multiplexed address/data bus in 8086.	$AD_7 - AD_0$ pins are used as time multiplexed address/data bus and $A_{15} - A_8$ pins are used as address bus only in 8088.
\overline{BHE} is present in 8086 and the external memory interfaces have even or odd address banks.	\overline{BHE} is not present in 8088. Therefore, the external memory interfaces will not have even or odd address banks. The external memory will therefore be byte oriented as 8085.
In 8086, I/O and memory pin is represented as $\overline{IO/M}$	In 8088, I/O and memory pin has been inverted and represented as $\overline{IO/\overline{M}}$.
The status signals of 8086 are $\overline{S_2}$, $\overline{S_1}$ and $\overline{S_0}$	The status signals of 8086 are $\overline{IO/\overline{M}}$, $\overline{DT/\overline{R}}$ and $\overline{SS_0}$.
The overall execution time of the instructions in 8086 is less compared to 8088 as 8086 has a 16-bit data bus and only 4 clock cycles are required to execute.	The overall execution time of the instructions in 8088 is more due to the 8-bit data bus, and the 16-bit operations require additional 4 clock cycles.

5.2 ARCHITECTURE OF 8086

The 8086 architecture has been implemented using two-stage pipelining in instruction execution. The processor logic unit has been divided into Bus Interface Unit (BIU) and Execution Unit (EU). These units are always operating asynchronously. The Bus Interface Unit (BIU) provides interface with external memory and I/O device addresses and data bus, and executes all bus operations. The BIU has a 6-byte instruction queue. On the other hand, the Execution Unit takes the instruction from the 6-byte instruction queue of BIU and executes it. Thus, the instruction fetch time has been drastically reduced.

The 8086 is a 16-bit microprocessor and it has a 20-bit address bus and a 16-bit data bus. Therefore, this processor can directly access $2^{20} = 1,048,567$ (1 MB) memory locations. It can read/write 8-bit data or 16-bit data from/to memory or Input/Output (I/O) devices. The 8086 has time-multiplexed address and data buses. Hence, the number of pins can be reduced, but it slows down the data-transfer rate. The block diagram of the internal architecture of the 8086 processor is shown in Fig. 5.1. It is divided into two separate functional units

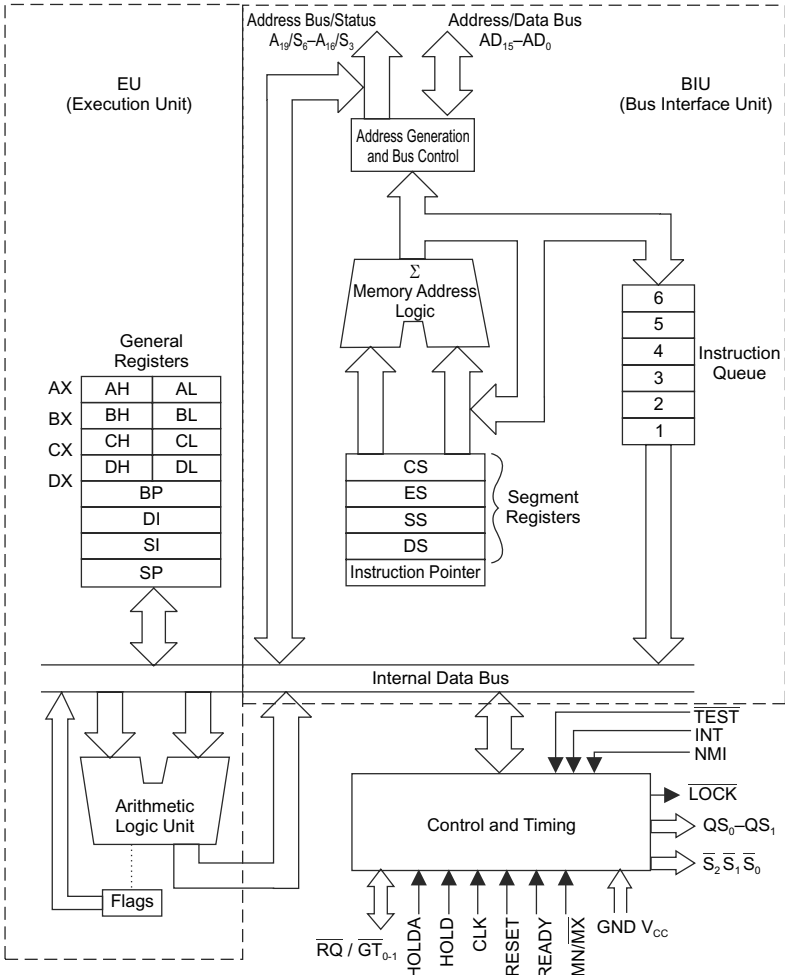


Fig. 5.1 Block diagram of 8086 microprocessor

such as Bus Interface Unit (BIU) and Execution Unit (EU). These two separate units are worked simultaneously for instruction execution based on two-stage instruction pipeline principles.

5.2.1 Bus Interface Unit (BIU)

The Bus Interface Unit (BIU) consists of bus interface logic, general-purpose registers, segment registers, stack pointer, base pointer and index registers, memory addressing logic and a 6-byte instruction queue. The BIU carries out all bus operations for the Execution Unit, and it is responsible for executing all external bus cycles.

The BIU performs data and addresses transfer between the processor and memory or I/O devices. This section computes and sends addresses, fetches instruction codes, stores fetched instruction codes in a First-In-First-Out (FIFO) register which is called a *queue*. The BIU is also used to read data from memory and I/O devices, and write data to memory and I/O devices. While the EU is busy in instruction execution, the BIU continues to fetch instructions from memory and stores them in the instruction queue.

This unit relocates addresses of operands while it gets unrelocated operand addresses from EU. The execution unit tells BIU from where to fetch instructions as well as from where to read data. When the EU executes an instruction the BIU resets the queue, fetches the next instruction from the new memory location, and passes the instructions to the EU. In this way, the 8086 BIU fills the queue when the queue becomes empty spaces of two bytes. This process is known as *pipeline flush*.

5.2.2 Execution Unit (EU)

The Execution Unit (EU) consists of Arithmetic Logic Unit (ALU), general-purpose registers, flag register (FLAGS), instruction decoder, pointers and index registers, and the control unit which are required to execute an instruction.

The EU gets the opcode of an instruction from the instruction queue. Then the EU decodes and executes it. The BIU and EU operate independently. When the EU is decoding or executing an instruction, the BIU fetches instruction codes from the memory and stores them in the queue. This type of overlapped operation of the BIU and EU functional units of a microprocessor is called *pipelining*. This process becomes faster except for *Jump* and *Call* instructions as the queue must be dumped and then reloaded from a new address. Hence, the function of the EU is to execute all instructions, provide address to the BIU for fetching opcodes and operands and perform ALU operations after using various registers as well as the flag register.

5.2.3 Fetch and Execute

During fetch and execute of instructions in the 8085 microprocessor, the nonpipeline concept follows so that instructions are fetched and execute sequentially as shown in Fig. 5.2(a). In the 8086 processor, the BIU and EU perform the fetch and execute operations with overlap. The fetch and execute operations of 8086 are given below:

- ◆ The BIU output is the content of the Instruction Pointer (IP), register which is put on the address bus. Therefore, a byte or word can be read from a specified address into the BIU.
- ◆ The content of the instruction pointer register is incremented by 1 to make it ready for the next instruction fetch.
- ◆ After receiving the opcode and operand of the instruction, the instruction code must be passed to the queue which is a FIFO (first-in first-out) register.
- ◆ Initially, the queue is empty. As soon as the BIU puts the instruction on the queue, the EU draws the instruction from the queue and starts execution.

- ◆ While the EU is executing one instruction, the BIU will continue to fetch new instructions. Depending upon the execution time of the instruction, the BIU can fill the queue with instructions. When execution time is more, the queue will be filled completely before the EU is ready to get the next instruction for execution. Figure 5.2(b) shows the pipeline concept of fetch and execution in BIU and EU. In this architecture, BIU and EU are operating independently. The advantage of this architecture is that the EU executes instructions continuously without waiting for fetching of the instruction in BIU.

But sometimes the EU can enter wait mode. There are three different conditions when the EU operates in wait mode. The first condition is when an instruction wants to access a memory location which is not in the queue. Then the BIU suspends fetching instructions and outputs the address of the memory location. After waiting for memory access, the BIU can start again filling the queue and the EU also starts to execute instruction codes from the instruction queue.

While executing a JUMP instruction, the control is to be transferred to a new address which is nonsequential. But it is known to us that instructions for a queue are executed sequentially. Due to the nonsequential new address of the JUMP instruction, the existing instruction codes in the queue will not be executed and the EU must wait while the instruction at the jump address is fetched. During this operation, the existing bytes in the queue will be discarded.

The last condition for wait-mode operation is possible when the BIU suspends the instruction-fetching operation. This is feasible when the EU operates slowly to execute an instruction. In case of AAM, ASCII adjusts for multiplication instruction require about 83 clock pulses to execute completely. Generally, four clock cycles are required per instruction fetch. Consequently, the queue will be completely filled during the execution of an AAM instruction. Then BIU must wait until the execution of slow instruction has been completed or the EU pulls one or two bytes from the queue.

Sometimes, an instruction requires to read data from a memory location which is not in the queue. Then BIU should suspend instruction fetching and wait for output from the address of the memory location. After waiting for reading data from memory, the EU can again start executing instruction codes from the queue.

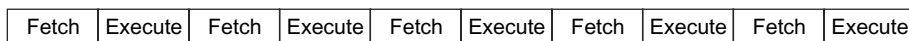


Fig. 5.2(a) General instruction fetch and execution for conventional processors (8085)

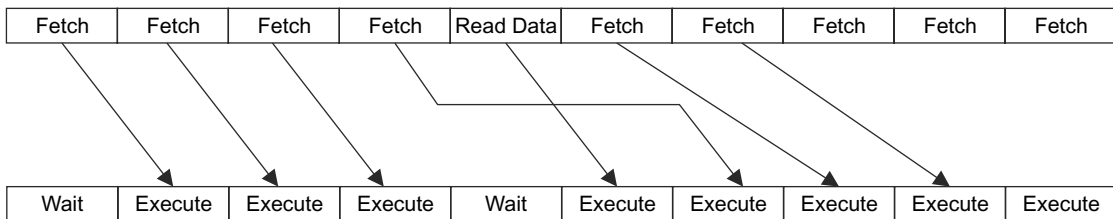


Fig. 5.2(b) Instruction fetch and execution of the 8086 processor

5.2.4 Process of Fetching and Decoding of Instructions

Initially, CS:IP must be loaded with the addresses from which the program will be executed. At first, the queue is empty and the microprocessor starts a fetch operation and the first byte, i.e., opcode of the instruction is loaded into the queue. Subsequently, data will also be fetched in the queue. When the first byte of the queue, i.e., opcode of instruction, is loaded into decoder for decoding, one byte becomes empty from the queue. Therefore, the queue must be updated with the next instruction opcode and data.

After decoding the opcode, the decoder takes a decision whether the instruction is of one byte or multi-byte. When the instruction is of one byte, the opcode can perform operations during executing the instruction. If the instruction is multi-byte, the first byte is opcode and other remaining bytes are data, or first two bytes are opcodes and other bytes are data. Therefore, the microprocessor should read the operand from queue and decode it. Subsequently, it executes the instruction by accepting data from the queue. After completion of fetching and decoding of an instruction, the next instruction will be fetched and decoded and executed. Figure 5.3 shows the queue operation in an 8086 microprocessor.

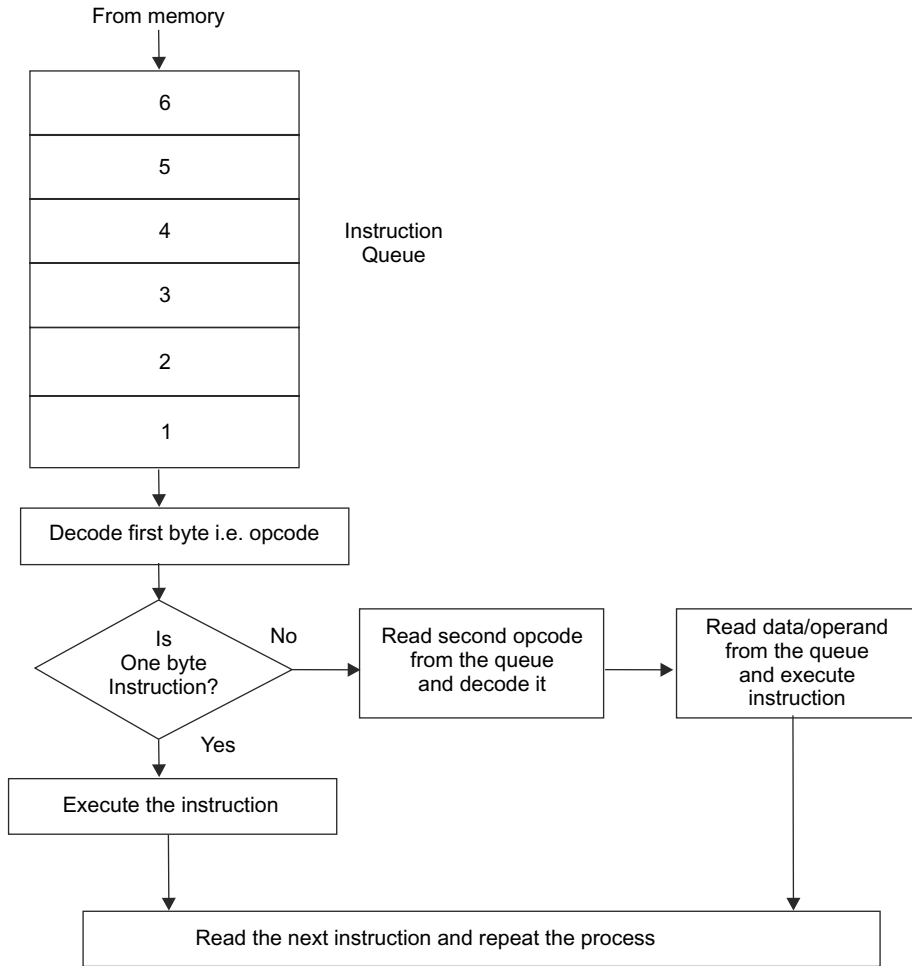


Fig. 5.3 Queue operation of 8086 microprocessor

5.3 REGISTERS

The 8086 CPU has fourteen 16-bit registers as depicted in Fig. 5.4. All these registers are subdivided into different groups, namely, Data Register Group (four registers), Segment Register Group (four registers), Pointer and Index Register Group (four registers), Instruction Register (Program Counter) and Flag Register. In this section all registers are discussed.

5.3.1 Data Registers

The 8086 has four 16-bit general-purpose registers (AX, BX, CX and DX). These registers can be used in arithmetic, logical operations and temporary storage. Each of these 16-bit registers is further subdivided into two 8-bit registers (upper and lower bytes) as shown in Table 5.3.

Table 5.3 General-purpose data registers

16-Bit Registers	8-Bit High-order Registers	8-Bit Low-order Registers
AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

The functions of each data register are discussed as follows:

AX Register The AX register serves as an accumulator. It performs input/output operations and processes data through AX or AH or AL. During execution of a 16-bit multiply and divide instruction, AX contains the one-word operand and the result is stored in the accumulator. In 32-bit multiply and divide instructions, AX is used to hold the lower-order word operand. Instructions involving AX or AH or AL can load data immediately and hence data usually require less program memory.

BX Register BX can be used as an index register for MOVE operation and base register while computing the data memory address.

CX Register CX register can be used as a count register for string operations and holds a count value during large number iterations. In LOOP instructions, CX holds the desired number of repetitions and is automatically decremented by one, after each iteration. While CX becomes zero, the execution of instructions should be terminated. In the same way, the 8-bit CL register is used as a count register in bit-shifting and rotate instructions.

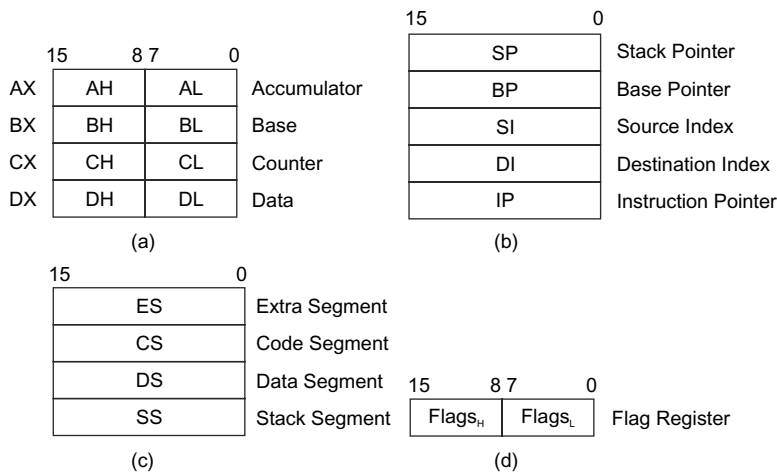


Fig. 5.4 Registers of 8086: (a) Data registers (b) Pointer and index registers (c) Segment registers (d) Flag registers

DX Register DX can be used as a port address for IN and OUT instructions. The DX may be used in I/O instructions, multiply and divide instructions. In 32-bit multiply and divide instructions, DX is used to hold the high-order word operand.

5.3.2 Segment Registers

The concept of memory segmentation was introduced in the 8086 processor. In memory segmentation, the complete 1 MB memory can be divided into 16 parts which are called segments. Each segment thus contains 64 KB of memory. In 8086, there are four segment registers such as Code Segment (CS) Register, Data Segment (DS) Register, Stack Segment (SS) Register and Extra Segment (ES) Register. The 8086 microprocessor-based system memory is divided into four different segments, namely, Code Segment (CS), Data Segment (DS), Stack Segment (SS) and Extra Segment (ES). Each segment has a memory space of 64 KB, as depicted in Fig. 5.5, and each segment can be addressed by 16-bit segment registers.

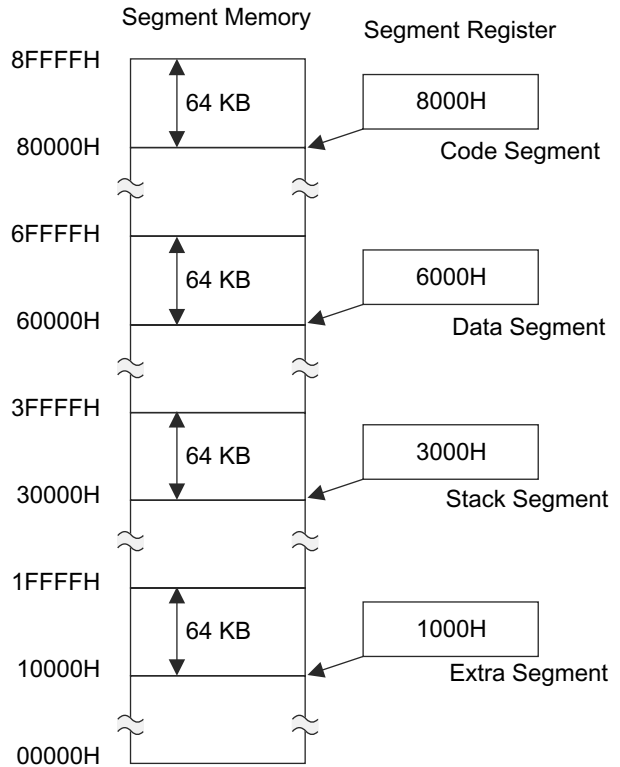


Fig. 5.5 Segment registers and segment memory

Code Segment (CS) The code segment register is used for addressing a memory location in the code segment of the memory in which the program is stored for execution.

Data Segment (DS) The data segment register points to the data segment of the memory, where data is stored.

Extra Segment (ES) The extra segment is a segment which can be used as another data segment of the memory. Therefore, extra segment contains data.

Stack Segment (SS) The stack segment register is used for addressing stack segment of memory in which stack data is stored. The CPU uses the stack for temporarily storing data, i.e., the content of all general purpose registers which will be used later.

5.3.3 Pointer and Index Registers

The pointer and index registers of 8086 are as follows:

- ◆ Stack Pointer (SP)
- ◆ Base Pointer (BP)

- ◆ Source Index (SI)
- ◆ Destination Index (DI)
- ◆ Instruction Pointer (IP)

Stack Pointer (SP) The stack pointer is used to locate the stack-top address. It contains an offset address. In PUSH, POP, CALL and RET instructions, the stack address is determined after adding the contents of the stack segment register, after 4-bit left-shift and the contents of SP.

Base Pointer (BP) The base pointer register can provide indirect access to data in a stack. The BP may also be used for general-purpose data storage.

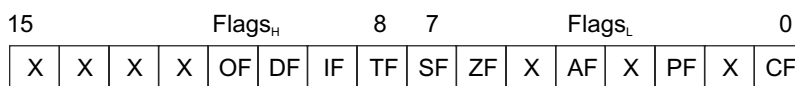
Source Index (SI) and Destination Index (DI) These registers are used in memory or stack-address computation for general data storage. The main purpose of these registers is to store offset or displacement. In memory address computation, the content of data segment and index registers depending upon addressing modes.

Sometimes SI is used as source index and DI as destination index. If the content of SI is added with the content of DS to determine the physical address, it will be used as source address of data. While the content of DI is added with the content of ES to find the destination address of the data, these registers can also be used as general purpose registers.

Instruction Pointer (IP) Generally, the instruction pointer register is used as a program counter. This is used for the calculation of memory addresses of instructions which will be executed. This register stores the offset for the instruction. The content of IP is automatically incremented while the execution of an instruction is going on. The address of the next instruction is computed after adding IP contents to the code segment register contents after 4-bit left-shift.

5.3.4 Flag Register

The 8086 has a 16-bit flag register. This register is also called Program Status Word (PSW). It has nine flags out of which six are status flags and three are control flags. The status flags are Carry flag (CF), Parity flag (PF), Auxiliary Carry flag (AF), Zero flag (ZF), Sign flag (SF) and Overflow flag (OF). These status Flags are affected after the execution of arithmetic or logic instructions. The control flags are Trap flag (TF), Interrupt Flag (IF) and Direction Flag (DF). Figure 5.6 shows the 16-bit flag register of the 8086 processor.



CF – Carry flag	SF – Sign flag
PF – Parity flag	TF – Trap flag
AF – Auxiliary carry flag	IF – Interrupt flag
ZF – Zero flag	DF – Directional flag
X – Underfined	OF – Overflow flag

Fig. 5.6 Flag registers of 8086

Carry Flag (CF) The carry flag is set to 1 if after arithmetic operation a carry is generated or a borrow is generated in subtraction. When there is no carry out, the carry flag is reset or zero. This flag can also be used in some shift and rotate instructions.

Parity Flag (PF) If the result of 8-bit operation or lower byte of the word operation contains an even number of 1s, parity flag is set.

Auxiliary Carry Flag (AF) This flag is set to 1 if there is a carry out of the lower nibble to the higher nibble of an 8-bit operation. It is used for BCD operations.

Zero Flag (ZF) The zero flag is set to 1 if the result of any arithmetic or logical operation is zero. While the result is zero, it is reset.

Sign Flag (SF) The sign flag is set to 1, if the MSB of the result is 1 after the arithmetic or logic operations. This flag represents a sign number. Logic 0 indicates a positive number and logic 1 is used to represent negative number.

Overflow Flag (OF) This flag is set to 1 if the signed result cannot be expressed within the number of bits in the destination operand. This flag is used to detect magnitude overflow in signed arithmetic operations. During addition operation, the flag is set when there is a carry into the MSB and the flag is reset if there is no carry out of the MSB. For subtraction operation, the flag is set when the MSB desires a borrow, and the flag is reset if there is no borrow from MSB.

Direction Flag (DF) The direction flag is used in string operations. When it is set to 1, string bytes can be accessed from a memory address in decrement order, i.e., high memory address to low memory address. If it is zero, string bytes can be accessed from memory address in increasing order, i.e., low memory address to high memory address. For example, in MOVS instruction if DF is set to 1, the contents of the index registers SI and DI are automatically decremented to access the string bytes. If DF = 0, index registers SI and DI are automatically incremented to access the string bytes.

Interrupt Enable Flag (IF) This flag can be used as an interrupt enable or disable flag. When this flag is set, the maskable interrupt is enabled and 8086 recognizes the external interrupt requests, and the CPU transfer control to an interrupt vector specified location. When IF is 0, all maskable interrupts are disabled and there will be no effect on nonmaskable interrupts as well as internally generated interrupts. If 8086 is reset, IF is automatically cleared.

Trap Flag (TF) TF is a single-step flag. When TF is set to 1, a single step interrupt occurs after the next instruction executes and the program can be executed in single-step mode. The TF will be cleared by the single-step interrupt.

5.4 LOGICAL AND PHYSICAL ADDRESS

The 8086 sends a 20-bits address on the address bus to detect a memory location for memory read or write operations. Addresses within the segment can be varied from 0000H to FFFFH (64 KB). To detect a memory

location, the segment register supplies the higher-order 16 bits of the 20-bit memory address. The lower-order 16 bits of the 20-bit memory address are stored in any of the pointers and index registers or BX register. Therefore, memory addresses of the 8086 are computed by summing the contents of the segment register which is shifted left by 4 bits and the content of offset address. The 20-bit address sent by the 8086 processor is called the physical address as depicted in Fig. 5.7.

The physical address is calculated from the segment address and offset address. The segment register contains the higher-order 16 bits of the starting address of a memory segment. The CPU shifts the content of the segment register left by four bits or inserts four zeros for the lowest four bits of the 20-bit memory address. For example, if the content of the code segment register is 4000H, the starting address of the code segment will be 40000H. Hence, the 64 KB memory segment may be anywhere within the complete 1 MB memory based on the content of the code segment register and the starting address should be divisible by 16.

The offset address is used to determine the memory location distance from the starting address within the memory segment. An offset can be determined depending upon the addressing modes. The offset address will be different in different addressing modes. To locate a memory location within a memory segment, the 8086 processor generates a 20-bit physical address.

To determine the 20-bit physical address with a segment register and offset, the content of the segment register is left shifted by 4 bits and then an offset is added to it. For example, if the content of CS is 4000H and an offset is 2000H, the computation of 20-bit physical address is 42000H. Then 42000H represents the starting address of the segment in memory. Figure 5.6 shows the computation of physical address.

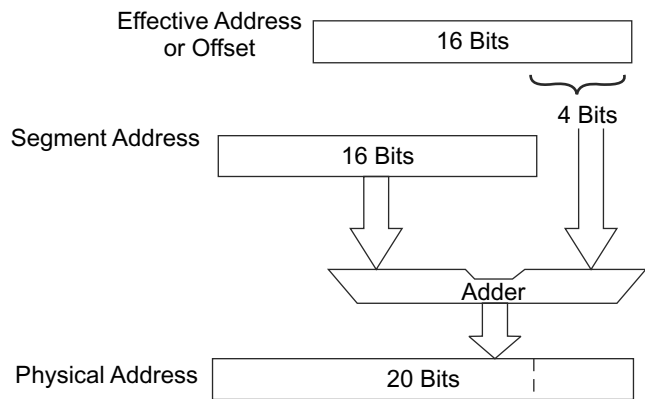


Fig. 5.7 Construction of physical address

Example 5.1

Determine the physical address when CS = 5300H and IP = 0200H. Write the starting and ending address of the code segment.

Solution

The content of the code segment is left shifted by 4 bits and the base address becomes 53000H. To determine the physical address, the content of IP will be added with base address. Hence physical address = 53000 + 0200 = 53200H.

The starting of code segment memory = 53200H.

As each segment memory consists of 64K memory locations, the end address will be computed after addition of 64K with the starting of code segment memory.

The ending address of code segment = 53200 + FFFF = 631FFH.

Example 5.2

Determine the physical address when ES is 6500H and offset address is 4767H

Solution

The content of the segment register ES is 6500H. When it is left shifted by 4 bits or multiplied by $(16)_D$ or $(10)_H$, the base address is equal to $6500H \times (10)_H = 65000H$.

$$\begin{aligned} \text{Physical address} &= \text{Content of segment register} \times (10)_H + \text{Offset address} \\ &= 6500H \times (10)_H + 4567H = 65000H + 4567H = 69567H \end{aligned}$$

Example 5.3

What is the content of data segment DS to locate the physical address 43657H? Assume the content of IP = 2057H.

Solution

The physical address is 43657H when the content of IP is 2057H

$$\text{Physical address} = \text{Content of data segment register} \times (10)_H + \text{IP address}$$

Therefore,

$$43657H = \text{Content of data segment register} \times (10)_H + 2057H$$

$$\text{Then the content of data segment register} \times (10)_H = 43657H - 2057H = 41600H$$

The content of data segment register (DS) is 4160H.

5.5 ADDRESS BUS, DATA BUS, CONTROL BUS

The 8086 processor is connected with memory and I/O devices through a set of parallel lines called *buses*. There are three different buses such as address bus, data bus and control bus which are explained below:

Address Bus

The 8086 CPU uses the address bus to select the desired memory or I/O device by generating a unique address which corresponds to the memory location or the location of I/O device of the system. The address bus is unidirectional and this processor has 20-bit address lines.

Data Bus

To transfer data between the CPU and memory and the CPU and I/O devices, a data bus is used. The data which is in the data bus can be used as instruction for the CPU, or the CPU sends data to an I/O device or the CPU receive data from I/O device. Therefore, a data bus is bidirectional.

Control Bus

The control bus of 8086 carries control signals which are used to specify the memory and I/O devices. The control signals of 8086 CPU are M/\overline{IO} , \overline{INTA} , ALE, and \overline{DEN} , etc.

5.6 MEMORY SEGMENTATION

Figure 5.8(a) shows the segment memory of an 8086 microprocessor. The address of 1st segment memory is 00000H to 0FFFFH. For these addresses, the content of segment register is 0000H and the offset address, i.e., the content of Instruction Pointer (IP) varies from 0000H to FFFFH. The physical address of memory is computed from

$$\text{The content of segment register} \times (10)_{16} + \text{offset address.}$$

Similarly, the second segment memory address is 10000H to 1FFFFH. In this case, the content of segment register is 1000H and the offset address is any value from 0000H to FFFFH. In the same way, other segment

memory addresses are $-20000H$ to $2FFFFH$, $30000H$ to $3FFFFH$, $40000H$ to $4FFFFH$, $50000H$ to $5FFFFH$, $E0000H$ to $EFFFFH$, and $F0000H$ to $FFFFFH$. In this case, the segments are non-overlapping and this type of memory segmentation is called non-overlapping memory segments.

Sometimes the segments are overlapping. For example, if a segment starts at a particular address, it will continue up to 64 K bytes. When another segment memory starts before 64K memory locations of the first segment, the two segments should be overlapped.

Assume the content of CS segment register $CS = 2500H$ and the content of DS segment register $DS = 2540H$ and the offset address varies from $0000H$ to $FFFFH$. The code segment register (CS) and offset address can represent the memory segment address starting from $25000H$ to $25000H + FFFFFH = 34FFFH$.

Similarly, the data segment DS and content of IP can address the memory segment address starting from $25400H$ to $25400H + FFFFFH = 353FFH$ as depicted in Fig. 5.8(b). It is clear from Fig. 5.8(b) that some portion of the segments are overlapping and this overlapping portion of the segment memory is called an *overlapped segment area*. The address locations of overlapping may be located from different sets of segments

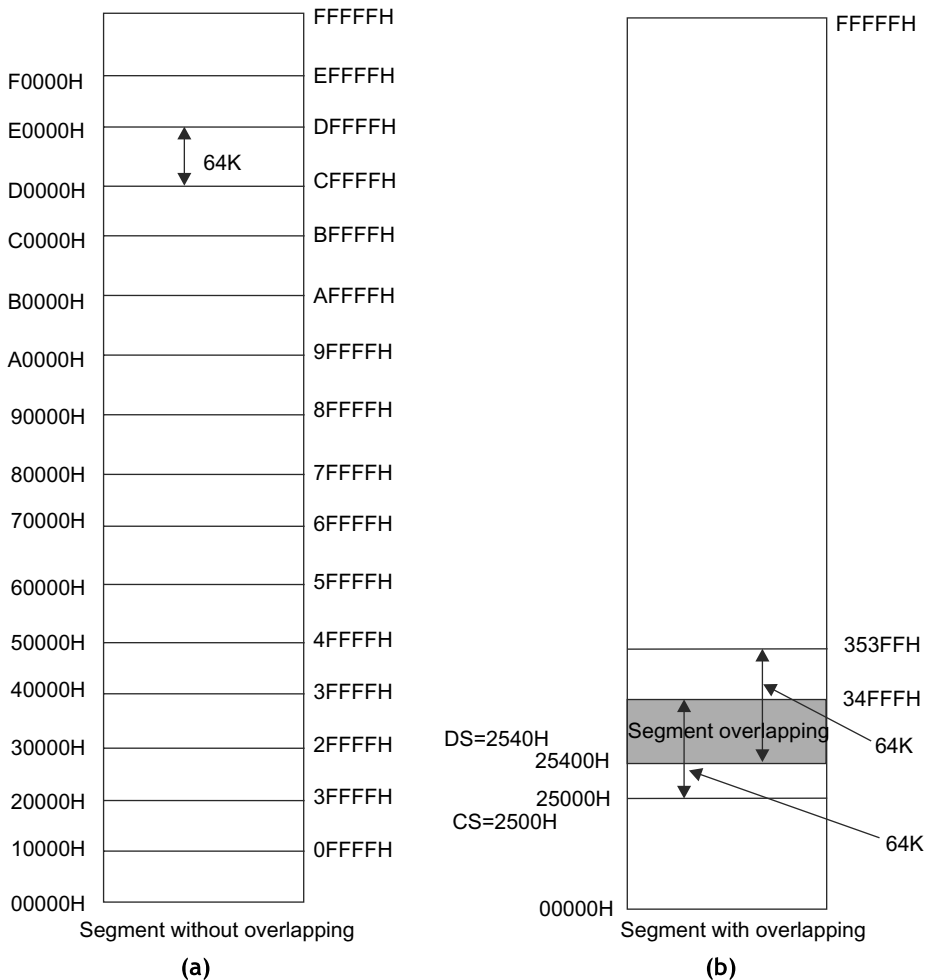


Fig. 5.8 (a) Segment memory without overlapping (b) Segment memory without overlapping

and offset addresses. Therefore, different sets of segments and offset address can locate the physical memory locations with non-overlapping segments and overlapping segments. The *advantages of segment memory* are given below:

- (i) Allow the memory capacity to be 1 MB even though the addresses associated with the individual instructions are 16 bits wide.
- (ii) Allow the use of separate memory areas for the program code and data and stack portion of the program
- (iii) Permit a program and/or its data to be placed into different areas of memory whenever the program is executed.
- (iv) Multitasking becomes easy
- (v) The advantage of having separate code and data segments is that one program can work on different sets of data. This is possible by reloading the data segment register (DS) to the point to the new data.
- (vi) The advantage of segment memory is that the reference logical addressed can be loaded into the instruction pointer (IP) and run the program anywhere in the segment memory as the logical address varies from 0000H to FFFFH.
- (vii) Programs are re-locatable so that programs can be run at any location in the memory.

5.7 8086 MEMORY ADDRESSING

The 8086/8088 processor has 20 bit address lines and it can allow 2^{20} or 1048567 (1MB) memory locations. Hence, the 8086 memory address space can be viewed as a sequence of 1MB as depicted in Fig. 5.9(a). Each memory location contains 8-bit data or one byte data and any two consecutive memory locations contain 16-bit data or a word. 524, 287 words are visualised in Fig. 5.9(b).

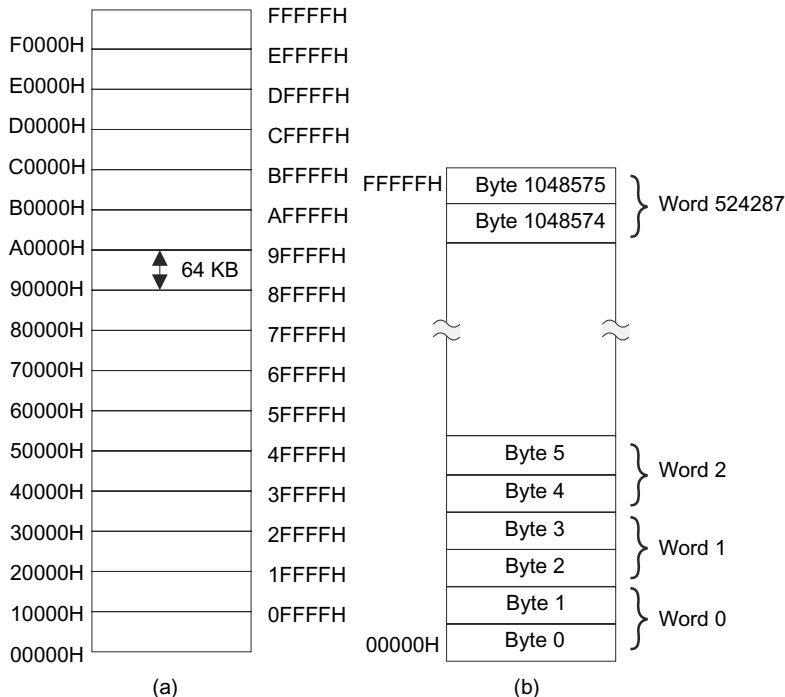


Fig. 5.9 (a) Memory map with 16 segments (b) Memory map with 524287 words

Physically, the memory can be organised as two banks such as even and odd bank and each bank consists of 512 KB memory size. The data lines D_7-D_0 are used for data transfer from even banks and $D_{15}-D_8$ are used for the odd banks. The even bank is selected by $A_0 = 0$ and $\overline{BHE} = 1$ and data bus D_7-D_0 is connected with this bank. When $A_0 = 1$ and $\overline{BHE} = 0$, the odd bank is selected and data bus $D_{15}-D_8$ is connected. The address space is physically connected to a 16-bit data bus by dividing the address space into two 8-bit banks, namely, odd-addressed bank and even-addressed bank as depicted in Fig. 5.10.

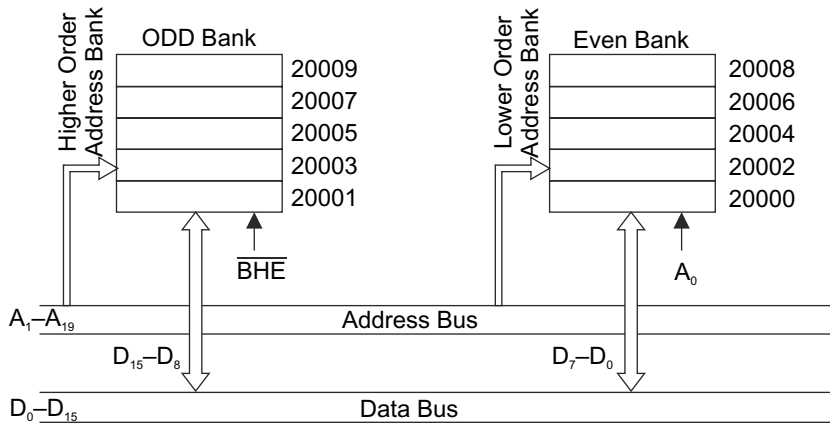


Fig. 5.10 Odd-bank and even-bank memory addressing

The 8086 reads 16-bit data from an odd-addressed bank memory and an even-addressed bank memory simultaneously. One bank is connected to the lower byte of the 16-bit data bus, D_7-D_0 and contains even address bytes, if $A_0 = 0$ and $\overline{BHE} = 1$ an even address bank is selected. The odd-addressed bank is connected to higher bytes of data bus $D_{15}-D_8$ and contains odd address bytes. When $A_0 = 1$ and Bus high Enable, \overline{BHE} is low ($\overline{BHE} = 0$) and the odd-address bank is selected. Any specific byte within even-addressed or odd-addressed banks can be selected by address lines A_1-A_{10} . Table 5.4 shows the memory processing depending upon A_0 and \overline{BHE} . Data can be accessed from memory in four different ways as given below:

- ◆ 8-bit data from even-address bank
- ◆ 8-bit data from odd-address bank
- ◆ 16-bit data starting from even-address bank
- ◆ 16-bit data starting from odd-address bank

Table 5.4 Selection of proper byte from even-addressed and odd-addressed memory banks for processing

\overline{BHE}	A_0	Processing
0	0	Both banks active. 16-bit data transfer, 16-bit word transfer on $AD_{15}-AD_0$
0	1	Only high bank active, one byte transfer on $AD_{15}-AD_8$
1	0	Only low bank active, one byte transfer on AD_7-AD_0
1	1	No bank active

5.7.1 8-Bit Data From Even-Address Bank

To access memory bytes from an even address, information is transferred over the lower half of the data bus D_7-D_0 , if $A_0 = 0$ and \overline{BHE} is high to enable the even bank. For example, assume loading one byte of data into

CH register from memory location within the even-address bank. The data will be accessed from the even bank through D_7-D_0 . Then this data will be transferred into the 8086 over lower 8-bit lines, the 8086 redirects the data over higher 8 bits of its internal 16-bit data path and hence data is loaded into CH register.

Assume 20-bit address is 20002H. $A_0 = 0$, $\overline{BHE} = 1$, and one byte data can be transferred from the memory. Only even bank is selected and only one byte will be transferred from 20002H to data bus as depicted in Fig. 5.11.

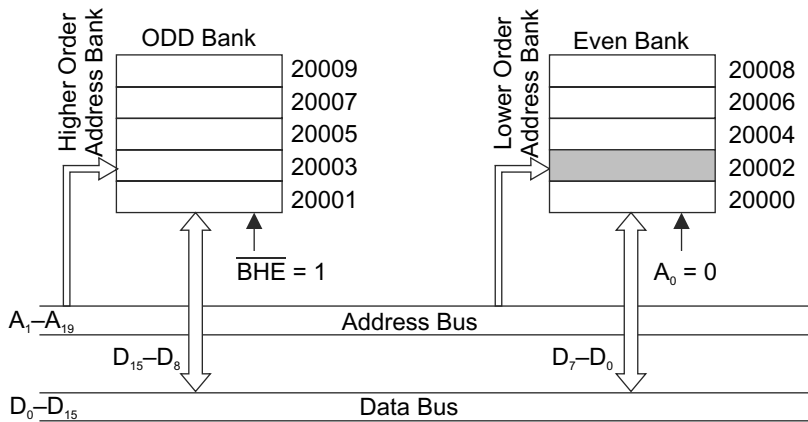


Fig. 5.11 Even bank memory addressing

5.7.2 8-Bit Data From Odd Address Bank

To read one byte from an odd address bank, information must be transferred over the higher-order data bus $D_{15}-D_8$. If $A_0 = 1$, the even memory bank is disabled and \overline{BHE} is low to enable the odd bank as depicted in Fig. 5.12.

Assume the 20-bit address is 20003H. As $A_0 = 1$ and $\overline{BHE} = 0$, only one byte has to be transferred from memory. As odd bank is selected for data transfer, one byte will be transferred from odd bank memory. The data bus $AD_{15}-AD_8$ contents data from memory.

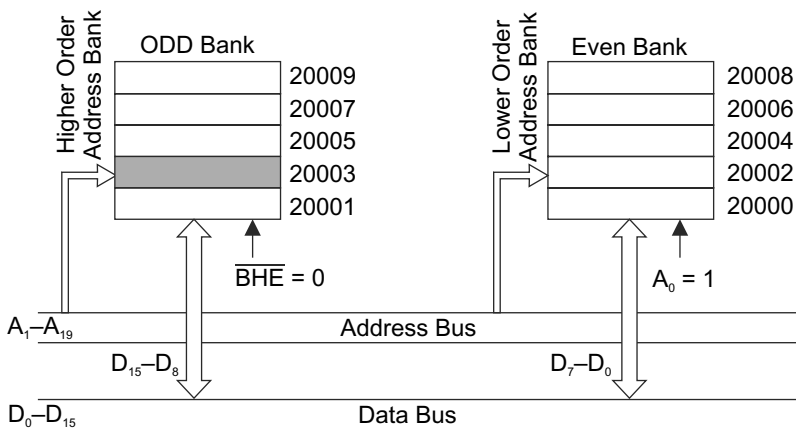


Fig. 5.12 Odd bank memory addressing

5.7.3 16-Bit Data Starting from Even Address Bank

Figure 5.13 shows that 16-bit data from an even address and an odd address respectively is accessed within a single bus cycle. The address lines $A_{19}-A_1$ select the appropriate byte within each bank. While $A_0 = 0$ and \overline{BHE} is low, the even and odd banks are enabled simultaneously. For example, the 20-bit address is 20002H. Since $A_0 = 0$ and $\overline{BHE} = 0$, one word or two bytes have to be transferred from memory locations 20002H and 20003H respectively. Data from an odd bank is transferred to $D_{15}-D_8$ and data from an even bank is transferred to D_7-D_0 data bus. Hence data bus $AD_{15}-AD_0$ contains two byte data from memory. As $\overline{WR} = 0$, $M/\overline{IO} = 1$, 16-bit data can be copied into the data bus from the memory bank.

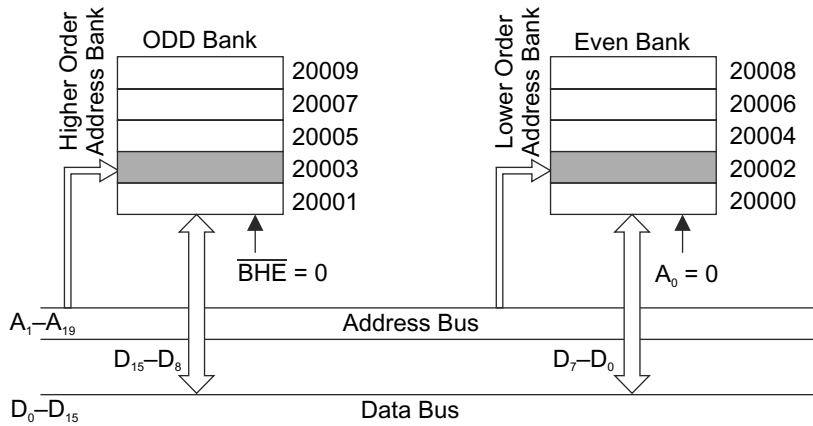


Fig. 5.13 Odd and Even bank memory addressing

5.7.4 16-Bit Data Starting from Odd Address Bank

Generally, a 16-bit word located at an odd address is accessed using two bus cycles. Assume the 20-bit physical address is 20003H and the 8086 transfers a word in two bus cycles. During the first cycle, $A_0 = 1$ and $\overline{BHE} = 0$; the odd bank becomes enabled for data transfer and even bank is disabled. $\overline{RD} = 0$ and $M/\overline{IO} = 1$ for 8086, the odd memory places data on $D_{15}-D_8$ bus. During the first bus cycle the lower byte is accessed from memory location 20003H as depicted in Fig. 5.14(a). In the second cycle, $A_0 = 0$ and $\overline{BHE} = 1$, the even bank

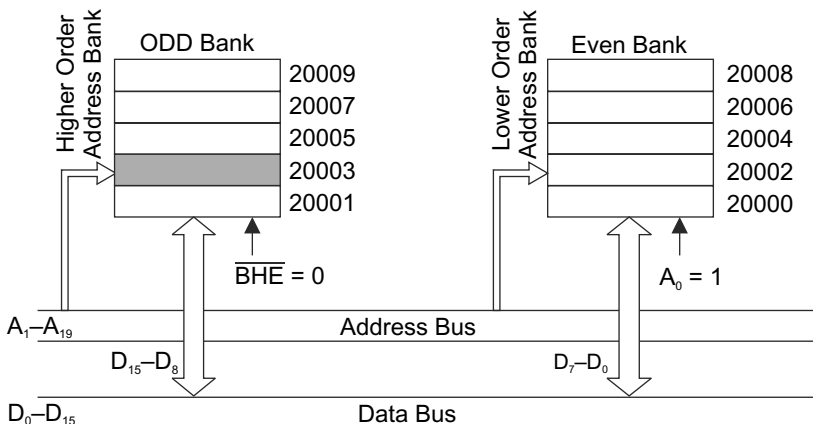


Fig. 5.14 (a) First cycle to access one byte from odd bank memory addressing

of memory becomes enabled and the odd bank is disabled. Then processor output $\overline{R}_D = 0$ and $M/\overline{IO} = 1$. The selected even-bank memory location content is on D_7-D_0 bus. Then data is to be accessed. Therefore, during the second bus cycle, the upper byte is accessed from the even address bank of memory location 20004H as depicted in Fig. 5.14(b)

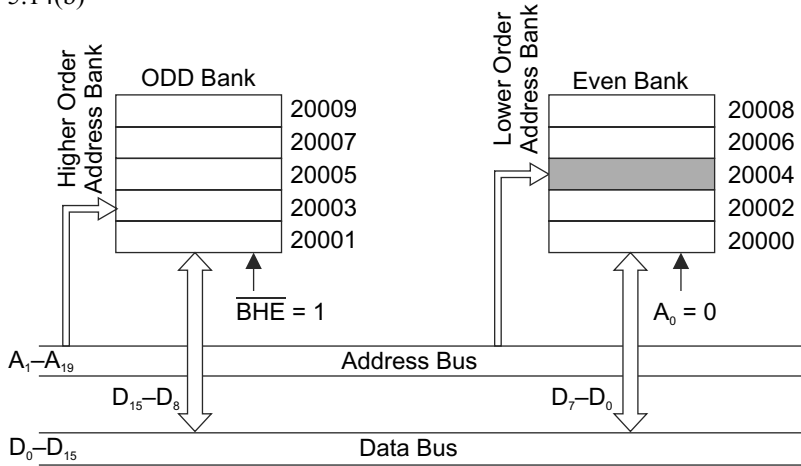


Fig. 5.14 (b) Second cycle to access one byte from even bank memory addressing

5.8 PIN DESCRIPTION OF 8086

The pin diagram of 8086 has been shown in Fig. 5.15. The 8086 can operate either in minimum mode or in maximum mode depending upon the status of the pin MN/\overline{MX} . When $MN/\overline{MX} = 5\text{ V}$, the 8086 works in minimum mode which means that 8086 operates in a single processor environment. If $MN/\overline{MX} = \text{GND}$, it works in maximum mode and the processor can be operated in multiprocessor environment. To differentiate the minimum and maximum mode operations, a set of the 8086 pins change their functions, but other pins have common functions in both the modes. The pin description of 8086 is as follows:

$AD_{15}-AD_0$ (Bi-directional) Address/Data Bus

These lines constitute the time-multiplexed address/data bus. These lines are low-order address bus. They act as an address bus during the first clock cycle multiplexed. When AD lines are used to transmit memory/IO address, the symbol A is used of AD. For example, A represents $A_{15}-A_0$. When data are transmitted through AD lines, the symbol D is used in place of AD. For example, D represents D_7-D_0 , $D_{15}-D_8$ or $D_{15}-D_0$.

$A_{19}-A_{16}$ (Output)

These are high-order address lines and they are time-multiplexed lines. During T_1 , these lines can be used as higher order 4 bits of memory address. But in I/O operation, these lines are low. During T_2 , T_3 , and T_4 , they carry status signals.

$A_{16}/S_3, A_{17}/S_4$ (Output)

A_{16} and A_{17} are time multiplexed with segment identifier signals S_3 and S_4 . During T_1 clock cycle, A_{16} and A_{17} are used as address bits. In T_2 to T_4 clock cycles, these lines carry status signals. Table 5.5 shows memory segment identification.

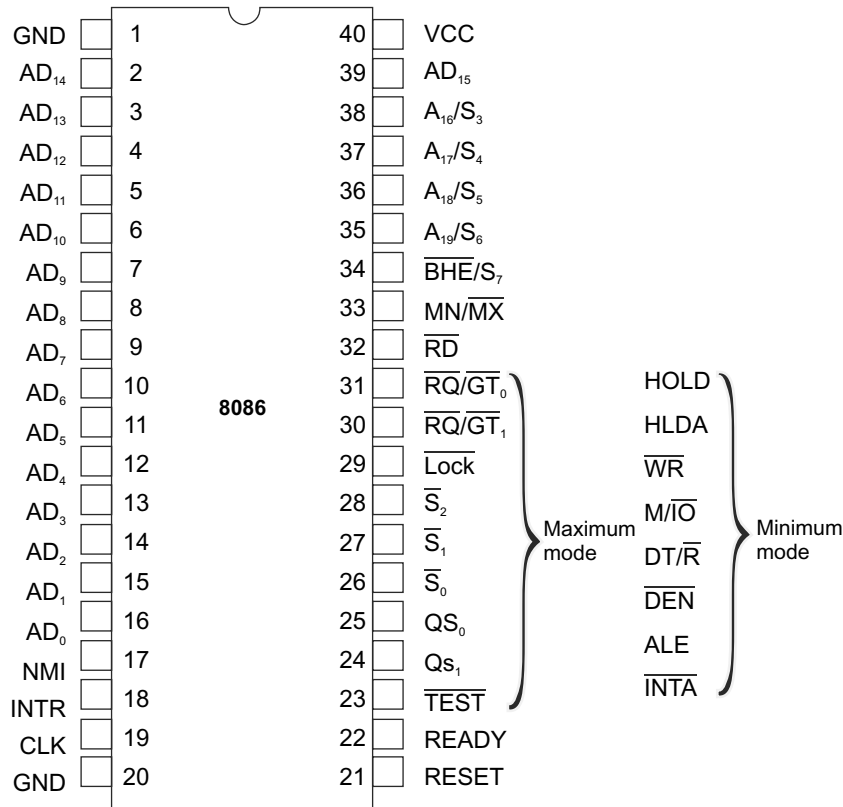


Fig. 5.15 Pin diagram of 8086

Table 5.5 Memory segment identification

S_4	S_3	Function
0	0	Extra segment memory access
0	1	Stack segment memory access
1	0	Code segment memory access
1	1	Data segment memory access

A_{18}/S_5 (Output) A_{18} is time multiplexed with interrupt status S_5 . During T_1 clock cycle, A_{18} is transmitted to the address bus. During other clock cycles (T_2 , T_3 and T_4), the status signal S_5 is transmitted through this line. S_5 is an interrupt-enable status signal. At the beginning of each clock cycle, the status of the interrupt enable flag S_5 is updated.

A_{19}/S_6 (Output) A_{19} is multiplexed with the status signal S_6 . During T_1 clock cycle, A_{19} is transmitted to an address bus. During T_2 to T_4 , the status signal S_6 is available on this line. It is low during T_2 to T_4 .

\overline{BHE}/S_7 (Output) Bus High Enable/Status During T_1 , the bus high enable signal \overline{BHE} can be used to enable data onto the most significant half of the data bus $D_{15}-D_8$. An 8-bit device connected to

the upper half of the data bus uses a \overline{BHE} signal to condition chip select functions. \overline{BHE} is low during T_1 for read, write and interrupt acknowledge cycles when a byte is to be transferred on the high portion of the bus. This pin is multiplexed with the status signal S_7 . The S_7 status signal is available during T_2 to T_4 . The signal is active low, and floats to 3-state OFF in hold. It is low during T_1 for the first interrupt acknowledge cycle. Table 5.6 shows the function of \overline{BHE} and A_0

Table 5.6 Function of \overline{BHE} and A_0

\overline{BHE}	A_0	Function
0	0	Whole word
0	1	Upper byte from/to odd address
1	0	Lower byte from/to even address
1	1	None

\overline{RD} (Output) (Read) This control signal is used for read operation. It is an output signal. It is active when LOW. The Read signal indicates that the processor is performing a memory or I/O read cycle, depending on the state of the S_2 pin. This signal is used to read devices which reside on the 8086 bus. \overline{RD} is active low during T_2 , T_3 and T_w of any read cycle and is guaranteed to remain high in T_2 until the 8086 local bus is floated. This signal floats to 3-state OFF in hold acknowledge.

READY (Input) The addressed I/O or memory devices send acknowledgment through this pin and it indicates that the data transfer is completed. The READY signal from memory or I/O is synchronized by the 8284A clock generator to provide READY input to 8086. This signal is active HIGH. The 8086 READY input is not synchronized. Correct operation is not guaranteed if the set-up and hold times are not met. When READY is HIGH, it indicates that the peripheral is ready to transfer data.

INTR (Interrupt Request) It is a level-triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt vector-look up table located in the system memory. It can be internally masked by software, resetting the interrupt enable bit. INTR is internally synchronized. This signal is active HIGH.

\overline{TEST} (Input) This is used in conjunction with the WAIT instruction. If the \overline{TEST} input is LOW, execution continues. Otherwise the processor waits in an idle state. This input is synchronized internally during each clock cycle on the leading edge of CLK. When it is low, the microprocessor continues execution otherwise it waits.

NMI (Input) Nonmaskable Interrupt This is an edge-triggered input which causes a type 2 interrupt. A subroutine is vectored to via an interrupt vector look-up table located in system memory. NMI is not maskable internally by software. A transition from LOW TO HIGH initiates the interrupt at the end of the current instruction. This input is internally synchronized.

$\overline{MN}/\overline{MX}$ (Input) The minimum/maximum signal indicates the operating mode of 8086. When it is high, the 8086 processor operates in minimum mode. If this pin is low, the processor operates in maximum mode.

RESET (Input)

The reset signal is active HIGH. The processor immediately terminates its present activity and system is reset. The signal must be active HIGH for at least four clock cycles. It restarts execution, as described in the instruction set, when RESET returns low. RESET is internally synchronised.

CLK (Input)

The CLK signal provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing. It is a nonmaskable interrupt request.

V_{CC}

Power supply, + 5 V dc

GND

Ground

5.8.1 Operating Modes of 8086

There are two different operating modes for Intel 8086, namely, the minimum mode and the maximum mode. When only one 8086 processor is to be used in a microcomputer system, the 8086 is used in the minimum mode of operation. In this mode, the CPU issues the control signals required by memory and I/O devices. In a multiprocessor system, the 8086 processor operates in the maximum mode. In maximum-mode operation, control signals are issued by the Intel 8288 bus controller which is used with 8086 for this purpose. The level of the pin MN/\overline{MX} decides the operating mode of 8086. When MN/\overline{MX} is high, the CPU operates in the minimum mode. When it is low, the CPU operates in the maximum mode. The pins 24 to 31 of 8086 issue two different sets of signals. One set of signals is issued when the processor operates in the minimum mode. The other set of signals is issued while the processor operates in the maximum mode. Thus, the pins from 24 to 31 have alternate functions. In this section, the pin description of minimum mode and maximum mode operations are discussed. The difference between minimum mode and maximum mode operations of 8086 microprocessor is given in Table 5.7.

Table 5.7 Difference between MAX mode operation and MIN mode operation in the 8086 microprocessor

<i>Min Mode Operation</i>	<i>Max Mode Operation</i>
The minimum/maximum MN/\overline{MX} signal indicates the operating mode of 8086. When it is high, the 8086 processor operates in minimum mode.	The minimum/maximum MN/\overline{MX} signal indicates the operating mode of 8086. If this pin is low, the processor operates in maximum mode.
In MIN mode operation, only one microprocessor will be in the system configuration.	In MAX mode operation, there may be more than one microprocessor in the system configuration. But the other components in the system are the same as in the minimum mode system.
In this mode, the CPU issues the control signals required by memory and I/O devices.	In maximum mode operation, control signals are issued by the Intel 8288 bus controller which is used with 8086 for this very purpose.
In this mode, PIN numbers 24 to 31 are used as \overline{INTA} (Output) Interrupt acknowledge, ALE (Output) Address latch enable, \overline{DEN} (Output) Data enable, $\overline{DT/R}$ (Output), Data Transmit/Receive, \overline{WR} (Output) Write, HLDA (Output) HOLD Acknowledge and HOLD (Input) Hold.	In this mode, PIN numbers 24 to 31 are used as QS_1 , QS_0 (Output) Instruction Queue Status, $\overline{S_2}$, $\overline{S_1}$ and $\overline{S_0}$ (Output) Status signals, \overline{LOCK} (Output) $\overline{RQ/GT_0}$ and $\overline{RQ/GT_1}$ Request/Grant.

5.8.2 Pin Description in Minimum Mode

In the minimum-mode operation, the pin MN/\overline{MX} is connected to 5 V dc supply. The minimum-mode operation is cheaper as all control signals for memory and I/O are generated by the microprocessor. The schematic pin diagram of 8086 for minimum-mode operation is illustrated in Fig. 5.16. The functions of all pins except pins 24 to 31 of 8086 are same in this mode. The pin descriptions of 24 to 31 for the minimum mode are as follows:

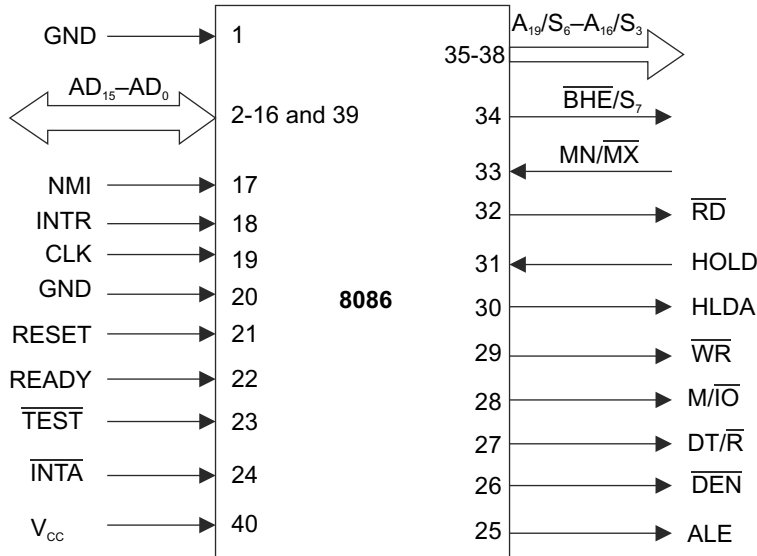


Fig. 5.16 Pin diagram of 8086 for minimum mode of operation

\overline{INTA} (Output) Interrupt Acknowledge This signal used as a read strobe for interrupt acknowledge cycles. It is active low during T_2 , T_3 and T_w of each interrupt acknowledge cycles. On receiving an interrupt signal, the processor issues an interrupt acknowledge signal. It is active LOW.

ALE (Output) Address Latch Enable This signal is provided by the processor to latch the address into the 8282/8283 address latch. It is HIGH during T_1 .

\overline{DEN} (Output) Data Enable This signal can be provided as an output enable for the 8286/8287 in a minimum system which uses the transceiver. \overline{DEN} is active low during each memory and I/O access and for \overline{INTA} cycles. For a read or \overline{INTA} cycle, it is active from the middle of T_2 until the middle of T_4 , while for a write cycle, it is active from the beginning of T_2 until the middle of T_4 . \overline{DEN} floats 3-state OFF in local bus-hold acknowledge. When Intel 8286/8287 octal bus transceiver is used, this signal acts as an output enable signal. It is active LOW.

DT/\overline{R} (Output) Data Transmit/Receive DT/\overline{R} signal is required in minimum-mode operation, when 8286/8287 data bus transceiver is used for data flow. This signal is used to control the direction of data flow through the transceiver. When it is HIGH, data are sent out. When it is LOW, the CPU wants to access the I/O device. Logically, DT/\overline{R} is equivalent to \overline{S}_1 in the maximum mode, and its timing is same as for M/\overline{IO} . This signal floats to 3-state OFF in local bus-hold acknowledge.

\overline{WR} (Output) Write

This pin indicates that the processor is performing a memory write or I/O write cycle, depending on the state of the M/\overline{IO} signal. \overline{WR} is active for T_2 , T_3 and T_w of any write cycle. It is active LOW and floats to 3 state OFF in local bus hold acknowledge. When it is LOW, the CPU performs memory or I/O write operations.

HLDA (Output) HOLD Acknowledge

This signal is issued by the processor when it receives a HOLD signal. This is an active HIGH signal. When the HOLD request is removed, HLDA goes LOW.

HOLD (Input) Hold

A HOLD signal indicates that another device (master) in a microcomputer system is requesting a local bus hold for using the address and data bus. Then the master sends a HOLD request to processor through this pin. It is an active HIGH signal.

5.8.3 Pin Description In Maximum Mode

For the maximum mode of operation, the pin MN/\overline{MX} is grounded. The maximum mode is designed to be used when a coprocessor exists in the system. The schematic pin diagram of 8086 for maximum-mode operation is depicted in Fig. 5.17. In this mode, the functions of all pins except pins 24 to 31 of 8086 are same. The description of the pins from 24 to 31 are as follows:

 QS_1 , QS_0 (Output) Instruction Queue Status

These two signals are decoded to provide instruction queue status as given below:

Table 5.8 Functions of queue status signals

QS_1	QS_0	Function
0	0	No operation
0	1	First byte of opcode from queue
1	0	Empty the queue
1	1	Subsequent byte from queue

 \overline{S}_2 , \overline{S}_1 and \overline{S}_0 (Output) Status Signals

These signals are connected to the bus controller Intel 8288. The bus controller decodes these signals to generate eight separate memory and I/O access control signals as depicted in Table 5.9 which shows the logic for status signals.

Table 5.9 Logic for status signals \overline{S}_2 , \overline{S}_1 and \overline{S}_0

\overline{S}_2	\overline{S}_1	\overline{S}_0	Function
0	0	0	Interrupt acknowledge
0	0	1	Read data from I/O port
0	1	0	Write data into I/O port
0	1	1	Halt
1	0	0	Code access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive

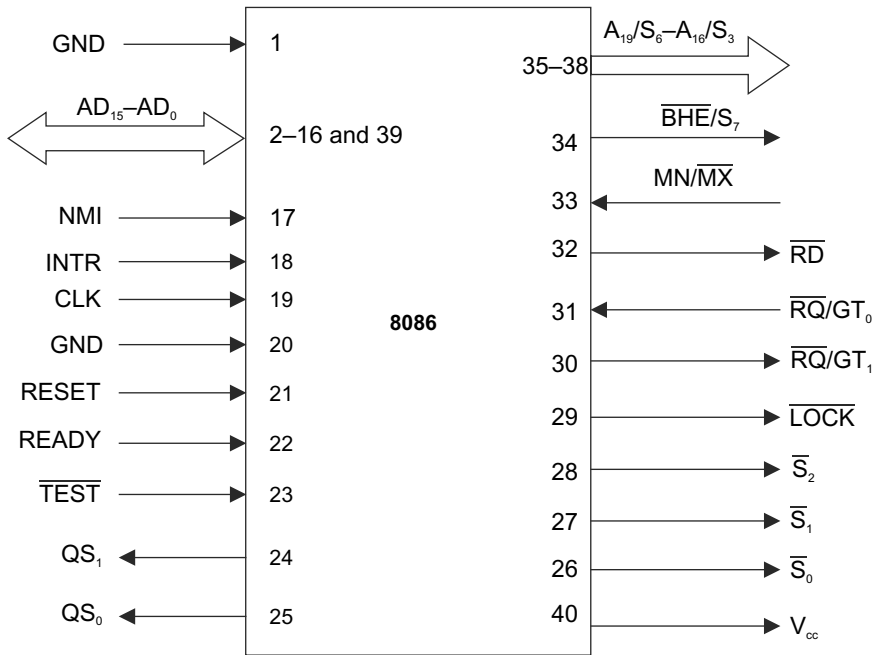


Fig. 5.17 Pin diagram of 8086 for maximum mode of operation

LOCK (Output) This output indicates that other system bus master are not to gain control of the system bus while \overline{LOCK} is active low. The \overline{LOCK} signal is activated by the LOCK prefix instruction and remains active until the completion of the next instruction. This signal is active LOW and floats to 3-states OFF in hold acknowledge.

$\overline{RQ/GT}_0$ and $\overline{RQ/GT}_1$ Request/Grant These pins are used by other processors in a multiprocessor environment. Local bus masters of other processors force the processor to release the local bus at the end of the processor's current bus cycle. Each pin is bidirectional and has internal pull-up resistors. Hence, they may be left unconnected.

5.9 MEMORY READ AND WRITE BUS CYCLE OF 8086

The bus cycle means the Bus Interface Unit (BIU) phenomenon. It is known to us that the EU and BIU work asynchronously. The EU takes instructions from the instruction queue and executes instructions in a number of clock periods continuously. These clock periods are not grouped to form machine cycles. Therefore, the EU does not use machine cycles. The EU waits for the BIU to handover the instruction whenever the program starts or the program executes any branch instruction or the BIU is executing data memory access for EU. The BIU fetches the instruction code from the memory, reads data from memory or I/O devices, and writes data into memory or I/O devices. The clock periods are grouped whenever the BIU accesses the memory and I/O devices for read and write operations. When any external memory or I/O devices are accessed, only four clock cycles are required to perform a read or write operation. These four clock cycles are grouped, which is called *bus cycle*. In this section, memory and I/O read and write bus cycles for both minimum and maximum modes are discussed.

5.9.1 Minimum-Mode Operation and Timing Diagram

Figure 5.18 shows the minimum-mode configuration of 8086. This figure shows a group of ICs which generate address-bus, data-bus and control-bus signals. The ROM and RAM ICs and I/O ports are connected to the CPU through these buses. The 8282 Latch ICs are used to latch the address from 8086 processor address/data bus. $A_{15}-A_0$, $A_{19}-A_{16}$ and \overline{BHE} are latched during T_1 state. The Output Enable (\overline{OE}) of 8288 I/O ports are grounded, therefore the bus will not be floated or have high impedance state. The ALE signal from 8086 is used to strobe address lines in 8282 latches. As the data bus is bidirectional, the bidirectional transceivers 8286 ICs are used. The working principle of 8086 in minimum mode can be explained with a timing diagram. The opcode fetch cycle is similar to the memory read cycle. In this section, memory read and write cycles are explained in detail. The types of data transfer depend on $\overline{M}/\overline{IO}$, \overline{RD} and \overline{WR} signals as depicted in Table 5.10.

Table 5.10 Types of data transfer

$\overline{M}/\overline{IO}$	\overline{RD}	\overline{WR}	Operation
0	0	1	I/O read
0	1	0	I/O write
1	0	1	Memory read
1	1	0	Memory write

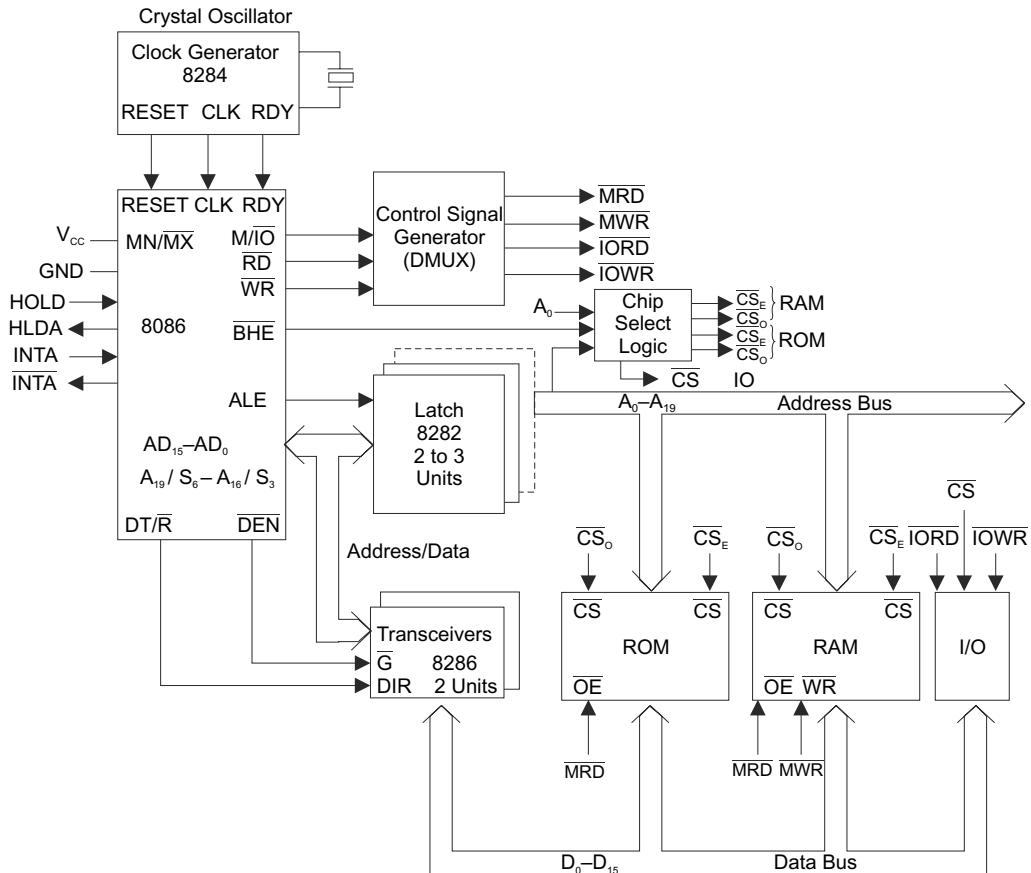


Fig. 5.18 Minimum-mode configuration of 8086

1. Memory Read Bus Cycle for Minimum Mode

During minimum-mode operation of 8086 processor, $\overline{MN}/\overline{MX}$ is +5Vdc. The timing diagram for memory read bus cycle is shown in Fig. 5.19. The following actions take place during four different clock cycles:

✓ **T_1 Clock Cycle** During T_1 clock cycle, the 8086 processor sends the 20-bit address on the address bus. The 16-bit least significant bits of address sends on $AD_{15}-AD_0$ and four most significant bits of address are put on $A_{19}-A_{16}$ lines.

- ◆ ALE is high only during T_1 . Address is put in the address bus. The falling edge of ALE is used to latch the address from the address bus.
- ◆ \overline{BHE} is high or low depending on 8 or 16-bit read from odd/even address.
- ◆ $\overline{M}/\overline{IO}$ is high to indicate memory operation. It remains high during the entire bus cycle.
- ◆ $\overline{DT}/\overline{R}$ is low and remains low throughout the complete bus cycle, to indicate the direction of data transfer as memory to the processor.

✓ **T_2 Clock Cycle** During T_2 clock cycle, the $AD_{15}-AD_0$ go into high-impedance state. This is indicated by dotted lines as depicted in Fig. 5.19 as bus drivers are disabled.

- ◆ \overline{RD} goes low during T_2 . This signal can be used by the selected memory IC to enable its output buffer. This is a read-control signal.
- ◆ When READY signal is high during T_2 , it indicates that the memory device is ready and the 8086 operates normally according to the bus cycle. The READY signal is used by slow memory devices. If the memory is not ready to transfer data, it does not make READY signal high, otherwise the READY is low. If READY is low during T_2 , the 8086 inserts a wait state between T_3 and T_4 . The 8086 processor always takes necessary steps to read data from the memory only when READY becomes high.
- ◆ \overline{DEN} goes high to enable the 8286/8287 transceiver.
- ◆ \overline{BHE} goes high.
- ◆ During T_2 to T_4 , status signals S_3, S_4, S_5 and S_6 are put on the address lines A_{16}/S_3 to A_{19}/S_6 . S_3 and S_4 are used to identify the memory segment which is to be accessed as illustrated in Table 5.5. S_5 indicates interrupt enable status. The status of the interrupt enable flag bit, S_5 is updated at the beginning of each clock cycle.

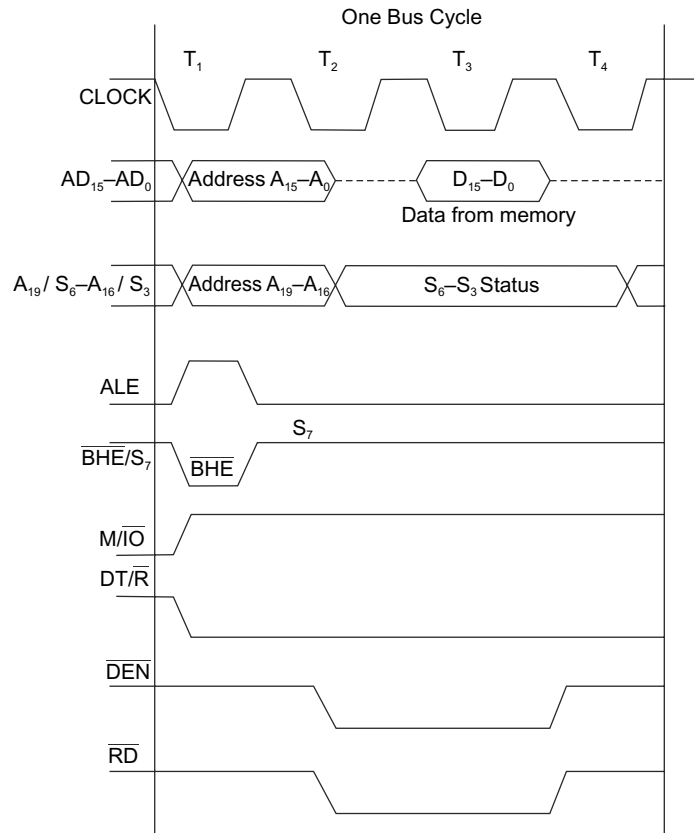


Fig. 5.19 Memory read bus cycle of 8086 for minimum mode

✓ **T₃ Clock Cycle**

- ◆ \overline{DEN} is low
- ◆ Data is put on lines AD₀—AD₁₅.

✓ **T₄ Clock Cycle**

- ◆ M/\overline{IO} goes low just after T₄ clock cycle.
- ◆ \overline{RD} goes high
- ◆ All bus signals are deactivated and be prepared for the next bus cycle.

2. Memory Write Bus Cycle for Minimum Mode

The timing diagram for memory write bus cycle is shown in Fig. 5.20. The following actions take place during four different clock cycles:

✓ **T₁ Clock Cycle** The write cycle is similar to read cycle, but there are some differences. During T₁ clock cycle, the 8086 processor sends the 20-bit address on the address bus. The 16-bit least significant bits of address sends on AD₁₅—AD₀ and four most significant bits of address put on A₁₆—A₁₉ lines.

- ◆ ALE is high only during T₁. Address is put in the address bus. The falling edge of ALE is used to latch the address from the address bus.
- ◆ \overline{BHE} is high or low depending on 8-or 16-bit read from odd/even address.
- ◆ M/\overline{IO} is high to indicate memory operation. It remains high during the entire bus cycle.
- ◆ DT/\overline{R} is high and remains high throughout the complete bus cycle, to indicate the direction of data transfer as the processor to memory.
- ◆ \overline{DEN} goes high to enable the 8286/8287 transceiver.

✓ **T₂ Clock Cycle**

- ◆ \overline{WR} is low as write control signal.
- ◆ \overline{BHE} goes high.
- ◆ Bus is turned around.
- ◆ Status is put on the A₁₉—A₁₆ lines. The activity starts during T₂ and continues till T₄.

✓ **T₃ Clock Cycle**

- ◆ Data is put on the AD₁₆—AD₀ lines.

✓ **T₄ Clock Cycle**

- ◆ \overline{WR} becomes high.
- ◆ M/\overline{IO} goes low.
- ◆ \overline{DEN} is low.
- ◆ DT/\overline{R} goes low.

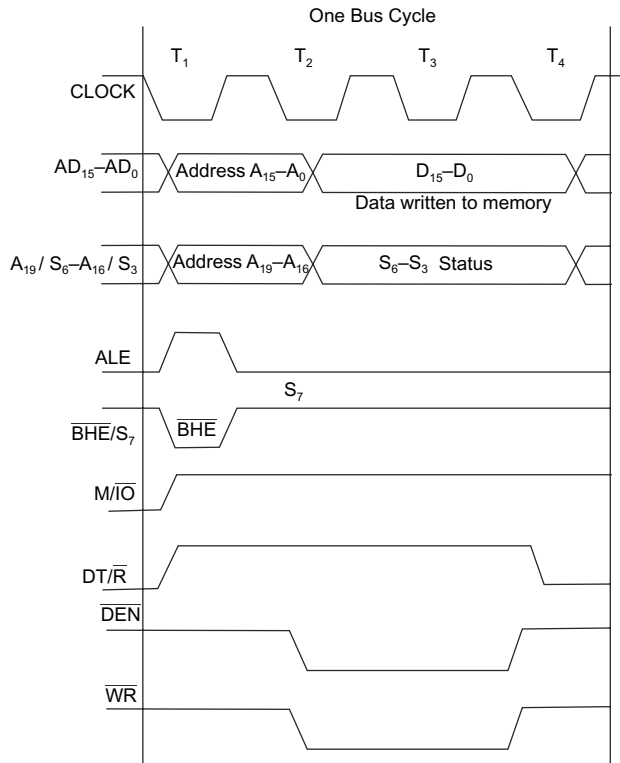


Fig. 5.20 Memory write bus cycle of 8086 for minimum mode

3. I/O READ and Write BUS CYCLE

The I/O read bus cycle is similar to the memory read cycle. The M/\overline{IO} signal is low for I/O read operation and all other signals are same as memory read operation. As illustrated in Fig. 5.19, it can be used as I/O read bus cycle by changing the M/\overline{IO} signal only. The I/O write bus cycle is also similar to memory write cycle. The M/\overline{IO} signal is low for I/O write operation and all other signals are same for memory write operation as depicted in Fig. 5.20.

5.9.2 Maximum-Mode Operation and Timing Diagram

Figure 5.21 shows the maximum-mode configuration of the 8086 processor. In the maximum mode, MN/\overline{MX} pin of 8086 is strapped to grounded and pins 24 to 31 of 8086 are active in maximum mode. In this mode, microprocessor status signals \overline{S}_2 , \overline{S}_1 , and \overline{S}_0 are fed to a 8288 bus controller. The bus controller interprets status signals \overline{S}_2 , \overline{S}_1 , and \overline{S}_0 to generate bus timing and control signals. Here the 8288 bus controller derive \overline{RD} , \overline{WR} , \overline{DEN} , $\overline{DT/R}$ and ALE, \overline{MRDC} , \overline{MWTC} , \overline{AMWC} , \overline{IORC} , \overline{IOWC} and \overline{AIOWC} control output signals.

\overline{IORC} , \overline{IOWC} are I/O read command and I/O write command signals respectively. These signals enable I/O read or write operations. \overline{MRDC} and \overline{MWTC} are used as memory read and memory write command signals respectively.

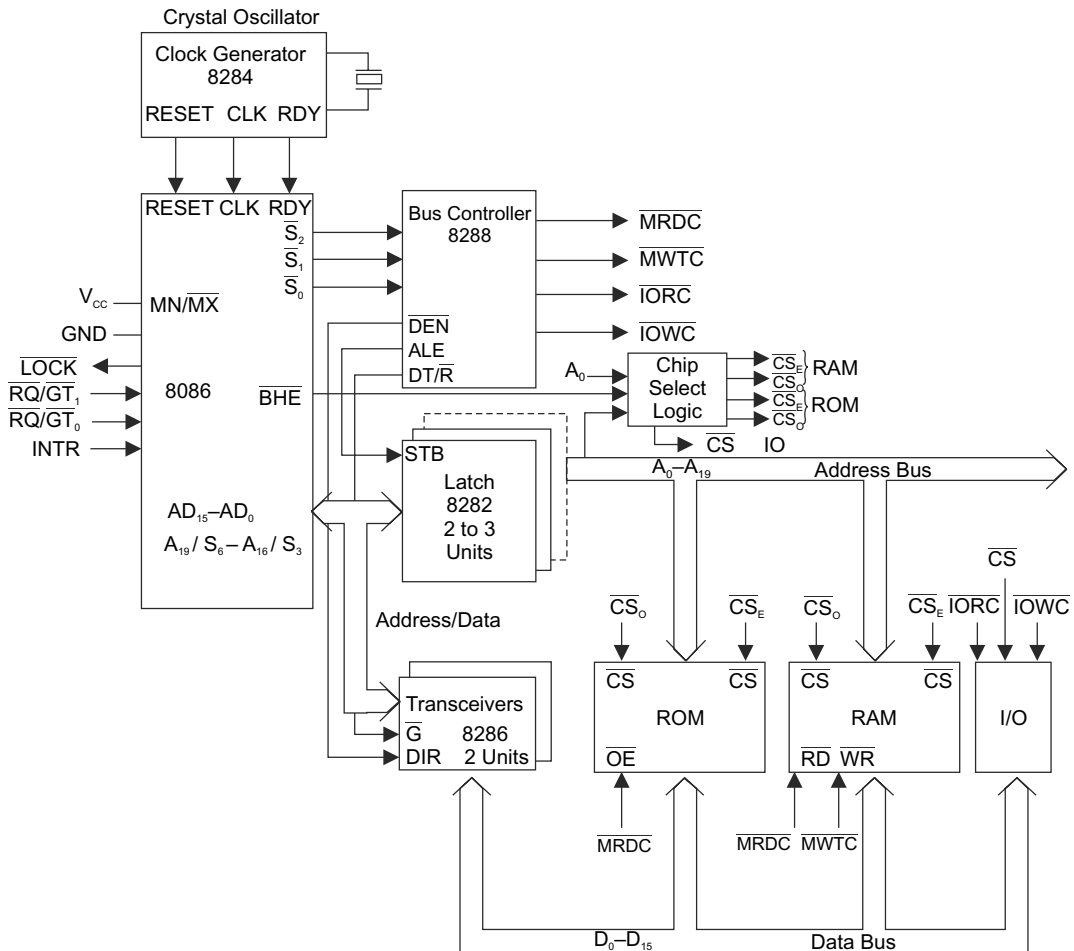


Fig. 5.21 Maximum-mode configuration of 8086

In this mode, more than one processor is interconnected within a system. \overline{AEN} , IOB and CEN pins are very useful for multiprocessor systems. \overline{AEN} , and IOB are grounded, but CEN pin is connected with + 5 V. The $\overline{MCE/PDEN}$ pin output depends upon the status of the IOB. When IOB is grounded, this pin works as a master cascade enable to control cascaded 8259A. When IOB is not grounded, this pin acts as a peripheral data enable. \overline{INTA} pin is used to generate two interrupt acknowledge signals and fed to the interrupt controller.

The working principle of 8086 in maximum mode can be explained with a timing diagram. The maximum-mode timing diagram is subdivided into the memory read cycle and memory write cycle. The address/data and address/status timings are similar to the minimum mode. In this section, memory read and write cycles are explained in detail.

1. Memory Read Bus Cycle for Maximum Mode

During minimum mode operation of 8086 processor, $\overline{MN}/\overline{MX}$ is grounded. The timing diagram for memory read bus cycle in maximum-mode operation of 8086 is shown in Fig. 5.22. The following actions take place during four different clock cycles:

✓ **T_1 Clock Cycle**

- ◆ $\overline{S}_2, \overline{S}_1$ and \overline{S}_0 are set by 8086 in the starting of clock cycle. These signals are decoded by the 8288 bus controller
- ◆ ALE is high only during T_1 . Address is put in the address bus. The falling edge of ALE is used to latch the address from the address bus.
- ◆ \overline{BHE} is high or low depending on 8-or 16-bit read from odd/even address.
- ◆ $\overline{DT}/\overline{R}$ goes low

✓ **T_2 Clock Cycle**

- ◆ \overline{BHE} is high.
- ◆ \overline{DEN} goes high to enable the 8286/8287 transceiver
- ◆ \overline{MRDC} is low for memory read control signal

✓ **T_3 Clock Cycle**

- ◆ Data is put on address/data lines from memory
- ◆ The status signals $\overline{S}_2, \overline{S}_1$ and \overline{S}_0 become active

✓ **T_4 Clock Cycle**

- ◆ \overline{MRDC} is high
- ◆ \overline{DEN} goes low to disable the 8286/8287 transceiver
- ◆ $\overline{DT}/\overline{R}$ goes high
- ◆ The READY signal is sampled at the end of T_2 . If it is low, wait states are inserted between T_3 and T_4

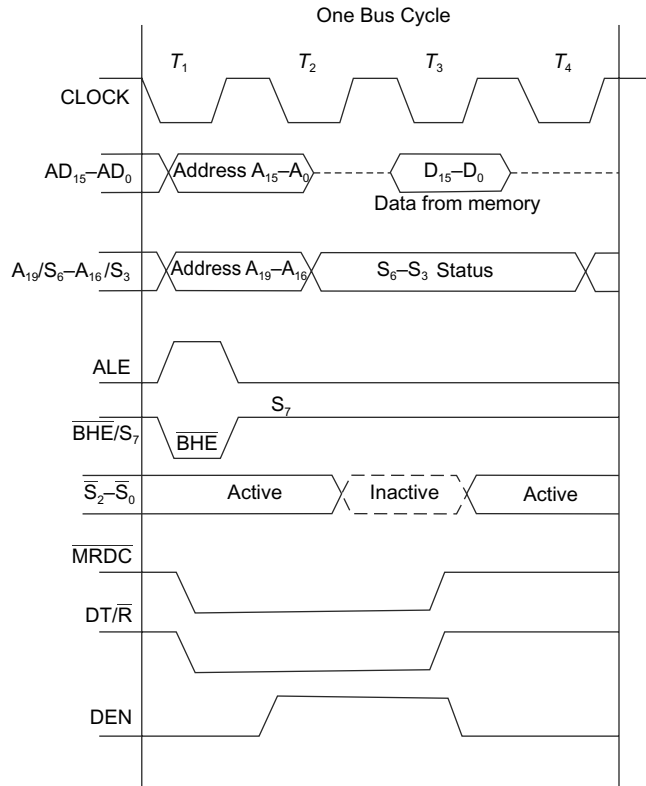


Fig. 5.22 Memory read bus cycle of 8086 for maximum mode

2. Memory Write Bus Cycle for Maximum Mode

bus cycle in maximum-mode operation of 8086 is shown in Fig. 5.23. The following actions take place during four different clock cycles:

✓ **T₁ Clock Cycle** $\overline{S}_2, \overline{S}_1$ and \overline{S}_0 are set by 8086 in the beginning of clock cycle. These signals are decoded by the 8288 bus controller

- ◆ ALE is high only during T₁. Address is put in the address bus. The falling edge of ALE is used to latch the address from the address bus.
- ◆ \overline{BHE} is high or low depending on 8- or 16-bit read from odd/even address.

✓ **T₂ Clock Cycle**

- ◆ \overline{BHE} is high
- ◆ \overline{DEN} goes high to enable the 8286/8287 transceiver

✓ **T₃ Clock Cycle**

- ◆ \overline{MWTC} is low for memory write control signal
- ◆ Data is put on address/data lines
- ◆ The status signals $\overline{S}_2, \overline{S}_1$ and \overline{S}_0 become active

✓ **T₄ Clock Cycle**

- ◆ \overline{MWTC} is high
- ◆ \overline{DEN} goes low to disable the 8286/8287 transceiver

The timing diagram for memory write

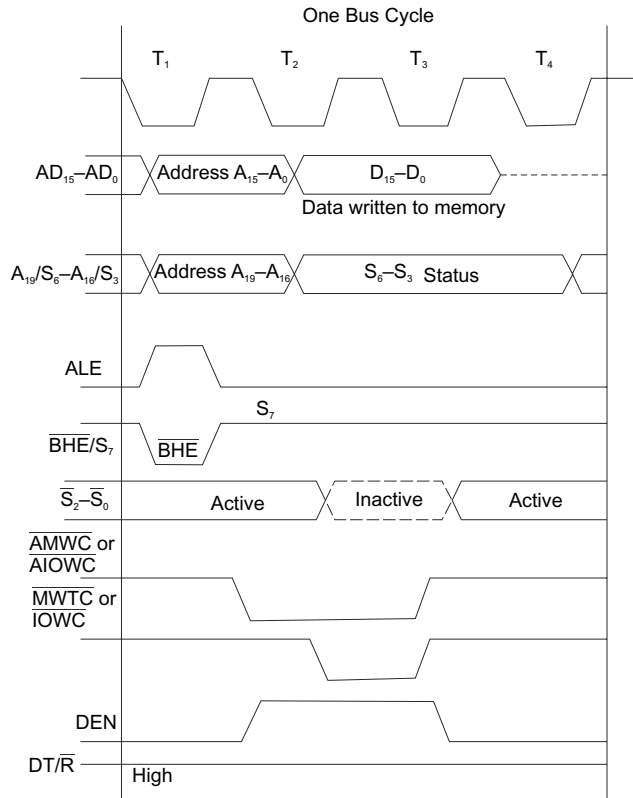


Fig. 5.23 Memory write bus cycle of 8086 for maximum mode

3. I/O READ and Write BUS CYCLE

The I/O write bus cycle is similar to memory write cycle. The memory write operation can be performed by write control signals \overline{AMWTC} , but \overline{AIOWC} control signal is used for I/O write. The \overline{AMWTC} signal is activated during T₂ to T₄, i.e., one clock cycle earlier than \overline{MWTC} . Therefore, in I/O write bus cycle using \overline{AIOWC} , \overline{AMWTC} will be replaced by \overline{AIOWC} as depicted in Fig. 5.23. Similarly, \overline{MRDC} will be replaced by \overline{IORC} in memory read cycle as I/O read bus cycle is similar to memory read cycle as given in Fig. 5.22.

5.10 INTEL 8088 PROCESSOR

The 8086 processor offers tremendous flexibility in programming as compared to 8085. Therefore, after the introduction of 8086, research work was done for a microprocessor chip which has programming flexibility like 8086 and external interface ICs of 8085, so that all existing circuits of 8085 could be compatible with the new processor. Then the 8088 processor was developed. The Intel 8088 is an 8-bit microprocessor. Its internal architecture is same as that of 8086, i.e., architecture of a 16-bit microprocessor. But the 8088 has only 8 data lines and hence it can use 8-bit I/O devices which are cheaper compared to 16-bit I/O devices. Personal computers based on the 8088 CPU are cheaper compared to personal computers based on 8086 and 80286 CPUs. The clock frequency of 8088 is 5 MHz and that for 8088-2 is 8 MHz. The 8088 microprocessor is

available in 40-pin IC and operates at + 5 V dc supply. Its register set, instructions and addressing modes are same as those of 8086. Its CHMOS version 80C88A operates at 8 MHz clock. The instruction queue length in case of the 8088 processor is of 4 bytes whereas that in 8086 is of 6 bytes. The 8088 CPU uses 8087 math coprocessor, 20-bit address bus and can directly address up to 1 MB of memory.

A computer built around the 8088 CPU uses 8284 clock generator, 8282 latches, 8286 transceivers, 8288 bus controller, 8087 math coprocessor, 8237 DMA controller, 8259 interrupt controller, etc. The 8088 CPU was very popular and widely used in personal computers, PC/XT. Presently, 8088-based computers are no longer manufactured. In this section, the architecture of the 8088 processor, pin description, addressing and timing diagram are explained.

5.10.1 Architecture of 8088 Microprocessor

Figure 5.24 shows the architecture of the 8088 processor. The set of registers of 8088 is approximately same

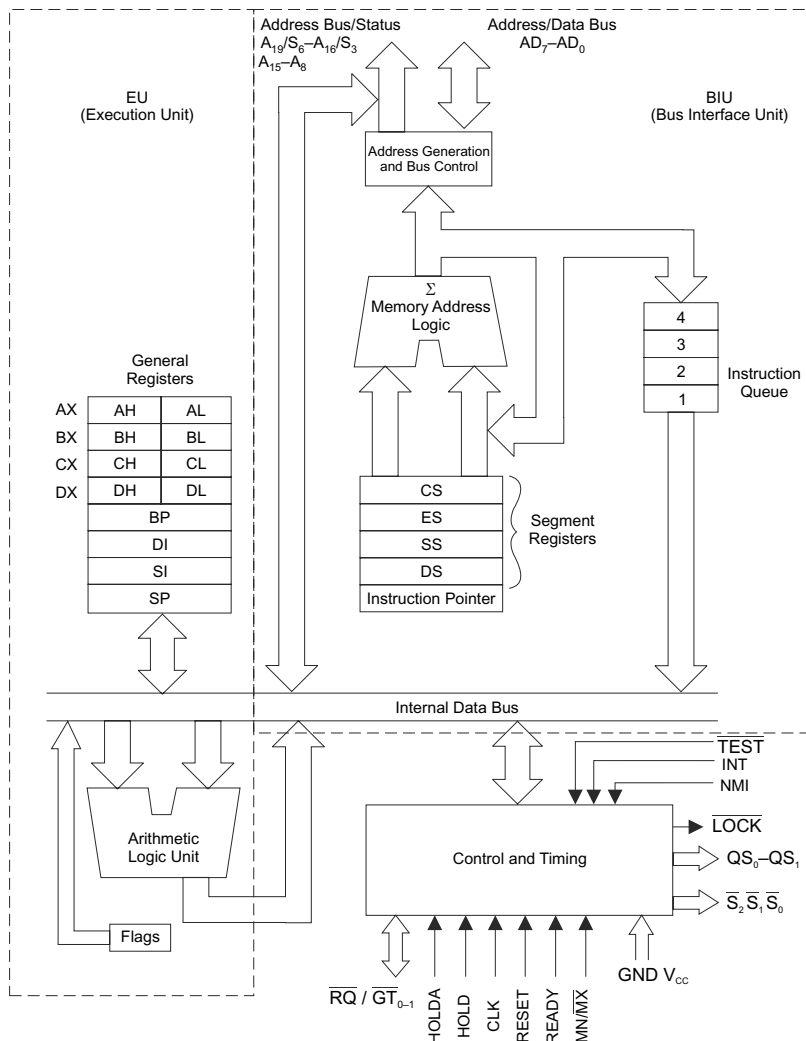


Fig. 5.24 Block diagram of 8088 microprocessor

as that of 8086. The architecture of 8088 is same as 8086 architecture, but there are two changes. The 8088 has a 4-byte instruction queue in place of 6-byte instruction queue in 8086 and the data bus of 8088 is 8-bit. The other function blocks are the same as the 8086 processor.

The 8088 processor has 1 Mbyte addressing capability and it has a 20-bit address or 20 addressing lines. The concept of segmented memory and the computational method of physical address are used in the 8088 processor without any change and it is same as the 8086 processor. The memory organization and addressing methods of 8088 are similar to 8086. There is no concept of even-address bank and odd-address bank of memory in 8088. Therefore, the complete memory is consistently addressed as a bank of 1Mbyte memory locations with the help of the segmented memory conception.

As the data bus is 8-bit, the 8088 can access only a byte at a time. Therefore, the speed of operation of 8088 will be reduced as compared to 8086, though internal data bus of 8088 is 16 bits and it can process the 16-bit data internally. Due to change in address and data bus structure, the timing diagrams of 8088 are different from 8086.

5.10.2 Pin Description of 8088 Processor

The pin diagram of 8088 is depicted in Fig. 5.25. The functions of 8088 pins except AD_7-AD_0 , $AD_{15}-AD_8$, \overline{SS}_0 and IO/\overline{M} pins are exactly same as the pins of 8086. Consequently, the pins functions of \overline{SS}_0 , IO/\overline{M} , AD_7-AD_0 , $A_{15}-A_8$ are explained in this section.

AD_7-AD_0 (Address/Data Bus)

The AD_7-AD_0 lines are operated as time-multiplexed address/data bus. When the ALE signal is high, these lines can be used as the address of the lower-order memory location address or I/O port address. When ALE is low, these lines are used as data bus and during T_1 clock cycle, the AD_7-AD_0 bus is used as address bus. During T_2 , T_3 , and T_4 states, these lines are used as data buses. These lines are in high impedance state in hold acknowledge and interrupt acknowledge cycles.

$A_{15}-A_8$ (Address Bus)

The $A_{15}-A_8$ lines are used as lower-order memory location address throughout the entire bus cycle. During hold acknowledge, these address lines are tristated or at high impedance state.

✓ \overline{SS}_0 This pin is newly introduced in the 8088 processor instead of the \overline{BHE} pin in 8086. During minimum-mode operation, the pin \overline{SS}_0 is equivalent to \overline{S}_0 . In the maximum mode, \overline{SS}_0 is always high in maximum mode.

✓ IO/\overline{M} The IO/\overline{M} pin is similar to the M/\overline{IO} pin of 8086. The function of this pin is to operate the 8088 processor as an 8085 processor, interfacing of memory and I/O devices.

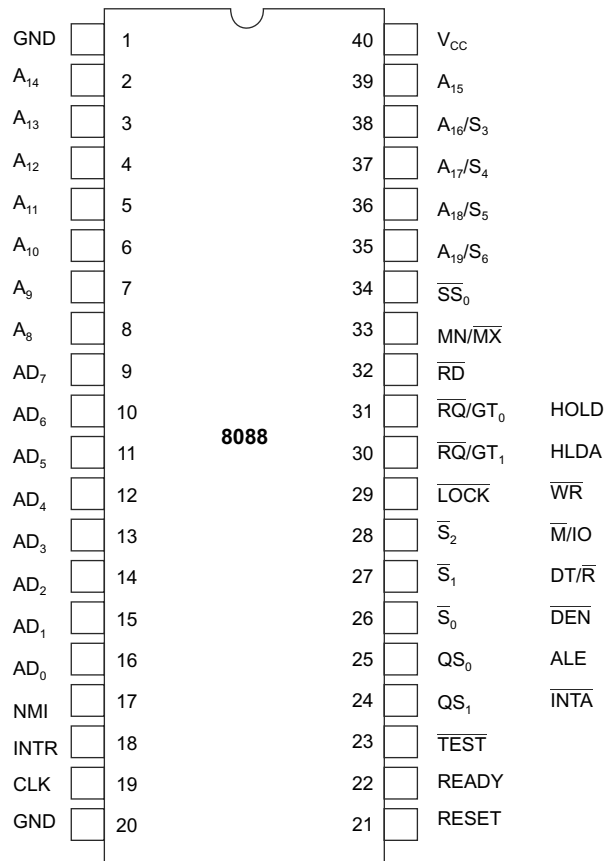


Fig. 5.25 Pin diagram of 8088

In the minimum mode, the operations of the 8088 microprocessor depend on control signals \overline{SS}_0 , IO/\overline{M} and DT/\overline{R} as given in Table 5.11. The \overline{S}_0 pin is always high in the maximum-mode operation. The other pin functions and timings of 8088 are same as 8086.

Table 5.11 Operation of 8088 processors based on control signals

IO/\overline{M}	DT/\overline{R}	\overline{SS}_0	Operation
0	0	0	Code access
0	0	1	Read memory
0	1	0	Write memory
0	1	1	Passive
1	0	0	Interrupt acknowledge
1	0	1	Read I/O port
1	1	0	Write I/O port
1	1	1	HALT

5.10.3 Timing Diagram of the 8088 Microprocessor

Each bus cycle of the 8088 processor consists of four T states: T_1 , T_2 , T_3 and T_4 . During the first clock cycle T_1 , ALE signal is high and $A_{19}/S_6-A_{16}/S_3$ are used as $A_{19}-A_{16}$ address buses, AD_7-AD_0 can be used as A_7-A_0 address buses. Hence, the leading edge of ALE is used to latch the valid 20-bit address during T_1 states. After T_1 state $A_{19}/S_6-A_{16}/S_3$ are used as status signals S_6-S_3 , the middle bus $A_{15}-A_8$ are always active as address buses but AD_7-AD_0 are tristated. During T_3 and T_4 data are read from memory and placed into the data bus. Therefore, AD_7-AD_0 is used as data bus in T_3 and T_4 durations. Figure 5.26 shows

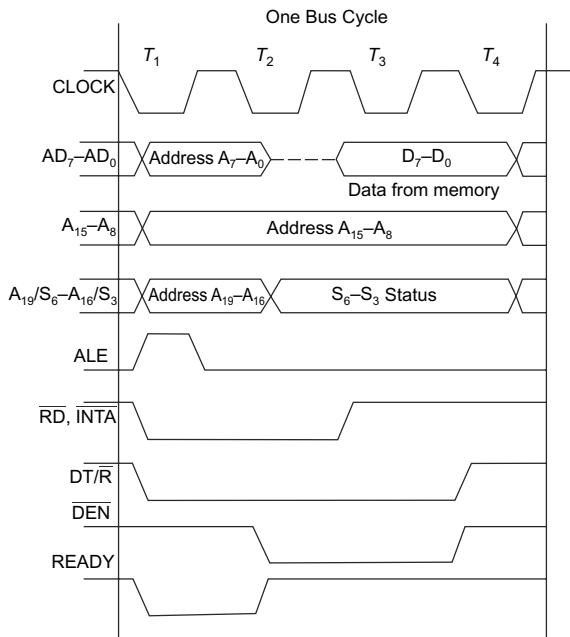


Fig. 5.26 Memory read bus cycle of 8088

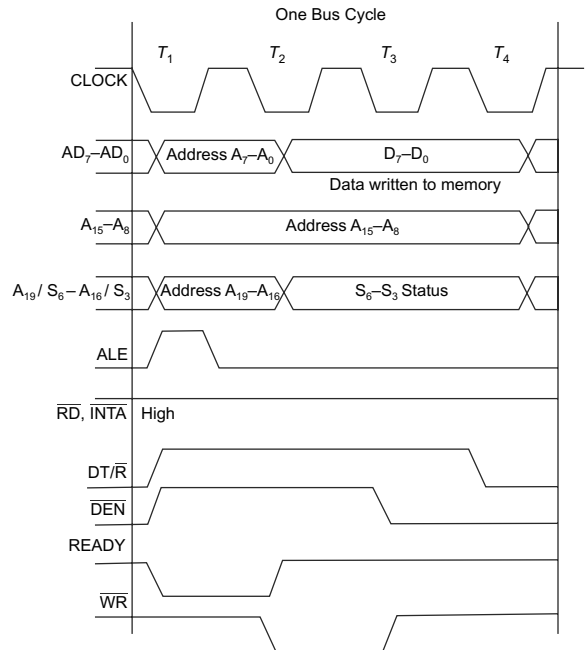


Fig. 5.27 Memory write bus cycle of 8088

the timing diagram for memory read operation of 8088 microprocessor. After T_4 , the next bus cycle will be started.

Figure 5.27 shows the timing diagram of 8088 for memory write bus cycles. In memory cycle, data will be available on the data bus during T_2 , T_3 and T_4 and the status signals are valid for T_2 , T_3 and T_4 durations. After T_4 address/data bus, AD_7-AD_0 are tristated.

5.11 DEMULTIPLEXING OF THE SYSTEM BUS IN 8086 AND 8088 MICROPROCESSORS

In 8086/8088 processors, there are three buses: address, data and control buses. The address/data buses are operated in time-multiplexed mode. The address bus is required to locate memory and I/O devices for data transfer through memory and I/O read or write cycles. The data bus is used to transfer data from microprocessor to memory/I/O devices or vice versa. The control bus provides control signals to memory/I/O devices for data-transfer operations.

5.11.1 Demultiplexing of System Bus in 8086

The 8086 microprocessor has time-multiplexed 16-bit address/data bus $AD_{15}-AD_0$ and 4-bit address/status bus $A_{19}/S_6-A_{16}/S_3$. The ALE signal is used to latch the address of 8086. Usually, latch ICs are available with eight separate latches. Therefore, three latch ICs should be used for demultiplexing 20-bit address lines. Figure 5.28 shows the circuit diagram for latching 20-bit address lines using three 74LS373 latch ICs. In this arrangement, two ICs are fully utilised and one latch is partially used.

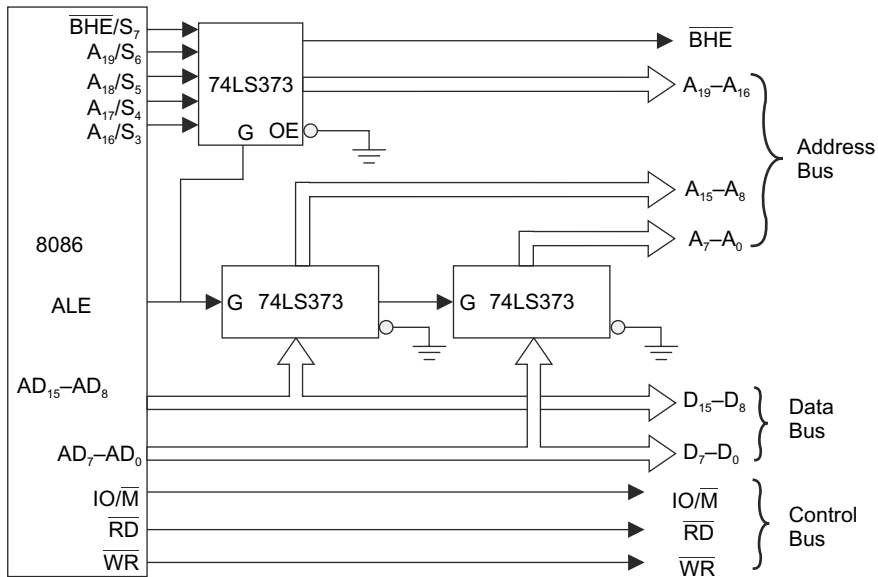


Fig. 5.28 Demultiplexing of 20-bit address bus in 8086 processor

The 8086 microprocessor has 16-bit time-multiplexed data bus which is available as Address/Data bus, $AD_{15}-AD_0$. The data bus is always separated from the address bus by using 74245 buffers as depicted in

Fig. 5.29. The data bus is bi-directional and data can be transferred from microprocessor to memory and memory to microprocessor for memory write and read operations respectively. The control signals \overline{DEN} and DT/\overline{R} represent the presence of data on the data bus and directional flow of data. These signals are used to connect the chip enable CE and directional pins of 74245 buffers. While \overline{DEN} is low, the data is available on the multiplexed bus.

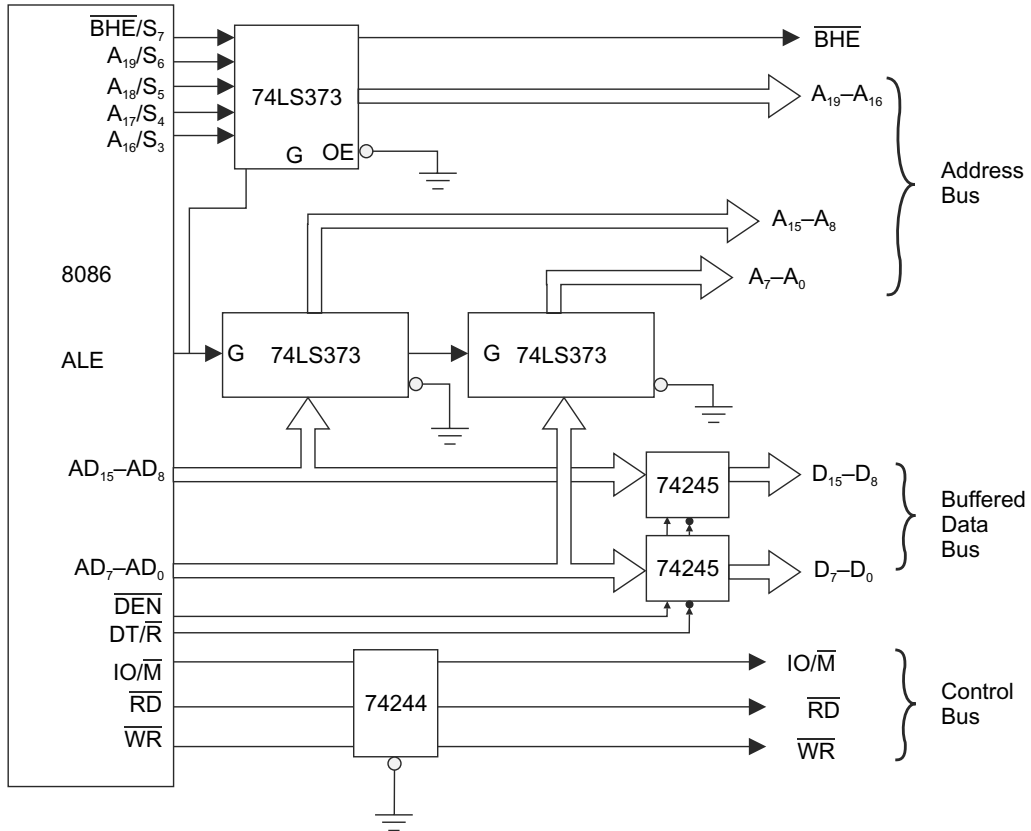


Fig. 5.29 Demultiplexing of fully buffered system bus in 8086 processor

5.11.2 Demultiplexing Of System Bus In 8088

In 8086 $A_{19}/S_6-A_{16}/S_3$, $AD_{15}-AD_0$ and \overline{BHE}/S_7 are multiplexed but in 8088 only A_7-A_0 and $A_{19}/S_6-A_{16}/S_3$ are time multiplexed. The demultiplexing of address bus of the 8088 microprocessor is shown in Fig. 5.30. Here, 74LS373 latches are used to demultiplex address/data bus. When $ALE = 1$, the latches pass the inputs to the outputs. After one clock time T_1 , ALE becomes logic 0. $A_{19}/S_6-A_{16}/S_3$ are connected into the top latch and A_7-A_0 are connected into the bottom latch. These address connections can address 1 MB memory space. The 8088 systems require only one data buffer due to the 8-bit data bus as depicted in Fig. 5.31.

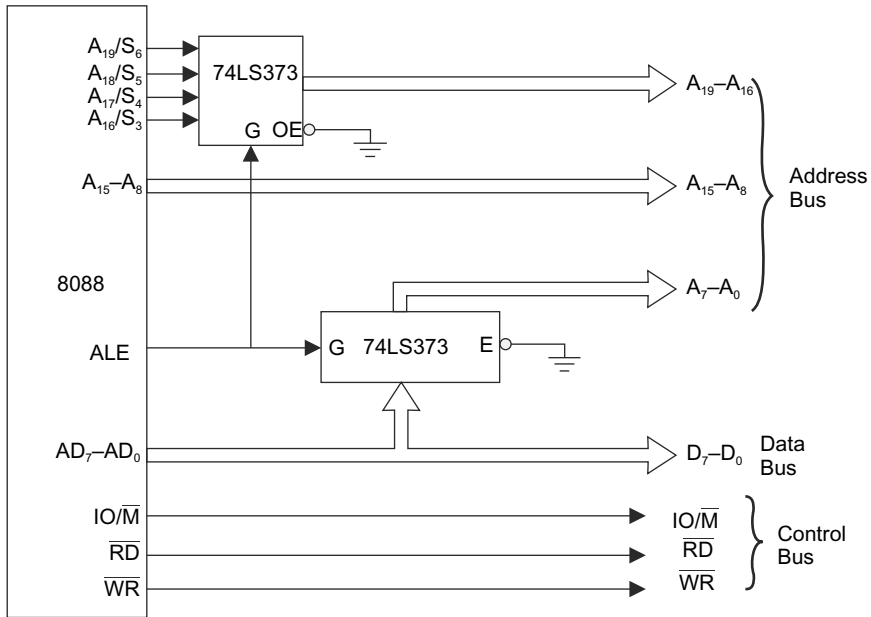


Fig. 5.30 Demultiplexing of address bus in 8088 processor

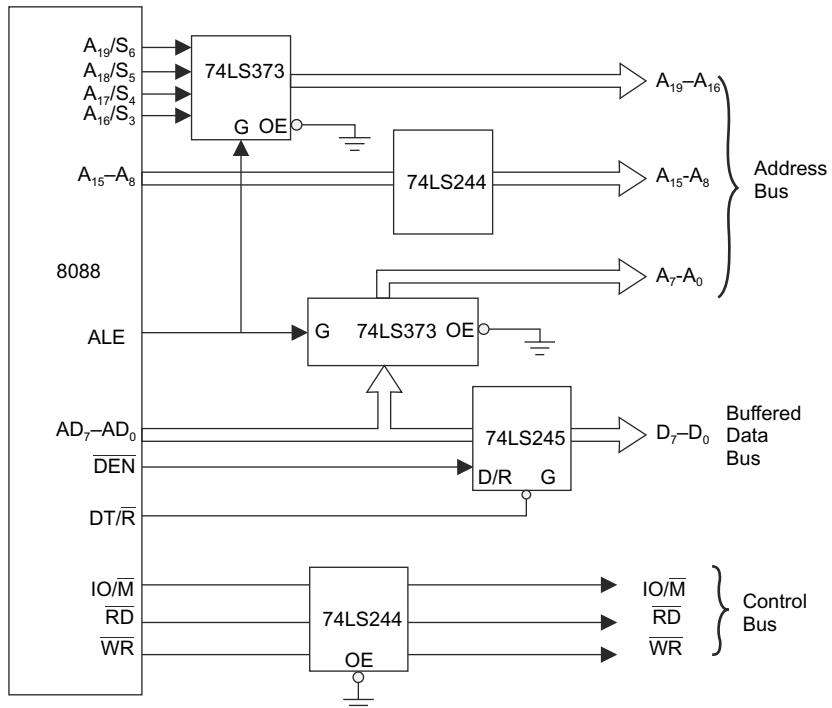


Fig. 5.31 Demultiplexing of fully buffered system bus in 8088 processor

5.12 SOME IMPORTANT ICs: 8284A, 8286/8287, 8282/8283, AND 8288

The 8086 processor requires a clock signal with very fast rise and fall times which is about 10 ns and a duty cycle of 33%. For proper operation, the 8086 processor RESET signal must be synchronised with clock signal and persist for four T states. Actually, the 8284A clock generator IC meets the requirement of CLOCK and RESET signals.

Buses of 8086 microprocessors require buffering techniques for reliable data transmission. When any receiver receives data, it requires a dc load current from the transmitter. Due to this load, the high-level output voltage V_{OH} will be reduced and low-level output voltage V_{OL} will be increased. Hence noise immunity of the system will be reduced. In addition, each receiver has an ac load current due to its input capacitance. Truly, the input capacitor must be charged and discharged whenever the transmitter output state changes from logic level 1 to 0 or logic level 0 to 1. As the propagation delay time of the transmitted signal increases, the time availability to the memory and I/O devices will be reduced. Therefore, to minimize the dc as well as ac loading effect, buffers are required for microprocessor buses. Usually, 8286/8287 buffer ICs are commonly used in microprocessor-based systems.

The 8282 and 8283 are 8-bit bipolar latches with three-state output buffers. These ICs can be used as latches, buffers or multiplexers. The 8282 provides non-inverted outputs whereas the 8283 inverts the input data at its output. In the maximum-mode operation, 8086 requires system control signals such as \overline{MRDC} , \overline{MWTC} , \overline{AMWC} , \overline{IORC} , \overline{IOWC} and \overline{AIOWC} . The 8288 bus controller chip generates all the control signals.

To design any microprocessor-based system, 8284A clock generator, 8286/8287 buffers, 8282/8283 I/O ports and the 8288 bus controller are used. In minimum- and maximum-mode operation, the first three ICs are extensively used but the 8288 bus control is used only in maximum mode operation. In this section, the detailed operation of 8284A clock generator, 8286/8287 buffers, 8282/8283 I/O ports and the 8288 bus controller are discussed in detail.

5.12.1 Clock Generator 8284A

During fetch and execute instructions, the 8086 and 8088 processors require clock pulse which has about 10 ns rise and fall times. The logic 0 level of the clock is 0.5 V to 0.6 V, logic-1 level is about 3.9 V to 5 V and the clock duty cycle is about 33%. The 8086 processor has no clock generator inside the chip. So, an external clock generator IC must be connected with the processor. The 8284A is a clock generator IC and it is a supporting component to the 8086/8088 processors. The 8284A IC has the following features or additional basic functions such as clock generation, RESET synchronization, READY synchronization, and a TTL level peripheral clock signal. The operation of 8284A IC has been explained in this section.

Operation of 8284A

The internal logic of the 8284A clock generator is depicted in Fig. 5.32. The upper half of the logic diagram represents the clock and reset synchronization section of the 8284A clock generator. It is depicted in Fig. 5.32 that the crystal oscillator has 2 inputs: X_1 and X_2 . When a crystal is connected to X_1 and X_2 terminals, the oscillator generates a square-wave signal and its frequency is same as the crystal frequency.

The output square-wave signal is fed to an AND gate and it is inverted by using an inverting buffer to generate the OSC output signal. The OSC signal can be used as an EFI input to other 8284A clock generators. When F/\overline{C} is a logic 0, the output of the AND gate is fed to divide-by-3 synchronous counters. If F/\overline{C} is a logic 1 then EFI is steered through to the counter.

The output of the divide-by-3 synchronous counters generates the ready signal for synchronization. The

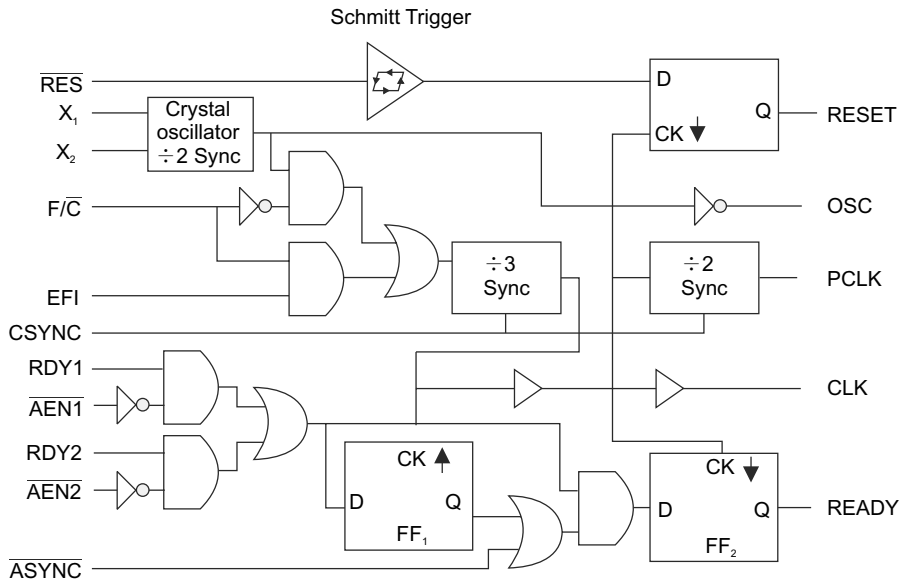


Fig. 5.32 Block diagram of 8284 clock generator

CLK signal is buffered before output from the clock generator. Another divide-by-2 synchronous counter generates the PCLK signal to the 8086/8088 microprocessor. When the output of the first ÷3 synchronous counters is fed to the second ÷2 synchronous counters, the two cascaded counters generate ÷6 outputs at PCLK. Figure 5.33 shows the connection between 8284A and the 8086/8088 processor. Usually, F/\bar{C} and CSYNC are connected with ground to select the crystal oscillator. Then a 15 MHz crystal generates the normal 5 MHz CLK clock signal and a 2.5 MHz peripheral clock signal PCLK.

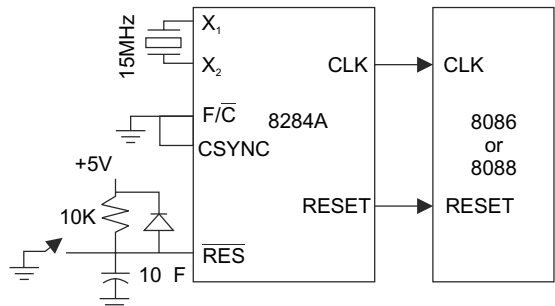


Fig. 5.33 CLK generator 8284A and 8086/88 microprocessor

The reset section of the 8284A consists of Schmitt trigger buffered and a D type flip-flop. The D flip-flop ensures that the timing requirements of the 8086/8088 RESET input are met. This circuit applies the RESET output signal of clock generator is fed to the microprocessor as shown in Fig. 5.32, and it is active on the negative edge the clocks. Hence, the reset section meets the timing requirements of the 8086/8088 microprocessor.

Pin Functions of 8284A

The 8284A is an 18 pin IC which is specifically designed for 8086/8088 microprocessors. The pin diagram of the chip is shown in Fig. 5.34. In this section, the functions of pins are explained.

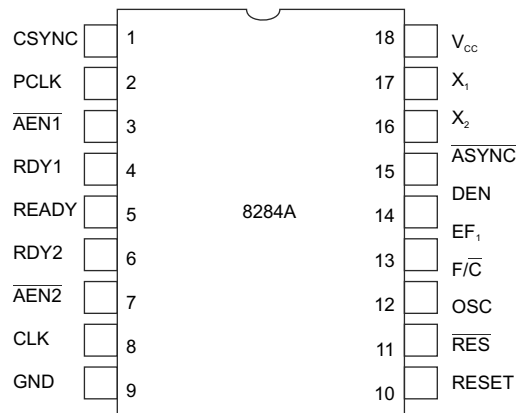


Fig. 5.34 Pin diagram of the 8284 A

- ✓ **X_1 and X_2 Crystal Inputs** These pins are connected to an external crystal which is used as a clock frequency source of the clock generator. The external crystal clock frequency will be about three times the required frequency. If the required frequency is 5 MHz, the crystal frequency will be 15 MHz.
- ✓ **CLK** CLK is an output pin that provides the clock (CLK) signal which is used as input signal to the 8086/8088 microprocessor system. The CLK pin has an output signal with 33% duty cycle as required by the 8086/8088.
- ✓ **EFI (External Frequency Input)** This is an alternate clock input when F/\overline{C} pin is pulled high. The externally generated clock signal is supplied to EFI whenever the F/\overline{C} pin is high.
- ✓ **PCLK (Peripheral Clock)** This is a clock output signal that is one sixth of the crystal. PCLK is half of the clock frequency and has a 50% duty cycle. The PCLK output signal can be used as a clock signal to the peripheral equipments in 8086/8088 system.
- ✓ **OSC (Oscillator Output)** This is an oscillator-output signal which is running at crystal or EFI frequency. This signal can be used to provide clock signal at EFI to the other 8284 clock generators in some multiple processor system.
- ✓ **F/\overline{C} (Frequency/Crystal)** The voltage on this pin determines the clocking source for the 8284A. If this input pin is high, an external clock at EFI is selected. While it is low, the internal crystal oscillator provides the clock frequency signal.
- ✓ **CSYNC (Clock Synchronization)** This pin is used for synchronization of clock signals in a multiprocessor system where all processors receive the clock at EFI. If the internal crystal oscillator is used, this pin must be grounded. When CSYNC is high, the 8284A clock generator stops working.
- ✓ **\overline{RES} (Reset Input)** To reset the 8086 processor, 8284A clock generator should send the RESET signal. Generally, this pin is connected to an RC network for generating RESET signal at power on.
- ✓ **RESET (Reset Output)** This signal is connected to the 8086/8088 RESETs input pin. The RESET signal must be synchronised with the clock.
- ✓ **RDY1, RDY2** The slow memory or I/O devices can request for extension of bus cycles using RDY1 or RDY2 pins. These two wait-state ready inputs are provided to support a multibus 8086/8088-based system.
- ✓ **READY** The READY output pin connects to the 8086 READY input which enables the bus cycle period insertion between T_3 and T_4 . The 8086 READY signal must be synchronized with the RDY1 and RDY2 inputs.
- ✓ **\overline{ASYNC} (Ready Synchronization Select)** This input pin is used to select either one or two stages of synchronization for the RDY1 and RDY2 inputs. If it is low, one level is selected. When it is high, two levels of synchronization are selected.
- ✓ **$\overline{AEN}_1, \overline{AEN}_2$** Two ready inputs RDY1, RDY2 have been provided in the 8284A to support the multibus system. The 8086 CPU may be connected to two separate system buses, on which data transfer takes place. The memory or I/O devices of any system bus may like to insert wait states. Hence, each system bus should have its own ready line. \overline{AEN}_1 and \overline{AEN}_2 are provided to arbitrate bus priorities whenever RDY1 and RD2 are active. The 8284A responds to RDY1 when \overline{AEN}_1 is low. In the same way, clock generator responds to RDY2 if \overline{AEN}_2 is low.
- ✓ **V cc (Power Supply Input)** This pin is connected to + 5 V $\pm 10\%$.
- ✓ **GND (Ground)** This pin must be grounded.

5.12.2 Bi-directional Bus Transceiver 8286/8287

The intel 8286 and 8287 are bi-directional system bus buffers-cum-drivers. Figure 5.35 shows the pin diagrams of 8286 and 8287 transceivers.

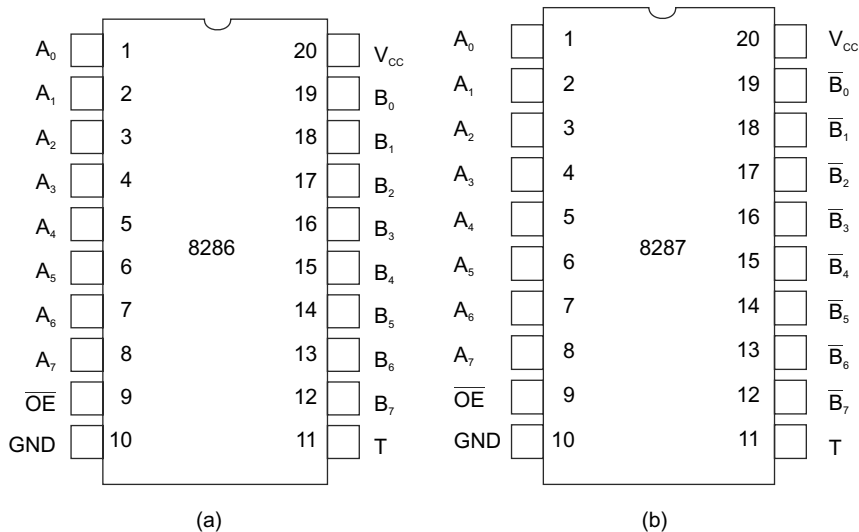


Fig. 5.35 Pin diagram of bi-directional bus transceivers (a) 8286(b) 8287

- ✓ **A₇-A₀ Bidirectional Tristate** These lines are connected to microprocessor data/address bus.
- ✓ **B₇-B₀ Bidirectional Tristate** These lines are connected to system bus.
- ✓ **OE** Output Enable
- ✓ **T** Direction Select
- ✓ **V_{cc}** Input Power Supply, + 5 V
- ✓ **GND (Ground)** This pin is grounded.

5.12.3 8-Bit Input-Output Port 8282/8283

The Intel 8282 and 8283 are unidirectional latch buffers. The difference between the 8282 and 8283 is that the IC 8282 does not change the data and the IC 8283 inverts the input data. Figure 5.36 shows the pin diagram of 8282 and 8283 input output ports. The pin functions are given below:

- ✓ **DI₇-DI₀** Data input
- ✓ **DO₇-DO₀** Data output
- ✓ **OE** Output enable
- ✓ **STB** Input data strobe. If STB is high, the data on output pins track the data on input pins. On high to low transition of STB, the data is latched. The data remains unchanged when STB is low. The data is latched internally till OE is low. When OE is low, the data is put on output lines. The 8282 outputs the data unaltered and the 8283 inverts the data.

- ✓ **V_{cc}** Input Power Supply, + 5 V
- ✓ **GND (Ground)** This pin is grounded.

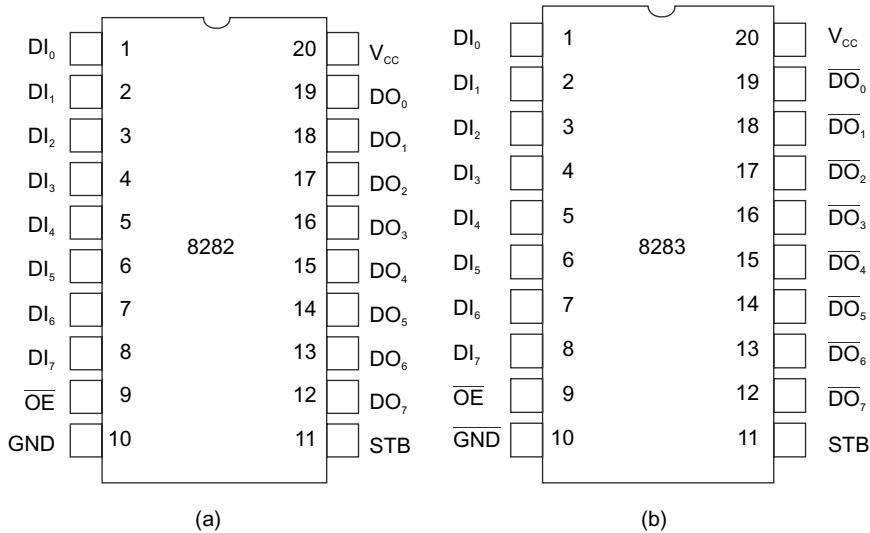


Fig. 5.36 Pin diagram of 8-bit I/O port: (a) 8282 (b) 8283

5.12.4 8288 Bus Controller

Figure 5.37 shows the 8288 bus controller which is used in the maximum-mode operation of 8086 CPU. This IC receives four inputs such as $\bar{S}_2, \bar{S}_1, \bar{S}_0$ status signals and CLK from 8086. There are two sets of output command signals.

The first set of output command signals are the MULTIBUS command signals. These are the conventional $\overline{MEMR}, \overline{MEMW}, \overline{IOR}$ and \overline{IOW} control signals. These signals are renamed as $\overline{MRDC}, \overline{MWTC}, \overline{IORC}$ and \overline{IOWC} respectively. Here, C stands for command. \overline{AMWC} and \overline{AIOWC} are advanced memory and I/O write commands. These outputs are enabled one clock cycle earlier than the normal write commands. Some memory and I/O devices require this wider write pulse width.

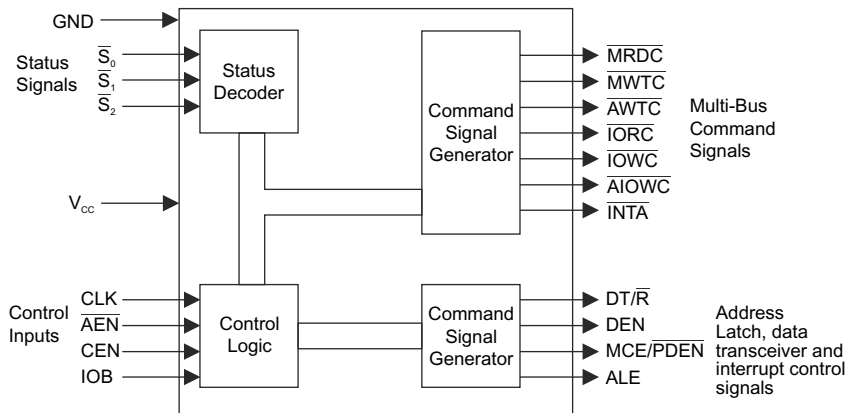


Fig. 5.37 Functional diagram of 8288 bus controller

The second set of output signals of the 8288 are the bus control signals DT/\overline{R} , DEN, ALE and MCE/\overline{PDEN} . MCE/\overline{PDEN} is an output signal which has two functions depending on the 8288's mode of operation such as I/O bus control or system bus control.

The three 8288 control inputs CEN, IOB and \overline{AEN} determine the operating mode as given in Table 5.12. When CEN and IOB are high or of logic level 1, the 8288 operates in the I/O bus mode and the MCE/\overline{PDEN} output acts as a peripheral data enable. The function of MCE/\overline{PDEN} is identical to DEN but it is active only during I/O instructions. This allows the 8288 to control two sets of buses such as the normal system buses and a special I/O bus dedicated to the processor.

During the system-bus mode, the control signals are active only while the address enable signal \overline{AEN} and IOB inputs are low. In this mode of operation, several 8288s IC and 8086 processors can be interfaced to the same set of bus lines. The bus mediator selects the active processor after enabling only one 8288 bus controller through an \overline{AEN} signal. The MCE/\overline{PDEN} signal becomes MCE (Master Cascade Enable). In this mode, the MCE/\overline{PDEN} signal is used to read the address from a master priority interrupt controller, PIC.

Table 5.12 Operating modes of 8288 bus controller

CEN	IOB	\overline{AEN}	Operations
0	x	x	All command outputs and the DEN and PDEN outputs are disabled or open-circuited
1	0	0	System bus mode and all control signals are active. The bus is free for use and $MCE/\overline{PDEN}=MCE$
1	0	1	System-bus mode but all control signals are disabled. The bus is busy, and is controlled by another bus master
1	1	x	I/O bus mode; all control lines are enabled and $MCE/\overline{PDEN} = \overline{PDEN}$

The pin diagram of the 8288 bus controller is shown in Fig. 5.38. The functions of the pins are described in this section.

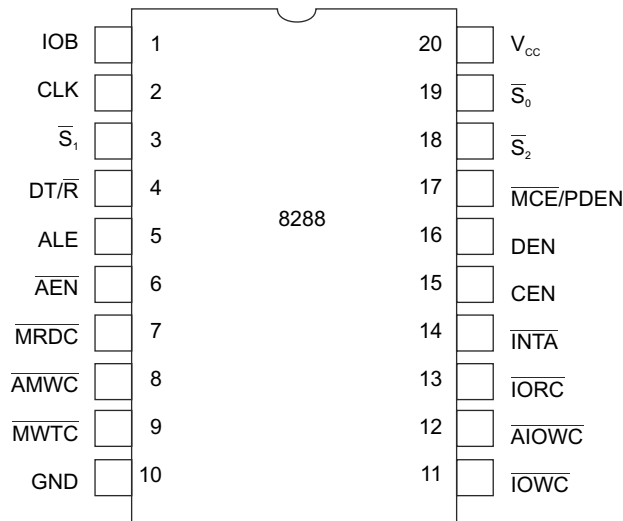


Fig. 5.38 Pin diagram of 8288 bus controller

✓ $\overline{S}_2, \overline{S}_1, \overline{S}_0$ (Status Input Signals) These are bus cycle status signals. These are decoded and control signals are generated.

- ✓ **CLK** This is an input signal. It is connected to CLK output of the clock generator 8284.
- ✓ **\overline{AEN} , CEN, IOB** These are bus priority and mode control signals. \overline{AEN} (bus priority control/enable), CEN(Command Enable) , and IOB (mode control) signals are used to generate various control signals.
- ✓ **\overline{MRDC} (Memory Read Control Signals)** These command signals are used to load the content of memory location on the data bus.
- ✓ **\overline{MWTC} (Memory Write Control Signals)** These command signals are used to store the available data on the data bus to the specified memory location.
- ✓ **\overline{IORC} (I/O Read Control Signals)** The I/O device is able to put the available data of the addressed port on the data bus.
- ✓ **\overline{IOWC} (I/O Write Control Signals)** The I/O device is able to accept the available data port on the data bus and send to the addressed port.
- ✓ **\overline{AMWC} (Advance Memory Write Control Signal)** This signal is activated one clock period earlier than \overline{MWTC} .
- ✓ **\overline{AIOWC}** This signal is activated one clock period earlier than \overline{IOWC} .
- ✓ **$\overline{MCE/PDEN}$** Cascade/Peripheral data Enable. This signal is used in Priority Interrupt Controller 8259A.
- ✓ **\overline{INTA} (Interrupt Acknowledge)** This is used as output signal during two interrupt acknowledge bus cycles and is used as memory read control signal.
- ✓ **ALE** Address Latch Enable signal.
- ✓ **$\overline{DT/\overline{R}}$** Data Direction Control signal.
- ✓ **DEN** Data buffer control signal.
- ✓ **V_{cc}** Power Supply Input + 5 V.
- ✓ **GND (Ground)** This pin is connected system ground.

SUMMARY

- In this chapter, the architectures of the 8086 and 8088 processors have been presented. The functional details such as registers, flags, segment memory and pin descriptions of 8086 and 8088 have been discussed.
- The minimum-mode and maximum-mode operations of 8086 microprocessors are explained briefly. The timing diagram for memory read, write, I/O read and write operations are discussed elaborately.
- The functions of 8284A clock generator, 8288 bus controller, 8286 bi-directional bus transceiver, 8-bit input-output port 8282 and comparisons between 8085, 8086 and 8088 are also incorporated.

MULTIPLE-CHOICE QUESTIONS

- 5.1 The 8086 has a
 (a) 16-bit data bus and 20-bit address bus
 (b) 8-bit data bus and 20-bit address bus
 (c) 16-bit data bus and 16-bit address bus
 (d) 8-bit data bus and 16-bit address bus
- 5.2 The 8088 has a
 (a) 16-bit data bus and 20-bit address bus
 (b) 8-bit data bus and 20-bit address bus
 (c) 16-bit data bus and 16-bit address bus
 (d) 8-bit data bus and 16-bit address bus
- 5.3 The 16-bit register of 8086 consists of
 (a) 16 flags (b) 8 flags
 (c) 9 flags (d) 7 flags
- 5.4 The instruction queue of 8086 consists of
 (a) 6 data (b) 8 data
 (c) 4 data (d) 10 data
- 5.5 The Instruction queue of 8088 consists of
 (a) 6 data (b) 8 data
 (c) 4 data (d) 10 data
- 5.6 The 8086 has
 (a) 6 memory segments
 (b) 8 memory segments
 (c) 4 memory segments
 (d) 10 memory segments
- 5.7 The physical memory of 8086 is
 (a) 1 MB (b) 64 KB
 (c) 2 MB (d) 4 MB
- 5.8 The memory map of 8086 is
 (a) 0000H to FFFFH (b) 00000H to FFFFFH
 (c) 0000H to FFFFH (d) 0000H to FFFFH
- 5.9 The segment memory capacity of 8086 is
 (a) 1 MB (b) 64 K
 (c) 2 MB (d) 4 MB
- 5.10 Control signals are generated from \overline{RD} , IO/\overline{M} , and \overline{WR}
 (a) \overline{MRD} , \overline{MWR} , \overline{IORD}
 (b) \overline{MWR} , \overline{IRD} , \overline{IOWR}
 (c) \overline{MWR} , \overline{IRD} , \overline{IOWR}
 (d) \overline{MRD} , \overline{MWR} , \overline{IORD} , \overline{IOWR}
- 5.11 8086 has the following units:
 (a) EU and BIU (b) EU only
 (c) BIU only (d) CU and BIU
- 5.12 Physical address of 8086 is
 (a) 8-bit (b) 16-bit
 (c) 20-bit (d) 32-bit
- 5.13 Clock frequency of 8086 and 8088 is
 (a) 5–10 MHz (b) 2–3 MHz
 (c) 1–3 MHz (d) 2–5 MHz
- 5.14 Clock generator has output frequency
 (a) 15–10 MHz (b) 2–3 MHz
 (c) 1–3 MHz (d) 2–5 MHz
- 5.15 The 8086/88 can be operated in single step when
 (a) TF set (b) DF set
 (c) SF set (d) AF set
- 5.16 The physical address when DS = 2345H and IP = 1000H is
 (a) 23450H (b) 24450H
 (c) 12345H (d) 2345H

SHORT-ANSWER-TYPE QUESTIONS

- 5.1 What are the general-purpose registers of the 8086 and 8088 microprocessors?
- 5.2 What are the functions of index registers, pointer registers and instruction pointers?
- 5.3 Write the differences between (a) 8085 and 8086, and (b) 8086 and 8088.
- 5.4 What is the purpose of the queue? How many bytes can be stored in the queue of 8086 and 8088?

- 5.5 What is pipelined architecture? How is it implemented in 8086?
- 5.6 Explain the concept of segmented memory. What are its advantages?
- 5.7 How do you select the minimum and maximum modes of operation in the 8086/8088?
- 5.8 Determine the physical address when CS = 6000H and offset address = 2300H.

REVIEW QUESTIONS

- 5.1 Draw the schematic block diagram of 8086 and explain the function of each block.
- 5.2 Define logical address and physical address. What are the differences between the logical and physical memory of the 8086?
- 5.3 What is the content of DS and IP to locate the physical address location 35678H? Assume the value of offset address.
- 5.4 Why is the 8086 memory organized into two banks of even and odd addresses? Explain how the even and odd bank are selected using the BHE and A₀ signals.
- 5.5 Draw the pin diagram of the 8086 microprocessor. Explain the function of the following pins of 8086: (i) ALE (ii) NMI (iii) INTR (iv) HOLD (v) HLDA (vi) BHE (vii) LOCK (viii) M/\overline{IO} (ix) \overline{DEN} (x) DT/\overline{R}
- 5.6 Draw the minimum-mode system configuration of 8086 with memory and I/O interface and give a list about the functions performed by each chip.
- 5.7 Draw bus-cycle timing diagrams for memory read and write operations in minimum mode and explain briefly.
- 5.8 Draw the maximum-mode system configuration of 8086 with memory and I/O interface and a list the functions performed by each chip.
- 5.9 Draw bus-cycle timing diagrams for memory read and write operation in the maximum mode and explain briefly.
- 5.10 Draw the schematic block diagram of 8088 and explain the function of each block. Write the differences between 8086 and 8088.
- 5.11 What is demultiplexing of buses in 8086? Explain demultiplexing of address bus in 8086 and 8088.
- 5.12 What are the functions of the 8284 clock generator in the 8086/8088 systems? If a crystal of frequency 15 MHz frequency is attached to the 8284, what will be the frequency of signals at CLK and PCLK pins?

Answers to Multiple-Choice Questions

-
- 5.1 (a) 5.2 (b) 5.3 (c) 5.4 (a) 5.5 (c) 5.6 (c) 5.7 (a) 5.8 (b) 5.9 (b)
- 5.10 (d) 5.11 (a) 5.12 (c) 5.13 (a) 5.14 (d) 5.15 (a) 5.16 (b)

Chapter 6

Instruction Set and Addressing Modes of the 8086 Microprocessor

6.1 INTRODUCTION

An instruction is a basic command given to a microprocessor to perform a specified operation with given data. Each instruction has two groups of bits. One group of bits is known as *operation code (opcode)*, which defines what operation will be performed by the instruction. The other field is called operand, which specifies data that will be used in arithmetic and logical operations. The addressing mode is used to locate the operand or data. There are different types of addressing modes depending upon the location of data in the 8086 processor.

The instruction format should have one or more number of fields to represent instruction. The first field is called operation code or opcode fields, and other fields are known as operand fields. The microprocessor executes the instruction based on the information of opcode and operand fields.

In this chapter, the general instruction format and different addressing modes of 8086/8088 processor along with examples are discussed. All types of instructions with examples are discussed elaborately. This chapter creates a background for assembly-language programming in 8086/8088 processor.

6.2 ADDRESSING MODES

An instruction is divided into operation code (opcode) and operands. The opcode is a group of bits which indicates what operation can be performed by the processor. An operand is also known as data (datum) and it can identify the source and destination of data. The operand can specify a register or a memory location in any one of the memory segments or I/O ports. Figure 6.1 shows a general instruction format which consists

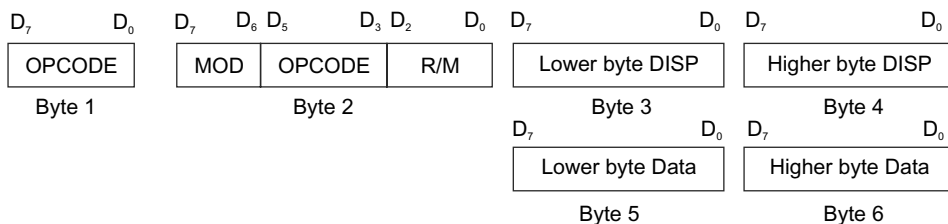


Fig. 6.1 General 8086 instruction format

of six bytes. Some instructions have only an opcode and such instructions are called single-byte instructions. Some instructions contain one- or two- or three- or four-byte operands. The detailed operation of instruction sets is explained in Section 6.3.

There are different ways to specify an operand. Each way of how an operand can be specified is called an addressing mode. The different addressing modes of 8086 microprocessors are as follows:

- ✦ Immediate addressing
- ✦ Register addressing
- ✦ Memory addressing
- ✦ Branch addressing

6.2.1 Immediate Addressing

In this mode of addressing, the 8-bit or 16-bit operand is a part of the instruction. For example, MOV AX, 4000H. In this instruction, the data 4000H can be loaded to the AX register immediately. Some other examples are given below:

MOV BX, 7000H; load 7000H in BX register
MOV CX, 4500H; store 4500H in CX register

6.2.2 Register Addressing

In the 8086 microprocessor, some instructions are operated on the general-purpose registers. The data is in the register specified by the instruction. The format for register addressing is

MOV Destination, Source

In this instruction, the data from the source register can be copied into the destination register. The 8-bit registers (AL, AH, BL, BH, CL, CH, DL, DH) and 16-bit registers (AX, BX, CX, DX, SI, DI, SP, BP) may be used for this instruction. The only restriction is that both operands must be of the same length. For example,

MOV AL, BL; Copies the value of BL into AL
MOV AX, BX; Copies the contents of BX into AX

6.2.3 Memory Addressing

Memory addressing requires determination of physical address. The physical address can be computed from the content of segment address and an effective address. The segment address identifies the starting location of the segment in the memory and effective address represents the offset of the operand from the beginning of this segment of memory. The 20-bit effective address can be made up of base, index, and displacement. The basic formula for the 16-bit effective address (EA) and the 20-bit physical address (PA) is given below:

16-bit EA = Base + Index + Displacement
20-bit PA = Segment × 10 + Base + Index + Displacement

Memory addressing has the following combinations:

- ✦ Direct addressing
- ✦ Register indirect addressing
- ✦ Based addressing
- ✦ Indexed addressing
- ✦ Based Indexed addressing
- ✦ Based Indexed with displacement addressing

Direct Addressing

In this mode of addressing, the instruction operand specifies the memory address where data is located. This addressing mode is similar to the immediate addressing mode, but the opcode follows an effective address instead of data. This is the most common addressing mode. The displacement-only addressing mode consists of an 8-bit or 16-bit constant that specifies the offset of the actual address of the target memory location. For example, `MOV AX, [5000]` copies 2 bytes of data starting from memory location $DS \times 10 + 5000H$ to the AX register. The lower byte is at the location $DS \times 10 + 5000H$ and the higher byte will be at the location $DS \times 10 + 5001H$. Another example is `MOV AL, DS:[5000H]`. In this instruction, the content of the memory location $DS \times 10 + 5000H$ loads into the AL register.

The instruction `MOV DS:[2000H], AL` means that the content of the AL register will move to memory location $DS \times 10 + [2000H]$. The computation of memory location for the operand is illustrated in Fig. 6.2. In this figure, the effective address (EA) is 2000H and the physical address (PA) is $PA = DS \times 10 + EA = 4000 \times 10 + 2000 = 42000$. The physical address (PA) computation for other segment registers with same effective address is given below:

$$PA = CS \times 10 + EA, PA = SS \times 10 + EA, \text{ and } PA = ES \times 10 + EA.$$

Generally, all displacement values or offsets are added with the data segment to determine the physical address. If something other than a data segment is required, we must use a segment override prefix before the address. For example, to access memory location 4000H in the Stack Segment (SS), the instruction will be `MOV AX, SS:[2000H]`. Similarly, to access the memory location in the Extra Segment (ES), the instruction will be written as `MOV AX, ES:[2000H]`.

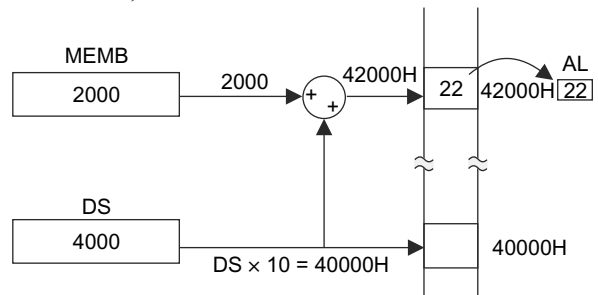


Fig. 6.2 Direct memory addressing

Register Indirect Addressing

This instruction specifies a register containing an address, where data is located. The effective address of the data is in the base register BX or an index register that is specified by the instruction. This addressing mode works with index registers SI, DI, and base registers BX and BP registers.

The examples of this addressing mode in the 8086 microprocessor are as follows:

`MOV AL, [BX]`

`MOV AH, [DI]`

`MOV AL, [SI]`

`MOV AH, [BP]`

The BX, BP, SI, or DI registers are using the DS segment by default. The base pointer uses stack segment by default. The segment override prefix symbols are used to access data in different segments. The examples of segment override instructions are as follows:

`MOV AL, CS:[BX]`

`MOV AL, DS:[BP]`

`MOV AL, SS:[SI]`

`MOV AL, ES:[DI]`

The effective address EA may either be in a base register (BX or BP) or in an index register (SI and DI). The physical address can be computed based on contents of segment register, BX, BP, SI and DI registers as given below: $PA = CS \times 10 + BX$, $PA = DS \times 10 + BP$, $PA = SS \times 10 + DI$, and $PA = ES \times 10 + SI$.

The general physical address expression for register indirect memory operand is depicted in Fig. 6.3. The content of BX is 1000H and CS is 2000. Then the physical address is $CS \times 10 + BX = 2000 \times 10 + 1000 = 21000H$. After executing the MOV AL, [BX] instruction, the contents of the memory location 21000H is 44H which will be stored in the AL register.

Based Addressing

The 8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP), the resulting value is a pointer to the location where the data resides. The effective address in based addressing mode is obtained by adding the direct or indirect displacement to the contents of either the base register BX or the base pointer BP. The effective address and physical address computation are given below:

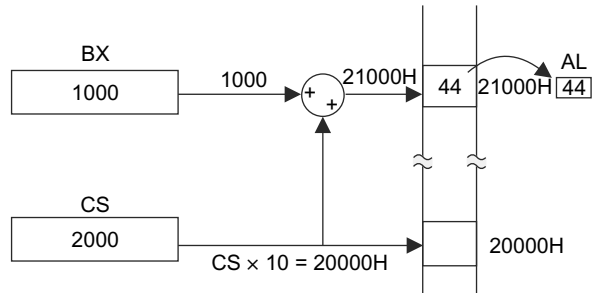


Fig. 6.3 Register indirect addressing

- EA = BX + 8-bit displacement EA = BP + 8-bit displacement
- EA = BX + 16-bit displacement EA = BP + 16-bit displacement
- PA = Segment \times 10 + BX + 8-bit displacement, PA = Segment \times 10 + BP + 8-bit displacement
- PA = Segment \times 10 + BX + 16-bit displacement, PA = Segment \times 10 + BP + 16-bit displacement

Segment will be any one of the segments CS, DS, SS and ES. Figure 6.4 shows the physical address computation in base addressing mode. When 16-bit displacement DISP = 0025H, the contents of the BX register is 0500H and the contents of the DS register is 4000H, the physical address = $DS \times 10 + BX + DISP = 4000H \times 10 + 0500 + 0025 = 40525H$. After execution of MOV AL, DS: [BX+DISP] instruction, the contents of the memory location 40525H will be copied into the AL register. The examples of base addressing mode instructions in the 8086 microprocessor are

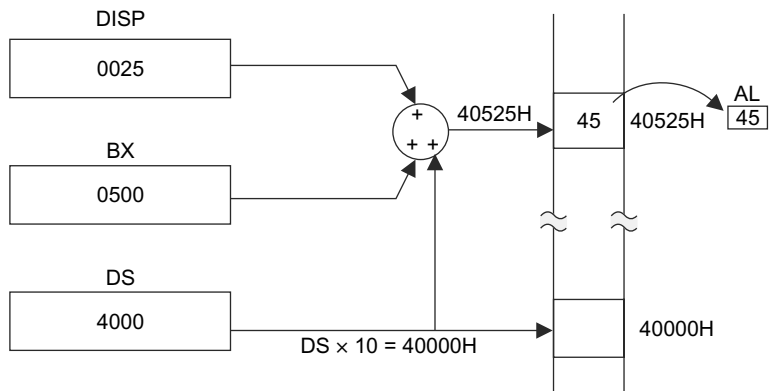


Fig. 6.4 Base addressing

Indexed Addressing

These addressing modes can work similar to the based addressing mode. The 8-bit or 16-bit instruction operand is added to the contents of an index register (SI or DI), and the resulting value is a pointer to the location where the data resides.

The displacement value is used as a pointer to the starting point of an array of data in memory and the contents of the specified register is used as an index. The EA and PA in the indexed addressing are as follows:

EA = SI + 8-bit displacement

EA = SI + 16-bit displacement

PA = Segment × 10 + SI + 8 bit displacement

PA = Segment × 10 + SI + 16 bit displacement

EA = DI + 8-bit displacement

EA = DI + 16-bit displacement

PA = Segment × 10 + DI + 8 bit displacement

PA = Segment × 10 + SI + 16 bit displacement

Segment will be any one of segment registers (CS, DS, SS and ES).

The index addressing modes generally involve BX, SI, and DI registers with the data segment. The [BP+DISP] addressing mode uses the stack segment by default. In the register indirect addressing modes, the segment override prefixes can be used to specify different segments. The examples of these instructions are as follows:

MOV AL, SS: [BX + DISP]

MOV AL, ES: [BP + DISP]

MOV AL, CS: [SI + DISP]

MOV AL, SS: [DI + DISP]

Figure 6.5 shows the physical address computation in index addressing mode. When 16-bit displacement DISP = 0055H, the contents of SI is 0100H and the contents of DS register is 4000H, the physical address = DS × 10 + SI + DISP = 4000H × 10 + 0100 + 0055 = 40155H. If MOV AL, DS: [SI + 0025] is executed, the contents of the memory location 40155H, FF will be loaded into the AL register. Figure 6.5 shows the indexed addressing.

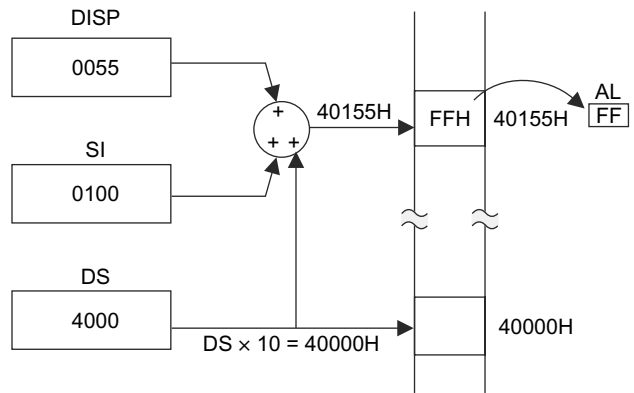


Fig. 6.5 Indexed addressing

Based Indexed Addressing

The contents of a base register (BX or BP) is added to the contents of an index register (SI or DI), the resulting value is a pointer to the location where the data resides. The effective address is the sum of a base register and an index register which are specified in the instruction. The based indexed addressing modes are simply combinations of the register indirect addressing modes. These addressing modes form the offset by adding together a base register (BX or BP) and an index register (SI or DI). The EA and the PA computation are given below:

EA = BX + SI, EA = BX + DI

EA = BP + SI, EA = BP + DI

PA = Segment × 10 + BX + SI

PA = Segment × 10 + BX + DI

PA = Segment × 10 + BP + SI

PA = Segment × 10 + BP + DI

Figure 6.6 shows the physical address computation in based indexed addressing mode. For example, if the content of BX register is 0200H and SI contains 0100H. Then the instruction MOV AL, [BX + SI] loads the content of the memory location DS × 10 + BX +

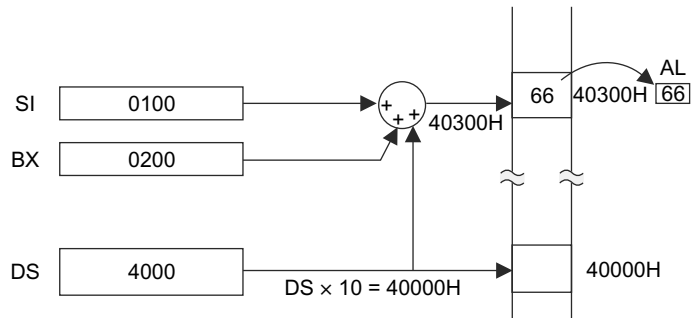


Fig. 6.6 Based indexed addressing

SI into the AH register. If DS = 4000H, the memory location address is $4000 \times 10 + 0200 + 0100 = 40300H$ whose content 66H will be loaded into the AH register. The examples of this addressing mode instruction are as follows:

```
MOV AL, [BX + DI]           MOV AL, [BX + SI]
MOV AL, [BP + SI]          MOV AL, [BP + DI]
```

Based Indexed with Displacement Addressing

The 8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP) and index register (SI or DI), the resulting value is a pointer to the location where the data resides. The effective address is the sum of an 8-bit or 16-bit displacement and based index address. The computation of EA and the PA are given below:

$$EA = BX + SI + \text{8-bit or 16-bit instruction} \quad EA = BX + DI \text{ 8-bit or 16-bit instruction}$$

$$EA = BP + SI + \text{8-bit or 16-bit instruction} \quad EA = BP + DI \text{ 8-bit or 16-bit instruction}$$

$$PA = \text{Segment} \times 10 + BX + SI + \text{8-bit or 16-bit instruction}$$

$$PA = \text{Segment} \times 10 + BX + DI + \text{8-bit or 16-bit instruction}$$

$$PA = \text{Segment} \times 10 + BP + SI + \text{8-bit or 16-bit instruction}$$

$$PA = \text{Segment} \times 10 + BP + DI + \text{8-bit or 16-bit instruction}$$

Figure 6.7 shows the physical address computation in based indexed with displacement addressing mode. When 16-bit displacement DISP = 0020H, the contents of BX register is 4000H, the contents of SI is 0300 and the contents of DS register is 5000H, the physical address = $DS \times 10 + BX + SI + DISP = 5000H \times 10 + 4000 + 0300 + 0020 = 54320H$. When MOV AL, DS:[BX + SI + DISP] is executed, the content of the memory location 54320H will be copied into the AL register. The examples of this addressing mode instruction are as follows:

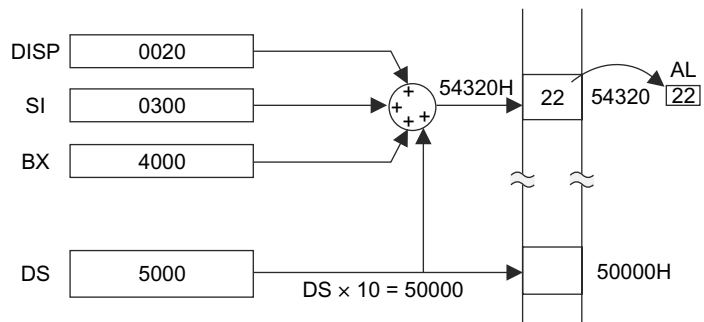


Fig. 6.7 Based indexed with displacement addressing

```
MOV AL, [BX + DI + DISP]   MOV AL, [BX + SI + DISP]
MOV AL, [BP + SI + DISP]   MOV AL, [BP + DI + DISP]
```

String Addressing Mode

String is a sequence of bytes or words which are stored in memory. Stored characters in word processors and data table are examples of string. Some instructions are designed to handle a string of characters or data. These instructions have a special addressing mode where DS : SI is used as a source of string and ES:DI is used to locate the destination address of the string. For example, MOV SB instruction is used to move a string of source data to the destination location.

When any MOV instruction is executed, data is always transferred from source to destination. The MOV instruction for different addressing modes is depicted in Table 6.1. The effective address computation depends

on MOD and R/M bit patterns as shown in Table 6.2. Segment registers for different addressing modes may be different and its selection also depends on MOD and R/M as illustrated in Table 6.3.

Table 6.1 Addressing modes of 8086/8088 microprocessors

Addressing mode	Mnemonic	Symbolic Operation	Destination of Operand	Source of Operand	Functions
Immediate addressing mode	MOV AX, 2000H	AH←20H; AL←00	AX register	Data 2000	Source of data is within instruction
Register addressing mode	MOV AX, BX	AX←BX	AX register	BX register	Source and destination of data are registers of microprocessors
Direct addressing mode	MOV AH, [0400]	AH←[0400H]	AH register	0400H = Displacement	Memory address is available within the instruction.
Register indirect addressing mode	MOV AX, [SI]	AL←[SI]; AH←[SI + 1]	AX register	SI + DS × 10 = memory location	Memory address is supplied in any index or pointer registers.
Indexed addressing mode	MOV AX, [SI + 06]	AL←[SI + 6]; AH←[SI + 7]	AX register	[SI + 06] + DS × 10 = memory location	Memory address is the sum of the indexed register and a displacement within the instruction.
Based addressing mode	MOV AX, [BP]	AL←[BP]; AH←[BP + 1].	AX register	BP + DS × 10 = memory location	Memory address is the content of BX or BP register within instruction.
Based and indexed addressing mode	MOV [BX + SI], AX.	[BX + SI]←AL; [BX + SI + 1]←AH.	BX + SI + DS × 10 = memory location	AX register	Memory address is the sum of an index register and a base register.
Based and indexed with displacement addressing mode	MOV AX, [BX + SI + 10]	AL←[BX + SI + 10]; AH←[BX + SI + 11]	AX register	[BX + SI + 10] + DS × 10 = memory location	Memory address is the sum of an index register, a base register and a displacement within instruction.
String addressing mode	MOV SB	[ES : DI]←[DS : SI] If DF = 0, then SI←SI + 1; DI←DI + 1. If DF = 1, then SI←SI - 1; DI←DI - 1.	DI + ES × 10 = memory location	SI + DS × 10 = memory location	The memory source address is the SI register in the data segment. The memory destination address is the DI register in the extra segment.

Table 6.2 Effective addressing computations corresponding to MOD and R/M fields

R/M	MOD 00	MOD 01	MOD 10	MOD 11	
				W = 0	W = 1
000	[BX] + [SI]	[BX] + [SI] + 8-bit DISP	[BX]+[SI] +16-bit DISP	AL	AX
001	[BX]+[DI]	[BX]+[DI] + 8-bit DISP	[BX]+[DI] +16-bit DISP	CL	CX
010	[BP]+[SI]	[BP]+[SI] + 8-bit DISP	[BP]+[SI] +16-bit DISP	DL	DX
011	[BP]+[DI]	[BP]+[DI] + 8-bit DISP	[BP]+[DI] +16-bit DISP	BL	BX
100	[SI]	[SI] + 8-bit DISP	[SI] +16-bit DISP	AH	SP
101	[DI]	[DI] + 8-bit DISP	[DI] +16-bit DISP	CH	BP
110	[Direct Address]	[Direct Address] + 8-bit DISP	[Direct Address] +16-bit DISP	DH	SI
111	[BX]+[SI]	[BX]+[SI] + 8-bit DISP	[BX]+[SI] +16-bit DISP	BH	DI
Memory Mode				Register Mode	

Table 6.3 Segment registers for different addressing modes corresponding to MOD and R/M fields

R/M	MOD 00	MOD 01	MOD 10	Segment Register
000	[BX] + [SI]	[BX] + [SI]+8-bit DISP	[BX] + [SI] +16-bit DISP	DS
001	[BX] + [DI]	[BX] + [DI] + 8-bit DISP	[BX] + [DI] +16-bit DISP	DS
010	[BP] + [SI]	[BP] + [SI] +8-bit DISP	[BP] + [SI] +16-bit DISP	DS
011	[BP] + [DI]	[BP] + [DI] +8-bit DISP	[BP] + [DI] +16-bit DISP	DS
100	[SI]	[SI] + 8-bit DISP	[SI] +16-bit DISP	DS
101	[DI]	[DI] + 8-bit DISP	[DI] +16-bit DISP	DS
110	[Direct Address]	[Direct Address] + 8-bit DISP	[Direct Address] +16-bit DISP	DS or SS
111	[BX] + [SI]	[BX] + [SI] + 8-bit DISP	[BX]+[SI] + 16-bit DISP	DS
8-bit DISP = 8-bit displacement			16-bit DISP = 16-bit displacement	

6.2.4 Branch Addressing

The basic types of branch addressing are shown in Fig. 6.8. The *intra-segment* mode is used to transfer the control to a destination that lies in the same segment where the control transfer instruction itself resides. In the *inter-segment* mode, address is used to transfer the control to a destination that lies in a different segment.

For the branch-control transfer instructions, the addressing modes depend upon whether the destination location is within the same segment or in a different one. It depends upon the method of passing the destination address to the processor. There are two types of branch control instructions: intersegment and intrasegment addressing modes.

During execution of program instruction, when the location to which the control to be transferred lies in a different segment other than the current one, the mode is called *inter-segment mode*. If the destination location lies in the same segment, the mode is called *intra-segment mode*.

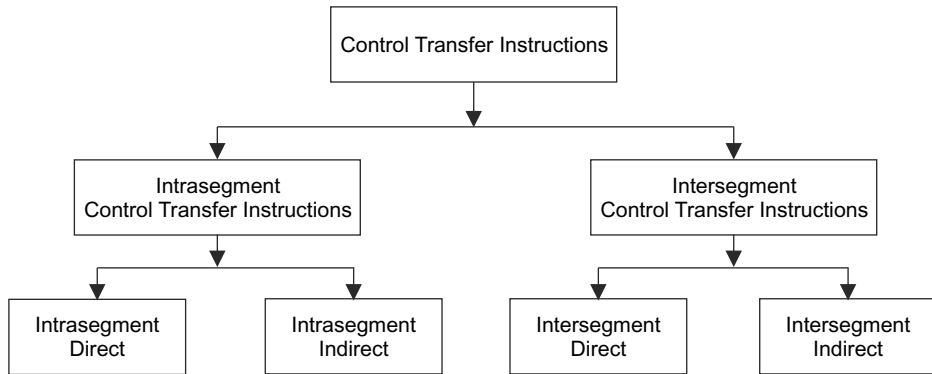


Fig. 6.8 Branch control transfer instructions

Intrasegment Direct

The effective branch address is sum of an 8-bit or 16-bit displacement and the current contents of IP. When the displacement is 8-bit long, it is referred to as a *short jump*. Intrasegment direct addressing is what most computer books refer to as relative addressing because the displacement is computed 'relative' to the IP. It may be used with either conditional or unconditional branching, but a conditional branch instruction can have only 8-bit displacement.

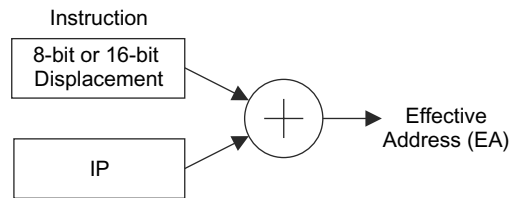


Fig. 6.9 Intrasegment direct addressing

Figure 6.9 shows intrasegment direct addressing. In the intrasegment direct mode, the destination location to which the control is transferred lies in the same segment where the control-transfer instruction lies and appears directly in the instruction as an immediate displacement value. The displacement is relative to the contents of the IP. The expression for effective address in which the control is transferred is given below:

$$EA = \text{Contents of IP} + 8\text{- or }16\text{-bit displacement.}$$

Intrasegment Indirect

The effective branch address is the contents of a register or memory location that is accessed using any of the above data-related addressing modes except the immediate mode. The contents of IP are replaced by the effective branch address. This addressing mode may be used only in unconditional branch instructions. Figure 6.10 shows intrasegment indirect addressing.

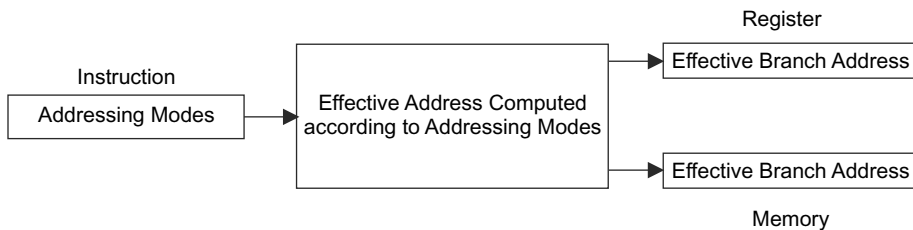


Fig. 6.10 Intrasegment indirect addressing

In this mode, the control to be transferred lies in the same segment where the control instruction lies and is passed indirectly to the instruction. It uses unconditional branch instructions. The effective branch address is that of the register or memory location that is accessed using any of the above data-related addressing modes except the immediate mode. The contents of the IP are replaced by the effective branch address.

Intersegment Direct This replaces the contents of IP with a part of the instruction and the contents of CS with another part of the instruction. The purpose of this addressing mode is to provide a means of branching from one code segment to another. Figure 6.11 shows intersegment direct addressing.

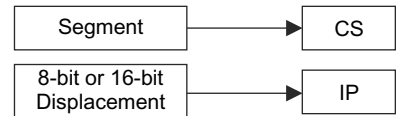


Fig. 6.11 Intersegment direct addressing

If the location to which the control is to be transferred lies in the different segment, than in which the control transfer instruction lies, it is called intersegment. This addressing mode provides the facility of branching from one segment to another segment. The CS and IP specify the destination address directly in the instruction. It replaces the contents of IP with the part of the instruction and the contents of CS with another part of the instruction.

Intersegment Indirect This mode replaces the contents of IP and CS with the contents of two consecutive words in memory that are referenced using any of the above data-related addressing modes except the immediate and register modes. Figure 6.12 shows intersegment indirect addressing.

The location to which the control is to be transferred lies in the different segment than the segment where the transfer control instruction lies and is passed to the instruction indirectly, i.e., it replaces the contents of IP and CS with the contents of 2 consecutive words in the memory. The starting address of the memory block may be referred using any of the addressing modes except immediate.

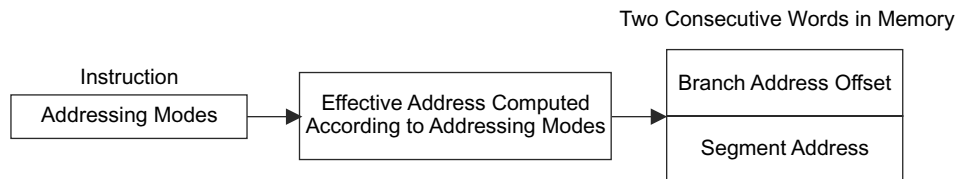


Fig. 6.12 Intersegment indirect addressing

Example 6.1

Find the addressing modes of the following instructions:

- (i) MOV CX, BX
- (ii) MOV BX, 1234
- (iii) MOV AX, [SI]
- (iv) MOV [Offset Address], 2345
- (v) MOV CX, [BX+SI]
- (vi) MOV AX, [BX+SI+1234]

Solution

- (i) MOV CX, BX instruction is an example of register addressing mode
- (ii) MOV BX, 1234 instruction is an example of immediate addressing mode

- (iii) MOV AX, [SI] instruction is an example of indexed addressing mode
- (iv) MOV [offset address], 2345 instruction is an example of memory addressing mode
- (v) MOV CX, [BX + SI] instruction is an example of based indexed addressing mode
- (vi) MOV AX, [BX + SI + 1234] instruction is an example of based indexed with displacement addressing mode

Example 6.2

Determine the memory location accessed by the following instructions:

- (i) MOV AL, [0100]
- (ii) MOV CL, [BX + 0200] Assume CS = 2300, BX = 1000H, SI = 0100

Solution

- (i) Physical address of the memory location accessed by MOV AL, [0100] is $CS \times 10 + 0100 = 2300 \times 10 + 0100 = 23100H$
- (ii) Physical address of the memory location accessed by MOV CL, [BX + 0200] is $CS \times 10 + BX + 0200 = 2300 \times 10 + 1000 + 0100 = 24100H$

Example 6.3

The contents of different registers are AX = 1000H, BX = 2000H, SI = 3000H, DI = 4000H, BP = 5000H, SP = 6000H, CS = 8000H, DS = 1000H, SS = 2000H, IP = 7000H.

Determine the 16-bit effective addresses and 20-bit physical address for the following addressing modes:

- (i) Direct addressing
- (ii) Register indirect addressing
- (iii) Based Indexed addressing
- (iv) Based Indexed with displacement addressing

Assume Offset (displacement) = 0500H

Solution

- (i) *Direct addressing mode*
MOV AX, [0100H] is the example of direct addressing mode instruction. The 16-bit effective addresses = 0500 and 20-bit physical address = $CS \times 10 + 0500 = 8000 \times 10 + 0500 = 80500H$
- (ii) *Register indirect addressing*
MOV AX, [BX] is the example of register indirect addressing mode instruction. The 16-bit effective addresses is the content of BX register = 2000 and 20-bit physical address = $CS \times 10 + BX = 8000 \times 10 + 2000 = 82000H$
- (iii) *Based indexed addressing*
MOV AX, DS:[BX + SI] is the example of base indexed addressing mode instruction. The 16-bit effective addresses is the content of BX plus SI register = $BX + SI = 2000 + 3000 = 5000H$ and 20-bit physical address = $DS \times 10 + BX + SI = 1000 \times 10 + 2000 + 3000 = 15000H$
- (iv) *Based indexed with displacement addressing*
MOV AX, DS:[BX + SI + DISP] is the example of base indexed addressing mode instruction. The 20-bit physical = $DS \times 10 + BX + SI + DISP$

$$= 1000 \times 10H + 2000H + 3000H + 0500H$$

$$= 15500H$$

Example 6.4

If SI = 0200H, what will be the content of the register AH after the execution of MOV AX, [SI] instruction? Assume DS = 4000H.

Solution

As DS = 4000H and the content of SI pointer register is 0200, the content of memory location 40200H ($DS \times 10 + SI = 4000 \times 10 + 0200$) will be copied into the AL register. Then the content of memory location 40201H ($DS \times 10 + [SI + 1] = 4000 \times 10 + 0201$) will be copied into the AH register.

Example 6.5

Determine the starting and ending address for data segment and code segment. Assume DS = 5000H and CS = 7000H.

Solution

The starting address is found by multiplying 10 with the content of the segment register. The ending address can be determined after adding FFFFH (64K) with the starting address. Then the starting address of data segment memory = $DS \times 10 = 5000 \times 10 = 50000H$ and the ending address of data segment memory is equal to = $50000H + FFFFH = 5FFFFH$. Similarly the starting address of code segment memory = $CS \times 10 = 7000 \times 10 = 70000H$ and the ending address of code segment memory is equal to = $70000H + FFFFH = 7FFFFH$.

Example 6.6

What physical address can be accessed by the instruction MOV [BP], AL if BP = 2500H. Assume the content of stack segment register is 4578H.

Solution

The physical address accessed by the instruction MOV [BP], AL is $SS \times 10 + BP = 4578 \times 10 + 2500H = 47C80H$.

6.3 8086 INSTRUCTION SET

The Intel 8086 Instruction Set is the core of the entire series of processors created by Intel. Some instructions have been added to this set to accommodate the extra features of later designs, but the set shown here contains the basic instructions understood by all of the processors. The 8086 instruction set consists of the following instructions:

- ✦ *Data Transfer Instructions* move, copy, load exchange, input and output
- ✦ *Arithmetic Instructions* add, subtract, increment, decrement, convert byte/word and compare
- ✦ *Logical Instructions* AND, OR, exclusive OR, shift/rotate and test
- ✦ *String Manipulation Instructions* load, store, move, compare and scan for byte/word.
- ✦ *Control Transfer Instructions* conditional, unconditional, call subroutine and return from subroutine
- ✦ *Input/Output Instructions*
- ✦ *Other Instructions* setting/clearing flag bits, stack operations, software interrupts, etc.

The instruction format consists of opcode and operand. Depending upon the opcode and number of operand present in an instruction, instructions are one byte to six bytes long. The general format of an instruction is illustrated in Fig. 6.13.

The first byte of any instruction is the opcode. The bits D_7 to D_2 specify the operation which will be carried out by the instruction. D_1 is the register direction bit (D). This bit defines whether the register operand is in byte 2 or is the source or destination operand. While $D = 1$, the register operand is the destination operand.

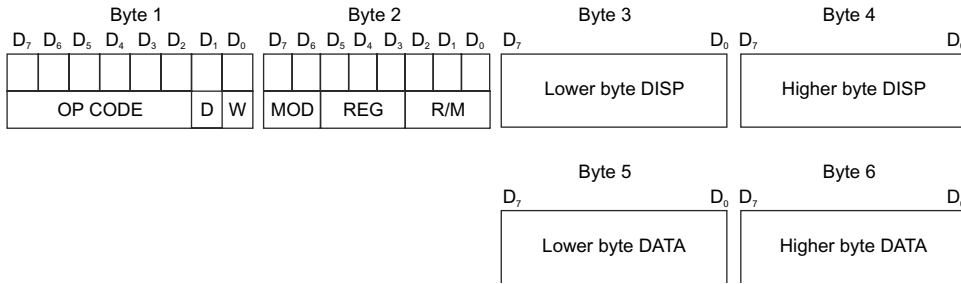


Fig. 6.13 Instruction format of 8086/8088 microprocessor

If D = 0, the register operand is source operand. D₀ represents data size (W), whether the data is 8-bit or 16-bit. When W = 0, data is 8-bit and if W = 1, data will be 16-bit.

The second byte of the instruction specifies whether the operand is in the memory or in the register. This byte consists of Mode (D₇–D₆ bits), Register (D₅, D₄, D₃ bits) and R/M (D₂, D₁, D₀). The third and fourth bytes of the instruction specify lower 8-bit displacement and higher 8-bit displacement of the memory respectively. Then last two bytes (fifth and sixth) represent lower 8-bit data and higher 8-bit data.

6.3.1 Classification of Instructions

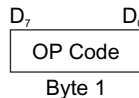
Instructions are performed operations with 8-bit data and 16-bit data. 8-bit data can be obtained from a register or a memory location or input port. In the same way, 16-bit data may be available from any register pair or two consequent memory locations. Hence binary codes of instructions are different. Due to different ways of specifying data for instructions, the machine or binary codes of all instructions are of different lengths. The Intel 8086/8088 instructions are classified into the following groups based on number of bytes in the instruction as given below:

- ✦ One-byte instructions
- ✦ Two-byte instructions
- ✦ Three-and four-byte instructions
- ✦ Five-and six-byte instructions

One-byte Instructions

An one-byte instruction is used as opcode as well as data or operand. The least three bits of the opcode are used to specify the register operand as shown in Fig. 6.14

One-byte instruction–implied operand



One-byte instruction–register mode

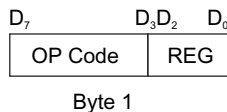
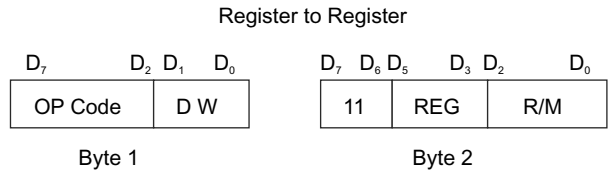


Fig. 6.14 One byte instruction

Examples of one-byte instructions are XLAT, LAHF, SAHF, PUSH AX, POP DS, PUSHF and POPF. The opcodes of these instructions are

D7 for XLAT,
 9F for LAHF,
 9E for SAHF,
 50 for PUSH AX, 1F for POP DS,
 9C for PUSHF,
 9D for POPF



Two-byte Instructions

Register-to-register and register to/from memory with no displacement instructions are two bytes long as shown in Fig. 6.15. In a register-to-register instruction, the first byte of the code specifies the instruction operation (D_7-D_2) and width of the operand is specified by D_1-D_0 (W). The second byte represents the register operand and R/M field as given in Table 6.2.

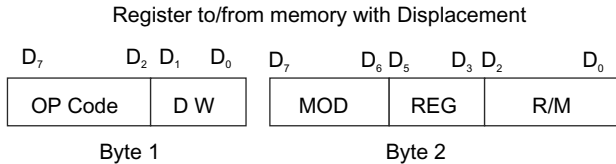


Fig. 6.15 Two-byte instructions

The register to/from memory with no displacement instructions are same as register to register instructions except MOD field as depicted in Fig. 6.14. The MOD field can be used to represent different modes of addressing as given in Table 6.4.

Table 6.4 Different modes of addressing

MOD	Mode of addressing
00	Memory addressing without displacement
01	Memory addressing with 8-bit displacement
10	Memory addressing with 16-bit displacement
11	Register addressing with W = 0 for 8-bit and W = 1 for 16-bit

Examples of two-byte instructions are MOV AX, BX; MOV AL, BL; IN AL, 01; and OUT 02, AL, etc. The opcodes and operands of these instructions are as follows:

- 89 D8 for MOV AX, BX
- 88 D8 for MOV AL, BL
- E4 01 for IN AL, 01
- E6 02 for OUT 02, AL

Three-Byte and Four-Byte Instructions

Register to/from memory with displacement and immediate operand to register instructions are four-byte instructions as depicted in Fig. 6.16. The register to/from memory with displacement instruction consists of one or two additional bytes for displacement and the 2 bytes of the register to/from memory without displacement as given below:

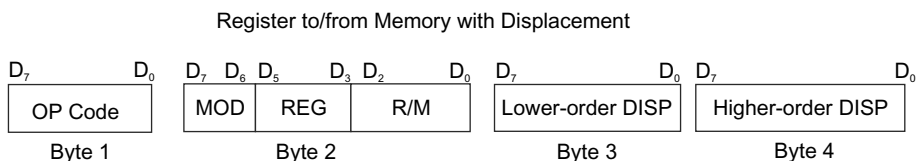
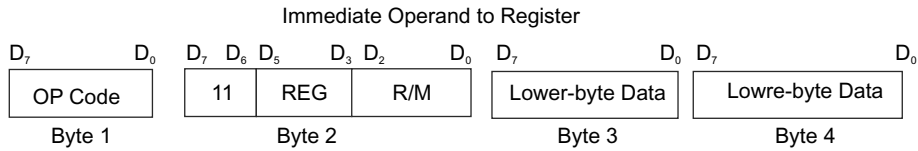


Fig. 6.16 Four-byte instructions

In immediate operand to register instruction, the first byte and the three bits D_5 – D_3 of the second byte are used for opcode. This instruction consists of one or two bytes of immediate data. The format of the instruction is given below:



When the instruction consists of one-byte immediate data, it acts as a three-byte instruction. If the instruction has two bytes of immediate data, it works as a four-byte instructions. Examples of three-byte instructions are MOV SI, 0300; MOV CX, 0005; MOV DI, 0100; MOV AL, [0300], and MOV [0400], AL, etc. The opcodes and operands of these three-byte instructions are given below:

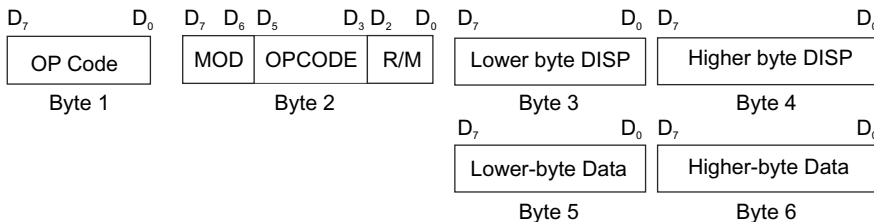
- BE 00 03 for MOV SI, 0300
- B9 05 00 for MOV CX, 0005
- BF 00 01 for MOV DI, 0100
- A0 00 03 for MOV AL, [0300]
- A2 00 04 for MOV [0400], AL

The four-byte instructions are MOV [BX + SI + 1000], AX; MOV [BX + DI + 0447], CL; MOV [BX + SI + 0300], SP; MOV [BP + SI + 0400], DL; and MOV [BP + DI + 0100], BL. The opcodes and operands of these four-byte instructions are as follows:

- 89 80 00 10 for MOV [BX + SI + 1000], AX
- 88 89 47 04 for MOV [BX + DI + 0447], CL
- 89 A0 00 03 for MOV [BX + SI + 0300], SP
- 88 92 00 04 for MOV [BP + SI + 0400], DL
- 88 9B 00 01 for MOV [BP + DI + 0100], BL

Five- and Six-Byte Instructions

Immediate operand to memory with 16-bit displacement instructions are six-byte instructions. The first two bytes represent OPCODE, MOD, OPCODE and R/M fields. The other four bytes consists of 2 bytes for displacement and 2 bytes for data as given below:



Immediate Operand to Memory with 16-bit displacement

Fig. 6.17 Six-byte instructions

The example of five byte instructions are MOV [BX + SI] + DISP, 22; MOV [BX + DI] + DISP, 66; MOV [SI] + DISP, 44, etc. The opcodes and operands of these three byte instructions are given below:

- C6 80 $DISP_L$ $DISP_H$ 22 for MOV [BX + SI] + DISP, 22
- C6 81 $DISP_L$ $DISP_H$ 66 for MOV [BX + DI] + DISP, 66
- C6 84 $DISP_L$ $DISP_H$ 44 for MOV [SI] +DISP, 44

The six-byte instructions are MOV [BX + SI + 1000], 2345; MOV [BX + DI + 0447], 2000; MOV [SI + 0300], 4466. The opcodes and operands of these four byte instructions are as follows:

```
C7 80 00 10 45 23   for   MOV   [BX + SI + 1000], 2345
C7 81 47 04 00 20   for   MOV   [BX + DI + 0447], 2000
C7 84 00 03 66 44   for   MOV   [SI + 0300], 4466
```

The 8086 instruction set can be divided into different categories based on their functions as follows:

- ✓ **Data Transfer Instructions** These types of instructions are used to transfer data from the source operand to the destination operand. All copy, store, move, load, input and output instructions fall in this category. Examples of instructions of this group are MOV, LDS, XCHG, PUSH and POP.
- ✓ **Arithmetic and Logical Instructions** All the instructions performing arithmetic, logical, increment, decrement, compare and scan instructions belong to this category. For example, ADD, SUB, MUL, DIV, INC, CMP and DAS, AND, OR, NOT, TEST, XOR.
- ✓ **Branch Instructions** These instructions transfer control of execution to the specified address. All CALL, JUMP, interrupt and return instructions belong to this category.
- ✓ **Loop Instructions** These instructions have REP prefix with CX used as count register, and they can be used to implement unconditional and conditional loops. The LOOP, LOOPNZ and LOOPZ instructions belong to this category. Usually, these instructions are used to implement different delay loops.
- ✓ **Processor Control Instructions** These instructions control the machine status. CLC, CMC, CLI, STD, STI, NOP, HLT, WAIT and LOCK instructions are example of machine control instructions.
- ✓ **Flag Manipulation Instructions** All these instructions which directly affect the flag register come under this group of instructions. Instructions like CLD, STD, CLI, STI, etc; belong to this category.
- ✓ **Shift and Rotate Instructions** These instructions involve the bitwise shifting OR rotation in either direction with or without a count in CX. The examples of instructions are RCL, RCR, ROL, ROR, SAL, SHL, SAR and SHR.
- ✓ **String Instructions** These instructions involve various string manipulation operation like load, move, scan, compare, store, etc. These instructions are only operated upon the strings. The examples of string instructions are MOVS, LODS and STOS.

6.3.2 Data-Transfer Instructions

The data-transfer instructions are used to transfer data between registers, registers and memory, registers and immediate data, or memory and immediate data. All data-transfer instructions are explained in this section.

MOV Destination, Source (Copy data from source to destination)

Destination ← Source, Flag affected: None

The instruction perform datas movement between registers, registers and memory, registers and immediate data, memory and immediate data, between two memory locations, between I/O port and registers and between the stack and memory or a register. Both 8-bit and 16-bit data registers are used in data transfer instructions.

In case of immediate addressing mode, a segment register cannot be a destination register. Direct loading of the segment registers with immediate data is not permitted. To load the segment registers with immediate data, one will have to load any general-purpose register with data and then it will have to be moved to that particular segment register.

<i>Destination</i>	<i>Source</i>
Register	Immediate data
Memory	Immediate data
Register	Register
Register	Memory
Memory	Register
Segment	Memory
Memory	Segment
Register	Segment
Segment	Register

The MOV instruction will not be able to

- ✦ set the value of the CS and IP registers
- ✦ copy the value of one segment register to another segment register (should copy to general register first).
- ✦ copy an immediate value to segment register (should copy to general register first)

MOV Register, Immediate Data This instruction moves immediate 8-bit/16-bit data to the specified register. Its object code is either 2 or 3 bytes based on data.

The format of its object code is as follows:

$w = 0$ for 8-bit data

1011	w	r r r	8-bit Data
------	---	-------	------------

$w = 1$ for 16-bit data

1011	w	r r r	Lower 8-bit Data	Higher 8-bit Data
------	---	-------	------------------	-------------------

r r r = address of register as illustrated in Table 6.2

For example, MOV AL, 8-bit data

object code is 1011 w r r r = 1011 0000 = B0 as $w = 0$ and r r r = 000

If the instruction is MOV AL, FFH, the object code will be B0, FF H

MOV mem/reg, data When this instruction is executed, immediate 8-bit or 16-bit data moves to a specified register or a memory location(s) though this instruction is not used to transfer immediate data to a register. The format of its object code is given below:

$w = 0$ for 8 bit data

1100 011w	Mod 000 R/M	8-bit Data
-----------	-------------	------------

$w = 1$ for 16 bit data

1100 011w	Mod 000 R/M	Lower -8 bit Data	Higher -8 bit Data
-----------	-------------	-------------------	--------------------

Mod R/M and data are explained in Section 6.2.

In this instruction, memory location can be addressed directly or by a register or a combination of register and displacement.

For example, MOV [0345], 23H

When this instruction is executed, 23H will be loaded into the memory location $DS \times 10 + 0345$. The object code is as follows:

1100 011w	Mod 000 R/M	8 bit data
-----------	-------------	------------

This instruction is direct addressing of a memory location. As per Table 6.2, for direct addressing Mod = 00, R/M = 110 and $w = 0$ for 8-bit operation. Then object code is

1100 0110	00 000 110	23
-----------	------------	----

= C6, 06, 23

Another example is MOV [0345], 2345H

When this instruction is executed, 45H will be loaded into the memory location $DS \times 10 + 0345$ and 23H will be loaded into $DS \times 10 + 0346$. The object code format of this instruction as given below:

1100 011w	Mod 000 R/M	Lower 8-bit Data	Higher 8-bit Data
-----------	-------------	------------------	-------------------

This instruction is direct addressing of a memory location. As per Table 6.2, for direct addressing Mod = 00, R/M = 110. $w = 1$ for 16-bit operation.

Then object code

1100 0111	00 000 110	46	23
-----------	------------	----	----

= C7, 06, 46, 23

The example of MOV[reg], data instruction is MOV [BX], 45H. When this instruction is executed, data will be moved to the memory location specified by the content of BX register. The object code is

1100 011 w	Mod 000 R/M	8-bit Data
------------	-------------	------------

Here, Mod = 00, R/M = 111 and $w = 0$ for 8-bit operation. Then object code is

1100 0110	00 000 111	45
-----------	------------	----

= C6, 07, 45

MOV ACC, Memory

When MOV ACC, Memory instruction is executed, the 8-bit data moves from a memory location to AL, or 16-bit data moves from two consecutive memory locations to AX register. The object code of this instruction is

1010 000w	16-bit offset address
-----------	-----------------------

For $w = 0$, Object code = A0, Offset address

For $w = 1$, Object code = A1, Offset address

For example, MOV AL, [2340]. This instruction moves the content of offset address 2340H to AL and the object code is A0, 40, 23. If the content of the memory location is 25H, after execution of MOV AL, [2340], 25 will be stored in AL.

MOV Memory, ACC

The content of the accumulator will be stored into the memory. This means that the content of AL is stored in memory and contents of AX will be stored in two consecutive memory locations. The object code is

1010 001w	16-bit Offset Address
-----------	-----------------------

For $w = 1$, Object code = A3, offset address

For $w = 0$, Object code = A2, offset address

For example, MOV [4000], AL Content of AL is stored in the memory location represented by offset address 4000H. Then object code is A2, 00, 40.

Another example is MOV [4000], AX. Content of AX is stored in the two consecutive memory locations represented by offset address 4000H.

MOV Memory/Register, Memory/Register

This instruction is used to move 8-bit or 16-bit data from one register to another register, memory to register and register to memory. This instruction cannot be used for data transfer from memory to memory. The object code is

1010 10 d w	Mod reg R/M
-----------------	-------------

The direction flag d is either 0 or 1. When $d = 0$, the specified register is the source of the operand. The Mod and R/M are used for the first operand (the content of memory/register-1) and reg represents the second operand (the content of memory/register-2). If $d = 1$, register specifies a register which works as the destination of the operand. The Mod and R/M are also used for the second operand (memory/register-2) and reg defines the first operand (mem/register-1).

For example, MOV BX, CX

This instruction is used for CX register to BX register data transfer. If $d = 0$, register specifies a register which is the source for the operand. When $d = 0$, Mod and R/M are used for the first operand, i.e., BX. As per Table 6.2 for BX, Mod = 11, R/M = 011. The register defines the second operand, CX (source for the operand). Then reg is 001 and $w = 1$. In that case the object code is

1010 1001	Mod reg R/M
-----------	-------------

= 1000 1001, 11 001 011 = 89, CB

Mod reg R/M

for for for

BX CX BX

When $d = 1$, reg specifies a register which is used as the destination for the operand. If $d = 1$, Mod and R/M are used for the second operand CX; and reg stands for the first operand BX as destination and $w = 1$. Then object code is 1000 1011, 11 011 001 = 8B, D9

Mod reg R/M

for for for

CX BX CX

Hence both codes 89, CB and 8B, D9 are valid for MOV BX, CX instruction.

Another example is MOV CX, [0500]. The object code of this instruction is

1000 10 d w	Mod reg R/M
-----------------	-------------

In this case, reg will specify register CX which acts as destination for the operand when $d = 1$. If $d = 1$, Mod and R/M are used direct addressing.

The mod = 00, R/M = 111, CX = 001 and $w = 1$

Then the object code is 1000 1011, 00 001 111 = 8B, 0F

Example 6.7

Write instructions for the following operations:

- (i) Move the content of DX register into SS register
- (ii) Load 16-bit data from memory location offset address 0300 to AX
- (iii) Load 8-bit data, FF in the BL register
- (iv) Source index address 0100 is stored in SI
- (v) Destination index address 0400 is stored in DI

Solution

- (i) MOV SS, DX; Move the content of the DX register into the SS register
- (ii) MOV AX, [0300]; Load 16-bit data from memory location offset address 0300 to AX
- (iii) MOV BL, FF; Load 8-bit data, FF in the BL register
- (iv) MOV SI, 0100; Source index address 0100 is stored in SI
- (v) MOV DI, 0400; Destination index address 0400 is stored in DI

XCHG Destination, Source (Exchange data between source to destination)

Destination ↔ Source, Flag affected: None

<i>Destination</i>	<i>Source</i>
Accumulator	register
Memory	register
Register	register

This instruction is used to exchange the contents of the specified source and destination operands, which may be registers or a memory location. But the exchange of contents of two memory locations is not allowed. Immediate data is not allowed in XCHG instructions. For example, XCHG [4000], AX exchange data between AX and a memory location represented by offset address 5000 H with the content of data segment. XCHG AX, BX instruction exchanges data between AX and BX. The data format for register to register and register to memory is

1000 011 w | Mod reg R/M

and the data format for register to accumulator is

1001 0 reg

The object code of XCHG AX,BX is 1000 011w Mod reg R/M = 1000 0111 11 000 011 = 87 C3 as $w = 1$, Mod = 11, reg = 000 and R/M = 011. Similarly the object code of XCHG AL, [BX] = 1000 0110, 00 000 111 = 86, 07 where $w = 0$, Mod = 00, reg = 000 and R/M = 111

LAHF (Loads the lower flags byte into AH)

AH ← Flags, Flag not affected: O A C Z P

Loads the low byte of the flags word into the AH register. Provides support for 8085 processor. Load AH from 8 low bits of flags register. This instruction loads the AH register with the lower byte of the flag register. This command may be used to observe the status of all the condition code flags at a time. The LAHF instruction followed by a PUSH AX instruction has the same effect of PUSH PSW instruction in 8085

AH = flags register

AH bit: 7 6 5 4 3 2 1 0

[SF] [ZF] [0] [AF] [0] [PF] [1] [CF]

Here bits 1, 3, and 5 are reserved.

The object code of LAHF instruction is 9F

1001 1111

SAHF (Saves AH into lower flags byte)

Flags ← AH, Flag affected: None

Saves the AH register bit pattern into the low byte of the flags register.

The lower byte of 8086 flag register is exactly same as flag register of 8085. The SAHF instruction replaces the equivalent flag byte of 8085 with a byte from the AH register. POP PSW instruction of 8085 will be translated to POP AX SAHF on a 8086 processor. SAHF changes the flags in the lower byte of flag register. The object code of SAHF instruction is 9E

1001 1110

IN port8 or DX (Input data from I/O device)

byte: AL ← port

word: AL ← [port]; AH ← [port+1] or AX ← (DX) Flag affected: None

This instruction is used to read data from an input port. The address of the input port can be specified within the instruction directly or indirectly. AL and AX registers can be used as destinations for 8-bit and 16-bit input operands respectively. DX is the only register which is allowed to carry the port address. It fetches a byte or word into AL or AX from an 8-bit port or the 16-bit address contained in DX. The 8-bit port supports the I/O technique of earlier processors such as 8085. Input a byte or word from direct I/O ports 00H to FFH (0 to 255). When the port address consists of 16 bits, it must be addressed by DX. Input a byte or word from indirect I/O ports 0000H to FFFFH (0-65535); port address is in DX and flags are not affected. The object code of this instruction is

For fixed port

1110010 w	port
-----------	------

For variable port

1110110 w

The examples of IN instructions are

IN AL, 01; Load the content of 8-bit port address 01H to AL register. The object code of IN AL, 01 is 1110010w, port address = E4, 01 as w = 0.

IN AX, DX; Read data from a 16-bit port address specified by DX register and stores it in AX register. The object code of IN AX, DX is 1110 110w = ED as w = 1.

OUT port8 or DX (Output data to I/O device)

byte: [port] ← AL

word: [port] ← AL [port+1]*AH or (DX←AX)

Flag affected: None

This instruction is used to write on an output port. The address of the output port may be specified in the instruction directly or implicitly in DX. Therefore, the contents of AX or AL are transferred to a directly or indirectly addressed port after execution. This instruction can output a byte or word to direct I/O ports 00H to FFH (0 to 255).

It can send a byte or word to an 8-bit port address or the 16-bit port address contained in DX. The registers AL and AX are the allowed source operands for 8-bit and 16-bit operations respectively. If the port address is of 16-bit, it must be in DX. Output a byte or word to indirect I/O ports 0000H to FFFFH (0 to 65535); port address is in DX and flags are not affected. The object code of this instruction is

For fixed port

1110 011w	port
-----------	------

For variable port

1110 111w

The examples of OUT instructions are OUT 02, AL and OUT DX, AX.

OUT 02, AL

After execution of this instruction, sends the content of AL to a port address 02H. The object code of OUT 02, AL is 1110 011w, port address = E6, 02 as w = 0.

OUT DX, AX

This instruction sends data available in AX to a port address which is specified by the DX register. The object code of OUT DX, AX is 1110 111w = EF as w = 1.

Example 6.8

Write instructions for the following operations:

- (i) Exchange the byte between memory location offset address 0300 and the AL register
- (ii) Load 8 bits of flags in the AH register
- (iii) Exchange the word between DX and AX registers
- (iv) Copy a byte from the port address 03 to the AL register
- (v) Output the content of accumulator to port address 01

Solution

- (i) XCHG AL, [0300]; Exchange the byte between memory location offset address 0300 and the AL register
- (ii) LAHF; Load 8 bits of flags in the AH register
- (iii) XCHG DX, AX; Exchange the word between DX and AX registers
- (iv) IN AL, 03; Copy a byte from the port address 03 to the AL register
- (v) OUT 01, AX; Output the content of accumulator to port address 01

LEA reg16, addr (load effective address)

reg16 ← effective address (offset) of addr. Flag affected: None

Loads the effective address or offset of memory variable into reg16. This type of data-transfer operation is important to load a segment or general-purpose register with an address directly from memory. The LEA instruction is used to load a specified register with 16-bit offset address. This instruction is very useful for assembly language. The object code is

1000 1101	Mod reg R/M
-----------	-------------

For example, LEA SI, address states that the 16-bit effective address loads in the SI register.

LEA BX, ADR

Effective address of ADR will be transferred to BX register. The object code of LEA BX, [0245] = 1000 1101 00 000 001 45 02 = 8D 1E 45 02.

LDS reg16, memory (load data segment)

reg16 ← [memory16]; DS ← [memory16+2] Flag affected: None

Loads the DS register and reg16 from memory with the segment and offset values. This instruction loads the specified register in the instruction with the content of the memory location specified as source in the instruction. It also loads the contents of the memory locations following the specified memory locations into DS register. The object code is

1100 0101	Mod reg R/M
-----------	-------------

= C5, mod reg r/m

The example of LDS instruction is LDS AX, [BX]

Load the contents of memory locations specified by the content of BX register into AX. Here, Mod for [BX] = 00, R/M for [BX] = 111 and reg for AX = 000. Then the object code = C5, 00000111 = C5, 07

LES reg16, memory (Load extra segment)

reg16 ← [mem16]; ES ← [mem16 + 2] Flag affected: None

Loads the ES register and reg16 with the segment and offset values for the variable in memory. The object code is

1100 0100	Mod reg R/M
-----------	-------------

The example of LES instruction is LES CX, [4000]. The object code of LES CX, [4000] instruction is 1100 0100 00 001 110 00 40 = C4 0E 00 40.

XLAT (Translate byte in AL by table look-up)

AL ← DS: [BX+AL], Flag affected: None

This instruction is used to translate the byte in the AL register by adding it to a base value in BX which has been set to locate the look-up table and the byte located is returned in AL. The physical address of memory location of look-up table is computed from DS: [BX+AL]. After execution of XLAT, the data from the memory location of the look-up table is loaded into the AL register. Using look-up table technique, this instruction is able to find out the codes in case of code-conversion problems. The object code of XLAT instruction is 1101 0111 = D7.

Example 6.9

Write instructions for the following operations:

- (i) Load the content of specified memory location represented by BX into the AX register.
- (ii) Load the content of specified memory location represented by SI into the AL register.
- (iii) Load the content of specified memory location represented by SI into the AX register.
- (iv) Replace a byte in the AL register with a byte from the of look-up table

Solution

- (i) LDS AX, [BX]; Load the content of specified memory location represented by BX into the AX register
- (ii) LODSB; Load the content of specified memory location represented by SI into the AL register
- (iii) LODSW; Load the content of specified memory location represented by SI into the AX register
- (iv) XLAT; Replace a byte in AL register with a byte from the of look-up table.

6.3.3 PUSH and POP Instructions

These instructions are used to manipulate stack-related operations. All stack instructions are explained in this section.

PUSH Source Push source register onto stack

SP = SP-2; SS:[SP] ← source register, Flag affected: None

After execution of this instruction, the content of a specified register is pushed on to the stack. The Stack Pointer (SP) is decremented by 2 after execution, and then stores the two-byte contents of the operand onto stack. Initially the higher byte is pushed and then the lower byte is pushed, so that the higher byte is stored

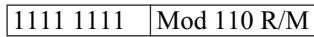
in the higher address and the lower byte is stored in the lower address. The actual operation of PUSH BX instruction is shown in Fig. 6.15. The sequence of PUSH operation is as follows:

- ✦ The current stack top is already occupied in the stack segment memory, so that SP is decrement by one then store the content of BH into the address pointed by SP and the stack segment SS.
- ✦ Again, decrement SP by one and store BL into the memory location pointed by SP and the stack segment SS.

In this way, SP is decrement by 2 and BH-BL contents are stored in the stack as shown in Fig. 6.18.

Thereafter, the contents of SP point to a new stack top. Assume the content of BX = 1234, SS = 4000 and SP = 01FF. Then content of BH, 12H is stored in 410FEH and content of BL, 34H is also stored in 410FDH.

The object code of PUSH instruction is
For register/memory



For register 01010 reg

For segment register 000 reg 110

The examples of these instructions are PUSH AX, PUSH BX, PUSH CX, PUSH DS, and PUSH [4000]. The object code for PUSH AX is 50H, for PUSH BX is 53H, for PUSH CX is 51H, for PUSH DS is 1E, for PUSH [4000] is FF 36 00 40.

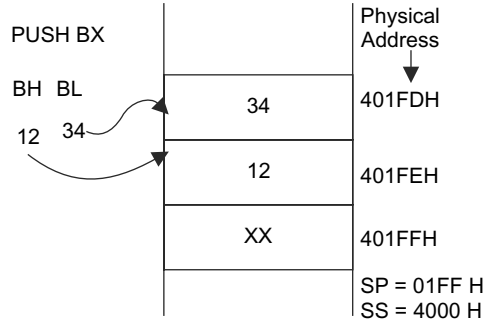


Fig. 6.18 The operation of PUSH BX instruction

PUSHF (Push flags word onto stack)

SP = SP-2; SS: [SP] ← flags, SP = SP-2; SS:[SP] ← Source, Flag affected: None

The push flag instruction pushes the content of the flag register on the stack. First the upper byte FLAG_U and then the lower byte FLAG_L is pushed on it. The SP is decremented by 2 for each push operation. The general operation of this operation is similar to the PUSH operation. The object code of PUSHF is 100 111 00 = 9C.

POP destination (Pop word at top of stack to destination)

Destination ← SS:[SP]; SP = SP + 2, Flag affected: None

When this instruction is executed, it loads the specified register/memory location with the contents of the memory location of which the address is formed using the current stack segment SS and the stack pointer SP. The stack pointer is incremented by 2. The operation of POP instruction is exactly opposite the PUSH instruction. The actual operation of POP BX instruction is shown in Fig. 6.16. The sequence of POP operation is as follows:

- ✦ The content of stack-top memory location is stored in the BL register and SP is incremented by one.
- ✦ Then contents of memory location as pointed by SP are copied to the BH register and SP is also incremented by 1.

Hence SP is incremented by 2 and points to the next stack top as depicted in Fig. 6.19. Assume 12H is stored in 410FEH and 34H is stored in 410FDH, the content of SS = 4000 and SP = 01FD. After execution of POP BX instruction, the content of the memory location 410FDH is copied into BL and content of the memory location 410FEH is also copied into BH.

The object code of POP is

For register/memory	1000 1111	Mod 000 R/M
For register	0101 1 reg	
For Segment register	000 reg 111	

The examples of POP instructions are POP AX, POP BX, POP CX, POP DS, and POP [4000]. The object code for POP AX is 58H, for POP BX is 5BH, for POP CX is 59H, for POP DS is 1F, for POP [4000] is 8F 06 00 40.

POPF (Pop word at top of stack to flags register)

flags ← SS:[SP]; SP = SP + 2, Flag affected: All
 The pop flags instruction loads the flag register completely from word contents of the memory location currently addressed by SP and SS. The SP is incremented by 2 for each pop operation.

The object code of POPF is 100 111 01 = 9D

Example 6.10

Write instructions for the following operations:

- (i) Push the content of AX register on to the stack.
- (ii) Push the content of memory location offset address 0500 on to the stack.
- (iii) Store the content of stack top memory locations in AL and AH registers.
- (iv) Store the 16-bit flags on to the stack.
- (v) Pop the top of the stack into the 16-bit flag word.

Solution

- (i) PUSH AX; Push the content of the AX register on to the stack.
- (ii) PUSH [0500]; Push the content of memory location offset address 0500 on to the stack.
- (iii) POP AX; Store the content of stack-top memory locations in AL and AH registers.
- (iv) PUSHF; Store the 16-bit flags on to the stack.
- (v) POPF; Pop the top of the stack into the 16-bit flag word.

6.3.4 Arithmetic Instructions

These instructions perform the arithmetic operations such as addition, subtraction, increment, decrement, negation, multiplication, division and comparing two values. The ASCII adjustment and decimal adjust instructions also belong to this type of instructions. The 8086/8088 instructions that handle these operations are ADD, ADC, SUB, SBB, INC, DEC, NEG, MUL, IMUL, DIV, IDIV, and other instructions such as AAA, AAD, AAM, AAS, DAA, and DAS. In this section, all arithmetic instructions are discussed below in significant details.

ADD Destination, Source

(Add two operands, result remains in destination)

Destination ← (Source + Destination), Flag affected: O S Z A P C

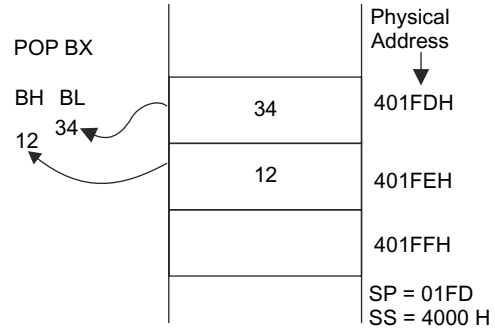


Fig. 6.19 The operation of POP BX instruction

The ADD instruction adds the contents of source operand (register or a memory location) specified in the instruction or an immediate data to the contents of destination (another register or memory location). After addition, the result is in the destination operand. But both the source and destination operands cannot be memory operands. It means that memory-to-memory addition is not possible. After addition, the condition code flags O, S, Z, A, P and C are affected, depending upon the result. The object code of the ADD instruction is as follows:

Register/Memory with Register	0000 00dw	Mod reg R/M		
Immediate to register/memory	1000 00sw	Mod 000 R/M	data	data
Immediate to Accumulator	0000 010w	data	data	

For example, ADD AX, 0100H instruction can add 16-bit immediate data (0100H) with the content of AX register and result is stored in the AX register. The object code 0000 010w, 16-bit data. Here $w = 1$, the object code is 05, 00, 01.

The example of other ADD instructions are ADD AL, 22H, ADD AX, BX, ADD AL, [BX], ADD [BX],CL and ADD [BX],CX. The object code for ADD AL, 22H is 04 22, for ADD AX, BX is 01 D8, for ADD AL, [BX] is 02 07, for ADD [BX], CL is 00 0F and for ADD [BX], CX is 01 0F.

ADC Destination, Source

(Add two operands with carry from previous add)

Destination \leftarrow (Source + Destination + CF), Flag affected: O S Z A P C

The ADC instruction performs the same operation as ADD instruction, although the carry flag bit is added with the result. All the condition flags are affected after execution of this instruction. The object code of ADC instructions are as follows:

Register/Memory with Register	0001 00dw	Mod reg R/M		
Immediate to register/memory	1000 00sw	Mod 010 R/M	data	data
Immediate to Accumulator	0001 010w	data	data	

The examples of ADC instructions are ADC AX, 1234H; ADC AX, CX; ADC AX, [SI]; ADC AX, 4000]; ADC [SI], AX; and ADC [4000], BX. The object code for ADC AX, 1234H is 15 34 12; for ADC AX, CX is 11 C8; for ADC AX, [SI] is 13 04; for ADC AX, [4000] is 13 06 00 40; for ADC [SI], AX is 11 04; and for ADC [4000], BX is 11 1E 00 40.

SUB Destination, Source

(Subtract source from destination, store result in destination)

Destination \leftarrow (Destination-Source), Flag affected: O S Z A P C

The *SUB destination, source* instruction subtracts the source operand from the destination operand and the result is stored in the destination operand. The source operand may be a register, memory location or immediate data. The destination operand may be a register or a memory location. But in an instruction, source and destination operands both will not be memory operands and the destination operand must not be an immediate data. After execution of this instruction, all the condition code flags, O, S, Z, A, P and C are affected.

The object code of SUB instruction is as follows:

Register/Memory with Register	0010 10dw	Mod reg R/M		
Immediate to register/memory	1000 00sw	Mod 101 R/M	data	Data
Immediate to Accumulator	0010 110w	data	data	

For example, SUB AX, 0100 Load 0100H to the AX register immediately. The object code is

0010 110w	data	data
-----------	------	------

Here $w = 1$, object is 0010 1101, 16-bit data = 2D, 00, 01H

The other examples of SUB instructions are SUB AL, 44H; SUB AX, BX; SUB AL, [BX]; SUB [BX], CL and SUB [BX], CX. The object code for SUB AL, 44H is 2C 44; for SUB AX, BX is 29 D8; for SUB AL, [BX] is 2A 07; for SUB [BX], CL is 28 0F and for SUB [BX], CX is 29 0F.

SBB Destination, Source (Subtract source and the carry flag bit from destination)

Destination ← ((Destination – Source) – CF) Flag affected: O S Z A P C

The SBB represents subtract with borrow. In this instruction, subtracts the source operand and the borrow flag which is the result of the previous operations, from the destination operand. The subtraction with borrow means that subtract 1 from the subtraction obtained by SUB. After subtraction, if carry is generated, carry flag is set. The result is stored in the destination operand. All the flags O, S, Z, A, P and C are affected by this instruction. The object code is

Register/Memory with Register	0001 10dw	Mod reg R/M		
Immediate to register/memory	1000 00sw	Mod 011 R/M	data	Data
Immediate to Accumulator	0000 111w	data	data	

For example, SBB AX, 0010. Subtract 0010H and the carry flag from AX register immediately. The object code is

0000 111w	data	data
-----------	------	------

Here $w = 1$, object is 0000 1111, 16-bit data = 0F, 00, 01H

The other examples of SBB instructions are SBB AX, BX; SBB AL, [BX]; SBB [BX], CL and SBB [BX], CX, and SBB AX, [4000]. The object code for SBB AX, BX is 19 D8; for SBB AL, [BX] is 1A 07; for SBB [BX], CL is 18 0F; for SBB [BX], CX is 19 0F and for SBB AX, [4000] is 1B 06 00 40.

Example 6.11

Write instructions for the following operations:

- (i) Add 2345 to the contents of the AX register
- (ii) Add 22H to the content of the specified memory location represented by the contents of the BX register
- (iii) Subtract the content of the AX register from the AX register
- (iv) Subtract immediately 2345 from the BX register with borrow
- (v) Subtract immediately 1000 from memory with offset address 0100H

Solution

- (i) ADD AX, 2345; Add 2345 to the contents of the AX register
- (ii) ADD [BX], 22; Add 22H to the content of the specified memory location by the BX register
- (iii) SUB AX, BX; Subtract the content of the AX register from AX register
- (iv) SBB AX, 2345; Subtract immediately 2345 from BX register with borrow
- (v) SUB [0100], 1000; Subtract immediately 1000 from memory with offset address 0100H

INC Destination (Add 1 to destination)

Destination ← (Destination + 1), Flag affected: O S Z A P

When this instruction is executed, the contents of the specified register or memory location increases by 1. After execution, the condition flags O, S, Z, A and P are affected but the carry flag is not affected by this instruction. In this instruction, immediate data cannot be operand. The object code of instruction is

Register/Memory

1111 111w	Mod 000 R/M
-----------	-------------

Register

01 000 reg

For example, INC AX. The object code is

01 000 reg

Here reg = 000 for the AX register. Then object code is 0100 0000 = 40

Other examples of INC instructions are INC BX; INC CX; INC DX and INC [BX]. The object code for INC BX is 43; for INC CX is 41; for INC DX is 42 and for INC [BX] is FF 07.

DEC Destination (Decrement destination by 1)

Destination \leftarrow (Destination - 1), Flag affected: O S Z A P C

This instruction decrements the contents of the specified register or memory location by one or subtracts 1 from the contents of the specified register or memory location. After execution, all the condition flags O, S, Z, A, P and C are affected depending upon the result. But the carry flag is not affected. In this instruction, immediate data can not be used as operand. The object code of instruction is

Register/Memory

1111 111w	Mod 001 R/M
-----------	-------------

Register

01 001 reg

For example, DEC AX. The object code is

01 001 reg

Here reg = 000 for AX register. Then object code is 0100 1000 = 48

Other examples of INC instructions are DEC BX; DEC CX; DEC DX and DEC [BX]. The object code for DEC BX is 4B; for DEC CX is 49; for DEC DX is 4A and for DEC [BX] is FF 0F.

NEG Destination (Changes the sign of an operand (Negate))

Destination \leftarrow (0-Destination), Flag affected: O S Z A P C

This instruction performs a 2's complement of destination. To obtain 2's complement, it subtracts the contents of the destination from zero. Then result is stored in the destination operand which may be a register or a memory location. After execution of this instruction, all the condition flags O, S, Z, A, P and C are affected. While OF is set, it means that the operation has not been completed successfully. The object code is

1111 011w	Mod 011 R/M
-----------	-------------

The example of this instructions are NEG AX; NEG BX; NEG CX; NEG DX; NEG AL; NEG BL; NEG CL and NEG DL. The object code for NEG AX is F7 D8; for NEG BX is F7 DB; for NEG CX is F7 D9; for NEG DX is F7 DA; for NEG AL is F6 D8; for NEG BL is F6 DB; for NEG CL is F6 D9 and for NEG DL is F6 DA.

CMP Destination, Source (Compare by subtracting source from destination)

Destination-Source; Flag affected: O S Z A P C

This instruction performs a nondestructive subtraction of source from destination but the result is not stored. Actually, the source operand and destination operand are compared. The source operand will be a register or an immediate data or a memory location and the destination operand may be register or a memory location. After comparison, the result will not be stored anywhere but the flags are affected depending upon the result of the subtraction. When both the source and destination operands are equal, zero flag is set. While the source operand is greater than the destination operand, carry flag is set; otherwise carry flag is reset. The object code is

Register/Memory with Register	0011 10dw	Mod reg R/M	
Immediate to register/memory	1000 00sw	Mod 111 R/M	data data
Immediate to Accumulator	0011 110w	data	data

For example, CMP AX, 0100 Compare 0100H with the content of AX register immediately. The object code is

0011 110w	data	data
-----------	------	------

Here $w = 1$, object is 0011 110w, 16-bit data = 0011 1101, 16-bit data = 3D, 00, 01H

The other examples of CMP instructions are CMP BX, 1234; CMP AL, 22; CMP BX, [SI]; CMP [0100], BX and CMP [BX], CX. The object code for CMP BX, 1234 is 81 FB 34 12; for CMP AL, 22 is 3C 22; for CMP BX, [SI] is 3B 1C; for CMP [0100], for BX is 39 1E 00 01 and for CMP [BX], CX is 39 0F.

Example 6.12

Write instructions for the following operations:

- Compare 16-bit immediately available data (4567H) from the AX register
- Increment the contents of the CX register by one
- Decrement the contents of memory location specified by the BX register
- 2's complement of the accumulator
- Compare 8-bit immediately available data (FFH) from contents of memory location specified by source index address 0400

Solution

- CMP AX,4567 ; Compare 16-bit immediately available data (4567H) from the AX register
- INC CX; Increment the contents of the CX register by one
- DEC [BX]; Decrement the contents of memory location specified by the BX register
- NEG AX; 2's complement of the accumulator
- MOV SI,0400; load 0400 in SI
- CMP [SI],FF; Compare 8-bit data (FFH) with the contents of memory location specified by source index

6.3.5 Multiplication and Division Instructions

MUL Source (Multiply 8- or 16-bit source by 8-bit (AL) or 16-bit (AX) value (unsigned))

$AX \leftarrow (AL * \text{source } 8)$

$DX:AX \leftarrow (AX * \text{source } 16)$, Flag affected: O C; the S, Z, A, and P flags are left in an indeterminate condition

This instruction is an unsigned byte or word multiplication by the contents of AL or AX. An 8-bit source is multiplied by the contents of AL to generate a 16-bit result in AX. A 16-bit source is multiplied by the contents of AX to generate a 32-bit result. The most significant word of the result is stored in DX and the least significant word of the result is stored in AX. The unsigned byte or word will be one of the general purpose registers or memory locations. All the flags are modified depending upon the result. In this instruction immediate operand is not allowed.

The object code of MUL instruction is

1111 011w	Mod 100 R/M
-----------	-------------

Here, mod and R/M are for memory/register; i.e., for the second operand. The first operand is always in AL or AX.

The example is MUL BL. Assume the content of the AL register is 22, and register of BL is 11H

The object code = 1111 011w, mod 100 R/M = 1111 0110, 11 100011 = F6, E3 as Mod and R/M for BL are 11 and 011 respectively and $w = 0$.

The other examples of MUL instructions are MUL CL; MUL BX; MUL CX; MUL DX and MUL [BX+10]. The object code for MUL CL is F6 E1; for MUL BX is F7 E3; for MUL CX is F7 E1; for MUL DX is F7 E2 and for MUL [BX + 10] is F7 67 10.

IMUL Source (Multiply 8-bit or 16-bit source by 8-bit (AL) or 16-bit (AX) value (signed))

$AX \leftarrow (AL * \text{source } 8)$

DX: $AX \leftarrow (AX * \text{source } 16)$, Flag affected: O C; the S, Z A and P flags are left in an indeterminate condition.

This instruction is a signed multiplication of two signed numbers. A signed byte in source operand is multiplied by the contents of AL to generate a 16-bit result in AX. The source can be a general-purpose register, memory operand, index register or base register, but it cannot be an immediate data. A 16-bit source operand is multiplied by the contents of AX to generate a 32-bit result. In case of 32 bit results, the higher-order word or the higher 16 bits is stored in DX and the lower-order word or the lower 16 bits is stored in AX. The AF, PF, SF and ZF flags are undefined after IMUL. If AH and DX contain parts of 16-bit and 32-bit results respectively, CF and OF both will be set. AL and AX are the implicit operands in case of 8-bit and 16-bit multiplications respectively. The unused higher bits of the result are filled by the sign bit and CF, AF are cleared.

The object code of IMUL is

1111 011w	Mod 101 R/M
-----------	-------------

Here, Mod and R/M are for the second operand which is either memory or register. The first operand is always in AL or AX. During 8-bit multiplication, 7 bits are used to represent a number and the eighth bit represents its sign. When the sign bit is 0, it represents a positive number. If the sign bit is 1, it represents a negative number. In case of 16-bit multiplication, 15 bits are used to represent a number and the sixteenth bit represents its sign.

The example of IMUL instruction is IMUL BL. Here $w = 0$, Mod and R/M for BL are 11, R/M = 001 respectively. Then object code = 1111 011w, mod 101 R/M = 1111 0110, 11 101 011 = F6, EB

The other examples of IMUL instructions are IMUL CL; IMUL BH; IMUL BX; IMUL CX; IMUL DX and IMUL [BX+10]. The object code for IMUL CL is F6 E9; for IMUL BH is F6 EF; for IMUL BX is F7 EB; for IMUL CX is F7 E9; for IMUL DX is F7 EA and for IMUL [BX+10] is F7 2F 10.

DIV Source (Divide of 16-bit or 32-bit number by 8- or 16-bit number (unsigned))

$AL \leftarrow (AX \div \text{Source } 8)$

AH \leftarrow Remainder

$AX \leftarrow (DX: AX \div \text{Source } 16)$

DX \leftarrow Remainder

Flag affected: The O, S, Z, A, P, and C flags are left in an indeterminate condition.

This is an unsigned divide instruction. This instruction is used to divide a 16-bit unsigned number by an 8-bit unsigned number. When a 16-bit number in AX is divided by an 8-bit source operand, the quotient is stored in AL and the remainder is stored in AH. If the result is too big to fit in AL, a divide by zero (type 0) interrupt is generated.

This instruction is also used to divide a 16-bit unsigned number by a 16-bit or 8-bit operand. The divided must be in AX for 16-bit operation and divisor may be specified using any one of the addressing modes except immediate. A 32-bit number in DX: AX is divided by a 16-bit source with the quotient remaining in

AX and the remainder in DX. When the quotient of a 16-bit operation is greater than FFFFH, a divide-by-zero (type 0) interrupt is generated. This instruction does not affect any flag.

The object code DIV instruction is

1111 011w	Mod 110 R/M
-----------	-------------

The example of DIV instruction is DIV BL. This instruction consists of 8-bit divisor in BL and AX contains 16-bit dividend. The object code is 1111 011w, Mod 110 R/M = 1111 0110, 11 110011 = F6, F3 as Mod = 11 and R/M = 011 for BL and $w = 0$.

The other examples of DIV instructions are DIV CL; DIV BX; DIV CX; DIV DX and DIV [BX+10]. The object code for DIV CL is F6 F1; for DIV BX is F7 F3; for DIV CX is F7 F1; for DIV DX is F7 F2 and for DIV [BX+10] is F7 37 10.

IDIV Source (Divide of signed 16-bit or 32-bit number by 8- or 16-bit number (signed))

AL \leftarrow (AX \div source 8)

AH \leftarrow Remainder

AX \leftarrow (DX: AX \div source 16)

DX \leftarrow Remainder

Flag affected: The O, S, Z, A, P, and C flags are left in an indeterminate condition.

This is a signed divide. This instruction performs the same operation as DIV instruction. A 16-bit value in AX is divided by an 8-bit source with the quotient remaining in AL and the remainder in AH. If the result is too big to fit in AL, a divide by zero (type 0) interrupt is generated.

A 32-bit number in DX: AX is divided by a 16-bit source with the quotient remaining in AX and the remainder in DX. Divide by 0 interrupt is generated, if the result (quotient) is too big to fit in AX, a divide by zero (type 0) interrupt is generated. All the flags are undefined after IDIV instruction.

The object code of IDIV is

1111 011w	Mod 111 R/M
-----------	-------------

The example of IDIV instruction is IDIV BL. The operation of this instruction is dividing AX by CL, both operands are signed numbers. The object code is 1111 011w, Mod 111 R/M = 1111 0110, 11 111011 = F6, FB as Mod = 11 and R/M = 011 for BL and $w = 0$.

The other examples of DIV instructions are IDIV BH; IDIV CL; IDIV BX; IDIV CX ; IDIV DX and IDIV [BX+10]. The object code for IDIV BH is F6 FF; for IDIV CL is F6 F9; for IDIV BX is F7 FB; for IDIV CX is F7 F9; for IDIV DX is F7 FA and for IDIV [BX+10] is F7 3F 10.

Example 6.13

Write instructions for the following operations:

- (i) Multiply the content of AL by the content of CL
- (ii) Multiply the content of AX by the content of CX
- (iii) Signed multiplication of AL and DL
- (iv) Divide AX by the content of memory location represented by BX
- (v) Signed division of AX and BL

Solution

- (i) MUL CL; Multiply the content of AL by the content of CL
- (ii) MUL CX; Multiply the content of AX by the content of CX
- (iii) IMUL DL; Signed multiplication of AL and DL
- (iv) DIV [BX]; Divide AX by the content of memory location represented by BX
- (v) IDIV BL; Signed division of AX and BL

6.3.6 Arithmetic Adjust Instructions

DAA (Decimal adjustment after addition)

$AL \leftarrow$ (AL adjusted for BCD addition), Flag affected: S Z A P C; the O flag is left in an indeterminate condition.

The DAA instruction is used to transfer the result of the addition of two packed BCD numbers to a valid BCD number. The result will be stored in the AL register only. If after addition, the lower nibble is greater than 9, AF is set. Then 06 will be added to the lower nibble in AL. After addition of 06 in the lower nibble of AL, if the upper nibble of AL is greater than 9 or if the carry flag is set, 60H will be added to AL through DAA instruction. After execution of this instruction, AF, CF, PF, and ZF flags are affected. The OF is undefined.

The object code of DAA is 00100111 = 27H

Example 6.14 Write instructions to add two numbers 54 and 26 and use DAA for adjustment of the result.

Solution

- (i) MOV AL, 54; 54 in AL.
- (ii) MOV BL, 26; 26 in BL.
- (iii) ADD AL, BL; add AL and BL, content of AL = 7A.
- (iv) DAA; adjust result in BCD. AL 80 = 7A + 06

DAS (Decimal adjust for subtraction)

$AL \leftarrow$ (AL adjusted for BCD subtraction); Flag affected S Z A P C; the O flag is left in an indeterminate condition.

The DAS instruction is used to convert the result of subtraction of two packed BCD numbers to a valid BCD number. The subtraction will be stored in the AL register only. While the lower nibble of AL is greater than 9, 06 will be subtracted from lower nibble of AL. If the result of subtraction sets the carry flag or if the upper nibble is greater than 9, 60H will be subtracted from AL. AF, CF, SF, PF and ZF flags are affected after execution of this instruction. The OF is undefined after DAS instruction.

The object code of DAS is 00101111 = 2FH

AAA (ASCII adjust for addition)

$AL \leftarrow$ (AL adjusted for ASCII addition); Flag affected: A, C; the O, S, Z, and P flags are left in an indeterminate state.

This instruction follows an addition of 'unpacked' ASCII data. After execution of ADD instruction, this AAA instruction is executed for ASCII adjustment of result of addition of two numbers. The result will be stored in the AL register. The AAA instruction converts the resulting contents of AL to unpacked decimal digits. When the AAA instruction is executed, the lower 4 bits of AL will be checked whether it is a valid BCD number in the range 0 to 9. If the lower 4 bits of AL is between 0 to 9 and AF is zero, AAA sets the higher order 4 bits of AL to 0. The content of AH must be cleared before addition. If the value in the lower 4 bits of AL is greater than 9 then the AL is incremented by 06, AH is incremented by 1, the AF and CF flags are set to 1, and the higher 4 bits of AL are cleared to 0. After the addition of 05H and 09H, the result 0E is stored in AL. As lower nibble of AL (E) is greater than 9, the AL is to be incremented by 06 and AH is incremented by 1. Hence the content of AL is 04H and the content of AH is 01H.

The object code of AAA is 00110111 = 37H.

AAS (ASCII adjust for subtraction)

AL ←(AL adjusted for ASCII subtraction); Flag affected: A C; the O, S, Z, and P flags are left in an indeterminate state.

This instruction follows a subtraction of 'unpacked' ASCII data. The AAS instruction is used to convert the result in the AL register after subtracting two unpacked ASCII operands. The result is stored in the AL register which is an unpacked decimal number. When the lower 4 bits of the AL register are greater than 9 or the AF flag is set or 1, the AL will be decremented by 6 and the AH register is decremented by 1, the CF and AF are set to 1. If not, the CF and AF are set to 0, the result does not require any correction. Hence, the upper nibble of AL is 0 and lower nibble may be any number from 0 to 9.

The object code of AAS is 00111111 = 3FH.

AAM (ASCII adjust for multiplication)

AH: AL ←(AH: AL adjusted for ASCII multiplication); Flag affected: S Z P; the O, A, and C flags are left in an indeterminate condition.

The AAM instruction is executed to convert the product available in AL into an unpacked BCD format. The AMM (ASCII Adjustment after multiplication) instruction follows a multiplication instruction that multiplies two unpacked BCD numbers, i.e., and higher nibbles of the multiplication number should be 0. Usually, the multiplication is performed using MUL instruction and the result of multiplication is available in AX. After execution of AAM instruction, the content of AH is replaced by tens of the decimal multiplication and the content of AL is replaced by ones of the decimal multiplication. The object code of AAM is 1101 0100 0000 1010 = D4 0A.

Example 6.15

Write instructions to multiply two unpacked BCD numbers 4 and 6 and use AAM for adjustment of the result.

Solution

- (i) MOV AL, 04; 04 in AL.
- (ii) MOV CL, 06; 06 in CL.
- (iii) MUL CL; multiply AL by CL, content of AL = 18.
- (iv) AAM; adjust result in BCD. AH ←01 and AL ←08.

CBW (Convert from byte to word (16-bit ← 8-bit))

AH← (filled with bit-7 of AL), AX ←(AL * source 8) Flag affected: None

This instruction converts a signed byte in AL to a signed word in AX. Actually, it copies the sign bit of a byte to be converted to all the bits in the higher byte of the result word. Flags are not affected after execution of CBW. The object code of CBW is 1001 1000 = 98H.

CWD (Convert from word to double word)

DX ← (filled with bit-15 of AX), AX ←(AL * source 8), Flag affected: None

Converts a 16-bit word in AX to a 32-bit word in DX: AX by sign extension of bit 15 of AX through DX. Usually, this operation is to be done before signed division. Flags are not affected after execution of CWD. The object code of CWD is 1001 1001 = 99H.

6.3.7 Logical and Bit Manipulation Instructions

These type of instructions are used for

- ✦ Basic logical operations such as NOT, AND, OR, and XOR;
- ✦ Bit by bit shift operations such as SHL (shift logical left), SHR (shift logical right), SAL (shift arithmetic left), and SAR (shift arithmetic right); and
- ✦ Rotate operations such as ROR (rotate right without carry), ROL (rotate left without carry), RCR (rotate right through carry), and RCL (rotate left through carry).

After execution of above instructions, all the condition-code flags are affected depending upon the result. In this section, the operations of logical and bit manipulation instructions are discussed in detail.

NOT Destination (1's complement of destination)

Destination ← (~Destination), Flag affected: None

Converts 1's to 0's and 0's to 1's in destination.

The NOT instruction is used to generate complement of the contents of an operand register or a memory location, bit by bit.

The object code of NOT is

1111 011w	Mod 010 R/M
-----------	-------------

The example of NOT instruction is NOT AL. The object code of NOT AL is 1111 011w 1101 000 = F6 D0 as Mod = 11 and R/M = 000 for AL and $w = 0$. The other NOT instructions are NOT BL, NOT CL, NOT DL, NOT AX, NOT BX, NOT CX, and NOT [BX]. The object code for NOT BL is F6 D3; for NOT CL is F6 D1; for NOT DL is F6 D2; for NOT AX is F7 D0; for NOT BX is F7 D3; for NOT CX is F7 D1, and for NOT [BX] is F7 17.

AND Destination, Source (Logical AND)

Destination ← (Destination AND Source)

This instruction performs a bitwise logical AND of source and destination with the result remaining in the destination. The source operand may be immediate data or a register or a memory location and the destination operand may be a register or a memory location. For AND operation, at least one of the operands must be a register or a memory operand. For this instruction, both the operands will not be memory locations and immediate operands and a destination operand should not be an immediate operand.

The object code of AND instruction is as follows:

Register/Memory with register	0010 00dw	Mod reg R/M		
Immediate to register/memory	1000 000w	Mod 100 R/M	data	Data
Immediate to accumulator	0010 010w	data	data	

The example of AND instruction is AND AX, 045B and its object code is 0010 010w 5B 04 = 25 5B 04 as $w = 1$. The other examples are AND AX, BX; AND CX, DX; AND AX, [BX] and AND AX, [SI]. The object code for AND AX, BX is 21 D8; for AND CX, DX is 21 D1; for AND AX, [BX] is 23 07 and for AND AX, [SI] is 23 04.

OR Destination, Source (Logical OR)

Destination ← (Destination OR Source)

The OR instruction performs a bitwise logical OR of source and destination with the result remaining in the destination. The OR operation is same as described in case of AND operation. The limitations of OR instruction based on source and destination operands are also the same as in case of AND operation.

The object code of OR instruction is as follows:

Register/Memory with Register	0000 10dw	Mod reg R/M		
Immediate to register/memory	1000 000w	Mod 001 R/M	data	data
Immediate to accumulator	0000 110w	data	data	

The example of OR instruction is OR AX, 2345 and its object code is 0000 110w 45 23 = 0D 45 23 as $w = 1$. The other examples are OR AX, BX; OR CX, DX; OR AX, [BX] and OR AX, [SI]. The object code for OR AX, BX is 09 D8; for OR CX, DX is 09 D1; for OR AX, [BX] is 0B 07 and for OR AX, [SI] is 0B 04.

XOR Destination, Source (Exclusive logical OR)

Destination \leftarrow (Destination XOR Source)

The XOR instruction performs a bitwise logical exclusive OR of source and destination with result remaining in destination. This instruction is carried out in a similar way to the AND and OR operation. The limitations of XOR instruction are also the same as in case of AND/OR operation.

The object code of XOR instruction is as follows:

Register/Memory with Register	0011 00dw	Mod reg R/M		
Immediate to register/memory	1000 000w	Mod 110 R/M	data	Data
Immediate to Accumulator	0011 010w	data	data	

The example of XOR instruction is XOR AX, 1234 and its object code is 0011 010w 34 12 = 35 34 12 as $w = 1$. The other examples are XOR AX, BX; XOR CX, DX; XOR AX, [BX] and XOR AX, [SI]. The object code for XOR AX, BX is 31 D8; for XOR CX, DX is 31 D1; for XOR AX, [BX] is 33 07 and for XOR AX, [SI] is 33 04.

TEST Destination, Source (Non-destructive logical AND)

Flags \leftarrow (Destination AND Source)

The TEST instruction performs a nondestructive bitwise logical AND of source and destination, setting flags and leaving destination unchanged. The result of this ANDing operation will not be available, but the flags are affected. Generally, OF, CF, SF, ZF and PF flags are affected. The source operands may be a register or a memory or immediate data and the destination operands may be a register or a memory.

The object code of TEST is given below:

Register/Memory and Register	1000 010w	Mod reg R/M		
Immediate data and register/memory	1000 000w	Mod 110 R/M	data	data
Immediate data and Accumulator	1010 100w	data	data	

The example of TEST instruction is TEST AX, 6789 and its object code = 1010 100w 89 67 = A9 89 67 as $w = 1$. The other example of TEST instructions are TEST AX, BX; TEST CX, DX; TEST AX, [BX]; TEST AX, [DI]. The object code for TEST AX, BX is 85 C3; for TEST CX, DX is 85 CA; for TEST AX, [BX] is 85 07; and for TEST AX, [DI] is 85 05.

Example 6.16

Write instructions for the following operations:

- (i) 1's complement of the content of the DX register
- (ii) AND 1234H with the content of the AX register

- (iii) XOR operation between AL and DL registers
- (iv) OR operation between BX and CX registers
- (v) Perform TEST operation between AL and BL registers

Solution

- (i) NEG DX; 1’s complement of the content of the DX register
- (ii) AND AX, 1234; AND 1234H with the content of the AX register
- (iii) XOR AL,DL; XOR operation between AL and DL registers
- (iv) OR BX,CX; OR operation between BX and CX registers
- (v) TEST AL,BL; Perform TEST operation between AL and BL registers

SHL/SAL (Shift Logical/ Arithmetic Left)

$$A_{n+1} \leftarrow A_n, A_{15} \leftarrow A_{14}, A_0 \leftarrow 0, CF \leftarrow A_{15}, \text{ All flags are affected}$$

These instructions shift each bit in the destination operand (word or byte) to the left and insert zeros in the newly introduced least significant bits. The highest order bit shifts into the carry flag as shown in Fig. 6.20. The common format of SHL/SAL instruction is SAL Operand-1, Operand-2.

The operand-1 will be the content of register or the content of memory. The number of shifts is set by operand-2. The operand-2 will be an immediate data or content of CL register. The object code of SHL/SAL instruction is

1101 00vw	Mod 100 R/M
-----------	-------------

The example of SHL instructions are SHL AX, CL and SHLAX, 1. Since shifting an integer to the left one position is equivalent to the multiplication of specified operand by 2. Actually, the shift left instruction for multiplication by powers of two as given below:

- SHL AX, 1; Result is equivalent to AX*2
- SHL AX, 2; Result is equivalent to AX*4
- SHL AX, 3; Result is equivalent to AX*8
- SHL AX, 8; Result is equivalent to AX*256

Assume the content of AX register is 1010 1010 1010 1010 = AAAA. After execution of SHL AX, 1 the content of AX will be 5554 and CY flag set. The object code of SHL AX, 1 is D1 E0. After execution of SHL AX, 2 the content of AX will be AAA8. All flags are affected depending upon the result.

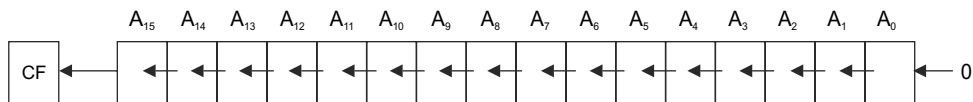


Fig. 6.20 Shift left operation

SHR (Shift logical right) $A_n A_{n+1}, CF \leftarrow A_0, A_{15} \leftarrow 0, \text{ All flags are affected}$

This instruction performs bitwise right shifts on the destination operand word or byte and inserts zeros in the shifted positions. The general format of SHR instruction is SHR Operand-1, Operand-2. The operand-1 may be a register or a memory location. The number of shifts is set by operand-2. The operand-2 will be an immediate data or content of CL register. The result is always stored in the destination operand. Figure 6.21 shows the shift right operation.

The object code of SHR instruction is



The example of SHR instructions are SHR AX, CL and SHR AX, 1. If the SHR instruction shifts an integer to the right one position, it performs an unsigned division of the destination operand by 2. Actually, each shift to the right is equivalent to dividing the value by 2 as given below:

SHR AX, 1; Result is equivalent to $AX/2$

SHR AX, 2; Result is equivalent to $AX/4$

SHR AX, 3; Result is equivalent to $AX/8$

SHR AX, 8; Result is equivalent to $AX/256$

When the content of the AX register is 1010 1010 1010 = AAAA, after execution of SHR AX, 1 the content of AX will be 5555. The object code of SHR AX,1 is D1 E8. After execution of MOV CL, 02 and SHR AX, CL the content of AX will be 2AAA. All flags are affected depending upon the result. This shift operation shifts the operand through the carry flag.

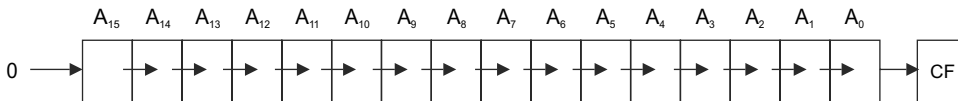


Fig. 6.21 Shift right operation

SAR (Shift arithmetic right)

$$A_n \leftarrow A_{n+1}, CF \leftarrow A_0, A_{15} \leftarrow A_{15}, \text{ All flags are affected}$$

The SAR instruction performs right shifts all the bits in the destination operand (word or byte) to the right one bit. This instruction is replicating the most significant bit of the operand in the newly inserted positions.

The common format of SAR instruction is SAR Operand-1, Operand-2. The operand-1 may be a register or a memory location. The number of shifts is set by the operand-2. The operand-2 will be an immediate data or content of the CL register. The result is always stored in the destination operand. Figure 6.22 shows the arithmetic shift right operation.

The object code of SAR instruction is



The example of SAR instructions are SHR AX, CL and SAR AX,1. If the SHR instruction shifts an integer to the right one position, it performs an unsigned division of the destination operand by 2. Actually, each shift to the right divides the value by 2 as given below:

SAR AX, 1; Result is equivalent to signed division by 2

SAR AX, 2; Result is equivalent to signed division by 4

SAR AX, 3; Result is equivalent to signed division by 8

SAR AX, 8; Result is equivalent to signed division by 256

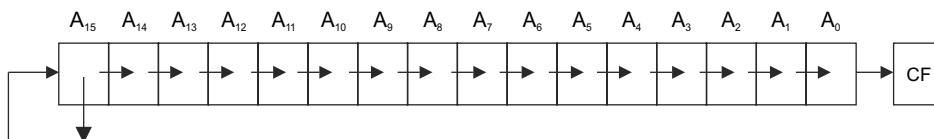


Fig. 6.22 Arithmetic shift right operation

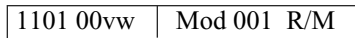
If the content of the AX register is 1010 1010 1010 1010 = AAAA, after execution of SAR AX, 1 the content of AX will be D555. The object code of SAR AX, 1 is D1 F8. After execution of SAR AX, 2 the content of AX will be EAAA. All flags are affected depending upon the result. This shift operation shifts the operand through the carry flag.

ROR(Rotate right without carry) $A_n \leftarrow A_{n+1}, A_{15} \leftarrow A_0, CF \leftarrow A_0$, All flags are affected

The ROR instruction rotates the contents of the destination operand to the right bit wise either by one or by the count specified in CL without carry. The least significant bit is stored into the carry flag and simultaneously it is transferred into the most significant bit position after each shift operation as shown in Fig. 6.23.

The common format of ROR instruction is ROR Operand-1, Operand-2. The operand-1 may be a register except segment register or a memory location. The operand-2 will be an immediate data or content of the CL register. The number of shifts is set by operand-2. The result is always stored in the destination operand.

The object code of ROR instruction is



The example of ROR instructions are ROR AX, CL and ROR AX, 1.

The PF, SF and ZF flags are left unchanged by this instruction.

Consider the content of the AX register is 1010 1010 1111 1010 = AAFA. After execution of ROR AX, 1 the content of AX will be 557D. The object code of ROR AX, 1 is D1 C8. After execution of MOV CL, 02 and ROR AX, CL the content of AX will be AABE.

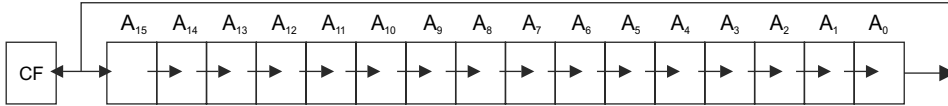


Fig. 6.23 Rotate right without carry

ROL (Rotate left without carry) $A_{n+1} \leftarrow A_n, A_0 \leftarrow A_{15}, CF \leftarrow A_{15}$, All flags are affected

The ROL instruction rotates the content of the destination operand to the left by one or by the specified number of bits in CL without carry. The most significant bit is pushed into the carry flag as well as the least significant bit position after each bit shift operation. The other bits are shifted left subsequently as depicted in Fig. 6.24. The PF, SF, and ZF flags are left unchanged by this operation. Its format is same as ROR. The object code of ROL instruction is



The example of ROL instructions are ROL AX, 1 and ROL AX, CL.

Assume the content of AX register is 1010 1010 1111 1010 = AAFA. After execution of ROL AX, 1 the content of AX will be 55F5. The object code of ROL AX,1 is D1 C0. After execution of MOV CL, 02 and ROL AX, CL the content of AX will be ABEA

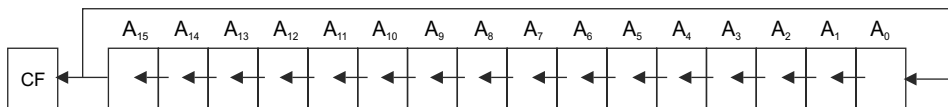


Fig. 6.24 Rotate left without carry

RCR (Rotate right through carry)

$$A_n \leftarrow A_{n+1}, A_{15} \leftarrow CF, CF \leftarrow A_0; \text{ All flags are affected}$$

This instruction rotates the contents of the destination operand bits right by one or by the specified number of bits in CL through the Carry Flag (CF). After each rotate operation, the carry flag is pushed into the MSB of the operand and the LSB is pushed into carry flag and the other bits are subsequently shifted right as given in Fig. 6.25. The SF, PF, ZF are left unchanged. Its format is same as ROR. The object code of RCR instruction is

1101 00vw	Mod 011 R/M
-----------	-------------

The example of RCR instructions are RCR AX, 1 and RCR AX, CL.

When the content of AX register is 1010 1010 1111 1010 = AAFA, after execution of RCR AX, 1 the content of AX will be 557D. The object code of RCR AX, 1 is D1 D8. After execution of MOV CL, 02 and RCR AX, CL the content of AX will be 2ABE.

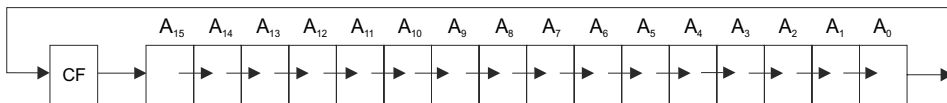


Fig. 6.25 Rotate right through carry

RCL (Rotate left Through Carry)

$$A_{n+1} \leftarrow A_n, CF \leftarrow A_{15}, A_0 \leftarrow CF \text{ All flags are affected}$$

The RCL instruction rotates the contents of the destination operand left by one or by the specified number of bits in CL through Carry Flag (CF). After each rotate operation, the carry flag is pushed into LSB and the MSB of the operand is pushed into carry flag. The remaining bits are subsequently shifted left as shown in Fig. 6.26. The SF, PF, ZF are left unchanged. The object code of RCL instruction is

1101 00vw	Mod 010 R/M
-----------	-------------

The example of RCL instructions are RCL AX, 1 and RCL AX, CL.

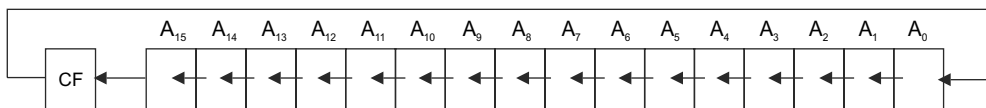


Fig. 6.26 Rotate left through carry

When the content of AX register is 1010 1010 1111 1010 = AAFA, after execution of RCL AX, 1 the content of AX will be 55F4. The object code of RCL AX, 1 is D1 D0. After execution of MOV CL, 02 and RCL AX, CL the content of AX will be ABE9.

Example 6.17

Write instructions for the following operations:

- (i) Shift content of BL left two times.
- (ii) Rotate content of BX left without carry one time.
- (iii) Shift logical right of the memory location represented by CS: SI one time.
- (iv) Rotate right without carry of AX registers three times.

Solution

- (i) MOV CL, 02
SHL BL, CL; Shift content of BL left two times
- (ii) ROL BX, 1; Rotate content of BX left without carry one time
- (iii) SHR [SI], 1; Shift logical right of the memory location represented by CS:SI one time
- (iv) MOV CL, 03
ROR AX, CL; Rotate right without carry of AX registers three times

6.3.8 Jump Instructions

The jump instructions are generally used to change the sequence of the program execution. There are two types of jump instructions, namely, conditional and unconditional. The conditional jump instructions transfer the program to the specified address when condition is satisfied only. The unconditional jump instructions transfer the program to the specified address unconditionally. All conditional and unconditional jump instructions are discussed in this section.

JMP target (Unconditional jump to target) This jump instruction unconditionally transfers the control of execution to the specified address using an 8-bit or 16-bit displacement or CS: IP. After execution of this instruction, no flags are affected. The jump instructions have different formats to specify the jump address.

Sort: $IP \leftarrow (IP + (\text{target displacement sign-extended}))$

Near: $IP \leftarrow (IP + (\text{target displacement}))$

Indirect: $IP \leftarrow (\text{register or value in memory})$

Far: $CS \leftarrow \text{targ_seg};$

$IP \leftarrow \text{targ_offset}, AX \leftarrow (AL * \text{source } 8)$

Flag affected: None

Short jumps are within ± 128 bytes of JMP instruction—only IP is affected.

Near jumps are within same segment—only IP is affected. Near jump allows a jump within ± 32 KB

Indirect jumps are within the same segment—only IP is affected.

Far jumps are to a different segment—both CS and IP are affected.

The object code of unconditional JMP instruction is

Direct with segment	1110 1001	disp-low	disp-high
Direct within segment short	1110 1011	disp	
Indirect within segment	1111 1111	Mod 100 R/M	
Direct intersegment	1110 1010	Offset-low	Offset-high
		Seg-low	Seg-high
Indirect intersegment	1111 1111	Mod 101 R/M	

JCXZ Target(short) (Jump short if CX register is 0) Flag affected: None
The object codes of JCXZ jump on CX zero instructions are

1110 0011	disp
-----------	------

Jcond (Jump on condition) $IP \leftarrow (IP + (\text{8-bit displacement sign-extended to 16 bits}))$, Flag affected: None

If conditional jump instructions are executed, program control can be transferred to the address specified by instruction itself. If the condition is not satisfied, instructions are executed sequentially. Here, condition is the status of flag. After execution of these instructions, no flags are affected. The address will be specified in the instruction which will be varied from -80H (-128) bytes to 7FH (127) bytes. Therefore, only short jumps can be implemented using conditional branch instructions. The conditions of jump instructions are given in Table 6.5.

Table 6.5 Conditional JUMP instructions

<i>Instruction</i>	<i>Condition</i>	<i>Operation</i>
JO	O = 1	Jump on overflow set
JNO	O = 0	Jump on overflow clear
JB / JNAE	C = 1	Jump if below / Jump if not above or equal
JAE / JNB	C = 0	Jump if above or equal / Jump if not below
JE / JNZ	Z = 1	Jump if equal / Jump if not zero
JNE / JZ	Z = 0	Jump if not equal / Jump if not zero
JBE / JNA	C = 1 or Z = 1	Jump if below or equal / Jump if not above
JA / JNBE	C = 0 and Z = 0	Jump if above / Jump if not below or equal
JS	S = 1	Jump on sign set
JNS	S = 0	Jump on sign clear
JP / JPE	P = 1	Jump on parity bit set (parity even)
JNP / JPO	P = 0	Jump on parity bit clear (parity odd)
JL / JNGE	S = 1 or O = 1	Jump if less / Jump if not greater than or equal to
JGE / JNL	S = 0	Jump if greater than or equal to / Jump if not less
JLE / JNG	Z = 1 or S and O = 1	Jump if less than or equal to / Jump if not greater than
JG / JNLE	Z = 0 or S = 0	Jump if greater than / Jump if not less than or equal to

The object codes of all conditional jump instructions are as follows:

JE/JZ Jump on equal/zero	0111 0100	disp
JL/JNGE Jump on less/not greater or equal	0111 1100	disp
JLE/JNZ Jump on less or equal/not greater	0111 1110	disp
JB/JNAE Jump on below/not above or equal	0111 0010	disp
JBE/JNA Jump on below or equal/not above	0111 0110	disp
JP/JPE Jump on parity/parity even	0111 1010	disp
JO Jump on overflow	0111 0000	disp
JS Jump on Sign	0111 1000	disp
JE/JZ Jump on equal/zero	0111 0100	disp

6.3.9 Loop Instructions

The LOOP instruction executes the part of the program from the level or address specified in the instruction up to the loop instruction, CX number of times. After each iteration, CX is decremented automatically. If

the content of CX is not zero, the LOOP instruction transfers control to starting address of the LOOP for execution. If CX is zero, the execution of LOOP instruction is completed and then the next instruction of the program will be executed. The LOOP instruction can be explained with an example as follows:

```

    LEA SI, 0100;    Load SI with source address of data
    LEA DI, 0200;    Load DI with destination address of data
    MOV CX, 0009;    Number of bytes 9 is loaded in CX register
START LODSB;        Data byte to AL and increment SI by 1
    STOSB;           The content of AL is stored in destination address represented by DI and and
                    increment DI by 1
    LOOP START;      repeat until CX = 0

```

The above example shows how a string of bytes can be shifted from one memory block specified by SI to other memory block specified by DI using LOOP instructions.

The LODSB instruction is equivalent to

```

MOV AL, [SI]
INC SI

```

The STOSB instruction is equivalent to

```

MOV [DI], AL
INC DI

```

And the LOOP instruction is equivalent to

```

DEC CX
JNZ START

```

In this case, LODSB and STOSB instructions are executed 9 times and a block of 9-byte data will be copied from source memory to destination memory sequentially.

LOOP Target(short) (Loop to short target) $CX \leftarrow (CX-1)$; Jump if $CX \neq 0$. The CX register is decremented by 1. If CX now is not equal to 0, the loop is back to target. Otherwise, continue.

The object code of LOOP is

1110 0010	DISP
-----------	------

LOOPE/Z Target(short) (Loop to short target if Z bit set) $CX \leftarrow (CX-1)$; jump if $CX \neq 0$ and $ZF = 1$. The CX register is decremented by 1. If CX is not equal to 0 or if the Z bit is set, the loop is back to short target.

The object codes of all conditional jump instructions are

1110 0001	DISP
-----------	------

LOOPNE/NZ (Loop to short target if Z bit is clear) $CX \leftarrow (CX-1)$; jump if $CX \neq 0$ and $ZF = 0$
The CX register is decremented by 1. If CX is not equal to 0 or if the Z bit is clear, the loop is back to short target.

The object codes of all conditional jump instructions are

1110 0000	DISP
-----------	------

6.3.10 CALL and RETURN Instructions

The CALL and RET (return) instructions are used to call a subroutine or a procedure that can be executed several times from a main program. The starting address of the subroutine or procedure can be specified

directly or indirectly depending upon the addressing mode. There are two types of procedures, namely, intra-segment and intersegment. The subroutine within a segment is known as intrasegment subroutine or NEAR CALL. The subroutine from one segment to another segment is known as intersegment subroutine or FAR CALL. These instructions are unconditional branch instructions. After execution of these instructions, the incremented IP and CS are stored onto the stack and loads the CS and IP registers with the segment and offset addresses of the procedure to be called. For NEAR CALL, only the IP register is stored on stack. But for FAR CALL, both IP and CS are stored onto the stack. Hence the NEAR and FAR CALLs can be discriminated using opcode. The operation of CALL and RET instructions are explained below:

CALL target (Call a procedure)

Near call: PUSH IP, JMP to target

$SP \leftarrow IP, SP \leftarrow SP-2, IP \leftarrow IP + DISP$

Far call: PUSH CS, PUSH IP, JMP to target :Flag affected: None

$SP \leftarrow CS, SP \leftarrow SP-2, SP \leftarrow IP, SP \leftarrow SP-2, IP \leftarrow 16 \text{ bit DATA}$

$CS \leftarrow 16\text{-bit DATA}$

The syntax for a near call (same segment) is CALL target.

The syntax for a far call (different segment) is CALL FAR target.

The object code of call instructions are given below:

Direct within segment	1110 1000	disp-low	disp-high
Direct Intersegment	1001 1010	Offset-low	Offset-high
		Seg-low	Seg-high
Indirect within segment	1111 1111	Mod 010 R/M	
Indirect Intersegment	1111 1111	Mod 011 R/M	

RET (Return from procedure)

RET n (return from procedure and add n to SP)

Near return: POP IP

$IP \leftarrow SP, SP \leftarrow SP + 2$

The syntax for a near return is RET.

Far return: POP IP, POP CS; Flag affected: None

$IP \leftarrow SP, SP \leftarrow SP + 2, CS \leftarrow SP, SP \leftarrow SP + DISP$

The syntax for a far return is RET FAR.

During execution of CALL instruction, initially the IP and CS of the next instruction is pushed onto stack, then the control is transferred to the procedure. At the end of execution of procedure, RET instruction must be executed. When RET instruction is executed, the previously stored content of IP and CS along with flags are retrieved into CS, IP and flag registers from the stack respectively. After that the execution of the main program again starts. Usually, the procedures are two types, namely, a near procedure or a far procedure. While in case of NEAR procedure, the current contents of SP points to IP but for a FAR procedure, the current contents of SP points to IP and CS at the time of return. Actually, the RET instructions are of four types such as

- ✦ Return within segment
- ✦ Return within segment adding 16-bit immediate displacement to the SP contents
- ✦ Return intersegment
- ✦ Return intersegment adding 16-bit immediate displacement to the SP contents

The object codes of RET instructions are as follows:

Within segment

1100 0011

Within Seg Adding Immed to SP

1100 0011	disp-low	disp-high
-----------	----------	-----------

1001 1010	Offset-low	Offset-high
-----------	------------	-------------

Seg-low	Seg-high
---------	----------

Indirect within segment

1111 1111	Mod 010 R/M
-----------	-------------

The RET n form adds n to the SP to compensate for stack growth when arguments are pushed onto the stack prior to a procedure call.

INT n (Interrupt (software))

PUSHF; IF \leftarrow 0; TF \leftarrow 0; PUSH CS; PUSH IP

IP \leftarrow 0000: [type * 4];

CS \leftarrow 0000: [(type * 4) + 2]; Flag affected: IT

INT n is a software interrupt to be serviced. The flags and current CS: IP are pushed onto the stack. The CS: IP stored in the vector indicated by the interrupt number are then loaded and the next instruction is fetched from that interrupt service routine address. There are 256 interrupts corresponding to the types from 00H to FFH in the interrupt structure of 8086. When an INT n instruction is executed, the TYPE byte n is multiplied by 4 and the contents of IP and CS of the interrupt service routine will be taken from the hexadecimal multiplication ($n \times 4$) as offset address and 0000 as segment address.

INTO (Interrupt on overflow) If OF = 1, then perform INT through vector 4; Flag affected: None

Interrupts the system if the overflow bit is set following a mathematical instruction. This indicates a carry from a signed value.

This instruction is executed, when the overflow flag OF is set. The new contents of IP and CS are taken from the address 0000: 0010 as explained in INT type instruction. This is equivalent to a type 4 interrupt instruction.

The object code of INTO instruction is

Interrupt on overflow

1100 1110

IRET (Return from interrupt service routine)

POP IP; POP CS; POPF AX (AL * src8); Flag affected: All

When an interrupt service routine is to be called, before transferring control to it, the IP, CS and flag register are stored on to stack to indicate the location from where the execution is to be continued after the ISR is executed. This instruction appears at the bottom of all Interrupt Service Routines (ISR).

When IRET is executed, the values of IP, CS and flags are retrieved from the stack to continue the execution of the main program.

The object code of IRET instruction is

1100 1111

6.3.11 String Instructions

Usually, a series of data bytes is known as a string of bytes and a series of data words is known as a string of words. For moving a string of bytes or words, 8086/8088 processors have five instructions such as STOS (store string byte or word), LODS (load string byte or word), MOVS (move string byte or word), SCAS (scan

string byte or word) and CMPS (compare string byte or word). For these instructions, a source of string byte or word is DS : SI and destination of string byte or word is ES : DI. After execution of these instructions, the offset memory pointer SI and DI are incremented or decremented by one or two depending upon direction flag. In this section STOS, LODS, MOVS, SCAS and CMPS are explained.

MOVSB (Move string byte) ES: [DI] ← DS: [SI]; DI = DI ± 1; SI = SI ± 1; Flag affected: None.

Moves a string a byte at a time from source memory DS : SI to destination memory ES: DI. SI and DI are incremented or decremented by 1, depending on Direction Flag (DF). The object code of MOVSB is 1010 010_w = A4 as $w = 0$.

MOVSW (Move string word) ES: [DI] ← DS: [SI]; DI = DI ± 2; SI = SI ± 2; Flag affected: None

Moves a string a word at a time from source memory DS: SI to destination memory ES: DI. SI and DI are incremented or decremented by 2, depending on Direction Flag (DF). The object code of MOVSW is 1010 010_w = A5 as $w = 1$.

STOSB (Store string byte) ES: [DI] ← AL; DI = DI ± 1; Flag affected: None

Moves a string one byte at a time from AL to destination memory address ES: DI. DI is then incremented or decremented by 1, depending on Direction Flag (DF). The object code of STOSB is 1010 101_w = AA as $w = 0$.

STOSW (Store string word) ES: [DI] ← AX; DI = DI ± 2; Flag affected: None

Moves a string one word at a time from AX to destination memory address ES: DI. DI is then incremented or decremented by 2, depending on Direction Flag (DF). The object code of STOSW is 1010 101_w = AB as $w = 1$.

LODSB (Load string byte) AL ← DS: [SI]; SI = SI ± 1; Flag affected: None

Moves a string one byte at a time from source memory address DS: SI to AL. Then SI is incremented or decremented by 1, depending on Direction Flag (DF). The object code of LODSB is 1010 110_w = AC as $w = 0$.

LODSW (Load string word) AX ← DS: [SI]; SI = SI ± 2; Flag affected: None

Moves a string one word at a time from source memory address DS: SI to AX. Then SI is incremented or decremented by 2, depending on Direction Flag (DF). The object code of LODSW is 1010 110_w = AD as $w = 1$.

CMPSB (Compare string byte) Flags ← (result of CMP DS: [SI], ES: [DI]); DI = DI ± 1; SI = SI ± 1; Flag affected: as CMP

The byte or 8-bit data at DS: SI is compared with the byte or 8-bit data at ES:DI and the flags are set accordingly. Both SI and DI are incremented or decremented by 1, depending on the Direction Flag (DF). This instruction is combined with a REP prefix, so that we can compare two strings and we can also find at what point two strings no longer are equal. The object code of CMPSB is 1010 011_w = A6 as $w = 0$.

CMPSW (Compare string word) Flags ← (result of CMP DS: [SI], ES: [DI]); DI = DI ± 2; SI = SI ± 2; Flag affected: as CMP

The word or 16-bit data at DS: SI is compared with the word or 16-bit data at ES: DI and the flags are set accordingly. Both SI and DI are incremented or decremented by 2, depending on the Direction Flag (DF).

This instruction is combined with REP prefix, so that we can compare two strings and we can also locate at what point two strings no longer are equal. The object code of CMPSW is 1010 011w = A7 as w = 1.

SCASB (Scan string byte) Flags ← (result of CMP ES: [DI], AL); DI = DI ± 1; Flag affected: same as CMP instruction

The byte or 8-bit data at ES: DI is compared to the contents of AL and correspondingly flags are set. DI is incremented or decremented by 1 depending upon the Direction Flag (DF). The SCASB can be combined with a REP prefix, so that we can be able to scan a string looking for the first occurrence of a particular byte. The object code of SCASB is 1010 111w = AE as w = 0.

SCASW (Scan string word) Flags ← (result of CMP ES: [DI], AX); DI = DI ± 2; Flag affected: same as CMP instruction

The word or 16-bit data at ES: DI is compared to the contents of AX and correspondingly flags are set and correspondingly flags are set. DI is incremented or decremented by 2 depending upon the Direction Flag (DF). The SCASW can be combined with a REP prefix, so that we can be able to scan a string looking for the first occurrence of a particular word. The object code of SCASW is 1010 111w = AF as w = 1.

REPEAT Instructions The string instructions are used to operate on large blocks of data. To refer a string, two parameters are required such as (i) starting/end address of the string, and (ii) length of the string. Usually starting/end address of the string is represented by DS: SI and the length of a string is stored as count in the CX register. After each iteration, the incrementing or decrementing of the pointer (SI or DI) depends upon the direction flag (DF) and the counter is decremented by one. To perform the string instructions repeatedly, REP (repeat) instructions are used. Hence the string instruction with the REP prefix is executed repeatedly until the CX register becomes zero. If CX becomes zero, the execution proceeds to the next instruction in sequence. The most commonly used REP instructions are REP (repeat), REPE (repeat while equal), REPZ (repeat while zero), REPNE (repeat while not equal), and REPNZ (repeat while not zero) which are explained below.

REP / REPE / REPZ (Repeat string instruction (prefix)) CX ← (CX-1); until CX = 0; Flag affected: Z

This is a prefix byte that forces a string operation to be repeated as long as CX is not equal to 0. CX is decremented once for each repetition. The object code of REPZ/REPE instruction is 1111 001Z = F3 as Z = 1.

REPNE / REPNZ (Repeat string instruction while not zero (prefix)) ZF ← 0; CX ← (CX-1); String Operation repeats while (CX! = 0 and ZF! = 0); Flag affected: Z. This is a prefix byte that keeps a string operation repeating while CX is not zero and Z! = 0. The object code of REPNZ/REPNE instruction is 1111 001Z = F2 as Z = 0.

Example 6.18

Write instruction to move a string of 9 bytes from source address DS:SI to destination address ES:DI. Assume DS = 4000H, ES = 6000H, SI = 0100H, DI = 0200H

Solution

MOV AX, 4000;	Load 4000H in the AX register
MOV DS, AX;	Load data segment address 4000H
MOV AX, 6000;	Load 6000H in the AX register
MOV ES, AX;	Load extra segment address 6000H

MOV CX, 0009;	Store number of data in the CX register
MOV SI, 0100;	SI register is loaded with 0100H
MOV DI, 0200;	DI register is loaded with 0200H
CLD;	Clear direction flag DF
REP MOVSB;	move a string of 9 bytes from source address DS:SI to destination address ES:DI.

6.3.12 Processor Control Instructions

These instructions control the operation of processor and set or clear the status indicators. These instructions are classified into two types such as flag-manipulation instructions and machine-control instructions. The flag-manipulation instructions directly change some flags of the 8086 processor but the machine-control instructions control the system bus functions.

The Carry (CF), Direction (DF) and Interrupt (IF) flags can be set or reset directly and the carry flag can be inverted by these instructions. The DF and IF are processor control bits. DF is used with the string instructions to change the content of pointer registers. When DF = 0, pointer register (DI) is incremented. When DF = 1, the pointer register (DI) is decremented. The STD (set direction flag) and CLD (clear direction flag) instructions are used to set or clear this flag. The STI (set interrupt flag) and CLI (clear interrupt flag) are used to enable or disable maskable interrupts on INTR line. When TF (trap flag) is set, a type 1 interrupt is generated after execution of each processor instructions. There are no specific instructions to set or reset the TF. POPF and SAHF instructions, which are termed as data-transfer instructions are used to modify flags. The machine-control instructions are HLT, WAIT, NOP ESC and LOCK. Some instructions are specially used for coprocessors. There are three coprocessor instructions as WAIT, LOCK and ESC. In this section all processor control instructions are explained.

CLC (Clear the carry flag) $CF \leftarrow 0$; Flag affected: C.

The CLC instruction is used in the carry flag low. The object code of CLC instruction is 1111 1000 = F8

CMC (Complement the carry flag) $CF \leftarrow \sim CF$; Flag affected: C

The CMC instruction is used to complement the carry flag. The object code of CMC instruction is 1111 0101 = F5

STC (Set the carry flag) $CF \leftarrow 1$; Flag affected: C

The CMC instruction is used to set the carry flag. The object code of STC instruction is 1111 1001 = F9

CLD (Clear direction flag) $DF \leftarrow 0$; Flag affected: D

Clear direction flag to 0. When DF = 0, pointer register (SI or DI) is automatically incremented by 1.

The object code of CLD instruction is 1111 1100 = FC

STD (Set direction flag) $DF \leftarrow 1$; Flag affected: D

Sets direction flag to 1. When DF = 1, pointer register (SI or DI) is automatically decremented by 1.

The object code of STD instruction is 1111 1101 = FD

CLI (Clear interrupt flag) $IF \leftarrow 0$; Flag affected: IF

Clears the interrupt enable flag which disables interrupts.

The object code of CLI instruction is 1111 1010 = FA

STI (Set interrupt flag) IF \leftarrow 1; Flag affected: IF

Sets the interrupt enable flag which enables interrupts. The object code of STI instruction is 1111 1011 = FB

HLT (Halt) Flag affected: None.

Halt instruction is used to ask the processor to stop execution. Actually, it hangs the processor in a series of self-inflicted NOP's until an interrupt occurs. The object code of HLT instruction is 1111 0100 = F4

WAIT (Wait) Flag affected: None.

It causes the processor to wait for a completion signal from the coprocessor. The object code of WAIT instruction is 1001 1011 = 9B

LOCK (Lock bus) Flag affected: None.

This instruction is used to avoid any other processors. Actually, it locks the bus attached to LOCK pin of device while a multicycle instruction completes. The object code of LOCK instruction is 1111 0000 = F0

NOP (No operation) When NOP instruction is executed, this instruction does not allow the processor to perform any operation except for incrementing the IP by one. The object code of NOP instruction is 1001 0000 = 90

TEST The TEST input is examined by a WAIT instruction. When the WAIT instruction is executed, it holds the operation of the processor with the current status till the logic level on the TEST pin is low. Therefore, the processor remains in idle state and the TEST pin goes low.

ESC (Escape) The ESC instruction is used as a prefix to the coprocessor instructions. The 8086 processor puts the source operand on the data bus but no operation further takes place. The coprocessor continuously examined the data bus content and it is activated by ESC instruction and it reads two operands and thereafter starts execution. The detailed operation is illustrated in coprocessor chapter.

SUMMARY

- In this chapter the different addressing modes of 8086/8088 microprocessors are explained with examples. The 8086 microprocessor instruction format is discussed in detail.
- The instruction set can be classified into different categories depending upon its functions such as data-transfer instruction, arithmetic instruction, logical instruction, string instruction, bit-manipulation instruction, control transfer instruction and machine/processor control instruction.
- In this chapter, all data-transfer instruction, arithmetic instruction, logical instruction, string instruction, bit-manipulation instruction, control transfer instruction and machine/processor control instructions are discussed with examples.

MULTIPLE-CHOICE QUESTIONS

- 6.1 What is the addressing mode of the instruction `MOV AX, [BX]`?
- Register direct
 - Register indirect
 - Immediate addressing
 - Indirect addressing
- 6.2 What is the addressing mode of the instruction `MOV AX, [BX+SI+06]` ?
- Index addressing
 - Base addressing
 - Base index addressing
 - Base index displacement addressing
- 6.3 Which of the following instructions is immediate addressing?
- `MOV AX, [2000]`
 - `MOV BX, 2000`
 - `MOV AX, [SI]`
 - `MOV AX, BX`
- 6.4 Which of the following instruction is base with 16-bit displacement addressing?
- `MOV AX, [BX + 06]`
 - `MOV AX, [BP + 2000]`
 - `MOV AX, [BP + 06]`
 - `MOV AX, [BP]`
- 6.5 Which of the following instruction is a four-byte instruction?
- `MOV AX, 2345`
 - `MUL BX`
 - `DIV CL`
 - `ADD AX, [BP + 0200]`
- 6.6 Which of the following instruction is a six-byte instruction?
- `MOV [BX + DI + 0200], 2345`
 - `MOV [SI], 5665`
 - `DIV CL`
 - `ADD BX, [BP + 0200]`
- 6.7 Which of the following instruction is a logical instruction?
- `DIV AB`
 - `TEST`
 - `CALL`
 - `AAM`
- 6.8 Which of the following instruction affects carry flag?
- `RCR`
 - `MUL AB`
 - `JZ`
 - `INC AX`
- 6.9 Which of the following instruction is an arithmetic instruction?
- `DIV AB`
 - `ROR`
 - `STI`
 - `WAIT`
- 6.10 The example of the string instruction is
- `MOV DX, [SI]`
 - `XLAT`
 - `MOVSB`
 - `AAD`
- 6.11 Which of the following instructions is not true?
- `MOV [2000], [4000]`
 - `MOV AX, [2000]`
 - `MOV [2000], AX`
 - `MOV AX, BX`
- 6.12 2's complement instruction is
- `NEG`
 - `NOT`
 - `CMP`
 - `CMC`
- 6.13 Direction flag is used with which of the following instructions?
- Data transfer
 - Branch control instructions
 - String instructions
 - Logical instructions
- 6.14 Which of the following instruction does not allow the interrupt request signal to interrupt the instruction which follows `NOP` ?
- `ESC`
 - `HALT`
 - `WAIT`
 - `LOCK`
- 6.15 Which of the following instruction is used to read a string of bytes and send it to another memory location?
- `SCASB`
 - `MOVSB`
 - `LODSB`
 - `LODSB`
- 6.16 A procedure can be called using the instruction
- `JMP`
 - `CALL`
 - `RET`
 - `INT n`

- 6.17 To return a procedure, we use the instruction
 (a) JMP (b) CALL
 (c) RET (d) INT n
- 6.18 The LODSB instruction is used which of the following register combinations?
 (a) ES:SI (b) ES:DI
 (c) DS:SI (d) DS:DI
- 6.19 When PUSH instruction is executed, initially
 (a) upper byte of data is stored on stack and $SP = SP - 1$
 (b) upper byte of data is stored on stack and $SP = SP + 1$
 (c) lower byte of data is stored on stack and $SP = SP - 1$
 (d) lower byte of data is stored on stack and $SP = SP + 1$
- 6.20 Coprocessor control instructions are
 (a) WAIT, LOCK, ESC
 (b) HALT, STC, CLC
 (c) ROR, RCR, ROL
 (d) DAA

SHORT-ANSWER-TYPE QUESTIONS

- 6.1 What is an instruction format? What are the types of instructions of 8086 microprocessors based on format?
- 6.2 Write the classification of 8086 instructions based on functions. Give a list of examples of different instructions.
- 6.3 Explain the difference between FAR CALL and NEAR CALL instructions.
- 6.4 Which registers are affected by the MUL, IMUL, DIV, and IDIV instructions?
- 6.5 Which of the shift, rotate, and logical instructions do not affect the zero flag?
- 6.6 Why does the SAR instruction always clear the overflow flag?
- 6.7 What does the NEG instruction do? What instruction is most similar to CMP? What instruction is most similar to TEST?

REVIEW QUESTIONS

- 6.1 Define addressing modes of 8086 processors. What are the different addressing modes of 8086 microprocessors? Explain each addressing mode with examples.
- 6.2 Write the procedure to determine physical address for the following instructions as given below:
 (i) MOV AX, [SI + 03] (ii) MOV AL, CS:[BX + 0400]
 (iii) MOV AX, [3000] (iv) MOV AL, [BX+SI + 22]
 Assume CS = 4000H, IP = 2300, SI = 02300 and DS = 5000
- 6.3 Write the difference between the following instructions:
 (i) MUL and IMUL (ii) DIV and IDIV
 (iii) JUMP and LOOP (iv) Shift and Rotate
- 6.4 Give a list of processor control instructions and explain briefly.
- 6.5 Explain the execution of data-transfer instructions with suitable examples.
- 6.6 What is a procedure? What are the different types of procedure in 8086? Discuss each type procedure with examples.

- 6.7 Discuss string instructions with suitable examples. Explain why REP prefix is added with string instructions. Which string instruction should be used to ensure that two strings in the memory are equal?
- 6.8 Explain operation of the following instructions:
- | | | | |
|------------------|-------------|--------------|---------------------|
| (i) ADD AX, [BX] | (ii) INC SI | (iii) MUL BX | (iv) IMUL DX |
| (v) NEG AL | (vi) DEC DI | (vii) XLAT | (viii) PUSH and POP |
- 6.9 Write the difference between following instructions
- | | | |
|----------------------------|--|------------------------|
| (i) CBW and CWD | (ii) MOV reg, immediate and LEA reg, address | |
| (iii) DEC AX and SUB AX, 1 | (iv) RCL and ROL | (v) IRET and RET (far) |
- 6.10 Explain the operation of the LOOP, LOOPE/LOOPZ, and LOOPNE/LOOPNZ instructions. What does the INT n instruction push onto the stack that the CALL FAR instruction does not? What is the JCXZ instruction typically used for?
- 6.11 Explain the operation of the 80x86 CALL and RET instructions. Describe step by step with suitable examples.
- 6.12 Explain how the XLAT instruction is used to convert an alphabetic character in the AL register from lower case to upper case and leave all other values in AL unchanged.
- 6.13 Find the object codes for the following instructions
- | | | | |
|------------------|-------------------------|------------------|----------------------|
| (i) MOV AX, 2345 | (ii) MOV [BX +SI], 4444 | (iii) ADD AL, FF | (iv) ADD [BX], 45 67 |
| (v) MUL CL | (vi) IMUL CX | (vii) DIV BX | (viii) IDIV CL |
- 6.14 Write instructions to perform the following operations:
- Copy content of BX to a memory location in the data segment with offset 0234H
 - Increment content of CX by 1
 - Multiply AX with 16 bit data 2467H
 - Rotate left the content of AL by two bits
- 6.15 Write results after execution of following instructions:
- MOV AL, 22; MOV BL, 44; ADD AL, BL;
 - MOV AX, 1002; MOV BX, 44; MUL AX, BL;
 - MOV CL, 34; MOV AL, FF; SUB AL, CL;
 - MOV AX, 8796; MOV CL, 2; ROR AX, CL;

Answers to Multiple-Choice Questions

-
- | | | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| 6.1 (b) | 6.2 (d) | 6.3 (b) | 6.4 (b) | 6.5 (d) | 6.6 (a) | 6.7 (b) | 6.8 (a) | 6.9 (a) |
| 6.10 (c) | 6.11 (a) | 6.12 (a) | 6.13 (c) | 6.14 (b) | 6.15 (b) | 6.16 (b) | 6.17 (c) | 6.18 (c) |
| 6.19 (a) | 6.20 (a) | | | | | | | |

Chapter 7

Assembly-Language Programs of the 8086 Microprocessor and 8087, 80287 and 80387 Numeric Data Processors

7.1 INTRODUCTION

Machine-language programming is coding of a program in terms of 0 and 1. During this programming, the memory control is directly in the hands of the programmer and the programmer is able to manage the memory of the system more efficiently. But the programming, coding and memory-management techniques in machine-language programming are very tedious. As the programmer writes all functions in terms of 0 and 1, the possibility of human errors is more. To write and understand the programs, the programmer should have thorough knowledge about the architecture and instruction set of the processor. The disadvantages of machine-language programming are given below:

- ✦ Writing a program is very complicated and time consuming.
- ✦ Possibility of errors in programming are large and a human can only feed the program byte by byte into the system.
- ✦ Debugging the program is very difficult.
- ✦ Only a program designer is able to understand the program. Therefore, such programs are not user friendly.

Assembly-language programming is comparatively simpler than machine-language programming. In the assembly-language programs, instruction mnemonics are used to write programs directly. These programs are more readable and understandable than machine-language programs. In assembly language, the address values and the constants can be identified by labels. As the labels are clear, the program becomes more understandable. The tedious byte handling and manipulations are reduced as address and constants are available inside the program, and it is not required to remember them. However, different logical segments and routines may be assigned with the labels rather than the different addresses. The memory-control feature of machine-language programming is left unchanged by providing storage defined facilities in assembly-language programming. The documentation facility is now available in assembly language.

An *assembler* is a program which converts any assembly-language program into the equivalent machine codes. During conversion, firstly the address of each label is initialised and the values for each of the constants and variables are substituted. Thereafter, the assembler generates the equivalent machine code for all mnemonics and data. The assembler also generates information about syntax errors in the program but the assembler cannot find out any logical errors in the program. The advantages of an assembler are as follows:

- ✦ Writing a program in assembly language is comparatively easier than machine level language programming.
- ✦ The chances of errors during editing a program are less as mnemonics are used to write program.
- ✦ It is very easy to enter the program in assembly language.
- ✦ Debugging is also easier than machine code programming as mnemonics are purpose suggestive.
- ✦ Such programs are more user friendly as the constants and memory address locations can be labelled. Macros make the task of programming easier.
- ✦ After execution, the results of programs are stored in a more user-friendly form.
- ✦ Flexibility of programming in assembly language is more than in machine language.

In any assembly-language program, the programmer should mention constants, variables, logical names of the segments, types of the different routines and modules, and end of file. Such help is given to the assembler using predefined alphabetical strings called *assembler directives*. Actually, assembler directives help the assembler understand the assembly-language programs properly and generate the machine codes. Usually, the following directives are commonly used in assembly-language programming:

DB: Define Byte
 DW: Define Word
 DQ: Define Quadword
 DT: Define Ten Bytes
 ASSUME: Assume Logical Segment Name
 END: End of Program
 ENDP: End of Procedure
 ENDS: End of Segment
 EQU: Equate
 LABEL: Label
 LENGTH: Byte Length of a Label
 NAME: Logical Name of a Module
 OFFSET: Offset of a Label
 ORG: Origin
 PROC: Procedure
 SEG: Segment of a label

To edit an assembly-language program on an IBM PC in the DOS operating system, different text editors such as Norton's Editor (NE.Com), Microsoft Assembler (MASM.EXE), Linker (LIINK.EXE) and Debugger (DEBUG.EXE) are commonly used. In this section, the basic operations of these editors are explained briefly.

7.1.1 Norton's Editor

To start Norton's Editor, type NE after C and enter the directory.

C > NE ↵

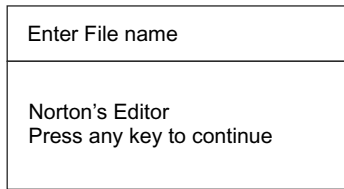


Fig 7.1 Norton's Editor's opening page

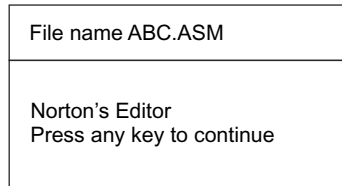


Fig. 7.2 Norton's Editors with file name ABC.ASM

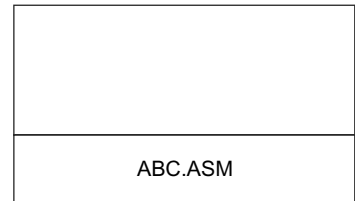


Fig. 7.3 Norton's Editor: Open a file ABC.ASM

After pressing the enter key, Norton's Editor's opening page will be displayed as shown in Fig. 7.1. Then type the file name. For example, assume the file name is ABC and the screen display is shown in Fig. 7.2. When any key is pressed, the ABC.ASM file will be opened as depicted in Fig. 7.3. After that, enter text to edit the assembly-language program. A sample program ABC.ASM is edited to subtract two numbers as shown in Fig. 7.4. If we want to open a file directly, the command is C>NE ABC.ASM. Then ABC.ASM file will be opened and displayed on the CRT screen. After editing the program or modifying the existing program, the F3-E command is used to save the program and exit from Norton's Editor by using the F3-Q command.

7.1.2 MASM Editor

The Microsoft Assembler MASM is a most popular assembler and it is very easy to use. To enter an assembly-language program, the command is

C > MASM ABC ↵

or

C > MASM ABC.ASM ↵

When the above commands are executed, Fig. 7.5 will be displayed on screen. If we enter the Command C > MASM ↵, Fig. 7.6 will be displayed as the opening page of MASM.

In Fig. 7.6, the source filename should be typed in the source filename with or without extension the .ASM. Then valid filename will be accepted if the enter key is pressed. Thereafter enter the .OBJ file name which creates the object file of the assembly-language program. The listing file is identified by the source filename and an extension .LST. This file consists of Levels, Offset Address, Mnemonics, Directives and other necessary assembly-language related information. The cross-reference filename is also entered in the same way as the listing file. This file is used for debugging the source program. The .CRF file contains information such as size of the file in bytes, number of labels, list of labels, and routines of the source program. After entering the cross-reference file name, the assembly-language process starts. Then syntax errors of the program are displayed using error code

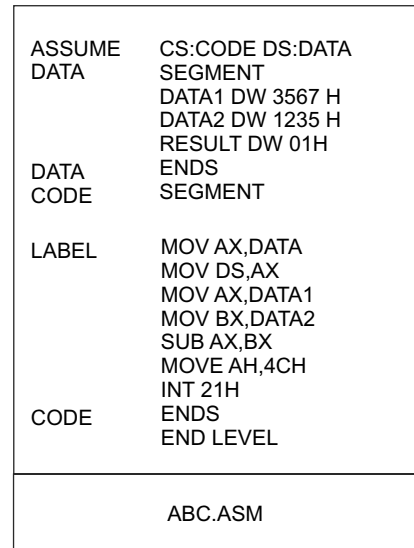


Fig. 7.4 Program for subtraction of two numbers in Norton's Editor

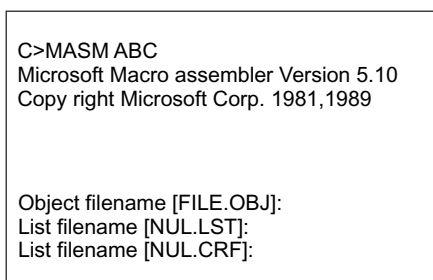


Fig. 7.5 MASM opening page

number and the corresponding line number, if the program has any syntax errors. When the programmer removes all syntax errors, the assembly process will be completed successfully. After successful assembly process, the .OBJ, .LST and .CRF files, are generated and these files can be linked by the linker programmer to link the object modules and generate an executable (.EXE) file.

7.1.3 LINK

The LINK. EXE file links the different object modules of the source program and function library routines to generate executable code of the source program. The input to the linker is the .OBJ file. The linker program is executed by the command `C > LINK ↵` or `C>LINK ABC.OBJ ↵` and the display on the screen is shown in Fig. 7.7.

7.1.4 DEBUG

DEBUG.COM is a program which allows the programmer to write, execute, debug and trouble-shoot assembly-language programs. This command is used to examine the contents of registers and memory. It can also be used to execute a program, but one instruction at a time. To start DEBUG, it is required to type DEBUG after the DOS prompt as given below:

```
C: \> DEBUG
```

After typing the above command, press the 'enter key. Then the DEBUG command is executed and a hyphen which is known as DEBUG prompt – (dash) will be displayed as given below:

```
C: \>DEBUG ↵
```

```
-
```

Any valid command such as `-R`, `-M`, `-A` and `-U`, etc., is accepted using the enter key. All the valid commands of DEBUG are discussed in the next section.

7.2 ASSEMBLY-LANGUAGE COMMANDS

DEBUG has a number of valid commands which are used to write, execute and debug programs. Each command is designed to perform an important task. There are commands to assemble and execute programs, display the contents of registers and memories, to run a program in single-step mode, etc. Generally, the DEBUG command is represented by a one-letter symbol such as A for assemble, E for entry, D for display, etc.

7.2.1 A (Assemble Command)

The A command is used to enter mnemonics of an assembly-language program and convert it into machine codes. When the A command is executed, the machine codes are generated and directly stored in the memory.

```
C>MASM
Microsoft Macro assembler Version 5.10
Copy right Microsoft Corp. 1981,1989
```

```
Object filename [.ASM]:
List filename [FILE.OBJ]:
List filename [NUL.LST]:
List filename [NUL.CRF]:
```

Fig. 7.6 MASM opening page

```
C>LINK
Microsoft Linker Version 3.64
Copy right Microsoft Corp. 1981,1989
```

```
Object Module [.OBJ]:
List File [.EXE]:
List filename [NUL.LST]:
Libraries [LIB]:
```

Fig. 7.7 Link command screen display

After getting DEBUG prompt, the starting address is initialized as-A address. Assume the starting offset address as 1000H, then the assemble can be written as

- A 1000

When the above instruction is executed, the display on the screen will be

17DA : 1000

Here, 17DA is the content of the Code Segment (CS) register and the offset address will be 1000H.

When offset is not written after the A command, the DEBUG command will be executed assuming the offset address 1000H.

To edit some instructions, write instruction after address as given below:

Example 7.1

Load 16-bit data 2045H in the AX register and another 16-bit data 6545 in the BX register.

Solution

C: \>DEBUG

- A 1000

17DA : 1000 MOV AX,2045; Load 16-bit data 2045H in the AX register

17DA : 1003 MOV BX,6545; Load 16-bit data 6545H in BX register

17DA : 1006 HLT

17DA : 1007

7.2.2 U (Un-assemble Command)

The U command is used to disassemble machine codes of specified memory locations and generates corresponding mnemonics and machine codes. This command displays machine codes and mnemonics on the screen. The default register is CS : IP and the general format is

-U address range

In Example 7.1, the program is written from 17DA: 1000 to 17DA: 1006. To display machine codes on the screen the U command can be written as follows:

-U 1000 1006

After execution of the above command, the display on the screen will be as given below:

17DA : 1000 B8 45 20 MOV AX, 2045

17DA : 1003 BB 45 65 MOV BX, 6545

17DA : 1006 F4 HLT

-

When the U command is applied without specifying any address range, DEBUG un-assembles the first 32 bytes starting from the address which is located by the content of IP register or it un-assembles 32-bytes since the last U. The default register for U command is the CS: IP. For example, the display is shown on the screen after execution of U command without mentioning the address.

Assume the following instructions are stored in the memory locations from 17DA: 1000 to 175A:1020

C: \>DEBUG

- A 1000

```

17DA : 1000 MOV AX,2045;   Load 2045H in the AX register
17DA : 1003 MOV BX,6545;   Load 6545H in the BX register
17DA : 1006 MOV CX,1234;   Load 1234H in the CX register
17DA : 1009 MOV DX,1234;   Load 1234H in the DX register
17DA : 100C ADD AX, BX;    Add content of BX with AX
17DA : 100E ADD AX, CX;    Add content of CX with AX
17DA : 1010 ADD AX, DX;    Add content of DX with AX
17DA : 1012 MOV SI, 0100;  Store 0100 in SI register
17DA : 1015 MOV DI, 0100;  Store 0100 in DI register
17DA : 1018 NEG AX ;      2's complement of AX
17DA : 101A NEG BX;      2's complement of BX
17DA : 101C NEG CX ;      2's complement of CX
17DA : 101E NEG DX;      2's complement of DX
17DA : 1020 HLT
17DA : 1021

```

If we execute the -U 1000 command, the display on screen will be

-U 1000

```

17DA : 1000 B8 45 20 MOV AX,2045
17DA : 1003 BB 45 65 MOV BX,6545
17DA : 1006 B9 34 12 MOV CX,1234
17DA : 1009 BA 34 12 MOV DX,1234
17DA : 100C 01 D8 ADD AX, BX
17DA : 100E 01 C8 ADD AX, CX
17DA : 1010 01 D0 ADD AX, DX
17DA : 1012 BE 00 01 MOV SI, 0100
17DA : 1015 BF 00 02 MOV DI, 0100
17DA : 1018 F7 D8 NEG AX
17DA : 101A F7 DB NEG BX
17DA : 101C F7 D9 NEG CX
17DA : 101E F7 DA NEG DX

```

7.2.3 R (Register Command)

The register command R can be used to display the contents of one or more registers. This instruction also displays the status of the flags. The general format of the R command is

-R register name

For example, the execution of -R AX command, the DEBUG displays the content of the AX register as follows:

C: \>DEBUG

- A 1000

17DA : 1000 MOV AX, 2045; Load 2045H in AX register

17DA : 1003 MOV BX, 6545; Load 6545H in BX register

17DA : 1006 HLT

17DA : 1007

-GCS: 1006

AX = 20445 BX = 6545 CX = 0000 DX = 0000 SP = FFEE BP = 0000 SI = 0000 DI = 0000

DS = 17DA ES = 17DA SS = 17DA CS = 17DA IP = 0100 NV UP EI PL NZ NA PO NC

17DA : 1006 F4 HLT

-R AX

AX 2045

:

When the name of a register is given after the R command, the name and the content of that register is displayed in the next line. A colon is then displayed as a prompt in the subsequent line. If a new value after the colon is typed and enter key pressed for execution, the content of the register will be changed. After that the debug prompt is displayed. Again to see the content of the register, write the R command with register name as given below.

-R AX

AX 2045

:5000

AX 5000

:

Usually, the R command is given without the name of a register; the DEBUG displays the contents of all registers including status flags.

-R

AX = 5000 BX = 6545 CX = 0000 DX = 0000 SP = FFEE BP = 0000 SI = 0000 DI = 0000

DS = 17DA ES = 17DA SS = 17DA CS = 17DA IP = 0100 NV UP EI PL NZ NA PO NC

17DA : 1006 F4 HLT

Status Flags

The status flags with their codes for RESET and SET are illustrated in Table 7.1. F is the flag register-name to visualise the status of flags. When -R F command is executed, the status of the flags are displayed in the beginning of the next line. At the end of the list of status of the flags, a hyphen (-) is displayed as given below. If we want to change status flags, after the hyphen type the desired flags in any order as per requirement, and then press the enter key.

-R F

After execution, the flag register will display the format as given below:

NV UP EI PL NZ NA PO NC-

If we want to change NC to CY, enter desired status of the desired flags after the hyphen as given below:

NV UP EI PL NZ NA PO NC-CY

To check the changed condition after giving R F command, we execute -R F command and the display on screen will be

- R F

OV UP EI NG ZR AC PO CY-

Table 7.1 Status flags for RESET and SET

<i>Name of the flag</i>	<i>Reset</i>	<i>Set</i>
Overflow	NV	OV
Direction	UP	DN
Interrupt	DI	EI
Sign	PL	NG
Zero	NZ	ZR
Auxiliary Carry	NA	AC
Parity	PO	PE
Carry	NC	CY

7.2.4 G (Go Command)

The Go command is used to execute any program. The general format of the G command is G = address. The equal sign (=) is put before the specified address which indicates the starting address of the program. The default register for G command is CS. A program for addition of two 16-bit numbers which are loaded in AX and BX registers is written from

```
C:\>DEBUG
-A 1000
17DA : 1000 MOV AX, 2000; Load 2000H in AX register
17DA : 1003 MOV BX, 3000; Load 3000H in BX register
17DA : 1006 ADD AX, BX; Add the content of BX with AX
17DA : 1008 HLT
17DA : 1009
-G=1000
Program terminated normally
```

To execute the above program and to visualize the results, we should write the command as follows:

```
-G 1008
```

In the above command, 1008 is the end address of the program.

Then the program will be starting from the address CS: 1000 and the contents of all registers and flags will be displayed as given below:

```
AX = 5000 BX = 3000 CX = 0000 DX = 0000 SP = 0004 BP = 20CD SI = 0000 DI = 0000
DS = 17DA ES = 17DA SS = 9FFF CS = 17DA IP = 1008 NV UP EI PL NZ NA PE NC
17DA:1008 F4 HLT
```

Sometimes, GCS command is used to execute the program. The general format of GCS command is

-GCS: end address of program. The example of GCS command for execution of above program is given below:

```
C:\>DEBUG
-A1000
17DA: 1000 MOV AX, 2000; Load 2000H in AX register
17DA:1003 MOV BX, 3000; Load 3000H in BX register
17DA:1006 ADD AX, BX; Add the content of BX with AX
17DA:1008 HLT
```

17DA:1009

-GCS: 1008

After execution, the result will be displayed as given below:

AX = 5000 BX = 3000 CX = 0000 DX = 0000 SP = 0004 BP = 20CD SI = 0000 DI = 0000

DS = 17DA ES = 17DA SS = 9FFF CS = 17DA IP = 1008 NV UP EI PL NZ NA PE NC

17DA:1008 F4 HLT

7.2.5 T (Trace Command)

The trace command T is used to run a program in single-step mode. The default register is the CS: IP and the general format of trace command is

-T = address

To testing the trace command, we should write a simple program as given below:

C:\>DEBUG

-A1000

17DA:1000 MOV AX, 2456; Load 2456H in AX register

17DA:1003 MOV BX, 6000; Load 6000H in BX register

17DA:1006 ADD AX, BX; Add the content of BX with AX

17DA:1008 HLT

17DA:1009

To execute the above program in single-step mode, we should enter the command

-T = 1000

When the enter key is pressed after T=1000 command, the first instruction of the program will be executed and the result will be displayed on the screen as given below:

AX = 2456 BX = 0000 CX = 0000 DX = 0000 SP = FFEE BP = 0000 SI = 0000 DI = 0000

DS = 17DA ES = 17DA SS = 17DA CS = 17DA IP = 1003 NV UP EI PL NZ NA PO NC

17DA: 1003 BB0060 MOV BX, 6000

The last line of the display is the next instruction which will be executed. To execute the next instruction, the T command is used in the following format as given below:

-T

AX = 2456 BX = 6000 CX = 0000 DX = 0000 SP = FFEE BP = 0000 SI = 0000 DI = 0000

DS = 17DA ES = 17DA SS = 17DA CS = 17DA IP = 1006 NV UP EI PL NZ NA PO NC

17DA: 1006 01D8 ADD AX, BX

-

When the T command is entered and the enter key pressed, 17DA: 1003 address instruction MOV BX, 6000 will be executed by default as the content of IP is 0103 and the results are displayed as given above.

To execute any instruction of the program use T = address command. For example, if we want to execute the third instruction at the memory location 17DA: 1006 ADD AX, BX, we should write the following command:

-T = 1006

AX = 8456 BX = 6000 CX = 0000 DX = 0000 SP=FFEE BP = 0000 SI = 0000 DI = 0000

DS = 17DA ES = 17DA SS = 17DA CS = 17DA IP = 1008 OV UP EI NG NZ NA PE NC

17DA:1008 F4 HLT

-

Before typing the T command, the contents of AX and BX registers are 2456 and 6000 respectively. After execution of the T command, the third instruction will be executed and result will be stored in the AX register as shown above.

If want to execute more than one instruction by a single command in single-step mode, the common format of the T command is

-T = address location value

Here address location is the starting address of first instruction from which execution will be started and value = n , the number of instructions to be executed by single command in single-step mode. Suppose we want to execute the first and second instructions of a program as illustrated by single command in single-step mode, we write the command as given below:

C:\>DEBUG

-A 1000

17DA:1000 MOV AX, 1234; Load 1234H in AX register

17DA:1003 MOV BX, 3456; Load 3456H in BX register

17DA:1006 ADD AX, BX; Add the content of BX with AX

17DA:1008 HLT

17DA:1009

-T=1000 02

AX = 1234 BX = 0000 CX = 0000 DX = 0000 SP = FFEE BP = 0000 SI = 0000 DI = 0000

DS = 17DA ES = 17DA SS = 17DA CS = 17DA IP = 1003 NV UP EI PL NZ NA PO NC

17DA:1003 BB0060 MOV BX,6000

AX = 1234 BX = 3456 CX = 0000 DX = 0000 SP = FFEE BP = 0000 SI = 0000 DI = 0000

DS = 17DA ES = 17DA SS = 17DA CS = 17DA IP = 1006 NV UP EI PL NZ NA PO NC

17DA: 1006 01D8 ADD AX, BX

After applying the T=1000 02 command, the first instruction 17DA: 1000 MOV AX, 1234 will be executed and the result will be displayed on screen. After that the second instruction 17DA:1003 MOV BX, 3456 will be executed and again the result will be displayed on screen as shown above.

7.2.6 D (Display Command)

The D command is used to display the contents of specified memory locations. The default register is DS. The general format of the D command is

-D or -D address

C:\>DEBUG

-A1000

17DA:1000 MOV AX, 2000; Load 2000H in AX register

17DA:1003 MOV BX, 3000; Load 3000H in BX register

17DA:1006 HLT

17DA:1007

-D

The display on screen will be from starting address 17DA: 0100 by default as given below:

```
17DA:0100  00 00 00 00 00 00  00 00-00 00 00 00 00 00 00  .....
17DA:0110  00 00 00 00 00 00  00 00-00 00 00 00 00 00 00  .....
17DA:0120  00 00 00 00 00 00  00 00-00 00 00 00 00 00 00  .....
17DA:0130  00 00 00 00 00 00  00 00-00 00 00 00 00 00 00  .....
```

```
17DA:0140  00 00 00 00 00 00  00 00-00 00 00 00 00 00 00  .....
17DA:0150  00 00 00 00 00 00  00 00-00 00 00 00 00 00 00  .....
17DA:0160  00 00 00 00 00 00  00 00-00 00 00 00 00 00 00  .....
17DA:0170  00 00 00 00 00 00  00 00-00 00 00 00 00 00 00  .....
```

Each line can display about 16 bytes. There is a hyphen between the eighth and ninth bytes. When the `-D 1000` is executed, the following data will be displayed on the screen.

```
17DA:1100  B8 00 20 BB 00 30  F4 00-00 00 00 00 00 00 00  .....
17DA:1110  00 00 00 00 00 00  00 00-00 00 00 00 00 00 00  .....
17DA:1120  00 00 00 00 00 00  00 00-00 00 00 00 00 00 00  .....
17DA:1130  00 00 00 00 00 00  00 00-00 00 00 00 00 00 00  .....
17DA:1140  00 00 00 00 00 00  00 00-00 00 00 00 00 00 00  .....
17DA:1150  00 00 00 00 00 00  00 00-00 00 00 00 00 00 00  .....
17DA:1160  00 00 00 00 00 00  00 00-00 00 00 00 00 00 00  .....
17DA:1170  00 00 00 00 00 00  00 00-00 00 00 00 00 00 00  .....
```

The other format of the D command is

`-D address range`

The D command incorporating address range is written as `D 1000 1005`, the display on screen will be

```
17DA: 1100 B8 00 20 BB 00 30 F4
```

When we write the D command with starting address, the DEBUG is executed and by default 80 (hex) bytes or 128 bytes (80 hex) starting from the given address will be displayed. The command `D 0100` will display 128 bytes starting from the memory location DS: 0100.

7.2.7 E (Enter Command)

The E command is used to enter machine codes or data. This instruction operates with the register DS by default. This command can be used in the following ways such as:

- ✦ Sequentially enter data or machine codes
- ✦ Replace data or machine codes of certain memory locations

The common format to enter data or machine codes sequentially is

`-E address list`

When the data are to be entered in DS segment starting from the offset address 0500, the command will be written as

```
-E 0500
```

After writing the E command, data should be entered in the same line with one space between two adjacent data as given below:

```
-E 0500 01 02 03 04 05 06 07 08
```

Here, data are 01 02 03 04 05 06 07 08 and the starting address is DS: 0500. After execution the above command, data will be entered into memory location DS: 0500 to DS: 0507. The content of DS: 0500 is 01 and the content of DS: 0507 is 08.

When we want to replace data in DS: 0400, we should write the command as E address. The example of replacing command is given below:

```
-E 0400
```

```
17DA:0400  00.12      00.34      00.56      00.78      00.99      00.AA      00.BB      00.CC
```

-E 0400

17DA:0400 12.01 34.02 56.03 78.04 99.05 AA.06 BB.07 CC.08

-E 0400

17DA: 0400 01. 02. 03. 04. 05. 06. 07. 08.

When the E 0400 command is executed, this will show the contents of the memory location DS: 0400 in the next line as

17DA: 0400 00

where, 17DA is the initial setting for DS by default. 00 is the content of the memory location 17DA:0400. If we want to change the existing data, enter the new data 12 as shown below:

17DA:0400 00.12

If we want to change data of the next memory location, the space bar is to be pressed. It will show the content of the next memory location. To change the existing data 00, enter the new data 34. In this way, data replacement can be preceded. The maximum number of bytes that can be entered in a line is 8. After replacing the desired number of data, the enter key is pressed. While data replacement is continued up to the end of the line, the next line with current memory address comes automatically.

7.2.8 F (Fill Command)

The Fill command is used to fill the specified range of memory locations with the values which are entered in a list. The default segment register is the DS. The general format is

-F address range list of data

The example of Fill command is

-F 0300 0304 12 34 56 78 90

Then a list of values from 12 to 90 will be filled in the memory locations DS: 0300 0304. This can be verified using D command as given below:

C:\>DEBUG

-F 0300 0304 12 34 56 78 90

-D 0300 0304

17DA:0300 12 34 56 78 90

-

7.2.9 M (Move Command)

Generally, data movement operation is performed with the M command. The M command is usually used to copy/move the block of data from one memory block into another memory block. By default, the DS register is used to locate data. The general format of the M command is

-M range address

The example of move command is -M 0100-0105 0200

or -M 0100 L 06 0200

After execution of this command, the data starting from DS: 0100 to DS: 0105 are copied into the memory location address beginning from DS: 0200 to DS: 0205.

7.2.10 S (Search Command)

The search command S is used to search the specified memory locations for the specified list of bytes. By default, the data segment register DS is used to locate data. The general format is

-S address range list

The list may contain one byte or more than one byte of data. If the list contains only one byte, all addresses of the byte in the specified range will be displayed. If the list contains more than one byte, then only the first addresses of the byte string are returned.

To search a byte (44H) in a specified memory range from DS: 0100 to DS: 0200, the command may be written as follows:

```
S 0100 0200 44.
```

7.3 ASSEMBLY-LANGUAGE PROGRAMS

7.3.1 Program for Addition of Two 8-Bit Numbers with a 16-Bit Sum

The first number FFH is stored in the AL register and the second number 22H is stored in the BL register. The result after addition will be stored in AX. The program flow chart for addition of two 8-bit numbers with a 16-bit sum is shown in Fig. 7.8.

Algorithm

1. Store first data in Register AL.
2. Store second data in Register BL.
3. Add the contents of AL and BL.
4. CY flag will be set, if result is more than 8 bits

```
C:\>DEBUG
```

```
-A 1000
```

```
17DA: 1000 MOV AL, FF; Load FFH in AL register
```

```
17DA: 1002 MOV BL, 22; Load 22H in AL register
```

```
17DA: 1004 ADD AL, BL; Add content of BL to AL
```

```
17DA: 1006 HLT
```

```
17DA: 1007
```

```
-U 1000 1006
```

```
17DA: 1000 B0 FF MOV AL,FF
```

```
17DA: 1002 B3 22 MOV BL,22
```

```
17DA: 1004 00 D8 ADD AL,BL
```

```
17DA: 1006 F4 HLT
```

```
-G 1006
```

```
AX = 0021 BX = 0022 CX = 0000 DX = 0000 SP = 0004 BP = 20CD SI = 0000 DI = 0000
```

```
DS = 17DA ES = 17DA SS = 9FFF CS = 17DA IP = 1006 NV UP EI PL NZ AC PE CY
```

```
17DA: 1006 F4 HLT
```

Initially the program is loaded in the memory location 17DA:1000 to 17DA:1006. Then the object codes of the program will be visualized after execution of the U 1000 1006 command. If the above program is executed by the G1006 command, the result will be displayed on the screen. As the result is more than 8 bits, the content of AL is 21 which is LSB and the CY flag is set to detect the MSB.

7.3.2 Program for Addition of Two 16-Bit Numbers with Sum is more than 16 Bits

Assume the first 16-bit number FFFFH is stored in the AX register. The second 16-bit number, 2333H is stored in the BX register. After addition, the result will be stored in AX. The program flow chart for addition

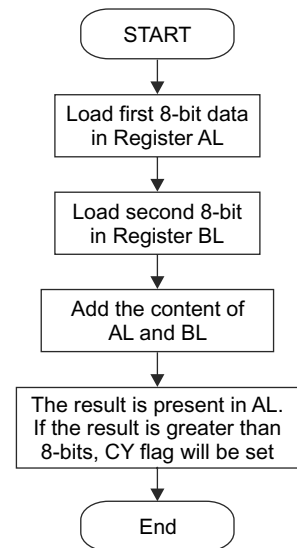


Fig. 7.8 Flow chart for addition of two 8-bit numbers

of two 16-bit numbers with sum is more than 16 bits, is depicted in Fig. 7.9.

Algorithm

1. Load the first 16-bit number in AX.
2. Store second 16-bit number in BX.
3. Addition of first and second numbers
4. Result is stored in AX and carry flag is set if sum is more than 16 bits.

C:\>DEBUG

-A 1000

17DA: 1000 MOV AX, FFFF; 16-bit data in AX

17DA: 1003 MOV BX, 2333; 16-bit data in BX

17DA: 1006 ADD AX, BX; Contents of BX is added to AX

17DA: 1008 HLT

17DA: 1009

-U 1000 1008

17DA: 1000 B8 FF FF MOV AX, FFFF

17DA:1003 BB 33 23 MOV BX, 2333

17DA:1006 01 D8 ADD AX, BX

17DA:1008 F4 HLT

-G 1008

AX = 2332 BX = 2333 CX = 0000 DX = 0000 SP = 0004 BP = 20CD SI = 0000 DI = 0000

DS = 17DA ES = 17DA SS = 9FFF CS = 17DA IP = 1008 NV UP EI PL NZ AC PO CY

17DA: 1008 F4 HLT

-

The above program is loaded in the memory location 17DA:1000 to 17DA:1008. After editing the program, the U 1000 1008 command is used to display the object codes of the program. This program is executed by G1008 command and result will be displayed on the screen. The content of AX is 2332 which is the addition of FFFFH and 2333H. As the sum is more than 16 bits, carry flag CY is set.

7.3.3 Program for Addition of a String of Words with 16-Bit Sum

Assume the number of 16-bit data is stored in the CX register and a string of words are stored in 17DA: 0300 to 17DA: 0309. After addition, the result is stored in BX. Initially, the content of BX is 0000H. The program flow chart for addition of a string of words with 16-bit sum is given in Fig. 7.10.

Algorithm

1. Initialize the SI register with 0300H as source address of data.
2. Load number of bytes to be added in the CX register.
3. Load a word in AX from the source specified by SI and SI is incremented by 2.
4. Addition of AX content and BX content.
5. Move content of AX to BX.
6. Continue steps-3 to 5 until CX = 0.

C:\>DEBUG

-A 0100

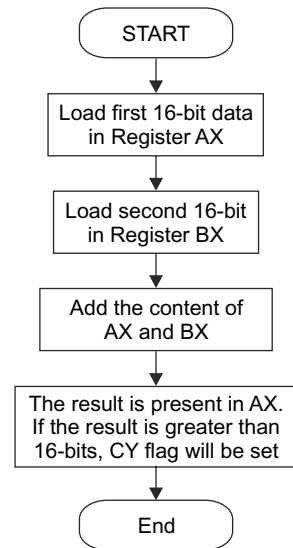


Fig. 7.9 Flow chart for addition of two 16-bit numbers

```

17DA: 0100 MOV SI, 0300; Source address in SI
17DA: 0103 MOV CX, 0005; Count value is loaded in CX
17DA: 0106 MOV AX, [SI]; Load AX with data which is located by SI
17DA: 0108 ADD AX, BX; Contents of BX in AX
17DA: 010A INC SI; Increment SI
17DA: 010B INC SI; Increment SI
17DA: 010C MOV BX, AX; Contents of BX in AX
17DA: 010E DEC CX; Decrement CX
17DA: 010F JNZ 0106; Jump to 0106 if CX≠0
17DA: 0111 HLT
17DA: 0112
-ECS:0300
17DA:0300 00.01 00.01 00.02 00.02 00.03 00.03 00.04 00.04
17DA:0308 00.05 00.05
-G 0111
AX = 0F0F BX = 0F0F CX = 0000 DX = 0000 SP = FFEE BP = 0000
SI = 030A DI = 0000
DS = 17DA ES = 17DA SS = 17DA CS = 17DA IP = 0111 NV UP EI
PL ZR NA PE NC
17DA: 0111 F4 HLT
-
    
```

The above program is entered in the memory location 17DA: 0100 to 17DA: 0111. The U 0100 0111 command can be used to display the object code of the program. Five 16-bit data 0101, 0202, 0303, 0404 and 0505 are entered by the command -ECS: 0300 17DA: 0300 00.01 00.01 00.02 00.02 00.03 00.03 00.04 00.04 17DA: 0308 00.05 00.05. After addition of 0101, 0202, 0303, 0404 and 0505, we get 0F0F. Therefore, when the program is executed by the G0111 command, the result will be displayed on the screen as the content of BX and AX registers, 0F0F.

7.3.4 Program for Subtraction of Two 16-Bit Numbers

Consider first 16-bit number is in the AX register and the second number is in the BX register. After subtraction, the result will be stored in AX. The program flow chart for subtraction of two 16-bit numbers is illustrated in Fig. 7.11.

Algorithm

1. Load first number in Register AX.
2. Load second number in Register BX.
3. Subtract BX from AX.

```

C:\>DEBUG
-A1000
17DA: 1000 MOV AX, FFFF ; 16 bit data in AX
17DA: 1003 MOV BX, 6666 ; 16 bit data in BX
    
```

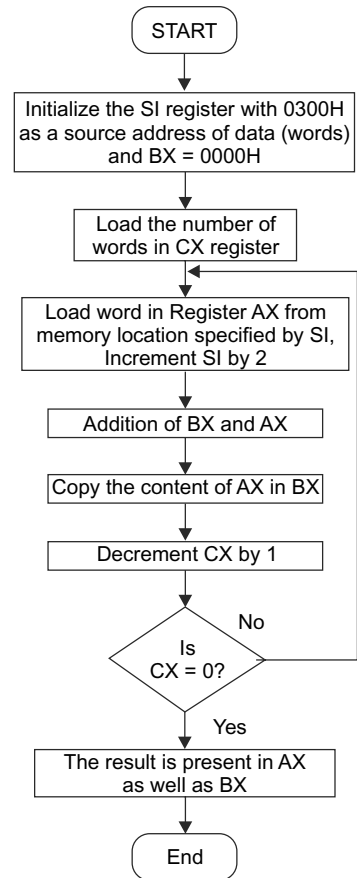


Fig. 7.10 Flow chart for addition of a string of words

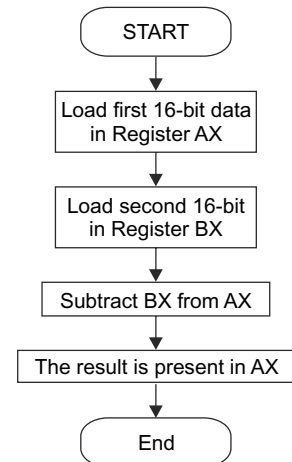


Fig. 7.11 Flow chart for subtraction of two 16-bit numbers

```

17DA: 1006 SUB AX, BX    ; Contents of BX is subtracted from AX
17DA: 1008 HLT
17DA: 1009
-U 1000 1008
17DA: 1000 B8FFFF    MOV AX,FFFF
17DA: 1003 BB6666    MOV BX,6666
17DA: 1006 29D8     SUB AX,BX
17DA:1008 F4       HLT
-G1008
AX = 9999 BX = 6666 CX = 0000 DX = 0000 SP = 0004 BP = 20CD SI = 0000 DI = 0000
DS = 17DA ES = 17DA SS = 9FFF CS = 17DA IP = 1008 NV UP EI NG NZ NA PE NC
17DA: 1008 F4 HLT
-

```

The program for subtraction of two 16-bit numbers is entered in the memory location 17DA: 1000 to 17DA: 1008. The U 1000 1008 command is used to display the object codes of the program. When the above program is executed by the G1008 command, the result will be available in AX. Therefore, the result is content of AX, i.e., 9999H which is the subtraction of FFFFH and 6666H.

7.3.5 Program for 2's Complement of a 16-Bit Number

Assume that the 16-bit number is stored in the AX register. Find the two's complement of the number and store it in the BX register. The program flow chart to find 2's complement of a 16-bit number is shown in Fig. 7.12.

Algorithm

1. Store the 16-bit number in Register AX.
2. Determine 2's complement of AX.
3. Store two's complement of AX in BX.

```

C:\>DEBUG
-A 1000
17DA: 1000 MOV AX, 2244 ; 16-bit data in AX
17DA: 1003 NEG AX      ; 2's complement of 16-bit data
17DA: 1005 MOV BX, AX  ; Result is stored in BX
17DA: 1007           HLT
17DA: 1008
-U 1000 1007
17DA: 1000 B8 44 22    MOV AX,2244
17DA: 1003 F7 D8      NEG AX
17DA:1005 89 C3      MOV BX,AX
17DA: 1007 F4       HLT
-G 1007
AX = DDBC BX = DDBC CX = 0000 DX = 0000 SP = 0004 BP = 20CD SI = 0000 DI = 0000
DS = 17DA ES = 17DA SS = 9FFF CS = 17DA IP = 1007 NV UP EI NG NZ AC PO CY
17DA: 1007 F4 HLT
-

```

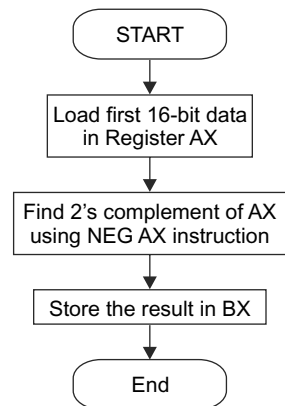


Fig. 7.12 Flow chart for 2's complement a 16-bit numbers

The program for 2's complement of a 16-bit number is stored in the memory location 17DA:1000 to

17DA:1007. The U 1000 1006 command is used to display the object codes of program. When this program is executed by G1007 command and result will be available in AX. Therefore the content of AX is copied into BX.

7.3.6 Program for 2's Complement of a String of Words

Assume a string of words are stored in 17DA: 0300 to 17DA: 030B. The number of words is stored in the CX register. Determine the 2's complement of the string words and store at the destination address 17DA: 0400 to 17DA: 040B. The program flow chart to find 2's complement of a string of words is depicted in Fig. 7.13.

Algorithm

1. Initialize SI with 0300H as source address of data.
2. Load number of words in CX register and initialize DI with 0400H as destination address.
3. Load a word in AX from the source specified by SI and SI is incremented by 2.
4. Determine 2's complement of AX and store at destination address represented by DI. DI is incremented by 2.
5. Continue steps-3 and 4 until CX = 0.

C:\>DEBUG

-A 0100

17DA: 0100 MOV SI, 0300 ; Load 0300H in source index register

17DA: 0103 MOV CX, 0005 ; Load number of bytes in the CX register

17DA: 0106 MOV DI, 0400 ; Load 0400H in destination index register

17DA: 0109 LODSW ; Load AX with data addressed by SI ans SI = SI+2

17DA: 010A NEG AX ; 2's complement of 16-bit data

17DA: 010C STOSW ; Store AX addressed by DI and DI = DI+2 Result is stored in memory

17DA: 010D LOOPNZ 0109 ; Loop unless CX = 0

17DA:010F HLT

17DA:0110

-ECS:0300

17DA: 0300 00.01 00.01 00.02 00.02 00.03 00.03 00.04 00.04

17DA: 0308 00.05 00.05 00.06 00.06

-G 010F

AX = FAFB BX = 0000 CX = 0000 DX = 0000 SP = FFEE BP = 0000 SI = 030A DI = 040A

DS = 17DA ES = 17DA SS = 17DA CS = 17DA IP = 010F NV UP EI NG NZ AC PO CY

17DA: 010F F4 HLT

-ECS: 0400

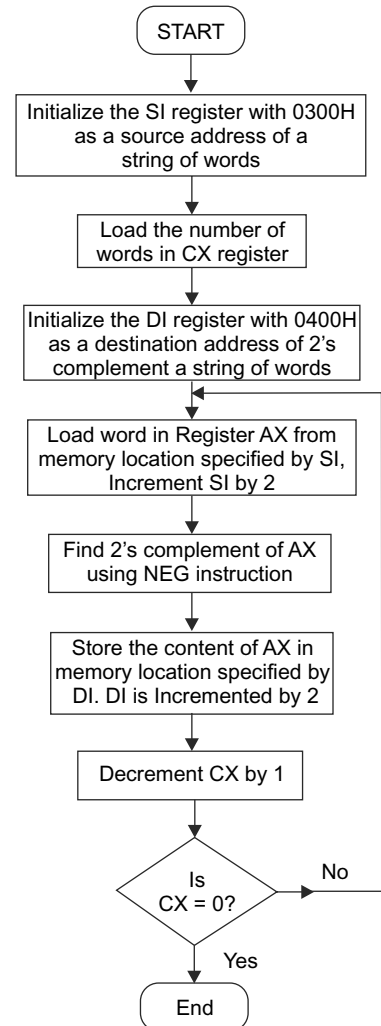


Fig. 7.13 Flow chart for 2's complement of string of words

17DA: 0400 FF. FE. FE. FD. FD. FC. FC. FB.

17DA: 0408 FB. FA. 00.

The above program is used to find 2's complement of a string of five 16-bit numbers which are stored in the memory location 17DA: 0300 to 17DA: 0309 and the program is stored in the memory location 17DA: 0100 to 17DA: 010F. The U 0100 010F command can be used to display the object codes of the program. ECS: 0300 command is used to five 16-bit numbers such as 0101, 0202, 0303, 0404 and 0505. This program can be executed by the G010F command and result will be stored in destination memory location 17DA: 0400 to 17DA: 0409. To see the result, the ECS: 0400 command should be used and result will be displayed as given above.

7.3.7 Program to Multiply Two 16-Bit Numbers

Assume that the first number 1111H is stored in the AX register and the second number 2222H is stored in the BX register. Multiply contents of AX by BX and the result is to be stored in DX and AX registers. The program flow chart to multiply two 16-bit numbers is given in Fig. 7.14.

Algorithm

1. Load first number in Register AX.
2. Store the second data in Register BX.
3. Multiply contents of AX by BX.

C:\>DEBUG

-A 1000

17DA: 1000 MOV AX, 1111 ; 16-bit multiplicand in AX

17DA: 1003 MOV BX, 2222 ; 16-bit multiplicand in AX

17DA: 1006 MUL BX ; Multiply contents of AX by BX

17DA: 1008 HLT

17DA: 1009

-U 1000 1008

17DA: 1000 B8 11 11 MOV AX,1111

17DA: 1003 BB 22 22 MOV BX,2222

17DA: 1006 F7 E3 MUL BX

17DA:1008 F4 HLT

-G 1008

AX = 8642 BX = 2222 CX = 0000 DX = 0246 SP = 0004 BP = 20CD SI = 0000 DI = 0000

DS = 17DA ES = 17DA SS = 9FFF CS = 17DA IP = 1008 OV UP EI NG NZ NA PO CY

17DA: 1008 F4 HLT

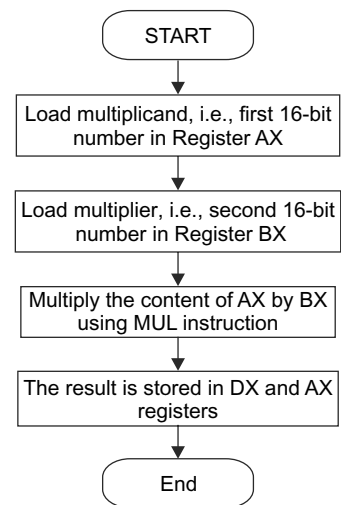


Fig. 7.14 Flow chart to multiply two 16-bit numbers

The above program is used to multiply two 16-bit numbers and it is stored in the memory location 17DA: 1000 to 17DA: 1008. The U 0100 0108 command is used to display the object codes of the program. This program can be executed by G1008 command and result will be stored in DX and AX. The result after multiplication of 1111H and 2222H is more than 16 bits. Then lower 16 bits result is stored in AX and upper 16 bits (most significant bit) of result is stored in DX. Here, the result is the content of DX = 0246 and AX = 8642.

7.3.8 Program to Divide Two 16-Bit Numbers

Assume that the first number FFFFH is stored in the AX register and the second number 2222H is stored in the CX register. Divide AX by CX and the result is to be stored in DX and AX registers. The program flow chart to divide two 16-bit numbers is depicted in Fig. 7.15.

Algorithm

1. Load first number in Register AX.
2. Store the second data in Register CX.
3. Divide content of AX by CX and result in Registers AX and DX.

C:\>DEBUG

-A 1000

17DA: 1000 MOV AX, FFFF ; 16-bit dividend in AX

17DA: 1003 MOV CX, 2222 ; 16-bit divisor in CX

17DA: 1006 DIV CX ; Divide contents of AX by CX

17DA: 1008 HLT

17DA:1009

-U 1000 1008

17DA:1000 B8 FF FF MOV AX, FFFF

17DA:1003 B9 22 22 MOV CX, 2222

17DA:1006 F7 F1 DIV CX

17DA:1008 F4 HLT

-G 1008

AX = 0007 BX = 0000 CX = 2222 DX = 1111 SP = 0004 BP = 20CD SI = 0000 DI = 0000

DS = 17DA ES = 17DA SS = 9FFF CS = 17DA IP = 1008 NV UP EI NG NZ NA PO NC

17DA:1008 F4 HLT

-

Quotient = 0007 in AX and remainder 1111 in DX register

The above program is used to divide two 16-bit numbers and is stored in the memory location 17DA: 1000 to 17DA: 1008. The U 0100 0108 command is used to display the object codes of the program. If the program is executed by G1008 command, the result will be stored in DX and AX. Quotient is stored in AX and remainder is stored in DX. Hence the result is quotient = content of AX = 0007H and remainder = content of DX = 1111H.

7.3.9 Program for Decimal Addition of two 16-bit Numbers and the Sum is 16 Bits

Two decimal numbers 4477H and 2299H are stored in DX and BX registers respectively. After addition, the result is to be stored in the CX register. The program flow chart for decimal addition of two 16-bit numbers and the sum is 16 bits, is shown in Fig. 7.16

Algorithm

1. Load first number in Register DX.
2. Load the second number in Register BX.
3. Move content of BL into AL and add content of DL with AL.
4. Decimal adjustment of AL after addition and store AL content in CL.

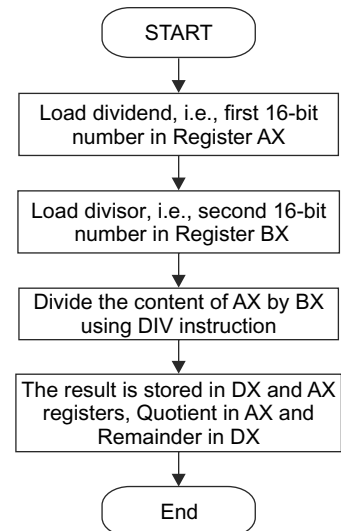


Fig. 7.15 Flow chart to divide two 16-bit numbers

5. Move content of BH into AL and add content of DH and AL with carry.
6. Decimal adjustment of AL after addition and store AL content in CH.

C:\>DEBUG

-A0100

17B3: 0100 MOV DX, 4477 ; 16-bit data in DX

17B3: 0103 MOV BX, 2299 ; 16-bit bit data in BX

17B3: 0106 MOV AL, BL ; Move the content of BL to AL

17B3: 0108 ADD AL, DL ; Contents of DL is added to AL

17B3: 010A DAA ; Decimal adjustment after addition

17B3: 010B MOV CL, AL ; Move the content of AL to CL

17B3: 010D MOV AL, BH ; Move the content of BL to AL

17B3: 010F ADC AL, DH ; Contents of DH is added to AL

17B3: 0111 DAA ; Decimal adjustment after addition

17B3: 0112 MOV CH, AL ; Move the content of AL to CH

17B3: 0114 HLT

17B3: 0115

-G 0114

AX = 0067 BX = 2299 CX = 6776 DX = 4477 SP = FFEE BP = 0000 SI = 0000 DI = 0000

DS = 17B3 ES = 17B3 SS = 17B3 CS = 17B3 IP = 0114 NV UP EI PL NZ NA PO NC

17B3: 0114 F4 HLT

The program for decimal addition of two 16-bit numbers is stored in the memory location 17B3: 0100 to 17B3: 0114. The U 0100 0114 command can be used to display the object codes of the program. This program is executed by G0114 command and result will be stored in the CX register. Hence, the result is content of CX = 6776 which is decimal addition of 4477 and 2299.

7.3.10 Program for Addition of Two String Decimal Numbers

The first string decimal numbers 11, 22, 33, 44 and 55 are stored in memory locations 17B3: 0100 to 17B3: 0104. The second string decimal numbers 22, 33, 55, 66 and 77 are stored in memory locations 17B3: 0400 to 17B3: 0404. After addition of two string decimal numbers, the result will be stored in memory locations 17B3: 0500 to 17B3: 0504. The program flow chart for addition of two string decimal numbers is depicted in Fig. 7.17.

Algorithm

1. Initialize SI and BX as offset address of first and second string decimal numbers.
2. Initialize DI as offset address of result and number of decimal numbers in a string is loaded in CX.
3. Move first decimal number from first string to AL.
4. Add first decimal number from second string decimal numbers with AL.
5. Decimal adjustment of AL after addition and store AL content in destination address represented by DI. Decrement CX.
6. Increment SI, BX and DI.
7. Move next decimal number from first string to AL.

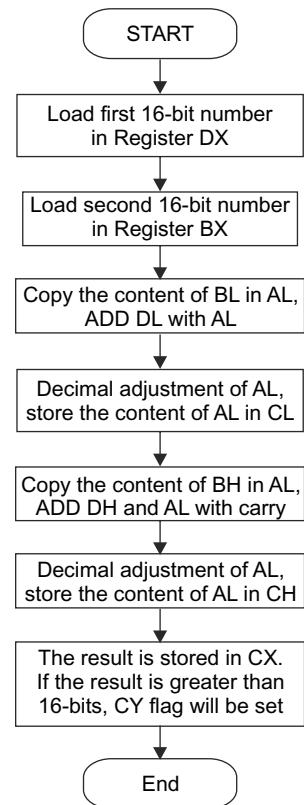


Fig. 7.16 Flow chart for decimal addition of two 16-bit numbers

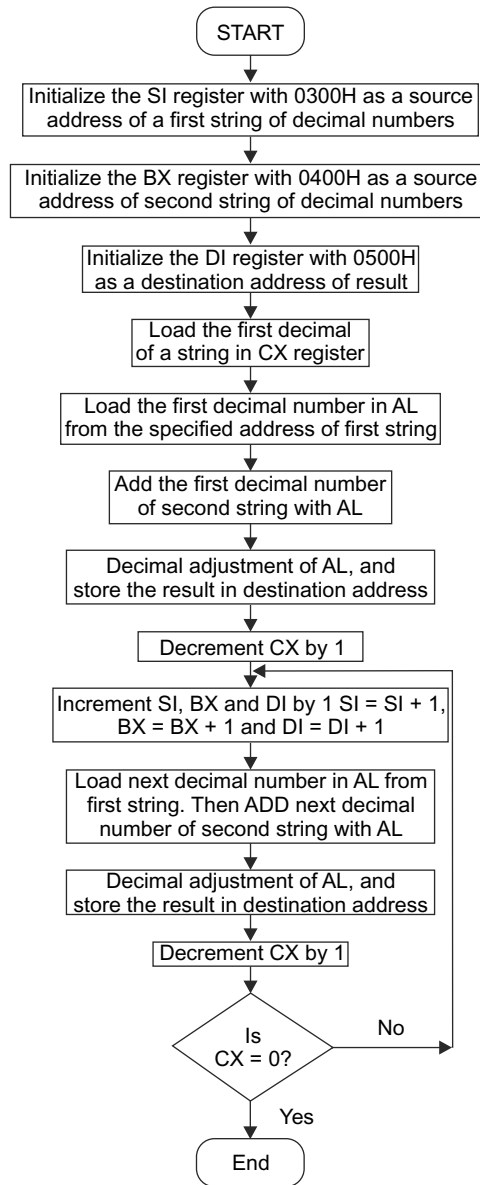


Fig. 7.17 Flow chart for decimal addition of two string decimal numbers

8. Next decimal number from second string decimal numbers is added with AL and carry.
9. Decimal adjustment of AL after addition and store AL content in destination address represented by DI.
10. Decrement CX. If CX≠0, jump to step 6.

C:\>DEBUG

-A0100

17B3: 0100 MOV SI, 0300 ; Load 0300H in SI as offset address of first string


```

17B3: 0103 MOV BX, 0400 ; Load 0400H in BX as offset address of second string
17B3: 0106 MOV DI, 0500 ; Load 0500H in DI as offset address of destination to store result
17B3: 0109 MOV CX, 0005 ; Number of decimal numbers in a string is loaded in CX
17B3: 010C MOV AL, [SI] ; Load decimal number from first string
17B3: 010E ADD AL, [BX] ; Add decimal number of second string to AL
17B3: 0110 DAA ; Decimal adjustment after addition
17B3: 0111 MOV [DI], AL ; Store the content of AL into destination memory location
17B3: 0113 DEC CX ; Decrement CX by 1
17B3: 0114 INC SI ; Increment SI by 1
17B3: 0115 INC BX ; Increment BX by 1
17B3: 0116 INC DI ; Increment DI by 1
17B3: 0117 MOV AL, [SI] ; Load next decimal number from first string
17B3: 0119 ADC AL, [BX] ; Add next decimal number of second string to AL with carry
17B3: 011B DAA ; Decimal adjustment after addition
17B3: 011C MOV [DI], AL ; Store the content of AL into destination memory location
17B3: 011E DEC CX ; Decrement CX by 1
17B3: 011F JNZ 0114 ; if CX ≠ 0, jump to offset address 0114
17B3:0121 HLT
17B3:0122
-ECS:0300
17B3:0300 00.11 00.22 00.33 00.44 00.55
-ECS:0400
17B3:0400 00.22 00.33 00.55 00.66 00.77
-G0121
AX = 0033 BX = 0404 CX = 0000 DX = 0000 SP = FFEE BP = 0000 SI = 0304 DI = 0504
DS = 17B3 ES = 17B3 SS = 17B3 CS = 17B3 IP = 0121 NV UP EI PL ZR NA PE CY
17B3: 0121 F4 HLT
-ECS: 0500
17B3: 0500 33. 55. 88. 10. 33.

```

The above program can be used for addition of two string decimal numbers. The U 0100 0121 command can be used to display the object codes of the program. The ECS: 0300 and ECS: 0400 command are used to enter first string 11 22 33 44 55 and second string 22 33 55 66 77 respectively. The program is executed by G0121 command and after addition result 33 55 88 10 33 will be stored in memory location 17B3: 0500 to 17B3: 0404.

7.3.11 Program to Rotate a 16-Bit Number Left through Carry by One Bit

The 16-bit number is stored in AX and rotate left one bit through carry. Store the result in AX. The program flow chart to rotate a 16-bit number left through carry by one bit is illustrated in Fig. 7.18.

Algorithm

1. Load 16-bit data, 1234 in Register AX.
2. Rotate the content of AX left through carry by one bit.

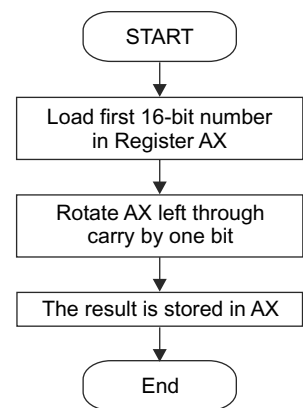


Fig. 7.18 Flow chart to rotate left a 16-bit number by one bit

```
C:\>DEBUG
-A0100
17B3: 0100 MOV AX, 1234 ; LOAD 1234 IN AX
17B3: 0103 RCL AX, 1 ; Rotate AX left by 1 bit
17B3: 0105 HLT
-G 0105
AX = 2468 BX = 0000 CX = 0000 DX = 0000 SP = FFEE BP = 0000 SI = 0000 DI = 0000
DS = 17B3 ES = 17B3 SS = 17B3 CS = 17B3 IP = 0105 NV UP EI PL NZ NA PO NC
17B3: 0105 F4 HLT
```

The program for rotating a 16-bit number left through carry by one bit is stored in the memory location 17B3: 0100 to 17B3: 0105. The U 0100 0105 command can be used to display the object codes of the program. After execution of this program by G0105 command, the result will be stored in AX register. So the result is content of AX = 2468.

7.3.12 Program to Rotate a 16-Bit Number Left through Carry by Four Bits

The 16-bit number is stored in AX and rotate left four bits through carry. Store the result in AX. The program flow chart to rotate a 16-bit number left through carry by four bits is shown in Fig. 7.19.

Algorithm

1. Load 16-bit data, 1234 in Register AX.
2. Load number of bits rotate in CL.
3. Rotate the content of AX left through carry by four bits.

```
C:\>DEBUG
-A0100
17B3: 0100 MOV AX, 1234 ; Load 1234 in AX
17B3: 0103 MOV CL, 04 ; Load count value in CL
17B3: 0105 RCL AX, CL ; Rotate AX left by 4 bits
17B3:0107 HLT
-U 0100 0107
17B3: 0100 B8 34 12 MOV AX, 1234
17B3: 0103 B1 04 MOV CL, 04
17B3: 0105 D3 D0 RCL AX, CL
17B3: 0107 F4 HLT
-G0107
AX = 2340 BX = 0000 CX = 0004 DX = 0000 SP = FFEE BP = 0000 SI = 0000 DI = 0000
DS = 17B3 ES = 17B3 SS = 17B3 CS = 17B3 IP = 0107 NV UP EI PL NZ NA PO CY
17B3: 0107 F4 HLT
```

The above program is used for rotating a 16-bit number left through carry by four bits and is stored in the memory location 17B3: 0100 to 17B3: 0107. The U 0100 0107 command is used to display the object codes of the program. When this program is executed by G0107 command, the result will be stored in AX register. Consequently the result is content of AX = 2340.

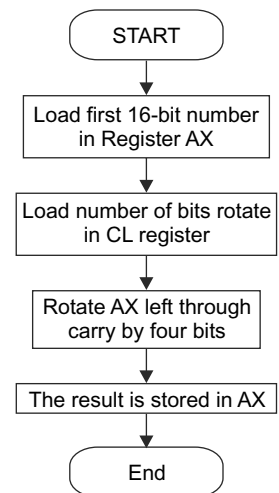


Fig. 7.19 Flow chart to rotate left a 16-bit number by four bits

7.3.13 Program for Left Shift of a 16-Bit Number by Two Bits

The 16-bit number is stored in AX and left shifted two bits. Store the result in memory location starting from 17DA: 1010. The program flow chart for left shift of a 16-bit number by two bits is shown in Fig. 7.20.

Algorithm

1. Load 16-bit data, 4567 in Register AX.
2. Load number of bits shift in CL.
3. The content of AX left shifted through carry by two bits.

C:\>DEBUG

-A1000

17DA: 1000 MOV AX, 4567 ; 16-bit data 4567H in AX

17DA: 1003 MOV CL, 02 ; Load number of shift in CL

17DA: 1005 SHL AX, CL ; Data is shifted left by two bit

17DA: 1007 MOV [1010], AX; Save the content of AX in 17DA: 1010

17DA:100A HLT

-G100A

AX = 159C BX = 0000 CX = 0002 DX = 0000 SP = 0004 BP = 20CD SI = 0000 DI = 0000

DS = 17DA ES = 17DA SS = 9FFF CS = 17DA IP = 100A NV UP EI PL NZ NA PE CY

17DA:100A F4 HLT

-ECS: 1010

17DA: 1010 9C. 15.

The above program is used for shifting a 16-bit number left by two bits and is stored in the memory location 17DA: 1000 to 17DA: 100A. The U 1000 100A command can be used to display the object codes of the program. When this program is executed by G100A command and result will be available in the AX register and the content of AX = 159C is stored in 17DA: 1010 and it can be displayed by ECS: 1010 command.

7.3.14 Program to Find Out the Largest Number from a String of Bytes

The count value of numbers 08H is stored in the CX register and the numbers are stored in the memory locations starting from 17B3: 0300 to 17B3:0307. The largest number will be stored in the memory location 17B3: 0400. The program flow chart to find out the largest number from a string of bytes is given in Fig. 7.21.

Algorithm

1. Initialize SI as source offset address of data and load numbers of bytes in CX register.
2. Initialize Register AL with 00.
3. Compare the content of memory with content of accumulator , if number is above and equal to AL jump to Step 5.
4. Move number from memory to AL.
5. Increment SI.
6. Decrement CX. If CX≠0, jump to Step 3.
7. Store result in memory location 17B3: 0400.

C:\>DEBUG

-A0100

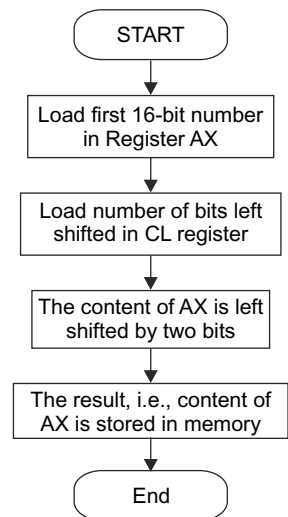


Fig. 7.20 Flow chart for left shift of a 16-bit number by two bits

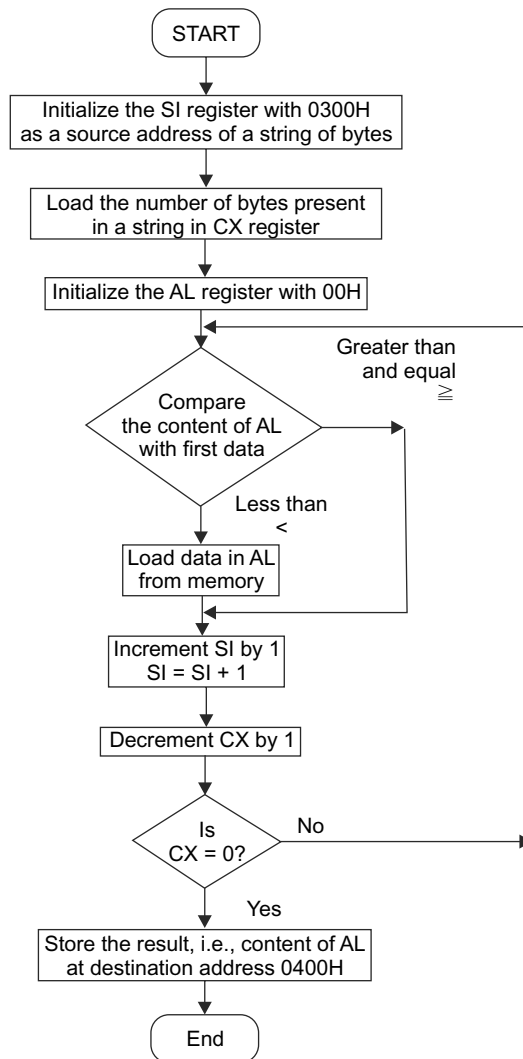


Fig. 7.21 Flow chart to find out the largest number from a string of bytes

17B3: 0100 MOV SI, 0300 ; Initialize SI with 0300H as source offset address of data
 17B3: 0103 MOV CX, 0008 ; Count value of numbers in CX
 17B3: 0106 MOV AL, 00 ; Initialize AL with 00H
 17B3: 0108 CMP AL, [SI] ; compare first data with AL
 17B3: 010A JAE 010E ; Jump to 010E if number is above and equal to AL
 17B3: 010C MOV AL, [SI] ; Load data from memory
 17B3: 010E INC SI ; Increment SI
 17B3: 010F LOOPNZ 0108 ; If CX≠0, jump to 0108
 17B3: 0111 MOV [0400], AL ; Store the content of AL at destination address
 17B3: 0114 HLT
 -ECS: 0300

17B3: 0300 00.11 00.22 00.33 00.FF 00.66 00.AA 00.BB 00.99

-G0114

AX = 00FF BX = 0000 CX = 0000 DX = 0000 SP = FFEE BP = 0000 SI = 0308 DI = 0000

DS = 17B3 ES = 17B3 SS = 17B3 CS = 17B3 IP = 0114 NV UP EI PL NZ NA PO NC

17B3: 0114 F4 HLT

-ECS: 0400

17B3: 0400 FF.

The eight 8-bit data 11, 22, 33, FF, 66, AA, BB and 99 are stored in the memory location starting from 17B3: 0300 to 17B3: 0307. So the largest number is FFH. After execution of above program, the largest number is stored in memory location 17B3: 0400 and it can be displayed by ECS: 0400 command as shown above.

7.3.15 Program to Find Out the Largest Number from a String of Words

The count value of number of words 05H is stored in the CX register. A string of words is stored in the memory locations starting from 17B3: 0300 to 17B3: 0309. The largest word will be stored in memory location 17B3: 0400. The program flow chart to find out the largest number from a string of words is depicted in Fig. 7.22.

Algorithm

1. Initialize SI as source offset address of word and load numbers of words in the CX register.
2. Initialize Register AX with 0000H.
3. Compare the word from memory with content of accumulator AX. If the word is above and equal to AX, jump to Step 5.
4. Move word from memory to AL.
5. Increment SI by 2.
6. Decrement CX. If CX≠0, jump to Step 3.
7. Store the result in the memory location 17B3: 0400.

C:\>DEBUG

-A0100

17B3: 0100 MOV SI, 0300 ; Initialize SI with 0300H as source offset address of word

17B3: 0103 MOV CX, 0005 ; Count value of words in CX

17B3: 0106 MOV AX, 0000 ; Initialize AX with 0000H

17B3: 0109 CMP AX, [SI] ; Compare word from memory with AX

17B3: 010B JAE 010F ; Jump to 010F if above and equal

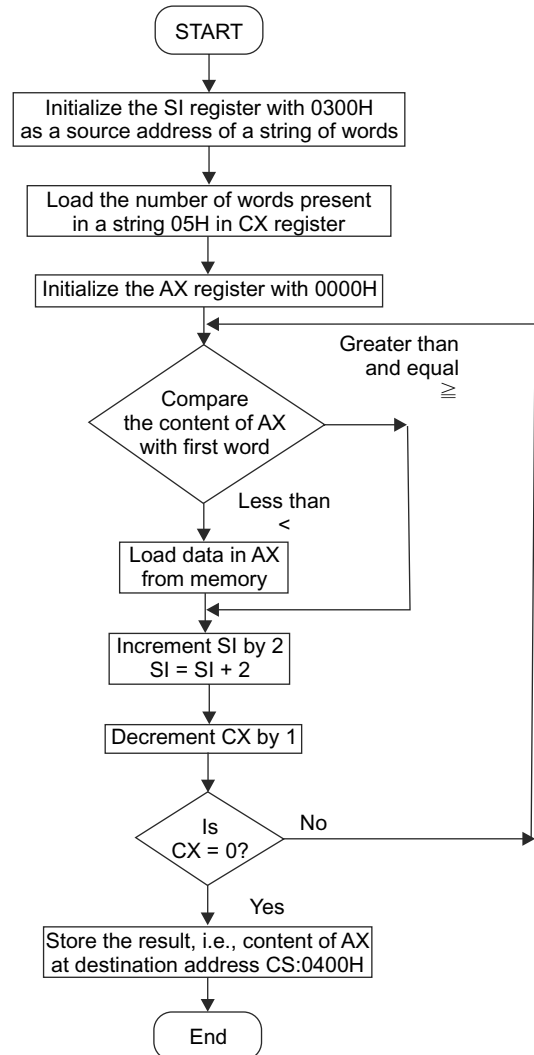


Fig. 7.22 Flow chart to find out the largest number from a string of words

```

17B3: 010D MOV AX, [SI] ; Load word from memory
17B3: 010F INC SI ; Increment SI
17B3: 0110 INC SI ; Increment SI
17B3: 0111 LOOPNZ 0109 ; If CX≠0, jump to 0109
17B3: 0113 MOV [0400], AX ; Store the content of AX at destination address
17B3: 0116 HLT
-ECS: 0300
17B3: 0300 00.11 00.11 00.22 00.22 00.33 00.33 00.FF 00.FF
17B3: 0308 00.99 00.99
-G0116
AX = FFFF BX = 0000 CX = 0000 DX = 0000 SP = FFEE BP = 0000 SI = 030A DI = 0000
DS = 17B3 ES = 17B3 SS = 17B3 CS = 17B3 IP = 0116 NV UP EI PL NZ NA PE NC
17B3: 0116 F4 HLT
-ECS: 0400
17B3: 0400 FF. FF.
    
```

The five 16-bit data or word 1111, 2222, 3333, FFFF, and 9999 are stored in the memory location starting from 17B3: 0300 to 17B3: 0309. Therefore, the largest number is FFFFH. After execution of the above program, the largest number is stored in memory location 17B3: 0400 and it can be displayed by ECS: 0400 command as given above.

7.3.16 Program to Transfer a Block of Data from one Section of Memory to The Other Section of Memory

Assume that a string of bytes is stored in the memory locations starting from 17B3: 0300 to 17B3: 030E. The count value of bytes 0FH is stored in the CX register. Move the block of data starting from memory location 17B3: 0300 to 17B3: 030E to other memory location starting from 17B3: 0400 to 17B3: 040E. The count value of words 0FH is stored in the CX register. The program flow chart to transfer a block of data from one section of memory to other section of memory is shown in Fig. 7.23.

Algorithm

1. Initialize SI as source offset address of bytes and load numbers of bytes in the CX register.
2. Initialize DI as destination offset address of bytes.
3. Move byte from source to destination.
4. Increment SI and DI by 1.
5. Decrement CX. If CX≠0, jump to Step 3.

```

C:\>DEBUG
-A0100
17B3: 0100 MOV SI, 0300 ; Load source address of data in SI
17B3: 0103 MOV DI, 0400 ; Load destination address of data
                        in DI
    
```

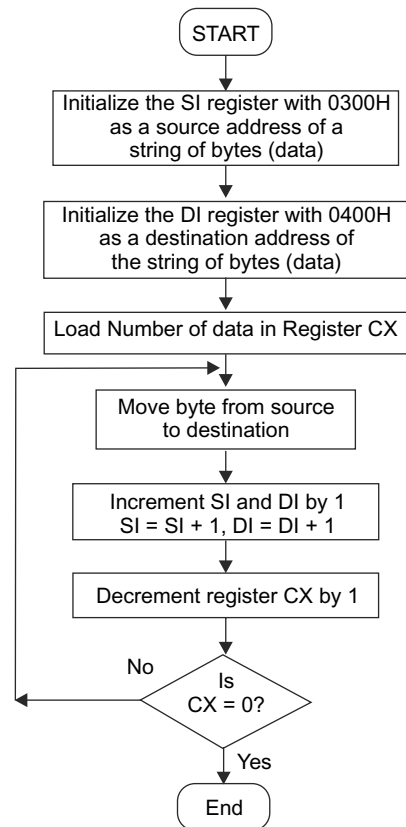


Fig. 7.23 Flow chart to transfer a block of data (bytes) from a section of memory to the other section of memory

```

17B3: 0106 MOV CX, 000F ; Count value for number of bytes in CX
17B3: 0109 MOVSB ; Move byte from source to destination
17B3: 010A LOOPNZ 0109 ; Decrement CX. If CX≠0, jump 0109
17B3: 010C HLT
17B3: 010D
-ECS: 0300
17B3: 0300 00.FF 00.EE 00.DD 00.CC 00.AA 00.BB 00.99 00.88
17B3: 0308 00.77 00.66 00.55 00.44 00.33 00.22 00.11 00.00
17B3: 0310 00.12 00.13
-G010C
AX = 0000 BX = 0000 CX = 0000 DX = 0000 SP = FFEE BP = 0000 SI = 030F DI = 040F
DS = 17B3 ES = 17B3 SS = 17B3 CS = 17B3 IP = 010C NV UP EI PL NZ NA PO NC
17B3: 010C F4 HLT
-ECS: 0400
17B3: 0400 FF. EE. DD. CC. AA. BB. 99. 88.
17B3: 0408 77. 66. 55. 44. 33. 22. 11. 00.
17B3: 0410 00.

```

A block of fifteen 8-bit data FF, EE, DD, CC, AA, BB, 99, 88, 77, 66, 55, 44, 33, 22 and 11 are stored in the memory location starting from 17B3: 0300 to 17B3: 030E. After execution of the above program by the command G010C, the said data is stored in a new memory location string from 17B3: 0400 to 17B3: 040E and it can be displayed by ECS: 0400 command as shown above.

7.3.17 Program to Add Two ASCII Numbers

ASCII numbers are in values from 30H to 39H for representing numbers 0 to 9. Add 39 and 38 which are ASCII numbers representing 9 and 8 respectively. The AAA instruction is used for ASCII adjustment after addition. The program for addition of two ASCII numbers is stored in the memory location 17B3: 0100 to 17B3: 0109. After execution of the ADD AL, BL instruction, the result is 0071H which is stored in the AX register. Thereafter execution of AAA instruction result is available in AX and it is 0107. The program flow chart to add two ASCII numbers is shown in Fig. 7.24.

```

C:\>DEBUG
-A0100
17B3: 0100 MOV AL, 39 ; Load first ASCII number in AL
17B3: 0102 MOV BL, 38; ; Load second ASCII number in BL
17B3: 0104 MOV AH, 00 ; Initialize AH with 00H
17B3: 0106 ADD AL, BL ; Add content of BL with AL
17B3: 0108 AAA ; ASCII adjustment of AX after addition
17B3: 0109 HLT
-G0109
AX = 0107 BX = 0008 CX = 0000 DX = 0000 SP = FFEE BP = 0000 SI = 0000 DI = 0000
DS = 17B3 ES = 17B3 SS = 17B3 CS = 17B3 IP = 0109 NV UP EI PL NZ AC PE CY
17B3: 0109 F4 HLT

```

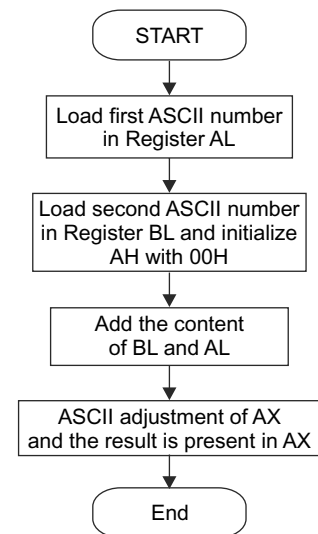


Fig. 7.24 Flow chart for addition of two ASCII numbers

7.3.18 Program to Subtract Two ASCII Numbers

Subtract 39H and 34H which are ASCII numbers representing 9 and 4 respectively. The AAS instruction can be used for ASCII adjustment after subtraction. The program for subtraction of two ASCII numbers is stored in the memory location 17DA: 0100 to 17DA: 010B. After the execution of the SUB AL, BL instruction, the result is 0005H which is stored in the AX register. Afterward execution of AAS and OR AL,30 instruction, the result 0035 is available in AX. The program flow chart to subtract two ASCII numbers is depicted in Fig. 7.25.

C:\>DEBUG

-A0100

```

17DA: 0100  MOV  AL, 39 ; Load first ASCII number in AL
17DA: 0102  MOV  BL, 34 ; Load second ASCII number in BL
17DA: 0104  MOV  AH, 00 ; Initialize AH with 00H
17DA: 0106  SUB  AL, BL ; Subtract content of BL from AL
17DA: 0108  AAS                ; ASCII adjustment of AX after subtraction
17DA: 0109  OR  AL, 30 ; OR 30H with AL
17DA: 010B  HLT
    
```

-G010B

AX = 0035 BX = 0034 CX = 0000 DX = 0000 SP = FFEE BP = 0000 SI = 0000 DI = 0000

DS = 17DA ES = 17DA SS = 17DA CS = 17DA IP = 010B NV UP EI PL NZ NA PE NC

17DA: 010B F4 HLT

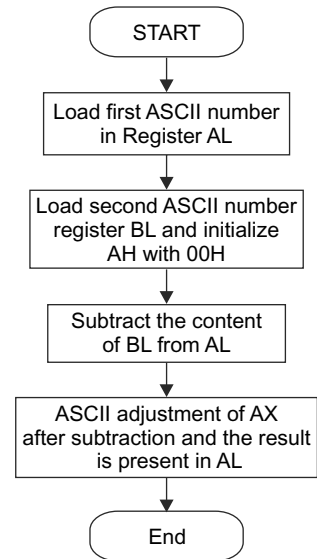


Fig. 7.25 Flow chart for subtraction of two ASCII numbers

7.3.19 Program to Multiply Two ASCII Numbers

The multiplication of ASCII numbers should not be done unless the Most Significant Number (MSN) is cleared. To multiply two ASCII numbers 38H and 32H, initially after removing MSN, the numbers are represented as 08 and 02. The AAM instruction can be used for ASCII adjustment after multiplication. The program for multiplication of two ASCII numbers is stored in the memory location 17B3: 0100 to 17B3: 0108. After execution of the MUL BL instruction, the result 0005H is stored in the AX register. Subsequently execution of AAM instruction, final result 0106 is available in AX. The program flow chart to multiply two ASCII numbers is shown in Fig. 7.26.

C:\>DEBUG

-A0100

```

17B3: 0100  MOV  AL, 08 ; Load first ASCII number in AL
17B3: 0102  MOV  BL, 02 ; Load second ASCII number in BL
17B3: 0104  MUL  BL ; Multiply BL with AL
17B3: 0106  AAM                ; ASCII adjustment after multiplication
17B3: 0108  HLT
    
```

-G 0108

AX = 0106 BX = 0002 CX = 0000 DX = 0000 SP = FFEE BP = 0000

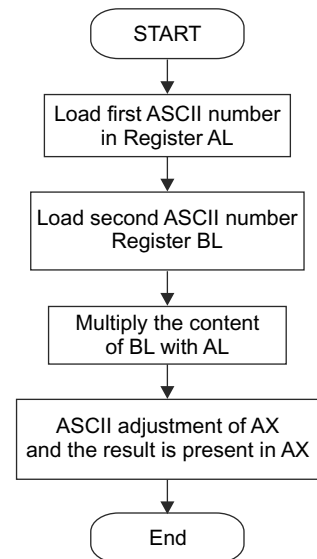


Fig. 7.26 Flow chart to multiply two ASCII numbers

SI = 0000 DI = 0000

DS = 17B3 ES = 17B3 SS = 17B3 CS = 17B3 IP = 0108 NV UP EI PL NZ NA PE NC

17B3: 0108 F4 HLT

7.3.20 Program to Divide Two ASCII Numbers

The division of ASCII numbers should not be done unless the Most Significant Number (MSN) is cleared. Therefore, the AAD instruction requires a two-digit unpacked BCD number before execution. After entering the two-digit unpacked BCD number, AAD is executed to adjust the content of AX. Then AX is divided by an unpacked BCD number to generate results. The program for division of two ASCII numbers (divide 75 by 8) is stored in the memory location 17B3: 0100 to 17B3: 0109. After execution of the above program, result is stored in the AX register. The quotient is 09 is in AL and remainder 03 in AH register. The result is also unpacked BCD. The program flow chart to divide two ASCII numbers is depicted in Fig. 7.27.

C:\>DEBUG

-A0100

17B3: 0100 MOV AX, 0705; Load ASCII number (two digit
unpacked BCD) in AX

17B3: 0103 AAD ; ASCII adjustment before division

17B3: 0105 MOV BL, 08 ; Load second ASCII number in BL

17B3: 0107 DIV BL ; Divide AX by BL

17B3:0109 HLT

-G0109

AX = 0309 BX = 0008 CX = 0000 DX = 0000 SP = FFEE BP = 0000 SI = 0000 DI = 0000

DS = 17B3 ES = 17B3 SS = 17B3 CS = 17B3 IP = 0109 NV UP EI PL NZ NA PE NC

17B3: 0109 F4 HLT

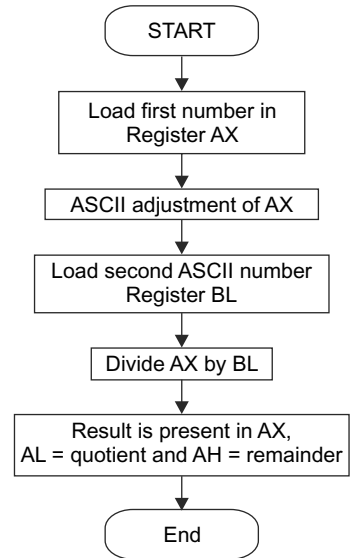


Fig. 7.27 Flow chart to divide two ASCII numbers

7.3.21 Program to Arrange a String of Words in Descending Order

A series of five words 1111H, 5555H, 3333H, 2222H, and 4444H are stored in memory locations from 17B3: 0302 to 17B3: 030B and the number of words is stored in the memory location 17B3: 0300. Arrange the above words in descending order. The program flow chart to arrange a string of words in descending order is shown in Fig. 7.28.

Algorithm

1. Store 0005H, number of words to be arranged in the DX register from memory and store number of comparisons in the CX register.
2. Load the first word in accumulator from memory.
3. Increment SI register by 2 for addressing next word.
4. Compare the next word from memory with the accumulator. Store the smallest word in the accumulator and largest word in the memory.
5. Then next number (word) is compared with the accumulator and store the largest number in memory and smallest number in accumulator.

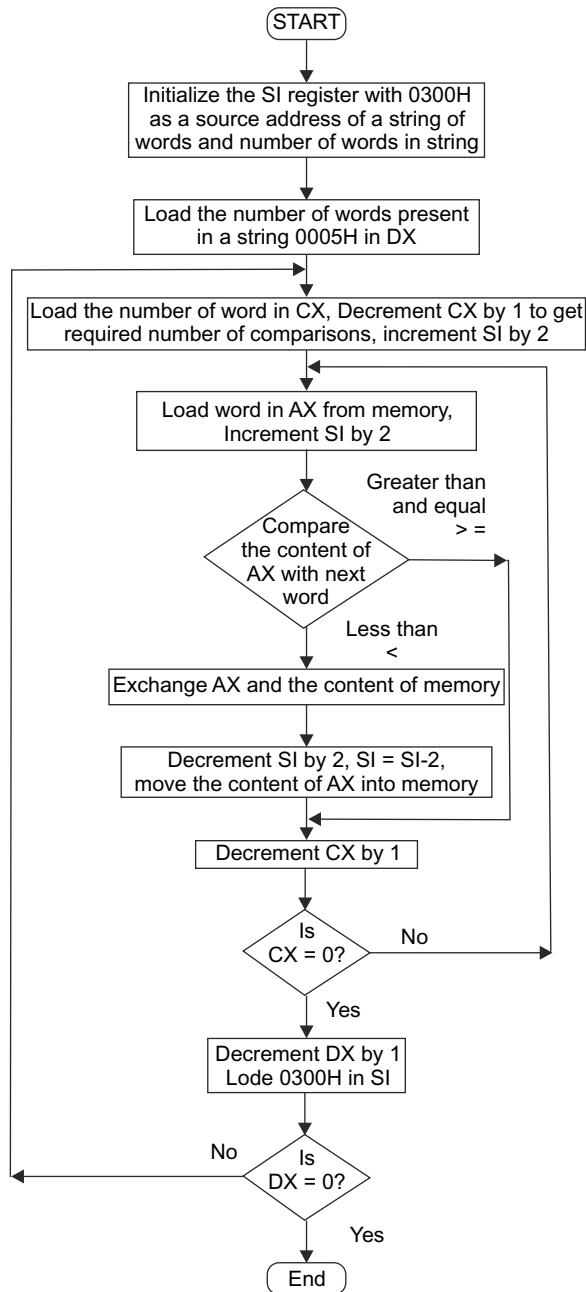


Fig. 7.28 Flow chart to arrange a string of words in descending order

6. This process will continue, till comparisons of all numbers have been completed. After completion of comparison of all numbers, the smallest number in accumulator is stored in memory. In this way, the first process will be completed.

7. At the starting of the second process, the DX register is decremented by one and store number of comparisons in the CX register. Then repeat Steps 2 to 6. After completion of this process, the smallest number is stored in 17B3: 030A and second smallest number in 17B3: 0308.
8. Register DX is decremented by one and the next process starts, if the content of Register DX is not zero. Then repeat steps 2 to 6.

C:\>DEBUG

-A0100

```

17B3: 0100    MOV SI, 0300    ; SI loaded with 0300 as source address
17B3: 0103    MOV DX, [SI]   ; DX loaded with the content of SI
17B3: 0105    MOV CX, [SI]   ; CX loaded with the content of SI
17B3: 010    DEC CX         ; Decrement CX
17B3: 0108    INC SI         ; Increment SI
17B3: 0109    INC SI         ; Increment SI
17B3: 010A    MOV AX, [SI]   ; AX is loaded with word from memory represented by SI
17B3: 010C    INC SI         ; Increment SI
17B3: 010D    INC SI         ; Increment SI
17B3: 010E    CMP AX, [SI]   ; Compare AX with the content of memory represented by SI
17B3: 0110    JNB 0118       ; Jump to 0118
17B3: 0112    XCHG AX, [SI] ; Exchange AX and word stored at memory represented by SI
17B3: 0114    DEC SI         ; Decrement SI
17B3: 0115    DEC SI         ; Decrement SI
17B3: 0116    MOV [SI], AX   ; Move AX to memory represented by SI
17B3: 0118    LOOP 010A      ; CX decrement by 1. If CX≠0, jump to 010A
17B3: 011A    DEC DX         ; Decrement DX
17B3: 011B    MOV SI, 0300   ; SI loaded with 0300 as source address
17B3: 011E    JNZ 0105       ; Jump not zero to 0105
17B3: 0120    HLT

```

-ECS: 0300

```

17B3: 0300  00.05  00.00  00.11  00.11  00.55  00.55  00.33  00.33
17B3: 0308  00.22  00.22  00.44  00.44

```

-G0120

AX = 3333 BX = 0000 CX = 0000 DX = 0000 SP = FFEE BP = 0000 SI = 0300 DI = 0000

DS = 17B3 ES = 17B3 SS = 17B3 CS = 17B3 IP = 0120 NV UP EI PL ZR NA PE NC

17B3:0120 F4 HLT

-ECS: 0300

```

17B3:0300  05.   00.   55.   55.   44.   44.   33.   33.
17B3:0308  22.   22.   11.   11.

```

The above program is used to arrange a string of words in descending order and this program is stored in memory locations 17B3: 0100 to 17B3: 0120. The U 0100 0120 command can be used to display the object codes of the above program. The 5 words (1111H, 5555H, 3333H, 2222H, and 4444H) are entered by the command ECS: 300. Thereafter G0120 is used to execute the program and words will be stored in descending order (5555H, 4444H 3333H, 2222H, and 1111H) as given above.

7.3.22 Program to Arrange a String of Words in Ascending Order

A series of five words 4444H, 1111H, 5555H, 2222H, and 3333H are stored in memory locations from 17B3:

0302 to 17B3: 031B and the number of words is stored in memory location 17B3: 0300. Arrange the above words in ascending order. The program flow chart to arrange a string of words in ascending order is depicted in Fig. 7.29.

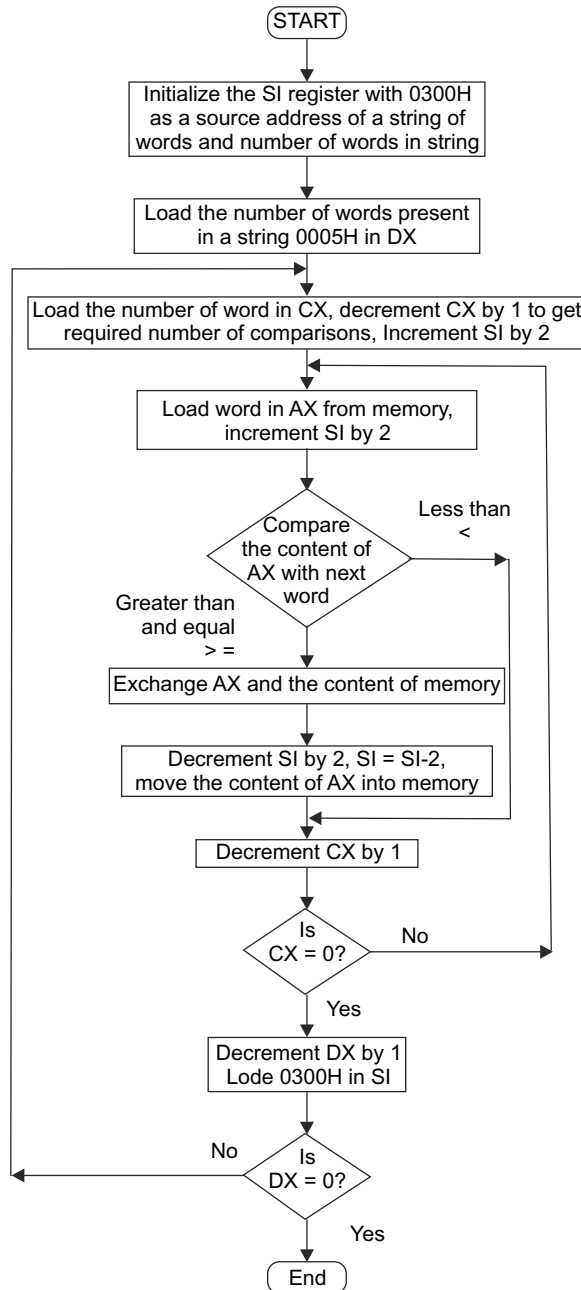


Fig. 7.29 Flow chart to arrange a string of words in ascending order

Algorithm

1. Store 0005H, number of words to be arranged in the DX register from memory and store number of comparisons in the CX register.
2. Load the first word in accumulator from memory.
3. Increment SI register for addressing next word.
4. Compare the next word from memory with the accumulator. Store the largest number in the accumulator and smallest number in the memory.
5. Then next number is compared with the accumulator and store the smallest number in memory and largest number in accumulator.
6. This process will continue, till comparisons of all numbers have been completed. After completion of comparison of all numbers, the largest number in accumulator is stored in memory. In this way, the first process will be completed.
7. At the starting of second process the DX register is decremented by one and store number of comparisons in the CX register. Then repeat steps 2 to 6. After completion of this process, the largest number is stored in 17B3: 0302A and the second largest number in 17B3: 0308.
8. Register DX is decremented and the next process starts, if the content of the DX register is not zero. Then repeat steps 2 to 6.

C:\>DEBUG

-A0100

```

17B3: 0100  MOV SI, 0300    ; SI loaded with 0300 as source address
17B3: 0103  MOV DX, [SI]    ; DX loaded with the content of SI
17B3: 0105  MOV CX, [SI]    ; CX loaded with the content of SI
17B3: 0107  DEC CX          ; Decrement CX
17B3: 0108  INC SI          ; Increment SI
17B3: 0109  INC SI          ; Increment SI
17B3: 010A  MOV AX, [SI]    ; AX is loaded with word from memory represented by SI
17B3: 010C  INC SI          ; Increment SI
17B3: 010D  INC SI          ; Increment SI
17B3: 010E  CMP AX, [SI]    ; Compare AX with the content of memory represented by SI
17B3: 0110  JB 0118         ; Jump to 0118
17B3: 0112  XCHG AX, [SI]   ; Exchange AX and word stored at memory represented by SI
17B3: 0114  DEC SI          ; Decrement SI
17B3: 0115  DEC SI          ; Decrement SI
17B3: 0116  MOV [SI], AX    ; Move AX to memory represented by SI
17B3: 0118  LOOP 010A       ; CX decrement by 1. If CX≠0, jump to
17B3: 011A  DEC DX          ; Decrement DX
17B3: 011B  MOV SI, 0300    ; SI loaded with 0300 as source address
17B3: 011E  JNZ 0105        ; Jump not zero to 0105
17B3: 0120  HLT

```

-ECS: 0300

```
17B3:0300  00.05  00.00  00.44  00.44  00.11  00.11  00.55  00.55
```

```
17B3:0308  00.22  00.22  00.33  00.33  00.66  00.66
```

-G0120

AX = 4444 BX = 0000 CX = 0000 DX = 0000 SP = FFEE BP = 0000 SI = 0300 DI = 0000
DS = 17B3 ES = 17B3 SS = 17B3 CS = 17B3 IP = 0120 NV UP EI PL ZR NA PE CY

```
17B3:0120 F4 HLT
-ECS:0300
17B3:0300 05. 00. 11. 11. 22. 22. 33. 33.
17B3:0308 44. 44. 55. 55. 66. 66.
```

The above program is used to arrange a string of words in ascending order stored in memory locations 17B3: 0100 to 17B3: 0120. The U 0100 0120 command can be used to display the object codes of the above program. The 5 words (4444H, 1111H, 5555H, 2222H, and 3333H) are entered by the command ECS: 300. Thereafter, G0120 is used to execute the program and all words will be stored in ascending order (1111H, 2222H, 3333H, 4444H and 5555H) as given above.

7.3.23 Program to Find Out Square of a Number using Look-up Table

Load the decimal number in the accumulator. Find the square of the decimal number and store it in the memory location 17B3: 0400. The square values of decimal numbers from 0 to 9 are stored in 17B3: 0300 to 17B3: 0309H and used for look-up table. The program flow chart to find out square of a number using look-up table is shown in Fig. 7.30.

Algorithm

1. Store the decimal number in accumulator.
2. Load 03H in Register AH.
3. If the decimal number is 02, the content of AH and AL registers are 03 and 02H respectively. Then the offset address of memory location will be 0302H denoted by the AX register. Move AX content into Register SI.
4. Move the square of decimal number in AL from memory location 17B3: 0302
5. Store the result, square value in 17B3: 0400.

```
C:\>DEBUG
-A0100
17B3: 0100 MOV AL, 02
17B3: 0102 MOV AH, 03
17B3: 0104 MOV SI, AX
17B3: 0106 MOV AL, [SI]
17B3: 0108 MOV DI, 0400
17B3: 010B MOV [DI], AL
17B3: 010D HLT
-ECS: 0300
17B3: 0300 00.00 00.01 00.04 00.09 00.16 00.25 00.36 00.49
-G010D
AX = 0304 BX = 0000 CX = 0000 DX = 0000 SP = FFEE BP = 0000 SI = 0302 DI = 0400
DS = 17B3 ES = 17B3 SS = 17B3 CS = 17B3 IP = 010D NV UP EI PL NZ NA PO NC
17B3:010D F4 HLT
```

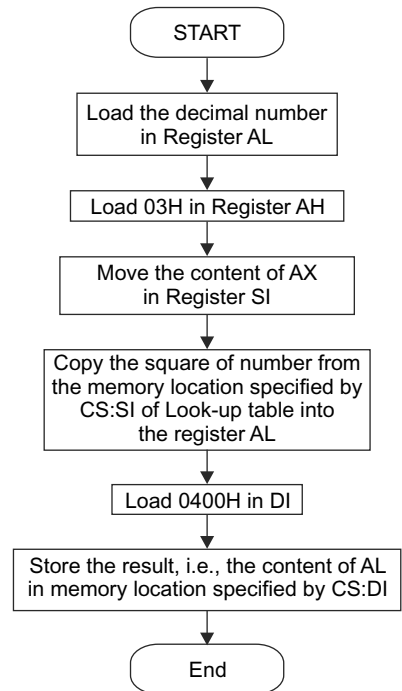


Fig. 7.30 Flow chart to find square of a number using look-up table

```

-ECS: 0400
17B3: 0400 04.
-ECS: 0300
17B3:0300 00.00 00.01 00.04 00.09 00.16 00.25 00.36 00.49
17B3:0308 00.64 00.81
-G010D
AX = 0304 BX = 0000 CX = 0000 DX = 0000 SP = FFEE BP = 0000 SI = 0302 DI = 0400
DS = 17B3 ES = 17B3 SS = 17B3 CS = 17B3 IP = 010D NV UP EI PL NZ NA PO NC
17B3:010D F4 HLT
-ECS: 0400
17B3: 0400 04.

```

The above program can be used to find the square of a decimal number and it is stored memory locations 17B3: 0100 to 17B3: 010D. The U 0100 010D command can be used to display the object codes of the above program. When G010D is used to execute the program and the square value of 02 is 04, it will be stored in memory location 17B3: 0400.

7.3.24 Program to Find the Addition of Two 3 x 3 Matrices

During addition of two matrixes, the corresponding matrix elements are added and a developed new matrix as shown below:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \text{ and } A + B = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & a_{13} + b_{13} \\ a_{21} + b_{21} & a_{22} + b_{22} & a_{23} + b_{23} \\ a_{31} + b_{31} & a_{32} + b_{32} & a_{33} + b_{33} \end{bmatrix}$$

The matrix A is stored in the memory location 17B3: 0200 to 17B3: 0208. The matrix B is stored in the memory location 17B3: 0300 to 17B3: 0308. After addition of two matrixes, the result is stored in the memory location 17B3: 0400 to 17B3: 0311. The program flow chart to find the addition of two matrixes is depicted in Fig. 7.31.

Algorithm

1. Store source address of matrices A and B in SI and DI respectively.
2. Store destination address of $A+B$ in Register BX.
3. Load number of elements of a matrix in Register CX.
4. Initialize Register AX.
5. Load elements of the matrix A from memory to AL and add the corresponding elements of the matrix B with AL.
6. Store addition of two corresponding elements into destination address.
7. Increment SI, and DI by one. Increment BX by 2.
8. Decrement CX by one. If $CX \neq 0$, execute steps 4 to 8

```
C:\>DEBUG
```

```

-A0100
17B3: 0100  MOV SI, 0200 ; SI loaded with 0200 as source address of the matrix A
17B3: 0103  MOV DI, 0300 ; DI loaded with 0300 as source address of the matrix B
17B3: 0106  MOV BX, 0400 ; BX loaded with 0300 as destination address of matrix A + B
17B3: 0109  MOV CX, 0009 ; Load number of elements in a matrix

```

```

17B3: 010C  MOV AX, 0000 ; Initialize AX with
                                0000H
17B3: 010F  MOV AL, [SI] ; Load element of the
                                matrix A in AL
17B3: 0111  ADD AL, [DI] ; Add corresponding
                                element of the matrix
                                B with matrix A
17B3: 0113  JNB 0118 ; Jump no carry to 0118
17B3: 0115  ADD AH, 01 ; Add 01 with AH if
                                carry generated
17B3: 0118  MOV [BX], AX ; store addition of cor-
                                responding elements
                                of (A+B) in destina-
                                tion address
17B3: 011A  INC SI ; Increment SI
17B3: 011B  INC DI ; Increment DI
17B3: 011C  ADD BX, +02 ; Increment BX by 2
17B3: 011F  LOOPNZ 010C ; CX decrement by 1. If
                                CX≠0, jump to 010C

17B3: 0121  HLT
-ECS: 200
17B3: 0200  00.11 00.22 00.33 00.44 00.55 00.6
6 00.77 00.88
17B3: 0208  00.99
-ECS: 300
17B3: 0300  00.11 00.11 00.33 00.44 00.55 00.
66 00.77 00.88
17B3: 0308  00.99
-G0121
AX = 0132 BX = 0412 CX = 0000 DX = 0000 SP = FFEE
BP = 0000 SI = 0209 DI = 0309
DS = 17B3 ES = 17B3 SS = 17B3 CS = 17B3 IP = 0121
NV UP EI PL NZ NA PE NC
17B3: 0121 F4 HLT
-ECS: 0400
17B3: 0400  22. 00. 33. 00. 66. 00. 88. 00.
17B3: 0408  AA. 00. CC. 00. EE. 00. 10. 01.
17B3: 0410  32. 01.
    
```

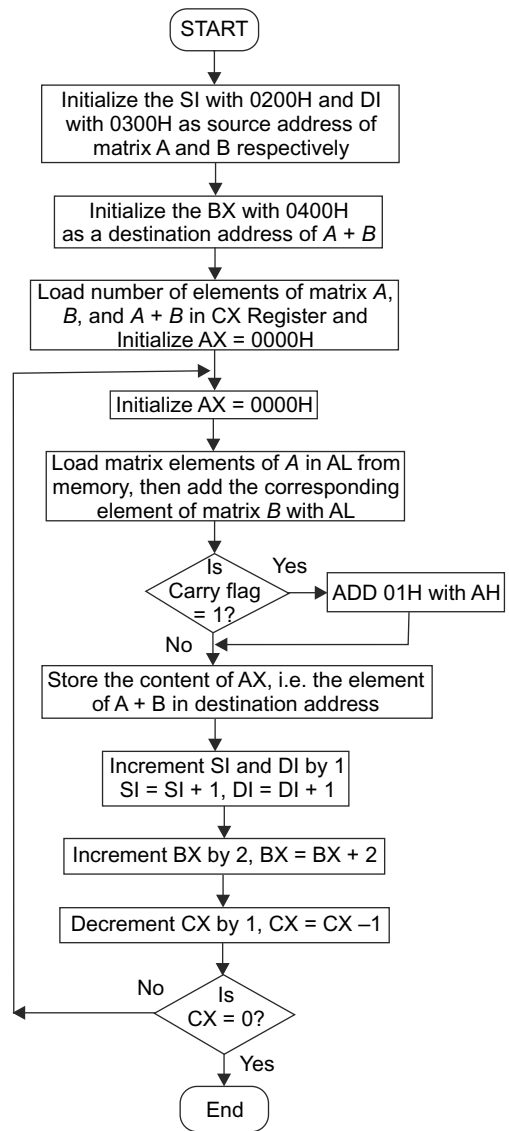


Fig. 7.31 Flow chart for addition of two matrices A and B

The above program can be used to add two 3×3 matrixes and it is stored in memory locations 17B3: 0100 to 17B3: 0121. The U 0100 0121 command will be used to display the object codes of the above program. ECS: 200 is used to enter the matrix elements of A such as 11, 22, 33, 44, 55, 66, 77, 88 and 99. Similarly, the matrix elements of B such as 11, 22, 33, 44, 55, 66, 77, 88 and 99 are entered by ECS: 300 command. When G0121 is executed, the result will be displayed as 0022, 0033, 0066, 0088, 00AA, 00CC, 00EE, 0110 and 0132.

7.3.25 Program to Find the Multiplication of Two 3 x 3 Matrices

Assume two matrixes are $A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$ and $B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$

Then multiplication of two matrix is

$$A \times B = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} & a_{11}b_{13} + a_{12}b_{23} + a_{13}b_{33} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} & a_{21}b_{13} + a_{22}b_{23} + a_{23}b_{33} \\ a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31} & a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} & a_{31}b_{13} + a_{32}b_{23} + a_{33}b_{33} \end{bmatrix}$$

The matrix A is stored in the memory location 17B3: 0200 to 17B3: 0208. The matrix B is stored in the memory location 17B3: 0300 to 17B3: 0308. After addition of two matrixes, the result is stored in the memory location 17B3: 0400 to 17B3: 0411. The program flow chart for multiplication of two matrixes A and B is illustrated in Fig. 7.32.

C:\>DEBUG

-A100

```

17DA: 0100  MOV  BX, 0400  ; BX loaded with 0400 as destination address of matrix A x B
17DA: 0103  MOV  SI, 0200  ; SI loaded with 0200 as source address of the matrix A
17DA: 0106  MOV  DH, 03    ; Load number of row/column in DH
17DA: 0108  MOV  DI, 0300 ; DI loaded with 0300 as source address of the matrix B
17DA: 010B  MOV  CL, 03    ; Load number of row/column in CL
17DA: 010D  MOV  DL, 03    ; Load number of row/column in DL
17DA: 010F  MOV  BP, 0000 ; Initialize BP with 0000H
17DA: 0112  MOV  AX, 0000 ; Initialize AX with 0000H
17DA: 0115  SAHF          ; Save the AH register bits into lower byte of flag register
17DA: 0116  MOV  AL, [SI] ; Load element of matrix A into AL
17DA: 0118  MOV  CH, [DI] ; Move corresponding element of B into CH
17DA: 011A  MUL  CH       ; Multiply corresponding element of B with AL
17DA: 011C  ADD  BP, AX   ; Add content of AX with BP
17DA: 011E  INC  SI      ; Increment SI by one
17DA: 011F  ADD  DI, +03 ; Add 03 with DI
17DA: 0122  DEC  DL      ; Decrement DL by one
17DA: 0124  JNZ  0116   ; If DL ≠ 0, Jump to 0116
17DA: 0126  SUB  DI, +08 ; Subtract 08 from DI
17DA: 0129  SUB  SI, +03 ; Subtract 03 from SI
17DA: 012C  MOV  [BX], BP ; Store element of A x B into destination memory address
17DA: 012E  ADD  BX, +02 ; Add 02 with BX
17DA: 0131  DEC  CL      ; Decrement CL by one
17DA: 0133  JNZ  010D   ; If CL ≠ 0, Jump to 0106
17DA: 0135  ADD  SI, +03 ; Add 03 with SI
17DA: 0138  DEC  DH      ; Decrement DH by one
17DA: 013A  JNZ  0108   ; If DH≠0, Jump to 0106
17DA: 013C  HLT

-ECS: 200
17DA: 0200  00.1  00.1  00.1  00.1  00.1  00.1  00.1  00.1
17DA: 0208  00.1  00.1  00.1

```

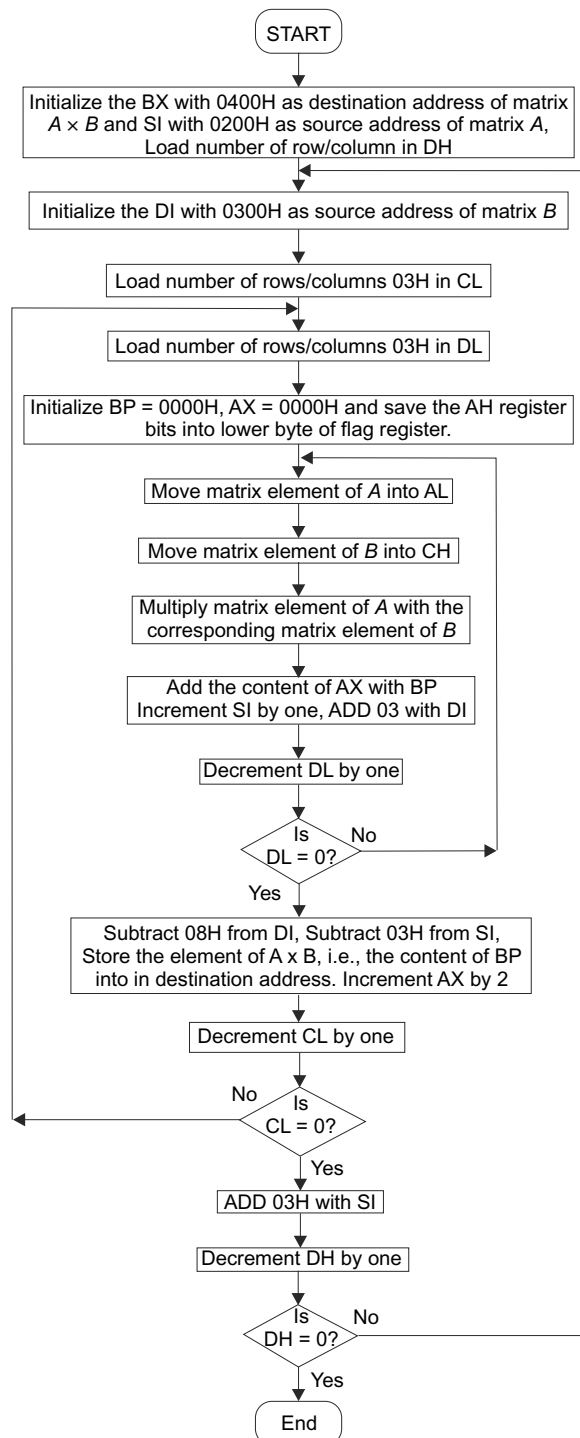


Fig. 7.32 Flow chart for multiplication of two matrices A and B

```

-ECS: 300
17DA: 0300    00.1    00.1    00.1    00.1    00.1    00.1    00.1    00.1
17DA: 0308    00.1    00.1    00.1
-G013C
AX = 0001 BX = 0412 CX = 0100 DX = 0000 SP = FFEE BP = 0003 SI = 0209 DI = 0303
DS = 17DA ES = 17DA SS = 17DA CS = 17DA IP = 013C NV UP EI PL ZR NA PE NC
17DA:013C F4 HLT
-ECS:400
17DA:0400    03.     00.     03.     00.     03.     00.     03.     00.
17DA:0408    03.     00.     03.     00.     03.     00.     03.     00.
17DA: 0410    03.     00.
    
```

The above program can be used to multiply two 3×3 matrices. To verify the result very easily, we assume both A and B matrixes are unit matrices. The ECS: 200 is used to enter the matrix elements of A , and the matrix elements of B are entered by ECS: 300. When G0121 is executed, the result will be stored in destination memory location and ECS: 400 command will display the result on the screen as shown above.

7.3.26 Program to Find the Gray Code Equivalent of a Binary Number

The 8-bit binary number is $B_7 B_6 B_5 B_4 B_3 B_2 B_1 B_0$. The Gray code number is $G_7 G_6 G_5 G_4 G_3 G_2 G_1 G_0$. The relationship between Gray code and binary code is $G_7 = B_7$ and $G_i = B_i \oplus B_{i+1}$ where $i = 0$ to 6 . The program flow chart to find the gray code of a binary number is shown in Fig. 7.33. The program for binary to Gray conversion is given below:

```

C:\>DEBUG
-A100
17DA: 0100  MOV  AL, 89      ; Load the binary number in AL
17DA: 0102  MOV  BL, AL       ; Move AL to BL
17DA: 0104  CLC                ; Clear carry
17DA: 0105  RCR  AL, 1      ; Rotate right through carry by one bit
17DA: 0107  XOR  BL, AL       ; XORing the content of BL and AL
17DA: 0109  MOV  DL, BL     ; Store content of BL in DL register
17DA: 010B  HLT
-G010B
AX = 0044 BX = 00CD CX = 0000 DX = 00CD SP = FFEE BP = 0000
SI = 0000 DI = 0000
DS = 17DA ES = 17DA SS = 17DA CS = 17DA IP = 010B NV UP EI
NG NZ NA PO NC
17DA: 010B F4 HLT
    
```

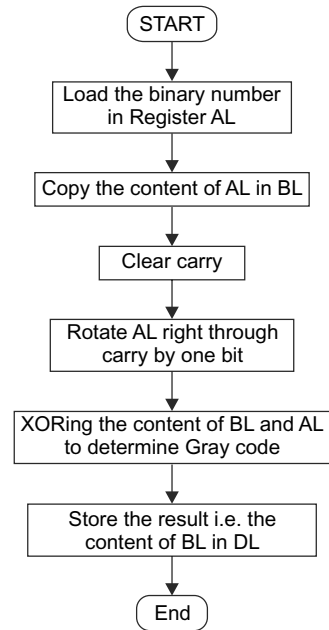


Fig. 7.33 Flow chart to find the Gray code of a binary number

The above program can be used to convert the binary number 89H into equivalent Gray code. When the program is executed by the G010B command, the equivalent Gray code of the binary number 89H will be stored in the DL register, i.e., CD.

7.3.27 Program to Convert a BCD Number to Equivalent Binary Number

The program for converting a BCD number into its binary equivalent number is given below. Assume the

BCD number is 2345. When the program is executed by the G011E command, the equivalent binary of BCD number 2345 will be stored in the DX register, i.e., 0929H. The program flow chart to convert BCD number to binary number is shown in Fig. 7.34.

C:\>DEBUG

-A100

17DA: 0100 MOV BX, 2345 ; The BCD number is stored in BX

17DA: 0103 MOV CX, 0000 ; Initialize CX as 0000H

17DA: 0106 CMP BX, + 00 ; Compare BX with 0000H

17DA: 0109 JZ 011C ; Jump zero to 011C

17DA: 010B MOV AL, BL ; Copy the content of BL to AL

17DA: 010D SUB AL, 01 ; Subtract 01 from AL

17DA: 010F DAS ; Decimal adjustment after subtraction

17DA: 0110 MOV BL, AL ; Copy the content of AL to BL

17DA: 0112 MOV AL, BH ; Copy the content of BH to AL

17DA: 0114 SBB AL, 00 ; Subtract 00 from AL with borrow

17DA: 0116 DAS ; Decimal adjustment after subtraction

17DA: 0117 MOV BH, AL ; Copy the content of AL to BH

17DA: 0119 INC CX ; Increment CX

17DA: 011A JMP 0106 ; Jump to 0106

17DA: 011C MOV DX, CX ; Store the Binary number in DX register

17DA: 011E HLT

-G11E

AX = 0000 BX = 0000 CX = 0929 DX = 0929 SP = FFEE BP = 0000 SI = 0000 DI = 0000

DS = 17DA ES = 17DA SS = 17DA CS = 17DA IP = 011E NV UP EI PL ZR NA PE NC

17DA: 011E F4 HLT

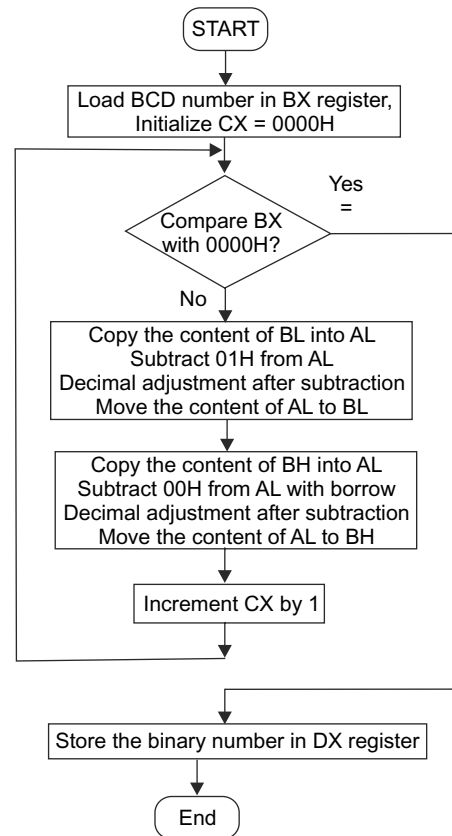


Fig. 7.34 Flow chart to convert a BCD number to a binary number

7.3.28 Program to Find Factorial of a BCD Number

The program to determine the factorial of a one-digit BCD number is developed based on the algorithm as given below:

Algorithm

1. If $N = 1$, then factorial = 1
2. If $N > 1$, the factorial = $N \times$ (factorial of $N - 1$)

The program flow chart to find factorial of a BCD number is depicted in Fig. 7.35.

```

C:\>DEBUG
-A0100
17DA: 0100  MOV  CX, 0005; Store number in the Register CX
17DA: 0103  MOV  AX, CX ; Copy the content of CX in Register
                AX
17DA: 0105  DEC  CX    ; Decrement CX
17DA: 0106  MUL  CX    ; Multiply CX with AX
17DA: 0108  DEC  CX    ; Decrement CX
17DA: 0109  JNZ  0106 ; Jump not zero to 0106
17DA: 010B  MOV  BX, AX ; Store factorial value in Register BX
17DA: 010D  HLT
-G010D
AX = 0078 BX = 0078 CX = 0000 DX = 0000 SP = FFEE BP = 0000
SI = 0000 DI = 0000
DS = 17DA ES = 17DA SS = 17DA CS = 17DA IP = 010D NV UP EI
PL ZR NA PE NC
17DA: 010D F4  HLT

```

The above program can be used to determine factorial 5 or 5!. After editing the program, G010D is used to execute. Then factorial 5 will be stored in BX register, i.e., 78H.

7.3.29 Program to Find Out the Number of Positive Numbers and Negative Numbers in a Series of Signed Numbers

The algorithm to find the number of positive and negative numbers in a series of signed numbers is illustrated below:

Algorithm

1. Load the number in the AX register from memory location.
2. Rotate number left through carry. Carry flag represents the most significant bit of the number.
3. If the carry flag = 1, the number is negative.
4. If the carry flag = 0, the number is positive.

The program flow chart to find out the number of positive numbers and negative numbers in a series of signed numbers is shown in Fig. 7.36.

```

C:\>DEBUG
-A100
17DA: 0100  MOV  SI, 0200 ; SI is loaded 0200 as source address of data
17DA: 0103  MOV  BX, 0000 ; Initialize BX register with 0000H
17DA: 0106  MOV  DX, 0000 ; Initialize DX register with 0000H
17DA: 0109  MOV  CL, 05 ; Load number of data in CL register
17DA: 010B  MOV  AX, [SI] ; Move data in to AX register from memory
17DA: 010D  SHL  AX, 1 ; Shift left through carry
17DA: 010F  JB  0114 ; Jump carry to 0114
17DA: 0111  INC  BX ; Else increment BX register
17DA: 0112  JMP  0115 ; Jump to 0115
17DA: 0114  INC  DX ; Increment DX register

```

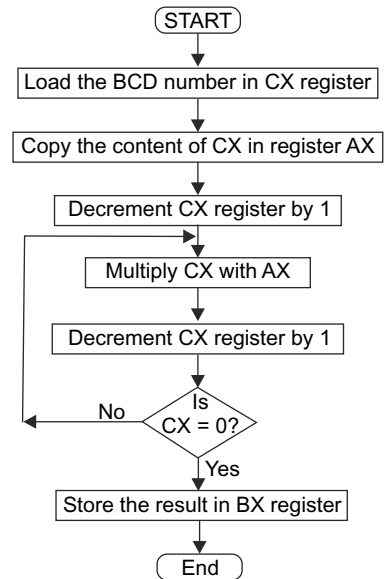


Fig. 7.35 Flow chart to find factorial of a BCD number

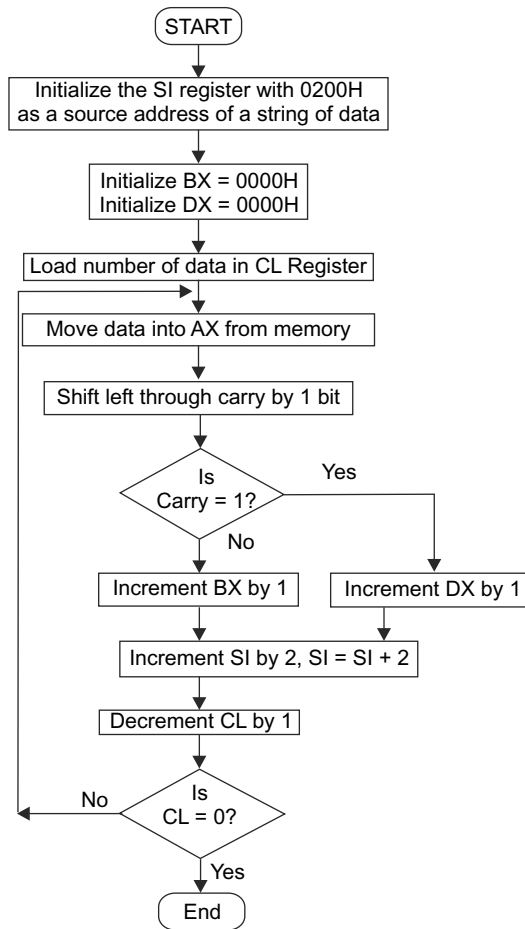


Fig. 7.36 Flow chart to find out the number of positive numbers and negative numbers in a series of signed numbers

```

17DA:0115  ADD    SI,+02    ; Add 02 with SI to locate next data in memory
17DA:0118  DEC    CL        ; Decrement CL
17DA:011A  JNZ    010B      ; Jump not zero 010B
17DA:011C  HLT
    
```

-ECS:200

```

17DA:0200  00.23  00.67  00.01  00.24  00.90  00.90  00.44  00.98
    
```

```

17DA:0208  00.26  00.98
    
```

-G11C

```

AX = 304C BX = 0002 CX = 0000 DX = 0003 SP = FFEE BP = 0000 SI = 020A DI = 0000
    
```

```

DS = 17DA ES = 17DA SS = 17DA CS = 17DA IP = 011C NV UP EI PL ZR NA PE NC
    
```

```

17DA:011C  F4 HLT
    
```

-

The above program is used to determine the number of positive as well as negative numbers in a series of signed numbers. The ECS:200 command is used to enter five signed numbers 6723, 2401, 9090, 9844 and

9826. After execution of the program through G11C command, number of positive numbers will be stored in CX register i.e., 2 and number of negative numbers will be stored in DX register i.e., 3.

7.3.30 Program to Find out the Number of Even Numbers and Odd Numbers in a Series of Signed Numbers

The algorithm to find the number of even and odd numbers in a series of signed numbers is illustrated below:

Algorithm

1. Load the number in AX register from memory location
2. Rotate number right through carry. Carry flag represents whether the number is even or odd. Actually the carry flag is least significant bit of number.
3. If the carry flag = 1, the number is odd.
4. If the carry flag = 0, the number is even.

The program flow chart to find out the number of even numbers and odd numbers in a series of signed numbers is given in Fig. 7.37.

C:\>DEBUG

-A100

17DA:0100 MOV SI,0200 ; SI is loaded 0200 as source address of data

17DA:0103 MOV BX,0000 ; Initialize BX register with 0000H

17DA:0106 MOV DX,0000 ; Initialize DX register with 0000H

17DA:0109 MOV CL,05 ; Load number of data in CL register

17DA:010B MOV AX,[SI] ; Move data in to AX register from memory

17DA:010D ROR AX,1 ; Rotate right through carry

17DA:010F JC 0114 ; Jump carry to 0114

17DA:0111 INC BX ; Else increment BX register

17DA:0112 JMP 0115 ; Jump to 0115

17DA:0114 INC DX ; Increment DX register

17DA:0115 ADD SI,2 ; Add 02 with SI to locate next data in memory

17DA:0118 DEC CL ; Decrement CL

17DA:011A JNZ 010B ; Jump not zero to 010B

17DA:011C HLT

-ECS:200

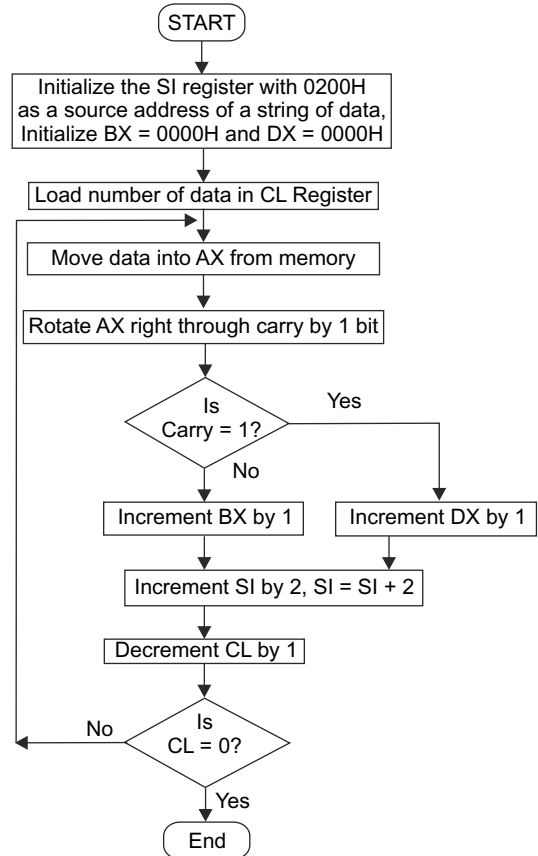


Fig. 7.37 Flow chart to find out the number of even numbers and odd numbers in a series of signed numbers

```

17DA:0200 00.01 00.23 00.00 00.45 00.41 00.34 00.61 00.92
-ECS:208
17DA:0208 00.00 00.89
-G11C
AX = 4480 BX = 0002 CX = 0000 DX = 0003 SP = FFEE BP = 0000 SI = 020A DI = 0000
DS = 17DA ES = 17DA SS = 17DA CS = 17DA IP = 011C NV UP EI PL ZR NA PE NC
17DA:011C F4 HLT
-
    
```

The above program can be used to determine the number of even as well as odd numbers in a series of signed numbers. The ECS:200 command is used to enter five signed numbers 2301, 4500, 3441, 9261 and 8900. When the program is executed by the G11C command, number of even numbers will be stored in BX register i.e., 2 and number of odd numbers will be stored in DX register i.e., 3.

7.3.31 Program to Find out Square Root of a Number using Look-up Table

Load the decimal number in the accumulator. Find the square root of decimal number and store it in the memory location 17B3:0400. The square values of decimal numbers 0, 1, 4, 9, and 16 are stored in 17B3:0300, 03001, 0304, 0309, and 0316 and used as Look-up table. The program flow chart to find out square root of a number using look-up table is shown in Fig. 7.38.

Algorithm

1. Store the decimal number in Accumulator.
2. Load 03H in AH register.
3. If the decimal number is 09, the content of AH and AL registers are 03 and 09H respectively. Then the offset address of memory location will be 0309H denoted by AX register. Move AX content into SI register.
4. Move square root of decimal number in AL from the memory location 17B3:0309.
5. Store the result, square value in 17B3:0400.

```

C:\>DEBUG
-A0100
17B3:0100 MOV AL,09
17B3:0102 MOV AH,03
17B3:0104 MOV SI,AX
17B3:0106 MOV AL,[SI]
17B3:0108 MOV DI,0400
17B3:010B MOV [DI],AL
17B3:010D HLT
-ECS:0300
17B3:0300 00. 00.01
-ECS:0304
17B3:0304 00.02
-ECS:0309
    
```

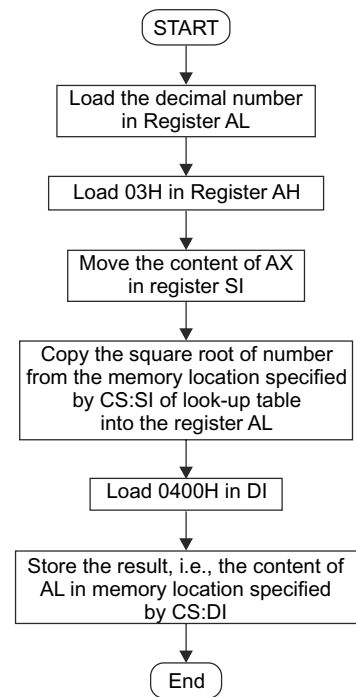


Fig. 7.38 Flow chart to find square root of a number using look-up table


```

17B3:0309 00.03
-ECS:0316
17B3:0316 00.04
-G010D
AX=0303 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0309 DI=0400
DS=17B3 ES=17B3 SS=17B3 CS=17B3 IP=010D NV UP EI PL NZ NA PO NC
17B3:010D F4 HLT
-ECS:0400
17B3:0400 03.
    
```

The program to find out the square root of a decimal number is stored into memory locations 17B3:0100 to 17B3:010D. The U 0100 010D command is used to display the object codes of the above program. When G010D is used to execute the program and the square root value of 09 is 03 will be stored in memory location 17B3:0400.

7.3.32 Program to Convert Binary Number to Equivalent BCD Number

The program for converting a binary number into an equivalent BCD number is given below. Assume the binary number is 00FFH. When the program is executed by G011F command, the equivalent BCD of binary number 00FFH will be stored in DX register, i.e., 0255. The program flow chart to convert binary number to equivalent BCD number is given in Fig. 7.39.

C:\>DEBUG

```

-A100
17DA:0100 MOV BX,00FF ; The Binary number is
                    ; stored in BX
17DA:0103 MOV AX,0000 ; Initialize AX as 0000H
17DA:0106 MOV CX,0000 ; Initialize CX as 0000H
17DA:0109 CMP BX,+00 ; Compare BX with 0000H
17DA:010C JZ 011F ; Jump zero to 011C
17DA:010E DEC BX ; Decrement BX
17DA:010F MOV AL,CL ; Copy the content of CL to
                    ; AL
17DA:0111 ADD AL,01 ; Add 01H with AL
17DA:0113 DAA ; Decimal adjustment after
                    ; addition
17DA:0114 MOV CL,AL ; Copy the content of AL to
                    ; CL
17DA:0116 MOV AL,CH ; Copy the content of CH to
                    ; AL
17DA:0118 ADC AL,00 ; Add 00H AL with carry
17DA:011A DAA ; Decimal adjustment after
                    ; addition
    
```

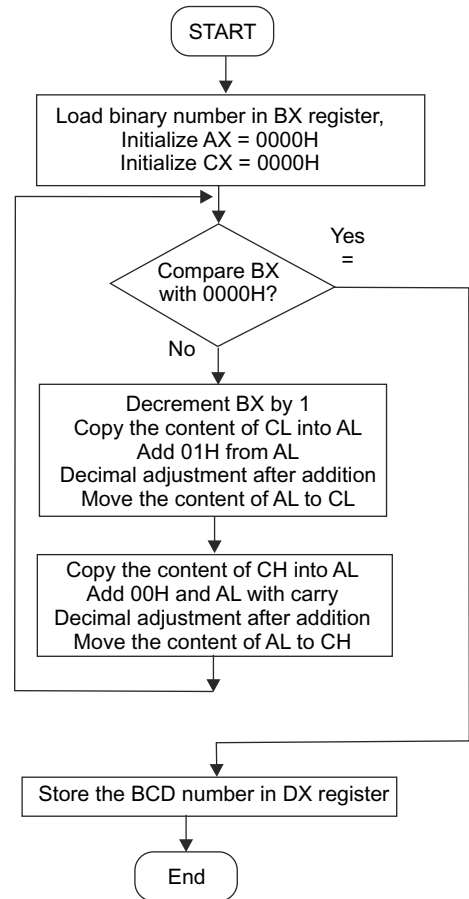


Fig. 7.39 Flow chart to convert binary number to BCD number

```

17DA:011B MOV  CH,AL      ; Copy the content of AL to CH
17DA:011D JMP  0109      ; Jump to 0109
17DA:011F MOV  DX,CX     ; Store BCD number in DX register
17DA:0121 HLT
-G0121
AX=0002 BX=0000 CX=0255 DX=0255 SP=FFEE BP=0000 SI=0000 DI=0000
DS=17DA ES=17DA SS=17DA CS=17DA IP=0121 NV UP EI PL ZR NA PE NC
17DA:0121 F4          HLT
    
```

7.4 8087, 80287 AND 80387 NUMERIC DATA PROCESSORS

The general-purpose microprocessors have an upper limit of data-processing capability and these general processors require complex programming to perform any mathematical calculations. For any mathematical operations, processors use high-level language for programming and a library of floating point objects has to be obtained from the manufacturer.

To increase the operating speed, several microprocessors are connected together using a certain network topology. Then the system is called a *multiprocessor system*. The simplest multiprocessors system consists of a microprocessor and a Numeric Data Processor (NDP). The numeric data processor has an independent math processing unit and it can do complex numeric calculations very fast compared to the main processor. The most commonly used numeric processors are 8087, 80287 and 80387. The 8087 was first released in 1980 by Intel and this can work with 8086, 8088, 80186 and 80188 processors. After introducing 80286 CPU, Intel developed a redesigned 80287 NDP to operate with 80286 and 80386 processors. To improve the performance of 80386, the 80387 NDP was developed. The Intel 80486 and Pentium chips provide high performance as a numeric data processor is directly in built in the CPU. In this section, the block diagram, operating principle, instruction set and programming of 8087, 80287 and 80387 numeric data processors are discussed elaborately.

7.5 8087 NUMERIC DATA PROCESSOR

The 8087 Numeric Data Processor (NDP) is called a high-speed math co-processor. This math co-processor is also known as Numeric Processor Extension (NPX) or Numeric Data Processor (NDP) or Floating Unit Point (FUP). The 8087 is available in 40-pin DIP packages in 5 MHz, 8 MHz, and 10 MHz versions and it is compatible with 8086 and higher-version processors.

7.5.1 Block Diagram of 8087

Figure 7.40 shows the block diagram of the internal architecture of 8087. The 8087 co-processor consists of

- ✦ Control Unit (CU) and
- ✦ Numeric Execution Unit (EU).

Control Unit (CU)

This unit is used to synchronize the operation between the main processor and co-processor. The control unit receives the instruction opcode, and then it decodes the instructional opcode and reads or writes operands from memory. The control unit provides the communication between the processor and memory, and it also coordinates the internal coprocessor execution. This unit continuously monitors the data bus to find instructions for the 8087 co-processor.

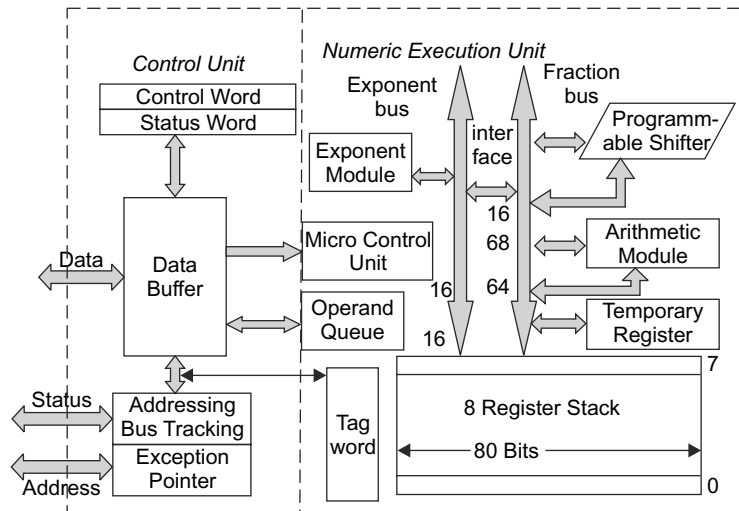


Fig. 7.40 Block diagram of 8087 co-processor

The control unit of 8087 maintains a parallel queue just like the queue of the main processors. The control unit also always monitors the \overline{BHE}/S_7 line to detect the processor type that is either 8086 or 8088. According to the main processor, the queue length will be adjusted. For the 8086 processor, the queue length will be 6 bytes, but for 8088 processor, the queue length will be 4 bytes. The 8087 uses the queue status input pins QS_0 and QS_1 to identify the instructions fetched by the main processor. All instruction codes of 8087 have 11011 as the most significant bits of their first code byte. Actually, the main processor identifies the co-processor instruction using the ESCAPE code. Once the main processor recognizes the ESCAPE code, it sends a trigger signal so that the execution of the numeric processor instruction starts in 8087.

During execution, the ESCAPE code identifies the co-processor instruction which can be operated with or without a memory operand. If the co-processor instruction requires a memory operand that will be fetched from memory then the physical address of the memory location will be computed using any one of the addressing modes in 8086 and a dummy read cycle must be initiated by the main processor. Thereafter, the 8087 co-processor reads the operand and proceeds for execution. If the co-processor instruction does not require any operand then the instruction can be directly executed. After execution of the instruction, the 8087 is ready with the execution results and the control unit obtains the control of the bus from 8086 and executes a memory write cycle to write the results at the specified memory location. The control unit consists of a control word, a status word and a data buffer which are explained later.

Numeric Execution Unit (NEU)

The Control Unit (CU) consists of a data bus buffer, status and control register and the Numeric Execution Unit (EU) has eight data register stacks, microcode control unit and a programmable shifter. These units duplicate the functions performed by the microprocessor control and ALU blocks. The 8087 NEU and CU can work independently. The CU works to maintain synchronization with the main 8086/8088 processor while the NEU is performing numeric operations.

The numeric extension unit performs all operations that access and manipulate the numeric data in the co-processor's registers. In NEU, the numeric registers are 80 bits wide and the numeric data is routed by a 64-bit mantissa bus and a 16-bit sign/exponent bus. The Numeric Execution Unit (NEU) executes all numeric processor instructions such as arithmetic, logical, transcendental and data-transfer instructions. The operation

of CU and NEU is asynchronous with each other. The internal data bus is 84 bits wide which consist of 68-bit fraction, 15-bit exponent and a sign bit.

When the NEU starts execution of an instruction, it always pulls up the BUSY signal. Usually the BUSY signal is connected with the \overline{TEST} input signal of the 8086 processor. When the BUSY signal of 8087 is verified by the main processor, the CPU is able to distinguish that the instruction execution is not yet completed. Therefore, 8086 must be waiting till the BUSY pin of 8087 or the \overline{TEST} input pin of 8086 becomes low. The microcode control unit generates the control signals which are required for execution of the instructions. The programmable shifter is used for shifting the operands during the execution of instructions. The data bus interface is able to connect the internal data bus of 8087 with the main processor data bus.

Registers of 8087 Co-processor

The 8087 co-processor has additional 13 registers to the 8086 and higher processors such as eight 80-bit floating point data registers, a control register, a status register, a tag register, an instruction pointer and a data pointer register. Figure 7.41 shows the registers of 8087 numeric processor and 8086/8088 main processor.

Floating Point Data Registers

The 8087 has eight 80-bit individually addressable data registers organized as stack registers R_1 to R_8 as shown in Fig. 7.41. The mathematical operations are performed in these registers. 8087 instructions can access these registers independently.

The data registers are theoretically divided into three fields such as sign (1-bit), exponent (15-bits) and significant (64-bits). The actual use of these fields varies with the type of data being operated with the instruction. When the 8087 receives numeric data words, it stores and holds them in a format called *temporary real form*. This number is expressed as the product of a 64-bit significant base and a 15-bit exponent and the Most Significant Bit (MSB) of the register is reversed as a sign bit to represent either a positive or a negative number. The 8087 instructions automatically convert data into this format when loading the registers and return back to the other format when returning them to the system memory.

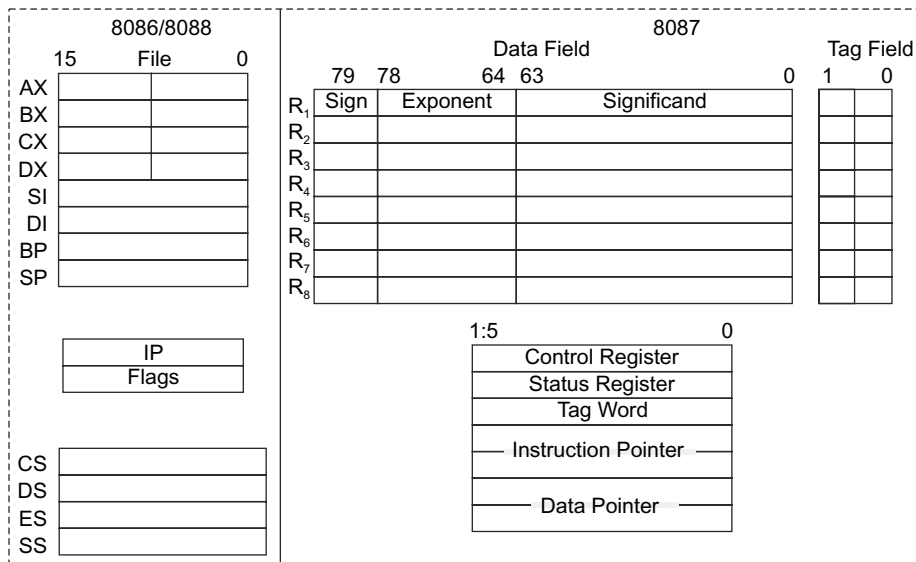


Fig. 7.41 Registers of 8087 numeric processor and 8086/8088 main processor

These registers can be represented as ST (0), ST (1), ST(2), ST(3) ST(4), ST(5) ST(6), and ST(7). When the 8087 co-processor is reset, ST (0) becomes the top of the stack and ST(1) refers to the next register in the stack and other registers will be referred accordingly as shown in Fig. 7.42 (a). After the first push operation, the register 000 becomes ST(1) and the register 111 refers ST(0). Similarly, other registers are referred as shown in Fig. 7.42 (b). During programming, any register can be used as the top of the stack and other registers will be referred according to their position.

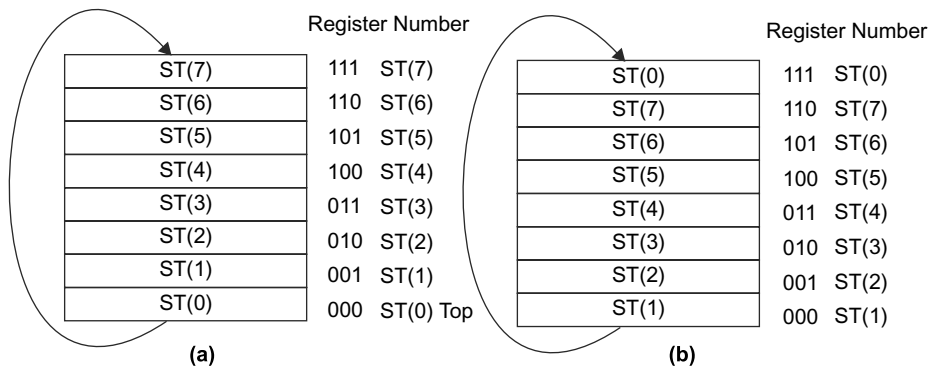
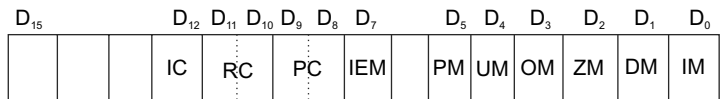


Fig. 7.42 (a) Registers after reset (b) Register after first push operation

Control Word Register

The 16-bit control word register is used to control the operation of the 8087. The control word register bits can select the required data processing options such as precision control, rounding control and infinity control. Figure 7.43 shows the format of the control word register. The bits D_5 – D_0 are used for masking the different exceptions. An exception may be masked by setting the respective bit in the control word register. The IEM bit is used as a common interrupt mask for all the interrupts. When IEM is set, all the exceptions generated will be masked and the execution may continue.



- IC — Infinity control
- RC — Rounding control
- PC — Precision control
- IEM — Interrupt enable mask
- PM — Precision mask
- UM — Underflow mask
- OM — Overflow mask
- ZM — Division by zero mask
- DM — Denormalized operand mask
- IM — Invalid operand mask

Fig. 7.43 Control word register

✓ **IC** Infinity control selects either affine or protective infinity. Affine allows positive and negative infinity, while protective assumes infinity is unsigned. Hence the Infinity Control (IC) provides control over the number size on both sides, i.e., either $+\infty$ or $-\infty$.

Infinity Control	
0 = Protective	1 = Affine

✓ **RC** Rounding control determines the type of rounding as given below.

Rounding Control	
00 =	Round to nearest or even
01 =	Round down towards minus infinity
10 =	Round up towards plus infinity
11 =	Chop or truncate towards zero

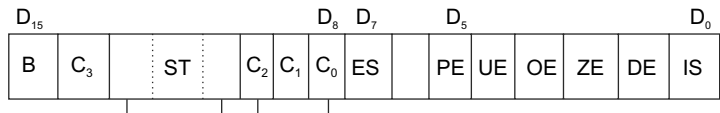
✓ **PC** Precision control bits control the precision of the result as given in Table 7.2.

Table 7.2

<i>Precision Control</i>	
00 =	Single precision (short)
01 =	Reserved
10 =	Double precision (long)
11 =	Extended precision (temporary)

Status Register

The operation of the 8087's status register is similar to the 8088's flag register. The co-processor status register contains the conditional code bits and the floating point flags as shown in Fig. 7.44. During executing an instruction, the 8087 co-processor generates six different exceptions. These are reflected in the status register format. Whenever any exception is generated, an interrupt take place to the CPU provided it is not masked. Then the 8086 processor will respond if the interrupt flag of the CPU is set. When the exceptions are masked, the 8087 continues the execution. Therefore, the status register reflects the over all operations of the 8087 co-processor.



- B — Busy bit
- C₃–C₀ — Condition code bits
- ST — Top-of-stack (ST)
- ES — Error summary
- PE — Precision error
- UM — Underflow error
- OM — Overflow error
- ZM — Zero error
- DE — Denormalized error
- IE — Invalid error

Fig. 7.44 Status register

✓ **B** Busy bit (B) indicates that 8087 co-processor is busy executing an instruction. Busy can be tested by examining the status or by using the FWAIT instruction.

✓ **C₃–C₀** Condition code bits indicates conditions about the co-processor as given Table 7.3(a) and Table 7.3.(b).

Table 7.3(a)

C ₃	C ₂	C ₁	C ₀	Result
0	0	x	0	ST > Source
0	0	x	1	ST < Source
1	0	x	0	ST = Source
1	1	x	1	ST is not comparable

Table 7.3(b)

C ₃	C ₂	C ₁	C ₀	Result
0	0	0	0	+Un-normal
0	0	0	1	+NAN
0	0	1	0	–Un-normal
0	0	1	1	–NAN
0	1	0	0	Normal
0	1	0	1	+∞
0	1	1	0	– Normal
0	1	1	1	–∞

(Contd.)

(Contd.)

1	0	0	0	+0
1	0	0	1	Empty
1	0	1	0	-0
1	0	1	1	Empty
1	1	0	0	+De-normal
1	1	0	1	Empty
1	1	1	0	-De-normal
1	1	1	1	Empty

- ✓ **ST** Top of the stack (ST) bit shows the current register address as the top of the stack.
- ✓ **ES** Error summary bit is set if any unmasked error bit (PE, UE, OE, ZE, DE, or IE) is set. In the 8087 co-processor, the error summary is also caused a co-processor interrupt.
- ✓ **PE** Precision error indicates that the result or operand executes selected precision.
- ✓ **UE** Under flow error indicates that the result is too small to fit in the specified format 8087 generates these exceptions. When this exception is masked, the 8087 denormalizes the fraction until the exponent fits in the specified destination format.
- ✓ **OE** Over flow error indicates that result is too large to represent in the specified format. If this error is masked, the 8087 co-processor generates infinity for an overflow error.
- ✓ **ZE** The zero error indicates if any non-zero finite operand is divided by zero. In this case the divisor is zero while the divided is a non-infinity or non-zero number. The zero error bit indicates that the result is infinity, even if the exception is masked.
- ✓ **DE** The denormalized error indicates, if at least one of the operand is denormalized. This error may be generated, if the result is denormalized. When this bit is masked, 8087 continues the exception normally.
- ✓ **IE** Invalid error shows a stack overflow or a stack underflow and uses "NAN" as operand. For example, if we find the square root of a negative number, this flag error will be generated.

TAG Register

The tag register contains several groups of 2 bits that can determine the state of the value of the eight 80-bit stack registers as valid, zero, special or empty. Tag register is a two-bit register called TAG field as shown in Fig. 7.45. The tag word register presents the entire TAG field to the CPU. The tag values are 00 = valid, 01 = zero, 10 = special and 11 = empty.

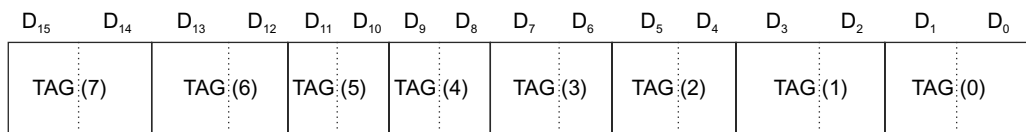


Fig. 7.45 TAG word

Instruction and Data Pointer Registers

The instruction and data pointer registers are used to hold the information about the last executed floating point instruction. Actually, the information is address of instruction, op-code and operand address. Prior to execution of a mathematical instruction, the 8087 forms a table in the memory. The table contains the instruction address in the fields of the instruction pointers, the opcode of the instruction, and operand address in the field of data pointers. The TAG word, status word, and control word will also be present in their respective fields as shown in Table 7.4. Therefore,

Table 7.4 Respective fields of TAG, control and status words

Control Word			← (DST + 0)
Status Word			← (DST + 2)
TAG Word			← (DST + 4)
Instruction Pointer (15–0)			← (DST + 6)
Instruction Pointer 19–16	0	Instruction Opcode (10–0)	← (DST + 8)
Data Pointer (15–0)			← (DST + 10)
Data Pointer 19–16			← (DST + 12)

the instruction pointer and the data pointer registers contain the address of the currently executed instruction and the corresponding data.

7.5.2 PIN Descriptions of 8087

Figure 7.46 shows the pin diagram of 8087 processor. The operations of all pins of 8087 are explained below:

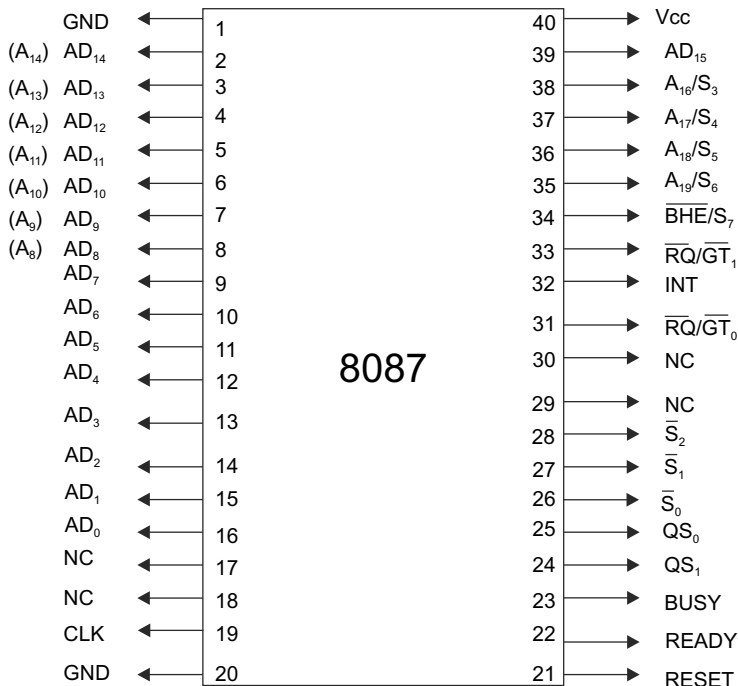


Fig. 7.46 Pin diagram of 8087 processor

$AD_{15}-AD_0$ The $AD_{15}-AD_0$ are the time multiplexed address/data lines. During T_1 , these lines are used as address bus $A_{15}-A_0$ and these lines can be used as data bus $D_{15}-D_0$ during T_2, T_3, T_w and T_4 states. A_0 is also used as the chip select signal whenever the data transfer is on lower byte (D_7-D_0) of data bus.

$A_{19}/S_6 - A_{16}/S_3$ These lines are the time-multiplexed address/status lines and their function are same as the corresponding pins of 8086. The S_6, S_4 and S_3 are high, when the S_5 is low.

\overline{BHE}/S_7 The \overline{BHE}/S_7 pin is used to select data on the higher byte of the 8086 data bus during T_1 . During T_2, T_3, T_w and T_4 , this signal is a status line S_7 .

QS_1-QS_0 The queue status input signals, QS_1 and QS_0 are used to allow the co-processor to track the progress of an instruction through the 8086 queue and help 8087 co-processor to determine when to access the bus for the escape opcode and operand. These signals can also maintain synchronism with main CPU and indicate the status of the internal instruction queue as given in the Table 7.5.

Table 7.5 Status signals of QS_1-QS_0

QS_1	QS_0	Queue Status
0	0	Queue is idle
0	1	First byte of op-ode from queue
1	0	Queue is empty
1	1	Subsequent byte of op-ode from queue

INT The interrupt output signal INT is used by 8087 to indicate an unmasked exception that has been received during execution. Usually this signal is handled by 8259A programmable interrupt controller.

BUSY The BUSY output signal is used to indicate to the CPU that 8087 co-processor is busy with the execution of an allotted instruction.

READY The READY input signal can be used to indicate the 8087 co-processor that the addressed device has completed the data transfer and the bus becomes free for the next bus cycle. Usually, this signal is synchronized by the clock generator 8284.

RESET This input signal is used to rest the co-processor after escaping the all internal activities and is ready for execution of any instruction send by the main processor.

S_2, S_1 and S_0 These lines are bus status output signals that encode the type of the current bus cycle as given in Table 7.6.

Table 7.6 Bus status output signals

S_2	S_1	S_0	Queue Function
0	0	0	Interrupt acknowledge
0	0	1	I/O read
0	1	0	I/O write
0	1	1	Halt
1	0	0	Opcode fetch
1	0	1	Memory read
1	1	0	Memory write
1	1	1	Passive

RQ/GT₀ The bus request/grant output signal RQ/GT₀ is used by the 8087 to control of the bus from the host 8086/8088 for operand transfers. This pin must be connected to the request/grant pin of the host processor.

RQ/GT₁ The RQ/GT₁ bidirectional pin is used by the other bus masters like DMA controllers to convey their need of the local bus access to 8087. This request must be further conveyed to the host processor.

CLK The input clock signal CLK provides the basic timings for the co-processor operation.

V_{CC} This is a +5 V supply line which is used for the circuit operation.

GND This pin is connected with the ground terminal of the power supply.

7.5.3 Interfacing The 8087 With 8086 Microprocessor

The physical connection between the 8087 co-processor and 8086 main processor is shown in Fig. 7.47. The 8087 can be connected with the main CPU only in their maximum mode of operation of processor when the MN/\overline{MX} pin of the processor is grounded. In maximum mode operation, all the control signals are generated by an 8288 bus controller. Multiplexed address-data bus lines, AD₁₅–AD₀ are connected directly from 8086 to 8087. The queue status QS₀ and QS₁ lines may be directly connected to the corresponding pins in case of 8086-based systems. The request/grant signal RQ/GT₀ of 8087 is connected to RQ/GT₁ of 8086. The BUSY pin of 8087 is connected with the TEST pin of the host 8086 CPU. The clock pin of the 8087 may be connected with the CPU 8086 clock input. The interrupt output of 8087 is routed to 8086/8088 via a programmable interrupt controller.

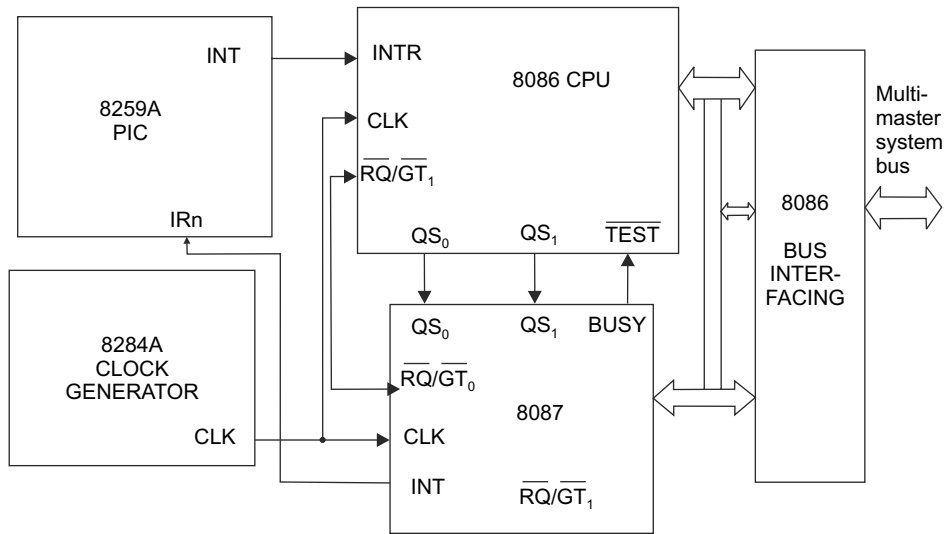


Fig. 7.47 Interconnection of 8087 and 8086

7.5.4 Data Formats for the 8087 Co-processor

The 8087 co-processor can support 7 different types of data such as

- ✦ **Three Signed Integers**
 - 16-bit (word) – range 32768 to + 32767

- 32-bit (short integer) range $-2 \times 10^{+9}$ to $+2 \times 10^{+9}$
- 64-bit (long integer) range $-9 \times 10^{+18}$ to $+9 \times 10^{+18}$
- ✦ **A18-digit BCD data and**
- ✦ **Three floating point type numbers**
 - 32-bit (extended precision) floating-point numbers
 - 64-bit (extended precision) floating-point numbers
 - 80-bit (extended precision) floating-point numbers

Figure 7.48 shows three signed integers supported by 8087

The floating-point unit hold signed integers, fractions and mixed numbers. The floating-point numbers has 3 parts such as sign bit, biased exponent and significand. The 8087 co-processor can support 3 types of floating point numbers such as

- Short (32 bits) : single precision, with a bias of 7FH
- Long (64 bits) : double precision, with a bias of 3FFH
- Temporary (80 bits) : extended precision, with a bias of 3FFFH and Fig. 7.49 shows the floating point number supported by 8087

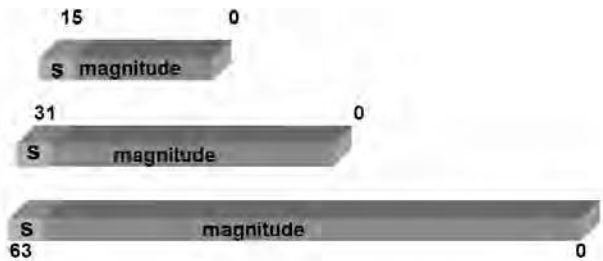


Fig. 7.48 Signed integer supported by 8087

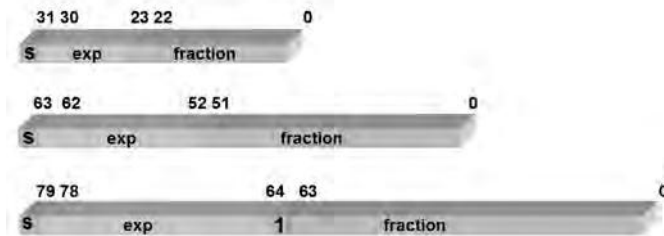


Fig. 7.49 Floating-point number supported by 8087

7.5.5 Instruction Set of 8087

The 8087 co-processor has 68 additional instructions to the instruction set of 8086. These instructions are fetched by 8086 but are executed by 8087. When the 8086 comes across an 8087 instruction, it executes the ESCAPE instruction code to bypass the instruction opcode and control of the local bus to 8087 co-processor. The execution of 8087 instructions is transparent to the programmer. The 8087 co-processor instructions are divided into six different groups as given below:

1. **Data Transfer Instructions**
 - Real Transfers (Example : FLD)
 - Integer Transfers (Example : FILD)
 - Packed Decimal Transfers (Example : FBLD, FBSTP)
2. **Arithmetic Instructions**
 - Addition (Example : FADD, FADDP)

- Subtraction (Example : FSUB, FSUBP)
 - Reversed Subtraction (Example : FSUBR, FSUBRP)
 - Multiplication (Example : FMUL, FMULP)
 - Division (Example : FDIV, FDIVP)
 - Reversed Division (Example : FDIVR, FDIVRP)
3. **Other Arithmetic Operations** (Example : FSQRT, FABS)
 4. **Compare Instructions** (Example : FCOM, FCOMP, FCOMPP)
 5. **Transcendental (Trigonometric and Exponential) Instructions** (Example : FPTAN, FPATAN)
 6. **Processor Control Instructions** (Example : FLD)

Some of the most commonly used co-processor instructions are discussed in this section.

Floating-Point Data Transfer Instructions

✓ **FLD (Load Real)** Loads 32-bit, 64-bit or 80-bit floating-point data to Top of Stack (ST). Stack pointer is then decremented by 1. Data can be retrieved from memory, or another stack register. The examples of FLD instructions are as follows:

FLD ST(2); Top of stack \leftarrow [ST(2)]; Copies the contents of register ST(2) to top of the stack.

FLD Memory_32; Top of stack \leftarrow [Memory_32]; Copies the contents of memory_32 to top of the stack.

✓ **FST (Store Real)** This instruction stores the content of the top of the stack into memory or a specified co-processor register. During copy, the data rounding occurs using the rounding control bits in floating point control register. The examples of FST instructions are as follows:

FST ST(1) ; [ST(1)] \leftarrow Top of stack; Copies the contents of top of the stack to register ST(1).

FST Memory_64; [Memory_64] \leftarrow Top of stack; Copies the contents of top of the stack to memory_64.

✓ **FSTP (Store Floating Point Number and Pop)** This instruction stores a copy of the top of the stack into memory or any specified co-processor register and pop the data from the top of stack. The examples of FSTP instructions are as follows

FSTP ST(4) ; [ST(4)] \leftarrow Top of stack, Copies the contents of top of the stack to register ST(4).

FSTP Memory_32; [Memory_32] \leftarrow Top of stack; Copies the contents of top of the stack to memory_32.

✓ **FXCH (Exchange)** The FXCH instruction exchanges the contents of specified register with top of stack. The example of FXCH instruction is

FXCH ST(2) ; [ST(2)] \leftrightarrow Top of stack; Exchanges the contents of top of the stack with register ST(2).

Integer Data-Transfer Instructions

During transferring the data, the co-processor automatically converts extended floating-point number to integer data. There are three types integer data transfer instructions such as

- ✦ FILD (Load integer)
- ✦ FIST (Store integer)
- ✦ FISTP (Store integer and pop)

For example,

FILD ST(3) ; Top of stack \leftarrow [ST(3)]; Copies the contents of register ST(3) to top of the stack.

FILD Memory ; Top of stack \leftarrow [Memory]; Copies the contents of memory to top of the stack.

The instructions FIST and FISTP work in an exactly similar manner as FST/FSTP except that the operands are integer operands. For example,

FIST ST(5) ; ST(5) ← Top of stack; Copies the contents of top of the stack to register ST(5).
 FST Memory_64; [Memory_64] ← Top of stack; Copies the contents of top of the stack to memory_64.

BCD Data-Transfer Instructions

The 8087 has two BCD data transfer instructions such as

- ✦ FBLD (loads the top of stack with BCD memory data), and
- ✦ FBSTP (stores top of the stack and does a pop).

Both the instructions work in an exactly similar manner as FLD and FSTP except that the operands are BCD numbers.

Arithmetic Instructions

The 8087 co-processor has 11 arithmetic instructions that can be used to perform arithmetic operations such as addition, subtraction, multiplication and division or supporting operations like scaling, rounding, negation, absolute value, etc. Some arithmetic instructions are explained below:

✓ **FADD and FADDP** These two instructions perform real or integer addition of the specified operand with the stack top. After addition, the results are stored in the destination operand. The operand may be any of the stack registers or a memory location. For example,

FADD ST(0), ST(1) ; ST(0) ← ST(0) + ST(1), destination = destination + source.

FADDP ST(3), ST(0) ; ST(3) ← ST(3) + ST(0), destination = destination + source.

✓ **FSUB, FSUBP, FSUBR and FSUBRP** These four instructions perform real or integer subtraction of the specified operand from the stack top. The operand may be any of the stack register or memory. After subtraction, the result of the operation is stored in the destination operand. For example,

FSUB ST(0), ST(1) ; ST(0) ← ST(1) - ST(0), destination = destination - source.

FSUBP ST(3), ST(0) ; ST(3) ← ST(0) - ST(3), destination = destination - source.

The FSUBR and FSUBRP instructions work in a similar manner but these instruction perform reverse subtraction such as destination = source - destination. For example,

FSUBR ST(0), ST(1) ; ST(0) ← ST(1) - ST(0), destination = source - destination.

FSUBRP ST(3), ST(0) ; ST(3) ← ST(0) - ST(3), destination = source - destination.

✓ **FMUL and FMULP** These two instructions perform real or integer multiplication of the specified operand with stack top. The specified operand may be a register or a memory location. After multiplication, the result will be stored in the destination operand. For example,

FMUL ST(0), ST(1) ; ST(0) ← ST(0) × ST(1), destination = destination × source.

FMULP ST(3), ST(0) ; ST(3) ← ST(3) × ST(0), destination = destination × source.

✓ **FDIV, FDIVP, FDIVR and FDIVRP** These four instructions perform real or integer division. When the destination is not specified, the top of stack is the destination operand and source operand may be a memory operand of short real or long real type. If both destination and source operands are specified then compute the division and store the result in the destination. For example,

FDIV ST(0), ST(1) ; ST(0) ← ST(0) / ST(1), destination = destination / source.

FDIVP ST(3), ST(0) ; ST(3) ← ST(3) / ST(0), destination = destination / source.

The FDIVR and FDIVRP instructions work in similar way but these instructions perform reverse division such as destination = source / destination. For example,

FDIVR ST(0), ST(1) ; ST(0) ← ST(1) / ST(0), destination = source / destination.

FDIVRP ST(3), ST(0) ; ST(3) ← ST(0) / ST(3), destination = source / destination.

✓ **FSQRT** The FSQRT instruction finds out the square root of the content of the top of stack and stores the result on the stack top. The value of the top of stack must be zero or positive otherwise FSQRT generates an invalid exception.

✓ **FABS** The FABS instruction computes the absolute value of the content of the stack top and the result is stored in the top of stack.

The following letters are used to additionally qualify the arithmetic operations:

- ✦ *P*— Perform a register pop after the operation.
- ✦ *R*— Reverse mode for subtraction and division.
- ✦ *I*— Indicates that the memory operand is an integer. ‘I’ appears as the second letter in the instruction, such as FIADD, FISUB, FIMUL, FIDIV.

Comparison Instruction The comparison instructions of the 8087 co-processor are FCOM, FCOMP, FCOMPP, FUCOM, FUCOMP, and FUCOMPP which are used to compare the two values on the top of stack and set the condition code flags appropriately. The example of comparison instruction is

FCOM ST(1); compare ST(0) against ST(1) and set the processor accordingly

Transcendental Instructions The 8087 co-processor has eight transcendental operation instructions such as FTAN, FPTAN, F2XMI, FLY2X, FLY2XP1, FSIN, FCOS and FSINCOS. In this section only FTAN, FPTAN, FLY2X, and FLY2XP1 are explained.

✓ **FPTAN** The FPTAN instruction is used to compute the partial tangent of an angle θ , where θ must be in the range $0^\circ \leq \theta \leq 90^\circ$. The value of angle must be stored at the stack top. The result is computed in the form of a ratio of ST/ST(1).

✓ **FPATAN** The FPATAN instruction calculates the arc tangent (inverse tangent) of a ratio ST(1)/ST(0). The stack is popped and the result is stored on the top of stack. Its function can be expressed as

$$ST(0) = \tan^{-1} \left(\frac{ST(1)}{ST(0)} \right)$$

✓ **FYL2X** This instruction is used to calculate $\log_2 X$ where X must be in the range of $0 \leq X \leq \infty$ and Y must be in the range $0 \leq Y \leq \infty$.

✓ **FYL2XP1** This instruction is used to compute the function $\log_2 (X + Y)$. This instruction is almost identical to FYL2X except that it gives more accurate results when computation log of a number very close to one.

Co-processor Control Instructions The co-processor control instructions are used to program the numeric processor or to handle the internal functions like flags manipulations, exception handling, processor environment maintenance and preparation, etc. The 8087 coprocessor control instructions are FINIT, FENI, FDISI, FLDCW, FSTCW, FSTSW, FCLEX, FINCSTP, FDECSTP, FFREE, FNOP, FWAIT, FSTENV, FLDENV, FRSTOR and FSAVE. In this section, operation of FINIT, FENI, FDISI and FWAIT instructions are discussed.

✓ **FINIT** The FINIT instruction initializes the 8087 for further execution. In other words, this instruction must be execute and the hardware will be reset before executing FPU instructions. The instruction initializes the control word register to 03FFH, the status register to 0 and the TAG status is set empty. All the flags are cleared and the stack top is initialized at ST (0).

✓ **FENI** The FENI instruction enables the interrupt structure and response mechanism of 8087. Therefore, the interrupt mask flag is cleared.

✓ **FDISI** The FDISI instruction sets the interrupt mask flag to disable the interrupt response mechanism of 8087.

✓ **FWAIT** The FWAIT instruction is used by the 8087 co-processor causes the microprocessor to wait for the coprocessor to finish an operation.

7.5.6 Assembly-Language Programs of 8087

Example 7.2

Write an assembly-language program to find out $z = \sqrt{x^2 + y^2}$. Assume x is stored in memory location 0200H and y is stored in memory location 0202H and the result z will be stored at 0300H.

Solution

<i>Mnemonics</i>	<i>Comments</i>
MOV BX,0200H	Store memory location (0200H) of first data x in Register BX
FLD (BX)	Load first data x into top of stack
FMUL	Multiply x with x and get x^2
FSTP ST(1)	Load x^2 in ST(1)
MOV BX,0202H	Store memory location (0202H) of second data in Register BX
FLD (BX)	Load second data y into top of stack
FMUL	Multiply y with y and get y^2
FADD ST(1)	Add x^2 with y^2 and result is stored in the top of stack
FSQRT	Find $z = \sqrt{x^2 + y^2}$
MOV BX,0300H	Store memory location 0300H in Register BX
FST (BX)	Store the result from top of stack to memory location 0300H
INT 3	Break

Example 7.3

Write a procedure in assembly-language to compute volume of a sphere $V = \frac{4}{3}\pi.r^3$ where r is the radius of sphere.

Solution

DATA SEGMENT

```
RADIUS DD 2.57
CONSTANT EQU 1.333
VOLUME DD 01 DUP(?)
```

DATA ENDS

ASSUME CS: CODE, DS:DATA

Volume PROC NEAR

Code SEGMENT

```
Start MOV AX,DATA           Initialize data segment
      MOV DS,AX
      FILD RADIUS           Load radius of sphere into top of stack
      FSTP ST(2)           Store top of stack into register ST(2)
      FMUL ST(2)           Multiply  $r$  with  $r$  and get  $r^2$ , ST(0) = ST(0) × ST(2) = ST(2)2
      FMUL ST(2)           Multiply  $r$  with  $r^2$  and get  $r^3$ , ST(0) = ST(0)2 × ST(2)
      FSTP ST(1)           Load  $r^3$  in ST(1)
      FLD CONSTANT         Load  $\frac{4}{3} = 1.333$  into top of stack
      FMUL ST(0), ST(1)    Multiply 1.333 with  $r^3$ 
```

```

        FSTP ST(3)          Store the result of 1.333 r3 in ST(3)
        FLDPI              Load the value of π into top of stack
        FMUL ST(0), ST(3)  Multiply π with 1.333 r3
        FST VOLUME        Store volume of sphere
        RETP
Volume ENDP
Code   ENDS
END    Start
    
```

Example 7.4

Write an assembly-language program to find out $\frac{xy}{x^2 + y^2}$. Assume x and y are integers.

Solution

<i>Mnemonics</i>	<i>Comments</i>
FILD x	Load first data x into top of stack
FMUL	Multiply x with x and get x^2
FSTP ST(1)	Load x^2 in ST(1)
FILD y	Load second data into top of stack
FMUL	Multiply y with y and get y^2
FADD ST(1)	Add x^2 with y^2 and result is stored in the top of stack
FSTP ST(2)	Store the result of $x^2 + y^2$ in ST(2)
FILD x	Load first data x into top of stack
FISTP ST(1)	Load x in ST(1)
FILD y	Load second data y into top of stack
FMUL ST, ST(1)	Multiply x with y and get $x y$
FDIV ST,ST (2)	Find $z = \frac{xy}{x^2 + y^2}$ and store the result into top of stack
INT 3	Break

7.6 80287 NUMERIC DATA PROCESSOR

The 80287 numeric data co-processor is an advanced version of its predecessor, the 8087, and it is specially designed to operate with the processor 80286. The 80287 provides about 70 additional instructions to the instruction set of 80286. These instructions are executed coherently by 80287 under the control of 80286. The 80287 co-processor instruction set can support integer, floating point, BCD, trigonometric and logarithmic calculations. Like 8087, the 80287 is designed using HMOS technology and it is available in a 40-pin DIP package. The block diagram of the internal architecture of 80287 is shown in Fig. 7.50. The internal architecture of 80287 consists of three sections such as

- ✦ Bus Control Logic,
- ✦ Data interface and Control Unit, and
- ✦ Floating Point Unit.

Bus Control Logic The bus control logic controls the interface between the internal data bus of 80287 and the 80286 bus through data buffer.

Data Interface and Control Unit The data interface and control unit consists of status word, control word, TAG word, error pointers, DATA FIFO, instruction decoder and a micro-instruction sequencer. The status word represents the present status of the 80287 co-processor. The control word is used to select

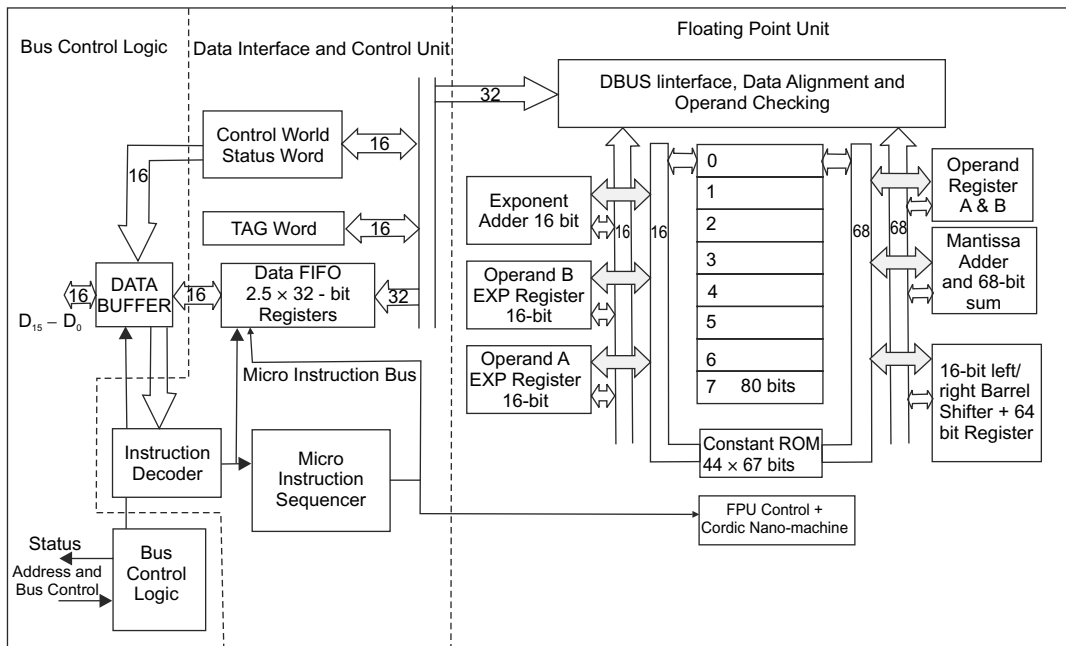


Fig. 7.50 Block diagram of internal architecture of 80287

any processing options provided by the 80287 co-processor and it is programmed by the main processor. The TAG word is used to improve the performance of 80287 numeric data co-processor by maintaining a record of empty and non-empty register locations. The error pointer points to the address of the instruction which generates the exception. The instruction decoder and micro-instruction sequencer decodes and forwards the instructions to the floating point unit for execution.

Floating-Point Unit

The floating-point unit is the actual data-processing section of the 80287 numeric data co-processor. This section consists of DBUS interface, data alignment and operand checking, exponent adder, operand registers (A and B), mantissa adder, sum register, 16-bit left/right barrel shifter, operand A and B exp registers. The data bus interface, data alignment and operand checking section is used to check the alignment and validity of the data. Whenever any error is found, a suitable error exception must be generated by the 80287. Usually the eight 80-bit stack registers are used to store operand data. The 80-bit stack registers always maintain 80-bit operands which are required for 80287 operations. The data bus of the floating-point unit consists of 84 bits. The lower 68 bits are mantissa data bit and the next 16 bits are used for exponent. The exponential operand registers A and B are used to store the operands in exponential form. The barrel shifter is used to shift the data which is required for execution.

Status Word

The status word is a set of 16 flags which reflects the current status of 80287 as shown in Fig. 7.5. The operations of the various flags are explained below:

✓ **B Flag** The BUSY flag has the same status as ES flag. This is used to maintain the compatibility with 8087.

✓ **TOP** The D_{13} , D_{12} and D_{11} bits are used to select one of the eight stack registers as a stack top. If TOP is 000, Register 0 is the top of stack. When TOP is 111, Register 7 is the top of stack. Similarly other registers will be selected depending upon the status of TOP.

- ✓ **C₃, C₂, C₁ and C₀** The condition code bits C₃, C₂, C₁ and C₀ are similar to the flags of a main processor. Usually, these bits are modified depending upon the result of the execution of arithmetic instructions.
- ✓ **ES** This is the error summary bit. If an unmasked exception is generated, it is set.
- ✓ **SF** This bit is used as stack flag. When the operation becomes invalid due to stack overflow or underflow, the stack flag is set.
- ✓ **Exception Flags** All exception flags are depicted in Fig. 7.51. These are used to show the generation of an exception when 80287 is executing.

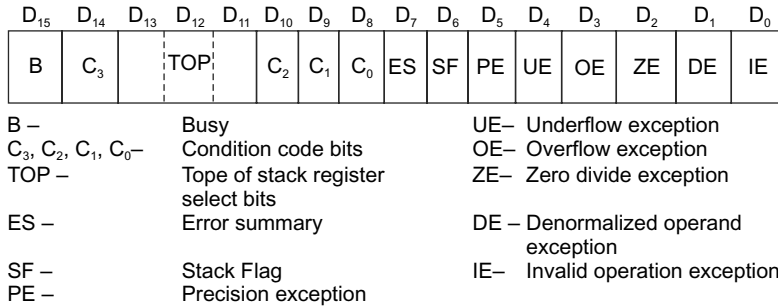
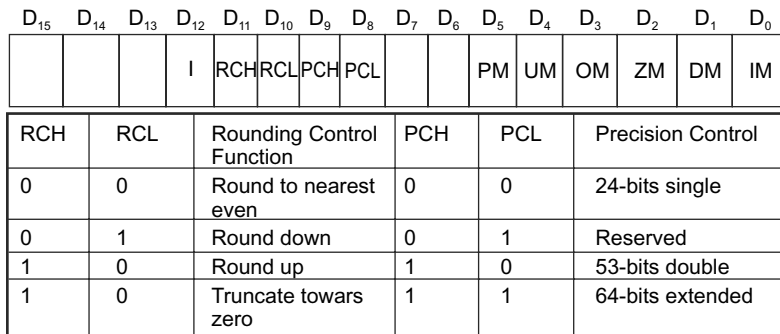


Fig. 7.51 Status word of 80287

Control Word

The control word is used to select any processing options of the 80287 co-processor. The control word the 80287 co-processor is shown in Fig. 7.52.



- | | |
|-------------------------|--------------------------------|
| I – Infinity control | PM – Precision mask |
| RCH – Rounding control | UM – Overflow mask |
| RCL – | – |
| PCH – Precision control | OM – Overflow mask |
| PCL – | – |
| | ZM – Division by zero mask |
| | DM – Denormalized operand mask |
| | – |
| | IM – Invalid operand mask |

Fig. 7.52 Control word of 80287

- ✓ **Masking Bits** The six masking bits (PM, UM, OM, ZM, DM and IM) are used to mask the six exceptions as shown in the status register. When the masking bit is , ‘1’, the respective exception is masked.

- ✓ **Precision Control Bits** The precision control bits are used to set the internal precision of 80287.
- ✓ **Rounding Control Bits** The rounding control bits are used to set rounding operation in arithmetic and transcendental instructions.
- ✓ **Infinity Control Bit** The infinity control bit is programmed for compatibility with 80287. This is initialized to zero after reset.

7.6.1 PIN Descriptions of 80287

The pin diagram of the 80287 co-processor is shown in Fig. 7.53 and the function of pins are given below:

- ✓ **$D_{15}-D_0$** The $D_{15}-D_0$ is used as a 16-bit data bus which is connected with the 80286 data bus.
- ✓ **CLK** This is a clock input pin through which 80287 co-processor receive the required clock for deriving the system.
- ✓ **RESET** The RESET is used to reset 80287 co-processor.
- ✓ **\overline{NPRW} (Numeric Processor Write)** The numeric processor write input pin enables a data transfer from the 80286 main processor to 80287 co-processor.
- ✓ **\overline{NPRD} (Numeric Processor Read)** The numeric processor read active-low input signal is used to enable data transfer from 80287 coprocessor to 80286 main processor.
- ✓ **\overline{NPS}_1 and \overline{NPS}_2** Numeric processor select input pins \overline{NPS}_1 and \overline{NPS}_2 indicate that the CPU is performing an escape operation. These pins are also used to enable 80287 co-processor to execute the next instruction.
- ✓ **CMD_0 and CMD_1** The command pins CMD_0 and CMD_1 are used to indicate that the CPU is performing an ESC instruction and to control the operations of 80287.

- ✓ **ERROR** The error status output signal represents the ES bit of the internal status register. When this pin is active low, it indicates that an exception has occurred.
- ✓ **BUSY** The BUSY output pin indicates to the main processor that 80287 co-processor is busy with the execution of an instruction. This pin is connected with the TEST pin of 80286.
- ✓ **PEREQ (Processor Extension Request)** The processor extension request active HIGH output pin is used to indicate to the 80286 that the 80287 NDP is ready for data transfer.
- ✓ **\overline{PEACK} : (Processor Extension Acknowledge)** The processor extension acknowledge active LOW input pin is used by the main processor to acknowledge a receipt of a valid processor extension request signal.
- ✓ **CKM (Clock Mode)** When the clock mode input pin, CKM is high, the CLK input is directly used for deriving the internal timings.

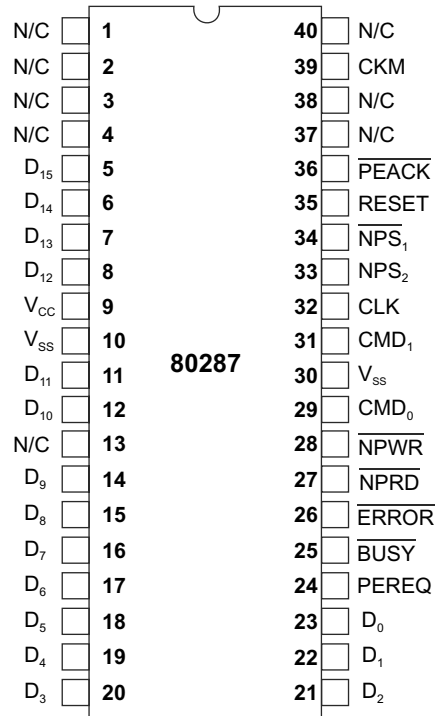


Fig. 7.53 Pin diagram of 80287

7.7 80387 NUMERIC DATA PROCESSOR

The 80387 numeric data co-processor is an advanced version of 80287 and it is a high performance numeric data processor and it is specifically designed to operate with 80386 CPU. The instruction set of the 80387 co-processor is transparent to the programmers. The 80387 provides six to eleven times better performance as compared to 80287. The block diagram of 80387 architecture is shown in Fig. 7.54.

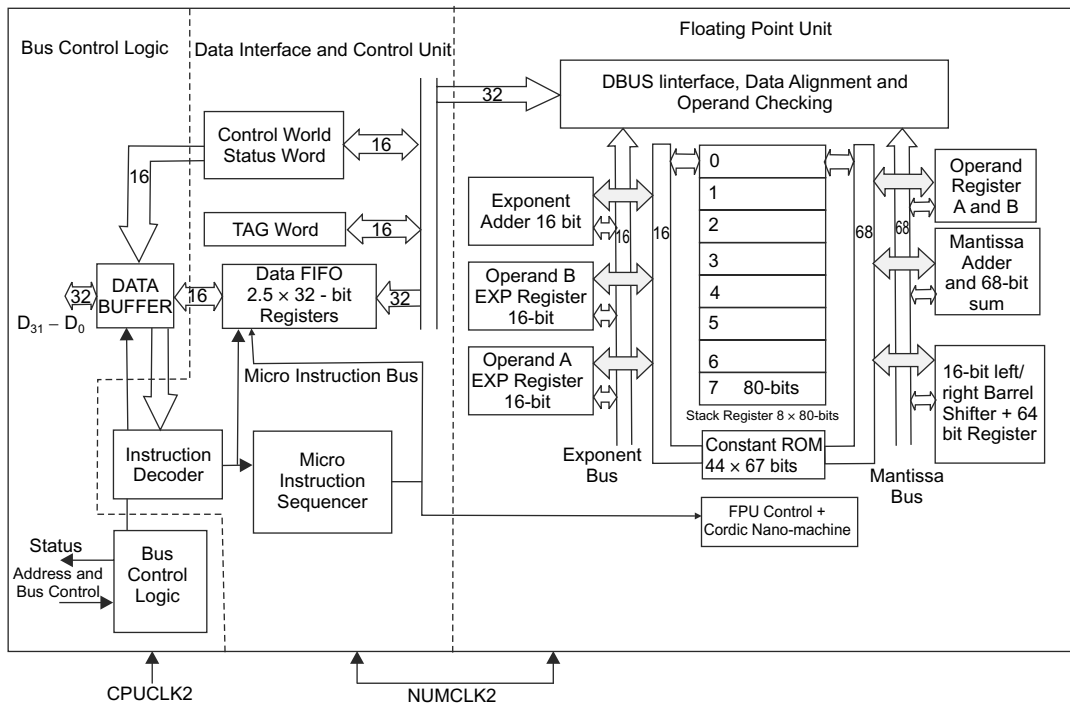


Fig. 7.54 Block diagram of internal architecture of 80387

The 80387 has three functional units such as *bus control logic unit*, *data interface and control unit* and *floating-point unit*. These units operate in parallel to increase the performance. The operation of 80387 is just like 80287, but the data bus size is different. The data bus of 80387 co-processor is 32 lines D_0-D_{31} .

The two clock input signals of 80387 are used for asynchronous or synchronous operations with 80386 main processor. Usually these operations are selected using the CKM pin of 80387. When the CKM is high, the 80387 operates in synchronous mode. If the CKM is low, it operates in the asynchronous mode.

- ✦ The *bus control unit* operates synchronously with 80386, and its operation is independent of the mode of operation of the floating point unit. With the help of READY input pin, the ADS input pin is used to provide delay the bus cycles with respect to CPUCLK2 pin. The status enable pin is used as a chip select for the 80387 numeric data co-processor. The other pins of 80387 work as the analogous pins of 80287.
- ✦ The *data interface and control unit* can handle the data through FIFO or instruction decoder depending upon the bus control logic directive. The decoder decodes the instruction and generates the control signals to control the data flow within the 80387. This unit also generates the synchronization signals for 80386.

✦ The *floating-point unit* is used for all the floating point operations in the 80287 coprocessor.

The 80387 co-processor has eight 80-bit floating point data registers, which are used to store signed 80-bit data. Each register has a corresponding 2-bit tag field. The 80387 has a 16-bit control, status and tag word registers. The 80387 has two more 48-bit registers known as instruction and data pointers. The tag word register and the status registers have exactly similar formats as those of 80287 respectively. The 80387 can be operated as per required by the programmer, when a control word is loaded from memory to its control word register. The control word register has exactly similar format as that of 80287. The data types of 80387 are just like the data types of 80287.

SUMMARY

- In this chapter a brief introduction to machine languages and assembly languages with their advantages and disadvantages are discussed.
- An assembler is a program that converts any assembly-language program into the equivalent machine codes. In this chapter, the most commonly used assemblers such as Norton's Editor, Microsoft Assembler, Linker and Debugger are explained briefly. All commands of DEBUG with examples are discussed elaborately.
- Emphasis on different programming techniques such as use of looping, data transfer, arithmetic and logical instructions are also incorporated in the chapter.
- Assembly-language programs for addition, subtraction, multiplication, division and conversion from BCD to binary, arranging a string in ascending order and descending order, finding the largest and smallest values of a string, block movement, and look-up table applications are illustrated
- The block diagram, operating principle, instruction set and programming of 8087, 80287 and 80387 numeric data processors are discussed elaborately.

MULTIPLE-CHOICE QUESTIONS

- 7.1 What is the output of DL after execution of the following instructions?
 MOV DL, 36
 AND DL, 0F
 (a) DL = 06H (b) 60H
 (c) 36H (d) 0FH
- 7.2 What is the content of AX and DX after execution of the following instructions?
 MOV BL, 9
 MOV AX, 0702
 AAD
 DIV BL
 (a) AX = 0080H BX = 0009H
 (b) AX = 0008H BX = 0009H
 (c) AX = 0008H BX = 0090H
 (d) AX = 0800H BX = 0900H
- 7.3 What is the result after addition of two ASCII number 33 and 37 by the following instructions
 MOV AX, 0037
 ADD AX, 0033
 AAA
 OR AX 3030
 (a) 303 1 (b) 1303
 (c) 3130 (d) 3310
- 7.4 After multiplication of two numbers, the result in AX will be
 MOV AL,05
 MOV CL,05

- MUL CL
AAM
(a) AX = 0205 (b) AX = 0250
(c) AX = 0025 (d) AX = 2500
- 7.5 Which one of the following program is the right program for finding the complement of a number?
(a) MOV AX,2345 (b) MOV AX,2345
 NEG AX CMP AX
(c) MOV AX,2345 (d) MOV AX,2345
 NOT AX CMC
- 7.6 The result of unsigned multiplication of two numbers is
MOV CL,25
MOV AL,35
MUL CL
(a) AX = 7A90H (b) AX = 907AH
(c) AX = A907H (d) AX = 07A9H
- 7.7 The result for addition of two numbers is
MOV AX,A233
MOV BX,A455
ADD AX,BX
(a) 4688H (b) 4886H
(c) 8846H (d) 6884H
- 7.8 How many *T* states are required to execute for the following instructions?
Start: MOV CX,2244 4 *T* states
 DEC CX 2 *T* states
 NOP 3 *T* states
 JNZ Start 16 *T* states
(a) 256 *T* states (b) 184216 *T* states
(c) 2560 *T* states (d) 2000 *T* states
- 7.9 Content of AL after execution of the following instructions
MOV AH, 06H
MOV AL, 06H
MOV BL, 08H
SUB AX, BX
(a) 08 (b) FE
(c) F8 (d) 80
- 7.10 To execute a program which command is used?
(a) R (b) G (c) E (d) F
- 7.11 Which command is used to see the flag register status?
(a) U (b) A (c) R (d) F
- 7.12. After execution of MOV AX, 9535 and RCL AX,1 the content of AX is
(a) 2A6A Carry = 0 (b) 2A6A Carry = 1
(c) A26A Carry = 1 (d) 2AA6 Carry = 1
- 7.13. If the following instructions are executed
MOV CL,02;
MOV AX,9535;
RCL AX,CL;
Result of AX register will be
(a) AX = 54D5 carry = 0
(b) AX = 54D5 carry = 1
(c) AX = 45D5 carry = 0
(d) AX = 545D carry = 1
- 7.14. After execution of the following instructions, the content of AX register is
MOV AL,07;
MOV BL,09;
MUL BL
(a) 0063 (b) 003F
(c) 6300 (d) 3F00
- 7.15. If the following instructions are executed
MOV CX,55
MOV AX,3535
DIV CX
(a) Quotient AL = 15 Remainder AH = A0
(b) Quotient AL = 0A Remainder AH = 51
(c) Quotient AL = A0 Remainder AH = 15
(d) Quotient AL = A0 Remainder AH = 51
- 7.16 When the MOV BX, 2467 and ADD BX, 4 instructions are executed, the result will be
(a) 2468 (b) 246A
(c) 246D (d) 246B
- 7.17 2's complement of AX is
(a) NEG AX (b) NOT AX
(c) CMP AX (d) XOR AX
- 7.18 The content of AX after execution of MOV AX, 3957; NOT AX is
(a) C68A (b) 6CA8
(c) CA86 (d) C6A8

- 7.19 The 8087 coprocessors operate in ____ with an 8086 processor and with the same instruction ____
 (a) Series, byte (b) Parallel, byte.
 (c) Series, bits (d) Parallel, bits
- 7.20 The synchronization between processor and coprocessor can be done by ____ connection and the ____ instruction.
 (a) RQ/GT₀ and RQ/GT₁, FWAIT
 (b) INT and NMI, WAIT
 (c) BUSY and TEST, FWAIT
 (d) S₀ and QS₀, WAIT
- 7.21 The 8087 Co-processor has ____ registers at the stack and registers are used as ____ stack.
 (a) 8, 80 bit, LIFO (b) 8, 60 bit, LILO
 (c) 7, 40 bit, FIFO (d) 7, 80 bit, FILO
- 7.22 8087 can be connected with another coprocessor through ____ pin and the co-processor operate in ____ with the 8087. .
 (a) QS₁ and QS₀, parallel
 (b) RQ/GT₁, parallel
 (c) RQ/GT₀, parallel
 (d) S₀ and S₁, parallel.

SHORT-ANSWER-TYPE QUESTIONS

- 7.1 What is an assembler? What are the different assemblers used in 8086 programming?
- 7.2 What are the advantages of assembly-language programming over machine language?
- 7.3 Why 8087 is referred to as co-processor?
- 7.4 What are the different sections in 8087, 80287 and 80387 architecture?
- 7.5 Which instruction is used by 8087 to fetch co-processor instructions?
- 7.6 What are the different types of instructions in 8087 co-processor instruction set?
- 7.7 What are the different data types supported by 8087?
- 7.8 What are the different exceptions generated by 8087?
- 7.9 How can the main processor differentiate the 8087 instructions from its own instructions?

REVIEW QUESTIONS

- 7.1 Explain DEBUG with some of its important commands.
- 7.2 What is A command? What is U command? Explain the operation of the following commands:
 (i) -U 0100 0120 (ii) -U 0100 L10 (iii) -D0100 L 10
 (iv) -T = 0100 05 (v) -ECS:100 (vi) -M 0100 0105 0300
 (vii) S 0200 0235 46
- 7.3 Write the commands for the following operations:
 (i) To display the content of Register AX
 (ii) To display the content of memory locations CS : 0200 to CS : 0250
 (iii) To enter data 22H, 44H, 66H 77H and FFH in the memory location starting from DS : 0300
 (iv) To search a byte 45H from a string DS : 0400 to DS : 0500
 (v) To display the status of flags
- 7.4 Write an assembly-language program to find the largest number in a data array.
- 7.5 Write an assembly-language program to find the smallest number in a data array.
- 7.6 Write an assembly-language program to arrange numbers in descending order.

- 7.7 Write an assembly-language program to arrange numbers in ascending order.
- 7.8 Write an assembly-language program to block move from one memory location to another location.
- 7.9 Write an assembly-language program to find the sum of a series of 16-bit numbers.
- 7.10 Write an assembly-language program to find the subtraction of two 3×3 matrices.
- 7.11 Write an assembly-language program for addition, subtraction, multiplication and division of two numbers.
- 7.12 Write an assembly-language program for addition of first 100 decimal numbers.
- 7.13 A block of 16 data is stored at the memory location starting from DS:0100. Move this block to the memory location starting from DS:0500.
- 7.14 Write an assembly-language program to convert a 16-bit binary number into its equivalent GRAY code.
- 7.15 Write an assembly-language program to find out the number of occurrences of a byte 44H in a string of bytes which is stored in the memory location starting from CS:0300 to CS:0320.
- 7.16 Write assembly-language programs to find
- (i) $n!$ (ii) $\frac{n!}{(n-r)!r!}$ (iii) ${}^n C_r$, assume $N = 9$, $r = 2$ and $p = 3$.
- 7.17 Write an assembly-language program for addition of two 8-bit numbers and with a sum of 8 bits.
- 7.18 Write an assembly-language program for addition of a string of bytes whose sum is 8 bits.
- 7.19 Write an assembly-language program for addition of two 16-bit numbers whose sum is 16 bits.
- 7.20 Write an assembly-language program for subtraction of two 8-bit numbers.
- 7.21 Write an assembly-language program for finding 1's complement of an 8-bit number.
- 7.22 Write an assembly-language program for finding 1's complement of an 16-bit number.
- 7.23 Write an assembly-language program for finding 2's complement of an 8-bit number.
- 7.24 Write an assembly-language program for finding 2's complement of a string of bytes
- 7.25 Write an assembly-language program to multiply two 8-bit numbers.
- 7.26 Write an assembly-language program to multiply -24 and 11, Two 8-bit numbers.
- 7.27 Write an assembly-language program to divide two 8-bit numbers.
- 7.28 Write an assembly-language program for decimal addition of two 16-bit numbers whose sum is 16 bits.
- 7.29 Write an assembly-language program for shift left of a 16-bit number by two bits.
- 7.30 Write an assembly-language program to find out the smallest number from a string of bytes.
- 7.31 Write an assembly-language program to find out the largest number from a string of words.
- 7.32 Write an assembly-language program to find out the smallest number from a string of words.
- 7.33 Write an assembly-language program to subtract two ASCII numbers.
- 7.34 Write an assembly-language program to arrange a string of bytes in ascending order.
- 7.35 Write an assembly-language program to arrange a string of bytes in descending order.
- 7.36 Write an assembly-language program to find out square root of a number using a look-up table.
- 7.37 Write an assembly-language program to find the transpose of a 3×3 matrix.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \text{ and } A^T = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{bmatrix}$$

- 7.38 Write an assembly-language program to convert a binary number to its equivalent BCD number.
 7.39 Draw the block diagram of the 8087 co-processor architecture and explain briefly.
 7.40 Discuss the register organization of the 8087 NDP.
 7.41 Draw the interface diagram between 8086 processor and 8087 co-processor and explain briefly.
 7.42 Write assembly-language programs to perform the following operations using 8087:
 (a) Calculate area of a circle (b) Compute x^y
 (c) Compute $\log_2 x$ (d) Compute 2^y

Answers to Multiple-Choice Questions

-
 7.1 (a) 7.2 (b) 7.3 (c) 7.4 (a) 7.5 (c) 7.6 (d) 7.7 (a) 7.8 (b) 7.9 (b)
 7.10 (b) 7.11 (d) 7.12 (b) 7.13 (a) 7.14 (b) 7.15 (c) 7.16 (d) 7.17 (a) 7.18 (d)
 7.19 (b) 7.20 (b) 7.21 (a) 7.22 (b)

Chapter 8

I/O and Memory Interfacing Using 8085/8086

8.1 INTRODUCTION

The microprocessor is a very powerful IC which is used to perform various ALU functions with the help of data from the environment. For this, the microprocessor is connected with memory and input/output devices to form a microcomputer. The technique of connection between input/output devices and the microprocessor is known as *interfacing*. Special attention must be always be given during the connection of pins of peripheral devices and microprocessor pins, as ICs cannot be simply connected. In the development of a microprocessor-based system, all memory ICs and input/output devices are selected as per requirement of the system and then interfaced with the microprocessor. Actually address, data and control lines are used for connecting peripherals. After connecting them properly, programs are embedded in the microprocessor. Programs will be different for different applications. When a program is executed, the microprocessor communicates with input/output devices and performs system operations. In this chapter the interfacing of memory, Programmable Interrupt Controller 8259A, Programmable Peripheral Interface (82C55 PPI) and Programmable Interval Timer Intel 8253 are explained.

8.2 MEMORY INTERFACING

Memory devices are used to store digital information. The simplest type of digital memory device is the flip-flop, which is capable for storing single-bit data, and is volatile and very fast. This device is generally used to store data in the form of registers. Registers are also used as main memory of computers for internal computational operations. The basic goal of digital memory is to store and access binary data, which is a sequence of 1's and 0's. In this section, different types of memories and their interfacing with microprocessors are explained.

8.2.1 Types of Memories

There are two types of semiconductor memories, namely, ROM and RAM. ROM stands for Read Only Memory. Data are permanently stored in memory cells. We are able to read data from the memory. ROM cannot be reprogrammed. This memory is nonvolatile and data is retained when power is switched off. But the data contents of ROM are accessed randomly just like the volatile memory circuits. Vinyl records and compact audio disks are typically referred as read only memory or ROM in the digital system.

Classification of ROM

ROMs are manufactured with bipolar technology and MOS technology. Figure 8.1 shows the classification of ROM. The different features of ROM, PROM, EPROM and EEPROM are explained below:

✓ **ROM (Read Only Memory)** The data is permanently stored in the memory and these devices are mask programmed during manufacturing. ROMs cannot be reprogrammed and are of nonvolatile type. These devices are cheaper than programmable memory devices. The applications of ROM are fixed programmed instructions, look-up tables, conversions, and some specific operations.

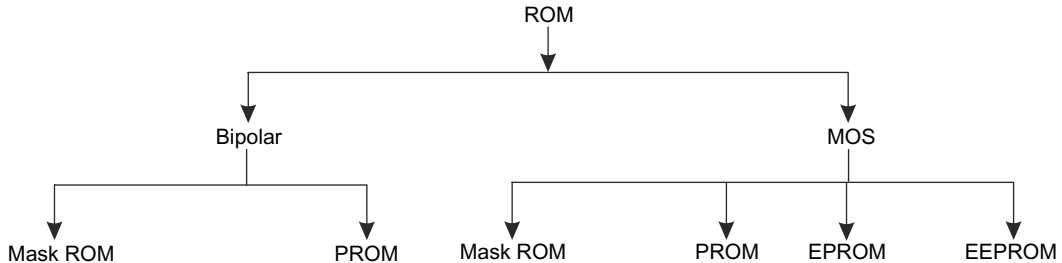


Fig. 8.1 Classification of ROM

✓ **PROM (Programmable Read Only Memory)** The data can be electrically stored. It can be programmed by blowing built-in fuses and can be reprogrammed and is of nonvolatile type. These memory devices are of very low memory density and occupy more space.

✓ **EPROM (Erasable Programmable Read Only Memory)** These are strictly MOS devices and programmed by storing charge on insulated gates. These devices are erasable with ultraviolet rays and become reprogrammable after erasing. These memory devices are of nonvolatile type.

✓ **EEPROM (Electrically Erasable Programmable Read Only Memory)** These memory devices are electrically programmable by the programmer and the stored data can be erased by ultraviolet light. They are of nonvolatile type. This is also called Electrically Alterable Programmable Read Only Memory (EAPROM).

Classification of RAM

Read only memory is used only for reading data stored in the memory. ROMs can be programmed only once and data once recorded cannot be erased. In a RAM, data can be written into its memory as often as desired and the data stored in a RAM can be read without destroying the contents of the memory.

Data can be written into and read from a RAM at any selected address in any sequence. When data are written into a given address in the RAM, the data previously stored at that address are destroyed and replaced by the new data. When data are read from a given address in the RAM, the data at that address are not destroyed. The nondestructive read operation can be thought of as copying the contents of an address while leaving the content intact. A vinyl record platter is an example of a random-access device. RAM memory is typically randomly accessed; it is actually virtually/volatile memory.

There are two types of RAMs, static and dynamic. The basic memory cell in a static RAM is a flip-flop, bipolar or MOS. After a bit has been stored in the flip-flop of a memory cell, it will remain there while power is available. Dynamic RAMs (called DRAMs) are based on charge, which is stored by using MOS devices. Since this charge is dissipated by passage of time, DRAMs need periodical recharging or refreshing. RAMs

and dynamic RAMs are volatile devices. The comparison between different memories based on category, erasing property, writing mechanism and volatility is illustrated in Table 8.1.

Table 8.1 Comparison of memories

Type of Memory	Category	Erasing property	Writing mechanism	Volatility
Read only Memory(ROM)	Read only Memory	Not possible	Masks	Nonvolatile
Programmable ROM(PROM)	Read only Memory	Not possible	Electrically	Nonvolatile
Erasable PROM(EPROM)	Read only Memory	Ultraviolet light and chip level	Electrically	Nonvolatile
Electrically Erasable PROM(EEPROM)	Read only Memory	Electrically and byte level	Electrically	Nonvolatile
Flash memory	Read only Memory	Electrically and block level	Electrically	Nonvolatile
Random access memory(RAM)	Read and write memory	Electrically and byte level	Electrically	Volatile

RAMs are also manufactured with bipolar technology and MOS technology. The bipolar RAMs are static RAMs but MOS RAMs are of static and dynamic types. Figure 8.2 shows the classification of RAM. The different features of static and dynamic RAMs are explained below:

Static RAMs

These RAMs are built with static or dynamic cells. Five or six transistors are used to store a single bit. Data can be written and read in nanoseconds. TTL, ECL, NMOS and CMOS technology are used to manufacture static RAMs. When the power is shut off, data stored in cells can be lost.

Dynamic RAMs

In a dynamic memory, data can be stored on capacitors and to retain data every cell has to be refreshed periodically. One transistor is used to build a memory cell and requires less space. These memories consume less power compared to static RAMs. The comparison between SRAM and DRAM is given below:

Table 8.2 Comparison of Static RAM and Dynamic RAM

Static RAM	Dynamic RAM
<ul style="list-style-type: none"> Stored data is retained as long as power is ON 	<ul style="list-style-type: none"> Stored data will be erased and repeated refreshing is required to store data
<ul style="list-style-type: none"> Stored data will not be changed with time 	<ul style="list-style-type: none"> Stored data will be changed with time
<ul style="list-style-type: none"> Consume more power 	<ul style="list-style-type: none"> Consume less power than static RAM
<ul style="list-style-type: none"> SRAM is expensive 	<ul style="list-style-type: none"> SRAM is less expensive
<ul style="list-style-type: none"> SRAM has less packing density 	<ul style="list-style-type: none"> DRAM has high packing density
<ul style="list-style-type: none"> Not easy to construct 	<ul style="list-style-type: none"> Construction is simple
<ul style="list-style-type: none"> No refreshing is required 	<ul style="list-style-type: none"> As refreshing is required, additional circuit is incorporated with the memory
<ul style="list-style-type: none"> No maintenance is required 	<ul style="list-style-type: none"> Maintenance is required

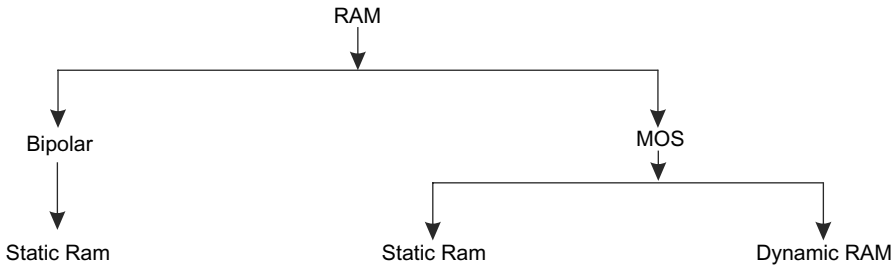


Fig. 8.2 Classification of RAM

8.2.2 Memory Organisation

Figure 8.3 shows the block diagram of a $M \times K$ bits memory structure. It has N bit input lines to locate an address of memory and each address line can store K bits. So the total number of bits in the memory is $M \times K$ bits. Each memory location is represented by address lines to locate M address locations. Here N bit inputs are required to locate M address locations. The relationship between address locations and input lines is $2^N = M$. To generate an address line, an N lines to M lines decoder is used. Actually, the decoder decodes M locations depending upon inputs. The number of locations and number of bits may be varied for different memories. When M is the number of locations and K is the number of bits in each location, the size of the memory is $M \times K$ bits. The size of commonly used memory devices are 64, 256, 512, 1024 (1K), 2048 (2K), 4096(4K), 16384 (16K) but the common values of word size are 1, 2, 4, 8, 12, 16, etc. The Chip Enable (CE) signal is used to enable the address lines for selecting a bit or a group of bits.

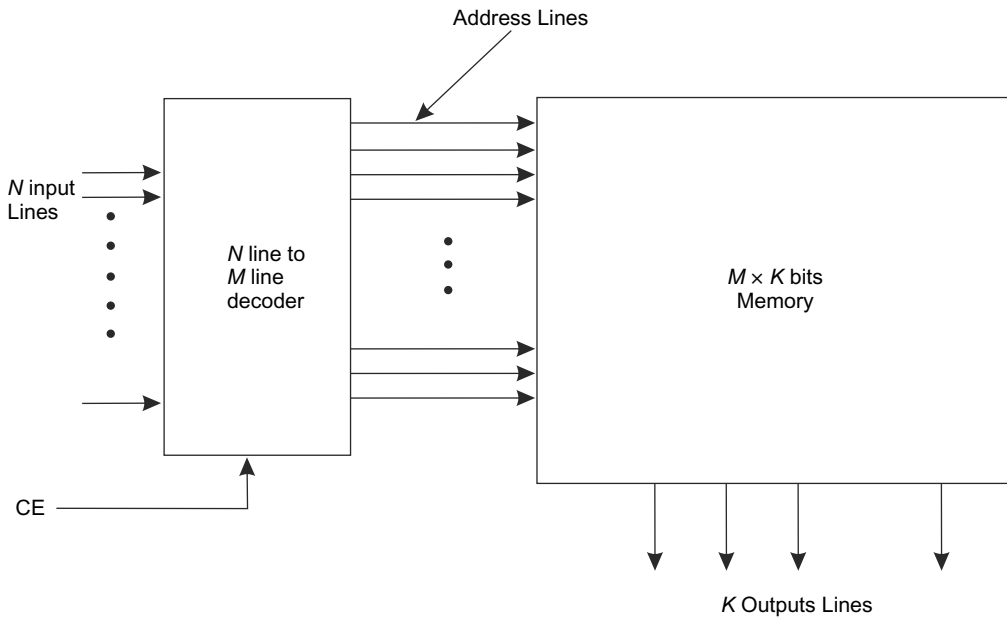


Fig. 8.3 Block diagram of $M \times K$ bits memory

Nowadays all the semiconductor memory devices are now available in Integrated Circuit (IC) form. Each memory IC can store a large number of words. Memory ICs are available in various sizes. The examples of ICs are 64×4 (64 words of 4 bits each), 256×8 (256 words of 8 bits each), $1 K \times 8$ (1024 words of 8 bits each), $1 M \times 8$ (1,048,576 words of 8 bits each).

Each memory IC should have address and data lines including chip select \overline{CS} , output enable \overline{OE} and Read/Write R/\overline{W} control signals. Figure 8.4 shows the memory organization of an IC. This chip has address lines, data lines, chip select signals and read and write control signals which are explained below:

Address Lines

The memory ICs should have address lines to receive the address values. For a 1K-byte memory, ten address lines A_0 – A_9 exist. The relationship between number of address lines and size of memory is 2^n , where n is the number of address lines. Similarly, for 64K byte memory, number of address lines is 16, A_0 to A_{15} address line, and A_0 – A_{n-1} to select one of the 2^n memory locations.

Data Lines

Data lines are provided for data input to the IC during write operation, and data output from IC during read operation. M data lines D_0 – D_{m-1} are used for data transfer between the microprocessor and memory IC.

Chip Select Signal \overline{CS}

The chip select signal \overline{CS} can enable the chip. When the \overline{CS} is low, memory access within the chip is possible.

Read or Write R/\overline{W}

The read or write operation can be performed based on R/\overline{W} control signal. If $R/\overline{W} = 1$, data will be read from memory. When $R/\overline{W} = 0$, data will be stored in the memory IC.

Output Enable \overline{OE}

The output enable signal is used to connect the output with the data bus.

For example, the memory organization of 256×4 memory IC is depicted in Fig. 8.5. This memory IC has 8 address lines A_0 – A_7 to select 256 memory locations. As there are four data lines, 4-bit data can be stored in each location. Therefore, the size of the memory is 256×4 bits.

Memory ICs are available in four-bit and eight-bit word configurations. In some applications, sixteen bits and more than sixteen bits are also used. The memory capacity of each IC is limited. Therefore, memory expansion is required. The memory size can be expanded by increasing the word size and address locations. The memory expansion can also be possible by proper interconnections of decoder and memory ICs. Figure 8.6 shows $2K \times 8$ bits memory using two $1K \times 8$ bits.

A 2K byte RAM can be developed using two 1K byte RAM ICs. In this case, \overline{CS} is directly connected with IC₁ and the complement of \overline{CS} is connected to IC₂. R/\overline{W} and \overline{OE} control signals of both ICs are directly interconnected as given in Fig. 8.6. The address lines A_0 – A_9 of the ICs are connected in parallel. Chip 1

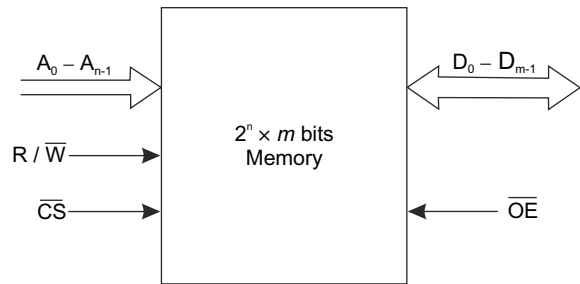


Fig. 8.4 Memory organization of $2^n \times m$ bits memory IC

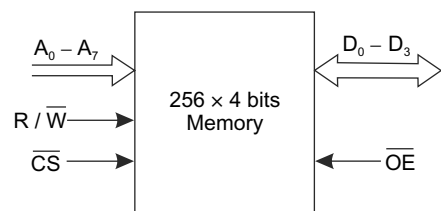


Fig. 8.5 Memory organization of 256×4 bits memory IC

provides the 1K addresses from 0 to 1023, and Chip 2 provides the next 1K addresses from 1024 to 2047. As for the first 1K addresses, Chip 1 is activated and for the next 1K addresses, Chip 2 is activated. The chip select signal is connected with the address line A_{10} . Each chip also provides 8 data lines D_0-D_7 . So that memory size is increased from 1K byte to 2K bytes.

Another example is that two $1K \times 4$ bits RAMs can be combined to develop a 1K byte RAM as depicted in Fig. 8.7. Chip 1 and Chip 2 have ten address lines which are connected in parallel. The chip select \overline{CS} , read/write R/\overline{W} and output enable \overline{OE} are also connected together. In this case, memory size is fixed, but word size is increased from 4 bits to 8 bits. IC_1 and IC_2 are selected at a time for 8-bit data storage or data read operation.

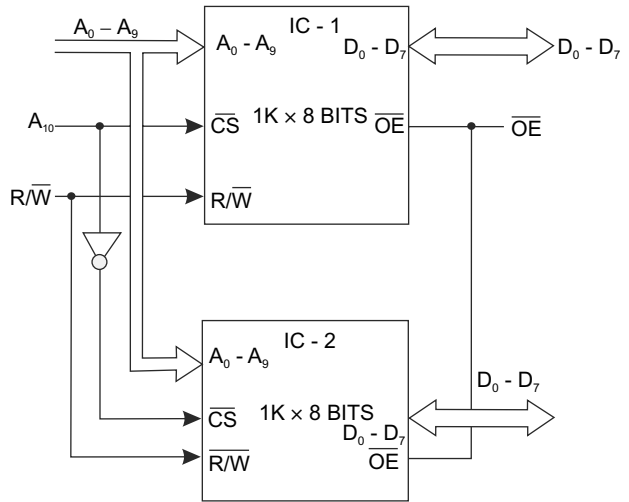


Fig. 8.6 Memory organization of $2K \times 8$ bits memory using two $1K \times 8$ bits.

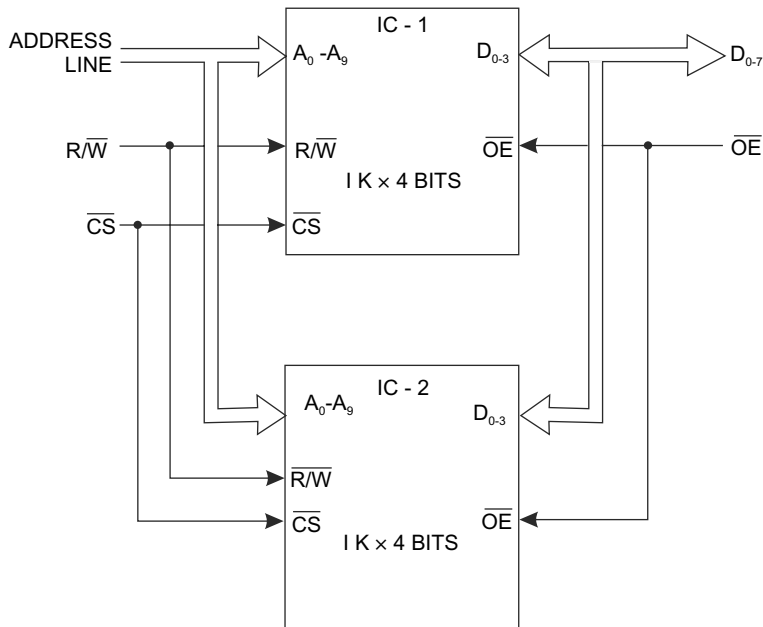


Fig. 8.7 Memory organization of $1K \times 8$ bits memory using two $1K \times 4$ bits

The memory organization of 2K bytes using four chips of $1K \times 4$ bits is illustrated in Fig. 8.8. In this case, the memory size as well as word size are increased. The memory size is increased from 1K to 2K and the word size is also increased from 4 bits to 8 bits. IC_1 and IC_2 are selected at a time for 8-bit data storage or data read operation. Similarly, IC_3 and IC_4 are used for $1K \times 8$ bits memory read/write operations.

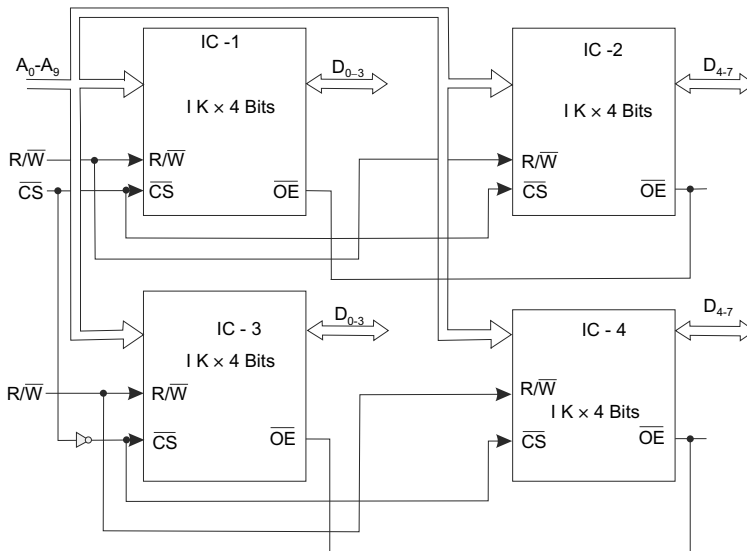


Fig. 8.8 Memory organization of $2K \times 8$ bits memory using four $1K \times 4$ bits

8.2.3 ROM and RAM ICs

The organization and operation of memory is already explained briefly in Section 8.2.2. Most commonly used ROM and RAM ICs are given in tables 8.3 and 8.4 respectively with their category, organization, package, access time, technology and power dissipation, etc.

Table 8.3 Commonly used ROM ICs

IC No.	Category	Package	Organization	Technology	Access time	Power dissipation
6206D	Mask PROM	16-pin DIP Package	512×4	Bipolar	60 ns	625 mW
23C1010	MASK ROM	32-pin DIP/ PDIP/SOP/ PLCC/TSOP Package	$128K \times 8$	MOS	45 ns	250 mW
23C2000	MASK ROM	32-pin PDIP/ PLCC/SOP/ TSOP Package	$256K \times 8$	MOS	70 ns	250 mW
23C4000	MASK ROM	32-pin PDIP/ PLCC/ SOP/ TSOP Package	$512K \times 8$	MOS	90 ns	210 mW
23C6410	Mask ROM	44-pin SOP and 48 pin TSOP Package	$8M \times 8$ $4M \times 16$	MOS	100 ns	420 mW
Am1702A	PROM	24-pin duel in-line hermetic cerdip package	256×8	MOS	550 ns	1000 mW

(Contd.)

(Contd.)

3602A	PROM	16-pin DIP package	512×4	Bipolar	70 ns	750 mW
3605	PROM	18-pin DIP package	1024×4	Bipolar	70 ns	800 mW
27C256	EPROM	28-pin DIP package and a 32-pin windowed LCC	$32,768 \times 8$	Low-power CMOS	250 ns	55 mW
2716	EPROM	24-pin DIP package	2048×8	MOS	450 ns	525 mW
2732A	EPROM	24-pin DIP package	4096×8	MOS	250 ns	790 mW
2764	EPROM	28-pin DIP package	8192×8	MOS	250 ns	790 mW
24AA00/ 24LC00/ 24C00	Serial EEPROM	8L DIP, SOIC, TSSOP and 5L SOT-23 packages	16 bytes \times 8 bits	Low-power CMOS	1000 ns	10 mW

Table 8.4 Commonly used RAM ICs

<i>IC No.</i>	<i>Category</i>	<i>Package</i>	<i>Organization</i>	<i>Technology</i>	<i>Access time</i>	<i>Power dissipation</i>
7489	Static RAM	16-pin DIP Package	16×4	Bipolar	33 ns	500 mW
2114	Static RAM	18-pin DIP Package	$2K \times 4$	MOS	200 ns	300 mW
74189	Static RAM	16 pin	16×4	Bipolar	50 ns	550 mW
74289	Static RAM	DIP Package	16×4	Bipolar	35 ns	250 mW
6116	Dynamic RAM	24-pin DIP, Thin Dip, SOIC and SOJ package	$2K \times 4$	CMOS	15 ns	4 W
4166	Dynamic RAM	16 pin DIP Package	16384×1	NMOS	200 ns	460 mW
2104A	Dynamic RAM	16 pin DIP Package	4096×1	MOS	150 ns	420 mW
2164		16 pin DIP Package	$64K \times 1$	MOS	450 ns	330 mW

**Fig. 8.9(a)** EPROM**Fig. 8.9(b)** RAM

8.2.4 Memory Map

As 8085 microprocessor has 16 address lines, it has an address capability of 64K ($2^{16} = 65,536$) from 0000H to FFFFH. This 64K memory will be used by EPROMs, and RAM ICs. The assignment of address to various memory ICs is known as a memory map. The memory map of 2K EPROM and a 2K static RAM is depicted in Fig. 8.10. The address of EPROM is from 0000H to 07FFFH and the address of RAM is 8000H to 87FFFH. If 2K RAM is not sufficient for programming; another 2K RAM may be connected from 8800H to 9000H as shown in Fig. 8.10 (b). Figure 8.10 (c) shows the memory map of a 4K EPROM and a 4K RAM. The address 0000H to 1000H are specified for EPROM and 4K RAM occupies addresses from 8000H to 9000H.

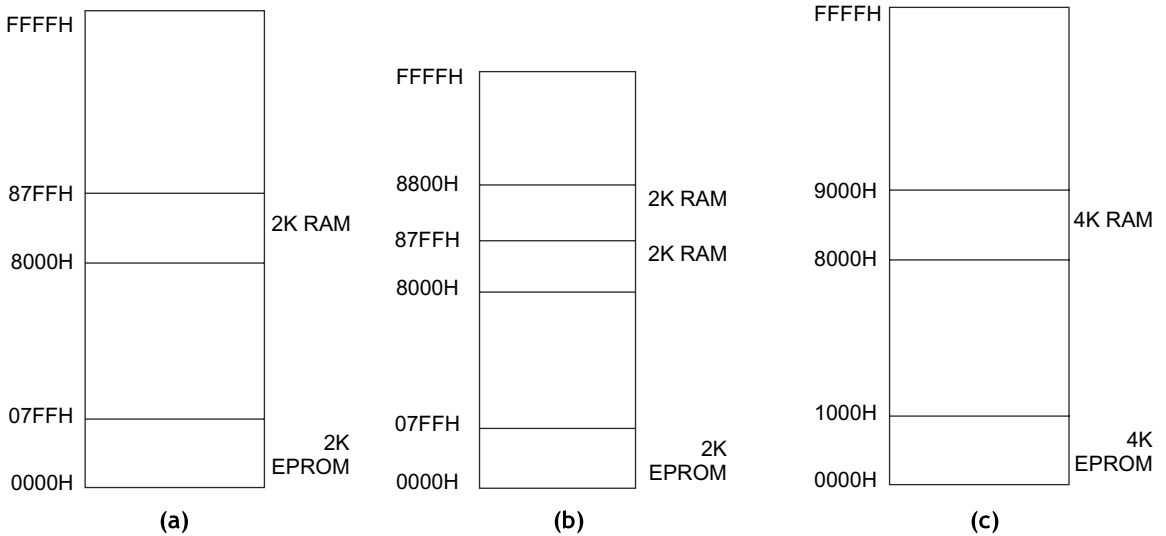


Fig. 8.10(a) Memory map of 2K EPROM and 2K RAM, **(b)** Memory map of 2K EPROM and 2 x 2K RAM **(c)** Memory map of 4K EPROM and 4K RAM

The 2K memory ICs have 11 address lines $A_{10}-A_0$, which are used to locate the memory location where data will be stored or read. The other address lines $A_{12}-A_{15}$ of the microprocessor can be used for the chip select signal. The memory map of 2K memories is given below:

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	= 0000H
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	= 07FFFH

The memory map from 0000H to 07FFFH can be expressed in terms of page analogy as given below:

0000	}	Page	0400	}	Page
00FF	}	0	04FF	}	4
0100	}	Page	0500	}	Page
01FF	}	1	05FF	}	5
0200	}	Page	0600	}	Page
02FF	}	2	06FF	}	6
0300	}	Page	0700	}	Page
03FF	}	3	07FF	}	7

8.2.5 Address Decoding

Figure 8.11 shows the address decoding technique of the 8085 microprocessor. A_0 – A_{10} are used for addressing the EPROM IC. A_{11} – A_{15} address lines are applied to a NAND gate to generate the chip select \overline{CS} . The memory map of EPROM is given below:

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	$= 0000H$
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
⋮													⋮			
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	$= 07FFH$

Figure 8.12 shows the complete memory and address decoder circuit. In this case, a 3-line to 8-line decoder can be used to select any one output. Based on inputs at A_{11} , A_{12} , A_{13} any one output of O_0 – O_7 will be active low and other output lines remain high. The output lines are connected to the chip select of memory ICs. It is depicted in Fig. 8.12 that O_0 is connected with the chip select of 2K EPROM and O_7 is connected with the 2 K RAM. The output lines and corresponding memory address capability is given in Table 8.5.

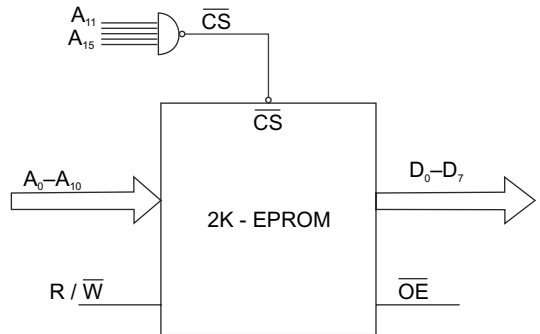


Fig. 8.11 Address decoding of 2K EPROM

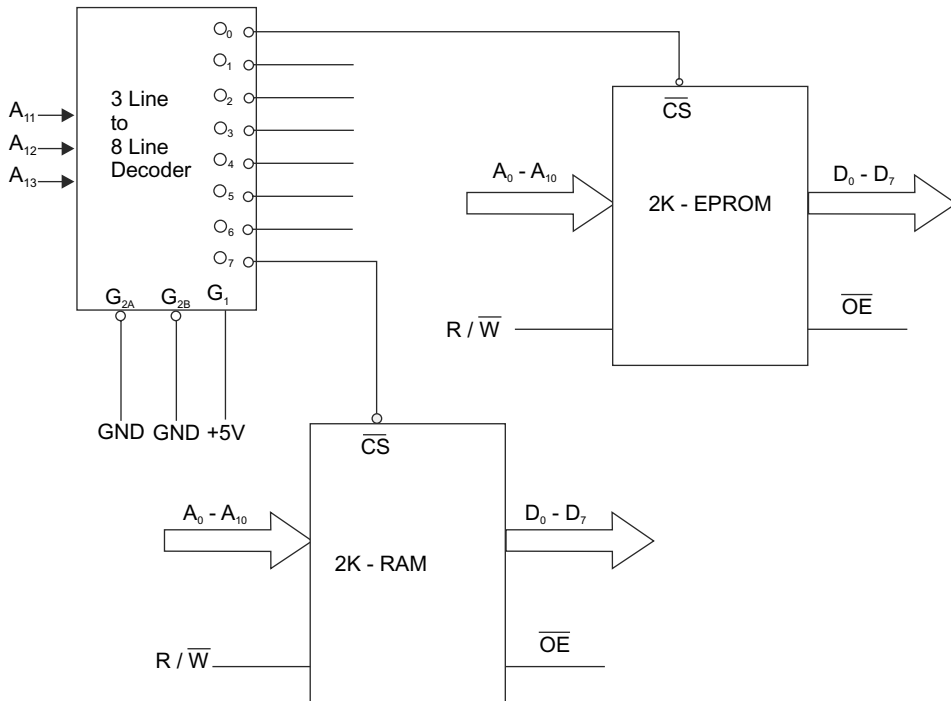


Fig. 8.12 Address decoding EPROM using decode

Table 8.5 Memory address selected by the decoder

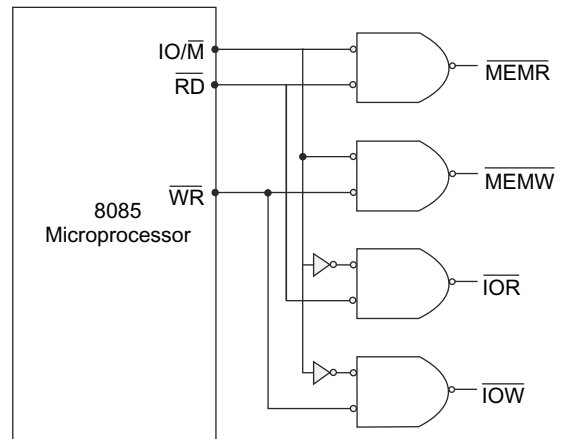
Output lines	Memory address
O ₀	0000H–07FFH
O ₁	0800H–0FFFH
O ₂	1000H–17FFH
O ₃	1800H–1FFFH
O ₄	2000H–27FFH
O ₅	2800H–2FFFH
O ₆	3000H–37FFH
O ₇	3800H–3FFFH

8.2.6 Control Signals for Memory and I/O Devices

The 8085 microprocessor has control signals \overline{RD} , \overline{WR} for read and write operations of memory and I/O devices. This IC also has a status signal IO/\overline{M} to distinguish the read/write operation of memory or I/O devices. The memory and I/O devices require the following control signals:

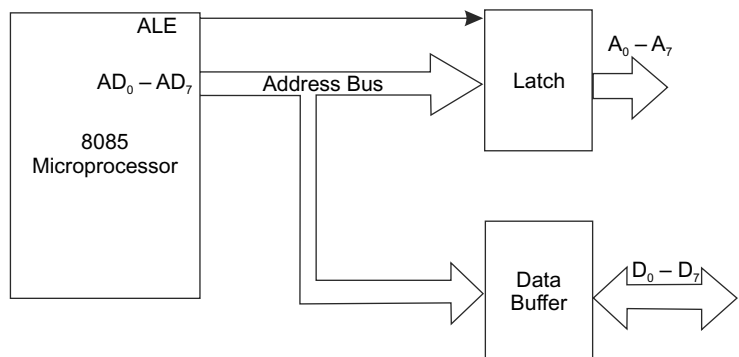
\overline{MEMR} (memory read), \overline{MEMW} (memory write), \overline{IOR} (I/O read) and \overline{IOW} (I/O write)

The above control signals are generated from \overline{RD} , \overline{WR} and IO/\overline{M} using gates as depicted in Fig. 8.13.

**Fig. 8.13** Control signal of memory and I/O read and write operations

8.2.7 Memory Interfacing to Microprocessor

The 8085 microprocessor has higher-order address bus A_8-A_{15} and lower-order address/data bus AD_0-AD_7 . The lower-order address data bus is multiplexed as address bus and data bus. During the first clock pulse of a machine cycle, the program counter releases the lower-order address in AD_0-AD_7 and higher order address in A_8-A_{15} . Then ALE signal is high; the AD_0-AD_7 can be used as lower-order address bus and not a data bus. The external latch circuit makes the difference between the address and data bus as shown in Fig. 8.14.

**Fig. 8.14** Multiplexing of lower-order address and data bus

The microprocessor communicates with various memory ICs. The interfacing between the microprocessor, memory and I/O devices through an address bus, data bus and control bus is depicted in Fig. 8.15. The address decoder is used to select proper memory and I/O devices as given Fig. 8.16.

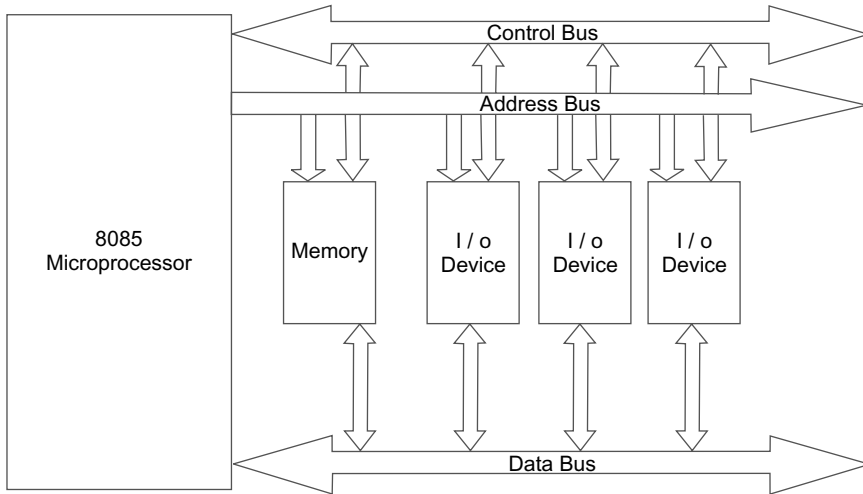


Fig. 8.15 Schematic block diagram for memory and I/O interfacing with microprocessor

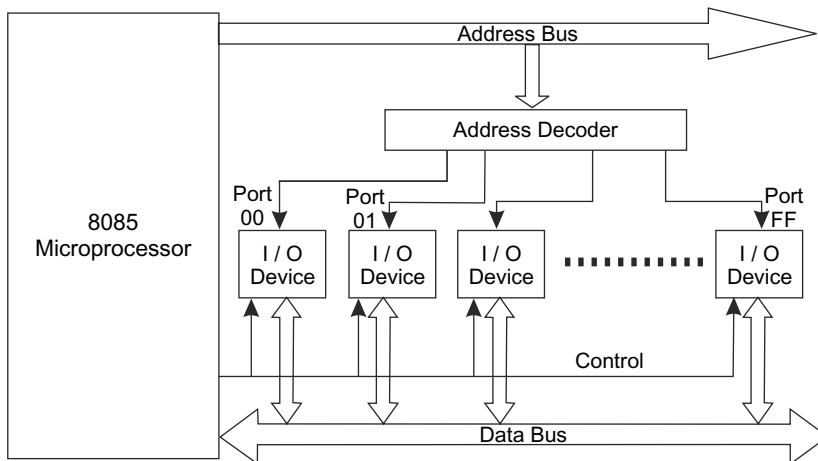


Fig. 8.16 I/O mapped I/O devices

When the address decoder is enabled and chip select signals are applied to the decoder, RAM or EPROM or I/O devices are selected. Data will be stored or read from memory devices or I/O devices. The total 64K addresses are to be assigned to memories and I/O devices. There are two types of address mapping: memory mapped I/O and I/O mapped I/O.

In some microprocessors, memory and I/O operation can be differentiated by control signals. The control signal IO/\overline{M} is available to distinguish between memory and I/O operations. When the control signal IO/\overline{M} is high, I/O operation can be performed. If the control signal IO/\overline{M} is low, memory operations will be performed. In this case, the same address can be assigned to I/O devices as well as memory locations. Generally,

two separate address spaces exist and each address space can be entirely assigned to either memory or I/O devices. This technique is known as I/O-mapped I/O.

In I/O-mapped-I/O scheme, an I/O device cannot be considered as a memory location. The I/O-mapped-I/O scheme requires special instructions like IN/OUT to access I/O devices and special signals $\overline{IO/\overline{M}}$. In this scheme, 8085 can access 256 I/O ports. In the 8085 microprocessor, this scheme requires 8-bit address lines. It requires less hardware to decode an 8-bit address. Arithmetical or logical operations cannot be directly performed with the input data.

Figure 8.16 shows the connection between the microprocessor and I/O devices. The I/O devices are identified by port addresses. The I/O read and write operations are performed by using software instructions such as IN and OUT. The I/O read and write operations are controlled by control signals \overline{IOR} and \overline{IOW} respectively. In this scheme, port addresses are varied from 00H to FFH. Therefore, 256 I/O devices may be connected with the microprocessor in I/O-mapped-I/O devices. The advantages and disadvantages of I/O-mapped-I/O are given below.

I/O-mapped-I/O has the following advantages:

- ✦ The total 256 address spaces are available for I/O devices.
- ✦ Program writing is easy as special instructions are used for I/O operations. In the 8085 microprocessor, IN and OUT instructions are usually used for data transfer with I/O devices.
- ✦ In I/O-mapped-I/O scheme, the I/O address length is usually one byte and instructions are two bytes long. Therefore, the program requires less memory and shorter execution time compared to memory-mapped I/O.
- ✦ The memory reference instructions can be easily distinguished from I/O reference instructions, which make program debugging easier.

I/O-mapped-I/O has the following disadvantages:

- ✦ One microprocessor pin must be used to distinguish between memory and I/O operations. The additional control signal, \overline{IOR} and \overline{IOW} must be generated for read and write operations.
- ✦ In data transfer with I/O devices and microprocessor, the data has to be transferred to the accumulator only to perform arithmetic or logical operations. Different addressing modes are not used in I/O-mapped-I/O.

Figure 8.17 shows the connection between microprocessor and memory. The memory location can be identified by I/O devices. The memory read and write operations are also performed by using software instructions such as LDA 9000H and STA 8000H respectively. In memory read and write operations, \overline{MEMR} and \overline{MEMW} control signals are used.

When memory location address is the same as port address of I/O devices, an I/O device will be selected for read and write operations. The memory read and write instructions employ various addressing modes. As the I/O device is considered as a memory location, this type of interfacing is known as memory-mapped-I/O. The advantages and disadvantages of memory-mapped-I/O are given below:

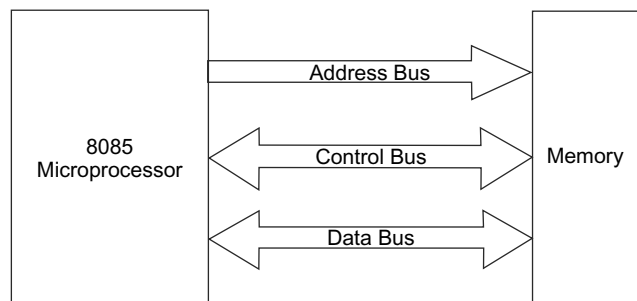


Fig. 8.17 Memory-mapped I/O scheme

Memory-mapped-I/O has the following advantages:

- ✦ The memory-mapped-I/O scheme can provide more than 256 input-output ports, as the port addresses are 16 bits.
- ✦ All the memory-related instructions can be used in read and write operations of memory-mapped I/O devices. The arithmetic and logical operations can be performed on available I/O data directly.
- ✦ CPU registers can exchange transfer of data with I/O devices directly without accumulator.
- ✦ Therefore, memory-mapped-I/O simplifies and increases speed of data transfer.

Memory-mapped I/O has the following disadvantages:

- ✦ Memory-mapped I/O scheme utilizes memory reference instructions, which are three-byte instructions and are longer than I/O instructions.
- ✦ Due to wider port address, the interface of hardware is also complicated
- ✦ The complexity of the program is large.

The method of a memory map is memory-mapped I/O. In the memory-mapped I/O scheme, actually a part of the memory space is allocated to the I/O devices. In this scheme, all the memory reference instructions can be used in the case of I/O devices and the arithmetic and logical operations are directly performed on I/O data.

In memory-mapped I/O schemes, the I/O device is treated as a memory location. This scheme does not require any special instruction. The microprocessor can access the I/O device by memory instruction. It does not require special signals. \overline{MEMR} , \overline{MEMW} signals can be used to access I/O devices. In this scheme, 8085 can access 64K memory locations or 64K I/O ports. In the 8085 microprocessor, this scheme requires 16 address lines. More hardware is required to decode a 16-bit address. Arithmetical or logical operations can be directly performed with the input/output data. The comparison between memory-mapped I/O and I/O mapped I/O is given in Table 8.6.

Table 8.6 Comparison between memory-mapped I/O and I/O-mapped I/O

Memory-Mapped I/O	I/O Mapped I/O
16-bit address	8-bit address
\overline{MEMR} memory read \overline{MEMW} memory write	\overline{IOR} I/O read \overline{IOW} I/O write
Memory related instructions MOV M,R, MOV R,M, ADDM, ANA M, SUB M, STA, LDA, LDAX, STAX	I/O related instructions IN and OUT
Data transfer between any register and I/O	Data transfer between accumulator and I/O
The memory map 64K is shared between I/Os and system memory	The I/O map is independent of the memory map; 256 input devices and 256 output devices
13 T states for execution of instructions (STA, LDA) 7 T states for execution of instructions (MOV M,R)	The IN and OUT instructions are required 10 T states for execution
More hardware is need to decode a 16-bit address	Less hardware is needed to decode 8-bit address.
Arithmetic or logical operations can be directly performed with I/O data	Not available

In any microprocessor-based system, the design of the interface is very important. In this section, the memory interface of memory and microprocessor is explained below.

The first step is to determine the number of address lines required for a memory interface. Find the

available memory address. and design the logic circuit for interfacing.

Figure 8.18 shows the 8K × 8 RAM interface to a 8085 microprocessor. For 8K memory interfaces, 13 address lines are required as $2^{13} = 8K$. The 8K memory has a starting address of 8000H. Then the end address will be 9FFFH. The following pins are used for interfacing between microprocessor and memory \overline{WR} , \overline{RD} , \overline{CS} and A_0-A_{12} .

But the pins available for memory interface on a microprocessor are \overline{WR} , \overline{RD} , IO/\overline{M} and A_0-A_{15} . A_0-A_{12} lines are directly connected with the microprocessor and the chip select signal is generated from remaining address lines $A_{13}-A_{15}$ and IO/\overline{M} .

Interfacing 4K byte RAM with microprocessor is illustrated in Fig. 8.19.

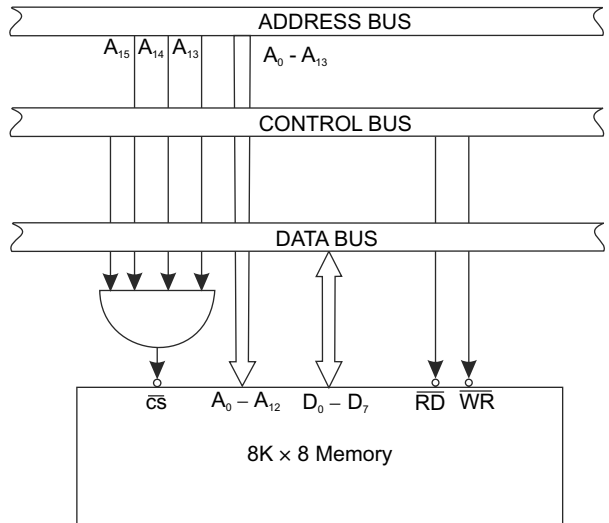


Fig. 8.18 Interfacing 8K byte RAM with microprocessor

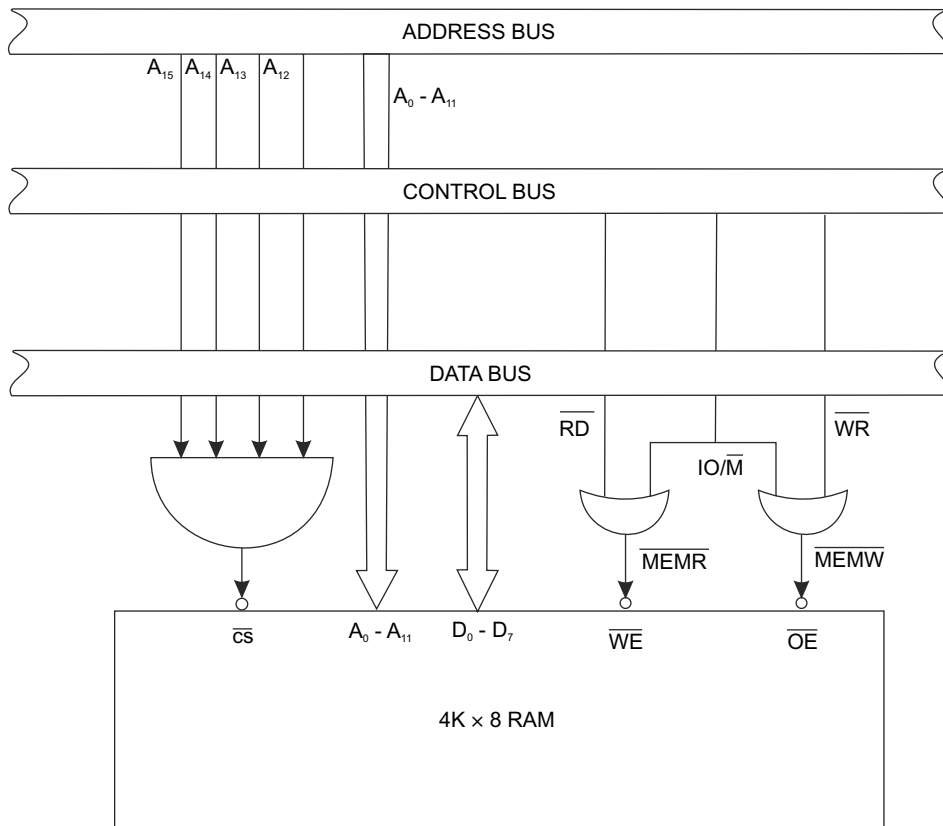


Fig. 8.19 Interfacing 4K byte RAM with a microprocessor

8.2.8 EPROM Interfacing with 8086

The most commonly used EPROM ICs are 27C256, 2716, 2732A and 2764. IC 27C256A is an erasable programmable read only memory and it is represented by 32K × 8 EPROM. The 32K is referred as the number of memory locations in the EPROM. As 1K = 1024, 32 × 1024 or 32,768 memory locations are available in the device. The 8 represents the number of bits in each memory location.

In IC27C256, 27 is the standard number for EPROMs and 256 is the number of K bits stored on the EPROM. Actually, the entire series of EPROMs is represented by 27XXX. The IC27C256 EPROM has 32K bytes of memory with 8-bit wide data bus AD₇–AD₀ and 15 address lines A₁₄ to A₀. Figure 8.20 shows the interfacing of EPROM IC27C256 with the 8086/8088 microprocessor.

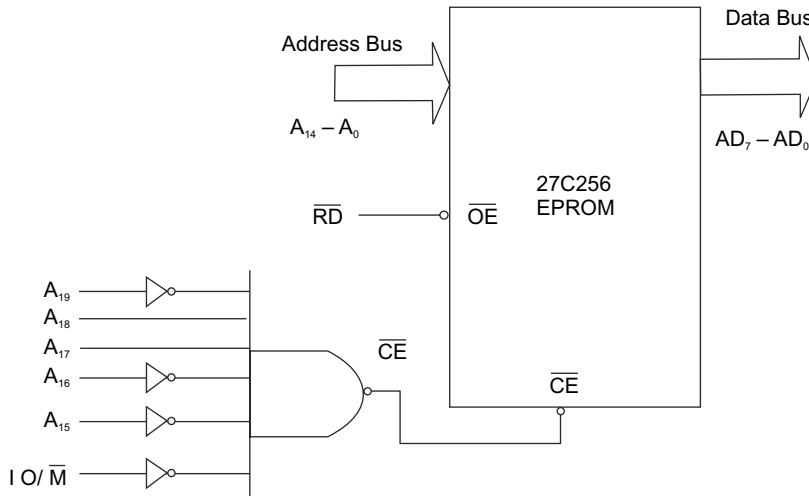


Fig. 8.20 EPROM 27C256 interfacing with 8086/8088 microprocessor

The data bus of the microprocessor AD₇–AD₀ is connected with the 8-bit data outputs of the EPROM. The address bus A₁₄–A₀ of the microprocessor is connected with the address lines A₁₄–A₀ of EPROM. The remaining address lines A₁₉–A₁₅ for 8086 processor are used to select the memory devices. The memory read signal \overline{RD} from the microprocessor is directly connected to the EPROM. The operation of EPROM is controlled by two pins such as chip enable \overline{CE} and output enable \overline{OE} . The chip enable signal is generated from IO/\overline{M} signal and address lines A₁₉–A₁₅. Therefore, a memory address decoder circuit is used to generate a \overline{CE} signal. The output enable \overline{OE} is used to enable the output buffers in the memory. This device has 8000H memory locations. When the starting address of EPROM is 60000H, the memory end address is 67FFFH. The memory map of EPROM 27C256 is given below:

	A ₁₉	A ₁₈	A ₁₇	A ₁₆	A ₁₅	A ₁₄	A ₁₃	A ₁₂	A ₁₁	A ₁₀	A ₉	A ₈	A ₇	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀
The starting address is 60000H =	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
The end address is 67FFFH =	0	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

To generate the chip enable signal, A₁₅ = 0, A₁₆ = 0, A₁₇ = 1, A₁₈ = 1 and A₁₉ = 0 as depicted in Fig. 8.20

The IC 2716 EPROM (2K × 8) has only 2 KB of memory and 11 address lines. A decoder can be used to decode the additional 9 address lines and generate a chip enable signal so that the EPROM can be placed in any 2KB section of the 1MB address space. If we assume the starting address of 2716 EPROM is FF800H, the end address will be FFFFFFH. A NAND gate and an OR gate are used to generate chip enable signal

using A_{19} - A_{11} , $\overline{IO/\overline{M}}$ and \overline{RD} from 8086/8088 microprocessor as depicted in Fig. 8.21. The memory map of IC 2716 is given below:

A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
The starting address is FF800H=	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
The end address is FFFFH =	0	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1

To generate the chip enable signal, $A_{11} = 0$, $A_{12} = 1$, $A_{13} = 1$, $A_{14} = 1$, $A_{15} = 1$, $A_{16} = 1$, $A_{17} = 1$, $A_{18} = 1$ and $A_{19} = 1$ as depicted in Fig. 8.21.

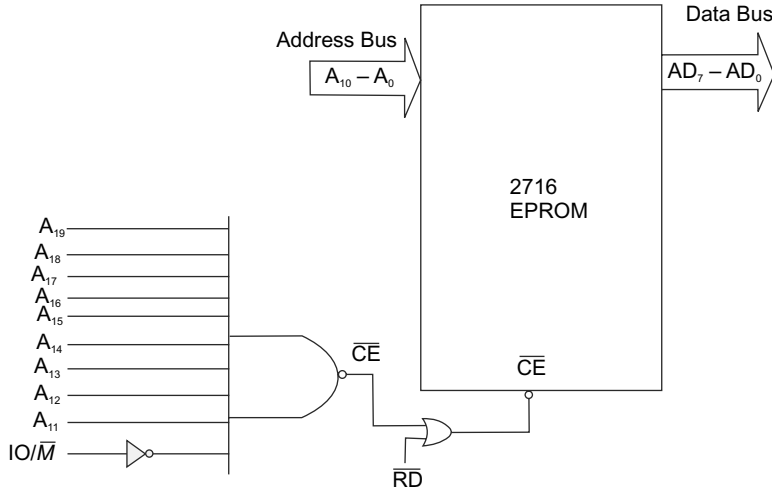


Fig. 8.21 EPROM 2716 interfacing with 8086/8088 microprocessor

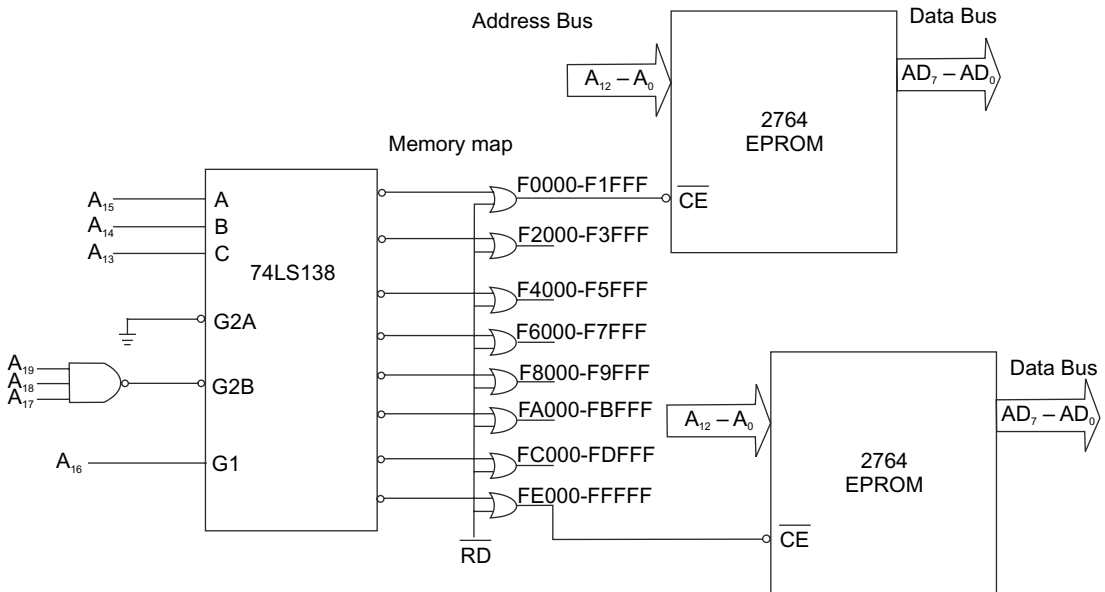


Fig. 8.22 EPROM 2764 interfacing with 8086/8088 microprocessor

In place of a NAND gate decoder, the 3-line to 8-line decoder IC74LS138 is used to select the memory devices and the 8086/8088 microprocessor can communicate with many EPROM ICs as shown in Fig. 8.22. There are three enable pins G2A, G2B and G1 which are active low, active low and active high respectively for proper operation of the decoder. Address lines A₁₂ to A₀ are directly connected to IC 2764. A₁₅ to A₁₃ are used as decoder input and each output of the decoder can select a 2764 EPROM (8K × 8). A₁₉–A₁₆ enable the decoder. The memory of each selected EPROM is depicted in Fig. 8.22.

8.2.9 Timing Diagram of Memory Read Cycle and Access Timing

Figure 8.23 shows the typical memory read cycle. The memory must respond with valid data t_{ACC} seconds after placing the memory address on the address bus. t_{ACC} is known as *address access time*. It represents the maximum amount of time that the memory requires to decode the address and place the data byte on the data bus. The address access time is in the range of 450 ns to 575 ns. t_{RD} is called as *memory read time*, which is the maximum amount of time after \overline{MEMR} becomes low and valid data will be placed on the data bus. This time is approximately 300 ns. t_{CA} is the minimum amount of time after \overline{MEMR} becomes high before a new address placed in the address bus. When minimum t_{CA} is not given in the system, a new memory address will not be placed in the address bus and the previous address's output data will be in the data bus. This is known as *bus contention*. The t_{CA} is approximately 20 ns. The detailed operation of memory read is explained below:

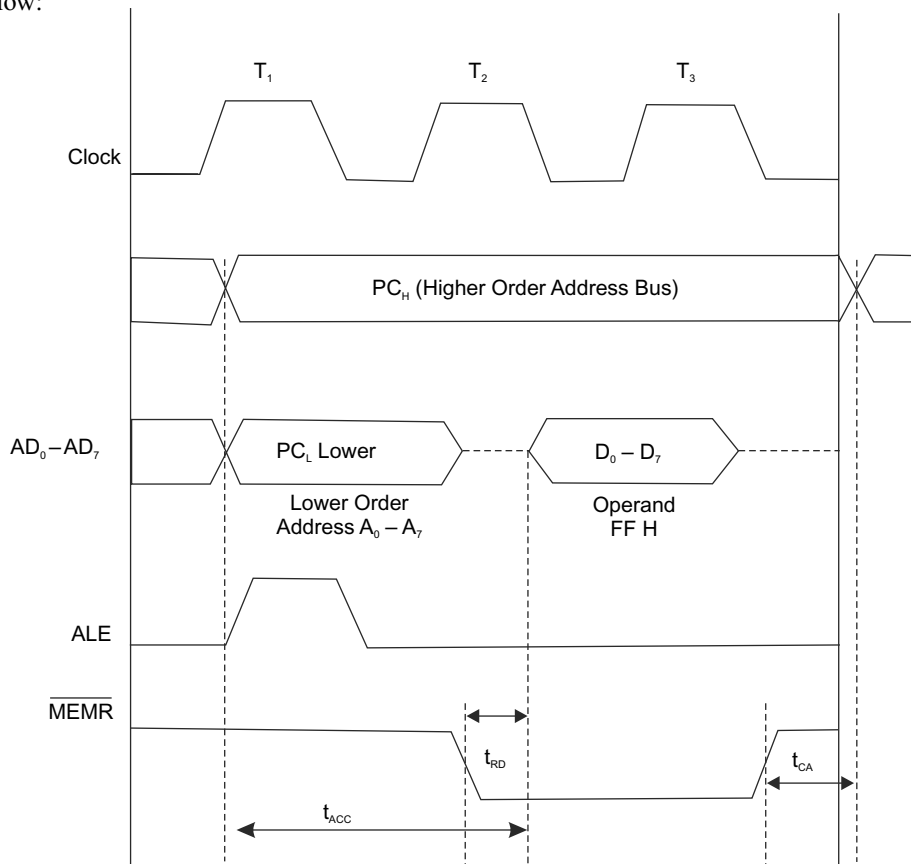


Fig. 8.23 memory read cycle

First Clock Cycle of Memory Read

- ✦ In the first clock cycle (T_1), the microprocessor places the content of program counter, 8000H, which is the address of operand on the 16-bit address bus. The 8 MSBs of the memory address, 80H are placed on the higher-order address bus, A_{15} – A_8 and 8 LSBs of the memory address, 00H are placed on the lower-order address bus, AD_7 – AD_0 .
- ✦ The microprocessor sends an Address Latch Enable (ALE) signal to go high and latch the 8 LSBs of the memory address. Then lower-order address bus is demultiplexed and the complete 16-bit memory address is available in the subsequent clock cycles to get the operand from memory location 8000H.
- ✦ The status signals $IO/\overline{M} = 0$, $S_0 = 0$ and $S_1 = 1$ to identify the memory read operation.

Second Clock Cycle of Memory Read

- ✦ The low-order bus AD_7 – AD_0 is ready to accept the operand from memory. The microprocessor sends the control signal $\overline{MEMR} = 0$ to enable memory and the program counter is incremented by 1 to 8001H. After that, the operand from the memory location 8000H is placed on the data bus.

Third Clock Cycle of Memory Read

- ✦ During T_3 , the microprocessor reads the operand. \overline{MEMR} signal becomes high during T_3 and the memory is disabled.
- ✦ The microprocessor also places the operand in any register.

8.2.10 Timing Diagram of Memory Write

In a memory write operation, the microprocessor sends data from the accumulator or any general-purpose register to the memory. The timing diagram of a memory write cycle is depicted in Fig. 8.24. The memory write cycle is similar to memory read cycle, but there are differences on status signals. The status signals $S_0 = 1$ and $S_1 = 0$ and write \overline{WR} is low during T_2 of the machine cycle which indicates that the memory write operation is to be performed.

During T_2 of the machine cycle M_2 , the low-order address bus AD_0 – AD_7 is not disabled as the data is to be sent out to the memory, which is placed on the low-order address bus. When \overline{MEMW} comes high in T_3 of machine cycle M_2 , the memory write operation is terminated. The following instructions use the memory write cycle: MOV M, B; MOV M, A and STA 8000 H, etc.

The memory write cycle begins by placing a valid address on the address bus A_0 – A_{15} . Valid data is also placed on the data bus D_0 – D_7 early in the memory write cycle which is known as *data write set-up time*. When \overline{MEMW} becomes low, the memory cycle is started and \overline{MEMW} will be low until writing operation is completed.

t_{AW} is the minimum amount of time that valid address will be held on the data bus before \overline{MEMW} becomes high. Generally, t_{AW} is 450 ns. t_{DW} is the minimum amount of time that valid data will be held on the bus before \overline{MEMW} becomes high and it is approximately 200 ns.

8.2.11 Timing Diagram of I/O Read Cycle

In an I/O read operation, the microprocessor reads the data from a specified input port or input device. The I/O read operation is similar to memory read cycle except for the control signal IO/\overline{M} . In a memory ready cycle IO/\overline{M} is low but IO/\overline{M} is high in case of an I/O read cycle operation because signal IO/\overline{M} goes high in case of I/O read.

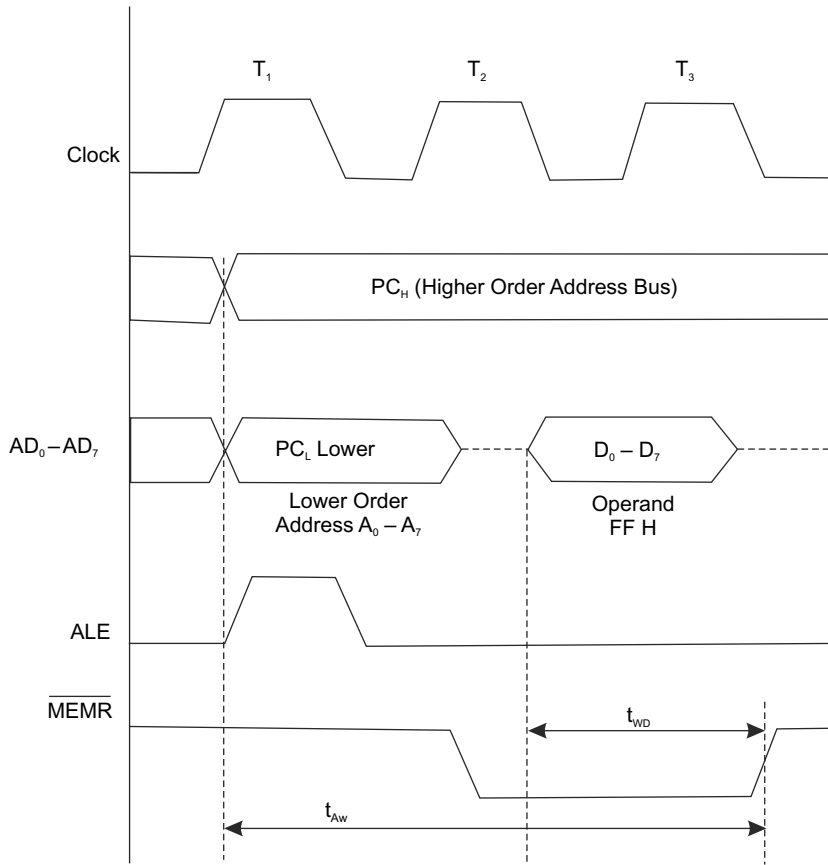


Fig. 8.24 Memory write cycle

The timing diagram of an I/O read operation is shown in Fig. 8.25. In this case, the address on the A-bus is for an input device. As an I/O device or I/O port the address is only 8 bits long, the address of an I/O device or I/O port is duplicated on both higher-order address bus A_8-15 and lower-order address bus AD_0-AD_7 .

For an I/O read operation, the IN instruction is used. One example is IN 00. This is a two-byte instruction. The code of this instruction is DB, 00 where DB is for IN and 00 is the input port address.

This instruction requires three machine cycles for execution. The first machine cycle is the *opcode fetch cycle*, and the second machine cycle is a *memory read cycle* to read the address of input device or input port. In the third machine cycle, the I/O read operation is performed, meaning the data is to be read from the input device or input port. After execution of this instruction, the data is placed in the accumulator. The opcode fetch cycle and memory read cycle are exactly similar to MVI C, FF H instruction. Figure 8.25 shows the machine cycle M_3 of I/O read operation and it is explained below:

T_1 State of M_3

- ✦ CPU places the address of I/O port or input-output peripheral devices.
- ✦ ALE signal is high.
- ✦ IO/\overline{M} becomes high to perform I/O operation.

T_2 State of M_3

- ✦ \overline{RD} is low for read operation.

T_3 State of M_3

- ✦ CPU reads data from I/O devices and places in Register A through a data bus.
- ✦ \overline{RD} Signal becomes high as I/O read operation has been completely performed.

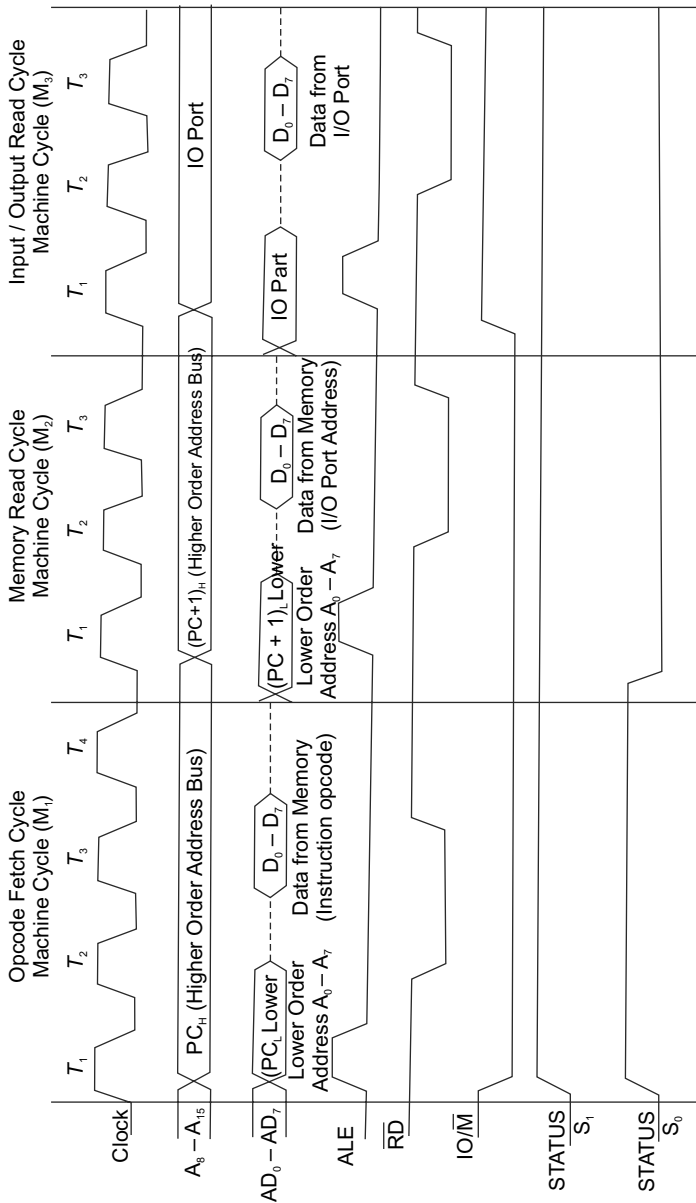


Fig. 8.25 Timing diagram of IN port address

8.2.12 Timing Diagram of I/O Write Cycle

The microprocessor sends the content of the accumulator to an I/O port or I/O device in an I/O write cycle. The operations of an I/O write cycle are similar to a memory write cycle. But the difference between memory write and I/O write cycle is that IO/\overline{M} becomes high in case of an I/O write cycle. When IO/\overline{M} is high, the microprocessor locates the address of any output device or an output port. The address of an output device or an output port is duplicated on both higher-order address bus A_8-A_{15} and lower-order address bus AD_0-AD_7 .

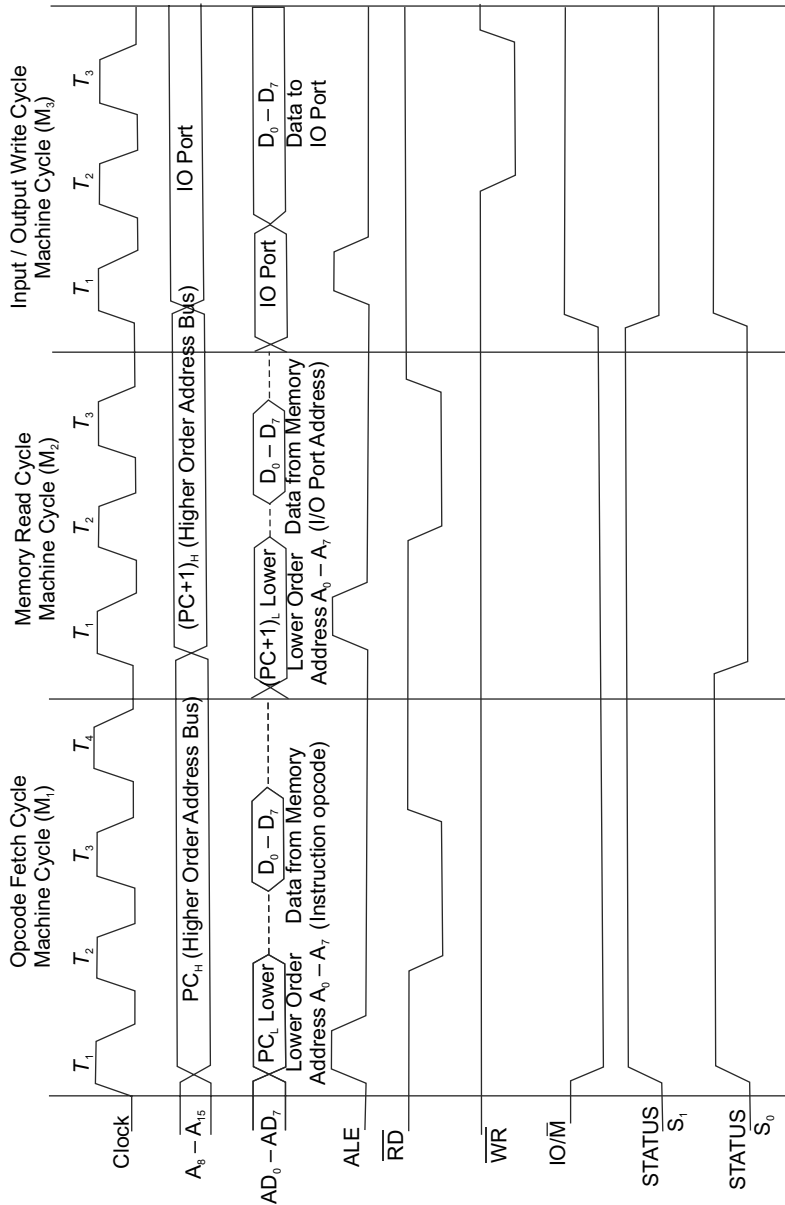


Fig. 8.26 Timing diagram of OUT port address

The OUT instruction is used for an I/O write operation. This is a two-byte instruction and it requires three machine cycles as depicted in Fig. 8.26. The first machine cycle is for opcode fetch operation and the second machine cycle is a memory read cycle for reading the address output device or output port from the memory. In the next third machine cycle, data will be written in the output device or output port. In other words, data is to be send to the I/O device. The third machine cycle is explained below:

T_1 State of M_3

- ✦ CPU places the address of I/O port or input-output peripheral devices.
- ✦ ALE signal is high.
- ✦ IO/\overline{M} signal is also high to perform I/O operation.

T_2 State of M_3

- ✦ \overline{WR} becomes low for write operation.

T_3 State of M_3

- ✦ CPU places the content of Register A in a data bus.
- ✦ Then it writes data to the I/O port.
- ✦ \overline{WR} signal becomes high as I/O read operation has been completed.

8.3 INTERRUPTS OF THE 8085 MICROPROCESSOR

Interrupts is the facility provided by the microprocessor to communicate with the outside environment, and the microprocessor can divert its operation based on priority. The interrupts can be used for various applications in different environments.

An *interrupt* is a process where an external device can get the attention of the microprocessor. The process starts from the I/O device and it is asynchronous-type data transfer. Figure. 8.27 shows the interrupt-driven data transfer. The microprocessor can initiate the data transfer after getting an interrupt signal from I/O device. The microprocessor can scan the interrupt pin on every machine cycle. When the interrupt signal is present, microprocessor suspends its present operation after storing the current status in the microprocessor so that the microprocessor can restart the suspended work again from where it left. Therefore, the stack is used to store the current status. Then the microprocessor provides services the interrupt request by executing interrupt service routine.

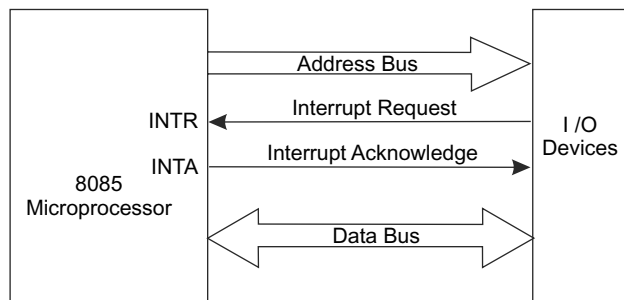


Fig. 8.27 Interrupt driven data transfer for an I/O device

An interrupt is considered to be an emergency signal which may be serviced. The microprocessor may respond to it as soon as possible. When the microprocessor receives an interrupt signal, it suspends the currently executing program and jumps to an Interrupt Service Routine (ISR) to respond to the incoming interrupt. Each interrupt will most probably have its own ISR. Figure 8.28 shows the interrupt execution. Responding to an interrupt may be immediate or delayed depending on whether the interrupt is maskable or

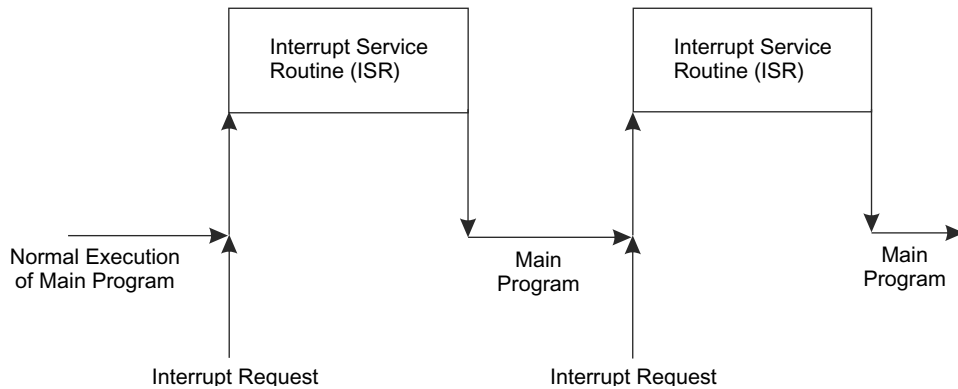


Fig. 8.28 Interrupt execution

nonmaskable and whether interrupts are being masked or not. There are two different ways of redirecting the execution to the ISR depending on whether the interrupt is vectored or nonvectored. In a vectored interrupt, the address of the subroutine is already known to the microprocessor. In case of a nonvectored interrupt, the I/O devices will have to supply the address of the subroutine to the microprocessor.

In any microprocessor-based system, I/O devices can use interrupt-driven data transfer. A microprocessor may have one interrupt level and more than one I/O devices share the interrupt level. The microprocessor may have many interrupt levels and several I/O devices. Each device can be connected to an interrupt level. When several I/O devices are connected with a single interrupt level, these devices are reconnected with an 8259 interrupt controller. Using an 8259 interrupt controller, only eight I/O devices can be connected. When more than eight I/O devices are connected to the 8085 microprocessor, more 8259 interrupt controllers are connected in cascade. If each I/O device is connected to an independent interrupt level, the microprocessor should have several interrupt levels. In this case, the number of I/O devices must be less than the number of interrupt levels.

8.3.1 Classification of Interrupts

Interrupts can be classified into two types: maskable interrupts and nonmaskable interrupts. The *maskable interrupts* can be delayed or rejected but the *nonmaskable interrupts* cannot be delayed or rejected. Interrupts can also be classified into vectored and nonvectored interrupts. In *vectored interrupts*, the address of the service routine is hard-wired but in *nonvectored interrupts*, the address of the service routine needs to be supplied externally by the device. All types of interrupts are explained in the 8085 interrupts section.

8.3.2 The 8085 Interrupts

When a device interrupts, it actually wants the microprocessor to give a service, which is equivalent to asking the microprocessor to call a subroutine. This subroutine is known as Interrupt Service Routine (ISR). Figure 8.29 shows the 8085 microprocessor interrupts and their detailed operations are depicted in Fig. 8.30. The 'EI' instruction is a one-byte instruction and is used to enable the nonmaskable interrupts. The 'DI' instruction is a one-byte instruction and is used to disable the nonmaskable interrupts.

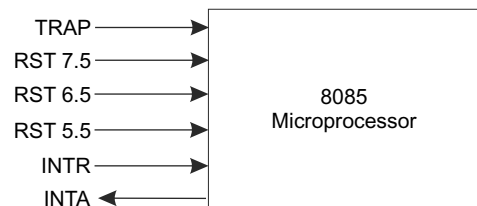


Fig. 8.29 8085 microprocessor interrupts

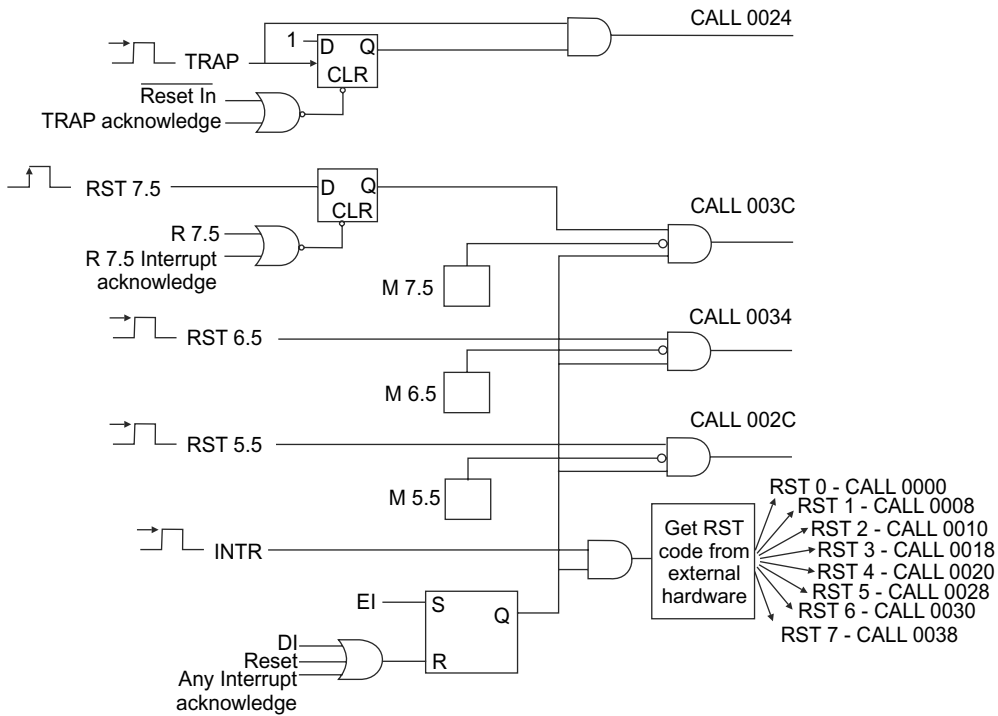


Fig. 8.30 Interrupts of 8085 microprocessor

The 8085 microprocessor has a single nonmaskable interrupt and the nonmaskable interrupt is not affected by the value of the interrupt enable flip-flop. The processor has five hardware interrupts such as INTR, RST 5.5, RST 6.5, RST 7.5, and TRAP. They are presented below in the order of their priority from lowest priority to highest priority:

INTR is a maskable interrupt. When the interrupt occurs, the processor fetches from the bus one instruction, usually one of EI and DI instructions.

The syntax for the interrupt instruction is RST *n*, where *n* is equal to 0 to 7. Any one of the 8 RST instructions (RST₀–RST₇) can be executed at a time. During execution, the processor saves the current program counter into stack and branches to the memory location. The vector address of this software interrupt is calculated from $N \times 8$, where *N* is a 3-bit number from 0 to 7 supplied with the RST instruction. For example, the vector address of RST 3 is $3 \times 8 = 24_{10} = 0018H$. Table 8.7 shows the vector address of RST instructions.

The 8085 recognize 8 RESTART instructions: RST₀–RST₇. Each of these would send the execution to a predetermined hard-wired memory location.

Table 8.7 RST *n* interrupts vector address

Restart instruction	Hex code	Equivalent Vector Address
RST0	C7	CALL 0000H
RST1	CF	CALL 0008H
RST2	D7	CALL 0010H

(Contd.)

(Contd.)

RST3	DF	CALL 0018H
RST4	E7	CALL 0020H
RST5	EF	CALL 0028H
RST6	F7	CALL 0030H
RST7	FF	CALL 0038H

When any of the above instructions is executed, a CALL instruction to the specified address is executed. The content of a program counter is saved in the stack and the control moves to the specified address. The vector addresses of the instructions are 8 bytes apart. Therefore, 8 bytes of instructions can be stored at any vector address. Generally, a 3-byte JMP instruction is stored for the corresponding ISR and program control is transferred to the desired ISR. CALL instruction is a 3-byte instruction. The processor calls the subroutine, the address of which is specified in the second and third bytes of the instruction. The INTR input is the only nonvectored interrupt. INTR is maskable using the EI/DI instruction pair.

RST 5.5 is a maskable interrupt. When this interrupt is received, the processor saves the contents of the Program Counter (PC) register into the stack and branches to 002CH address.

RST 6.5 is a maskable interrupt. When this interrupt is received, the processor saves the contents of the Program Counter (PC) register into the stack and branches to 0034H address.

RST 7.5 is a maskable interrupt. When this interrupt is received, the processor saves the contents of the Program Counter (PC) register into the stack and branches to 003CH address.

TRAP is a nonmaskable interrupt. It does not need to be enabled, as it cannot be disabled. It has the highest priority amongst all interrupts. This is edge and level sensitive. This TRAP signal needs to be high and stay high for recognition. Once it is recognized, it does not need to be recognized again until it becomes low and then high again. Generally, TRAP is used for power failure and emergency shutdown. When this interrupt is received, the processor saves the contents of the PC register into the stack and branches to 0024H address. Figure 8.30 shows the TRAP interrupts circuit with other interrupts. The positive edge of TRAP input signal sets the D flip-flop and Q becomes '1'. Then AND gate output will be '1' for the duration of high level of TRAP input. Jump to the vector memory location 0024H as the starting address of an interrupt service routine for TRAP interrupt is 0024H.

RST 5.5, RST 6.5, RST 7.5 are all automatically vectored. RST 5.5, RST 6.5, and RST 7.5 are all maskable. TRAP is the only nonmaskable interrupt in the 8085. TRAP is also automatically vectored. All maskable interrupts can be enabled or disabled using EI and DI instructions. RST 5.5, RST6.5 and RST7.5 interrupts can be enabled or disabled individually using SIM instruction. RST5.5 is a maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into the stack and branches to 2CH (hexadecimal) address.

Table 8.8 8085 interrupts

<i>Interrupt</i>	<i>Maskable</i>	<i>Vectored</i>
INTR	Yes	No
RST 5.5	Yes	Yes
RST 6.5	Yes	Yes
RST 7.5	Yes	Yes
TRAP	No	Yes

8.3.3 Interrupt Vectors and Vector Table

An interrupt vector is a pointer in which the Interrupt Service Routine (ISR) is stored in memory. All vectored interrupts are mapped onto a memory area called the Interrupt Vector Table as given in Table 8.7 and Table 8.9. The interrupt vector table is generally located in memory page 00 (0000H–00FFH). The purpose of the interrupt vector table is to hold the vectors that redirect the microprocessor to the right place when an interrupt appears.

For example, assume a device interrupts the microprocessor using the RST 7.5 interrupt line. As the RST 7.5 interrupt is a vectored-type interrupt, the microprocessor should know in which memory location it jumps using a call instruction to get the ISR address. RST7.5 is known as call 003CH to microprocessor. Microprocessor jumps to 003C H memory location and it also get a JMP instruction to the actual ISR address. After that the microprocessor jumps to the ISR location.

Maskable/Vectored Interrupts of 8085

The 8085 has 4 masked/vectored interrupt inputs. RST 5.5, RST 6.5, RST 7.5 are all maskable. They are automatically vectored according to Table 8.9. The vectors for these interrupt fall in between the vectors for the RST instructions. For this, they have names like RST 5.5 (RST 5 and a half).

Table 8.9 Maskable interrupts and vector locations

<i>Interrupt</i>	<i>Vector Address</i>
RST 5.5	002CH
RST 6.5	0034H
RST 7.5	003CH

The spaces between software interrupt RST 5 and hardware interrupt RST 5.5 is 4 bytes. In this space 3-byte JMP instruction is written. The vector address of the hardware interrupts is spaced 8 bytes. Usually, a three-byte jump instruction is written in this space. The hardware interrupt signals are directly vectored to the address specified in the interrupt vector table. These interrupts are called *vector interrupts*.

Masking RST 5.5, RST 6.5 and RST 7.5

RST 5.5, RST 6.5 and RST 7.5 interrupts are masked at two levels through the interrupt enable flip-flop and the EI/DI instructions. The interrupt enable flip-flop controls the whole maskable interrupt process through individual mask flip-flops that control the availability of the individual interrupts. These flip-flops control the interrupts individually. The 8085 maskable/vectored interrupt should process the following steps:

Step 1 The interrupt process must be enabled using the EI instruction.

Step 2 The 8085 should check for an interrupt during the execution of every instruction.

Step 3 When there is an interrupt and the interrupt is enabled using the interrupt mask, the microprocessor will complete the executing instruction, and then reset the interrupt flip flop.

Step 4 Thereafter, the microprocessor executes the CALL instruction which sends the execution to the appropriate memory location according to the interrupt vector table.

Step 5 When the microprocessor executes the call instruction, it saves the address of the next instruction on the stack.

Step 6 The microprocessor jumps to the specific service routine. The Interrupt Service Routine (ISR) must incorporate the instruction EI to re-enable the interrupt process.

Step 7 At the end of the service routine, the RET instruction returns the execution to where the program was interrupted.

Nonvectored Interrupt

The 8085 nonvectored interrupt process are completed by the following steps:

Step 1 The interrupt process should be enabled using the EI instruction.

Step 2 The 8085 checks for an interrupt during the execution of every instruction.

Step 3 If INTR is high, microprocessor completes current instruction, disables the interrupt and sends INTA (Interrupt acknowledge) signal to the device that interrupted.

Step 4 INTA allows the I/O device to send an RST instruction through data bus.

Step 5 After receiving the INTA signal, the microprocessor saves the memory location of the next instruction on the stack and the program is transferred to 'call' location (ISR Call) specified by the RST instruction

Step 6 Microprocessor performs the ISR. ISR must include the 'EI' instruction to enable further interrupt within the program.

Step 7 RET instruction at the end of the ISR allows the microprocessor to retrieve the return address from the stack and the program is transferred back to where the program was interrupted.

The 8085 recognizes 8 RESTART instructions: RST0–RST7. Each of these would send the execution to a predetermined hard-wired memory location as given in Table 8.7. The syntax for the interrupt instruction is RST n , where n is equal to 0 to 7. The restart sequence is made up of three machine cycles.

✓ **In the first Machine Cycle**

- The microprocessor sends the INTA signal.
- When INTA is active low, the microprocessor reads the data lines expecting to receive, from the interrupting device, the opcode for the specific RST instruction.

✓ **In the Second and Third Machine Cycles**

- The 16-bit address of the next instruction is saved on the stack.
- Then the microprocessor jumps to the address associated with the specified RST instruction.

The opcode is simply a collection of bits. The external device produces the opcode for the appropriate RST instruction. So, the device needs to set the bits of the data bus to the appropriate value in response to an INTA signal.

The timing diagram of RST 5 is shown in Fig. 8.31. Consider that the RST 5 is stored at the 8000H memory location. This instruction has three machine cycles. In the machine cycle M_1 , the opcode of RST 5 instruction is fetched. In the opcode fetch cycle, opcode will be read and decoded. As RST is an interrupt instruction, it is needed to execute its service routine and after execution it must return back to the next memory location of RST5 instruction. This return address must be stored in the stack.

The second machine cycle is M_2 , which is called memory write cycle. In this machine cycle, the higher-order byte of the program counter will be stored in the stack. For this, the content of the stack pointer is decremented by one and 16-bit content is placed on the address bus. Then the higher-order byte of the program counter is stored in that memory location.

In the third machine cycle, the content of the stack pointer is also decremented by one and again placed on the address bus. Thereafter, the lower order byte of the program counter can be stored in that memory location.

Figure 8.32 shows an interrupt acknowledge cycle for CALL instruction. M_2 and M_3 machine cycles are needed to call the 2-byte address of the CALL instruction. Then the content of program counter are stored in memory writes cycles M_4 and M_5 . After that a new instruction cycle starts.

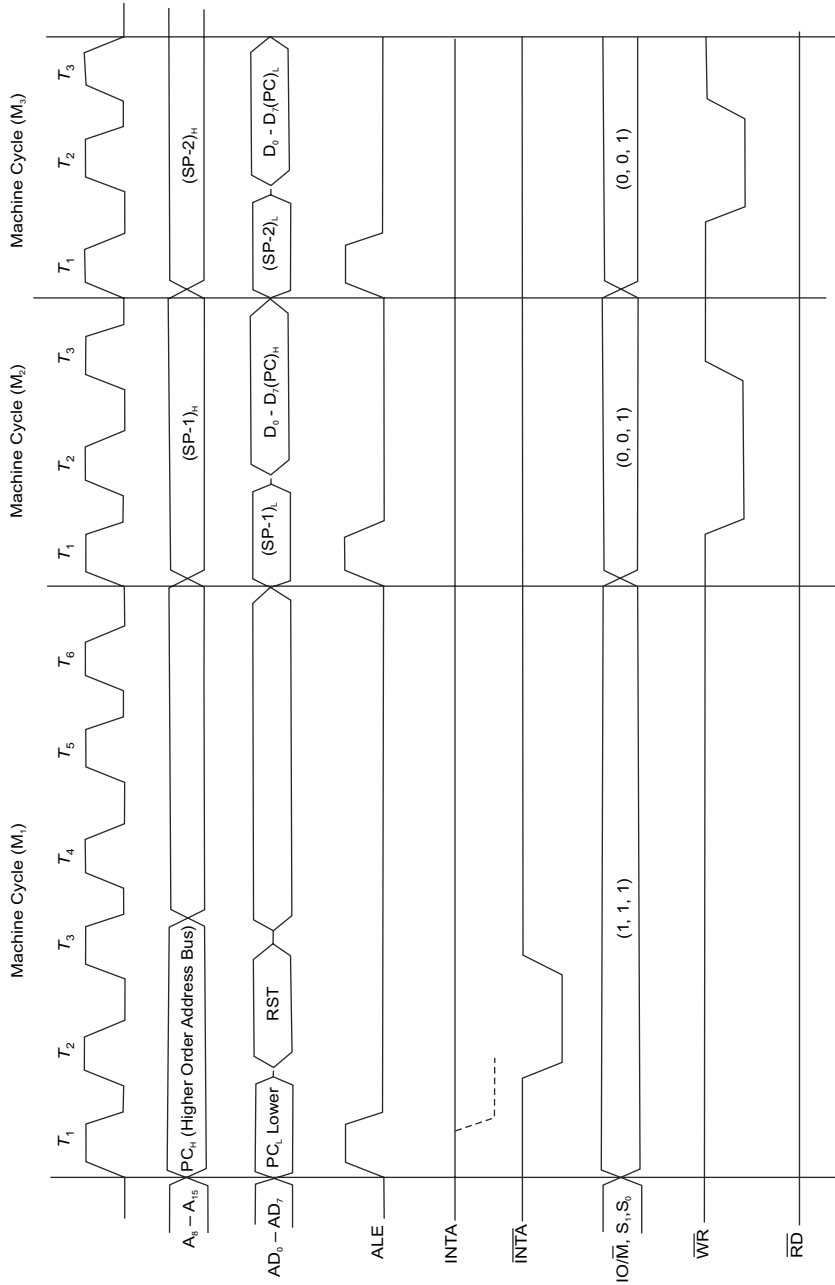


Fig. 8.31 Timing Diagram of RST 5 instruction

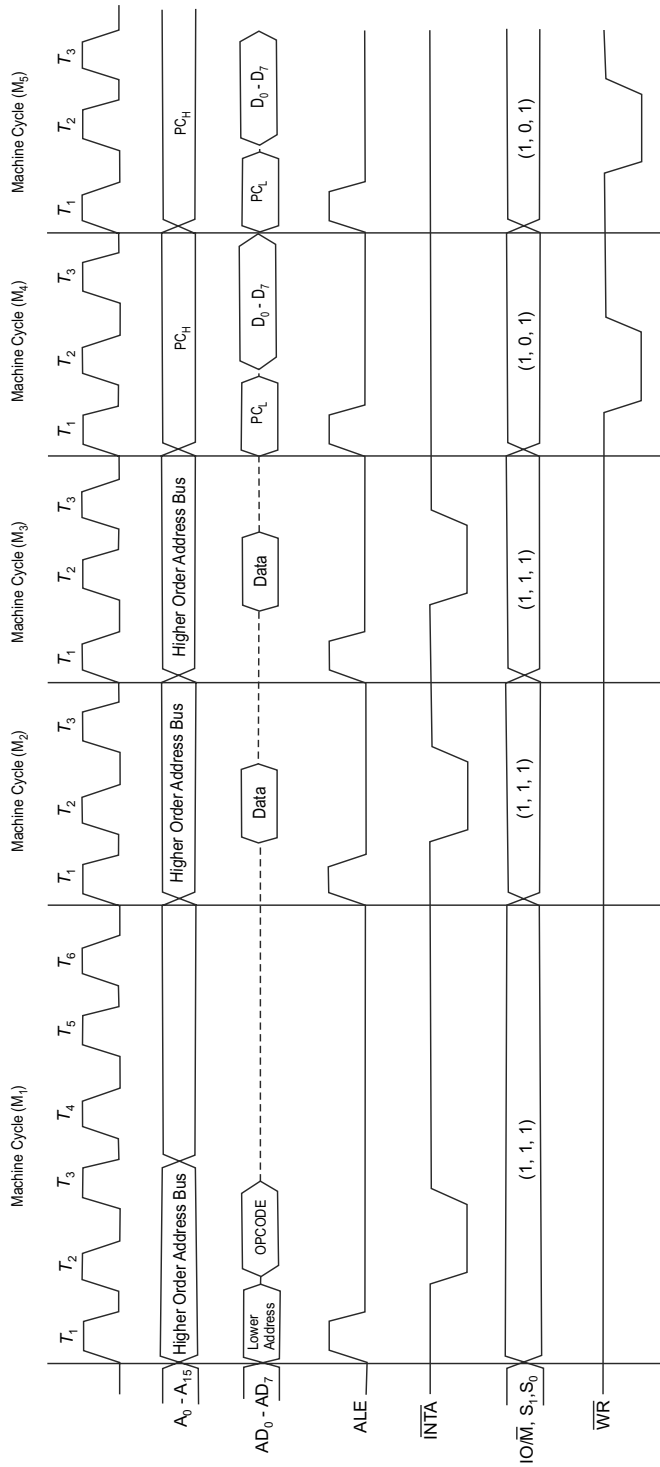


Fig. 8.32 Timing Diagram of INTA machine cycle and execution of CALL instruction

Figure 8.33 shows the generation of RST 5 opcode. RST 5 opcode is 11101111 (EFH). If INTR is acknowledged by the microprocessor, \overline{INTA} signal becomes low. The RST 5 is gated into the system bus. Then microprocessor saves the content of program counter in the stack and jumps to the memory location 0028H. In this address, the interrupt service starts and ends with RET instruction. After execution of RET instruction, the processor restores the saved address in the stack to the program counter so that normal execution of main program can be started.

In the first machine cycle of the RST operation, the microprocessor activates the INTA signal. This signal will enable the tri-state buffers, which will place the value EFH on the data bus. Therefore, it sends the microprocessor the RST 5 instruction. The RST 5 instruction is exactly equivalent to CALL 0028H.

Triggering Levels RST 7.5 is positive edge sensitive. When a positive edge appears on the RST 7.5 line, logic '1' is stored in the flip-flop as a 'pending' interrupt. Since the value has been stored in the flip-flop, the line does not have to be high when the microprocessor checks for the interrupt to be recognized. The line must go to zero and back to one before a new interrupt is recognized.

RST 6.5 and RST 5.5 are level sensitive. The interrupting signal must remain present until the microprocessor checks for interrupts.

TRAP is edge triggered as well as level triggered. Therefore, TRAP must be high until this is acknowledged. Figure 8.30 shows the TRAP interrupt. When the interrupt is acknowledged, the flip-flop of TRAP interrupt will be cleared so that the next new TRAP interrupt can be entertained. The summary of all 8085 interrupts is given in Table 8.10.

Table 8.10 8085 interrupts

Interrupt	Maskable	Masking method	Vectored	Memory	Triggering method
INTR	Yes	DI/EI	No	No	Level sensitive
RST 5.5	Yes	DI/EI SIM	Yes	No	Level sensitive
RST 6.5	Yes	DI/EI SIM	Yes	No	Level sensitive
RST 7.5	Yes	DI/EI SIM	Yes	Yes	Edge sensitive
TRAP	No	None	Yes	No	Level and edge sensitive

8.3.4 Interrupt Instructions

The Enable interrupts (EI) and Disable Interrupts (DI) instructions authorize the microprocessor to allow or reject interrupts. In case of EI, the interrupts will be enabled following the completion of the next instruction following the EI. This allows at least one more instruction like JMP or RET to be executed before the microprocessor allows itself to be interrupted again. In the DI, the interrupts are disabled immediately and no flags are affected.

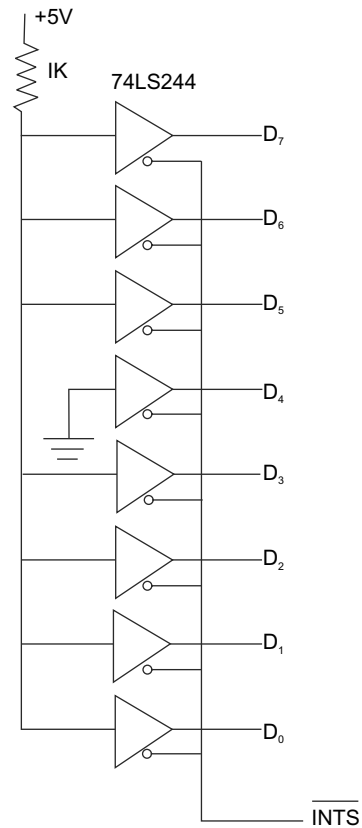


Fig. 8.33 Generation of RST 5 opcode

The Read Interrupt Mask (RIM) and Set Interrupt Mask (SIM) instructions are used to provide interrupt services of the 8085 microprocessor with the help of the Serial Input Data (SID) and Serial Output Data (SOD) pins on the device. The discussion of the above two instructions are as follows:

SIM Instruction

Sometimes it is required to enable some selected interrupts and disable some other interrupts. The selected interrupts are enabling through the set interrupt mask. The accumulator (A) is loaded with the specified mask bits. The SIM instruction reads the accumulator content and enables and disables the interrupts.

The individual masks for RST 5.5, RST 6.5 and RST 7.5 are manipulated using the SIM instruction. This instruction takes the bit pattern in the accumulator. The SIM instruction reads the accumulator content and enables or disables the specific interrupts. Figure. 8.34 shows the accumulator content for SIM instruction.

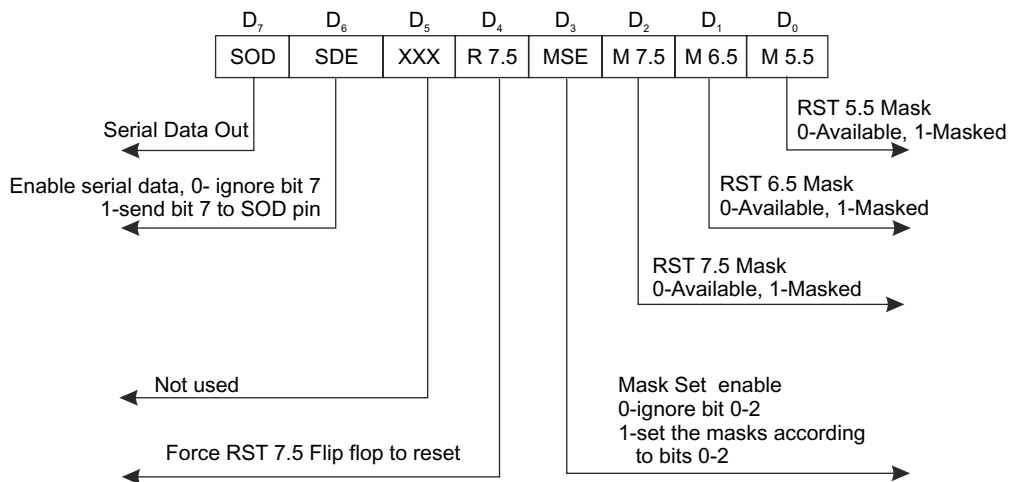


Fig. 8.34 Accumulator content for SIM

✓ **RST Masks Bits D₀, D₁, and D₂** Bit D₀ is the mask for RST 5.5, bit D₁ is the mask for RST 6.5 and bit D₂ is the mask for RST 7.5. If the mask bit is 0, the interrupt is available. If the mask bit is 1, the interrupt is masked. If bits D₀ or D₁ are set to 1, a signal applied to their respective pins causes no action. When D₀ or D₁ are set to 0, their respective bits will be visible through the RIM instruction, and the call to the interrupt vector will occur. In the case of bit D₂, the RIM instruction can indicate that RST 7.5 interrupt is pending, and an automatic call will not occur.

✓ **Mask Set Enable Bit D₃** Bit D₃ is Mask Set Enable (MSE) and this is an enable for setting the mask. If it is set to 0, the mask is ignored and the old settings remain. If it is set to 1, the new settings are applied. The SIM instruction is used for multiple purposes and not only for setting interrupt masks. It is also used to control functionality such as serial data transmission. Therefore, bit D₃ is necessary to tell the microprocessor whether or not the interrupt masks should be modified.

✓ **RST 7.5 RESET BIT D₄** Bit D₄ is RST 7.5. The RST 7.5 interrupt is the only 8085 interrupt that has memory. If a signal on RST 7.5 arrives while it is masked, a flip-flop will remember the signal. When RST 7.5 is unmasked, the microprocessor will be interrupted even if the device has removed the interrupt signal. This flip-flop will be automatically reset when the microprocessor responds to an RST 7.5 interrupt. Bit

4 of the accumulator in the SIM instruction allows explicitly resetting the RST 7.5 memory even if the microprocessor did not respond to it.

- ✓ **Undefined Bit D_5** Bit D_5 is not used by the SIM instruction.
- ✓ **SOD Enable Bit D_6** Bit D_6 is used for serial output data enable.
- ✓ **Serial Output Data Bit D_7** Bit D_7 is used for serial output data. The SIM instruction is used for serial data transmission. When the SIM instruction is executed, the content of bit D_7 of accumulator will be output on the SOD line.

Example 8.1

Set the interrupt masks so that RST5.5 is enabled, RST6.5 is masked, and RST7.5 is enabled.

Solution

Initially determine the contents of the accumulator:

Enable 5.5	bit $D_0 = 0$
Disable 6.5	bit $D_1 = 1$
Enable 7.5	bit $D_2 = 0$
Allow setting the masks	bit $D_3 = 1$
Don't reset the flip flop	bit $D_4 = 0$
Bit 5 is not used	bit $D_5 = 0$
Don't use serial data	bit $D_6 = 0$
Serial data is ignored	bit $D_7 = 0$

SOD	SDE	xxx	R7.5	MSE	M7.5	M6.5	M5.5
0	0	0	0	1	0	1	0

Content of accumulator is 0AH. The program for the above operation is given below:

PROGRAM 8.1

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	EB		EI		Enable all interrupts
8001	3E, 0A		MVI	A, 0AH	mask to enable RST 7.5, and 5.5, disable 6.5
8003	30		SIM		Apply the settings RST masks

RIM Instruction

The RIM instruction loads the accumulator with 8 bits, which consists of the status of the interrupt mask, the interrupt, enable, the pending interrupts and one bit of serial input data. Figure 8.35 shows the accumulator content for RIM instruction

- ✓ **Interrupt Mask Bits D_0 , D_1 , and D_2** Bits D_0 , D_1 and D_2 represent the current setting of the mask for each of RST 7.5, RST 6.5 and RST 5.5. A high level shows that interrupt is masked and low level means that interrupt is not masked. Bits D_0 , D_1 and D_2 return the contents of the three mask flip flops. These bits can be used by a program to read the mask settings in order to modify only the right mask.

- ✓ **Interrupt Enable Bit D_3** Bit D_3 is the interrupt enable flag. This bit shows whether the maskable interrupt process is enabled or not. When it is high, interrupt is enabled. If it is low, interrupt is disabled. It

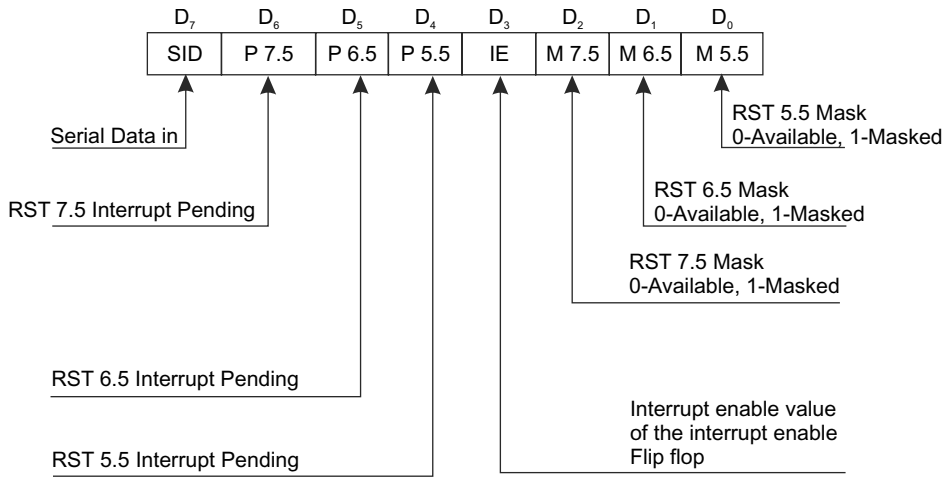


Fig. 8.35 Accumulator content for RIM instruction

returns the contents of the interrupt enable flip-flop. It can be used by a program to determine whether or not interrupts are enabled.

✓ **Interrupts Pending Bits D₄, D₅, D₆** Bits D₄, D₅ and D₆ represent the pending interrupts. Bits D₄ and D₅ return the current value of the RST5.5 and RST6.5 pins. Bit D₆ returns the current value of the RST7.5 memory flip-flop. A high level on bits D₄, D₅ and D₆ states that interrupt are pending. A low level on bits D₄, D₅ and D₆ states that interrupts are not pending.

✓ **Serial Input Data Bit D₇** Bit D₇ is used for serial data input. The RIM instruction reads the value of the SID pin on the microprocessor and returns it in this bit.

Example 8.2

Write instructions to call interrupt service subroutine 003CH corresponding to RST 7.5 if it is pending. Assume the content of accumulator is 20H on executing of RIM instruction.

Solution

The program for the above operation is given below:

PROGRAM 8.2

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
9000	20		RIM		Accumulator content is 20H on executing RIM instruction
9001	E6, 40		ANI	40H	AND immediate with 40H
9003	CD, 3C, 00		CALL	003CH	Call interrupts service routine for RST 7.5, when RST 7.5 is pending.

Initially, the contents of the accumulator is 20H.

SID	P7.5	P6.5	P5.5	IE	M7.5	M6.5	M5.5
0	0	1	0	0	0	0	0

After immediate ANDing with 40H, the content of accumulator is given below.

SID	P7.5	P6.5	P5.5	IE	M7.5	M6.5	M5.5
0	1	0	0	0	0	0	0

The 8085 microprocessor has two additional instructions such as Enable Interrupt (EI) and Disable Interrupt (DI). These instructions can enable or disable all the interrupts except TRAP interrupt. The interrupt enable flip-flop is manipulated using the EI/DI instructions. Actually, EI and DI instruction generates internally EI and DI signals. The connections of EI and DI are depicted in Fig. 8.30. The EI signal sets the SR flip-flop and generates an interrupt output signal. The interrupt enable signal enables the AND gates at the RST 7.5, RST 6.5 and RST 5.5 inputs. The DI signal can reset the SR flip-flop and makes the interrupt enable output becomes low. Then all maskable interrupts are disabled. The application of EI and DI is shown in Fig. 8.36.

Example 8.3

Write instructions to enable interrupt RST 7.5 and disable RST 6.5 and RST 5.5.

Solution

The content of accumulator for the SIM instruction to enable interrupt RST 7.5 and disable RST 6.5 and RST 5.5 are given below.

SOD	SOE	x	R7.5	MSE	M7.5	M6.5	M5.5
0	0	0	0	1	0	1	1

Bit D_2 is set to 0 and bits D_1 and D_0 are reset to 1 to enable interrupt RST 7.5 and disable RST 6.5 and RST 5.5 respectively. The content of accumulator is 0BH. The instructions for the above operation are given below:

PROGRAM 8.3

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8500	FB		EI		Enable all interrupts
8501	3E, 0B		MVI	A, 0BH	Load 0BH to enable RST 7.5, and disable RST5.5 and RST 6.5
8502	30		SIM		Apply the settings to enable RST 7.5, and disable RST5.5 and RST 6.5

Example 8.4

Write instructions for the following operations The microprocessor has completed the RST 7.5 interrupt. Check if RST 5.5 is pending. When RST 5.5 is pending, enable RST 5.5 interrupt.

Solution

The content of accumulator for the RIM instruction for RST 5.5 pending is 10H.

SID	P7.5	P6.5	P5.5	IE	M7.5	M6.5	M5.5
0	0	0	1	0	0	0	0

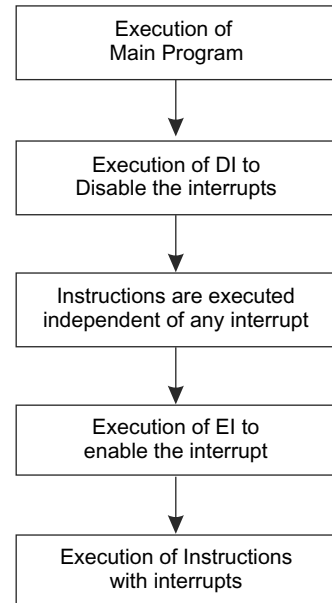


Fig. 8.36 Application of EI and DI

The content of accumulator for the SIM instruction to enable interrupt RST 5.5 and disable RST 6.5 and RST 7.5 are given below.

SOD	SOE	x	R7.5	MSE	M7.5	M6.5	M5.5
0	0	0	0	1	1	1	0

Bit D₀ is set to 0 and bits D₁ and D₂ are reset to 1 to enable interrupt RST 5.5 and disable RST 6.5 and RST 7.5 respectively. Then content of accumulator is 0EH. The instructions for the above operation are as follows:

PROGRAM 8.4

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
9000H	20		RIM		Read interrupt mask
9001H	47		MOV	B,A	Store mask information into B register
9002H	E6, 10		ANI	10H	Check whether RST 5.5 is pending or not
9004H	C2, Level		JNZ	Level	
9007H	FB		EI		
9008H	C9		RET		RST 5.5 is not pending, and return to main program
9009H	78		MOV	A,B	Read bit pattern and RST 5.5 is pending
900AH	E6, 0E		ANI	0EH	Enable RST 5.5
900CH	F6, 08		ORI	08H	Enable SIM by setting D3
900EH	30		SIM		
900FH	CD, 2C, 00		CALL	002CH	Call service routine from 002CH

The RIM instructions checks whether any interrupts are pending. The ANI 10H masks all bits except D₄ to check pending RST 5.5 interrupt. Bit D₄ indicates that RST 5.5 is pending. The ANI 0E and ORI 08H instructions enable RST 5.5 interrupt. The CALL instruction transfers the execution of program to the service routine of RST 5.5.

8.3.5 Pending Interrupts

The 8085 microprocessor has five interrupt lines. One interrupt may occur during an ISR and other interrupts remain pending.

If more than one interrupt wants servicing simultaneously, the microprocessor can only response to one interrupt at a time. Therefore, some priority has been assigned to different interrupt lines which allows their signals to reach the microprocessor according to the priority. This problem can be solved by one additional circuit known as the priority encoder, 74LS148. This circuit has 8 inputs and 3 outputs. The inputs are assigned increasing priorities according to the increasing index of the input so that input 7 has highest priority and input 0 has the lowest. Using the RIM instruction, the programmer can read the status of the interrupt lines and find if there are any pending interrupts.

8.4 INTERRUPTS OF 8086/8088 MICROPROCESSOR

An interrupt is an external signal which sends information to the CPU so that an external device gets service from CPU. This signal provides a mechanism for changing one program environment to an other. Due to an interrupt, the microprocessor stops execution of its current instruction and calls a procedure to provide

service to interrupt. At the end of the interrupt service procedure, an IRET instruction is executed to return back to the main program. The 8086/8088 processor has the following interrupts:

- ✦ Software interrupts
- ✦ Nonmaskable interrupts
- ✦ Internal interrupts
- ✦ External hardware interrupts
- ✦ Reset

8.4.1 Software Interrupts

When the source of interrupt is the execution of interrupt instructions, this interrupt is known as software interrupt. In 8086/8088, there are about 256 interrupts such as INT 00H, INT 01H, INT 02H to INT FFH. Whenever the INT interrupt instruction is executed, the microprocessor automatically saves the content of the flag register, Instruction Pointer (IP) and code segment register on the stack and jumps to a specified memory location. In 8086, the memory location is always four times the value of the interrupt number. When INT n is executed, the interrupt service routine is located at $n \times 4$ H memory address. For example, INT 02H goes to 00008H. Software interrupts are always generated by INT instructions and used for divide-by-zero error, single-step, NMI, break-point, and overflow interrupts.

For each interrupt, there is a program associated with a specified interrupt. This program is known as Interrupt Service Routine (ISR). It is also called *interrupt handler*. When the interrupt occurs, the processor runs the interrupt service routine according to the interrupt vector table as given in Table 8.11.

Table 8.11 *Interrupt vector table*

<i>INT Number</i>	<i>Physical address assuming CS = 0000H</i>
INT 00H	00000H
INT 01H	00004H
—	—
—	—
INT FEH	003F8H
INT FFH	003FCH

The interrupt vector table has 256 entries, each containing four bytes. Each interrupt vector consists of a 16-bit offset and the 16-bit segment address. The initial 32 interrupt vectors are spared for various microprocessor operations and the remaining 224 interrupt vectors are user defined. The lower vector number has the higher priority, when more than two interrupts occur simultaneously. Figure 8.37 shows the interrupt pointers. There are 256 address pointers. The starting addresses of their service routines are available in the program memory as depicted in Fig. 8.37.

Interrupt Type 0—INT 00H (Divide by Zero Error)

The 8086 generates a type 0 interrupt, if the result of DIV or IDIV operation is too large to fit in the destination register. For this interrupt, 8086 pushes the content of flag register on the stack, resets IF and TF and also pushes the content of CS and IP onto the stack. Then 8086 gets the starting address of the interrupt service procedure from the interrupt pointer table. Therefore, load the new value of CS from addresses 00002H and 00003H; also load the new value of IP from addresses 00000H and 00001H.

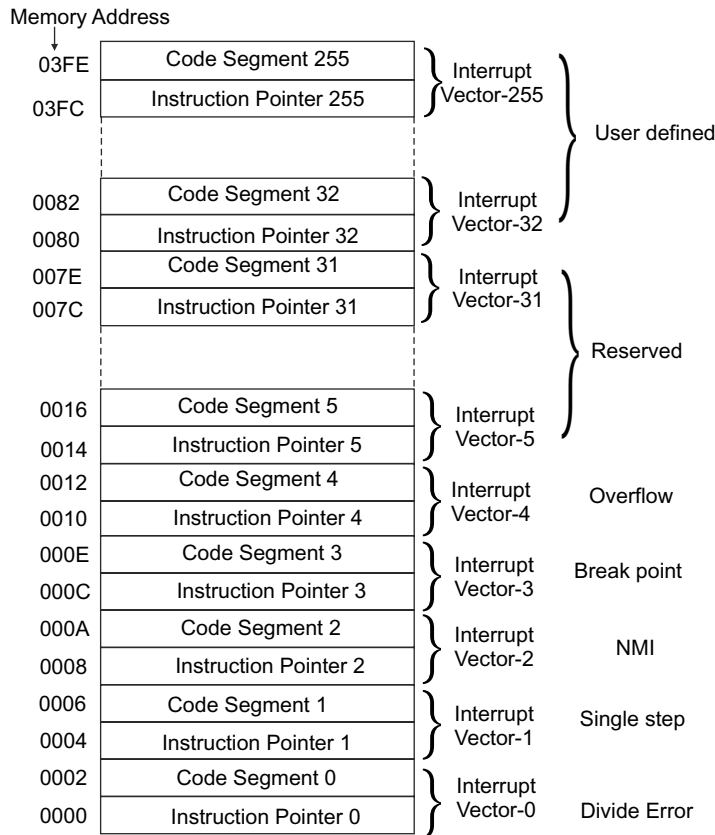


Fig. 8.37 Memory address of interrupt vectors

Interrupt Type 1—INT 01H (Single Step)

During execution of a sequence of instructions, there is frequently a need to examine the contents of the CPU's registers and system memory. This is done by executing one instruction at a time and then inspecting the registers and memory. If they are correct, the user can give the command to go on and execute the next instruction. This called the *single stepping*. When the 8086 Trap Flag (TF) is set, the 8086 perform a type 1 interrupt after execution of each instruction.

When the CPU gets a type 1 interrupt, initially it pushes the flag register onto the stack, changing the trap bit and pops the flag register back from the stack. Then it loads the CS value from starting address 00006H and the IP value from starting address 00004H for the type 1 interrupt service routine. The 8086 has no instruction to set or reset the trap flag. A sequence of instructions is used to set the trap flag as given below:

```

PUSHF                ; Push flags onto stack
MOV BP, SP           ; Copy SP to BP
OR [BP+0], 001000H  ; Set TF bit
POP F                ; Pop flag register and TF is set

```

To reset the trap flag, the OR instruction will be replaced by AND [BP + 0], 0FEFFH. After reset the trap flag, when the 8086 processor sends a type 1 interrupt, single-step mode will be disabled.

Interrupt Type 2—INT 02H (Nonmaskable Interrupt) The 8086 performs a type 2 interrupt when the NMI pin receives a low to high transition signal. Then the CPU, 8086 pushes the content of flag register on the stack, resets IF and TF and also pushes the content of CS and IP onto the stack. After that, the CPU jumps to 00008H to fetch the CS: IP of the ISR associated with NMI. As the type 2 interrupt response cannot be disabled (masked) by any instruction, this interrupt is called a nonmaskable interrupt. Usually, the type 2 interrupt is used to switch off a circuit for protection.

Interrupt Type 3—INT 03H (Break Point) The type 3 interrupt is generated by the execution of INT 03H instruction. This interrupt is used to implement a break-point function in a system. When we insert a break point in the main program, the system executes all instructions up to the break point and then jumps to the break-point subroutine. When 8086 executes the INT 03H instruction, it pushes the content of the flag register onto the stack, resets TF and IF and pushes the CS and IP values onto the stack. Then 8086 gets the new IP value from the starting address 0000CH and the CS value from the starting address 0000EH.

Interrupt Type 4—INT 04H (Overflow Interrupt) If the result of addition of two signed numbers is too large to represent in the destination register, the overflow (OF) flag will be set. For example, if we add 0110 1100 (108_{10}) and 0101 0001 (81_{10}), the result is 1011 1101. The above result will be correct only for unsigned binary numbers. For signed number addition, "1" in the MSB of the result represents that the result is negative and it is in 2's complement form. Hence the result, 1011 1101 represents -67_{10} , but the correct result is (189_{10}). If the overflow flag is set, the 8086 provides type 4 interrupt after executing the INTO instruction.

During execution of type 4 instruction, the 8086 pushes the content of flag register on the stack, resets TF and IF and pushes the values of CS and IP on the stack. Then 8086 gets a new IP value from the starting address 00010H and a new CS value from the starting address 00012H. After that instructions in the ISR perform the desired operation.

8.4.2 INTR Interrupts, Type 0 to 255

The INTR input pin of 8086 allows external signal to interrupt the execution of a program. INTR can be masked or disabled so that it cannot cause any interrupt. When the Interrupt Flag (IF) is cleared, INTR input pin becomes disabled. The IF can be cleared by Clear Interrupt Instruction (CLI).

When the IF is set, the INTR input will be enabled. The IF can be set by Set Interrupt Instruction (STI). After reset of 8086 microprocessor, the interrupt flag is automatically cleared. The INTR interrupt is sent to the 8086 from the 8259A interrupt controller as shown in Fig. 8.38.

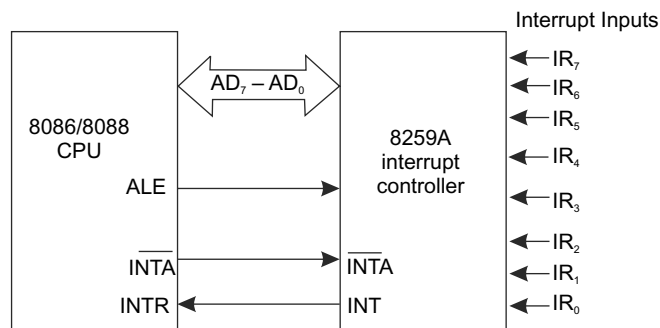


Fig. 8.38 8259A connected with 8086/8088 microprocessor

When the 8259A receives an interrupt signal on any one of the IR inputs, it provides an interrupt request signal to the INTR input of 8086. When the INTR input is enabled with an STI instruction, the 8086 processor sends an interrupt acknowledge signal. Figure 8.39. shows the interrupt acknowledge bus cycle of 8086.

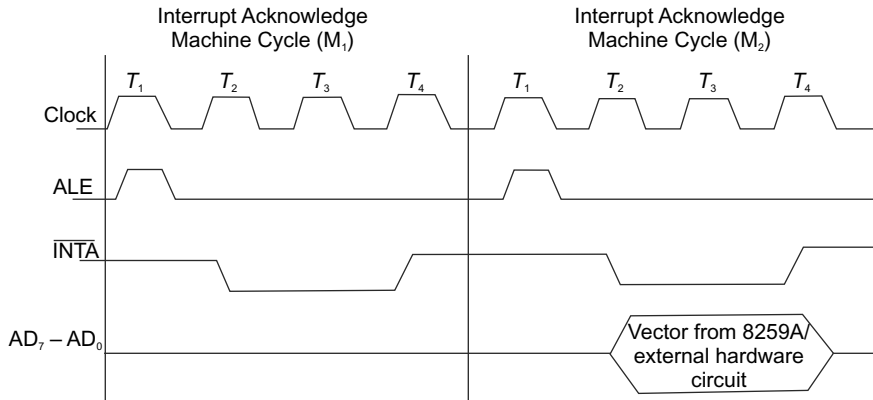


Fig. 8.39 8086/8088 interrupt acknowledge bus cycle

8.4.3 External Hardware Interrupt Interface

Figure 8.40 shows the minimum-mode hardware interrupt interface of 8086 microprocessor. The external hardware interrupt circuit can identify which of the pending interrupts has highest priority. Subsequently, hardware interrupt circuit passes its type number to the microprocessor. Then 8086 CPU samples INTR input during the last clock period of each interrupt execution cycle.

INTR is a level-triggered input. The logic level '1' must be maintained until it is sampled, but it must be removed before it is sampled next time. Otherwise the same interrupt service routine (ISR) will be repeated.

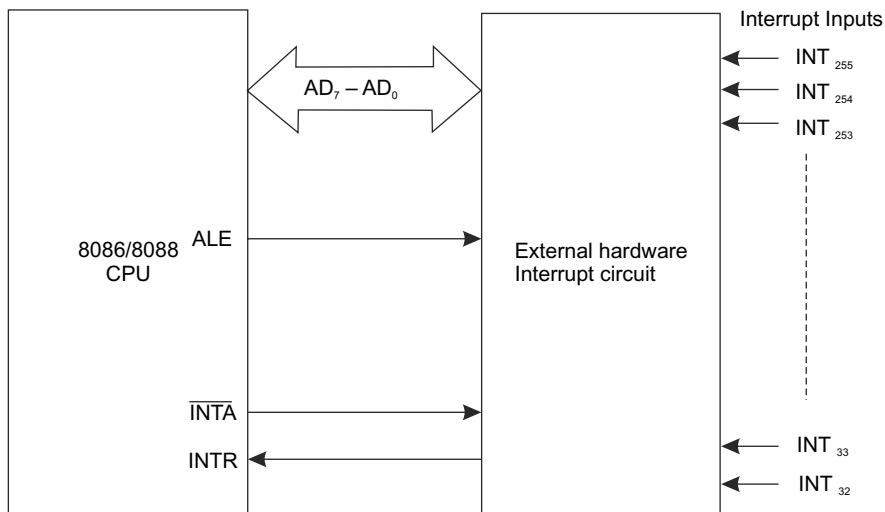


Fig. 8.40 External hardware interrupt interface with 8086/8088 CPU

\overline{INTA} becomes logic level '0' in the first interrupt bus cycle to acknowledge the interrupt as the 8086 CPU has decided to respond to the interrupt.

It goes to '0' again in the second bus cycle to request for the interrupt vector type from external device. Then interrupt-type number is read by the microprocessor and the new value of CS and IP are also read from the memory. Figure 8.39 shows the interrupt acknowledge bus cycle of 8086.

8.4.4 Priority of 8086/8088 Interrupts

In the 8086/8088 microprocessor, all interrupts must be serviced as priority order. The highest-priority interrupt will be serviced first and then the next highest-priority interrupt will be serviced. Therefore, lower-priority interrupt service will be provided after a higher-priority one. The priority of interrupts will be controlled by ISR. The priority order of 8086/8088 interrupts are

- ✦ Reset
- ✦ Internal interrupts
- ✦ Software interrupts
- ✦ Nonmaskable interrupts
- ✦ Hardware interrupts

8.4.5 Interrupt Instructions of 8086/8088

The interrupt instructions of 8086/8088 microprocessors are CLI, STI, INT n INTO, HLT and WAIT. The functional operation of interrupt instructions are given in Table 8.12.

Table 8.12 *Interrupt instructions*

<i>Mnemonics</i>	<i>Function</i>	<i>Operation</i>
CLI	Clear interrupt instruction, IF affected	IF ← 0
STI	Set Interrupt flag, IF affected	IF ← 1
INT n	Type n software interrupts. This interrupt initiates a vectored call of an interrupt service subroutine	(SP - 2) ← Flags, TF, IF ← 0, (SP - 4) ← CS, CS ← (2 + 4 × n), IP ← (SP - 6), IP ← 4 × n
IRET	Interrupt return, All flags affected	IP ← (SP), CS ← (SP + 2), Flags ← (SP + 4), SP ← (SP + 6)
INTO	Interrupt overflow, TF and IF affected	Same as INT 4
HLT	Halt	Wait for an external interrupt
WAIT	Wait	Wait for <i>TEST</i> input to become active

8.4.6 Interrupt Cycle of 8086/8088

The sequence of operations of any interrupt (interrupt cycle) is depicted in Fig. 8.41 and the step-by-step operation for interrupt is given below:

Step 1 The interrupt sequence starts when an external device requests service by sending an interrupt input.

Step 2 The external hardware circuit or interrupt controller evaluates the priority of the interrupt.

Step 3 The 8086 checks for the INTR at the last T state of the instruction.

Step 4 Check IF before sending interrupt acknowledge signal INTA.

Step 5 8086 initiates the INTA bus cycle. During T_1 of the first bus cycle, ALE is high and address/data bus AD_7-AD_0 is at high impedance (Z) state and stays high for the bus cycle. During the second interrupt acknowledge bus cycle, external circuit gates one of the interrupts.

Step 6 The contents of the flag register are pushed on the stack.

Step 7 The Interrupt Flag (IF) and Trap Flag (TF) are cleared. This disables the INTR pin and the trap or single-step feature.

Step 8 The contents of the Code Segment (CS) is pushed on the stack.

Step 9 The contents of the Instruction Pointer (IP) is pushed on the stack.

Step 10 The interrupt vector type number is multiplied by 4 and generates a memory address. The contents of this address are fetched and placed into IP. Subsequently, the contents of the memory address (interrupt vector type number $\times 4 + 2$) are fetched and placed into CS. After that the next instruction executes at the interrupt service procedure addressed by the interrupt vector.

Step 11 To return from the interrupt service routine, the IRET instruction is executed.

Step 12 Flags return to their state prior to the interrupt. Operation restarts at the prior IP address after CS and IP are popped.

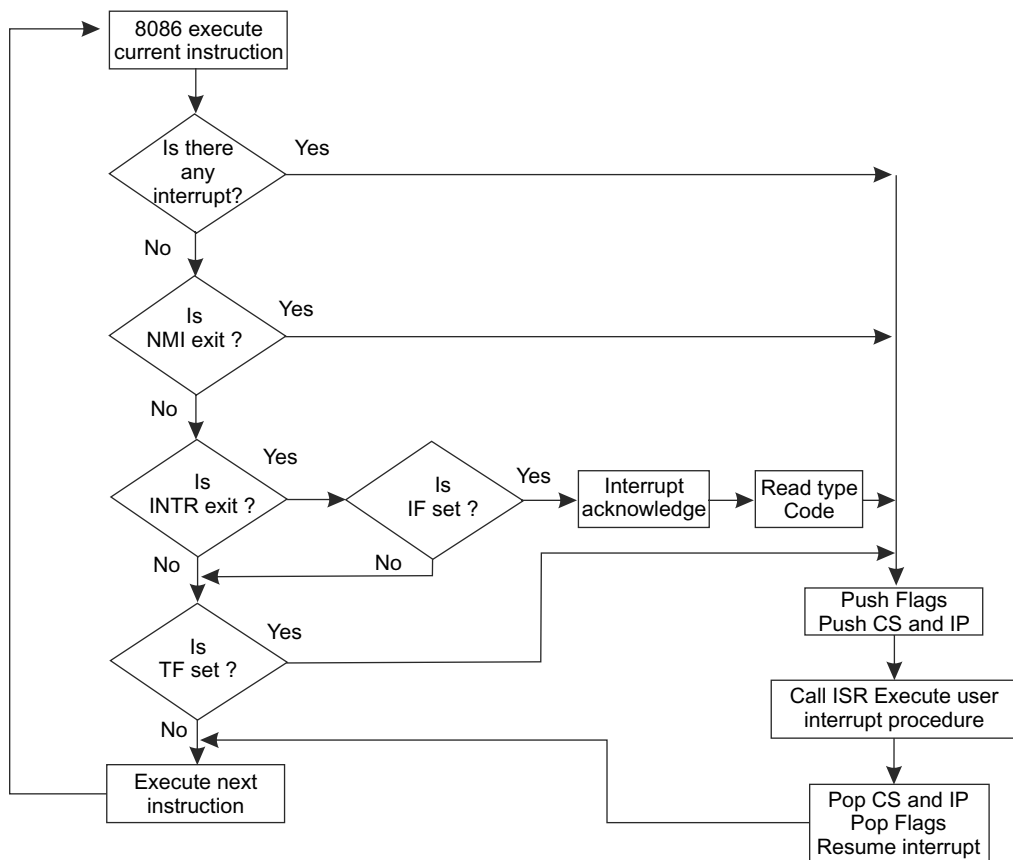


Fig. 8.41 Flowchart of interrupt operation

8.5 8259A PROGRAMMABLE INTERRUPT CONTROLLER

Microprocessor-based system design requires many I/O devices such as keyboards, displays, sensors and other components. These devices should receive servicing in an efficient manner from the CPU. The most common method of servicing such devices is known as the *polled approach*. In this approach, the processor must test each device in sequence and find the device that requires servicing. For this, a large portion of the main program is looped through this continuous polling cycle. Such a method would have a serious detrimental effect on system throughput, thus limiting the tasks that could be assumed by the microprocessor and reducing the cost effectiveness of using such devices. The other most desirable method is that the microprocessor can execute its main program and only stop to service peripheral devices when CPU receives a signal from the device itself.

Then the processor should complete whatever instruction is currently being executed and fetch a new routine that will service the requesting device. However, after completion of service, the processor would resume exactly where it left off. This method is known as *interrupt*. Interrupts are used in a microcomputer system for different applications. When the number of I/O devices are less, the already available interrupts of microprocessors are sufficient and there is no requirement of programmable interrupt controller as shown in Fig. 8.42.

The CPU can access many devices using interrupt signals. In multiple interrupt systems, the CPU must take care of the priorities for the interrupts and simultaneously occurred interrupts.

To overcome all difficulties, a Programmable Interrupt Controller (PIC) has been designed and can be used to handle many interrupts at a time. This controller handles all simultaneous interrupt requests along with their priorities and the microprocessor will be relieved from this task. The Programmable Interrupt Controller (PIC) functions as an overall manager in an interrupt-driven system environment as depicted in Fig. 8.43. It accepts requests from the peripheral equipment, and then it determines priority value of all incoming requests and issues an interrupt to the CPU based on this determination. The 8259A Programmable Interrupt Controller can be interfaced with 8085, 8086 and 8088 processors. The features of these devices are given below:

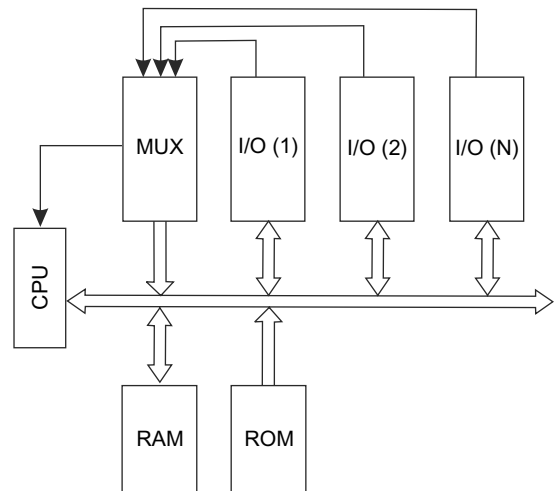


Fig. 8.42 Interrupt driven CPU using MUX

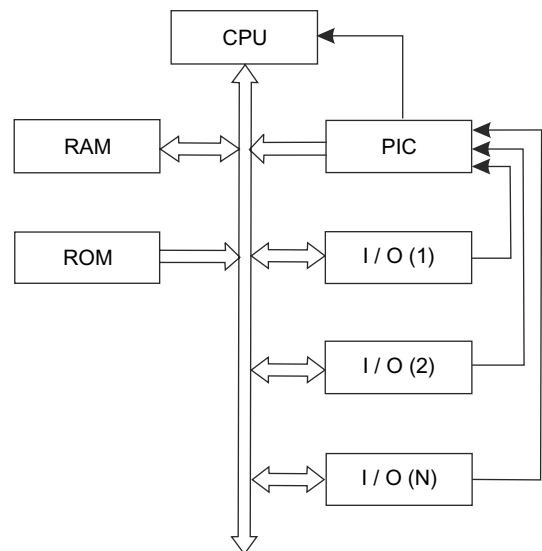


Fig. 8.43 PIC in an interrupt-driven environment

FEATURES

- 8085, 8086, and 8088 compatible
- This device is an eight-level priority controller
- Individual request mask capability
- This device is able to accept level-triggered or edge-triggered inputs
- Programmable interrupt modes
- Single a +5 V supply (no clocks)
- Available in 28-pin DIP and 28-lead
- This interrupt can be expandable to 64 levels

8.5.1 Pin Diagram

The Intel 8259A programmable Interrupt Controller handles up to eight-vectored priority interrupts for the CPU. This IC is cascadable for up to 64-vectored priority interrupts without additional circuitry. This IC is available in a 28-pin DIP package and uses NMOS technology and requires a single 5 V supply. Circuitry is static, requiring no clock input. The schematic diagram of 8259A is depicted in Fig. 8.44. The pin diagram of 8259A is also shown in Fig. 8.45 and the pin functions are explained below:

VCC 5 V supply.

GND GROUND

\overline{CS} (Chip Select) When the chip select pin is active low, this pin enables read RD and write WR operation between the CPU and the 8259A. INTA functions are independent of \overline{CS} .

\overline{WR} (Write) A low on this pin, when \overline{CS} is low, enables the 8259A for write operation. This pin also enables to accept command words from the CPU.

\overline{RD} (Read) When \overline{RD} is active low and \overline{CS} is low, this pin enables the 8259A to release status onto the data bus for the CPU.

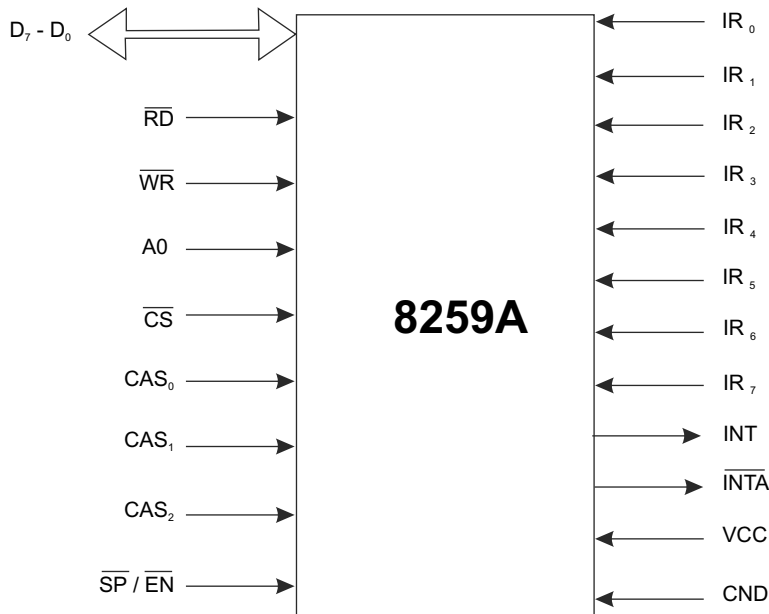


Fig. 8.44 Schematic pin diagram of 8259A

D_7 – D_0 (I/O Bidirectional Data Bus)

These pins are used as bidirectional data buses. The control, status and interrupt-vector informations are transferred through this bus.

 CAS_0 – CAS_2 (I/O Cascade Lines)

A 8279A has only eight interrupts. When the number of interrupts requirement is more, a multiple interrupt controller must be connected in cascade. The CAS lines of a 8259A bus is used to control a multiple 8259A structure. These pins are outputs for a master 8259A and inputs for a slave 8259A.

 $\overline{SP/EN}$ (I/O Slave Program/Enable Buffer)

This is a dual-function pin. When this IC is used in the buffered mode, it can be used as an output to control buffer transceivers (EN). If this IC is not in the buffered mode, it is used as an input to designate a master (SP = 1) or slave (SP = 0).

INT (Interrupt)

This pin goes high whenever a valid interrupt request is asserted. This pin signal is used to interrupt the CPU. Therefore, it is connected to the CPU's interrupt pin.

 IR_0 – IR_7 (Interrupt Requests)

These pins are used as asynchronous inputs. Each pin can be used to receive an interrupt request to the CPU by raising an IR input from low to high. The interrupt pin must be maintained at high level until this is acknowledged (edge-triggered mode), or just by a high level on an IR input (level-triggered mode).

 \overline{INTA} (Interrupt Acknowledge)

This pin becomes high when a valid interrupt request is asserted. This pin is used to enable 8259A interrupt vector data onto the data bus by a sequence of interrupt acknowledge pulses issued by the CPU.

 A_0 (Address Line)

This pin works in conjunction with the \overline{CS} , \overline{WR} , and \overline{RD} pins. This is also used by the 8259A to read various command words the CPU writes and status the CPU wishes to read. Generally, this is connected to the CPU A_0 address line.

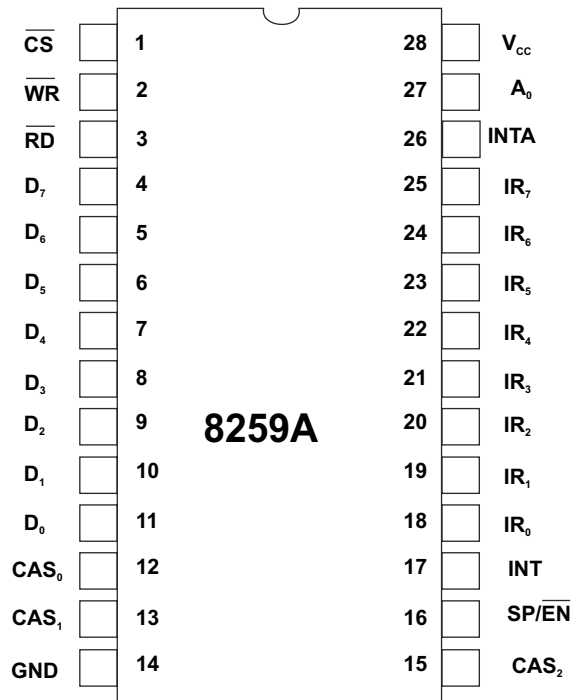


Fig. 8.45 Pin diagram of 8259A

8.5.2 Functional Description

The functional block diagram of 8259A Programmable Interrupt Controller is shown in Fig. 8.46. Each functional block has been explained below:

Interrupt Request Register (IRR)

The interrupts at the IR input lines are handled by two internal registers, the Interrupt Request Register (IRR) and the In-Service register (ISR). The IRR is used to store all the interrupt requests, which are requesting service and it provides service one by one on the priority basis.

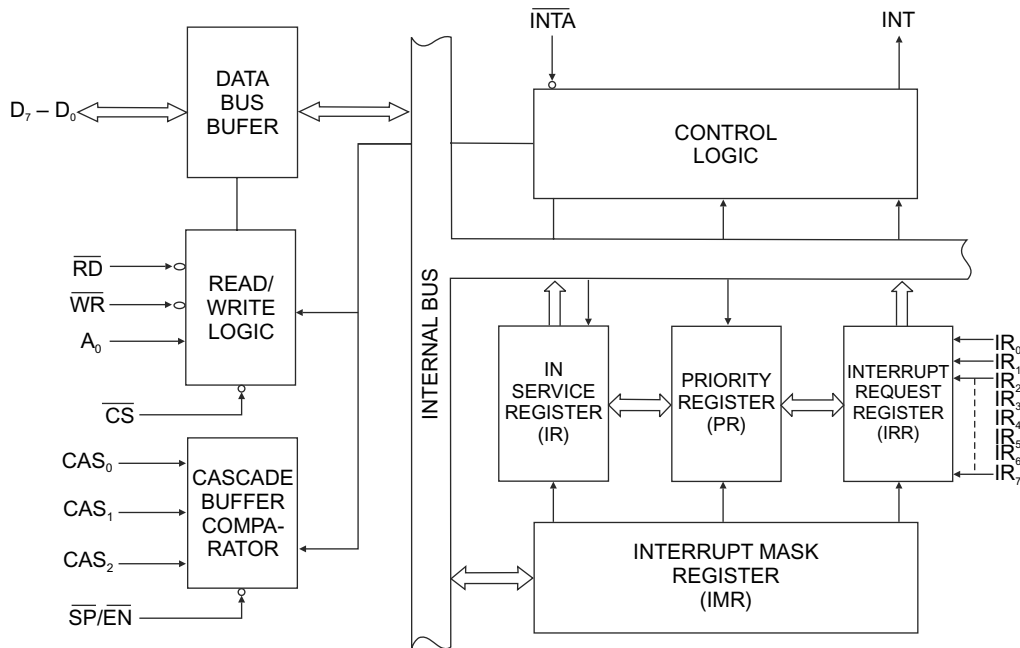


Fig. 8.46 Functional diagram of 8259A

In-Service Register (ISR) The In-Service Register (ISR) is used to store all the interrupt levels which are being serviced, and also keeps a track of the request being served.

Priority Resolver This logic block determines the priorities of the interrupt requests in the IRR. The highest priority is selected and strobed into the corresponding interrupt request of the ISR during INTA pulse.

Interrupt Mask Register (IMR) This Interrupt Mask Register (IMR) stores the bits that mask the interrupt lines to be masked. The IMR operates on the IRR based priority resolver.

Interrupt Control Logic The interrupt control logic block manages the interrupt and the interrupt acknowledge signals. The interrupt (INT) and interrupt acknowledge (INTA) signals are directly send to the CPU interrupt input. The INTA signal from CPU that will cause the 8259A to release vectoring information onto the data bus.

Data Bus Buffer This 3-state, bidirectional 8-bit buffer is used to interface the 8259A to the microprocessor data bus. Control words and status information are transferred through the data bus buffer.

Read/Write Control Logic This block is used to accept Output commands from the CPU. It contains the Initialization Command Word (ICW) registers and Operation Command Word (OCW) registers which store the various control formats for device operation. This function block also allows the status of the 8259A to be transferred onto the data bus.

Cascade Buffer/Comparator The cascade buffer/comparator block stores and compares the IDs of all 8259A's used in the microprocessor system. The associated three I/O pins (CAS₀, CAS₁ and CAS₂) are outputs when the 8259A is used as a master and are inputs when the 8259A is used as a slave. When the

8259A is used as a master, the 8259A sends the ID of the interrupting slave device onto the CAS_0 to CAS_2 lines. The slave thus selected will send its preprogrammed subroutine address onto the data bus during the subsequent INTA pulses.

8.5.3 Interrupt Sequence

The most powerful features of 8259A are programmability and the interrupt routine addressing capability. This device allows direct or indirect jumping to the specified interrupt service routine without any polling of the interrupting devices. The interrupt sequence for an interrupt in 8085 microprocessor system has been explained below:

1. One or more of the Interrupt Request lines (IR_7 to IR_0) are raised high, setting the corresponding IRR bits.
2. The 8259A evaluates these requests, and sends an INT to the CPU.
3. The CPU acknowledges the INT and responds with an INTA pulse.
4. After receiving an INTA from the CPU group, the highest priority ISR bit is set, and the corresponding IRR bit is reset. The 8259A will also release a CALL instruction code (11001101) onto the 8-bit data bus through its D_7 to D_0 pins.
5. This CALL instruction will initiate two more INTA pulses to be sent to the 8259A from the CPU group.
6. These two INTA pulses allow the 8259A to release its preprogrammed subroutine address onto the data bus. The lower 8-bit address is released at the first INTA pulse and the higher 8-bit address is released at the second INTA pulse.
7. This completes the 3-byte CALL instruction released by the 8259A. In the AEOI (automatic end of interrupt) mode, the ISR bit is reset at the end of the third INTA pulse. Otherwise, the ISR bit remains set until an appropriate EOI (end of interrupt) command is issued at the end of the interrupt sequence.

8.5.4 INTERFACING of 8259A with 8085

Figure 8.47 shows the interfacing of 8259A with 8085. When the 8259A PIC receives an interrupt, INT becomes active and an interrupt acknowledge cycle is started. If a higher-priority interrupt occurs between the two INTA pulses, the INT line goes inactive immediately after the second INTA pulse. After an unspecified amount of time, the INT line is activated again to signify the higher priority interrupt waiting for service. This inactive time is not specified and can vary between parts. The designer should be aware of this consideration when designing a system that uses the 8259A. It is recommended that proper asynchronous design techniques be followed.

The interfacing steps are explained below:

1. The address decoder output is connected with the \overline{CS} input of the IC.
2. A_0 line is used to select one of the two internal addresses in the device. This is connected to A_0 of the address lines of the microprocessor.
3. As the device operates in I/O-mapped I/O mode, the \overline{RD} and \overline{WR} signals are connected to \overline{IOR} and \overline{IOW} signals respectively.
4. The interrupt INT pin of 8259A is connected to the INTR input of the 8085 microprocessor.
5. INTA output of the processor is connected to the INTA input.
6. When $\overline{SP} / \overline{EN}$ pin is high, only one IC is used in the microprocessor-based system. If more than one ICs are connected in cascade, this pin must be low.

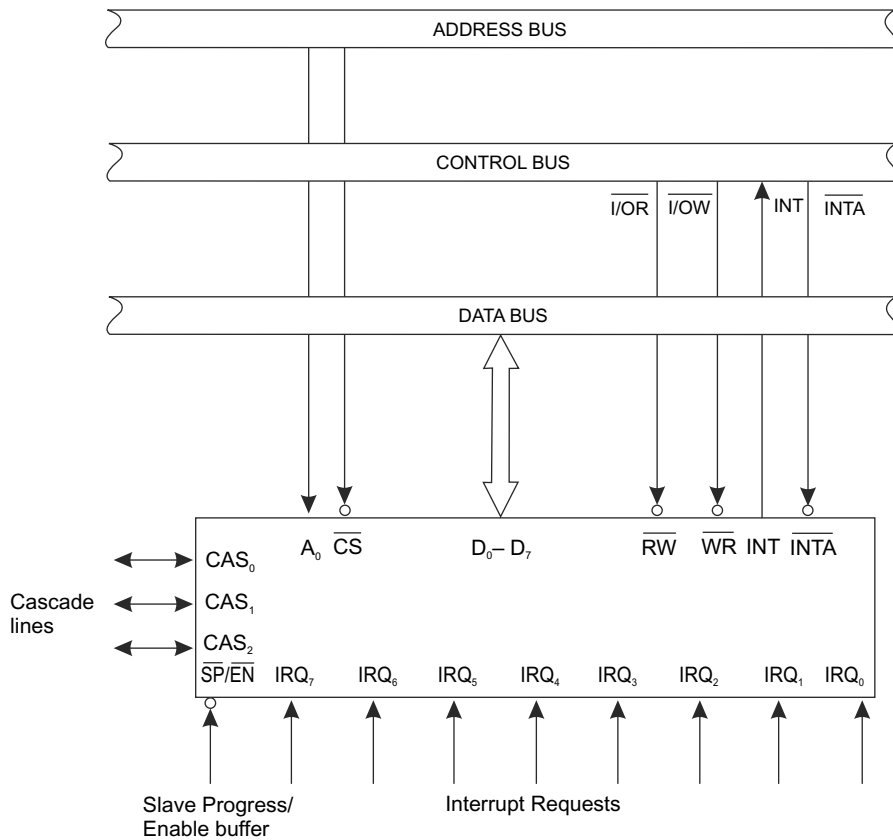


Fig. 8.47 Interfacing of 8259 with 8085 microprocessor

7. CAS_0 , CAS_1 and CAS_2 lines are generally opened.
8. There are eight IR input lines (IR_0 – IR_7) are available. When the IR inputs are not used, they must be grounded properly to avoid noise pulse in interrupt lines.

8.5.5 Programming of 8259A

The 8259A accepts two types of command words generated by the microprocessor. These two types of command words are Initialization Command Words (ICWs) and Operation Command Words (OCWs).

Initialization Command Words (ICW_s) The 8259A programmable interrupt controller can be initialized by sending a sequence of initialization control words (ICWs) to the controller. There are four initialization control words. The ICW_1 and ICW_2 always send to 8259 systems. When the system has any slave 8259A in cascade mode, ICW_3 must be used. In some special operations such as fully nested mode, ICW_4 can be used. All initialization command words are explained below:

Initialization Command Word 1 (ICW₁) Whenever a write command is received with $A_0 = 0$ and $D_4 = 1$, it is interpreted by 8259 as Initialization Command Word 1 (ICW_1). The ICW_1 starts the initialization sequence during which the following automatically occur.

1. The edge sense circuit is reset, which means that following initialization, an Interrupt Request (IR) input must make a low-to-high transition to generate an interrupt.
2. The Interrupt Mask Register (IMR) is cleared.
3. IR₇ input is assigned lowest priority 7.
4. The slave mode address is set to 7.
5. The special mask mode is cleared and status read is set to IRR.
6. If IC₄ is 0 then all functions selected in ICW₄ are set to zero. Master/Slave in ICW₄ is only used in the buffered mode.

The format of ICW₁ is shown in Fig. 8.48.

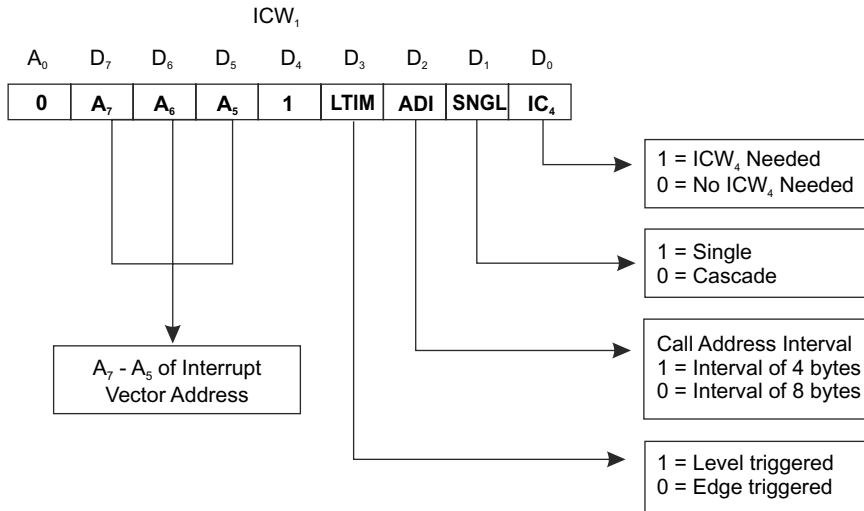


Fig. 8.48 Initialization Command Word 1

- ✓ **Bit D₀** It indicates whether ICW₄ is needed or not. If it is 1, ICW₄ is needed and if it is 0, ICW₄ is not needed.
- ✓ **Bit D₁** When this bit is 0, then only one 8259 is in the system. If it is 1, the additional 8259A are there in the system.
- ✓ **Bit D₂** ADI stands for address interval. If this bit is '0' then call address interval is 8 and if it is '1' then call address interval becomes 4.
- ✓ **Bit D₃** Bit D₃ determines recognition of the interrupts either in level triggered or edge-triggered mode. If this bit is 1 then the input interrupts will be recognized if they are in the level-triggered mode.
- ✓ **D₅-D₇** These are A₅-A₇ bits as shown in Fig. 8.48. For an interval spacing of 4, A₀-A₄ bits are automatically inserted by 8259A while A₀-A₅ are inserted automatically for an interval of 8. A₅-A₇ bits are programmable as set by the bits D₅-D₇ of ICW₁.

Initialization Command Word 2 (ICW₂)

The Initialization Command Word 2 (ICW₂) is shown in Fig. 8.49.

Bits D₇ - D₃ specify address bits A₁₅-A₁₁ interrupt vector address when operating in MCS 80/85 mode.

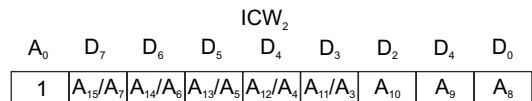


Fig. 8.49 Initialization Command Word 2

Bits D_2 – D_0 specify address bits A_{10} – A_8 for the interrupt vector address when operating in MCS 80/85 mode. These bits can be set to 0 when working on an 8086 system. T_3 – T_7 are interrupt vector address when the controller operates in 8086/8088 mode.

Initialization Command Word 3 (ICW₃)

The ICW₃ is used only when there is more than one 8259A in the system and cascading is used, in which case $SNGL = 0$. This will load the 8-bit slave register. The functions of this register are as follows:

In the master mode (either when $SP = 1$, or in buffered mode when $M/S = 1$ in ICW₄) a '1' is set for each slave in the system. The master then will release byte 1 of the call sequence (for MCS 80/85 system) and will enable the corresponding slave to release bytes 2 and 3 (for 8086 only byte 2) through the cascade lines.

In the slave mode (either when $SP = 0$, or if $BUF = 1$ and $M/S = 0$ in ICW₄) bits 2 ± 0 identify the slave. The slave compares its cascade input with these bits and, if they are equal, it releases bytes 2 and 3 of the call sequence for 8086 on the data bus.

$S_0 - S_7 = 1$ IR input has a slave and $S_0 - S_7 = 0$ IR -input does not have a slave

The ICW₃ is used only when there is more than Master Mode ICW₃

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

Fig. 8.50(a) Initialization Command Word 3 (master mode)

Slave Mode ICW₃

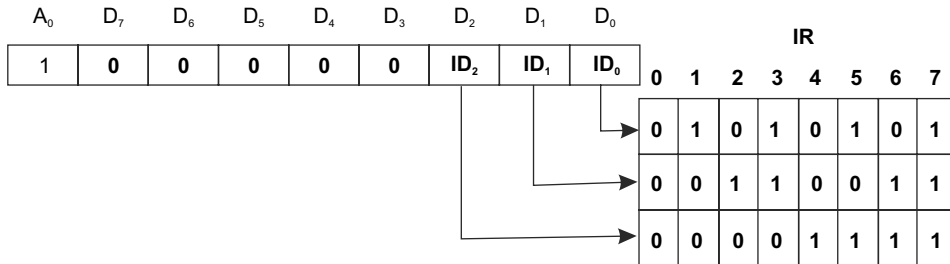


Fig. 8.50(b) Initialization Command Word 3 (slave mode)

Initialization Command Word 4 (ICW₄)

The format of ICW₄ is shown in Fig. 8.51. ICW₄ is loaded only if D_0 bit of ICW₁ (IC₄) is set. The position D_0 – D_4 are explained below:

- ✓ **Bit D_0 (MPM)** MPM stands for microprocessor mode. $MPM = 0$ sets the 8259A for MCS-80, 85 system operation, but $MPM = 1$ sets the 8259A for 8086 system operation
- ✓ **Bit D_1 (AEOI)** AEOI stands for the automatic end of interrupt mode. If $AEOI = 1$, then it is in auto EOI mode and if it is $= 0$, it is normal EOI mode.
- ✓ **Bit D_2 (M/S)** When buffered mode is selected, $M/S = 1$ means the 8259A is programmed to be a master, and $M/S = 0$ means the 8259A is programmed to be a slave. If $BUF = 0$, M/S has no function.
- ✓ **Bit D_3 (BUF)** Bit D_3 determines buffered/non-buffered mode of operation. If $BUF = 1$ the buffered mode is programmed. In buffered mode, SP/EN becomes an enable output and the master/ slave determination is by M/S.

✓ **Bit D_4 (SFNM)** SFNM stands for the Special Fully Nested Mode. If SFNM = 1 then this mode is programmed. In the cascaded mode of operation, when a slave receives a higher priority interrupt request than one, which is already in service (through the same slave), it would not be recognized by the master. This is happening as the master ISR bit is already in the set condition; thereby it ignores all requests of equal or lower priority. Therefore the higher priority interrupt won't be serviced until the master ISR bit is reset by an EOI command. This is most likely to happen after the completion of the lower priority routine.

If a truly fully nested structure is required within a slave 8259A, the especially fully nested mode should be used. The SFNM is programmed for the master mode only and done during master initialization through ICW₄. In this mode, the master will ignore interrupt requests of lower priority, and will respond to requests of equal or higher priority.

The Fully Nested Mode (FNM) is auto set after initialization is over. In this mode all interrupt requests are arranged from highest priority (IR₀) to lowest priority (IR₇). This mode can also be changed through operation command words (OCWs). When 8259 acknowledges an interrupt request through INTR pin, the device can find out the highest priority and the corresponding bit in the Interrupt Service Register (ISR) is set. The SFNM is different from FNM and the difference between SFNM and FNM are given below:

1. In SFNM, the slave is able to place an interrupt request, which has higher priority than the present interrupt being serviced. The master recognises the higher-level interrupt and can place this interrupt request to the CPU.
2. Before issuing an EOI command to the slave, the software must determine if any other slave interrupts are pending in SFNM. After that it reads its ISR (In Service Register). When the ISR contains all zeros, there is no interrupt from the slave in service and an EOI command can be sent to the master. If the ISR is not all zeros, an EOI command should not be sent to the master. When the master ISR bit is cleared with an EOI command while there are still slave interrupts in service, the lower-priority interrupt may be recognised by the master.

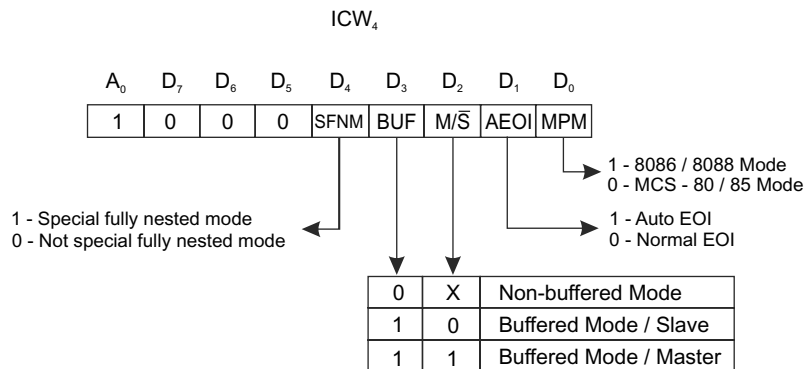


Fig. 8.51 Initialization Command Word 4

8.5.6 Programming Sequence of 8259

8259 is programmed by issuing initialization of command words and operation command words. Initialization of command words are issued in a sequence. The algorithm for initializing 8259 is given below:

1. Write ICW₁
2. Write ICW₂
3. If 8259 does not go in the cascade mode of operation, go to Step 5

4. Write ICW₃
5. If ICW₄ is not required, Go to step 7
6. Write ICW₄
7. Ready to accept interrupt sequence

The algorithm for initialization of 8259A programmable interrupt controller is depicted in Fig. 8.52.

Operation Command Words (OCWs)

These are the command words which command the 8259A to operate in various interrupts modes. These operating modes are:

1. Fully nested mode
2. Rotating priority mode
3. Special mask mode
4. Polled mode

There are three operation command words such as OCW₁, OCW₂ and OCW₃. These OCWs may be programmed to change the manner in which the interrupts are to be processed. The OCWs can be loaded into the 8259A any time after initialization.

Operation Command Word 1 (OCW₁)

The format of OCW₁ is shown in Fig. 8.53. OCW₁ sets and clears the mask bits by programming the Interrupt Mask Register (IMR). M₇–M₀ represents the eight mask bits. If M = 1, then corresponding interrupt is masked (inhibited) and M = 0 indicates the interrupt is unmasked. A write command with A₀ = 1 is interpreted as OCW₁, and written after ICW₂.

Operation Command Word 2 (OCW₂)

The OCW₂ enables the user to program 8259 in different modes. The format OCW₂ is shown in Fig. 8.54. Bits 3 and 4 are always set to 0. L₂, L₁, L₀ bits can be used to set the interrupt level acted upon which the controller must react. R, SL, EOI—these three bits control the rotate and end of interrupt modes and combinations of the two. A chart of these combinations is given below:

R	SL	EOI	Selection
0	0	0	Rotate in automatic EOI mode (CLEAR)
0	0	1	Non-specific EOI command
0	1	0	No Operation
0	1	1	Specific EOI command
1	0	0	Rotate in automatic EOI mode (SET)
1	0	1	Rotate in non-specific EOI
1	1	0	Set priority command
1	1	1	Rotate on specific EOI

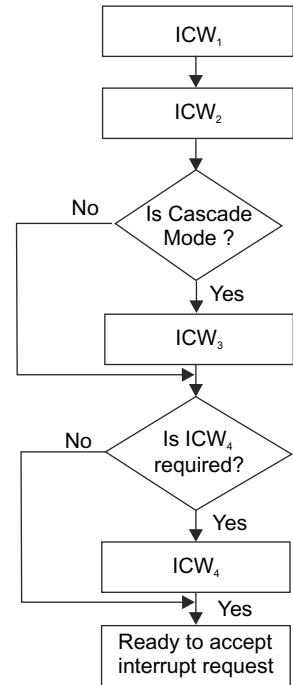


Fig. 8.52 Initialization sequence of 8259A

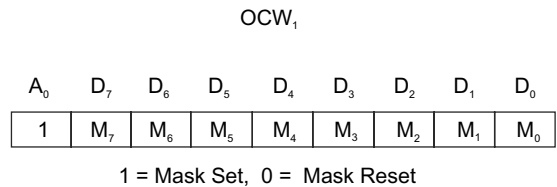


Fig. 8.53 Operation Command Word 1

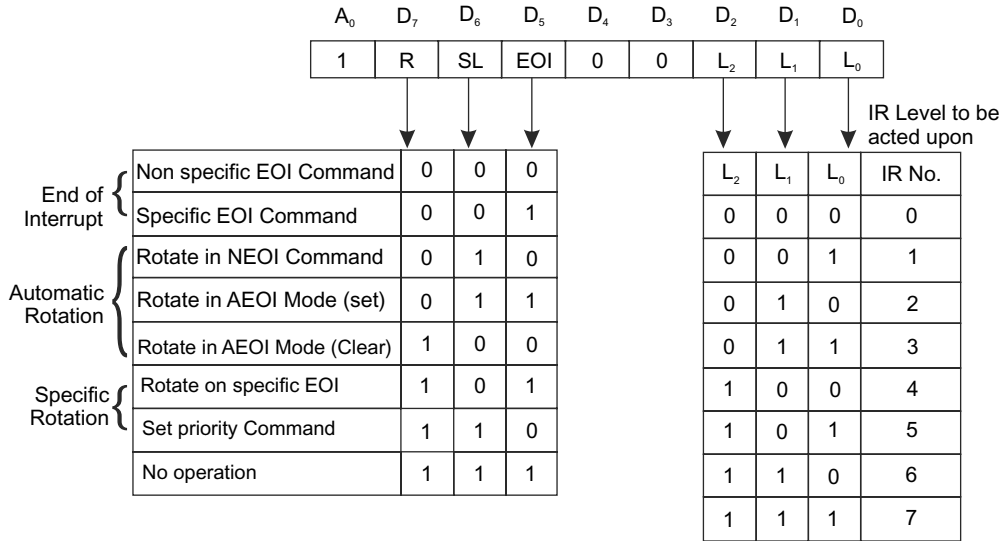


Fig. 8.54 Operation Command Word 2

The EOI command stands for End Of Interrupt. The interrupt service bit may be reset by an End of Interrupt command. Generally, this command is issued by the CPU just before sending the interrupt service routine. There are two different EOI commands such as non-specific EOI command and specific EOI command.

Non-specific EOI Command

The CPU issues a nonspecific EOI command. Actually, this is an OUT instruction by the CPU to 8259A. The 8259A automatically determine the interrupt level, i.e., highest priority interrupt in service and reset the correct bit in the ISR. The nonspecific EOI command must be used when the most recent level acknowledged and serviced is always the highest priority level. On receiving a nonspecific EOI command, 8259A simply resets the highest priority ISR bit, thus confirming to the 8259A that the highest priority routine of the routine in service is finished. The nonspecific EOI command is also known as *Fully Nested Mode (FNM)*. The advantage of the nonspecific EOI command is that IR level specification is not necessary as in the specific EOI Command. But some special consideration must be taken when deciding to use the nonspecific EOI.

Specific EOI Command (SEOI)

A SEOI command sent from the microprocessor to 8259A lets it know when a service routine of a particular interrupt level is completed. A SEOI command resets a specific ISR bit and any one of the eight IR levels can be specified. A SEOI command is required when 8259A is unable to determine the IR level. This command is best suited for situations in which priorities of the interrupt levels are changed during an interrupt routine (specific rotation).

Automatic EOI Mode (AEOI)

In AEOI mode, no command has to be issued. Therefore, AEOI mode simplifies programming and lower code requirements within interrupt routines.

AEOI mode must be used continuously as the ISR bit of a routine presently in service is reset right after its acknowledgement. Thus it leaves no designation in the ISR that a service routine is being executed. If any interrupt request occurs during this time and interrupts are enabled, it will be serviced regardless of its

priority, whether low or high. The problem of 'over-nesting' may happen in this case. It occurs when an IR input keeps interrupting its own routine. This results in unnecessary stack pushes, which could fill up the stack in a worst-case condition.

Automatic Rotation—Equal Priority When several communication channels are connected to a microcomputer system, all the channels should be accorded equal priority in sharing information with the microcomputer.

In this method once a peripheral is serviced, all other equal priority peripheral should be given a chance to be serviced before the original peripheral is serviced again. This is accomplished automatically assigning a peripheral the lowest priority after being serviced. In this way the device, presently being serviced, would have to wait until all other devices are serviced.

The automatic rotation is two types:

1. Rotate on nonspecific EOI Command
2. Rotate on automatic EOI Mode

Rotate on Nonspecific EOI Command When the rotate on NSEOI command is issued, the highest ISR bit is reset. Just after it is reset by a nonspecific EOI command, the corresponding IR level is assigned lowest priority. Figure 8.55 shows how the rotate on nonspecific EOI command effects the interrupt priorities.

Let us assume that the IR₀ has the highest priority and IR₇ has the lowest priority before command as shown in Fig. 8.55 (a). It is also assumed that IR₆ and IR₄ are already in service but neither is completed. As IR₄ has the highest priority, IR₄ routine be executed. When a NSEOI command is executed, Bit 4 in the ISR is reset. Then IR₄ becomes the lowest priority and IR₅ becomes the highest priority as depicted in Fig. 8.55 (b).

	IR ₇	IR ₆	IR ₅	IR ₄	IR ₃	IR ₂	IR ₁	IR ₀	
ISR Status	0	1	0	1	0	0	0	0	Before Command
Priority	7	6	5	4	3	2	1	0	

↑ Lowest Priority
↑ Highest Priority

Fig. 8.55(a) Priority of interrupts before command

	IR ₇	IR ₆	IR ₅	IR ₄	IR ₃	IR ₂	IR ₁	IR ₀	
ISR Status	0	1	0	1	0	0	0	0	After Command
Priority	2	1	0	7	6	5	4	3	

↑ Highest Priority
↑ Lowest Priority

Fig. 8.55(b) Priority of interrupts after command

Rotate in Automatic EOI Mode

The rotate in Automatic EOI Mode (AEOI) works like the rotate on nonspecific EOI (NSEOI) command. The difference between NSEOI and AEOI is that priority routine is done automatically after the last \overline{INTA} pulse of an interrupt request. To enter or exit from this mode, a rotate-in-automatic EOI set command and rotate-in-automatic EOI clear command are provided.

Set Priority Command

The set priority command is used to assign an IR level the lowest priority. All other interrupt levels will be in the fully rested mode based on the newly assigned low priority. The relative priorities of the interrupt levels before the set priority command and after the set priority command are depicted in Fig. 8.56(a) and 8.56(b) respectively. As IR_3 has the highest priority, IR_3 routine be executed next. When the IR_3 routine is executing and set priority command is issued to the 8259A, priorities will be changed. Then IR_6 is the highest and IR_5 as the lowest priority as given in Fig. 8.56 (b).

Rotate on Specific EOI Command

The rotate on specific EOI command is the combination of set priority command and specific EOI command. Just like the set priority command, a specified IR level will be assigned lowest priority. Similar to the specific EOI command, a specified level will be reset in the ISR. In this way the rotate on specific EOI command achieves two operations in only one command.

	IR_7	IR_6	IR_5	IR_4	IR_3	IR_2	IR_1	IR_0	
ISR Status	0	1	0	0	1	0	0	0	Before Command
Priority	7	6	5	4	3	2	1	0	

↑
↑
 Lowest Priority Highest Priority

Fig. 8.56(a) Rotating interrupts priority before set priority command

	IR_7	IR_6	IR_5	IR_4	IR_3	IR_2	IR_1	IR_0	
ISR Status	0	1	0	0	0	0	0	0	After Command
Priority	1	0	7	6	5	4	3	2	

↑
↑
 Lowest Priority
↓
↓
 Highest Priority

Fig. 8.56(b) Rotating interrupt priority after set priority command

Operation Command Word 3 (OCW₃)

The OCW_3 are used to perform the following operations:

1. To read the status of registers (IRR and ISR)
2. To set/reset the special mask and polled modes

The format of OCW_3 is shown in Fig. 8.57.

✓ **RIS** This bit is used to select the In service register (ISR) or the Interrupt Request Register (IRR) and then read register. When $RIS = 0$ and $RR = 1$, IRR is selected. If $RIS = 1$ and $RR = 1$, ISR is selected.

✓ **RR** This bit is used to execute the read register command. When $RR = 0$, the read register command will not issued and no action takes place. If $RR = 1$, the read register command is issued and the state of RIS decide the register to be read.

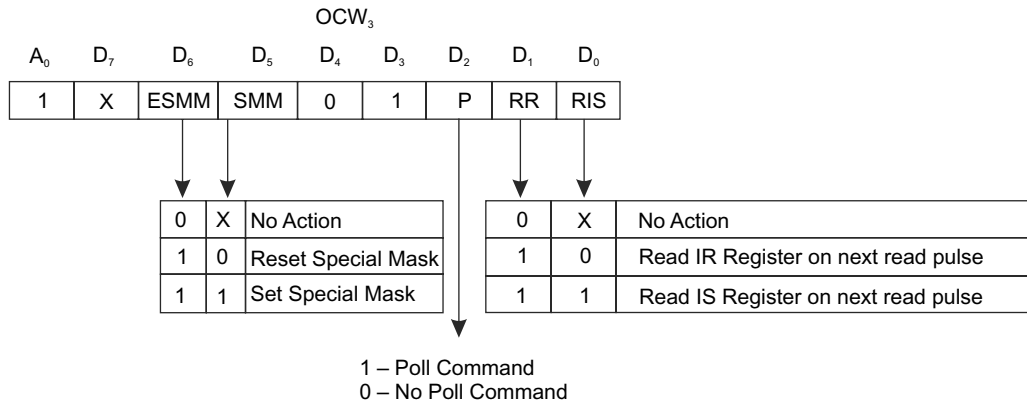


Fig. 8.57 Operation Command Word 3

✓ **P (Poll Mode)** This bit is used for POLL command. When $P = 1$, the poll command is issued. If $P = 0$, the poll command is not issued. In this mode, the interrupting devices seeking services from 8085 are polled one after another and to detect which device has sought for interrupt request? In the polled mode ($P = 1$), 8259A is then read by masking its \overline{RD} and \overline{CS} pins '0'. The ISR bit is set corresponding to the highest level interrupt in the IRR. The poll word is shown in Fig. 8.58.

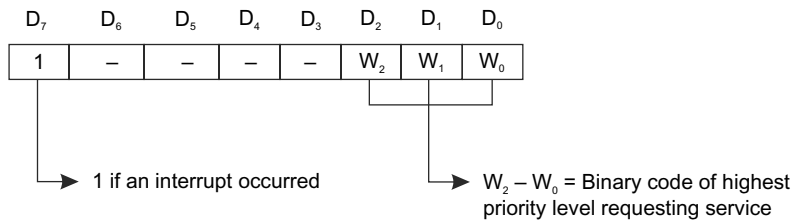


Fig. 8.58 Poll Word

Special Mask Mode (SMM)

The special mask mode enables interrupt from all levels except the level currently in service. When $SMM = 1$, the special mask mode is selected. Once the special mask mode is set, it remains in effect until reset. If $SMM = 0$, the special mask mode is not selected. The special mask mode can be set when ESSM bit enables SMM ($ESSM = 1$) in OCW₃. The special mask mode can be cleared by loading OCW₃ with $ESSM = 1$ and $SMM = 0$.

✓ **ESMM** This bit is used to enable/disable the effect of the special mask mode (SMM). When $ESMM = 1$, the special mask mode is enabled. If $ESMM = 0$, the special mask mode is disabled.

8.6 PROGRAMMABLE PERIPHERAL INTERFACE, 8255

8255 is a programmable peripheral interface IC and is a multipoint input/output device. This is a general-purpose programmable I/O device, which may be used with many different microprocessors. There are 24 I/O pins, which may be individually programmed in 2 groups of 12 and used, in 3 major modes of operation.

The I/O ports can be programmed in a variety of ways as per requirement of the programmer. The features of this device are given below:

Features

- ✦ 24 programmable I/O pins
- ✦ Fully TTL compatible
- ✦ High speed, no 'Wait State' operation with 5 MHz 8085, 8 MHz 80C86 and 80C88
- ✦ Direct bit set/reset capability
- ✦ Enhanced control word read capability
- ✦ 2.5 mA drive capability on all I/O ports
- ✦ Low standby power static CMOS circuit design insures low operating power

Generally, this device is used to read data from an external device and write data into an external device. Some interface circuits are available for reading/writing data in an external device. These interface circuits are called *peripheral interface circuits*. These circuits are also known as programmable I/O ports as I/O ports are programmed to perform specified functions.

8.6.1 Architecture of Intel 8255A

The schematic and pin diagrams of Intel 8255A are shown in Fig. 8.59 and Fig. 8.60 respectively. It is a 40-pin. IC package and operates on a single +5 V dc supply. The 8255A has 24 I/O pins, which may be individually programmed in two groups of twelve input/output lines or three groups of eight lines. The two groups of I/O pins are called Group A and Group B. Each group contains a subgroup of eight bits known as 8-bit port and a subgroup of four bits known as 4-bit port. This IC has three eight-bit ports: Port A (PA_7 – PA_0), Port B (PB_7 – PB_0), and Port C (PC_7 – PC_0). Port C is divided into subgroups such as Port C upper, (PC_7 – PC_4) and Port C lower, (PC_3 – PC_0). Group A consists of Port A and Port C upper. Group B consists of Port B and Port C lower. The internal block diagram of 8255 is depicted in Fig. 8. 61. The pins for various ports are as follows:

PA_0 – PA_7 are 8 pins of Port A, PB_0 – PB_7 are 8 pins of Port B, PC_0 – PC_3 are 4 pins of Port C lower and PC_4 – PC_7 are 4 pins of Port C upper

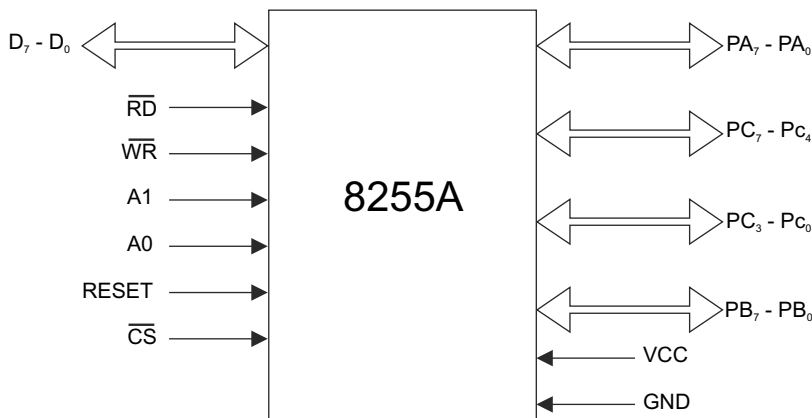


Fig. 8.59 Schematic Diagram of 8255A

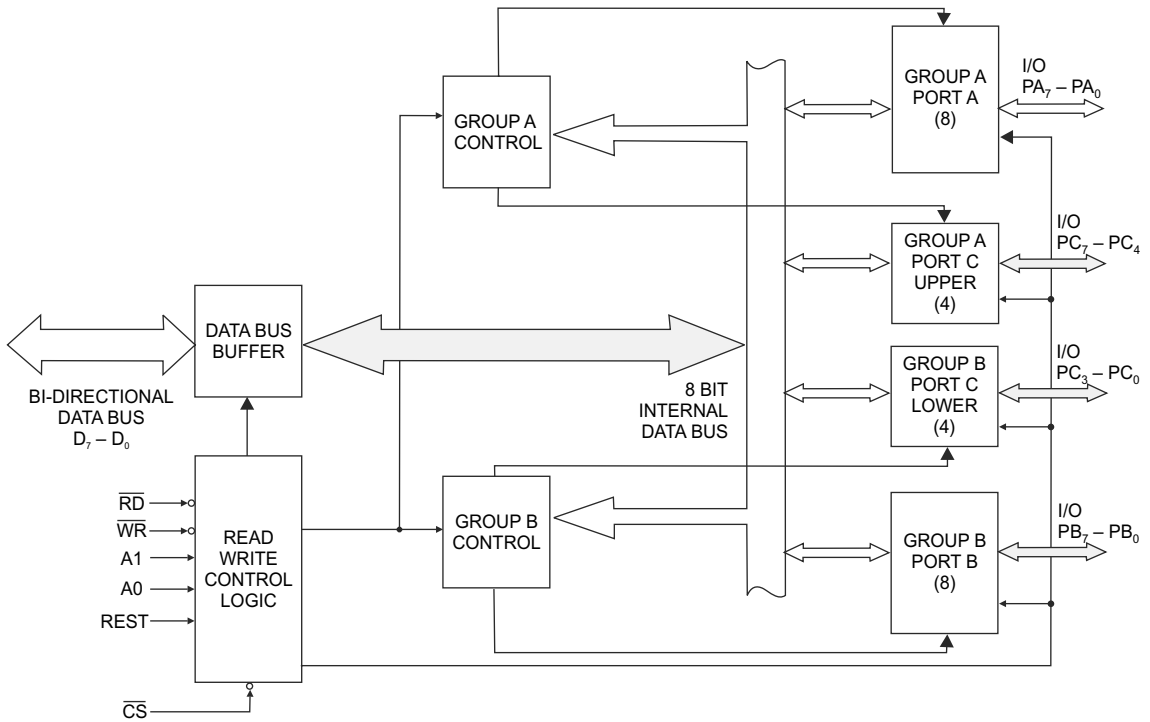


Fig. 8.60 Functional diagram of 8255A

The functional descriptions of pins are as follows:

Symbol	Type	Description
PA ₀ -PA ₇	I/O	PORT A: 8-bit input and output port. Depending upon the control words <i>bus hold</i> <i>highs</i> and <i>bus hold</i> <i>low</i> which are present on this port.
PB ₀ -PB ₇	I/O	PORT B: 8-bit input and output port. This port is used to hold high or low in the same way as Port A.
PC ₀ -PC ₇	I/O	PORT C: 8-bit input and output port. This port may be used as output latch or input buffer.
D ₀ -D ₇	I/O	DATA BUS: The data bus lines are bi-directional three-state pins connected to the system data bus. This three-state bi-directional 8-bit buffer is used to interface the 82C55A to the system data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the microprocessor. Control words and status information are also transferred through the data bus buffer.
RESET	I	RESET: A 'high' on this input initialises the control register to 9BH and all ports (A, B, C) are set to the input mode. 'Bus hold' devices internal to the 82C55A will hold the I/O port inputs to a logic '1' state with a maximum hold current of 400 µA.
\overline{CS}	I	CHIP SELECT: Chip select is an active low input used to enable the 82C55A on to the data bus for CPU communications.

(Contd.)

(Contd.)

\overline{RD} (Read)	I	READ: Read is an active low input control signal used by the CPU to read status information or data via the data bus. When \overline{RD} is LOW, the 8255 sends output data or status information to the microprocessor on the data bus or the microprocessor can read data from the input port of 8255.
\overline{WR}	I	WRITE: Write is an active low input control signal used by the CPU to load control words and data into the 82C55A. When \overline{WR} is LOW, the CPU writes data into the output port of 8255 or writes control word into the control word register of 8255.
A_0 - A_1	I	ADDRESS: These input signals, in conjunction with the \overline{RD} and \overline{WR} inputs, control the selection of one of the three ports or the control word register, A_0 and A_1 are normally connected to the least significant bits of the address bus A_0, A_1 . These lines are used to select input ports and control word register.

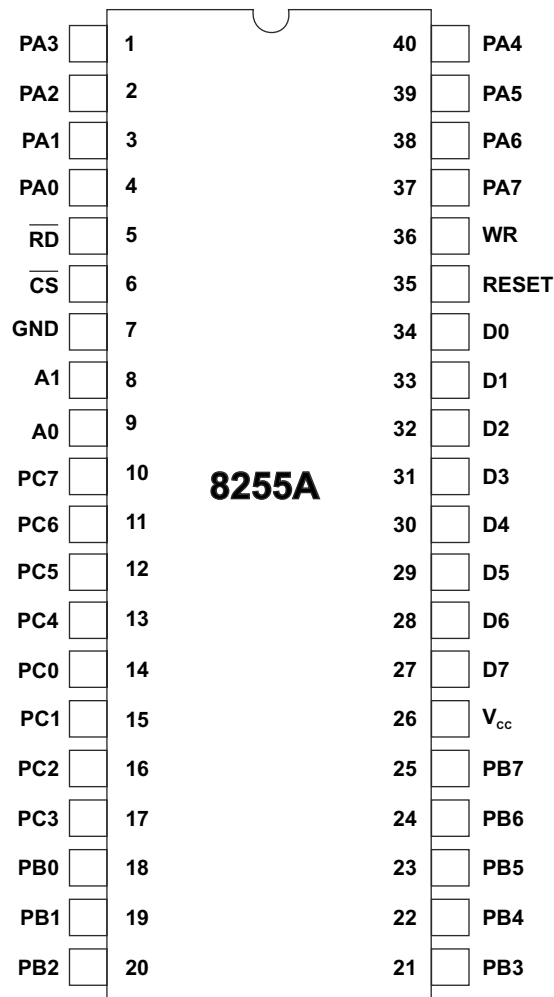


Fig. 8.61 Pin diagram of Intel 8255A

8.6.2 Group A and Group B Controls

The functional configuration of each port can be programmed by the instruction. For this, the CPU stores a control word to the 82C55A. The control word contains information about the mode of operation, bit set, bit reset, etc. The control word initialises the functional configuration of the 82C55A. Each of the control blocks, Group A and Group B, receive 'commands' from the control logic signals; \overline{RD} and \overline{WR} receive 'control words' from the internal data bus and issue the proper commands to their associated ports.

- ✦ Control Group A—Port A and Port C upper (PC_7-PC_4)
- ✦ Control Group B—Port B and Port C lower (PC_3-PC_0)

The control word register can be both written and read as shown in the 'Basic Operation' Table 8.13. The control word format for both read and write operations is depicted in Fig. 8.62. When the control word is read, the bit D_7 will always be a logic '1', as this implies control-word mode information.

Table 8.13(a) 82C55A basic input operation

A_1	A_0	\overline{RD}	\overline{WR}	\overline{CS}	Input Operation (READ Cycle)
0	0	0	1	0	Port A to data bus
0	1	0	1	0	Port B to data bus
1	0	0	1	0	Port C to data bus
1	1	0	1	0	Control word to data bus

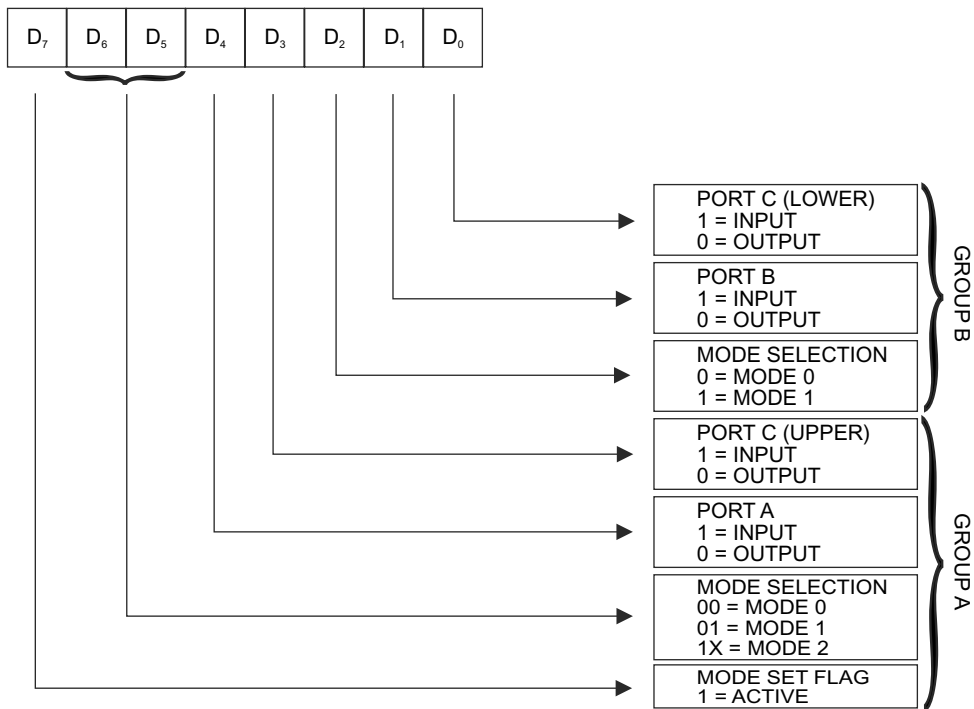


Fig. 8.62 Control word Bits of 8255A

Table 8.13(b) 82C55A basic output operation

A_1	A_0	\overline{RD}	\overline{WR}	\overline{CS}	Output Operation (WRITE)
0	0	1	0	0	Data bus to Port A
0	1	1	0	0	Data bus to Port B
1	0	1	0	0	Data bus to Port C
1	1	1	0	0	Data bus to control

Table 8.13(c) 82C55A disable operation

A_1	A_0	\overline{RD}	\overline{WR}	\overline{CS}	Disable Function
X	X	X	X	1	Data bus to three-state
X	X	1	1	0	Data bus to three-state

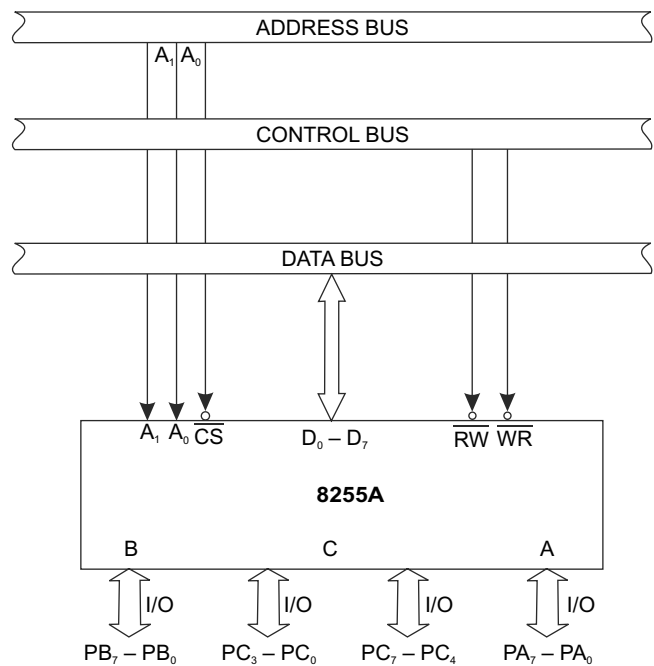
8.6.3 Operating Modes

There are three basic modes of operation of 8255 as follows:

- ✦ Mode 0—Basic input/output
- ✦ Mode 1—Strobed input/output
- ✦ Mode 2—Bi-directional bus

The system software can select the mode of operation. When the reset input becomes 'high', all ports will be set to the input mode with all 24-port lines held at logic 'one' level by internal bus hold devices. When the reset is removed, the 82C55A can remain in the input mode with no additional initialisation required. This eliminates the need to pull up or pull down resistors in all CMOS designs. Then the control word register will contain 9B H. During the execution of the system program, any of the other modes may be selected using a single output instruction. This allows a single 82C55A to service a variety of peripheral devices with a simple software maintenance routine. Any port programmed as an output port is initialised to all zeros when the control word is written.

The 8255A has two 8-bit ports (Port A and Port B) and two 4-bit ports (Port C upper and Port C lower). The modes for Port A and Port B can be separately defined, though Port C is divided into two portions as required by the Group A and Group B definitions.

**Fig. 8.63** Mode 0 operating mode of 8255A

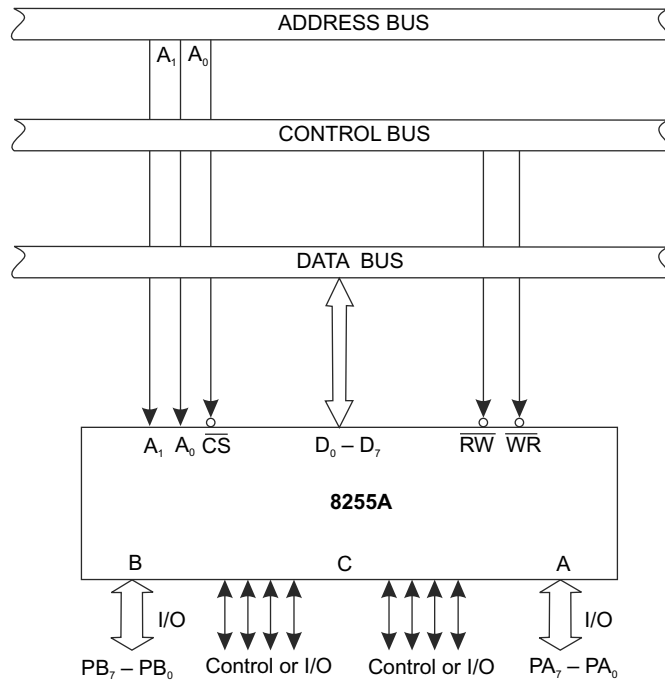


Fig. 8.64 Mode 1 operating mode of 8255A

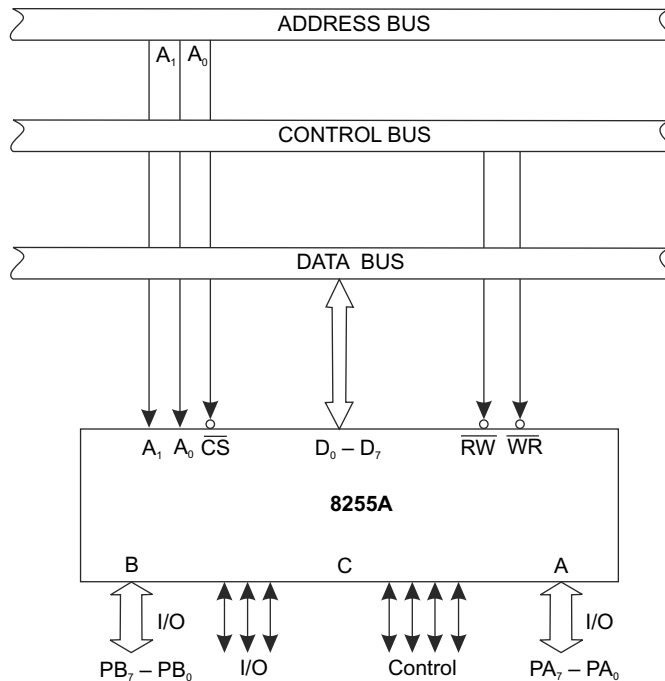


Fig. 8.65 Mode 2 operating mode of 8255A

Mode 0—Basic Input/Output

This functional configuration provides simple input and output operations for each of the three ports. Each of the four ports of 8255 can be programmed to be either an input or output port. No handshaking is required; data is simply written to or read from a specific port.

Basic functional definitions of Mode 0 are as follows:

- ✦ Two 8-bit ports and two 4-bit ports
- ✦ Any port can be input or output
- ✦ Outputs are latched
- ✦ Inputs are not latched
- ✦ 16 different input/output configurations possible

Mode 1—Strobed Input/Output

This is strobed input/output mode of operation. Only Port A and Port B both can be operating in this mode of operation. When Port A and Port B are programmed in Mode 1, six pins from Port C are used as control signals. These control signals are used for handshaking. PC_0 , PC_1 and PC_2 pins of PC lower are used to control Port B and PC_3 , PC_4 , and PC_5 pins of PC upper are used to control Port A. The other pins of Port C, i.e., PC_6 and PC_7 , can be used as either input or output. While Port A is operated as an output port, pins PC_3 , PC_6 and PC_7 are used for its control. The pins PC_4 and PC_5 can be used either as input or output. The combination of Mode 0 and Mode 1 operation is also possible. When Port A is programmed to operate in Mode 1, Port B can also be operated in Mode 0. Figure 8.66 shows the timing diagram of Mode 1 (input).

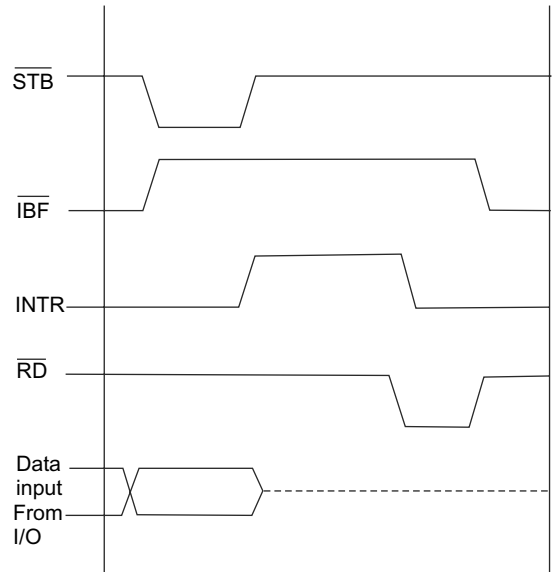


Fig. 8.66 Timing diagram of Mode 1

This functional configuration provides a means for transferring I/O data to or from a specified port in conjunction with strobes or handshaking signals. In Mode 1, Port A and Port B use the lines on Port C to generate or accept these handshaking signals. In Mode 1, the 8255A has two functional groups, namely, Group A and Group B. Each group contains one 8-bit port and a 4-bit control/data port. The 8-bit data port can be either input or output. Both inputs and outputs are latched. The 4-bit port can be used for control and status of the 8-bit port. Figure 8.67 shows the Mode 1 operation of Port A and Port B as input ports. Figure 8.68 shows the Mode 1 operation of Port A and Port B as output ports.

Input Control Signals

- ✓ **\overline{STB} (Strobe Input)** A low on this input loads data into the input latch.
- ✓ **\overline{IBF} (Input Buffer Full F/F)** A high on this output indicates that the data has been loaded into the input latch: in essence, and acknowledgment. \overline{IBF} is set by \overline{STB} input being low and is reset by the rising edge of the \overline{RD} input.
- ✓ **INTR (Interrupt Request)** A 'high' on this output can be used to interrupt the CPU when an input device is requesting service. INTR is set by the conditions: \overline{STB} is a 'one', \overline{IBF} is a 'one' and INTE is a 'one'. It is reset by the falling edge of \overline{RD} . This procedure allows an input device to request service from the CPU by simply strobing its data into the port.

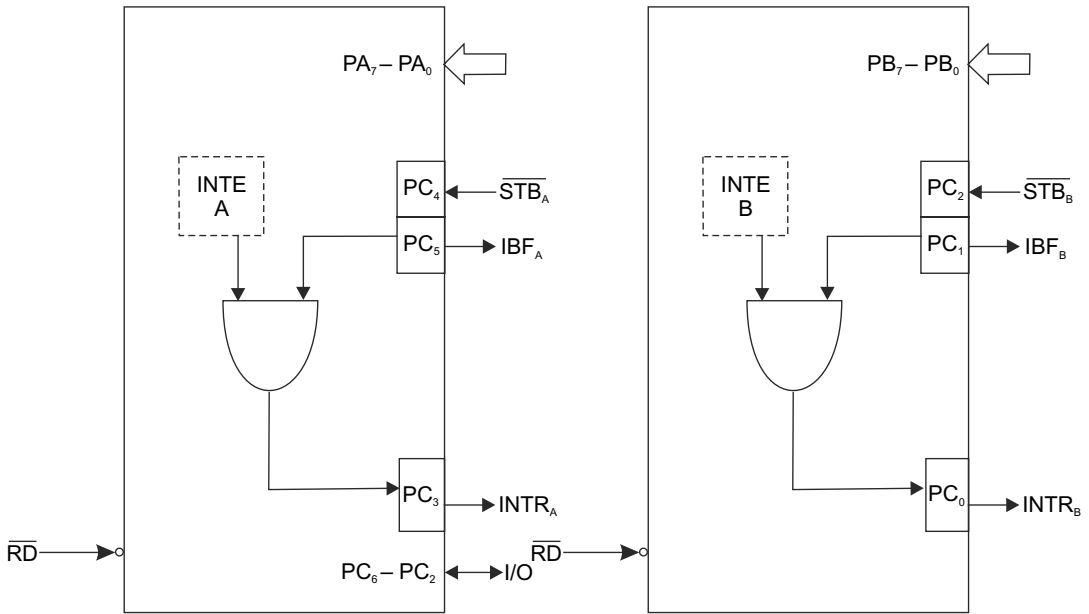


Fig. 8.67 Input mode of port A and port B in mode-1

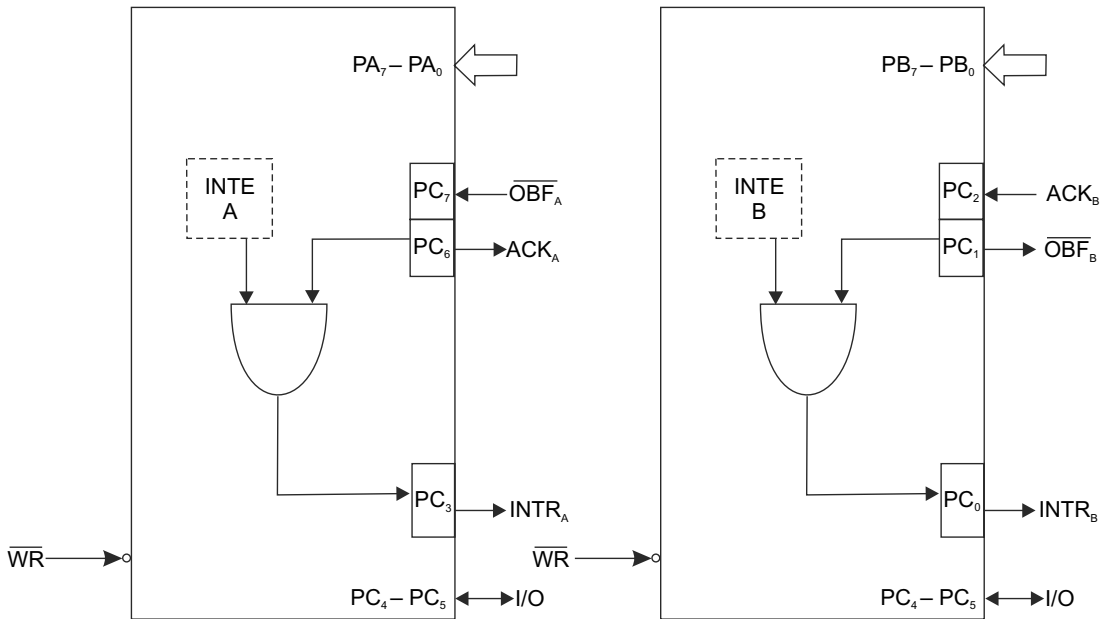


Fig. 8.68 Output of port A and port B in mode-1

- ✓ **INTE A** Controlled by bit set/reset of PC₄.
- ✓ **INTE B** Controlled by bit set/reset of PC₂.

Output Control Signals

- ✓ **\overline{OBF} (Output Buffer Full F/F)** The \overline{OBF} output will go 'low' to indicate that the CPU has written data out to be specified port. This does not mean valid data is sent out of the part at this time since \overline{OBF} can go true before data is available. Data is guaranteed valid at the rising edge of \overline{OBF} . The \overline{OBF} F/F will be set by the rising edge of the \overline{WR} input and reset by \overline{ACK} input being low.
- ✓ **\overline{ACK} (Acknowledge Input)** A 'low' on this input informs the 82C55A that the data from Port A or Port B is ready to be accepted. In essence, a response from the peripheral device indicates that it is ready to accept data.
- ✓ **INTR (Interrupt Request)** A 'high' on this output can be used to interrupt the CPU when an output device has been accepted by data transmitted by the CPU. INTR is set when \overline{ACK} is a 'one', \overline{OBF} is a 'one' and INTE is a 'one'. It is reset by the falling edge of \overline{WR} .
- ✓ **INTE A** Controlled by bit set/reset of PC₆.
- ✓ **INTE B** Controlled by bit set/reset of PC₂.

The strobe line is in a handshaking mode. The user needs to send \overline{OBF} to the peripheral device, generate an \overline{ACK} from the peripheral device and then latch data into the peripheral device on the rising edge of \overline{OBF} . The timing diagram of 8255 in Mode 1 operation of Port A and Port B as output port is illustrated in Fig. 8.69.

Mode 2-Bi-directional Bus

This mode is strobed bi-directional of operation of port with input and output capability. Mode 2 operation is only feasible for Port A. Therefore, Port A can be programmed to operate as a bi-directional port. If Port A is programmed in Mode 2, Port B can be used in either Mode 1 or Mode 0. In this mode of operation, PC₃ to PC₇ pins are used to control signals of Port A.

The basic functional definitions of Mode 2 are

- Used in Group A only
- One 8-bit, bi-directional bus port (Port A) and a 5-bit control port (Port C)

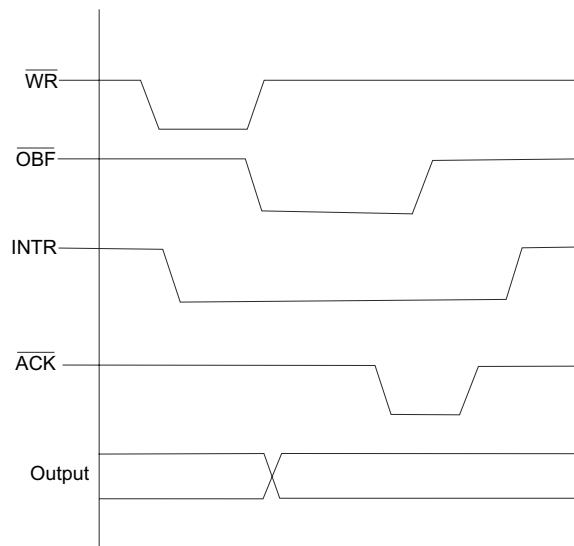


Fig. 8.69 Timing Diagram of Mode 1 when Port A and Port B are output ports

- Both inputs and outputs are latched
- The 5-bit control port (Port C) is used for control and status for the 8-bit, bi-directional bus port (Port A)

✓ **Bi-Directional Bus I/O Control Signal**

- INTR - (Interrupt Request) A high on this output can be used to interrupt the CPU for both input or output operations.

✓ **Output Operations**

- OBF (Output Buffer Full) The OBF output will go 'low' to indicate that the CPU has written data out to Port A.
- ACK (Acknowledge) A 'low' on this input enables the three-state output buffer of Port A to send out the data. Otherwise, the output buffer will be in the high-impedance state.
- INTE 1 (The INTE flip-flop associated with OBF)
Controlled by bit set/reset of PC₄.

✓ **Input Operations**

- STB (Strobe Input) A 'low' on this input loads data into the input latch.
- IBF (Input Buffer Full F/F) A 'high' on this output indicates that data has been loaded into the input latch.
- INTE 2 (The INTE Flip-Flop associated with IBF) Controlled by bit set/reset of PC₄. The timing diagram of Mode 2 operation of 8255 is depicted in Fig. 8.71.

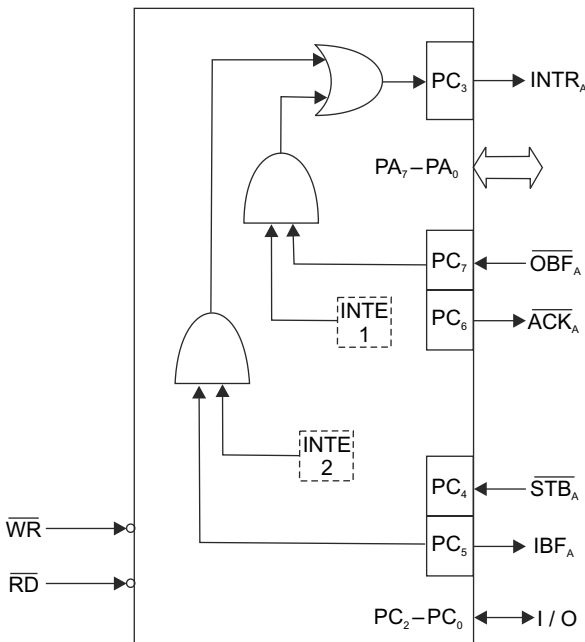


Fig. 8.70 Mode 2

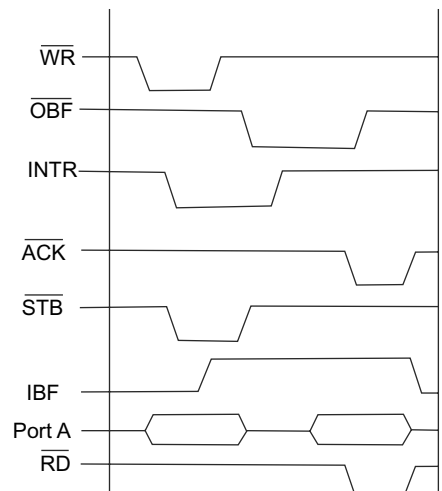


Fig. 8.71 Timing diagram of Mode 2

8.6.4 Single-Bit Set/Reset (BSR) Mode

In this mode, any of the eight bits of Port C can be set or reset using a single output instruction. This feature reduces software requirements in control-based applications.

When Port C is being used as status/control for Port A or B, these bits can be set or reset by using the bit set/reset operation just as if they were output ports. Figure 8.72 shows the bit set/reset format.

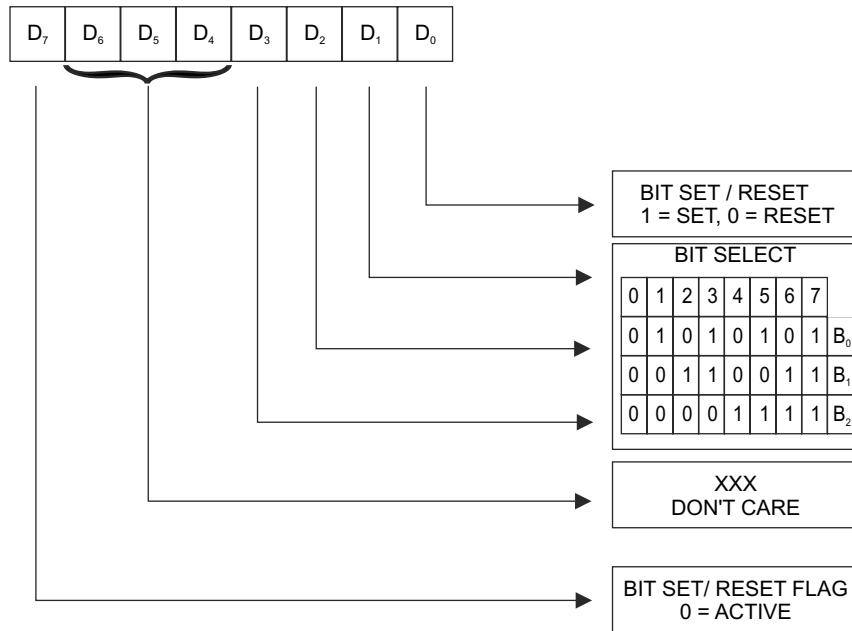


Fig. 8.72 Bit set/reset format

8.6.5 Control Word

The ports of 8255A can be operating any one mode by programming the internal register of 8255A. This internal register of 8255 PPI is known as Control Word Register (CWR). To program the ports of 8255, a control word is formed. Figure 8.72 shows the bits of the control word. Only write operation of the control word register is permissible and no read operation of the control word register is allowed. Writing the control word into control word register, the IC will be configured to operate specified modes of operation. The functional description of the bits of the control word is as follows:

- ✓ **Bit No. D₀** The D₀ bit is used to set Port C lower. When this bit is set to 1, Port C lower is an input port. If the bit is set to 0, Port C lower is an output port.
- ✓ **Bit No. D₁** This bit is used for Port B. When this bit is set to 1, Port B is an input port. If the bit is set to 0, Port B is an output port.
- ✓ **Bit No. D₂** The bit D₂ is used for the selection of the mode operation of Port B. If this bit is set to 0, Port B can be operating in Mode 0. For Mode 1 operation, D₂ is set 1.
- ✓ **Bit No. D₃** It is used for the Port C upper. If the bit is set to 1, Port C upper is an input port. When the bit is set to 0, Port C upper is an output port.

✓ **Bit No. D_4** The bit D_4 sets Port A for input or output operation. When this bit is 1, Port A can be used as input port. When it is 0, Port A becomes output port.

✓ **Bit No. D_5 and D_6** These bits are used to select the operating mode of Port A, for Port A can be operate in Mode 0, Mode 1 and Mode 2. The mode of operation is selected by D_5 and D_6 as given below:

Mode of Port A	Bit No. D_6	Bit No. D_5
Mode 0	0	0
Mode 1	0	1
Mode 2	1	0 or 1

For Mode 2, bit No. 5 is set to either 0 or 1; it is immaterial.

✓ **Bit No. D_7** This bit selects the I/O mode or bit set/reset mode. When it is 1 ports A, B and C are defined as input/output port. If it is set to 0, bit set/reset mode is selected.

Table 8.14 Control words of 8255A for Mode 0 operation

Control word bits								Control word	Port A	Port C lower	Port B	Port C lower
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0					
1	0	0	1	1	0	1	1	9B	input	input	input	input
1	0	0	1	1	0	1	0	9A	input	input	input	output
1	0	0	1	1	0	0	1	99	input	input	output	input
1	0	0	1	1	0	0	0	98	input	input	output	output
1	0	0	1	0	0	1	1	93	input	output	input	input
1	0	0	1	0	0	1	0	92	input	output	input	output
1	0	0	1	0	0	0	1	91	input	output	output	input
1	0	0	1	0	0	0	0	90	input	output	output	output
1	0	0	0	1	0	1	1	8B	output	input	input	input
1	0	0	0	1	0	1	0	8A	output	input	input	output
1	0	0	0	1	0	0	1	89	output	input	output	input
1	0	0	0	1	0	0	0	88	output	input	output	output
1	0	0	0	0	0	1	1	83	output	output	input	input
1	0	0	0	0	0	1	0	82	output	output	input	output
1	0	0	0	0	0	0	1	81	output	output	output	input
1	0	0	0	0	0	0	0	80	output	output	output	output

8.6.6 Examples to Determine Control Word

The determination of control word bit corresponding to a particular port is set to either 1 or 0 depending upon the definition of the port, whether it is to be made an input port or output port. If a particular port is to be made an input port, the bit corresponding to that port is set to 1. For making a port an output port, the corresponding bit for the port is set to 0.

The control words for various configurations of the ports of 8255 for Mode 0 operation are illustrated in Table 8.14. The following examples will illustrate how to make control words:

Example 8.5

Determine control words when the ports of Intel 8255 are defined as follows:

Port A as an input port. Mode of the Port A is Mode 0.

Port B as an input port. Mode of the Port B is Mode 0.

Port C upper and C lower are input ports.

Solution The control word bits for the above definition of ports are as shown in Fig. 8.73.

Bit No. D_0 is set to 1, as the Port C lower is an input port.

Bit No. D_1 is set to 1, as the Port B is an input port.

Bit No. D_2 is set to 0, as the Port B has to operate in Mode 0.

Bit No. D_3 is set to 1, as the Port C upper is an input port.

Bit No. D_4 is set to 1, as the Port A is an input port.

Bit No. D_5 and D_6 are set to 00 as the Port A has to operate in Mode 0.

Bit No. D_7 is set to 1, as the Ports A, B and C are used as simple input/output port.

Thus the control word for above operation is 9B H.

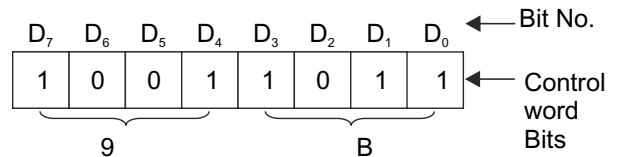


Fig. 8.73

Example 8.6

Determine the control word for the following configuration of the ports of Intel 8255 for Mode 1 operation:

Port A is used as input and operation mode of Port A is Mode 1.

Port B can be used as output and operates in Mode 1.

PC_6 and PC_7 act as input.

Solution

Six pins of Port C, PC_0 – PC_5 are used to control Port A and Port B in Mode 1 operation. PC_0 – PC_2 are used for the control of Port B. Port B can be programmed as an input or output port. When Port A is operated as an input port, PC_3 – PC_5 are used to control this port. In this operating mode, PC_6 and PC_7 may be used as input or output. Figure 8.74 shows the control word bits for the above configuration of the ports. The control word for the above definition of the ports of Intel 8255 is BD H.

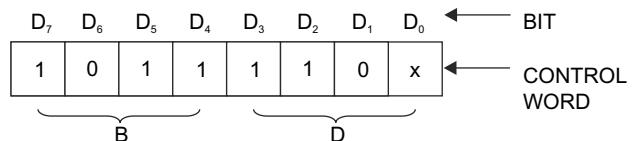


Fig. 8.74

8.6.7 Applications of 8255 PPI

The 8255 PPI IC is a very powerful tool for interfacing peripheral equipment to the microprocessor-based system. It represents the optimum use of available pins and is flexible enough to interface almost any I/O device without the need for additional external logic.

Each peripheral device in a microprocessor-based system usually has a 'service routine' associated with it. The routine manages the software interface between the device and the microprocessor. The functional definition of the 8255 can be programmed by the I/O service routine and becomes an extension of the system software. By examining the I/O device interface characteristics for both data transfer and timing, and matching this information to the examples and tables in the detailed operational description, a control word must

be developed and loaded into control word register to initialise the 8255 IC to get a specified operation. The typical applications of the 8255 are given below:

- ✦ Putting on LED as specified by the designer
- ✦ Generating a square wave at Port A
- ✦ Interfacing A/D converter
- ✦ Keyboard operation
- ✦ Sequential switching of lights
- ✦ Traffic light control
- ✦ Interfacing with dc motors and stepper motors

8.7 8253 PROGRAMMABLE COUNTER/INTERVAL TIMER

In the process control system or automation industry, a number of operations are generally performed sequentially. Between two operations, a fixed time delay is specified. In a microprocessor-based system, time delay can be generated using software. Sequences of operations are also performed based on software. Therefore time delay, sequence and counting can be done under the control of a microprocessor. These most common problems can be solved using the 8253 in any microcomputer system.

The 8253 is a programmable interval timer/counter specifically designed for use in real-time application for timing and counting function such as binary counting, generation of accurate time delay, generation of square wave, rate generation, hardware/software triggered strobe signal, one-shot signal of desired width, etc. The function of 8253 timer is that of a general purpose, multi-timing element which can be treated as an array of I/O ports in the system software.

The generation of accurate time delay using software control or writing instruction is possible. But instead of writing instructions for time delay loop, the 8253 timer may be used for this. The programmer configures the 8253 as per requirements. When the counters of the 8253 are initialising with the desired control word, the counter operates as per requirement. Then a command is given to the 8253 to count out the delay and interrupt the CPU. At the instant it has completed its tasks, the output will be obtained from the output terminal. Multiple delays can easily be implemented by assignment of priority levels in the microprocessor.

The counter/timer can also be used for non-delay in nature such as Programmable Rate Generator, Event Counter, Binary Rate Multiplier, Real Time Clock, Digital One-Shot, and Complex Motor Controller. The 8253 operates in the frequency range of dc to 2.6 MHz while the 8253 uses NMOS technology. The 8253 is compatible to the 8085 microprocessor. Generally, 8253 can be operating in the following modes.

- ✦ Mode 0 Interrupt on terminal count
- ✦ Mode 1 Programmable one-shot
- ✦ Mode 2 Rate generator
- ✦ Mode 3 Square-wave generator
- ✦ Mode 4 Software triggered mode
- ✦ Mode 5 Hardware triggered mode

The pin diagram, block diagram of 8253, interfacing with 8085 microprocessor and operation of each mode have been explained in this section.

8.7.1 Pin Diagram of 8253

The 8253 timer is a 24-pin IC and operates at +5 V dc. It consists of three independent programmable 16-bit counters: Counter 0, Counter 1, and Counter 2. Each counter operates as a 16-bit down counter and each counter consists of clock input, gate input and output as depicted in Fig. 8.75. The schematic block diagram is given in Fig. 8.75. The gate input is used to enable the counting process. Therefore the starting of counting may be controlled by external input pulse in gate terminal. After gate triggered, the counter starts count down. When the counter has completed counting, output signal would be available at the out terminal.

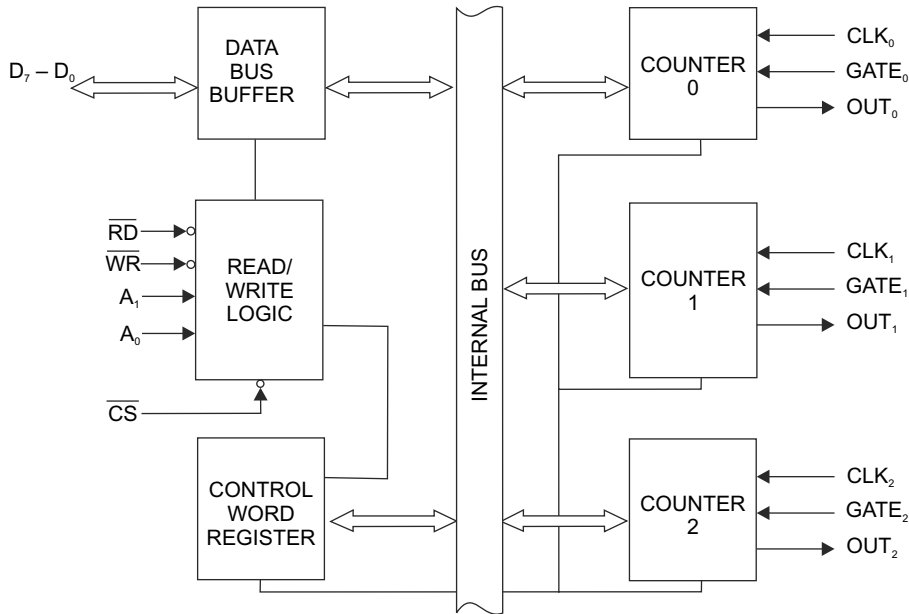


Fig. 8.75 Schematic block diagram of Intel 8253 timer/counter

The programmer can program 8253 using software in any one of the six operating modes: Mode 0, Mode 1, Mode 2, Mode 3, Mode 4, and Mode 5. The pin diagram of 8253 is shown in Fig. 8.76 and Fig. 8.77. The functional descriptions of pins are as follows:

- ✓ **\overline{RD} (Read)** When this pin is low, the CPU is inputting data in the counter.
- ✓ **\overline{WR} (Write)** When this is low, the CPU is outputting data in the form of mode information or loading of counters.
- ✓ **A_0, A_1** These pins are normally connected to the address bus. The function of these pins is used to select one of the three counters to be operated and to address the control word registers for mode selection as given below:

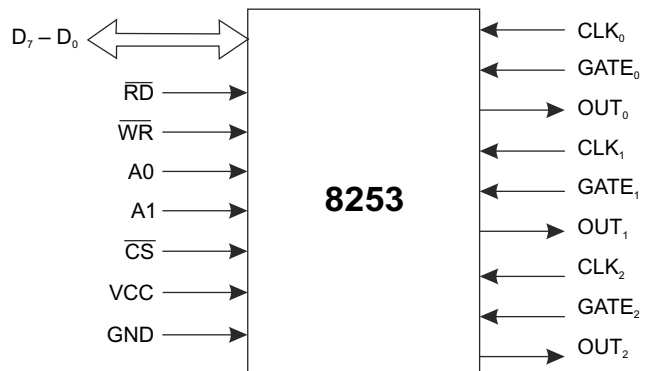


Fig. 8.76 Pin diagram of 8253

A_1	A_0	Selection of Counters and Control word register
0	0	Counter 0
0	1	Counter 1
1	0	Counter 2
1	1	Control Word Register

✓ **\overline{CS} Chip select** A 'low' on \overline{CS} input enables the 8253. No reading or writing operation will be performed until the device is selected. The \overline{CS} input signal is not used to control the actual operation of the counters.

✓ **Data Bus Buffer** The 3-state, bi-directional, 8-bit buffers exist in 8253. These buffers are used to interface the 8253 to the systems data bus D_0 - D_7 lines. Data can be transmitted or received by the buffer upon execution of input and output CPU instructions. The data bus buffer has three basic functions, namely, programming the Modes of the 8253, loading the count registers and reading the count values.

✓ **D_0 - D_7 Bi-directional Data Bus** There are eight data lines through which the control word will be written in the control word register of 8253 counter/timer during programming. The counter will be written and read through the data bus.

✓ **Read/Write Logic** The read/write Logic accepts inputs from the system bus and in turn generates control signals for operation of 8253. This is enabled by \overline{CS} . Therefore, no operation can take place to change the function unless the device has been selected by the system logic. Table 8.15 shows the various functions of 8253 based on the status of pins associated with read/write logic.

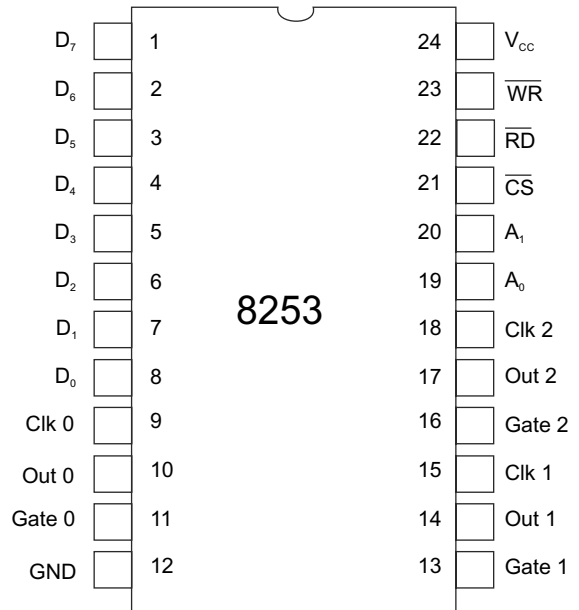


Fig. 8.77 Pin diagram of 8253

Table 8.15 Functions of 8253

\overline{CS}	\overline{RD}	\overline{WR}	A_1	A_0	Function
0	1	0	0	0	Load Counter 0
0	1	0	0	1	Load Counter 1
0	1	0	1	0	Load Counter 2
0	1	0	1	1	Write Mode word
0	0	1	0	0	Read Counter 0
0	0	1	0	1	Read Counter 1
0	0	1	1	0	Read Counter 2
0	0	1	1	1	No Operation 3-State
1	X	X	X	X	Disable 3 State
0	1	1	X	X	No Operation 3-State

- ✓ **CLK₀, CLK₁, CLK₂** CLK₀, CLK₁ and CLK₂ are clock for Counter 0, Counter 1 and Counter 2 respectively. The countdown of the counter takes place on each high to low transition of clock input.
- ✓ **GATE₀, GATE₁, GATE₂** GATE₀, GATE₁ and GATE₂ are gate terminals of Counter 0, Counter 1 and Counter 2 respectively. The function of the GATE in different modes is illustrated in Table 8.16.
- ✓ **OUT₀, OUT₁, OUT₂** OUT₀, OUT₁ and OUT₂ are output terminals of Counter 0, Counter 1 and Counter 2 respectively. The output of the 8253 timer depends upon the mode of operation.

Table 8.16 Different modes of operation corresponding to gate signal

Single status mode	Low or going low	Rising	High
0	Disables counting		Enables counting
1	--	Initiates counting Resets output after next clock	--
2	Disables counting Sets output immediately high	Initiates counting	Enables counting
3	Disables counting Sets output immediately high	Initiates counting	Enables counting
4	Disables counting	--	Enables counting
5	--	Initiates counting	--

8.7.2 Block Diagram

The functional block diagram is illustrated in Fig. 8.75. This device can be divided into functional blocks such as the counter section and the CPU interface section.

Counter Section

The 8253 consists of three programmable independent counters: Counter #0, Counter #1, and Counter #2. These three functional blocks of counters are identical in operation. Each counter consists of a single 16-bit down counter. The counter can operate in either binary or BCD and its input, gate and output are configured by the selection of modes stored in the Control Word Register. The counters are fully independent and each can have separate mode configuration and counting operation, binary or BCD. Each counter can be operated in any of six modes (Mode 0 to Mode 5).

The reading of the contents of each counter is available to the programmer with simple READ operations for event counting applications. Special commands and logic are incorporated in the 8253 so that the contents of each counter can be read without having to inhibit the clock input.

Systems Interface

The \overline{CS} input signal enables the 8253 timer/counter IC. The \overline{RD} and \overline{WR} signals are used to read and write operation respectively. The 8253 can be interfaced with the microprocessor in the same manner as all other peripherals of the family. The 8253 timer/counter, which consists of three counters and the control register, will be treated by the systems software as an array of peripheral I/O ports for all modes of programming.

Figure 8.78 shows the interfacing between microprocessor and 8253 timer. The data bus D₀-D₇ is connected with the data bus of the microprocessor. The select inputs A₀, A₁ of 8253 connect to the A₀, A₁ address bus signals of the CPU. The \overline{CS} can be derived directly from the address bus using a linear select method or it can be connected to the output of a decoder.

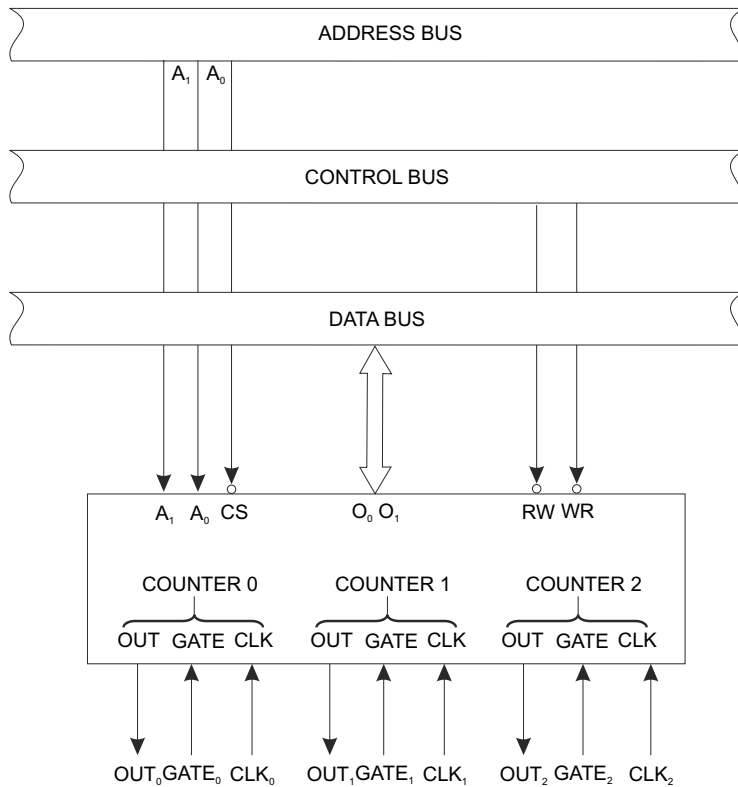


Fig. 8.78 8253 System interface

8.7.3 Control Word Register

The systems software programs are all functions of the 8253. This device is programmed to initialise the counter, select the specified counter mode, and read the count value. A control word must be sent out by CPU to initialise each counter of the 8253 to operate in the desired mode. Before initialisation, the mode count and output of all counters are undefined. The control words program the mode, loading sequence and selection of binary or BCD counting. Once programmed, the 8253 is ready to perform any operation which is assigned to carry out.

The control word register is selected when the pins A₀ and A₁ are 11. Then the control word register accepts information from the data bus buffer and stores it. The information stored in this register controls the operation of each counter. Each counter has three terminals: CLK, GATE and OUT. The output signal depends on the operating mode. The GATE signal controls the output signal.

All of the modes for each counter are programmed by a simple instruction. Writing a control word into the control word register individually programs each counter of the 8253. The control word format is shown below:

Control Word Format

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
SC ₁	SC ₀	RL ₁	RL ₀	M ₂	M ₁	M ₀	BCD

SC-Select Counter

The SC_0 and SC_1 bits of the control word select a counter. The selection of counters is given below:

SC_1	SC_0	Select Counter
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Illegal

RL-Read/ Load

The RL_0 and RL_1 are used to load/read counts as follows:

RL_1	RL_0	Read/Load
0	0	Counter latching operation
0	1	Read/Load least significant byte only
1	0	Read/Load most significant byte only
1	1	Read/Load least significant byte first, then most significant byte

M-Mode

Mode selecting bits M_0 , M_1 and M_2 select any one of six modes as given below:

M_2	M_1	M_0	Mode
0	0	0	Mode 0
0	0	1	Mode 1
×	1	0	Mode 2
×	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

BCD

- 0 Binary counter (16 bits)
- 1 Binary coded decimal (BCD) counter (4 decades)

Reading While Counting

The 8253 timer has an command for latching the content of a counter to read the count value without stopping the counting. This device has a special internal logic to achieve this. The count value can be read after loading a control word in the control word register. The bit pattern for the control word for this operation is as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC_1	SC_0	0	0	×	×	×	×

- ✦ SC_1 and SC_0 —specify counter to be latched.
- ✦ D_5 and $D_4=00$ makes counter latching operation
- ✦ X—indicates 'don't care'.

For example, the control word for reading the count value of Counter 1 is

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
0	1	0	0	x	x	x	x

Bits D₇ and D₆ are 0 and 1 respectively to represent Counter 1.

Bits D₅ and D₄ are 0 and 0 to represent latching operation.

Other bits are either 1 or 0.

Therefore, the control word for the above operation is 40H.

Whenever the microprocessor wants to latch the counter the latching command must be issued every time and then the content of counter read. In this mode of operation, counter operation will not be affected by the latching command. After receiving the latching command, the 8253 latches the content of the counter and stores it in a storage register. Then the microprocessor reads the content of the register by issuing a read instruction. This read operation is generally specified by RL₁ and RL₀ bits of the original mode set command. For counter latching operation, RL₁ and RL₀ are 0 and 0 respectively. If RL₁ = 0 and RL₀ = 1, only read least significant byte LSB of the counter. When RL₁ = 1 and RL₀ = 0, only read MSB of the count, If RL₁ = 1 and RL₀ = 1, read LSB of the count first, and thereafter read MSB of the count.

8.7.4 Operational Modes

The 8253 consists of three independent negative edge triggered 16-bit down counters, namely, Counter 0, Counter 1 and Counter 2. As the counters are fully independent, each counter of 8253 can be programmed in a different mode configuration and counting operation. The programmer must write the control word in the control word register and the load the count value in the selected count register. For writing the mode control word, the counter may be selected in any sequence. Each counter's mode control word register has a separate address so that it can be loaded independently. Usually, the 8253 is available on a microprocessor kit. Sometimes 8253 timer/counter is also connected with a microprocessor kit externally. The clock frequency is about 1.5 MHz, which is available on the kit. If the clock frequency is about 3 MHz, an edge-triggered flip-flop can be used to divide this clock frequency by two to obtain a desired clock frequency for operating 8253 properly.

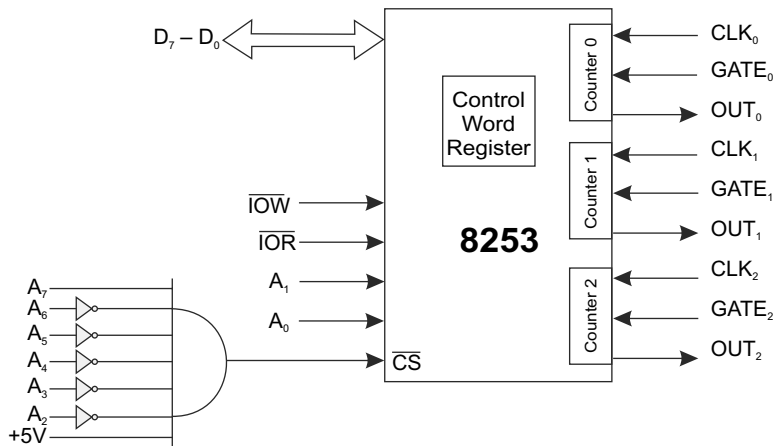


Fig. 8.79 Interfacing the 8253

The port address for control word register and the counters of 8253 are as follows:

A ₇	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀	
1	0	0	0	0	0	1	1	83H Address of control word register
1	0	0	0	0	0	0	0	80H Address of Counter 0
1	0	0	0	0	0	0	1	81H Address of Counter 1
1	0	0	0	0	0	1	0	82H Address of Counter 2

A counter can be used for various applications such as BCD/binary counter, programmable rate generator, square wave generator, hardware/software triggered strobe, programmable one-shot, to generate time delay, etc. Descriptions of some applications are given below.

MODE 0 Interrupt on Terminal Count

Generally, Mode 0 is used to generate accurate time delay under software control. Firstly any one counter of 8253 timer/counter is initialised and loaded with a suitable count to develop the desired time delay. After termination of counting, the counter interrupts the microprocessor. When counter interrupts the microprocessor, the specified operations will be performed by the microprocessor. When the control word is loaded into the control word register, the 8253 timer/counter sets the mode. In Mode 0 operation, the output of the counter becomes initially low after the mode is set. After mode set operation, the selected counter must be loaded by the desired count value N .

In this mode of operation, GATE is kept high. Therefore, just after loading the count registers, the counter starts to decrement. When counting is going on, the counter output terminal OUT remains low. As soon as the terminal count reaches 0, the output becomes high. The output remains high until the counter is reloaded or a new count value is loaded into the counter. When the count is reloaded or a new count is loaded, the counting restarts from new count value and again OUT becomes low. The timing diagram for Mode 0 operation is shown in Fig. 8.80.

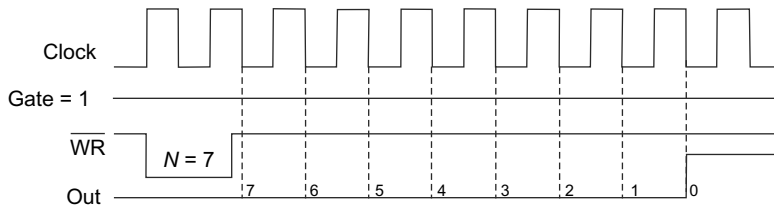


Fig. 8.80 Mode 0-interrupt on terminal count

While counting is going on, GATE becomes low suddenly and the counter stops counting operation. After some time if the GATE returns to 1, counting is resumed from the GATE count value at which the counting discontinued. The count value can be changed at any time. A new count value can also be loaded while counting is going on. But the changes will be effective only after the next GATE trigger. This mode of operation can be used to generate accurate time delay. This can also be used to perform specified operation after some delay. The output of OUT terminal may be used to interrupt the microprocessor. The examples of Mode 0 operation are given below:

Example 8.7

Write a subroutine program to initialise Counter 0 in Mode 0 with count value 8000H.

Solution

The control word for Mode 0 operation as given below:

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
0	0	1	1	0	0	0	0 = 30H

D₇ and D₆ have been set to 00 to initialise Counter 0.

D₅ and D₄ have been set 11 to load the least significant byte of the count first, then the most significant byte.

D₃, D₂, D₁ are set to 000 for Mode 0 operation.

D₀ is set to 0 as counting is to be done in binary.

The address of the control register is 83H and the address of Counter 0 is 80H

The program for loading the control word and 16-bit number in Counter 0 is given below:

PROGRAM 8.5

Memory address	Machine Codes	Mnemonics	Operands	Comments
9000	3E, 30	MVI	A, 30	Control word for Mode 0 to initialise counter 0
9002	D3, 83	OUT	83	Write the control word into control word register
9004	3E, 00	MVI	A, 00 H	Least significant byte of the count
9006	D3, 80	OUT	80	Load counter 0 by 00H, LSB of count
9008	3E, 80	MVI	A, 80	Most significant byte of the count
900A	D3, 80	OUT	80	Load counter 0 by 80H, MSB of count

After execution of the program, the gate signal becomes ONE

Example 8.8

Counter 0 of 8253 timers is used in Mode 0 to perform addition of an array after certain delay. Assume count value for delay = 2000H. After completion of counting, the counter interrupts the microprocessor to jump a memory location for addition of two 8-bit numbers.

Solution

It is a very simple task to jump from one memory location to another memory location. But, here the task is to jump from one memory location to another memory location after the microprocessor is interrupted. For this, all interrupts must be enabled. Then the content of the accumulator will be enable RST 7.5, 6.5 and 5.5 for SIM instruction as given below:

7	6	5	4	3	2	1	0
SOD	SOE	X	R7.5	MSE	M7.5	M6.5	M5.5
0	0	0	0	1	0	0	0 = 08H

The control word for Mode 0 operation of Counter 0 is as follows.

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
0	0	1	1	0	0	0	0 = 30 H

D₇ and D₆ are set to 0 and 0 respectively to initialise Counter 0.

D₅ and D₄ have been set 11 to load the least significant byte of the count first, after that most significant byte must be loaded.

D₃, D₂, D₁ are set to 0 0 0 for Mode 0 operation.

D₀ is set to 0 as counting is to be done in binary. Connect OUT terminal of Counter 0 with RST 7.5 of microprocessor.

PROGRAM 8.6

<i>Memory address</i>	<i>Machine Codes</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	FB	EI		All interrupts are enable
8001	3E, 08	MVI	A, 08	Load bit pattern to accumulator to enable RST 7.5, 6.5 and 5.5
8003	30	SIM		RST 7.5, 6.5 and 5.5 are enable
8004	3E, 30	MVI	A, 30	The control word for Mode 0 to initialise counter 0
8006	D3, 13	OUT	83	Write control word into control word register
8008	3E, 00	MVI	A, 00 H	Load least significant byte of the count in Counter 0
800A	D3, 11	OUT	80	Load Counter 0 by LSB of count value
800C	3E, 20	MVI	A, 20	MSB of count value
800E	D3, 11	OUT	80	Load Counter 0 by MSB of count value
8010	C3, 10, 80	JMP	8010	

As soon as the count value is loaded in Counter 0, the counter starts decrementing. When the counting has been completed, RST7.5 interrupts microprocessor and the program jumps to the memory location 003C. After that the monitor transfers the program from 003C location to 9000H. Then the jump instruction which is stored at 9000H location can transfer the program from 9000 H to the starting address of the subroutine, 8500H. The subroutine program has given below:

```

9000      C3, 00, 85  JMP      8500      Jump to subroutine at 8500H
SERVICE SUBROUTINE
8500      21 00 86   LXI H      8600      Load 8600 in H-L register pair
8503      7E        MOV      A,M      Move content of 8600 into Accumulator
8504      23        INX      H        Increment H-L pair
8505      86        ADD      M        Add the 2nd data with accumulator
8506      32 02 86   STA      8602      Store result in 8602 memory location
8509      FB        EI        Enable all interrupts
850A      C9        RET
DATA
8600      22        RESULT
8601      44        8602      66

```

After execution of subroutine, the program returns from the subroutine to the main program. Before it returns to the main program, all interrupts must be enabled so that any additional interrupts can be acknowledged.

MODE 1 Programmable One-Shot

In this mode, the counter acts as a retriggerable and programmable one-shot. The rising edge trigger signal is applied to GATE terminal of the counter. In this mode, initially OUT is high after the mode is set. The control word and the count value must be loaded into the counter. When the mode set operation is done and the counter is loaded by a count value, the counter starts decrement count. At the first negative edge of the clock after the rising edge trigger signal of the GATE input,

output becomes low. Then the output (OUT) will be low for a number of count clock cycles. After completion of count, the output becomes high as depicted in Fig. 8.81. The width of the output pulse depends upon the count value. Consequently, the width of the output pulse can be varied by changing the count value in the program. Therefore, this mode of operation can be called programmable one-shot.

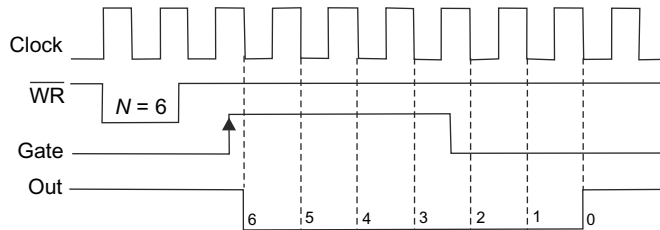


Fig. 8.81 Mode 1 Programmable one shot

Example 8.9

Write a program to operate Counter 0 of 8253 timer/counter in MODE 1

Solution

The control word for Counter 0 and Mode 1 was determined as follows:

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
0	0	0	1	0	0	1	0 = 12 H

D₇ and D₆ are set to 0 and 0 respectively to select Counter 0.

D₅ and D₄ are also set to 0 and 1 respectively for loading only Least Significant Bits (LSB) of the count.

D₃, D₂ and D₁ are set to 001 for Mode 1.

D₀ is set to 0 for binary counting.

PROGRAM 8.7

Memory address	Machine Codes	Mnemonics	Operands	Comments
8000	3E, 12	MVI	A, 12H	Load control word to initialise counter 0 in Mode 1.
8002	D3, 83	OUT	83	Write the control word in control word register
8004	3E, 10	MVI	A, 10	Get count
8006	D3 80	OUT	80	Load Counter 0 with the count

MODE 2 RATE Generator

In Mode 2, the counter behaves as a divide by *N* counter. Generally, it is used to generate a real-time clock interrupt. The control word and the count value are loaded into the control word register and counter respectively. After the mode is set, the output of the counter will be initially high. If the counter is loaded by a count of value *N*, the output remains high for (*N*-1) clock pulses. After (*N*-1) clock pulses, output will be low for one clock pulse and then output becomes high again as shown in Fig. 8.82. Thereafter the count value *N* is reloaded into counter and the output remains high for (*N*-1) clock pulses and will be low

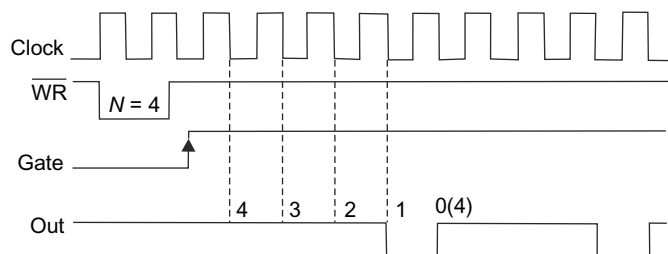


Fig. 8.82 Mode 2 Rate generator

for one clock pulse. If a new count value is reloaded into the count register between output pulses, the present period is not affected. The subsequent period reflects the new value of the count.

Example 8.10

Counter 1 of 8253 time operates in MODE 2, divide by N binary counter. Assume $N = 6$ in decimal.

Solution

When Counter 1 of 8253 time operates in Mode 2 and divide by N binary counter, the control word are as follows:

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
0	1	0	1	0	1	0	0 = 54 hex

D₇ and D₆ are set to 0 and 1 respectively to initialise Counter 1.

D₅ and D₄ have been set 0 and 1 to load the only least significant byte of the count.

D₃, D₂, D₁ are used for mode select. Mode 0 operation D₃, D₂, D₁ are set to 0 1 0.

D₀ is set to 0 as counting is to be done in binary.

54H is the control word for Mode 2 operation

In counter 1, GATE₁, OUT₁ and CLK₁ terminals are available at microprocessor kit. Apply + 5V to GATE₁ of counter 1 to make it high. Clock has been connected to terminal CLK₁. The clock frequency must be the desired frequency, approximately 1.5 MHz.

PROGRAM 8.8

Memory address	Machine Codes	Mnemonics	Operands	Comments
8000	3E, 54	MVI	A, 54H	Load control word for counter 1, MODE 2, binary counting.
8002	D3, 83	OUT	83	83H is address for writing control word in control word used.
8004	3E, 06	MVI	A, 06 H	Load count value N. A = 06H
8006	D3 81	OUT	81	81 H is the address for counter-1
8008	76	HLT	Stop	

MODE 3 Square-Wave Generators

In this mode, the counter operates as a square-wave generator. To operate Counter 1 in Mode 3, the control word must be loaded into control word register. After mode setting the counter is loaded by a count of value N . When the GATE becomes high, the counter starts counting. The output remains high for half of the count value, $N/2$ clock pulses and it remains low for the rest of the count values or $N/2$ clock pulses. Therefore, a continuous square wave of specified period can be generated at output terminal as depicted in Fig. 8.83.

For even values of N , the output is high for $N/2$ clock pulses and low for next $N/2$ clock pulses. For odd values of N , the output remains high for $\frac{N+1}{2}$ clock pulses and low for remaining clock pulses. By changing the count value, time period of square wave can be controlled. After completion of count, the output state is changed and the counter is automatically reloaded with the full count and the above process will be repeated.

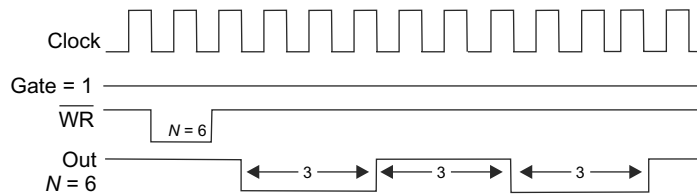


Fig. 8.83(a) Mode 3-square wave generator with even count value

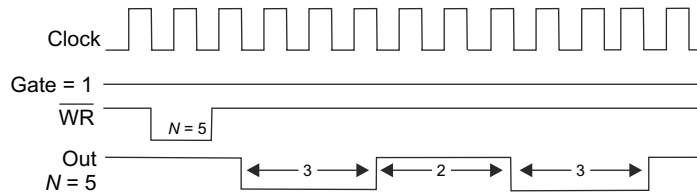


Fig. 8.83(b) Mode 3-square wave generator with odd count value

Example 8.11

Write a program to use Counter 1 of 8253 in Mode 3 as a square wave generator

Assume $N = 16$. The counter operates as a binary counter.

Solution

The counter 1 has been used as a square-wave generator and the control word for this operation is

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
0	1	0	1	0	1	1	0 = 56 H

D₇ and D₆ have been set to 0 and 1 to select counter 1.

D₅ and D₄ are set to 0 and 1 for loading only LSB of the count.

D₃, D₂ and D₁ are set to 011 for Mode 3 operation.

D₀ is set to 0 for binary counting.

PROGRAM 8.9

Memory address	Machine Codes	Mnemonics	Operands	Comments
9000	3E,56	MVI	A, 56H	Load the control word for Mode 3 in control word register to initialise Counter 1
9002	D3, 83	OUT	83	Write in control word register
9004	3E, 10	MVI	A, 10 H	Load the count value for binary counting.
9006	D3 81	OUT	81	Load counter 1 with the count value
9008	76	HLT	Stop	

If $N = 16$, the output will be high for 8 clock cycles and then it will be low for the next 8 clock cycles. When $N = 17$, the output remains high for 9 clock pulses and then low for remaining 8 clock pulses. A continuous square wave can be generated at output as the counter is reloaded by same value and the timer repeats the process repeatedly.

MODE 4 Software Triggered Strobe

In software triggered strobe operation, the counter output will be high after mode is set. The GATE is always high for this mode of operation. When the counter is loaded with a count value, the counter starts counting. As soon as the count value is loaded into the counter register, it triggers the generation of the strobe. Therefore, this mode of operation is known as software triggered strobe. When the counter content becomes 0, the output will be low for one clock period and thereafter, output will be remaining high as illustrated in Fig. 8.84.

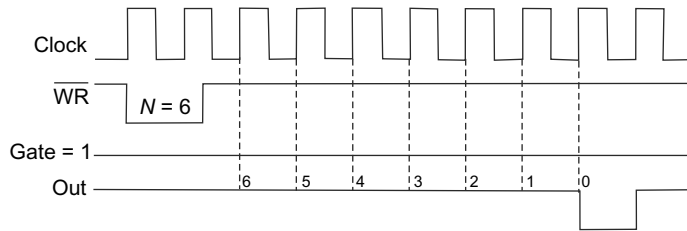


Fig. 8.84 Mode 4—Software triggered strobe

Example 8.12

Write a program using Counter 2 of 8253 in Mode 4.

Solution

The control word for Counter 2 for Mode 4 operation is as follows:

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
1	0	1	1	1	0	0	0 = B8 hex

D₇ and D₆ are set to 10 to select Counter 2.

D₅ and D₄ have been set to 11 to load the LSB of the count value first, then load MSB of the count value.

D₃, D₂ and D₁ have been set to 100 for Mode 4 operation.

D₀ is set to 0 for binary counting.

PROGRAM 8.10

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	3E,B8		MVI	A, B8H	Load control word to initialise Counter 2 for Mode 4 operation.
8002	D3, 83		OUT	83	Write control word in control word register
8004	3E, 05	LOOP	MVI	A, 05	Load LSB of the count.
8006	D3, 82		OUT	82	Load Counter 2 with LSB of the count.
8008	3E, 00		MVI	A, 00	Load MSB of the count
800A	D3, 82		OUT	82	Load counter 2 with MSB of the count
800C	C3, 04, 80		JMP	LOOP	

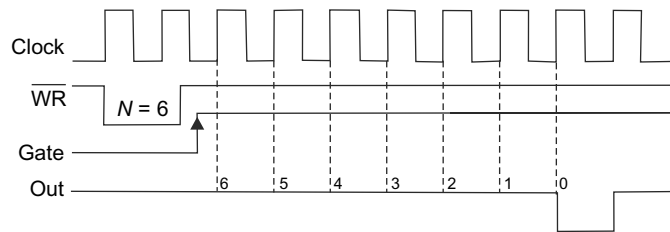


Fig. 8.85 Mode 5—hardware triggered strobe

MODE 5 Hardware Triggered Strobe

In this mode of operation, initially the output is high. The control word and count value are loaded in the counter register. Here GATE input acts as a trigger signal. When low to high transition of the GATE input occurs, the counter starts to decrement the count value and the output becomes initially high. The output goes low for one clock period, when the counter completes the count value. As the low to high transition of the GATE input or the rising edge of GATE trigger the counter, this mode is called as hardware triggered strobe.

This mode of counter operation is also retriggerable. When the GATE input becomes low to high again, the counter is reloaded by the count value N and the counter starts to decrement the count value once again. The output will also be low for one clock period on terminal count.

Example 8.13

Write a program to use Counter 2 of 8253 in MODE 5 operation.

Solution

The control word for Counter 2 in Mode 5 is as follows:

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
1	0	1	1	1	0	1	0 = BA H

D₇ and D₆ are set to 1 and 0 respectively to select Counter 2. D₅ and D₄ are set to 11 for loading least significant bit first, then most significant bit. D₃, D₂ and D₁ are set to 101 for MODE 5 operation. D₀ is set to 0 for binary counting.

PROGRAM 8.11

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	3E,BA		MVI	A, BA	Load control word to initialise Counter 2 in Mode 5 operations.
8002	D3, 83		OUT	83	Write the control word into control word register
8004	3E, 06		MVI	A, 06	Load LSB of the count.
8006	D3, 82		OUT	82	Load the Counter 2 with LSB of the count value.
8008	3E, 00		MVI	A, 00	Get MSB of the count
800A	D3, 82		OUT	82	Load MSB of counter into Counter 2
800C	3E, 80	LOOP	MVI	A,80 H	Initialise ports of 8255.2
800E	D3, 0B		OUT	0B	Load the Counter 2 with MSB of the count value

(Contd.)

(Contd.)				
8010	3E, 00	MVI	A, 00	Generate a pulse output at PC ₀ terminal, which is connected to GATE of 8253.
8012	D3, 0A	OUT	0A	
8014	3E, 01	MVI	A, 01	
8016	D3, 0A	OUT	0A	
8018	C3, 0C, 80	JMP	LOOP	

SUMMARY

- In the beginning of this chapter, the basic concept of memory and its organization have been discussed. Memory can be classified into two groups such as Read Only Memory (ROM) and Random Access Memory (RAM). ROM is nonvolatile and data is permanently stored in this memory and once ROM is programmed, its stored data cannot be changed. RAM is of volatile type and can be used to read and write. This is known as the *user memory*. List of ROM and RAM ICs are incorporated in this chapter.
- The memory addresses assigned to a memory IC in any microprocessor-based system are known as memory maps. Generally, the chip select logic and address decoding are used for the assignment of memory addresses.
- There are two types of address mapping such as I/O mapped I/O and memory mapped I/O. In I/O mapped I/O, the microprocessor uses an 8-bit address to identify an I/O device, and IN and OUT instructions for data transfer. In memory mapped I/O, the microprocessor uses a 16-bit address to identify an I/O device, and memory instructions such as LDA, STA, MOV M, R are used for data transfer.
- In this chapter, the concept of interfacing of I/O devices with 8085 microprocessor has been discussed. This chapter also covers the execution of I/O instructions, and the data transfer process in the peripheral devices.
- Interrupt is a process where an external device can get the attention of the microprocessor for communication and data transfer. Actually the interrupt process starts from peripheral devices and it is an asynchronous type data transfer as it can be started at any time.
- Interrupts are two types: maskable interrupt and nonmaskable interrupt. The maskable interrupts can be delayed or rejected but the nonmaskable interrupts cannot be delayed or rejected.
- Interrupts can also be classified into vectored and nonvectored interrupts. In vectored interrupts, the address of the interrupt service routine is hard-wired but in nonvectored interrupts, the address of the interrupt service routine needs to be supplied externally by the device. In the 8085 interrupts section, all types of interrupts and operation of EI, DI, SIM and SIM instructions are explained.
- In this chapter, architecture, functional block diagram, pin diagram and operation of programmable devices such as Programmable Peripheral Interface 8255, Programmable Timer 8253, Programmable Interrupt Controller 8259A, have been discussed. These general-purpose devices have been designed to provide services for different purposes in I/O communication and data transfer. Each device has a control word register and operation instructions. Therefore, each device must be initialised by writing control word in the control word register for appropriate operation. Interfacing of these devices with microprocessors has been incorporated in this chapter.
- 8255 Programmable Peripheral Interface (PPI) and 8253 timer/counter are two widely used

(Contd.)

(Contd.)

general-purpose programmable devices and they are compatible with any microprocessor. The 8255 PPI has three programmable ports, one of which can be used for bi-directional data transfer. The 8253 timer/counter has three 16-bit independent timers, which can operate in various modes such as programmable rate generator, square wave generator, software and hardware trigger strobe, etc.

MULTIPLE-CHOICE QUESTIONS

- 8.1 How many address lines are required to access 1 MB RAM using microprocessor?
 (a) 16 (b) 8
 (c) 20 (d) 12
- 8.2 What are the control signals of 8085 microprocessor used to interface I/O devices?
 (a) IO / \overline{M} , \overline{RD} , \overline{WR} (b) IO / \overline{M}
 (c) \overline{RD} (d) \overline{WR}
- 8.3 To design a 4 KB RAM with 1024 byte RAM ICs, how many ICs are required?
 (a) 4 (b) 8
 (c) 2 (d) None of these
- 8.4 In I/O mapped I/O device interfacing, the device has
 (a) 16-bit address lines
 (b) 8-bit address lines
 (c) 20-bit address lines
 (d) 12-bit address lines
- 8.5 In memory mapped I/O device interfacing, the device has
 (a) 16-bit address lines
 (b) 8-bit address lines
 (c) 20-bit address lines
 (d) 12-bit address lines
- 8.6 When the starting address of 4K RAM is 8000H, the memory map will be
 (a) 8000H–9000H (b) 8000H–8500H
 (c) 8000H–9500H (d) 8000H–A000H
- 8.7 If the starting address of 2K RAM is 4000H, the end address of memory map will be
 (a) 4800H (b) 8400H
 (c) 8000H (d) 4000H
- 8.8 A 64K bit memory device can be organized as
 (a) $64K \times 1$ (b) $14K \times 4$
 (c) $8K \times 8$ (d) all of these
- 8.9 How many address lines are used to identify an I/O port in I/O mapped I/O and memory mapped I/O?
 (a) 16-bit and 8-bit address lines
 (b) 8-bit and 16-bit address lines
 (c) 8-bit and 20-bit address lines
 (d) 16-bit and 12-bit address lines
- 8.10 The number of $4K \times 4$ memory devices are required for $16K \times 8$ memory
 (a) 2 (b) 3
 (c) 4 (d) 8
- 8.11 The memory map of a 4K-byte memory chip begins at the location 3000H. The last location of memory address and number of pages in the chip are
 (a) 2000, 2 (b) 3000, 4
 (c) 4000, 8 (d) 5000, 9
- 8.12 The number of address lines are required to access 2M byte of data from microprocessor
 (a) 16-bit address lines
 (b) 8-bit address lines
 (c) 20-bit address lines
 (d) 21-bit address lines
- 8.13 What is the vector address of a software interrupt?
 (a) vector address = interrupt number \times 8
 (b) vector address = interrupt number \times 16
 (c) vector address = interrupt number \times 12
 (d) vector address = interrupt number \times 4

- 8.14 If interrupt instruction RST 5 is executed, the program will jump to memory location
 (a) 2000H (b) 0020H
 (c) 0028H (d) 0016H
- 8.15 Which is the highest priority interrupt?
 (a) TRAP (b) RST 6.5
 (c) RST 5.5 (d) RST 7.5
- 8.16 SIM instruction is used to
 (a) enable RST 7.5, 6.5 and 5.5
 (b) disable RST 7.5, 6.5 and 5.5
 (c) enable or disable RST 7.5, 6.5 and 5.5
 (d) None of these
- 8.17 If A_0 and A_1 pins of 8255 are 00, which port will be selected?
 (a) Port A (b) Port B
 (c) Port C (d) None of these
- 8.18 To select Port B of 8255 A_0 and A_1 are
 (a) 00 (b) 01
 (c) 10 (d) None of these
- 8.19 When Port A input, Port B and Port C output, the control word of 8255 is
 (a) 80H (b) 90H
 (c) 85H (d) 86H
- 8.20 8259 is a
 (a) programmable interrupt controller
 (b) DMA controller
 (c) programmable keyboard display interface
 (d) programmable counter
- 8.21 8259 can handle
 (a) 8-vectored priority interrupt controller
 (b) 18-vectored priority interrupt controller
 (c) 16-vectored priority interrupt controller
 (d) 6-vectored priority interrupt controller
- 8.22 In cascade mode, 8259 can handle
 (a) up to 64-vectored priority interrupt
 (b) up to 46-vectored priority interrupt
 (c) up to 60-vectored priority interrupt
 (d) up to 40-vectored priority interrupt
- 8.23 Which pin used to control the output of counter 2 of 8253 in Mode 2?
 (a) GATE 0 (b) GATE 1
 (c) GATE 2 (d) GATE 3
- 8.24 What are the bits of the control word to select a counter?
 (a) SC_0 SC_1 (b) RW_0 RW_1
 (c) M_0 M_1 M_2 (d) BCD, RW_0 & RW_1
- 8.25 The control word register is selected by the read/write logic when
 (a) $A_0A_1 = 11$ (b) $A_0A_1 = 01$
 (c) $A_0A_1 = 10$ (d) $A_0A_1 = 00$
- 8.26 8253 has
 (a) 6 modes of operation
 (b) 5 modes of operation
 (c) 4 modes of operation
 (d) 3 modes of operation
- 8.27 8253 is capable to handle clock frequency at
 (a) 1 MHz (b) 2 MHz
 (c) 3 MHz (d) 4 MHz

SHORT-ANSWER TYPE QUESTIONS

- 8.1 What are the types of memory? Explain the comparison between different types of memory.
- 8.2 What are the advantages and disadvantages of memory mapped I/O over I/O mapped I/O?
- 8.3 What are the interrupt pins of 8085/8086/808 microprocessor ?
- 8.4 Give a list of applications of the 8253 timer.
- 8.5 What do you mean by maskable and nonmaskable interrupts?.
- 8.6 Define priority interrupts?
- 8.7 What is programmable interrupt controller?
- 8.8 What happens when the RESET pin of 8255 is made high?
- 8.9 What are the types of rotating priority mode of interrupt?.
- 8.10 Mention the different modes of operation of 8253 IC.
- 8.11 What the applications of 8255 PP1?

REVIEW QUESTIONS

- 8.1 Explain memory mapped I/O and I/O mapped I/O. Write the comparison between memory mapped I/O and I/O mapped I/O. What are the instructions available in memory mapped I/O and I/O mapped I/O scheme?
- 8.2 What are the advantages and disadvantages of I/O mapped I/O over CPU initiated data transfer? Explain why I/O mapped I/O data transfer technique is limited to 256 input and 256 out peripherals.
- 8.3 A semiconductor memory is specified as $4K \times 8$. Mention the number of words, word size and total capacity of this memory?
- 8.4 Design a memory 16×8 from 16×4 memory IC.
- 8.5 Develop a 32×4 memory using 16×4 memory ICs.
- 8.6 Draw a memory read cycle and explain briefly.
- 8.7 Draw a memory write cycle and explain briefly.
- 8.8 Draw the timing diagram of I/O read cycle and explain briefly.
- 8.9 Draw the timing diagram of I/O write cycle and explain briefly.
- 8.10 What are the control signals are used for memory and I/O read and writes operations?
- 8.11 Explain the generation of \overline{MEMR} , \overline{MEMW} , \overline{IOR} and \overline{IOW} control signals from IO / \overline{M} , \overline{RD} , \overline{WR} signals.
- 8.12 Compare the memory mapped I/O with peripheral mapped I/O.
- 8.13 Design to interface $2K \times 8$ RAM and $4K \times 8$ RAM to a 8085 microprocessor.
- 8.14 Discuss address decoding with a suitable example.
- 8.15 Explain memory interfacing with 8085 microprocessor. Design a memory interfacing circuit to interface the following memory ICs.
 - (i) $2K \times 8$ -bit EPROM 2716. Assume starting address is 8000H.
 - (ii) $2K \times 4$ -bit RAM 6116. Consider starting address is 9000H.
 Write the memory map of the above ICs.
- 8.16 Compare static RAM and dynamic RAM.
- 8.17 Design the interfacing of $8K \times 8$ bit RAM with 8085 microprocessor. Assume the starting address is 7000H. Show the memory map.
- 8.18 Design the interfacing circuit to interface two $8K$ RAM and two $4K$ EPROM with the 8085 microprocessor. Assume the starting address is 8000H. Show the memory map.
- 8.19 Explain in detail the interrupt system of 8085 microprocessor.
- 8.20 What are the software interrupts of 8085 microprocessor? Mention interrupts instructions with their Hex code and vector address. How is the vector address for a software interrupt determined?
- 8.21 Draw the TRAP interrupt and explain briefly. Why TRAP input the edge and level sensitive? Can the TRAP interrupt be disabled by a combination of hardware and software? Write some applications of TRAP interrupt.
- 8.22 Draw the interrupt circuit diagram for 8085 and explain in detail.
- 8.23 What are nested interrupts? How you can handle the nested interrupt. Write a short note on pending interrupt.
- 8.24 Draw the SIM instruction format and discuss with examples.
- 8.25 Draw the RIM instruction format and discuss with examples. "A RIM instruction should be performed immediately after TRAP occurs". Why?

- 8.26 Design and explain a scheme to interrupt on INTR of 8085.
- 8.27 Explain the software instructions EI and DI. 'When returning back to the main program for interrupt service subroutine (ISS), the software instruction EI is inserted at the end of the ISS'. Why?
- 8.28 What do you mean by priority interrupts? Explain the operation of different interrupts available in 8085 with the help of circuit diagram.
- 8.29 Distinguish between
- (i) Vectored and nonvectored interrupt
 - (ii) Maskable and nonmaskable interrupt
 - (iii) Internal and external interrupt
 - (iv) Software and hardware interrupt
- 8.30 Draw the timing diagram of RST 5 instruction and explain briefly.
- 8.31 Give the comparison between INTR, RST 5.5, RST 6.5, RST 7.5 and TRAP interrupts.
- 8.32 Write a program to call the interrupt service routine 003CH corresponding to RST 7.5 if it is pending. Assume the content of accumulator is 20H on executing the RIM instruction.
- 8.33 Write the software instructions EI and DI. Explain their operations.
- 8.34 Explain interrupt driven I/O technique. How does 8085 microprocessor respond to INTR interrupt signal?
- 8.35 Explain the triggering levels of RST7.5, RST 6.5 and RST 5.5.
- 8.36 Discuss the INTR and INTA signals of 8085.
- 8.37 Mention all the pins of 8085 microprocessor through which the processor can be interrupted. Write instructions to enable interrupt RST 6.5 and disable RST 7.5 and 5.5.
- 8.38 Define interrupt. Explain the importance of interrupt in a microprocessor-based system.
- 8.39 Draw the architecture of 8259A and explain briefly. Write the features of 8259.
- 8.40 Draw the functional block diagram of 8259. What are the different functional blocks in 8259?
- 8.41 Write the functions of the following pins of 8259A.
- (i) INT
 - (ii) INTA
 - (iii) IR_0 – IR_7
 - (iv) CAS_0 – CAS_2
 - (v) \overline{SP} / \overline{EN}
- 8.42 Draw the block schematic showing the interconnections between I/O devices with 8259, RAM, ROM.
- 8.43 How many interrupt levels can be handled by 8259. What are the ICWs and OCWs? Describe how the PIC 8259 responds to interrupts.
- 8.44 Write down the sequence of operation for programming 8259?
- 8.45 Write the functions of IR_0 – IR_7 pins. Describe the functions of pins CAS_0 – CAS_2 .
- 8.46 Describe the vector data formats when 8085 is interrupted via INTR.
- 8.47 Describe the following initialisation command words:
- (i) ICW_1
 - (ii) ICW_2
 - (iii) ICW_3
 - (iv) ICW_4
- 8.48 Discuss the operation command word 2 OCWs. Describe the fully nested mode.
- 8.49 Describe the EOI and AEOI command. In cascade mode, how many EOI commands are to be issued?
- 8.50 Explain Special Fully Nested Mode (SFNM). Describe special mask mode, and polled mode interrupt. Distinguish between NSEOI, SEOI, and AEOI.
- 8.51 Explain the interfacing of 8259A with 8085/8088/8086 microprocessor.
- 8.52 Draw the functional block diagram of 8255 and explain the operation of each sub-block.
- 8.53 What are the different operating modes of 8255? Explain any one operating mode of 8255.

- 8.54 Explain different ports and control words of 8255.
- 8.55 Explain BSR mode of 8255. Discuss the control word format in the BSR mode.
- 8.56 Write a BSR control word to set bits PC_7 and PC_0 and to reset them after 1 second delay.
- 8.57 Write the control word format for I/O mode operation of 8255. Explain the operation of 8255 PPI IC with its internal block diagram. Explain its Mode 0, Mode 1 and Mode 2 operations.
- 8.58 Write control word in Mode 0 operation for the following cases:
(i) Port A = Input port, port B = output port, Port C = output port
(ii) Port A = Input port, port B = output port, Port C_U = output port, Port C_L = input port
- 8.59 What are the control signals when ports A and B act as input ports during Mode 1 operation? Discuss the control signals. Draw the timing waveforms for a stored input.
- 8.60 What are the control signals when ports A and B act as input ports during Mode 1 operation?. Discuss the control signals. Draw the timing waveforms for a stored output.
- 8.61 Draw Port A and the control signals when 8255 is operated in mode 2. Explain the Mode 2 operation with a timing diagram.
- 8.62 What are input and output control signals in Mode 2? Discuss them. Write applications of 8255. Write a program to generate a square wave using 8255.
- 8.63 Write the difference between three different modes of 8255 PPI. Explain Mode 1 operation and draw the timing diagram of port A in mode 1 operation.
- 8.64 Discuss the control signals when ports A and B act as output ports. Draw the timing diagram for strobed input.
- 8.65 Determine the control word for the following configuration of the ports of 8255
(i) Port A—output and mode of Port A is Mode 1
(ii) Port B—output and mode of Port B is Mode 1
Remaining pins of Port C are used as input
- 8.66 Explain the operation of 8253 timer IC with its functional block diagram.
- 8.67 Explain the features of 8253. Briefly explain its different modes of operation.
- 8.68 Draw the functional block diagram of 8253. How many counters are there in 8253 and how many modes are there?
- 8.69 Explain Mode 0 operation with timing diagram.
- 8.70 Explain the importance of GATE signal. How it is used to control the operation of counters?
- 8.71 Explain Mode 1 and Mode 2 operations with timing diagrams. Write the difference between Mode 2 and Mode 3 of 8253.
- 8.72 Explain Mode 3 and Mode 4 of 8253 with timing diagrams.
- 8.73 Write the interfacing procedure to interface 8253 with 8085 microprocessor. Write control word format and explain for all modes of operation.
- 8.74 Show in a tabular form the conditions of different modes corresponding to the different status of gate signals.
- 8.75 Discuss different methods of reading the value of the count in a counter while the counting is in progress.
- 8.76 Write a program to read the count value of counter while counting is going on. Assume Counter 0 in Mode 0 with count value 7000H.
- 8.77 Write a program to generate a square wave using 8253.
- 8.78 Write a program to use Counter 2 of 8253 in Mode 5 operation.

- 8.79 What is the difference between software interrupt and hardware interrupt?
- 8.80 What are the different interrupts of 8086 microprocessor? Explain software interrupts in detail.
- 8.81 What is interrupt vector table of 8086 microprocessor?
- 8.82 Draw the flow chart for interrupt operator and discuss interrupt cycle of 8088 microprocessors.

Answers to Multiple-Choice Questions

-
- | | | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| 8.1 (c) | 8.2 (a) | 8.3 (a) | 8.4 (b) | 8.5 (a) | 8.6 (a) | 8.7 (a) | 8.8 (d) | 8.9 (b) |
| 8.10 (d) | 8.11 (d) | 8.12 (d) | 8.13 (a) | 8.14 (c) | 8.15 (a) | 8.16 (c) | 8.17 (a) | 8.18 (c) |
| 8.19 (b) | 8.20 (a) | 8.21 (a) | 8.22 (a) | 8.23 (c) | 8.24 (a) | 8.25 (a) | 8.26 (a) | 8.27 (b) |

Chapter 9

Communication and Bus Interfacing with the 8085/8086 Microprocessor

9.1 INTRODUCTION

The microprocessor is a very powerful IC, used to perform various ALU functions with the help of data from the environment. For this, the microprocessor is connected with the memory and input/output devices and forms a microcomputer. The technique of connection between input/output devices and a microprocessor is known as *interfacing*. Special attention must always be given during the connection of pins of peripheral devices and microprocessor pins, as ICs cannot be simply connected. In the development of a microprocessor-based system, all memory ICs and input/output devices are selected as per requirement of the system and then interfaced with the microprocessor. Actually address, data and control lines are used for connecting peripherals. After connecting properly, programs are written in the microprocessor. Programs will be different for different applications. When the program is executed, the microprocessor communicates with input/output devices and performs system operations. In this chapter, the interfacing of Programmable Keyboard and Display Interface 8279, Serial Communication Interface 8251, Direct Memory Access Controller 8257, 8275 CRT Controller, A/D converter and D/A converter, Bus interface, RS 232C, IEEE-488, parallel printer interface, 8250 UART, 16550 UART and 8089 I/O processor are explained.

9.2 SERIAL COMMUNICATION INTERFACE 8251

The serial data transfer is a method of data transfer in which one bit is transferred at a time. This transmission is always used when the distance is greater than five metres. This method of transmission requires very few data lines compared to parallel transmission. Serial data transmission can be classified as *simplex*, *half-duplex* and *full-duplex* data transfer. In the simplex serial-data-transfer system, data is transfer only in one direction. In the half-duplex system, data can be transferred in either direction but in one direction at a time only. In the full-duplex system, data can be transmitted in both directions simultaneously.

The serial data-transfer systems can also be classified based on timing signals such as synchronous and asynchronous data transfer. The difference between synchronous and asynchronous data transfer is given in Table 9.1.

Table 9.1 Differences between synchronous and asynchronous data transfer

<i>Asynchronous Data Transfer</i>	<i>Synchronous Data Transfer</i>
In asynchronous data transfer, a word or character is preceded by a start bit and is followed by a stop bit. The start bit is a logical 0. The stop bit(s) is (are) a logical 1.	In synchronous data transfer, the transmission begins with a block header, which is a sequence of bits.
Data can be sent one character at a time.	This can be used for transferring large amounts of data without frequent starts or stops.
When no data is sent over the line, it is maintained at an idle value, logic '1'.	Since the data sent is synchronous, the end of data is indicated by the sync character(s). After that, the line can be either low or high.
A parity bit can be included along with each word or character. Each character data can be of 5, 6, 7 or 8 bits.	A parity bit can be included along with each word or character. Each character data can be of 5, 6, 7 or 8 bits.
The start and stop bits are sent with each character. Generally, the stop bits may be either one or more bits. The stop bits must be sent at the end of the character. It is used to ensure that the start bit of the next character will cause a start bit transition on the line.	In synchronous data transfer, the transmitter sends synchronous characters, which is a pattern of bits to indicate end of transmission.
Asynchronous mode data transfer is used for low-speed data transfer. Data can move in simplex, half-duplex and full-duplex methods.	Synchronous mode data transfer is used for high-speed data transfer. Data can move in simplex, half-duplex and full-duplex methods.
In this data transfer, the transmitter is not synchronized with the receiver by the same clock. The clock is an integral multiple of the baud rate (number of bits per second). Generally, this multiplication factor is 1, 16, or 64.	In synchronous mode data transfer, the receiver and transmitter is perfectly synchronized on the same clock pulse.
Synchronization between the receiver and transmitter is required only for the duration of a single character at a time.	Synchronism between the transmitter and receiver is maintained over a block of characters.
Asynchronous data transfer can be implemented by hardware and software.	Synchronous data transfer can be implemented by hardware.

The 8251 is a powerful programmable communication interface IC through which the serial data transfer can be effectively carried out. The Programmable Communication Interface 8251 is a programmable I/O device designed for serial communication. This IC can be used either in synchronous mode or asynchronous mode. Therefore, it is called Universal Synchronous Asynchronous Receiver and Transmitter (USART).

The IC chip is fabricated using N-channel silicon gate technology. The 8251 can be used to transmit and receive serial data. It accepts data in parallel format from the microprocessor and converts them into serial data for transmission. This IC also receives serial data and converts them into parallel and sends the data in parallel. It is available in a 28-pin dual in-line package and has the following features:

- ◆ Synchronous and asynchronous operation
- ◆ Programmable data word length, parity and stop bits
- ◆ Parity, overrun and framing error-checking instructions and counting-loop interactions
- ◆ Programmed for three different baud rates
- ◆ Supports up to 1.750 Mbps transmission rates

- ◆ Divide-by 1, 16, 64 mode
- ◆ False start bit deletion
- ◆ Number of stops increase of asynchronous data transfer can be 1 bit 1 ½ or 2 bits
- ◆ Full-duplex double-buffered transmitter and receiver
- ◆ Automatic break detection
- ◆ Internal and external sync character detection
- ◆ Peripheral modem control functions

This device is mainly used as the asynchronous serial interface between the processor and the external equipment. This IC can also be used to generate the baud-rate clock using external clock and convert outgoing parallel data into serial data. This IC can also be used to convert incoming serial data into parallel data and it can control the modem.

9.2.1 Functional Block Diagram

The functional block diagram of 8251 IC is shown in Fig. 9.1. This IC consists of four major sections, namely, transmitter, receiver, modem control and microprocessor interface section. These four sections are communicated with each other on an internal data bus for serial data transfer.

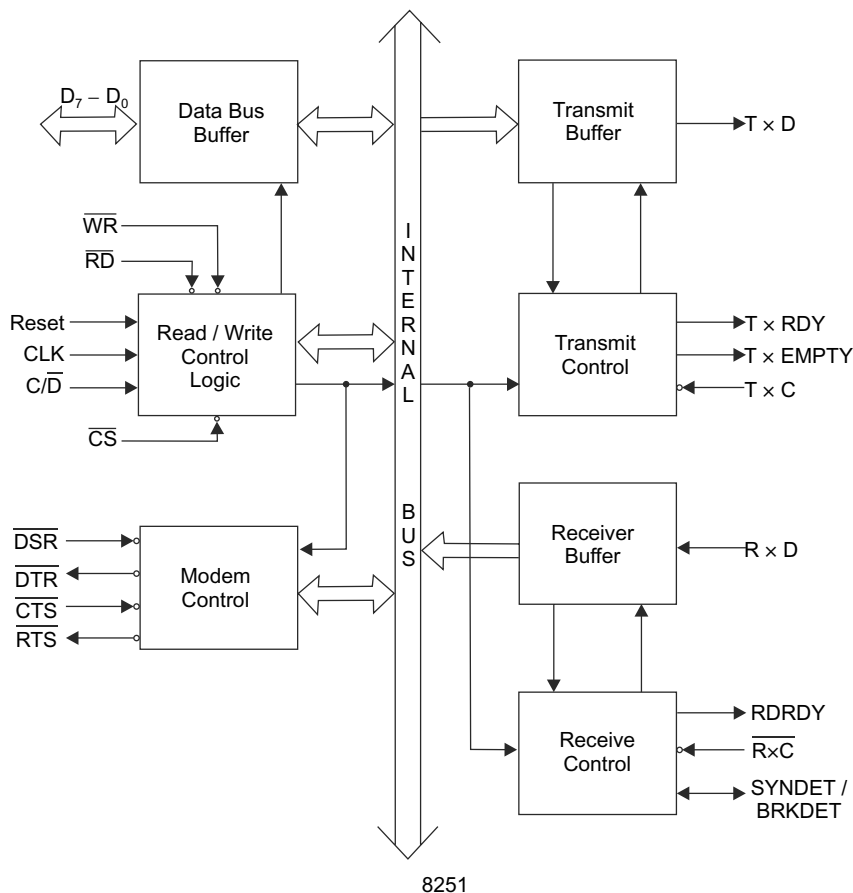


Fig. 9.1 Functional block diagram of 8251 IC

Transmitter

Figure 9.2 shows the transmitter section. The section consists of transmitter buffer register, output register and transmit control logic. This section has one input and three outputs. The transmitter buffer register accepts parallel data from the data bus of the microprocessor. Then data can be shifted out of the output register on the $T \times D$ pin after addition of framing bits. The serial data bits are preceded by the START bit and succeeded by the STOP bit, which are known as *framing bits*. For this operation the transmitter must be enabled and \overline{CTS} signal must be active low. The $T \times C$ signal is the transmitter clock signal which controls the bit rate on the $T \times D$ line. The clock frequency may be 1, 16 or 64 times the baud.

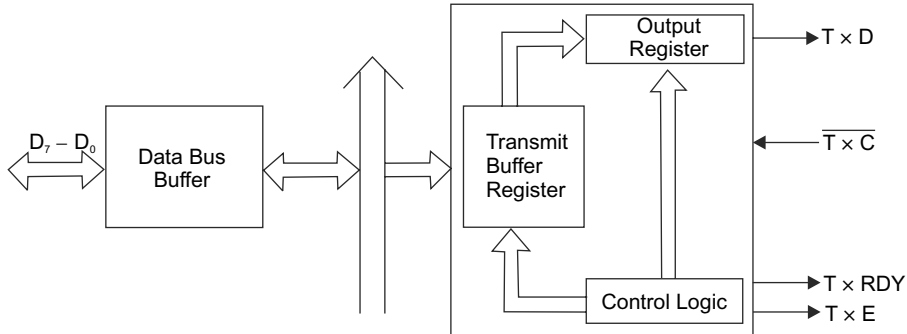


Fig. 9.2 Block diagram of transmitter

In the asynchronous mode, the transmitter adds a START bit, depending on how the unit is programmed; it also adds an optional even or odd parity bit, and either 1, 1 1/2 or 2 STOP bits.

When the transmitter buffer register is empty, it outputs a signal $T \times RDY$. It indicates to the CPU that the 8251 is ready to accept a data character.

When there is no data in the transmitter output register then it raises a $T \times E$ signal to indicate that the transmission is stopped. It is reset when the data is transferred from the buffer register.

Receiver

The block diagram of a receiver section is depicted in Fig. 9.3. This section consists of a receiver buffer register, receiver control logic and input register. The receiver section of 8251 USART accepts serial data on the $R \times D$ pin. Then it converts the data into parallel data according to the required format. When the 8251 is in the asynchronous mode and it is ready to accept a character, it looks for a low level on the $R \times D$ line.

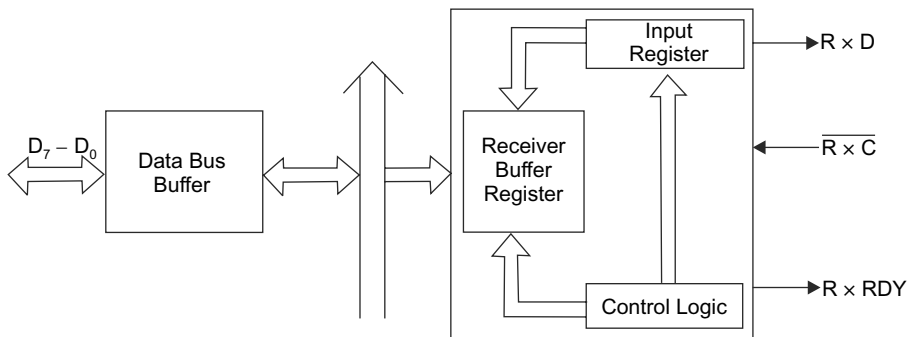


Fig. 9.3 Block diagram of receiver

When it gets a low level, it assumes that it is a START bit and enables an internal counter. At a count equivalent to one-half of a bit time, the $R \times D$ line is sampled again. If the line is still low, a valid START bit has probably been received and the 8251 proceeds to assemble the character. If the $R \times D$ line is high when it is sampled then either a noise pulse has occurred or the receiver has become enabled in the middle of the transmission of a character. In either case, the receiver aborts its operation and prepares itself to accept a new character. After the successful reception of a START bit, the 8251 clocks in the data, parity and STOP bits in the input register, the data is separated and converted into parallel data and then transfers the data to the receiver buffer register. The $R \times RDY$ signal is asserted to indicate that a character is available. The $R \times C$ falling edge clock signal is used to disassemble the bits from the serial data.

When this register is full, the $R \times RDY$ line becomes high. This line is then used either to interrupt the microprocessor or to indicate its own status. The microprocessor then accepts the data from the register.

The $R \times C$ line stands for receiver clock. This signal controls the rate at which bits are received by the input register. The clock can be set to 1, 16, or 64 times the baud in the asynchronous mode.

Modem Control

The modem connection of the IC 8251 is shown in Fig. 9.4. The modem control section provides the generation of \overline{RTS} (request to send) and the reception of \overline{CTS} (clear to send). This section also provides a general-purpose output \overline{DTR} (Data Terminal Ready) and a general-purpose input \overline{DSR} (Data Set Ready). \overline{DTR} is generally assigned to the modem, indicating that the terminal is ready to communicate and \overline{DSR} is a signal from the modem indicating that it is ready for communication.

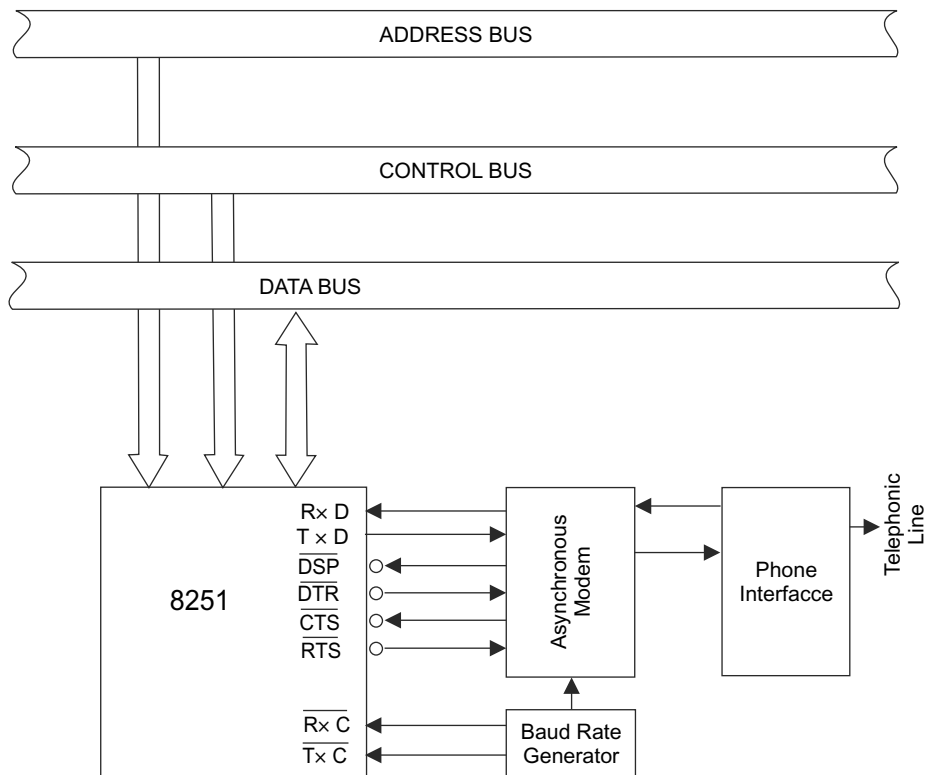


Fig. 9.4 Block diagram of modem control

9.2.2 Pin Diagram of 8251

Figure 9.5 shows a schematic diagram of Intel 8251 and the pin configuration of 8251 A is depicted in Fig. 9.6. The description of pins is as follows:

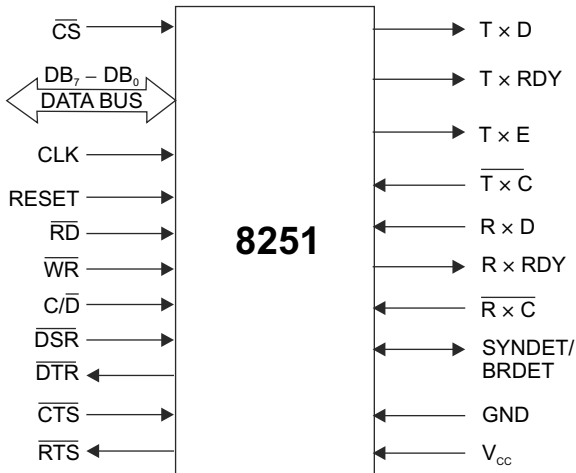


Fig. 9.5 Schematic diagram of Intel 8251

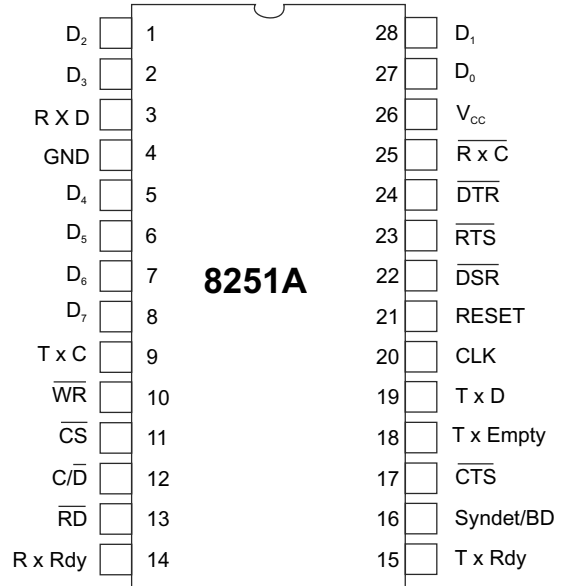


Fig. 9.6 Pin diagram of Intel 8251

D₀–D₇ (Data Bus)

The 8-bit data bus is used to read or write status, command word or data from or to the 8251 A.

Read/Write Control Logic Signals

The Read/Write Control Logic consists of three buffer registers such as data buffer register, control register and status register. It has six input signals \overline{CS} , C/\overline{D} , \overline{RD} , \overline{WR} , \overline{CS} (Chip select), RESET and CLK.

✓ **\overline{CS} (Chip select)** An active-low on this input select inputs 8251 A for communication. When \overline{CS} is high, no reading or writing operation can be performed. The data bus is tristated and \overline{RD} and \overline{WR} have no effect on the device.

✓ **C/\overline{D} , (Control Word/Data)** This pin is used to inform the 8251A that the word on the data bus is either data or control word/status information. When C/\overline{D} pin is high, either the control register or status register will be selected. If this pin is low, the data bus buffer is selected. The \overline{RD} and \overline{WR} signals are used to distinguish the control register and the status register respectively.

Pin Name	Function
D ₀ –D ₇	Data Bus
\overline{CS}	Chip select
C/\overline{D}	Control Word/Data
\overline{RD}	Read
\overline{WR}	Write
RESET	RESET

(Contd.)

(Contd.)

CLK	CLOCK
$T \times D$	Transmit data.
$\overline{T} \times \overline{C}$	Transmitter clock
$T \times RDY$	Transmitter ready
$T \times E$	Transmitter empty
$R \times D$	Receiving data
$\overline{R} \times \overline{C}$	Receiver clock
$R \times RDY$	Receiver ready
\overline{DSR}	Data set ready
\overline{DTR}	Data terminal ready
\overline{CTS}	Clear to send
\overline{RTS}	Request to send

- ✓ **\overline{RD} (Read)** An active-low on this input informs the microprocessor 8251A that the microprocessor is reading either data or status information from internal registers of 8251.
- ✓ **\overline{WR} (Write)** The active-low input on \overline{WR} is used to inform it that the microprocessor is writing data or control word to 8251.
- ✓ **RESET** A high on this input forces the 8251A into an 'idle' state. This device will remain idle until a new set of control words is written into it. The minimum required reset pulse width is 6 clock states for each reset operation.
- ✓ **CLK (CLOCK)** The CLK input is used to generate internal device timings and is normally connected to the output of a clock generator. The input frequency of CLK should be greater than 30 times the receiver or transmitter data-bit transfer rate.

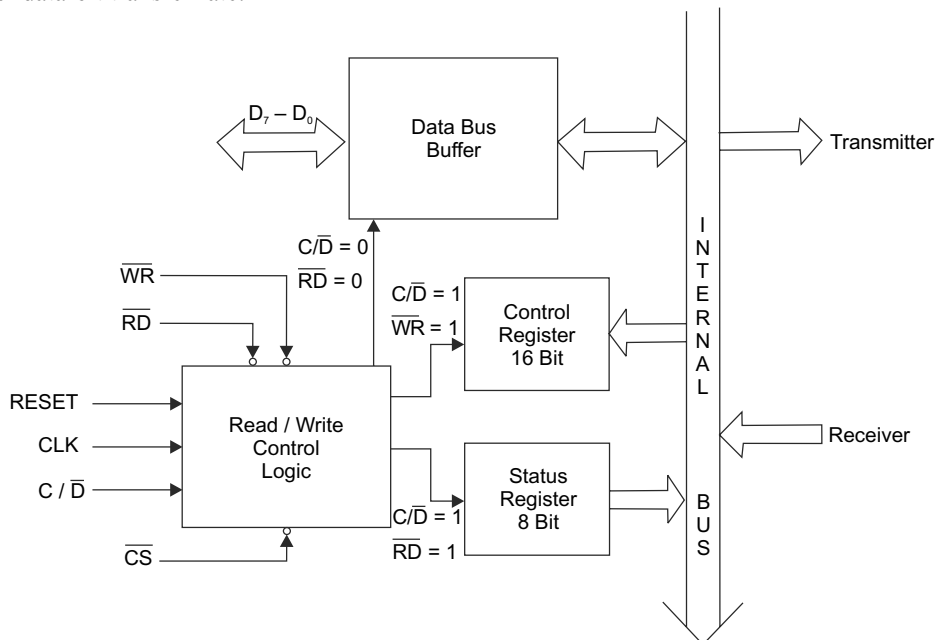


Fig. 9.7 Control logic and registers of 8251 IC

Table 9.2 Status of the control signals, \overline{CS} , C/\overline{D} , \overline{RD} , \overline{WR} and CLK for accessing the different registers.

\overline{CS}	C/\overline{D}	\overline{RD}	\overline{WR}	State
0	1	1	0	Microprocessor writes instructions in the control register
0	1	0	1	Microprocessor reads status from the status register
0	0	1	0	Microprocessor outputs data to the data buffer
0	0	0	1	Microprocessor accepts data from data buffer
1	x	x	X	USART is not selected for communication

Transmitter

Transmitter control pins are $T \times D$, $\overline{T \times C}$, $T \times RDY$, $T \times E$ which are explained below:

✓ **$T \times D$ (Transmit Data Output)** The serial data output from the output register is transmitted on $T \times D$ pin. The transmitted data bits consist of data along with other informations such as start bit, stop bits and parity bit.

✓ **$T \times C$ (Transmitter Clock Input)** The transmitter clock input $\overline{T \times C}$ controls the rate at which the data is to be transmitted. The baud rate is equal to the $T \times C$ frequency in synchronous transmission mode. In asynchronous transmission mode, the baud rate is 1, 1/16 or 1/64 times the $T \times C$. The serial data is transmitted out by the successive negative edge of the $T \times C$.

✓ **$T \times RDY$ (Transmitter Ready)** This is output signal, which indicates to the CPU that the transmitter buffer is empty. The internal circuit of the transmitter is ready to receive a byte of data from the CPU for transmission. The $T \times RDY$ signal is set only when \overline{CTS} and $T \times E$ are active. The $T \times RDY$ is reset when the CPU writes a byte into the buffer register by the rising edge of the \overline{WR} signal. The $T \times RDY$ status bit will indicate the empty or full status of the transmitter data input register.

✓ **$T \times E$ (Transmitter Empty)** When the $T \times E$ output is high, the 8251 has no characters to transmit. This automatically goes low when a character is received from the CPU for further transmission. If this pin is high in synchronous mode, it indicates that a character has not been loaded and the SYNC character or characters are being transmitted or about to be transmitted. The $T \times E$ signal can be used to indicate the end of a transmission mode.

Receiver

Receiver control pins are $R \times D$, $\overline{R \times C}$, $R \times RDY$ which are explained below:

✓ **$R \times D$ (Receive Data Input)** This input pin of 8251A receives serial data from outside environment, and delivers to the input register via $R \times D$ line which is subsequently put into parallel form and placed in the receiver buffer register.

✓ **$\overline{R \times C}$ (Receiver Clock Input)** The $\overline{R \times C}$ receiver clock input pin controls the rate at which the bits are received by the input register. In synchronous mode, the baud rate is equal to the $R \times C$ frequency. In asynchronous mode, the baud rate can be set to either 1 or 1/16 or 1/64th of the $R \times C$ frequency. The received data is read into the 8251 on rising edge of $R \times C$.

✓ **$R \times RDY$ (Receiver Ready)** This is an output pin which indicates that 8251A contains a character to be read by the CPU. This signal can be used to interrupt the CPU as well as polled by the CPU. In synchronous mode, the receiver must be enabled to set the $R \times RDY$ signal and a character must finish assembly and then be transferred to the data output register. When the data does not read properly from the receiver data output register before assembly of the next data byte, the overrun condition error flag is set and the previous byte is overwritten by the next byte of the incoming data and hence it is lost.

Modem Control Pins

8251 has four modem control pins: \overline{DSR} , \overline{DTR} , \overline{CTS} and \overline{RTS} . The \overline{DSR} and \overline{RTS} are inputs but \overline{DTR} and \overline{CTS} are output pins. All these pins are active low. The description of modem control pins are given below:

✓ **\overline{DSR} (Data Set Ready)** The \overline{DSR} input can be used as a general-purpose one-bit inverting input port. The CPU using a status read operation can test its status. This input is normally used to check the modem condition such as data set is ready.

✓ **\overline{DTR} (Data Terminal Ready)** This is a general-purpose one-bit inverting output port. This can be used by 8251 to signal the modem about the information that the device is ready to accept data. This port can be programmed using the command word.

✓ **\overline{CTS} (Clear to Send)** This is a one-bit inverting input port. When the \overline{CTS} input line is low, the 8251A will be enabled to send the serial data, provided D_0 , the enable bit in the command instruction word should be enabled. If D_0 becomes low in the command instruction word while data transmission takes place, \overline{CTS} is switched off, and the transmitter will complete sending the stored data.

✓ **\overline{RTS} (Request to Send Data)** This is a general-purpose one-bit inverting output port. This can be used by 8251 to indicate the modem that the receiver is ready to receive a data byte from the modem. Bit D_5 of the command instruction format controls the status of the pin.

✓ **$\overline{SYNDET/BD}$ (Synchronous Detect/Break Detect)** This pin is used for detection of synchronous characters in synchronous mode and break characters in asynchronous mode. This pin can be programmed using the appropriate control word. In the input mode or the external synchronous detect mode, a rising edge on this pin will cause 8251 to start collecting data characters on the rising edge of the next $R \times C$.

This pin can be used as a break detect in the asynchronous mode. When $R \times D$ pin remains low through two consecutive stop bit sequences, the stop bit sequence contains a stop bit, a start bit, data bits and parity bits. This is reset when master chip reset or the $R \times D$ becomes high.

9.2.3 8251 Interface With Microprocessor

The interfacing connection of 8251 with the microprocessor is shown in Fig. 9.8. This circuit consists of eight data lines, which are connected to the data bus of the CPU. The 8251 IC can be used either in I/O mapped I/O or memory mapped I/O mode. In I/O mapped I/O interface mode, the \overline{RD} and \overline{WR} lines are connected to \overline{IOR} and \overline{IOW} control lines. The CLK pin is connected with to the CLK OUT of the microprocessor to provide synchronization between the microprocessor and 8251. The RESET terminal is connected to the RESET OUT of the microprocessor. When RESET pin is in high level, the I.C 8251 is forced into the idle mode. The address decoder output is connected to the \overline{CS} terminal of 8251. C/\overline{D} terminal is used to select internal registers such as control register and data register. Generally this is connected to the A_0 address bus.

9.2.4 Programming and Operating Modes of 8251

The 8251 can be operated in different modes based on mode control words. A set of control words can be written into the internal registers of 8251A to make it operate in the

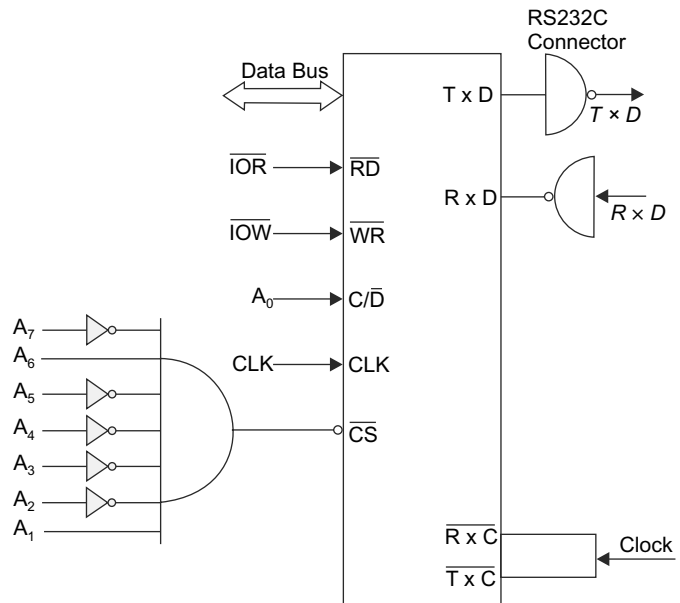


Fig. 9.8 Interfacing of 8251 with microprocessor

desired mode. The control words of 8251A are two functional types, namely,

- ◆ Mode Instruction Control word
- ◆ Command Instruction Control word

There are two 8-bit control registers in 8251 to load the mode word and command word. The control logic and registers of the 8251 IC are depicted in Fig. 9.7. The mode instruction word informs about the initial parameters such as mode, baud rate, stop bits and parity bit. The command instruction word explains about enabling the transmitter and receiver section. The mode instruction word and command instruction control word are explained below.

Mode Instruction Control Word

Mode instruction control word defines the general operational characteristics of 8251A. After reset by using internal reset command or external reset command, the mode instruction control word must be loaded into 8251 to configure the device as per requirements. These control words are different for synchronous and asynchronous mode operation. Once the mode instruction control word has been written into 8251, SYNC characters (synchronous mode only) or command instructions (synchronous or asynchronous mode) may be programmed. The mode of operation from synchronous to asynchronous or from asynchronous to synchronous can be changed by resetting the 8251. The typical data is given in Fig. 9.9. The mode instruction format for asynchronous mode is shown in Fig. 9.10.

$C/\bar{D} = 1$	Mode instruction	← Sync mode only
$C/\bar{D} = 1$	Sync character 1	
$C/\bar{D} = 1$	Sync character 2	
$C/\bar{D} = 1$	Command instruction	
$C/\bar{D} = 0$	Data	
$C/\bar{D} = 1$	Command instruction	
$C/\bar{D} = 0$	Data	
$C/\bar{D} = 1$	Command instruction	

Fig. 9.9 Typical data block

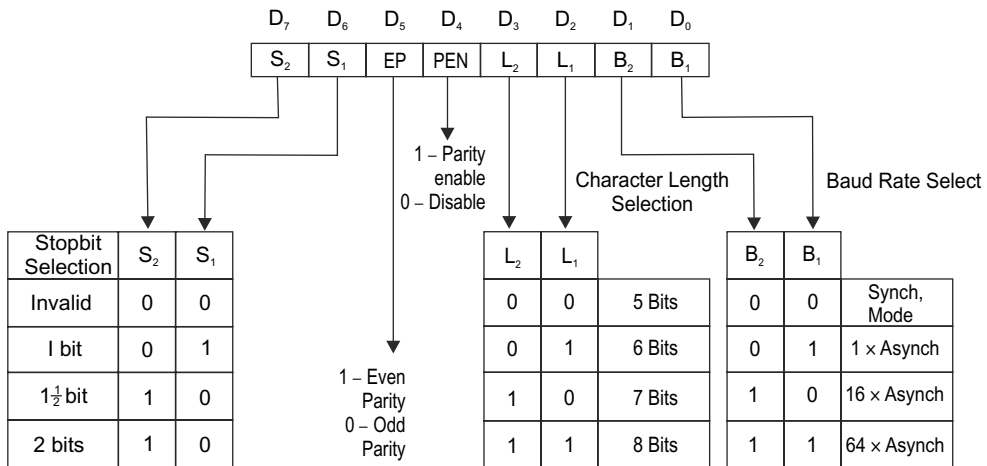


Fig. 9.10 Instruction format of asynchronous mode

Example 9.1

Find the mode instruction for the following operations:

- 8251 can be operated in asynchronous mode for data transmit
- The baud rate is 16 × Asynch
- The length of character is 8 bits and number of stop bits is 2

Assume odd parity, the address of the control register is 41H and the address of data register is 40H.

Solution

The mode instruction word for the above operations is DEH as given below:

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
1	1	0	1	1	1	1	0

To load the instruction word into the control word register, the following statements will be written:

```
MVI A, DEH
```

```
OUT 41H
```

Asynchronous Mode (Transmission)

The general transmission format for asynchronous communication is shown in Fig. 9.11. The transmission format consists of start bit, data character, parity

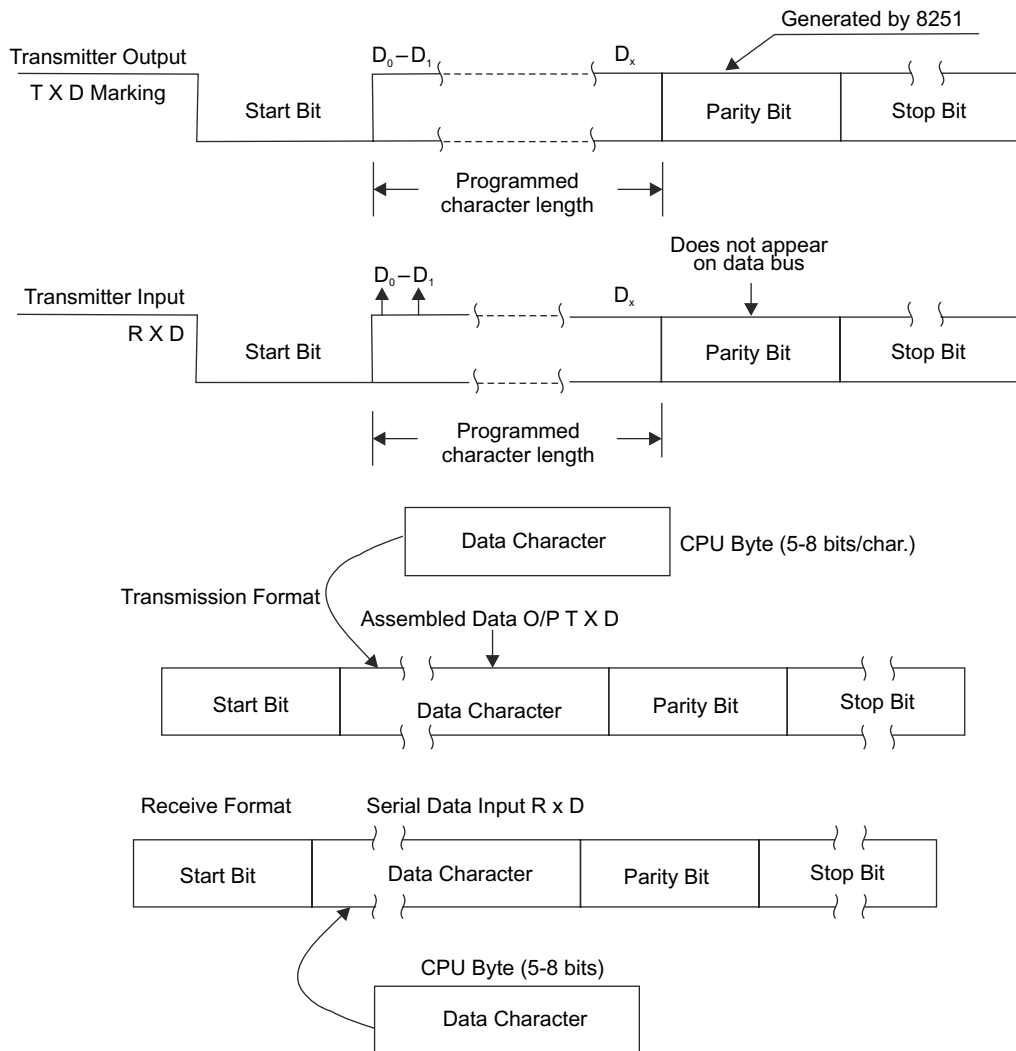
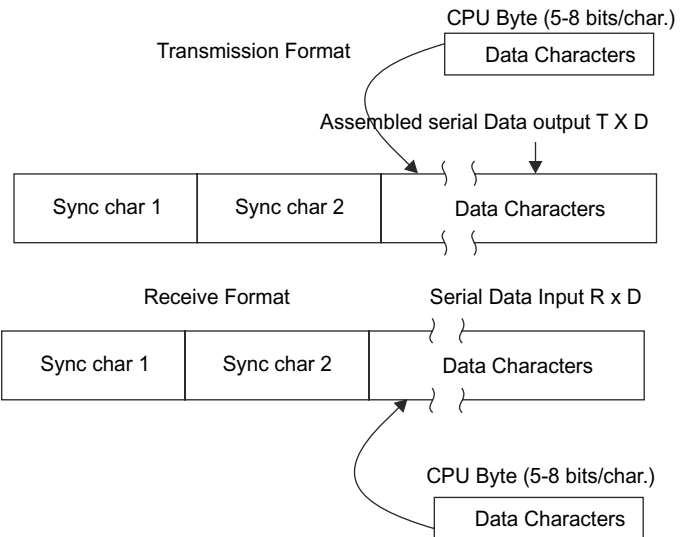


Fig. 9.11 Asynchronous mode transmission and receive formats

bit and stop bits. The 8251 starts to send data on the $T \times D$ pin after adding a start bit which is a 1 to 0 transmission. Then data bits are transmitted using the $T \times D$ pin on the falling edge of the transmitter clock ($\overline{T \times C}$) followed by stop bits. When no data is transmitted by the CPU to 8251, the $T \times D$ output pin remains 'high'. If a 'break' has been detected, $T \times D$ the line will go low.

Asynchronous Mode (Receive)

The general receive format for asynchronous communication is shown in Fig. 9.11. Data reception starts with a falling edge of $R \times D$ input which indicates the arrival of start bit. The high to low transition on the $R \times D$ line triggers the 'False start Bit Detection Circuit' and the output of this circuit samples the $R \times D$ line half-a-bit time later to confirm about the start-bit. If $R \times D$ is low, it indicates a valid start bit which starts counting. Then the bit counter locates data bits, parity bits and stop bits. If any error occurs during the receiving of data with regard to parity, framing or overrun, the corresponding flags in the status word will be set. The receiver requires only one stop bit to mark end of the data bit string though the number of stop bits affects the transmitter.



Synchronous Mode Instruction

Format The synchronous mode instruction format is shown in Fig. 9.12. For synchronous transmission/receiving of data both D_0, D_1 will be low. Bits D_2 and D_3 indicate the character length. Bit D_4 stands for Parity Enable (PEN) and D_5 stands for Even Parity (EP). Bit D_6 stands for External Synchronous Detect (ESD). $D_6 = 1$ for input and $D_6 = 0$ for output. Bit $D_7 = 1$ indicates a Single Synchronous Character (SCS). But $D_7 = 0$ represents double synchronous characters.

Fig. 9.12 Synchronous mode transmission and receive formats

Synchronous Mode (Transmission)

The general transmission/receive format for synchronous communication is shown in Fig. 9.13. In transmission format, one or two synchronous characters are sent followed by data characters. When the \overline{CTS} line becomes low, the first character is serially transmitted out. All the characters are shifted out of the serial output register on the falling edge of the transmitter clock ($\overline{T \times C}$) at the same rate as $\overline{T \times C}$. After completion of transmission, the CPU replenishes the transmitter buffer. When the CPU fails to provide a character before the transmitter buffer becomes empty, 8251 should send SYNC characters. In that case, $T \times E$ pin becomes high to indicate that the transmitter buffer is empty.

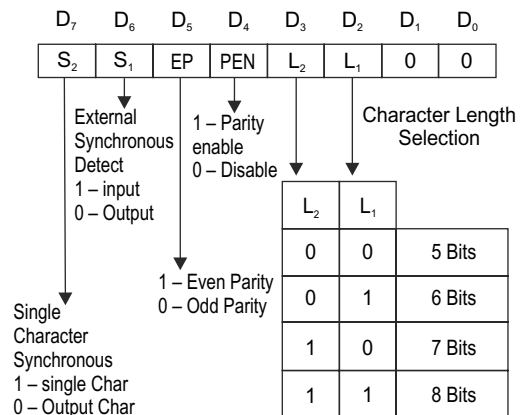


Fig. 9.13 Synchronous mode instruction format

Synchronous Mode (Receiver)

In synchronous receive mode, the character synchronization can be achieved internally or externally. In the internal SYNC mode, the receiver samples the data available as the $R \times D$ pin on the rising edge of $\overline{R \times C}$. When 8251 is programmed in this mode, 'ENTER HUNT' command should be included in the first command instruction word. The data on $R \times D$ pin is sampled on the rising edge of the $R \times C$. The receiver buffer content is compared with the first SYNC character at every edge till a match occurs. When 8251A is initially programmed for two sync characters, the process can be extended to two SYNC characters. When both the characters match, the hunting stops. After Hunting is over, the system goes for character boundary synchronization. The SYNDET pin is set and is rest automatically by a status read operation. The SYNDET pin gets set in the middle of the parity bit, if the parity is enabled; otherwise in the middle of the last data bit. In the external SYNC mode, synchronization can be achieved by applying a high level on the SYNDET input pin, which forces 8251A out of HUNT mode. The parity and overrun error can be checked in the same way as in asynchronous mode.

Command Instruction Word

The command instruction controls the actual operations of the selected format like enable transmit/receive, error reset and modem controls. Once the mode instruction has been written into 8251A and the SYNC characters are loaded (only in synchronous mode), the device is ready for data communication. The command instructions can be accepted only after mode instruction in case of asynchronous mode. All further control words written with C / \overline{D} will load a command instruction. A reset operation returns the 8251 back to mode instruction format from the command instruction format. The format of command instruction is depicted in Fig. 9.14.

When the 8251 has been programmed by the mode instruction word, the device is ready for data communication. The command instruction controls the actual operation of the selected format. The command Instruction format is explained below:

The D_6 bit is used as the internal reset. The command word with $D_6 = 1$ returns 8251 in the mode instruction format. When D_0 ($T \times EN$) is high, the transmitter becomes enable and data transmission is possible. If D_2 ($R \times EN$) is high, it enables the receiver for reception. The D_1 bit controls the data terminal ready. The D_3 bit forces the transmitter to send continuous break characters. A high on D_4 resets the error flags—PE, OE and FE (parity, overrun and framing errors respectively). The D_5 bit controls the request to send the pin of the device. The D_7 bit is used in synchronous mode. This pin enables the receiver to look for the synchronizing data.

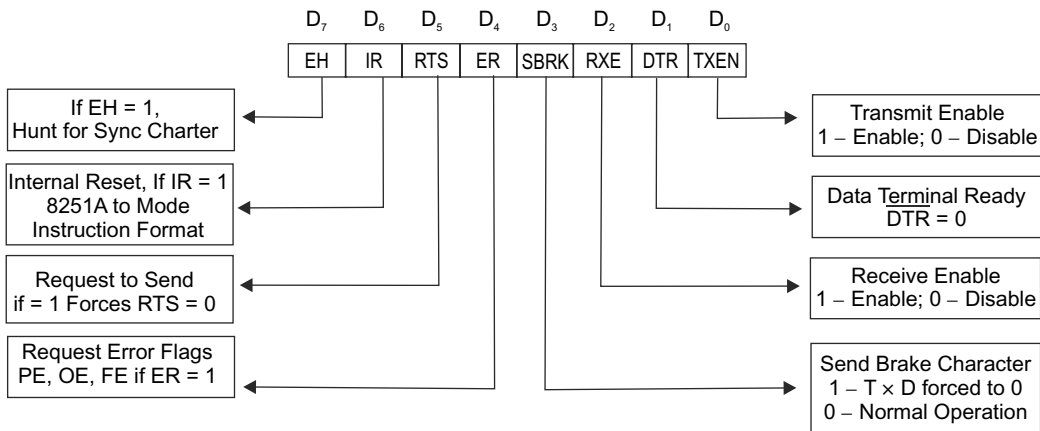


Fig. 9.14 Command instruction word

Status Word Register Format

The status word can be read with $C/\overline{D}=1$. The CPU requires various information to operate properly. All required information are provided by the status word. The status word is continuously updated by 8251, except when CPU reads the status word. The status word format is shown in Fig. 9.15.

- ◆ D_0 stands for the status of the pin $T \times RDY$.
- ◆ D_1 represents the status of the pin $R \times RDY$.
- ◆ D_2 correspond to the status of the pin $T \times E$.
- ◆ D_3 represents parity error. It is set when there is a parity error.
- ◆ D_4 stands for overrun error. It is set when the CPU does not read a character before the next one becomes available.
- ◆ D_5 represents framing error. It is set when a valid stop bit is not detected.
- ◆ D_6 is used for synchronous mode (SYNDET)
- ◆ D_7 reflects the logic level of the DSR (modem control) pin.

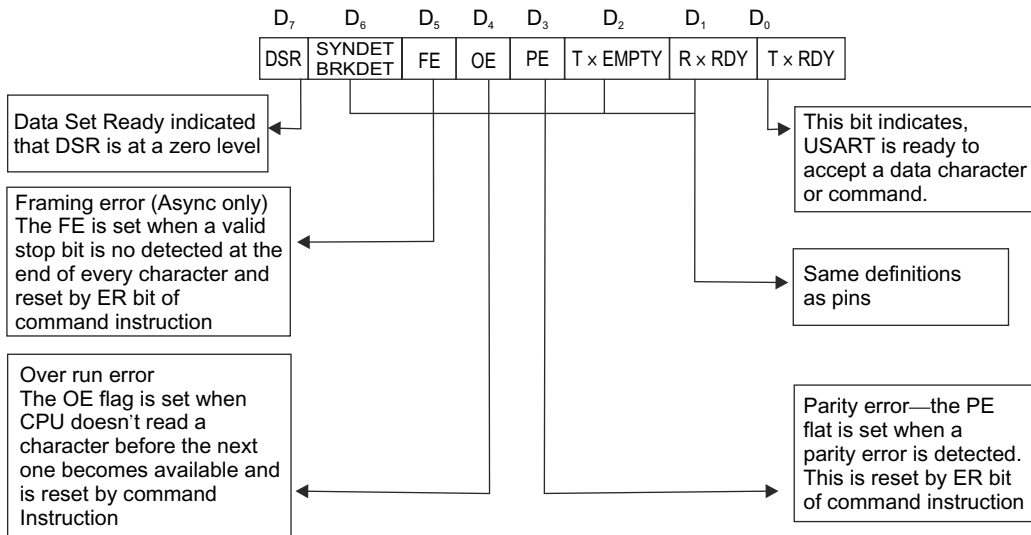


Fig. 9.15 Status word format

9.3 DIRECT MEMORY ACCESS (DMA) CONTROLLER 8257

When a bulk of data is transferred between memory and any peripheral devices, if I/O data transfer technique is used, this process takes more time as each byte of the data is transferred through the CPU. If we want to transfer data at a faster rate, the CPU must be isolated and data can be transferred between memory and peripheral devices directly. This I/O technique is known as Direct Memory Access (DMA) operation.

Figure 9.16 shows the DMA operation, which consists of CPU, memory devices, I/O peripheral devices, DMA controller and switches. Usually, the switch positions will be such that the memory and peripheral devices are connected to the CPU. Therefore, the data bus, address bus and the control bus of the memory and I/O peripheral devices are connected to the CPU. While the DMA operation is to be performed, the CPU is completely isolated and the address bus and control bus are taken over by the DMA controller circuitry. The DMA operation can be carried out by the following sequence as given below:

- ◆ Initially the device, which requires data transfer between the device and the memory, should send the DMA request (DRQ) to the DMA controller.
- ◆ The DMA controller sends a Hold Request (HRQ) line to the CPU and waits for the CPU to assert the HLDA.
- ◆ Then the microprocessor tri-states all the address bus, data bus and control bus. The CPU relinquishes the control of the bus and acknowledges the Hold input signal through Hold Acknowledge (HLDA) output signal. The CPU remains in the HOLD state; the DMA controller becomes the master of bus. Actually, DMA controller circuit manage the switching of address, data and control buses between CPU, Memory, and I/O devices.
- ◆ The HLDA signal is fed to the DMA controller. When the DMA controller receives the HLDA signal, the DMA controller takes care of direct data transfer operation between memory and I/O devices. The DMA controller sends DACK signal to the peripheral device, which requested for DMA operation.
- ◆ Then DMA operation can be performed by sending proper address to the memory and required control signals to transfer a bank of data.

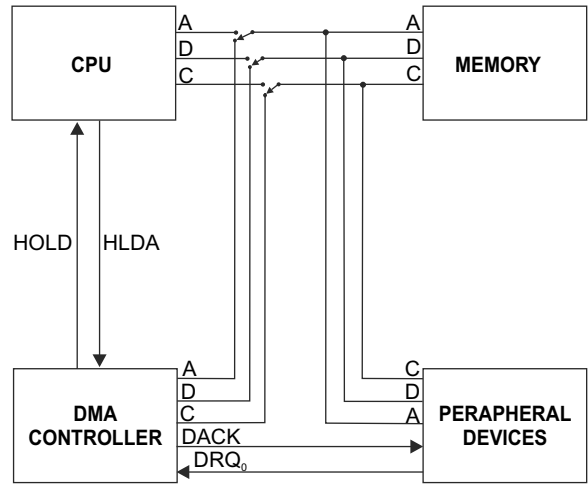


Fig. 9.16 DMA operation

At the starting of the DMA operation, the DMA controller should know the starting address of the memory location, number of bytes to be transferred and type of data transfer from memory to I/O or from I/O to memory.

9.3.1 Pin Diagram

The 8257 IC is a programmable DMA controller. This is available in a 40-pin dual in-line package. The schematic diagram of 8257 is shown in Fig. 9.17(a). The pin diagram of 8257 is depicted in Fig. 9.17(b) and the pin functions are described below:

DRQ₀–DRQ₃ These are the four separate DMA request lines. Any I/O device sends DMA request signal on one of the DRQ₀ – DRQ₃ lines. When DRQ is high, a DMA request signal is received by the DMA controller. Among four DMA request lines, DRQ₀ has the highest priority and DRQ₃ has the lowest priority in the fixed priority mode.

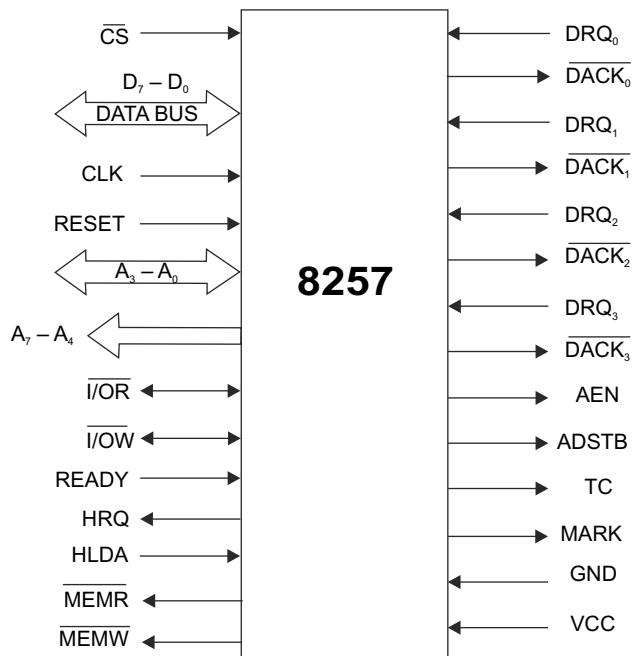


Fig. 9.17(a) Schematic diagram of 8257

$DACK_0-DACK_3$ These are the DMA acknowledge output lines which sends an acknowledged signal through any one of these lines to the I/O peripheral devices. When this signal is active-low, the line acknowledges the I/O devices.

A_0-A_3 These are the least significant address lines. A_0-A_3 are bi-directional lines. In master mode, these four least significant memory address lines are generated by 8257.

A_4-A_7 These are the four most significant address lines of lower byte address generated by 8257 in the master mode DMA operation.

D_0-D_7 These are bi-directional data lines. These lines are used to interface CPU with the internal data bus of 8257 DMA. During programming of the DMA controller, the CPU sends data through data lines for DMA address register; byte count register and mode set register. When the 8257 operates in master mode, these lines are used to send higher byte of the memory address. Then these 8 MSBs of address are latched using an ADSTB signal. During the first clock cycle of DMA operation, the address is transferred over $D_0 - D_7$. After that, the data bus is available for data transfer during the rest DMA cycle.

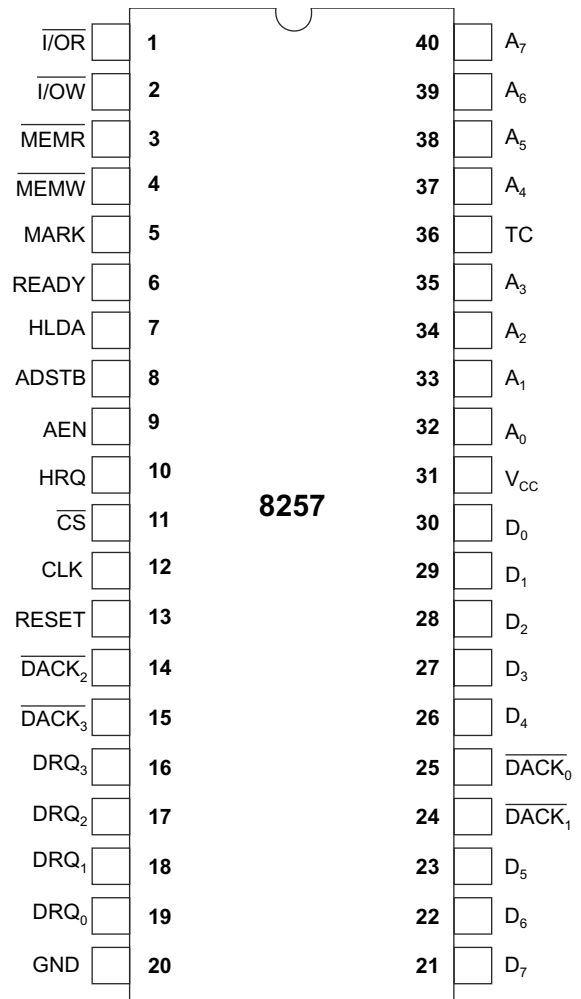
I/OR This is a bi-directional tristate input line. In the slave mode, IOR signal is used to read the internal registers of 8257 by the CPU. This line operates as output in master mode. In master mode, IOR is used to read data from I/O peripheral device during DMA write cycle.

I/OW This is a bi-directional tristate input line. In the slave mode, the I/OW signal is used to load the content of data lines to the upper or lower byte of a 16-bit DMA address register or terminal count register. In the master mode, data is transferred from memory to I/O devices during DMA memory read cycle.

$MEMR$ This is the memory read output signal. When this is active low, data will be read from memory during DMA read cycle.

$MEMW$ This is a memory write output signal. When this is active low, data will be written to the memory during DMA write cycle.

CLK A clock input is applied to 8257 for internal operation of 8257, which will be synchronized with the clock.



9.17(b) Pin diagram of 8257

RESET This is an asynchronous input signal. When this is active high, all DMA channels must be disabled, all mode registers will be cleared and all the control lines will be in tristates.

\overline{CS} This is an active low chip select signal, which enables the 8257 for read and write operations in slave mode. In the master mode, this is disabled to prevent the chip from getting selected (by CPU) during the DMA operation.

AEN (Address Enable) This is the address enable signal. When it is high, it indicates that the DMA operation will be performed. This AEN output can be used to disable the data bus and the control bus driven by the processor. This output can be used to disable the selection of an I/O device in the system.

ADSTB (Address Strobe) This output from 8257 strobes the most significant byte of the memory address generated by the DMA controller into the latches.

TC (Terminal Count) This output indicates that the terminal count register content is zero. If the TC STOP bit in the mode set register is set, the selected channel will be automatically disabled at the end of the DMA cycle. This pin will be activated when the 14-bit content of the terminal count register of the selected channel is equal to zero. The lower-order 14 bits of the terminal count register are to be programmed for the desired number of DMA cycles.

MARK When the MARK output is one level, it indicates that the current DMA cycle is the 128th cycle since the previous MARK output. The mark may be activated after each 128 cycles for the particular peripheral devices.

READY This is an asynchronous input signal. This is used to extend memory read and write cycles of 8257 by inserting wait states. This is suitable for interfacing slower I/O peripheral devices.

HRQ (Hold Request) The DMA controller sends the hold request as it is connected to the Hold signal input of the microprocessor. This output requests the microprocessor to access the system bus. In the master, this is connected with the HOLD pin of the CPU. In the slave mode, this pin of a slave is connected with a DRQ input line of the master 8257 and the master is connected with Hold input of the CPU.

HLDA (Hold Acknowledge) This pin is connected to the HLDA output of the CPU. This input is high indicates that the microprocessor tristates all the address bus, data bus and control bus.

V +5 V supply

GND Ground

9.3.2 Architecture of 8257

The 8257 is a programmable four-independent channel DMA controller. Therefore, four peripherals can send request data transfer simultaneously. The block diagram of the internal architecture of 8257 is depicted in Fig. 9.18. This consists of four DMA channels: control logic for data transfer, read/write logic and data bus buffer.

Register Organization of 8257 The 8257 performs the DMA operation using four independent DMA channels. Each DMA channel of 8257 has two 16-bit registers, namely, DMA address register and terminal count register. There are also two common registers for all the channels such as mode set register and status register. Therefore, there are ten registers of 8257. The CPU can select any one of the ten registers using address lines $A_0 - A_3$. Table 9.3 shows the selection of one of these registers based on $A_0 - A_3$. All registers are explained below:

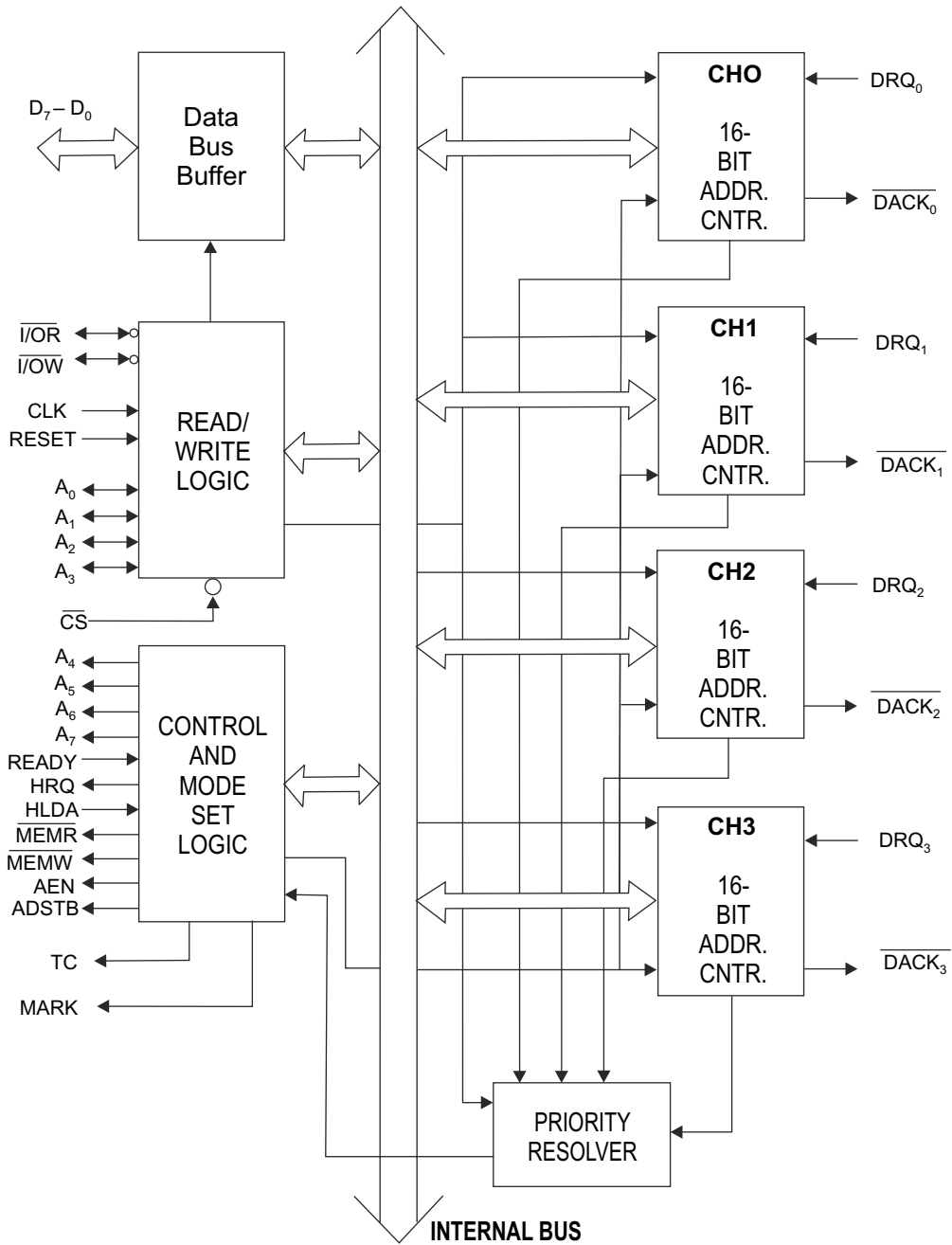


Fig. 9.18 Architecture of 8257

DMA Address Registers The 8257 has four separate DMA channels CH-0 to CH-3. Each channel has a DMA request and DMA acknowledge signal. Each DMA channel has one separate DMA address

register. The DMA address register is used to store the starting address of the memory location from where data will be accessed by the DMA channel. Therefore, the starting address of the memory block will be loaded in the DMA address register of the DMA channel. Generally, the 8257 DMA controller will access the block of memory with the starting address stored in the DMA Address Register and transfer to I/O devices through DMA channel.

Table 9.3 8257 register selection

Register	Byte	Address inputs				Bi-directional data bus							
		A ₃	A ₂	A ₁	A ₀	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
CH-0 DMA ADDRESS	LSB	0	0	0	0	A ₇	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀
	MSB	0	0	0	0	A ₁₅	A ₁₄	A ₁₃	A ₁₂	A ₁₁	A ₁₀	A ₉	A ₈
CH-0 Terminal Count	LSB	0	0	0	1	C ₇	C ₆	C ₅	C ₄	C ₃	C ₂	C ₁	C ₀
	MSB	0	0	0	1	Rd	Wr	C ₁₃	C ₁₂	C ₁₁	C ₁₀	C ₉	C ₈
CH-1 DMA ADDRESS	LSB	0	0	1	0	A ₇	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀
	MSB	0	0	1	0	A ₁₅	A ₁₄	A ₁₃	A ₁₂	A ₁₁	A ₁₀	A ₉	A ₈
CH-1 Terminal Count	LSB	0	0	1	1	C ₇	C ₆	C ₅	C ₄	C ₃	C ₂	C ₁	C ₀
	MSB	0	0	1	1	Rd	Wr	C ₁₃	C ₁₂	C ₁₁	C ₁₀	C ₉	C ₈
CH-2 DMA ADDRESS	LSB	0	1	0	0	A ₇	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀
	MSB	0	1	0	0	A ₁₅	A ₁₄	A ₁₃	A ₁₂	A ₁₁	A ₁₀	A ₉	A ₈
CH-2 Terminal Count	LSB	0	1	0	1	C ₇	C ₆	C ₅	C ₄	C ₃	C ₂	C ₁	C ₀
	MSB	0	1	0	1	Rd	Wr	C ₁₃	C ₁₂	C ₁₁	C ₁₀	C ₉	C ₈
CH-3 DMA ADDRESS	LSB	0	1	1	0	A ₇	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀
	MSB	0	1	1	0	A ₁₅	A ₁₄	A ₁₃	A ₁₂	A ₁₁	A ₁₀	A ₉	A ₈
CH-3 Terminal Count	LSB	0	1	1	1	C ₇	C ₆	C ₅	C ₄	C ₃	C ₂	C ₁	C ₀
	MSB	0	1	1	1	Rd	Wr	C ₁₃	C ₁₂	C ₁₁	C ₁₀	C ₉	C ₈
Mode Set (Program only)	-	1	0	0	0	AL	TCS	CH	RP	EN ₃	EN ₂	EN ₁	EN ₀
Status (Read only)	-	1	0	0	0	0	0	0	UP	TC ₃	TC ₂	TC ₁	TC ₀

A₁₀ – A₁₅ — DMA Starting Address, C₀ – C₁₃ — Terminal Count Value,

Rd and Wr—DMA verify (00), Write (01) or Read (10) cycle selection.

AL – Auto Load, TCS—TC STOP, EW—Extended Write, RP—Rotating Priority,

EN₃ – EN₀—Channel Mask Enable, UP—Update Flag, TC₃ – TC₀ — Terminal Count Status Bits.

Terminal Count Register

Each DMA channel of 8257 has one terminal count register (TC). The count register is a 16-bit register and is used to store the number of bytes which will be transferred through a DMA channel. Therefore, before starting the actual DMA operation, the register must be loaded the number of bytes. The first 14bits (D₀ – D₁₃) of the terminal count register are used for this purpose. The maximum data transfer using 8257 during one DMA operation can be 16K bytes. One of data transfer is known as a DMA cycle. Hence to transfer a block of data, large numbers of DMA cycles are required.

The most significant bits D₁₄ and D₁₅ of the count register indicate the type of the DMA function such as DMA write cycle, DMA read cycle or DMA verify cycle. The DMA operation selection and the corresponding bit configuration of the bits D₁₄ and D₁₅ of the terminal count register is shown in Table 9.4. In the DMA write operation, this device can able to transfer data from peripheral devices to the memory. During the DMA

read operation, the data is transferred from memory to peripheral devices. In DMA verify operation, the 8257 does not involve with the data transfer.

Table 9.4 Selection of DMA Operation Using A_{15}/RD and A_{14}/WR

Bit 15	Bit 14	DMA Operation
0	0	Verify DMA Cycle
0	1	Write DMA Cycle (I/O read and Memory write)
1	0	Read DMA Cycle (Memory read and I/O write)
1	1	(Illegal)

Mode Set Register

The mode set register of 8257 can be programmed as per requirement of the programmer. The control word in the mode set register is used to enable or disable DMA channels individually and also determines various modes of operation. To enable a DMA channel, the DMA address register and the terminal count register must be loaded with proper information. The mode set register format is shown in Fig. 9.19.

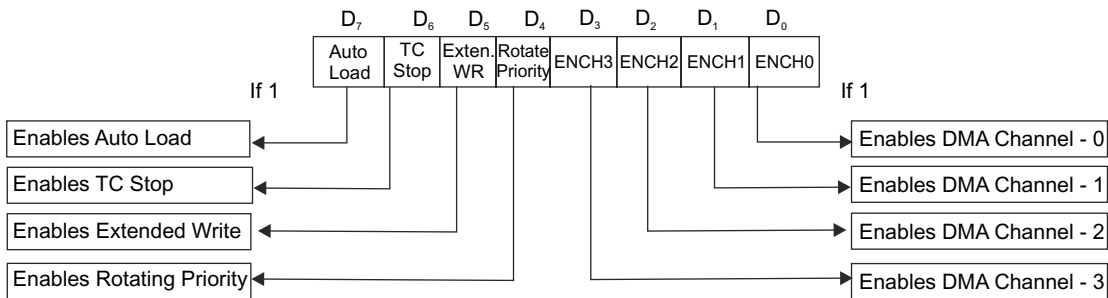


Fig. 9.19 Bit definitions of the mode Set Register

The bits $D_0 - D_3$ are used to enable or disable any one of the four DMA channels. When D_0 is '1', channel 0 is enabled.

If Bit D_4 of mode set register is set, rotating priority is enabled, or else the normal priority, i.e., fixed priority. In the rotating priority mode, the priority of the channels has a circular sequence. The rotating priority is depicted in Fig. 9.20. After each DMA cycle, the priority of channels are changed. The channel which has just been serviced will have the lowest priority. When the rotating priority bit is reset, each DMA channel has a fixed priority. In the fixed priority mode, Channel 0 has the highest priority and Channel 3 has the lowest priority. The priority of DMA operation is shown in Table 9.5.

Table 9.5 Priority operations of DMA channels

Priority	Channel Just Serviced			
	CH 0	CH 1	CH 2	CH 3
Highest priority	CH 1	CH 2	CH 3	CH 0
	CH 2	CH 3	CH 0	CH 1
	CH 3	CH 0	CH 1	CH 2
	CH 0	CH 1	CH 2	CH 3
Lowest priority	CH 0	CH 1	CH 2	CH 3

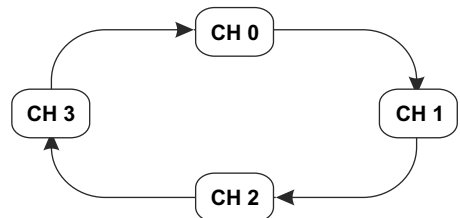


Fig. 9.20 Rotating priority of DMA channels

Bit D_5 can enable the ExTENDED WRITE operation. If the bit is set, the duration of MEMW and I/O signals are extended by activating them earlier in the DMA cycle. This is very useful to interface the peripheral devices with different access times. When the peripheral devices are not accessed within the stipulated time, it is requested to give one or more wait states in the DMA cycle.

Bit D_6 enables terminal count (TC) STOP. When the TC STOP bit is set, a channel is automatically disabled after the terminal count output goes high and prevent any further DMA operation on the same channel. If the DMA operation is to be continued or else if another operation is to begin, the DMA channel must be enabled by a fresh mode set operation in the mode set register.

Bit D_7 of mode set register enables auto load mode. This bit is set when some DMA operation is repeatedly desired—like sending data to CRT monitor. This is known as repetitive or chained DMA operation. Channel 2 and Channel 3 are used for repetitive DMA operation. Generally, Channel 2 registers are initialized as usual for the first data block, while Channel 3 registers are used to store the block re-initialization parameters, i.e., the DMA starting address and the terminal count. After the first data block is transferred using DMA Channel 2, the parameters stored in Channel 3 registers are transferred to Channel 2, if the update flag is set.

Status Register

The status word register of 8257 is shown in Fig. 9.21. The status word can be read to know the status of the terminal counts of the four channels $CH_0 - CH_3$. Any of the lower order 4-bits of the status word register ($D_0 - D_3$) is set, when the terminal count output corresponding to that channel becomes high. These bits remain set till the status register is read or the 8257 is reset.

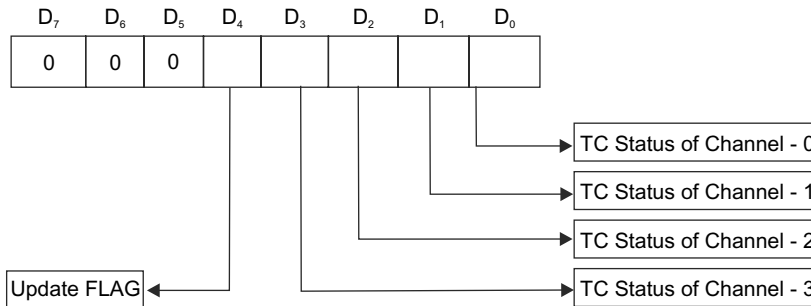


Fig. 9.21 Status register

The update flag is not affected when the status read operation is performed. This flag can be cleared either by resetting 8257 or by resetting the auto load bit in the mode set register. When the update flag is set, the contents of Channel 3 registers are reloaded to the corresponding registers of Channel 2.

✓ **Data Bus Buffer** The 8-bit, bi-directional data bus buffer is interfaced with the internal bus of 8257 and also with the external system bus. The data bus buffer is tri-state type.

✓ **Read/Write Logic** The 8257 can operate in either slave mode or master mode. In the slave mode, the read/write logic accepts the I/O Read ($\overline{I/O\overline{R}}$) or I/O Write ($\overline{I/O\overline{W}}$) control signals. It decodes the $A_0 - A_3$ lines and either writes the contents of the data bus to the addressed internal register or reads the contents of the selected register depending on $\overline{I/O\overline{W}}$ or $\overline{I/O\overline{R}}$. During the master mode operation, the read/write logic generates the $\overline{I/O\overline{R}}$ and $\overline{I/O\overline{W}}$ signals to control the data flow to or from the selected peripheral devices.

✓ **Control Unit** The control logic unit controls the sequences of DMA operations and generates the following control signals: AEN, ADSTB, MEMR, MEMW, TC and MARK. This unit generates the address lines $A_4 - A_7$ in master mode.

✓ **Priority Resolver** The priority resolver resolves the priority of DMA channels of 8257 depending on fixed priority or rotating priority.

9.3.3 DMA Operations

The 8257 is able to perform three types of operations such as verify DMA operation, write operation and read operation. This device operates in four different modes, namely, single-byte data transfer mode, burst mode, control overdrive mode and not ready mode.

Single-byte data Transfer Mode The complete DMA operation of 8257 in single-byte data transfer mode is described below with the help of a flowchart as shown in Fig. 9.22.

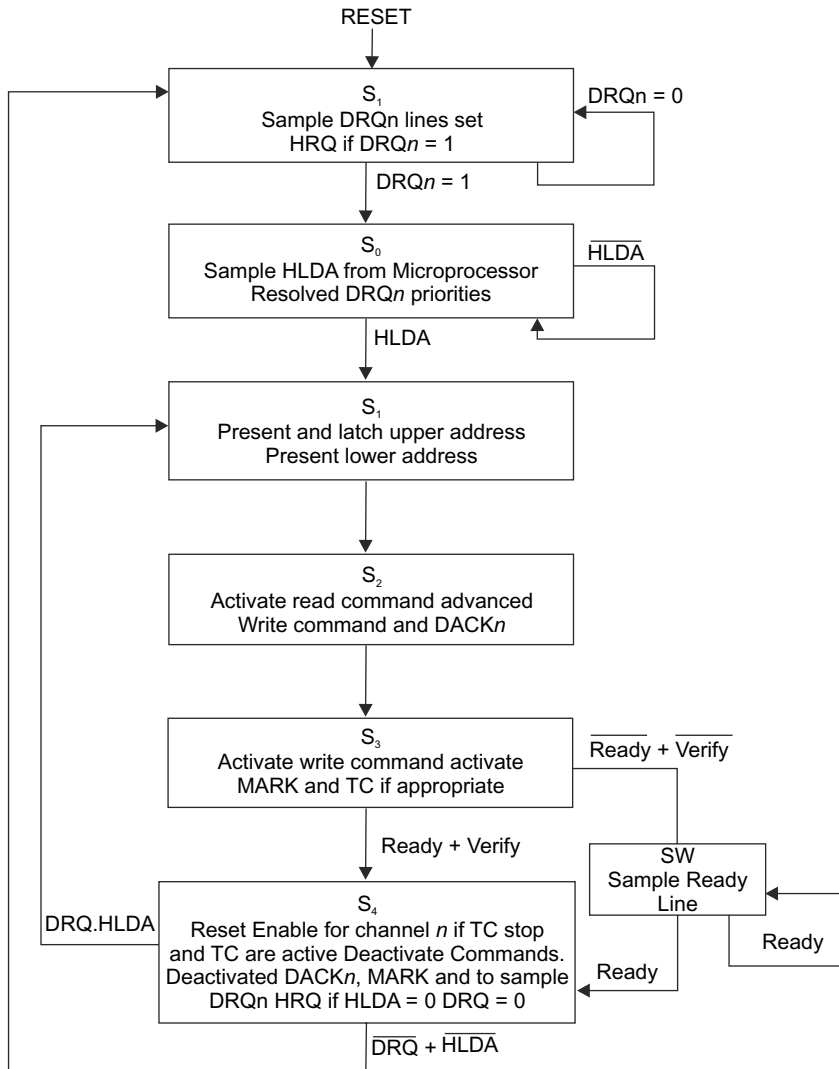


Fig. 9.22 DMA operation state diagram

- ✓ **S_1 State** S_1 is the idle state of DMA operation. In this state, the DMA controller will sample the DMA request inputs (DRQ_n) to check whether any peripheral device want to data transfer between the device and memory. When any DRQ request is received by the 8257, it sends HRQ (HOLD request) signal to the microprocessor and enters S_0 state.
- ✓ **S_0 State** In this state, the DMA controller waits for acknowledgement signal from the CPU at the HLDA input and resolves the priorities of the DMA requests. After detection of a valid HLDA, it exists S_0 and enters S_1 state.
- ✓ **S_1 State** When the DMA controller receives the HLDA signal, it indicates that the bus is available for the transfer. In the S_1 state, the DMA controller write the MSB of the DMA address register on its D_0 – D_7 pins and some external device is used to latch the MSB by making use of the ADSTB signal. The LSB of the DMA address register is put out on the A_0 – A_7 pins. After that it enters S_2 state.
- ✓ **S_2 State** In the S_2 state, the DACK line of the used channel is pulled down by the DMA controller to indicate the peripheral device, which already sends DMA request for the DMA transfer. The read command is activated in this state. I/OR is used for a DMA write operation, and MEMR is used for a DMA read operation. When the extended write option is set in this state, it activates the write command. MEMW is used for a DMA write operation, and I/OW is also used for a DMA read operation. After completion of S_2 state, it enters into S_3 state.
- ✓ **S_3 State** In the S_3 state, the write command is activated. Then DMA controller sets the TC and MARK outputs if the appropriate conditions are satisfied, it enters S_4 state. During S_3 state, it samples the ready input. If the device in which data will be written is not ready, the device makes the ready input to the DMA controller low. Then the DMA controller enters into wait state if the ready input is low. It continues to execute wait cycles until ready input becomes high. When ready input is high, it enters S_4 state.
- ✓ **S_4 State** In this state, if the TC stop and the TC are active (high), the channel just serviced is disabled. The DACK MARK and TC are deactivated. The DMA controller again samples the DMA inputs, and determines their priorities. If there is no DMA request, it resets the HRQ ($HLDA = 0$ or $DRQ = 0$) and enters S_1 state.
- ✓ **Burst Mode and Consecutive Transfers** When more than one channel request services at a time, the DMA controller operates in burst mode for data transfer. No overhead is required in switching from one channel to another. In the S_4 state, DRQ lines are sampled and the highest priority request must be indicated for the next DMA operation. After completion of highest priority DMA channel operations, the next higher priority DMA request must be serviced. The HRQ line is maintained active till all the DRQ lines become low.
- ✓ **Control Override Mode** An external device can interrupt the continuous or burst DMA transfer mode by lowering the HLDA line. After each DMA transfer, the 8257 samples the HLDA line to insure that it is still active. If it is not active, the 8257 completes the current transfer; releases the HRQ line and then returns to the idle state. When DRQ lines are still active, the 8257 may raise the HRQ line in the third cycle and proceed normally.
- ✓ **Not Ready Mode** 8257 uses four clock cycles to complete a transfer. Figure 9.23 shows the timing diagram of DMA operation. The READY input pin is used to interface 8257 with low-speed devices. The READY pin status is sampled in S_3 of the state diagram. If $READY = 0$, the 8257 enters a wait state. The status of READY pin is sampled in every state till it becomes high. Once the $READY = 1$, the 8257 proceeds to state S_4 from S_3 state to complete the transfer.

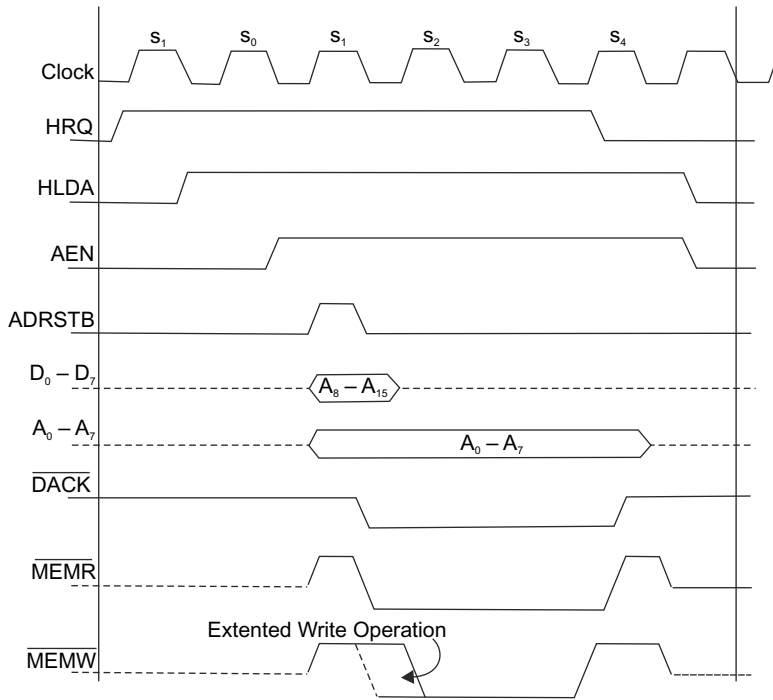


Fig. 9.23 Timing diagram of DMA operation

9.3.4 Interfacing of 8257 with 8085 Microprocessor

The 8257 can be interfaced as a memory mapped device or an I/O mapped device. This device can be operated in slave mode as well as master mode. In this section, the slave mode and master-mode operation are explained briefly with circuit diagram.

Slave-Mode Operation

The interfacing of 8257 with the 8085 processor in slave-mode operation is shown in Fig. 9.24. In this case, the 8257 IC is connected in I/O mapped I/O mode and the \overline{IOR} and \overline{IOW} pins of the IC are connected to the \overline{IOR} and \overline{IOW} control signals. The data lines D₀ - D₇ are connected to the data bus of the microprocessor.

The 8257 can also be connected to the system bus as a memory device instead of as an I/O device. This device operates in memory mapped I/O mode by connecting the system memory control lines to the 8257 I/O control

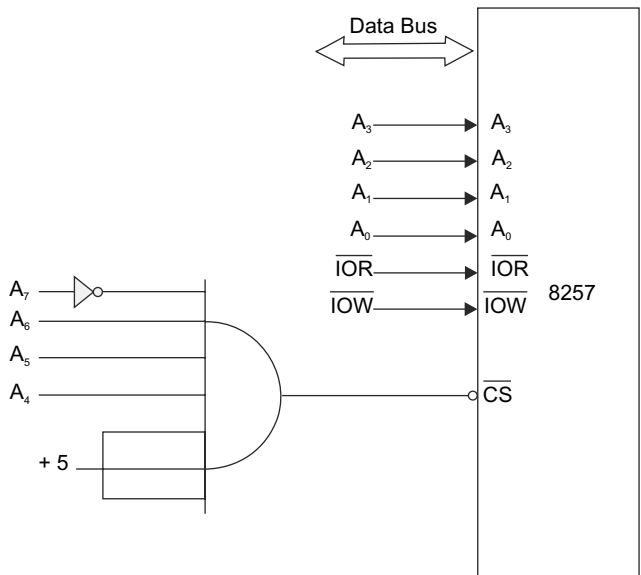


Fig. 9.24 Interfacing of 8257 with 8085 microprocessor in slave mode

lines and the system I/O control lines to 8257 memory control lines. In this case, the \overline{MEMR} and \overline{MEMW} control lines of the system should be connected to the \overline{IOR} and \overline{IOW} input lines of 8257 as shown in Fig. 9.25. The programming of bit 15 (D_{15}) and bit 14 (D_{14}) in the terminal count register is used for different purpose as shown in Table 9.4.

Master-Mode Operation

The 8257 operates in master mode when more than one DMA request lines become active simultaneously. In this mode, CPU is isolated; the DMA controller is activated for data transfer.

The DMA controller should send the address of memory location and control signals \overline{MEMR} , \overline{MEMW} , \overline{IOR} and \overline{IOW} . The Interfacing of 8257 with 8085 microprocessor in master-mode operation is depicted in Fig. 9.26.

In the master mode, the data lines D_0-D_7 acts as the higher order address line A_0-A_{15} . The 8257 enables the signal AEN (address enable). The AEN is used to disable the demultiplexed address bus of A_0-A_7 of the 8085 processor. The 8257 loads the low-order address byte of the DMA address register in A_0-A_7 lines. If the AEN signal is high, the ADSTB (Address Strobe) signal strobes the higher order byte of the DMA address register using the data lines D_0-D_7 .

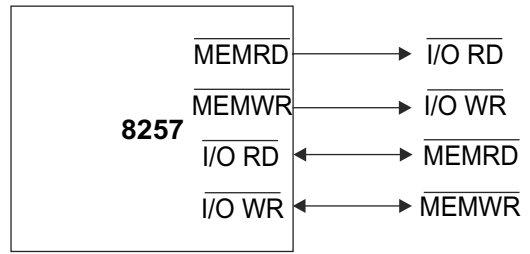


Fig. 9.25 System interface for memory mapped I/O

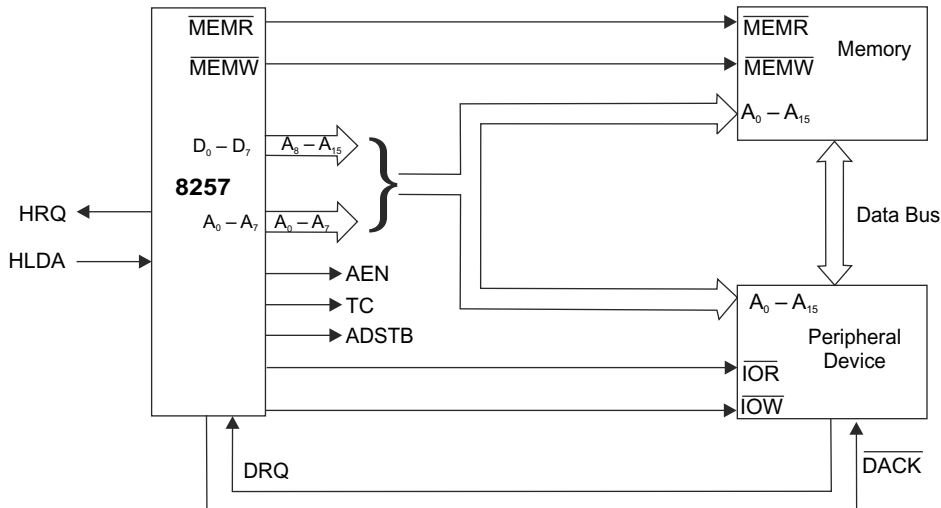


Fig. 9.26 Interfacing of 8257 with 8085 microprocessor in master mode

Depending upon the DMA read or DMA write operation, the control \overline{MEMR} , \overline{MEMW} , \overline{IOR} and \overline{IOW} are activated properly by the 8257. After completion of one-byte data transfer, the content of the count register is decremented by one and the address of the DMA address register is incremented by one. Then 8257 sends the necessary control signals to transfer next byte. For each byte of data transfer, the \overline{DACK} signal is active low. When all the bytes are transferred, the terminal count (TC) signal becomes high.

Example 9.2

Write a program for the data transfer from memory to a disk. Assume the starting address of memory location is 8000H and sixteen data will be transferred.

The address of the DMA address register is 70H and the terminal count (TC) register is 71H. The address of mode set register is 78H. Data transfer is done through Channel 0.

Solution

Figure. 9.27 shows the application of DMA for the data transfer from memory to a disk. The program for the data transfer from memory to a disk is given below:

PROGRAM 9.1

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
9000	3E, 41	MVI	A, 41H		Bit D ₀ = 1 to enable Channel 0. Bit D ₆ = 1 to enable terminal count stop bit. The control word of mode set register is 41 H
9002	D3, 78	OUT	78H		Load 41H into mode set register
9004	3E, 10	MVI	A, 10H		Number of data byte
9006	D3, 71	OUT	71H		(10H) will be loaded into least significant byte of terminal count register
9008	3E, 80	MVI	A, 80H		Bit D ₇ = 1 to indicate the read operation, 80H will be loaded into most significant byte of terminal count register
900A	D3, 71	OUT	71H		16 bit starting address of memory location
900C	3E, 00	MVI	A, 00H		(8000H) will be written into DMA address register of CH-0
900E	D3, 70	OUT	70H		
9010	3E, 80	MVI	A,80H		
9012	D3, 70	OUT	70H		

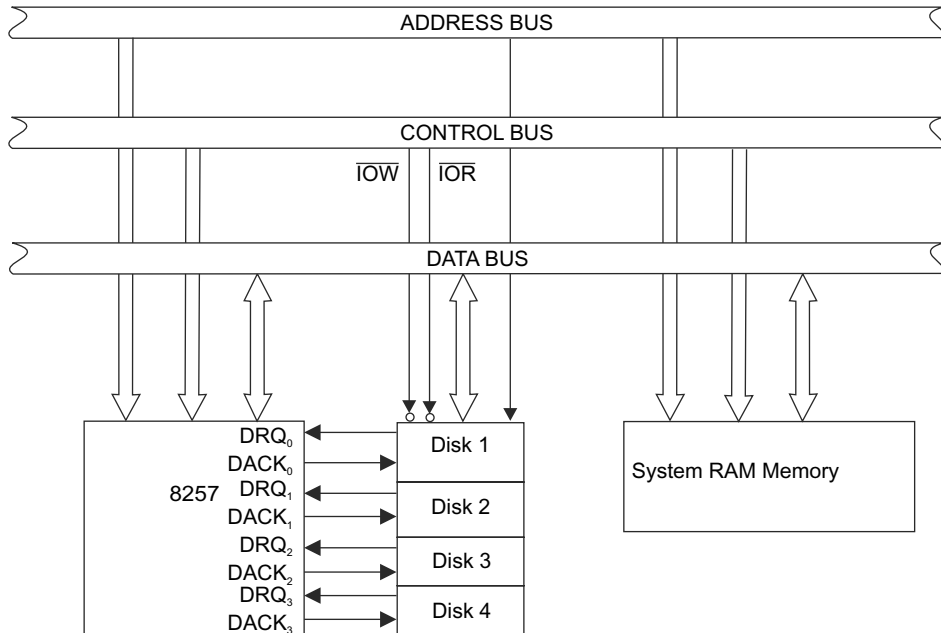


Fig. 9.27 Disk controller using 8257

Example 9.3

Write a program to transfer 45H byte data from a peripheral device to memory. Assume the starting address of the memory location is 8000H. The address of the DMA address register is 72H and the terminal count (TC) register is 73H. The address of mode set register is 78H. Data is to be input through Channel 1.

Solution**PROGRAM 9.2**

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
9000	3E, 41		MVI	A,42H	Bit D ₁ = 1 to enable channel 0. Bit D ₆ = 1 to enable terminal count stop bit. The control word of mode set register is 42 H
9002	D3, 78		OUT	78h	Load 42H into mode set register
9004	3E, 10		MVI	A,45H	Number of data byte (45H) will be loaded into least significant byte of terminal count register
9006	D3, 71		OUT	73H	Bit D ₁₅ = 0, D ₁₄ = 1 and D ₁₃ -D ₈ = 0 for write DMA cycle, 40H will be loaded into most significant byte of terminal count register
9008	3E, 80		MVI	A, 40H	
900A	D3, 71		OUT	73H	16 bit starting address of memory location (8000H) will be written into DMA address register of CH 1
900C	3E, 00		MVI	A, 00H	
900E	D3, 70		OUT	72H	
9010	3E, 80		MVI	A, 80H	
9012	D3, 70		OUT	72H	

9.4 8279—PROGRAMMABLE KEYBOARD AND DISPLAY I/O INTERFACE

In any microprocessor-based system, the keyboard is most commonly used as input device and seven-segment display is used as output device. The programmer presses the keys on the keyboard as desired to feed instruction or data to the CPU. Therefore, the board is constantly scanned to detect a pressed key. The display section is also constantly supplied with data to hold it steady while the CPU operates as scan key and displays it. The CPU will be heavily loaded and the system operation becomes slow, as less time will be available for data processing or ALU operations. If a specific IC performs these operations, then the CPU can handle data processing or ALU operations done very efficiently.

The 8279 is a general-purpose programmable keyboard and displays I/O interface device designed for use in microprocessors. The keyboard portion can provide a scanned interface to a 64-contact key matrix. The keyboard section can also be interfaced to an array of sensors or a strobed interface keyboard. Key depressions can be 2-key lockout or *N*-key rollover. Keyboard entries are debounced and strobed in an 8-character FIFO. When more than 8 characters are entered, the overrun status is set. Key entries set the interrupt output line to the CPU.

The display portion provides a scanned display interface for LED, or any popular display device. Both numeric and alphanumeric segment displays may be used as well as simple indicators. The 8279 has a

16 × 8 display RAM. This 16 × 8 display can be organized into dual 16 × 4. The CPU can load the RAM. Both right entry calculator and left entry typewriter display formats are possible. Both read and write of the display RAM can be done with auto-increment of the display RAM address. The features of 8279 are given below:

- ◆ Simultaneous keyboard and display operations
- ◆ Scanned keyboard mode
- ◆ Scanned sensor mode
- ◆ Strobed input entry mode
- ◆ 8-character keyboard FIFO
- ◆ Single 16-character display
- ◆ 2-key lockout or N-key rollover with contact debounce
- ◆ Dual 8- or 16-numerical display
- ◆ Right or left entry 16-byte display RAM
- ◆ Mode programmable from CPU
- ◆ Programmable scan timing
- ◆ Interrupt output on key entry

9.4.1 Pin Diagram of 8279

The 8279 is packaged in a 40-pin DIP. The pin configuration of 8279 is depicted in Fig. 9.28. The schematic diagram of 8279 is shown in Fig. 9.29. The following is a functional description of each pin.

✓ **DB₀–DB₇ (Bi-directional Data Bus)** All data and commands between the CPU and the programmable keyboard interface, of 8279 are transferred on these lines.

✓ **CLK (Clock)** Generally, a system clock is used to generate internal timing.

✓ **RESET** A high signal on this pin resets the 8279. After being reset, the 8279 is placed in the following mode:

- 16 8-bit character display—left entry
- Encoded scan keyboard—2 key lockout and the program clock prescaler is set to 31.

✓ **CS (Chip Select)** A low on this pin enables the programmable keyboard interface 8279 to receive or transmit data.

✓ **A0 (Buffer Address)** A high on this line indicates that the signals in or out are interpreted as a command or status. A low indicates that they are data.

✓ **RD (Read)** This output signal is activated from microprocessor to 8279 to receive data from external bus.

✓ **WR (Write)** This signal enables the data buffers to send data to the external bus.

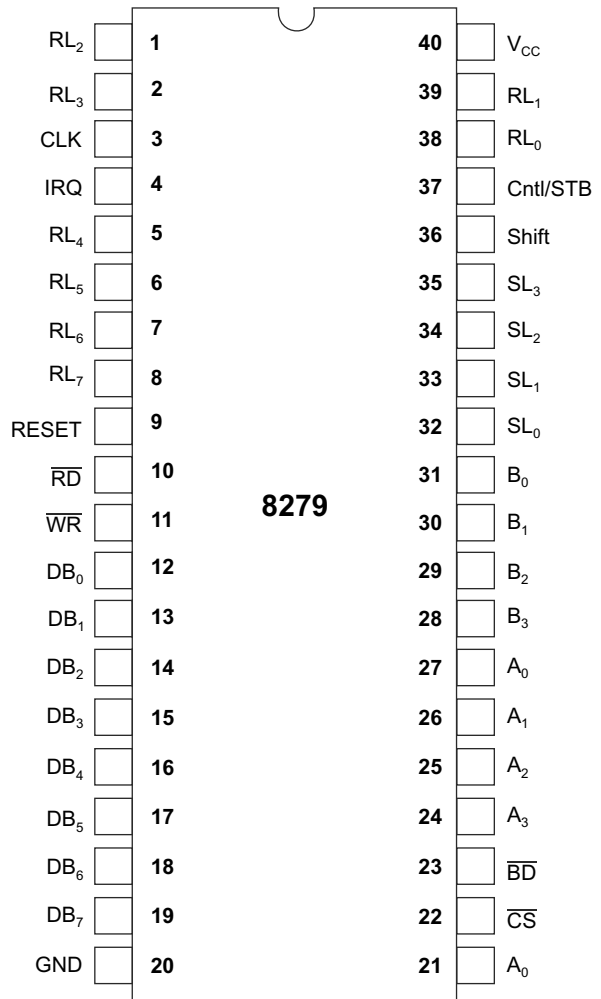


Fig. 9.28 Pin diagram of 8279

✓ **IRQ (Interrupt Request)** In a keyboard mode, the interrupt line is high when there is data in the FIFO/Sensor RAM. The interrupt line becomes low with each FIFO/Sensor RAM read and returns high if there is still information in the RAM. In a sensor mode, the interrupt line goes high whenever a change in a sensor is detected.

✓ **SL₀–SL₃ (Scan Lines)** Scan lines are used to scan the key switch or sensor matrix and the display digits. These lines can be either encoded (1 to 16) or decoded (1 of 4).

✓ **RL₀–RL₇ (Return Line)** These are return line inputs. These are connected to the scan lines through the keys or sensor switches. They have active internal pull ups to keep them high until switch closures are pulled down. The switches are connected between the scan lines and return lines. These lines also serve as an 8-bit input in the strobed input mode.

✓ **SHIFT (Shift)** The shift input status is stored along with the key position on key closure in the scanned keyboard modes. Till a switch closure pulled low, it has an active internal pull up to keep it high.

✓ **CNTL/STB (Control/Strobed Input Mode)** In keyboard modes, the CNTL/STB line is used as a control input and stored like status on a key closure. The line can be used as the strobe line that enters the data into the FIFO in the strobed input mode. It has an active internal pull-up to keep it high until the line is pulled down with a switch closure.

✓ **OUT A₀–OUT A₃ and OUT B₀–OUT B₃ (Outputs)** These are the output ports for the 16 × 4 display refresh registers. The data from these outputs is synchronized to the scan lines (SL₀–SL₃) for multiplexed digit displays. The two 4-bit ports may also be used as one 8-bit port. These two ports may be blanked independently.

✓ **BD (Blank Display)** This output pin is used to blank the display during digit switching or by a blanking command.

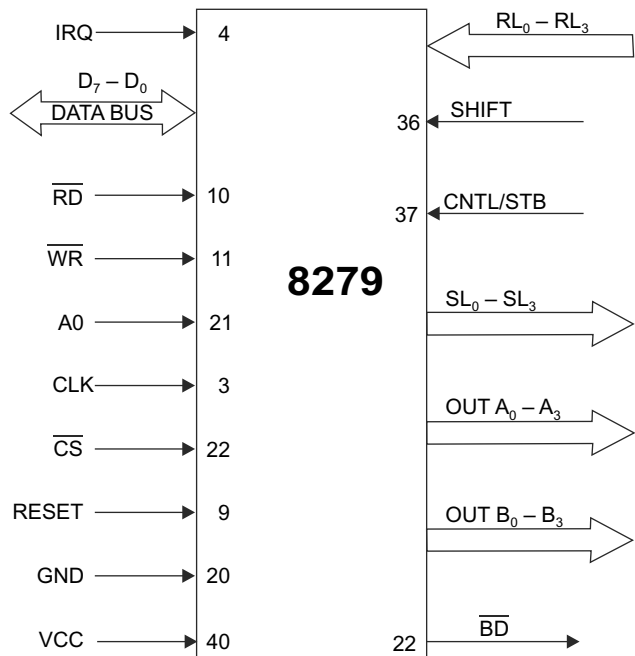


Fig. 9.29 The schematic diagram of 8279

9.4.2 Functional Description

As data input and display are an integral part of all microprocessor designs, the system designer needs an interface that can control these functions without placing a large load on the CPU. The 8279 provides this function for 8-bit microprocessors. The 8279 has two sections: keyboard section with a set of four scan lines and eight return lines and display section with a set of eight output lines for interfacing. The functional block diagram is shown in Fig. 9.30 and the description of the major elements of the 8279 programmable keyboard/display interface device is given below.

I/O Control and Data Buffers

The I/O control section uses the \overline{CS} , \overline{WR} , \overline{RD} and A_0 lines to control data flow to and from the various internal registers and buffers. All data flow to and from the 8279 is enabled by \overline{CS} . The character of the information, given or desired by the CPU, is identified by A_0 . A logic one means the information is a command or status. A logic zero means the information is data. \overline{RD} and \overline{WR} determine the direction of data flow through the data buffers. The data buffers are bi-directional buffers that connect the internal bus to the external bus. When the chip is not selected ($\overline{CS} = 1$), the devices are in a high impedance state. The drivers input using \overline{WR} , \overline{CS} and output using \overline{RD} , \overline{CS} control signals.

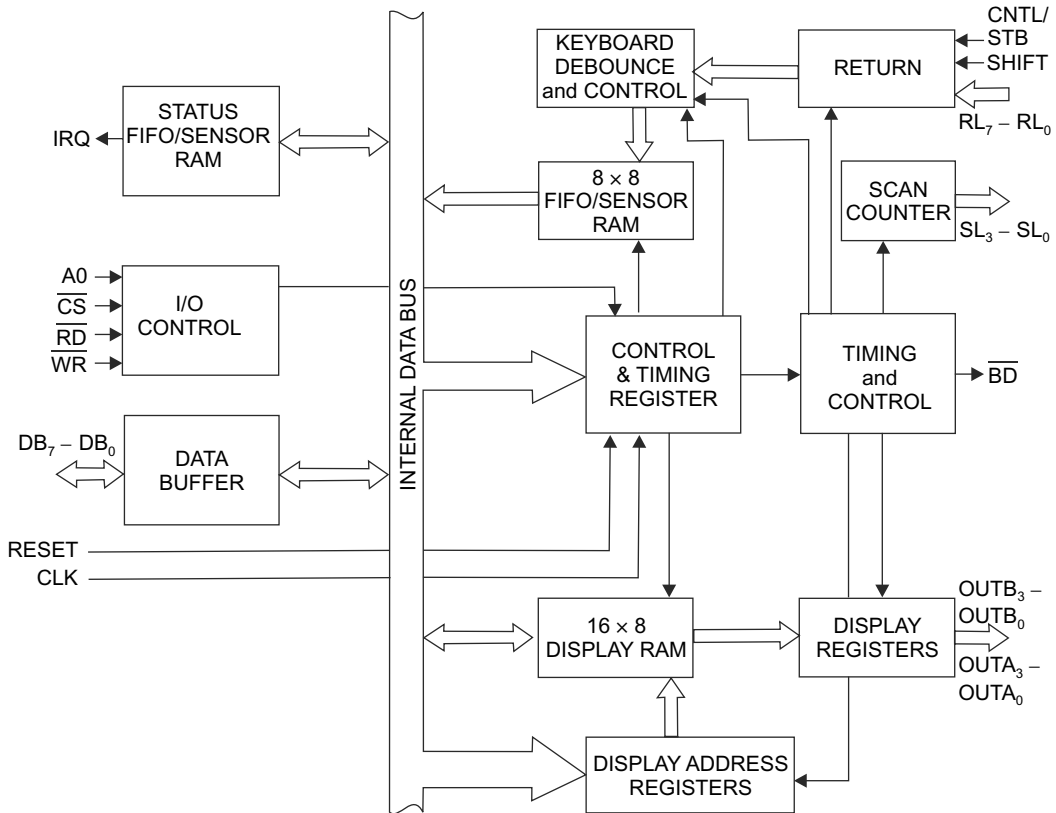


Fig. 9.30 Block diagram of 8279

Control and Timing Registers and Timing Control

These registers store the keyboard and display modes and other operating conditions programmed by the CPU. When $A_0 = 1$, the modes are programmed by presenting the proper command on the data lines and then sending a \overline{WR} . The command is latched on the rising edge of \overline{WR} . The command is then decoded and the appropriate function is set. The timing control unit controls the basic timing counter chain. The first counter is a $\div N$ pre-scaler that can be programmed to yield an internal frequency of 100 kHz which gives a 5.1 ms keyboard scan time and a 10.3 ms de-bounce time. The other counters divide down the internal operating frequency of 8279 to provide the proper key scan, row scan, keyboard matrix scan, and display scan times.

Scan Counter

The scan counter has two modes such as encoded mode and decoded mode. In the encoded mode, the counter provides a binary count that must be externally decoded to provide the scan lines

for the keyboard and display. In the decoded mode, the scan counter decodes the least significant 2 bits and provides a decoded 1 of 4 scan on SL_0 – SL_3 while the keyboard is in decoded scan, it can display. This means that only the first 4 characters in the Display RAM are displayed. In the encoded mode, the scan lines are active high outputs. In the decoded mode, the scan lines are active-low outputs.

Return Buffers and Keyboard Debounce and Control

The 8 return lines are buffered and latched by the return buffers. In the keyboard mode, these lines are scanned, for key closures in row wise. When the debounce circuit detects a closed switch, it waits about 10 ms to check if the switch remains closed. If the switch is closed, the address of the switch in the matrix, the status of SHIFT and CONTROL are transferred to the FIFO. In the scanned sensor matrix modes, the contents of the return lines are directly transferred to the corresponding row of the Sensor RAM (FIFO) each key scan time. In strobed input mode, the contents of the return lines are transferred to the FIFO on the rising edge of the CNTL/STB line pulse.

FIFO/Sensor RAM and Status

In keyboard or strobed input modes, this block is a dual function 8×8 RAM and it operates in FIFO. Each new entry is written into successive RAM positions and then can be read in order of entry. FIFO status keeps track of the number of characters in the FIFO and whether it is full or empty. Too many reads or writes will be recognized as an error. The status can be read by an \overline{RD} with \overline{CS} low and A_0 high. The status logic also provides an IRQ signal when the FIFO is not empty. In scanned sensor matrix mode, the memory unit acts as a Sensor RAM. Each row of the Sensor RAM is loaded with the status of the corresponding row of sensor in the sensor matrix. In this mode, IRQ is high if a change in a sensor is detected.

Display Address Registers and Display RAM

The display address registers hold the address of the word currently being written or read by the CPU and the two 4-bit nibbles can be displayed. The read/ write addresses are programmed by CPU command. The address can be automatically updated after each read or write operation. The CPU can directly read by the Display RAM after the address is set. The addresses for the A and B nibbles are automatically updated by the 8279 to match data entry by the CPU. The A and B nibbles can be entered independently or as one word, depending upon the mode set by the CPU. Data entry to the display can be set to either left or right entry.

9.4.3 Operating Modes of 8279

The 8279 is designed to directly connect to the microprocessor bus. Then CPU can program all operating modes for the 8279. The 8279 operates in input (keyboard) modes and output (display) modes:

Input (Keyboard) Modes

8279 has three input modes, namely, scanned keyboard, scanned sensor matrix, and strobed input.

✓ **Scanned Keyboard** In this mode, 8279 can be encoded (8×8 key keyboard) or decoded (4×8 key keyboard) by scan lines. A key depression generates a 6-bit encoding of key position. Position, and shift and control status are stored in the FIFO. Keys are automatically debounced with 2-key lockout or N -key rollover.

✓ **Scanned Sensor Matrix** In this mode, a sensor array will be interfaced with 8279 with encoded (8×8 matrix switches) or decoded (4×8 matrix switches) scan lines. Key status are stored in RAM addressable by CPU.

✓ **Strobed Input** Data on return lines during control line strobe is stored in the FIFO.

Output (Display) Modes

8279 provides two output modes such as display scan and display entry.

- ✓ **Display Scan** In this mode, Programmable Key Board and Display Controller 8279 provides 8 or 16 character multiplexed displays that can be organized as dual 4-bit or single 8-bit ($B_0 = D_0, A_3 = D_7$) display unit.
- ✓ **Display Entry** Right entry or left entry display formats are executable for 8279 IC.

9.4.4 Software Operation

The following commands program the 8279 operating modes. The commands are sent on the data bus with $\overline{CS} = 0$ and $A_0 = 1$ and are loaded to the 8279 on the rising edge of . All commands of 8279 are discussed below:

Keyboard/Display Mode Set

The command-word format to select different modes of operation of 8279 is given below:

MSB				LSB			
D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
0	0	0	D	D	K	K	K

where DD is the display mode and KKK is the keyboard mode.

✓ **DD Display Mode**

- 0 0 Eight 8-bit character display—Left entry
- 0 1 Sixteen 8-bit character display—Left entry
- 1 0 Eight 8-bit character display—Right entry
- 1 1 Sixteen 8-bit character display—Right entry

✓ **KKK Keyboard Modes**

- 0 0 0 Encoded Scan Keyboard—2 Key Lockout
- 0 0 1 Decoded Scan Keyboard—2-Key Lockout
- 0 1 0 Encoded Scan Keyboard—N-Key Rollover
- 0 1 1 Decoded Scan Keyboard—N-Key Rollover
- 1 0 0 Encoded Scan Sensor Matrix
- 1 0 1 Decoded Scan Sensor Matrix
- 1 1 0 Strobed Input, Encoded Display Scan
- 1 1 1 Strobed Input, Decoded Display Scan

Program Clock

The clock for operation of 8279 is programmable. All timing signals are generated by an internal prescaler, which divides the external clock by a programmable integer. Bits P P P P P determine the value of the integer from 2 to 31. When the system clock frequency of 2 MHz is divided by 20 (10100) to get the clock frequency 2/20 MHz or 100 kHz.

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
0	0	1	P	P	P	P	P

Read FIFO/Sensor RAM

The command format of Read FIFO/Sensor RAM is given below:

If the user requirements is to blank the display, the BL flags are available for each nibble. Both BL bits will be cleared for blanking both nibbles.

Clear Display RAM

This command format for clear display RAM operation is given below:

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
1	1	1	CD ₂	CD ₁	CD ₀	CF	CA

The CD₂, CD₁, CD₀ bits are available in this command to clear all rows of the Display RAM to a selectable blanking code as given below:

CD ₂	CD ₁	CD ₀	
1	0	x	All zeros x don't care
1	1	0	A ₃ -A ₀ = 2 (0010) and B ₃ -B ₀ = 0 (0000)
1	1	1	All ones

CD₂ must be 1 for enabling the clear display command. When CD₂ = 0, the clear display command is invoked by setting CA = 1 and CD₁, CD₀ bits must be same.

If the CF bit is 1, the FIFO status is cleared and the interrupt output line is reset. Also, the Sensor RAM pointer is set to row 0.

If the clear all bit (CA) is set to 1, this combines the effect of CD and CF bits. This CA uses the CD clearing code on the Display RAM and also clears the FIFO status.

End Interrupt/Error Mode Set

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
1	1	1	E	x	x	x	x

For the sensor matrix modes, this command lowers the IRQ line and enables further writing into RAM. Therefore, if any in-sensor value is detected, the IRQ line becomes high which inhibits writing into the RAM.

For the N-key rollover mode, if the E bit is programmed to '1', the 8279 IC can operate in the special error mode.

Data Format

In the scanned keyboard mode, the character entered into the FIFO corresponds to the position of the switch in the keyboard plus the status of the CNTL and SHIFT lines. CNTL is the MSB of the character and SHIFT is the next most significant bit. The next three bits D₅-D₃ are from the scan counter and indicate the position of the row the key was found in. The last three bits D₂-D₀ are from the column counter and indicate the position of the column on which the key is pressed.

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
CNTL	SHIFT	SCAN			RETURN		

In the sensor matrix mode, the data on the return lines (RL₇-RL₀) is entered directly in the row of the Sensor RAM that corresponds to the row in the matrix being scanned. Therefore, each switch position maps directly to a Sensor RAM position. The SHIFT and CNTL inputs are ignored in this mode and switches are not necessarily the only things that can be connected to the return lines. Any logic that can be triggered by the scan lines can enter data to the return line inputs. Eight multiplexed input ports could be tied to the return lines and scanned by the 8279.

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
RL ₇	RL ₆	RL ₅	RL ₄	RL ₃	RL ₂	RL ₁	RL ₀

Display

✓ **Left Entry** The left-entry mode is the simplest display format. This mode is just like typewriter mode. In this mode, each display position directly corresponds to a byte in the display RAM. The first entry goes to the left most display position of the Display RAM; the second entry to just the right of the first one. In this way, the 16th entry goes to the 15th address position of Display RAM. Entering characters from position zero causes the display to fill from the left. Therefore, the 17th entry goes to the Display RAM address 0 and 18th entry goes to Display RAM address 1 as depicted in Fig. 9.31.

✓ **Right Entry** The right entry mode is most commonly used in electronic calculators. The first entry is placed in the right most display character of Display RAM. The second entry goes to the right most character after the display is shifted left one character as shown in Fig. 9.32. In this way, the 17th entry goes to the Display RAM address 0 and the 18th entry goes to Display RAM address 1 and the left most character is shifted off the end and is lost. In this mode, there is no correspondence between Display RAM address and the display position.

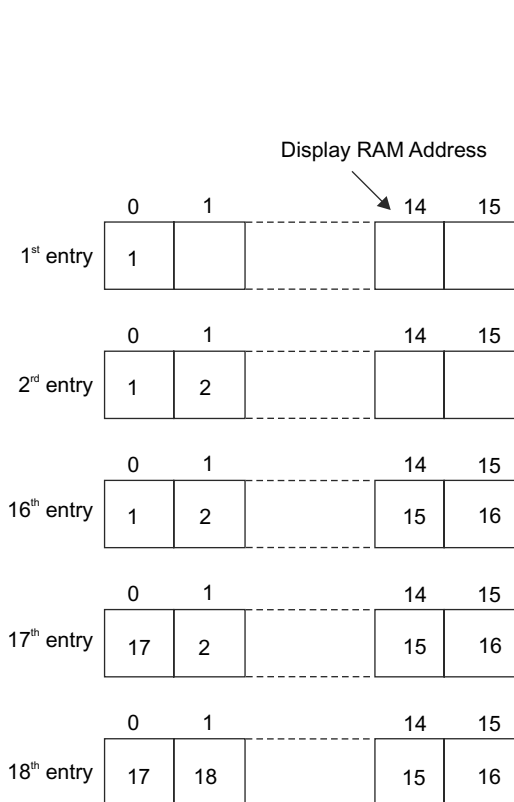


Fig. 9.31 Left-mode entry (auto increment)

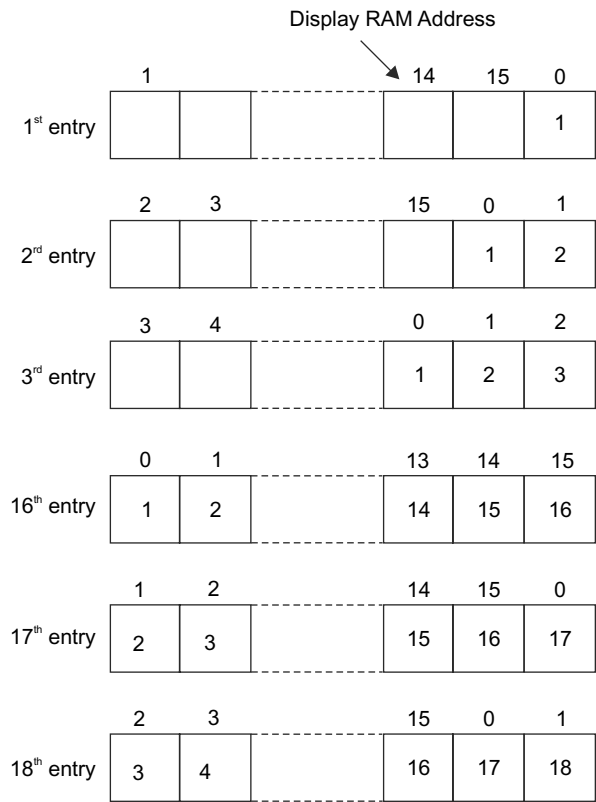


Fig. 9.32 Right mode entry (auto increment)

9.4.5 Interfacing 8279 with Microprocessor

The 8279 can be used as a memory mapped I/O or as an I/O mapped I/O device. The interfacing of 8279 with the microprocessor in I/O mapped I/O mode is depicted in Fig. 9.33. This circuit consists of 8 data lines, RESET, \overline{RD} , \overline{WR} , \overline{CS} , CLK and C/\overline{D} .

8 data lines D_7 – D_0 are connected to the data bus of the microprocessor.

The RESET signal is connected to the RESET OUT of the microprocessor. \overline{RD} and \overline{WR} signals are connected to \overline{IOR} and \overline{IOW} control signals.

The address decoder output is also connected to the \overline{CS} pin of 8279 in order to access the IC as an I/O mapped device. When the \overline{CS} signal is active low, the device can communicate with the microprocessor. The address of the data port is 30H and the address of the command port is 31H. The command/data signal is connected to A_0 address line of the microprocessor for addressing data register and command register sequentially. The clock signal CLK is linked to the system clock.

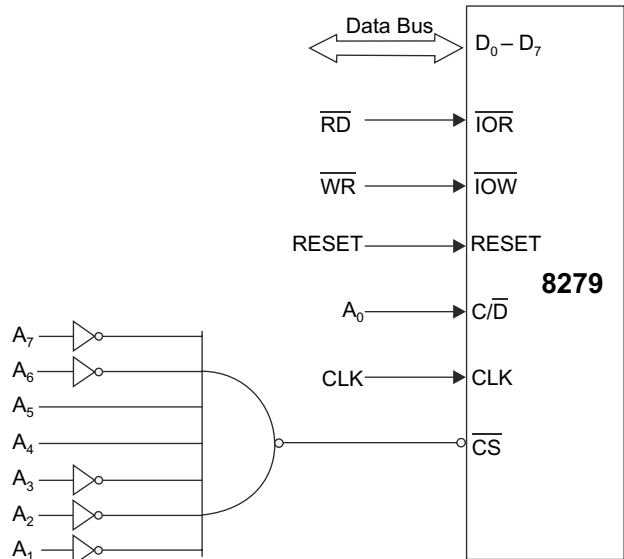


Fig. 9.33 Interfacing of 8279 with microprocessor

9.4.6 Keyboard Interface of 8279

The keyboard interface of 8279 is depicted in Fig. 9.34. To recognize the keyboard data, the device has a keyboard buffer RAM. The scan lines S_0 , S_1 and S_3 are used as inputs of 3 lines to 8 lines decoder. The decoder output lines drive eight rows of keys. The return lines RL_0 – R_7 are used as column lines for the keyboard. The 8 row and 8 column signals can enable the 8279 to interface a 64-key keyboard.

Sometimes one key may have more than one function. The shift and control keys are used generally for this purpose. When any key and either shift or control or both are pressed together, three more function keys are available from a key. In this way, 64 keys are able to provide 256 functions. For this purpose, control and shift lines take care of the control and shift keys. Once a key is pressed, an 8-bit code must be loaded into the buffer RAM and the interrupt request sends a signal to the microprocessor which can indicate that the buffer RAM is not empty.

9.4.7 Sixteen Digit Display Interface of 8279

Figure 9.35 shows the sixteen-digit display using 8279. This consists of sixteen 8 bit buffer RAM to hold the 8-bit data for 16 characters or digits. All display digits are multiplexed by four lines to sixteen lines decoder. Therefore, all digits are not turned ON at a time, but they are ON sequentially. The four scan lines output 0 to 15 in a cyclic manner, which can be decoded into 16 different lines by the decoder to select any one digit of the 16-character display. The lines A_0 – A_3 and B_0 – B_3 send the 8-bit information for display. The \overline{BD} is used to blank all display digits. If the first digit is selected for display, the content of the first display buffer will be placed in A_0 – A_3 and B_0 – B_3 lines.

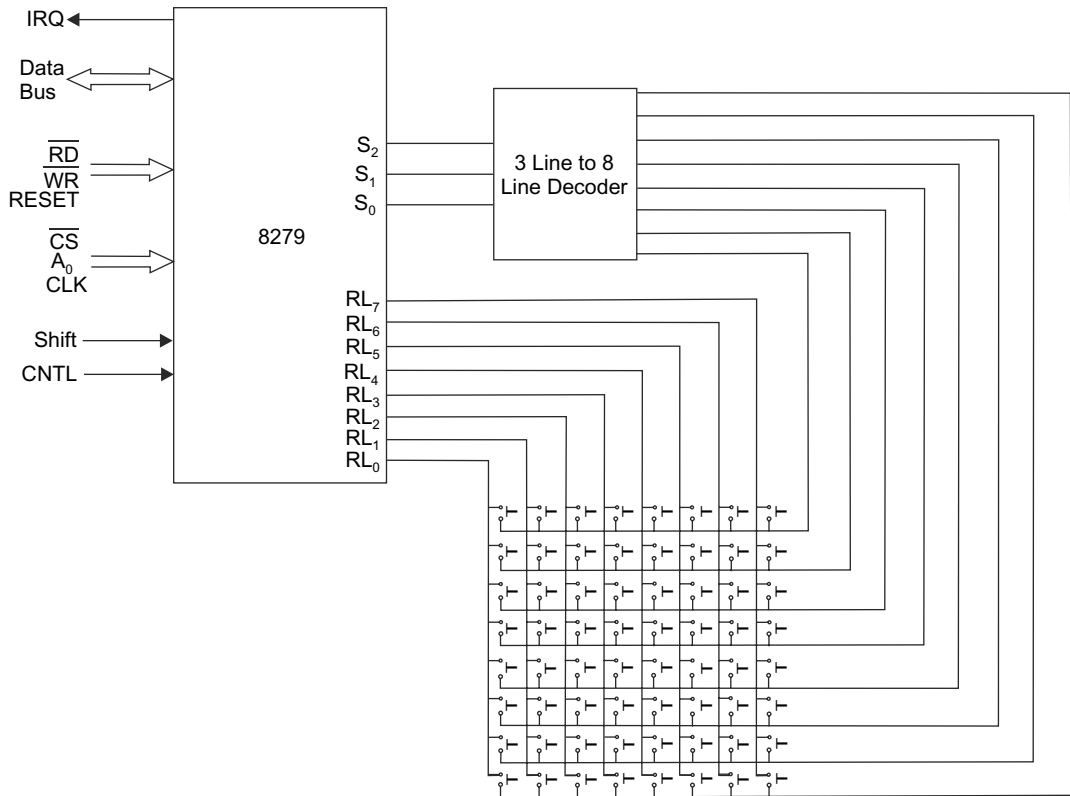


Fig. 9.34 Keyboard interface with 8279

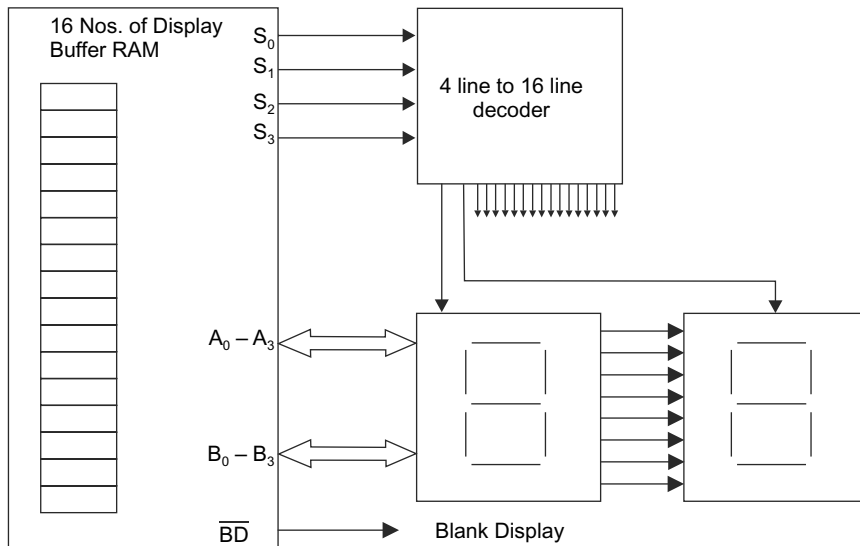


Fig. 9.35 Sixteen-digit display interface with 8279

9.5 8275 CRT CONTROLLER

The 8275 programmable CRT controller is a single-chip device that generates all the signals which are used for interfacing a microprocessor system. The main function of the 8275 CRT controller is to refresh the display by buffering the information from the main memory. This is also used to keep track of the current display position of the screen. This device has the following features:

- ◆ Programmable screen and character format
- ◆ Six independent visual field attributes
- ◆ Eleven visual character attributes
- ◆ Four types of cursor control
- ◆ Light pen detection and registers
- ◆ Dual row buffer
- ◆ Programmable DMA burst mode

9.5.1 Functional Description of 8275

Figure 9.36 shows the internal architecture of the 8275 CRT controller and its functional description of each block have been explained below:

Data Bus Buffer

This tri-state bi-directional 8-bit buffer is used to interface the 8275 with the system data bus. The data bus buffer accepts inputs from the system control bus and generates control signals for overall device operation. It consists of the command, parameter and status registers. This bus is also used to read or write the internal registers of 8275 CRT controller.

- ✓ **\overline{RD} (Read)** When \overline{RD} is active low, the CPU can read data or status information from the 8275.
- ✓ **\overline{WR} (Write)** If \overline{WR} is active low, the CPU can write data or control words to the 8275.
- ✓ **DRQ (DMA Request)** When the DRQ output pin is high, the 8275 desires a DMA transfer.
- ✓ **\overline{DECK} (DMA Acknowledge)** The active low input on \overline{DECK} pins informs the 8275 that a DMA cycle is in progress.
- ✓ **IRQ (Interrupt Request)** When the IRQ output pin is high, it informs the CPU that 8275 desires interrupt service. The operation of read/ write /DMA control logic function is given in Table 9.6.

Table 9.6

A_0	\overline{RD}	\overline{WR}	\overline{CS}	Operations
0	0	1	0	Write 8275 parameter Register
0	1	0	0	Read 8275 parameter Register
1	0	1	0	Write 8275 command Register
1	1	0	0	Read 8275 status Register
x	1	1	0	Tri-state
x	x	x	1	Tri-state

Character Counter

The character counter is a programmable counter. It is used to determine the number of characters to be displayed per row and the length of the horizontal retrace interval. This is driven by CCLK (Character Clock) input.

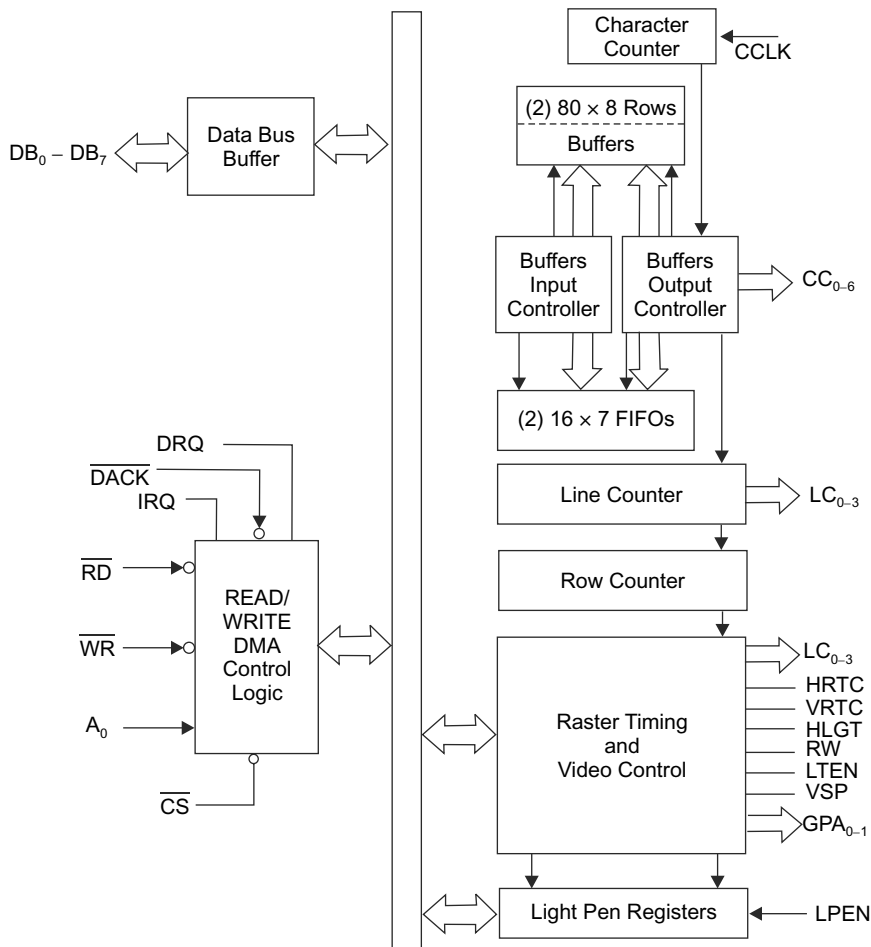


Fig. 9.36 Block diagram of internal architecture of 8275

Line Counter This is a programmable counter. It is used to determine the number of horizontal lines (sweeps) per character row.

Row Counter The row counter is a programmable counter. It is used to determine the number of character rows to be displayed per frame and the length of vertical retrace interval.

Light Pen Registers The light pen registers consists of two registers that can store the contents of the character counter and row counter whenever a rising edge is detected at the LPEN input pin.

Raster Timing and Video Control The raster timing circuit controls the timing of the Horizontal Retrace (HRTC) and Vertical Retrace (VRTC) outputs. The video control circuit generates LA_0 – LA_1 (Line Attribute), HGLT (Highlight), RVV (Reserve Video), LTEN (Light Enable), VSP (Video Suppress) and GPA_0 – GPA_1 (General Purpose Attribute) output signals.

Row Buffers

The row buffers have two 80 character buffers. These buffers are filled from the microcomputer system memory with the character codes to be displayed. When a row buffer is displaying a row of characters, the other is being filled with the next row of characters to be displayed later.

FIFOs

There are two sixteen-character FIFOs in the 8275 CRT controller. FIFOs are used to provide extra row buffer length in the transparent attribute mode.

Buffer Input/Output Controllers

The buffer input/output controllers are used to decode the character information being placed in the row buffers. When the character is a character attribute, field attribute or a special code, these controllers decide the appropriate action.

9.5.2 Pin Descriptions of 8275

The pin diagram of 8275 CRT controller is shown in Fig. 9.37 and the function of each pin is explained in this section:

✓ **Line Counter, LC₃–LC₀ (Output)** These are the output signals of the line counter which is used to address the character generator for the current line position on the screen.

✓ **DMA Request, DRQ (Output)** This output signal is used to request the 8257 DMA controller for a DMA operation.

✓ **DMA Acknowledge, DACK (Input)** The \overline{DACK} input signal is used to acknowledge that the requested DMA cycle has been accepted.

✓ **Horizontal Retrace, HRTC (Output)** The HRTC output signal is active during the programmed horizontal retrace interval. During the active HRTC, the VSP is high and the LTEN is low.

✓ **Vertical Retrace, VRTC (Output)** The VRTC output signal is active during the programmed vertical retrace interval. During this period, the VSP is high and the LTEN is low.

✓ **\overline{RD} , Read (Input)** This \overline{RD} is a control signal to read the internal registers of 8275.

✓ **\overline{WR} Write (Input)** The \overline{WR} input control signal is used to write commands into the control registers of 8275 or to write data into the row buffers during a DMA cycle.

✓ **Light Pen, LPEN (Input)** This is an input signal from the CRT controller which informs the 8275 that a light pen signal has been detected.

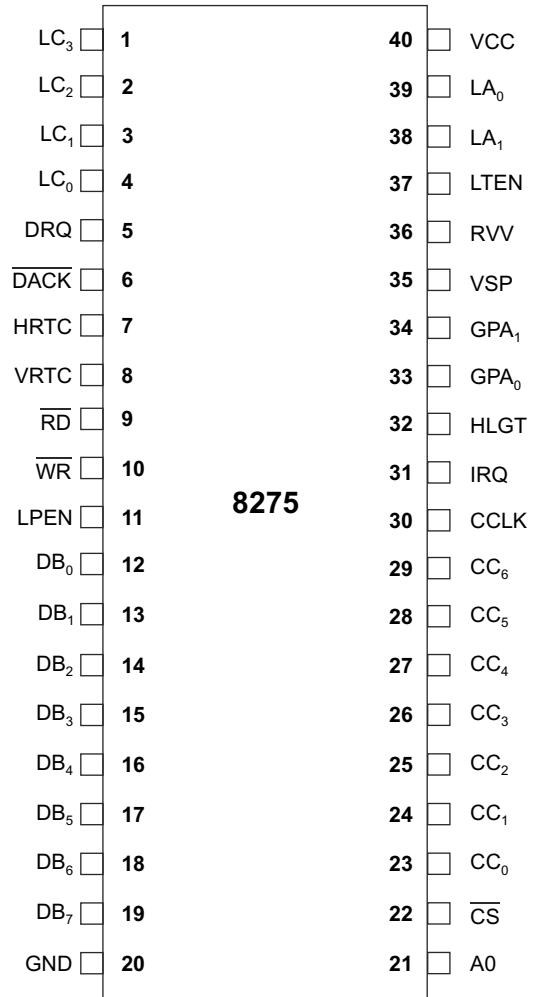


Fig. 9.37 Pin diagram of 8275

- ✓ **Data Bus, DB_7 – DB_0 (Input/Output)** The DB_7 – DB_0 lines are bidirectional tri-state data buses that are used for read or write operations from/to the 8275 internal registers.
- ✓ **Video Suppression, VSP , (Output)** This output signal is used to blank the video signal to the CRT. This signal is active high,
 - ◆ During the horizontal and vertical retrace intervals,
 - ◆ At the top and bottom lines of rows, if an underline is programmed to be at line number 8 or greater than 8
 - ◆ When an end-of-screen or an end-of-row is detected
 - ◆ When DMA under-run occurs
 - ◆ At regular intervals to create blinking displays if specified by the cursor character attribute or field attributes programmed
- ✓ **Reverse Video, RVV (Output)** This output signal is used to indicate the CRT circuit to reverse the video signal. This is active at the cursor positions, when a reverse video block cursor is programmed.
- ✓ **Light Enable, $LTEN$ (Output)** The $LTEN$ output signal is used to enable the video signal to CRT. This output is active at the programmed underline cursor position and at the position specified by the attribute codes.
- ✓ **A_0 Port Address (Input)** The input address line is A_0 . When this pin is high, it selects the ‘C’ port or the command register is selected and if it is low, the ‘P’ port or the parameter register of 8275 will be selected.
- ✓ **Character Code, CC_6 – CC_0 (Output)** These are output lines from the row buffers used for character selection from the character generator.
- ✓ **CS , Chip Select (Input)** The read and write operations are enabled by CS .
- ✓ **Character Clock, $CCLK$ (Input)** This is a clock input terminal that is driven by the dot/timing logic.
- ✓ **Interrupt Request, IRQ (Output)** The IRQ output pin is used to generate an interrupt request to the CPU.
- ✓ **Highlight, $HLGT$ (Output)** This output signal is used to intensify the display at the particular positions on the screen as specified by the character attribute codes or the field attributes codes.
- ✓ **General-Purpose Attribute Codes, GPA_7 – GPA_0 (Output)** These output signals are enabled by the general-purpose field attribute codes.
- ✓ **Line Attribute Codes LA_7 – LA_0 (Output)** These line attribute codes have to be decoded externally by the dot/timing logic to generate the horizontal and vertical line combinations for the graphic displays as specified by the character attribute codes.
- ✓ **Vcc** This pin connected with +5 V power supply.
- ✓ **Ground (GND)** The ground pin of 8257 is connected to power supply ground terminal.

9.5.3 System Operation

Figure 9.38 shows the interfacing of the 8275 CRT controller with an 8257 DMA controller. The 8275 CRT controller operates with an 8257 DMA controller and the standard character generator ROM for dot matrix decoding in a microprocessor based system. The dot-level timing signals are provided by an external circuit.

The 8257 is programmable to a large number of display formats. This controller provides raster timing, display row buffering, visual attributes decoding, cursor timing and light pen detection. Initially, the 8275

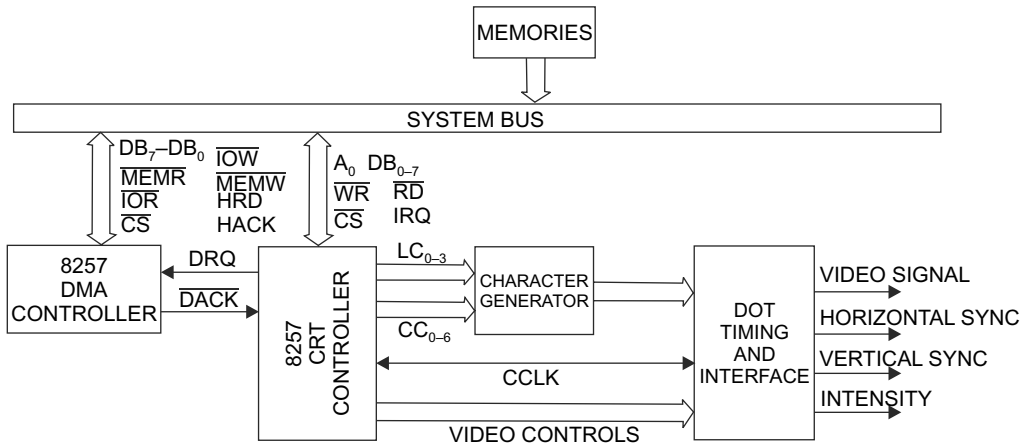


Fig. 9.38 Interfacing of 8275 with 8257 DMA

picks up a row of characters to be displayed from the system memory and load them into an 80-character row buffer. When one of the two buffers is filled up with characters, the other buffer is on display. In this way, the two row buffers are used to display one by one till the complete display is over.

The number of characters per row and the number of rows per display frame are programmable. While one buffer is being displayed and the other is busy in already displayed then the 8275 CRT controller sends a request for a DMA cycle to fill the already displayed buffer. This process will continue till the complete display frame is over.

The 8275 displays characters rows one line at a time as shown in Fig. 9.39. The 8275 controls the raster timing. This is possible by generating horizontal retrace (HRTC) and vertical retrace (VRTC) signals. The timing of these signals is also programmable. The 8275 generates a cursor and location of cursor is programmable.

9.6 ANALOG-TO-DIGITAL CONVERTER INTERFACING

The analog-to-digital conversion (ADC) is the reverse operation of digital-to-analog conversion (DAC). Figure 9.40 shows the block diagram of ADC, which consists of filter, sample and hold, quantizer and digital processor. The filter circuit is used to avoid the aliasing of high-frequency signals and passes the baseband frequency signal of ADC. Sometimes this filter is also called *antialiasing filter*. After the filter, a sample and hold circuit is used to maintain constant the analog input voltage of ADC during the period when the analog signal is converted into digital. This time period is also called *conversion time of ADC*. The quantizer circuit is used after sample and hold to segment the reference voltage into different

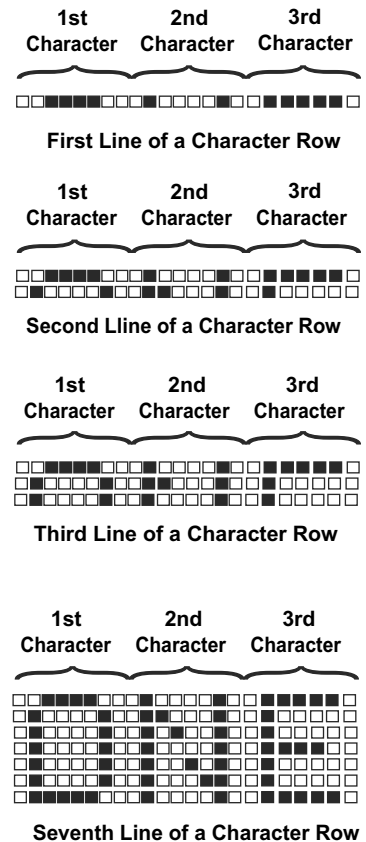


Fig. 9.39 Display of character using 8275 CRT controller

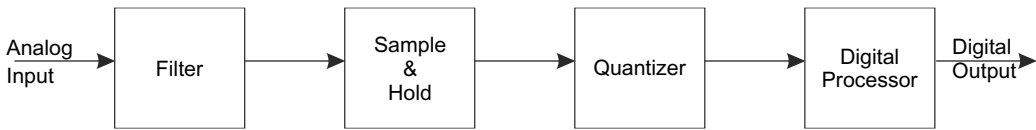


Fig. 9.40 *Block diagram of ADC*

ranges. If ‘N’ number of digital bits represents analog voltage, there are $2N$ possible subranges. The quantizer determines the specified subranges corresponding to an analog input voltage. The digital processor can encode the corresponding digital output. There are different types of ADCs. The classifications of ADC architectures based on speed are slow-speed ADCs, medium high-speed ADCs and fast speed ADCs. Single-slope and dual-slope serial ADCs are slow-speed type and their resolution is very high and accuracy is very good. Medium-speeds ADCs are successive approximation ADCs, and parallel or flash ADCs are high speed ADCs. Resolution is moderate for medium-speed ADCs and low for flash ADCs. Accuracy of medium-speed ADCs is good but flash ADCs have limited accuracy.

9.6.1 Counting-Type A/D Converter

Counting-type ADCs are of two types; single-slope serial ADC and dual-slope serial ADC. The operation of a counting A/D converter is given below:

The principle of operation of single-slope serial ADC is to generate a ramp voltage using DAC, which is compared with the analog input voltage. At the start of the ramp, the counter is started to count from the initial value. When the ramp reaches the analog input voltage, the counter is stopped. The digital value in the counter is directly related to the input voltage. This converter takes longer time to convert a large voltage than a small one and some control signals are required for the start of conversions and end of conversions. The maximum conversion time is $2NT$, when $2N$ clock pulses are required to convert, where N is the number of bits, T is the clock period. The disadvantage of this ADC is that it is unipolar due to single-slope ramp generator.

Figure 9.41 shows the block diagram of a single-slope analog-to-digital converter. This converter consists of a ramp generator, binary counter, comparator, and AND gate. Here, the counter is used to generate a digital output. Initially, an analog input is sampled and held and then applied to the positive terminal of the comparator. The counter is in reset condition and the clock is applied in AND gate and counter. When the first clock pulse is applied, the ramp generator starts to integrate a reference voltage V . When V_{in} is greater than the output of the ramp generator, the comparator output is high and a clock pulse is applied to the counter to count clock pulses. If the output of the ramp generator is equal to V_{in} , the comparator output is low. The

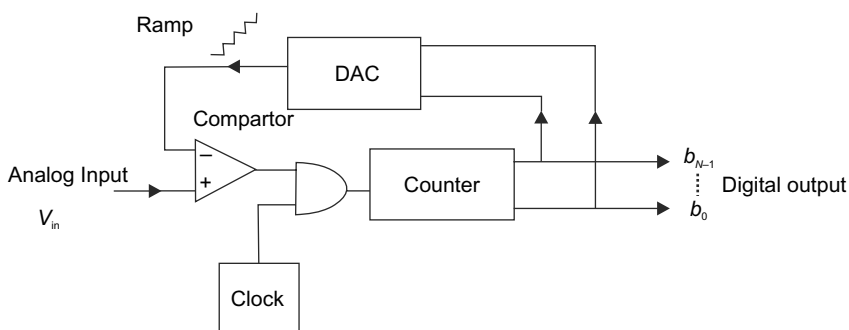


Fig. 9.41 *Block diagram of single-slope serial ADC*

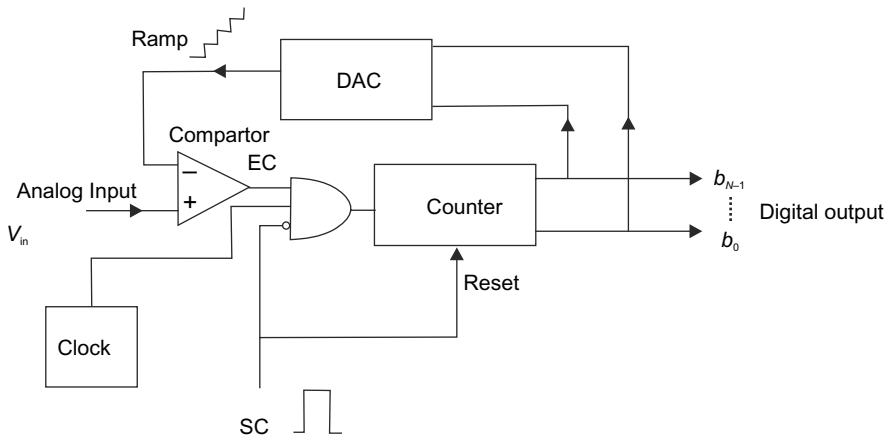


Fig. 9.42 Single-slope serial ADC with SC and EC signals

output of the counter is the desired digital output of analog voltage. Single-slope serial ADC with start of conversion (SC) and end of conversion (EC) is illustrated in Fig. 9.42. The conversion sequence of single-slope ADC is given below:

- (i) Start of conversion signal resets the counter to zero, and enables the gate to allow clock pulses to be counted in the counter.
- (ii) The counter outputs are fed into a DAC to generate a ramp output.
- (iii) Then the ramp output is compared with the sampled input signal. The gate output is high till the ramp voltage equals the input signal.
- (iv) When the ramp output voltage is equal to the input signal, the gate output becomes low and counting stops.
- (v) The gate-disabled signal can be used to indicate the end of conversion.

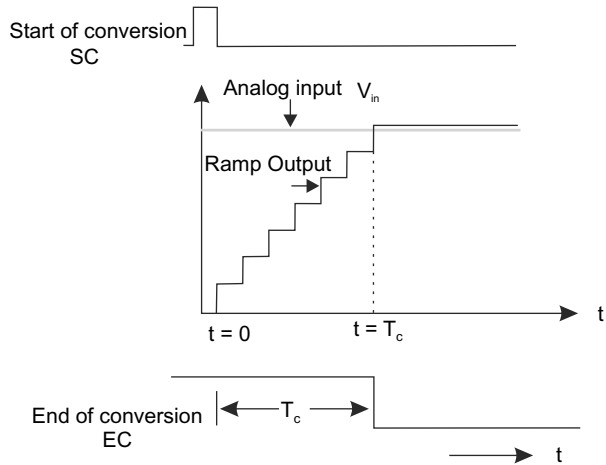


Fig. 9.43 Waveforms of single slope serial ADC with SC and EC signals

Figure 9.43 shows the timing diagram of single slope ADC.

9.6.2 Successive Approximation ADC

The major drawback of single-slope and dual-slope ramp and counter types of ADC is that the length of conversion time is very high. The maximum conversion time is $2N$ clock cycles, where ' N ' is the number of bits. To reduce the conversion time, successive approximation type ADC is very much useful. This converter is similar to counter-type ADC, but this converter uses a pattern generator rather than a clock to obtain digital equivalent value. The pattern generator simply sets one bit at a time starting with the MSB. Therefore, the approximation starts by placing logic 1 on the Most Significant Bit (MSB). Then the output of the DAC is compared with the sampled input signal. If the output of the DAC is too high, MSB is reset to logic 0, but if

it is too low it is left at logic 1 and the next bit is set. This process is repeated until all bits are at the correct logic levels in sequence. Consequently, an N -bit ADC will only need ‘ N ’ attempts before all the bits are corrected. Therefore, conversion time is independent of the size of the analog voltage but it depends upon the number of bits.

Figure 9.44 shows the successive-approximation-type ADC converter. This converter consists of a comparator, a DAC, digital control logic and Successive Approximation Register (SAR). The function of the

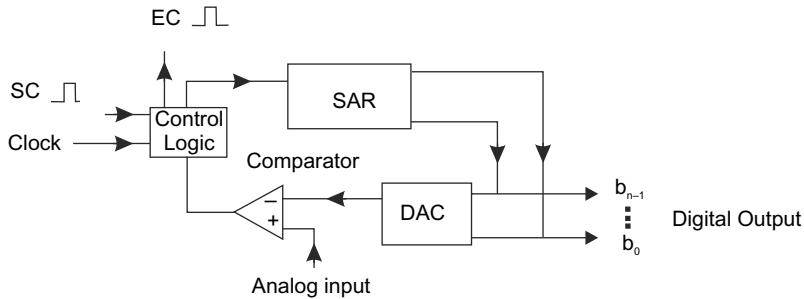


Fig. 9.44 Successive approximation type ADC

digital control logic is to determine the value of each bit in a sequential manner based on the output of the comparator. The conversion process starts with sampling and holding the analog voltage when the start of conversion signal is given. The digital control logic sets the MSB and resets all other bits. This digital data is fed to DAC, which generates analog voltage $V_{ref}/2$ and applies to the comparator to compare with the input voltage V_{in} . When the comparator output is high, the digital control logic makes the MSB 1. If the comparator output is low, the digital control logic makes the MSB 0. After completion of this step, the next MSB is 1 and other bits are 0. Again, the sampled input is compared to the output of the DAC with this digital data. When the comparator is high, the second bit is proven to be 1. If the comparator is low, the second bit is 0. In this way, the process will continue until all bits of digital data have not checked by successive approximation. The successive approximation process for converging to the analog output voltage of DAC is shown in Fig. 9.45 (a) and (b).

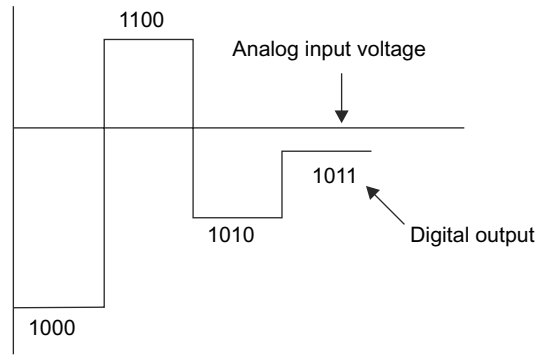


Fig. 9.45(a) Successive approximation ADC for a analog input voltage

The successive approximation process for converging to the analog output voltage of DAC is shown in Fig. 9.45 (a) and (b). The number of cycles for conversion for N -bit ADC is ‘ N ’. The bipolar analog to digital conversion can be achieved by using a sign bit either $+V$ or $-V$.

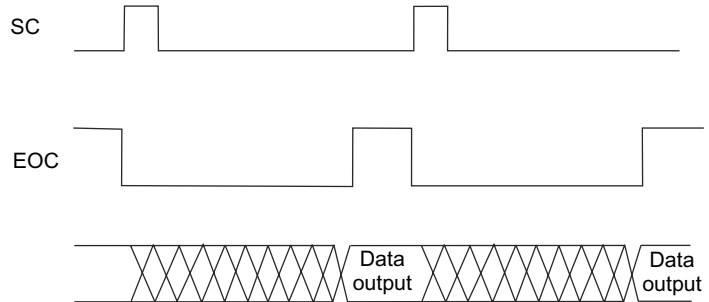


Fig. 9.45(b) Timing diagram of ADC

9.6.3 Parallel or Flash Converter

Figure 9.46 shows a flash ADC. In case of a three-bit flash ADC, reference voltage V is divided into eight different voltages $\frac{V}{14}, \frac{3}{14}V, \frac{5}{14}V, \frac{7}{14}V, \frac{9}{14}V, \frac{11}{14}V, \frac{13}{14}V$, and V . Each voltage is applied to the non-inverting terminal of a comparator. The outputs of comparators are fed to the encoder, and the encoder output is the digital data of analog input. When V_i is $.7V$, the output of comparator C_7 and C_6 are 1 and other comparators C_5, C_4, C_3, C_2, C_1 are low. In this case, the digital output of encoder is 101, which is equivalent to analog input voltage. Therefore, flash-type ADC converter converts analog voltage into digital output in one clock pulse but in two phases. In the first phase, the analog input voltage is sampled and applied to the comparator inputs. In the second phase, digital encoder determines the correct digital output and stores it in a register. Flash ADC can be used as bipolar converter when weighted resistances are connected between $+V$ and $-V$. Table 9.7 shows the analog input, comparator output, and digital output of flash-type ADC.

The advantage of a flash converter is high speed but many comparators are required. For a three-bit flash converter, 7 comparators are required and an 8-bit flash converter requires 255 comparators on a chip. Therefore, power dissipation is very large.

Table 9.7 Analog input, comparator output and digital output of flash converter

Analog input voltage	Comparator outputs							Digital output		
V_i	C_7	C_6	C_5	C_4	C_3	C_2	C_1	b_2	b_1	b_0
$0 \leq V_i < V/14$	0	0	0	0	0	0	0	0	0	0
$V/14 < V_i < 3V/14$	0	0	0	0	0	0	1	0	0	1
$3V/14 < V_i < 5V/14$	0	0	0	0	0	1	1	0	1	0
$5V/14 < V_i < 7V/14$	0	0	0	0	1	1	1	0	1	1
$7V/14 < V_i < 9V/14$	0	0	0	1	1	1	1	1	0	0
$9V/14 < V_i < 11V/14$	0	0	1	1	1	1	1	1	0	1
$11V/14 < V_i < 13V/14$	0	1	1	1	1	1	1	1	1	0
$13V/14 < V_i \leq V$	1	1	1	1	1	1	1	1	1	1

9.6.4 Specification of ADC

Generally, manufacturers use the following specifications of analog-to-digital converter:

- ◆ Analog input-voltage range
- ◆ Input impedance
- ◆ Accuracy
- ◆ Quantization error
- ◆ Resolution
- ◆ Conversion time
- ◆ Format of digital output
- ◆ Temperature stability

Analog input-voltage range

It is the maximum allowable input-voltage range in which ADC will operate properly. Actually, it is the difference between the smallest and largest analog input voltages to use the full range of digital outputs. Typical values are 0 to 10V, 0 to 12V, $\pm 5V$, $\pm 10V$, and $\pm 12V$.

Input Impedance

The input impedance of ADC varies from 1 Kohm to 1 Mohm, depending on type of ADC. Input capacitance of ADC is approximately some picofarads.

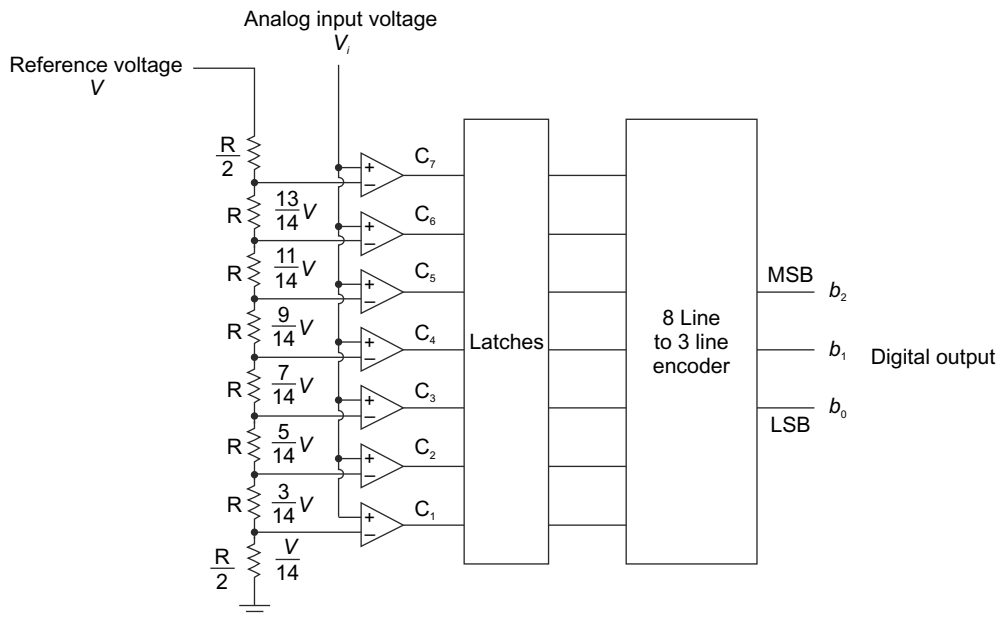


Fig. 9.46 Flash ADC

Quantization Error The full-scale range of analog input voltage is quantized for conversion to a finite number of steps. The error in process of quantization is called as quantization error. Generally, the quantization error is specified as $\frac{1}{2}$ LSB.

Accuracy The accuracy of an ADC depends on quantization error, digital system noise, gain error, offset error, and deviation from linearity, etc. Accuracy is determined from sum of all types of errors. Typical values of accuracy are $\pm 0.001\%$, $\pm 0.01\%$, $\pm 0.02\%$, and $\pm 0.04\%$ of full-scale value.

Resolution The resolution is defined by the ratio of reference voltage to number of output states. Actually it is smallest change in analog voltage for LSB.

Resolution = Reference voltage/($2N-1$) where N = Number of bits of the ADC.

Conversion Time The conversion time of medium speed ADC is about $50 \mu\text{s}$ and high speed ADC's conversion time is about some ns. Therefore conversion time varies from $50 \mu\text{s}$ to few ns for slow/medium speed to high-speed ADC.

Format of Digital Output Generally ADC always uses any standard code, namely, unipolar binary, bipolar binary, offset binary, one's complement and two's complement, etc.

Temperature Stability Accuracy of A/D converter depends on temperature variation. Typical temperature coefficients of error are $30 \text{ ppm}/^\circ\text{C}$.

9.6.5 ADC ICs

The simplified configuration of an ADC IC is shown in Fig. 9.47. The IC performs analog to digital conversion by using — Start of Conversion (SC), End of Conversion (EOC) and output enable signals. Commonly available ADC ICs are single channel 8-bit A/D converter ADC0800, eight channels 8-bit A/D converter ADC0808/0809, twelve channels 8-bit A/D converter ADC0816/0817, 12-bit A/D converter ADC80.

The ADC0800 is an 8-bit monolithic A/D converter using P channel ion-implanted MOS technology. It consists of a high input impedance comparator, 256 series resistors and analog switches, control logic and output latches as shown in Fig. 9.48. Conversion is performed using a successive approximation technique where the unknown analog voltage is compared to the voltage of R network using analog switches. When the appropriate R network voltage matches the unknown voltage, conversion is complete and the digital outputs will be an 8-bit complementary binary word corresponding to the unknown voltage. Figure 9.49 shows the timing diagram of this converter. The features of the ADC0800 are low cost, input ranges $\pm 5V$ to $\pm 10V$, no missing codes, ratiometric conversion, Tri-state outputs, contains output latches, TTL compatible, supply voltages 5 VDC and 12 VDC, resolution of 8 bits, linearity ± 1 LSB, conversion speed 40 clock periods, clock range 50 to 800 kHz. Table 9.8 shows the maximum values of ADC's performance characteristics.

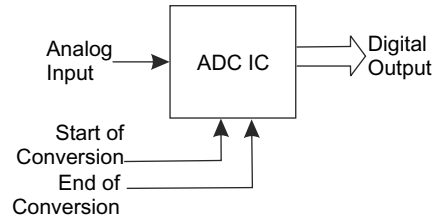


Fig. 9.47 Schematic diagram of ADC

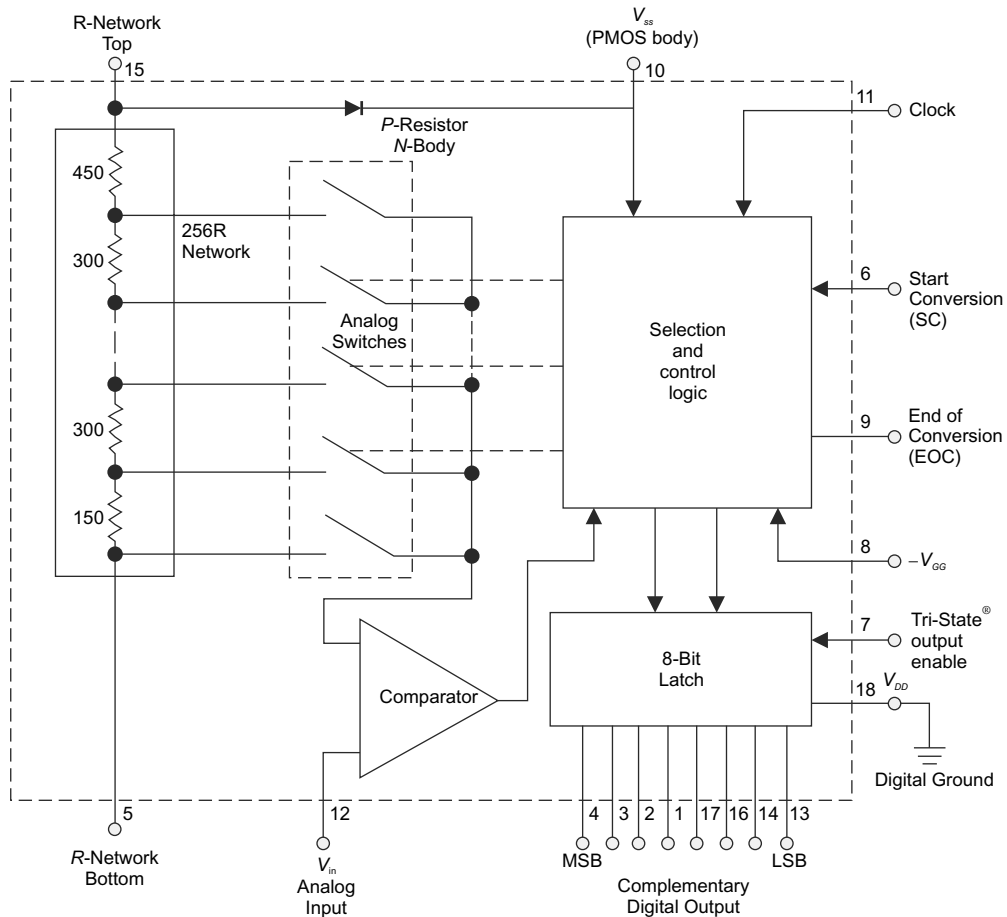


Fig. 9.48 Logic diagram of ADC0800

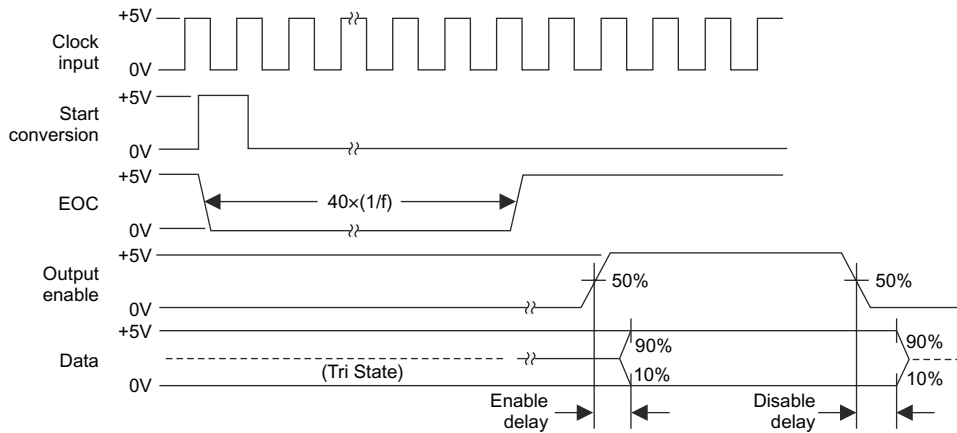


Fig. 9.49 Timing diagram of ADC0800

Table 9.8 Performance characteristics of ADC0800

Parameters	Maximum value
Non-Linearity	± 2 LSB
Differential Non-Linearity	$\pm \frac{1}{2}$ LSB
Zero Error	± 2 LSB
Zero Error Temperature Coefficient	0.01 %/°C
Full-Scale Error	± 2 LSB
Full-Scale Error Temperature Coefficient	0.01 %/°C
Input Leakage current	1 μ A
Clock Frequency	800KHz
Clock Pulse Duty Cycle	60 %
TRI-STATE Enable/Disable Time	1 μ s
Start Conversion Pulse	3½ clock pulse
Power Supply Current	20 mA

The ADC80 is a 12-bit successive approximation type A/D converter. It is available in a 32 pin DIP. The important performance characteristics of ADC80 is given in Table 9.9.

Table 9.9 Performance characteristics of ADC80

Parameters	Maximum value
Linearity error	$\pm 0.012\%$
Differential non-linearity	$\pm \frac{1}{2}$ LSB
Full-scale error temperature coefficient	30 ppm/°C
Conversion time	25 μ s
Analog input voltage	± 2.5 V, ± 5 V, ± 10 V, 0 to 5 V, 0 to 10 V
Digital output format	Unipolar and bipolar
Power loss	800 mW

9.6.6 Interfacing of ADC0800

Figure 9.50 shows the interfacing connections of ADC0800 with the 8085 microprocessor. Here, 8255 is used in between ADC 0800 and 8085 microprocessors. Port A and Port C lower of 8255 are used as inputs and Port B and Port C upper of 8255 are used as outputs. The control word of 8255 when Port A and Port C upper are used as inputs and Port B and Port C lower are used as outputs is 98H. The address of Port A is 00H, the address of Port B is 01H and the address of Port C is 02H. The control word address is 03H.

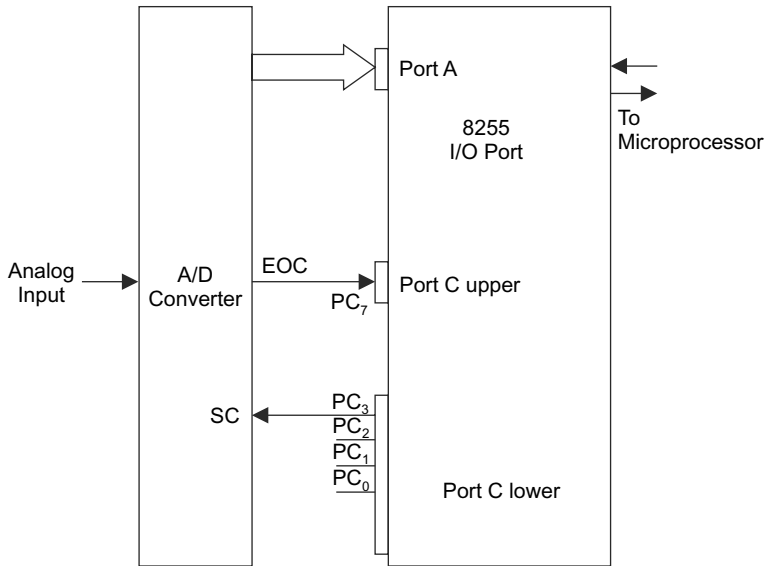


Fig. 9.50 Interfacing of ADC 0800 with 8255

The Start Of Conversion (SC) of ADC is connected with PC₃ of Port C lower, the End Of Conversion (EOC) is connected with the PC₇ of Port C upper. The output of ADC IC is also connected with Port A of 8255.

Initially, the start of conversion signal will be high to start the conversion process. For this, 08H is sent to Port C by the microprocessor. The port address of Port C is 02H. The instruction OUT 02 sends the content of the accumulator to Port C lower and the SC pin becomes high. This pin signal will be high only for clock pulse duration as it is used to start the conversion process only. Therefore, 00H is loaded into the accumulator by instruction MVI A,00H. The OUT 02 instruction makes the pin PC₃ low. When the analog-to-digital conversion has been started, some times are taken by the conversion process. At the End Of Conversion, the ADC sends the End of Conversion Signal (EOC). So the microprocessor should check the EOC signal from time to time. If EOC is high, ADC conversion has been completed. To check EOC signal IN 00H and RAL instructions are used. If carry is generated, the EOC becomes high and the conversion has been completed. When no carry is generated, it means that conversion is not completed, so it jumps to the level LOOP to recheck the status of PC₇. After completion of conversion, the microprocessor reads the output of ADC through the instruction IN 00H. As the ADC output is available in complement form, the CMA instruction is used to convert into the final result. Then the result, content of accumulator can be stored into a specified memory location. Result of ADC conversion is given in Table 9.10. The program for ADC interfacing is given below:

PROGRAM 9.3

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	3E, 98		MVI	A, 98H	Load control word (98H) of 8255 in accumulator
8002	D3, 03		OUT	03H	Write control word in control word register and initialize ports
8004	3E, 08		MVI	A, 08H	Send start of conversion signal through PC ₃
8006	D3, 02		OUT	02H	PC ₃ is high
8008	3E, 00		MVI	A,00H	As PC ₃ will be high for one or two clock pulse, make it 0
800A	D3,02		OUT	02H	PC ₃ becomes low
800C	DB, 02	LOOP	IN	02	Read end of conversion signal
800E	17		RAL		Rotate accumulator to check either conversion is over or not.
800F	D2, 0C, 80		JNC	LOOP	If conversion is not completed, jump to LOOP
8012	DB, 00		IN	00	Read digital output of A/D converter
8014	2F		CMA		Complement of ADC output
8015	21, 00, 81		LXI		H,8100H
8018	77		MOV	M, A	Store accumulator content in 8000H location
8019	76		HLT		Stop

Table 9.10 Result of ADC conversion

<i>Analog Input</i>	<i>Digital output</i>
5V	FF
4V	ED
3V	C7
2V	B9
1V	93
0V	80

It is clear from the above result that for 5 V analog input digital output is FFH and for 0 V input, the digital output is 80H. To modify the output result, 80 is subtracted from result. Then the modified result is given in Table 9.11. The modified program is given below.

PROGRAM 9.4

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	3E, 98		MVI	A, 98H	Load control word of 8255 in accumulator
8002	D3, 03		OUT	03H	Write control word in control word register and initialize ports
8004	3E, 08		MVI	A,08H	Send start of conversion signal through PC ₃
8006	D3,02		OUT	02H	PC ₃ is high
8008	3E, 00		MVI	A ,00H	As PC ₃ will be high for one or two clock pulse, make it 0

(Contd.)

(Contd.)

800A	D3,02		OUT	02H	PC ₃ becomes low
800C	DB, 02	LOOP	IN	02	Read end of conversion signal
800E	17		RAL		Rotate accumulator to check either conversion is over or not.
800F	D2, 0C, 80		JNC	LOOP	If conversion is not completed, jump to LOOP
8012	DB, 00		IN	00	Read digital output of A/D converter
8014	2F		CMA		Complement of ADC output
8015	D6, 80		SUI	80H	Subtract 80H
8017	21, 00, 81		LXI	H,8100H	
801A	77		MOV	M,A	Store accumulator content in 8100H location
801B	76		HLT		Stop

Table 9.11 Result of ADC

Analog Input	Digital Output
5V	7F
4V	6D
3V	47
2V	39
1V	13
0V	00

9.6.7 Interfacing of ADC 0800 and Multiplexer with 8255

An analog multiplexer is required for larger number of analog inputs. The microprocessor sends the channel select signals to the multiplexer to get the desired analog input voltage from the selected channel. Figure 9.51

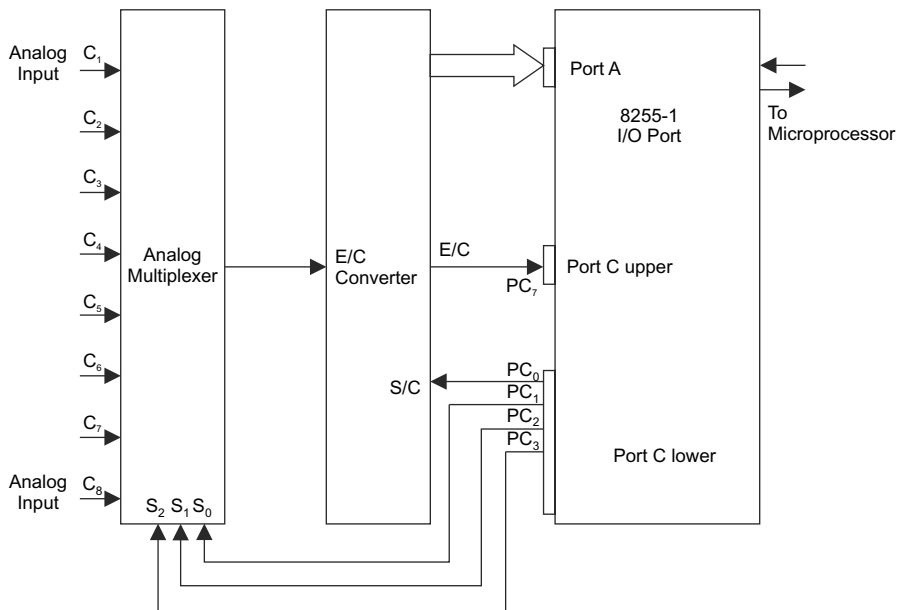


Fig. 9.51 Interfacing of ADC 0800 and multiplexer with 8255

shows the schematic circuit diagram, which consists of analog multiplexer, A/D converter and 8255. The analog multiplexer has eight channels. To select any one channel, send channel select signals through the Port C lower. The SC signal is connected with the pin PC₃ and EOC signal is connected with PC₇. Analog input is applied to Channel 1 of the multiplexer. When the microprocessor sends the 00H into Port C, Channel 1 will be selected and the input voltage of Channel 1 is fed to ADC converter IC. After that, the microprocessor sends the SC signal to ADC to start the conversion process. This SC signal will be high only for one clock pulse duration. After that the microprocessor checks the end of conversion signal, whether the conversion process is completed or not. When EOC is high, the conversion stands completed and the microprocessor read the ADC output and stores it in a memory location. If the analog voltage is applied to any one of the channels, the applied voltage will be converted into its digital equivalent value in the same way.

PROGRAM 9.5

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	3E, 98		MVI	A,98H	Load control word of 8255 in accumulator
8002	D3,03		OUT	03H	Write control word in control word register and initialize ports
8004	3E,00		MVI	A,00H	Load 00H to select the multiplexer channel
8006	D3, 02		OUT	02H	Channel 1 is selected
8008	3E, 08		MVI	A,08H	Send start of conversion signal through PC ₃
800A	D3, 02		OUT	02H	PC ₃ is high
800C	3E, 00		MVI	A,00H	As PC ₃ will be high for 1or two clock pulse, make it 0
800E	D3,02		OUT	02H	PC ₃ becomes low
8010	DB, 02		LOOP	IN 02	Read end of conversion signal
8012	17		RAL		Rotate accumulator to check whether conversion is over or not
8013	D2, 10, 80		JNC	LOOP	If conversion is not completed, jump to LOOP
8016	DB, 00		IN	00	Read digital output of A/D converter
8018	2F		CMA		Complement of ADC output
8019	D6, 80		SUI	80H	Subtract 80H
801B	21, 00, 81		LXI	H, 8100H	
801E	77		MOV	M,A	Store accumulator content in 8000H location
801F	76		HLT		Stop

9.6.8 Interfacing of 12 BIT ADC 0800

Figure 9.52 shows the interfacing of 12-bit ADC with microprocessor through 8255. Port A, Port B and Port C upper of 8255 are used as inputs and Port C lower of 8255 is used as output. The start of conversion (SC) of ADC is connected with PC₃ of Port C lower; the end of conversion (EOC) is connected with the PC₇ of Port C upper. The output of ADC IC is also connected with Port A and Port B of 8255. PA₀–PA₇ are considered as LSBs of digital output of ADC and PB₀ – PB₃ are used as MSBs of digital output of ADC. The PB₄–PB₇ are opened and considered as logic 1. The program for 12 bit ADC interfacing is given below:

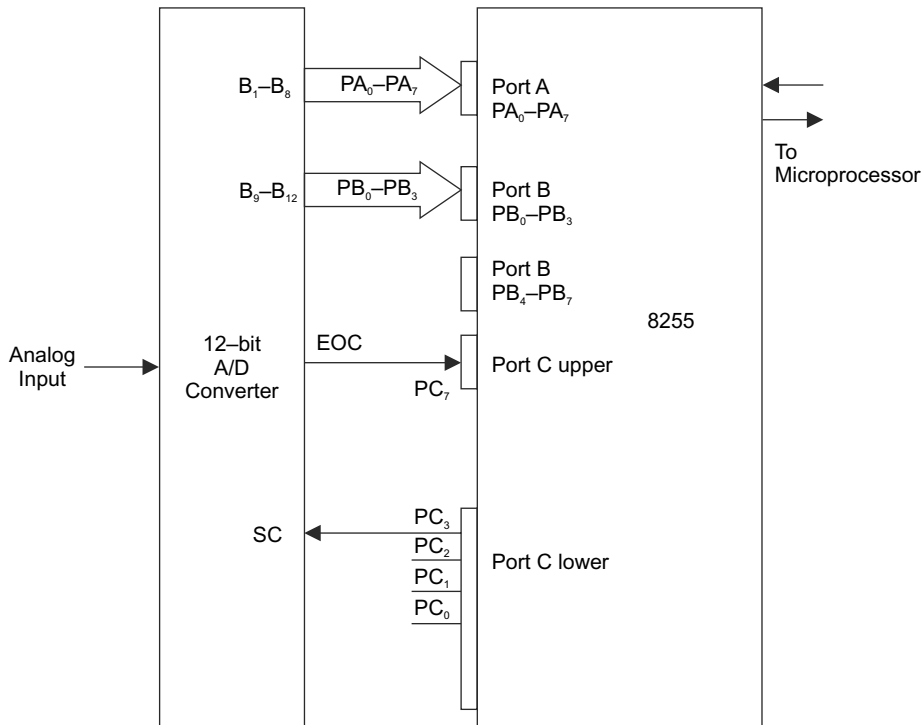


Fig. 9.52 12 bit ADC interfacing with microprocessor through 8255

PROGRAM 9.6

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	3E, 9A		MVI	A,9AH	Load control word (9AH) of 8255 in accumulator
8002	D3,03		OUT	03H	Write control word in control word register and initialize ports
8004	3E, 08		MVI	A,08H	Send start of conversion signal through PC ₃
8006	D3,02		OUT	02H	PC ₃ is high
8008	3E, 00		MVI	A,00H	As PC ₃ will be high for 1 or two clock pulse, make it 0
800A	D3,02		OUT	02H	PC ₃ becomes low
800C	DB, 02	LOOP	IN	02	Read end of conversion signal
800E	17		RAL		Rotate accumulator to check whether conversion is over or not.
800F	D2, 0C, 80		JNC	LOOP	If conversion is not completed, jump to LOOP
8012	DB, 00		IN	00	Read digital output of A/D converter from port A
8014	2F		CMA		Complement of ADC output (LSBs)
8015	21, 00, 81		LXI	H, 8100H	
8018	77		MOV	M,A	Store accumulator content (LSB of output) in 8100H location
8019	23		INX	H	
801A	DB, 01		IN	01	Read digital output of A/D converter from port B
801C	2F		CMA		Complement of ADC output (MSBs)
801D	77		MOV	M, A	Store accumulator content (MSB) in 8101H location
801E	76		HLT		Stop

9.7 DIGITAL-TO-ANALOG CONVERTER INTERFACING

The Digital-to-Analog Converter (DAC) has the ability to convert digital signals to analog signals. The digital-to-analog conversion is a process in which digital words are applied to the input of the DAC and an analog output signal is generated to represent the respective digital input. In this conversion process, an N -bit digital data can be mapped into a single analog output voltage. Therefore, the analog output of the DAC is a voltage that is some fraction of a reference voltage.

So,
$$V_{out} = K \times V_{Ref}$$

where, V_{out} is the analog voltage output, V_{Ref} is the reference voltage and K is the fraction.

Figure 9.53 shows the block diagram of a DAC converter. When a DAC has ' N '-bits digital inputs ($b_0, b_1, b_2, b_3 \dots b_{N-1}$) and a reference voltage, V_{Ref} , the voltage output, V_{out} can be expressed as

$$V_{Out} = K \times V_{Ref} \times \text{Digital inputs}$$

where, K is scaling factor

$$\text{Digital input} = 2^{N-1}b_{N-1} + 2^{N-2}b_{N-2} + 2^{N-3}b_{N-3} \dots \dots \dots b^{N-3} + 2^2b_2 + 2^1b_1 + 2^0b_0$$

N = number of bits, b_{N-1} = most significant bit, b_0 = least significant bit

The basic architecture of an ADC converter without S/H circuit is shown in Figure 9.54. It consists of binary switches, scaling network and output amplifier. The reference voltage, binary switches and scaling network convert digital inputs into voltage or current or charge signals. The output amplifier amplifies the output of the scaling the network to a desired measurable value.

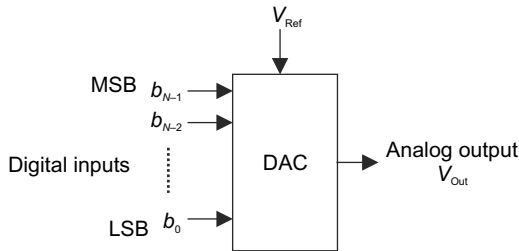


Fig. 9.53 Block diagram of a digital-to-analog converter

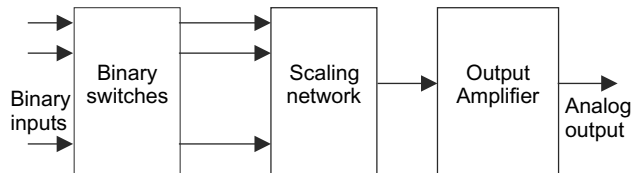


Fig. 9.54 Basic architecture of a digital to analog converter without S/H circuit

9.7.1 Binary Weighted or $R/2^N R$ DAC

Weighted binary DAC and $R-2R$ ladder are the two types of DAC. Each DAC converter input is a multi-bit digital signal, and generates an analog output signal equivalent to digital. Each bit of the signal has a different binary weight. The bit is multiplied by its weighting factor to give its contribution to the whole. The contribution from each bit is then summed, to give the analog equivalent.

The binary-weighted-input DAC circuit is a variation on the inverting summer op-amp circuit. The classic inverting summer circuit is an operational amplifier using negative feedback for controlled gain, with several voltage inputs and one voltage output. The output voltage is the inverted sum of all input voltages when all equal resistances are used in the circuit. If any of the input resistors were different, the input voltages would have different degrees of effect on the output, and the output voltage would not be a true sum. Assume the input resistor values are multiple powers of two: $R, 2R, 4R$ and $8R$, instead of R_1, R_2, R_3 and R_4 respectively

Figure 9.55 shows the circuit diagram of weighted 4-bit binary DAC The analog output voltage of a 4-bit weighted DAC can be expressed as follows.

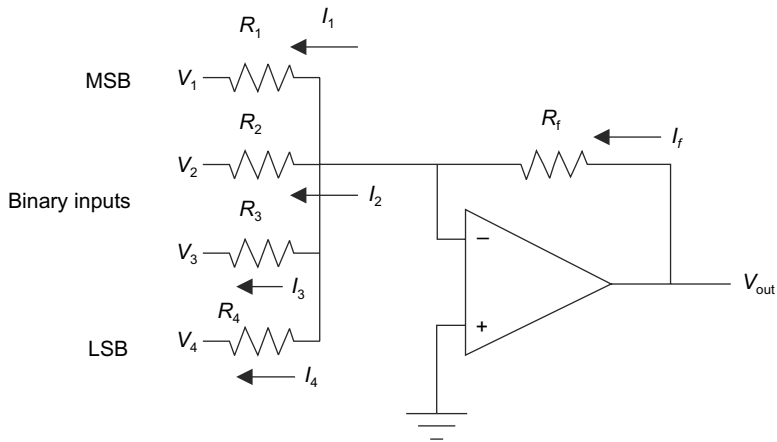


Fig. 9.55 Weighted 4-bit binary DAC

Four input currents I_1, I_2, I_3, I_4 and feedback current I_f are determined from the following expressions:

$$I_1 = V_1/R_1, I_2 = V_2/R_2, I_3 = V_3/R_3, I_4 = V_4/R_4 \text{ and } I_f = -V_{out}/R_f$$

The sum of the input currents is equal to feedback current I_f .

$$I_f = I_1 + I_2 + I_3 + I_4$$

After substituting all current values in the above expressions,

$$-\frac{V_{out}}{R_f} = \frac{V_1}{R_1} + \frac{V_2}{R_2} + \frac{V_3}{R_3} + \frac{V_4}{R_4}$$

or,

$$-V_{out} = \frac{R_f}{R_1} V_1 + \frac{R_f}{R_2} V_2 + \frac{R_f}{R_3} V_3 + \frac{R_f}{R_4} V_4$$

when, $\frac{R_f}{R_1} = 8, \frac{R_f}{R_2} = 4, \frac{R_f}{R_3} = 2, \frac{R_f}{R_4} = 1, V_1 = b_3, V_2 = b_2, V_3 = b_1$ and $V_4 = b_0$ the output voltage can be expressed as

$$-V_{out} = 8b_3 + 4b_2 + 2b_1 + b_0$$

or,

$$-V_{out} = 2^3 b_3 + 2^2 b_2 + 2^1 b_1 + 2^0 b_0$$

The weighting of bits b_0, b_1, b_2 and b_3 is 1, 2, 4 and 8 respectively. For this weighting, R_4 must be the largest resistor value, but the others resistances are half the value of the previous. If the 4-bit DAC has the following input resistances $R_1 = 1K, R_2 = 2K, R_3 = 4K, R_4 = 8K$ and the feedback resistor $R_f = 8K$, determine the output voltage for 1011 binary inputs. The analog output is ()V. To reduce the resolution of DAC, number of input bits will be increased.

The 'N' bit binary weighted DAC is given in Figure 9.56. The operational amplifier feedback resistance R_f can be selected to get the proper scale. Here, $R_f = K R/2$. The analog output voltage can be written as follows:

For N-bit inputs, current flowing through feed back resistance R_f is

$$I_f = -(I_{N-1} + I_{N-2} + I_{N-3} + \dots + I_2 + I_1 + I_0)$$

After substituting all current values in the above equation, we get

$$I_f = -V_{Ref} \left(\frac{b_{N-1}}{R} + \frac{b_{N-2}}{2R} + \frac{b_{N-3}}{4R} + \dots + \frac{b_2}{2^{N-3}R} + \frac{b_1}{2^{N-2}R} + \frac{b_0}{2^{N-1}R} \right)$$

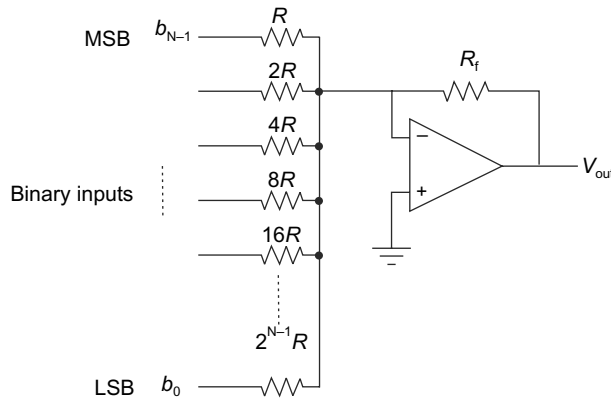


Fig. 9.56 N bit weighted DAC

$$I_f = -\frac{V_{Ref}}{2^{N-1}R} (2^{N-1} b_{N-1} + 2^{N-2} b_{N-2} + 2^{N-3} b_{N-3} + \dots + 2^2 b_2 + 2^1 b_1 + 2^0 b_0)$$

The output voltage V_{out} is

$$V_{out} = -I_f R_f$$

$$I_f R_f = -\frac{V_{Ref}}{2^{N-1}R} (2^{N-1} b_{N-1} + 2^{N-2} b_{N-2} + 2^{N-3} b_{N-3} + \dots + 2^2 b_2 + 2^1 b_1 + 2^0 b_0)$$

When an offset voltage applied in a DAC as shown in Figure 5.141. The output voltage can be expressed as

$$V_{out} = -\frac{R_f}{R_{off}} V_{off} - I_f R_f$$

$$V_{out} = \frac{R_f}{R_{off}} V_{off} - \frac{V_{Ref}}{2^{N-1}R} (2^{N-1} b_{N-1} + 2^{N-2} b_{N-2} + 2^{N-3} b_{N-3} + \dots + 2^2 b_2 + 2^1 b_1 + 2^0 b_0)$$

The resistance ratio of MSB and LSB is

$$\text{Resistance of MSB} / \text{Resistance of LSB} = \frac{R}{2^{N-1}R} = \frac{1}{2^{N-1}}$$

When $N = 8$, this ratio will be $1/128$. The larger component range creates problems for proper resistance matching and the binary-weighted resistance DAC is nonmonotonic.

9.7.2 R-2R Ladder Circuit

R-2R ladder circuit can eliminate the larger component spread in binary weighted DAC. The R-2R ladder circuit for converting digital to analog converter uses only resistances of R and 2R as shown in Figure 9.57. Mathematically analyzing this ladder network is a little bit difficult than doing so for a weighted resistance DAC. In weighted resistance DAC, each bit effect on output is very easily calculated. But in an R-2R ladder network, each binary input's effect on output can be determined using Thevenin's theorem for each binary input.

The effect of b_0, b_1, b_2 are determined as follows:

When $b_0 = 1, b_1 = 0, b_2 = 0$, the Thevenin's equivalent resistance and voltage can be determined as given below:

Looking from Section A-A', $2R \parallel 2R$ the equivalent resistance = R and equivalent voltage is $\frac{V_{Ref}}{2}$.

Looking from Section B-B', R is in series with R and sum of these resistance parallel with 2R the equivalent resistance = R and equivalent voltage is $\frac{V_{Ref}}{2}$.

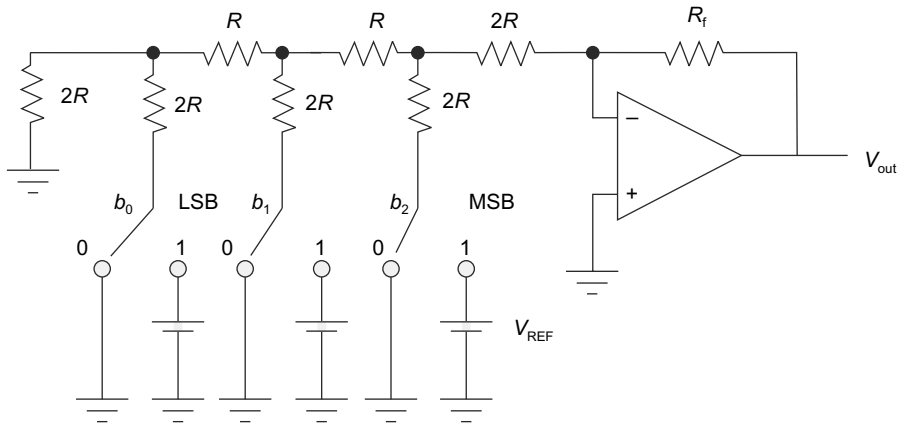


Fig. 9.57 R-2R ladder circuit of 3 bit DAC

Looking from Section C-C', R is series with R and sum of these resistances is parallel with 2R. Then equivalent resistance is equal to R and equivalent voltage is $\frac{V_{REF}}{2^3}$.

Similarly, the equivalent circuit for b_1 is $R_{equ} = 3R$ and $V_{in} = \frac{V_{REF}}{2^2}$ and the equivalent circuit for b_1 is $R_{equ} = 3R$ and $V_{in} = \frac{V_{REF}}{2}$. The complete equivalent circuit is shown in Figure 9.59.

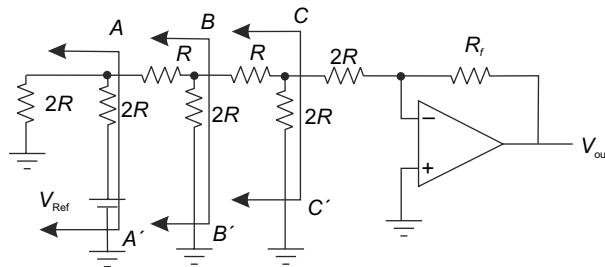
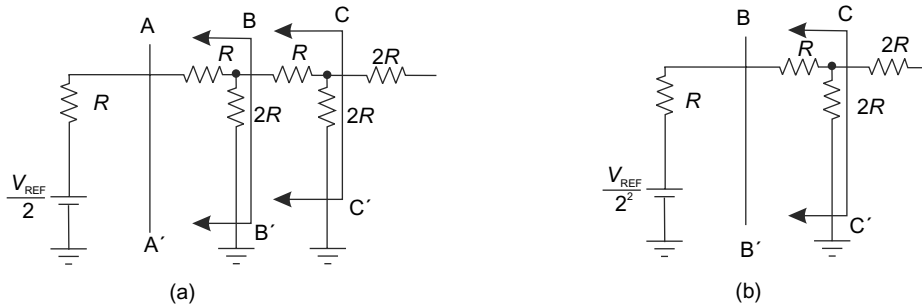


Fig. 9.58 R-2R ladder circuit of 3-bit DAC when digital input 001



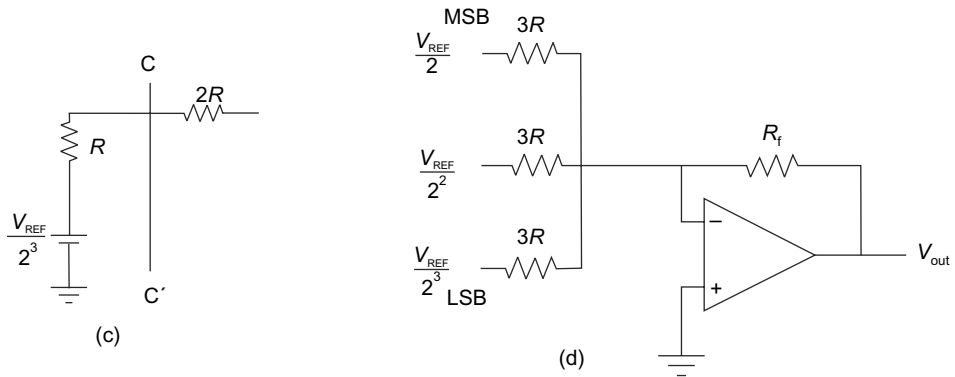


Fig. 9.59 Equivalent circuit of R-2R ladder DAC

The R – 2R ladder circuit works on the fact that the current is reduced by a factor of 2 for each digital input from LSB to MSB. The $I_0, I_1,$ and I_2 currents are as follows:

$$I_0 = \frac{V_{REF}}{2^3 \times 3R}, I_1 = \frac{V_{REF}}{2^2 \times 3R}, I_2 = \frac{V_{REF}}{2 \times 3R}$$

When all bits are 1, the currents flow into the operational amplifier and produce an output voltage:

$$\begin{aligned} V_{out} &= -(I_2 + I_1 + I_0) R_f \\ &= -\left(\frac{V_{REF}}{2 \times 3R} b_2 + \frac{V_{REF}}{2^2 \times 3R} b_1 + \frac{V_{REF}}{2^3 \times 3R} b_0\right) R_f \\ &= -\frac{V_{REF}}{2} \frac{R_f}{3R} (4b_2 + 2b_1 + 1b_0) \\ &= -K (4b_2 + 2b_1 + 1b_0) \end{aligned}$$

where,
$$K = \frac{V_{REF}}{2} \frac{R_f}{3R}$$

In this method of DAC, the larger component spread problem is eliminated and current flowing through the resistance cannot change due to switching and behaves as a constant voltage. This DAC is as fast as the binary weighted resistance DAC.

Figure 9.60 shows the N-bit R–2R ladder circuit and its equivalent circuit is depicted in Fig. 9.61. The output voltage of N bit R–2R circuit is

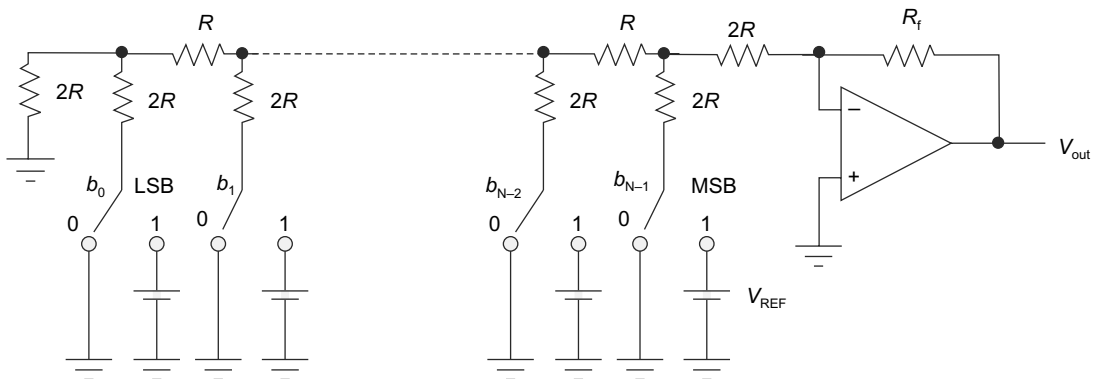


Fig. 9.60 N-bit R-2R ladder circuit

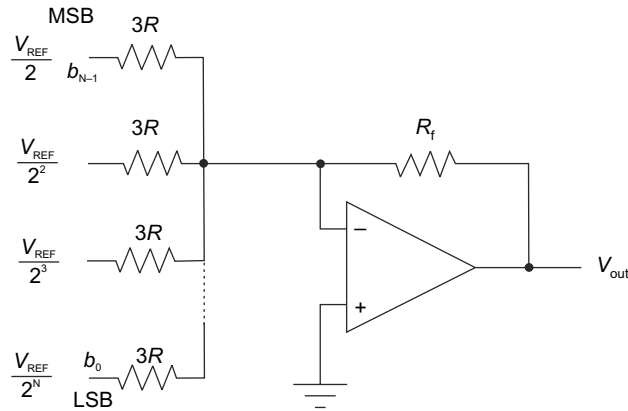


Fig. 9.61 Equivalent circuit of N bit R-2R ladder circuit

$$\begin{aligned}
 V_{out} &= (I_{N-1} + I_{N-2} + \dots + I_2 + I_1 + I_0) R_f \\
 &= \frac{V_{REF}}{2^N} \frac{R_f}{3R} (2^{N-1} b_{N-1} + 2^{N-2} b_{N-2} + 2^{N-3} b_{N-3} + \dots + 2^2 b_2 + 2^1 b_1 + 2^0 b_0)
 \end{aligned}$$

9.7.3 D/A Converter Specification

The performance of a D/A converter is measured based on the following parameters: resolution, accuracy, linearity, settling time and temperature sensitivity. The manufacturers generally specify these parameters in data sheets.

Resolution

The resolution of a D/A converter refers to the smallest change in the analog output voltage. It is equivalent to the value of the Least Significant Bit (LSB). For an N-bit D/A converter, the maximum number of steps is 2^{N-1} . When the reference voltage is V , the least significant bit value is given below:

$$\text{Resolution} = \frac{\text{Reference Voltage}}{\text{Number of Steps}} = \frac{V}{2^{N-1}}$$

For an 8-bit D/A converter with a full scale output of 10 V, the resolution is

$$\frac{10}{2^8 - 1} = \frac{10}{255} = 39.2 \text{ mV}$$

Accuracy

The output voltage of a D/A converter is different from an ideal case. Therefore, there is always some error. The accuracy is measured from the difference actual output voltage and voltage for ideal case. When the accuracy of a D/A converter is ± 0.25 per cent, the error of converter is $0.25 \times 12/100 = \text{mV}$ for full-scale voltage $V = 12 \text{ V}$.

Offset/Zero Scale Error

An input code of zero may be expected to give 0 V output. A small offset may be present and the transfer characteristic does not pass through the origin. The offset error of a D/A converter is depicted in Fig. 9.62.

Linearity

Figure 9.62 shows the input–output characteristics of a D/A converter. Zero offset and gain can develop the characteristic, which passes through the origin and full-scale points. But it is not sure that intermediate points will always lie on a straight line. A very small error in the weighting factor for

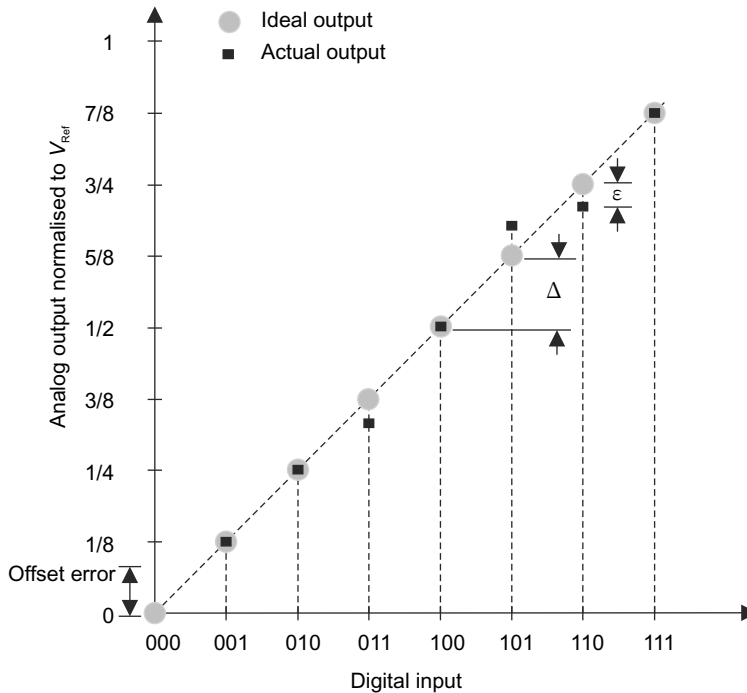


Fig. 9.62 Offset error of a D/A converter

a fraction LSB will cause non-linearity. Linearity can be expressed by deviation from the ideal line as a percentage, or a fraction of LSB. It is generally specified as $\pm \frac{1}{2}$ LSB or $|\epsilon| < \frac{1}{2} \Delta$.

Settling Time

This is usually expressed as the time taken to settle within half LSB. Generally, the settling time will be about 500 ns.

Temperature Sensitivity

The D/A converters are temperature sensitive. When the digital inputs are fixed, the analog output may be varied with temperature due to the temperature sensitivities of the reference voltages, the operational amplifier and converter circuit resistances, etc. Generally, temperature sensitivity of DACs is about ± 50 ppm/ $^{\circ}$ C in general-purpose converters.

9.7.4 Interfacing of DAC ICs

Most commonly DAC ICs are 8-bit DAC0800, 12-bit DAC80, 16-bit PCM54 and PCM55 etc. The DAC0800 ICs are monolithic 8-bit high-speed current output digital to analog converters with typical settling times of 100 ns. When used as a multiplying DAC, monotonic performance over a 40 to 1 reference current range is possible. These ICs have high compliance complementary current outputs to allow differential output voltages of 20Vp-p with simple resistance load as depicted in Fig. 9.63. The features of DAC0800 ICs are given below:

- ◆ Fast settling output current 100 ns
- ◆ Full-scale error ± 1 LSB
- ◆ Non-linearity over temperature $\pm 0.1\%$

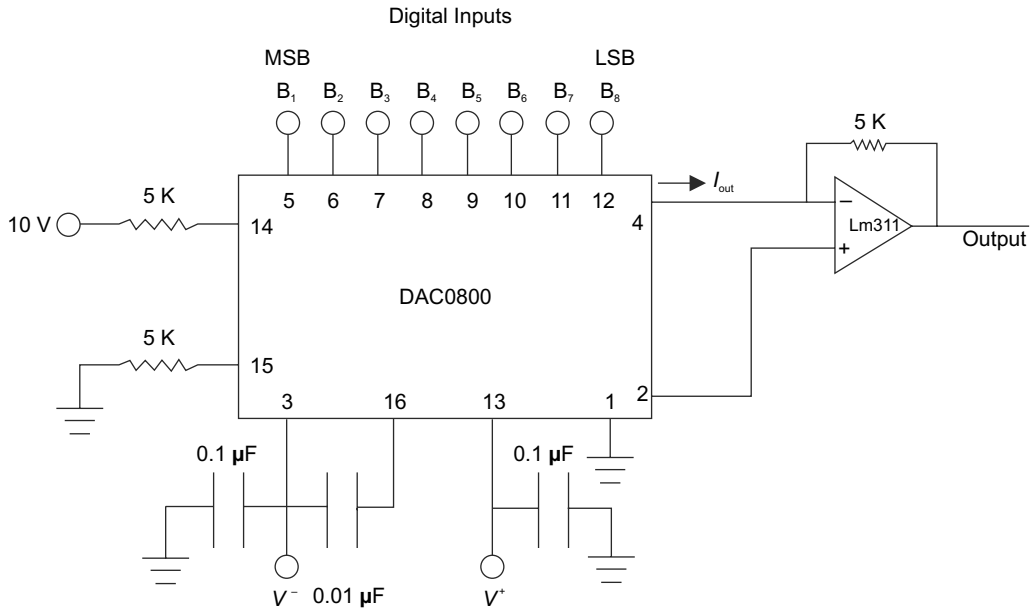


Fig. 9.63 8 bit DAC0800

- ◆ Full-scale current drift ± 100 ppm/C
- ◆ High output compliance $-10V$ to $+18V$
- ◆ Complementary current outputs
- ◆ Interface ability with TTL, CMOS, PMOS, etc.
- ◆ 2 quadrant wide range multiplying capability
- ◆ Power supply range $\pm 4.5V$ to $\pm 18V$
- ◆ Low power consumption 33 mW at $\pm 5V$
- ◆ Low cost

Figure 9.64 shows the circuit diagram for interfacing between DAC0800 and microprocessor using 8255. The output of Port A is directly interconnected with DAC. All ports of 8255 are operating in output mode and mode of operation is Mode 0. In this configuration, the control word is 80H. The programming of DAC is given below:

PROGRAM 9.7

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
9000	3E, 80		MVI	A,80H	Load control word of 8255 in accumulator
9002	D3,03		OUT	03H	Write control word in control word register
9004	3E,FF		MVI	A,FFH	Get FF for digital input to DAC
9006	D3, 00		OUT	00H	Send to port A for input into DAC
9008	76		HLT	Stop	

When the digital input is FFH, the analog output voltage is 10V. If the digital input is 00H the output will be 0V. Table 9.12 shows the analog output voltage with respect to digital input.

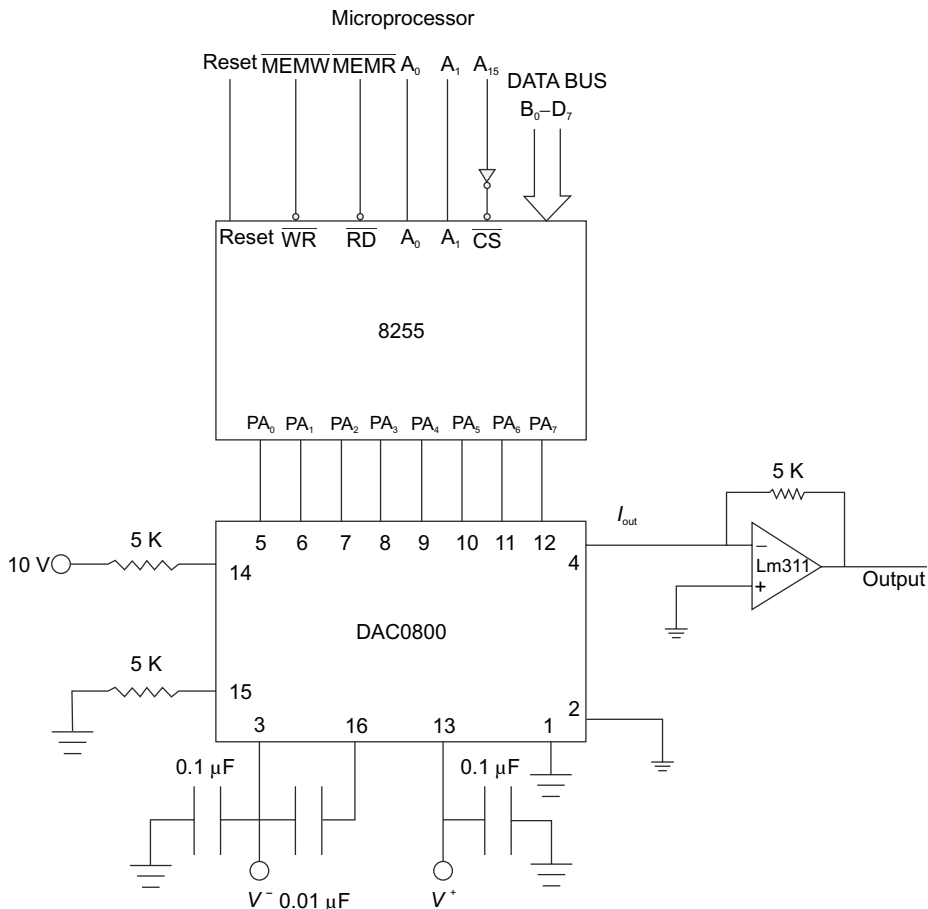


Fig. 9.64 Interfacing of DAC 0800

Table 9.12 Input and output relationship of DAC

Digital input of DAC	Analog output voltage
FFH	10V
80H	5V
00H	0V

The analog output voltage is $I_{out} \times R_L$

$$I_{out} = \frac{(\text{Digital Input})_{10}}{(256)_{10}} I_{ref}$$

The reference current = $I_{Ref} = \frac{V_{Ref}}{R_{Ref}}$

when V_{ref} is 10 V applied through 5 K ohms, reference current $I_{Ref} = \frac{V_{Ref}}{R_{Ref}} = \frac{10\text{ V}}{5\text{ K}} = 2\text{ mA}$.

The bipolar operation of DAC is shown in Figure 9.65. The Pin 2 of DAC is connected with the non-inverting terminal of operational amplifier. Pin number 4 is connected with the inverting terminal of operational amplifier. In this case, when FF input is applied to DAC, output is equal to 10 V. If input is 00H, output

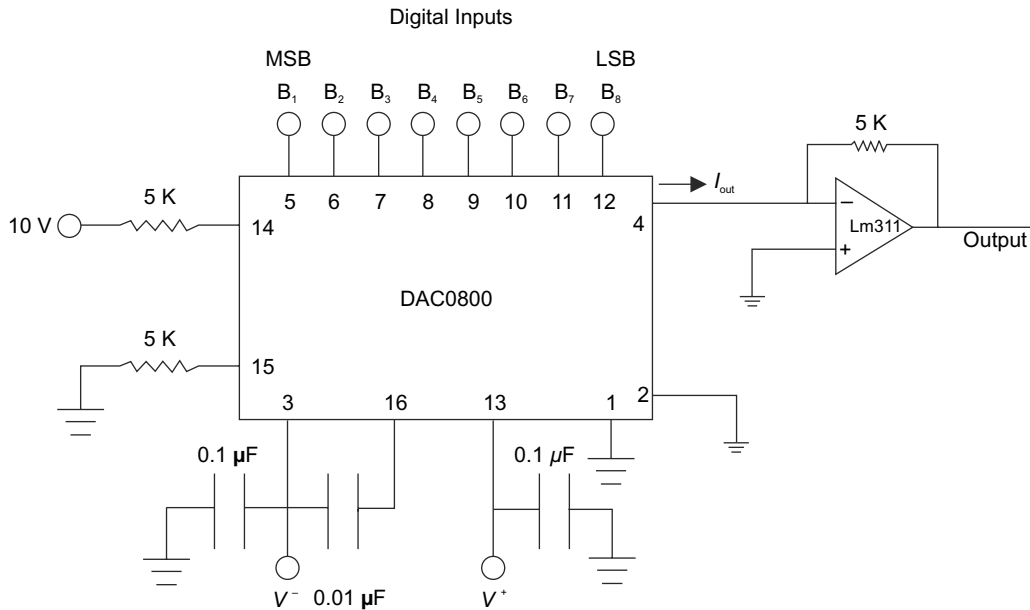


Fig. 9.65 Interfacing of DAC0800

will be -10 V . The input and output relationship of bipolar DAC is given in Table 9.13.

Table 9.13 Input and output relationship of biopolar DAC

Digital Input of DAC	Analog Output Voltage
FFH	10 V
80H	0 V
00H	-10 V

9.8 BUS INTERFACE

A microcomputer consists of a set of components such as CPU, memory, I/O device and these components communicate with each other to perform a specified task. The collection of paths which connect the various devices or modules is called the interconnection structure. The design of the interconnection structure depends on the exchange of data between different devices or modules as shown in Figure 9.66. There are different types of data transfer such as

- ◆ Memory devices to CPU
- ◆ CPU to memory devices
- ◆ I/O devices to CPU
- ◆ CPU to I/O devices
- ◆ I/O devices to or from memory

9.8.1 System Bus

A bus is a communication pathway which can connect two or more devices (CPU, memory and I/O). Actually, the bus is a set of circuits that runs throughout the board and connects all the expansion slots, memory, I/O

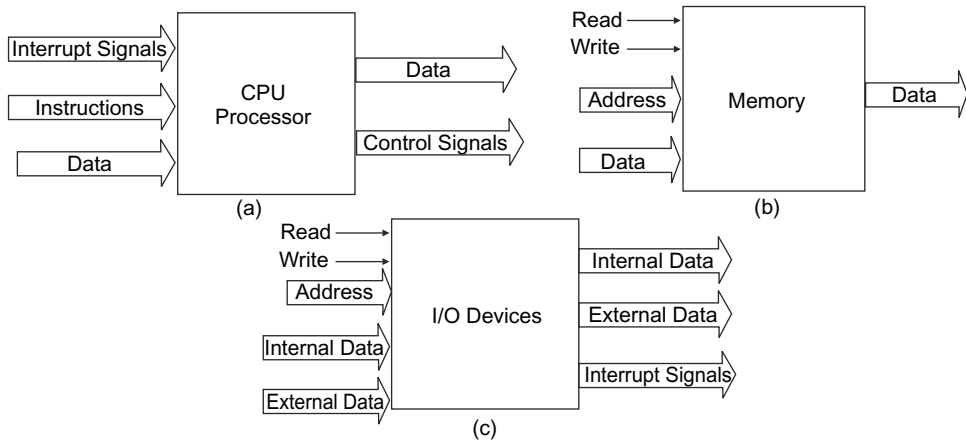


Fig. 9.66 Input and output signals of (a) CPU/Processor (b) Memory (c) I/O devices

devices and CPU together. The characteristic of a bus is that it is a shared transmission medium. A bus consists of multiple pathways or lines. Each line is capable to transmit digital signal representing a binary digit, either 1 or 0. A sequence of bits can be transmitted across a single line. Simultaneously, several lines can be used to transmit bits in parallel. A bus that connects major components such as the CPU, memory, and I/O is called system bus. The most commonly used microcomputer interconnection structures is shown in Fig. 9.67 and 9.68.

A system bus consists of 50–100 lines. Each line is assigned a particular function. The bus lines are classified into 3 groups such as data bus, address bus and control bus.

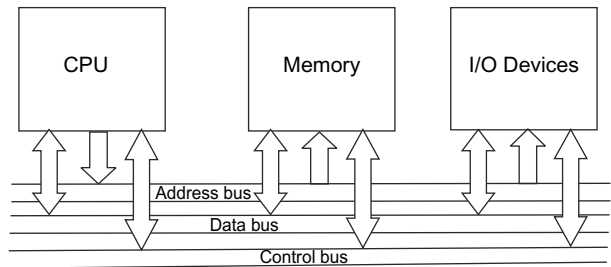


Fig. 9.67 Bus structure of a microcomputer

- ◆ The data bus provides a path for moving data between the system modules or devices. These lines are also called *data lines*. The data bus typically consists of 8, 16 or 32 separate lines, actually the numbers of lines being transferred to as the *width* of the data bus. Each line is capable of carrying only 1 bit at a time and the number of lines determines how many bits can be transferred at a time to increase overall system performance.
- ◆ The address bus is used to locate the source or destination of the data on the data bus. The width of the address bus determines the maximum possible memory capacity of the system. The address bus is also known as *address lines*.

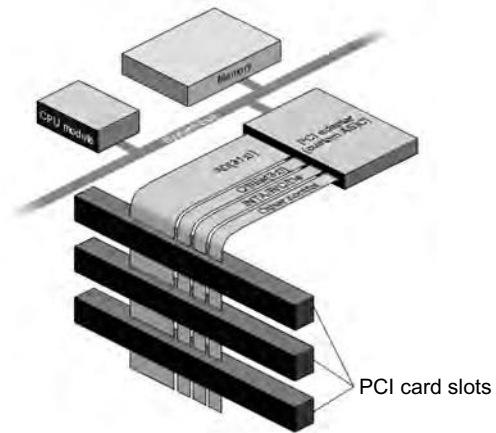


Fig. 9.68 Bus structure of a microcomputer

- ◆ The control bus is used to control the access to and the use of the data and address lines. The most commonly used control lines are memory read, memory write, I/O read, I/O write, clock, reset, bus request, bus grant, interrupt request, interrupt acknowledge (ACK) and transfer ACK.

The operation of the bus is to transfer data from one device to other device via the buses. In a physical bus architecture, the system bus is a number of parallel electrical conductors. The conductors are metal lines etched in a card or printed circuit board. The bus extends across all of the components to connect with the bus lines.

Generally, the motherboard of a microcomputer is the physical arrangement of a computer, which consists of all the computer's components such as CPU/microprocessor, co-processors, memory, BIOS, expansion slots and interconnecting circuitry. Some additional components can be added to a motherboard through its expansion slots. The electronic interface between the motherboard and the smaller cards in the expansion slots is also called the bus. The personal computer consists of several types of buses on the motherboard as shown in Fig. 9.69, and the block diagram representation of a typical motherboard is depicted in Fig. 9.70. Most commonly used buses are

- ◆ CPU bus or system bus
- ◆ Cache bus
- ◆ Memory bus
- ◆ I/O or Expansion Bus:
 - ISA (Industry Standard Architecture)
 - EISA (Extended ISA)
 - MCA (Micro Channel Architecture)
 - PCI Bus (Peripheral Connection Interface)
 - VL Bus (VESA Local Bus)
- ◆ External Buses which are used to connect devices through external cables

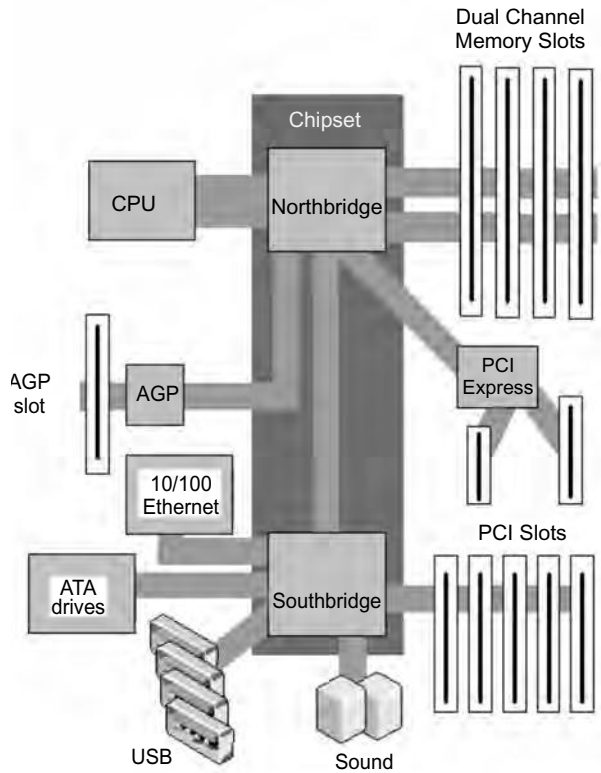


Fig. 9.69 Block diagram of a motherboard

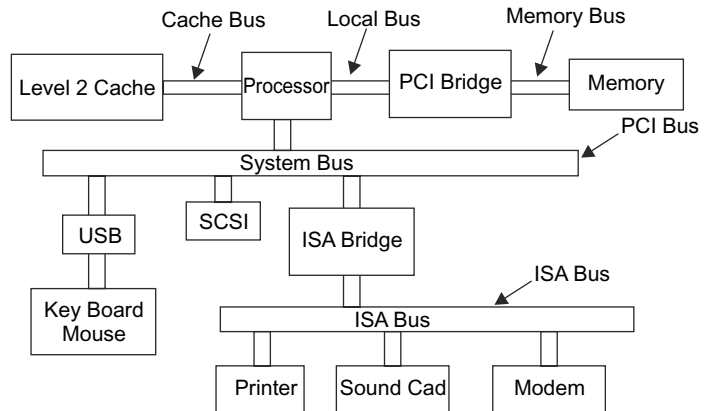


Fig. 9.70 System buses of a motherboard

- SCSI Bus
- PC Card Bus
- USB Bus

9.8.2 BUS Architecture

In multiple bus architecture, two or more devices are attached to the system bus, and propagation delays affect the performance of computer system. But the aggregate data transfer of the system bus is limited, and it is a bottleneck for data transfer as graphics and video controller. Therefore, buses are increased and they are interconnected in a certain configuration. The multiple bus architectures are classified as traditional bus architecture and high performance bus architecture. The traditional bus architecture has *local bus* between CPU and cache, *system bus* between main memory and cache, and *expansion bus* between I/O modules and main memory as shown in Figure 9.71. Figure 9.71(b) shows the high-performance bus architecture which consists of local bus between CPU and cache/bridge, system bus between cache/bridge and memory, high speed bus between high speed I/O devices and cache, and *expansion bus* between low speed I/O devices and expansion interface.

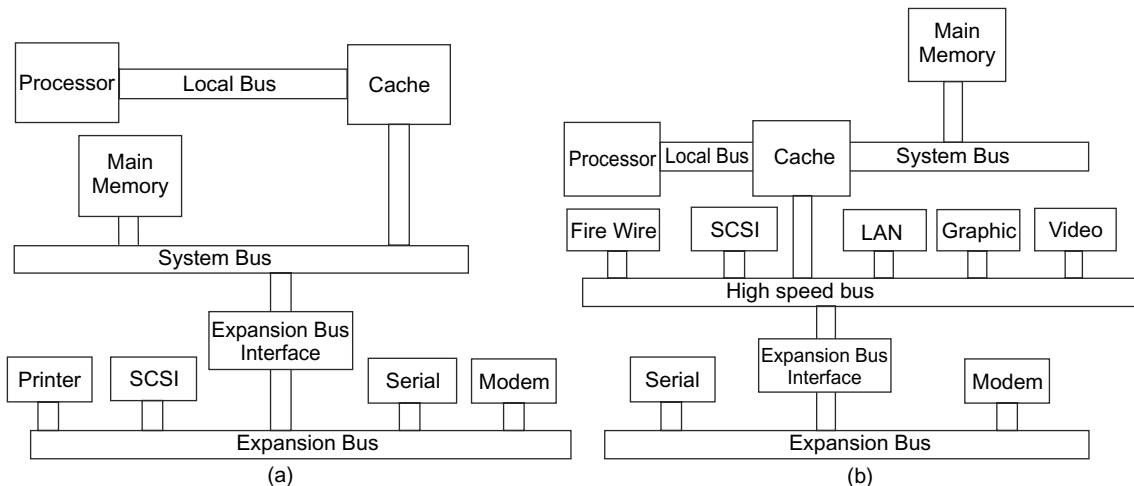


Fig. 9.71 (a) Traditional bus architecture and (b) High performance bus architecture

CPU or Processor Bus The CPU bus is the highest-level bus through which the chipset uses to send information to the processor and receive information from the processor.

Cache Bus In high-level bus architecture, the Pentium processors use a dedicated bus for accessing the system cache. This is also known as *backside bus*. In motherboards, the cache is connected to the standard memory through the cache bus.

Memory Bus The memory bus is a second-level system bus which connects the memory subsystem to the processor. In some systems, the processor bus is used as memory bus.

Local I/O Bus This bus is a high-speed input/output bus used for connecting peripherals to the memory, I/O devices and processor. For example, audio cards, video cards, disk storage devices, high-speed networks interfaces generally use the local I/O bus of this sort. The most commonly used local I/O buses are the VESA Local Bus (VLB) and the Peripheral Component Interconnect Bus (PCI).

Standard I/O Bus

The standard I/O buses are used for slower peripherals such as mice, modems, regular sound cards and low-speed networking. Nowadays, in all modern PCs, this is used as the Industry Standard Architecture (ISA) bus.

Expansion Bus

All the above buses are located on the motherboard of a computer. In the motherboard of all personal computers, expansion slots are available to add cards or boards for more memory, graphics capabilities, and support for special devices. The boards inserted into the expansion slots are called *expansion boards*, *expansion cards*, and *add-on-cards*. On each slot, we can insert expansion boards such as soundcards, graphics cards, TV cards, etc., to create additional facility to the personal computer. Actually, these boards can communicate with the other hardware devices in the system. There are different ‘slots’ on the motherboard as shown in Fig. 9.72 and the names of the slots are as follows:

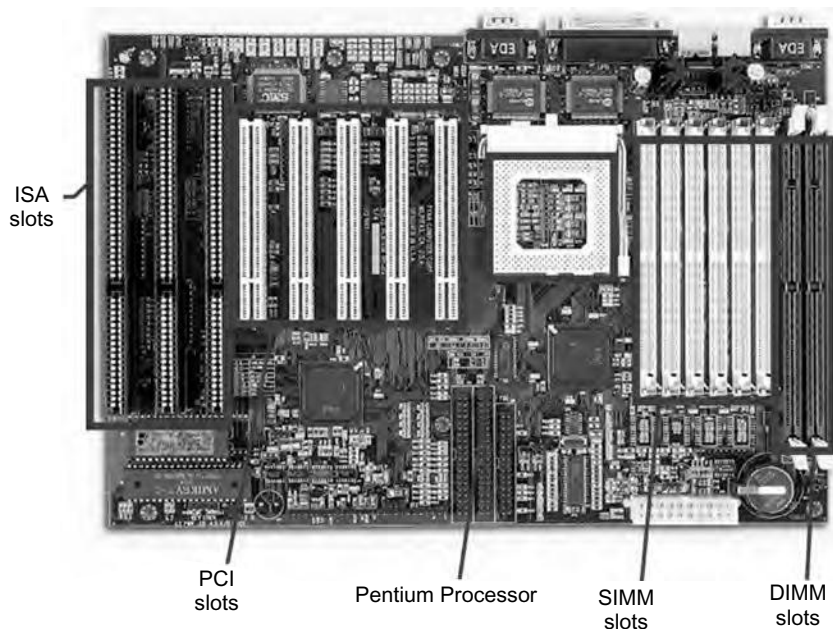


Fig. 9.72 Different expansion slots in a mother board

- ◆ ISA — Industry Standard Architecture
- ◆ EISA — Extended ISA
- ◆ MCA — Micro-Channel Architecture
- ◆ VESA — Video Electronics Standards Association
- ◆ PCI — Personal Component Interconnect
- ◆ AGP — Accelerated Graphics Port
- ◆ SIMM — Single Inline Memory Module
- ◆ DIMM — Dual Inline Memory Module
- ◆ PCMCIA — Personal Computer Memory Card International Association

In this section, ISA, EISA, MCA, VESA, PCI, and AGP are discussed elaborately.

9.8.3 ISA (Industry Standard Architecture)

The ISA stands for Industry Standard Architecture and it is pronounced as ‘eye-es-ch’. This bus architecture was developed by IBM in 1979. Actually, this bus was introduced in the IBM PC XT in 1981 as an 8 bit

expansion slot and this bus operates at 8.3 MHz with a data rate of 7.9 Mbytes/s. After that 16-bit ISA bus introduced in the IBM PC AT machines in 1984 and runs at 15.9 MHz with data rate of 15.9 Mbytes/s. This bus is called AT bus and presently all ISA slots are 16-bits. The ISA allows 16 bits of information at a time to flow between the motherboard circuit and an expansion slot card and its associated devices.

9.8.4 EISA (Extended Industry Standard Architecture)

EISA is a standard bus architecture that extends the ISA standard to a 32-bit interface. It was developed by IBM competitors such as HP, AST, Compaq, and Epson, etc., in 1987 and it was used as an alternative of the Micro Channel Architecture (MCA). EISA is hardware compatible with ISA. EISA data transfer can run at 33 Megabytes per second.

9.8.5 MCA (Micro Channel Architecture)

The Micro Channel Architecture was introduced by IBM in 1987 and used in PS/2 desktop computers. The MCA can be used as an interface between computers and their expansion cards and associated devices.

The pin connections in MCA are smaller than other bus interfaces. Therefore, MCA does not support other bus architectures.

9.8.6 VESA (Video Electronics Standards Association)

VESA stands for video electronics standards association. This bus is also called VL-Bus or VESA Local Bus (VLB). VESA Local Bus was developed in 1990 and supports 32-bit data flow at speeds of up to 40 MHz. VESA VL bus is a standard interface between a computer and its expansion slot. This bus provides faster data flow between the peripheral devices (video, disk, network) controlled by the expansion cards and microprocessor. This bus is a '*local bus*' and it provides a physical path on which data flows at the speed of the microprocessor. The VESA local bus architecture was very popular on 80486 based computer systems in 1993 and 1994. The advantages of VESA bus are as follows:

- ◆ Faster processing
- ◆ 32-bit data-transfer capability
- ◆ Direct access to the processor bus, which is local to the CPU
- ◆ Direct access to system memory at the speed of the processor
- ◆ Faster access
- ◆ Different physical slot that prevents plugging a slower card into a fast slot
- ◆ 128 MBps to 132 MBps maximum throughput

The limitations of VESA bus are

- ◆ Available only for 80486 processors
- ◆ Maximum speed of the VESA specification is 66 MHz, though it's speed is limited to 33 MHz
- ◆ Poor implementation of bus mastering
- ◆ This bus do not support plug-and-play
- ◆ VESA bus cannot be used for the speed of the Pentium
- ◆ Limited to a maximum of three cards depending on system resources

9.8.7 PCI (Peripheral Component Interconnect)

PCI (Peripheral Component Interconnect) bus was developed by Intel in 1993, ISA has been replaced by the PCI local bus architecture. These are the smaller and white-colored slots on the motherboard. PCI is a 64-bit

bus, but it is usually implemented as a 32-bit bus. This bus runs at clock speeds of 33 MHz or 66 MHz. At 32 bits and 33 MHz, the data transfer rate is 133 MBps.

PCI is an interconnection system between CPU or microprocessor and peripheral devices in which expansion slots are spaced closely for high-speed operation. Using PCI, any computer can support both new PCI cards and ISA expansion cards. PCI 2.0 is a local bus and is designed to be independent of the microprocessor. Presently, PCI is installed on all new desktop computers based on Pentium and Power PC processors. The PCI transmits 32 bits at a time in a 124-pin connection and 64 bits in a 188-pin connection in an expanded implementation. This bus uses all active paths to transmit both address and data signals, sending the address on one clock cycle and data on the next cycle. This bus provides better system performance for high-speed I/O subsystems such as graphic display adapters, network interface controllers, and disk controllers. Figure 9.73 shows the PIC bus for a single processor based computer system and the PIC bus for multiprocessor based server system is depicted in Fig. 9.74.

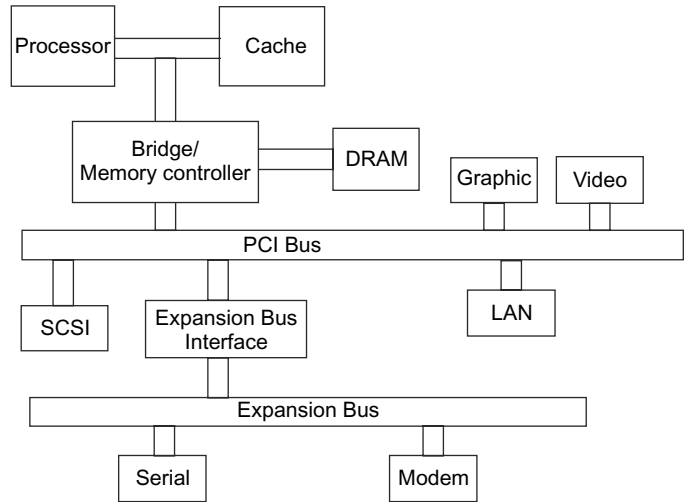


Fig. 9.73 Single processor system

9.8.8 AGP (Accelerated Graphics Port)

The Accelerated Graphics Port (AGP) was developed by Intel in 1998. Actually, AGP is based on PCI, but it is designed specially to display 3D graphics quickly on ordinary personal computers. AGP introduces a dedicated point-to-point channel, so that the graphics controller can directly access main memory. Usually, the AGP channel is 32 bits wide and runs at 66 MHz with a bandwidth of 266 MBps. The AGP allows 3D textures to be stored in the main memory rather than video memory. The AGP interface uses main storage (RAM) for refreshing the monitor image and to support the *texture mapping*, *z-buffering*, and *alpha blending* required for 3D image display. The Pentium processors can work with the AGP chipset for 3-dimensional applications.

9.8.9 Universal Serial Bus (USB)

In the mid 1990's, the Universal Serial Bus was invented by a group of companies such as IBM, Intel, Microsoft, Compaq, etc., and it was introduced as a high speed replacement of RS-232 serial port. It has very high bandwidth and operates at 1.5 MBps, 12 MBps and 480 MBps. This device can be used as daisy-chained as

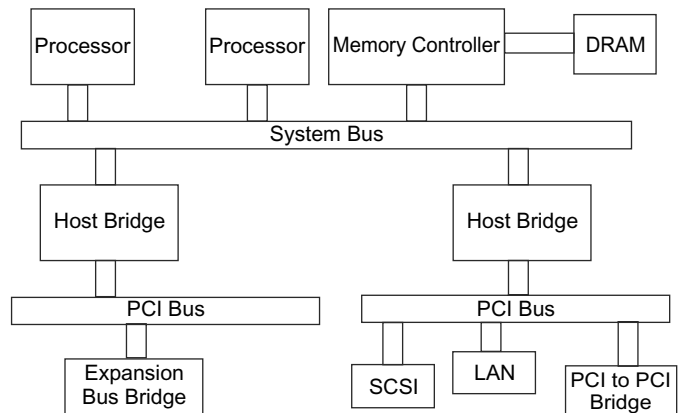


Fig. 9.74 Multiprocessor system

shown in Figure 9.75. The USB can also be used as multipoint bus and hubs provide multiple connection points for I/O devices as depicted in Figure 9.76. It can support about 127 devices. The commonly used USB devices are keyboard, monitors, digital cameras, mobile phones, digital video recorders, etc. The USB is available in different standards such as USB 1.1, USB 2.0 and USB on the go (OTG). This is low cost and hot plug-in play type. The hot plug-in play means the ability of USB to connect a device to the computer while a computer is in operation.

There are two basic requirements for USB implementation in microcomputers. The first is the presence of USB hubs to support USB ports on the microcomputer, and the second is the software support required from the operating system to operate the USB properly. Presently, all motherboards have built-in support for USB. The operating systems such as Window 98, Windows 95, OSR 2, Windows XP and other later versions can support USB. Whenever any device is connected into USB, the operating system would recognize the device and also configure the device simultaneously. The advantages of USB over the RS 232 are higher operating speed, ability to daisy chain connected different devices and support for hot plug and play.

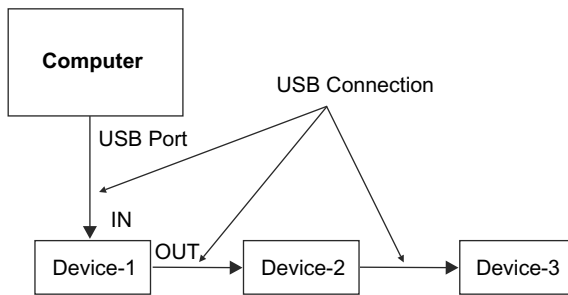


Fig. 9.75 Daisy chain of USB devices

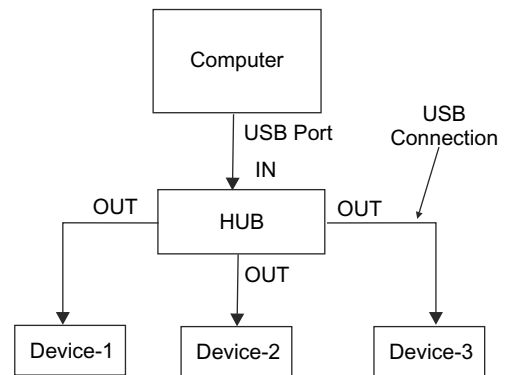


Fig. 9.76 USB devices using a HUB

9.8.10 Parallel Printer Interface

The parallel interface implies a Centronics-compatible printer interface. This interface was developed by the printer-manufacturing company Centronics and introduced on IBM PC in 1981 as line printer (LPT) port. The improved version of LPT was developed by Intel, Xircon and Xenith in 1994 and was known as Enhanced Parallel Port (EPP). The data-flow rate through a parallel interface is about 150 kbytes/s for LPT to 1.5 Mbytes/s for EPP. The typical parallel printer interface with a cable is shown in Figure 9.77. The pin description of DB 25 and Centronics are given in Table 9.14.

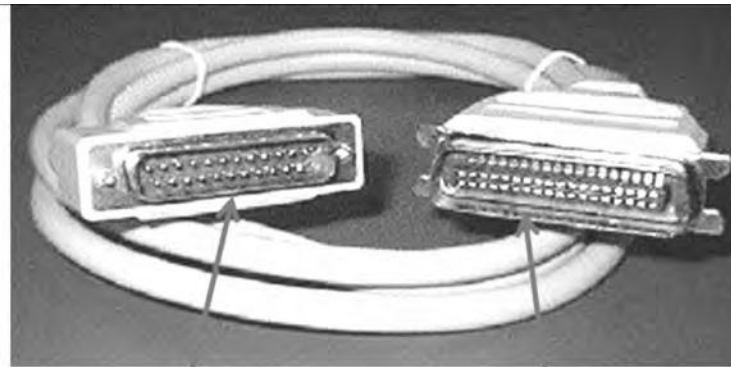
Table 9.14 Pin description of DB25P male and 36-pins Centronics male

DB25 Pins	Centronics 36 Pins	Direction	Signal	Function
1	1	Out	/Strobe	Low pulse ($>0.5 \mu\text{s}$) to send
2	2	Out	Data 0	LSB
3	3	Out	Data 1	
4	4	Out	Data 2	
5	5	Out	Data 3	
6	6	Out	Data 4	

(Contd.)

(Contd.)

7	7	Out	Data 5	
8	8	Out	Data 6	
9	9	Out	Data 7	MSB
10	10	In	/ACK	Low pulse acknowledge
11	11	In	Busy	High for busy/offline/error
12	12	In	Paper End	High for out of paper
13	13	In	Select In	High for printer selected
14	14	Out	/Auto Fd	Low to auto feed one line
15	32	In	/Error	Low for error
16	31	Out	/Init	Low pulse(>50s)to initialize
17	36	Out	/Select	Low to select printer
18-25	16,1719-30, 33		Ground	



DB25P male connects to PC

36-pins Centronics male connects to printer

Fig. 9.77 Parallel printer interface cable

9.8.11 RS-232C

The serial communication is the simplest form of communication between two devices. Serial data transmission is used for digital communication between computers and computers, computers and peripheral devices (modems, printers, etc.) and sensors and computers for data acquisition. In all personal computers, serial interface is implemented using a communication port or COM port. There are four COM ports such as COM-1, COM-2, COM-3, and COM-4. The COM ports conform to the RS-232C interface standard. The RS-232C was developed by the Electronics Industry Association (EIA) in the 1960s. Actually, RS-232C was intended as an electrical specification to connect computer terminals to modems through DTE and DCE as shown in Figure 9.78. DTE means Data Terminal Equipment, and DCE



Fig. 9.78 Data flow between computer terminals and modems

means Data Terminal Equipment, and DCE

stands for Data Communication Equipment. The modem is also a data set. In Figure 9.78, the data communication is the digital data exchange between a mainframe computer and a remote computer. These computers are linked by telephone lines and modems are used at each end for signal translation.

The maximum data flow rate of RS-232C is 20 kbits/s with a maximum cable length of 15 metres. The serial data transmission through RS-232C is possible in two different modes, namely, asynchronous and synchronous. In the asynchronous mode, the transmitting and receiving devices are not synchronized and the clock signal does not get transmitted along with the data. During synchronous mode, the transmitting and receiving devices are synchronized and the clock signal is transmitted along with the data. But most of the RS-232C operates in asynchronous mode. Data is transmitted on the Transmit Data (TD) line in packets (5 bits, 6 bits, 7 bits or 8 bits). Each packet consists of a start bit (0) at the beginning and a stop bit (1) at the end. A parity bit is also inserted at the end of the packet, but before the stop bit. The parity bit states either even parity or odd parity with the data bits in the packet.

The line voltages of RS-232C have two states such as ON state and OFF state. The ON state is also known as *marking state* which is identified by a negative voltage. The OFF state is called the *space state*. The positive voltage is used to represent space state. In RS 232C, 1 is called a *mark* and 0 is called a *space*. The voltage limits of mark state and space state are given in Table 9.15. Figure 9.79 shows the plot of the asynchronous RS-232C transmission of ASCII character 'A' with start bit, parity bit and two stop bits.

Table 9.15 RS232 Voltage levels

State	Transmitter voltage level	Receiver voltage level
Space state (Logical 0)	+5 V to +15 V	+3 V to +25 V
Mark state (Logical 1)	-5 V to -15 V	-3 V to -25 V
Undefined	—	-3 V to +3 V

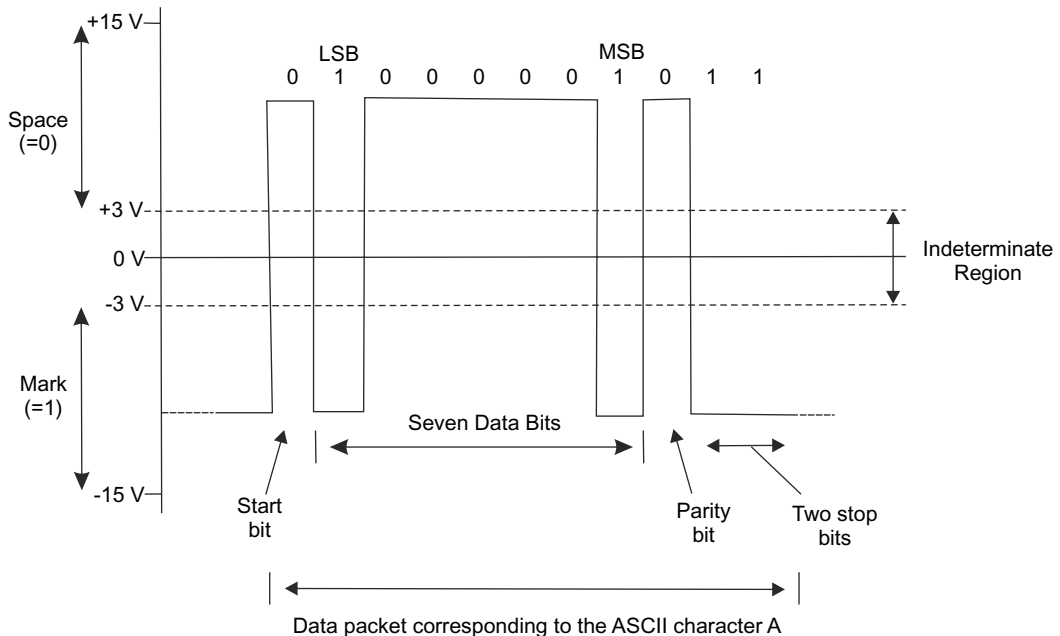


Fig. 9.79 Asynchronous RS-232C transmission of ASCII character 'A'

The original standard RS-232C is a 25-pin connector, but presently 9-pin RS-232C connectors are most commonly used. These connectors are available in male and female sockets. The DB25P and DB9P are RS-232C 25 pin and 9-pin male connectors respectively as depicted in Figure 9.80. The DB25S and DB9S are RS-232C 25 pin and 9-pin female connectors respectively as illustrated in Figure 9.81. The detail description of RS-232C pins are given in Table 9.16 and Fig. 9.82. The function of signals as follows:

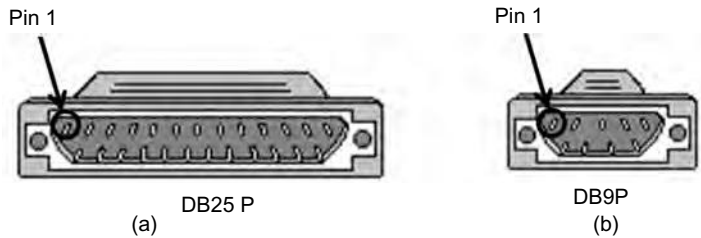


Fig. 9.80 RS-232 (a) 25-pin male connectors (b) 9-pin male connectors

- ✓ **TD** This signal is used to transmit data
- ✓ **RD** This signal is used to receive data
- ✓ **DSR** The DSR stands for data set ready. This signal indicates whether the DCE (modem) is powered ON.
- ✓ **DTR** This is a data terminal ready signal which indicates whether DTR is powered on and turning off DTR causes modem to hang up the line.
- ✓ **RI** The RI means Ring Indicator. This signal is activated by the modem when it detects incoming phone call on the telephone line.
- ✓ **DCD** The Data Carrier Detect (DCD) is used by the modem to signal the transmitter that the communication link is usable. This signal is ON when two modems have negotiated successfully and the carrier signal is established on the phone line.
- ✓ **RTS** RST stands for Request To Send. This signal is activated when DTE wants to send data. This is used to turn on and off the modem's carrier signal in multi-point lines, but usually it is constantly ON in point-to-point lines.
- ✓ **CTS** The CRS stands for Clear To Send. This signal is used by the receiver to inform the transmitter that DCE is ready to receive data.
- ✓ **SG** SG is Signal Ground.

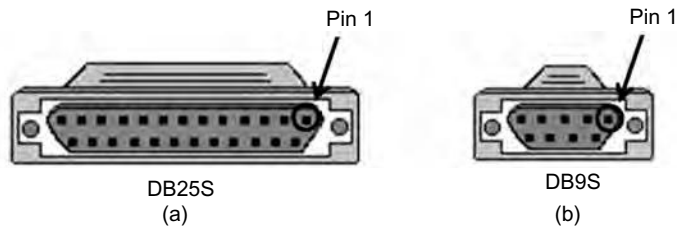


Fig. 9.81 RS-232 (a) 25-pin female connectors (b) 9-pin female connectors

Table 9.16 Pin description of RS-232C (a) 9 Pin (b) 25 pin

DB25P	DB9P	Signal	Signal Name	Direction
1	-	CD	Chassis ground	—
2	2	TD	Transmit data	DCE←DTE
3	3	RD	Receive data	DTE←DCE
4	7	RTS	Request to send	DCE←DTE
5	8	CTS	Clear to send	DTE←DCE

(Contd.)

(Contd.)

6	6	DSR	Data set ready	DTE←DCE
7	5	SG	Signal ground	—
8	1	DCD	Data carrier detect	DTE←DCE
20	4	DTR	Data terminal ready	DCE→DTE
22	9	RI	Ring indicator	DTE→DCE

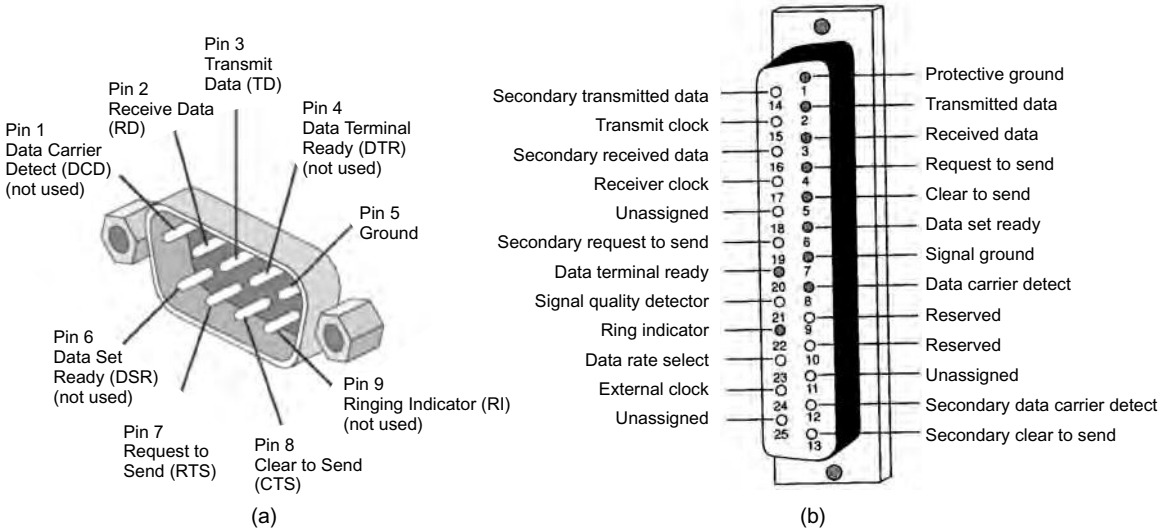


Fig. 9.82 Pin description of RS-232C (a)DB9P (b) DB25P

RS-232C with TTL Interfacing

Figure 9.83 shows the interfacing RS 232C with TTL via special line drivers and receivers. The line driver MC 1488 can accept TTL level inputs and generates RS-232C output levels. The MC 1488 converts logic level '1' into about 9 V and logic level '0' into +9 V. The MC 1489 converts RS-232C level inputs to TTL levels. The problem of RS-232C is that drivers and receivers are single ended. The input and output signals have a common ground.

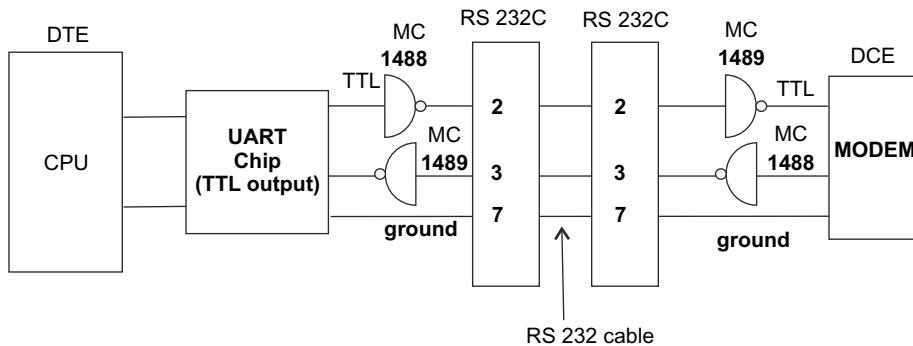


Fig. 9.83 RS 232C interfacing with TTL

To interface a computer (CPU) and a peripheral device (modem), the minimum three lines such as 2, 3 and 7 are required as shown in Fig. 9.84. In the DTE to DCE communication, transmits on pin 2 and receives on pin 3 are as shown in Fig. 9.84(a). In DTE to DTE communication, the terminal transmits on Pin2 and receives on Pin 3 as shown in Fig. 9.84(b).

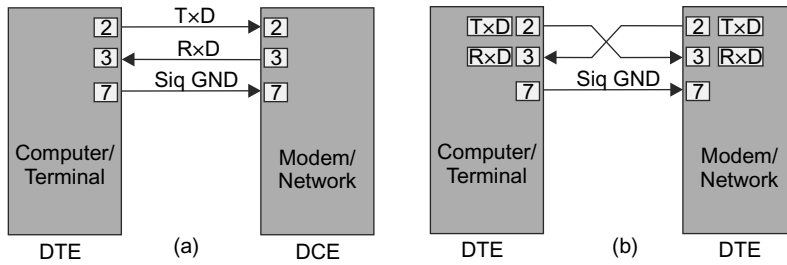


Fig. 9.84 RS-232C connections (a) DTE to DCE (b) DTE to DTE

For high-speed data transmission, the standards RS-422A and RS-423A are used. Differential amplifiers are used in these standards to reduce noise levels and can transmit data at higher speed for long-distance cable. The RS 422A has a maximum speed of 10M baud for 40-foot distance and 10 kbaud for 1000-foot distance.

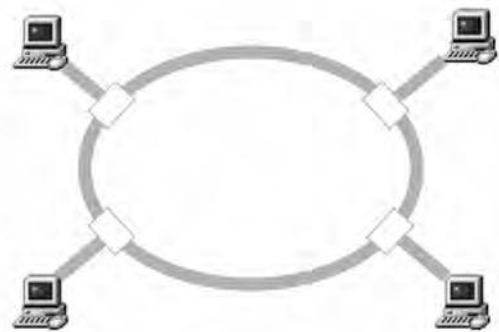
The speed of RS-423A is limited up to 100 kbaud for 130-foot distance and 10 kbaud for 300-foot distance.

9.8.12 IEEE-488 Bus

The General Purpose Interface Bus (GPIB) is known as Hewlett-Packard interface bus(HPIB) or IEEE488 bus. This was developed by Hewlett-Packard to interface testing equipments with a computer as shown in Fig. 9.85(a). These buses are used in a computer network as shown in Fig. 9.85(b). Usually, three types of standard devices such as *listener*, *talker* and *controller* can be connected on the GPIB.



(a)



(b)

Fig. 9.85 (a) IEEE-488 Bus (b) Computer network

The *listener* can receive data from other instruments or from the controller. For example, printers, display devices, programmable power supplies and programmable signal generators are used as listeners. A *talker* can send data to the instruments. The examples of talkers are tape readers, digital multimeters, frequency counters and measuring equipments. The third device, *controller*, determines who talks and who listens on the bus.

The IEEE-488 bus consists of a 24-wire cable with a connector such as that shown in Fig. 9.85(a). The IEEE-488 bus is an 8-bit parallel bus which allows up to 15 devices connected to the same computer port. This bus can be used as daisy-chain connectors and each cable is 2 m or less in length. Extensive handshaking

controls the bus and many testing and measuring devices are equipped with GPIB. This bus can transfer data at reasonably fast speed of about 1 Mbit/s.

The IEEE-488 bus has five bus management lines such as IFC, ATN, SQR, REN and EOI. The IFC stands for *interface clear line*. When this line is asserted by the controller, it resets all devices on the bus to a starting state. When the *Attention Line* (ATN) is active low, it indicates that the controller is putting a universal command or an address command on the data bus. If the ATN line is high, the data lines contain data or a status byte. The *Service Request* (SQR) can work as an interrupt. Whenever any device requires to transfer data on the data bus, the SQR line becomes low. After that the controller can pool all the devices to find the device which needs service. The *Remote Enable Signal* (REN) allows an instrument to be controlled directly by the controller. The *End Of Identify* (EOI) signal is used by a talker to indicate the completion of data transfer. This bus has three handshake lines such as *Data Valid* (DAV), *Not Ready For Data* (NRFD) and *Not Data Accepted* (NDAC). These lines are used to coordinate the transfer of data bytes on the data bus. The difference between RS-232C and IEEE-488 is given in Table 9.17.

Table 9.17 Differences between RS232C and IEEE 488

<i>RS-232C</i>	<i>IEEE-488</i>
RS-232C is a standard serial communication interface. This is used for long-distance transmission with lower baud rates.	IEEE-488 is a standard parallel communication interface. This is used for short-distance (20 m) transmission at higher baud rates.
The maximum data transmission rate is 20,000 baud or 20 kbits/second	The maximum data transmission rate is 1 Mbits/second.
Telephone line can be used for data transfer	Maximum length of interconnection between two devices is about 20 m.
RS-232C is connected between DTE and DCE.	IEEE-488 can be connected to up to 15 devices.
RS-232C is not TTL compatible.	IEEE-488 is TTL compatible.
RS-232C is a male connector to connect with DTE, and RS-232C is a female connector to connect with DCE.	At the end of cable, both male and female connectors are available so that cables may be connected in daisy chain.
This is used for interfacing a computer to terminal communication equipment, modem.	This is used for interfacing a computer to measuring equipment/instruments.

9.9 8250 UART

The UART stands for Universal Asynchronous Receiver Transmitter. The UART is used in parallel-to-serial and serial-to-parallel conversions. Actually, UART receives data bytes from the computer and converts it into a single serial bitstream for outward transmission. During inward transmission of UART converts the serial bitstream into the bytes to feed into the computer. UARTs are available in different channels such as 1, 2, 4 and 8 channels. The single channel is represented by UART. DUART is a dual-channel UART and four channels of UART are represented by QUART. The eight channels of UART are called as octal of UART. The first PC UART, 8250, was manufactured by National Semiconductors. The evolution of UART ICs is given below:

8250 → 16450 → 16550 → 16C650A → 16C850
 UART UART UART UART UART

The pin diagram of 8050 is shown in Fig. 9.86(a) and its schematic pin diagram is depicted in Fig.9.86(b). The block diagram of 8250 UART is shown in Fig. 9.87.

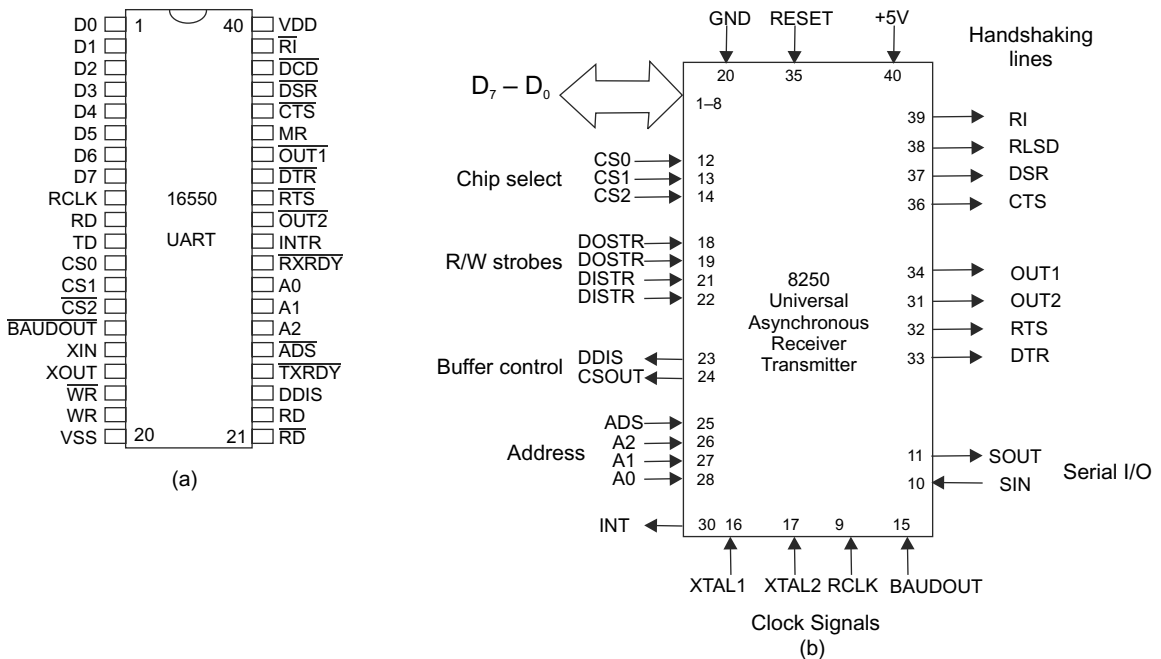


Fig. 9.86 (a) Pin diagram of 8250 UART (b) Schematic pin diagram of 8250 UART

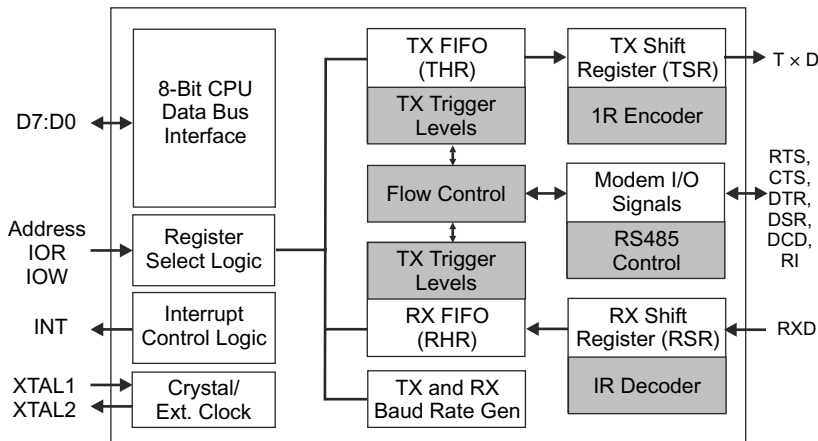


Fig. 9.87 Block diagram of 8250 UART

9.9.1 Registers

The 8250 UART has receive buffer register, transmitter holding register, interrupt enable register, interrupt identification register, line control register, modem control register, line status register, modem status register, scratch register, divisor latch register (LSB) and divisor latch register (MSB). Address signals A₂, A₁, A₀ select a UART register for the CPU to read or write to during data transfer. The state of the divisor latch access bit (DLAB) is the most significant bit of the line control register as shown in Fig.9.88. Table 9.18 shows the addresses of 8250 registers.

Table 9.18 8250 register address

DLAB	A ₂	A ₁	A ₀	Function
0	0	0	0	Receive buffer register for read, transmitter holding register for write
0	0	0	1	Interrupt enable register
x	0	1	0	Interrupt identification register read only
x	0	1	1	Line control register /data format register
x	1	0	0	Modem control register
x	1	0	1	Line status register
x	1	1	0	Modem status register
x	1	1	1	Scratch register
1	0	0	0	Divisor latch register LSB
1	0	0	1	Divisor latch register MSB

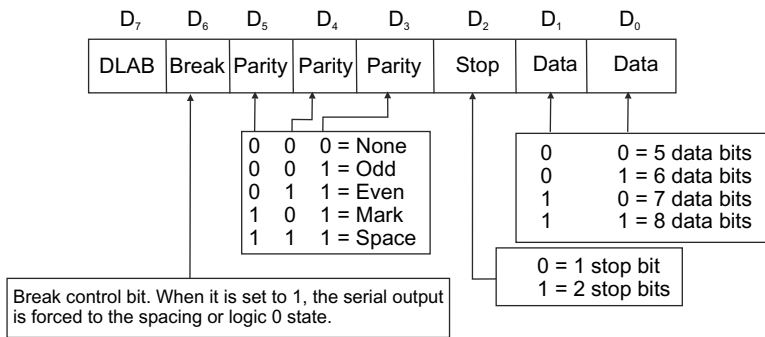


Fig. 9.88 Line control register

Line Status Register

This register provides status information to the CPU concerning the data transfer. Figure 9.89 shows the function of bits of the line status register.

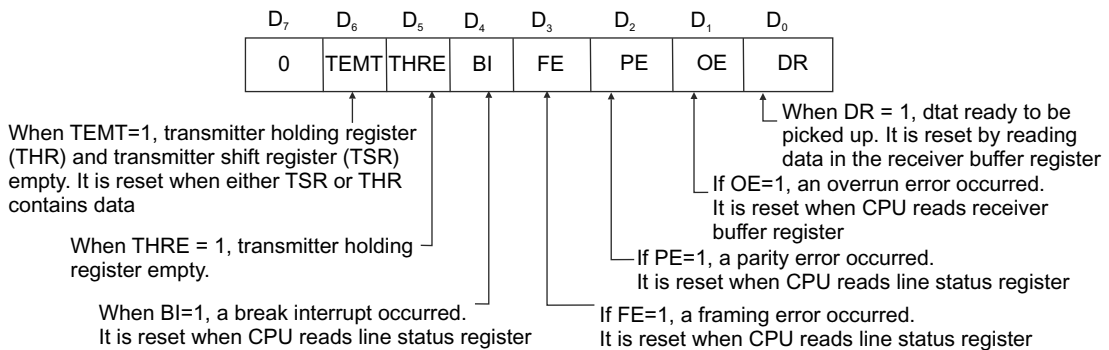


Fig. 9.89 Line status register

Interrupt Enable Register

This register is to indicate modem status, receiver line status, transmitter buffer empty and receive data available as shown in Fig. 9.90.

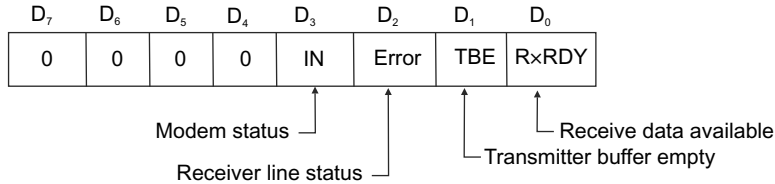


Fig. 9.90 Interrupt enable register

Interrupt Identification Register

To provide minimum software overload during data character transfers, the UART prioritizes interrupts into four levels such as serialization error, received data, transmitter buffer empty and modem status as shown in Fig. 9.91.

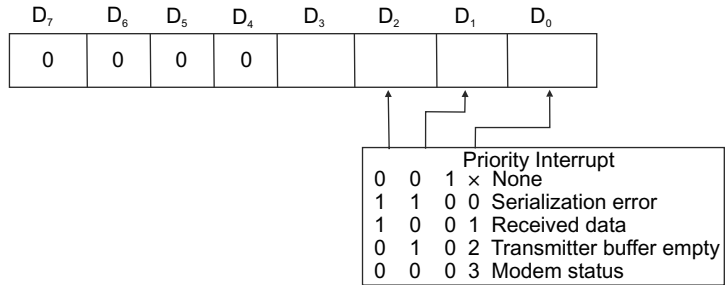


Fig. 9.91 Interrupt identification register

Modem Control Register This register controls the interfacing with the modem. The function of modem control register is depicted in Fig. 9.92.

Modem Status Register This register provides the current states of the control lines from the modem to the CPU. Figure 9.93 shows the modem status register.

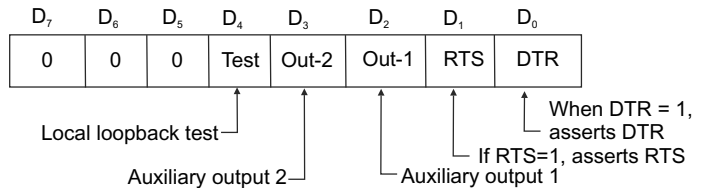


Fig. 9.92 Modem control register

Divisor Register The baud rate generator is programmed with a divisor that computes the baud rate of the transmitter section. Table 9.19 shows the baud rate and divisors at 1.8432 MHz.

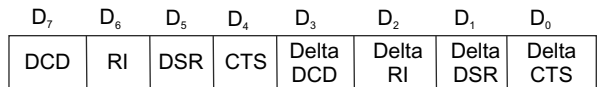


Fig. 9.93 Modem status register

Table 9.19 Baud rate and divisors at 1.8432MHz

Baud Rate	Decimal Divisor	Hex Divisor
110	1047	0417 H
300	384	0180 H
600	192	00C0 H
1200	96	0060 H
2400	48	0030 H
4800	24	0018 H
9600	12	000C H

9.9.2 Transmitter

A transmitter is used for parallel-to-serial conversion and transmits data from CPU to the serial port. This section has transmit(TX) FIFO register and transmit shift register (TSR). The transmitter operates in non-FIFO and FIFO mode. Figure 9.94(a) shows the non-FIFO mode of transmitter and FIFO mode of transmitter is depicted in Figure 9.94(b).

Initially, write data is sent to Transmit Holding Register (THR) and transmit data will be queued in TX FIFO. Then data is transferred to the Transmit Shift Register (TSR) and finally data is outputted from the transmitter. The TX character framing is done using start bit, data, stop bit and parity insertion as shown in Figure 9.95.

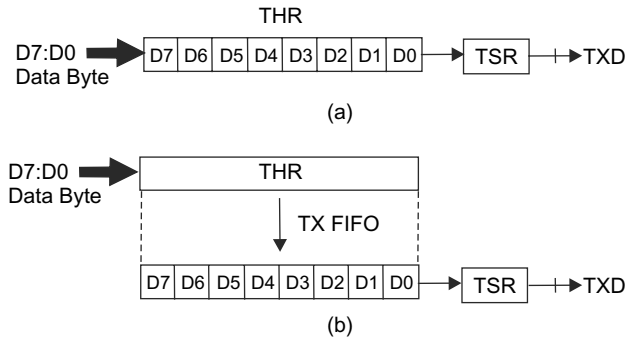


Fig. 9.94 Transmitter (a) non-FIFO mode (b) FIFO mode

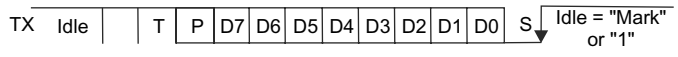


Fig. 9.95 TX character framing

9.9.3 Receiver

The receiver is used for serial-to-parallel conversion and receives data from serial port to CPU. This has received (RX) FIFO and Receive Shift Register (RSR). The receiver can be operated in non-FIFO and FIFO mode as shown in Figure 9.96. The incoming data is received in the receive shift register (RSR). Then the received data is queued in the RX FIFO. The error tags are associated with data in RHR which can be read through LSR. After that, Receive Holding Register (RHR) will be read and data is outputted from the receiver.

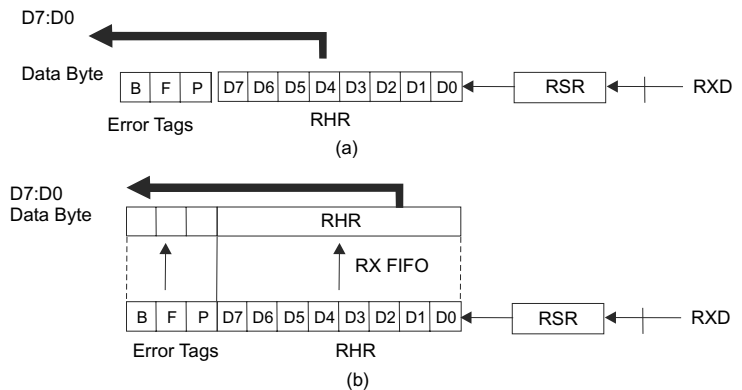


Fig. 9.96 Receiver (a) non-FIFO mode (b) FIFO mode

RX character validation is shown in Fig. 9.97. The high-to-low transition indicates a start bit. Start bit is validated if RX input is still low. During midbit sampling, the data, parity and stop bits are sampled at midbit. Line status errors are error tags and overrun error. Framing error exists if the stop bit is not detected. Parity error exists if the parity bit is incorrect. Break detected if RX input is LOW for duration of one character time. Overrun error exists if the character is received in RSR when RX FIFO is full.

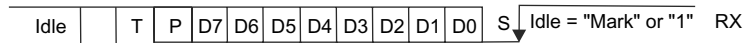


Fig. 9.97 RX character framing

9.9.4 UART Applications

UART is most commonly used in industry, telecom sector, automation, process control, remote access server, wireless applications, and entertainment systems as follows:

- ✓ **Industry** Building control, heating-ventilation-air-conditioning, security, telemetry, sensors, medical, testing and measurement, data terminals, video conference systems, photocopiers, printers, data recorder, and robot control, etc.
- ✓ **Telecom Sector** Network server management, hub, router, switch, console management, Bluetooth devices, keyboard-video-mouse switches and home networks, banking ATM, ticketing and vending, tolls collection systems, and car parking systems, etc.
- ✓ **Automation and Process Control** Processing, welding, printing and packaging, etc.
- ✓ **Remote Access Server** Modem servers and PC-based Internet-service-providers.
- ✓ **Wireless** Vehicle tracking, GPS, satellite, marine communication.
- ✓ **Entertainment** Set-top box, recreation and video-on-demand systems such as airplanes.

9.10 16550 UART

The 16550 UART (Universal Asynchronous Receiver/Transmitter) is an integrated circuit which is used for the interfacing for serial communications. Usually, it is used to implement the serial port for personal computers and it is also used to connect to an RS-232C for modems, printers, and other peripheral devices. The 16550 UART was manufactured by National Semiconductors as shown in Fig. 9.98 and it has the following features:

- ◆ The UART performs serial-to-parallel conversion on data received from modem and parallel-to-serial conversion on data received from the CPU using shift registers.
- ◆ The UART has Modem control capability. The Modem control functions are CTS, RTS, DSR, DTR, RI, and DCD.
- ◆ The UART can be operated in CHARACTER mode and FIFO mode. In FIFO mode, 16-bytes data to be stored in both receive and transmit modes.
- ◆ It has fully programmable serial-interface characteristics such as
 - 5-bit, 6-bit, 7-bit, and 8-bit characters
 - Even, odd, or no-parity bit generation and detection
 - 1-bit, 1½-bit and 2-stop bit generation
 - Baud generation up to 1.5 M baud
- ◆ A programmable baud generator divides the input clock by 1 to $(2^{16} - 1)$ and generates a $16 \times$ clock.
- ◆ False start bit detection, line break generation and detection
- ◆ Independently controlled transmit, receive, line status, and data set interrupts.
- ◆ The UART has complete status reporting capability.
- ◆ Internal diagnostic capabilities such as loopback controls for communications link fault isolation and break, parity, overrun and framing error simulation.
- ◆ UART has a processor interrupt system. Interrupts can be programmed as per programmer requirements.

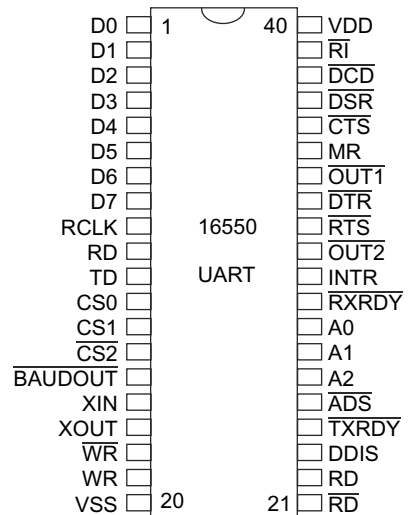


Fig. 9.98 16550 UART

In the FIFO mode, transmitter and receiver are buffered with 16-byte FIFO's to reduce the number of interrupts. In 16550 UART, asynchronous serial data are transmitted and received without any clock signal.

There are two separate sections of transmitters and receive for data communications. The transmitter and the receiver can work independently and the 16550 operates in simplex, half-duplex and full-duplex modes.

The 16550 UART provides different registers such as Transmit Holding Register (THR), Receive Holding Register (RHR), Interrupt Enable Register (IER), Interrupt Status Register (ISR), FIFO Control Register (FCR), Line Control Register (LCR), Modem Status Register (MSM) and Scratch Pad Register (SPR):

- ✓ **THR** The THR (Transmit Holding Register) is a write-only type register and its function is that loads data to be transmitted into TX FIFO.
- ✓ **RHR** The RHR (Receive Holding Register) is a read-only type register and its function is that reads out received data from the RX FIFO.
- ✓ **IER** The IER (Interrupt Enable Register) is a read/write type register and it is used to enable or disable interrupts.
- ✓ **ISR** The ISR (Interrupt Status Register) is a read-only type register and it can be used for highest priority pending interrupt.
- ✓ **FCR** The FCR (FIFO Control Register) is a write-only type register. The FIFO control register is used to enable the transmitter and receiver FIFOs. It clears the transmitter and receiver FIFOs and generates control for the 16550 interrupts. The transmitter and receiver are independently controlled. Set the RCVR FIFO trigger level and select the type of DMA signal. Figure 9.99 shows the FIFO control register.

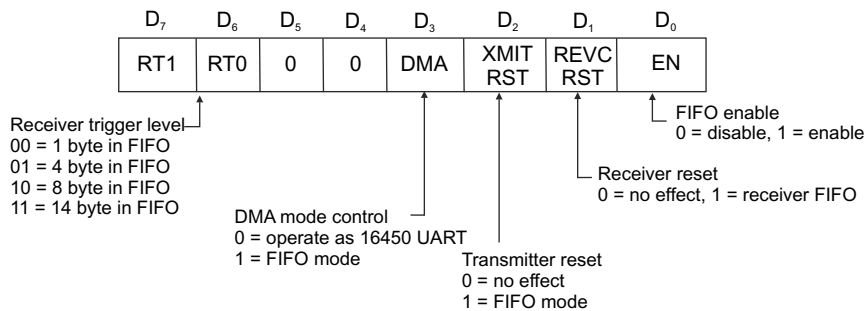


Fig. 9.99 FIFO control register

✓ **LCR** The LCR (Line Control Register) is a read/write type register. This register holds the number of data bits (word length), stop bit length, parity selection and break. Figure 9.100 shows the line control register, and the function of Stick Bit (ST) and parity bits is illustrated in Table 9.20.

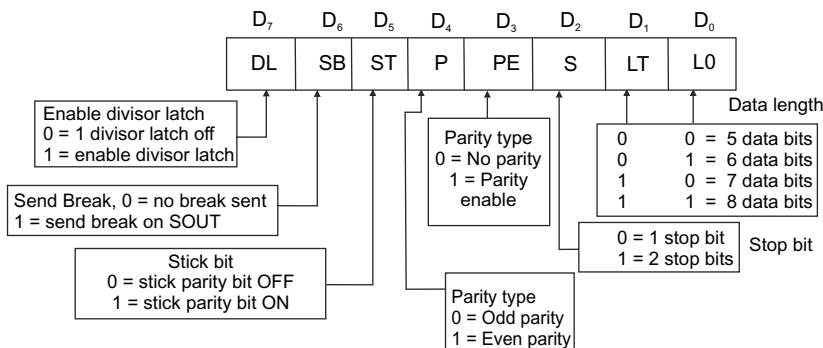


Fig. 9.100 Line control register

Table 9.20 Function of ST and parity bits

ST	P	PE	Function
0	0	0	No parity
0	0	1	Odd parity
0	1	0	No parity
0	1	1	Even parity
1	0	0	Undefined
1	0	1	Send/receive 1
1	1	0	Undefined
1	1	1	Send/receive 0

- ✓ **MSR** The MSR (Modem Status Register) is a read-only type register. This is used to store the state of modem inputs CD, RI, DSR, CTS and state changes since last read.
- ✓ **SPR** The SPR (Scratch Pad Register) is a read/write type register. This is used as a general-purpose read/write register.
- ✓ **DLL and DLM** These are read/write type registers and used as 16-bit divisors for the internal baud rate generator where DLL is LSB and DLM is MSB.
- ✓ **Line Status Register** This register provides status information to the CPU related with the data transfer. Figure 9.101 shows the line status register.

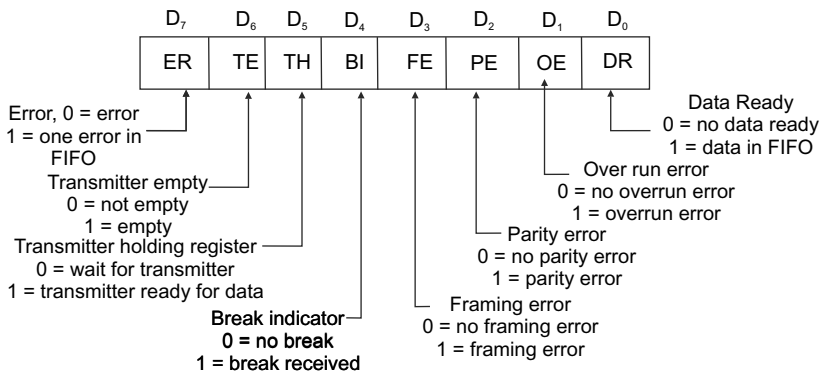


Fig. 9.101 Line status register

Programmable Baud Rate Generator

The UART consists of a programmable baud rate generator. The baud rate is the number of bits transferred per second including the start, data, parity and stop bits. The baud rate generator is programmed with a divisor and computes the baud rate. The input clock frequency is dc to 24 MHz which can be divided by any divisor from 2 to $2^{16} - 1$. The output frequency of the baud generator is $16 \times \text{baud}$. The 8-bit latches are used to store the divisor in a 16-bit binary format.

9.11 8089 I/O PROCESSOR

In a microprocessor-based system, there are many peripheral devices which are interconnected with the main processor as shown in Fig. 9.102(a). To perform any operation related to peripheral devices, the CPU must

initiate the specified operation and also track their operations. After completion of the input/output operation, CPU must be maintaining the post-operation status and records. Therefore, the CPU consumes some time for these operations and there will be always some delay in communication. To reduce the delay of operations related with peripheral devices, the I/O processor is used to control all input/output operations as depicted in Fig. 9.102(b). When operation is initiated by the main processor, the I/O processor receives a request from the system peripherals. I/O processors communicate with the main processor using its interrupt services.

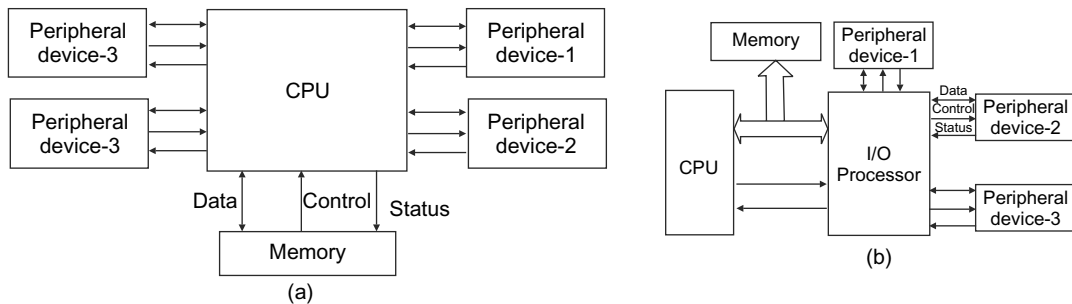


Fig. 9.102 (a) Peripheral devices handled by CPU (b) Peripheral devices handled by I/O Processor

The 8089 is an I/O processor in microprocessor input/output processing and is packaged in a 40-pin DIP package. The 8089 is a high-performance processor implemented in *N*-channel HMOS technology. The 8089s instruction sets are optimized for high speed, flexible and efficient I/O handling. It can work with 16-bit 8086 and 8-bit 8088 microprocessors with 8-bit and 16-bit peripherals. The communication with main processors is done using a memory table which states the details of the task to be executed. The memory table is prepared by the host processor to allot the specified task to the I/O processor. The host processor interrupts the I/O processor after sending a task to it. Then the I/O processor reads the memory table to get details of the allotted task. The memory table provides an address of the program written in the 8089 I/O processor instructions and it is called *channel program*. Then the 8089 I/O processor executes the channel program. Subsequently, the 8089 I/O processor can fetch and execute its own instructions. The features of 8089 I/O processors are as follows:

- High-speed DMA operations between I/O to memory, memory to I/O, memory to CPU, CPU to memory, and I/O to I/O
- Flexible, intelligent DMA functions such as translation, search, and word
- Memory-based communication with CPU
- Assembly/Disassembly
- Can support local or remote I/O processing
- Multi-bus compatible system interface
- Compatible with 8086/8088 processors
- 1 Mbyte addressability
- Interface with 8-bit and 16-bit peripherals

9.11.1 Pin Description

Figure 9.103 shows the pin diagram of 8089 and function of each pin is explained below:

$A_{15}/D_{15}-A_0/D_0$ (Input/Output) These lines are used as multiplexed address and data bus. $A_{15}-A_8$ are stable on transfers to a physical 8-bit data bus and are multiplexed with data on transfers to a 16-bit physical bus.

$A_{19}/S_6-A_{16}/S_3$ (Output) These are multiplexed most significant address lines and status information as given below:

S_6	S_5	S_4	S_3	Function
1	1	0	0	DMA cycle on Channel-1
1	1	0	1	DMA cycle on Channel-2
1	1	1	0	Non-DMA cycle on Channel-1
1	1	1	1	Non-DMA cycle on Channel-2

\overline{BHE} (Output) The Bus High Enable \overline{BHE} is used to enable data operations on the most significant half of the data bus $D_{15} - D_8$. when this signal is active low, a byte is to be transferred on the upper half of the data bus.

✓ **$\overline{S}_2, \overline{S}_1, \overline{S}_0$ (Output)** These are the status pins which state the different activities of I/O processor as given below:

\overline{S}_2	\overline{S}_1	\overline{S}_0	Function
0	0	0	Instruction fetch and I/O space
0	0	1	Data fetch and I/O space
0	1	0	Data store and I/O space
0	1	1	Not used
1	0	0	Instruction fetch and system memory
1	0	1	Data fetch; system memory
1	1	0	Data store; system memory
1	1	1	Passive

READY (Input/Output) This signal is received from the addressed device which indicates that the device is ready for data transfer.

\overline{LOCK} (Output) The lock output signal indicates to the bus controller that the bus requires more than one contiguous cycle.

RESET (Input) When the I/O processor receives the reset signal, the I/O processor has suspended its activities and enters an idle state until a channel attention is received.

The signal must be active for at least four clock cycles.

CLK (Input) Clock signal is required for internal I/O processor.

CA (Input) This is used as the channel attention signal of the I/O processor.

SEL (Input) The first CA received after reset informs the I/O processor through the SEL line, whether it is a master or slave and starts the initialization sequence.

$DRQ_2 - DRQ_1$ (Input) These are the DMA request input signals of the I/O processor. These signals indicate that a peripheral is ready to transfer/receive data using either Channel-1 or Channel-2.

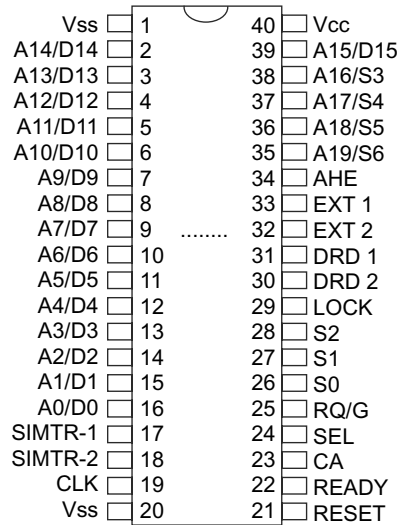


Fig. 9.103 Pin diagram of 8089

$\overline{RQ} / \overline{GT}$ (Input/Output)

The request grant signal is used to implement the communication dialogue required to arbitrate the use of the system bus between IOP and CPU, LOCAL mode or I/O bus when two I/O processors share the same bus. The $\overline{RQ} / \overline{GT}$ is active low signal.

 $SINTR_2$ - $SINTR_1$ (Output)

These are the interrupt output signals from Channel-1 and Channel-2 respectively.

 EXT_2 - EXT_1 (Input)

These are external terminate input signals for Channel-1 and Channel-2 respectively. The EXT signal is used to terminate the current DMA transfer operation if the channel is so programmed by the channel control register.

9.11.2 Functional Description of 8089 Architecture

The 8089 I/O processor has been designed to remove I/O processing, control and high-speed transfers from the CPU. This processor can support I/O peripheral devices and also support versatile DMA data transfer. The DMA function can boast flexible termination conditions such as external terminate, mask compare, single transfer and byte count. The DMA function of the 8089 I/O processor uses a two-cycle approach where the information actually flows through the 8089 I/O processor. This method of DMA data transfer vastly simplifies the bus timings and enhances compatibility with memory and peripherals.

Figure 9.104 shows the block diagram of 8089 architecture. The 8089 I/O processor has two internal I/O channels. These two channels can be programmed independently to handle two separate I/O tasks for the host CPU. The ALU and main control unit are shared by both the channels. The main control unit generates the control signals for the operation of the I/O processor channels. The bus control unit handles all the bus activities. The *Channel Control Pointer* (CCP) is used by the programmers and gets loaded with the 20-bit address of a memory table for the channel. Actually, this table is prepared by the host CPU to allot a task to the I/O processor. The address of the memory table for Channel 2 is computed by adding 8 to the memory table address of channel 1 or the contents of CCP.

The communication between the CPU and I/O processor is performed through a look-up table in shared memory. The CPU sends a hardware Channel Attention (CA) signal to the I/O processor so that 8089 executes a program by placing it in the 8089's memory space. The SEL pin indicates to the I/O processor about

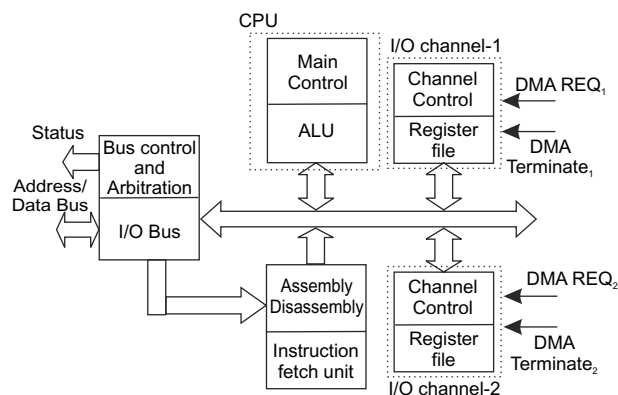


Fig. 9.104 Block diagram of 8089 I/O processors

the select channel. The communication from the I/O processor to the host processor can be performed in a similar way through system interrupt signals SINTR₂ and SINTR₁ when the CPU enables interrupts. The 8089 is able to store messages in the memory regarding its status and the status of any peripherals. Usually, this communication mechanism is supported by a hierarchical data structure as depicted in Figure 9.105.

After the first CA from RESET, if the I/O processor is initialized as the BUS MASTER, 5 bytes of information are read into the 8089 starting at location FFFF6. Then information about the type of system bus and pointers to the system configuration block are obtained. This is the only fixed location to be accessed by the 8089. The remaining addresses are obtained through the data structure hierarchy.

The 8089 computes addresses in the same way as 8086. The 20-bit address is generated by 4-bits left shifting of the 16-bit relocation pointer and added to the 16-bit address offset. Once 20-bit addresses are formed, they are stored in the 20-bit address registers of 8089. After the system configuration, pointer address is formed and the 8089 can access the *System Configuration Block (SCB)*.

The SCB is used only during startup, points to the *Control Block (CB)* and provides I/O processor system configuration data via the SOC byte. The SOC byte initializes the I/O processor I/O bus width to 8 bits or 16 bits, and states one of two $\overline{RQ}/\overline{GT}$ operating modes. During $\overline{RQ}/\overline{GT}$ mode 0 operation, the I/O processor is initialized as SLAVE, and its $\overline{RQ}/\overline{GT}$ is line attached to a Master CPU. In this mode, the CPU has control of the bus, grants control to the IOP. For mode 1, the I/O processor is useful only in remote mode between two I/O processors (MASTER and SLAVE).

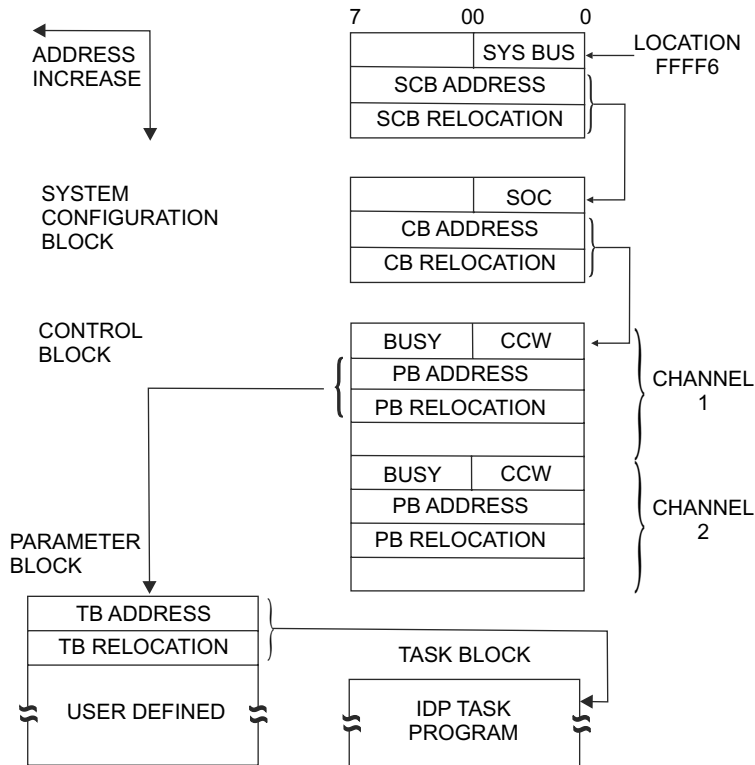


Fig. 9.105 Communication data structure

The *control block* performs the bus control initialization for the I/O processor operation and provides pointers to the parameter block or data memory for channels 1 and 2. The *Channel Control Word (CCW)* is retrieved and analyzed. The CCW byte is decoded to determine channel operation.

The *parameter block* contains the address of the task block and acts as a message center between the I/O processor and CPU. Parameters information is passed from the CPU to I/O processor in this block to adapt the software interface to the peripheral device. It is also used for transferring data and status information between the I/O processor and CPU.

The task block holds the instructions for the respective channel. This block can reside on the local bus of the I/O processor and the I/O processor can operate concurrently with the CPU. The advantage of the communication between the processor, I/O processor and peripherals is that it allows for a very clear method for the operating system to handle I/O routines.

The 8089 has separate registers for its two different I/O channels as shown in Figure 9.106. Each channel has two sets of registers such as pointers and registers. The pointers are 20-bit registers usually used to address memory, but the registers are 16-bit general-purpose data registers. Each of the pointers excluding PP register has a tag bit. This bit is used to indicate that either the 20-bit register content is to be used or the lower 16-bit register content is to be used as the pointer.

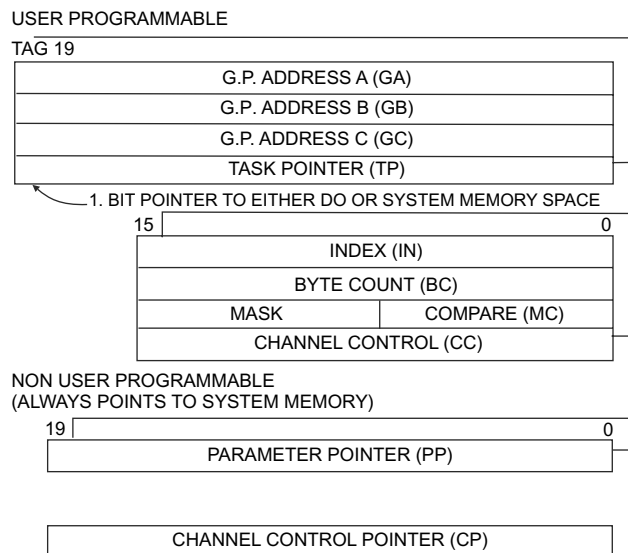


Fig. 9.106 Register model

The PP register is a 20-bit pointer. The registers GA, GB, GC, IN, BC and MC can be used as general-purpose registers, when they are not used as pointers. The memory operands can be accessed using one of the base pointers GA, GB, GC, and PP.

The basic DMA pointer registers (GA and GB) can point to the system bus or local bus, DMA source or destination, and can be automatically incremented. The register set GC can be used to allow translation during the DMA process through a look-up table. The channel control register can be accessed by MOV and MOVI instructions and the mode of the channel operation will be selected. Registers are also provided for a masked compare during the data transfer and can be set up to act as one of the termination conditions. Other registers can be used as general-purpose registers during program execution, while the I/O processor is not performing DMA cycles.

The 8089 I/O processor can be used in different applications such as file management in hard disk or floppy disk, soft error recovery routines and scan control, CRT control (cursor control and auto scrolling), keyboard control, communication control and general I/O applications.

SUMMARY

- General-purpose peripheral devices have been designed to provide services for different purposes in I/O communication and data transfer. Each device has a control word register and operation instructions. Therefore, each device must be initialized by a writing control word in the control word register for appropriate operation. Interfacing of these devices with a microprocessor has been incorporated in this chapter.
- In this chapter, the architecture, functional block diagram, pin diagram and operation of programmable devices such as Programmable communication interface 8251, Direct Memory Access (DMA) controller 8257, Programmable keyboard display controller 8279A, and 8275 CRT controller have been discussed.
- The implementation of direct memory access high-speed data transfers by using 8257 DMA controller are discussed. The 8251 is a programmable serial I/O IC known as USART and this device can perform synchronous and asynchronous serial data transmission. The functional details and operation of 8251 have also been explained in this chapter.
- In any microprocessor-based system, it is necessary to convert analog signal to digital signal and digital signal to analog form. The A/D converter converts analog signal to digital form, and a digital signal is converted into analog form by using D/A converter. In this chapter, the basic concept, operation and specification of A/D converter (ADC) and D/A converter (DAC) and the interfacing of data converter (ADC and DAC) ICs with 8085 microprocessor have been discussed.
- Bus interface devices such as ISA bus, VESA bus, USB, parallel printer interface, PCI bus, RS-232C, IEEE-488; 8250 UART, 16550 UART and 8089 I/O processors are also incorporated in this chapter.

MULTIPLE-CHOICE QUESTIONS

- | | |
|---|--|
| <p>9.1 8279 displays can operate in</p> <p>(a) 8 8-bit character display, left entry only</p> <p>(b) 16 16-bit character display, left entry only</p> <p>(c) 8 8-bit character display-right entry only</p> <p>(d) 8 8-bit character display-left and right entry and 16 16-bit character display, left and right entry</p> <p>9.2 How many bytes of memory are available for the seven segment display?</p> <p>(a) 16 × 8 Display RAM</p> <p>(b) 16 × 4 Display RAM</p> <p>(c) 16 × 1 Display RAM</p> <p>(d) None of these</p> | <p>9.3 How many bytes of memory are available for key pressed in 8279?</p> <p>(a) 8 × 8 RAM (b) 8 × 4 RAM</p> <p>(c) 8 × 1 RAM (d) None of these</p> <p>9.4 How many seven-segment displays can be connected with 8279?</p> <p>(a) 16 (b) 12</p> <p>(c) 10 (d) 8</p> <p>9.5 How many character keyboards can be connected with 8279 in mode?</p> <p>(a) 8 (b) 16</p> <p>(c) 20 (d) 4</p> |
|---|--|

- 9.6 The 8279 is a
 (a) DMA controller
 (b) programmable keyboard display interface
 (c) counter
 (d) interrupt controller
- 9.7 The 8257 is a
 (a) DMA controller
 (b) programmable keyboard display interface
 (c) counter
 (d) interrupt controller
- 9.8 The maximum number of data that can be transferred through 8257 is
 (a) 64K (b) 46K
 (c) 16K (d) 14K
- 9.9 DMA has
 (a) one channel (b) two channels
 (c) three channels (d) four channels
- 9.10 The signals are used for DMA operation are
 (a) HRQ (b) HLDA
 (c) HRQ and HLDA (d) None of these
- 9.11 At what speed is data transferred in 8251?
 (a) 300 (b) 256
 (c) 150 (d) 9600
- 9.12 The 8251 is a
 (a) USART IC (b) counter
 (c) interrupt controller
 (d) Programmable peripheral interface
- 9.13 The number of bits in case of asynchronous data transfer is
 (a) 2 bits (b) 3 bits
 (c) 4 bits (d) 5 bits
- 9.14 The 8251 operates in
 (a) synchronous mode
 (b) asynchronous mode
 (c) synchronous and asynchronous modes
 (d) none of these
- 9.15 When serial data can be transferred in either direction but in one direction at a time, the data transfer is known as
 (a) simplex (b) half duplex
 (c) full duplex (d) none of these
- 9.16 The UART performs
 (a) a serial-to-parallel conversion
 (b) a parallel-to-serial conversion
 (c) control and monitoring functions
 (d) all
- 9.17 The USART consists of
 (a) data bus buffer
 (b) control and logic function
 (c) transmit and receive buffer
 (d) all of these
- 9.18 The modems are used in serial data communication
 (a) as telephone circuits connected in some places of the circuit
 (b) as the switched telephone network connected in the circuit to reach anyone in the system
 (c) all of the above
- 9.19 The analog voltage corresponding to the LSB of 12-bit A/D converter is
 (a) $V/(2^{12} - 1)$ (b) $V/(2^{12} + 1)$
 (c) $V/2^{12}$ (d) None of these
- 9.20 The resolution of any 8 bit D/A converter with a full scale output of 10 V is
 (a) $\frac{10}{2^8 - 1}$ (b) $\frac{10}{2^8 + 1}$
 (c) $\frac{10}{2^8}$ (d) None of these
- 9.21 A digital instrument is used to measure analog voltage and display it in 7-segment display devices. The instrument has
 (a) an ADC at the input and a DAC at the output
 (b) an ADC at the input
 (c) a DAC at the input
 (d) an ADC at the output
- 9.22 Resolution of an 'N' DAC is
 (a) full-scale value/ 2^N
 (b) full-scale value/ $(2^N - 1)$
 (c) full-scale value/ $(2^N - 1)$
 (d) none of these
- 9.23 The minimum number of resistances required for any 8 bit weighted-resistor type DAC are

- (a) 8 (b) 9
 (c) 15 (d) 16
- 9.24 The resolution of a D/A converter is 0.4 per-
 cent of full-scale range. It is a
 (a) 8-bit converter (b) 10-bit converter
 (c) 12-bit converter (d) 16-bit converter
- 9.25 The input resistance of an $R-2R$ ladder D/A
 converter is
 (a) R for each digital input
 (b) $2R$ for each digital input
 (c) $3R$ for each digital input
 (d) none of these
- 9.26 A D/A converter's full scale output voltage is
 10V and it's accuracy is +0.4%. The maximum
 error of DAC will be
 (a) 20 mV (b) 30 mV
 (c) 40 mV (d) none of these
- 9.27 The speed of conversion is maximum in
 (a) successive approximation ADC
 (b) flash ADC
 (c) single slope serial ADC
 (d) Dual slope ADC
- 9.28 In an N -bit flash converter, the number of com-
 parators needed is
 (a) 2^N-1 (b) 2^N
 (c) 2^{N+1} (d) none of these
- 9.29 The N -bit successive approximation ADC
 requires
 (a) 2^N-1 clock pulses
 (b) 2^N clock pulses
 (c) N clock pulses
 (d) none of these
- 9.30 A 12-bit A/D converter has the input voltage
 signal from 0 V to +10 V. The voltage equiva-
 lent to 1 LSB will be
 (a) 0 (b) 1.2 mV
 (c) 2.4 mV (d) 0.833 V

SHORT-ANSWER-TYPE QUESTIONS

- 9.1 What is serial data transfer?
- 9.2 What is the difference between synchronous and asynchronous data transfer?
- 9.3 Define simplex, half-duplex and full-duplex data transfer.
- 9.4 What are the advantages of DMA controlled data transfer over interrupt-driven data transfer?
- 9.5 What are the building blocks of 8257?
- 9.6 What is the maximum value of kB of data that 8257 can transfer?
- 9.7 What are the different functions of 8279?
- 9.8 Define ADC. What are the types of ADC?
- 9.10 Write some applications of ADCs.
- 9.11 Define resolution. What is the resolution of 12-bit successive approximation ADC?
- 9.12 What is DAC? Write some applications of DACs.
- 9.13 What are the registers available in 8257? How is the 8257 is initialized?
- 9.14 What are the various input modes in which 8279 operates?
- 9.15 What is the difference between RS-232C and IEEE-488?

REVIEW QUESTIONS

- 9.1 Draw the block diagram of the 8251 chip and explain its working principles.
- 9.2 Write the functions of the following pins of 8251.
- | | | | |
|----------------------|-----------------------|--------------------------|-------------------------|
| (i) $T \times D$ | (ii) $T \times E$ | (iii) $R \times D$ | (iv) $T \times RDY$ |
| (v) \overline{DSR} | (vi) \overline{DTR} | (vii) C / \overline{D} | (viii) \overline{RTS} |
- 9.3 Draw the functional block diagram of 8251 and explain the operation of each block.
- 9.4 Describe the Read/Write control logic and registers.
- 9.5 Explain the operation of the transmitter section of 8251. How does the CPU know the transmitter buffer is empty?
- 9.6 Explain the operation of the receiving section of 8251. Why are modems used in case of digital transmission of data?
- 9.7 Explain the function of SYNDET/BD pin of 8251.
- 9.8 What are the modem control pins associated with 8251? Describe the functioning of these pins.
- 9.9 Discuss the mode instruction format for asynchronous transmission/reception.
- 9.10 Explain synchronous mode instruction format and command instruction format.
- 9.11 Draw the general transmission /receive format for synchronous communication.
- 9.12 Draw the status word format and explain the same.
- 9.13 Discuss the mode instruction format for synchronous transmission/reception case.
- 9.14 Show the command instruction format and explain briefly.
- 9.15 Explain how data can be transferred using 8251 USART at different baud rates. Write the features of 8251.
- 9.16 Explain the DMA operation with a suitable diagram. Why is DMA controlled data transfers faster?
- 9.17 Draw the functional block diagram of 8257 DMA and explain its operating principle.
- 9.18 Describe the features of 8257. How many I/O devices can access 8257?
- 9.19 Draw the architecture of 8257 and explain briefly.
- 9.20 Describe the flowchart of DMA mode of data transfer. What do you mean by DMA cycle?
- 9.21 Explain how the address registers and terminal count registers for each of CH0-CH3 are selected as also the mode set register and status word register.
- 9.22 Describe the status word register of 8257. Draw a timing diagram for DMA operation.
- 9.23 Explain the function of the following pins of 8257:
- | | | | |
|-----------------------|-----------------------|-----------|--------------|
| (i) HRQ | (ii) HLDA | (iii) TC | (iv) READY |
| (v) \overline{DACK} | (vi) \overline{DRQ} | (vii) AEN | (viii) ADSTB |
| (ix) MARK | | | |
- 9.24 Draw the functional block diagram of 8279 and explain the operation of each block.
- 9.25 Write the different features of programmable keyboard and display controller.
- 9.26 Write the functions of the following signals
- | | | | |
|---------------------|----------------------|----------------|--------------------------|
| (i) SL_0 - SL_3 | (ii) RL_0 - RL_3 | (iii) CNTL/STB | (iv) $OUTA_0$ - $OUTA_3$ |
|---------------------|----------------------|----------------|--------------------------|

- (v) OUTA₀–OUTA₃ (vi) IRQ (vii) SHIFT (viii) BD

- 9.27 What are the different modes of 8279 programmable keyboards and display controller? Explain each mode with an example.
- 9.28 Discuss the left-entry mode of display format.
- 9.29 Discuss the right-entry mode of display format.
- 9.30 Explain the seven-segment display interfacing with 8279. How are sixteen-digit displays interfaced with 8279?
- 9.31 Draw a circuit diagram to interface 8279 with a microprocessor and explain. Discuss the keyboard interface of 8279.
- 9.32 Write short notes on the following:
 (i) Right entry (ii) Left entry (iii) N key Roller (iv) Display RAM
 (v) 2 key Roller (vi) FIFO (vii) Command word format of 8279
- 9.33 Explain counting-type ADC with a suitable diagram. What are the limitations of this converter? How you can improve the performance of ADC?
- 9.34 Explain successive approximation type ADC. Compare dual-slope ADC and successive approximation ADC.
- 9.35 Draw an N -bit binary weight DAC and explain its operation. What are the disadvantages of binary weight DAC? What is the difference between binary weight DAC and R - $2R$ ladder DAC.
- 9.36 Draw R - $2R$ ladder circuit for 3 bits and explain with equivalent circuits.
- 9.37 Justify the following statements:
 (i) N -bit successive approximation ADC requires only N clock pulses for complete conversion.
 (ii) Successive approximation ADC is faster than counting type ADC
 (iii) Quantization error is $\pm\frac{1}{2}$ LSB.
 (iv) N -bit flash comparator requires 2^N-1 comparators.
- 9.38 Interface an A/D converter to 8085 and write a program to convert the analog input to digital.
- 9.39 Interface a D/A converter to 8085 and write a program to convert the digital input to analog output.
- 9.40 Draw the internal architecture of 8275 CRT controller and discuss briefly.
- 9.41 Discuss the general functions of 8275 CRT controller.
- 9.42 What is system bus? Explain the bus structure of a micro computer.
- 9.43 What is Expansion bus? What are the different slots on the mother board?
- 9.44 Write short notes on the following:
 (i) ISA (ii) EISA (iii) VESA (iv) PCI
 (v) AGP (vi) USB (vii) Parallel printer interface
- 9.45 What is RS-232C? Explain the function of different pins RS-232C.
- 9.46 How RS-232C is interfaced with TTL?
- 9.47 What is IEEE 488 Bus? Write the difference between RS 232C and IEEE 488.
- 9.48 Draw the block diagram of 8250 UART and explain in detail. Write the applications of UART.
- 9.49 What are the different features of 16550 UART? Explain the function of registers of 16550 UART.

Chapter 10

Applications of 8085/8086 Microprocessors

10.1 INTRODUCTION

Initially, standard logic gates, digital and analog ICs were used to measure any physical and electrical quantity in all electronics products. A product using standard logic gates can be replaced by interconnections of standard hardware with the logic stored in a ROM. When the logic is concentrated in only a few components, a high degree of design flexibility is possible. This type of system has limitations on size, weight, power consumption and price. The microprocessor makes it possible to improve old products in all directions and develop more sophisticated new industrial products incorporating new features. Microprocessor technology has been used to replace hardware designs, which were formally implemented with logic devices. Actually, microprocessor applications are limited only by the technology rather than by the imagination of the designers.

The microprocessor is a VLSI IC in which large numbers of transistors are placed. As microprocessors are relatively new devices, these devices should be used to implement various functions such as measurement of electrical and physical quantities, monitoring, controlling and protection of any process control system, motion control, servo control system and power system, etc. These devices are programmable and can substitute program logic for hardwired logic. Initially, microprocessor cost was too high, but due to rapid decrease in the microprocessor-based system cost, enormous logic power can be added with some additional integrated circuits in a microcomputer. The advantages of microprocessor-based design of a system are given below:

- ◆ The manufacturing costs of the electronic products are generally lower, but the typical microprocessor-based designs cost 60 to 20 per cent of their TTL implementation costs.
- ◆ The time and cost for the original development can be substantially lowered. Due to applications of microprocessors, the design time can be reduced by about two thirds. Presently, numbers of software are available to design a prototype system before implementation of the final product. Therefore, the design cycle will continue to decrease.
- ◆ Consequently, microprocessor-based products can be brought to the market very early as per consumer requirement.
- ◆ Microprocessor-based products have many complex functional capabilities and these products can be provided at reasonable cost. Therefore, the realization of better products for the same or lower prices are possible.
- ◆ The smaller number of components in a microprocessor system increase the reliability of the final product.

- ◆ Sometimes microprocessor-based products fail. The computational capability of a microprocessor can be used to perform self-diagnosing of the product to find error and help to remove faults. These devices also provide substantial reductions in service charges.

In industry, there are a variety of microprocessor applications such as instrumentation, industrial control, and aerospace, etc. Some of the actual applications come across industrial boundaries and these are more informative to about the type of function to be performed. Microprocessors are used in data-collection terminals, office equipment, business machines, calculators, point-of-sale terminals, and various kinds of data-communication equipments. As the incremental cost for additional functions is very small in a microprocessor-based system, always there is an increasing tendency to add greater functional capability. This tendency is most noticeable in the area of instrumentation, where increasingly sophisticated products are finding their way to the market in growing numbers. Presently, modern instruments have the additional features such as remote control, programmability, improved readout, and peripheral interfaces.

Generally, microprocessors are also used to control traffic lights, appliances, motion control, position control, servo control, elevators, automation, electric car, and control of AC/DC machines, etc. In this chapter, measurement and display of electrical and physical quantities such as voltage, current, frequency, phase angle, power factor, power, energy, force, displacement, speed, acceleration, temperature, pressure, stress, strain, deflection, water level, traffic-light control, overvoltage and overcurrent protection, speed control of dc and induction motors are discussed. Before discussion of measurement and control of some electrical and physical quantities, the seven-segment display will be discussed as it is used to display any quantity after measurement.

10.2 SEVEN-SEGMENT DISPLAY

Seven-segment display is widely used in calculators, digital watches, and measuring instruments, etc. Generally, Light Emitting Diode (LED), Liquid Crystal Display (LCD) segments provide the display output of numerical numbers and characters. To display any number and character, seven-segment display is most commonly used. Figure 10.1(a) shows the segment identification, and display of decimal numbers from 0 to 9 is given in Fig. 10.1(b). The light emitting diodes emit light when the anode is positive with respect to the cathode. There are two possible connections, namely, common anode and common cathode. In *common-anode connection*, seven anodes connected to a common voltage and cathode will be controlled individually to get the proper display. But in *common cathode connection*, anodes can be controlled individually for display when all cathodes are connected to a common ground of supply voltage as depicted in Fig. 10.2.

Figure 10.3 shows the block diagram of a 7-segment display. The decimal number 0 to 9 can be displayed by the binary coded decimal input. For example, the segments a, b, c, d, e, and f will be bright for decimal number 0. Table 10.1 shows the different segments, which will be bright for decimal numbers 0 to 9. IC 7447 can be used as a decoder circuit for converting binary coded decimal inputs into seven-segment display. Figure 10.4 shows the pin configuration of IC 7447 and the pin description is given below:

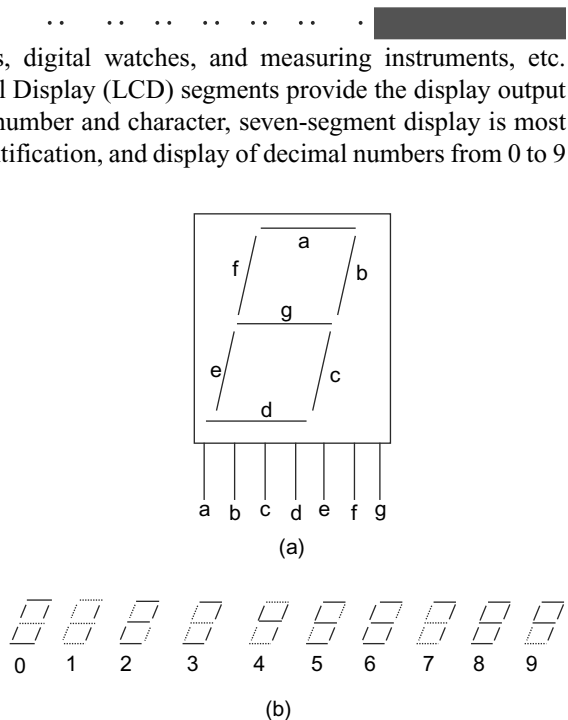


Fig. 10.1 (a) Segment identification
(b) Numerical displays

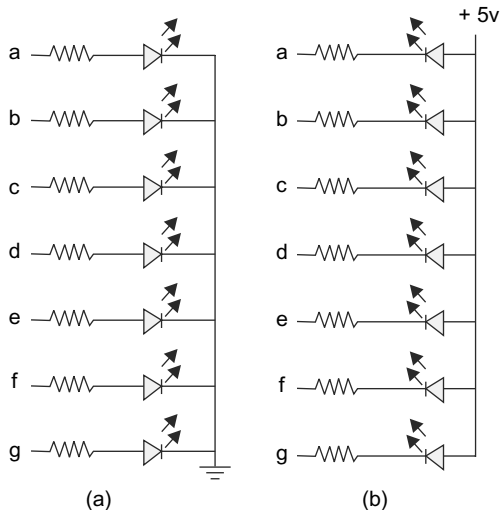


Fig. 10.2 (a) Common-cathode connection, and (b) Common-anode connection of seven segment display

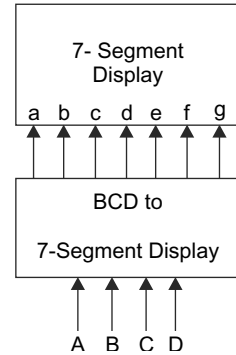


Fig. 10.3 Block diagram of seven-segment display

A_0-A_3 BCD inputs, $\overline{RB1}$ Ripple blanking, \overline{LT} Lamp test input
 $\overline{BI} / \overline{BRO}$ Blanking input/Ripple blanking output, $\overline{a} - \overline{g}$ Segment outputs

The IC 7447 decodes the input data given in the truth table 10.1. IC 7447 is BCD to 7-segment decoder with open-collector outputs. The 74LS47 has four input lines of BCD (8421) data, and it generates their complements internally. Then the decoder decodes the data and decoder outputs can be used to drive indicator segments directly. Each segment output sinks about 24 mA in the ON/LOW state and can withstand 15 V in the OFF/HIGH state. Some auxiliary inputs, namely, ripple blanking, lamp test and cascadable zero-suppression functions are also provided in IC 7447. Zero suppression logic is very useful in multi-seven-segment decoder.

Table 10.1 Truth table for seven-segment display

Decimal Number	Inputs				Outputs						
	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1

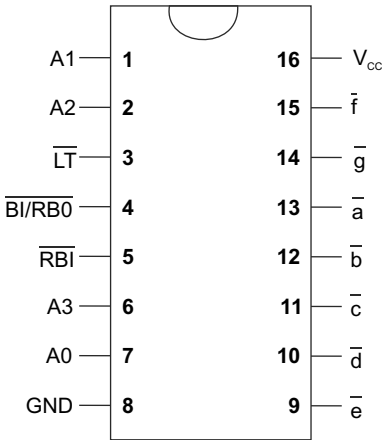


Fig. 10.4 Pin configuration of IC 7447

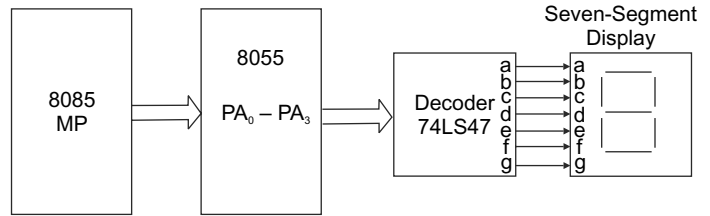


Fig. 10.5 Block diagram of Single-digit display

10.2.1 Single-Digit Display

The seven-segment displays are not directly connected with the I/O ports of 8255. Actually they are connected through either buffers or drivers or decoders. Figure 10.5 shows the interfacing of a decoder and a seven-segment display with microprocessor. Assume all I/O ports of 8255 operate as output port. Then control word of 8255 will be 80H. The pins PA₀–PA₃ of Port A are connected to the decoder 74LS47. Therefore, binary inputs corresponding to the decimal number 0 to 9 are applied to 74LS47 and the decimal number 0 to 9 will be displayed in the seven-segment display. The program for displaying the decimal number is given below:

PROGRAM 10.1 for Single Digit Display

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	3E, 80		MVI	A, 80H	Load control word of 8255 in accumulator
8002	D3, 03		OUT	03H	Write control word in control word register and initialize ports
8004	3E, 09		MVI	A, 09H	Load 09 in accumulator
8006	D3, 00		OUT	00	Send 09 to Port A for display
8008	76		HLT		Stop

Initially control word of 8255, 80H is loaded in accumulator and writes the control word in control word register to initialise all ports as output ports. After that, 09H is loaded in the accumulator and microprocessor outputs 09H in Port A. The binary logic for 9 is output on the pins PA₀–PA₃, which are connected with seven-segment display for display. The pins PA₄–PA₇ are the MSD of the decimal number 09. Hence the logic for 0 is output on the pins PA₄–PA₇. These pins are not connected anywhere and consequently ‘0’ is not displayed.

10.2.2 Two-Digit Display

In two-digit display, two decoder drivers and two seven-segment displays are used as shown in Fig. 10.6. The LSB will be displayed in one and MSD will be displayed in another seven-segment display. Thus, two

display units will display two-digit decimal numbers. To display 99H, the program is illustrated. After execution of this program, PA₀–PA₃ will be 9H and PA₄–PA₇ will be 9H. As Port A outputs are connected to two seven-segment display units through decoder IC 74LS47, 99H will be displayed in the seven-segment display.

PROGRAM 10.2 for Two-Digit Display

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	3E, 80		MVI	A, 80H	Load control word of 8255 in accumulator
8002	D3, 03		OUT	03H	Write control word in control word register and initialize ports
8004	3E, 99		MVI	A, 99H	Load 99 in accumulator
8006	D3, 00		OUT	00	Send 99 to Port A for display
8008	76		HLT		Stop

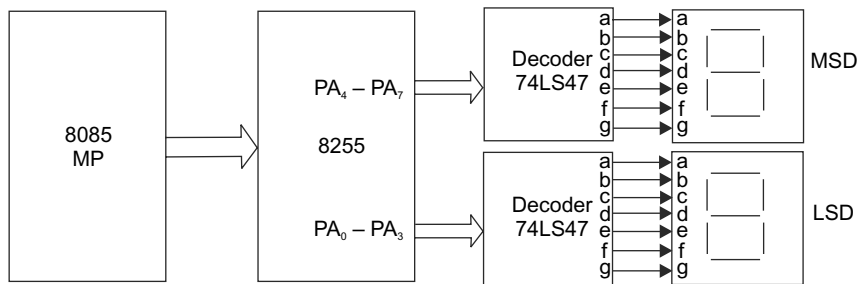


Fig. 10.6 Block diagram of two-digit display

10.2.3 Four-Digit Display

Figure 10.7 shows the four-digit display, which consists of four decoder drivers and four seven-segment display units. Port A and Port B are used for driving the decoder. The program for four-digit display is as follows:

PROGRAM 10.3 for Four-Digit Display

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	3E, 80		MVI	A, 80H	Load control word of 8255 in accumulator
8002	D3, 03		OUT	03H	Write control word in control word register and initialize ports
8004	3E, 12		MVI	A, 12H	Load 12 in accumulator
8006	D3, 00		OUT	00	Send 12 to Port A for display
8008	3E, 34		MVI	A, 34H	Load 34 in accumulator
800A	D3, 01		OUT	01	Send 34 to Port B for display
800C	76		HLT		Stop

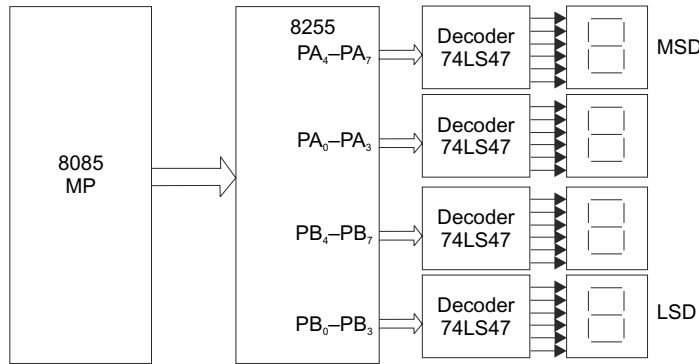


Fig. 10.7 Block diagram of four-digit display

Here, Port A is used to display most significant digits and Port B displays the least significant digits. After execution of the above program, 1234 will be displayed in the seven-segment display section.

To display more than four digits, more I/O ports and large number interfacing devices are required. To reduce the number of I/O ports and interfacing components, multiplexing technique is used to display large number of digits or letters or characters. Figure 10.8 shows the block diagram of a multi-digit display system. In this system, only one digit will display at a time. PA₀–PA₃ of Port A is connected to the decoder, and the decoder outputs are connected to seven-segment display. PB₀–PB₃ of Port B are fed to the multiplexer for selecting any one of the several seven-segment display units. Each seven-segment display unit will be turned ON and OFF in a sequence and this process will be repeated continuously. In this scheme, sixteen digits can be displayed simultaneously as the multiplexer has four inputs. Firstly, one of the sixteen seven-segment display units will be selected through the multiplexer and afterward the desired segments of the seven segments of the LED are to be turned ON to display any number. The same process may be repeated for the second, third and other seven-segment LEDs. The process can be repeated in cyclic order with a minimum time delay.

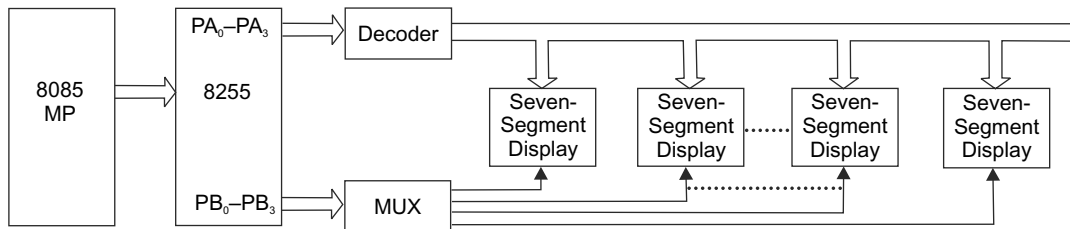


Fig. 10.8 Block diagram of multi-digit display using multiplexer

10.3 MEASUREMENT OF ELECTRICAL QUANTITIES

10.3.1 Measurement of Voltage

Figure 10.9 shows the schematic block diagram of dc voltage measurement. Firstly, the dc voltage is applied to the peak detector circuit to detect the peak value of dc voltage when the dc voltage varies instantaneously. Usually, the average and rms values of input voltage are directly proportional to peak of dc voltage. The

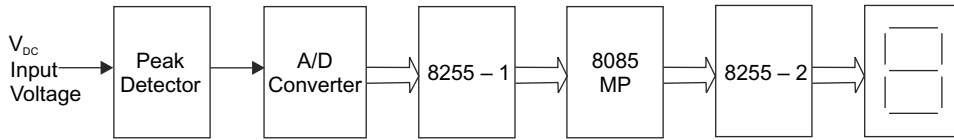


Fig. 10.9 The dc voltage measurement

peak detector circuit is depicted in Fig. 10.10. When input voltage is positive, output of the operational amplifier drives the diode *D*, and the capacitor *C* will be charged to the positive peak value of dc input voltage. In this circuit, when the diode *D* is forward bias, operational amplifier operates as a voltage follower. If input voltage becomes negative, the diode *D* is reverse biased and voltage across the capacitor *C* will be retained as the capacitor discharges through resistance *R_L* only. For satisfactory operation of peak detector circuit, the charging and discharging time constant must follow the given condition.

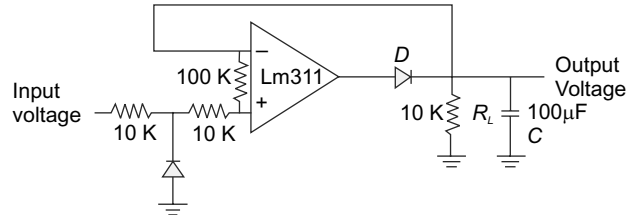


Fig. 10.10 Peak voltage detector circuit

$$CR_D \leq CR_L$$

where, *CR_D*—charging time constant,
CR_L—discharging time constant
R_D—resistance of forward bias diode,
R_L—load resistance

The output of the peak detector circuit is fed to the A/D converter. Subsequently, analog dc voltage is applied to the analog input terminal of the A/D converter. The output of the A/D converter is digital equivalent to the analog input voltage. The outputs of the A/D converter are connected to the I/O ports of 8255 and the microprocessor can read this digital output of ADC through 8255 and transfer the digital data to the accumulator. ADC interfacing with 8255 and the microprocessor is illustrated in Fig. 10.11. In this section, the program for dc voltage measurement in the range of 0 to 5 V has been incorporated and displayed in the mV range. Therefore, 0000 will be displayed for 0 V and similarly 5000 will be displayed at 5 V dc input voltage. While writing the program, the given steps are followed for dc voltage measurement:

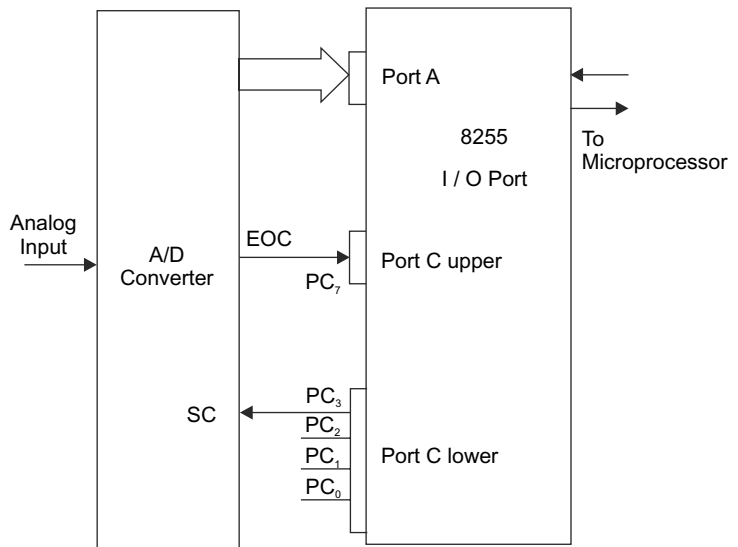


Fig. 10.11 ADC interfacing with 8255 and microprocessor

1. Convert analog input voltage into digital form and store in memory location
2. Find the memory location of the look-up table where the calibrated data of digital equivalent voltage is stored. The address is calculated from the following expressions. Actual address = Hex code of analog input voltage × 2 + Initial address

- Call data stored in the two consecutive memory locations and display in the address field of the microprocessor kit or in the seven-segment display.

To measure 0 to 5 V dc, the A/D converter IC operates in unipolar mode and the digital equivalent value of 0–5 V will be 00H to 80H. 00H is equivalent to 0 V and 80 is equivalent to 5 V. When an analog input voltage is converted to a digital equivalent by the A/D converter then the microprocessor reads a digital equivalent voltage. The look-up table or multiplication factor will be used for calibration of digital equivalent voltage to display. The resolution of 8-bit 0–5 V ADC is 20 mV. Then relationship between analog input voltage and hex output is

$$V = X \frac{20}{1000} = \frac{X}{50} \text{ V}$$

where, X is the decimal equivalent of a hex number.

Two I/O ports, 8255-1 and 8255-2, are used for measurement of dc voltage. Port A and Port C of 8255-1 are used for ADC interfacing, and Port A and Port B of 8255-2 are used for display interface. Assume Port A, and Port C upper of 8255-1 operate as input ports, and Port B and Port C lower as output ports. It is also assumed that all ports of 8255-2 operate as output ports. The control word of 8255-1 is 98H and the control word of 8255-2 is 80H. Assume the following port addresses of 8255-1 and 8255-2 as shown in Table 10.2. The program for dc voltage measurement with the help of a look-up table is given below.

Table 10.2 Port addresses

8255-1		8255-2	
Port	Port Address	Port	Port Address
Control word register	03H	Control word register	0BH
Port C	02H	Port C	0AH
Port B	01H	Port B	09H
Port A	00H	Port A	08H

PROGRAM 10.4 for dc Voltage Measurement

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	3E, 98		MVI	A, 98H	Load control word of 8255-1 in accumulator
8002	D3, 03		OUT	03H	Write control word in control word register and initialize ports
8004	3E, 08	START	MVI	A, 08H	Send start of conversion signal through PC ₃
8006	D3, 02		OUT	02H	PC ₃ is high
8008	3E, 00		MVI	A, 00H	As PC ₃ will be high for one or two clock pulse, make it 0
800A	D3, 0A		OUT	02H	PC ₃ becomes low
800C	DB, 02	LOOP	IN	02	Read end of conversion signal
800E	17		RAL		Rotate accumulator to check either conversion is over or not.
800F	D2, 0C, 80		JNC	LOOP	If conversion is not completed, jump to LOOP

(Contd.)

(Contd.)

8012	DB, 00	IN	00	Read digital output of A/D converter
8014	2F	CMA		Complement of ADC output
8015	D6, 80	SUI	80H	Subtract 80H
8017	21, 50, 80	LXI	H, 8050H	Load 8050H in HL-register pair
801A	77	MOV	M, A	Store accumulator content in 8050H location
		CALL	8100	Call calibration subroutine
		CALL	8150	Call display subroutine
8000	76	JMP	START	Jump to start

Subroutine to find initial address of memory location of look-up table for calibration and stored calibrated data in two consecutive memory location 8200 and 8201H

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8100	21, 50, 80		LXI	H, 8050H	Load 8050H in HL-register pair
8103	3E, 00		MVI	A, 00H	Store 00H in accumulator and C register
8105	0E, 00		MVI	C, 00H	
8107	86		ADD	M	Add content of memory (digital equivalent of analog input voltage) with accumulator
8108	86		ADD	M	Add content of memory with accumulator
8109	D2, 0E, 81		JNC	LEVEL-1	If no carry, jump to Level-1
810C	0E, 01		MVI	C, 01	Load 01H in C register
810E	6F	LEVEL-1	MOV	L, A	Move accumulator content into L register
810F	26, 90		MVI	H, 90H	Load 90H in H register
8111	79		MOV	A, C	Content of C register with H register
8112	84		ADD	H	
8113	7E		MOV	A, M	Move content of memory location specified by HL register pair into accumulator
8114	32, 00, 82		STA	8200	Store two least significant digits of display voltage stored in 8200H
8117	24		INX	H	Increment the HL register pair
8118	32, 01, 82		STA	8201	Store two most significant digits of display voltage stored in 8201H
811B	C9		RET		

Subroutine for Display

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8150	3E, 80		MVI	A, 80H	Load control word of 8255-2 in accumulator
8152	D3, 0B		OUT	0BH	Write control word in control word register and initialize ports
8154	3A, 00, 82		LDA	8200H	Load data in accumulator from 8200H
8157	D3, 09		OUT	09	Send data to Port B for display LSD
8159	3A, 01, 82		LDA	8201H	Load data in accumulator from 8201H
815C	D3, 08		OUT	08	Send data to Port A for display MSD
815E	C9		RET		

Table 10.3 Look-up table between hex code and display voltage

Memory Address	Hex code of Voltage	Voltage in Display (mV)	Memory Address	Hex code of Voltage	Voltage in Display (mV)
9000	00	0000	9080	40	2500
9002	01	0039	9082	41	2539
9004	02	0078	9084	42	2578
9006	03	0117	9086	43	2617
9008	04	0166	9088	44	2666
...
907E	3F	2461	9100	80	4999

The schematic block diagram of ac voltage measurement (peak value) is illustrated in Fig. 10.12. A rectifier is used to convert ac voltage into dc voltage. Therefore, rectifier output is a dc voltage source, which can be represented as constant voltage source in series with a resistance. In a microprocessor-based system, it is necessary to rectify millivolt high-frequency signal. When conventional diodes are used in a rectifier circuit, there is always some voltage drop across forward bias diode 0.2 V for Ge diode and 0.7 V for Si diode, so that

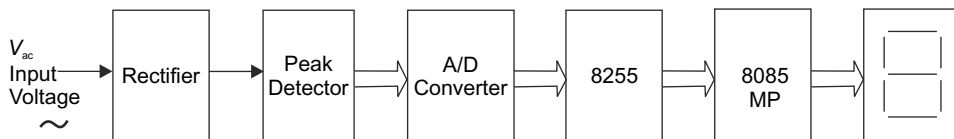


Fig. 10.12 An ac voltage measurement

a millivolt signal cannot be able to forward bias the conventional diode. The switching speed of a conventional diode is also low. If a diode is used in the forward path of an operational amplifier, cut-in-voltage of diode will be divided by the open-loop gain, which is very large in an operational amplifier. Then the diode can operate in ideal mode with zero cut-in voltage. If a conventional diode is used in feedback path of operational amplifier, high-frequency millivolt can be rectified. This rectifier is known as *precision rectifier*. There are two types of precision rectifiers such as half-wave rectifier and full-wave rectifier. Figure 10.13 shows the half-wave and full-wave precision rectifiers.

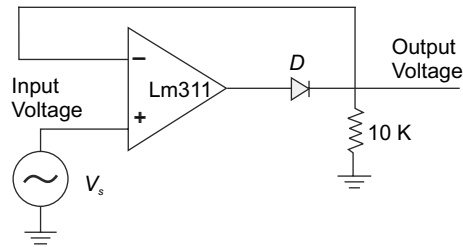


Fig. 10.13(a) Half-wave precision rectifier

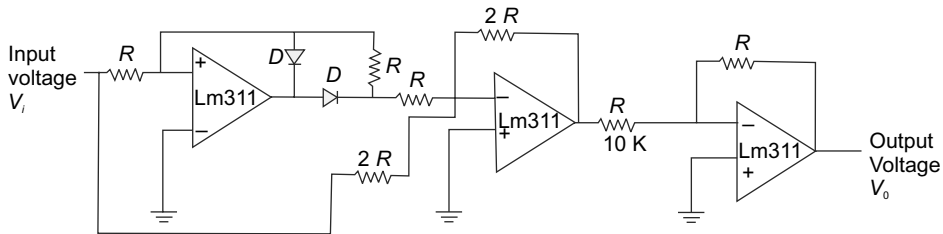


Fig. 10.13(b) Full-wave precision rectifier

The magnitude of ac voltage changes with time. So a sample hold circuit is required to maintain analog voltage during the A/D conversion process. A bi-directional A/D converter must be used to measure positive as well as negative voltage to display the instantaneous value of analog ac voltage. To measure the average as well as rms value of ac voltage, a precision rectifier is used to convert ac voltage to dc. Then measure the dc voltage with the help of an A/D converter, 8255 and microprocessor and after proper calibration it will be displayed in the address field of the microprocessor or in the seven-segment display.

A similar process will be followed for writing program to measure the ac voltage. To measure high voltage, a Potential Transformer (PT) is required. The primary of PT is connected to high-voltage supply and the output from PT secondary winding is fed to precision rectifier.

10.3.2 Measurement of Current

In current measurement, firstly the current must be converted into voltage using I to V converter and then voltage is measured by the microprocessor with the help of an A/D converter and 8255 programmable peripheral interface IC. Figure 10.14 shows the dc current-to-voltage converter circuit. Current flow through the R_s resistance is zero as operational amplifier is virtually grounded. Therefore, I_2 current flows through resistance R_2 and output voltage is equal to $V_o = -I_2 R_2$. The ac current-to-voltage converter circuit using Current Transformer (CT) and operational amplifier is depicted in Fig. 10.15. If output voltage at secondary of CT is V_1 , output voltage will be equal to $V_o = -\frac{R_2}{R_1} V_1$.

Figure 10.16 shows the current measurement of a dc motor. For the measurement of current, the current signal is converted to voltage signal. A very low resistance of approximately 0.1 ohms may be connected in series with the load. Then voltage across the resistance should be equal to 0.1 times of load current (I). After amplifying by ten times, the amplifier output voltage must be equal to the current. Then output analog voltage is applied to the input of the peak detector circuit so that it can deliver 5 volts when 5 A current flows through the circuit. Thereafter, by measuring voltage, we can measure analog input current in digital form. The program for voltage measurement may be used in current measurement also. But there will be some changes in the look-up table. In the look-up table for each digital input signal due to corresponding input, current calibrated data must be stored in two successive memory locations as given in Table 10.4.

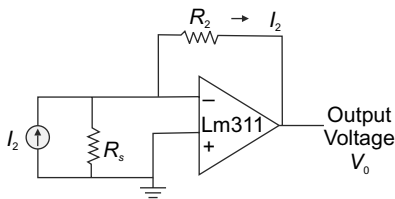


Fig. 10.14 A dc current to voltage converter

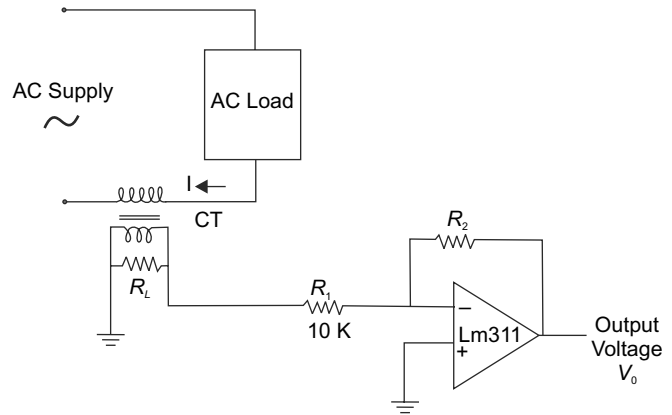


Fig. 10.15 An ac current-to-voltage converter

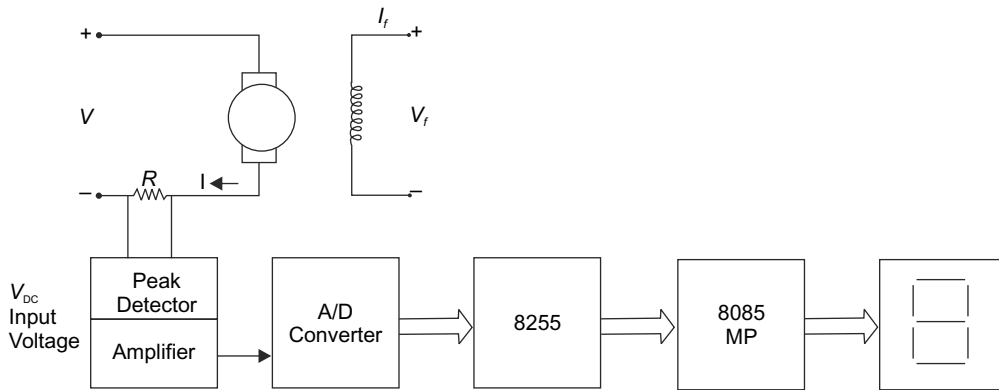


Fig. 10.16 A dc current measurement

In ac current measurement, a Current Transformer (CT) may replace the sensor resistance. The current transformer (CT) or a core-balanced transformer should be connected with a burden as depicted in Fig. 10.17. The output signal in the voltage drop across the secondary of CT or burden resistance is applied to an A/D converter through precision rectifier and peak detector circuit. Then the microprocessor measures analog voltage and displays it in seven-segment display units.

For dc current measurement, assume $R = 0.1 \Omega$ and maximum current $I = 5 \text{ A}$. Then IR drop = 0.5 V. If an amplifier is used to increase voltage with a gain = 10, output voltage will be 5 V. Subsequently, the look-up table for voltage measurement can be used and current displayed in mill-amperes.

In ac current measurement assume CT ratio = 10 : 1, and primary current $I_p = 10 \text{ A}$. Then current flow through secondary $I_s = 1 \text{ A}$, and voltage across resistance $V_R = 0.1 \text{ V}$. Two amplifiers circuits will be used to produce 5 V output. Gain of the first-stage amplifier is 10 and the second-stage amplifier gain is 5. Here full-scale analog voltage is 5 V for 10 A current and its digital equivalent output is 80H. The program for voltage can be used in this case but the look-up table must be modified. The modified look-up table is given below:

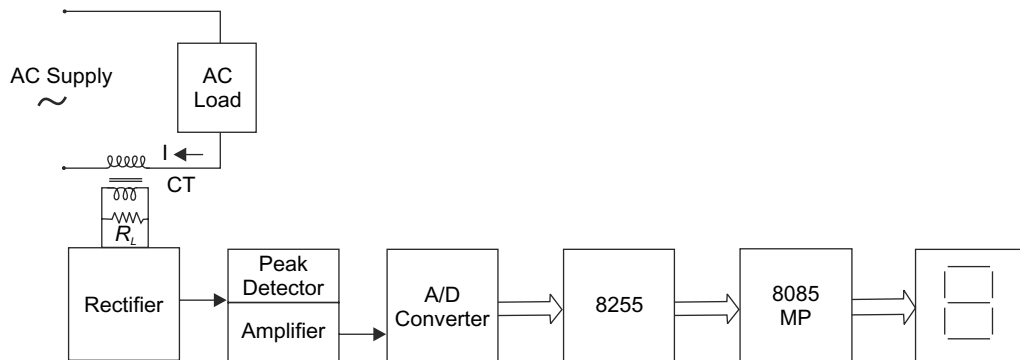


Fig. 10.17 An ac current measurement

Table 10.4 Look-up table between Hex code of current and current (mA)

Memory Address	Hex code of Current	Current (mV)	Memory Address	Hex code of Current	Current (mV)
9000	00	0000	9080	40	5000
9002	01	0078	9082	41	5078
9004	02	0156	9084	42	5156
9006	03	0234	9086	43	5234
9008	04	0312	9088	44	5312
...
907E	3F	4921	9100	80	9999

10.3.3 Frequency Measurement

To measure the frequency of an ac signal, firstly the ac signal is converted into dc square waveform using Zero Crossing Detectors (ZCD) as shown in Fig. 10.18. Then the time period of half cycle of square wave is measured, which is directly related with frequency. The relation between frequency (f) and time period (T) is $f = \frac{1}{T}$.

Usually, the Zero Crossing Detector (ZCD) is an open-loop or saturation-mode operation of the operational amplifier and it is basically a comparator with zero reference voltage. Zero crossing detectors are of two types—inverting and non-inverting type as shown in Fig. 10.18 (a) and (b) respectively. Whenever the input voltage crosses the zero axis, the output voltage changes abruptly. In an inverting zero crossing detector, input voltage is applied to the inverting terminal, and the output voltage signal is out of phase (180 phase shift from the input voltage). In case of non-inverting zero crossing detector, the input voltage is applied to the non-inverting terminal. Therefore, the output voltage is in phase with the input signal.

As depicted in Fig. 10.18(b), ZCD converts the positive half-cycle of ac input voltage to a rectangular waveform. Output is low in the negative half-cycle and high in the positive half-cycle. The amplitude of a square wave will be either $+V_{CC}$ or $-V_{CC}$. When the operational amplifier operates in $+12\text{ V}$ and -12 V , the output voltage will vary in between $+12\text{ V}$ and -12 V . But microprocessors can operate at 5 V only. Therefore, the output square wave must be converted into $+5\text{ V}$ and 0 V using some additional arrangement in ZCD as depicted in Fig. 10.19.

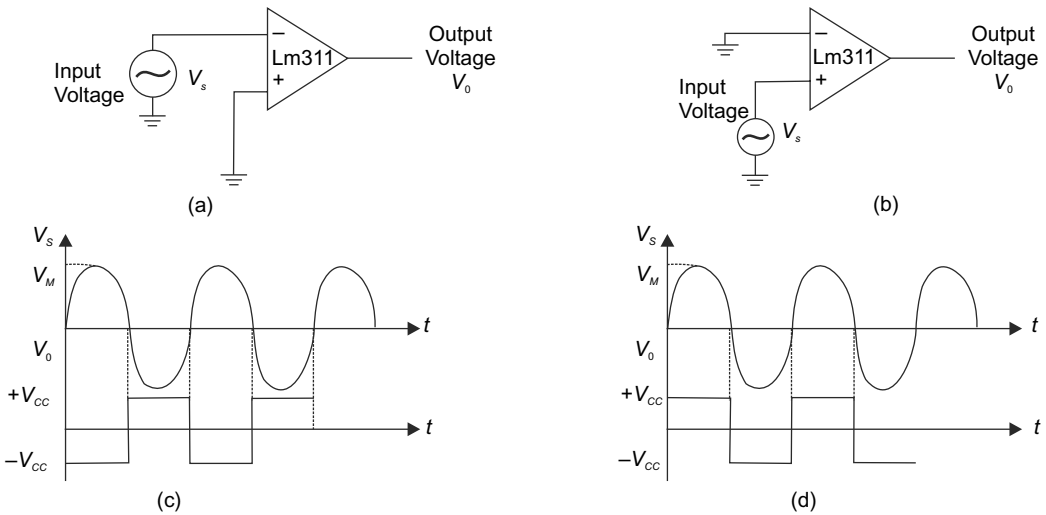


Fig. 10.18 (a) Inverting ZCD (b) Non-inverting ZCD (c) Input and output voltage of inverting ZCD (d) Input and output voltage of non-inverting ZCD

A sinusoidal signal is converted to square wave using a voltage comparator LM 311 or operational amplifier LM 747 or LM 324 as shown in Fig. 10.19. A diode is used to rectify the output signal. A potential divider is used to reduce the magnitude to 5 volts. The microprocessor receives the output signal of ZCD through I/O ports of 8255 to detect the rising edge and falling edge of square waveform using an assembly-language program. The program for zero crossing detectors to detect rising edge and falling edge of a square waveform is given below.

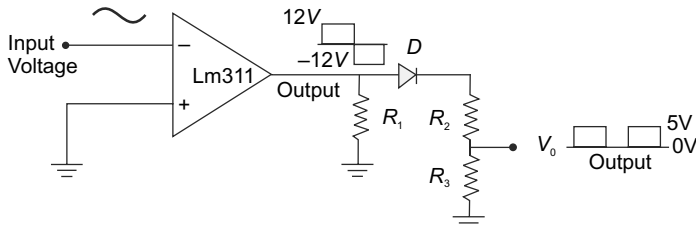


Fig. 10.19(a) Sine-wave-to-square-wave converter whose output varies between 0 to +5 V.

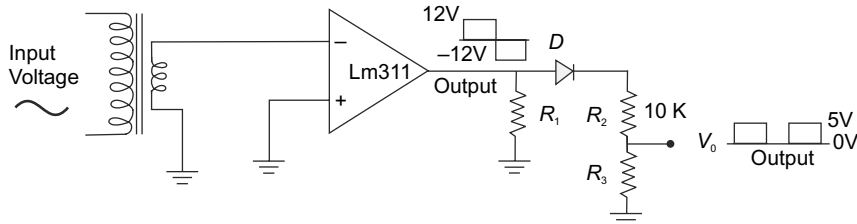


Fig. 10.19(b) Sine-wave-to-square-wave converter whose output varies between 0 to +5 V

PROGRAM 10.5 for ZCD

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	3E, 90		MVI	A, 90H	Load control word of 8255 in accumulator
8002	D3, 03		OUT	03H	Write control word in control word register and initialize ports
8004	01, 00, 00		LXI	B, 0000H	
8007	11, 00, 00		LXI	D, 0000H	
800A	DB, 00	Level-1	IN	00	Input through Port A
800C	E6, 01		ANI	01H	Logical AND 01H with accumulator
800E	C2, 0A, 80		JNZ	Level_1	
8011	DB, 00	Level_2	IN	00	Input through Port A
8013	E6, 01		ANI	01H	Logical AND 01H with accumulator
8015	CA, 11, 80		JZ	Level_2	
8018	03		INX	B	Increment B-C register pair as positive edge zero crossing is detected
8019	DB, 00	Level_3	IN	00	Input through Port A
801B	E6, 01		ANI	01H	Logical AND 01H with accumulator
801D	C2, 19, 80		JNZ	Level_3	
8020	13		INX	D	Increment DE register pair as negative edge zero crossing is detected
8021	C3, 11, 80		JMP	Level_2	
8024	76		HLT		Stop

The above program has been written to detect the rising and falling edges of a square waveform. Increment BC register pair, after detecting rising edge and increment DE register pair after detecting falling edge.

Initially, assume Port A as input and Port B and Port C as output ports. Find the control word of 8255 and load it into control word register. The ZCD output is connected to the PA₀ of 8255. The PA₀ pin will be either low or high. If it is high, it waits till the signal becomes low. As soon as it receives the low signal, the counter DE register pair will be incremented. When the signal is low, it waits till the signal becomes high. After that, the counter BC register pair will be incremented.

Figure 10.20 shows the schematic block diagram of frequency measurement, which consists of a step-down transformer, a Zero Crossing Detector (ZCD), 8255 PPI and 8085 microprocessor. The step-down transformer changes the supply voltage to 5 V signal and fed to a ZCD. The output of ZCD is a 5 V square wave, which is connected with the PA₀ of 8255 PPI.

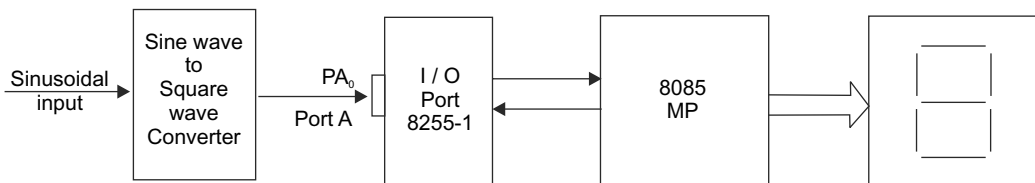


Fig. 10.20 Interface for frequency measurement

The microprocessor checks the status of PA_0 pin of 8255. At point P_1 , the $PA_0 = 0$. $PA_0 = 1$ at point P_2 . As soon as P_2 becomes 1 and PA_0 status has been changed from 0 to 1, the counter starts counting but the microprocessor checks the status of PA_0 continuously as the counter should count until PA_0 changes from 1 to 0. At point P_4 , PA_0 becomes 0 from 1 and the processor terminates the counting process. In this way, half-cycle time is measured in terms of count value, which is stored in the BC register pair. A look-up table will be stored into a microprocessor for conversion from the count value of time period to frequency. Actually, the hexadecimal count value is compared with a look-up table and frequency will be measured and displayed in data field or address field of microprocessor or in the seven-segment display units. The program for frequency measurement is given below.

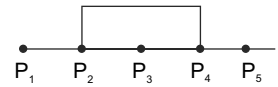


Fig. 10.21 Rectified square wave from 0 to 5V.

PROGRAM 10.6 for Frequency Measurement

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	3E, 90		MVI	A, 90H	Load control word of 8255 in accumulator
8002	D3, 03		OUT	03H	Write control word in control word register and initialize ports
8004	01, 00, 00	Start	LXI	B, 0000H	
8007	DB, 00	Level-1	IN	00	Input through Port A
8009	E6, 01		ANI	01H	Logical AND 01H with accumulator
800B	C2, 07, 80		JNZ	Level_1	
800E	DB, 00	Level_2	IN	00	Input through Port A
8010	E6, 01		ANI	01H	Logical AND 01H with accumulator
8012	CA, 0E, 80		JZ	Level_2	
8015	03	Level_3	INX	B	Increment B-C register pair as positive edge zero crossing is detected
8016	DB, 00		IN	00	Input through Port A
8018	E6, 01		ANI	01H	Logical AND 01H with accumulator
801A	C2, 15, 80		JNZ	Level_3	
801D	79		MOV	A, C	Store the content of B-C register pair in 8050H and 8051H memory location
801E	32, 50, 80		STA	8050H	
8021	78		MOV	A, B	
8022	32, 51, 80		STA	8051H	
8025	CD, 00, 81		CALL	8100	Call look-up table subroutine
8028	CD, 50, 81		CALL	8150	Call display subroutine
802B	C3, 04, 80		JMP	Start	

Subroutine for Frequency Calibration

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8100	21, 50, 80		LXI	H, 8050H	Load 8050H in HL-register pair
8103	5E		MOV	E, M	Store LSB of count value in E register;
8104	23		INX	H	increment HL register pair to locate next memory location
8105	56		MOV	D, M	Store MSB of count value in D register
8106	2E, 00		MVI	L,00	Starting memory location of look-up table 9000H is loaded in HL register pair
8108	26, 00		MVI	H,90	
810A	19		DAD	D	Add content of DE and HL register pair as well store the result in HL register pair
810B	19		DAD	D	Add content of DE and HL register pair as well store the result in HL register pair
810C	7E		MOV	A,M	Move LSDs of frequency to accumulator from lookup table
810D	32, 00, 82		STA	8200	Store two least significant digits (LSDs) of display frequency in 8200H
8110	23		INX	H	Increment HL register pair
8111	7E		MOV	A,M	Move MSDs of frequency to accumulator from lookup table
8112	32, 01, 82		STA	8201	Store two most significant digits of display frequency in 8201H
8115	C9		RET		

PROGRAM 10.7 for Display

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8150	3E, 80		MVI	A, 80H	Load control word of 8255-2 in accumulator
8152	D3, 0B		OUT	0BH	Write control word in control word register and initialize ports
8154	3A, 00, 82		LDA	8200H	Load data in accumulator from 8200H
8157	D3, 09		OUT	09	Send data to Port B for display
8159	3A, 01, 82		LDA	8201H	Load data in accumulator from 8201H
815C	D3, 08		OUT	08	Send data to Port A for display
815E	C9		RET		

✓ **Look-up-table between count value and frequency.**

Memory Address	Hex code of Count value	Frequency Display (Hz)	Memory Address	Hex code of Count value	Frequency Display (Hz)
9000	0000	0000	9300	0180	5020
9002	0001	9999	9302	0181	5007
9004	0002	9986	9304	0182	4994
9006	0003	9973	9306	0183	4981
...
92FE	017F	5033	95FE	02FF	0050

In the above program, instructions from memory location 8007H to 800CH can be used to check whether the level of ZCD output is 0 or 1. If it is in level high or '1', again execute instructions from memory location 8007 H to 800C H. Otherwise, start execution from 800E H memory location. Instructions are written in between 800E H and 8013 H to detect the rising edge of ZCD output (square wave signal). As soon as the rising edge is detected, the counter starts counting and again reads PA_0 to check the status of ZCD output. Here, the BC register pair is used as a counter and it counts continuously until falling edge of ZCD output square wave is detected. Parts of the main program from memory location 8015H to 801BH are used for this purpose. In this way, the time period for half cycle is measured in terms of count value. The count value is inversely proportional to the frequency. When the count value is low, the frequency will be high and vice versa. The count value can be converted into frequency by computation or by using a look-up table. The flowchart for frequency measurement is depicted in Fig. 10.22.

This frequency measurement program can be used to measure frequency up to 10 kHz. The look-up table between count value and frequency is given above. Here, we assume maximum count value is about 02FF and minimum count value 0000H while the program is executed. When the count value is 0001, the frequency output is 9999 Hz, which will be displayed in seven-segment display units. If the count value is 02FF, frequency output is 0050 Hz and display is in seven-segment display units.

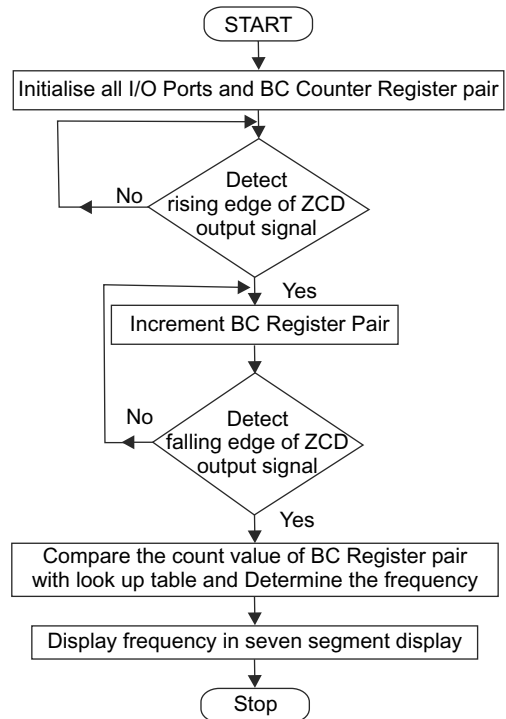


Fig. 10.22 Flowchart for frequency measurement

10.3.4 Phase-Angle Measurement

The schematic block diagram of phase-angle measurement is shown in Fig. 10.23. Assume the voltage and current waveform has a phase angle difference of ϕ as depicted in Fig. 10.24. The voltage waveform converted into square waveform using ZCD. Initially, the current signal is converted into voltage signal and then voltage signal is converted into square waveform using another ZCD. The ZCD output corresponding to voltage signal is connected with PA_0 of 8255 and the ZCD output due to current signal is also fed to PB_7 of 8255.

A program for phase-angle measurement should be loaded in the microprocessor. When the program is executed, the microprocessor firstly detects the instant of positive zero crossing of input voltage. Then the

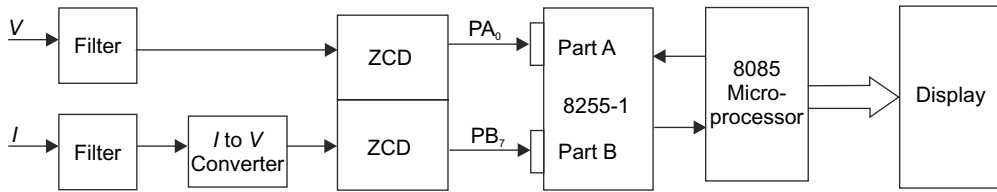


Fig. 10.23 Block-diagram interface for phase-angle measurement

counter starts counting at the instant of positive zero crossing point of square wave corresponding to voltage. After that, the microprocessor checks the status of PB_7 to detect the positive zero crossing point of current signal and the counter terminates counting as soon as zero crossing of the current waveform is detected. The count value of the counter is directly proportional to phase-angle difference. A look-up table, i.e., the relationship between count value and phase angle is already stored in the microprocessor. Therefore, using the said look-up table, the digital count value can be calibrated into phase angle and finally displayed in seven-segment display. The program for phase-angle measurement is given below:

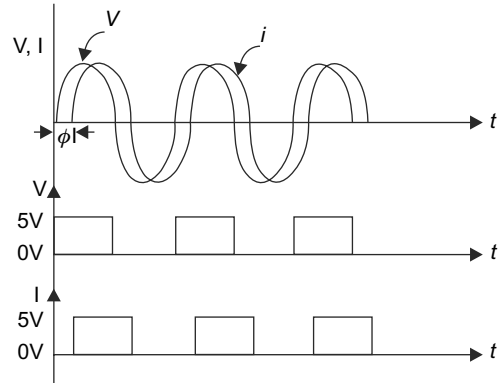


Fig. 10.24 Voltage and current waveforms and their corresponding ZCD outputs

PROGRAM 10.8 for Phase Angle Measurement

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	3E, 90		MVI	A, 90H	Load control word of 8255 in accumulator
8002	D3, 03		OUT	03H	Write control word in control word register and initialize ports
8004	01, 00, 00	Start	LXI	B, 0000H	
8007	DB, 00	Level-1	IN	00	Input through Port A
8009	E6, 01		ANI	01H	Logical AND 01H with accumulator
800B	C2, 07, 80		JNZ	Level_1	Detect the zero level of square wave of V
800E	DB, 00	Level_2	IN	00	Input through Port A
8010	E6, 01		ANI	01H	Logical AND 01H with accumulator
8012	CA, 0E, 80		JZ	Level_2	If positive zero crossing of V is not detected, jump to Level_2
8015	03	Level_3	INX	B	Increment B-C register pair
8016	DB, 01		IN	01	Input through Port B
8018	E6, 80		ANI	80H	Logical AND 80H with accumulator
801A	C2, 15, 80		JNZ	Level_3	Detect the zero level of square wave of I
801D	03	Level_4	INX	B	Increment B-C register pair

(Contd.)

(Contd.)

801E	DB, 01	IN	01	Input through Port B
8020	E6, 80	ANI	80H	Logical AND 01H with accumulator
8022	CA, 1D, 80	JZ	Level_4	If positive zero crossing of I is not detected, jump to Level_4
8025	79	MOV	A, C	Store the content of B-C register pair in 8050H and 8051H memory location
8026	32, 50, 81	STA	8050H	
8029	78	MOV	A, B	
802A	32, 51, 81	STA	8051H	
802D	CD, 00, 81	CALL	8100	Call look-up table subroutine
8030	CD, 50, 81	CALL	8150	Call display subroutine
8033	C3, 04, 80	JMP	Start	

✓ Look-up table for phase-angle measurement

<i>Memory Address</i>	<i>Hex code of Phase Angle</i>	<i>Phase Angle Display in degrees</i>	<i>Memory Address</i>	<i>Hex code of Phase Angle</i>	<i>Phase Angle Display in degrees</i>
9000	0000	00 00	9180	00C0	45 00
9002	0001	00 24	9182	00C1	45 24
9004	0002	00 47	9184	00C2	45 47
9006	0003	00 71	9186	00C3	45 71
...
917E	00BF	44 70	9180	0180	90 00

The above program for frequency measurement is written by using the following steps:

- Step 1** Initialization of I/O ports of 8255-1 and load 0000H in BC register to initialize counter.
- Step 2** Detect the instant of rising edge of square wave corresponding to input voltage.
- Step 3** After detecting the positive zero crossing of voltage signal, start counting. Counting will be continuing until positive zero crossing of current signal is detected.
- Step 4** Detect the rising-edge zero crossing of current wave and terminate counting. Count value will be stored in BC register pair and stored in 8100 and 8101 memory locations.
- Step 5** Call subroutine for calibration. The count value can be converted into phase angle. After that, phase can be displayed by display subroutine.

In this method, phase angle 0 to 90 degree lagging can be measured. To measure the leading phase angle, the program will be modified. Counting will be started at rising edge of a square wave corresponding to current signal and terminate counting at rising edge zero crossing of input voltage. Here, only the basic concept of phase-angle measurement has been incorporated for better understanding and program simplification.

10.3.5 Power-Factor Measurement

The current waveform is either lag or lead from the voltage waveform by a phase angle (ϕ). The power factor is a cosine of phase angle i.e., $\cos \phi$. Therefore, the power factor will be either lagging or leading. Figure 10.25 shows the schematic block diagram interface for frequency measurement.

To measure the power factor, first of all phase angle is measured as explained in Section 10.3.4. After measuring phase angle, the power factor can be measured. Generally, the phase angle is well-known in terms

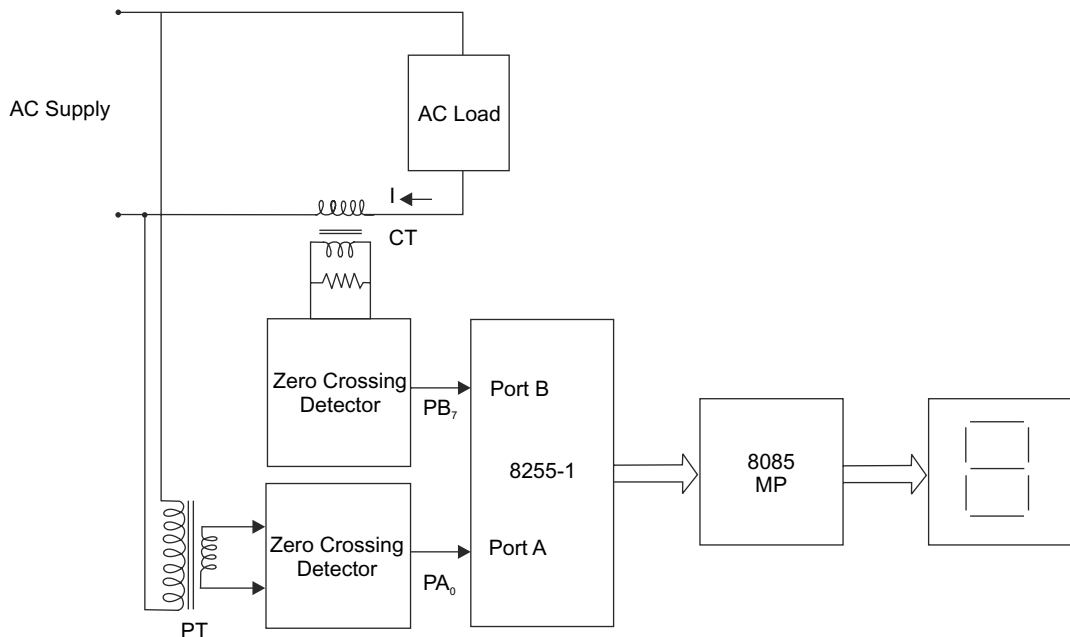


Fig. 10.25 Schematic block diagram interface for power factor measurement

of count value. Therefore digital value of the counter can be converted into power factor with the help of a look-up table and displayed in seven-segment display. Actually, the look-up table is the relationship between the count value and power factor and it is used for better accuracy and simplification of program. The program for phase-angle measurement can also be used in lagging power factor measurement after incorporating the new look-up table as given below. To measure the leading power factor, the same program can be used with some modification in the main program as well as the look-up table.

✓ Look-up table for power-factor measurement

Memory Address	Hex code of Phase Angle	Power factor Display	Memory Address	Hex code of Phase Angle	Power factor Display
9000	0000	1000	9180	00C0	0707
9002	0001	0999	9182	00C1	0704
9004	0002	0999	9184	00C2	0701
9006	0003	0999	9186	00C3	.0698
...
9080	0040	0964	9200	0100	0478
90E0	0070	0891	9260	0130	0293
9120	0090	0823	92C0	0160	0096
...
917E	00BF	0710	9180	0180	0000

10.3.6 Impedance Measurement

Impedance of a circuit (Z) is defined as the ratio of V and I . It can be expressed by $Z = \frac{V}{I}$. To measure impedance, firstly we measure rms voltage and current using a multiplexer-based A/D converter. After measuring V_{rms} and I_{rms} , the ratio $V_{\text{rms}}/I_{\text{rms}}$ can be determined by using a division subroutine. Figure 10.26 shows the schematic block diagram of impedance measurement.

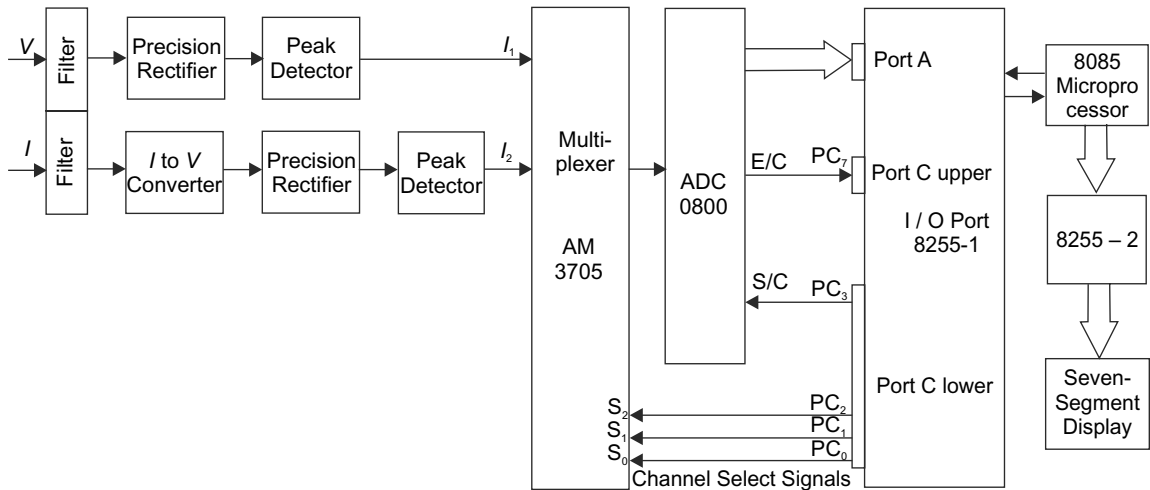


Fig. 10.26 Schematic block diagram for impedance measurement

PC_0 , PC_1 and PC_2 of Port C are connected to channel select terminals of multiplexer. Input 1 of the multiplexer is connected with voltage signal and Input 2 of the multiplexer is connected with the current signal. When PC_0 , PC_1 and PC_2 are 000, Channel 1 will be selected and analog input voltage is applied to A/D converter. Start Of Conversion (SC) pin and End Of Conversion (EOC) pin of A/D converter are connected with PC_3 and PC_7 respectively. When the microprocessor sends SC signal to A/D converter through 8255, A/D converter starts to convert analog voltage into digital form. After that, the microprocessor reads EOC signal to detect the end of A/D conversion process. After proper A/D conversion, the microprocessor reads the digital equivalent of analog input voltage and stores the digital data in a memory location. Subsequently, the microprocessor sends $PC_0 = 1$, $PC_1 = 0$, and $PC_2 = 0$ signals to select Channel 2 and analog voltage corresponding to current is fed to A/D converter. The A/D converter converts analog voltage into its digital equivalent value and stores it in the memory location. Then the microprocessor program calls a division subroutine to find the V/I ratio and displays it in the seven-segment display.

Assume dividend, i.e., digital equivalent voltage is stored in the memory location 8050H and divisor i.e. digital equivalent of current is also stored in the memory location 8051. After division, Quotient is written in 8200H and remainder in 8201H locations. In the display subroutine, the quotient will be loaded into the accumulator from 8200H and displayed as most significant digits MSDs. Similarly, the remainder can be read from memory location 8201H and loaded into the accumulator. Then remainder is displayed as least significant digits LSDs. The program for impedance measurement is given below:

PROGRAM 10.9 for Impedance Measurement

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	3E, 98		MVI	A, 98H	Load control word of 8255-1 in accumulator
8002	D3, 03		OUT	03H	Write control word in control word register and initialize ports
8004	3E, 00		MVI	A, 00H	Send $PC_0 = 0$, $PC_1 = 0$, $PC_2 = 0$ to select Channel 1
8006	D3, 02		OUT	02H	$PC_0 = 0$, $PC_1 = 0$, $PC_2 = 0$
8008	3E, 08		MVI	A, 08H	Send start of conversion signal through PC_3
800A	D3, 02		OUT	02H	PC_3 is high
800C	3E, 00		MVI	A, 00H	As PC_3 will be high for 1 or two clock pulse, make it 0
800E	D3, 02		OUT	02H	PC_3 becomes low
8010	DB, 02	LOOP	IN	02	Read end of conversion signal
8012	17		RAL		Rotate accumulator to check either conversion is over or not
8013	D2, 10, 80		JNC	LOOP	If conversion is not completed, jump to LOOP
8016	DB, 00		IN	00	Read digital output of A/D converter corresponding to voltage
8018	2F		CMA		Complement of ADC output
8019	D6, 80		SUI	80H	Subtract 80H
801B	21, 50, 81		LXI	H, 8050H	Load 8050H in HL-register pair
801E	77		MOV	M, A	Store accumulator content, i.e., digital equivalent of voltage in 8050H location
801F	3E, 01		MVI	A, 01H	Send $PC_0 = 1$, $PC_1 = 0$, $PC_2 = 0$ to select Channel 2
8021	D3, 02		OUT	02H	$PC_0 = 1$, $PC_1 = 0$, $PC_2 = 0$
8023	3E, 08		MVI	A, 08H	Send start of conversion signal through PC_3
8025	D3, 02		OUT	02H	PC_3 is high
8027	3E, 00		MVI	A, 00H	As PC_3 will be high for 1 or two clock pulse, make it 0
8029	D3, 02		OUT	02H	PC_3 becomes low
802C	DB, 02	LOOP	IN	02	Read end of conversion signal
802D	17		RAL		Rotate accumulator to check either conversion is over or not

(Contd.)

(Contd.)

802E	D2, 2C , 80	JNC	LOOP	If conversion is not completed, jump to LOOP
8031	DB, 00	IN	00	Read digital output of A/D converter corresponding to current
8033	2F	CMA		Complement of ADC output
8034	D6, 80	SUI	80H	Subtract 80H
8036	21, 51, 81	LXI	H, 8051H	Load 8051H in HL-register pair
8039	77	MOV	M, A	Store accumulator content, i.e., digital equivalent of current in 8051H location
803A	CD, 00, 81	CALL	8100	Call for division subroutine
803D	CD, 50, 81	CALL	8150	Call display subroutine
8040	86	HLT		Stop

Subroutine for V/I

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8100	21, 50, 80		LXI	H, 8050H	Address of Dividend in H-L pair.
8103	0E, 00		MVI	C, 00H	Load 00H in C register for initial value of quotient
8105	7E	LOOP	MOV	A,M	Load Dividend in accumulator from memory
8106	7A		MOV	D,A	Copy Dividend in D register
8107	23		INX	H	Increment HL register pair
8108	96		SUB	M	Subtract divisor from dividend
8109	DA, 13, 81		JC	LEVEL_1	If there is carry, jump to LEVEL_1
810C	CA, 18, 81		JZ	LEVEL_2	If there is zero, jump to LEVEL_2
810F	2B		DCX	H	Decrement HL register pair
8110	77		MOV	M,A	Store Modified Dividend in memory location from accumulator
8111	0C		INR	C	Increment C register
8112	C3		JMP	LOOP	
8113	37	LEVEL_1	STC		Clear the carry flag using set carry status and then complement the carry status
8114	3F		CMC		
8115	C3, 19, 81		JMP	LEVEL_3	Jump to LEVEL_3
8118	0C	LEVEL_2	INR	C	
8119	21, 00, 82	LEVEL_3	LXI	H, 8200H	
811C	71		MOV	M,C	Store Quotient in 8200H from C register

(Contd.)

(Contd.)

811D	23	INX	H	Increment HL register pair
811E	72	MOV	M,D	Store Remainder in 8201H from D register
811F	C9	RET		

Subroutine for Display

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8150	3E, 80		MVI	A, 80H	Load control word of 8255-2 in accumulator
8152	D3, 0B		OUT	0BH	Write control word in control word register and initialize ports
8154	3A, 00, 82		LDA	8200H	Load data in accumulator from 8200H
8157	D3, 09		OUT	09	Send Quotient to Port B for display as MSDs
8159	3A, 01, 82		LDA	8201H	Load data in accumulator from 8201H
815C	D3, 08		OUT	08	Send Remainder to Port A for display as LSDs
815E	C9		RET		

The above program for impedance measurement is written by using the following steps:

- Step 1** Initialize I/O ports of 8255.
- Step 2** Select Channel 1 and analog voltage is applied to A/D converter to convert analog voltage into digital value. Stores digital value of voltage in 8050H memory location. The part program of the above program from memory locations 8004H to 801EH is used for this purpose.
- Step 3** Select Channel 2 for current input. Convert analog equivalent voltage of current into digital value by A/D converter and store in 8051H memory location. The part program of the above program from memory locations 801FH to 8039H is used for this function.
- Step 4** Call division subroutine to determine the V/I ratio. Store the quotient in 8200H memory location and the remainder in 8201H memory locations. The division subroutine is given in the memory locations 8100H to 811FH.
- Step 5** Call display subroutine to display in seven-segment display.

10.3.7 VA Measurement

Figure. 10.27 shows the schematic block diagram of VA measurement. The VA can be expressed as

$$S = VI \text{ VA}$$

$$S = \frac{VI}{1000} \text{ kVA}$$

To measure VA, voltage V and current I are measured separately. Then the product of V and I can be computed by using a multiplication subroutine.

Precision rectifiers are used to convert ac voltage signals into dc voltages. The I to V converter is used to generate a voltage signal which is directly proportional to current. The output of rectifiers is fed to the peak detector circuit whose outputs are applied to a multiplexer as depicted in Fig. 10.27. Channel 1 and Channel

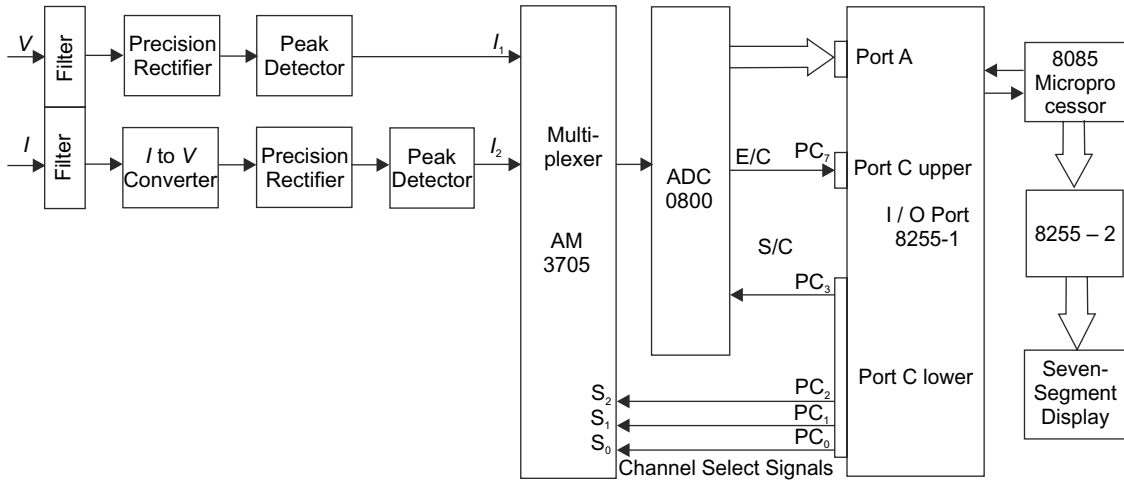


Fig. 10.27 Schematic block diagram VA measurement

2 inputs of a multiplexer are V and voltage corresponding to I respectively. Then voltage and current both are converted into digital form using A/D converter and stored in memory location. Afterwards, a multiplication subroutine is called to determine the product VI . Then it is displayed in seven-segment display.

PROGRAM 10.10 for VA Measurement

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	3E, 98		MVI	A, 98H	Load control word of 8255-1 in accumulator
8002	D3, 03		OUT	03H	Write control word in control word register and initialize ports
8004	3E, 00		MVI	A, 00H	Send $PC_0 = 0, PC_1 = 0, PC_2 = 0$ to select Channel 1
8006	DB, 02		OUT	02H	$PC_0 = 0, PC_1 = 0, PC_2 = 0$
8008	3E, 08		MVI	A, 08H	Send start of conversion signal through PC_3
800A	D3, 02		OUT	02H	PC_3 is high
800C	3E, 00		MVI	A, 00H	As PC_3 will be high for 1 or two clock pulse, make it 0
800E	D3, 02		OUT	02H	PC_3 becomes low
8010	DB, 02	LOOP	IN	02	Read end of conversion signal
8012	17		RAL		Rotate accumulator to check either conversion is over or not.
8013	D2, 10, 80		JNC	LOOP	If conversion is not completed, jump to LOOP
8016	DB, 00		IN	00	Read digital output of A/D converter

(Contd.)

(Contd.)

8018	2F		CMA		Complement of ADC output
8019	D6, 80		SUI	80H	Subtract 80H
801B	21, 50, 81		LXI	H, 8050H	Load 8050H in HL-register pair
801E	77		MOV	M, A	Store accumulator content in 8050H location
801F	3E,01		MVI	A, 01H	Send $PC_1 = 0$, $PC_1 = 0$, $PC_2 = 0$ to select channel 2
8021	D3, 02		OUT	02H	$PC_1 = 0$, $PC_1 = 0$, $PC_2 = 0$
8023	3E, 08		MVI	A, 08H	Send start of conversion signal through PC_3
8025	D3, 02		OUT	02H	PC_3 is high
8027	3E, 00		MVI	A, 00H	As PC_3 will be high for 1 or two clock pulse, make it 0
8029	D3, 02		OUT	02H	PC_3 becomes low
802B	DB, 02	LOOP-1	IN	02	Read end of conversion signal
802D	17		RAL		Rotate accumulator to check either conversion is over or not.
802E	D2, 2Bn, 80		JNC	LOOP-1	If conversion is not completed, jump to LOOP
8031	D2, 00		IN	00	Read digital output of A/D converter
8033	2F		CMA		Complement of ADC output
8034	D6, 80		SUI	80H	Subtract 80H
8036	21, 51, 80		LXI	H, 8051H	Load 8050H in HL-register pair
8039	77		MOV	M, A	Store accumulator content in 8050H location
803A	CD, 00, 81		CALL	8100	Call for multiplication
803D	CD, 50, 81		CALL	8150	
8040	76		HLT		Stop

Subroutine for Multiplication

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8100	21, 50, 80		LXI	H, 8050H	Address of multiplicand in H-L pair
8103	5E		MOV	E, M	Store multiplicand in E register from memory
8104	23		INX	H	Address of multiplicand in H-L pair
8105	56		MOV	D, M	Multiplicand in D register
8106	0E, 08		MVI	C, 08H	Load 08H in C register
8108	3A, 51, 80		LDA	8051H	Load multiplier in accumulator
810B	21, 00, 00		LXI	H, 0000H	Initial value of product = 00 in H-L pair

(Contd.)

(Contd.)

810E	0F	LOOP	RRC		Rotate accumulation left
810F	D2, 13, 81		JNC	LEVEL	If there is no carry, jump to level
8112	19		DAD	D	Add content of DE with content of H
8113	EB	LEVEL	XCHG		The content of DE register pair and HL register pair exchanged, result in DE register
8114	19		DAD	H	Multiplicand shifted one bit right
8115	EB		XCHG		The content of DE register pair and HL register pair exchanged; result in DE register
8116	37		STC		Clear the carry flag using set carry status and then complement the carry status
8117	3F		CMC		
8118	0D		DCR	C	Decrement C register
8119	C2, 0E, 81		JNZ	LOOP	If content of C register is not zero, jump to LOOP
811C	22, 00, 90		SHLD	9000H	Store the content of HL register pair in 9000H and 9001 memory location
811F	C9		RET		

10.3.8 Power Measurement

To find the power consumption in a circuit, the power is calculated from the expression

$$P = VI \cos \phi \text{ when the voltage and current waveforms are sinusoidal.}$$

Usually, the instantaneous value of the voltage is $V \cos \phi$ when current reaches the peak. To get a pulse at the moment of peak current, a phase shifter and zero-cross detector are generally used. Firstly, the current signal is fed to the phase shifter to obtain a 90° phase shift. After that the output of the phase shifter is applied to the zero-cross detector. The instant where rising edge of square wave occurs, is the peak instant of current. The microprocessor reads the status of ZCD to detect the rising edge. As soon as the microprocessor detects the rising edge of ZCD square wave output, the microprocessor generates a sample and hold signal so that $V \cos \phi$ voltage is applied to A/D converter. At this instant, the microprocessor sends a command to the multiplexer to select Channel 1, and the instantaneous value of the voltage $V \cos \phi$ is fed to an A/D digital converter. Then the A/D converter converts $V \cos \phi$ into its digital equivalent value and stores in memory location. Afterwards, find the average value of the current using the same A/D converter and store in a memory location. Subsequently, call multiplication subroutine to determine the $VI \cos \phi$ and display the power value in a seven-segment display.

10.3.9 VAR Measurement

Reactive power can be expressed as

$$VI \sin \phi = \sqrt{(VI)^2 - (VI \cos \phi)^2}$$

To measure the reactive power, initially $V \sin \phi$ and I are measured by microprocessor and then $VI \sin \phi$ is determined. At the instant of zero current, the instantaneous value of the voltage is $V \sin \phi$. To detect the instant of zero current, current signal is fed to ZCD and converted into square wave. The microprocessor reads and examines the ZCD output to detect the rising edge of square wave. When the microprocessor detects the rising edge of ZCD square wave, the microprocessor generates a sample and hold signal so that $V \sin \phi$ voltage is applied to an A/D converter. At this instant the microprocessor also sends a command to the multiplexer to select Channel 2 and the instantaneous value of the voltage, $V \sin \phi$ is fed to an A/D digital converter. A/D converter converts $V \sin \phi$ into digital equivalent value. Afterwards, the current is also converted into digital form using the same A/D converter. Subsequently, call the multiplication subroutine to determine the $VI \sin \phi$ ratio and display the reactance value in seven-segment display.

10.3.10 Energy Measurement

The electrical energy consumed by a circuit is expressed as

$$E = \int_0^t VI \cos \phi dt$$

Energy consumed by an electrical circuit = $E = VIt$ where V is voltage, I is current and t is time.

If V and I are constants, E will be a constant. The method of energy measurement is given below:

- Step 1** Determine the digital equivalent of analog voltage by an A/D converter.
- Step 2** Determine digital value of analog voltage proportional to current using an A/D converter.
- Step 3** Multiply the digital values of voltage and current and stored in memory location after summation of product value with previous value.
- Step 4** Call delay for 1minute.
- Step 5** Increment counter.
- Step 6** Repeat steps 1 to 5 until count value is equal to $(60)_D$.
- Step 7** If count value is $(60)_D$, the sum value will be energy consumed in watt hours. If we want to represent in kWh, the result must be divided by $(1000)_D$.

Assume voltage and current waveform is sinusoidal. Voltage and current are fixed for every one minute.

After summation, decimal adjustment is required for display.

The block diagram of the microprocessor-based energy measurement is shown in Fig. 10.28. In any measurement system, it is necessary to interact the system with the microprocessor through a proper sensor generating voltage signal, depending upon the system performance and compatibility with the microprocessor.

The voltage across 0.01 resistances has been taken as the reference signal, which after isolation has been conveyed to the microprocessor through ADC and I/O ports. The voltage signal is also fed to the microprocessor through ADC and I/O ports.

Software development is the most interesting feature of the device since the monitoring system is solely dependent on software. The microprocessor initially assigns I/O ports as input ports and a clearing counter m for storing units of the energy consumed, receives electrical signals in succession from the respective sensors and keeps them in specified memory locations. Then allowing a very small pre-calculated time-delay, it continues to receive current signals, adds them to the previous ones, preserves the results in the same memory location and continues the cycle operations for sixteen complete cycles.

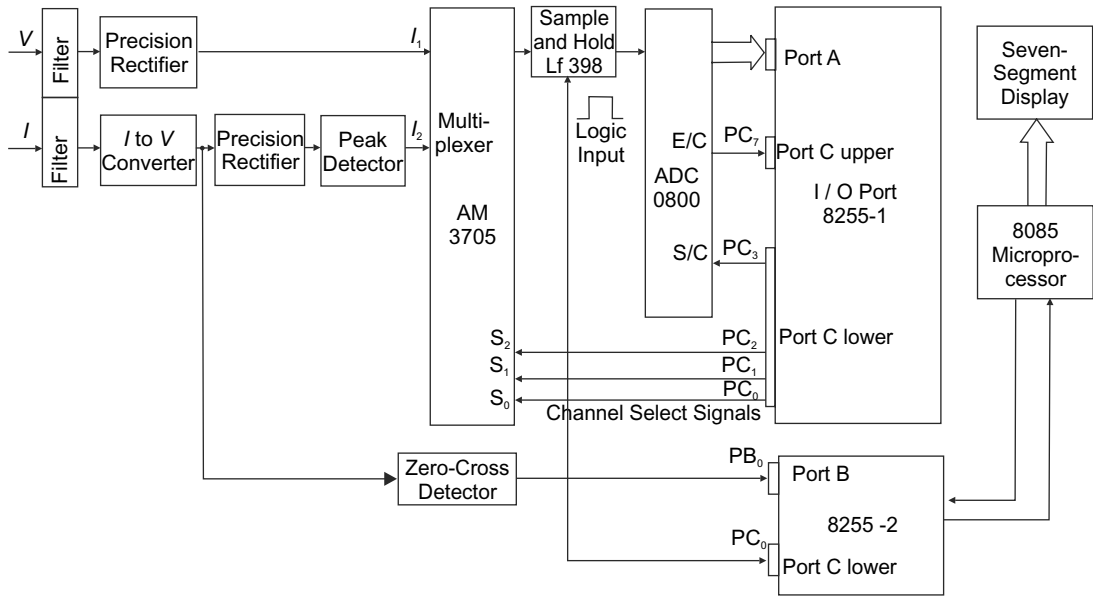


Fig. 10.28 Schematic block diagram for energy measurement

10.4 MEASUREMENT OF PHYSICAL QUANTITIES

Nowadays, physical quantities such as force, displacement, acceleration, velocity, speed, temperature, pressure, flow and level, etc., are measured and displayed using microprocessors and interfacing devices in industry. For the measurement of any physical quantity, transducers are used to convert energy from motion, displacement, acceleration, velocity, flow, pressure, level, heat, light, sound and any other physical quantities into electrical energy. A transducer consists of sensor and signal conditioning circuit. Most commonly used transducers are potentiometers, capacitive and inductive transducers, level transducers, strain gauge, accelerometer, Linear Variable Differential Transformer (LVDT), piezoelectric crystals and diaphragm, etc. Electrical output of a transducer is very small and it is not in measurable condition; therefore it should be amplified by using amplifiers. Figure 10.29 shows the schematic block diagram of a physical quantity measurement.

The output electrical signal from transducer is fed to an A/D converter, which converts analog signal to digital form and then applies to the 8085 microprocessor through 8255 PPI. The 8085 microprocessor reads this digital data and displays it in seven-segment display. When it is required to measure and display more than one physical quantity, a multiplexer should be incorporated in between transducers and the A/D converter. In

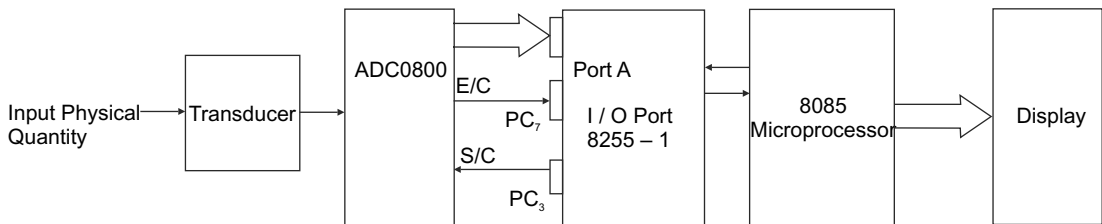


Fig. 10.29 Schematic block diagram of any physical quantity measurement

this section, the working principle of measuring displacement, strain, pressure, force, torque, speed and temperature are discussed in detail.

10.4.1 Displacement Measurement

In a displacement-measurement potentiometer, capacitive transducers, and Linear Variable Differential Transformers (LVDT) are generally used. In a potentiometer, the object moves the tap on a variable resistance and output voltage is directly proportional to displacement. Pots are used as potentiometers, shown in Fig. 10.30. In a pot, an electrically conductive wiper slides against a fixed resistor element. To measure displacement, the potentiometer is typically wired in a voltage divider configuration as depicted in Fig. 10.31. The output voltage is a function of the wiper's position and it is an analog voltage. The output voltage V_0 can be expressed as $V_0 = V_r \frac{x}{x_p}$ where V_r = the reference

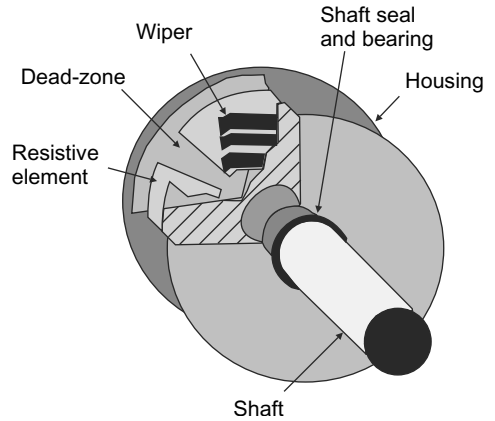


Fig. 10.30 Pot

$$= V_r \frac{x}{x_p} \text{ where } V_r = \text{the reference}$$

voltage, V_0 = output voltage, x_p = the maximum wiper position, and x = displacement. This type of resistive displacement

sensor has some advantages such as ease of use, low cost, high-amplitude voltage signal and passivity. But its disadvantages are limited bandwidth, frictional loading, inertial loading and wear. The potentiometer is commonly used in positioning of robotics like artificial limbs and servo systems.

In capacitive displacement transducer, one plate of the capacitor is mounted to a fixed surface and the other plate mounted to the object. With the position of object, capacitance value changes. The capacitive displacement sensor generates an output signal due to change in capacitance. The capacitance is a function of distance (d) between the electrodes, the surface area (A) of the electrodes, and the permittivity ϵ as given below:

$$C = \epsilon \frac{A}{d} = \epsilon_0 \epsilon_r \frac{A}{d} \text{ where } \epsilon_0 \text{ is permittivity of air, and } \epsilon_r \text{ is the relative permittivity. The change in capacitance due to change in distance is } \Delta C = \epsilon_0 \epsilon_r \left(\frac{A}{d} - \frac{A}{d + \Delta d} \right)$$

Capacitor sensors are variable-distance displacement sensors, variable-area displacement sensors, and variable-dielectric displacement sensors as depicted in Fig. 10.32 (a), (b) and (c) respectively.

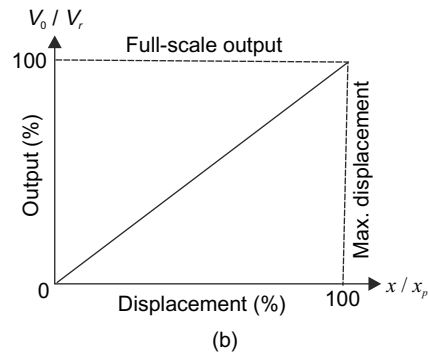
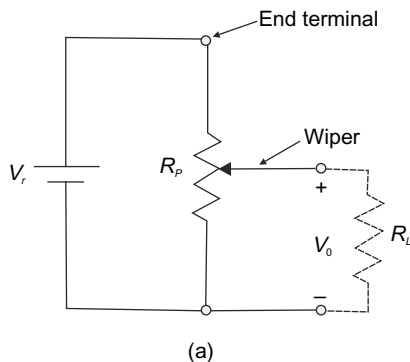


Fig. 10.31 (a) Potentiometer (b) Linear output function

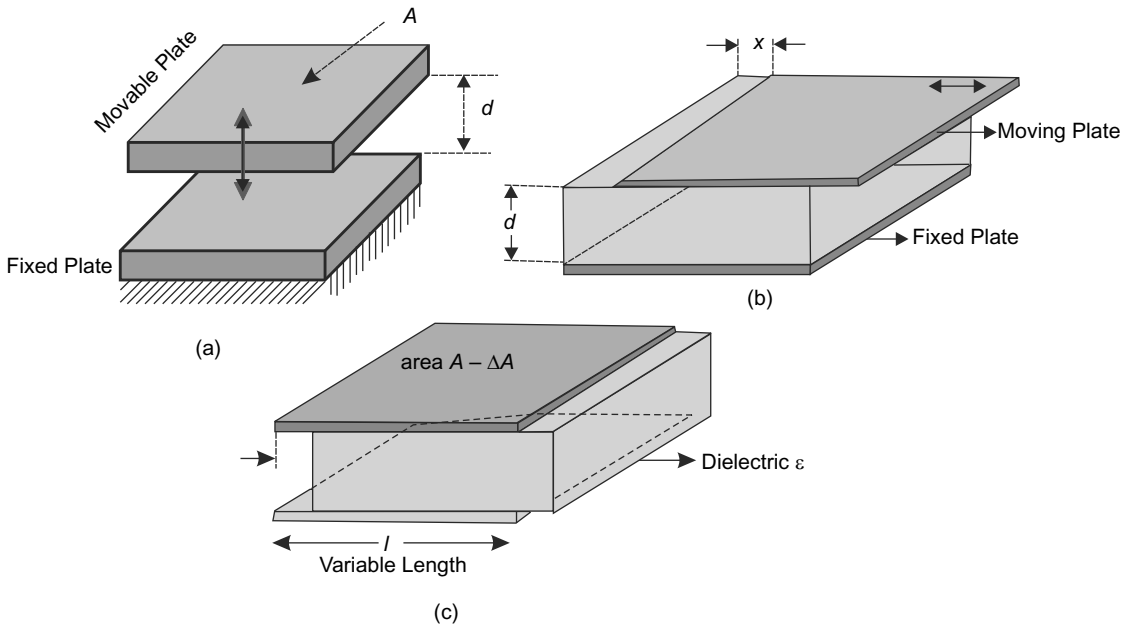


Fig. 10.32 Capacitor sensors

Capacitor value in variable-area displacement sensors is $C = \epsilon_0 \epsilon_r \frac{A - wx}{d}$ where w width, wx the reduction in the area due to movement of the plate. In variable dielectric displacement capacitive sensors, $C = \epsilon_0 w[\epsilon_0 l - (\epsilon_2 - \epsilon_1)x]$ where ϵ_2 is the permittivity of the displacing material, and ϵ_1 is the relative permittivity of the dielectric material. Generally, a capacitive transducer can be placed in a bridge circuit and ac voltage is connected across the bridge. Then the bridge output voltage is amplified, rectified and measured.

A Linear Variable Differential Transformer (LVDT) is a three-coil inductive transducer and object moves core of three winding. A three-winding transformer (one primary and two secondary) with a movable core is shown in Fig. 10.33. It is a passive inductive transducer. The two secondaries are having equal sizes, shapes and number of turns. Primary winding of transformer is supplied by 1–10 V, 50 Hz –25 kHz ac signal. Each secondary winding covers one half of transformer, secondaries connected to oppose each other and the object is connected to the core.

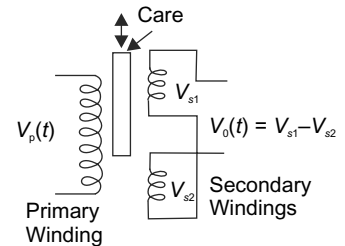


Fig. 10.33 LVDT

The mutual inductance between the primary and secondary windings is changed with the change in position of a high permeability rod. The induced voltages at secondaries are $V_{s1}(t) = K_1 V_p(t)$, $V_{s2}(t) = K_2 V_p(t)$. As secondary windings are connected in series opposition, the output voltage

$$V_0(t) = V_{s1}(t) - V_{s2}(t). \text{ As the rod moves from the centre, } K_1 \text{ increases while } K_2 \text{ decreases.}$$

When the core is centered, voltages at two secondaries are equal and the output voltage is zero. While core is off center, voltage at one secondary is higher than other one. Thus output voltage is linearly related to core position as depicted in Fig. 10.34.

The model function of LVDT is $V_0(x, t) = KV_p(t)$ where K is a constant, t is time, $V_p(t)$ is the primary voltage, x is displacement and it will be either positive or negative. If x is negative, phase of output voltage is reversed.

The schematic block diagram of the displacement or deflection measurement is shown in Fig. 10.35. LVDT is used to sense the deflection of a beam as the movable core of the linear variable differential transformer is connected to the beam. When the core is in centered position, the voltages induced in two secondaries of the LVDT are equal. Hence output voltage will be zero. While the core is moved in upward or downward directions, the voltage induced in two secondaries will not be equal and output voltage is equal to the difference between secondary induced voltages as expressed by $V_0 = V_{S1} - V_{S2}$. This output voltage V_0 is directly proportional to the displacement of core. The output voltage of LVDT is low and in the range of 100–500 mV. Therefore, an amplifier should be used to amplify LVDT output and fed to a precision rectifier for rectification. Then precision rectifier output voltage is applied to A/D converter for analog to digital conversion. The digital output of the A/D converter is fed to Port A of 8255-1 as depicted in Fig. 10.35 (a).

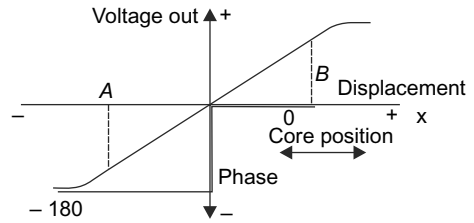


Fig. 10.34 Plot of output voltage against core position

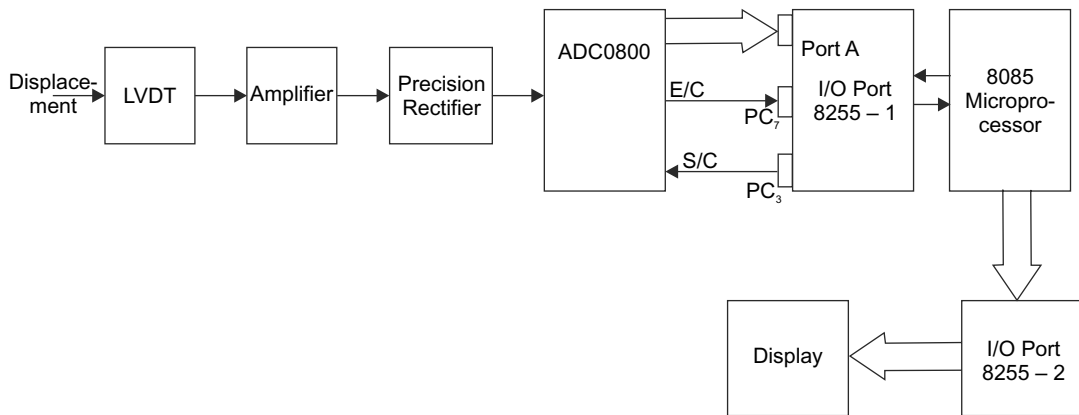


Fig. 10.35(a) Schematic block diagram of the displacement measurement

A look-up table between the digital output of the A/D converter and displacement/deflection is stored in the memory of the microprocessor. By using a look-up table, the microprocessor measures deflection for a particular A/D converter digital output and displays the same on seven-segment display through 8255-2 PPI IC. The desirable characteristics are low wear, less repeatability error, high speed and ability to measure small displacements of approximately 0.04 mm. This method can also be used for vibration measurement. The undesirable characteristics are complex conditioning circuit and expense. The program flow chart to measure displacement is illustrated in Fig. 10.35(b).

PROGRAM 10.11 for Displacement Measurement

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	3E, 98		MVI	A,98H	Load control word of 8255-1 in accumulator <i>(Contd.)</i>

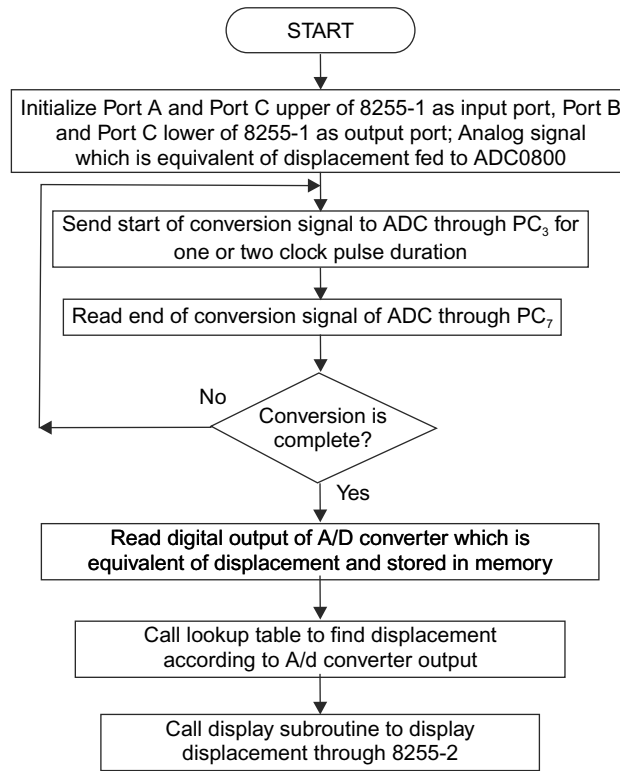


Fig. 10.35(b) Flow chart to measure displacement

(Contd.)

8002	D3, 03	OUT	03H	Write control word in control word register and initialize ports
8004	3E, 08	MVI	A, 08H	Send start of conversion signal through PC ₃
8006	D3, 02	OUT	02H	PC ₃ is high
8008	3E, 00	MVI	A, 00H	As PC ₃ will be high for one or two clock pulse, make it 0
800A	D3, 0A	OUT	02H	PC ₃ becomes low
800C	DB, 02	LOOP	IN 02	Read end of conversion signal
800E	17	RAL		Rotate accumulator to check either conversion is over or not
800F	D2, 0C, 80	JNC	LOOP	If conversion is not completed, jump to LOOP
8012	DB, 00	IN	00	Read digital output of A/D converter
8014	2F	CMA		Complement of ADC output
8015	D6, 80	SUI	80H	Subtract 80H
8017	21, 00, 81	LXI	H, 8050H	Load 8050H in HL-register pair
801A	77	MOV	M, A	Store accumulator content in 8050H location

(Contd.)

(Contd.)

801B	CD, 00, 81	CALL	8100	Call lookup table to determine displacement
801E	CD, 50, 81	CALL	8150	Call display subroutine for display
8021	76	HLT		Stop

Subroutine for calibration look-up table to find displacement and stored in memory location 8200 and 8201

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8100	21, 50, 80		LXI	H, 8050H	Load 8050H in HL-register pair
8103	3E, 00		MVI	A, 00H	Store 00H in accumulator and C register
8105	0E, 00		MVI	C, 00H	
8107	86		ADD	M	Add content of memory (digital equivalent of analog input voltage) with accumulator
8108	86		ADD	M	Add content of memory with accumulator
8109	D2,0E, 81		JNC	LEVEL-1	If no carry, jump to Level-1
810C	0E, 01		MVI	C,01	Load 01H in C register
810E	6F	LEVEL-1	MOV	L,A	Move accumulator content into L register
810F	26, 90		MVI	H, 90H	Load 90H in H register
8111	79		MOV	A, C	Content of C register with H register
8112	84		ADD	H	
8113	7E		MOV	A, M	Move content of memory location specified by HL register pair into accumulator
8114	32, 00, 82		STA	8200	Store two least significant digits of displacement in 8200H
8117	24		INX	H	Increment the HL register pair
8118	32, 00, 82		STA	8201	Store two most significant digits of displacement in 8201H
811B	C9		RET		

Subroutine for Display

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8150	3E, 80		MVI	A, 80H	Load control word of 8255-2 in accumulator
8152	D3, 0B		OUT	0BH	Write control word in control word register and initialize ports
8154	3A, 00, 82		LDA	8200H	Load data in accumulator from 8200H
8157	D3, 09		OUT	09	Send LSDs of displacement to Port B for display

(Contd.)

(Contd.)

8159	3A, 01, 82	LDA	8201H	Load data in accumulator from 8201H
815C	D3, 08	OUT	08	Send MSDs of displacement to Port A for display
815E	C9	RET		

✓ **Look-up Table for Hex code of displacement and display of displacement**

Memory Address	Hex code of Displacement	Displacement Display (mm)	Memory Address	Hex code of Displacement	Displacement Display (mm)
9000	00	00 00	9080	40	02. 56
9002	01	00.04	9082	41	02. 60
9004	02	00.08	9084	42	02. 64
...
9032	19	01.00	90B2	59	03.56
...
9064	32	02.00	90E4	72	04.56
...
907E	3F	02.52	9100	80	05. 00

10.4.2 Strain Measurement

Strain is the change in shape of an object due to some force. Assume an object in two conditions: with and without a force applied. When an external force is applied along a dimension, there will be some deformation in the object.

Let L_1 be the length of the object along the dimension when no force is applied and L_2 be the length when the force is applied. Then the object's strain is $\frac{L_2 - L_1}{L_1} = \frac{\Delta L}{L_1}$ where $\Delta L = L_2 - L_1$ change in length.

A strain transducer is used for strain measurement. Strain gauge is a strain transducer and is used to measure strains and stresses in any structures. A strain gauge is a flexible card with strip of some copper–nickel alloy conductor wires arranged in special pattern as shown in Fig. 10.36. The grids of fine wires forming a strain gauge are cemented to a thin paper membrane. The strain gauge is mounted on the object being measured. A strain-gauge conductor is usually made of metal or semiconductor. The pattern is chosen in such a way that the conductor maintains an almost constant volume with strain. That is, the conductor is not compressible. The resistance of a conductor is $R = \rho \frac{L}{A}$, where L is its length, A is its area, and ρ is its resistivity.

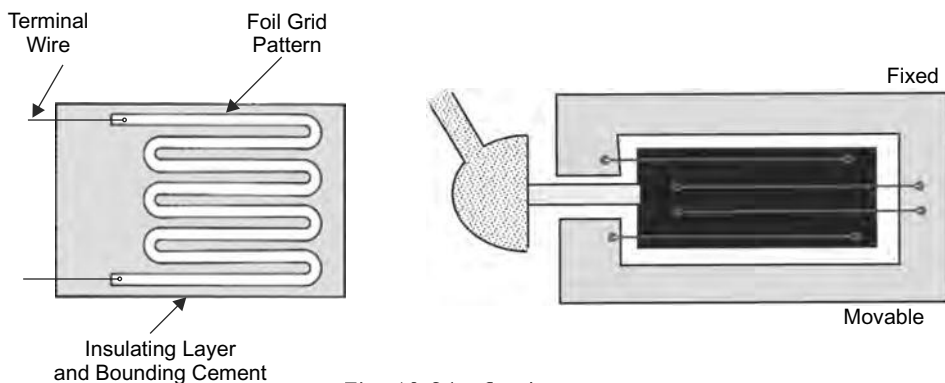


Fig. 10.36 Strain gauge

Assume force causes length of the conductor to decrease. Since volume does not change much, area must increase. Thus, resistance decreases. The model function of resistance is equal to $R_1 = R_0 (1 + G_f \varepsilon)$ where G_f is a constant and known as the *gauge factor* which is the ratio $\frac{\Delta R/R}{\Delta L/L}$ and may be considered as the sensitivity of the sensor, R_0 is resistance without strain, R_1 is resistance due to strain, and ε is strain.

For metal-wire strain gauges (constantan), $G_f = 2$ while semiconductor strain gauges have much higher G_f of about 200. Bonded strain gauges have folded wires bonded to a semiflexible backing material, with unbonded gauges having flexible wires connected between fixed and movable frames as shown in Fig. 10.36.

The sensitivity of a strain gauge is very low which is approximately 1% over full operating range. But it is very important that the above change must be accurately detected.

To increase the sensitivity, two and more active sensor elements can be used in a bridge circuit. In strain measurement, a Wheatstone bridge is used a device, which can read a difference voltage directly. The output voltage of the Wheatstone bridge as shown in Fig. 10.37 is expressed as

$$V_0 = \left(\frac{R_4}{R_2 + R_4} - \frac{R_3}{R_1 + R_3} \right) V_B$$

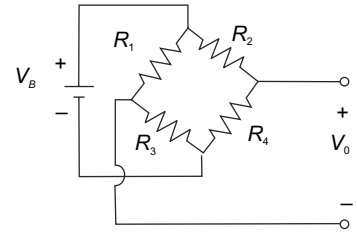


Fig. 10.37 Wheatstone bridge

The sensor bridge usually consists of four identical sensor elements. Assume that only one of these is sensitive to the strain, which can be measured, and other sensors are ‘dummy’ sensors. For example, for a maximum of 1% change in resistance, the output voltage is about $0.0025V_B$. This is approximately 2.5 mV for a 10 V supply. Therefore, the range of V_0 in this case is 0–2.5 mV.

$$V_0 = \left(\frac{R + \Delta R}{2R + \Delta R} - \frac{R}{2R} \right) V_B = \frac{\Delta R}{2(2R + \Delta R)} V_B \cong \frac{\Delta R}{4R} V_B$$

Increased measurement accuracy and possibility for increased sensitivity are shown in Fig. 10.39 and Fig. 10.40 respectively. Elimination of ‘noise’ effects on a sensor output; i.e. if a sensor is sensitive to changes in both temperature and strain, if 4 identical elements are used:

$$V_0 = \left(\frac{R + \Delta R - R}{2R + \Delta R} \right) V_B \cong \frac{\Delta R}{2R} V_B$$

with only one of them subjected to the strain, the temperature effect is cancelled. The output voltage can be expressed

- ◆ for two Strain gauges in Wheatstone bridge, and
- ◆ for four Strain gauges in Wheatstone bridge

$$V_0 = \left(\frac{R + \Delta R}{2R} - \frac{R - \Delta R}{2R} \right) V_B = \frac{\Delta R}{R} V_B$$

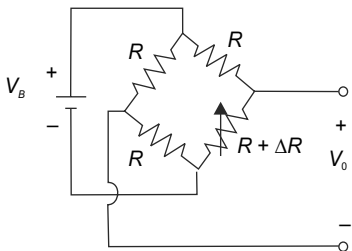


Fig. 10.38 One strain gauge in Wheatstone bridge

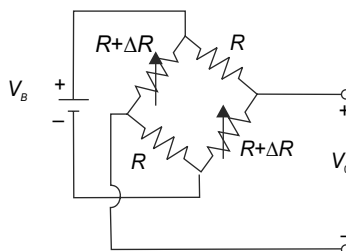


Fig. 10.39 Two strain gauges in Wheatstone bridge

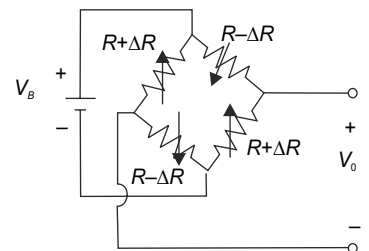


Fig. 10.40 Four strain gauges in Wheatstone ridge

In some cases, the strain in two places on the object will be of equal magnitude but of opposite sign. For example, a cantilever beam is as shown in Fig. 10.41. The upper part of the beam is stretched (positive strain) and the lower part of the beam is compressed (negative strain). The two strain gauges, therefore, form complementary pairs.

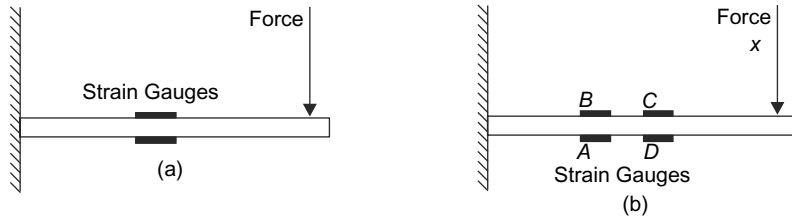


Fig. 10.41(a) Complementary pairs of strain gauge (b) Four strain gauges

Physically, a strain gauge is not much different from an RTD and so, a strain gauge is affected by temperature. Hence, temperature compensation is required. The model function including temperature.

$R_1 = R_0(T)(1 + G_f \epsilon)$, where $R_0(T)$ is a function of temperature. Conditioning circuit can remove $R_0(T)$. A bridge circuit does this very well.

For a complimentary pair, $R_1 = R_0(T)(1 \pm G_f \epsilon)$

Four strain gauges are placed in the bridge form as shown in Fig. 10.42. The temperature terms will be canceled and the output voltage $V_0 = AV_E G_f \epsilon$, where $A =$ gain of amplifier, $V_E =$ input voltage, $G_f =$ gauge factor, and $\epsilon =$ strain.

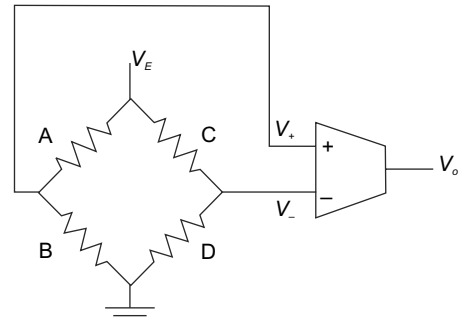


Fig. 10.42 Bridge circuit with amplifier

Figure 10.43 shows the block diagram for strain measurement when two gauges are mounted to a cantilever beam. Due to change in resistance, bridge output voltage changes and its magnitude is directly proportional to strain. Then output of the bridge is fed to a instrumentation amplifier and amplified to a certain voltage in the range of 0–5 V so that it can be processed by the microprocessor. Output is 0 V for no strain and 5 V for the maximum strain. When four strain gauges are mounted in a cantilever beam, the output voltage increases two times. A look-up table between the hex code of digital voltage and strain is stored in memory. After converting the analog voltage into digital form through an A/D converter, find the strain from the look-up table and display it in seven-segment display unit. The program for displacement measurement may also be used in strain measurement with the modification in look-up table only.

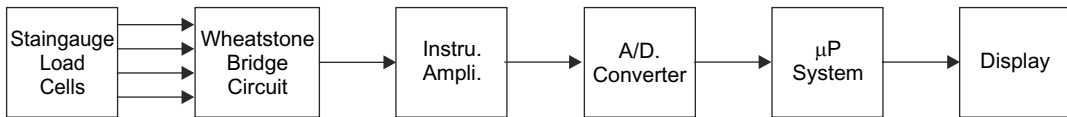


Fig. 10.43 The block diagram for strain measurement

Assume the dimension of cantilever beam Length $L = 22$ cm, Breadth $B = 2.8$ cm and thickness $T = 0.3$ cm and Young’s modulus Y of stainless steel $= 2.1 \times 10^6$. The strain is calculated by the following expression:

$$\epsilon = \frac{6WL}{BT^2Y}$$

where $W =$ the weight applied at the end of the cantilever beam

If $W = 1$ kg, the strain is about 248. When load variation is 100 g to 1 kg, the output voltage of strain gauge in mV is given in Table 10.5.

Table 10.5 Strain gauge output voltage and its digital equivalent

Weight in gm	Output voltage in mV	Amplified Voltage in Volts	Digital Equivalent of Voltage
100	25	0.50	0C
200	50	1.00	19
300	74	1.48	25
400	100	2.00	33
500	123	2.46	3E
600	149	2.98	4C
700	175	3.50	59
800	201	4.02	66
900	222	4.44	71
1000	248	4.96	7E

✓ Look-up table between Hex code of strain and strain display

Memory Address	Hex code of strain	Strain Display	Memory Address	Hex code of strain	Strain Display
9018	0C	0025	9098	4C	0149
...
9032	19	0050	90B2	59	0175
...
904A	25	0074	90CC	66	0201
...
9066	33	0100	90E2	71	0222
...
907C	3E	0123	90FC	7E	0249

10.4.3 Force Measurement

There are many types of sensors which can be used to measure force. Resistance-type force sensors, such as gauges and load cells, are very commonly used in force measurement. The force can be measured by the following ways:

Elastic sensing	$F = E.x$
Strain sensing	$F = \sigma.A$
Pressure sensing	$F = P.A$
Acceleration sensing	$F = m.a$

Force can move a part of the transducer. This movement can be measured using displacement sensors.

Piezoelectric crystals are also used in force measurement. A piezoelectric crystal consists of a crystal of a material with piezoelectric properties, i.e., a piezoelectric material emits charge when compressed. The material may be quartz, or special ceramics. Contacts are always placed along two faces of the crystal.

When force or pressure is applied to the crystal, a charge appears on the surface of the crystal and the amount of charge directly proportional to the force. Therefore, the output of the transducer is charge. The output of the crystal is converted to voltage using a capacitor. The voltage generated from piezoelectric crystals is V , which can be expressed as

$$V = \frac{Q}{C}$$

The capacitor should be chosen in such a way, so as to get the desired voltage range. The capacitor

voltage is fed to very high input impedance amplifier for amplification, and amplifier output is applied to A/D converter for analog-to-digital conversion.

When an unchanging force is applied, the voltage will decrease over time. Therefore, piezoelectric crystals are best used for measuring changes in force and vibration.

A *load cell* is most commonly used to measure mechanical force. Strain gauges are called load cells and normally used for force measurement. The force bends, compresses, or stretches a part of the transducer and change in shape is usually measured using strain gauges. Two common load-cell configurations are illustrated in Fig. 10.45. Usually, load cells are sold including a bridge circuit.

Resistance of a strain gauge increases if it is stretched. Strain gauges are cemented over the mechanical structure whose deformation under the influence of stress is to be measured. Figure 10.46 shows a cantilever beam with four strain gauges. The force is applied at a predetermined point. Strain gauges are placed at locations chosen so that their output is linearly related to force. The choice of location for the strain gauges and the derivation of the resulting load-cell model function are beyond the scope of this book. Load cells are usually packaged with strain gauges connected in bridge configuration. Strain gauges 1 and 2 are mounted so that after applying load, they come under tension. Similarly, strain gauges 3 and 4 will be under compression under loaded condition. Strain gauges are normally used in a full bridge to give the bridge output proportional to the applied force. To maximise the bridge sensitivity, the strain gauge is connected in a bridge. Under loaded conditions, resistance of strain gauges 1 and 2 increases and 3 and 4 decreases. Therefore, the potential at Point *A* of the bridge will be elevated much as compared to *B*. As all four gauges are at the same temperature, this system also provides temperature compensation. Strain gauges are bonded in such a way that can provide the maximum output deformation ratio. The strain gauges are wired in full-bridge configuration for temperature compensation and for better accuracy of measurement. The complete assembly must be housed within a protective case and properly sealed so that the external environment cannot affect strain gauges though strain gauges are capable to deform after application of the force.

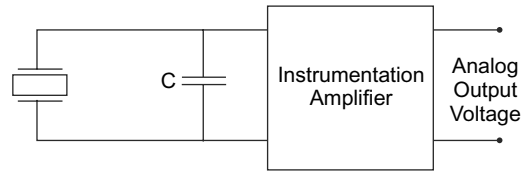


Fig. 10.44 Piezoelectric crystals in force measurement

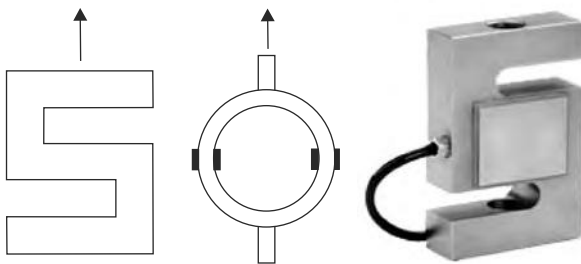


Fig. 10.45 Load-cell configurations

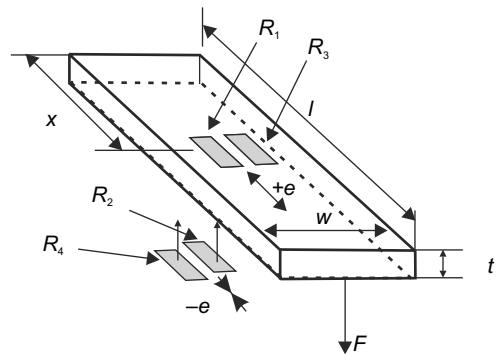


Fig 10.46 Cantilever beam with four strain gauges

When strain gauges are used in cantilever type load cells, the strain can be expressed as $e = \frac{6(l-x)}{wt^2E} F$ where F = force, l = length, w = width, t = thickness, and E Young's modulus.

The output voltage of the bridge will be $V_0 = \frac{V_s R_2 R_3}{(R_2 + R_3)^2} \left[\frac{\Delta R_3}{R_3} + \frac{\Delta R_1}{R_1} - \frac{\Delta R_2}{R_2} - \frac{\Delta R_4}{R_4} \right]$

The block diagram for force measurement is shown in Fig. 10.48. The program for displacement measurement can be used in force measurement, but there will be some modification in the look-up table. A look-up table between the hex code of digital voltage corresponding to force is stored in memory. The 8085 microprocessor reads the digital output of A/D converter for a force input and determines the force from the look-up table and display it in seven-segment display unit.

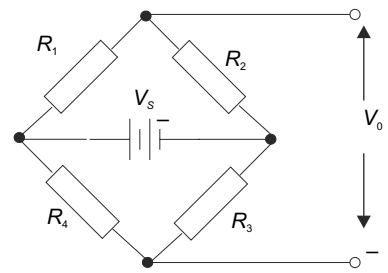


Fig 10.47 Bridge Configuration

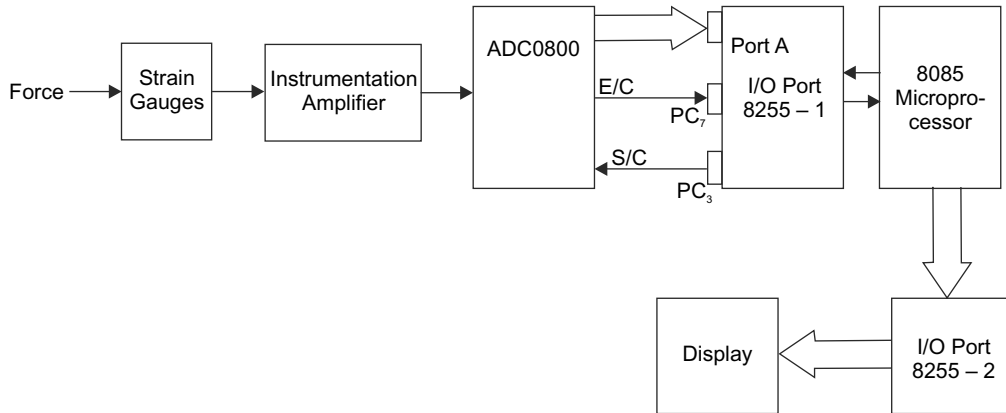


Fig. 10.48 Schematic block diagram of force measurement

✓ **Look-up table of force measurement**

Memory Address	Hex code of Force	Force Display in g	Memory Address	Hex code of Force	Force Display in g
9018	0C	0100	9098	4C	0600
...
9032	19	0200	90B2	59	0700
...
904A	25	0300	90CC	66	0800
...
9066	33	0500	90E2	71	0900
...
907C	3E	0500	90FC	7E	1000

10.4.4 Torque Measurement

Generally, torque is transmitted through a rotating shaft between a power source and a power sink. Strain gauges are commonly used in torque cells. Figures 10.49 (a) and (b) show the torque measurement using

strain gauges. Here, four strain gauges are mounted on the shaft. Strain gauges 1 and 3 are compressed, but strain gauges 2 and 4 are under tension due a torque in the shaft. The strain of the strain gauge 1 is approximately

$$e_1 = \frac{T}{\pi.G.r^3}$$

where T = torque, $G = \frac{E}{2(1 + \nu)}$ = shear modulus, r = radius of shaft.

The relationship between strains of all four gauges is $e_2 = e_4 = -e_1 = -e_3$. When all four gauges are connected in bridge form as shown in Fig. 10.50, the output voltage can be expressed as $\frac{V_0}{V} = G_f \cdot e_1 = \frac{G_f \cdot T}{\pi \cdot G \cdot r^3}$

The output of bridge circuit fed to a differential amplifier using an operational amplifier as shown in Fig. 10.50, and output voltage becomes measurable. The microprocessor can be used to measure this voltage and display it in seven-segment display after proper calibration based on a look-up table.

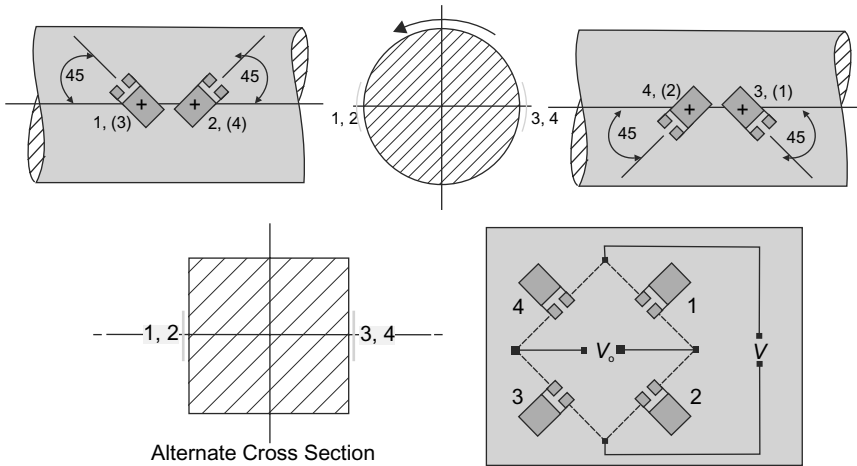


Fig. 10.49(a) Torque measurement using strain gauges

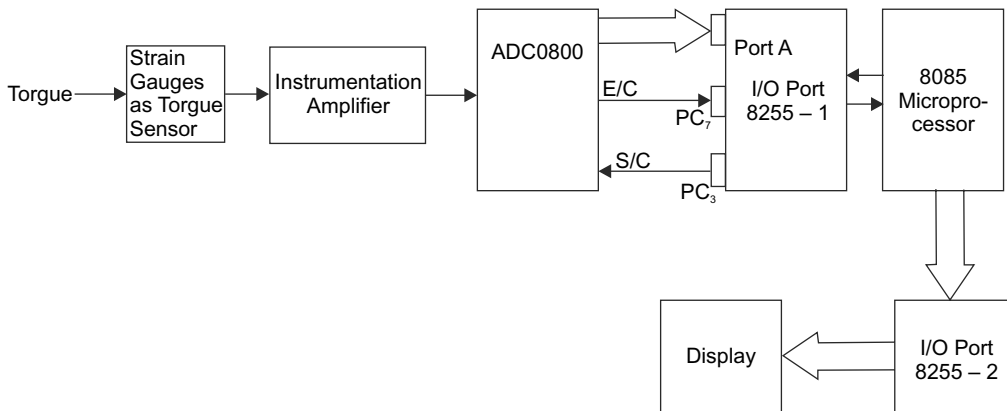


Fig. 10.49(b) Torque measurement using strain gauges

$$R_1 = R_2$$

$$R_3 = R_F$$

Gain: $R_F = R_1$

10.4.5 Pressure Measurement

Pressure is the force per unit area. Pressure is usually specified as a difference between the process variable measured and some reference pressure. This can be visualized as the pressure on a diaphragm, with the pressure being measured on one side and the reference pressure on the other. That reference pressure defines the type of pressure measured:

Absolute: Reference pressure is zero.

Gauge: Reference pressure is the environment air pressure. Automobile tire pressure is gauge pressure.

Differential: The reference pressure is a second process variable being measured.

Types of pressure transducers are large-displacement transducers and small-displacement transducers. Large-displacement transducer consists of a variety of flexible containers that change size with pressure. Small-displacement transducers usually consist of a diaphragm and a strain gauge.

Small-displacement pressure transducers consist of a diaphragm, one side exposed to pressure being measured and the other side exposed to reference pressure. Displacement of a diaphragm measured by a capacitive or inductive displacement transducer or strain of diaphragm measured using strain gauges. The placements of strain gauges are shown in Fig. 10.51. Center of the diaphragm is convex, and a part near the edge is convex. Strain gauges can be placed so that there are complementary pairs.

Nowadays integrated pressure sensors are available in the market and are used as a small-displacement pressure transducer. The entire sensor is fabricated on one silicon chip and the diaphragm is etched into silicon. Strain gauges are fabricated on silicon and signal conditioning circuit is present on the same chip. The schematic block diagram for pressure measurement is shown in Fig. 10.52. The programming of pressure measurement will be same as force measurement, but the look-up table must be modified as per required calibration.

10.4.6 Temperature Measurement

Temperature is widely measured and controlled in industrial process control system. For temperature measurement, one of the following devices are used:

<i>Device Name</i>	<i>Temperature range</i>
Resistance thermometers	-100°C to + 300°C
Platinum Resistance thermometers	-0°C to 700°C
Thermocouples	-250 °C to + 2000°C
Thermistors	-100°C to + 100°C)
Pyrometers	+ 100°C to + 5000°C

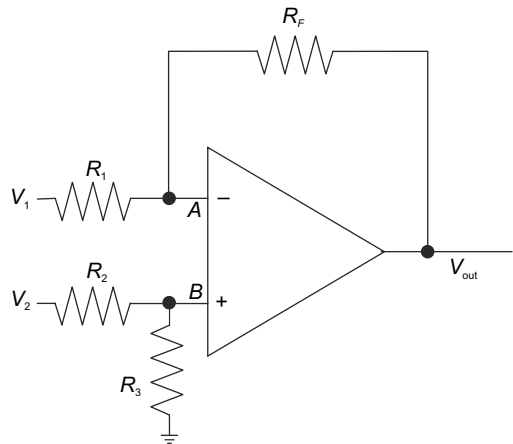


Fig. 10.50 Bridge configuration

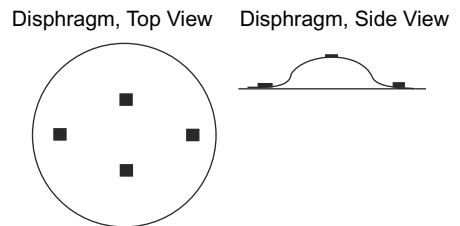


Fig. 10.51 Placement of strain gauges

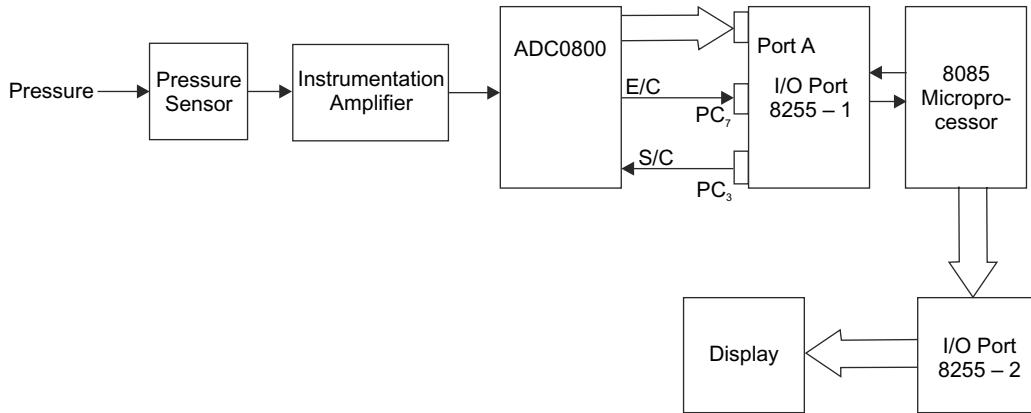


Fig. 10.52 *Schematic block diagram of pressure measurement*

Platinum wires are frequently used in resistance thermometers for industrial application because of their greater resolution, and mechanical and electrical stability as compared to copper or nickel wires. A change in temperature causes a change in resistance. The resistance thermometer is placed in an arm of a Wheatstone bridge to get a voltage proportional to temperature. A thermistor is a semiconductor device fabricated from a sintered mixture of metal alloys, having a large negative temperature coefficient. A thermistor is used in a Wheatstone bridge to get a voltage proportional to temperature. The thermistor is a thermally sensitive variable resistor made of semiconductor material. The substance used may be oxides of nickel, copper, manganese, iron, cobalt, etc., usually a high negative temperature coefficient. It can be used in the range of -100 to $+100^{\circ}\text{C}$ for greater accuracy as compared to a platinum resistance thermometer. Positive thermistors are also used but in the low range of 50°C to $+100^{\circ}\text{C}$.

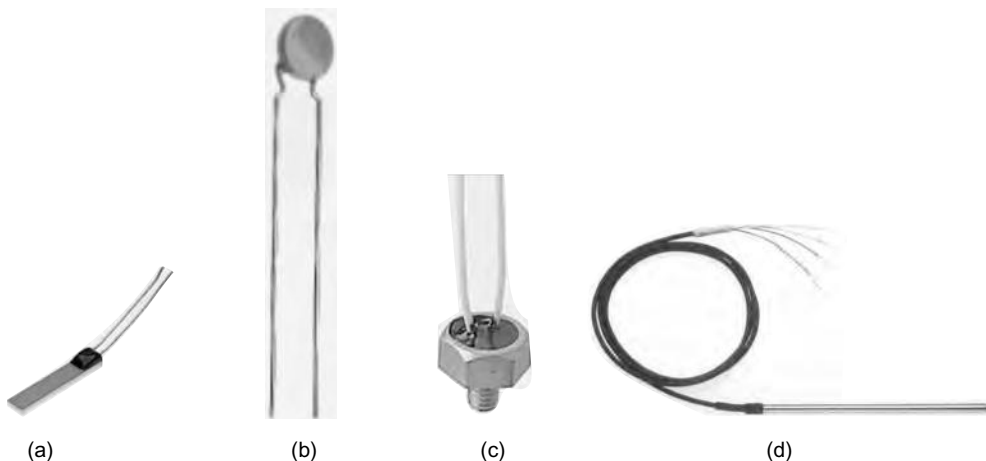


Fig. 10.53 (a) *Typical platinum resistance thermometer (PRT)* (b) *Disk thermistor*
(c) *Threaded thermistor* (d) *Sheathed PRT*

In industry, the most widely used temperature transducer is the *thermocouple*. This temperature transducer works on the principle that contact potential between two dissimilar metals changes with temperature. When two dissimilar metals are joined and the junctions are placed at two different temperatures, an emf is induced which will be used for temperature measurement. Thermocouple materials for different ranges of temperature are given below:

Material	Temperature Range
Copper–Constantan	–200°C to + 350°C
Iron–Constantan	–200°C to + 100°C
Iron–Nickel	+300°C to + 600°C
Nickel Chrome	+600°C to +1000°C
Platinum–Platinum Rhodium	+1000°C to +1750°C
Tungsten–Rhenium	0°C to + 2000°C

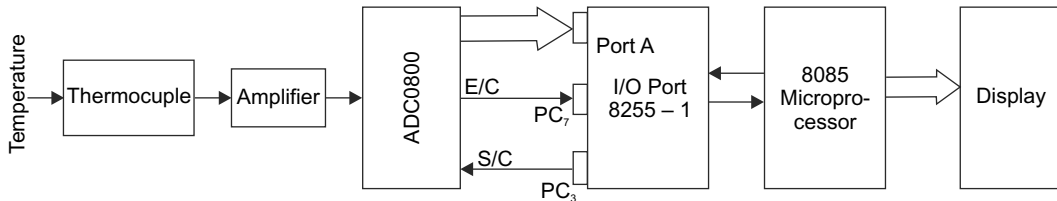


Fig. 10.54 Schematic block diagram of temperature measurement

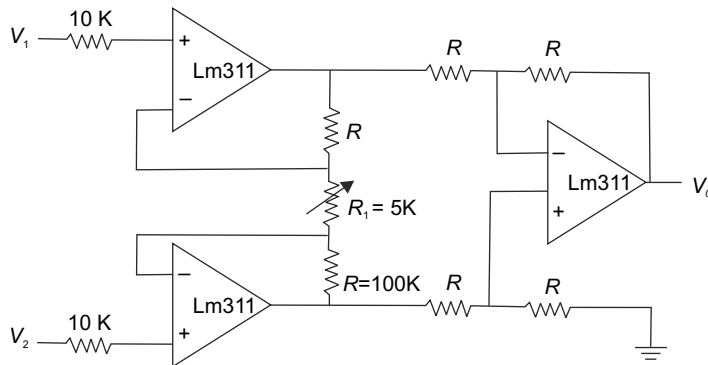


Fig. 10.55 Instrumentation amplifier

The microprocessor-based temperature measurement of an electrical furnace is shown in Fig. 10.54. Here, a thermocouple is used as a sensor for temperature measurement. The output of a thermocouple is directly proportional to the furnace temperature, which is in millivolt range. As output voltage is not in a measurable condition, it must be amplified using an instrumentation amplifier. The amplified voltage is applied to an A/D converter. The microprocessor sends a start of conversion signal to the A/D converter through the port of 8255 PPI. When an A/D converter completes conversion, it sends an end-of-conversional signal to the microprocessor. Having received an end-of-conversion signal from the A/D converter, the microprocessor reads the output of the A/D converter, which is a digital quantity proportional to the temperature to be measured. Then the microprocessor displays the measured temperature. The flowchart for temperature measurement is illustrated in Fig. 10.56. The program for temperature measurement is as follows:

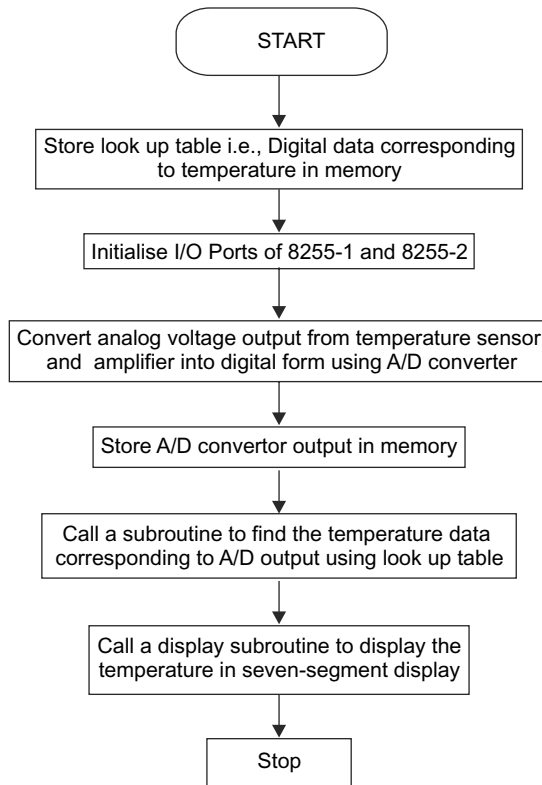


Fig. 10.56 Program flowchart for temperature measurement and display

PROGRAM 10.12 for Temperature Measurement

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	3E, 98		MVI	A,98H	Load control word of 8255-1 in accumulator
8002	D3, 03		OUT	03H	Write control word in control word register and initialize ports
8004	3E, 08		MVI	A, 08H	Send start of conversion signal through PC ₃
8006	D3, 02		OUT	02H	PC ₃ is high
8008	3E, 00		MVI	A, 00H	As PC ₃ will be high for 1or two clock pulse, make it 0
800A	D3, 0A		OUT	02H	PC ₃ becomes low
800C	DB, 02	LOOP	IN	02	Read end of conversion signal
800E	17		RAL		Rotate accumulator to check either conversion is over or not.
800F	D2, 0B, 80		JNC	LOOP	If conversion is not completed, jump to LOOP

(Contd.)

(Contd.)

8012	DB, 00	IN	00	Read digital output of A/D converter
8014	2F	CMA		Complement of ADC output
8015	D6, 80	SUI	80H	Subtract 80H
8017	21, 50, 80	LXI	H, 8050H	Load 8050H in HL-register pair
801A	77	MOV	M, A	Store accumulator content in 8050H location
801B	CD, 00, 81	CALL	8100	Get temperature from look-up table
801E	CD, 50, 81	CALL	8150	Call the display subroutine routine,
8021	76	HLT		Stop

Table 10.6 Temperature and output voltage in mV for thermocouple (chromel-constantan)

Temperature in °C	Output voltage in mV	Amplified Voltage in Volts	Digital equivalent of Voltage
100	4	0.27	06
200	8	0.55	0E
400	16	1.10	1C
600	24	1.65	2A
800	32	2.2	38
1000	40	2.76	46
1200	48	3.31	54
1400	56	3.86	62
1600	64	4.41	70
1800	72	4.95	7E

✓ **Look-up table for temperature measurement**

Memory Address	Hex code of Temperature	Temperature Display °C	Memory Address	Hex code of Temperature	Temperature Display °C
9018	06	0100	9098	46	1000
...
9032	0E	0200	90B2	54	1200
...
904A	1C	0400	90CC	62	1400
...
9066	2A	0600	90E2	70	1600
...
907C	38	0800	90FC	7E	1800

10.4.7 Water-Level Indicator

A water-level indicator works by converting water levels into electrical signals and measures them by electrical or electronics circuits. The most simplest type water level indicator is the resistive method. This is also known as

contact point type. A number of resistances of suitable values have their one end inserted in the column. Resistance may be a function of level.

The microprocessor-based water level indicator is shown in Fig. 10.57. The contact-type level sensors are connected to + 5 V through series resistors and other terminals are grounded and assume water tank is also at ground potential. When the level sensor is immersed in water, it will be at ground potential and its output will be logic 0. If the level sensor is not immersed in water, its output is + 5 V or logic 1. As shown in Fig.10.57, there are eight level sensors, which are used to indicate eight different levels of the tank. The output of level sensors are connected to a buffer and buffer outputs are applied to Port A of 8255. The microprocessor reads the buffer output through Port A of 8255 and determines the water level based on look-up table, which is stored in the memory. After finding out the level, it will be displayed in a seven-segment display. The program for water-level measurement is given below.

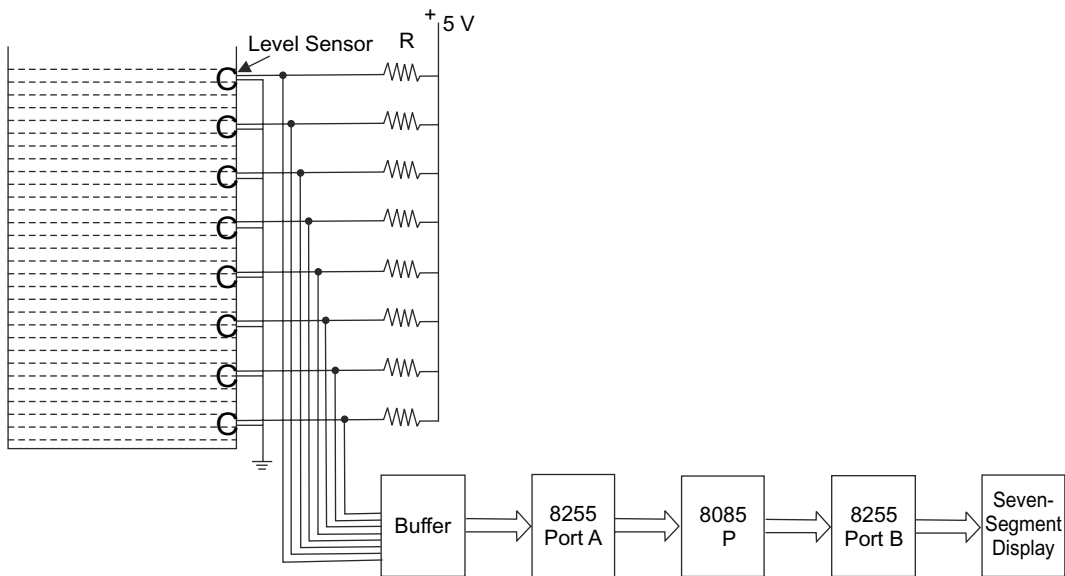


Fig. 10.57 Schematic block diagram of water level measurement

PROGRAM 10.13 for Level Measurement

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	3E, 98		MVI	A,98H	Load control word of 8255-1 in accumulator
8002	D3, 03		OUT	03H	Write control word in control word register and initialize ports
8004	3E, 08		MVI	H,90	
8006	DB, 00	START	IN	00H	
8008	3E, 00		MOV	L,A	
800A	7E		MOV	A,M	
800B	D3, 01		OUT	01	
800D	17, 06, 80		JMP	START	

✓ **Look-up table between buffer output and level in cm**

Memory Address	Hex code of Buffer Output	Level Display (cm)	Memory Address	Hex code of Buffer output	Level Display (cm)
9000	00	90	90F8	F8	40
9080	80	80	90FC	FC	30
90C0	C0	70	90FE	FE	20
90E0	E0	60	90FF	FF	10
90F0	F0	50			

10.4.8 Measurement and Display of Speed of a Motor

Figure 10.58 shows the microprocessor-based speed measurement. A tachogenerator is coupled at the shaft of the motor and generates a voltage proportional to the speed. The output of the tachogenerator (TG) used in this measurement scheme is 0 to 5 volts dc for speed variation of 0 to 1500 RPM. The output voltage is connected to ADC 0808 for analog-to-digital conversion. The output of the A/D converter is applied to Port A of 8255-1. Seven-segment display units are connected to Port A and Port B of 8255-2. The control word for both 8255-1 is 98H and 8255-2 is 80H. The look-up table consists of the hex code of techo-generator voltage and its corresponding speed in rpm. The microprocessor reads the digital output of the A/D converter for different speeds of the motor. After that, the microprocessor measures the speed using a look-up table and displays the speed of the motor in a seven-segment display.

For better accuracy of measurement, the search-table properly calibrated. For this, each digital input voltage, and corresponding speed is measured accurately and stored in the memory location. For one digital input voltage, two memory locations are located in the search table where decimal data corresponding to the speed are stored. After getting a digital input corresponding to a speed, the speed will be searched from memory and displayed in the displayed screen. If we want to measure speed accurately, a train of pulses can be generated using a photoelectric switch sensor. Opto-switches consist of a light source and a light sensor within a single unit as shown in Fig. 10.59. This scheme of speed measurement has a light source, a semiconductor device sensitive to light and an attachment disc on the shaft containing a hole to pass light. The microprocessor will count the number of pulses per second, which is directly proportional to the speed.

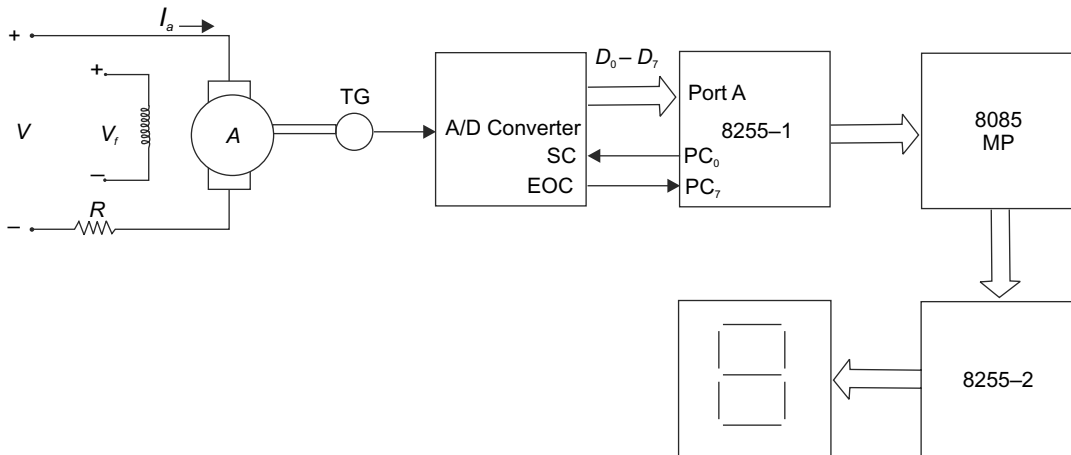


Fig. 10.58 Schematic block diagram of speed measurement

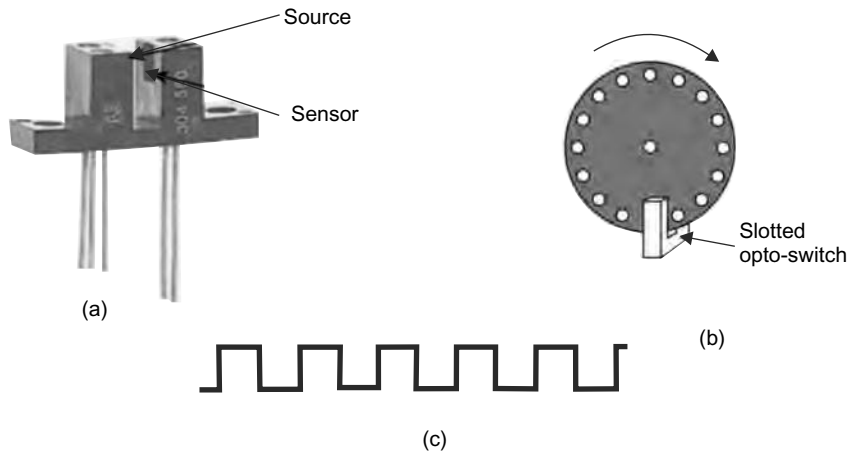


Fig. 10.59 (a) Slotted opto-switch (b) Opto-switch sensor (c) Train of output pulse

PROGRAM 10.14 for Speed Measurement

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	3E, 98		MVI	A, 98H	Load control word of 8255-1 in accumulator
8002	D3, 03		OUT	03H	Write control word in control word register and initialize ports
8004	3E, 08		MVI	A, 08H	Send start of conversion signal through PC ₃
8006	D3, 02		OUT	02H	PC ₃ is high
8008	3E, 00		MVI	A, 00H	As PC ₃ will be high for 1 or two clock pulse, make it 0
800A	D3, 0A		OUT	02H	PC ₃ becomes low
800C	DB, 02	LOOP	IN	02	Read end of conversion signal
800E	17		RAL		Rotate accumulator to check either conversion is over or not
800F	D2, 0C, 80		JNC	LOOP	If conversion is not completed, jump to LOOP
8012	DB, 00		IN	00	Read digital output of A/D converter
8014	2F		CMA		Complement of ADC output
8015	D6, 80		SUI	80H	Subtract 80H
8017	21, 50, 80		LXI	H, 8050H	Load 8050H in HL-register pair
801A	77		MOV	M, A	Store accumulator content in 8050H location
801B	CD, 00, 81		CALL	8100	Get temperature from look-up table
801E	CD, 50, 81		CALL	8150	Call the display subroutine routine
8021	76		HLT		Stop

✓ **Look-up table for speed measurement**

<i>Memory Address</i>	<i>Hex code of Speed</i>	<i>Speed Display in RPM</i>	<i>Memory Address</i>	<i>Hex code of Speed</i>	<i>Speed Display in RPM</i>
9000	00	0000	9080	40	0768
9002	01	0012	9082	41	0780
9004	02	0024	9084	42	0792
...
9032	19	0300	90B2	59	1068
...
9064	32	0600	90E4	72	1368
...
907E	3F	0756	9100	80	1536

10.5 MICROPROCESSOR-BASED PROTECTION

Reliable and accurate protection schemes are required for any system. Microprocessors can fulfill these requirements without fail. In addition to the system protection, microprocessors can perform all control operations, analysis as well as measurement. The cost of a protective scheme should be about 1% of the cost of the equipment to be protected. When the microprocessor is used to control the system in addition to system protection, it will be very economical. Presently, microprocessor-based protective schemes are developed. Therefore, microprocessor applications will result in availability of faster, more accurate and reliable relays than conventional relays. These relays are known as *static relays*. A microprocessor increases the flexibility of static relays due to its programmable approach. A number of desired characteristics such as overvoltage, undervoltage, overcurrent, directional, impedance, reactance, and mho can be used in microprocessor-based relays. In this section, microprocessor-based overvoltage protection has been discussed.

10.5.1 Overvoltage Protection

Electrical appliances or any electrical and electronics instruments always require protection against over and undervoltage. The conventional relays are already used for the under and overvoltage, and the maximum and minimum level of voltages are not changeable. Though a micro-processor-based system is of high cost, but the advantage of this system is that the same system may provide protection against maximum and minimum allowable current and voltage with a scope to adjust maximum and minimum limits.

The schematic block diagram of the system overvoltage protection is shown in Fig. 10.60. It is depicted in Fig. 10.60 that a single-phase ac supply is connected to a load (electrical appliance) through an electromagnetic relay. This electrical appliance must be protected from overvoltage as well as undervoltage. For this, a Potential Transformer (PT) has been used to collect the voltage signal. The output of PT is fed to the input of a peak detector circuit to detect the peak value of the voltage. The output of a peak detector circuit is applied to the A/D converter for analog-to-digital conversion. For protection against over and undervoltage, an opto-coupler circuit, MCT2E is used to connect to the pin PB_0 of the I/O port. A 5 V dc supply has been connected to the opto-coupler circuit and the output has been connected to the energizing coil of an electromagnetic relay through a diode IN 4007.

In microprocessor-based protection, initially the upper and lower limiting values of voltage are stored in memory. Initialize Port A and Port C upper as input ports and Port B and Port C lower as output ports. The microprocessor receives the output of the A/D converter and compares the same with the upper and

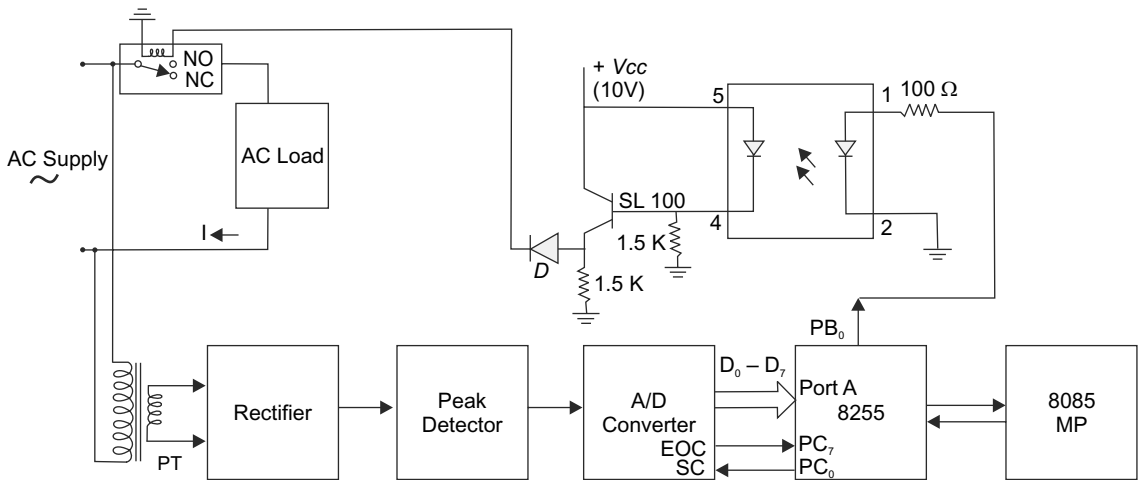


Fig. 10.60 Schematic block diagram of overvoltage protection

lower limiting values of voltage (V_{UL} and V_{LL}). Within the safe limit, the microprocessor sends signals 0 through PB_0 and relay is OFF and supply current flows through load. If voltage value is either less than V_{LL} or greater than V_{UL} , the microprocessor sends '1' signal through PB_0 and the relay coil is energized. As the relay becomes ON, supply voltage is disconnected from the system and the system will be protected. The flowchart of the program is given in Fig. 10.61 and the assembly-language programming for voltage protection is given below.

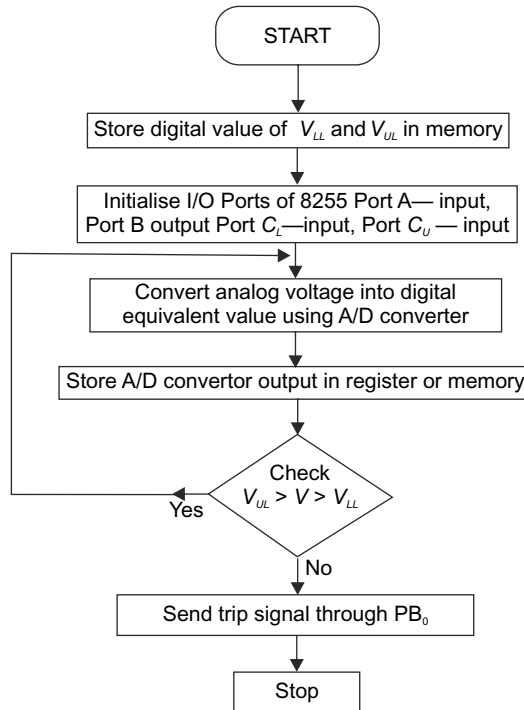


Fig. 10.61 Flow chart for voltage protection

PROGRAM 10.15 for Voltage Protection

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	21, 00, 81		LXI	H, 8100H	Initialize memory location 8100H
8003	3E, 50		MVI	A, 50H	Store digital equivalent of V_{LL} in 8100H location
8005	77		MOV	M, A	
8006	23		INX	H	Increment HL register pair
8007	3E, 70		MVI	A, 70H	Store digital equivalent of V_{UL} in 8100H location
8009	77		MOV	M, A	
800A	3E, 98		MVI	A, 98H	Load control word of 8255-1 in accumulator
800C	D3, 03		OUT	03H	Write control word in control word register and initialize ports
800E	3E, 08	START	MVI	A, 08H	Send start of conversion signal through PC_3
8010	D3, 02		OUT	02H	PC_3 is high
8012	3E, 00		MVI	A, 00H	As PC_3 will be high for 1 or two clock pulse, make it 0
8014	D3, 02		OUT	02H	PC_3 becomes low
8016	DB, 02	LOOP	IN	02	Read end of conversion signal
8018	17		RAL		Rotate accumulator to check either conversion is over or not
8019	D2, 16, 80		JNC	LOOP	If conversion is not completed, jump to LOOP
801C	DB, 00		IN	00	Read digital output of A/D converter corresponding to voltage
801E	2F		CMA		Complement of ADC output
801F	D6, 80		SUI	80H	Subtract 80H
8021	4F		MOV	C, A	Store digital equivalent of voltage in C register
8022	21, 00, 81		LXI	H, 8100H	Load 8100H in HL-register pair
8025	7E		MOV	A, M	Move digital equivalent of V_{LL} voltage into accumulator from 8100H location
8026	B9		CMP	C	Compare C with V_{LL}
8027	DA, 33, 80		JC	TRIP	If carry flag is set, jump to trip

(Contd.)

(Contd.)

802A	79		MOV	A, C	Move content of C register to accumulator
802B	23		INX	H	Increment HL register pair
802C	BE		CMP	M	Compare the content of memory V_{UL} with accumulator
802D	DA, 33, 80		JC	TRIP	If carry flag is set, jump to trip
8030	C3, 0E, 80		JMP	START	Jump to start
8033	3E, 01	TRIP	MVI	A, 01H	
8035	D3, 01		OUT	01H	Send $PB_0=1$ and trip the circuit
8037	76		HLT		Stop

10.6 MICROPROCESSOR-BASED TRAFFIC CONTROL

Nowadays microprocessors are used to implement the traffic control system. Figure 10.62 shows the simple model of microprocessor-based traffic control system. The various control signals such as red, green, orange, forward arrow, right arrow and left arrow are used in this scheme. The forward, right and left arrows are used to indicate forward, right and left movement respectively. The red (R) signal is used to stop the traffic in

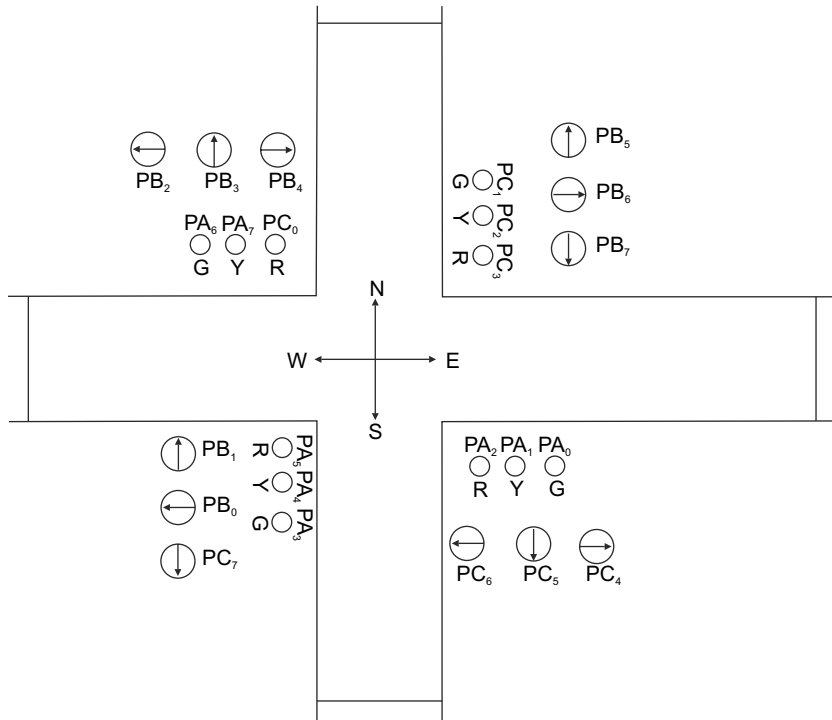


Fig. 10.62 Traffic light control

the required lane and the yellow (Y) signal is used as standby, which indicates that the traffic must wait for the next signal. The green (G) light for a particular lane remains ON for DELAY-1 seconds followed by the standby signal for DELAY-2 seconds. However, at a time for 3 out of the four roads, the left signal or the left arrow remains on even though that lane may have a red signal. The traffic light control is implemented using the 8085 microprocessor kit having 8255 on board and the interfacing circuit is illustrated in Fig. 10.63. Each

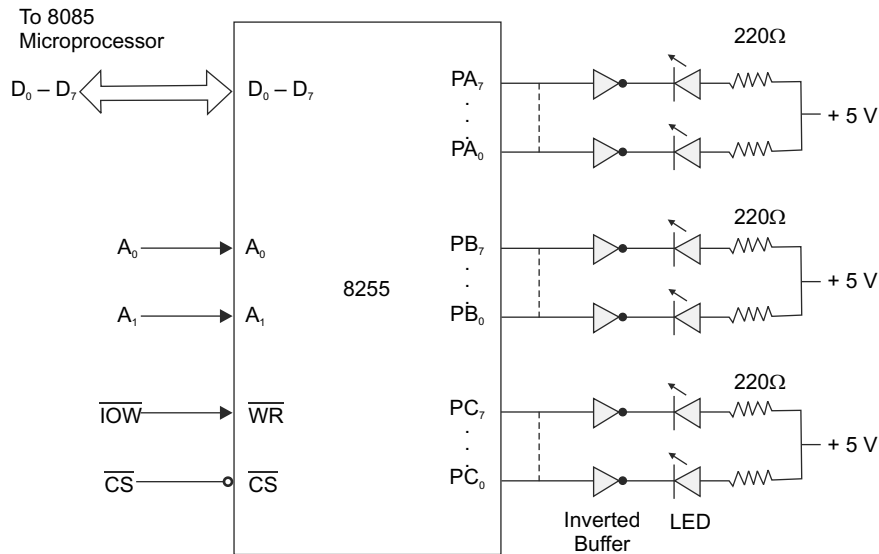


Fig. 10.63 The interfacing circuit for traffic light control

signal is controlled by a separate pin of I/O ports. The total number of logic signals required for this arrangement is twenty-four. The programmable peripheral interface device 8255 is used to interface these 24 logic signals with the lamps. The logic '0' and '1' represent the state of the lamp. Logic '1' represents ON and '0' represents OFF. All ports of 8255 are used as output ports. The control word to make all ports as output ports for Mode 0 operation is 80H. The traffic light control program can be written by the following steps:

Step 1 Initialize all ports of the 8255 as output ports.

Step 2 Determine the required status of Port A, Port B and Port C of 8255 for north to south traffic movement. Load data into accumulator and send to Port A, Port B and Port C for north to south traffic movement.

Step 3 Call delay subroutine-1.

Step 4 Before starting east to west traffic movement, north to south traffic movement will be ready to stop, and east to west traffic must be ready for movement. Therefore, determine the required status of Port A, Port B and Port C for this operation. Then load data into accumulator and send to Port A, Port B and Port C for north to south traffic movement which will be ready to stop and east to west traffic must be ready for movement.

Step 5 Call delay subroutine-2.

Step 6 For east to west traffic movement, determine the required status of Port A, Port B and Port C of 8255. Load data into accumulator and send to Port A, Port B and Port C for east to west traffic movement.

Step 7 Call delay subroutine-1.

Step 8 Prior to starting south to north traffic movement, east to west traffic will be ready to stop, and south to north traffic must be ready for movement. For this operation, determine the status of Port A, Port B and Port C of 8255. Load required data into accumulator and send to Port A, Port B and Port C for east to west traffic will be ready to stop and south to north traffic must be ready for movement.

Step 9 Call delay subroutine-2.

Step 10 Determine the status of Port A, Port B and Port C for south to north traffic movement. Load required data into accumulator and send to Port A, Port B and Port C for south to north movement.

Step 11 Call delay subroutine-1.

Step 12 Before starting west to east traffic movement, south to north traffic movement will be ready to stop and west to east traffic must be ready for movement. Find out the status of Port A, Port B and Port C for this operation. Load required data into accumulator and send to Port A, Port B and Port C for south to north traffic movement which will be ready to stop and west to east traffic must be ready for movement.

Step 13 Call delay subroutine-2.

Step 14 For west to east traffic movement, determine the status of Port A, Port B and Port C of 8255. Load necessary data into accumulator and send to Port A, Port B and Port C for west to east traffic movement.

Step 15 Call delay subroutine-1.

Step 16 Subsequently, west to east traffic movement will be ready to stop and north to south traffic must be ready for movement. Determine the status of Port A, Port B and Port C for this operation. Load needed data into accumulator and send to Port A, Port B and Port C west to east traffic movement will be ready to stop and north to south traffic must be ready for movement.

Step 17 Call delay subroutine-2.

Step 18 Jump to step-2.

The chart shows the bit assignment of ports. Putting 0s and 1s in the required position, the data byte for each port can be derived. For example, during north to south traffic movement, the statuses of Port A, Port B and port C are as follows:

PA ₇	PA ₆	PA ₅	PA ₄	PA ₃	PA ₂	PA ₁	PA ₀
0	0	1	0	0	0	0	1
PB ₇	PB ₆	PB ₅	PB ₄	PB ₃	PB ₂	PB ₁	PB ₀
0	0	0	0	0	1	0	0
PC ₇	PC ₆	PC ₅	PC ₄	PC ₃	PC ₂	PC ₁	PC ₀
1	1	1	1	1	0	0	1

When north to south traffic movement will be ready to stop and east to west traffic must be ready for movement, the statuses of Port A, Port B and Port C are as follows:

PA ₇	PA ₆	PA ₅	PA ₄	PA ₃	PA ₂	PA ₁	PA ₀
0	0	0	1	0	0	1	0
PB ₇	PB ₆	PB ₅	PB ₄	PB ₃	PB ₂	PB ₁	PB ₀
0	0	0	0	0	1	0	0
PC ₇	PC ₆	PC ₅	PC ₄	PC ₃	PC ₂	PC ₁	PC ₀
0	0	0	0	1	0	0	1

The calculated necessary data bytes of Port A, Port B and Port C for all types of traffic movement are illustrated in Table 10.7 as given below.

Table 10.7 Traffic Movement and Status of Ports

Traffic Movement	Status of Port A	Status of Port B	Status of Port C
North to south traffic movement	21H	04H	F9H
North to south traffic movement be ready to stop, and east to west traffic be ready for start	12H	04H	09H
East to west traffic movement	0CH	27H	89H
East to west traffic movement to be ready to stop, and south to north traffic be ready for start	94H	20H	08H
South to north traffic movement	64H	3CH	18H
South to north traffic movement to be ready to stop, and west to east traffic be ready for start	A4H	00H	14H
West to east traffic movement	24H	D0H	93H
West to east traffic movement to be ready to stop, and north to south traffic ready for start	22H	00H	85H

The program for traffic light control as follows:

PROGRAM 10.16 for Traffic Light Control

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	3E, 80		MVI	A, 80H	Load control word of 8255 in accumulator
8002	D3, 0B		OUT	03H	Write control word in control word register and initialize ports
8004	3E, 21	START	MVI	A, 21H	Send 21H in Port A, F9H in Port C and 04H in Port B for north to south traffic movement
8006	D3, 00		OUT	00H	
8008	3E, F9		MVI	A, F9H	
800A	D3, 02		OUT	02H	
800C	3E, 04		MVI	A, 04H	
800E	D3, 01		OUT	01H	
8010	CD, 00, 81		CALL	DELAY_1	Delay-1 memory location is 8100
8013	3E, 12		MVI	A, 12H	Send 12H in Port A, 09H in Port C and 04H in Port B for north to south traffic movement will be ready to stop and east to west traffic movement is ready to start
8015	D3, 00		OUT	00H	
8017	3E, 09		MVI	A, 09H	
8019	D3, 02		OUT	02H	
801B	3E, 04		MVI	A, 04H	

(Contd.)

(Contd.)

801D	D3, 01	OUT	01H	
801F	CD, 00, 82	CALL	DELAY_2	Delay-2 memory location is 8200
8022	3E, 0C	MVI	A, 0CH	Send 0CH in Port A, 89H in Port C and 27H in Port B for east to west traffic movement
8024	D3, 00	OUT	00H	
8026	3E, 89	MVI	A, 89H	
8028	D3, 02	OUT	02H	
802A	3E, 27	MVI	A, 27H	
802C	D3, 01	OUT	01H	
802E	CD, 00, 81	CALL	DELAY_1	
8031	3E, 94	MVI	A, 94H	Send 94H in Port A, 08H in Port C and 20H in Port B for east to west traffic movement will be ready to stop and south to north traffic movement is ready to start
8033	D3, 00	OUT	00H	
8035	3E, 08	MVI	A, 08H	
8037	D3, 02	OUT	02H	
8039	3E, 20	MVI	A, 20H	
803B	D3, 01	OUT	01H	
803D	CD, 00, 82	CALL	DELAY_2	
8040	3E, 64	MVI	A, 64H	Send 64H in Port A, 18H in Port C and 3CH in Port B for south to north traffic movement
8042	D3, 00	OUT	00H	
8044	3E, 18	MVI	A, 18H	
8046	D3, 02	OUT	02H	
8048	3E, 3C	MVI	A, 3CH	
804A	D3, 01	OUT	01H	
804C	CD, 00, 81	CALL	DELAY_1	
804F	3E, A4	MVI	A, A4H	Send A4H in Port A, 14H in Port C and 00H in Port B for south to north traffic movement is ready to stop and west to east traffic movement will be ready to start
8051	D3, 00	OUT	00H	
8053	3E, 14	MVI	A, 14H	
8055	D3, 02	OUT	02H	
8057	3E, 00	MVI	A, 00H	
8059	D3, 01	OUT	01H	
805B	CD, 00, 82	CALL	DELAY_2	
805E	3E, 24	MVI	A, 24H	Send 24H in Port A, 93H in Port C and D0H in Port B for west to east traffic movement

(Contd.)

(Contd.)

8060	D3, 00		OUT	00H	
8062	3E, 93		MVI	A, 93H	
8064	D3, 02		OUT	02H	
8066	3E, D0		MVI	A, D0H	
8068	D3, 01		OUT	01H	
806A	CD, 00, 81		CALL	DELAY_1	
806D	3E, 22		MVI	A, 22H	Send 22H in Port A, 85H in Port C and 00H in Port B for west to east traffic movement is ready to stop and north to south traffic movement will be ready to start
806F	D3, 00		OUT	00H	
8071	3E, 85		MVI	A, 85H	
8073	D3, 02		OUT	02H	
8075	3E, 00		MVI	A, 00H	
8077	D3, 01		OUT	01H	
8079	CD, 00, 82		CALL	DELAY_2	
807C	C3, 04, 80		JMP	START	

✓ **DELAY SUBROUTINE-1 (DELAY-1)****Subprogram for Delay Subroutine-1**

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8100	11, 00, 80		LXI	D, 8000	Load suitable delay value in DE register pair
8103	1B	Level_1	DCX	D	Decrement the DE register pair by 1
8104	7A		MOV	A, D	Move the content of D into A
8105	B3		ORA	E	OR operation between A and D
8106	C2, 03, 81		JNZ	Level_1	If DE is not equal to zero, jump to Level-1
8109	C9		RET		

✓ **DELAY SUBROUTINE-2 (DELAY_2)****Subprogram for Delay Subroutine-2**

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8200	16, FF		MVI	D, FF	Load suitable delay value FF in D register
8202	15	Level-2	DCR	D	Decrement the D register by 1
8203	C2, 02, 82		JNZ	Level-2	If D is not equal to zero, jump to Level-2
8206	C9		RET		

10.7 MICROPROCESSOR-BASED FIRING CIRCUIT OF A THYRISTOR

Figure 10.64 shows the circuit diagram for triggering a thyristor using a microprocessor. In a microprocessor-based firing circuit, it is necessary to synchronize the microprocessor software with the ac input voltage. For this purpose, a zero crossing detector is necessary. A potential transformer is used to reduce the ac supply voltage into 5 V ac and is also used to isolate the power circuit and control circuit. The output voltage of PT is applied to ZCD. Figure 10.65 shows the ZCD circuit, which generates a pulse on positive zero crossing as well as negative zero crossing of supply voltage. The microprocessor reads the output of ZCD through an input port and detects the zero crossing at the rising edge of the ac voltage waveform. After that, the microprocessor executes a time delay loop and subsequently, it generates the trigger or firing pulse holding it for

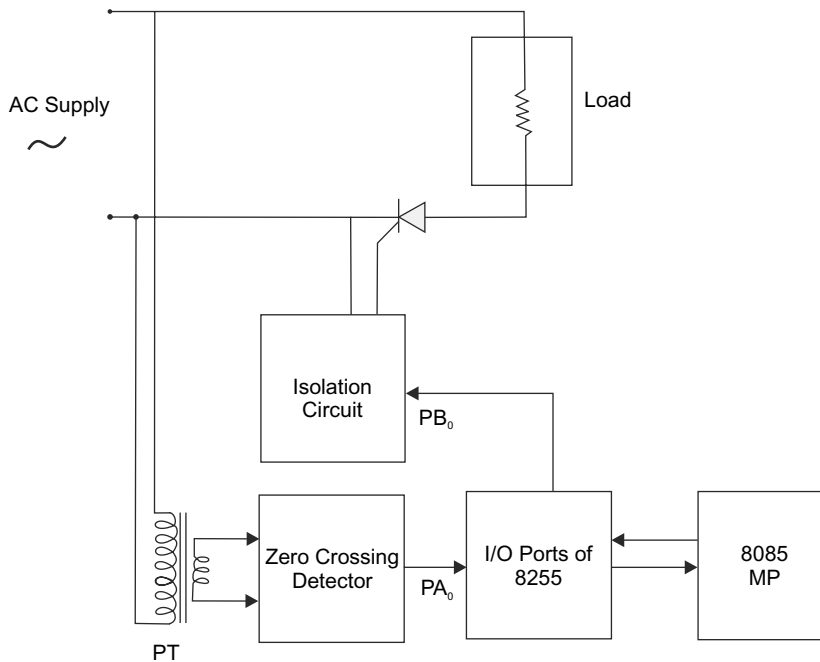


Fig. 10.64 Schematic block diagram of thyristor trigger circuit

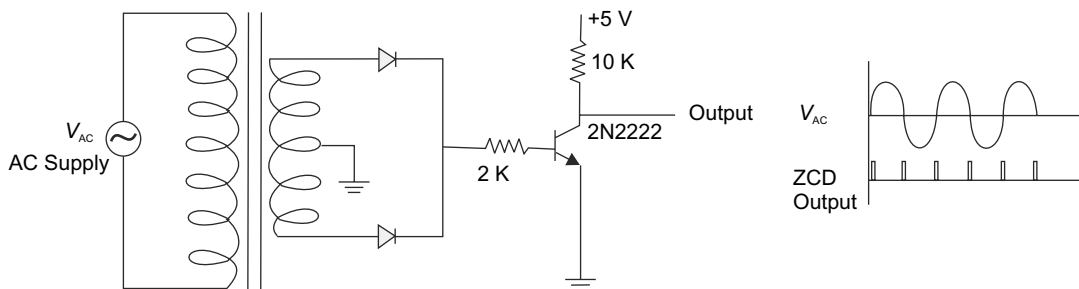


Fig. 10.65 ZCD circuit and its output waveform

small time interval. As the output trigger or firing pulse has low power capacity, it cannot be used to trigger the thyristor directly. For that reason, an opto-isolator circuit using MCT2E has been connected to the microprocessor output port and the output of MCT2E is applied to gate-cathode terminals of the thyristor. As the thyristor is forward biased and gate pulse is applied, the thyristor becomes ON and current flows through the load. The ac input voltage, ZCD output, firing pulse and output voltage waveform for a delay angle α are shown in Fig 10.66(a). Again, the microprocessor examines the ZCD output and detects the next rising edge at zero crossing of the ac waveform. After following the time delay, the microprocessor generates the next trigger pulse. Consequently, the above operation will be repeated in cyclic order. The program flow chart for generating firing pulse to turn on a thyristor is depicted in Fig. 10.66(b).

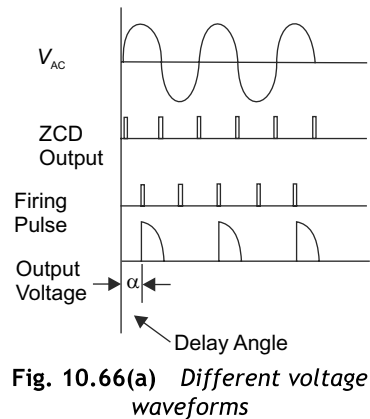


Fig. 10.66(a) Different voltage waveforms

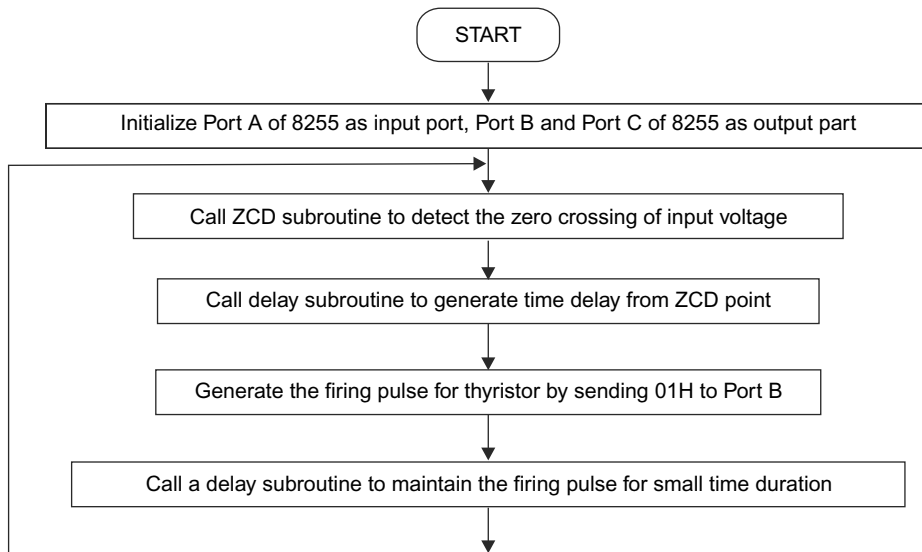


Fig. 10.66(b) Flow chart for generating firing pulse to turn on a thyristor

The following steps are used to develop the assembly-language program for firing circuit of a thyristor.

- Step 1** Initialize all ports A, B and C.
- Step 2** Call ZCD subroutine to find the positive zero crossing of ac input voltage.
- Step 3** Load HL register pair or any register with 16-bit or 8-bit data respectively and call delay subroutine for creating delay.
- Step 4** After delay generation, one pin PB₀ will be high. This high output should be maintained for a certain duration. For this, another delay loop will be called.
- Step 5** Load any register with 8-bit data and the delay subroutine will be executed for creating delay.
- Step 6** After completion of delay, send logic '0' signal at PB₀ and then PB₀ will be low.
- Step 7** Jump to Step 2 to find the next ZCD point and repeat the above process continuously.

This firing circuit can be used for a half-wave rectifier, full-wave rectifier, and ac voltage controller. The program for firing or triggering a thyristor is as follows:

PROGRAM 10.17 for Thyristor Firing

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	31, 00, 90		LXI	SP, 9000	Initialize stack pointer
8003	3E, 90		MVI	A,90	Assign Port A as input port and Port B and C as output ports
8005	D3, 03		OUT	03	Select I/O chip no.1
8007	CD, 00, 81	START	CALL	ZCD	Call subroutine ZCD to detect the zero crossing of input voltage for synchronization; ZCD address is 8100
800A	CD, 50, 81		CALL	DELAY_1	Call delay loop-1 to generate time delay from ZCD. Address of DELAY-1 is 8150
800D	3E, 01		MVI	A,01	Send 01 to Port B and PB0 becomes 1
800F	D3, 01		OUT	01	
8011	CD, 00, 82		CALL	DELAY_2	Call delay loop-2, as PB0 will be high for specified duration; Address of DELAY-1 is 8200
8014	3E, 00		MVI	A,00	Send 00 to Port B and PB0 becomes 0
8016	D3, 01		OUT	01	
8018	C3, 07, 80		JMP	START	Jump to start

Subroutine for ZCD

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8100	DB, 00	Level-1	IN	00	Input through Port A
8102	E6, 01		ANI	01H	Logical AND 01H with accumulator
8104	C2, 00, 81		JNZ	Level_1	Jump not zero to Level-1
8107	DB, 00	Level-2	IN	00	Input through Port A
8109	E6, 01		ANI	01H	Logical AND 01H with accumulator
810B	C2, 07, 81		JZ	Level_2	Jump not zero to Level-1
810E	C9		RET		

✓ DELAY SUBROUTINE-1 (DELAY_1)

Subprogram for Delay Subroutine-1

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8150	0E, 20	Level_1	MVI	C,20	Load 20H in C register
8152	06, FF	Level_2	MVI	B,FF	Load FFH in B register

(Contd.)

(Contd.)

8154	05	DCR	B	Decrement B register
8155	C2, 52, 81	JNZ	Level_2	If B register is not equal to zero, jump to Level-2
8158	0D	DCR	C	Decrement C register
8159	C2, 50, 81	JNZ	Level_1	If C register is not equal to zero, jump to Level-1
815C	C9	RET		

✓ DELAY SUBROUTINE-2 (DELAY_2)

Subprogram for Delay Subroutine-2

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8200	16, FF		MVI	D, FF	Load suitable delay value FF in D register
8202	15	Level-3	DCR	D	Decrement the D register by 1
8203	C2, 02, 82		JNZ	Level-3	If D is not equal to zero, jump to Level-3
8206	C9		RET		

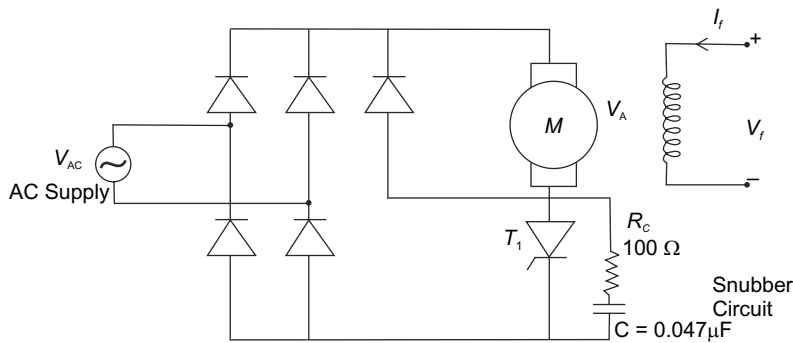


Fig. 10.67 Full-wave controlled rectifier fed dc motor

Figure 10.67 shows a bridge-rectifier fed dc motor, which consists of five diodes and one thyristor. The triggering pulses of the thyristor can be controlled by the microprocessor. The assembly-language program for triggering a thyristor, which is already given in this section, can be used for bridge-rectifier-fed dc motor control. Initially, the stack pointer should be defined and initializes Port A as input and ports B and C as output ports. Zero-cross detectors are used to sense the positive going zero and negative going zero of the ac supply voltage. The microprocessor can sense the zero-instant of the ac supply voltage and generates a trigger pulse after a certain delay. The trigger signal from the microprocessor is applied to the input of the opto-coupler circuit through PB₀ and this trigger pulse must be hold

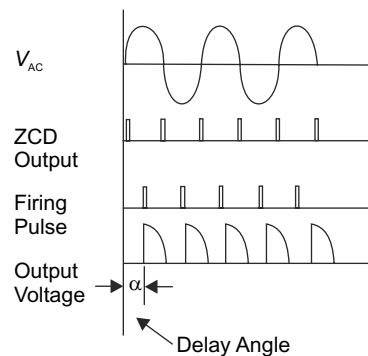


Fig. 10.68 Different voltage waveforms

for a certain time interval. The opto-coupler output signal is applied to the gate of the thyristor to make the thyristor ON. Once the thyristor becomes ON, it will be ON for the +ve half cycle of supply voltage. As the rectified dc voltage is fed to the dc motor, the thyristor will also be ON in the -ve half cycle of supply voltage when microprocessor sends the trigger pulse with the same delay in negative half cycle of supply voltage. The ac input voltage, ZCD output, firing pulse and output voltage waveforms are shown in Fig. 10.68.

10.8 SPEED CONTROL OF DC MOTOR

The speed of a dc motor can be controlled by varying the duty cycle of the transistor. Each duty cycle has a definite ON time and OFF time. When the transistor is ON, the voltage applied across the armature of a dc motor and motor accelerates. If the transistor is OFF, no voltage across the dc motor and motor decelerates. While ON and OFF of transistor cycle is repeated continuously at a fixed frequency, the motor will run at a definite constant speed as fixed average voltage is applied. But motor instantaneous speed will be variable due to voltage or load fluctuation.

Assume motor load torque is constant and ON or OFF period of the armature voltage remains constant. If T_{on} and T_{off} are the on-period and off-period of the transistor respectively, the duty ratio, δ can be expressed as

$$\text{duty - ratio } (\delta) = \frac{T_{on}}{T_{on} + T_{off}}$$

The average voltage applied across the armature of a dc motor is

$$V_{out} = V_{in} \times \frac{T_{on}}{T_{on} + T_{off}} = V_{in} \times \delta$$

where V_{in} is the supply voltage, V_{out} is output voltage, and δ is the duty ratio.

The speed of a dc motor is directly proportional to the average voltage applied across armature. Therefore, speed will be changed with variation in duration of on- or off-period of the armature voltage.

As the motor is operating at constant load torque, motor speed increases with increase in T_{on} . It should be noted that the motor speed varies with the change in load torque, when duty ratio δ is retained constant. While load torque increases but this pulse duration or duty ratio δ is constant, motor speed decreases. On the other hand, any drop in load torque with constant pulse duration, motor speed will lead to increase.

The dc motor speed can be controlled either open loop or closed loop. The open speed control of the dc motor is shown in Fig. 10.69. In open-loop control strategy, the pulse duration generated by the microprocessor

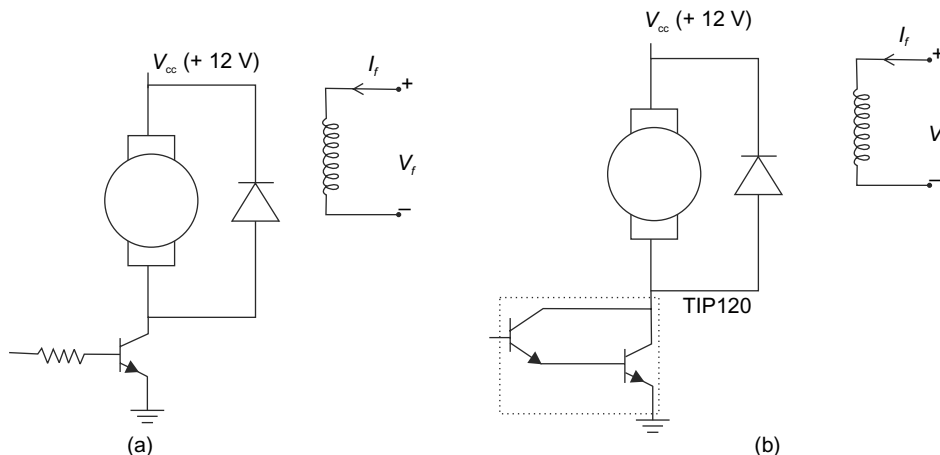


Fig. 10.69 (a) Open-loop speed control of dc motor (b) Transistor-based drive

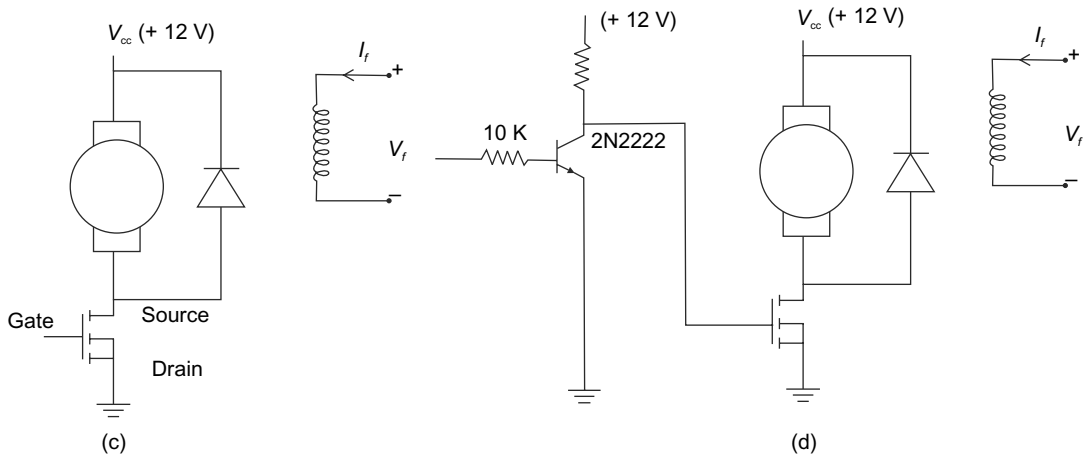


Fig. 10.69 (c) Darlington-pair-based drive (d) Power MOSFET based drive

is fixed. But in closed-loop control, the microprocessor generates pulse, which will be controlled by using software. In this method, the speed of the dc motor can be controlled.

The schematic block diagram of the closed-loop speed control of a dc motor is shown in Fig. 10.70. In this control scheme, by controlling the turn-on period of the supply voltage the average armature voltage can be controlled. Therefore, the microprocessor-based controller is used to generate the controllable pulse and the program of the dc motor closed loop speed control can be developed by the following steps.

Step 1 Initialize Port A as input port and Port B and Port C as output ports.

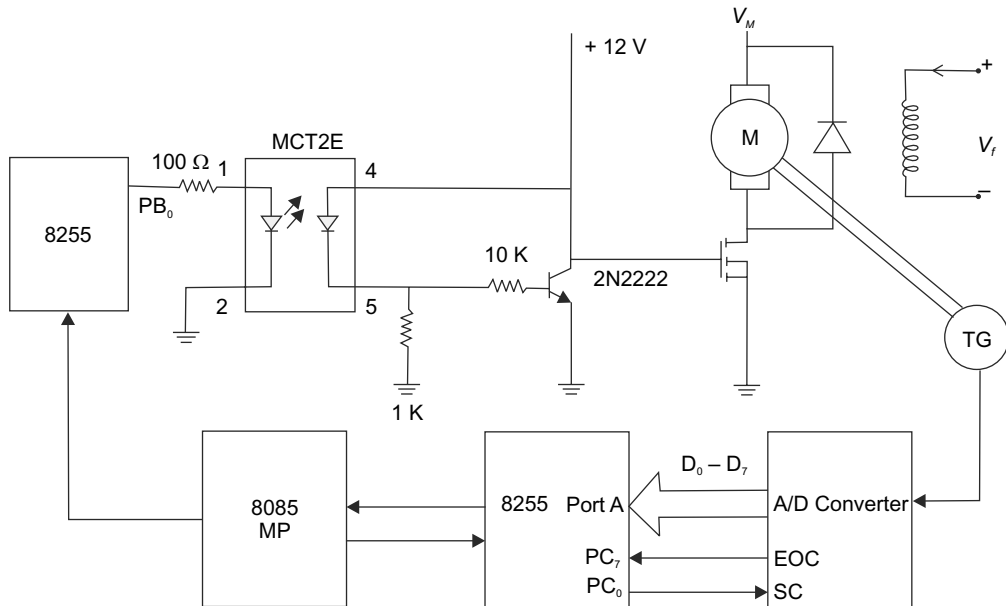


Fig. 10.70 Closed-loop speed control of dc motor

Step 2 Initialize the on time delay (T_{on}) in register pair or memory location. Initialize the off time delay (T_{off}) in the register pair or memory location. Initialize C register as counter.

Step 3 Send 1 high to Port B_0 and call ON time delay subroutine to execute time delay program.

Step 4 Send 0 low to Port B_0 and call OFF time delay subroutine to execute time delay program.

Step 5 Decrement D register. If $D \neq 0$, repeat steps 3 to 5 for initial starting of dc motor.

Step 6 When the dc motor is rotating at a certain speed, tacho-generator generates dc voltage, which is fed to ADC for analog to digital conversion.

Step 7 The microprocessor reads the ADC output and subtracts or compare from a reference input corresponding to speed. If the output speed is less than the reference speed, increase in time is by a very small duration or is proportionate to error in speed. This is done through incrementing the content of on time delay register. When output is more than reference, decrease in time is by small duration or proportionate to difference between reference and actual.

Step 8 Load C register with a count value.

Step 9 Send 1 PB_0 and call for on time delay subroutine with new on time delay.

Step 10 Send 0 to PB_0 and call for the off time delay which is not changed or remains unchanged.

Step 11 Decrement C register. If not equal to zero, repeat steps 9 to 11.

Step 10 Jump to Step 7 and repeat the process.

In the same way, closed-loop control is also possible and the programming of dc motor control is incorporated in this section. The output voltage waveform at specified T_{on} and T_{off} is shown in Fig 10.71(a). The program flow chart for closed loop speed control of dc motor is depicted in Fig. 10.71(b).

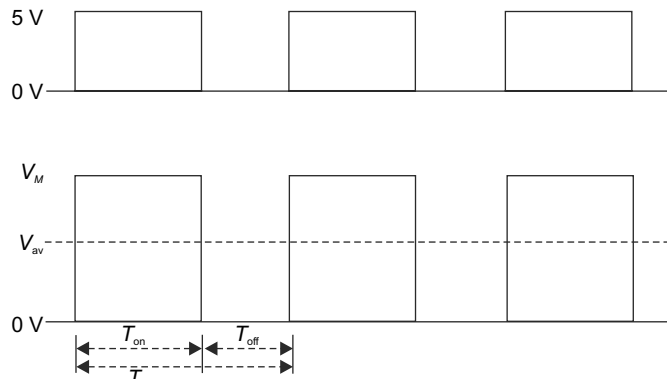


Fig. 10.71(a) *Output voltage waveform*

✓ OPEN-LOOP CONTROL OF DC MOTOR

PROGRAM 10.18 for dc Motor Control

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	31, 00, 90		LXI	SP, 9000H	Define stack pointer at 9000H
8003	3E, 80		MVI	A, 80	Assign Port A and Port B and C as output ports

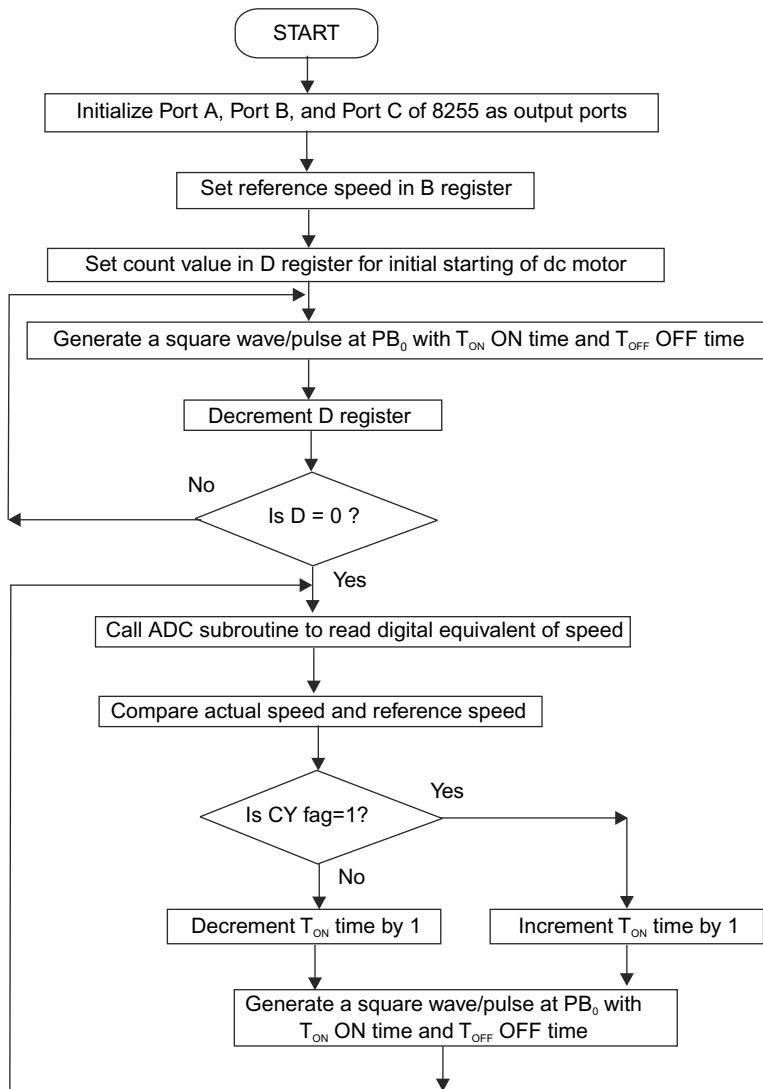


Fig. 10.71(b) Flow chart for closed loop speed control of dc motor

8005	D3, 03		OUT	03	Write control word 80H in control word register
8007	3E, 01	LOOP	MVI	A, 01	Keep PB ₀ high
8009	3E, 01		OUT	01	Output through Port B
800B	0E, FF		MVI	C, FF	Store on time delay (T_{on}) in Register C
800D	00	LEVEL-1	NOP		No operation
800E	00		NOP		No operation
800F	0D		DCR	C	Decrement C register

(Contd.)

(Contd.)

8010	C2, 0D, 80		JNZ	LEVEL-1	Jump to level-1
8013	3E, 00		MVI	A, 00	Keep PB ₀ low
8015	D3, 01		OUT	01	Output through Port B
8017	0E, 80		MVI	C, 80	Store off time delay (T_{off}) in Register C
8019	00	LEVEL-2	NOP		No operation
801A	00		NOP		No operation
801B	0D		DCR	C	Decrement Register C
801C	C2, 19, 80		JNZ	LEVEL-2	Jump on zero to LEVEL-2
801F	C3, 07, 80		JMP	LOOP	Jump to loop

✓ CLOSED-LOOP CONTROL OF DC MOTOR

PROGRAM 10.19 for Speed Control of dc Motor

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	31, 00, 90		LXI	SP, 9000H	Define stack pointer at 9000H
8003	3E, 80		MVI	A, 80	Assign Port A and Port B and C as output ports
8005	D3, 03		OUT	03	Write control word 80H in control word register
8007	06, 50		MVI	B, 50	Store reference speed value, 50H in Register B
8009	16, 20		MVI	D, 20	Store count value 20H in Register D
800B	3E, 01	LOOP_1	MVI	A, 01	Keep PB ₀ high
800D	D3, 01		OUT	01	Output through Port B
800F	0E, FF		MVI	C, 30	Store on time delay (T_{on}) in Register C
8011	00	LEVEL-1	NOP		No operation
8012	00		NOP		No operation
8013	0D		DCR	C	Decrement Register C
8014	C2, 01, 80		JNZ	LEVEL-1	Jump to level-1
8017	3E, 00		MVI	A, 00	Keep PB ₀ low
8019	D3, 01		OUT	01	Output through Port B
801B	0E, 80		MVI	C, 80	Store off time delay (T_{off}) in Register C
801D	00	LEVEL-2	NOP		No operation
801E	00		NOP		No operation
801F	0D		DCR	C	Decrement Register C
8020	C2, 1D, 80		JNZ	LEVEL-2	Jump on zero to Level-2
			DCR	D	
8023	C3, 0B, 80		JNZ	LOOP_1	Jump not zero to loop_1
8026	CD, 00, 81	CONTROL	CALL	ADC	Read the digital equivalent of speed from tacho-generator and ADC output Address of ADC is 8100

(Contd.)

(Contd.)

8029	21, 00, 83		LXI	H, 8300	Move the content of 8300H in accumulator
802C	7E		MOV	A, M	
802D	B8		CMP	B	Compare B and A
802E	DA, 36, 80		JC	NEXT	If carry flag is set, jump to trip
8031	23		INX	H	
8032	34		INR	M	
8033	C3, 3A, 80		JMP	NEXT_1	
8036	23		NEXT	INX H	Increment HL register pair
8037	35		DCR	M	
8038	0E, 10	LOOP_2	MVI	C, 10	
803A	CD	NEXT_1	NOP		
803B	3E, 01		MVI	A, 01	Send 01 to Port B and PB0 becomes 1
803D	D3, 01		OUT	01	
803F	CD, 00, 82		CALL	DELAY_1	Call delay loop-2, as PB0 will be high for specified duration. Address of DELAY-1 is 8200
8042	3E, 00		MVI	A, 00	Send 00 to Port B and PB0 becomes 0
8044	D3, 01		OUT	01	
8046	3E, 01		MVI	A, 01	Send 01 to Port B and PB0 becomes 1
8048	D3, 01		OUT	01	
804A	CD, 50, 82		CALL	DELAY_2	Call delay loop-2. Address of DELAY-1 is 8250
804D	0D		DCR C		Decrement Register C
804E	C2, 38, 80		JNZ	LOOP_2	Jump not zero to loop_2
8051	C3, 26, 80		JMP	CONTROL	Jump to control

Subroutine for ADC

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8100	3E, 08		MVI	A, 08H	Send start of conversion signal through PC ₃
8102	D3, 02		OUT	02H	PC ₃ is high
8104	3E, 00		MVI	A, 00H	As PC ₃ will be high for 1 or two clock pulse, make it 0
8106	D3, 02		OUT	02H	PC ₃ becomes low
8108	DB, 02	LOOP	IN	02	Read end-of-conversion signal
810A	17		RAL		Rotate accumulator to check either onversion is over or not.
810B	D2, 08, 81		JNC	LOOP	If conversion is not completed, jump to LOOP

(Contd.)

(Contd.)

810E	DB, 00	IN	00	Read digital output of A/D converter corresponding to voltage due to speed
8110	2F	CMA		Complement of ADC output
8111	D6, 80	SUI	80H	Subtract 80H
8113	57	MOV	D, A	Store digital equivalent of speed in Register D
8114	21, 00, 83	LXI	H, 8300	Store digital equivalent of speed in 8300 memory location
8117	77	MOV	M, A	Copy accumulator content in memory
8118	23	INX	H	Increment H-L register pair
8119	77	MOV	M, A	Copy accumulator content in memory
811A	C9	RET		Return to main program

✓ **DELAY SUBROUTINE-1 (DELAY_1)****Subprogram for Delay Subroutine-1**

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8200	21, 00, 83		LXI	H, 8300H	
8203	4E	Level_1	MOV	C, M	The new delay value is loaded in Register C from memory location 8301H
8204	06, FF	Level_2	MVI	B, FF	Load FFH in Register B
8206	05		DCR	B	Decrement Register B
8207	C2, 04, 82		JNZ	Level_2	If Register B is not equal to zero, jump to Level-2
820A	0D		DCR	C	Decrement Register C
820B	C2, 03, 82		JNZ	Level_1	If Register C is not equal to zero, jump to Level-1
820E	C9		RET		Return to main program

✓ **DELAY SUBROUTINE-2 (DELAY_2)****Subprogram for Delay Subroutine-2**

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8250	16, FF		MVI	D, 80	Load suitable delay value 80 in Register D
8252	15	Level-3	DCR	D	Decrement the Register D by 1
8253	C2, 52, 82		JNZ	Level-3	If D is not equal to zero, jump to Level-3
8256	C9		RET		Return to main program

10.9 STEPPER MOTOR

Stepper motors are electromechanical devices, which convert electrical pulses into proportionate discrete mechanical rotational movement. To rotate the stepper motor's shaft, a sequence of pulses is required to be applied to stator windings of a stepper motor. When a given number of command pulses are supplied to the motor, the shaft will have turned through a known angle. Therefore, the motor can be used to control position by keeping count of the number of command pulses. Each revolution of the stepper motor's shaft is made up of a series of discrete individual steps. A step is defined as the angular rotation produced by the shaft each time when the motor receives a step pulse. Due to each step, the shaft can rotate a specified angle in degrees. The rotation of the shaft due to each step is called *step angle*. The stepper motors are usually used in position control of robot arms, paper-drive mechanism in a printer, machine-tools control, process control system, textile industry, integrated circuit fabrication, electric watches, tape as well as disk-drive systems, etc. Further, the average motor speed is proportional to the rate at which the pulse command is delivered. At low command pulse rate, the rotor moves in steps, but when the pulse rate is made sufficiently high, because of the inertia, the rotor moves smoothly, as in case of dc motors. As motor speed is proportional to rate of command pulses, it can be used for speed control. The advantages and disadvantages of stepper motors are as follows:

Advantages

- ◆ The angle of rotation of the motor is directly proportional to the number of input pulses.
- ◆ The motor has full torque at standstill when the windings are energized.
- ◆ Precise position control with an accuracy of 3–5% of a step is possible and the error is noncumulative from one step to the next step.
- ◆ This motor has very good response during starting, stopping and reversing operation.
- ◆ This motor drive is very reliable since there are no contact brushes in the motor. Consequently, the life of the motor depends on the life of the bearing.
- ◆ The stepper motor drives provide open-loop control, making the drive system simpler and less costly to control.
- ◆ It is possible to achieve very low speed synchronous rotation with a load, which is directly coupled to the shaft.
- ◆ A wide range of speed control can be possible as the motor speed is directly proportional to the frequency of the input pulses.
- ◆ Repeatability of operation is also feasible in stepper motors.

Disadvantages

- ◆ It is not easy to operate at extremely high speeds.
- ◆ Resonance may occur if not properly controlled.

There are three basic stepper motor types: permanent magnet, variable reluctance–single stack or multi stack type—and hybrid.

The *permanent magnet stepper motors* are very popular and most commonly used. The permanent-magnet stepper motor is a low-cost and low-resolution type motor with typical step angles of 7.5° to 15° (48–24 steps/revolution). It operates on the reaction between a permanent-magnet rotor and an electromagnetic field developed by a stator. When a coil of stator winding is energized, an electromagnetic field with a north and south pole is developed. Therefore, the stator carries the magnetic field. The magnetic field can be altered by sequentially energizing the stator coils, which generate rotary motion. The permanent magnet stepper motor with four stator windings is shown in Fig. 10.72 (a) and its rotor is also depicted in Fig. 10.72(b). When a sequence of pulses is applied to the windings of the stepper motor, the shaft of the stepper motor will rotate. For one complete rotation of shaft, the required number of pulses are equal to number of internal teeth of the rotor.

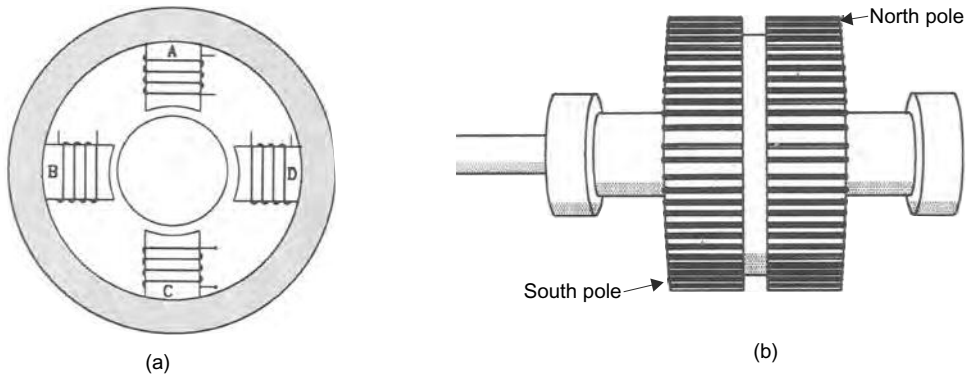


Fig. 10.72 (a) Permanent-magnet stepper motor with four stator windings, and (b) its rotor structure

The *variable-reluctance (VR) stepper motors* are different from the permanent-magnet stepper motors in that they have no permanent-magnet rotor and no residual torque to hold the rotor at any position while turned off. When the stator coils are energized, the rotor teeth will align with the energized stator poles. This type of motor operates on the principle of minimizing the reluctance along the path of the applied magnetic field. The stator field changes when stator windings are sequentially energized and the rotor is moved to a new position. The stator teeth and the rotor teeth of the stepper motor are locked to each other.

When a pulse is applied to the winding, the rotor rotates by an angle, which can be expressed as

$$\theta = \frac{N_s - N_r}{N_s \times N_r} \times 360^\circ$$

where θ —step angle in degree, N_s —number of stator teeth, and N_r —number of rotor teeth. Figure 10.73 shows a typical variable-reluctance stepper motor. If $N_s = 8$, and $N_r = 6$ number of rotor teeth, step angle in degree will be $\theta = \frac{8-6}{8 \times 6} \times 360^\circ = 15^\circ$. Therefore, the rotor will turn each time a pulse is applied. The switching sequence to complete a full of rotation is illustrated in Table 10.9. The motor can rotate in a clockwise direction by repeating the sequence. This motor can also rotate in a reverse direction by changing the sequence of switching.

The stepping sequence is illustrated in Table 10.8 and switching sequence waveform in Fig. 10.74 (a) is known as ‘one-phase on’ stepping. In this method, one phase is switched at a time. The most common method of stepping is ‘two-phase on’ where both phases of the motor are always energized and switching sequence waveform is shown in Fig. 10.74(b). In two-phase, on stepping, the rotor aligns itself between the average north and average south magnetic poles. As both phases are always on, this method gives about 40% more torque than ‘one-phase on’ stepping. The motor can also be operated in half stepping mode by inserting an off state between transitioning phases. Actually, this method reduces the stepper’s full step angle in half. For example, a 15° stepping motor should move 7.5° on each half step. But in half stepping, there will be 20%–30% loss of torque depending on step rate compared to the two-phase on-stepping sequence. As one of the windings is not energized during each alternating half step, the less electromagnetic force can be exerted on the rotor. As a result, there will be huge loss of torque.

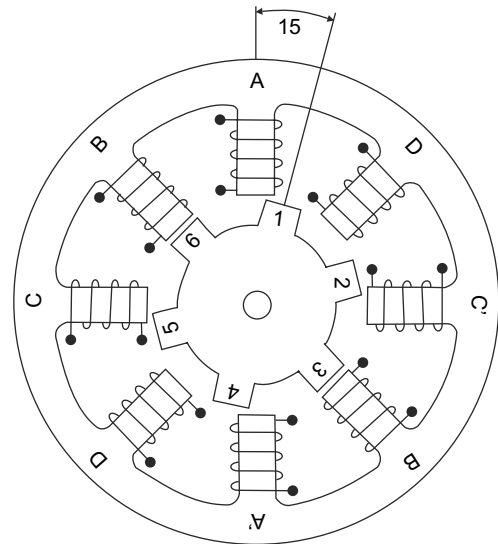


Fig. 10.73 Variable-reluctance stepper motor

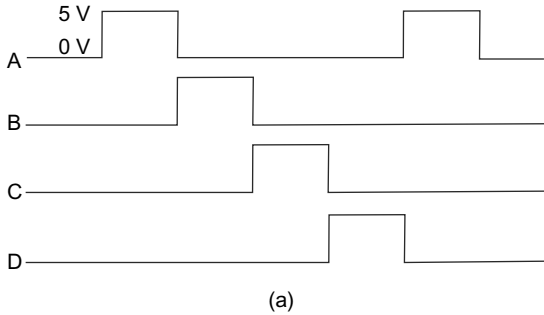


Fig.10.74(a) Switching sequence waveform for single-phase switching

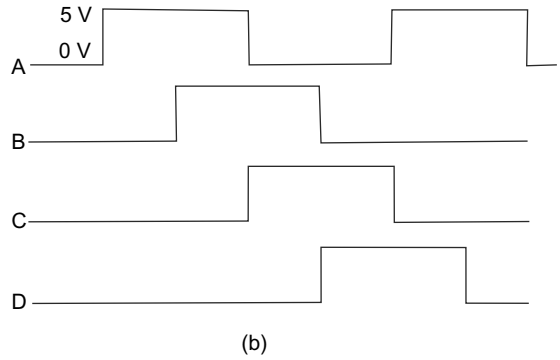


Fig.10.74(b) Switching sequence waveform for two-phase switching

Table 10.8(a) Switching sequence when one phase is switched at a time

Cycle	Phase				Hex-Code	Position
	A	B	C	D		
Cycle -1	1	0	0	0	08	0°
	0	1	0	0	04	15°
	0	0	1	0	02	30°
	0	0	0	1	01	45°
Cycle -2	1	0	0	0	08	60°
	0	1	0	0	04	75°
	0	0	1	0	02	90°
	0	0	0	1	01	105°
Cycle -3	1	0	0	0	08	120°
	0	1	0	0	04	135°
	0	0	1	0	02	150°
	0	0	0	1	01	165°
Cycle -4	1	0	0	0	08	180°
	0	1	0	0	04	195°
	0	0	1	0	02	210°
	0	0	0	1	01	225°
Cycle -5	1	0	0	0	08	240°
	0	1	0	0	04	255°
	0	0	1	0	02	270°
	0	0	0	1	01	285°
Cycle -6	1	0	0	0	08	300°
	0	1	0	0	04	315°
	0	0	1	0	02	330°
	0	0	0	1	01	345°

Table 10.8(b) Switching sequence when two phases are switched at a time

Cycle	Phase			
	A	B	C	D
Cycle-1	1	1	0	0
	0	1	1	0
	0	0	1	1
Cycle-2	1	0	0	1
	1	1	0	0
	0	1	1	0
Cycle-3	0	0	1	1
	1	0	0	1
	1	1	0	0
Cycle-4	0	1	1	0
	0	0	1	1
	1	0	0	1
	1	1	0	0
	0	1	1	0
	0	0	1	1
	1	0	0	1
	1	1	0	0
	0	1	1	0
	0	0	1	1
	1	0	0	1
	1	1	0	0

The hybrid stepper motor is operated under the combined principles of the Permanent Magnet (PM) and Variable Reluctance (VR) stepper motors. The stator core structure of a hybrid motor is same as variable-reluctance stepper motor. The rotor of a hybrid motor is multi-toothed like the variable reluctance motor and contains an axially magnetized concentric magnet around its shaft. The teeth on the rotor provide a better path that helps guide the magnetic flux to preferred locations in the air gap so that the detent, holding and dynamic torque characteristics of the motor will be better than the VR and PM types stepper motors. These motors are more expensive than the PM stepper motor but provide better performance with respect to step resolution, torque and speed. Typical step angles for the hybrid stepper motor range from 3.6° to 0.9° (100–400 steps per revolution).

Stepper-motor operating performance depends upon holding torque, step angle (degree), steps/revolution, operating voltage, and resistance of windings, length, diameter, shaft size and other mechanical specifications available from the manufacturer. This motor is not only used in position or servo-control system but also low-speed operation. The actual speed of a stepper motor is dependent on the step angle and step rate. The number of steps per second is known as the *step rate*. The speed of motor is expressed as

$$N = \frac{\psi \times \text{No. of steps per second}}{6}$$

where N = motor speed in RPM, and ψ = step angle in degrees.

Figure 10.75 shows the interfacing of four-phase stepper motor windings. The four windings A, B, C and D are connected PA_0 , PA_1 , PA_2 and PA_3 respectively. When PA_0 is level '1', the coil A is energized and motor will rotate by one step clockwise. Similarly, the coil B will be energized when PA_1 is in level '1' and again the motor rotates by one step. In the same way, the other phases are energized sequentially as per Table 10.9. The assembly-language program for a stepper motor control in clockwise as well as anti-clockwise rotation is illustrated below.

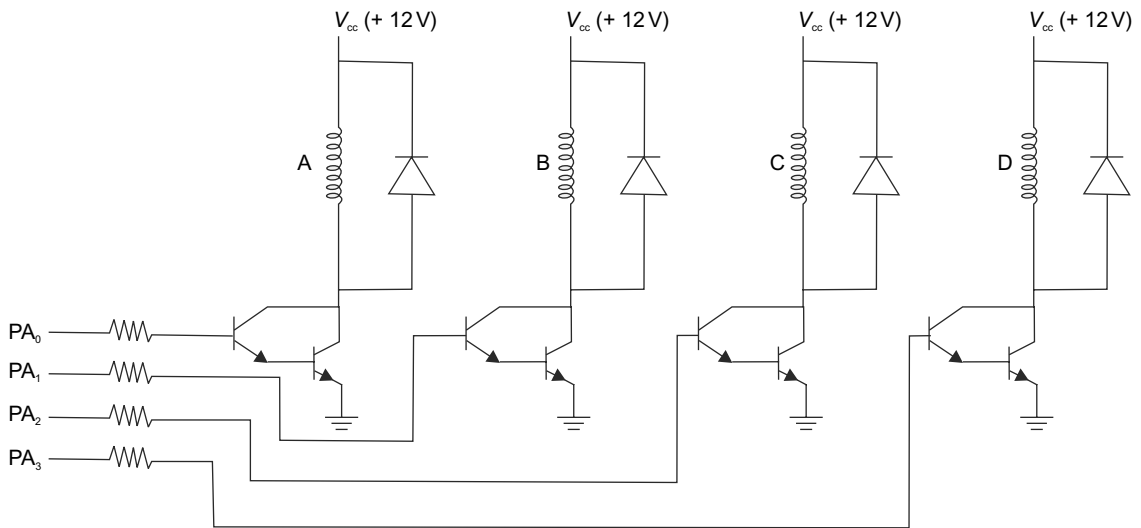


Fig. 10.75 Interfacing circuit

PROGRAM 10.20 for Stepper Motor Control in Clockwise Direction

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	3E, 80		MVI	A, 80H	Load control word of 8255 in accumulator
8002	D3, 03		OUT	03H	Write control word in control word register and initialize ports
8004	3E, 08	START	MVI	A, 08H	Send 08H in Port A
8006	D3, 00		OUT	00H	
8008	CD, 00, 81		CALL	DELAY_1	Call Delay-1 subroutine whose memory location is 8100
800B	3E, 04		MVI	A, 04H	Send 04H in Port A
800D	D3, 00		OUT	00H	
800F	CD, 00, 81		CALL	DELAY_1	Call Delay-1 subroutine
8012	3E, 02		MVI	A, 02H	Send 02H in Port A
8014	D3, 00		OUT	00H	
8016	CD, 00, 81		CALL	DELAY_1	Call Delay-1 subroutine
8019	3E, 01		MVI	A, 01H	Send 01H in Port A
801B	D3, 00		OUT	00H	
801D	CD, 00, 81		CALL	DELAY_1	Call Delay-1 subroutine
8020	C3, 04, 80		JMP	START	Jump to start

PROGRAM 10.21 for Stepper Motor Control in Anticlockwise Direction

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	3E, 80		MVI	A, 80H	Load control word of 8255 in accumulator
8002	D3, 03		OUT	03H	Write control word in control word register and initialize ports
8004	3E, 01	START	MVI	A, 01H	Send 01H in Port A
8006	D3, 00		OUT	00H	
8008	CD, 00, 81		CALL	DELAY_1	Call Delay-1 subroutine whose memory location is 8100
800B	3E, 02		MVI	A, 02H	Send 02H in Port A
800D	D3, 00		OUT	00H	
800F	CD, 00, 81		CALL	DELAY_1	Call Delay-1 subroutine
8012	3E, 04		MVI	A, 04H	Send 04H in Port A
8014	D3, 00		OUT	00H	
8016	CD, 00, 81		CALL	DELAY_1	Call Delay-1 subroutine
8019	3E, 08		MVI	A, 08H	Send 08H in Port A
801B	D3, 00		OUT	00H	
801D	CD, 00, 81		CALL	DELAY_1	Call Delay-1 subroutine
8020	C3, 04, 80		JMP	START	Jump to start

✓ **DELAY SUBROUTINE - 1 (DELAY_1)****Subprogram for Delay Subroutine-1**

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8100	16, 80		MVI	D, 80	Load suitable delay value 80H in Register D
8103	15	Level_1	DCR	D	Decrement Register D by 1
8104	00		NOP		
8105	00		NOP		
8106	C2, 03, 81		JNZ	Level_1	If DE is not equal to zero, jump to Level-1

SUMMARY

- Nowadays microprocessors are used in different applications like instrumentation, industrial control, aerospace engineering, etc. Microprocessors are also used in data-collection terminals, office equipment, business machines, calculators, point-of-sale terminals, and various kinds of data communication equipments, traffic light control system, appliances, motion control, position control, servo control, elevators, automation, electric car, and control of ac/dc machines.
- In this chapter, a broad overview of measurement and display of electrical and physical quantities such as voltage, current, frequency, phase angle, power factor, power, energy, force, displacement, speed, temperature, pressure, torque stress, strain, water level, traffic light control, overvoltage protection, speed control of dc motors are discussed.
- The servo-control mechanism using a stepper motor is also incorporated. The seven-segment display has been discussed as it is used to display any quantity after measurement.

MULTIPLE-CHOICE QUESTIONS

- 10.1 The required torque to move the rotor of a stepper motor one full step is called
 (a) holding torque (b) residual torque
 (c) detent torque (d) none of these
- 10.2 The most popular types of stepper motors are
 (a) shunt and series
 (b) PM and VR
 (c) shunt and compound
 (d) hybrid motor
- 10.3 A thermocouple is a transducer that converts
 (a) voltage to temperature
 (b) temperature to voltage
 (c) analog signals to digital signals
 (d) temperature to resistance
- 10.4 The Bourdon tube is a
 (a) temperature transducer
 (b) pressure transducer
 (c) flow transducer
 (d) none of these
- 10.5 A load cell
 (a) generates a voltage which is proportional to force
 (b) converts a physical displacement into a pressure differential
 (c) converts flow into proportional electrical output signal
 (d) is a piezoelectric transducer
- 10.6 A strain gauge is capable of measuring
 (a) pressure, temperature, resistance
 (b) force, torque, temperature
 (c) displacement, resistance, pressure
 (d) torque, weight, pressure
- 10.7 If the speed of the motor decreases, the thyristor will trigger
 (a) earlier (b) later
 (c) at the same angle (d) none of the above
- 10.8 Isolation can be achieved by
 (a) transformer (b) opto-cupler
 (c) reed relay (d) all
- 10.9 Speed of an induction motor can be controlled by using
 (a) cyclo converter (b) chopper
 (c) PWM converter (d) ac voltage controller
 (e) (a), (c) and (d)
- 10.10 Instrument transformers are
 (a) potential transformers
 (b) current transformers

- (c) power transformers
 - (d) (a) and (b)
- 10.11 The commonly used material for thermocouples
- (a) platinum-rhodium
 - (b) chromel-alumel
 - (c) chromel-copal
 - (d) any of the above
- 10.12 Which device cannot be used to measure pressure?
- (a) Strain gauge
 - (b) Pyrometer
 - (c) LVDT
 - (d) Pirani gauge
- 10.13 Which device is used to measure only pressure?
- (a) Diaphragm
 - (b) Radioactive method
 - (c) Belt-type meter
 - (d) Bubble gauge method
- 10.14 Which of the following is the best method for the measurement of temperature of hot bodies radiating energy in the visible spectrum?
- (a) Thermopile
 - (b) Bolometer
 - (c) Optical pyrometers
 - (d) thermocouple
- 10.15 A differential transformer is a
- (a) constant-pressure transducer
 - (b) constant-displacement transducer
 - (c) variable-inductance transducer
 - (d) variable-pressure transducer
- 10.16 Which one is not a signal conditioner?
- (a) Amplifier
 - (b) Signal converter
 - (c) Equalizing network
 - (d) Damping network
- 10.17 Which of the following can be measured with the help of a piezoelectric crystal?
- (a) Acceleration
 - (b) Temperature
 - (c) Velocity
 - (d) Force
- 10.18 Thermistors have temperature coefficient
- (a) low and positive
 - (b) high and positive
 - (c) low and negative
 - (d) high and negative

SHORT-ANSWER-TYPE QUESTIONS

- 10.1 Is it possible to measure frequency through SID line of microprocessor? If yes, explain.
- 10.2 What is ZCD? Mention some applications of ZCD.
- 10.3 What are the physical quantities measured using a microprocessor ?
- 10.4 How can you measure and display the speed of a motor using a microprocessor ?
- 10.5 Why do you need to execute a delay loop while sending a new 7-segment code to a 7-segment display?
- 10.6 What is a stepper motor? Mention advantages and disadvantages of a stepper motor.
- 10.7 How can you use a stepper motor in position control of a robot arm?

REVIEW QUESTIONS

- 10.1 What is a seven-segment display? Draw the diagram for common-cathode type and common-anode type seven-segment displays and explain their operations. Write some applications of a seven-segment display.

- 10.2 Draw the interfacing circuit of a seven-segment display with its decoder to the 8085-microprocessor. Explain principles of multidigit display. Write assembly language program for
(a) single-digit display (b) two-digit display (c) four-digits display
- 10.3 What are the electrical quantities measured by a microprocessor? Draw the schematic block diagram of dc voltage (5 V) measurement and discuss briefly. Write an assembly-language program for this measurement. What modification is required in hardware and software to measure very high voltage ac and dc?
- 10.4 Draw full-wave precision rectifier circuit and explain its operating principle.
- 10.5 Discuss dc current measurement using a microprocessor with diagrams. Write an assembly-language program for this measurement scheme.
- 10.6 Explain ac current measurement using a microprocessor with a diagram. Write an assembly-language program for this measurement scheme.
- 10.7 Draw the interface connections of a microprocessor-based overvoltage and under voltage relay and explain how it operates. Draw the flowchart for this scheme and write the assembly-language program for voltage protection.
- 10.8 Draw the interface connections of a microprocessor-based overcurrent relay and explain how it operates. Draw the flowchart for this scheme and write the assembly-language program for current protection.
- 10.9 Draw a circuit diagram of ZCD and discuss its operating principle with waveforms.
- 10.10 Discuss a microprocessor-based frequency measurement and display scheme. Draw the flowchart for frequency measurement. Give the assembly-language program for frequency measurement.
- 10.11 Discuss a microprocessor-based technique to measure resistance and reactance of an electric circuit.
- 10.12 Draw a neat sketch and suitable interface for a microprocessor-based scheme to measure the following quantities:
(a) Phase angle (b) Power factor (c) Impedance (d) Resistance
(e) Reactance
- 10.13 Discuss a microprocessor-based active power (W) and energy (WH) measurement scheme.
- 10.14 Discuss a microprocessor-based technique to measure VA and VAR.
- 10.15 Draw a schematic block diagram of any physical quantity measurement.
- 10.16 Discuss the microprocessor-based displacement measurement scheme with a suitable assembly-language program.
- 10.17 Design a strain measurement scheme using the 8085 microprocessor with a suitable assembly-language program.
- 10.18 Explain the microprocessor-based temperature measurement scheme with a suitable assembly-language program.
- 10.19 Draw a neat sketch and suitable interface for a microprocessor-based scheme to measure the following physical quantities:
(a) Force (b) Torque (c) Pressure (d) Resistance
- 10.20 Discuss the microprocessor-based water-level indicator with a suitable assembly-language program.
- 10.21 Draw a traffic light control system and explain its operation briefly. Write the assembly-language program for the same traffic light control system.

- 10.22 Write an assembly-language program to generate a trigger pulse
 (a) a single-phase full wave controlled rectifier
 (b) a single-phase ac voltage controller
 (c) Assume the power supply frequency is 50 Hz
- 10.23 Show the interface connections of a microprocessor based firing-circuit of thyristors which are placed in
 (a) a half-wave controller rectifier (b) a bridge-rectifier (c) ac voltage controller
 Explain why a zero-cross detector is required in this scheme.
- 10.24 Elaborate how to measure, display and control speed of a dc motor using microprocessor.
- 10.25 Explain with a suitable block diagram the control of an induction motor using a microprocessor.
- 10.26 Mention the advantages of a microprocessor-controlled ac and dc drives. Give a list of some industrial applications where such drives are widely used.
- 10.27 Show interface connections for a microprocessor-based stepper motor control scheme. Write the assembly-language program for this scheme.
- 10.28 Design a stepper motor control interface circuit. A stepper motor has a step angle of 15° and is required to rotate at 400 rpm. Determine the pulse rate for the motor. Write a program to rotate the shaft at a speed of 20 revolutions per minute.
- 10.29 How to generate a square wave form using I/O port. Write a program to generate a square wave form.
- 10.30 Draw a interface connections to control firing circuit of thyristors bridge rectifier. Write a program to control the firing angle of thyristors.

Answers to Multiple-Choice Questions

-
- 10.1 (a) 10.2 (b) 10.3 (b) 10.4 (b) 10.5 (a) 10.6 (d) 10.7 (b) 10.8 (b) 10.9 (e)
 10.10 (d) 10.11 (a) 10.12 (b) 10.13 (a) 10.14 (c) 10.15 (c) 10.16 (d) 10.17 (d) 10.18 (d)

Chapter 11

80186, 80286, 80386 and 80486 Microprocessors

11.1 INTRODUCTION

In 1982, the 80186 microprocessor was developed by Intel. This is an improved 8086 with several common functions built in blocks such as clock generator, system controller, interrupt controller, DMA controller, and timer/counter. This processor has 8 new instructions and executes instructions faster than the 8086. Just like the 8086 processor, it has a 16-bit external data bus. It is also available with an 8-bit external data bus, and then the processor name is 80188 microprocessor. The initial clock frequency of the 80186 and 80188 is about 6 MHz. Generally, these processors are used as embedded processors and also used as the CPU of personal computers.

The second generation of the 80186 family, such as the 80C186/C188 processors, have been developed by Intel in 1987. The 80186 was redesigned as a static, stand-alone module known as the 80C186 Modular Core and its pin configuration is compatible with the 80186 family. The high-performance CHMOS III process allowed the 80C186 to operate at twice the clock rate of the NMOS 80186, but this processor consumes less than one-fourth the power.

The 80C186 Modular Core family was further developed and the 80C186XL processor was developed in 1991. The 80C186XL/C188XL is a higher performance and lower power replacement for the 80C186/C188.

The 80186 and 80188 processor series are generally intended for embedded systems such as modems, public and private PBX switching systems, cellular phones, etc. The architecture of 80186 can also be found in many real-time environments such as robotics, automation industry, measurement control systems, sensors and test equipments, fax machines, copiers, printers and medical equipment. Therefore, Intel 80186 processor family, 80186, 80C186XL, 80C186EA/EB/EC, have been accepted in a wide range of applications. In this section, the detailed architecture, addressing modes and instruction sets have been discussed.

11.2 80186 MICROPROCESSOR ARCHITECTURE

The Intel 80186 processor is a high-performance, highly integrated 16-bit microprocessors. Usually, the 80186 processors is intended for embedded systems, as microcontrollers with external memory. To reduce the number of chips in systems, it is required to include clock generator, interrupt controller, timers, wait state generator, DMA channels, and external chip select lines within a chip. The 80186 is a natural successor to the 8086 in personal computers. However, because of its integrated hardware was incompatible with the hardware used in the original IBM PC. The architecture of 80186 processor is shown in Fig. 11.1.

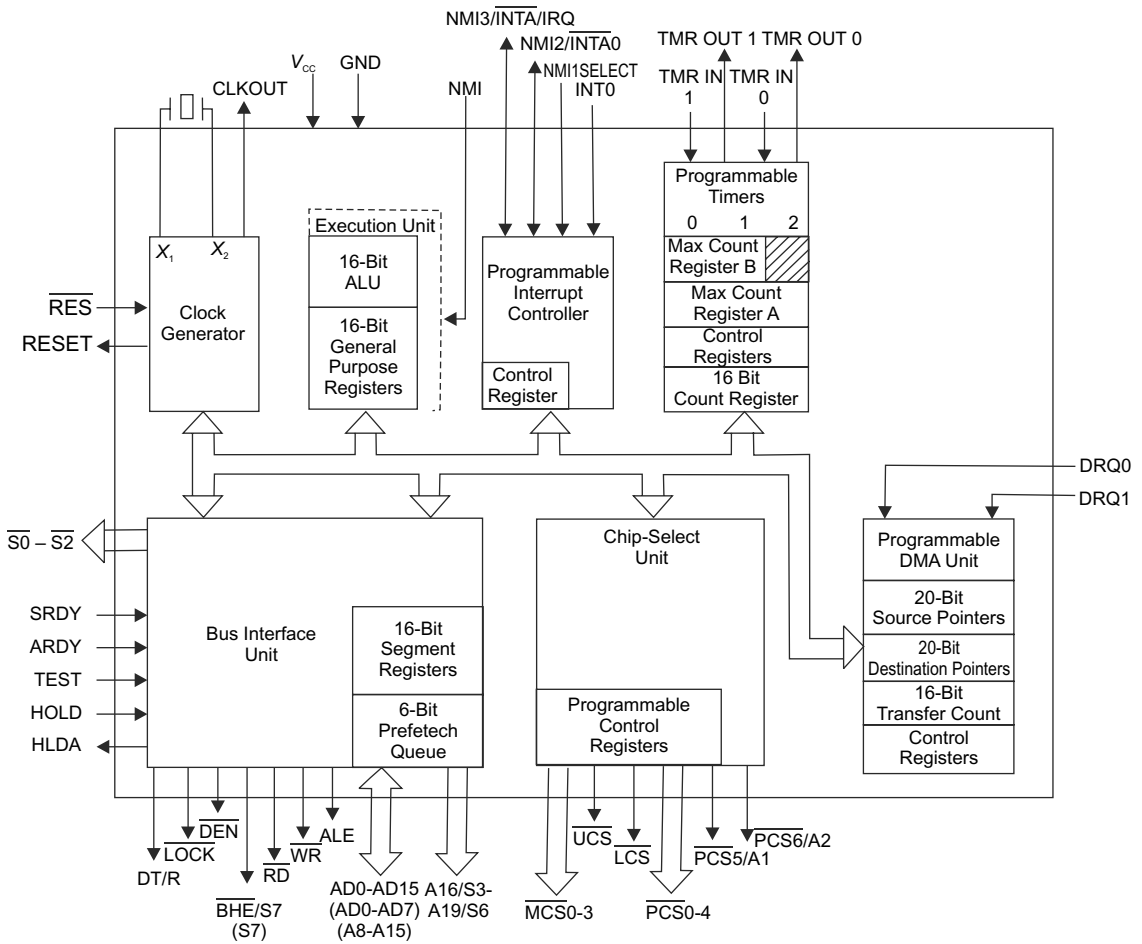


Fig. 11.1 Block diagram of Intel 80186/80188 architecture

The architecture of 80186 is common with the 8086 and 8088 microprocessors. This processor is a very high-integration 16-bit microprocessor. This IC combines 15 to 20 most common microprocessor components onto a single chip and provides twice the performance of standard 8086. The object code of 80186 is compatible with 8086 and 8088 microprocessors but it adds 10 new instructions. The following functional description describes the basic architecture of 80186:

- Clock generator
- Bus interface unit
- DMA controller
- 16-bit programmable timer consisting of three timers
- Programmable interrupt controller
- Chip select unit into a single chip

The 8086, 8088, 80186, and 80286 families all contain the same basic set of registers, instructions, and addressing modes. The 80186 processor is also compatible with the 8086, 8088, and 80286 CPUs.

11.2.1 Register Set

The 80186 base architecture has fourteen registers as shown in Fig. 11.2. These registers are grouped into the five categories such as segment registers, base and index registers, status and control registers and status word.

General Registers

The 80186 processor consists of eight 16-bit general-purpose registers which are used to perform all arithmetic and logical operations. Four of these registers AX, BX, CX, and DX can be used as 16-bit registers or split into pairs of separate 8-bit registers.

Segment Registers

Four 16-bit special-purpose registers (CS, DS, SS and ES) are used to select the segments of memory at any given time. The memory can be immediately addressable as code, stack, and data.

Base and Index Registers

Four of the general-purpose registers BX, BP, SI and DI can be used to determine offset addresses of operands in memory. These registers may contain base addresses or indexes to particular locations within a segment. The different addressing modes of the 80186 microprocessor selects the specific registers to determine the physical address of memory for an operand.

Status and Control Registers

Two 16-bit special-purposes registers (IP and Status word) are also used to record or alter different aspects of the 80186 processor state. The IP (Instruction Pointer) register contains the offset address of the next sequential instruction to be executed, but the status word register contains status and control flag bits as depicted in Fig. 11.2 and Fig. 11.3 respectively.

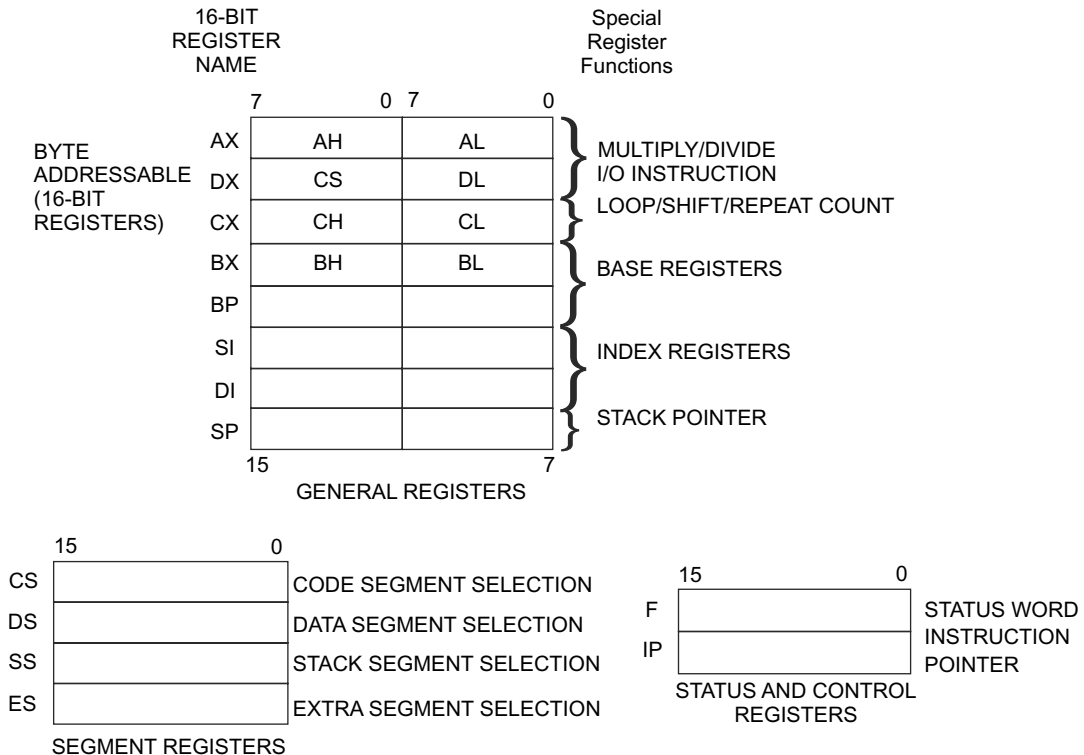


Fig. 11.2 General-purpose register set

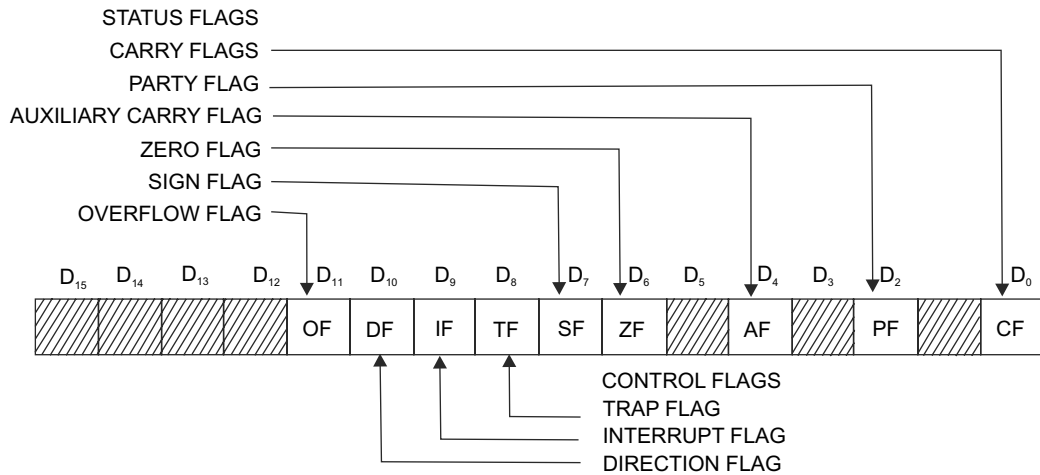


Fig. 11.3 Status word

11.2.2 Status Word

The status word is used to record specific characteristics of the result of logical and arithmetic instructions and controls the operation of the 80186 within a specified operating mode. The status word register is 16 bits wide. Bits 0, 2, 4, 6, 7, and 11 are used for logical and arithmetic operations and bits 8, 9, and 10 are used to control the processor operation. The function of the status word bits is given below.

CF (Carry Flag) The carry flag is set whenever a carry/borrow is generated after arithmetic operations such as addition and subtraction; cleared otherwise.

PF (Parity Flag) This flag is set if low-order 8 bits or the final result after arithmetic/logical operations contain an even number of 1 bit. If there is odd number of ones, it is reset.

AF (Auxiliary Flag) This AF is set whenever there occurs a carry or borrow at the low-order four bits of AL during any operations; cleared otherwise.

ZF (Zero Flag) This flag is set if the result is zero after any operation; otherwise it is reset.

SF (Sign Flag) The sign flag is equal to high-order bit of result. It is set if the result is negative or MSB = 1. It is reset if the result is positive or MSB = 0.

TF (Single-Step Flag) It is used in the processor in single-step mode. When this flag is set, a single-step interrupt occurs after the next instruction executes. TF is cleared by the single-step interrupt.

IF (Interrupt-Enable Flag) When this flag is set, maskable interrupts is enabled cause the CPU to transfer control to an interrupt vector specified location.

DF (Direction Flag) If DF is set, it causes string instructions to auto-decrement the appropriate index register. Clearing DF causes auto-increment. When DF = 1, index register automatically decrement. If DF = 0, index register automatically increment.

OF (Overflow Flag)

When the signed result is too large and cannot be expressed within the number of bits in the destination operand, overflow flag is set; cleared otherwise.

11.2.3 Clock Generator

The 80186 has an on-chip clock generator for both internal and external clock generation. The features of the clock generator are a crystal oscillator, a divide-by-two counter, synchronous and asynchronous ready inputs, and reset circuitry.

Oscillator

Figure 11.4 shows the 80186 crystal oscillator configurations. The oscillator circuit of the 80186 is designed to be used with a parallel resonant fundamental mode crystal. This can be used as the time base for the 80186. The crystal frequency selected will be double the CPU clock frequency. When an external oscillator is used, it can be connected directly to the input pin X1 in lieu of a crystal. The output of the oscillator is not directly available outside the 80186 microprocessor.

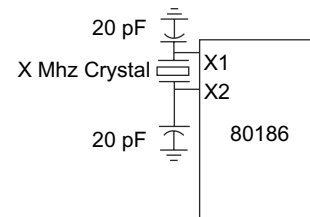


Fig. 11.4 80186 crystal oscillator configuration

Clock Generator

The 80186 clock generator provides the 50% duty cycle processor clock for the 80186. It is possible by dividing the oscillator output by 2 forming the symmetrical clock. When an external oscillator is used, the state of the clock generator will change on the falling edge of the oscillator signal. If an external crystal of 12 or 16 MHz is connected with 80186, then it generates a 6 or 8 MHz internal clock. Hence 80186/80188 is able to operate either at 6 or 8 MHz internal clock. The requirement of crystal frequencies for 80186/80188 is 16 MHz and for advanced version, the 80186/80188 is 12 MHz. The CLKOUT pin gives the processor clock signal for use outside the 80186. This can be used to drive other system components. Always all timings should be referenced to the output clock.

11.2.4 DMA Channels

The 80186 DMA controllers have two independent high speed DMA channels. Data transfers can take place between memory and I/O devices (memory to I/O or I/O to memory) or within the same space (memory to memory or I/O to I/O). Data can be transferred either in 8 bits (bytes) or in 16 bits (words) from even or odd addresses. Each DMA channel consists of a 20-bit source and destination pointer. The content of the index pointer can be incremented or decremented depending upon data-transfer byte or word. When data transfer is one byte, the pointer is incremented by one. If data transfer is one word, the pointer is incremented by two. Each data transfer always takes 2 bus cycles, i.e., minimum 8 T states. The first bus cycle is used to fetch data and the other bus cycle can be used to store data. The maximum data transfer rate is about one Mword/s or 2 MBytes/s. Figure 11.5 shows the DMA block diagram of 80186.

DMA Operation

Each DMA channel consists of six registers in the control block which defines each channel's specific operation. The control register has a 20-bit source pointer, a 20-bit destination pointer, a 16-bit transfer counter, and a 16-bit control word. The source pointer and destination pointer have two-words capacity as shown in Table 11.1 The number of DMA transfers to be performed is specified by the Transfer Count Register (TC). A maximum of 64 K byte or word transfers can be performed with automatic termination. The control word defines the channel's operation. The content of all registers may be changed or modified during any DMA operation. Any changes in the registers must be reflected immediately in DMA operation. The DMA control register is depicted in Fig. 11.6.

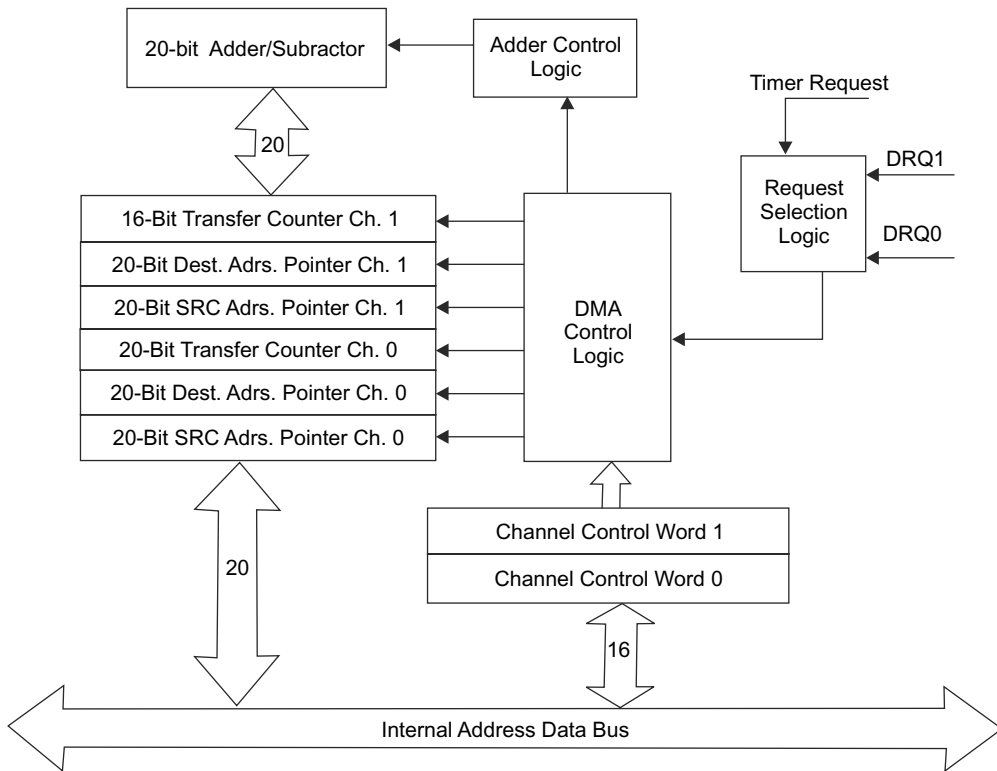
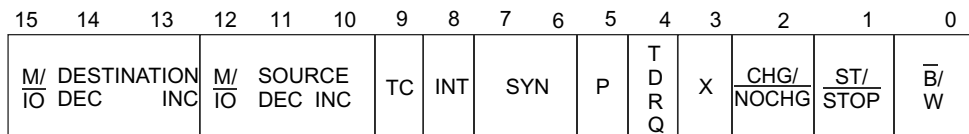


Fig. 11.5 DMA Block diagram of 80186



X = DON'T CARE

Fig. 11.6 DMA control register

Table 11.1 DMA control

<i>Register Name</i>	<i>Register Address</i>	
	<i>Ch. 0</i>	<i>Ch. 1</i>
Control Word	CAH	DAH
Transfer Count	C8H	D8H
Destination Pointer (upper 4 bits)	C6H	D6H
Destination Pointer	C4H	D4H
Source Pointer (upper 4 bits)	C2H	D2H
Source Pointer	C0H	D0H

DMA Channel Control Word Register

Each DMA channel control word specifies the mode of operation for any particular 80186 DMA channel. The control word registers states

- The mode of synchronization
- The number of bytes or words to be transferred
- Interrupts to be generated after the last DMA data transfer
- Ceasing of DMA operation after a programmed number of DMA cycles
- The relative priority of one DMA channel with respect to the other DMA channel
- Whether the source pointer will be incremented, decremented, or maintained constant after each DMA operation
- Whether the source pointer addresses memory or I/O devices
- Whether the destination pointer will be incremented, decremented, or maintained constant after each DMA operation
- Whether the destination pointer will address memory or I/O devices

Usually, the DMA channel control registers can be changed at the same time when the channel is operating. But any changes in the DMA channel control registers made during operation will affect the current DMA transfer.

DMA Control Word Bit Descriptions

✓ **\overline{B}/W : Byte/Word Transfers.** If $\overline{B}/W = 0$, a byte will be transferred. When $\overline{B}/W = 1$, a word will be transferred.

✓ **$\overline{ST/STOP}$: Start/stop Channel.** When $\overline{ST/STOP} = 1$, the specified channel starts data transfer. While $\overline{ST/STOP} = 0$, the specified channel should stop data transfer.

✓ **$\overline{CHG/NOCHG}$: Change/Do not change $\overline{ST/STOP}$ bit.** When $\overline{CHG/NOCHG}$ bit is set during writing to the control word, the $\overline{ST/STOP}$ bit will be programmed by the write to the control word. If $\overline{CHG/NOCHG}$ bit is cleared when writing the control word, the $\overline{ST/STOP}$ bit will not be changed. This bit cannot be stored and will always will be 0 on read.

✓ **\overline{INT}** This bit is used as enable interrupts to CPU on byte count termination.

✓ **\overline{TC}** While \overline{TC} is set, DMA operation will be terminated if the contents of the transfer count register becomes zero. The $\overline{ST/STOP}$ bit will be reset, if \overline{TC} is set. If \overline{TC} bit is cleared, the content of transfer count register in the DMA unit will be decrement for each DMA cycle operation. Although the DMA transfer will not stop when the content of the \overline{TC} register becomes zero.

✓ **\overline{SYN} : Synchronization bits (2 bits)** The operation of synchronization bits is given below:

SYN Bits	Operation
00	No synchronization
01	Source synchronization
10	Destination synchronization
11	Unused

✓ **$\overline{SOURCE\ INC}$** Depending \overline{B}/W , on increment source pointer by 1 or 2 after each transfer. If $\overline{B}/W = 0$, increment source pointer by 1. When $\overline{B}/W = 1$, increment source pointer by 2.

- ✓ **M/\overline{IO}** The source pointer is in M/\overline{IO} space. If M/\overline{IO} is set (1), the source pointer represents memory. When $M/\overline{IO} = 0$, the source pointer represents input/output device address.
- ✓ **DEC** Depending on \overline{B}/W , decrement the source pointer by 1 or 2 after each transfer. If $\overline{B}/W = 0$, decrement the source pointer by 1. When $\overline{B}/W = 1$, decrement the source pointer by 2.
- ✓ **DEST: INC** Increment the destination pointer by 1 or 2 after each transfer. When $\overline{B}/W = 0$, increment the destination pointer by 1. If $\overline{B}/W = 1$, increment the destination pointer by 2.
- ✓ **M/\overline{IO} Destination pointer is in M/\overline{IO} space.** When M/\overline{IO} is set (1), the destination pointer represents memory. If $M/\overline{IO} = 0$, the destination pointer represents an input/output device address.
- ✓ **DEC** Depending on \overline{B}/W , decrement the destination pointer by 1 or 2 after each DMA data transfer.
- ✓ **P** This bit stands for channel priority relative to other channel. Logic level 0 represents low priority and logic level 1 represents high priority. If channels are set at the same priority level, the operation of channels will alternate cycles.
- ✓ **TDRQ** If this bit is reset (0), disable DMA requests from the timer 2. When it is set (1), enable DMA requests from timer 2.
- ✓ **Bit 3** Bit 3 of DMA control register is not used.

11.2.5 Timers

The 80186 microprocessor has three internal 16-bit programmable timers as depicted in Fig. 11.7. Two of these timers are highly flexible and programmable to count the external events and they are connected to four external pins (2 per timer). These two timers can be used to count external events, time external events, and generate non-repetitive waveforms. The third timer is not connected to any external pins. This timer can be

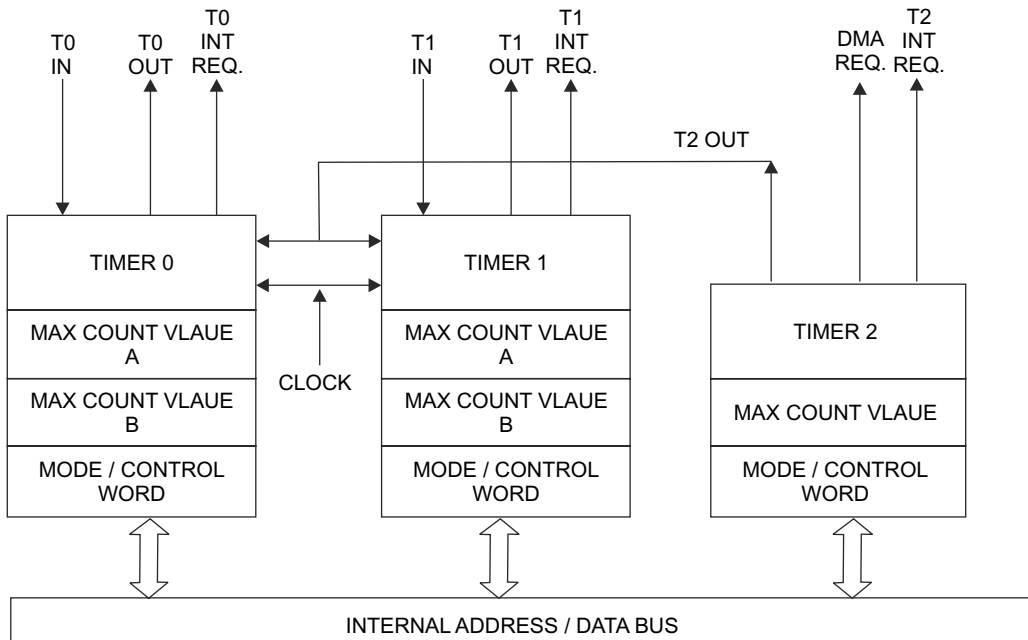


Fig. 11.7 Block diagram of a timer

used to interrupt the 80186 after a programmed interval of time to provide a count pulse to DMA unit, real-time coding and time delay applications.

Timer Operation

Usually, timers are controlled by 11 16-bit registers in the internal peripheral control block. The timer control block format is demonstrated in Table 11.2. The count register contains the current value of the timer and the timer count value can be read or written at any time when the timer is running or not running. The value of this register can be incremented after each timer event. Each of the timers is incorporated with a MAX COUNT register. In general, the MAX COUNT register is used to express the maximum count value of the timer. After reaching the MAX COUNT register value, the timer count value will reset to zero; so that the maximum count value is never stored in the count register itself. A second MAX COUNT register is also present in the timers 0 and 1. This second MAX COUNT register enables the timers 0 and 1 to swap their count between two different MAX COUNT values which is programmed by the programmer.

Table 11.2 Timer control block format

Register name	Register offset		
	Timer-0	Timer-1	Timer-2
Count register	50H	58H	60H
Maximum count A	52H	5AH	62H
Maximum count B	54H	5CH	Not present
Mode/Control word	56H	5EH	66H

As each timer gets a signal on every fourth CPU clock cycle, it can operate at speeds of up to one-quarter the internal clock frequency. External clocking of the timers can be done at the rate of one-quarter of the internal CPU clock rate. When the internal clock frequency is about 8 MHz, the timer operating frequency is 2 MHz. Any timer output can take maximum six clocks to respond to any individual clock or gate input due to internal synchronization and pipelining of the timer circuitry. As the count registers and the maximum count registers are 16 bits wide, 16 bits of resolution are provided in timers. For any read or write operation, the timers will add one wait state to the minimum four-clock bus cycle. This operation is required for synchronization and coordination between the internal timers and the internal bus for the internal data flows. The timers can be programmed in different modes as given below:

- Timer 0, Timer 1 and Timer 2 can be set to halt or continue on a terminal count.
- Timer 0 and Timer 1 can select between internal and external clocks, exchange between MAX COUNT registers and be set to retrigger on external events.
- The timers may be programmed to detect an interrupt on terminal count.

All the above modes of timer operation are selectable through the timer mode/control word register.

Timer Mode/Control Register

Usually, the mode/control register is used to allow the user to program in the specific mode of operation. It is also used to check the current programmed status for any of the three timers. Figure 11.8 shows the timer mode/control register. The operation of timer mode/control register bits are discussed in this section.

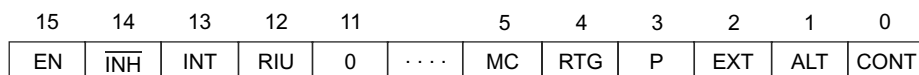


Fig. 11.8 Timer mode/control register

- ✓ **ALT** The ALT bit is used to find out which of two MAX COUNT registers is used for count comparison. If ALT is logic level zero, Register A for that timer is always used, while if ALT is logic level one, the comparison will alternate between Register A and Register B when each maximum count is reached. This ALT bit can also be used to determine the function of the timer output pin. If ALT is logic level zero, the output pin will go LOW for one clock, the clock after the maximum count is reached, If ALT is logic level one, the output pin will reflect the current MAX COUNT register being used. Logic level 0 is used for Register B and logic level 1 is used for Register A.
- ✓ **CONT (Count)** When the CONT bit is set, the associated timer runs continuously. If the CONT bit is reset, the timer will halt upon maximum count. While $CONT = 0$ and $ALT = 1$, the timer will count to the MAX COUNT Register A value.
- ✓ **EXT (External Bit Selects)** This bit selects between internal and external clocking for the timer. When this bit is set, the timer will count low-to-high transitions on the input pin. If it is cleared, it will count an internal clock while using the input pin for control. This external bit signal may be asynchronous with respect to the 80186 clock.
- ✓ **P (Prescaler)** The prescaler bit must be ignored when internal clocking is not selected (EXT-0). When the P bit is at logic level zero, the timer will count at one-fourth the internal clock frequency. If the P bit is at logic level one, the output of Timer 2 can be used as a clock for the timer.
- ✓ **RTG (Retrigger)** The RTG (Retrigger) bit is only active for internal clocking when $EXT = 0$. When $RTG = 0$, the input level gates the internal clock on and off. If the input pin is HIGH, the timer will count otherwise the timer will hold its value. If $RTG = 1$, the input pin is used to detect the low-to-high transitions. The first such transition starts the timer running, clearing the timer value to zero on the first clock, and subsequently increase. The next transitions on the input pin will again reset the timer to zero, from which it will start counting up again. The input signal may be asynchronous with respect to the 80186 clock.
- ✓ **EN (Enable)** The EN (enable) bit gives the programmer control over the timer's RUN/HALT status. If this bit is set, the timer is always enabled to increment depending upon the input pin constraints in the internal clock mode. When this bit is reset, the timer will be inhibited from counting.
- ✓ **INH (Inhibit)** The INH (inhibit) bit is used to allow for selective updating of the EN (enable) bit. If INH is a logic level one while writing to the mode/control word, the state of the EN bit will be modified by the write operation. When INH is a logic level zero during the write, the EN bit will not be affected by the operation. This bit will always be a 0 on a read operation.
- ✓ **INT (Interrupts)** When this bit is set, the INT bit enables interrupts from the timer, which will be generated on every terminal count. If this enable bit is reset after the interrupt request has been generated, but before a pending interrupt is serviced, the interrupt request is latched in the Interrupt Controller.
- ✓ **MC (Maximum Count)** The MC (Maximum Count) bit is set when the timer reaches its final maximum count value. This bit is set regardless of the timer's interrupt-enable bit. The MC bit provides information to the programmer about the ability to monitor timer status through software instead of through interrupts.
- ✓ **RIU (Register In Use)** The RIU (Register In Use) bit is used to indicate which MAX COUNT register is currently being used for comparison to the timer count value. When $RIU = 0$, it indicates Register A. The RIU bit cannot be written and its value is not changed when the control register is written. This bit is always cleared when the ALT bit is zero.
- ✓ **Count Registers** Each timer has a 16-bit count register. The contents of this register can be read or written by the processor at any time. If we write in the register while the timer is counting, the new value will take effect in the current count cycle.

✓ **Max Count Registers** Timer 0 and Timer 1 have two MAX COUNT registers, but Timer 2 has a single MAX COUNT register. The MAX COUNT registers are used to store the number of events that the timer will count. In Timer 0 and Timer 1, the MAX COUNT register can exchange between the two maximum count values whenever the current maximum count is reached.

11.2.6 Interrupt Controller

The 80186 can receive interrupts from both internal and external sources. The internal interrupt controller is able to merge all interrupt requests on a priority basis and provides individual interrupt service by the CPU. Timers and DMA channels are the internal interrupt sources and these sources can be disabled by their own control registers. These can also be disabled by mask bits within the interrupt controller. The 80186 interrupt controller has control registers which can set the mode of operation for the controller.

The interrupt controller can always resolve priority among all pending requests simultaneously. So interrupt service routines for lower priority interrupts might be interrupted by higher priority interrupts. The block diagram of the interrupt controller is shown in Fig. 11.9.

The interrupt controller is able to operate in two different modes such as

- non-iR MX 86 (Master) mode, and
- non-iR MX 86 (Master-Slave) mode.

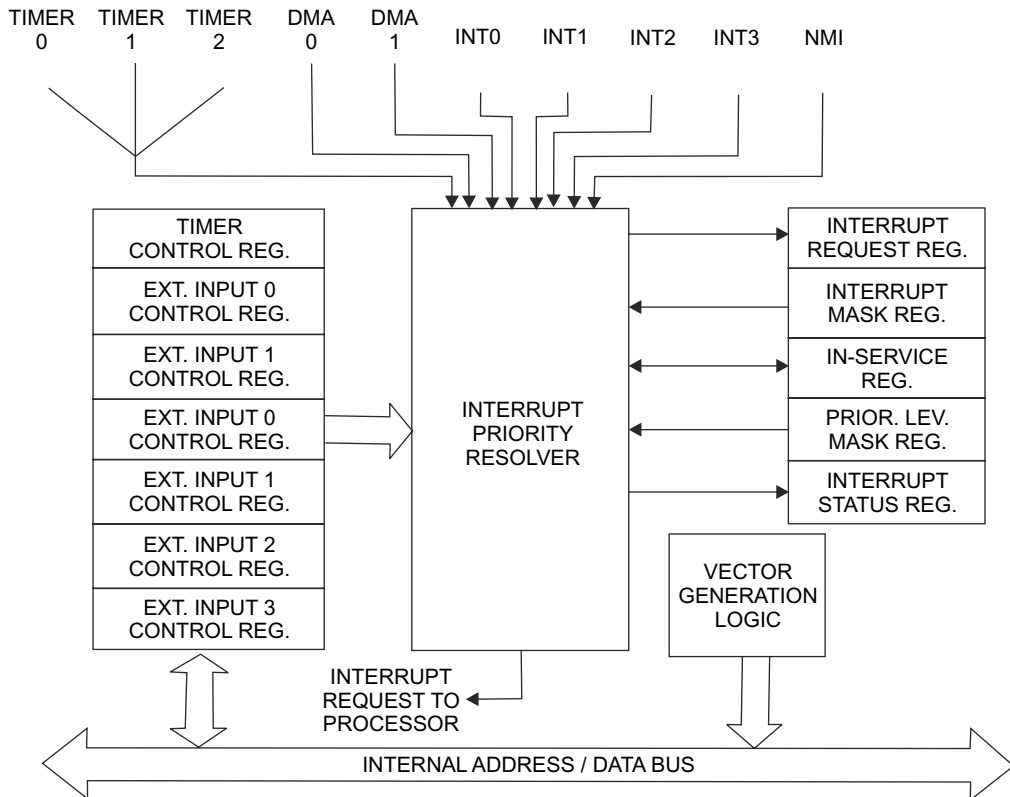


Fig. 11.9 Block diagram of the interrupt controller

Non-iRMX (Master) Mode Operation

The interrupt controller provides five dedicated pins for external interrupt sources. One of these pins is Nonmaskable Interrupt (NMI). This pin is used for power-fail interrupts, etc. The other four pins may function either as four interrupt input lines with internally generated interrupt vectors, as an interrupt line and an interrupt acknowledge line along with two other input lines with internally generated interrupt vectors. While the interrupt lines are used in cascade mode, the 80186 interrupt controller will not generate internal interrupt vectors. In the cascade mode, external sources use externally generated interrupt vectors. If an interrupt is acknowledged, two INTA cycles are initiated and the vector is used to read into the 80186 on the second cycle. In cascade mode, external 8259A programmable interrupt controllers can be interfaced with the 80186 processor.

The basic modes of operation of the interrupt controller in non-iRMX mode (Master) are like the 8259A. The interrupt controller responds identically to internal interrupts in all three modes, namely, fully nested mode, cascade mode and special fully nested mode. But the difference of three modes is only in the interpretation of function of the four external interrupt pins. The interrupt controller can be set into one of the three modes after programming the INT0 and INT1 control registers.

Fully Nested Mode

In the fully nested mode, four pins are used as direct interrupt requests. The vectors for these four inputs are generated internally. An in-service bit is provided for every interrupt source. If a lower-priority device requests an interrupt while the in-service bit (IS) is set, no interrupt will be generated by the interrupt controller. While interrupts are received and enabled, higher-priority interrupts will be serviced. When a service routine is completed, the proper IS bit must be reset by writing the proper pattern to the EOI register. An EOI command is issued at the end of the service routine just before the issuance of the return from interrupt instruction.

Cascade Mode

In cascade mode, the 80186 has four interrupt pins and two of them have dual functions. In the fully nested mode, the four pins are used as direct interrupt inputs and the corresponding vectors are generated internally. In the cascade mode, the four pins are configured into interrupt input-dedicated acknowledge signal pairs. The interconnection between 80186 and 8259A is shown in Fig. 11.10. INT0 is an interrupt input which is used to interface an 8259A and INT2/INTA0 provides the dedicated interrupt acknowledge signal. The same is true for INT1 and INT3/INTA1. The primary cascade mode allows the capability to serve up to 128 external interrupt sources through the use of external master and slave 8259As. Three levels of priority are created, requiring priority resolution in the 80186 interrupt controller, the 8259A masters, and the 8259A slaves.

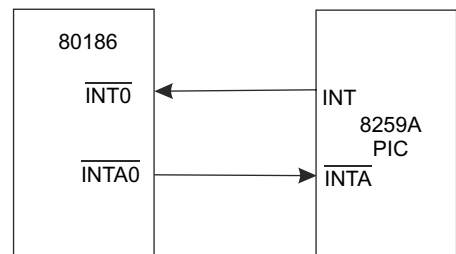


Fig. 11.10 Interrupt connection in cascade mode

Special Fully Nested Mode

The interrupt controller operates in this mode after setting the SFNM bit in INT0 or INT1 control register. This mode enables complete nest ability with the external 8259A masters. Usually, any interrupt request from an interrupt source will not be recognized until the in-service bit for that source is reset.

Interrupt Controller Registers

The interrupt controller register is depicted in Fig. 11.11. It consists of fifteen registers such as in-service register, interrupt request register, mask register, priority mask register, interrupt status register, timer control register, DMA 0, 1 control registers, INT0-INT3 control registers, poll register, poll status register and EOI register. All registers can both be read or written to unless specified.

iRMX 86 Mode

The interrupt controller has a special iRMX 86 compatibility mode that allows the use of the 80186 within the iRMX 86 operating system interrupt structure. This interrupt model requires one master and multiple slaves 8259A in cascade connection. In the iRMX mode, the internal 80186 interrupt controller will be used as a slave controller to an external master interrupt controller. Figure 11.12 shows the iRMX 86 interrupt controller interconnection. The INT0 input is used as the 80186 CPU interrupt input. INT3 functions as an output to send the 80186 slave-interrupt request to one of the eight master PIC inputs. To get correct master–slave interfacing, decoding of slave address CAS0–CAS2 are required. INT1 is used as slave-select input. INT2 is used as an acknowledge output and it is used to drive the INTA input of an 8259A.

The iRMX mode of operation allows nesting of interrupt requests. Vector generation in the iRMX mode is exactly like that of an 8259A slave. In iRMX mode, the specific EOI command operates to reset an in-service bit of a specific priority. All control and command registers such as interrupt vector register, specific EOI register, mask register, priority-level mask register, in-service register, interrupt request register, and Level 0–Level 5 control registers are

	OFFSET
INT3 CONTROL REGISTER	3EH
INT2 CONTROL REGISTER	3CH
INT1 CONTROL REGISTER	3AH
INT0 CONTROL REGISTER	38H
DMA 1 CONTROL REGISTER	36H
DMA 0 CONTROL REGISTER	34H
TIMER CONTROL REGISTER	32H
INTERRUPT CONTROLLER STATUS REGISTER	30H
INTERRUPT REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY MASK REGISTER	2AH
MASK REGISTER	28H
POLL STATUS REGISTER	26H
POLL REGISTER	24H
EOI REGISTER	22H

Fig. 11.11 Interrupt controller register

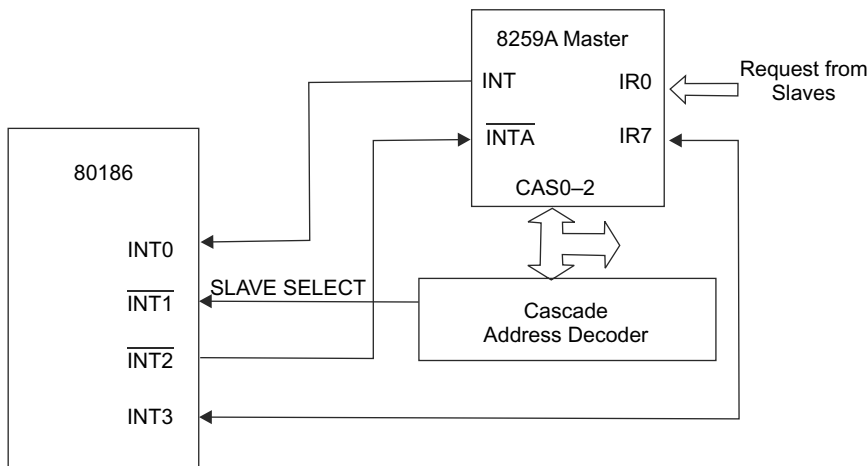


Fig. 11.12 iRMX86 interrupt controller

located inside the internal peripheral control block. Figure 11.13 shows the interrupt controller registers in iRMX86 mode.

11.2.7 Memory Organization

The 80186 has 20-bit address lines and it can directly address $2^{20}=1\text{MB}$ memory. The memory of 80186 is organized in sets of segments. Each segment is available in a linear contiguous sequence of up to 64K bytes. Memory is addressed using a two-component address such as a 16-bit base segment and a 16-bit offset. The 16-bit base values are stored in any one of four internal segment registers: code segment, data segment, stack segment, and extra segment. The physical address is calculated by shifting the base value left by four bits and adding the 16-bit offset value to determine a 20-bit physical address just like the 8086 processor.

	OFFSET
LEVEL 5 CONTROL REGISTER (TIMER 2)	3AH
LEVEL 4 CONTROL REGISTER (TIMER 1)	38H
LEVEL 3 CONTROL REGISTER (DMA 1)	36H
LEVEL 2 CONTROL REGISTER (DMA 0)	34H
LEVEL 0 CONTROL REGISTER (DMA 0)	32H
INTERRUPT REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY MASK REGISTER	2AH
MASK REGISTER	28H
SPECIFIC EOI REGISTER	22H
INTERRUPT VECTOR REGISTER	20H

Memory Chip Select

The 80186 has six memory chip select outputs for three different address spaces such as upper memory, midrange memory and lower memory. One memory chip select signal is used for upper memory, four memory chip select signals are provided for midrange memory and one memory chip select signal is used for lower memory.

The range for each memory chip select signal is programmable and the range can be set to 1K, 2K, 4K, 8K, 16K, 32K, 64K, 128K and 256K. In addition, the starting or base address of the midrange memory chip select can also be selected. At a time, only one chip select can be programmed to be active for any memory location. Generally, all chip select sizes are in bytes though the memory of 80186 is arranged in words. For example, sixteen $64\text{K} \times 1$ memories are used to develop the 128K memory block.

Fig. 11.13 iRMX86 interrupt controller registers

UCS (Upper Memory CS)

The 80186 provides the upper memory chip select signal \overline{UCS} to select the top of memory. Generally, the top of memory is used as the system memory as the 80186 starts executing at memory location FFFF0H after reset. The upper limit of memory is represented by the chip select and it is FFFFH, but the lower limit is programmable. Actually, the lower limit and the size of the select block can be varied by programming as given in Table 11.3.

Table 11.3 Upper memory chip select programming values

Starting Address	Memory Block Size	UMCS Value
FFC00H	1K	FF8H
FF800H	2K	FFB8H
FF000H	4K	FF38H
FE000H	8K	FE38H
FC000H	16K	FC38H
F8000H	32K	F838H
F0000H	64K	F038H
E0000H	128K	E038H

\overline{MCS} (Mid-range Memory \overline{CS})

The 80186 provides four midrange memory chip select signals which are active within a user locatable memory. This memory block may be within 1 Mbyte memory address space exclusive the area defined by upper-memory chip-select block and lower-memory chip-select block. The base address and the size of the select memory block for programming are shown in Table 11.4.

Table 11.4 Midrange lower-memory chip-select programming values

Total Memory Block Size	Individual Memory Block Size	MMCS Bits
8K	2K	0000001B
16K	4K	0000010B
32K	8K	0000100B
64K	16K	0001000B
128K	32K	0010000B
256K	64K	0100000B
512K	128K	1000000B

 \overline{LCS} (Lower Memory)

The 80186 provides the lower-memory chip-select signal \overline{LCS} to select the bottom of memory. Usually, the bottom of memory is used as the interrupt vector table starting from memory location 00000H. The lower limit of memory is always defined by this chip select and it is 0H. The upper limit and the size of the select memory block can be defined by programming as given in Table 11.5.

Table 11.5 Lower-memory chip-select programming values

Starting Address	Memory Block Size	LMCS Value
003FFH	1K	0038H
007FFH	2K	0078H
00FFFH	4K	00F8H
01FFFH	8K	01F8H
03FFFH	16K	03F8H
07FFFH	32K	07F8H
0FFFFH	64K	0FF8H
1FFFFH	128K	1FF8H
3FFFFH	256K	3FF8H

Peripheral Chip Selects

In general, the 80186 generates chip-select signals for up to seven peripheral devices. These chip selects are active for seven adjacent blocks of 128 bytes above a programmable base address. This base address can be located in either memory or I/O space. The 80186 generates seven \overline{CS} lines called \overline{PCS}_{0-6} . The base address is user-programmable but it will be a multiple of 1k bytes. Therefore, the least significant 10 bits of the starting address are always 0.

Usually \overline{PCS}_5 and \overline{PCS}_6 are programmed to provide latched address bits A_1, A_2 . If these signals are programmed, they cannot be used as peripheral selects. These outputs can be connected directly to the A_0, A_1 pins and are used to select internal registers of 8-bit peripheral chips. Then the hardware interface becomes simplified as the 8-bit registers of peripherals are simply treated as 16-bit registers located on even boundaries in I/O or memory space. In this case the lower 8-bits of the register are significant, but the upper 8-bits are 'don't cares'

The starting address of the peripheral chip-select block can be described by the Peripheral Chip-Select (PACS) register. This register is located at offset A4H in the internal control block. Bits 15–6 of this register correspond to bits 19–10 of the 20-bit Programmable Base Address (PBA) of the peripheral chip-select block. Bits 9–0 of the PBA of the peripheral chip-select block are all zeros. If the chip-select block is located in I/O space, bits 12–15 must be programmed zero, since the I/O address is only 16 bits wide. Table 11.6 shows the address range of each peripheral chip select with respect to the PBA contained in PACS register.

Table 11.6 PCS address range

PCS Line	Active between locations	
PCS0	PBA	PBA+127
PCS1	PBA+128	PBA+255
PCS2	PBA+256	PBA+383
PCS3	PBA+384	PBA+511
PCS4	PBA+512	PBA+639
PCS5	PBA+640	PBA+767
PCS6	PBA+768	PBA+895

11.3 PIN DESCRIPTION OF 80186

The 80186 is a 68-pin IC and it is available in Plastic Leaded Chip Carrier (PLCC), ceramic Leadless Chip Carrier (LCC) and Pin Grid Array (PGA) packages. Figure 11.14 shows the pin diagram of 80186 in ceramic leadless chip carrier package. The pin functions are discussed elaborately as given below:

V_{CC} (Input) +5 volt power supply

V_{SS} (Input) Ground

✓ **RESET (Output)** The output of RESET pin indicates that the 80186 CPU is being reset. It can also be used as a system reset. When it is logic level HIGH (1) and synchronized with the processor clock, it lasts an integer number of clock periods corresponding to the length of the \overline{RES} signal.

✓ **X1, X2 (Input)** The X1 and X2 are crystal input terminals. These pins provide an external connection for a fundamental mode parallel resonant crystal for the internal crystal oscillator, but X1 can be used for interfacing an external clock instead of a crystal.

✓ **CLKOUT (Output)** The clock output provides the system clock with a 50% duty-cycle waveform. Usually, the clock signal (CLKOUT) is generated when the input or oscillator frequency is internally divided by two, so that frequency of CLKOUT is one half of the crystal oscillator.

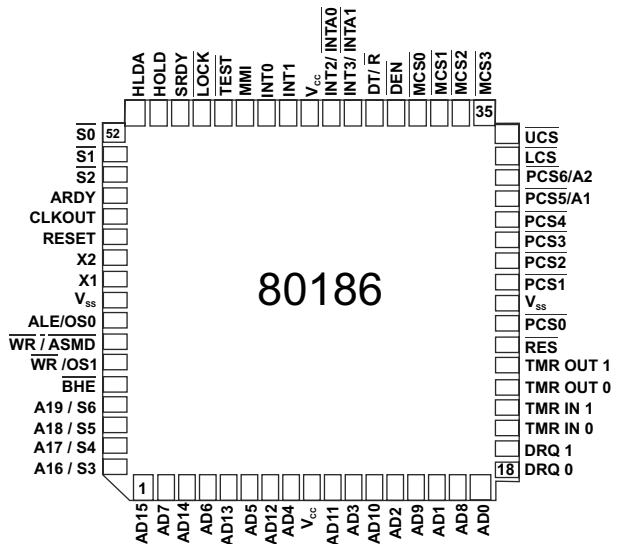


Fig. 11.14 Pin diagram of 80186

- ✓ **\overline{RES} (I)** When input is logic level LOW (0), the system will be reset, so that the 80186 immediately terminates its present operation, clears the internal logic, and enters into a dormant state. For proper reset operation, \overline{RES} must be low for at least 50 ms. This signal may be asynchronous to the 80186 clock. Generally, this pin is connected to an RC circuit which generates a reset signal after application of power supply. When \overline{RES} occurs, the 80186 will drive the status lines to an inactive level for one clock, and then tri-state them.
- ✓ **\overline{TEST} (Input)** The function of \overline{TEST} input signal can be examined by the WAIT instruction. When the \overline{TEST} input is HIGH, WAIT execution starts and instruction execution will be suspended. This input signal is synchronized internally.
- ✓ **$TMR\ IN0$, $TMR\ IN1$ (Input)** These are timer input signals which are used either as clock or control signals, depending upon the programmed timer mode. Usually, these inputs are active HIGH or logic level 1 and internally synchronized.
- ✓ **$TMR\ OUT\ 0$, $TMR\ OUT\ 1$ (Output)** These are timer output signals which are used to provide a single pulse or a continuous waveform generation, depending upon the selected timer mode.
- ✓ **$DRQ0$, $DRQ1$ (Input)** The $DRQ0$ and $DRQ1$ are DMA request inputs for two internal DMA channels. These pins are driven HIGH by an external device whenever it is required to perform data transfer through DMA channel 0 or 1. These signals are active HIGH, level-triggered, and internally synchronized.
- ✓ **NMI (Input)** NMI (Non-Maskable Interrupt) is an positive edge-triggered input which causes a type 2 interrupt. It is not maskable internally. This signal is latched internally and is also internally synchronized. NMI duration of one clock or more will guarantee service.
- ✓ **$INT0$, $INT1$ (Input) $INT2/INTA0$ (I/O), $INT3/INTA1$ (I/O)** The maskable interrupt requests can be requested by one of $INT0$, $INT1$, $INT2/INTA0$ and $INT3/INTA1$ pins. Usually, these input pins are active HIGH and are synchronized internally. $INT2$ and $INT3$ can be configured through software to provide active-LOW interrupt-acknowledge output signals. All interrupt inputs can be configured through software to be either edge-triggered or level-triggered. To ensure the interrupt operation, all interrupt requests must remain active until the interrupt is acknowledged.
- ✓ **$A19/S6$, $A18/S5$, $A17/S4$, $A16/S3$ (Output)** $A19$ – $A16$ are address bus outputs and S_6 – S_3 are bus cycle statuses. These signals are used as the four most significant address bits during T_1 . Generally, these signals are active HIGH. During T_2 , T_3 , T_W and T_4 , status information is available on these lines. S_6 differentiates processor cycle and DMA cycle. $S_6 = 0$ indicates processor cycle and $S_6 = 1$ indicates DMA Cycle. S_3 , S_4 , and S_5 are defined as LOW during T_2 – T_4 .
- ✓ **AD_{15} – AD_0 (I/O)** These signals are time multiplexed address/data bus. During T_1 , the 80186 places A_{15} to A_0 signals on these pins to locate the memory or I/O address. During T_2 , T_3 , T_W and T_4 , these lines work as data bus. The bus is always active HIGH.
- ✓ **\overline{BHE}/S_7 (Output)** The bus high enable signal can be used to determine if data is to be enabled onto the most significant half of the data bus, pins D_{15} – D_8 during T_1 clock cycle. For any read, write, an interrupt acknowledge operations \overline{BHE} is LOW during T_1 and a byte is to be transferred on the higher half of the bus. The S_7 status is available during T_2 , T_3 and T_4 clock cycle. S_7 is always at logic level 1. Hence no latch is required to de-multiplex S_7 . The function of processor depends on \overline{BHE} and A_0 as shown in Table 11.7.
- ✓ **ALE/QS_0 (Output)** The ALE/QS_0 (Address Latch Enable/Queue Status 0) is provided by the 80186 to latch the address into the 8282 / 8283 address latches. When ALE is active HIGH, addresses are valid on the trailing edge of ALE .

Table 11.7 \overline{BHE} and A_0 encoding

\overline{BHE}	A_0	Function
0	0	Word Transfer
0	1	Byte transfer on upper half of data bus (D15–D8)
1	0	Byte transfer on lower half of data bus (D7–D0)
1	1	Reserved

✓ **\overline{WR}/QSI (Output)** The \overline{WR}/QSI (Write Strobe/Queue Status 1) is used to indicate that the data on the bus is to be written into a memory or an I/O device. The \overline{WR} signal is active for T_2 , T_3 , and T_W of any write cycle. When the 80186 is in queue status mode, the ALE/QSO and \overline{WR}/QSI pins give information about processor and instruction queue interaction as depicted in Table 11.8.

Table 11.8 Queue operation based on QSI and QSO

QSI	QSO	Queue Operation
0	0	No queue operation
0	1	First opcode byte fetched from the queue
1	1	Subsequent byte fetched from the queue
1	0	Empty the queue

✓ **$\overline{RD} / \overline{QSMD}$ (O)** This signal is used indicate that the 80186 is performing a memory or I/O read cycle. The \overline{RD} is active LOW during T_2 , T_3 and T_W of any read cycle. \overline{RD} is driven HIGH for one clock during reset, and then the output driver is floated. During RESET, the pin is sampled to determine whether the 80186 should provide ALE, \overline{WR} , and \overline{RD} , or the queue-status should be provided. \overline{RD} must be connected to GND (ground) to provide queue-status data.

✓ **ARDY (Input)** The asynchronous ready input signal is used to inform the 80186 processor that the addressed memory or I/O device will complete a data transfer. This signal is internally synchronized by the 80186 on rising edge of the clock. The ARDY input pin will accept an asynchronous input, and is active HIGH. When this pin is connected to +Vcc (+5 V), the 80186 functions normally. If this pin is connected to ground, the 80186 enters WAIT states.

✓ **SRDY (Input)** The synchronous ready input signal should be synchronized externally to the 80186. This signal can be used as the ready input. This line is active HIGH. When this line is connected to Vcc, no WAIT states are inserted. The ARDY (Asynchronous Ready) or SRDY (synchronous ready) must be active before a bus cycle is terminated. When SRDY line is unused, it may remain connected to +5 V or it may be connected to ground.

✓ **\overline{LOCK} (Output)** The \overline{LOCK} output signal is used to prevent other bus masters from accessing the system bus. Usually, the LOCK signal is requested by the \overline{LOCK} prefix instruction while \overline{LOCK} is active LOW for the duration of locked instruction. This is activated at the beginning of the first data cycle connected with the instruction following the \overline{LOCK} prefix. It will be active until the completion of the instruction following the \overline{LOCK} prefix.

✓ **$\overline{S2}$, $\overline{S1}$, $\overline{S0}$ (Output)** The bus cycle status output signals S2–S0 are used to provide bus-transaction information as given in Table 11.9.

Table 11.9 80186 Bus Cycle Status Information

\overline{S}_2	\overline{S}_1	\overline{S}_0	Bus Cycle Initiated
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O
0	1	0	Write I/O
0	1	1	Halt
1	0	0	Instruction Fetch
1	0	1	Read Data from Memory
1	1	0	Write Data to Memory
1	1	1	Passive (no bus cycle)

\overline{S}_2 can be used for a logical M/\overline{IO} signal, and \overline{S}_1 is used for DT/\overline{R} indicator. These status lines are active HIGH for one clock during reset, and then floated until a bus cycle begins.

- ✓ **HOLD (Input)** The HOLD signal is used to indicate whether any other bus master is requesting the local bus. This input is active HIGH and it may be asynchronous with respect to the 80186 clock.
- ✓ **HLDA (Output)** After receiving a HOLD request, the 80186 will issue an HLDA hold acknowledge signal at the end of T_4 or T_1 . Then the 80186 will float the local bus and control lines.
- ✓ **\overline{UCS} (Output)** The \overline{UCS} (Upper Memory Chip Select) output signal is an active LOW whenever a memory reference is made to select the upper portion of the memory map. This output line is software programmable to enable memory size of 1KB–256KB.
- ✓ **\overline{LCS} (Output)** The \overline{LCS} (Lower Memory Chip Select) is active LOW whenever a memory reference is made to select the lower portion of the memory map. This output line is also software programmable to enable memory size of 1 KB–256 KB.
- ✓ **\overline{MCS}_{0-3} (Output)** The midrange memory chip select signals, \overline{MCS}_{0-3} are active LOW when a memory reference is made to the specified midrange portion of memory (8K–512K). These lines are not floated during bus HOLD and are software programmable.
- ✓ **\overline{PCS}_{0-4} (Output)** The peripheral chip select signals \overline{PCS}_{0-4} are active LOW when a reference is made to the specified peripheral area about 65k byte I/O space. These lines are not floated during bus HOLD, but the address ranges are software programmable.
- ✓ **$\overline{PCS5}/A1$ (Output)** The peripheral chip select 5 or latched address signal A1 can be programmed to provide a sixth peripheral chip select, or to supply an internally latched A1 signal. The address range activated by PCS5 is software programmable.
- ✓ **$\overline{PCS6}/A2$ (Output)** The peripheral chip select 6 or latched address signal A2 may be programmed to provide a seventh peripheral chip select, or to give an internally latched A2 signal. The address range activated by PCS6 is also software programmable.
- ✓ **DT/\overline{R} (Output)** The DT/\overline{R} (Data Transmit/Receive) signal controls the direction of data flow through the external data bus transceiver. If DT/\overline{R} is LOW, data is transferred to the 80186. When DT/\overline{R} is HIGH, the 80186 writes data on the data bus.
- ✓ **\overline{DEN} (Output)** The \overline{DEN} (Data Enable) is used enable the external data bus transceiver output. This is active LOW during each memory and I/O access. If \overline{DEN} is HIGH, DT/\overline{R} changes state.

11.4 ADDRESSING MODES OF 80186

The 80186 provides eight different types of addressing modes to specify operands. The operand (data) is used in the instruction to specify the addressing modes. The addressing mode byte is always the second byte of the instruction code. Usually, the two addressing modes are provided for instructions that operate on the register or immediate operands:

✓ **Register Operand Addressing Mode** The operand is located in one of the 8- or 16-bit general registers. The examples of 8-bit and 16-bit registers addressing are MOV AL, BL and MOV AX, BP respectively.

✓ **Immediate Operand Addressing Mode** The operand is included in the instruction. The examples of immediate operand mode are MOV AL, FFH and MOV BX, 2354H.

The remaining six modes are provided to specify the location of an operand in a memory segment. Any memory operand address consists of a segment base and an offset. The segment base must be supplied by a 16-bit segment register. The offset is called the *effective address*, and it is determined by summing any combination of the following three address elements as given below:

- the 8- or 16-bit immediate displacement provided in the instruction
- the contents of base registers such as BX or BP base registers
- the contents of index registers such as SI or DI index registers

The combinations of the above three address elements define the six memory addressing modes as given below.

✓ **Direct Addressing Mode** The operand's offset is contained in the instruction as an 8- or 16-bit displacement element. The example of direct addressing mode instruction is MOV AX, [3000].

✓ **Register Indirect Addressing Mode** In this addressing mode, the operand's offset is stored in one of the registers SI, DI, BX, or BP. The example of register indirect addressing mode instruction is MOV AX, [BX].

✓ **Based Addressing Mode** The operand's offset is the sum of an 8- or 16-bit displacement and the contents of a base register (BX or BP). The example of based addressing mode instruction is MOV AX, [BX+66].

✓ **Indexed Addressing Mode** In this addressing mode, the operand's offset is the sum of an 8-bit or 16-bit displacement and the contents of an index register (SI or DI). The example of indexed addressing mode instruction is MOV AX, [SI + 0300].

✓ **Based Indexed Addressing Mode** The operand's offset is the sum of the contents of a base register and an index register. The example of based indexed addressing mode instruction is MOV AX, [BX+SI].

✓ **Based Indexed Mode with Displacement Addressing Mode** In this addressing mode the operand's offset is the sum of a base register's contents, an index register's contents, and an 8- or 16-bit displacement. The example of based indexed with displacement addressing mode instruction is MOV AX, [BX+SI+0200].

11.5 DATA TYPES OF 80186

The 80186 microprocessor can directly support the different data types such as integer, ordinal, pointer, string, ASCII and BCD.

- ✓ **Integer** Integer data is a signed binary numeric value represented in an 8-bit byte or a 16-bit word. During all arithmetic operations, assume that the data is a 2's complement representation. The signed 32-bit and 64-bit integer's data can be supported by using a numeric data processor.
- ✓ **Ordinal** It is an unsigned binary numeric value contained in an 8-bit byte or a 16-bit word.
- ✓ **Pointer** The pointer is a 16-bit or 32-bit quantity, and it consists of a 16-bit offset element or a 16-bit segment base element in addition to a 16-bit offset element.
- ✓ **String** String is a contiguous sequence of bytes or words. Any string may contain from 1 KB to 64 KB.
- ✓ **ASCII** It is a byte representation of alphanumeric and control characters using the ASCII standard of character representation.
- ✓ **BCD** BCD is an unpacked byte representation of the decimal digits 0–9.
- ✓ **Packed BCD** It is a packed byte representation of two decimal digits (0–9). One digit is stored in each nibble (4 bits) of the byte.
- ✓ **Floating Point** Floating point is used for a signed 32-bit, 64-bit or 80-bit real number representation. Usually, floating point operands are supported by using a numeric data processor.

11.6 INSTRUCTION SET OF 80186

Intel 80186 is the improved version of 8086, hence it has faster instruction execution time compared to the 8086 microprocessor. The 80186 processor is compatible with all instruction of 8086 microprocessors and a few new instructions are introduced with the 80186. The new instructions are ENTER/LEAVE which is used to replace several instructions when handling stack frames, PUSHA/POPA which is used for push/pop all general registers, BOUND which can check array index against bounds, INS/OUTS which is used for input/output of string. A very useful immediate addressing mode instruction is also added for the push, imul, and multi-bit shift instructions. The detailed operations of all new instructions are described below:

Data-Transfer Instructions

- ✓ **PUSHA** This instruction is used to push the content of all registers onto the stack. Actually, it copies the content of AX, CX, DX, BX, SP, BP, SI, and DI onto the stack. The contents of the SP register are pushed before AX is pushed.
- ✓ **POPA** This instruction can be used to pop the content of all registers from the stack. In fact, it pops DI, SI, BP, SP, BX, DX, CX, and AX from the stack. Usually, the popped SP value is discarded.
- ✓ **PUSH immediate** This instruction can be used to push an immediate 16-bit number onto the stack. The example of PUSH immediate instruction is PUSH 4000H; Store 4000HH on stack immediately and decrement the SP by 2.

Arithmetic Instructions

- ✓ **IMUL** This instruction is used for signed multiplication immediately. The common format of IMUL instruction is

IMUL Destination Register, Source, Immediate data.

This instruction has three operands. The first operand is the destination register. The second operand is the source of data which may be either the memory or the register. The third operand is the immediate data.

When the IMUL instruction is executed, the multiplication between the contents of the source operand and immediate data is performed, and the lower 16 bits of the final result will be stored in the destination register. But the upper 16 bits of the final result is ignored. The example of IMUL instruction is

IMUL BX, CX, 22; Multiply the content of CX register with 22H and result (lower 16-bits) is stored in BX register

Logical Instructions

SHIFT/ROTATE Destination, immediate data, Immediate data states number of bit shift.

String Instructions

✓ **INS** This instruction is used to input a string byte or string word. The common format of INS instruction is
INS Destination, Port.

Actually, it copies a byte using INSB or a word using INSW from I/O port to the destination memory. The port address is represented by DX and the contents of the port move to the memory location pointed by DI. After completion of data transfer, DI must be updated.

✓ **OUTS** This instruction is used to output a string byte or string word. The common format of OUTS instruction is

OUTS Port, Source

Usually, this instruction copies a byte using OUTSB or word using OUTSW from a specified memory location pointed by SI to the specified I/O port. The port address is represented by the content of the DX register. After completion of data transfer, SI must be updated.

✓ **BOUND** This instruction is used to detect values outside the predefined range. The format of BOUND instruction is

BOUND REG, SRC.

(REG) ← (SRC) and an interrupt of type 5 occurs. BOUND REG, SCR must point to 2 consecutive words in the memory.

✓ **ENTER** This instruction is used to prepare the stack for procedure entry. The format of ENTER instruction is

ENTER SIZE, Level. The stepwise operation of this instruction is given below:

Step 1 Push the content of BP. Save SP to a register TEMP.

Step 2 When Level = 0, go to Step 8.

Step 3 If LEVEL – 1 = 0, go to Step 7.

Step 4 (BP) ← [(BP) – 2]

Step 5 Push to the stack pointed by BP.

Step 6 Level ← Level-1 and go to Step 3.

Step 7 Push TEMP.

Step 8 (BP) ← (TEMP.)

Step 9 (SP) ← (SP) - SIZE .

(SP) ← (BP) and Pop stack into BP

In this instruction, SIZE is a non-negative integer less than 65, 536 and Level is a non-negative integer less than 256.

11.7 COMPARISON BETWEEN 8086 AND 80186

The comparison between 8086 and 80186 is given Table 11.10.

Table 11.10 Comparison between 8086 and 80186

<i>8086 microprocessor</i>	<i>80186 microprocessor</i>
Intel 8086 was developed in 1978 and it has about 3000 different instructions for programming.	Intel 80186 was developed in 1982 and it is the improved version of 8086, in the sense that it has faster instruction execution time, includes a few more instructions as compared to 8086.
DMA channels, programmable interrupt controller, programmable timers, programmable chip select logic are not incorporated in the 8086 processor.	High-speed DMA channels, programmable interrupt controller, programmable timers, programmable chip select logic are incorporated in the 8086 processor.
8086 has an operating frequency of 5 to 10 MHz.	80186 has an operating frequency of 8 MHz to 10 MHz.
The operation codes 63H and 64H are not available in 8086.	80186 accepts the operation code 63H and 64H and these codes will present an interrupt of byte-6.
8086 has arithmetic and logic instructions and numbers of instructions are more than 3000.	80186 is compatible with all existing instructions of 8086 and it has ten new instructions.
Slow performance with respect to 80186. Power consumption is more than 80186.	80186 is high-performance processor, with two times the performance of the standard 8086. The power consumption is less than 8086.
8086 is a 40-pin IC and available in plastic leaded chip carrier (PLCC), ceramic leads chip carrier (LCC) and pin grid array (PGA) packages.	80186 is a 68-pin IC and available in plastic leaded chip carrier (PLCC), ceramic leads chip carrier (LCC) and pin grid array (PGA) packages.
PUSH and POP instructions are available in 8086.	PUSH, POP and PUSH immediate instructions are available in 80186.
IMUL instruction is available for signed multiplication and its format is IMUL BL	IMUL instruction is available for signed multiplication and its format is IMUL BX, CX, 22.
8086 has no high-level instructions.	80186 has three high-level instructions such as BOUND, LEAVE and ENTER.
For 8086 to execute a rotate or shift instruction, it will need more than 1000 cycles to complete, and, therefore, there is a long delay if an interrupt of highest priority is requested.	In order to execute a rotate or shift instruction, the number of bits to shift is the count specified in the instruction modulo 32. This limits the number of shifts to 31. It requires less execution time compared to 8086.
In 8086, the LOCK signal can be initiated by a lock instruction prefix and is maintained until the end of the next instruction.	The LOCK signal in 80186 will not be activated until the locked instruction starts its operand reference bus cycles.

11.8 INTRODUCTION TO 80286

The Intel 80286 was introduced in early 1982. This is also known as iAPX 286 and it is an x86 16-bit microprocessor with 134,000 transistors. It was the first Intel processor that could run all the software written for its predecessor. It was widely used in IBM PC compatible computers such as IBM PC/AT during the mid 1984 to early 1990s.

Initially, 80286 was released with 6 MHz and 8 MHz, it was subsequently scaled up to 12.5 MHz. The 80286 had an average speed of about 0.21 instructions per clock. The 6 MHz model usually operated at 0.9 MIPS, the 10 MHz model at 1.5 MIPS, and the 12 MHz model at 1.8 MIPS.

The 80286's performance is more than twice that of its predecessors, i.e., Intel 8086 and Intel 8088 per clock cycle. The 80286 processors have a 24-bit address bus. Therefore, it is able to address up to 16 MB of RAM, whereas the 8086 could directly access up to 1 MB. The 80286 CPU was designed to run multitasking applications, digital communications, real-time process control systems, and multi-user systems.

This processor is the first x86 processor, which can be used to operate in protected mode. The protected mode enabled up to 16 MB of memory to be addressed by the on-chip linear memory management unit (MMU) with 1 GB logical address space. The memory management unit is able to provide some degree of protection from applications writing outside their allocated memory zones. But the 80286 could not revert to the 8086 compatible real mode without resetting the processor.

80286 is a high-performance 16-bit microprocessor with on-chip memory management and protection capabilities. Actually, this processor has been designed for a multi-user as well as a multitasking system. Usually, the 80286 processor is booted in real mode, and thereafter it works in protected mode by software command. But it is not possible to switch the 80286 from protected mode to real mode. To shift from protected mode to real mode, 80286 microprocessors must be reset. The 80286 with 8 MHz clock provides up to 6 times higher than the 5 MHz 8086.

There is no on-chip clock generator circuit in 80286. Therefore, an external 82284 chip is required to generate the external clock. The 80286 has a single CLK pin for single-phase clock input. Usually, the external clock is divided by 2 internally to generate the internal clock. The 82284 provides the 80286 RESET and READY signals.

The 80286 operates in two different modes such as real mode and protected mode. The real mode is used for compatibility with existing 8086/8088 software base, and the protected mode is used for enhanced system level features such as memory management, multitasking, and protection.

The 80286 is the first advanced microprocessor with memory management and protection abilities. In this chapter, the architecture, memory management and other functional details of 80286 are discussed.

11.9 ARCHITECTURE OF 80286

The 80286 processor is an advanced, high-performance microprocessor with specially optimized capabilities for multi-user and multitasking systems. The 80286 has built-in memory protection that supports operating system and task isolation as well as program and data privacy. A 12 MHz 80286 provides about six times more than the 5 MHz 8086. The 80286 includes memory management capabilities that map 2^{30} (one gigabyte) of virtual address space per task into 2^{24} bytes (16 megabytes) of physical memory.

The 80286 is compatible with 8086 and 8088 operating software. The 80286 has two operating modes: real address mode and protected virtual address mode. In real address mode, the 80286 is object code compatible with existing 8086, and 8088 software. In protected virtual address mode, the 80286 is source code compatible with 8086, 8088 software and sometimes it may require upgrading to use virtual addresses supported by the 80286's integrated memory management and protection scheme. Both modes operate at full 80286 performances and execute all instructions of the 8086 and 8088 processors.

The internal block diagram of the 80286 processor's architecture is depicted in Fig. 11.15. The CPU of the 80286 processor consists of four functional units such as

- ◆ Address Unit (AU)

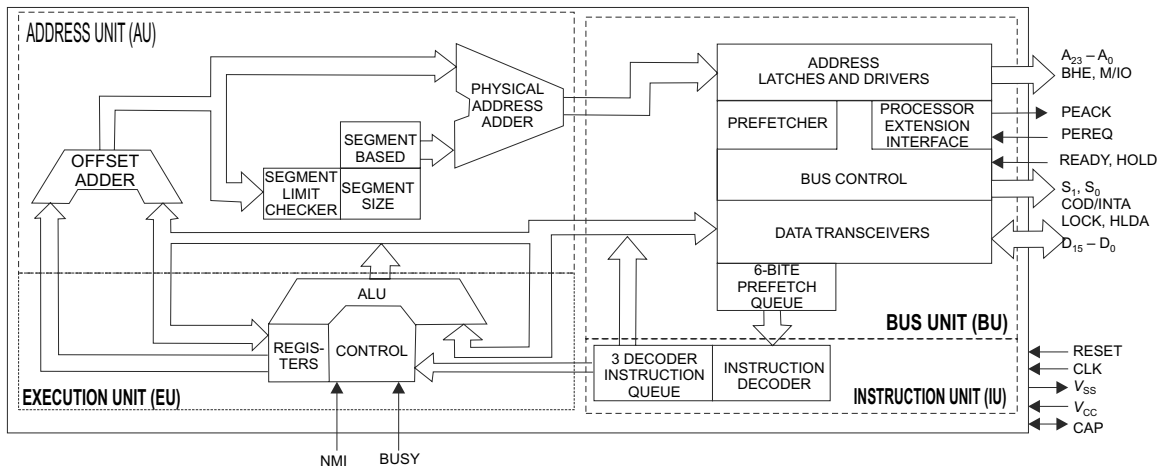


Fig. 11.15 Internal block diagram of the 80286 processor

- ◆ Bus Unit (BU)
- ◆ Instruction Unit (IU)
- ◆ Execution Unit (EU)

Address Unit (AU) The address unit (AU) is used to determine the physical addresses of instructions and operands which are stored in memory. The AU computes the 20-bit physical address based on the contents of the segment register and 16-bit offset just like 8086. The address lines derived by AU can be used to address different peripheral devices such as memory and I/O devices. This physical address computed by the address unit is sent to the Bus Unit (BU) of the CPU.

Bus Unit (BU) The bus unit interfaces the 80286 with memory and I/O devices. This processor has a 16-bit data bus, a 24-bit address bus, and a control bus. The bus unit is responsible for performing all external bus operations. This unit consists of latches and drivers for the address bus, which transmit the physical address $A_{19}-A_0$. The $A_{19}-A_0$ facilitates all memory and I/O devices for read and write operations.

The bus unit is used to fetch instruction bytes from the memory. Generally, the instructions are fetched in advance and stored in a queue for faster execution of the instructions. This concept is known as *instruction pipelining*.

Hence, to fetch instruction, the CPU will not wait till the completion of execution of the previous instruction. While one instruction is being executed, the subsequent instruction is to be prefetched, decoded and kept ready for execution. The prefetcher module in the bus unit performs this task of prefetching. The bus unit has a bus control module which controls the prefetcher module. The fetched instructions are arranged in a 6-byte prefetch queue. In this way, the CPU prefetches the instructions, to enhance the speed of execution.

Instruction Unit (IU) The 6-byte prefetch queue forwards the instructions sequentially to the Instruction Unit (IU). The instruction unit receives instructions from the prefetch queue and an instruction decoder decodes them one by one. The decoded instructions are latched onto a decoded instruction queue. The IU decodes maximum 3 prefetched instructions and loads them into decoded instruction queue for execution by execution unit.

Execution Unit (EU)

The output of the decoded instruction queue is fed to a control circuit of the execution unit. This unit is responsible for executing the instructions received from the decoded instruction queue. The execution unit consists of the register bank, arithmetic and logic unit (ALU) and control block. The register bank is used for storing the data as a scratch pad. The register bank can also be used as special-purpose registers. The ALU is the core of the EU, and perform all the arithmetic and logical operations and sends the results either over the data bus or back to the register bank. The control block controls the overall operation of the execution unit.

The 80286 CPU family contains all the basic set of registers, instructions, and addressing modes of 8086. The 80286 processor is upward compatible with the 8086, 8088, and 80186 CPU's. In this section, operations of registers are explained elaborately.

The 80286 base architecture has fifteen registers as depicted in Fig. 11.16. These registers can be grouped into the four categories as given below:

- ◆ General-purpose registers
- ◆ Segment registers
- ◆ Base and index registers
- ◆ Status and control registers

General-Purpose Registers

Eight 16-bit general-purpose registers are used to store arithmetic and logical operands. Four of these (AX, BX, CX, and DX) can be used either as 16-bit words or split into pairs of separate 8-bit registers.

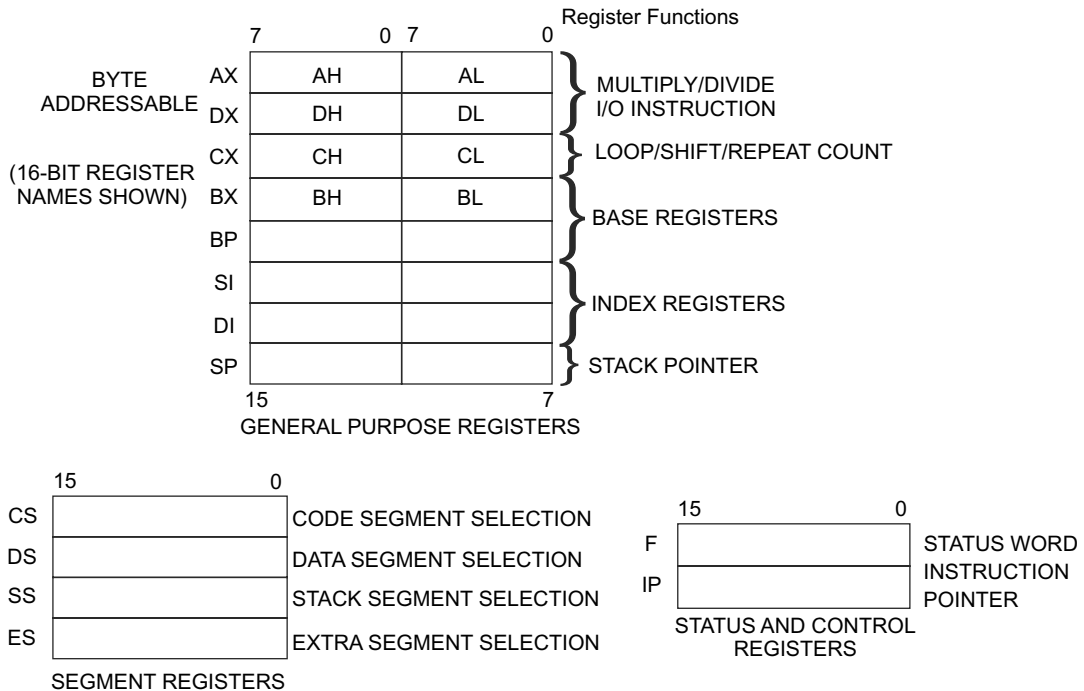


Fig. 11.16 Register set of 80286

Segment Registers

Four 16-bit special-purpose registers are used to select the segments of memory that are immediately addressable for code, stack, and data.

Base and Index Registers

Four of the general-purpose registers can also be used to determine offset addresses of operands in memory. Usually, these registers hold base addresses or indexes to particular locations within a segment. Any specified addressing mode determines the specific registers used for operand address calculations.

Status and Control Registers

The three 16-bit special-purpose registers are used of record and control of the 80286 processor. The instruction pointer contains the offset address of the next sequential instruction to be executed.

Flags Word Description

The flags word register records the specific characteristics of the result of arithmetic and logical instructions. The flag register bits D_0 , D_2 , D_4 , D_6 , D_7 , and D_{11} are modified as per result of the execution of arithmetic and logical instructions. These are called *status flag bits*. Bits D_8 and D_9 control the operation of the 80286 within a given operating mode and these bits are called *control flags*. The flag register is a 16-bit register. Figure 11.17 shows the flag register of 80286 and the function of the flag bits are explained below:

- ✓ **CF Carry Flag (bit D_0)** Set on high-order bit carry or borrow; cleared otherwise.
- ✓ **PF Parity Flag (bit D_2)** Set if low-order 8 bits of result contain an even number of 1bit; cleared otherwise.

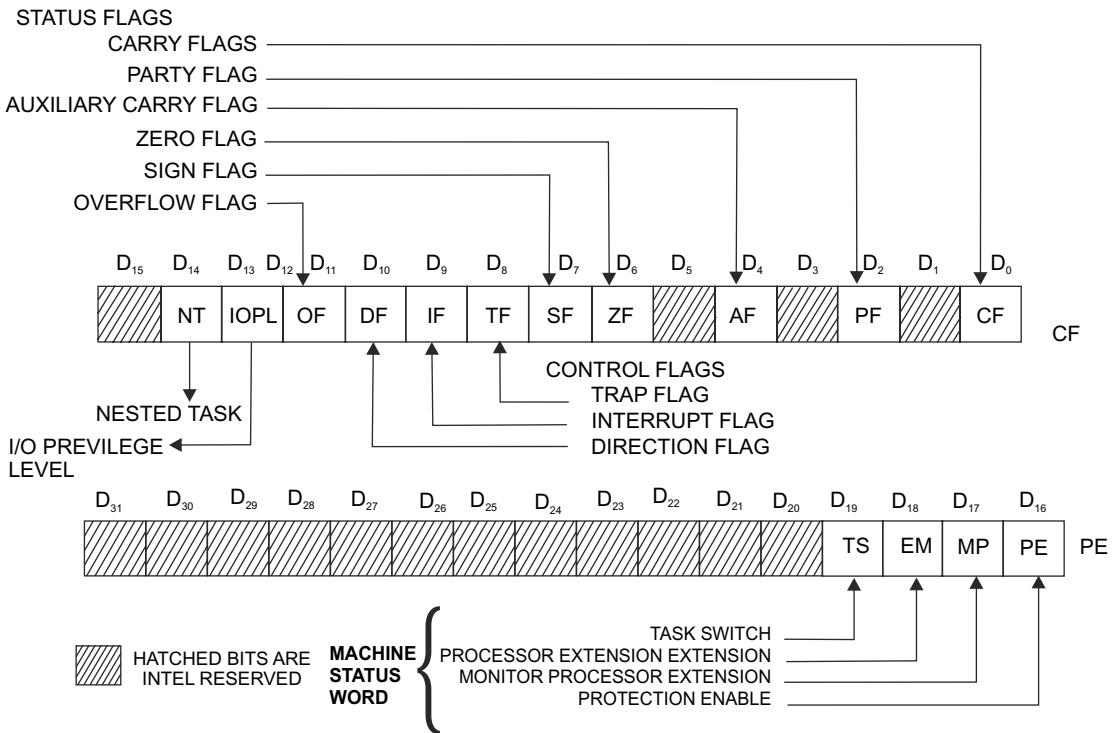


Fig. 11.17 80286 flag registers

- ✓ **AF (bit D_4)** Set on carry from or borrow to the lower-order four bits of AL; cleared otherwise.
- ✓ **ZF Zero Flag (bit D_6)** Set if result is zero; cleared otherwise.
- ✓ **SF Sign Flag (bit D_7)** Set equal to high-order bit of result (0 if positive, 1 if negative).
- ✓ **TF Single Step Flag (bit D_8)** Once set, a single-step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
- ✓ **IF Interrupt-enable Flag (bit D_9)** When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
- ✓ **DF Direction Flag (bit D_{10})** Causes string instructions to auto-decrement the appropriate index registers when set. Clearing DF causes auto increment.
- ✓ **OF Overflow Flag (bit D_{11})** Set if result is a too-large positive number or a too-small negative number (excluding sign-bit) to fit in destination operand; cleared otherwise.
- ✓ **PE (bit D_{16})** Protection enable flag places the 80286 in protected mode, when PE is set. This can only be cleared by resetting the CPU.
- ✓ **MP (bit D_{17})** When MP is set, the monitor processor extension flag allows WAIT instruction to generate a processor extension not present in the exception, i.e., exception number 7.
- ✓ **EM (bit D_{18})** If EM is set, the emulate processor extension flag causes a processor extension absent exception and permits the emulation of processor extension by CPU.
- ✓ **TS (bit D_{19})** When TS set, this flag indicates the next instruction using extension will generate exception 7, permitting the CPU to test whether the current processor extension is for the current task.

Machine Status Word (MSW)

The machine status word consists of four flags such as PE, MP, EM and TS of the four lower-order bits D_{16} to D_{19} of the upper word of the flag register. The LMSW and SMSW instructions are available in the instruction set of 80286 and these instructions are used to read and write the MSW in real address mode.

11.10 PIN DIAGRAM OF 80286

The 80286 processor is available in 68-pin PLCC (Plastic Leaded Chip Carrier), 68-pin Ceramic LCC (Lead Less Chip Carrier), and 68-pin PGA (Pin Grid Array) packages. In PLCC, conducting leads are provided for external connections but in LCC, only conducting pads are provided in place of each pin. The pin diagram of 80286 for PLCC packages is depicted in Fig. 11.18. The pin functions of 80286 are briefly discussed below:

- ✓ **CLK (I)** The CLK is the system clock input pin. The system clock frequency applied at this pin is divided by two inside the 80286 to generate the processor clock. The internal divide-by-two circuits must be synchronized with the external clock generator. The clock is generated using 82284 clock generator.
- ✓ **D_{15} – D_0 (I/O)** These are sixteen bidirectional data bus lines. The data bus inputs data during memory, I/O and interrupt acknowledge read cycles. The data bus outputs data during memory and I/O write cycles. The data bus is active high and floats to 3-state off during bus hold acknowledge.
- ✓ **A_{23} – A_0 (O)** These are the physical address output lines used to address memory or I/O port address or I/O devices. A_{23} – A_{16} are low during I/O transfers. A_0 is low when data is to be transferred on pins D_7 – D_0 . The address bus is active high and floats to tri-state off during bus hold acknowledge.

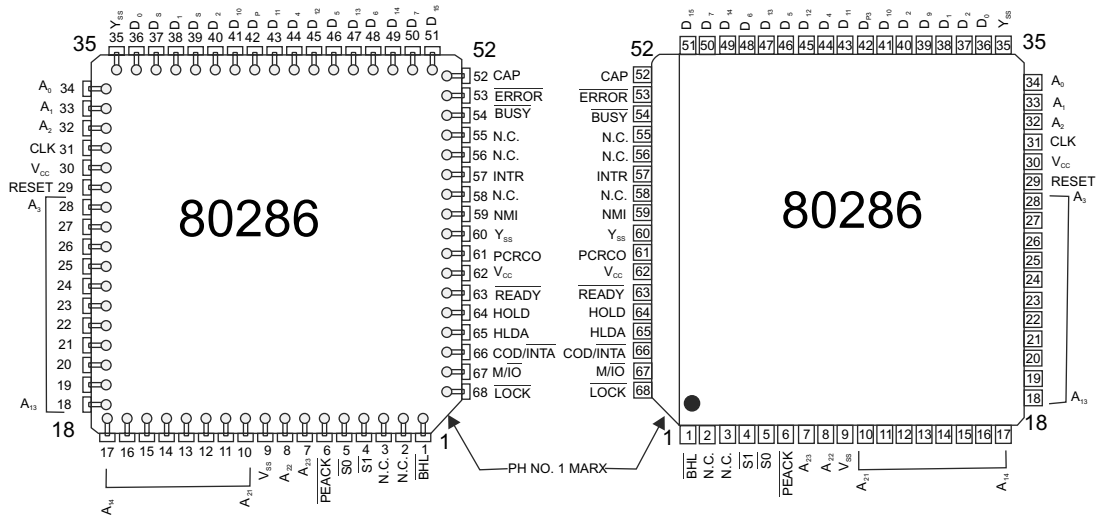


Fig. 11.18 Pin diagram of 80286 in PLCC

✓ **\overline{BHE} (O)** The bus high enable is an output signal and indicates that there is a transfer on the higher byte of the data bus (D_{15} – D_8). Eight-bit oriented devices assigned to the upper byte of the data bus normally use \overline{BHE} , which is active low and floats to 3-state off during bus hold acknowledge.

Table 11.11 \overline{BHE} and A_0 encodings

\overline{BHE}	A_0	Function
0	0	Word transfer
0	1	Byte transfer on upper half of data bus D_{15} – D_8
1	0	Byte transfer on lower half of data bus D_7 – D_0
1	1	Will never occur

✓ **$\overline{S_0}, \overline{S_1}$ (O)** These are bus cycle status signals. These signals are the active-low status output signals which indicate initiation of a bus cycle. Along with $M/\overline{I/O}$ and COD/\overline{INTA} , these signals define the type of the bus cycle as shown in Table 11.12.

✓ **$M/\overline{I/O}$ (O)** The memory I/O select output signal differentiates memory operations from I/O operations. If this signal is "1", a memory cycle or a halt/shutdown cycle is in progress. If it is "0", an I/O cycle or interrupt acknowledge cycle is in progress.

✓ **COD/\overline{INTA} (O)** The code/interrupt acknowledges output signal distinguishes instruction fetch cycle from memory data read cycles. This signal also differentiates interrupt acknowledge cycles from I/O cycles.

Table 11.12 80286 bus cycle status

COD/\overline{INTA}	$M/\overline{I/O}$	$\overline{S_1}$	$\overline{S_0}$	Bus Cycle
0	0	0	0	Interrupt acknowledge
0	0	0	1	Will not occur
0	0	1	0	Will not occur

(Contd.)

(Contd.)

0	0	1	1	None; not a status cycle
0	1	0	0	IF AI = 1 then halt; else shutdown
0	1	0	1	Memory data read
0	1	1	0	Memory data write
0	1	1	1	None; not a status cycle
1	0	0	0	Will not occur
1	0	0	1	I/O read
1	0	1	0	I/O write
1	0	1	1	None; not a status cycle
1	1	0	0	Will not occur
1	1	0	1	Memory instruction read
1	1	1	0	Will not occur
1	1	1	1	None; not a status cycle

✓ **\overline{LOCK} (O)** This active-low output signal is used to indicate that the other system masters are not to gain control of the system bus for the current and the following bus cycles. This signal is activated by \overline{LOCK} instruction prefix, or automatically by hardware during XCHG, interrupt acknowledge or descriptor table access. \overline{LOCK} is active low and floats to 3-state off during bus hold acknowledge.

✓ **\overline{READY} (I)** This is an active-low input signal. It is used to terminate a bus cycle. Bus cycles are extended without limit until terminated by \overline{READY} low. This signal is neglected during HLDA cycle.

✓ **HOLD and HLDA** The hold request (HOLD) and hold acknowledge (HLDA) control ownership of the local bus of 80286.

The HOLD input signal allows another local bus master to request control of the local bus. While the control is granted, the 80286 will float its bus drivers to 3-state off and then activate the hold acknowledge (HLDA) signal.

The local bus must be remain granted to the requesting master until HOLD becomes inactive so that the 80286 deactivates HLDA and regains control of the local bus. This can terminate the hold acknowledge condition. These signals are active high.

✓ **INTR (I)** The interrupt request (INTR) signal requests the 80286 to suspend the execution of current instruction and provide service to a pending interrupt request. Its function is just like that of INTR pin of 8086.

Whenever the interrupt enable bit in the flag word is cleared, interrupt requests are masked. When the 80286 responds to an interrupt request, it performs two interrupt acknowledge bus cycles to read an 8-bit interrupt vector. INTR is active high and may be asynchronous to the system clock.

✓ **NMI (I)** The non-maskable interrupt (NMI) request is an active-high, edge-triggered input signal. This signal interrupts the 80286 with an internally supplied vector value of 2. No acknowledge cycles are needed to be carried out. For proper recognition of NMI, this input signal must be previous low for at least four system clock cycles and remain high for at least four system clock cycles.

✓ **PEREQ (I) and \overline{PEACK} (O)** The processor extension request and acknowledgement (PEREQ) and acknowledge (\overline{PEACK}) extend the memory management and protection capabilities of the 80286 to coprocessor (80287 in case of 80286 CPU).

The PEREQ input requests the 80286 to perform a data-operand transfer for a processor extension. The \overline{PEACK} is an active-low output signal that indicates to the processor extension that the requested operand is being transferred.

- ✓ **\overline{BUSY} (I) and \overline{ERROR} (I)** The processor extension busy (\overline{BUSY}) and error (\overline{ERROR}) are active-low input signals that indicate the operating conditions of a processor extension to 80286. When the \overline{BUSY} becomes low, 80286 stops the program execution and waits until the \overline{BUSY} becomes inactive. During this time, the processor extension is busy with its allotted job. After completion of the job, the processor extension drives the \overline{BUSY} input high indicating 80286 to continue with the program execution. An active \overline{ERROR} input signal causes the 80286 to perform the processor extension interrupt while executing the WAIT or ESC instructions.
- ✓ **CAP (I)** A substrate filter capacitor, 0.047 μf , 12 V must be connected between this input pin and ground to filter the output of the internal substrate bias generator. For correct operation of 80286 the capacitor must be charged to its operating voltage. Till this capacitor charges to its full capacity, the 80286 may be kept stuck to reset to avoid any spurious activity. A maximum dc leakage current of 1 μA is allowed through the capacitor.
- ✓ **V_{SS} (I)** This pin is ground at 0 volts.
- ✓ **V_{CC} (I)** This pin is used to apply + 5 V power supply voltage of 80286.
- ✓ **RESET (I)** The system RESET input clears the internal logic of 80286, and it is active high. Due to a low to high transition on RESET, the 80286 may be reinitialized. For proper reinitialize, the active-high reset input pulse width should be at least 16 clock cycles.

11.11 ADDRESSING MODES OF 80286

The 80286 has eight addressing modes for instructions to access operands from memory. The eight different addressing modes are as follows:

- ◆ Register operand mode
- ◆ Immediate operand
- ◆ Direct mode
- ◆ Register indirect mode
- ◆ Based mode
- ◆ Indexed mode
- ◆ Based indexed mode
- ◆ Based indexed mode with displacement

The first two operating modes are related with the register and immediate operands. The remaining six modes are provided to specify the location of an operand in a memory segment. A memory operand address consists of two 16-bit components, namely, segment selector and offset. The segment selector is supplied by a segment register either implicitly chosen by a segment override prefixes. The offset is determined by summing any combination of the following three address elements.

- ◆ The displacement (8- or 16-bit immediate value)
- ◆ The base (content of the BX or BP)
- ◆ Any carry out from the 16-bit addition is ignored; eight-bit displacements are sign extended to 16-bit values

Combinations of these three address elements define the six memory addressing modes. All above modes are explained in this section.

- ✓ **Register Operand Mode** In this mode, the operand is located in one of the 8- or 16-bit general-purpose registers.
- ✓ **Immediate Operand Mode** In immediate operand mode, the operand is included in the instruction itself.
- ✓ **Direct Mode** In direct addressing mode, the operand's offset is containing in the instruction as an 8- or 16-bit immediate displacement.
- ✓ **Register Indirect Mode** In register indirect addressing mode, the operand's offset is stored in one of the general-purpose registers or in SI, DI, BX or BP.
- ✓ **Based Mode** In this mode, the operand's offset is computed after adding an 8- or 16-bit displacement with the contents of a base register (BX or BP).
- ✓ **Indexed Mode** In index addressing mode, the offset is determined by adding a displacement with the contents of an index register (SI or DI).
- ✓ **Based Indexed Mode** In this mode, the operand's offset is calculated by the sum of the contents of a base register and an index register.
- ✓ **Based Indexed Mode With Displacement** In based indexed with displacement addressing mode, the operand's offset is obtained by adding an 8-bit or 16-bit immediate displacement with contents of a base register and an index register.

11.12 DATA TYPES OF 80286

The 80286 directly supports seven types of data such as integer, ordinal, pointer, string, ASCII, BCD, packed BCD, and floating-point types of data as given below:

- ✓ **Integer** It is a signed binary numeric value contained in an 8-bit byte or 16-bit word. All operations are performed assuming 2's complement representation of the operand. The signed 32 and 64-bit integers are supported using the 80287 numeric data processor.
- ✓ **Ordinal (unsigned)** Ordinal is an 8-bit or 16-bit unsigned binary numeric value.
- ✓ **Pointer** Pointer is a 32-bit quantity, composed of a segment selector component and an offset component; each component is 16-bit word.
- ✓ **String** String is a contiguous sequence of bytes or words. A string may contain from 1 byte to 64 k bytes or 32k words.
- ✓ **ASCII** A byte representation of alphanumeric and control characters using ASCII standard of character representation.
- ✓ **BCD** BCD represents decimal digits 0–9. All operations are performed based on decimal digits 0–9.
- ✓ **Packed BCD** A byte representation of two decimal digits 0–9 storing one digit in each nibble of the byte.
- ✓ **Floating Point** A signed 32, 64 or 80 real-number representation.

11.13 80286 INSTRUCTION SET

The instruction sets of the 80286 processor are upwardly compatible with that of the 8086 processor. Most of the instructions of 80286 are the same as the instructions of 8086. The additional instructions of 80286 processor are as follows:

- ◆ ARPL (Adjust RPL Field of Segment Selector)
- ◆ CLTS (Clear Task Switched Flag in CRO)
- ◆ LAR (Load Access Rights Byte)
- ◆ LGDT/LIDT (Load Global/Interrupt Descriptor Table Register)
- ◆ LLDT (Load Local Descriptor Table Register)
- ◆ LMSW (Load Machine Status Word)
- ◆ LOADALL (Load All Registers)
- ◆ LSL (Load Segment Limit)
- ◆ LTR (Load Task Register)
- ◆ SGDT (Store Global Descriptor Table Register)
- ◆ SIDT (Store Interrupt Descriptor Table Register)
- ◆ SLDT (Store Local Descriptor Table Register)
- ◆ SMSW (Store Machine Status Word)
- ◆ STR (Store Task Register)
- ◆ VERR/VERW (Verify a Segment for Reading or Writing)

In this section, all new instructions are explained elaborately with examples.

ARPL (Adjust Requested Privilege Level of the Selector) The ARPL instruction enables the lower privileged routines to access higher privileged routines or data. The common format of ARPL instruction is ARPL destination, source

This instruction compares the RPL bits of 'destination' against 'source'. If the RPL bits of 'destination' are less than 'source', the destination RPL bits are set equal to the source RPL bits and the zero flag is set. Otherwise, the zero flag is cleared. The example of ARPL instruction is

ARPL register, register

CLTS (Clear Task Switch Flag) The operation of CLTS instruction is to clear the task switched flag of the status flag word. This instruction is a privileged instruction to be executed at the level 0 by the operating system software.

LAR (Load Access Rights Byte) When this instruction is executed, the access rights byte of the descriptor associated with the source (Operand 2) as a selector is loaded into the higher byte of the destination register (Operand 1) and the lower byte of the operand 1 is set to 00. The zero flag is set if the load operation is successful. The format of LAR instruction is

LAR destination, source

The example of LAR instruction is LAR AX, 4000H.

✓ **LGDT (Load Global Descriptor Table)** This instruction loads a value from an operand into the Global Descriptor Table (GDT) register. No flags are affected.

The format of LGDT instruction is

LGDT source

LIDT (Load Interrupt Descriptor Table) This instruction loads a value from an operand into the Interrupt Descriptor Table (IDT) register. No flags are affected. The format of LIDT instruction is

LIDT source

LLDT (Load Local Descriptor Table) The LLDT instruction loads a value from an operand into the Local Descriptor Table Register (LDTR). This instruction is used by operating systems. No flags are affected. The format of LLDT instruction is LLDT source. The example of LLDT instruction is

LLDT BP

LMSW (Load Machine Status Word) This instruction loads the MSW from the effective address of the operand. The example LMSW instruction is

LMSW BP; Load MSW from address DS:BP

LSL (Load Segment Limit) This instruction loads the segment limit of a selector into the destination register if the selector is valid and visible at the current privilege level. The zero flag is set, if loading is successful. Otherwise, zero flag is cleared. The format of LSL instruction is

LSL destination, source

The example of LSL instruction is LSL AX, Selector or LSL register_16, register_16.

LTR (Load Task Register) This instruction loads the current task register with the value specified in 'source'. No flags are affected. The format of LTR instruction is

LTR source

The example of LTR instruction is LTR [5000H]

SGDT (Store Global Descriptor Table) This instruction stores the Global Descriptor Table (GDT) register into the specified operand. No flags are affected. The format of SGDT instruction is

SGDT destination

SIDT (Store Interrupt Descriptor Table) This instruction stores the Interrupt Descriptor Table (IDT) register into the specified operand. No flags are affected. The format of SIDT instruction is

SIDT destination

SMSW (Store Machine Status Word) This instruction stores the MSW to the effective address of the operand. A general protection error exception is generated, if the operand points to a write protected segment or is if invalid memory reference. The stack-fault exception is generated for stack-segment limit overrun. The example of SMSW instruction is

SMSW BP; Store MSW to address ES:BP

STR (Store Task Register) This instruction stores the current task register to the specified operand. No flags are affected. The format of STR instruction is

STR destination

The example of STR instruction is STR [5000H]

A general protection exception is generated when there is an attempt of a write operation in a write protected segment or an invalid memory reference. A stack-fault exception is generated for usual reasons.

VERR (Verify for Read) The VERR instruction sets the zero flag if the segment pointed to by the selector (a 16-bit register or a memory operand) can be read. The example of VERR instruction is VERR BP.

VERW (Verify for Write) The VERW instruction sets the zero flag if the segment pointed to by the selector (a 16-bit register or a memory operand) can be written. The example of VERW instruction is VERW MEMORY.

ENTER (Enter Procedure)

This instruction modifies the stack for entry to procedure for high-level language. This instruction which is used by most of the structured high-level languages requires two operands. The format of Enter instruction is

ENTER locals, level

Operand 'locals' specifies the amount of storage to be allocated on the stack. 'Level' specifies the nesting level of the routine. No flags are affected by this instruction. Paired with the LEAVE instruction, this is an efficient method of entry and exit to procedures. The example of Enter instruction is

ENTER immed_16, immed_8

LEAVE (Leave the Procedure)

The operation of LEAVE is exactly the opposite operation of ENTER instruction. This instruction is used with high-level languages to exit a procedure. When RET instruction is executed after LEAVE, it returns the control to the calling program.

BOUND (Array Index Bound Check)

When this instruction is executed, the array index in the source register is checked against upper and lower bounds in memory source. The format of BOUND instruction is

BOUND source, limit

The first word located at 'limit' is the lower boundary and the word at 'limit+2' is the upper array bound. When the source value is less than or higher than the source, Interrupt 5 occurs. None of the flags are affected. The example of BOUND instructions are BOUND register_16, memory_32, and BOUND BX, memory_32

Memory_32 is a memory block starting address containing four bytes, two bytes for the starting index and the other two for ending index.

LAHF (Load Register AH From Flags)

This instruction copies bits 0–7 of the flags register into AH. This includes flags AF, CF, PF, SF and ZF other bits are undefined.

AH: = SF ZF xx AF xx PF xx CF

PUSH IMD (Push Immediate)

When this instruction is executed, a 16-bit immediate data is pushed to the stack after decrementing the stack pointer (SP) by 2. If the new value of SP is outside the stack segment, a stack fault exception is generated. As the new segment reference is illegal, usually a general protection exception is generated for a push operation. Flags are not affected after execution of PUSH IMD instruction.

PUSH A (Push All)

After execution of PUSH A instruction, the contents of AX, CX, DX, BX, SP, BP, SI, and DI are pushed onto the stack. Hence the stack pointer (SP) is decremented by 16. As the structure of the stack is Last In First Out (LIFO), the last pushed register contents appear first, in the stack memory segment. During execution of PUSH A, if the stack segment limit is overrun, a stack fault exception is generated. None of the flags are affected.

POP A (Pop All)

When this instruction is executed, the contents of the registers DI, SI, BP, SP, BX, DX, CX and AX are popped from the stack in the sequence that is exactly opposite to that of pushing. No flags are affected. Exceptions are exactly the same as PUSH A instruction.

IMUL lmd-Oper

When this instruction is executed, the content of AL is multiplied with a signed immediate operand and the signed 16-bit result is stored in AX. The flags CF and OF are cleared, while the

AH is a sign extension of AL, else CF and OF are set. When the immediate operand is a signed 16-bit data, the contents of AX is multiplied with signed 16-bit data and the signed result is stored in DX: AX combination, with DX as MSB and AX as LSB. Other flags are undefined.

Rotate Source, Count

This is a group of four instructions such as RCL, RCR, ROL, and ROR. These instructions work as in 8086, but an additional mode of count is allowed. In 8086, the count value is either 1 or CL, but in 80286 the value can be an immediate count value 0 to 31 (decimal). After execution of rotate instruction, only the OF and CF flags are changed.

INS (INSB, INSW)

When INS instruction is executed, a string of byte data or word from a variable port address specified only in DX will be read. The port address may be of 16 bits. No flags are affected by this instruction. The data string read by this instruction must be stored in memory at the address pointed by ES:DI, in the sequence in which they were read. After execution of INS instruction, the DI is automatically advanced depending upon the direction flag DF. The example of INS instruction is INSB ES:DI, DX.

OUTS (OUTSB/OUTSW)

When OUTS instruction is executed, a string of bytes or words from the memory location specified by DS:SI will be write to a port pointed by DX. The SI is automatically incremented by 1 for byte or incremented by 2 for word operations. The example of OUTS instruction is OUTS DX, DS:SI.

11.14 80286 ADDRESSING MODE

The 80286 microprocessor operates in two different modes such as Real Address Mode and Protected Virtual Address Mode (PVAM). In this section, the real address mode as well as virtual address mode are discussed.

11.14.1 Real Addressing Mode

In the real addressing mode of operation, 80286 executes a fully upward-compatible instruction of the 8086 instruction set. It works as a fast 8086. In real address mode, the 80286's object code is compatible with 8086 and 8088 software. In this addressing mode, the 80286 addresses a contiguous array of up to 1 M (1,048,576 bytes) of physical memory using $A_{19}-A_0$ and \overline{BHE} . The lines $A_{23}-A_{20}$ should be ignored.

During addressing the physical memory of up to 1MB, the 80286 uses \overline{BHE} along with $A_{19}-A_0$ in the real address mode. The address bits $A_{23}-A_{20}$ will not be always zero in this mode. Hence $A_{23}-A_{20}$ should not be used by the processor when the 80286 operates in real address mode. The 20-bit physical address is generated in the same way as that in 8086. Figure 11.19 shows the address calculation in real address mode.

The segment selector portion of the pointer is used as the upper 16 bits of a 20-bit segment address. The lower four bits of the 20-bit segment address are always zero. Therefore, segment addresses begin on a multiple of 16 bytes. In this addressing mode, all segments are 64 kB in size and may be read, written or executed. An exception or interrupt can occurs if data operands or instructions attempt to wrap around the end of a segment.

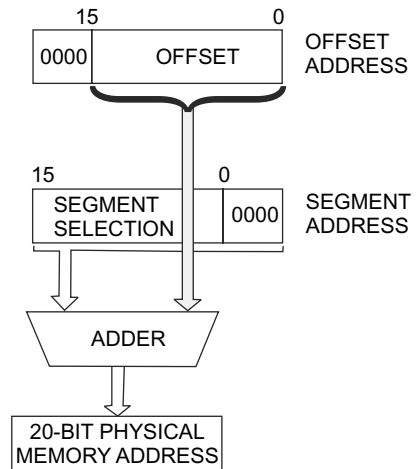


Fig. 11.19 Address calculation in 80286 Real Address Mode

In real address mode, the 80286 reserves two fixed areas of memory, namely, system initialization area and interrupt table area. Memory locations from addresses FFFF0(H) to FFFFF(H) are reserved for system initialization. The memory location from 00000(H) through 0003FF(H) are reserved for interrupt vectors as depicted in Fig. 11.20.

11.14.2 Protected Virtual Address Mode (PVAM)

The 80286 executes a fully upward-compatible superset of the 8086 instruction set in protected virtual address mode. The PVAM operation of the 80286 processor provides memory management and protection mechanisms and associated instructions.

The 80286 enters into protected virtual address mode from real address mode when the PE (Protection Enable) bit of the machines status word is set with the execution of LMSW (Load Machine Status Word) instruction. This operating mode also provides extended physical and virtual memory address space, memory protection mechanisms and new operations to support operating systems and virtual memory.

In this mode, 80286 provides a 1-gigabyte virtual address space per task mapped into a 16-megabyte physical address space defined by the address pin $A_{23}-A_0$ and \overline{BHE} . The virtual memory address space must be larger than the physical address space since any use of an address that does not map to a physical memory location will cause a restartable exception.

The protected mode uses a 32-bit pointer which consists of 16-bit selector and 16-bit offset components. The selector specifies an index into a memory resident table rather than the upper 16 bits of a real memory address. The 24-bit base address of the desired segment can be obtained from the tables in memory. The 16-bit offset will be added to the segment base address to form the physical address. Figure 11.21 shows the

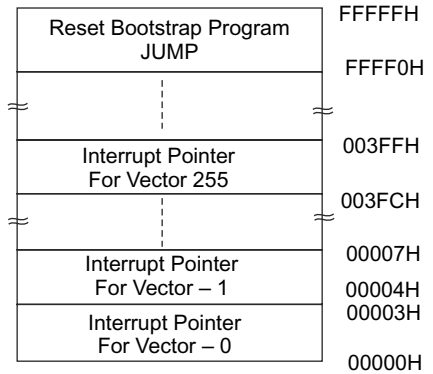


Fig. 11.20 Reserved memory locations in 80286 Real Address Mode

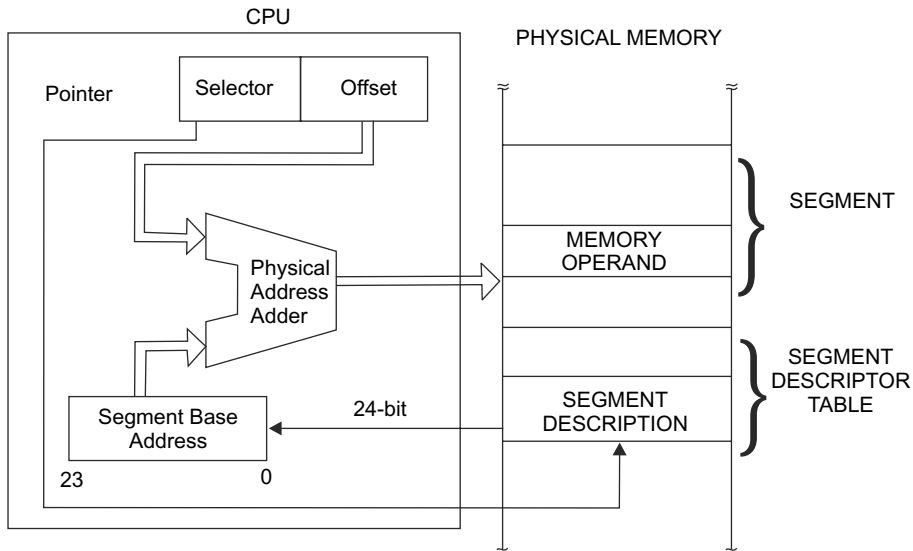


Fig. 11.21 Physical Address Calculation in PVAM

memory addressing in PVAM. The segment descriptor tables are referenced by the CPU whenever a segment register is loaded with a selector. The memory-based tables hold 8-byte values called *descriptors*.

Descriptors

Descriptors state how to use the memory by the 80286 processor. Some special types of descriptors are used to define new functions for *transfer of control* and *task switching*. The 80286 has *segment descriptors* for code, stack and data segments. This processor also has *system control descriptors* for special system data segments and control transfer operations. In this section, code and data segment descriptors, system segment descriptors, gate descriptors, segment descriptor cache registers are explained.

Code and Data Segment Descriptors (S = 1)

The code and data segment descriptors contain segment base addresses, segment attributes including segment size up to 64 KB; access rights such as read, read/write, execute, and execute/read; and presence in memory for virtual memory system. Figure 11.22 shows the code and data segment descriptors and access-rights byte definitions are illustrated in Table 11.13.

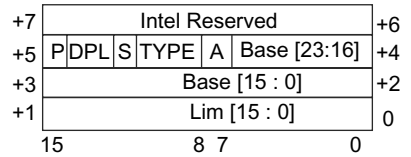


Fig. 11.22 Code and data segment descriptor

Table 11.13 Access rights byte definition

Bit Position	Name		Function	
7	Present (P)	P = 1	Segment is mapped into physical memory	
		P = 0	No mapping to physical memory exists, base and limit are not used.	
6-5	Descriptor Privilege Level (DPL)		Segment privilege attribute used in privilege tests	
4	Segment Descriptor(S)	S = 1	Code or data segment descriptor	
		S = 0	System segment descriptor or gate descriptor	
3	Executable (E)	E = 0	Data segment descriptor type is	
2	Expansion Direction	ED = 0	Expand up segment, offsets must be ≤ limit	If S = 1, E = 0, Data Segment
		ED = 1	Expand down segment, offsets must be > limit	
1	Writable (W)	W = 0	Data segment may not be written into	
		W = 1	Data segment may be written into	
3	Executable (E)	E = 1	Code segment descriptor type is	If S = 1, E = 1, Code Segment
2	Conforming (C)	C = 1	Code segment may only be executed when CPL ≥ DPL and CPL remains unchanged.	
1	Readable (R)	R = 0	Code segment may not be read	
		R = 1	Code segment may be read.	
0	Accessed (A)	A = 0	Segment has not been accessed	
		A = 1	Segment selector has been loaded into segment register or used by selector test instructions.	

System Segment Descriptors (S = 0, TYPE = 1–3)

The protected mode 80286 states the system segment descriptors which contain a table of descriptors (*Local descriptor table descriptor*) and segments which holds the execution state of a task (*task state segment descriptor*). Figure 11.23 shows the

system segment descriptor and system segment descriptor fields are given in Table 11.14. This descriptor consists of 24-bit base address of the segment and 16-bit limit. The access byte defines the type of descriptor, its states and privilege level. Bit 4 of the access byte is always 0 to indicate the system control descriptor. The functions of P, DPL and Type (1–3) are given in Table 11.14.

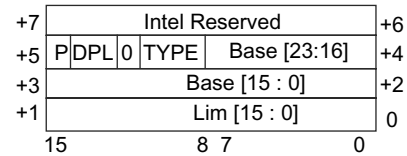


Fig. 11.23 System segment descriptor

Table 11.14 System segment descriptor fields

Name	Value	Description
P	0	Descriptor contents are not valid
	1	Descriptor contents are valid
DPL	0-3	Descriptor privilege level
TYPE	1	Available Task State Segment (TSS)
	2	Local descriptor table
	3	Busy Task State Segment (TSS)
BASE	24-bit number	Base address of special system data segment in real memory
LIMIT	16-bit number	Offset of last byte in segment

Gate Descriptors ($S = 0$, Type 4–7)

The gate descriptors are used to control the access to entry points within the target code segment. There are four types of gate descriptors such as *call gates*, *task gates*, *interrupt gates* and *trap gates*. The gate descriptors provide information regarding indirection between the source and the destination of control transfer. *Call gates* are used to change privilege levels. *Task gates* are used to perform a task switch. The *interrupt and trap gates* are used to specify interrupt service routines. Figure 11.24 shows the gate descriptor and the gate descriptor fields are depicted in Table 11.15. The gate descriptor consists of a 16-bit destination selector, 16-bit destination offset and access byte format. The operation of access byte format is given in Table 11.15.

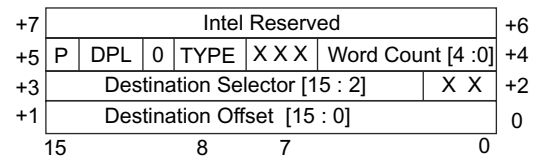


Fig. 11.24 Gate descriptor

Table 11.15 Gate descriptor fields

Name	Value	Description
P	0	Descriptor contents are not valid
	1	Descriptor contents are valid
DPL	0-3	Descriptor privilege level
TYPE	4	Call gate
	5	Task gate
	6	Interrupt gate
	7	Trap gate
Word Count	0-31	Number of words to copy from callers stack to called procedures stack; only used with call gate
Destination Selector	16-bit selector	Selector to the target code segment (Call, Interrupt or Trap Gate) selector to the target task state segment (Task Gate)
Destination Offset	16-bit offset	Entry point within the target code segment

Segment Descriptor Cache Registers

Figure 11.25 shows the segment descriptor cache register which is assigned to each of the four segments registers, i.e., CS, DS, SS, and ES. The segment descriptors are automatically loaded into a segment descriptor cache register, whenever the associated segment register is loaded with a selector. Once segment descriptors are loaded into segment descriptor cache registers, the segment of memory uses the cached descriptor information instead of accessing the descriptor. The segment descriptor cache registers are invisible to programs.

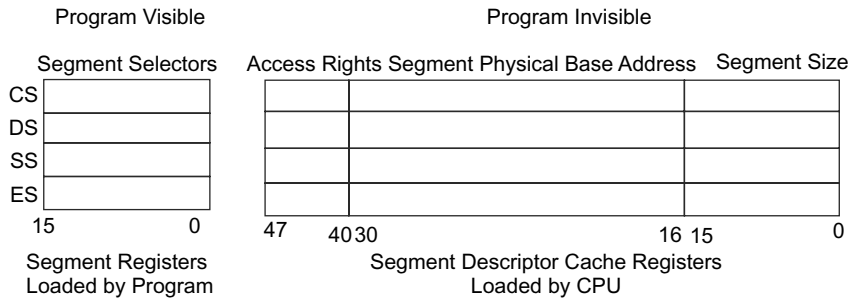


Fig. 11.25 Descriptor cache registers

Selector Fields

In the protected mode of 80286, the selector has three fields such as *descriptor entry index*, *local or global descriptor table indicator (TI)*, and *selector privilege (RPL)* as shown in Fig. 11.26. The first two bits D_{15} – D_0 are called requested privilege level RPL field. The D_2 bit states the descriptor table type. The index D_{15} – D_3 bits are used to indicate descriptor base in the descriptor table. The function of the fields is given in Table 11.16.

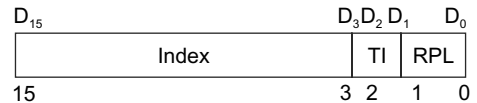


Fig. 11.26 Sector fields

Table 11.16 Function of sector fields

Bits	Name	Function
0-1	Request privilege level (RPL)	Indicates selector privilege level desired
2	Table indicator (TI)	TI = 0, use global descriptor table TI = 1, use local descriptor table
15-3	Index	Select descriptor entry in table

Local And Global Descriptor Tables

The local and global descriptor tables contain all descriptors accessible by a task at any given time. Actually, a descriptor table is a linear array of up to 8K (8192) descriptors. The upper 13 bits of the selector field are an index into a descriptor table. Each descriptor table has a 24-bit base register to locate the descriptor table in physical memory and it has a 16-bit limit register which confines descriptor access to the defined limits of the table. Figure 11.27 shows the local and global descriptor tables.

The Global Descriptor Table (GDT) contains descriptors available to all tasks. The Local Descriptor Table (LDT) contains descriptors that can be private to a task. All tasks may have their private LDTs. The GDT may contain all descriptor types except interrupt and trap descriptors. The LDT contains segment, task gate, and call gate descriptors. A segment cannot be accessed by a task if its segment descriptor does not exist in either GDT or LDT at the time of access.

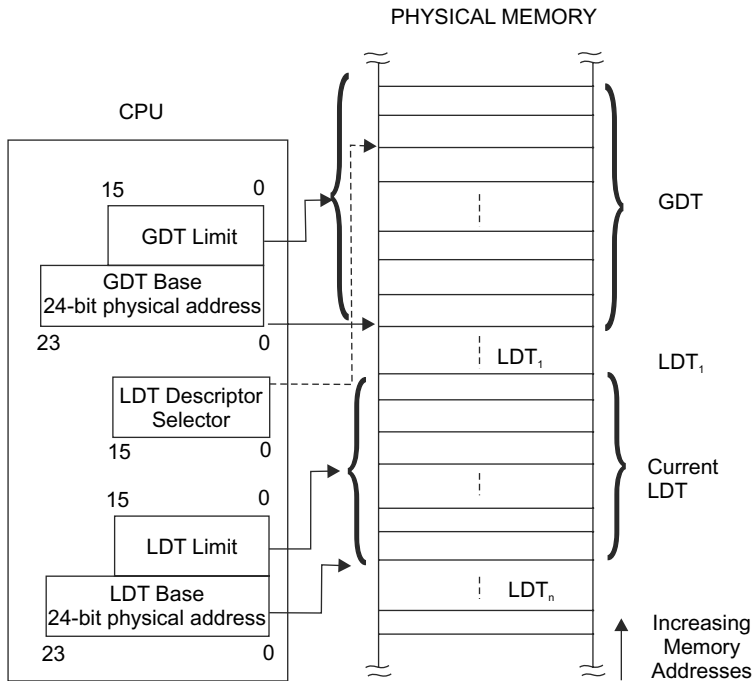


Fig. 11.27 Local and global descriptor table definition

The LGDT (Load Global Descriptor Table) and LLDT (Load Local Descriptor Table) instructions load the base and limit of the GDT and LDT. The LGDT and LLDT are privileged, and these instructions may only be executed by programs at privilege level 0. The LGDT instruction loads a six-byte field containing the 16-bit limit and 24-bit physical base address of the GDT as depicted in Fig. 11.27. The LLDT instruction loads a selector which refers to an LDT descriptor containing the base addresses and limit as shown in Fig. 11.28.

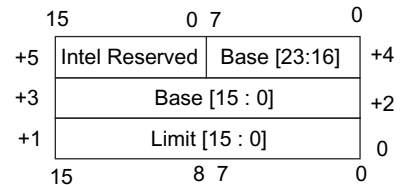


Fig. 11.28 Global descriptor table

Interrupt Description Table

In the protected mode, the 80286 processor has a third descriptor table known as Interrupt Descriptor Table (IDT). The IDT can be used to define up to 256 interrupts. Figure 11.29 shows the Interrupt Descriptor Table. The IDT contains task gates, interrupt gates and trap gates. The IDT has a 24-bit physical base and a 16-bit limit register in the CPU. The privileged LIDT (Load Interrupt Descriptor Table) instruction loads these registers with a 6-byte value in same way of the LGDT instruction. Usually, the IDT entries are made through INT instructions, external interrupt vectors, or exceptions. The IDT should have 256 bytes in size to allocate space for all reserved interrupts.

Privilege

The 80286 processor can support a four-level hierarchical privilege system which controls the use of privileged instructions and access to descriptors within a task. Figure 11.30 shows four-level privilege mechanism. The privilege levels are numbered 0 through 3. Level 0 is the most privileged level whereas Level 4 is the least privileged level. Privilege levels provide protection within a task. The operating

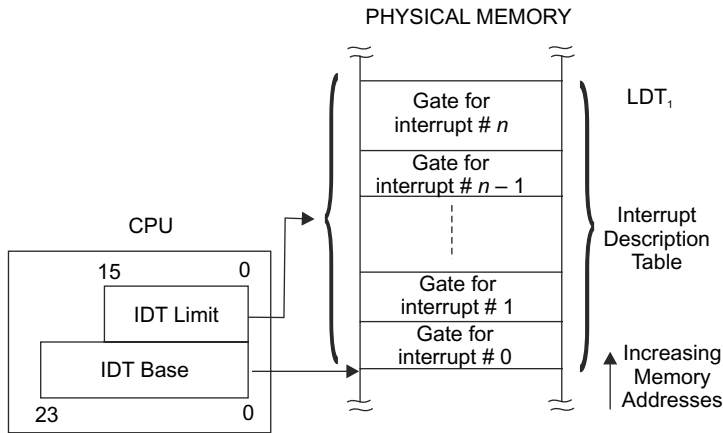


Fig. 11.29 Interrupt descriptor table

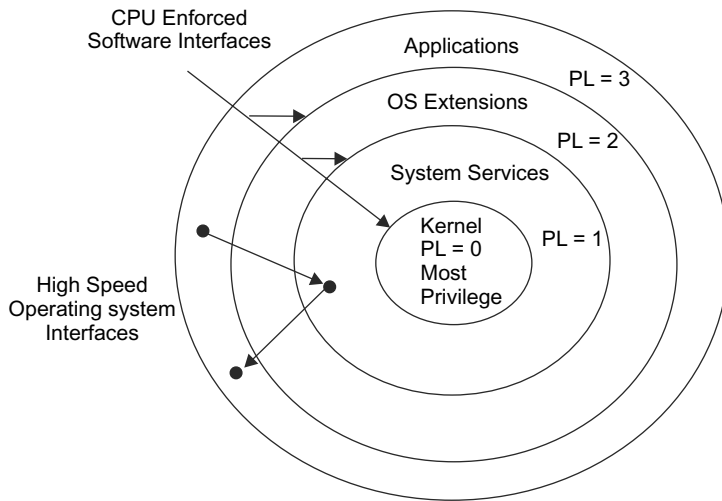


Fig. 11.30 Four-level privilege

system routines interrupt handlers, and other system software can be protected from unauthorized accesses within the virtual address space of each task using the four levels of privilege. Each task in the system has a separate stack for each privilege levels. Tasks, descriptors, and selectors have a privilege level attribute that can find out whether the descriptor may be used. The task privilege has an effect on the use of instructions and descriptors. The descriptor and selector privilege only effect access to the descriptor.

11.15 COMPARISON BETWEEN 8086 AND 80286

The comparison between 8086 and 80286 is given Table 11.17.

Table 11.17 Comparison between 8086 and 80286

<i>8086 microprocessor</i>	<i>80286 microprocessor</i>
Intel 8086 was developed in 1978 and it has about 3000 different instructions for programming	Intel 80286 was developed in 1983 by Intel and it is the improved version of 80186, in the sense that it has faster instruction execution time, includes a few more instructions as compared to 80186
DMA channels, programmable interrupt controller, programmable timers, programmable chip select logic are not incorporated in the 8086 processor	High-speed DMA channels, programmable interrupt controller, programmable timers, programmable chip select logic are incorporated in the 8086 processor
8086 has operating frequency of 5 to 10 MHz	80286 has an operating frequency 8 MHz to 12.5 MHz
8086 has no memory management capability	80286 has memory management capability that maps 2^{30} (1GB) of virtual address
The 8086 has 20-bit address lines and can able to access $2^{20} = 1\text{MB}$ memory.	The 80286 has 24-bit address lines and can able to access $2^{24} = 16\text{MB}$ memory.
8086 operates in real addressing mode	80286 operates in real mode as well as protected virtual address mode
8086 has arithmetic and logic instructions and numbers of instructions are more than 3000	80286 is compatible with all existing instructions of 8086 and it has some new instructions.
Slow performance with respect to 80286. Power consumption is more than 80286	80286 is a high-performance processor. Six times the performance of the standard 8086. The power consumption is less than 8086
8086 is a 40-pin IC and available in plastic leaded chip carrier (PLCC), ceramic leads chip carrier (LCC) and pin grid array (PGA) packages.	80286 is a 68 pin IC and available in plastic leaded chip carrier (PLCC), ceramic leads chip carrier (LCC) and pin grid array (PGA) packages
PUSH and POP instructions are available in 8086	PUSH, POP and PUSH immediate instructions are available in 80286
IMUL instruction is available for signed multiplication and its format is IMUL BL	IMUL instruction is available for signed multiplication and its format is IMUL BX, CX, 22
8086 has no high-level instructions	80286 has three high-level instructions such as BOUND, LEAVE and ENTER
In case of 8086 to execute a rotate or shift instruction, it will need more than 1000 cycles to complete and, therefore, there is a long delay if an interrupt of highest priority is requested	In order to execute a rotate or shift instruction, the number of bits to shift is the count specified in the instruction modulo 32. This limits the number of shifts to 31. It requires less execution time compared to 8086.
In 8086, the LOCK signal can be initiated by a lock instruction prefix and is maintained until the end of the next instruction	The LOCK signal in 80286 will not be activated until the locked instruction starts its operand reference bus cycles

11.16 COMPARISON BETWEEN 80186 AND 80286

The comparison between 80186 and 80286 is given Table 11.18.

Table 11.18 Comparison between 80186 and 80286

80186 microprocessor	80286 microprocessor
Intel 80186 was developed in 1982 and it is the improved version of 8086, in the sense that it has faster instruction execution time, includes a few more instructions as compared to 8086.	Intel 80286 was developed in 1983 by Intel and it is the improved version of 80186, in the sense that it has faster instruction execution time, includes a few more instructions as compared to 80186.
80186 has an operating frequency of 8 MHz to 10 MHz.	80286 has an operating frequency of 8 MHz to 12.5 MHz.
80186 has no memory management capability.	80286 has memory management capability that maps 2^{30} (1GB) of virtual address.
The 80186 has 20-bit address lines and is able to access $2^{20} = 1\text{MB}$ memory.	The 80286 has 24-bit address lines and can able to access $2^{24} = 16\text{ MB}$ memory.
80186 operates in real addressing mode.	80286 operates in real mode as well as protected virtual address mode.
80186 is compatible with all existing instructions of 8086 and it has ten new instructions.	80286 is compatible with all existing instructions of 80186 and it has some new instructions
80186 is two times the performance of the standard 8086. The power consumption is less than 8086.	80286 is six times the performance of the standard 8086. The power consumption is less than 8086.
CAP is not available in the 80186 microprocessor.	80286 has a new pin CAP. An external capacitor is connected to the CAP pin for filtering the bias voltage.

11.17 INTRODUCTION TO 80386

The concepts of memory management, privilege and protection was introduced with 80286. The 16-bit word length of 80286 provides limitations on its operating speed. But for advanced applications, technology demanded high-speed machines with more powerful instruction sets incorporating all the features of 80286. Subsequently, a CPU with a 32-bit word size and higher operating frequency and high speed of operation, has been developed to overcome all the limitations of 80286. The new processor is called the 80386 processor. This is the third-generation processor and is introduced by Intel in 1985. The features of 80386 processor are as follows:

- ◆ The 80386 is a *32-bit microprocessor* that can support 8-bit, 16-bit and 32-bit operands. It has 32-bits registers, 32-bits internal and external data bus, and 32-bit address bus.
- ◆ Due to its *32-bit address bus*, the 80386 can address up to *4GB of physical memory*. The physical memory of this processor is organised in terms of segments of 4 GB size at maximum.
- ◆ The 80386 CPU is able to support 16k number of segments and the total virtual memory space is 4 giga bytes x 16k = 64 terrabytes.
- ◆ *It has a 16-byte prefetch queue.*
- ◆ It is manufactured by Intel using 0.8-micron CHMOS technology.
- ◆ It is available with 275k transistors in a 132-Pin PGA package.
- ◆ It operates at clock speeds of 16 MHz to 33 MHz.
- ◆ This processor has *memory management unit with a segmentation unit and a paging unit.*
- ◆ It operates in real, protected and virtual real mode. The protected mode of 80386 is fully compatible with 80286.

- ◆ The 80386 instruction set is upward compatible with all its predecessors.
- ◆ The 80386 can run 8086 applications under a protected mode in its virtual 8086 mode of operation.
- ◆ The 80386 processor supports Intel 80387 numeric data processor.

In this chapter, the architectural and operational features of 80386 are presented.

11.18 ARCHITECTURE OF 80386

The simplified block diagram of the 80386 processor is depicted in Fig. 11.31, and Fig. 11.32 shows the detailed architecture of 80386. The internal architecture of the 80386 processor consists of three different sections such as Central Processing Unit (CPU), Memory Management Unit (MMU) and Bus Interface Unit (BIU).

Central Processing Unit (CPU)

The central processing unit consists of an Execution Unit (EU) and an Instruction Unit (IU). The Execution Unit (EU) has eight general-purpose registers and eight special-purpose registers. These registers are used for handling data or calculating offset addresses. The Instruction Unit (IU) is used to decode the opcode bytes received from the 16-byte instruction code queue and followed by arranging them into a 3-instruction decoded-instruction queue. After decoding opcode bytes of instructions, information passes to the control section to provide the necessary control signals. The barrel shifter increases the speed of all shifts and rotate operations. The multiply or divide logic implements the bit-shift-rotate algorithms to complete the instruction operations within minimum time. The 32-bit multiplications/divisions can also be executed within one microsecond by the multiply/divide logic.

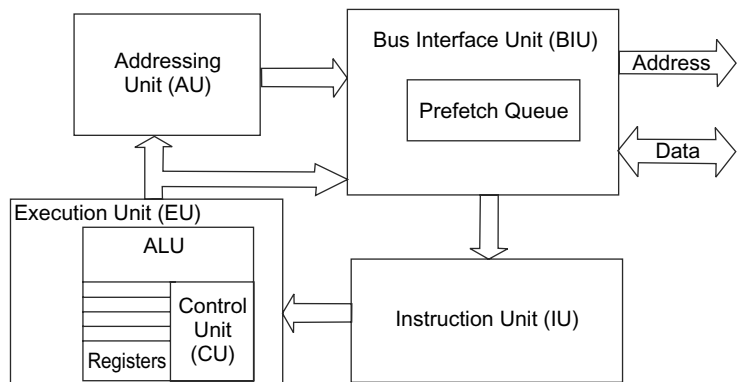


Fig. 11.31 The simplified block diagram of 80386 processor

Memory Management Unit (MMU) The Memory Management Unit (MMU) has a Segmentation Unit (SU) and a Paging Unit (PU).

✓ **Segmentation Unit (SU)** The segmentation unit uses two address components, namely, segment and offset to relocate and sharing of code and data. The segmentation unit allows a maximum size of 4 GB segments. The segmentation unit has four-level protection mechanisms to protect and isolate the system's code and data from the application programs. The 'limit and attribute PLA' is used to check segment limits and attributes at segment level to keep away from invalid accesses to code and data.

✓ **Paging Unit (PU)** The paging unit arranges the physical memory in terms of pages of 4 KB size each. The paging unit always acts under the control of the segmentation unit and each segment is divided into pages. The virtual memory is also arranged in terms of segments and pages by the memory management unit. Usually, the paging unit converts linear addresses into physical addresses. The 'control and attribute PLA' is used to check the privileges at the page level. Each page always maintains the paging information of the task.

✓ **Bus Interface Unit (BIU)** The bus interface unit interfaces the 80386 processor with memory and I/O devices. To fetch instructions and transfer data from code fetcher unit, the processor provides address, data and control signals through BIU. The code prefetch is used for fetching instructions from the memory while BIU is not executing any bus cycle. The bus control section has a 'request prioritizer' to decide the priority of the various bus requests. This section controls the bus access. The address driver is used for bus enable signals BE₃–BE₀ and address signals A₃₁–A₀. The pipeline and bus size control units handle the related control signals.

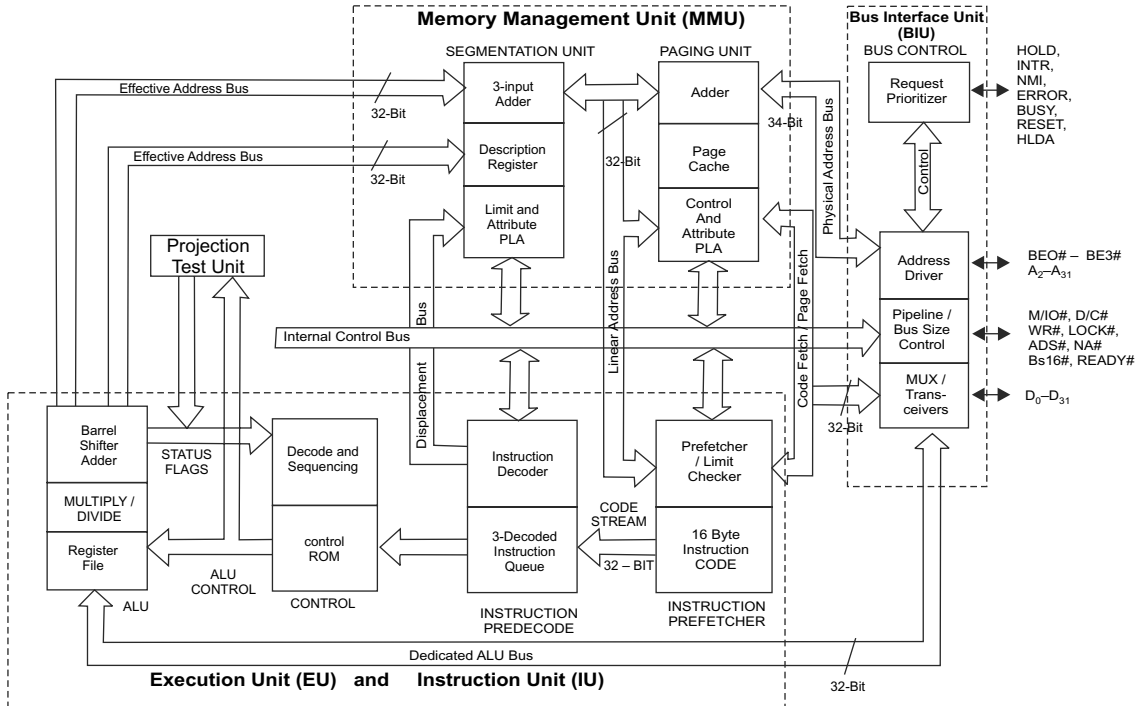


Fig. 11.32 The detailed internal architecture of 80386 processor

11.19 REGISTERS OF 80386

The 80386 processor has significantly extended the 8086 register set. All the registers of 80286 are existing in the 80386 processor and some new registers have been added in 80386. Generally, the registers of 80386 are of 32 bits and they can be used as 8-bit and 16-bit registers. Figure 11.33 and Table 11.19 show the registers of 80386.

The registers of 80386 are divided as *general-purpose registers* and *special registers*.

- *General-purpose registers* are 32-bit EAX, EBX, ECX, EDX, ESI, SDI, EBP, and ESP.
- *Special registers* are
 - Segment (selector) registers 16-bit CS, DS, ES, SS, FS, and GS
 - 32-BIT EIP
 - EFLAGS

32-BIT EIP

The IP of 8086 and 80286 can only support program segments of 64 KB but the EIP register is able to support programs up to 4 GB. The CS register enables larger programs and its content can be changed under program control. The instruction pointer cannot be written directly by the program. The IP can be changed by execution of JUMP, CALL, RETURN and INTERRUPT instructions. For example, during execution of FAR CALL and JUMP instructions, the values of CS as well as IP are changed to locate a new physical memory location.

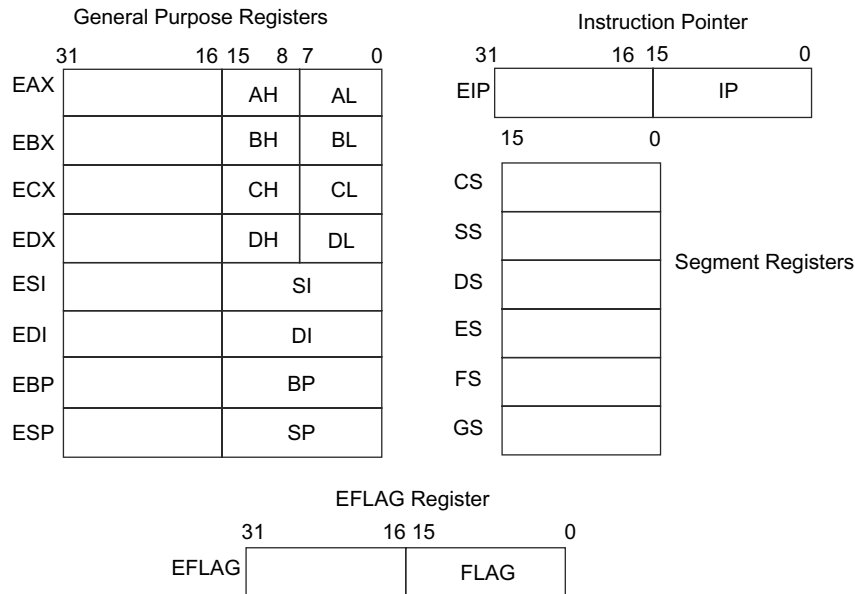


Fig. 11.33 Registers of 80386

Stack Segment and Stack Pointer

Each program has a stack segment. In the 8086 microprocessor, the stack grows downward and the value of the stack pointer decreases with execution of PUSH instruction and increases with execution of POP instruction. In the 80386 processor, when the data is stored on the stack, the value of ESP is decreased by 4 as the 80386 always writes a complete double word or 4 bytes. If the 80386 operates in 16-bit mode, two bytes will be stored on the stack and the value of SP will be reduced by 2 after each PUSH command.

General-Purpose Registers

The general-purpose registers is able to hold 8-, 16-, or 32-bit data. The 8086 microprocessor has byte and word-sized registers, but the 80386 contains double-word sized or extended registers. The 8- and 16-bit registers can be addressed just like the 8086 processor.

The AX, BX, CX, DX, SI, DI, BP, SP, FLAGS and IP registers are 16-bit registers and they have been extended to 32 bits. A 32-bit register is called an extended register and it is represented by the register name with prefix E. For example, a 32-bit register corresponding to AX is represented by EAX. Similarly, all 32-bit general-purpose registers are represented by EAX, EBX, ECX, EDX, ESI, EDI. The other 32-bit registers are EBP, ESP, EFLAGS, and EIP.

Segment Registers

Besides the above 32-bit registers, the 80386 also provides 2 new 16-bit segment registers such as FS and GS. Therefore, all segment registers of 80386 are CS, DS, ES, SS, FS, and GS. The FS and GS registers are additional extra segment registers which allows access 6-different segments

in memory without reloading a segment register. In real-mode operation, segment registers contain a segment address and in protected mode operation, they contain a selector just as in the 8086.

Table 11.19 Register of the 80386 processor

Name	32-bit register	16-bit register	8-bit register	Applications
Accumulator	EAX	AX	AH, AL	Multiplication, division, input/output and shifts
Base Register	EBX	BX	BH, BL	Pointer to base address in data segment
Count Register	ECX	CX	CH, CL	Counting, rotates and shifts
Data Register	EDX	DX	DH, DL	Multiplication, division, input/output

Name	32-bit register	16-bit register	Applications
Base Pointer	EBP	BP	Pointer-to-base address in stack segment
Source Index	ESI	SI	Index pointer and source string
Destination Index	EDI	DI	Index pointer and destination string
Stack pointer	ESP	SP	Stack operation
Instruction pointer	EIP	IP	Instructions offset
Flag	EFLAG	FLAG	Processor status

Control Registers

The 80386 processor has four 32-bit control registers: CR0–CR3. These registers are used to hold global machine status. The load and store instructions are used to access these registers. In 80386, these registers perform paged memory management, cache enable/disable and protected mode operation. Figure 11.34 shows the control registers.

Debugging Registers

The 80386 has eight 32-bit debug registers DR7–DR0 for hardware debugging as depicted in Fig. 11.34. Among the eight debugging registers, two registers DR5 and DR4 are reserved by Intel. The first four registers DR3 to DR0 are used to store four program controllable breakpoint addresses. The DR7 and DR6 hold breakpoint control and breakpoint status information respectively.

Test Registers

Two test registers TR6 and TR7 exist in the 80386 processor for page caching as shown in Fig. 11.34. TR6 is known as test control and TR7 is called a test status register.

System Address Registers

The 80386 has four system address registers to refer the descriptor tables as shown in Fig. 11.34. The four different types of descriptor tables are Global Descriptor Table (GDT), Interrupt Descriptor Table (IDT), Local Descriptor Table (LDT), Task State Segment descriptor (TSS).

The system address and segment registers are used to hold the addresses of descriptor tables GDT, IDT, LDT and TSS and their respective segments. These registers are called GDTR, IDTR, LDTR and TR respectively. The GDTR and IDTR are known as *system address*, but the LDTR and TR are known as *system segment registers*.

FLAG Register

The flag register of the 80386 is a 32-bit register as shown in Fig. 11.35. Among these 32 bits, D31 to D18, D15, D5 and D3 are reserved by Intel and D₁ is always 1. The lower fifteen bits of flag register of 80386 are same as 80286. Only two flags are newly added to the 80286 flag register to get the flag register of 80386. The two new flags are VM and RF flags.

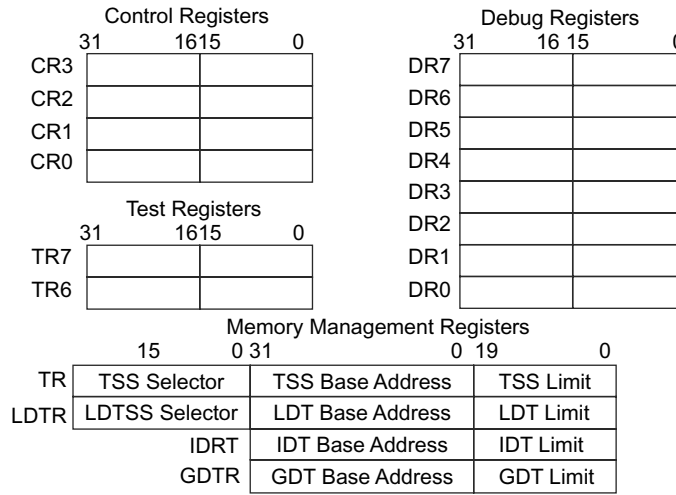


Fig. 11.34 Control, debug and system address (memory management) registers

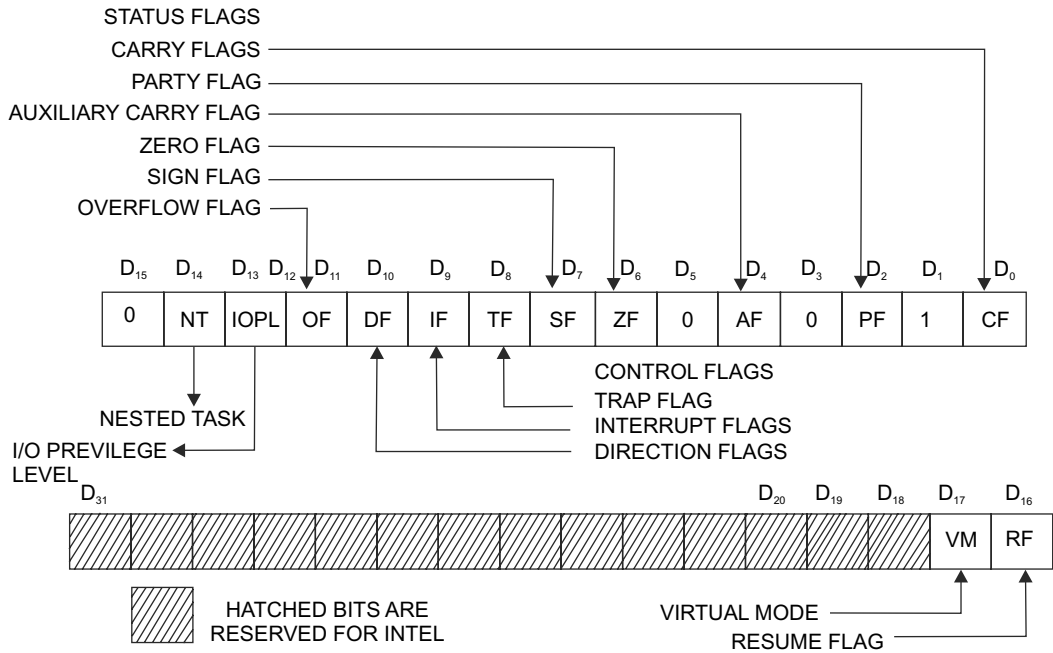


Fig. 11.35 FLAG register of the 80386

RF (Resume Flag)

This is the first bit in the extended EFLAGS register. It is used with the debug register breakpoints. At the starting of each instruction cycle, the status of RF is always checked. If RF = 1, any debug fault will be ignored while executing any instruction. This flag is automatically reset after execution of instructions except IRET and POPF.

VM (Virtual Mode Flag)

When this flag is set, the 80386 enters in the virtual 8086 mode within the protected mode. If VF is set, 80386 operates in protected mode. When this flag is cleared or reset, the 80386 operates in real address mode.

11.20 PIN FUNCTIONS OF 80386

The 80286 family has a sinking current of 2.0 mA but each output pin of the 80386 is capable of sinking 4.0 mA current for address and data buses or 5.0 mA for other connections. Therefore, there is significant improvement in the current handling capability. Figure 11.36 shows the pin diagram of 80386 processor.

- ✓ **A₃₁–A₂ (Address Bus)** Address lines A₃₁–A₂ are used as the upper 30 bits of the 32-bit address bus. These lines can address any memory locations of 1 GB x 32 memory in the 80386 memory system.
- ✓ **A₁–A₀** A₁ and A₀ are encoded to generate BE₃, BE₂, BE₁ and BE₀.
- ✓ **D₃₁–D₀ (Data Bus)** The data bus D₃₁–D₀ are used to transfer data between the microprocessor and memory and input/output devices.

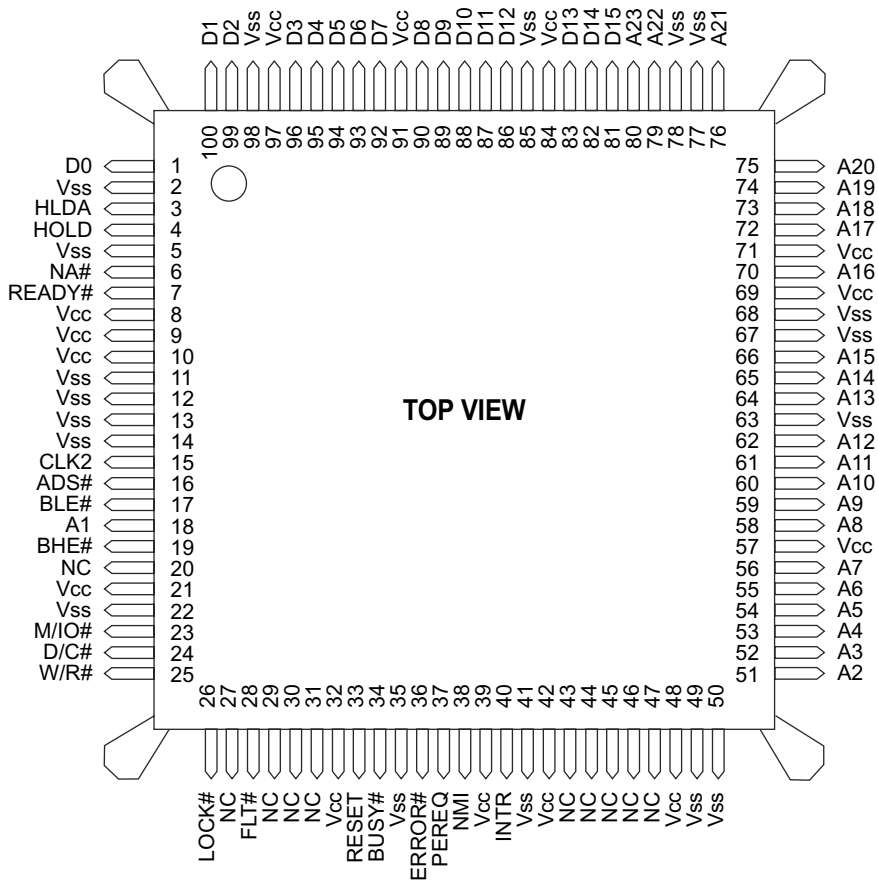


Fig. 11.36 The pin diagram of 80386 processor

- ✓ **BE_3 – BE_0 (Bank Enable Signals)** The memory system of 80386 can be observed as a 4-byte wide memory access mechanism. The four byte enable signals are BE_3 , BE_2 , BE_1 , and BE_0 . These signals are used to enable four banks. These signals are generated by the microprocessor from address bits A_1 and A_0 and used to access 1 byte, 2 bytes or word, and 4bytes or double word of data simultaneously.
- ✓ **M/\overline{IO} (Memory/Input/Output)** The M/\overline{IO} pin is used to select the memory and I/O related operations. When it is logic level '1', memory devices will be selected. If it is logic level '0', I/O devices will be selected for data transfer. During I/O operation, the address bus consists of a 16-bit I/O address.
- ✓ **W/\overline{R} (Write/Read)** The W/\overline{R} signal is used to indicate the read and write bus cycles. When $W/\overline{R} = 1$, the current bus cycle is a write cycle. If $W/\overline{R} = 0$, the current bus cycle is a read cycle.
- ✓ **\overline{ADS} (Address Data Strobe)** This pin indicates that the 80386 issues a valid memory or I/O address and the address bus and bus cycle pins W/\overline{R} , D/\overline{C} , M/\overline{IO} and BE_3 – BE_0 are holding their respective valid signals.
- ✓ **RESET** When RESET pin is high, the processor suspends the current operation and becomes reset. Therefore, the processor restarts the execution from the starting memory location FFFFFFF0H.
- ✓ **CLK2 (Clock time 2)** The CLK2 input pin provides a clock signal for the operation of 80386 and the clock frequency is two times of the operating frequency of the 80386.
- ✓ **\overline{READY}** The \overline{READY} signal indicates that the previous bus cycle has been completed and the bus is ready for the next bus cycle. This signal controls the number of WAIT states inserted into the bus cycle. Usually, this signal is used for interfacing slow I/O devices with CPU.
- ✓ **\overline{LOCK}** When the \overline{LOCK} output pin is at logic level 0, the bus lock prevents the other bus masters from gaining control of the system bus. This signal is most frequently used during DMA accesses.
- ✓ **D/\overline{C} (Data and Control)** The D/\overline{C} output signal indicates whether the current bus cycle is data cycle, i.e., memory and I/O read or write cycles, or the current bus cycle is a control cycle, i.e., interrupt acknowledge, halt or code fetch operations.
- ✓ **BS_{16}** Bus size of 16 bits BS_{16} indicates the interfacing of 16-bit devices with the 32-bit wide 80386 data bus.
- ✓ **\overline{NA}** The \overline{NA} represents the Next Address input signal. When this signal is active low, it allows address pipelining in the 80386 bus cycles.
- ✓ **HOLD** The Hold request signal allows another bus master to request control of the local bus. This signal is commonly used in DMA operation just like it did on the 8086 microprocessor.
- ✓ **HLDA** The Hold Acknowledge (HLDA) indicates that the 80386 processor has surrendered control of its local bus to another bus master.
- ✓ **\overline{PEREQ}** The Processor Extension Request (\overline{PEREQ}) indicates that the 80387 arithmetic coprocessor has data to transfer to the processor.
- ✓ **\overline{BUSY}** The BUSY input signal indicates that the math coprocessor is busy.
- ✓ **ERROR** The Error input signal indicates that the math coprocessor has an error condition.
- ✓ **INTR** The Interrupt Request (INTR) pin is used as a maskable interrupt input. This pin will be masked using the IF of the flag register.
- ✓ **NMI** The Non-maskable Interrupt (NMI) signal requests a nonmaskable interrupt just like the 8086 microprocessor.

- ✓ V_{CC} This pin is connected to a $+V_{CC}$ system power supply.
- ✓ V_{SS} System ground provides the 0 V connection. This pin is connected to a ground terminal of system power supply.

11.21 ADDRESSING MODES OF 80386

The 80386 can operate in all the addressing modes which were available with the 80286 processor. The 80386 processor can also operate in all addressing modes of 80286 with 32-bit immediate or 32-bit register operands or displacements. Besides all addressing modes of 80286, the 80386 has a family of scaled modes. In the scaled modes, the index register values will be multiplied by a valid scale factor to get the final displacement. The valid scale factors are 1, 2, 4 and 8. In this section, all scaled modes are briefly explained.

✓ **Scaled Indexed Mode** The content of an index register is multiplied by a scale factor 1, 2, 4 or 8 and subsequently the computed value will be added to get the final operand offset. For example,

```
MOV ECX, [ESI*2]
MOV ECX, [ESI*4]
MOV ECX, [ESI*8]
```

✓ **Based Scaled Indexed Mode** The based scaled indexed mode instruction is the content of an index register is multiplied by a scale factor and the computed value is added with the base register to find the offset.

```
MOV EAX, [EBX*2] [ECX]
MOV EBX, [EDX*4] [ECX]
MOV EAX, [EBX*8] [ECX]
```

✓ **Based Scaled Indexed Mode with Displacement** The based scaled indexed mode with displacement instruction is the content of an index register multiplied by a scaling factor and the computed value is added with the content of base register and a displacement to obtain the address of an operand. The offset can be expressed as

$$\text{Offset} = \text{Index Register} \times \text{Scale factor} + \text{Displacement}$$

and the 32-bit memory address is computed by

$$[\text{Base Register}] + [\text{Index Register}] \times \text{Scale factor} + [\text{Displacement}]$$

The above expression can be expressed for different registers as given below:

$$\begin{bmatrix} EAX \\ EBX \\ ECX \\ ESP \\ EBP \\ ESI \\ EDI \end{bmatrix} + \begin{bmatrix} EAX \\ EBX \\ ECX \\ ESP \\ EBP \\ ESI \\ EDI \end{bmatrix} \times \begin{bmatrix} 1 \\ 2 \\ 4 \\ 8 \end{bmatrix} + \begin{bmatrix} 0 - \text{bit displacement} \\ 8 - \text{bit displacement} \\ 16 - \text{bit displacement} \end{bmatrix}$$

The example of based scaled indexed mode with displacement instructions are

```
MOV ECX, [ESI*2] [EBX+0111]
MOV ECX, [ESI*4] [EAX+0FFF]
MOV ECX, [ESI*8] [EBX+0400]
```

11.22 DATA TYPES OF 80386

The 80386 processor is able to support the following data types:

- ◆ *Bit*
- ◆ *Bit Field*: A group of 4 bytes (32 bits)
- ◆ *Bit String*: A string of contiguous bits of maximum 4 GB in length
- ◆ *Signed Byte*: Signed byte data
- ◆ *Unsigned Byte*: Unsigned byte data
- ◆ *Integer Word*: Signed 16-bit data
- ◆ *Long Integer*: 32-bit signed data
- ◆ *Unsigned Integer Word*: Unsigned 16-bit data
- ◆ *Unsigned Long Integer*: Unsigned 32-bit data
- ◆ *Signed Quad Word*: Signed 64-bit or four word data
- ◆ *Unsigned Quad Word*: Unsigned 64-bit data
- ◆ *Offset*: 16 or 32-bit displacement used in any of the addressing modes
- ◆ *Pointer*: A pair of 16-bit selector and 16/ 32-bit offset
- ◆ *Character*: ASCII equivalent of any alphanumeric or control characters
- ◆ *Strings*: Sequences of bytes, words or double words; it contains minimum one byte and maximum 4 gigabytes
- ◆ *BCD*: Decimal digits (0–9) represented by unpacked bytes
- ◆ *Packed BCD*: Two packed BCD digits representing 00 to 99

11.23 OPERATING MODE OF 80386

The 80386 processor is able to operate in three different modes as given below:

- ◆ Real addressing mode,
- ◆ Protected mode, and
- ◆ Virtual 386 mode.

In this section, the above three modes are explained elaborately.

11.23.1 Real Addressing Mode

The 80386 always starts from the memory location FFFFFFF0H in the real address mode whenever the processor is reset. In this mode, 80386 works as 8086 processor with 32-bit registers and data types. The addressing modes, memory size, interrupt handling of 80386 are same as the real address mode of 80286. Initially, the 80386 starts with real mode and then prepares for protected mode operation. All the instructions of 80386 are available in this mode except protected address mode instructions. In this mode, the operand size is 16 bits by default. The 32-bit operands and addressing modes can be used with the help of override prefixes. In this mode, the segment size is 64 k.

During real addressing mode, the 80386 can address up to 1 MB of physical memory using address lines A_{19} – A_0 . In this address mode, the paging unit is disabled so that the real addresses are the same as the physical addresses. To compute a physical memory address, the contents of the segment register are shifted left by

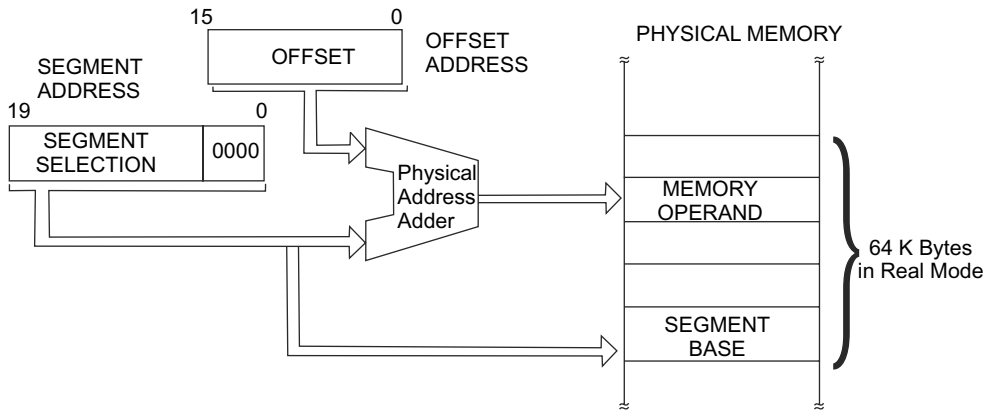


Fig. 11.37 Physical address computation in real mode of 80386

four bit positions and then added to the 16-bit offset address formed using one of the addressing modes just like the 8086 real address mode. Figure 11.37 shows the physical address computation in real mode of 80386. In real-mode operation of 80386, the segments can be read, written or executed. The segments in 80386 real modes can be overlapped or non-overlapped.

11.23.2 Protected Mode Addressing

In protected mode, the 80386 can able to address 4 gigabytes of physical memory and 64 terrabytes of virtual memory. In this mode, the 80386 has capability to support all programs written for 80286 and 8086 and to be executed. The controls of memory management and protection abilities of 80386 are possible in this operating mode. All additional instructions and addressing modes of 80386 feasible in protected mode.

In protected mode addressing, the contents of segment registers are used as selectors which can address the segment descriptors. The segment descriptors consist of the segment limit, base address and access rights byte of the segment. The effective or offset address is added with segment base address to determine linear address. When the paging unit is disabled, the linear address is used as physical address. If the paging unit becomes enabled, the paging unit converts the linear address into physical address. Figure 11.38 and Fig. 11.39 show the protected mode addressing without paging and with paging unit respectively. In general, the

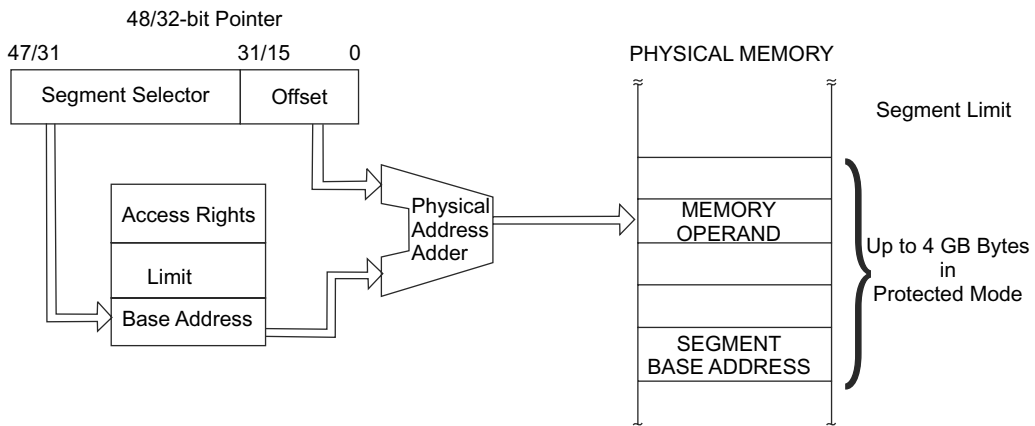


Fig. 11.38 Protected mode addressing of 80386 without paging unit

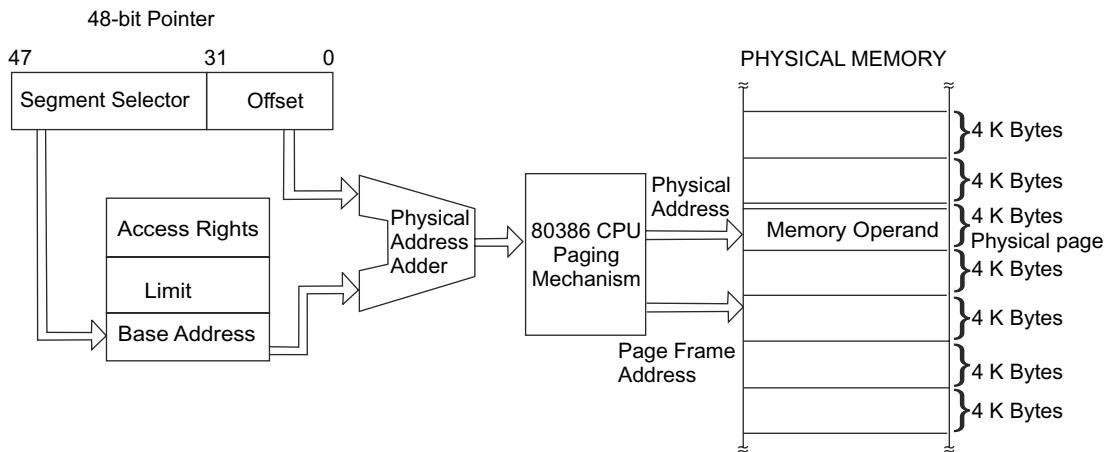


Fig. 11.39 Protected mode addressing of 80386 with paging unit

paging unit is a memory management unit which is enabled only in the protected mode. The paging mechanism is able to handle memory segments in terms of pages of 4 KB size. Usually, a paging unit operates under the control of segmentation unit.

The 80386 starts with real mode and then changes the operation from real mode to the protected mode operation. To change the operation from real mode to the protected mode, the following steps must be followed:

- Step 1** Initialize the IDT so that it contains valid interrupt gates for at least the first 32 interrupt type numbers. Usually, IDT contains up to 256 8-byte interrupt gates to define all 256 interrupt type.
- Step 2** Initialize the GDT so that it contains a null descriptor at Descriptor 0. The valid descriptors are used for at least one code, one stack and one data segment.
- Step 3** Switch to protected mode after setting the PE bit in CR0.
- Step 4** Perform an intrasegment near JMP operation to flush the internal instruction queue and load the TR with the base TSS descriptor.
- Step 5** After that, load all the segment registers with their initial selector values.
- Step 6** 80386 operates in the protected mode using segment descriptors that are defined in GDT and IDT.

11.23.3 Memory Management

The function of the memory-management unit is to convert a linear address into physical address. This unit uses paging mechanism to locate any physical address on memory.

Segmentation

The segmentation provides protection to different types of data and code. The 80386 has three types of segment descriptor tables as already exist in the 80286. But, there are some differences between the 80386 and the 80286 segment descriptor structures. The three types of the 80386 segment descriptor tables are given below:

- ◆ Global Descriptor Table (GDT)
- ◆ Local Descriptor Table (LDT)
- ◆ Interrupt Descriptor Table (IDT)

The registers used for descriptor tables GDT, LDT, and IDT are Global Descriptor Table register (GDTR), Local Descriptor Table Register (LDTR) and Interrupt Descriptor Table register (IDTR) respectively. LGDT, LLDT and LIDT instructions are used to load the three corresponding registers.

Descriptors

The descriptors of 80386 are a series of 8 bytes which is used to describe and locate a memory segment. Figure 11.40 shows the structure of the 80386 descriptor. The 80386 segment descriptors have a 20-bit segment limit and a 32-bit segment address. The descriptors of 80386 contain access right or attribute bits along with the base and limit of the segments. The function of bits of segment descriptors is given in Table 11.20.

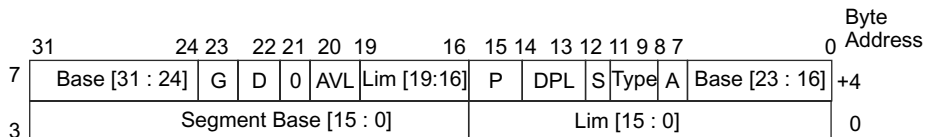


Fig. 11.40 Structure of 80386 segment descriptors

Table 11.20 Bit definition of segment descriptors

Name	Function
BASE ($B_{31}-B_0$)	Base address of the segment
LIMIT ($L_{19}-L_0$)	Length of the segment
P	Present Bit P = 1 for Present, P = 0 for Not Present
DPL	Descriptor privilege level from 0 to 3
S	Segment descriptor S = 0 for System descriptor, S = 1 for Code or Data Segment Descriptor
TYPE	Type of segment
A	Accessed bit
G	Granularity bit If G = 1, Segment length is page granular If S = 0, Segment length is byte granular
D	Default operation size D = 1 for 32-bit segment, D = 0 for 16-bit segment
0	This bit must be '0' for compatibility with other processors
AVL	Available field for user or OS

The 80386 processor has five types of descriptors as follows:

- ◆ Code or data segment descriptors
- ◆ System descriptors
- ◆ Local descriptors
- ◆ TSS (Task State segment) descriptors
- ◆ GATE descriptors

The structures of the above descriptors are slightly different from the general segment descriptor structure of 80286. The 80386 also provides a four-level protection mechanism which is exactly in the same way as the 80286 works.

Page Table The size of each page table is 4 Kbytes and each page table has a maximum 1024 entries. Usually, the page table entries contain the starting address of the page and the information about the page. Figure 11.43 shows the page table entry. Generally, the upper 20 bits of the page frame address of the page table entry is combined with the lower 12 bits of the linear address. The address bits A_{21} – A_{12} of linear address are used to choose the 1024 page table entries.

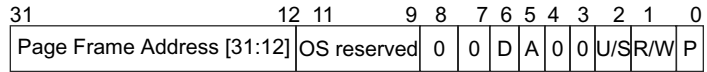


Fig. 11.43 Page table entry

✓ **P-bit** The P-bit can be used in address translation. When $P = 1$, the entry can be used in address translation. If $P = 0$, the entry cannot be used. The P-bit of the presently executed page is always high.

✓ **U/S and R/W Bit** The User/Supervisor (U/S) bit and Read/Write (R/W) bit are used to provide the four level protections as shown in Table 11.21. The level 0 has the highest privilege level, but the level 3 has the lowest privilege level.

Table 11.21 Four-level protections

<i>U/S</i>	<i>R/W</i>	<i>Permitted for privilege level 3</i>	<i>Permitted for privilege level 2, 1, 0</i>
0	0	None	Read Write
0	1	None	Read Write
1	0	Read only	Read Write
1	1	Read Write	Read Write

✓ **A-bit** The A-bit (access bit) must be set by the 80386 processor before accessing any page. If $A = 1$, the page is accessed. When $A=0$, the page is not accessed.

✓ **D-bit** The D-bit (Dirty bit) is set before a write operation to the page. The D-bit is undefined for page directory entries.

The OS reserved bits are defined by the operating system.

11.23.5 Conversion of Linear Address to Physical Address

Initially the paging unit gets a 32-bit linear address from the segmentation unit. Then the upper 20 bits of the linear address (A_{31} – A_{12}) are compared with the 32 entries in the translation look-aside buffer to find if any matches exist with the 32 entries. Whenever it matches, the 32-bit physical address is computed from the matching TLB entry and put on the address bus.

During the linear-addresses-to-physical-addresses conversion, each conversion process uses the two-level paging and a certain amount of time is always wasted in the conversion process. To optimize conversion process, a 32 entry or 32×4 bytes page table cache is used. This page table cache stores the just now accessed 32-page table entries. The page table cache is also called Translation Look-aside Buffer (TLB). When a linear address is converted to a physical address, firstly check the page table cache entries to find the corresponding address. Figure 11.44 shows the paging operation with TLB.

If the page table entry does not exist in TLB, the 80386 processor reads the page directory entry. After that it checks the P-bit of the directory entry. When $P = 1$, the page table exists in the memory. Subsequently, 80386 uses the appropriate page table entry and sets the A bit (access bit). If $P = 1$ in the page table entry, it is confirmed that the page is available in the memory. After that the 80386 processor updates the A and D bits and accesses the page. Then upper 20 bits of the linear address will be read from the page table and will be

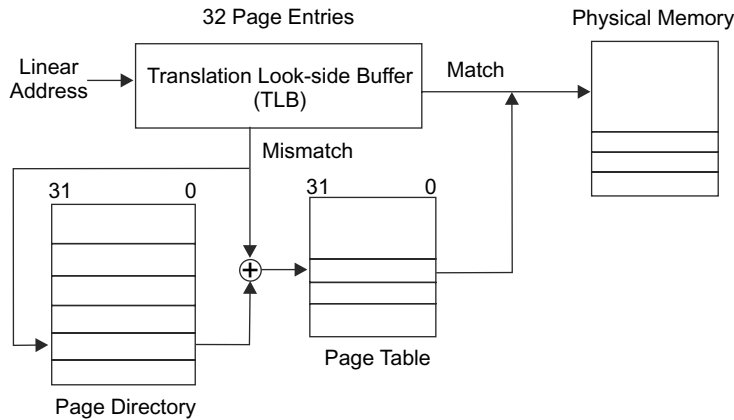


Fig. 11.44 The paging operation with TLB

stored in TLB for future access. If P = 0, the processor generates a page fault exception to indicate that page protection rules are violated.

11.23.6 Virtual 8086 Mode

In real mode, 80386 is able to execute the 8086 programs along with all capabilities of 80386. But once the 80386 processor enters into the protected mode from real mode, it cannot revert back to the real mode without a reset operation. During the protected mode of operation, 80386 processors confer a virtual 8086 operating environment to execute the application programs of 8086. Therefore, the virtual mode operation of 80386 provides an advantage of executing 8086 programs although the 80386 processor is in protected mode.

The address computation mechanism in virtual 80386 modes is same as 8086 real mode. In this mode, 80386 can address 1 Mbytes of physical memory, which will be within the 4 Gbytes memory address of the protected mode of 80386. The paging mechanism and protection capabilities are also available in this mode of operation. In the virtual mode, the paging unit provides 256 pages, each of 4 Kbytes size. Each of the pages will be anywhere within the maximum 4 Gbytes physical memory. The 80386 can support multiprogramming; hence the multiple 8086 real-mode software applications can be executed at a time. Figure 11.45 shows the memory management in virtual 386 modes in a multitasking virtual 8086 environment.

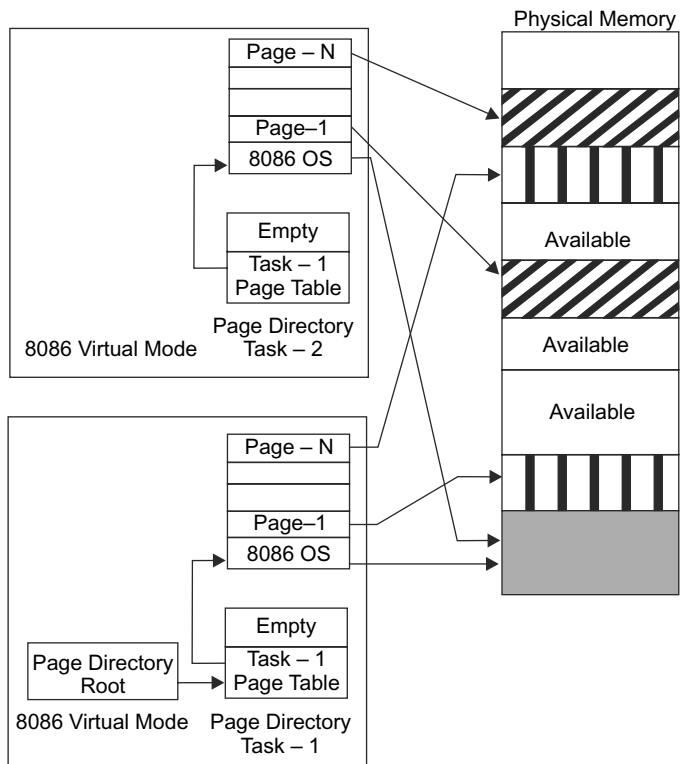


Fig. 11.45 Memory management in virtual 386 Mode

11.24 INSTRUCTION SET

The 80386 processor can support all the instructions of 80286. Usually, the instruction sets of 80286 are designed to operate with 8-bit or 16-bit data, but the same mnemonics may be executed with 8-bit, 16-bit, and 32-bit operands in the 80386 processor. Due to the enhanced architecture of 80386 over 80286, with additional general-purpose registers, segment registers and flag register, some new instructions were incorporated in the instruction set of 80286, to get the instruction set of 80386. As scaled addressing mode is added with 80386 processor, the number of instructions of 80386 increased significantly. The newly added instructions can be divided into the following functional groups as given below:

- ◆ Bit scan instructions
- ◆ Bit test instructions
- ◆ Conditional set byte instructions
- ◆ Shift double instructions
- ◆ Control transfer via gates instructions

Bit-Scan Instructions

The 80386 processor has two bit-scan instructions such as BSF (Bit-Scan Forward) and BSR (Bit-Scan Reverse). The BSF and BSR instructions can scan the operand for a '1' bit, without rotating the operand. The BSF instruction scans the operand from right to left whereas the BSR instruction scans the operand from left to right. When a '1' is encountered during the scan, a zero flag will be set and the bit position of '1' will be stored into the destination operand. If there is no '1', zero flag is reset.

Bit-Test Instructions

The 80386 CPU has four bit-test instructions, namely, BT (Test a Bit), BTC (Test a Bit and Complement), BTS (Test and Set a Bit) and BTR (Test and Reset a Bit). The BT, BTC, BTS and BTR instructions test the bit position in the destination operand which is specified by the source operand. The carry flag is affected whenever the bit position of the destination operand satisfies the condition specified in the mnemonics. If the bit position in the destination operand, specified by the source operand is '1' for BT instruction, the carry flag is set, or else the carry flag is cleared.

Conditional Set Byte Instructions

The conditional set byte instructions can set all the operand bits when the condition specified by the instruction is true. There are 16 conditional set byte instructions as given in Table 11.22.

Table 11.22 Conditional set byte instructions

<i>Instruction</i>	<i>Function</i>
SETO	Set on with overflow
SETNO	Set on without overflow
SETB/SETNAE	Set on below/ Set on not above or equal
SETNB/SETAE	Set on not below/ Set on above or equal
SETE/SETZ	Set on equal/ Set on zero
SETNE/SETNZ	Set on not equal/ Set on not zero
SETBE/SETNA	Set on below or equal/ Set on not above
SETNBE/SETA	Set on not below or equal/ Set on above
SETS	Set on sign
SETNS	Set on not sign

(Contd.)

(Contd.)

SETP/SETPE	Set on parity/ Set on parity even
SETNP	Set on not parity/parity odd.
SETL/SETNGE	Set on less/ Set on not greater or equal
SETNL/SET GE	Set on not less/ Set on greater or equal
SETLE/SETNG	Set on less or equal/ Set on not greater
SETNLE/SETG	Set on not less or equal/ Set on greater

Shift Double Instructions

The shift double instructions shift the specified number of bits from the source operand into the destination operand. The 80386 has two shift double instructions such as SHLD (Shift Left Double) and SHRD (Shift Right Double). The SHLD instruction is used to shift the number of bits specified in the instruction from the upper side. The SHRD instruction is used to shift the number of bits specified in the instruction from the lower side. For example,

<i>Instruction</i>	<i>Function</i>
SHLD EAX, ECX, 2	This instruction can shift 2 MSB bits of ECX into the LSB positions of EAX one by one starting from the MSB of ECX.
SHRD EAX, ECX, 6	This instruction is used to shift 6 LSB bits of ECX into the MSB positions of EAX one by one starting from the LSB of ECX.

Control Transfer Instructions

The 80386 instruction set does not have any additional instructions for the intrasegment jump. However, for intersegment jumps, it has got a set of new instructions which are variations of the previous CALL and JUMP instructions, and are to be executed only in the protected mode. These instructions are used by 80386 to transfer the control either at the same privilege or at a different privilege level. Also, different versions of control transfer instructions are available to switch between the different task types and TSS (Task State Segment). The corresponding RET instructions are also available to switch back from the new task initiated via CALL, JMP or INT instructions to the parent task.

11.25 COMPARISON BETWEEN 80286 AND 80386

The comparison between 80286 and 80386 is given Table 11.23.

Table 11.23 Comparison between 80286 and 80386

<i>80286 microprocessor</i>	<i>80386 microprocessor</i>
Intel 80286 was developed in 1983 by Intel and it is the improved version of 80186.	Intel 80386 was developed in 1985 using CHMOS III technology and it is the improved version of 80286.
80286 is a 16-bit processor.	80386 is the first 32-bit processor.
It is available with 134K transistors in a 68 Pin PGA package.	It is available with 275K transistors in a 132-pin PGA package.
80286 has operating frequency 8 MHz to 12.5 MHz.	80386 operates at clock speed of 16 MHz to 33 MHz.
The 80286 has a 16-bit data bus and a 24-bit address bus.	The 80386 has a 32-bit data bus and a 32-bit address bus.
The 80286 has 24-bit address lines and can able to access $2^{24}=16$ MB of physical memory	The 80386 has 32-bit address lines and is able to address up to $2^{32}=4$ GB physical memory
The 80286 processor supports Intel 80287 numeric data processor.	The 80386 processor supports Intel 80387 numeric data processor.
The 80286 processor has a 6-byte prefetch queue.	The 80386 processor has a 16-byte prefetch queue.

11.26 INTRODUCTION TO 80486

Due to the increasing demand for more sophisticated processing capability in advanced applications, the 80387 numeric data processors became compulsory for processors. Subsequently, the designers developed a new processor after incorporating the floating-point unit inside the CPU itself. The Intel 80486 is the first processor with an in-built 80387 floating-point unit and it is developed in 1989 using CHMOS IV technology. The features of the 80486 processor are given below:

- ◆ It has complete 32-bit architecture which can support 8-bit, 16-bit and 32-bit data types.
- ◆ 8 KB unified level 1 cache for code and data has been added to the CPU. In advanced versions of the 80486 processor, the size of level 1 cache has been increased to 16 KB.
- ◆ The 80486 is packaged in a 168-pin grid array package. The 25 MHz, 33 MHz, 50 MHz and 100 MHz (DX-4) versions of 80486 are available in the market.
- ◆ Execution time of instructions is significantly reduced. Load, store and arithmetic instructions are executed in just one cycle when data already exists in the cache.
- ◆ Intel 80486 operates at much faster bus transfers.
- ◆ This processor retains all complex instruction sets of 80386, and more pipelining has been introduced to improve performance in speed.
- ◆ Floating-point unit is integrated with 80486 processor. Hence the delay in communications between the CPU and FPU has been eliminated and all floating-point instructions are executed within very few CPU cycles.
- ◆ For fast execution of complex instructions, the 80486 has a five-stage pipeline. Two out of the five stages are used for decoding the complex instructions and the other three stages are used for execution.
- ◆ Clock-doubling and clock-tripling technology has been incorporated in faster versions of Intel 80486 CPU. These advanced i486 processors can operate in existing motherboards with 20–33 MHz bus frequency, while running internally at two or three times of bus frequency.
- ◆ Power management and System Management Mode (SMM) of 80486 became a standard feature of the processor.

The different variations of 80486 processors are manufactured, but two most common versions are 80486DX with integrated FPU and 80486SX without integrated FPU. The Intel 80486 microprocessor was developed for speeds up to 100 MHz. AMD486 produced at 120 and 133 MHz versions of the 80486, and also manufactured in small quantities the 150 MHz and possibly 166 MHz versions. In this section basic architecture, pin functions of 80486 are explained.

11.27 ARCHITECTURE OF 80486

The 80486DX is a 32-bit processor. Figure 11.46 shows the simplified block diagram of 80486 and the internal architecture of 80486 is depicted in Fig. 11.47. The architecture of Intel's 80486 can be divided into three different sections such as

- ◆ Bus interface unit (BIU),
- ◆ Execution and control unit (EU), and
- ◆ Floating-point unit (FU).

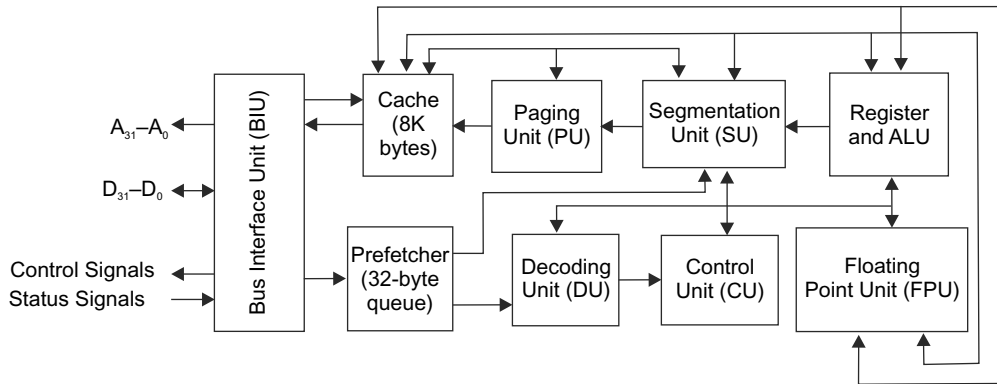


Fig. 11.46 The simplified block diagram of 80486

Bus Interface Unit (BIU)

The bus interface unit is used to organize all the bus activities of the processor. The address driver is connected with the internal 32-bit address output of the cache and the system bus. The data bus transceivers are interconnected between the internal 32-bit data bus and system bus. The write data buffer is a queue of four 80-bit registers and is able to hold the 80-bit data which will be written to the memory. Due to pipelined execution of the write operation, data must be available in advance. To control the bus access and operations, the following bus control and request sequencer signals \overline{ADS} , W/\overline{R} , D/\overline{C} , M/\overline{IO} , PCD , PWT , \overline{RDY} , \overline{LOCK} , \overline{PLOCK} , \overline{BOFF} , $A20M$, $BREQ$, $HOLD$, $HLDA$, $RESET$, $INTR$, NMI , \overline{FERR} and \overline{IGNNE} are used.

Execution Unit (EU) and Control Unit (CU)

The burst control signal updates the processor that the burst is ready. This signal works as a ready signal in the burst cycle. The \overline{BLAST} output shows that the previous burst cycle is over. The bus size control signals $\overline{BS16}$ and $\overline{BS8}$ indicates dynamic bus sizing. The cache control signals \overline{KEN} , \overline{FLUSH} , \overline{AHOLD} and \overline{EADS} are used to control the cache control unit.

The parity generation and control unit generates the parity and carries out the checking during the processor operation. The boundary scan control unit of the processor performs boundary scan tests operation to ensure the correct operation of all components of the circuit on the mother board.

The prefetcher unit fetches the codes from the memory and arranges them in a 32-byte code queue. The function of the instruction decoder is to receive the code from the code queue and then decodes the instruction code sequentially. The output of the decoder is fed to the control unit to derive the control signals, which are used for execution of the decoded instructions. Before execution, the protection unit should check all protection norms. If there is in any violation, an appropriate exception is generated.

The control ROM stores a microprogram to generate control signals for execution of instructions. The register bank and ALU are used for their usual operation just like they perform in 80286. The barrel shifter is used to perform the shift and rotate algorithms. The segmentation unit, descriptor registers, paging unit, translation look aside buffer and limit and attribute PLA are worked together for the virtual memory management. These units also provide protection to the op-codes or operand in the physical memory.

Floating-point Unit (FPU)

The floating-point unit and register banks of FPU communicate with the bus interface unit (BIU) under the control of memory management unit (MMU), through a 64-bit internal data bus. Generally, the FPU is used for mathematical data processing at very high speed as compared to the ALU.

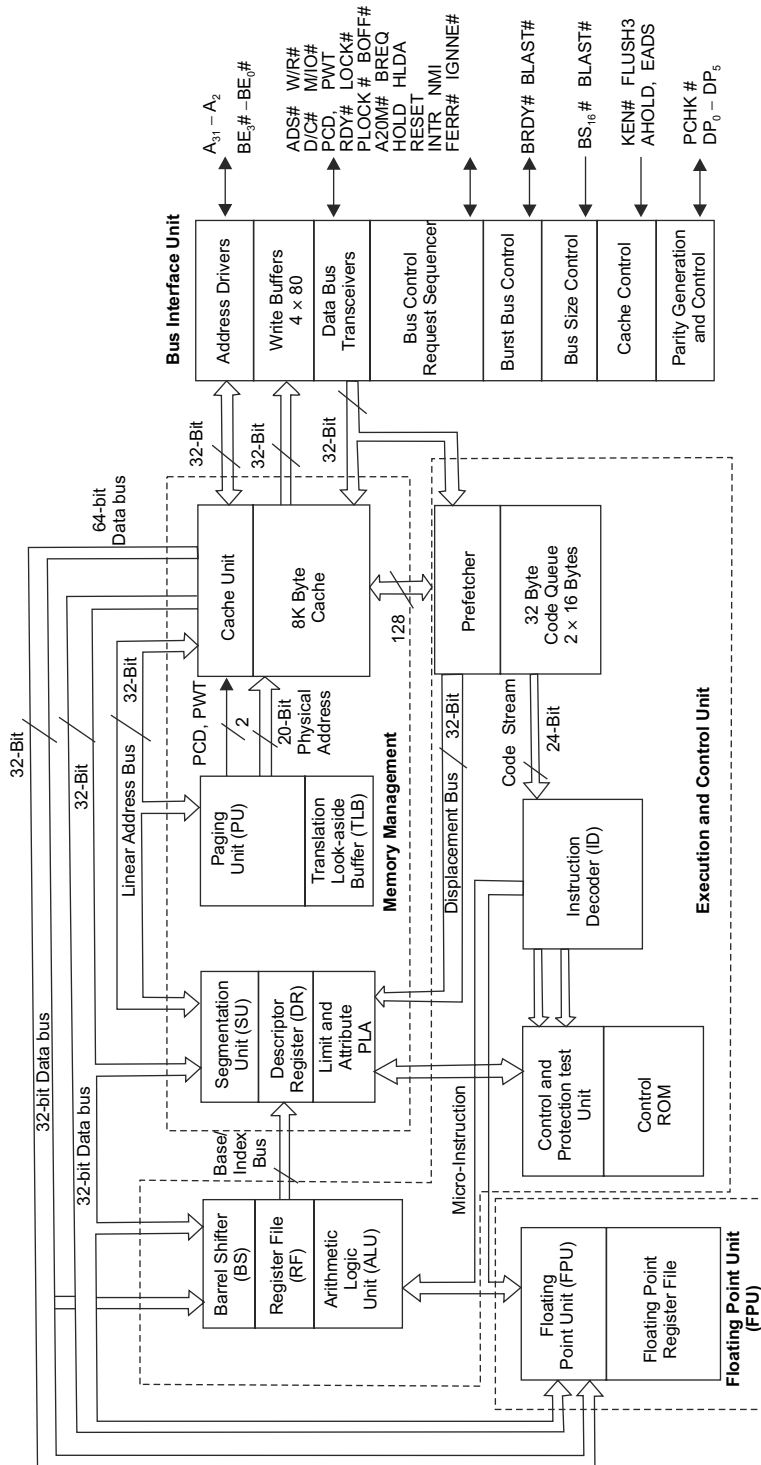


Fig. 11.47 Internal architecture of 80486

11.27.1 Register of 80486

The registers of the 80486 processor are same as the 80386 processor, except for the flag register. Figure 11.48 shows the flag register. As compared to the flag register of 80386, the flag register of 80386 has only one additional flag called alignment check flag or AC flag. The D₁₈ position of the flag register is AC flag as depicted in Fig. 11.48. When the AC flag bit is set to '1', there is an access to a misaligned address and an exception (fault) will be generated. The alignment faults are generated only at privilege level 3.

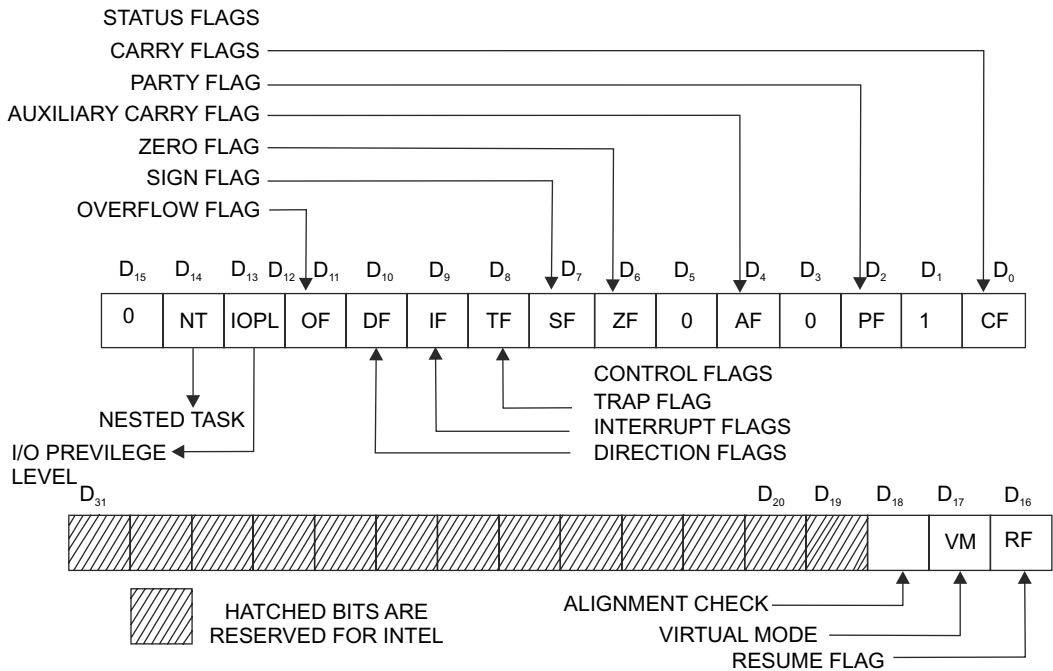


Fig. 11.48 Flag register of 80486

11.28 PIN DESCRIPTIONS OF 80486

The pin diagram of 168-pin PGA (Pin Grid Array) package 80486 is shown in Fig. 11.49 and the schematic pin diagram of 80486 processor is depicted in Fig. 11.50. All signals of 80486 are grouped according to their functions. Some of the most important groups of signals are explained in this section.

✓ **CLK** The CLK input signal provides the timing for the operation of 80486.

Address Bus

✓ **A₃₁–A₂** The address lines A₃₁–A₂ are used for selecting memory and I/O devices.

✓ **\overline{BE}_3 – \overline{BE}_0** For memory and I/O addressing, byte enable signals \overline{BE}_3 – \overline{BE}_0 are required. When the byte enable signals \overline{BE}_3 – \overline{BE}_0 are active-low, these indicate which byte of the 32-bit data bus is active during the read or write cycle. For example, when $\overline{BE}_0 = 0$, the least significant byte is active. In the same way, if $\overline{BE}_3 = 0$, the most significant byte in 32-bit data is accessed.

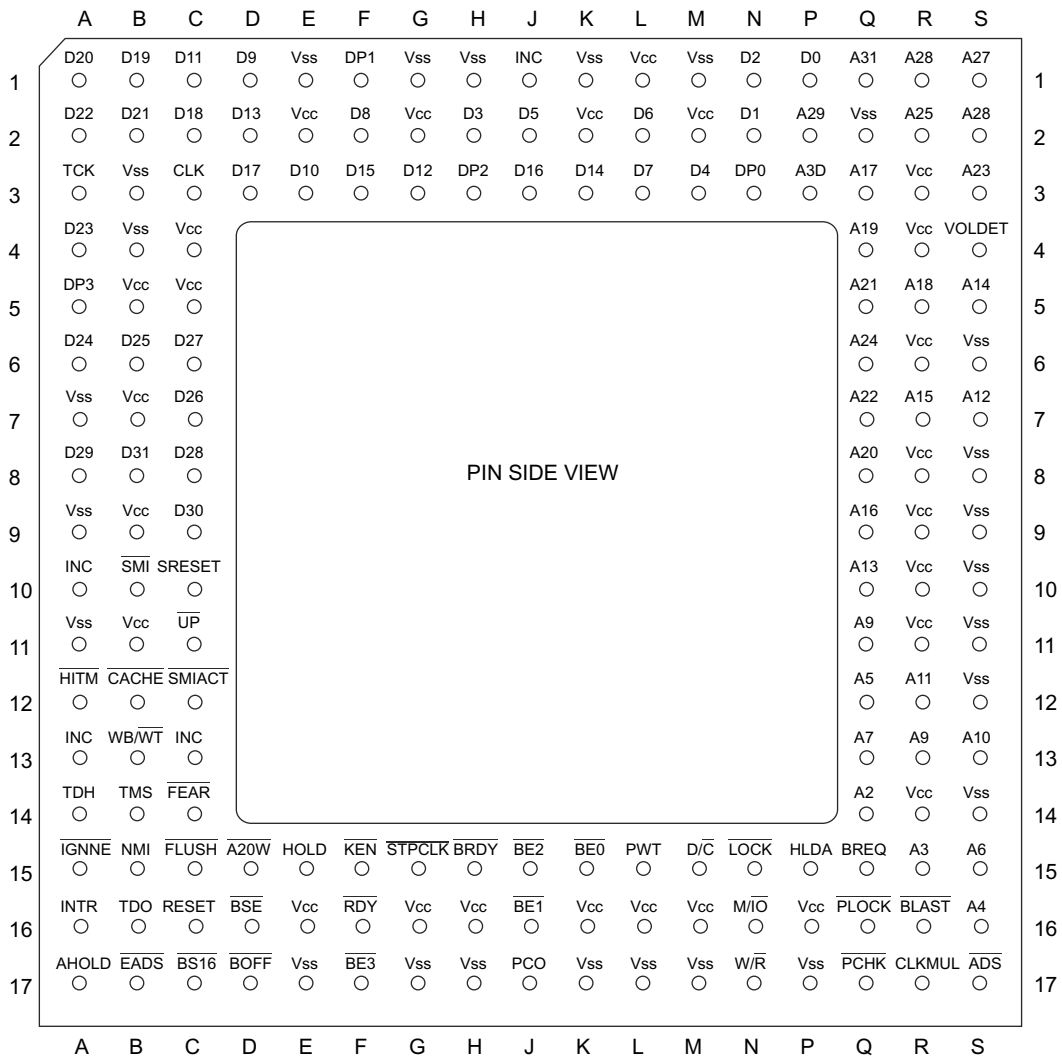


Fig. 11.49 Pin diagram of 80486 processor

Data Bus

✓ **D₃₁–D₀** The data lines D₃₁–D₀ are a bidirectional data bus. D₃₁ is the most significant data bit and D₀ is the least significant data bit.

Data Parity Group

These pins are used to detect the parity during the memory read and write operations.

✓ **DP₃–DP₀** The four data parity input/output pins are DP₃–DP₀. These pins are used to represent the individual parity of 32 bits (4 bytes) of the data bus.

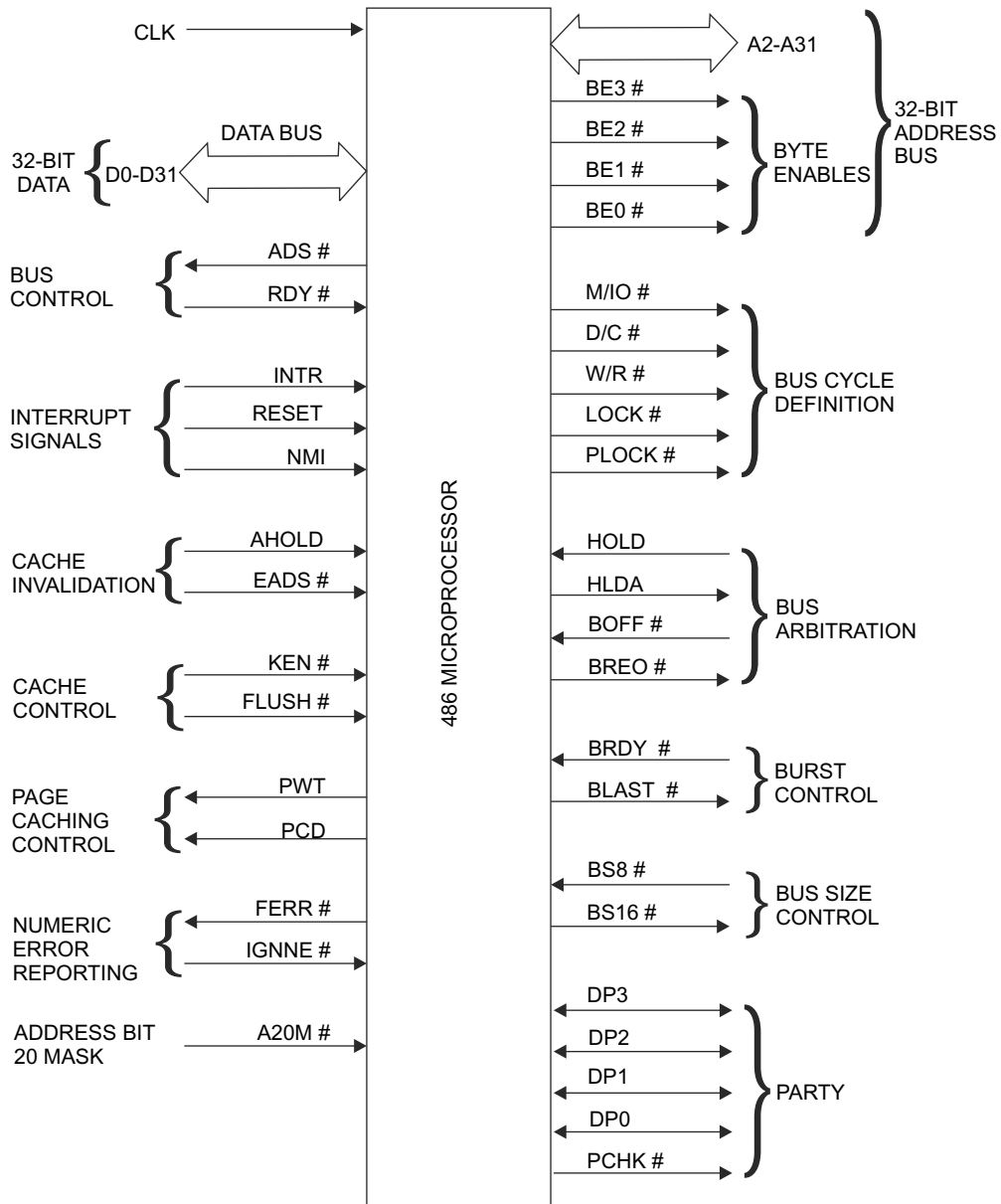


Fig. 11.50 Schematic pin diagram of 80486 processor

Bus Cycle Definition Group

- ✓ M/\overline{IO} The M/\overline{IO} output pin is used to differentiate between memory and I/O operations.
- ✓ W/\overline{R} The W/\overline{R} output pin is used to differentiate between read and write bus cycles.
- ✓ D/\overline{C} : This output pin can be used to differentiate between data and control operations.

- ✓ **\overline{LOCK}** This output pin is used to state that the current bus cycle is locked.
- ✓ **\overline{PLOCK}** The \overline{PLOCK} (pseudo lock) output pin indicates that the current operation requires more than one bus cycle to complete the task. This signal is at logic level 0 for arithmetic co-processor operations.

Bus Control Group

- ✓ **\overline{ADS}** The \overline{ADS} (address data strobe) output pin indicates that the address bus contains a valid memory address.
- ✓ **\overline{RDY}** The \overline{RDY} (ready) input pin acts as a ready signal and this signal is used for the current nonburst cycle.

Burst Control Group

- ✓ **\overline{BRDY}** The \overline{BRDY} (burst ready) input pin indicates the burst mode of memory read or memory operation. During burst mode, the speed of memory access may be doubled compared to normal memory read/write operations.
- ✓ **\overline{BLAST}** When the \overline{BLAST} (burst last) output signal is high, it indicates that CPU initiates the burst mode of memory access. If this signal is low, it indicates that the burst bus cycle is completed and the \overline{BRDY} signal is next asserted for starting the next memory access operation.

Bus Arbitration Group

- ✓ **$HOLD$** The $HOLD$ pin acts as a local bus hold input. This pin may be activated by another bus master like DMA controller. This pin is functionally similar to the $BREQ$ pin.
- ✓ **$HLDA$** The $HLDA$ output signal is used to acknowledge the receipt of a valid $HOLD$ request.
- ✓ **$BREQ$** When the $BREQ$ (bus request) output signal is active-high, it indicates that the 80486 CPU has generated a internal bus request.
- ✓ **\overline{BOFF}** When the \overline{BOFF} (back off) input pin is at logic level 1, 80486 CPU places its buses at hold state. The active-high back off input signal forces the current bus master of 80486 CPU to release the bus in the next clock cycle.

Interrupts

- ✓ **$INTR$** The $INTR$ pin is a maskable interrupt input. It is controlled by the IF in the flag register.
- ✓ **NMI** This is a type-2 nonmaskable interrupt input.
- ✓ **$RESET$** The $RESET$ input pin is used to reset the processor when it becomes high.

Cache Control Group

- ✓ **\overline{KEN}** The \overline{KEN} (cache enable) input pin is used to decide whether the current cycle is cacheable or not.
- ✓ **\overline{FLUSH}** The \overline{FLUSH} is a cache flush input signal. When this pin is activated, it clears the cache contents and validity bits.

Cache Invalidation Group

- ✓ **$AHOLD$** The $AHOLD$ (address hold request) input signal enables other bus masters to use the 80486 system bus for a cache invalidation cycle.

✓ **\overline{EADS}** The \overline{EADS} (external address) input signal is used to indicate that a valid address for external bus cycle is available on the address bus.

Page Cache Ability Group

✓ **PCD** The PCD (page cache disable) output pin reflect the status of the PCD attribute bit in page table or page directory entry.

✓ **PWT** The PWT (page write-through) output pin indicates the status of the PWT attribute bit in page table or page directory entry.

Bus Size Control Group

✓ **$\overline{BS8}$** The $\overline{BS8}$ (bus size 8) input pin is used for the 8-bit dynamic bus sizing feature of 80486 and the 80486 can be interfaced with 8-bit external devices.

✓ **$\overline{BS16}$** The $\overline{BS16}$ (bus size 16) input pin is used for the 16-bit dynamic bus sizing feature of 80486 and the 80486 can be interfaced with 16-bit external devices.

✓ **$\overline{A20M}$** When the $\overline{A20M}$ (address bit 20 mask) input pin is activated, the 80486 masks the physical address line A20 before performing any memory or cache cycle. This is very useful to wrap the physical address space from 00FFFFH to 000000H as the 8086 microprocessor does and it works around the 1 Mbyte memory size, i.e. physical memory space of 8086 in virtual 8086 mode.

FPU Error Group

✓ **FERR** The FERR output signal is activated whenever the floating point unit generates any error.

✓ **IGNNE** When the IGNNE (Ignore Numeric Processor Extension) input pin is activated, the 80486 ignores the floating processor (FPU) errors and execute non-control floating-point instructions continuously.

Test Access Port Group

The test access is a unique facility, which is available in the 50 MHz version 80486. The test access signals are used to check the fault conditions of the components on the motherboard.

✓ **TDI** The TDI (test data) input of 80486 is the serial input pin which is used to shift the JTAG instructions and data into components.

✓ **TCK** The TCK (test clock) input provides the basic clock signal, which is required by the boundary test feature.

✓ **TDO** The TDO (test data output) pin is the serial output pin which is used to shift the JTAG instruction and data out of components under test.

✓ **TMS** The TMS (test mode select) input is decoded by the JTAG TAP (tap access port) to select the operation of this test logic.

Supply Lines

✓ **V_{CC}** The V_{CC} pin is connected to +5 V power supply. There are 24 V_{CC} pins in the 80486 processor.

Ground Lines

✓ **V_{SS}** The V_{SS} pin is connected to the ground terminal of power supply. There are 28 V_{SS} pins in the 80486 processor.

11.29 COMPARISON BETWEEN 80386 AND 80486

The comparison between 80386 and 80486 is given Table 11.24.

Table 11.24 Comparison between 80386 and 80486

80386 Microprocessor	80486 Microprocessor
Intel 80386 was developed in 1985 using CHMOS III technology and it is the improved version of 80286.	Intel 80486 was developed by Intel in 1989 using CHMOS IV technology and it is the improved version of 80386.
It is available with 275K transistors in a 132-pin PGA package.	The 80486 is available with 1200K transistors packaged in a 168-pin grid array package.
80386 operates at clock speed of 16 MHz to 33 MHz.	The 25 MHz, 33 MHz, 50 MHz and 100 MHz (DX-4) versions of 80486 are available in the market.
On-Chip cache is not available in 80386.	8 KB unified level-1 cache for code and data is available to the CPU. In advanced versions of the 80486 processor, the size of level 1 cache has been increased to 16 KB.
4.0 MIPS at 25 MHz. Most of the instructions require 2 CLK for execution.	15.0 MIPS at 25 MHz. Most of the instructions require 1 CLK for execution
The 80386 processor supports Intel 80387 numeric data processor. There is no on-chip numeric data processor.	Numeric data processor (floating point unit) is integrated with the 80486 processor. Therefore the delay in communications between the CPU and FPU has been eliminated and all floating point instructions are executed with in very few CPU cycles.
The 80386 processor has a 16-byte prefetch queue.	80486 processor has a 32-byte prefetch queue.
The 80386 processor has no multiprocessing support capability.	The 80386 processor has multiprocessing support capability.
The 80386 has less power management capability compared to 80486.	The 80486 has 2–3 times more power management capability compared to 80386
There is no RISC feature in 80386.	RISC feature is incorporated in 80486.
The alignment check (AC) flag does not exist in 80386.	The Alignment Check (AC) flag exists in 80486.

SUMMARY

- This chapter starts with an introduction of the 80186 microprocessor. The basic architecture, register set, memory organisation, addressing modes, data types and instruction set of 80186 has been discussed elaborately. The pin diagram of 80186 microprocessors and function of each pin have been discussed.
- The 80286 processor is an advanced, high-performance microprocessor with specially optimized capabilities for multi-user and multitasking systems. The 80286 has built-in memory protection that supports the operating system and task isolation as well as program and data privacy. In this chapter, architecture features of 80286 such as register set, addressing modes, and data types are discussed in detail. The pin descriptions, instruction set and memory management feature of the 80286 microprocessor are also incorporated in this chapter.

- The first 32-bit processor is 80386 which was introduced after 80286. The internal architecture and register set of the 80386 microprocessor are explained in this chapter. The signal descriptions, addressing modes, data types and instruction set of 80386 are discussed. The real address mode of operation and protected virtual address mode of operation has been explained in detail. The paging unit, a new feature of 80386 is also discussed elaborately along with the virtual 8086 mode of operation of 80386.
- With the increasing demand for more sophisticated processing capability in advanced applications, the 80387 numeric data processor has been incorporated with processors. The Intel 80486DX is the first processor with an in-built 80387 floating-point unit. In this chapter, the basic features of 80486 processors are enlisted. The internal architecture, flag register and signal descriptions of 80486 processors are discussed. The comparison between 8086 and 80186, 8086 and 80286, 80186 and 80286, 80286 and 80386, 80386 and 80486 are given in this chapter.

MULTIPLE-CHOICE QUESTIONS

-
- | | |
|--|--|
| <p>11.1 80286 has</p> <p>(a) 24-bit address bus and 16-bit data bus
 (b) 14-bit address bus and 16-bit data bus
 (c) 16-bit address bus and 16 bit data bus
 (d) 16-bit address bus and 24-bit data bus.</p> <p>11.2 The bus unit of 80286 has</p> <p>(a) 32-byte prefetch queue
 (b) 16-byte prefetch queue
 (c) 12-byte prefetch queue
 (d) 6-byte prefetch queue</p> <p>11.3 Instruction unit of 80286 consists of</p> <p>(a) 6 decoded instruction queues
 (b) 3 decoded instruction queues
 (c) 2 decoded instruction queues
 (d) 1 decoded instruction queues</p> <p>11.4 The format of a cache register is</p> <p>(a) 6-byte or 48-bit format
 (b) 5-byte or 40-bit format
 (c) 4-byte or 32-bit format
 (d) 3-byte or 24-bit format</p> <p>11.5 The concept of memory management, privilege and protection are incorporated in</p> <p>(a) 8088 (b) 8086
 (c) 80186 (d) 80286</p> <p>11.6 80286 can be operated in</p> <p>(a) real mode only
 (b) protected virtual mode only</p> | <p>(c) both real and protected virtual modes
 (d) virtual 8086 mode</p> <p>11.7 80386Dx is a _____ processor</p> <p>(a) 16-bit (b) 24-bit
 (c) 32-bit (d) 64-bit</p> <p>11.8 80386 has</p> <p>(a) 24-bit address bus and 16-bit data bus
 (b) 32-bit address bus and 32-bit data bus
 (c) 24-bit address bus and 32-bit data bus
 (d) 16-bit address bus and 32-bit data bus</p> <p>11.9 80486 is the combination of</p> <p>(a) 80386 and 80387
 (b) 80386 and 80287
 (c) 80286 and 80387
 (d) 80286 and 80287</p> <p>11.10 The memory management unit consists of</p> <p>(a) segmentation unit
 (b) paging unit
 (c) both segmentation unit and paging unit</p> <p>11.11 The instruction prefetcher of 80386 processor consists of</p> <p>(a) 16-byte instruction code queue
 (b) 12-byte instruction code queue
 (c) 10-byte instruction code queue
 (d) 6-byte instruction code queue</p> |
|--|--|

- 11.12 80386 can be operated in
 (a) real mode only
 (b) protected virtual mode only
 (c) real and protected virtual modes only
 (d) real, protected virtual mode and virtual 8086 mode
- 11.13 The segmental unit allows maximum
 (a) 4 GB segment (b) 4 MB segment
 (c) 64 KB segment (d) 4 KB segment
- 11.14 The paging unit organises the physical memory in terms of pages of
 (a) 8 KB size page (b) 4 KB size page
 (c) 4 MB size page (d) 4 GB size page
- 11.15 The difference between the 80386 and 80486 flag register is
 (a) alignment check flag
 (b) virtual mode flag
 (c) resume flag
 (d) trap flag

SHORT-ANSWER-TYPE QUESTIONS

- 11.1. Write the length of the address and data buses of the following processors:
 (i) 80186 (ii) 80286 (iii) 80386 (iv) 80486
- 11.2. What are the different registers of the 80186 microprocessor?
- 11.3. What are the different types of data supported by the 80186 microprocessor?
- 11.4. How many new instructions are available in 80186 with respect to 8086?
- 11.5. What are different features of the 80286 processor?
- 11.6. What are the different addressing modes of 80286?
- 11.7. Write the difference between Real Address Mode and Protected Virtual Address Mode (PVAM).
- 11.8. What are the different interrupts available in 80286?
- 11.9. What are the advantages the 80286 microprocessor with respect to the 8086 microprocessor?
- 11.11. Mention the three operating modes of Intel 80386 processor.
- 11.12. What do you mean by paging? What are the advantages of paging?
- 11.13. What are the difference between logical address, linear address and physical address?
- 11.14. What is translation look-aside buffer? How can it increase the speed of execution of programs?
- 11.15. What are the new features of 80486 over 80386?
- 11.16. What is the difference between flag register of 80486 and 80386?

REVIEW QUESTIONS

- 11.1. Draw the block diagram of the 80186 microprocessor and explain its operation.
- 11.2. Explain memory organization of the 80186 microprocessor.
- 11.3. Write the functions of the following pins of 80186:
 (i) DRQ0, DRQ1 (ii) \overline{WR} / OS1 (iii) \overline{LOCK} (iv) \overline{DEN}
 (v) \overline{RD} / \overline{QSMD}

- 11.4. Discuss the different addressing modes of 80186 microprocessor with suitable examples.
- 11.5. Explain the operation of the following instructions of 80186:
 (i) IMUL (ii) BOUND (iii) ENTER (iv) INS
 (v) OUTS
- 11.6. Write the difference between 8086 and 80186.
- 11.7. Draw the internal block diagram of the 80286 microprocessor and discuss its operation.
- 11.8. Draw and discuss the flag register of 80286.
- 11.9. What are different registers of 80286?
- 11.10. Explain the function of the following pins of 80286:
 (i) \overline{ERROR} (ii) \overline{PEACK} (iii) CAP (iv) COD / \overline{INTA}
 (v) $A_{23}-A_0$
- 11.11. Write the operations of the following instructions of 80186:
 (i) ARPL (ii) VERR/VERW (iii) LAR (iv) LGDT/LIDT
 (v) SGDT (vi) SMSW
- 11.12. Discuss Real Address Mode and Protected Virtual Address Mode (PVAM) operation of 80286.
- 11.13. How is the physical address computed in Real Address Mode of 80286.
- 11.14. Explain the concept of virtual memory.
- 11.15. What do you mean by a descriptor? Discuss different types of descriptor supported by the 80286 and their applications.
- 11.16. Write short notes on the following:
 (i) Interrupt descriptor table (ii) Local and global descriptor tables
 (iii) Segment descriptor cache registers (iv) Privilege
- 11.17. What are the different addressing modes of 80286? How can the 80286 enter into PVAM ?
- 11.18. What are the different data types supported by 80286?
- 11.19. Show how the virtual to physical address translation takes place in 80286.
- 11.20. Give a list of features of 80386
- 11.21. Draw the internal block diagram of the 80386 microprocessor and discuss its operation in detail.
- 11.22. Draw the register set of 80386 processor and explain the operation of each register in brief.
- 11.23. Draw the flag register of the 80386 processor and discuss operation of each flag.
- 11.24. Explain the function of the following pins of 80386:
 (i) $ADS\#$ (ii) $BE3\#-BE0\#$ (iii) $W/R\#$ (iv) $D/C\#$
 (v) \overline{LOCK} (vi) BS_{16} (vii) M / \overline{IO} (viii) $D_{31}-D_0$
- 11.25. Write short notes on the following:
 (i) Segment descriptor registers (ii) Control registers
 (iii) System address registers (iv) Debug and test registers
- 11.26. Discuss the different addressing modes of 80386.

Chapter 12

Pentium and RISC Processors

12.1 INTRODUCTION

After the development of the 80486 processor in 1989, Intel started to work on the next generation of processors and in 1993, Intel developed the fifth-generation processor 80586 (P5) known as Pentium processor. The name Pentium was derived from the Greek word *pentē*, meaning 'five', and the Latin ending *-ium*. The term 'Pentium processors' refers to a family of microprocessors which can share a common architecture and instruction set. Since 1993, various versions of Pentium processors have been evolved incorporating new features. The features of the Pentium processor are as follows:

- ◆ 5 V processor fabricated in 0.8-micron Bipolar Complementary Metal Oxide Semiconductor (BiCMOS) technology
- ◆ Runs at a clock frequency of 60 MHz or 66 MHz and has 3.1 million transistors
- ◆ Two independent integer pipelines and a floating-point pipeline
- ◆ Branch prediction
- ◆ Compatibility with 80386 instructions through a microprogrammed CISC unit
- ◆ Separate code and data caches
- ◆ Wider 64-bit data bus with pipelined burst mode for quicker cache line fills and write backs
- ◆ Memory management unit for paging
- ◆ System management to implement power-save functions
- ◆ Compatibility with all x 86 and x 87 predecessors
- ◆ Dual processor support with on-chip Advanced Programmable Interrupt Controller (APIC)

12.2 PENTIUM INTERNAL ARCHITECTURE

The Pentium is a 32-bit processor, but it has a 32-bit address bus and a 64-bit data bus. This processor's data bus serves the on-chip caches, but not the 32-bit registers. The internal and external data buses are connected through the caches. Figure 12.1 shows the internal architecture of the Pentium processor which consists of 8K byte code cache, 8K byte data cache, Translation Look-aside Buffer (TLB), Branch Trace Buffer (BTB), Integer pipelines U and V, floating-point pipeline, Microcode ROM, and Control Unit (CU).

Code and Data Cache

There are separate code and data caches, and the cache line size is 32 bits just like the 80486 processor. Each cache is connected with its own Translation Look-aside Buffer (TLB). Therefore, the paging unit of the Memory Management Unit (MMU) can rapidly convert linear code or data addresses into physical addresses. Due to two separate caches, the pre-fetches cannot conflict with data access cycles.

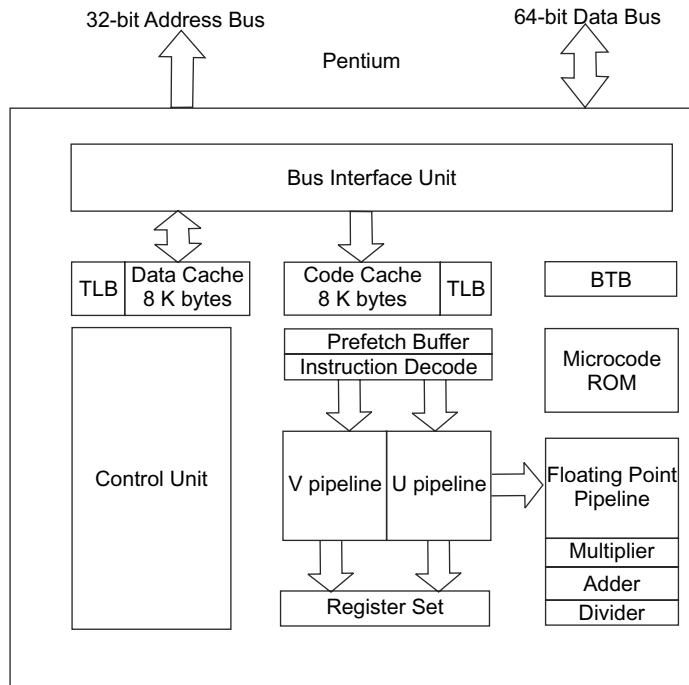


Fig. 12.1 Block diagram of Pentium architecture

Branch Prediction

Branch prediction consists of a Control Unit (CU) and a Branch Trace Buffer (BTB). The function of control unit and Branch trace buffer are as follows:

✓ **Branch Trace Buffer (BTB)** The BTB is used to store the target address and statistical information about the branch operation. Hence, the branch prediction is able to predict branches and cause the Pentium to use the most likely target address for instruction fetching. Pipeline freeze up caused by pipeline flushes and the subsequent fetching operations are reduced and the program execution is accelerated.

✓ **Control Unit (CU)** The control unit controls the five-stage integer pipelines U and V, and the eight-stage floating-point pipeline. In the Pentium processor, the integer pipelines are used for all instructions which are not involved in any floating-point operations. Therefore, the Pentium can transmit two integer instructions in the same clock cycle and performance of the processor is improved. This method is called *superscalar architecture*. Figure 12.2 shows the superscalar organization of the Pentium processor.

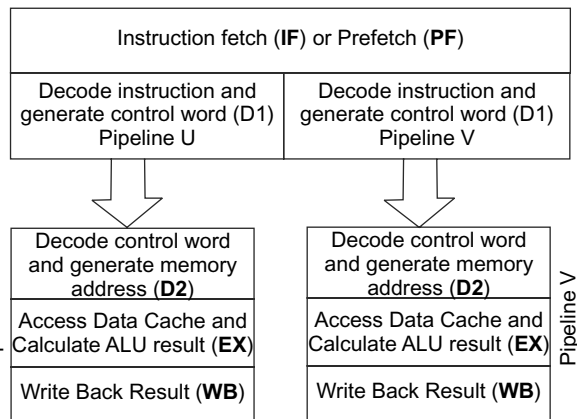


Fig. 12.2 Superscalar organization of Pentium processor

The first four stages of the floating-point pipeline overlap with the U pipeline and the

parallel operation of the integer and floating-point pipelines is possible only under some specified conditions. If the operating clock frequency of Pentium is same as 80486, the Pentium floating-point unit is able to execute floating-point instructions 3 to 5 times faster than 80486. This is possible as a hardware multiplier, divider and quicker algorithms are incorporated in the microcode floating-point unit.

The Pentium has a microcode support unit to support complex functions. The support unit controls the pipelines with the microcode. Actually, this unit uses both pipelines together. Therefore, complex microcode instructions run very fast on a Pentium than on a 80486.

12.2.1 Integer Pipelines U and V

The Pentium is a superscalar processor and it has two integer pipelines, called U and V. The process of issuing two instructions in parallel is known as *pairing*.

The U-pipeline is able to handle the full instruction set of the Pentium but the V-pipeline has limited handling capability. The V-pipeline is able to handle only simple instructions without any microcode support. The V-pipeline is used to execute 'simple integer instructions' such as load/store type instructions and the FPU instruction FXCH, but the U-pipeline executes any legitimate Pentium instructions. Actually, Pentium processors use a set of pairing rules to select a simple instruction which can go through the V pipeline. When instructions are paired, initially the instruction is issued to the U-pipe and then the next sequential instruction is issued to the V-pipe.

There are two integer pipelines and a floating-point unit in the Pentium processor. Figure 12.3 shows an integer pipeline. Each integer unit has the basic five-stage pipeline as given below:

- ◆ Prefetch (PF)
- ◆ Decode-1 (D1)
- ◆ Decode-2 (D2)
- ◆ Execute (E)
- ◆ Write Back (WB)

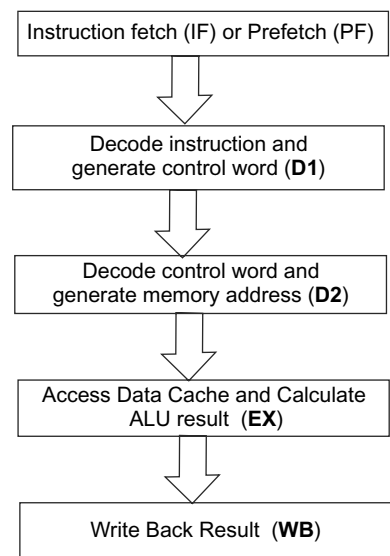


Fig. 12.3 Integer pipeline

Prefetch (PF) In the prefetch stage of integer pipeline of the Pentium processor, instructions are fetched from the instruction cache as instructions are stored initially in the instruction cache. After fetching, the CPU aligns the codes properly. As the instructions are of variable lengths, the initial opcode bytes of each instruction must be properly aligned. After completion of the prefetch stage, the decode stages D1 and D2 will be executed.

Decode-1 (D1) In the decode-1 (D1) pipeline stage, the CPU decodes the instruction and generates a control word. The D1 pipeline stage has two parallel instruction decoders. These implement the pairing rules. Only a single control word may be sufficient to start execution of the data transfer, arithmetic and logical operations in RISC processor. This processor supports complex CISC instructions with the help of microcoded control sequencing.

Decode-2 (D2) The decode-2(D2) pipeline stage is required whenever the control word from D1 stage is decoded to complete the instruction decoding. In this stage, the CPU generates addresses for data memory.

Execute (E)

The execution stage is used for both ALU operations and data cache access. The data cache is used for data operands and ALU performs arithmetic logic computations or floating-point operations. In the execution stage, all U-pipe and V-pipe instructions, except conditional branches, are verified for correct branch prediction. A microcode is designed to use both the U and V pipes. Therefore, microcode instructions are executed faster on the Pentium than on the 80486.

Write Back (WB)

The final stage of the five-stage pipeline is Write Back (WB). In the WB stage, the CPU updates the contents of registers and status of the flag register after completion of execution. In this stage, the V-pipeline conditional branch instructions are verified for correct branch prediction.

The Pentium pipeline structure is similar to 80486 pipeline structure. Usually, the 80486 takes two clock cycles to decode instructions, but the Pentium processor takes only one clock cycle as Pentium processor has an additional integrating hardware in each pipeline stages to speed up the process.

12.2.2 Superscalar

The Pentium processor architecture has been designed based on Superscalar. In Superscalar architecture, two instructions are executed in parallel. Figure 12.2 shows the superscalar architecture. Two independent integer pipelines are depicted in Fig. 12.2. In the PF and D1 stages, the microprocessor can fetch, instructions decode instructions and generate control words. In this stage, decoded instructions issue them to two parallel U and V pipelines. For complex instructions, D1 generates microcoded sequences for U and V pipelines. Several techniques are used to resolve the pairing of instructions.

12.2.3 Floating-Point Unit

The 80486DX CPU is the first processor in which the 80387 math co-processor has been incorporated on-chip to reduce the communication overhead. The 80486 CPU contains a floating-point unit, but this floating-point unit is not pipelined. The Pentium processor has been designed for incorporating on the chip numeric data processor. The Floating-Point Unit (FPU) of Pentium has an eight-stage pipeline as shown in Fig. 12.4. The eight pipeline stages are

- ◆ Prefetch (PF)
- ◆ Decode-1 (D1)
- ◆ Decode-2 (D2)
- ◆ Execute (dispatch)
- ◆ Floating Point Execute-1 (X1)
- ◆ Floating Point Execute-2 (X2)
- ◆ Write Float (WF)
- ◆ Error Reporting (ER)

The first five stages of the pipeline are similar to the U and V integer pipelines. During the operand fetch stage, the FPU fetches the operands either from the floating-point register or from the data cache. The floating-point unit has eight general-purpose floating point registers. There are two execution stages in Pentium such as the first execution stage (X1 stage) and the second execution stage (X2 stage). In the X1 and X2 stages, the floating-point unit reads the data from the data cache and executes the floating-point calculation.

Prefetch (PF)

The prefetch stage is same as the integer pipeline of Pentium processor.

Decode-1 (D1)

The decode-1 (D1) pipeline stage is also same as the integer pipeline of Pentium processor.

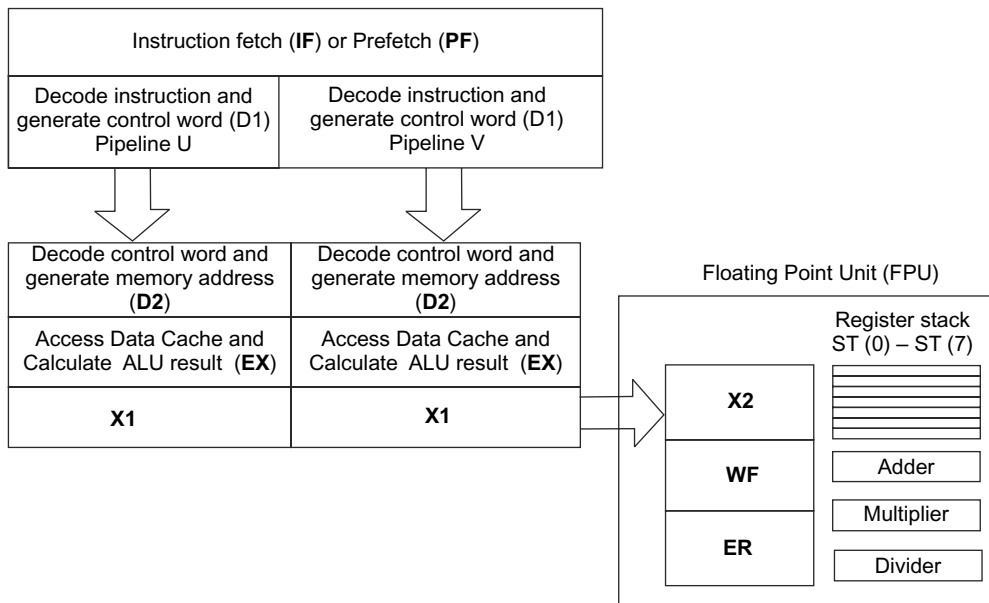


Fig. 12.4 Floating-point pipeline

Decode-2 (D2) The decode-2 (D2) pipeline stage is worked as required whenever the control word from D1 stage is decoded to complete the instruction decoding. In this stage, it is the integer pipeline of Pentium processor.

Operand Fetch During the execution stage (E), the floating-point unit accesses the data cache and the floating-point register to fetch operands. Before writing the floating-point data to the data cache, the floating-point unit converts internal data format into appropriate memory representation format.

Floating Point Execute-1 (X1) In the Floating Point Execute-1 (X1) stage, the floating-point unit executes the first steps of the floating-point calculations. While reading the floating-point data from the data cache, the floating-point unit writes the data into the floating-point register.

Floating Point Execute-2 (X2) During the Floating Point Execute-2 (X2) stage, the Floating Point unit execute the remaining steps of the floating-point computations.

Write Float (WF) In the Write Float (WF) stage, the floating-point unit completes the execution of the floating-point calculations and then writes the computed result into the floating-point register file.

Error Reporting (ER) In the error reporting(ER) stage, the floating-point unit generates a report about the internal special situations and updates the floating point status.

The floating-point unit of Pentium consists of a dedicated adder, multiplier and division units. All independent circuits are used to perform addition, multiplication, division and other mathematical operations within very few clock cycles.

The block diagram of the floating-point unit is depicted in Fig. 12.5. There are five segments such as Floating-point Adder Segment (FADD), Floating-point Multiplier Segment (FMUL), Floating-point Divider

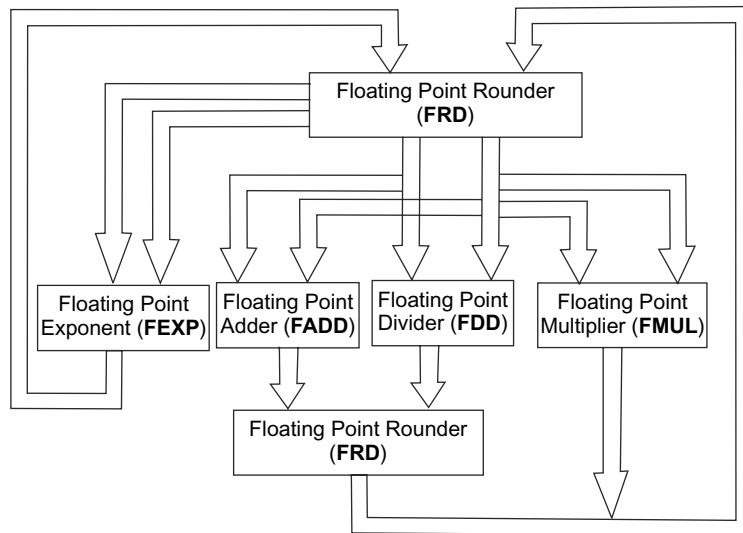


Fig. 12.5 Block diagram of floating-point unit

Segment (FDIV), Floating-point Exponent Segment (FEXP) and Floating-point Runder Segment (FRD) in the floating-point unit of Pentium processors. These segments are used to perform five different floating-point computations. In this section, the functions of the above segments are discussed.

Floating-point Adder Segment (FADD) The floating-point adder segment is used for addition of floating-point numbers and execution of floating-point instructions such as addition, subtraction and comparison. During X1 and X2 stages of the pipeline, the FADD segment is active and executes floating-point instructions based on single-precision, double-precision and extended precision data.

Floating-point Multiplier Segment (FMUL) The floating-point multiplier segment executes floating-point multiplication in single-precision, double-precision and extended precision modes.

Floating-point Divider Segment (FDIV) This segment performs the floating-point division and executes square-root instructions.

Floating-point Exponent Segment (FEXP) The floating-point exponent segment calculates the floating-point exponent. This segment communicates with all other floating-point segments for proper adjustment of mantissa and exponent fields in the final stage of a floating-point computation.

Floating-point Runder Segment (FRD) After the floating-point addition or division operations, it is required to round off the computed results before write back to the floating-point registers. The floating-point runder segment is used to perform the round-off operation before write-back stage.

12.2.4 Floating-Point Exceptions

There are six possible floating-point exceptions in Pentium processors during integer arithmetic computations. The six different floating-point exceptions are divide by zero, overflow, underflow, denormal operand and invalid operation. The divide-by-zero exception, invalid operation exception and denormal operand exception may be detected before the actual floating-point computation.

The Safe Instruction Recognition (SIR) mechanism is used in a Pentium processor to determine that any floating-point operation can be executed without creating any exception. Whenever an instruction can be executed safely without any exception, the SIR mechanism can be used to allow the instruction for execution. When a floating-point instruction is not safe, the pipeline halts the instruction for three cycles and the exception is generated.

12.2.5 Instruction Pairing

Initially an instruction is loaded into the U pipeline. After that the next instruction will be loaded into V and it must be part of a pair. As per the Pentium processor's pairing rules, if it is not part of a pair then it cannot be loaded into the V pipeline. Then the instruction has to wait till the next slot is available in the U-pipeline. Usually, the instruction decoding and pairing decisions are done in hardware.

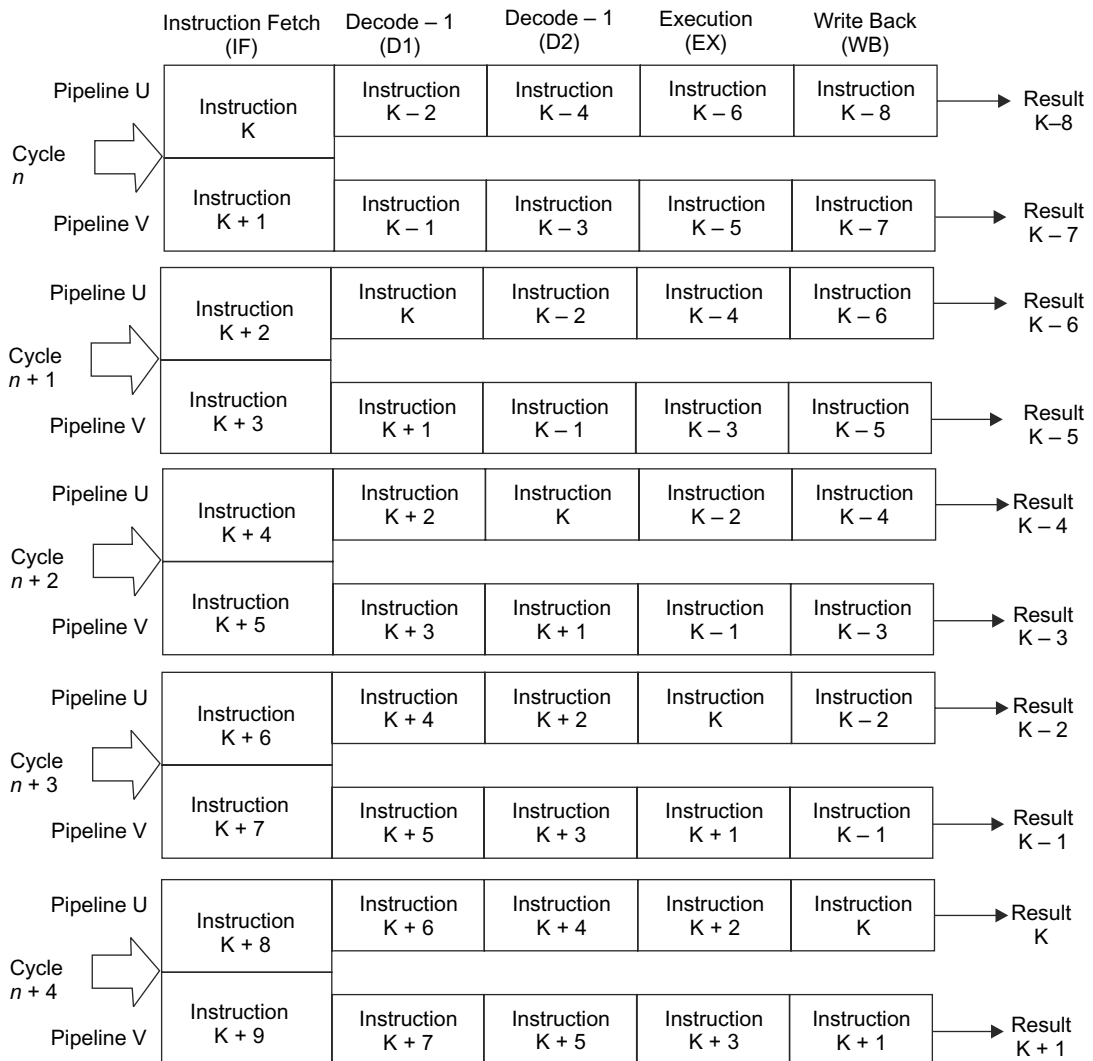


Fig. 12.6 Pairing of instructions

The Pentium pipeline must be transparent to programmers. Whenever the compiler is aware of the Pentium pipeline strategy then instruction throughput can be improved. Figure 12.6 shows the instruction pairing in a Pentium processor. Figure 12.7 shows the example where an instruction cannot be paired.

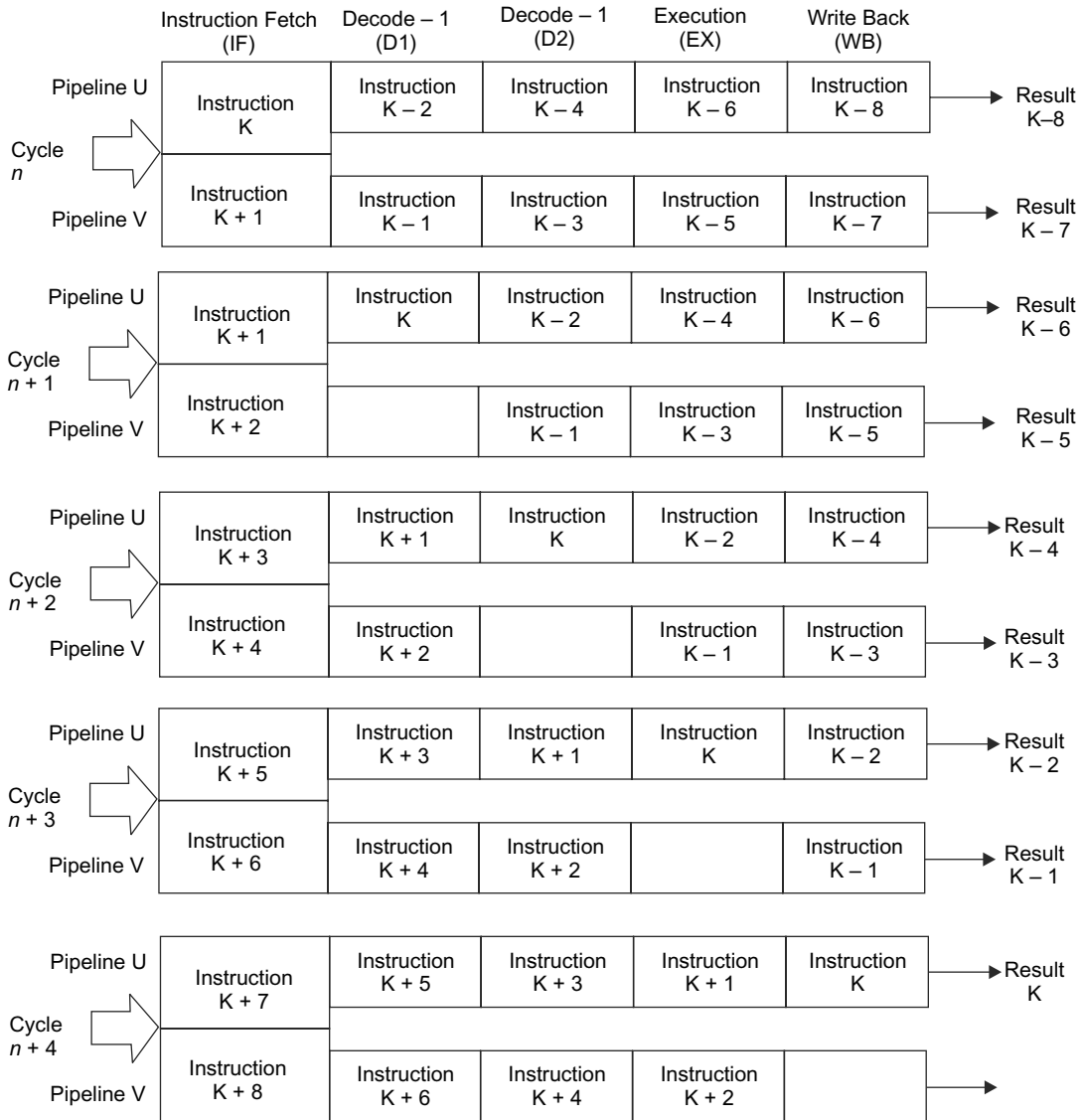


Fig. 12.7 An instruction without pairing

12.2.6 Pentium Register Set

Figures 12.8(a) and (b) show the register set of a Pentium processor. The Pentium has the same register sets as the 80386 processor, but it is clear from Fig 12.8(b) that two new registers CR4 and TR12 are added in the register set of Pentium processor.

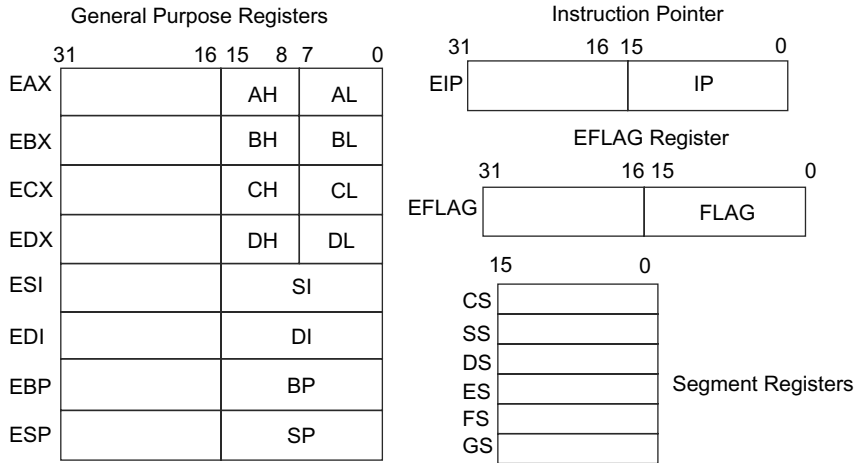


Fig. 12.8(a) Registers of Pentium processor

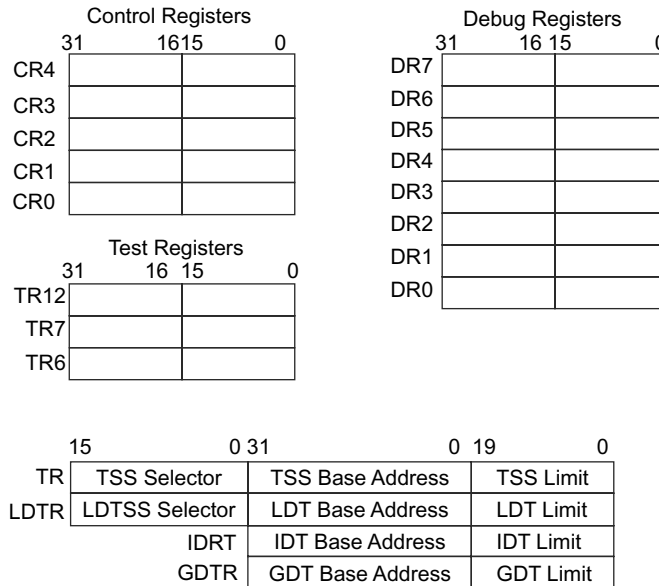


Fig. 12.8(b) Registers of Pentium processor

The control register CR4 controls the Pentium processors extensions for virtual-8086 mode operation. The CR4 register is also used for debugger support and it is used to support up to 4 Mbyte pages. The test control register TR12 enables the selective activation of new features of Pentium processors such as branch prediction, and superscalar operation, etc.

Three new flags are also added in the EFLAGS register of Pentium processor. Two flags are used to support virtual 8086 mode operation and the third flag indicates if the processor supports the CPU ID instruction. When the processor can set and clear the ID flag, it can execute the CPUID instruction.

12.3 PENTIUM OPERATING MODES

The Pentium processor architecture supports three operating modes such as *protected mode*, *real-address mode* and *System Management Mode (SMM)* and one ‘quasi-operating mode’ or *virtual-8086 mode*. In this section, all operating modes are explained in detail.

Protected Mode The protected mode is the local operating mode of the processor. In this mode, all instructions and architectural features are available; the processor is able to provide the highest performance capability.

Real-address Mode The real-address operating mode provides the programming environment of the 8086 processor incorporating the ability to switch to the protected mode or system management mode.

System Management Mode (SMM) The system management mode of the Pentium processor provides an operating system with a transparent mechanism for implementing power management. When an external system interrupt pin (SMI#) is activated, a System Management Interrupt (SMI) is generated and the processor has to be entered in the system management mode. In this mode, the processor switches to a separate address space while saving the context of the currently running program or task. Then the system management mode’s specific code can be executed transparently. Upon returning from SMM, the processor can be back to the real-address-mode state, or protected-mode state or virtual 8086 mode state from the system management mode by using RESET or RSM signal.

Virtual-8086 Mode When the processor operates in protected mode, it can support a quasi-operating mode known as *virtual 8086 mode*. This mode allows the processor to execute 8086 software in a protected as well as multitasking environment.

Figure 12.9 shows how the processor changes the operating modes. Initially, the Pentium processor enters the real-address mode through a power-up or a reset operation. After that, the PE flag in the control register CR0 controls whether the processor is in real-address or protected mode. The switching between real-address mode and protected mode requires some initialization before the mode is changed. When PE = 1, the processor operating mode changes from real address mode to protected mode.

The VM flag in the EFLAGS register decides whether the processor will be operated in protected mode or virtual 8086 mode. The transitions from protected mode to virtual 8086 mode are generally carried out as part of a task switch or a return from an interrupt or exception handler. When VM = 1, the processor-operating mode changes from protected mode to virtual 8086 mode.

12.3.1 Real Mode

The Pentium processor can support real mode for backward compatibility with the 8086

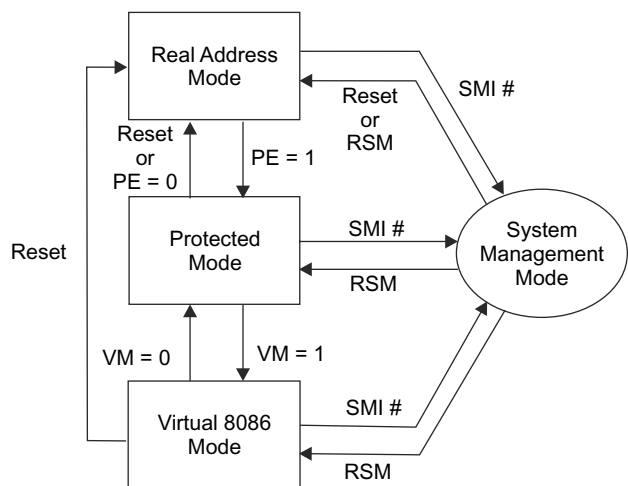


Fig. 12.9 Operating modes of Pentium processor

microprocessor. In real mode, the processor computes the physical addresses from the content of CS and IP registers just like the 8086. But the difference is that while the 8086 had a 20-bit address, the Pentium processors have a 32-bit address. Therefore, in real mode, the IA processors can address over 1 MB with a 21-bit address bus. Hence in real mode, the only extra memory that is available from 100000H to 10FFEFH cannot be used in full 21-bit address. Figure 12.10 shows the real-mode operation of 32-bit IA processors. The physical address computation is given below when the content of segment register is FFFFH and offset is equal to FFFFH.

$$\begin{array}{r}
 \text{Content of segment register} \times 2^4 = \text{FFFFH} \times 2^4 = \text{FFFF0 H} \\
 + \text{Offset} = \text{FFFF H} \\
 \hline
 \text{Physical address} = \text{10FFEF H}
 \end{array}$$

In case of 8086, the leading '1' will be lost as a carry and the address that appears on the 20-bit address bus is 0FFEFH. This is a case of wrap around. In the 32-bit IA/Pentium processors, the address that appears is 10FFEFH. Consequently, in real mode, Pentium processor can access addresses greater than 1 MB causing a pseudo-protection exception and software interrupt.

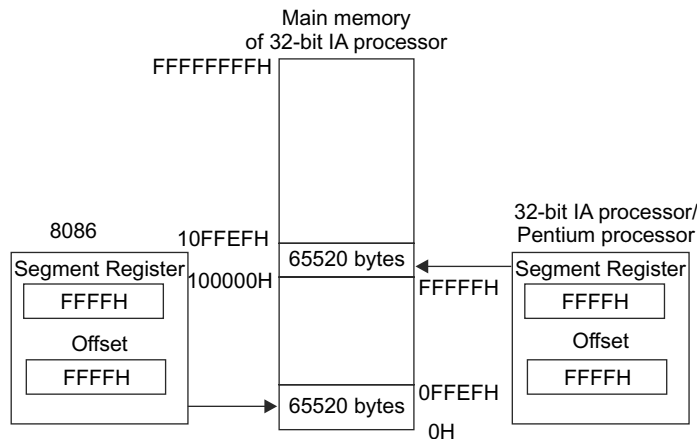


Fig. 12.10 Real mode of 32-bit IA processors and 8086 processor

12.3.2 Protected Mode

The protected-mode operation of the Pentium processor protects different tasks in multitasking operating systems from invalid accesses. The processor hardware checks all accesses of a program to code and data and provides access rights based on four different privilege levels. Task switching is very fast in protected mode.

Memory Management of Pentium During protected-mode operation of the Pentium processor, memory management is done in two different ways, namely, *segmentation* and *paging*.

✓ **Segmentation** Segmentation is used to isolate individual code, data, and stack modules so that multiple programs can run on the same processor without interfering with other programs.

✓ **Paging** Paging is one of the memory management techniques which allows the processor to address a range of virtual memory that is greater than the physical memory that can be addressed using the processor's address bus alone. This is done by swapping pages in and out of the main memory and on and off the disk.

In protected mode, segmentation cannot be disabled but the use of paging is optional.

Actually, segmentation divides the processor's addressable memory space into small protected address spaces called *segments*. This address space is also known as the *linear address space*. Usually, segments are used to hold the code, data, and stack for a program or it can also hold system data. Whenever more than one program is executed on a processor, each program must be assigned its own set of segments. Then the processor enforces the boundaries between these segments so that one program does not interfere with the execution of other programs.

All segments must exist in the processor's linear address space. The *logical address* is used to locate a byte in a particular segment. The logical address is also called a far pointer. Figure 12.11 shows the memory management of a Pentium processor in protected mode. It is clear from Fig. 12.11 that the logical address consists of a segment selector and an offset.

Each segment has a unique segment selector and it provides an offset into the Global Descriptor Table (GDT) to a data structure called a *segment descriptor*. The segment descriptor is used to specify the *size* of the segment, the *access rights and privilege level* for the segment, the *segment type*, and the location of the first byte of the segment in the linear address space which is known as the *base address of the segment*. The offset part of the logical address or far pointer must be added to the base address of the segment and generates an address to locate a byte within the segment. In this way, the addition of base address and the offset determines a *linear address* in the Pentium processor's linear address space as depicted in Fig. 12.11.

When the paging technique is not used in memory management, the linear address space of the processor is mapped directly into the physical address space of the processor. Then the physical address space can be described as the range of addresses that the Pentium processor can generate on its address bus.

When the paging technique is used, IA-32 bit Pentium processors have a linear address space. A linear address is stored in the linear address space. Actually, the linear address consists of page directory, page table

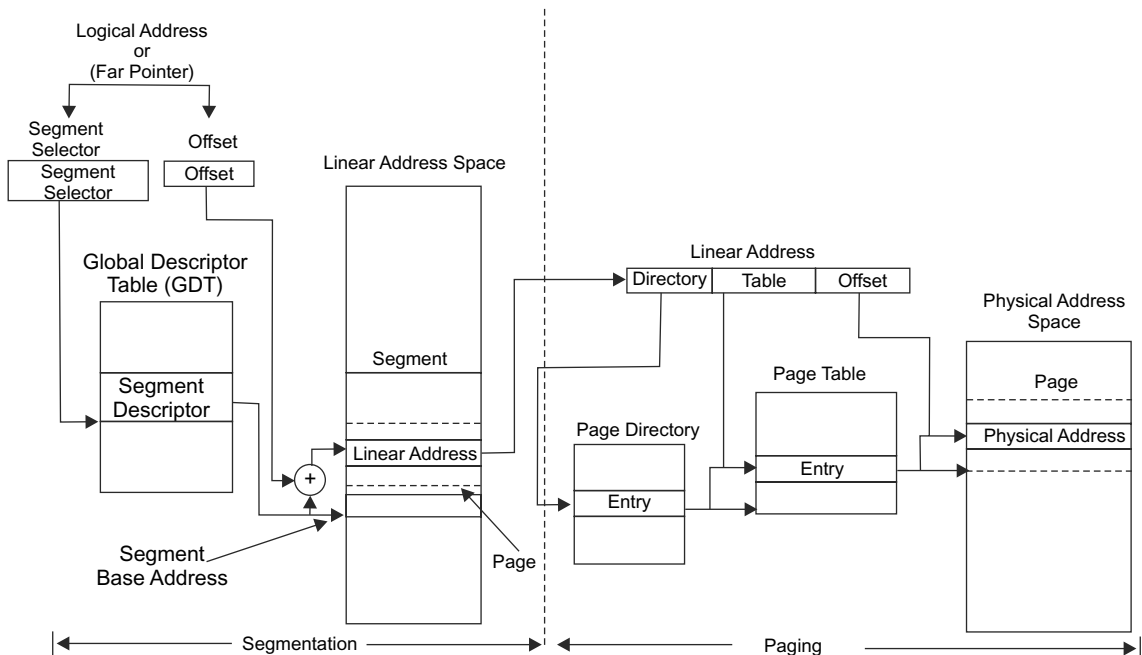


Fig. 12.11 Memory management of Pentium processor

and offset. The physical address is computed by using the contents of page directory, page table and offset to locate a memory location in the physical memory as depicted in Fig. 12.11. In fact, *paging* supports a ‘virtual memory’ environment where a large linear address space is replicated with a small size of physical memory, which is either RAM or ROM or disk storage. With application of paging, each segment must be divided into pages and each page size is about 4 Kbytes, which can be stored either in physical memory or on the disk.

12.4 SEGMENTATION

When segmentation is used in memory management of the Pentium processor, the processor’s linear address space is broken up into a set of segments. Each segment must be specified by a segment descriptor, which defines the *base address* of the segment, the *size or limit* of the segment and its *access rights*. Figure 12.12 shows the simplified representation of the segment descriptor and Fig. 12.13 shows how the segment descriptor is used to define a segment.

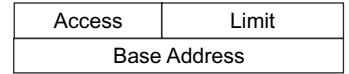


Fig. 12.12 Representation of segment descriptor

Using different settings of the segmentation registers, the processor’s linear address space can be organized into three different memory models such *basic flat model*, *protected flat model*, and *multisegment model*. Paging can be operated with any of the segmentation models. In this section, all three modes are discussed elaborately.

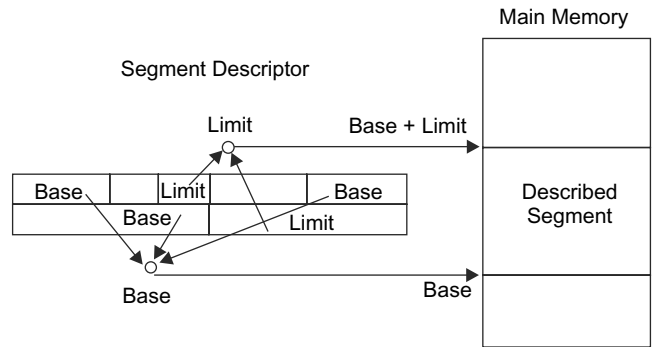


Fig. 12.13 The segment descriptor is used to define a segment

12.4.1 Basic Flat Model

The most simple memory model for a Pentium system is the *basic flat model*. In this system model, the operating system and application programs use a continuous, unsegmented address space. Usually, the basic flat model hides the segmentation mechanism from both the system designer as well as the application programmer. The segmentation cannot be disabled. To use the segmentation, proper setting up of the segmentation registers is required. For implementation of basic flat memory model, at least two segment descriptors are required, one for a code segment and the other for a data segment. Subsequently, both code and data segments can be used to map the entire linear address spaces as shown in Fig. 12.14. The entire linear address spaces have the same base address value of 00000000H and the segment limit of FFFFFFFFH or 4 Gbytes.

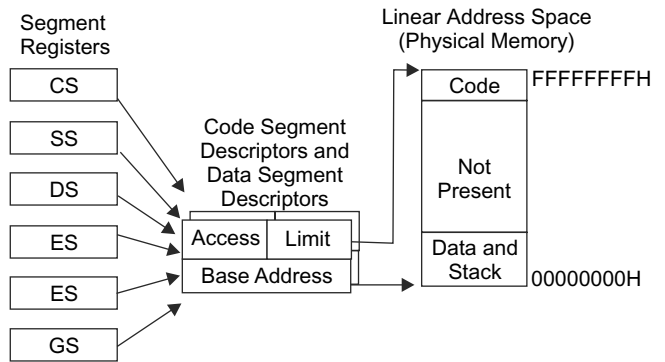


Fig. 12.14 Basic flat model

12.4.2 Protected Flat Model

Figure 12.15 shows the protected flat memory model for a system. The protected flat model is just like the basic flat model, except the segment limits are set to include only the range of addresses for which physical memory actually exists. The general-protection exception, \overline{GP} (#GP) is generated when any attempt has been taken to access non-existent memory. This model provides a minimum level of hardware protection against program bugs. If the protected flat model is combined with the paging mechanism, a higher level of protection can be achieved. Usually, Network Technology (NT) uses this technique to get a high level of protection.

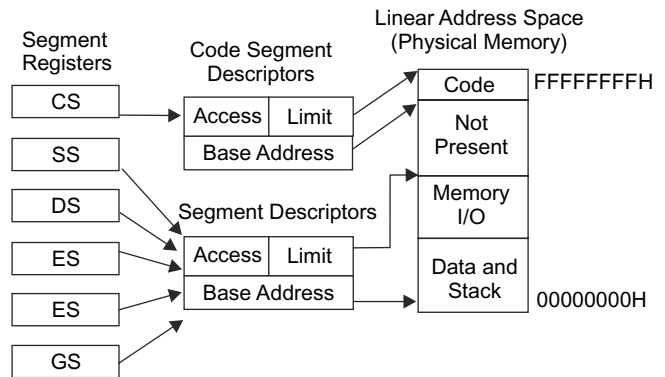


Fig. 12.15 Protected flat model

12.4.3 Multisegment Model

Figure 12.16 shows the multisegment memory model for a system. The multi-segment model has the segmentation capabilities to provide hardware-enforced protection of code, data structures, and programs and tasks. Each program (or task) should have its own table of segment descriptors and its own segments. The segments will be completely private to their corresponding assigned programs. Access to all segments of the system and the execution environments of all individual programs running on the system are hardware controlled.

The access check operations are used to protect against referencing an address outside the limit of a segment, and also against performing disallowed operations in certain segments. Usually, code segments are read-only segments. As a result, hardware can prevent writes into code segments. The access rights information can also be used to set up protection levels. Actually, the protection levels are used to protect operating-system procedures from any unauthorized access by application programs.

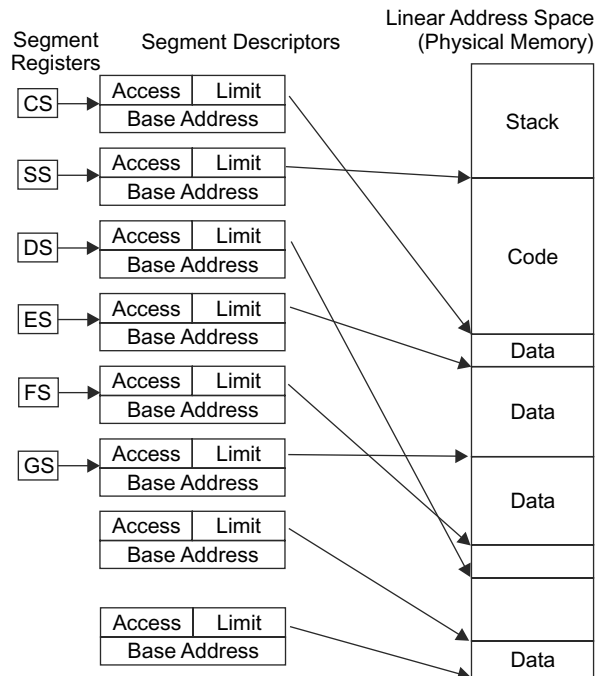


Fig. 12.16 Multi-segment model

12.5 PHYSICAL, LINEAR AND LOGICAL ADDRESS

12.5.1 Physical Address

In protected mode, the Intel Architecture (IA) 32-bit processors/Pentium processors provide a usual physical address space of 2^{32} bytes or 4 Gbytes. Therefore, the processor can address up to 4 Gbytes memory locations

through its address bus. This 4 Gbytes address space is a basic flat model of memory and these address spaces are unsegmented. The address range or memory map of 4 Gbytes address space is from 00000000H to FFFFFFFFH.

The memory mapping divides the 4 Gbytes physical memory into different segments and pages. When paging technique is used for memory management, the paging unit takes the output of the segmentation unit, which is called the *linear address*, and subsequently converts the linear address into a physical address. If paging is not in use, the linear address simply maps directly onto the physical address.

12.5.2 Logical and Linear Addresses

During protected mode, the segmentation unit generates a linear address from a logical address. Each byte which is stored in the processor's address space, can be accessed with a logical address. The logical address consists of a 16-bit segment selector and a 32-bit offset as depicted in Fig. 12.17. The segment selector is used to identify the segment descriptor which provides the base address of the segment. The offset specifies the location of the byte in the segment relative to the base address of the segment. The segment descriptor has a limit field. When the linear address is outside the size of the segment, an exception will be generated. Hence, the segments will be protected from invalid accesses.

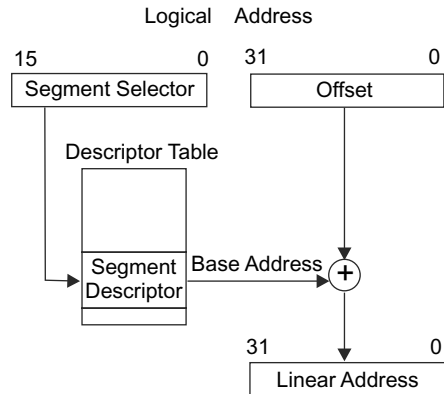


Fig. 12.17 Logical address and linear address

A linear address is a 32-bit address in the processor's linear address space. Just like the physical address space, the linear address space is also a flat memory model and it is unsegmented. The linear address space consists of all the segments and system tables which are used to define a system. The linear address space is 2^{32} bytes or 4 Gbytes of address spaces. The address range of linear address spaces is from 00000000H to FFFFFFFFH. The IA-32 bit Pentium processors convert each logical address into a linear address. There are three steps to translate a logical address into a physical address.

Step 1 Find the index field from the segment selector and use the index field to locate the segment descriptor for the segment in the Global Descriptor Table (GDT) or Local Descriptor Table (LDT). This step is required in the processor whenever a new segment selector is loaded into a segment register.

Step 2 After that, test the access and limit fields of the descriptor to make sure that the segment is accessible and the offset is within the limits of the segment.

Step 3 The base address of the segment will be obtained from the segment descriptor. Then the base address of the segment will be added to the offset to determine a linear address.

When the paging is not used in the system, the processor maps the linear address directly to a physical address. Therefore, the linear address directly placed on the processor's address bus when paging is not used. If the paging is incorporated in the linear address space, a second level of address translation is used to translate the linear address into a physical address. Figure 12.18 shows the physical address generation from logical and linear addresses.

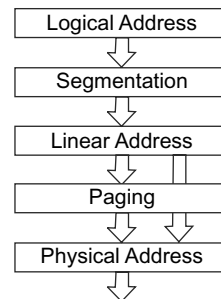


Fig. 12.18 Physical-address generation

12.5.3 Segment Registers or Segment Selectors

In protected mode, the segment registers or segment selectors consist of index field, table indicator and RPL (Requested Privilege Level) as depicted in Fig. 12.19. The segment registers are used as segment selectors rather than segment base addresses.

The *index* field (bit 15 to bit 3) is used to locate into the current table of segment descriptors. IA-32 Pentium processors can hold segment descriptors either in a Global Descriptor Table (GDT) or in a Local Descriptor Tables (LDT).

The *Table Indicator* (TI) field or bit 2 of the segment selector decides which table to be used. When TI = 0, the Global Descriptor Table (GDT) is selected. If TI = 1, the Local Descriptor Table (LDT) is selected. Actually, GDT and LDT are in the processor's linear address space. The LDTR (Local Descriptor Table Register) and GDTR (Global Descriptor Table Register) hold the base addresses for the local and global descriptor tables respectively.

The *Requested Privilege Level* (RPL) bits '1' and '0' are used to find out the privilege level of a program which must be accessed by the segment. These bits are called the *current privilege level* and these bits are used to define the privilege level of the currently active program. There are four different privilege levels (PL) from 0 to 3. The privilege level '0' has the highest priority and the privilege level '3' has the lowest priority.

Application programs with lower privilege levels can only access a segment; whereas programs with a higher privilege level, by using special gates, can access to higher privilege segments. The Operating System (OS) or kernel has the highest privilege level as PL = 0 and application programs have the lowest privilege level as PL = 3. For example, the instruction LGDT (Load Global Descriptor Table) can only execute when the current program has a privilege level '0'.

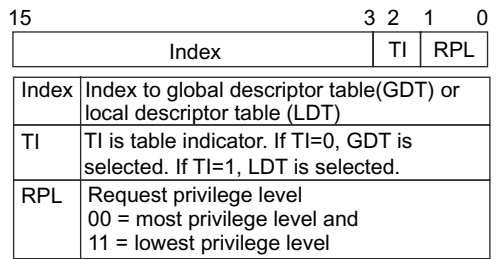


Fig. 12.19 Format of segment registers or segment selectors

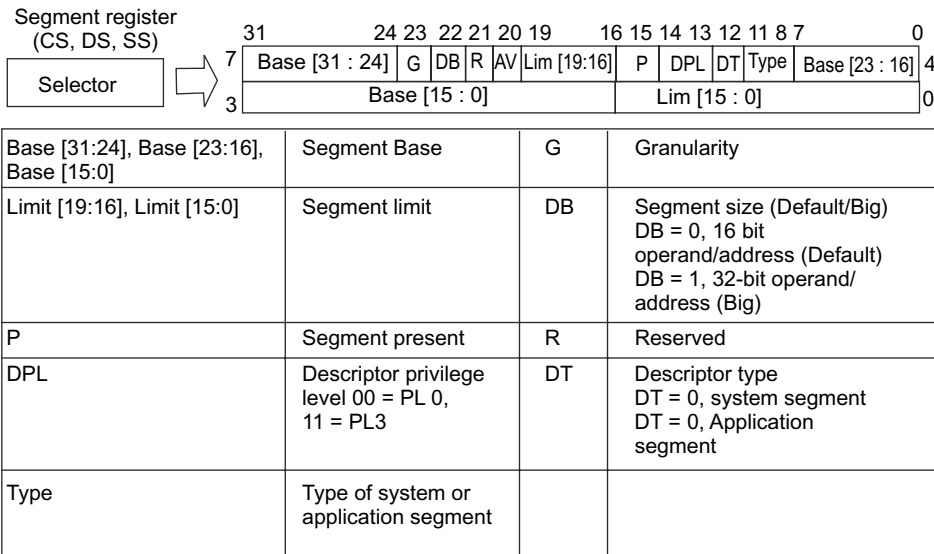


Fig. 12.20 Segment descriptors

12.5.4 Global and Local Descriptor Tables and Cache Registers

The global descriptor table (GDT) is a list in memory which describes the segment sizes and addresses in memory in terms of segment descriptors. Each segment descriptor has 8 bytes as shown in Fig. 12.20. The global descriptor table holds segments available to all tasks, but the local descriptor table (LDT) is only available to the currently active task. The segment descriptor provides the 32-bits segment base address (Base [31:24], Base [23:16], and Base [15:0]) and 20-bits limit (Limit [19:16] and Limit [15:0]) for the size of the segment. The function of other fields of the segment descriptor are given in Fig. 12.21. The 20-bits of the limit field can be used to represent either a memory size of up to 1 MByte or up to 4 GByte, depending on the granularity (G) bit. When G = 0, byte granularity means that the 20 bits represent 1 Mbyte. If G = 1, the granularity is 4K in size and it represents 20 bits × 4K = 4 Gbyte. The DB bit is used to make instructions as either 32-bit instructions or 16-bit instructions. For example, a 32-bit instruction is *mov eax, mem32* when DB = 0 and a 16-bit instruction is *mov ax, mem32* if DB=1.

Segment Registers	64-bit Segment Descriptor cache Registers		
CS	Access	Base Address	Limit
SS	Access	Base Address	Limit
DS	Access	Base Address	Limit
ES	Access	Base Address	Limit
FS	Access	Base Address	Limit
GS	Access	Base Address	Limit

Fig. 12.21 64-bit segment descriptor cache registers

During accessing memory, if the processor does not want to use the descriptor tables frequently then the processor loads the base address and limit for a given selector into a cache register which is associated with each selector. Figure 12.21 shows the 64-bit segment descriptor cache registers. Whenever the contents of a segment register change, the local descriptor table (LDT) or global descriptor table (GDT) is accessed for the base address and limit. After that, the base addresses and limits are placed in the segment descriptor cache register. Subsequently, the cache register is used for the base addresses and limits.

12.5.6 Interrupts in Protected Mode

In protected mode of IA 32-bit Pentium processors, the interrupt vector table is an Interrupt Descriptor Table (IDT). The interrupt descriptor table works in the same way to the segment descriptors. The IDT is able to hold up to 256 interrupt levels. However, each interrupt level can be accessed through an interrupt gate rather than an interrupt vector.

Usually, the interrupt type number is located at the interrupt descriptor table. The IDT may be located anywhere in the memory system where the first 1K hold the interrupt vectors as depicted in Fig. 12.22. The content of IDT is interrupt or trap gate. The format of an interrupt or gate descriptor is shown in Fig. 12.23. The segment selector is used to locate

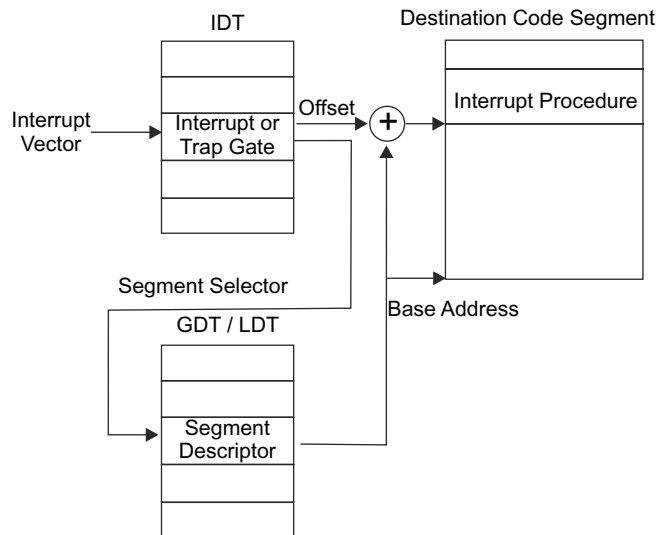


Fig. 12.22 Interrupts in protected mode

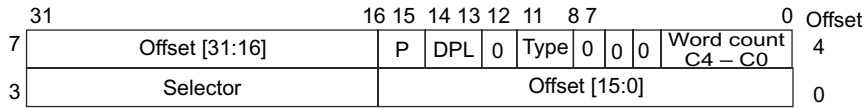


Fig. 12.23 Interrupt or gate descriptor format

segment descriptor on the global descriptor table or the local descriptor table and generates base address. The base address and offset are combined to find interrupt procedure at the destination code segment.

12.5.7 Task State Segment (TSS)

The task state segment descriptor consists of information about the location, size and privilege level of the task state segment. Usually, the task state segment is described by the TSS descriptor and it does not contain code or data. Actually, the TSS holds the state of the task and linkage so that tasks may be nested during operation. Figure 12.24 shows the task state segment.

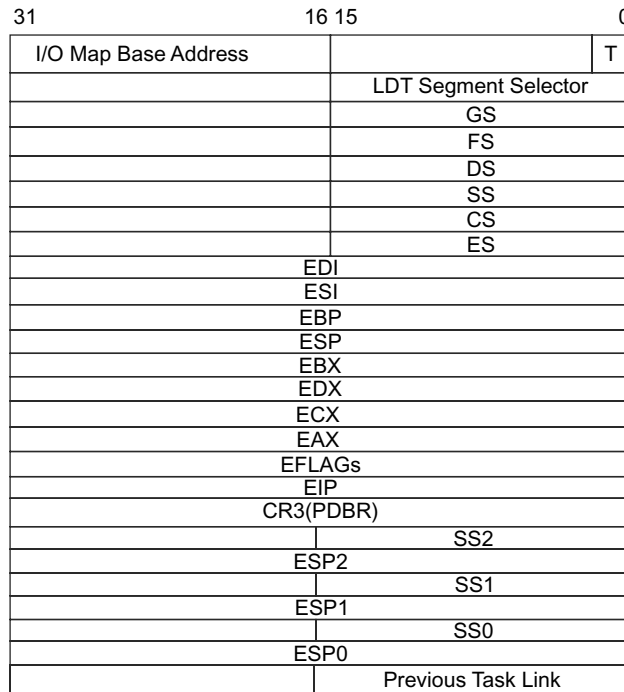


Fig. 12.24 Task state segment

12.6 VIRTUAL 8086 MODE

The *virtual 8086 mode* is a special operating mode of the Pentium processor and it is a subset of protected mode. In this mode, two 8086 application programs can be executed at the same time. During virtual 8086 mode operation, the segment registers are used in the similar way as in real mode. Consequently, the segment and an offset address are used to address 1 Mbyte memory locations starting from 00000H to FFFFFH. After incorporating paging, any program can use addresses below 1 Mbyte memory space, but the Pentium

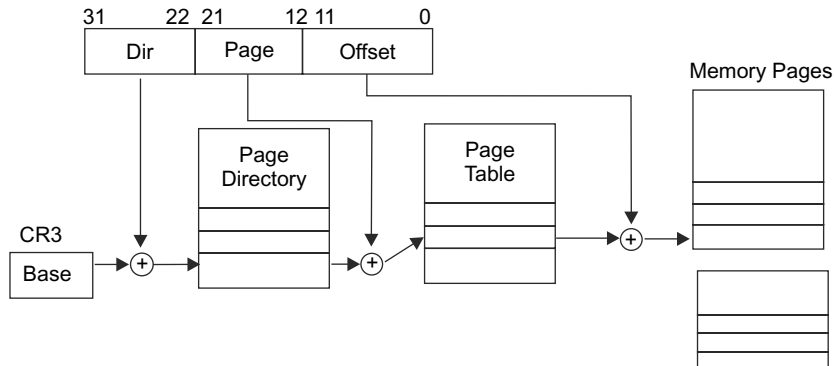


Fig. 12.26 The page translation process

Page table is an array of 32-bit page-table entries restricted in a 4 Kbyte page. About 1024 page-table entries can be held in a page table. The page tables cannot be used for 2 Mbyte or 4 Mbyte pages. The 2 Mbyte or 4 MByte page sizes are mapped directly from one or more page-directory entries.

Usually the page directory is about $1K \times 4$ bytes or 4 Kbytes. Each page table is also 4 Kbytes. As there are 1024 (1K) possible page tables, a fully paged Pentium system requires 4 Mbytes + 4K to express the paging. But most of the operating systems do not use a fully paged memory system. Pentium processors support the 4M paging. When the Pentium processor supports 4M paging, there is no need of page tables and the page directory can be used to addresses a 4 Mbyte memory page. Figure 12.27 shows the 4M memory page supported by Pentium processors.

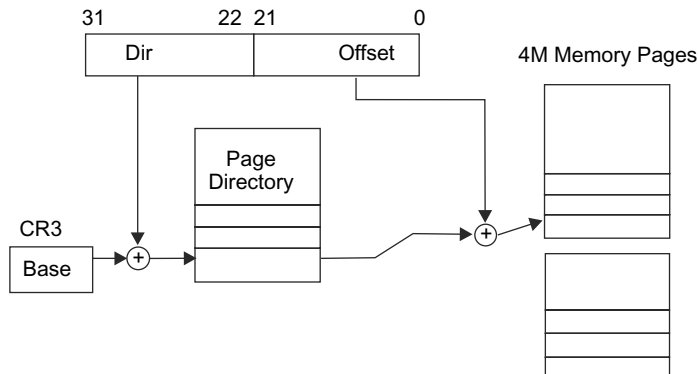


Fig. 12.27 4M memory page is supported by Pentium processors.

12.6.2 Translation Look-aside Buffers

It is clear from the paging system of Pentium processor that two memory accesses are required before addressing the specified physical address. This is a huge bottleneck of a paging technique. To improve the paging system, IA-32 Pentium processors use a Translation Look-aside Buffer (TLB), which is a cache of recently used page translations. When a page-table entry is in the TLB, the paging unit uses the TLB to implement the paging system. If a page-table entry is not present in the TLB, it is required to read memory to get the page directory and page-table contents which are needed for the translation. Usually, TLBs are smaller than normal caches. The Pentium processors have separate TLBs for code cache and data cache.

12.7 PIN DESCRIPTION OF PENTIUM PROCESSOR

The Pentium's memory and I/O interfaces are more advanced than the simple models. They are very different from the 8086 bus interfaces. Figure 12.28(a) shows the Pentium processor and the schematic pin diagram of Pentium processor with most signals depicted in Fig. 12.28(b). The pin functions of the Pentium processor are described below:



Fig. 12.28(a) Pentium processor

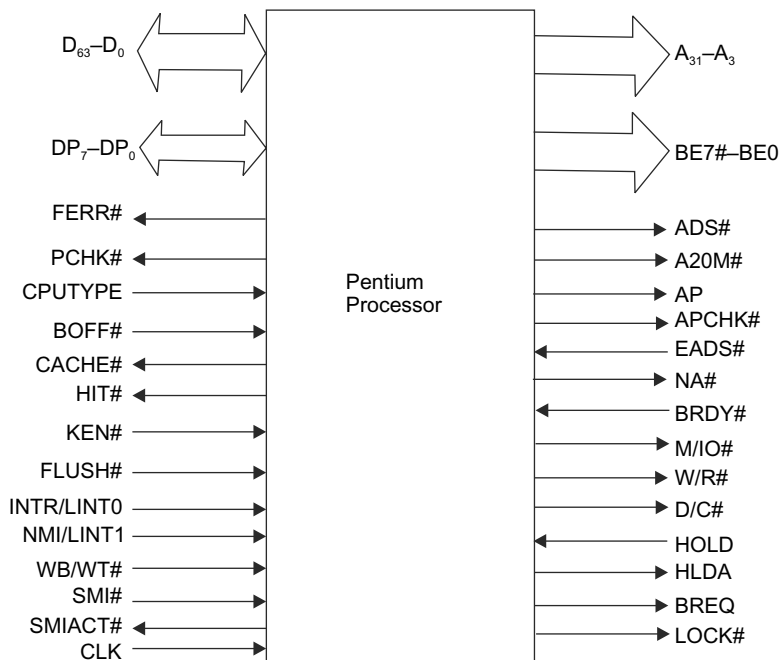


Fig. 12.28(b) Schematic pin diagram of Pentium processor

✓ **$A_{31}-A_3$ (Output)** The address bus ($A_{31}-A_3$) is used to drive an external memory, I/O devices and decoders. Generally, the address buses $A_{31}-A_3$ are used along with the byte enables signals $BE7#-BE0#$.

✓ **ADS#(Output)** The address status signal (ADS#) shows that a new valid bus cycle is driven by the Pentium processor.

- ✓ **A20/M#(Input)** Whenever the *address bit 20 mask* (A20/M#) pin is asserted, the Pentium imitates the address wrap-around of the 8086. The A20/M#signal is only used in real mode.
- ✓ **APCHK# (Output)** When the Pentium processor has detected address parity errors on an inquire or snooping cycle, the *Address Parity Check* (APCHK#) status signal is asserted 2 clock cycles after EADS#.
- ✓ **AP (Input/Output)** The *Address Parity* (AP) signal provides an even parity information for all address cycles of Pentium processor. This signal can also be used along with APCHK# and EADS# for snooping cycles.
- ✓ **BE7#–BE0# (Output)** The *byte enable signals* BE7#–BE0# are used with address buses A_{31} – A_3 to indicate the bytes which will be written to the external memory.
- ✓ **BF1–BF0 (Input)** For the duration of reset, BF1–BF0 pins are sampled and fix the ratio of the external and internal clocks.
- ✓ **BOFF#(Input)** The *Back-Off* (BOFF#) input signal terminates all uncompleted bus cycles. The Pentium floats the pins normally affected by a bus hold, a clock cycle after BOFF#. When BOFF# signal is removed, the Pentium restarts all uncompleted bus cycles in their entirety.
- ✓ **BRDY#(Input)** The *Burst Ready* (BRDY#) input signal is used to indicate an external device which has currently completed the read or write when BRDY#=0. If BRDY#=1, an external device needs more time. Usually this signal is sampled in the T2, T12 and T2P cycles.
- ✓ **BREQ(Output)** The *Bus Request* (BREQ) output signal informs the external system that the Pentium processor has generated a bus request internally.
- ✓ **BUSCHK#(Input)** The *Bus Check* (BUSCHK#) input signal allows the external system so that a bus cycle becomes unsuccessful. Therefore, a machine check exception will be generated depending on the state of CR4.
- ✓ **CACHE#(Output)** When the Pentium processor initiates a cycle's operation, the *cache* (CACHE#) output signal indicates that the current read cycle is internally cacheable. This signal is also used to indicate a burst write back cycle for a write cycle operation.
- ✓ **D₆₃–D₀(Input/Output)** The 64-bit *data bus* of the Pentium processor is D₆₃–D₀. D₆₃–D₅₆ lines are the most significant byte and D₇–D₀ lines are the least significant byte.
- ✓ **DP₇–DP₀** The DP₇–DP₀ lines are called *data parity* bits. Actually, there is one bit for each byte of the data bus. For example, DP₇ is used for D₆₃–D₅₆ and DP₀ is used for D₇–D₀. During a write operation, the Pentium processor generates even parity in the same cycle. But during a read operation, the external system has to drive even parity back into the Pentium.
- ✓ **CLK(Input)** The *clock* (CLK) input gives the fundamental timing for Pentium cycles. All timings signal except the JTAG signals are always referred to the clock input.
- ✓ **CLK(Input)** In a single-processor based system, CPUTYPE is connected to ground. In a dual-processor based system, the primary processor's CPUTYPE will be connected to ground and the secondary processor's CPUTYPE must be connected to +V_{CC}.
- ✓ **PCHK#(Output)** The *Parity Check* (PCHK#) output signal is used to indicate the result of a parity check on a specified data read.
- ✓ **PEN#(Input)** The *Parity Enable* (PEN#) input signal is used with CR4MCE to determine whether a machine cycle exception can be taken as a result of a parity error on a read cycle.

- ✓ **WR#(Dual-Input/Output)** The W/R# signal is used to differentiate the read and write operations. If W/R# =1, this signal represents a write operation. When W/R#=0, this signal represents read operations.
- ✓ **D/C#(Output)** The D/C# output signal is used to represent data or code. If D/C# =1, this signal represents data. When D/C# =0, this signal represents a code/special cycle. Actually, D/C#=0, M/IO#=0, W/R#=1 and the BE7#–BE0# signals are used to identify the exact type of a special cycle.
- ✓ **M/IO#(Dual-Input/Output)** The M/IO# signal is used to distinguish the memory and input/output related operations. If M/IO# =1, this signal represents memory-related operation. When M/IO# =0, this signal represents input/output device related operations.
- ✓ **EADS#(Input)** The EADS# signal determines that a *valid external address* has been driven into the Pentium processor for a snooping or inquire cycle.
- ✓ **FLUSH#(Input)** When the FLUSH# input signal becomes low, the Pentium processor writes back to memory all changed cache lines and invalidates its code and data internal caches.
- ✓ **HIT#(Output)** The hit signal provides the result of a snooping or inquiry cycle. When HIT# is low, it indicates a hit. If HIT# is high, it indicates a miss.
- ✓ **HOLD(Input)** The bus hold request (HOLD) is used for bus arbitration.
- ✓ **HLDA(Output)** The bus hold acknowledges (HLDA) is used for bus arbitration. When an external bus master asserts HOLD, the Pentium processor tri-states its output and input/output lines and also asserts HLDA after completing all outstanding bus cycles. Then external bus master releases HOLD and the Pentium leaves bus hold and release HLDA.
- ✓ **LOCK#(Output)** When the *Lock* (LOCK#) is active low, the Pentium processor does not recognize HOLD request signal.
- ✓ **KEN#(Input)** The *Cache Enable* (KEN#) input signal is used to verify whether the current cycle is cacheable or not cacheable.
- ✓ **NA#(Input)** The Next Address (N/A#) input decides whether any external system is ready to accept a new address even though all the current bus cycles have not completed. This is used to speed up memory access cycles significantly.
- ✓ **INTR/LINT0 (Input)** When INTR input signal is high, it indicates a maskable interrupt request from an external device. If the IE flag is set in the EFLAG register, the processor will accept the interrupt. The LINT0 states local interrupt 0 when the on-chip APIC is used in the Pentium processor.
- ✓ **NMI/LINT1(Input)** NMI stands for nonmaskable interrupt. The LINT1 states local interrupt 1 when the on-chip APIC is used in a Pentium processor.
- ✓ **RESET(Input)** When Reset is high, it forces the Pentium processor to start execution at a known state and all internal caches are invalidated.
- ✓ **SMI#(Input)** If the *System Management Interrupt* (SMI#) input signal is low, the Pentium processor enters the system management mode.
- ✓ **SMIACT#(Output)** When the *System Management Interrupt Active* (SMIACT#) output signal goes low, it indicates that the processor is in System Management Mode (SMM).
- ✓ **WB/WT#(Input)** The *Write-Back/Write-Through* (WB/WT#) input signal is used to define whether a data cache line may be used as write-back (1) or write-through (0) on a line-by-line basis. This signal also

decides whether a cache line is initially in the S or E state in the data cache. Usually, it is used with the PWT bit.

12.8 ADDRESSING MODES OF THE PENTIUM PROCESSOR

The Pentium processors can support the following addressing modes:

- ◆ Register mode
- ◆ Immediate mode
- ◆ Register direct mode
- ◆ Direct mode
- ◆ Base displacement mode
- ◆ Base indexed mode
- ◆ PC relative mode

The effective address can be computed as

Effective address = Base Register + Index register x Scaling factor + displacement

where,

- ◆ base registers are EAX, EBX, ECX, EDX, ESP, and EBP
- ◆ Index register is EDI, and ESI
- ◆ Scaling factor is 1, 2, 4 and 8

For example, the addressing mode looks like

[EBX] [EDI x2] +FF

- ✓ **Register Mode** In this mode, the operand is in a register. For example, MOV EAX, EBX. The content of EBX register is copied to EAX register.
- ✓ **Register Direct Mode** In this mode, the operand is in effective-address-use-register direct mode. For example, MOV EAX, [ESP]. The content of the EBX register is copied to the EAX register.
- ✓ **Immediate Mode** In this mode, the operand is in the instruction. For Example, MOV EAX, 88.
- ✓ **Direct Mode** In direct address mode, the effective address is within the instruction. For example, MOV EAX, address. The content of the effective address will be copied into the register EAX.
- ✓ **Base Displacement Mode** The effective address is the sum of the contents of the register and a constant. For example, MOV EAX, [ESP+4]. The contents of memory location specified by the effective address must be copied into the EAX register.
- ✓ **Base-Indexed Mode** The effective address is the sum of the contents of two registers. For example, MOV EAX, [ESP][ESI]. The contents of ESP and ESI registers are added to generate the effective address of the memory location. The content of the memory location specified by the effective address must be copied into the EAX register.
- ✓ **PC Relative Mode** In this addressing mode, the effective address is computed by the sum of the contents of PC and a constant within the instruction. For example, JMP address. The contents of the program counter are added with an offset which is existing within the instruction. The computed result of addition is placed into the program counter (PC).

12.9 PENTIUM BUS INTERFACING

The Pentium processor has a 64-bit data bus which increases data transfer rates over previous generation processors, and the processor bus is used for connection to a fast L2-cache. The Pentium processor is able to address bytes, words and double words as processor instruction set supports bytes, words and double-word type data. This processor uses the byte enable signals, BE7#–BE0#. Each bus cycle uses address lines A31–A3 to access up to 8 bytes at a time. The byte enable signals BE7#–BE0# is able to address individual bytes as shown in Fig. 12.29.

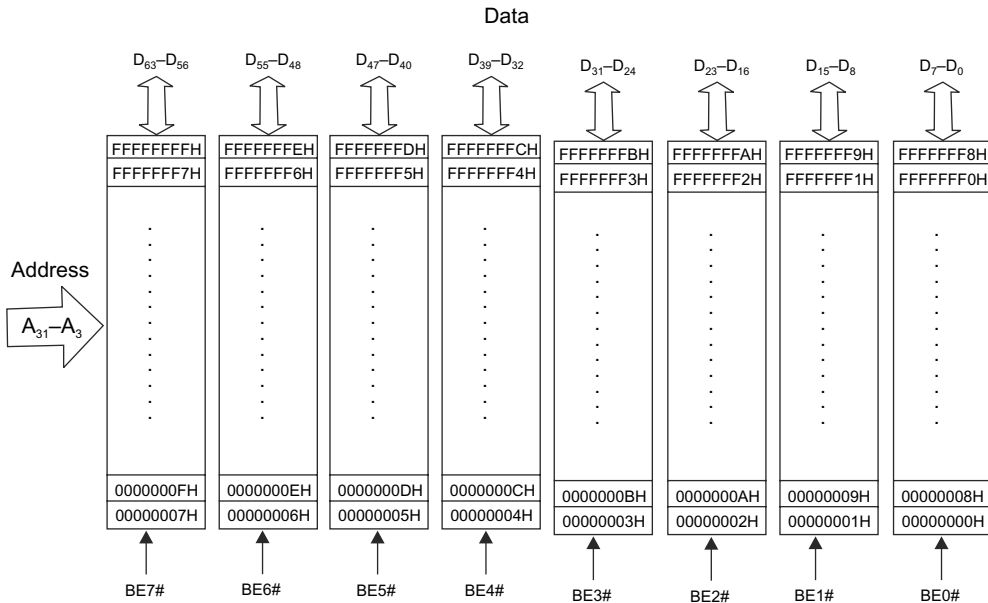


Fig. 12.29 Individual bytes addressed by the Pentium processor

The Pentium performs the memory accesses as a cache line fills or write-backs. The Pentium processor requires single transfer cycles to I/O device addresses and noncacheable memory areas. Usually, the I/O device address space is 32 bytes wide as I/O device accesses do not work through the D-cache. The I/O subsystem generates A₂–A₀ signals from BE7#–BE0# for I/O devices which requires address bits A₂–A₀.

12.9.1 Single Transfer Cycles

In a single transfer read or write cycle, 8 bits, 16 bits, 32 bits or 64 bits of data are transferred to or from the Pentium processor. When the Pentium processor holds CACHE# line at logic level high, it indicates that no line fill operation will be performed. Figure 12.30 shows the single transfer cycle of Pentium processor.

It is clear from Fig. 12.30 that a no-wait single cycle transfer takes at least two CLK cycles. The Pentium processor starts a bus cycle by asserting the Address Status Signal (ADS#) during the first clock pulse (T₁) as depicted in Fig. 12.30. The Address Status (ADS#) output signal determines a valid bus cycle and address will be available on the cycle definition pins and the address bus. If the CACHE# output signal is high, the bus cycle will be a single transfer cycle.

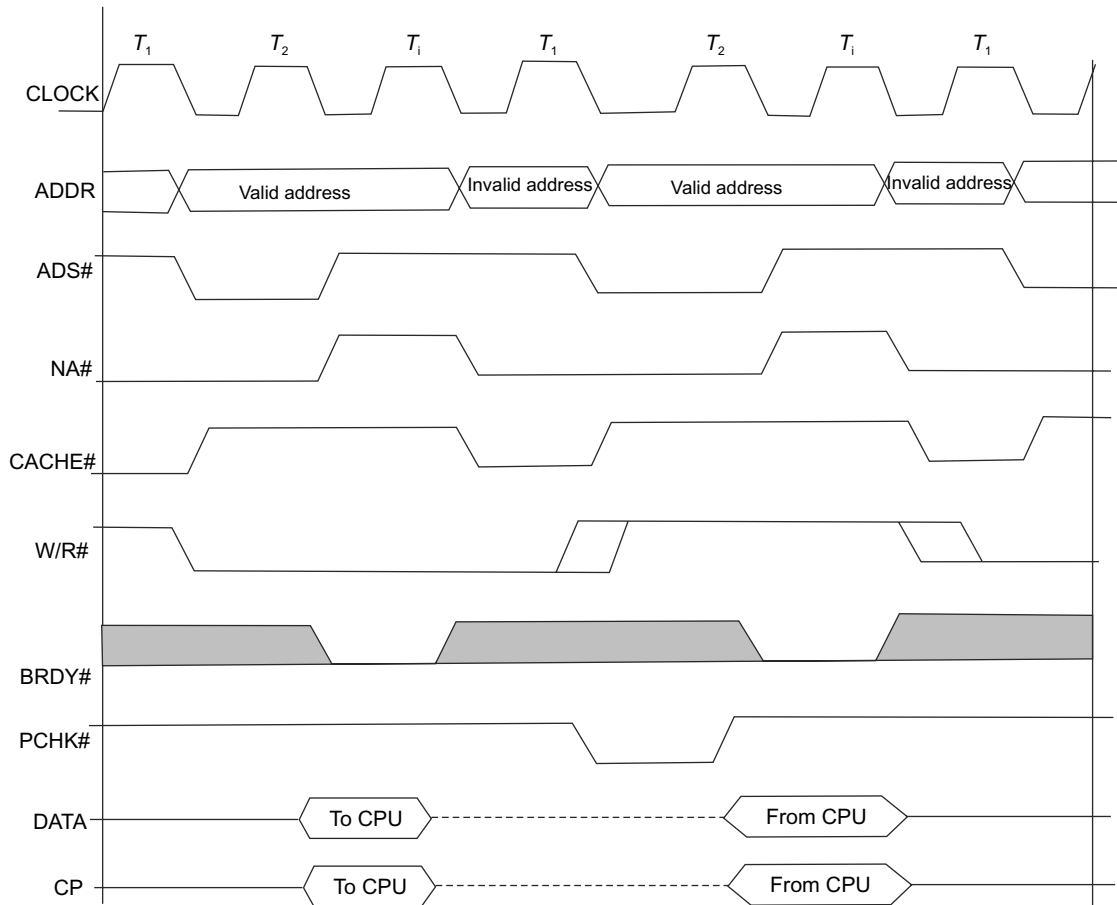


Fig. 12.30 Single transfer cycle of Pentium processor

During the second clock pulse of the bus cycle, **BRDY#** is returned by the external system for a zero wait state transfer. The **BRDY#** signal is used to detect that the external system has valid data on the data pins due to a read operation, or the external system has received valid data due to a write operation. Actually, the Pentium processor samples the **BRDY#** input during second clock pulse and subsequent clock pulses of a bus cycle.

The timing diagram of the parity check (**PCHK#**) output signal and the data parity input are shown in Fig. 12.30. The Pentium processor drives the Data Parity (DP) and returns to the Pentium processor in the same clock as the data. After two clock pulses from **BRDY#** are returned for reads with the results of the parity, the **PCHK#** becomes low.

When the Pentium processor is not ready to drive or receive data, wait states must be added to the bus cycle and the **BRDY#** will not be returned to the processor at the end of the second clock. Figure 12.31 shows a bus cycles with one and two wait states. At the end of the second clock pulse, **BRDY#** must be driven inactive. Any number of wait states can be added to Pentium processor bus cycles when **BRDY#** signal is inactive.

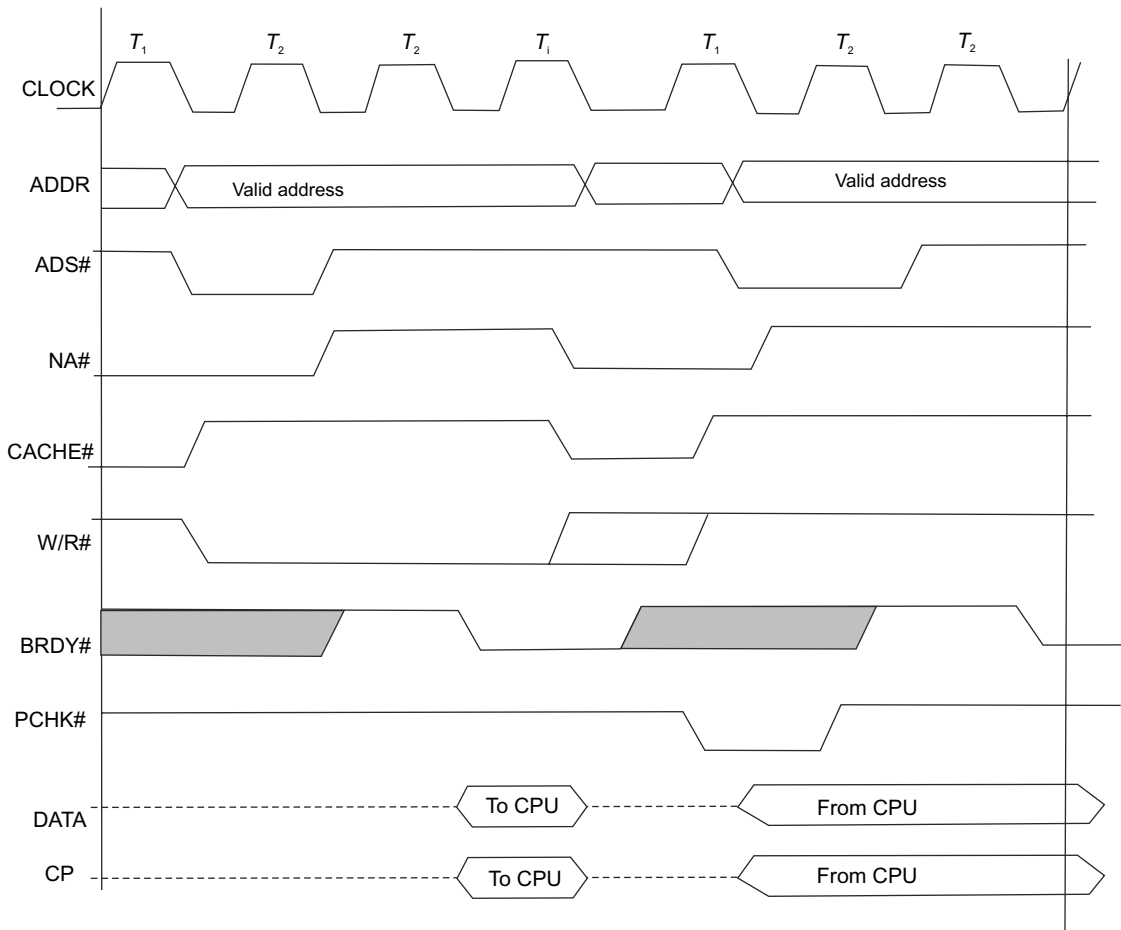


Fig. 12.31 Single transfer cycle of Pentium processor with wait states

12.9.2 Burst Cycles

If there is a requirement to transfer more than a single data using cacheable and write-back bus cycles, the Pentium processors generally use the burst data transfer technique. During the burst transfers, a new data item can be sampled or driven by the Pentium processor in consecutive clock pulses. The 64-bit data bus of Pentium processors represents 8 bytes. Therefore, a burst is able to transfer 32 bytes within four cycles. The data are contiguous and must be aligned to 32-byte boundaries, equivalent to an internal Pentium processor cache line. Figure 12.32 shows the burst cycle of a Pentium processor.

In the burst read cycle, the **CACHE#** signal becomes low to indicate the memory subsystem that Pentium processor wants to transfer the address into the cache. When the **KEN#** signal is returned by the memory subsystem as an active low then the Pentium processor extends the single transfer to a cache line fill to store a complete block of data in the on-chip cache.

The burst cycle is also limited to a 32-byte boundary. Therefore, when the first address of data has been sent out, the other three addresses of the next three data are already fixed. This means that the subsystem can independently calculate the other three burst addresses without decoding any other bus addresses from the

Pentium processor. Hence, the speed of data transfer will be increased considerably. During the burst cycle, the Pentium processor sends the address ADDR and BEX# signal in the first clock cycle and these signals are not changed after the first clock pulse.

The first address which is sent out by the processor will not inevitably lie on a 32-byte boundary, but can be anywhere in the memory. In the first transfer of the data, the KEN# signal identifies that a burst transfer is going to happen. During the burst transfer stage, the first 8 bytes have already moved out. As a result, the next three cycles should come in a fixed sequence. Actually, the sequence is optimized for the 2-way interleaving of DRAM memory and then the subsystem sends data in a defined order without changing addresses of the processor.

The fastest burst cycle possible needs two clock pulses for the first data item to be returned and all succeeding data items will be returned on each clock pulse. Figure 12.32 shows a burst read cycle without wait or pipelining.

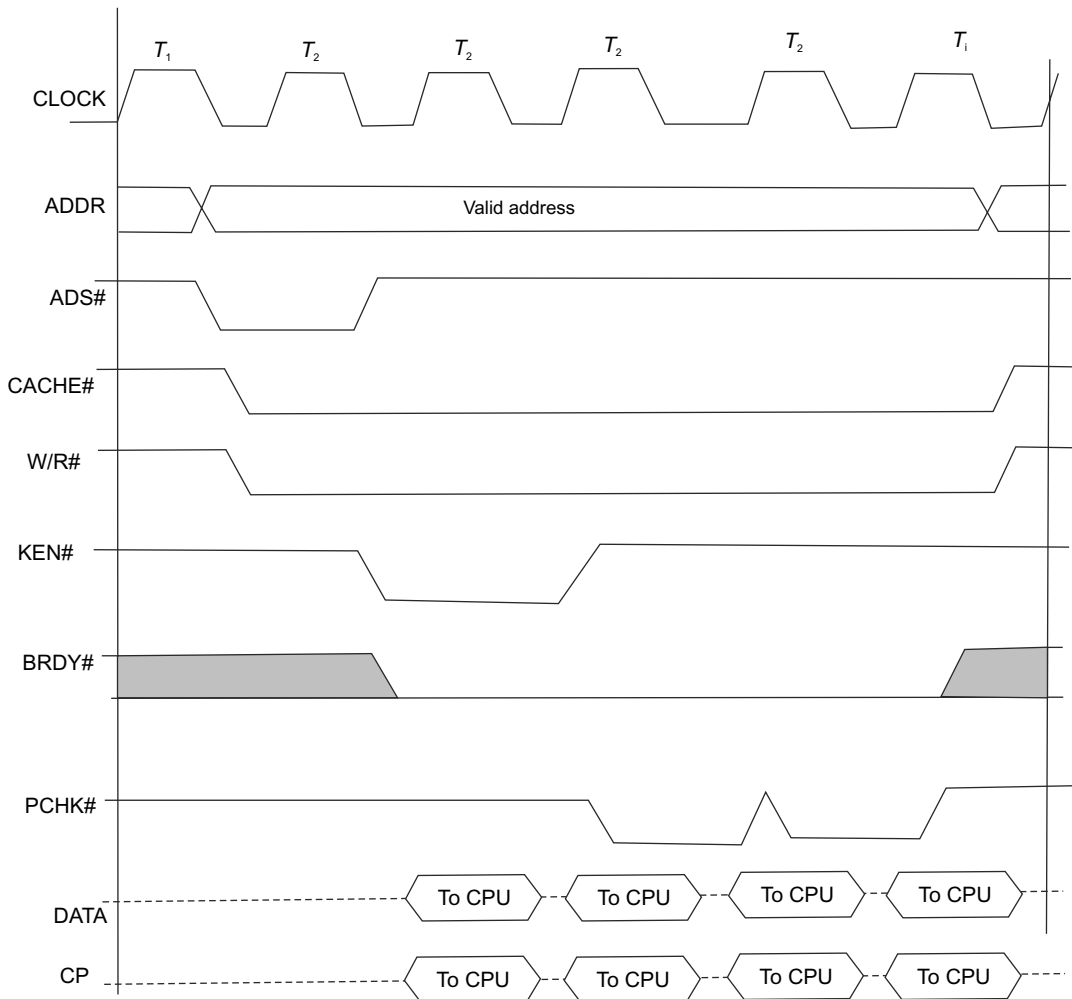


Fig. 12.32 Burst read cycle without waits or pipelining

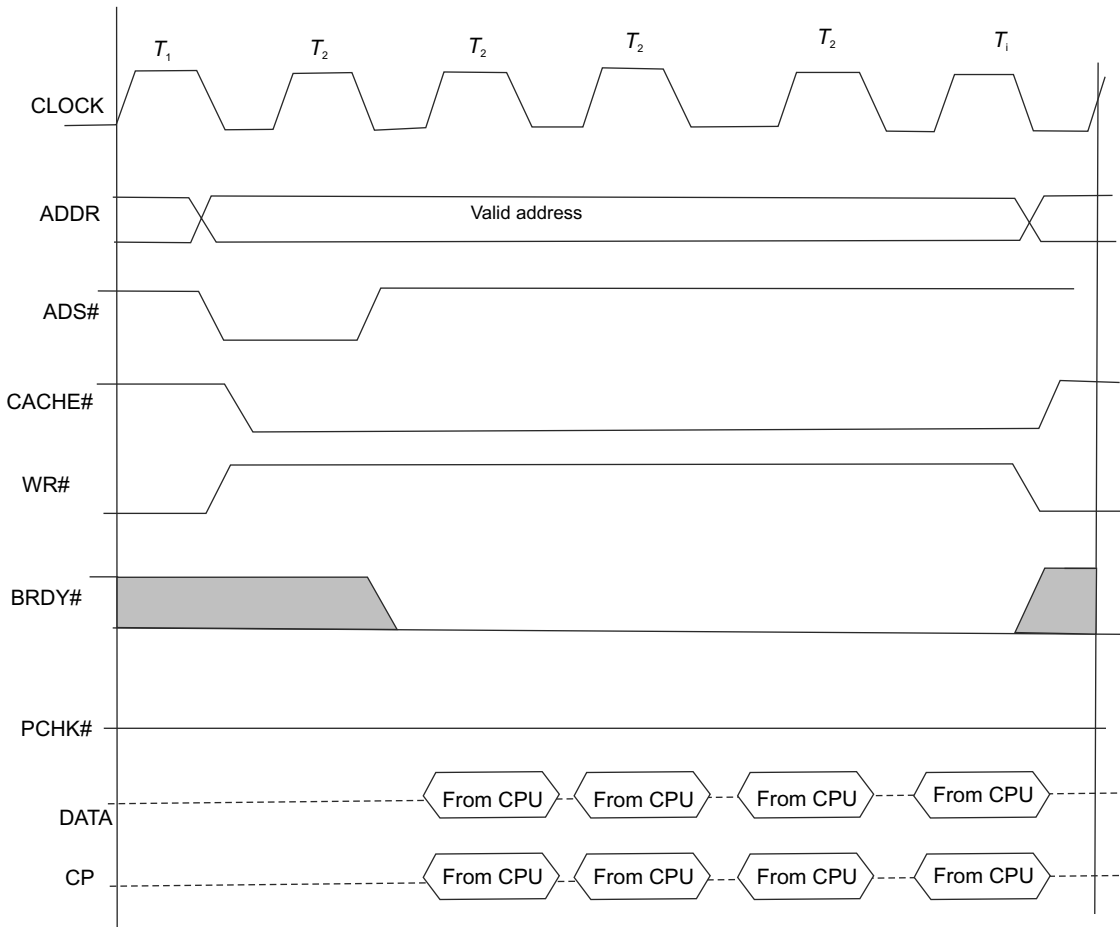


Fig. 12.33 Burst writes cycle without waits or pipelining

Burst write cycles are always write-backs of modified lines in the data cache. The burst writes always follow the sequence $00H \rightarrow 08H \rightarrow 10H \rightarrow 18H$. A burst write is depicted in Fig. 12.33. If there are no wait states, the Pentium processor is able to transfer four bytes in five clocks, called a 2-1-1-1 burst as shown in Fig. 12.33. When a wait state is added in each cycle of data transfer, the burst writes will be known as a 3-2-2-2 burst.

12.9.3 Pentium Address Pipelining

In the Pentium processors, address pipelining can be used to further increase data throughput. For this function, the Pentium uses the $NA\#$ signal. Then the first cycle in a bus cycle can be reduced to one clock pulse. The $NA\#$ input is used to indicate the Pentium processor that it can drive another cycle before the current one is completed.

When the memory subsystem has already decoded an address, it is ready to take the next cache-line fill address for decoding if the memory subsystem asserts the $NA\#$ signals to the Pentium processor. Then the Pentium processor sends the next address and the subsystem starts the decoding operation of the new address although the third data transfer is still going on. Hence two burst cycles can be done as depicted in Fig. 12.34.

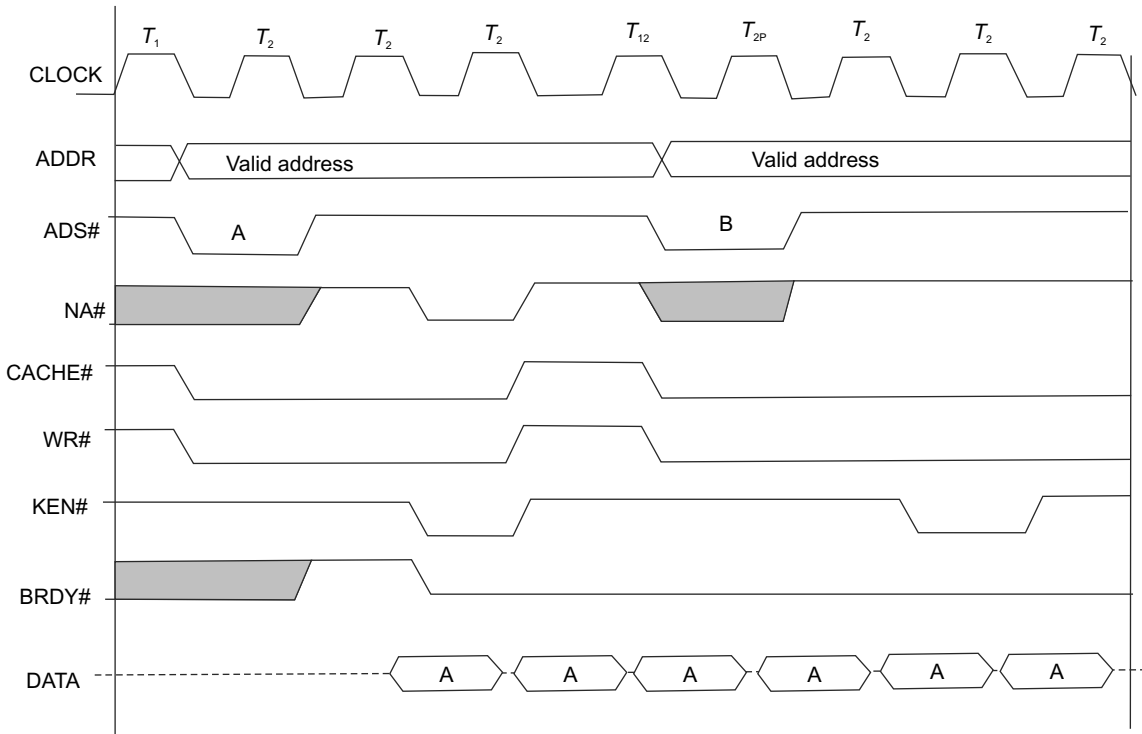


Fig. 12.34 Two burst cycles of Pentium processor using address pipelining

12.9.4 Special Cycles

The Pentium processors use control signals to indicate that a special cycle is in progress. The system control signals are $D/C\#=0$, $M/IO\#=0$ and $W/R\#=1$. Usually, the identity of the special cycle is sent out on the byte enable signals $BE7\#-BE0\#$. Table 12.1 shows the special cycles of a Pentium processor. When the $BRDY\#$ is active low, the external system must take care of special cycles such as Shutdown, Halt/ Stop, Flush, Flush Acknowledge, Write-back and Branch Trace Message. $INVD$ and $WBINVD$ instructions are used for flush bus cycle, the $WBINVD$ instruction is applied for write-back bus cycle and $FLUSH\#$ must be active low for flush acknowledge bus cycle.

Table 12.1 The special cycles of a Pentium processor

Byte Enable Signals								Special Bus Cycles
$BE7\#$	$BE6\#$	$BE5\#$	$BE4\#$	$BE3\#$	$BE2\#$	$BE1\#$	$BE0\#$	
1	1	1	1	1	1	1	0	Shutdown
1	1	1	1	1	1	0	1	Flush
1	1	1	1	1	0	1	1	Halt/ Stop
1	1	1	1	0	1	1	1	Write-back
1	1	1	0	1	1	1	1	Flush acknowledge
1	1	0	1	1	1	1	1	Branch trace message

12.9.5 Inquiry Cycles

Generally, multiprocessor-based systems use inquiry or snooping cycles to realize the MESI (Modified, Exclusive, Shared, Invalid) protocol. The external unit can be used to check whether data at a specified address is available in the on-chip cache of the Pentium processor. The external unit can invalidate the stored data and it can also invalidate the whole corresponding cache line.

Usually, any inquire cycle is performed by asserting a HOLD to force the Pentium processor to float its address bus, waiting two clocks, and then driving the inquire address and INV and asserting EADS#. Inquire cycles can be executed if the Pentium processor is forced off the bus due to HLDA or BOFF# signal. Forcefully, the entire cache line is affected by an inquire cycle; address lines A_{31} – A_5 will be driven with the valid inquire address. The INV pin must be driven along with the inquire address to indicate whether the line is invalidated or marked as shared. If INV is 1 (high), the line is invalidated. When INV is 0 (low), the line is marked as shared during an inquire hit operation.

When the Pentium processor finds out that the inquire cycle hits a line either in internal code or data cache, it drives the HIT# pin. Two clocks after EADS# is asserted, HIT# is asserted (0 or low) if the inquire cycle hit a line in the code or data cache as shown in Fig. 12.35. HIT# is de-asserted or high, two clocks after EADS# is asserted if the inquire cycle missed in internal code and data caches. Due to an inquire cycle, the

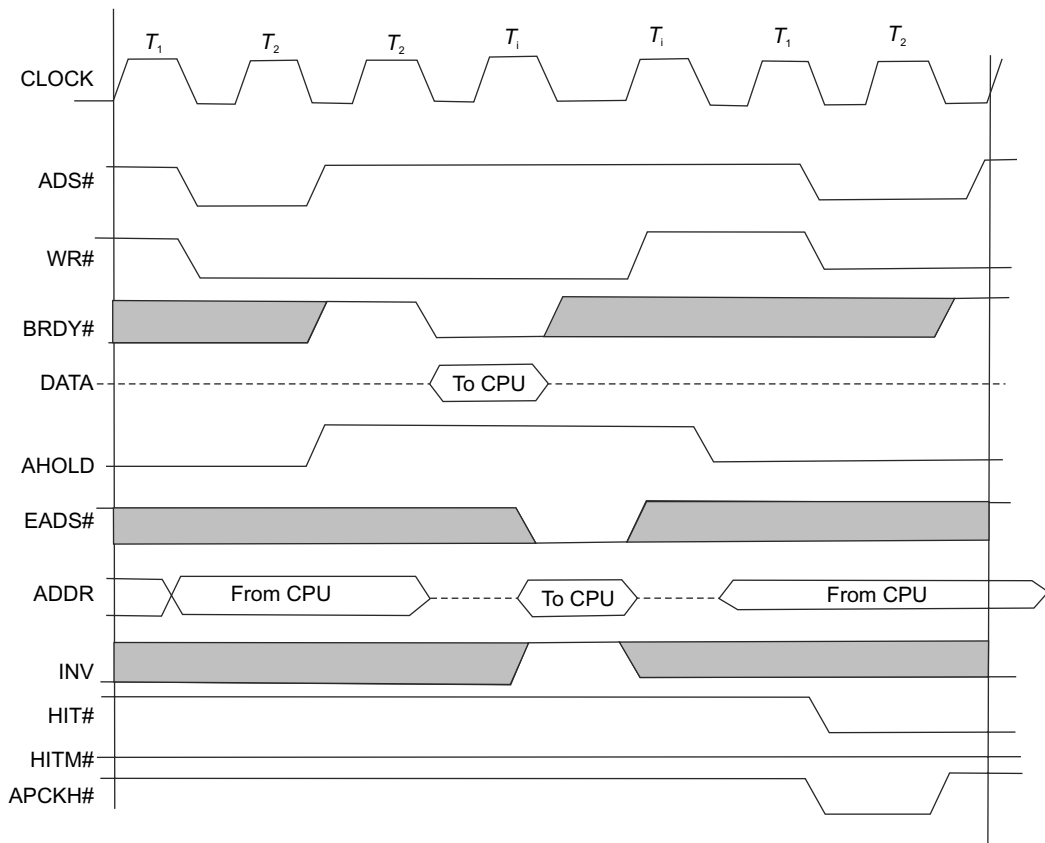


Fig. 12.35 An inquire cycle of the Pentium processor

HIT# output changes its value. But HIT# retains its value between inquire cycles. Usually, the HITM# pin is asserted just two clocks after EADS# when the inquire cycle hits a modified line in the data cache. HITM# is low to indicate to the external system that the Pentium processor holds the most recent copy of the data and any device wants to read that data.

12.10 SYSTEM MANAGEMENT MODE (SMM) OF THE PENTIUM PROCESSOR

In system management mode, the high power is used up by the Pentium processors. In this mode, the Pentium processors use SMI# (input) and SMIACK# (output) signals. The Pentium can only leave SMM after execution of RSM instruction. In SMM, the SMI# is used as an interrupt.

12.10.1 System Management Mode (SMM) RAM

Generally, the system management mode uses a battery-backed up SRAM at addresses 30000H to 3FFFFH. Usually, this covers up some DRAM addresses. Whenever the SRAM is used, the DRAM must be disabled in the system management mode and the SRAM must be disabled after exiting the system management mode. The SMIACK# signal may be used for this purpose. Figure 12.36 shows the SMM structure supported by the Pentium.

In this operating mode, the Pentium processor operates in an extended real mode. The difference between the normal real mode and an extended real mode is that the SMM address space is not limited to 1 Mbyte, as the offset registers can use 32-bit length. NMI and INTR will not provide any service until SMM has been exited.

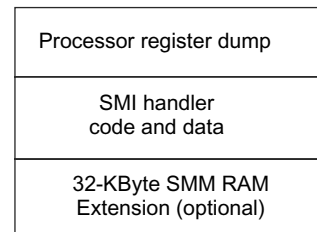


Fig. 12.36 SSM structure

12.10.2 Dual Processing

Figure 12.37 shows the dual processing of two Pentium processors. It is clear from Fig. 12.37 that the private bus between two CPUs consists of PBREQ#, PBGNT#, PHIT# and PHITM# signals. The control bus has CACHE#, KEN#, PHIT# and PHITM# signals. The processor address bus consists of BEX# and A_{31} – A_3 . D_{63} – D_0 are used as processor data bus. During dual processing, one processor is used as the primary Pentium, the other processor is the secondary or dual Pentium. After reset, only the primary Pentium processor starts to execute. After that, the primary Pentium processor checks the presence of the secondary Pentium processor in the system. If the secondary Pentium processor is present, the primary Pentium processor can enable the private bus and on-chip APICs.

Bus Arbitration

One Pentium processor is used as the Most Recent Master (MRM) and the other Pentium processor is the Least Recent Master (LRM). When the LRM wants to use the bus, the following sequence of operations will happen:

- Step 1** The least recent master asserts PBREQ# signal.
- Step 2** The most recent master completes any pending bus cycles and grants the bus.
- Step 3** The most recent master asserts PBGNT# signal.
- Step 4** The least recent master becomes the new MRM. After that LRM controls the signals to the common L2-cache, main memory and I/O devices.

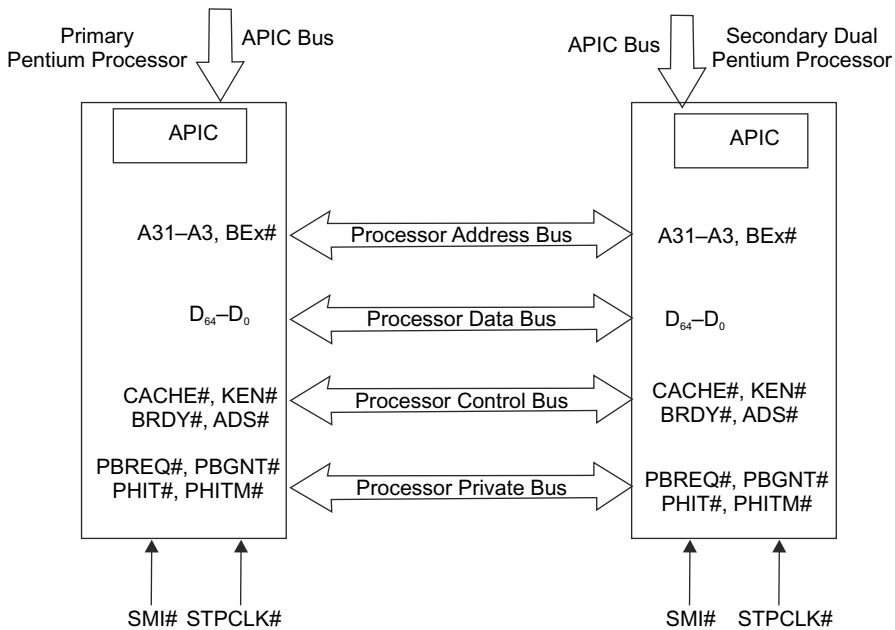


Fig. 12.37 Dual processing

On-Chip Advanced Programmable Interrupt Controllers

Each Pentium processor has an on-chip Advanced Programmable Interrupt Controller (APIC), which is enabled by the active high voltage level on the APICEN pin during reset. Before using the APIC, the BIOS must be used to initialize the APIC. In a dual Pentium system, the APICs of primary and secondary Pentium processors must be enabled. The interrupt subsystems of the two Pentiums appear as a single interrupt system to the computer system.

Two APICs of primary and secondary Pentium processors are communicated by the PIC bus, which consists of PICD0, PICD1, and PICCLK signals as depicted in Fig. 12.38. Usually, external interrupt request signals are fed to the 8259A PIC (Programmable Interrupt Controller) and after that interrupts are applied to the external I/O APIC (82489DX). The output signals of I/O APIC are distributed to the on-chip APICs of primary and secondary Pentium processors. The 8259A (PIC) is only incorporated in the circuit for compatibility

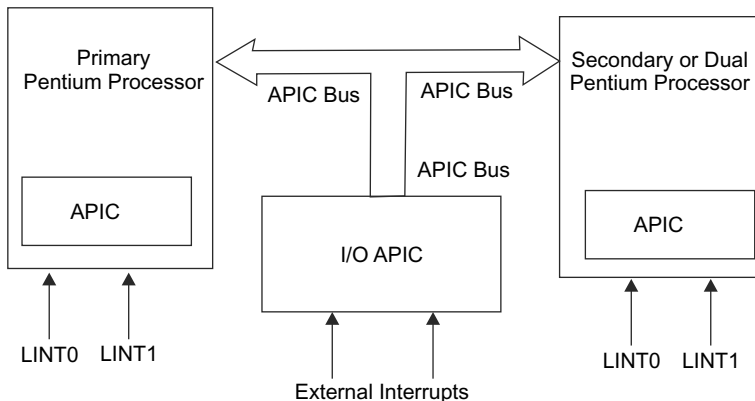


Fig. 12.38 External interrupts and on-chip APICs

features. The local APICs process local interrupts on the LINT0 and LINT1 pins. Actually, LINT0 and LINT1 pins are NMI and INTR in a single processor system.

Performance of Pentium Processor The Pentium has a time stamp counter, a control and event select register, and two programmable event counters. These counters have specific pins associated with them and allow programmers to measure the code execution time and performance parameters.

12.11 CACHE MEMORIES

Pentium processors operate at very high clock speeds to provide very good performance. But reading and writing memory off-chip is much slower than accessing CPU registers on-chip. Therefore, the access time of the memory in a system has a critical effect on the system's performance. Presently, two types of read and write memories, such as dynamic RAMs (DRAMs) and static RAMs (SRAMs), are commonly used in computer systems. DRAMs are very large in size and are relatively cheap. DRAMs are slower as compared to the clock frequency of Pentium processors and it is very difficult to make DRAMs faster. SRAMs are much faster, but they are small in size and more expensive per byte than DRAMs. SRAMs have less access time as compared to the bus speed of the processor. To achieve the benefit of fast access times and also large memory capability in a system, a combination of SRAMs and DRAMs are used. Consequently, the system designer designs a memory system using a compromise combination of SRAMs and DRAMs. The SRAMs within the larger memory system of any computer system is known as the *cache memory*.

The SRAM is used as cache memory in the system and DRAM is used within the large memory system. A cache memory is small in size and very easily accessible. Usually, SRAMs are used to hold copies of recently executed instructions and data. The SRAM always operates transparently to the programmer. The cache may be on-chip or off-chip. The on-chip caches are too much faster as the signal delay is much shorter. Therefore, cache memory works as most successive memory accesses affect only a very small address area. This is called *locality*. Consequently, at any time the program can be executing the same instructions over and over in a loop or accessing the same areas of data. In this way, the reduced access time over many cycles improves system performance significantly.

Therefore, code and data should not be placed in the slower main memory. Figure 12.39 shows the block diagram of SRAM cache interfacing with the CPU and cache controller. It is clear from Fig. 12.39 that the

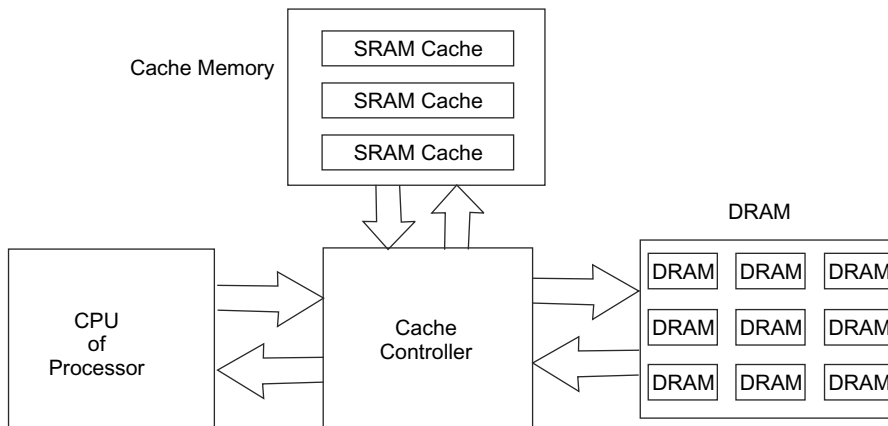


Fig. 12.39 SRM cache and cache controller

fast SRAM cache is placed between the CPU and the slower DRAM through a cache controller. The SRAM cache is used to hold the most frequently accessed instructions as well as data and make it available very quickly. The cache controller controls the complete process. The cache controller uses different strategies such as *write-through*, *write buffering*, or *modified write-through*, and *write back* to maintain the data in the cache and the main memory.

There are two types of cache organization at the highest levels of memory organization. The first is *unified cache* and other is *separate instruction and data caches*. The unified cache is commonly used for instructions and data in the 80486. The unified cache is very efficient in terms of the use of the available cache, but it is slower. Pentium and later IA-32 processors use separate instructions and data caches called a *Harvard cache*. Figure 12.40 shows the separate code and data caches (Harvard cache).

When the Pentium processor wants to read data, it sends out the memory address of desired data. Then cache controller decides whether the address of data is in the SRAM cache or in the main memory. When the data is in the cache, it is called a *cache hit* and the address is passed to cache memory without delay. The effectiveness of a cache memory is measured in terms of % cache hits measured against total memory accesses. In case of tightly written code, the cache hit rate can be about 90%. When the processor sends out an address of data which does not exist in the cache memory, it is called a *cache miss*. Subsequently, the cache controller must go out to the main memory. The proportion of memory accesses which are satisfied by the cache is known as the *cache hit rate*, and the proportion of memory accesses which are not satisfied by the cache is called the *cache miss rate*. Usually, the miss rate of a Pentium processor will be a few per cent to remove the memory bottleneck.

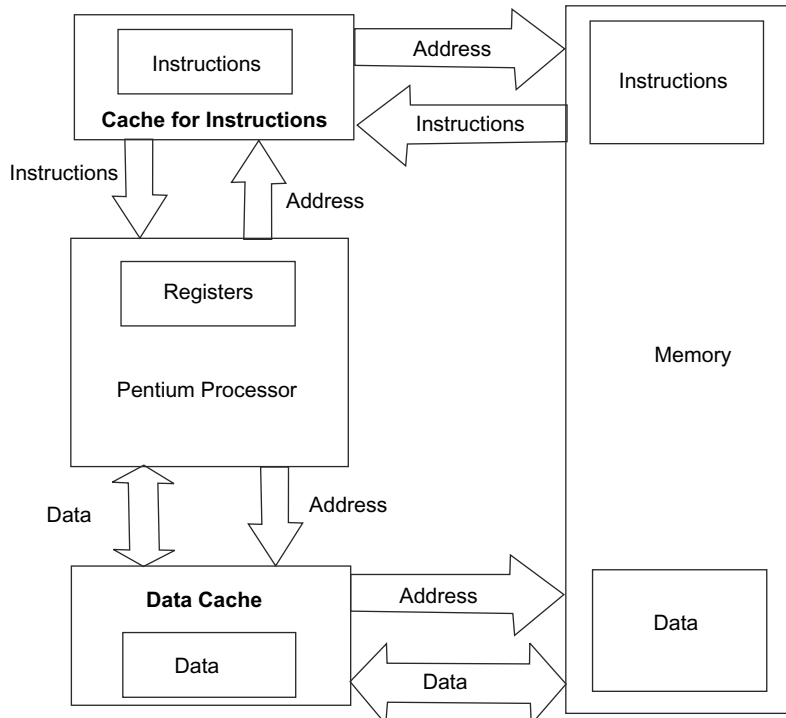


Fig. 12.40 Separate code and data caches of Pentium processor

When there is a hit, the cache controller reads data from the fast SRAM cache and sends the data to the CPU without any wait states. If there is a cache miss, the cache controller first reads the main memory, so that the read is switched to the main memory. This operation requires wait states so that the cache controller must de-assert the RDY# signal. Subsequently, the processor will insert wait states until the data is available. The cache controller simultaneously controls the cache and main memory. When a cache miss occurs, not only the requested data bytes but also a small block of data, called the *cache line*, will be read from the main memory into the cache memory. This process is known as *cache line fill*. To get best performance, the data bytes which are requested by the CPU for read, must be passed on before the cache line fill is completed.

Generally, cache lines are 32 bytes in size, as data and code exist in blocks within memory. The cache controllers use a burst mode to read cache lines and multiple bus cycles are used to read a large number of bytes. Actually, burst mode almost doubles the bus transfer rate. As a result, a cache line can be read quicker than single values. In this way, cache lines improve the system performance.

When the Pentium processor does a write operation, the cache controller always checks if the data is available in the cache. If so, the data is written to the cache. It is also required to write data in the main memory. For the case of a cache hit, two different strategies such as *write-through* and *write-back* are used for this purpose.

Write-through Strategy

In a write-through strategy, the CPU always stores data to the main memory for any write operation even though there is a cache hit. All writes are switched through to the main memory as depicted in Fig. 12.41(a). When there is a cache hit, the cache memory will be updated with the new data. The disadvantage of a write-through strategy is that all writes go to the main memory and wait states are inevitable. Therefore, the writing speed becomes slower. By using write buffers (FIFO), the writing speed can be increased up to a certain extent as shown in Fig. 1.1(b). But this process works until the buffers fill up. After filling the FIFO buffer, multiple write operations enter wait states.

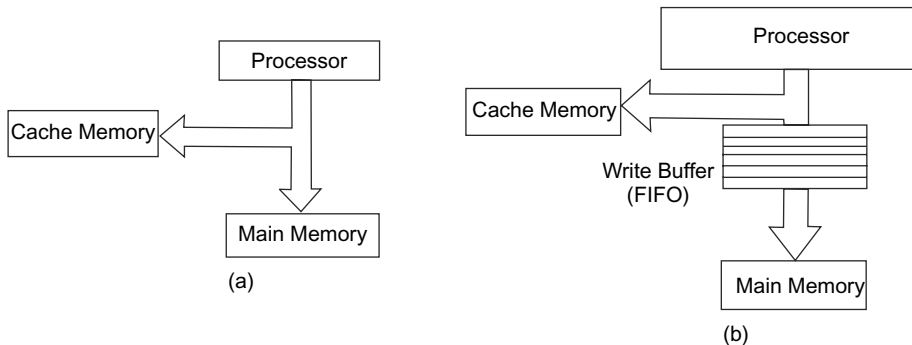


Fig. 12.41(a) Write through (b) Modified Write through using write buffering

Write-Back Strategy

Figure 12.42 shows the write-back strategy. In a write-back strategy, the cache memory receives all the write operations and updates the cache entry only. The cache lines always remember that they have been modified using a *dirty* bit on each line. The cache line which has been already changed is only written to the main memory after some specific events. Hence, Pentium processor updates the main memory by the following ways:

- (i) When WBINVD (write-back and invalidate data cache) instruction is executed
- (ii) Though hardware signal FLUSH#

- (ii) When there is a cache miss, a new cache line is replaced by an old one
- (iii) Whenever an internal or external inquiry cycle is executed

The disadvantage of a write-back strategy is that exchanging cache lines takes longer time during exchanging data, as data will be written to the main memory before the new memory is loaded into the cache memory.

Write-allocate

In the write-allocate approach, if data is being written out to main memory, the cache controller should try to cache it even in the case of a cache miss. Consequently, the latest data will be waiting in the cache the next time it is required. Therefore, the cache controller allocates a new cache entry for the write cycle. After that, if the cache is fully used, the new allocation may displace a more useful old cache entry. The main application of the write-allocate strategy is for bus snooping. But the fact is that if there are some cache misses, most caches simply switch through to main memory and just ignore the cache.

When the processors or DMA controllers can access the main memory, the cache controller must allow the cache that its data become invalid, when the other devices want to write to the main memory. This process is known as *cache invalidation*. Figure 12.43 shows the invalid write cycles without caching. While the cache controller uses a write-back strategy, there are certain times when the contents of the cache memory have to be transferred to main memory. For example, a DMA controller wants to write data from the main memory out to a peripheral, but the latest data is only in the cache. This case is called a *cache flush*.

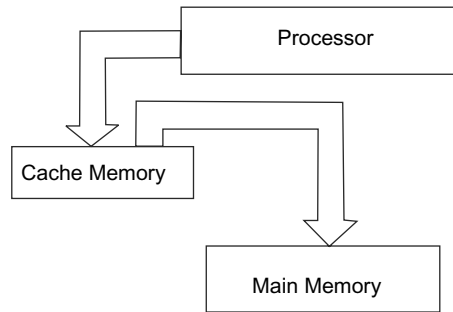


Fig. 12.42 Write back strategy

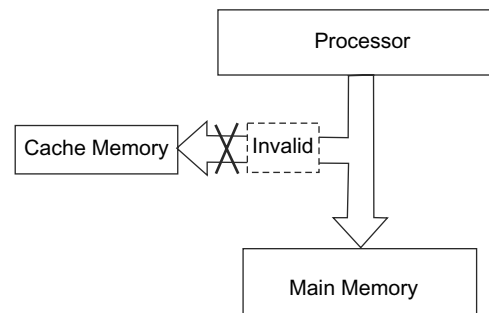


Fig. 12.43 Invalid write cycles without caching

12.11.1 Cache Organization

A cache memory is used to store both the data and the address where the data is stored in the main memory. There are methods of cache organization such as *direct mapped cache* and *two-way set-associative cache*. In this section, both direct mapped cache and two-way set-associative cache are discussed elaborately.

Direct Mapped Cache

The simplest cache organization is the direct mapped cache as shown in Fig. 12.44. The features of a direct-mapped cache are given below:

- (i) A specified data or a memory item is stored in a unique location in the cache memory and two items with the same cache address will compete for use of that location. For example, when a cache line is already placed at an address in the cache memory, we also want to put another cache line at another address in the cache memory. These two addresses should have the same $A_{12}-A_4$ address bits all the time. Therefore, these addresses cannot both be placed in the cache at the same time. This is known as *contention*.

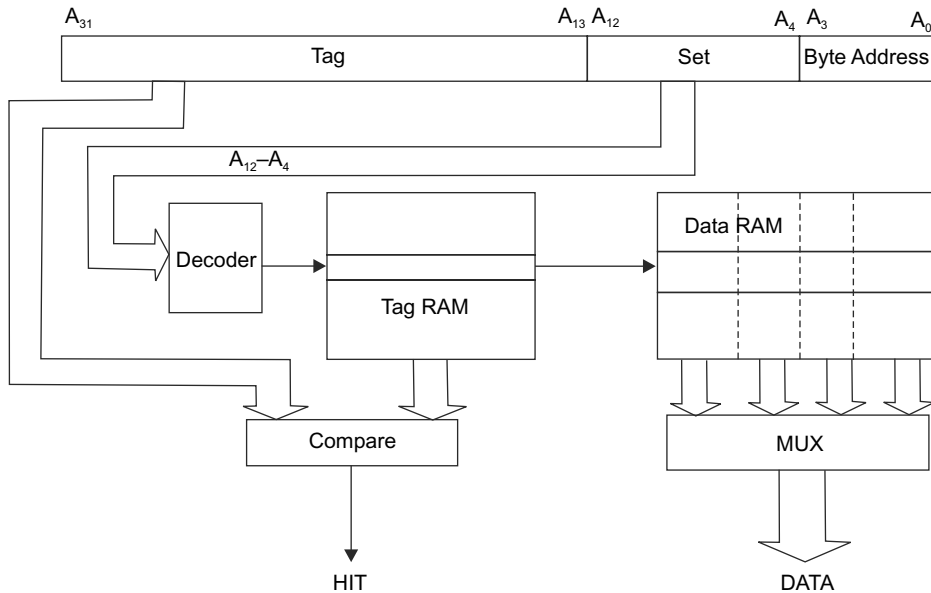


Fig. 12.44 Direct mapped cache organization

- (ii) Those bits are not used to select within the line or to address the cache RAM, these bits will be stored in the tag field.
- (iii) The tag and data access can be performed simultaneously and also provide the fastest cache access time of any organization.
- (iv) The tag RAM is always smaller than the data RAM. Hence, the access time of the tag RAM is shorter than data RAM and the tag comparison can be completed within the data access time.
- (v) The direct mapped cache organization can store 8 Kbytes in 16-byte cache lines. The direct mapped cache has 9 address bits A_{12} – A_4 or 512 lines. The first 4 bits (A_3 – A_0) of the 32-bit address bus A_{31} – A_0 are used to address the bytes within the line and there are 19 tag bits (A_{31} – A_{13}). Therefore, 512×19 (9728) bits are required for storing the tag.
- (vi) While data is loaded into the cache memory, a block of data will be fetched from memory using burst cycles. Normally, line size is not smaller than the block size. When the block size is smaller than the line size, a valid bit will be used to indicate each block within the line. To get the simplest cache organization, the line will be equal to block size.

Two-Way Set-Associative Cache

To reduce the problems due to contention, the *set-associative cache* is used by allowing a particular memory item to store in more than one cache location. Figure 12.45 shows the *2-way set-associative cache* organization. In this cache organization, two direct-mapped caches work in parallel. The address applied to the cache may find its data and each memory address can be stored in one of the two places. Each of the two items, which were in contention for a single location in the direct-mapped cache organization, may be stored at one of the two places after allowing the cache to hit on both.

The two-way set-associative caches have 8 Kbyte caches with 16-byte cache lines and eight address bits A_{11} – A_4 and 256 lines in either half of the cache. Therefore, four bits of the 32-bit address, A_3 – A_0 , are used to select a byte from the cache line and eight bits A_{11} – A_4 are used to select one line from each half of the

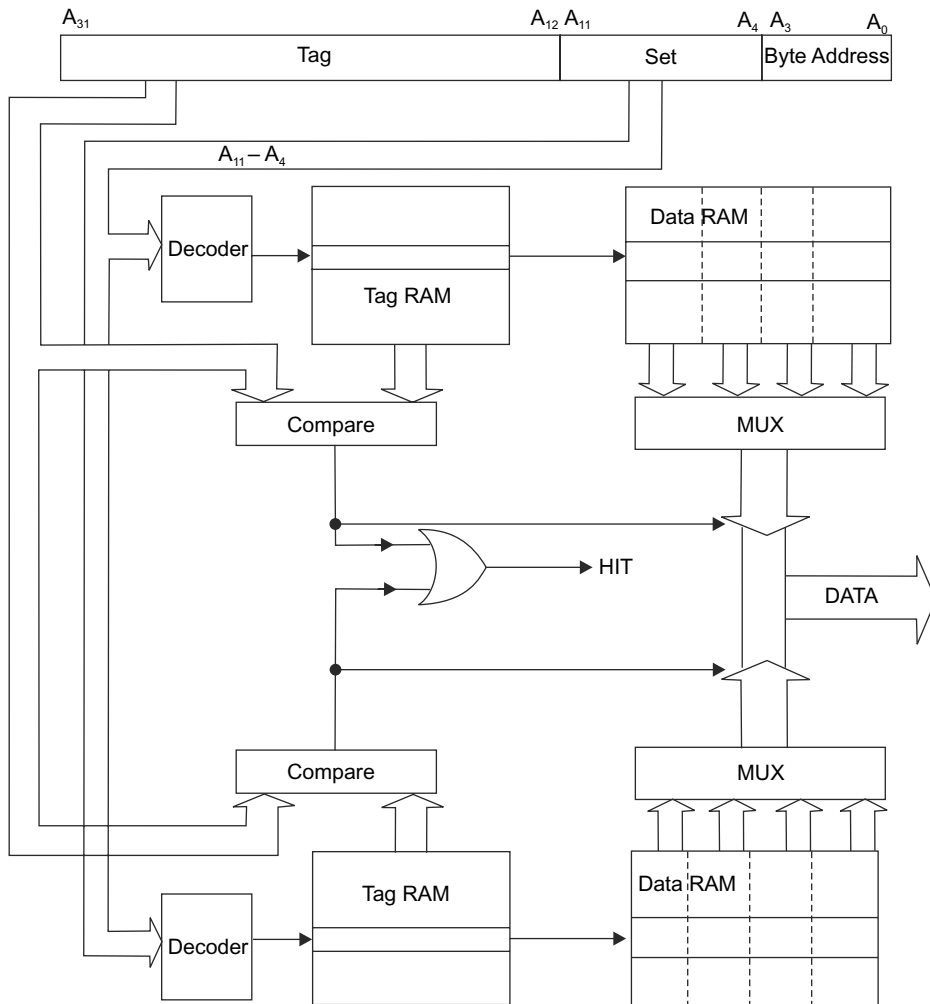


Fig. 12.45 Two-way set-associative cache organization

cache. Consequently, there are 20 address bits A_{31} – A_{12} for the address tag. The access time of a two-way set-associative cache is slightly larger than the direct-mapped cache as the extra time is required to perform the multiplexing operation.

12.11.2 Cache Consistency

The cache consistency problem occurs when data can be modified by more than one source. When a copy of data is held in both main memory and a cache memory, one copy is changed and the other copy is stale and the system consistency will be lost. If there are two caches in the system, the problem becomes very complicated. Assume a multiprocessor system consists of two processors, namely, Pentium-A and Pentium-B. If the secondary Pentium processor (Pentium-B) overwrites a common memory location, the other processor (Pentium-A) should know that this has occurred. Usually, the MESI protocol for cache lines is used in Pentium and advanced processors to ensure cache consistency. Figure 12.46 shows the bus snooping when two processors each with a local cache have access to a common main memory.

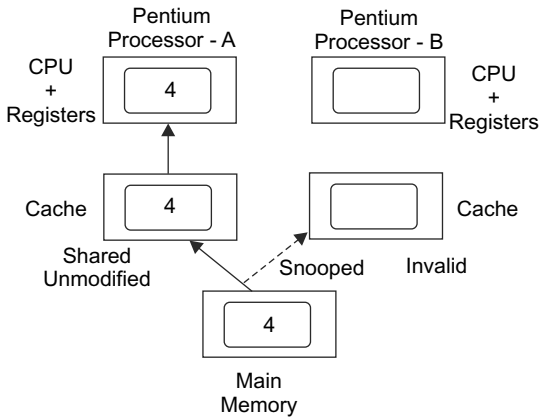


Fig. 12.46(a) Bus snooping when Pentium processor-A reads

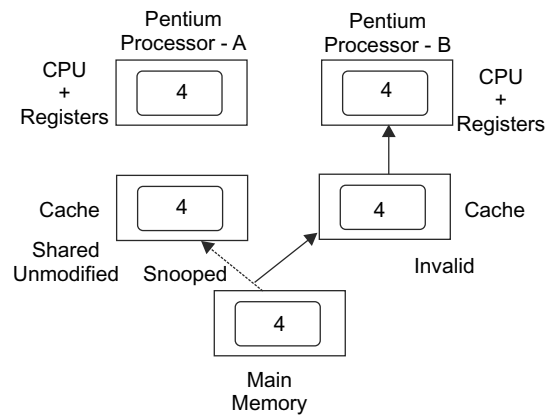


Fig. 12.46(b) Bus Snooping when Pentium processor-B reads

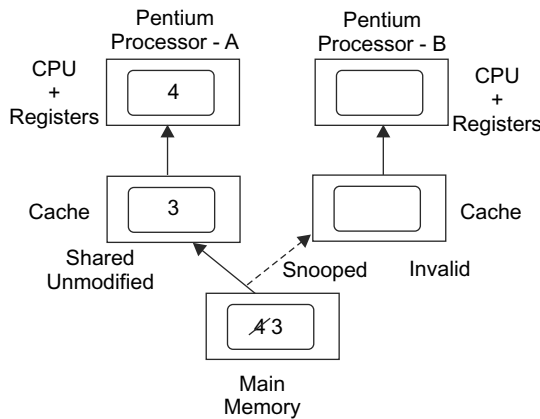


Fig. 12.46(c) Bus snooping when Pentium processor-A writes

12.11.3 MESI (Modified, Exclusive, Shared, Invalid) Protocol

The abbreviation of MESI is *Modified, Exclusive, Shared, and Invalid*, which are the four possible states of a cache line. The MESI protocol is a general mechanism to control cache consistency, using snooping techniques. The Pentium processors can change the state of a cache line through read or write cycles or internal snooping and other devices such as L2-cache controller can change the state through external snooping. The MESI protocol provides each cache line that can be one of the four states, and the MESI protocol is managed by the two MESI bits. Figure 12.47 shows the state transition diagram of MESI protocol. The modified, exclusive, shared, and invalid stages of the protocol are explained in this section.

✓ **Modified (M)** When the data of a cache line is marked as modified (M), it is available in a single cache of the complete system only. This cache line can be read or written to without an external cycle.

✓ **Exclusive (E)** The exclusive (E) cache line is always stored in only one of the caches in a computer system, though it has not been modified. Hence its values are the same as in the rest of the system. The cache line can be read or written to without an external cycle. Once it is written, to the cache line should be set to modify.

✓ **Shared (S)** The shared (S) line can be stored in other caches of the system. The shared line always has the current value so that read accesses can be obtained from the cache. Write accesses to a shared cache line are switched through the external data bus, whenever any cache write strategy is used. Therefore, the shared cache lines in the other caches are invalidated.

✓ **Invalid (I)** The cache line which is marked as invalid is not available in the cache. The cache lines marked as invalid (I) lines might be empty or could have invalid data in the cache. Each access to an invalid cache line generates a cache miss. During read access, the cache controller starts a cache line fill and the cache controller switches the write through to the external bus, rather than a write-allocate.

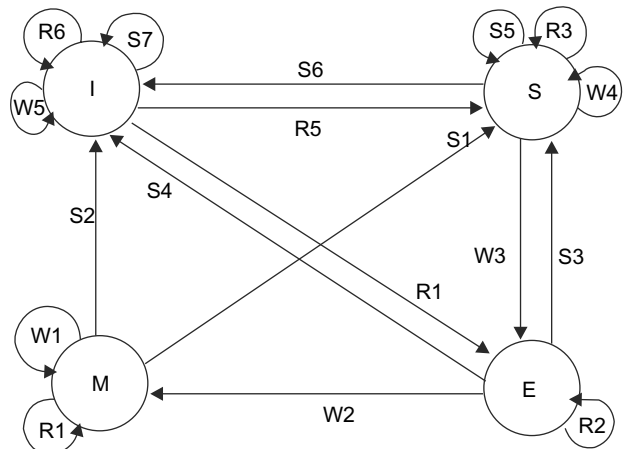


Fig. 12.47 State transition diagram of MESI protocol

It is clear from the state transition diagram of MESI protocol that there are six read access states, R1 to R6; five write access states, W1 to W5, and seven inquiry or snooping states S1 to S7. The function of each state is given below:

Six Read Access States R1 to R6

- ◆ **R1** Read access leads to a hit. Data is available in the cache and it is transferred to the CPU of Pentium processor ($M \leftarrow M$).
- ◆ **R2** Read access leads to a hit. Data is available in the cache and it is transferred to the CPU of Pentium processor ($E \leftarrow E$).
- ◆ **R3** Read access leads to a hit. Data is available in the cache and it is transferred to the CPU of Pentium processor ($S \leftarrow S$).
- ◆ **R4** Read access leads to a miss. Data is not available in the cache. The cache controller performs an external read and a line fill ($E \leftarrow I$).
- ◆ **R5** Read access leads to a miss. Data is not available in the cache. The cache controller performs an external read and a line fill ($S \leftarrow I$).
- ◆ **R6** Read access leads to a miss. Data is not available in the cache. The cache controller is not able to perform a line fill and the cache line remains invalid ($I \leftarrow I$).

Five Write Access States, W1 to W5

- ◆ **W1** Write access leads to a hit. Data is available in the cache. As MESI protocol operates with write-back cache strategy, no write cycle is sent to the external bus ($M \leftarrow M$).
- ◆ **W2** Write access leads to a hit. Data is available in the cache and it was not previously overwritten. As MESI protocol operates with write-back cache strategy, no write cycle is sent to the external bus ($M \leftarrow E$).
- ◆ **W3** Write access leads to a hit. Data is available in the cache but it was shared. The cache controller sends a write cycle to the external bus. Then cache line is used in one cache. The main memory is updated, and the cache becomes exclusive ($E \leftarrow S$).

- ◆ **W4** Write access leads to a hit. This state will operate for a write-through cache. At this time, all writes switched to external bus and the cache line stay as shared ($S \leftarrow S$).
- ◆ **W5** Write access leads to a miss. Data will be written to the main memory, but not in the cache. The cache line remains invalid ($I \leftarrow I$).

Seven Inquiry or Snooping States, S1 to S7

- ◆ **S1** The snooping (inquiry) cycle hits a modified cache line. The reference cache line will be written back to main memory ($S \leftarrow M$).
- ◆ **S2** The snooping (inquiry) cycle hits a modified cache line and this state becomes invalidated. The reference cache line is written back to main memory anyway ($I \leftarrow M$).
- ◆ **S3** The snooping (inquiry) cycle hits an exclusive cache line. This state has not been modified and it does not require writing back to main memory. The content of main memory is written to another cache line and the previously exclusive line is now shared ($S \leftarrow E$).
- ◆ **S4** This snooping (inquiry) cycle hits an exclusive cache line. This state has not been modified and does not need to write back to main memory. The content of the main memory is written to another cache line. Due to some reason, this line becomes invalidated ($I \leftarrow E$).
- ◆ **S5** The snooping (inquiry) cycle hits a shared cache line. This snooping cycle informs the system that this cache line is available in the cache ($S \leftarrow S$).
- ◆ **S6** The snooping (inquiry) cycle hits a shared cache line. For some reason, this line will be invalidated ($I \leftarrow S$).
- ◆ **S7** The snooping (inquiry) cycle hit an invalid cache line ($I \leftarrow S$).

12.11.4 L2 Caches and MESI Protocol

Generally, L1 caches are small in size and available as local. L2 caches are larger in size but a little bit slower. Presently, L2 caches are available on-chip Pentium processors. For example, L1 cache of Pentium-II processor can provide data in one cycle, but L2 cache takes two cycles to supply data. The *Modified, Exclusive, Shared, and Invalid* (MESI) protocols can be applied to the multicache system. The cache consistency can be achieved when all the addresses in the L1 cache are available in the L2 cache as depicted in Fig. 12.48. The data in the two caches may not be the same as the L1 cache may be a write-back cache.

The data transfer between the CPU and main memory takes place over the L2 cache. The MESI protocol can be performed over the three stages such as (i) L1 cache, (ii) L2 cache, and (iii) main memory. The MESI state of an L2 cache line always operates one step ahead of the MESI state of the related L1 cache line. An inquiry or snooping cycle is used by the L2 cache to decide whether a certain cache line is available in the L1 cache and it has been changed. If there is an inquiry cycle in a modified line, the L2 cache will be updated by a write-back cycle to the L2 cache. After that, the L2 cache can also ask for a back-invalidation cycle and this can indicate to the hit line being made invalid. Figure 12.49 shows the caches of IA-32 bit Pentium processors.

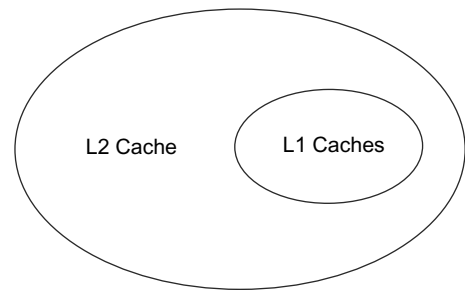


Fig. 12.48 L1 caches are within the L2 cache

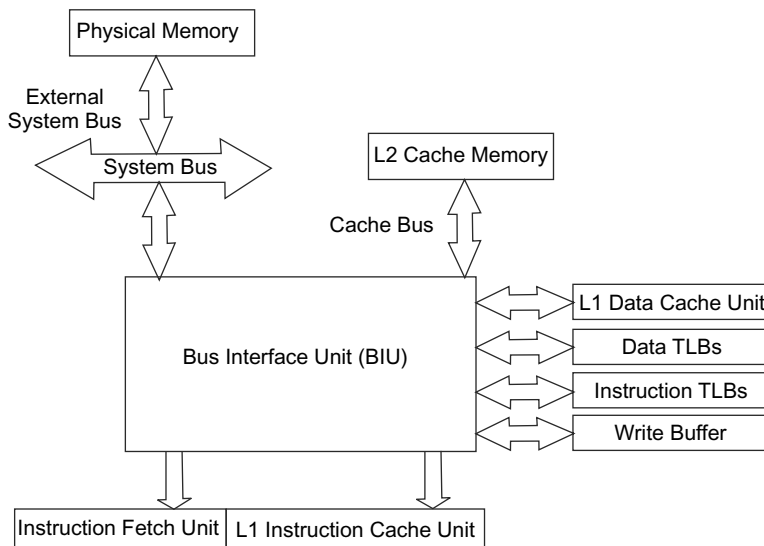


Fig. 12.49 Caches of IA-32 bit Pentium processors

12.11.5 Pentium On-Chip Caches

The Pentium processor supports an 8 Kbyte data cache and an 8 Kbyte code cache. The data cache (D-cache) is used to store data and the instructions are stored in code cache (I-cache). The code and data caches are two-way set-associative. Both D and I caches can be accessed simultaneously. The code cache provides up to 32 bytes of opcodes, but the data cache provides data for two data references in the same clock pulse. Both caches are parity protected. The cache page size is 4K or 128 lines and the cache line size is 32 bytes. Each cache line is filled by a burst of four read cycles through the processor's 64-bit data bus. The two-way set-associate scheme consists of two different ways such as Way 0 and Way 1 as shown in Fig 12.50. Each cache way contains 128 cache lines.

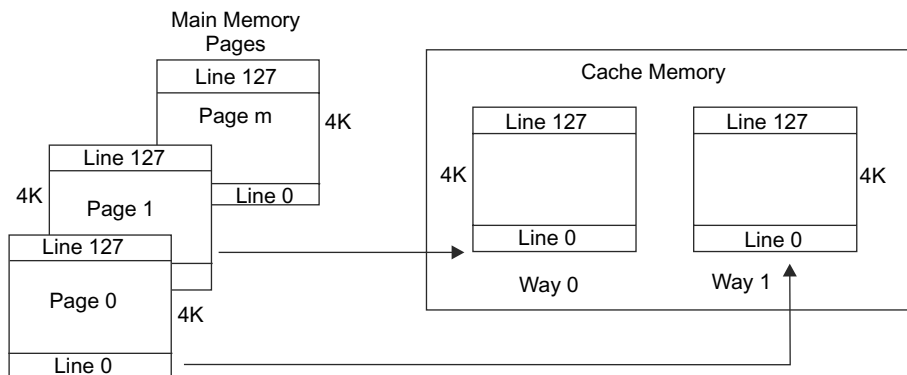


Fig. 12.50 Two-way set-associate cache

The code cache is write-protected to prevent code corruption. The code cache can support the two states of the MESI protocol, namely, the S and I states. The data cache fully supports the MESI protocol. The data cache is configurable as write-back or write-through strategy. Memory areas can be defined as noncacheable

by external hardware or software. Cache line consistency as well as replacement can be done using hardware. Usually, the LRU method is used for cache line replacement in both the data-cache and code-cache. This LRU method requires one bit per set in each of the caches. Figure 12.51(a) shows the data cache and the code cache is depicted in Fig. 12.51(b).

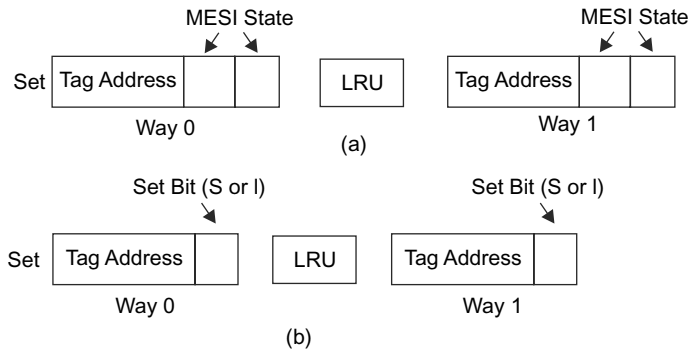


Fig. 12.51 (a) Data cache (b) Code cache

Physical Addresses

Data cache and code cache can be accessed by a physical address. Therefore, each cache has its own TLB to translate linear addresses to physical addresses. For example, the data cache has separate TLBs for 4 Kbyte and 4 Mbyte pages, but the code cache has only one TLB for both 4 Kbyte and 4 Mbyte pages. Actually, the TLBs are 4-way set associative caches.

Cache Operating Modes

The cache operating modes are controlled by the CD and NW bits of CR0 register. For normal operation and best performance, CD = 0 and NW = 0. When both CD and NW bits are set to 1 after reset. When both CD and NW are 0, the cache performs the following operation:

- (i) Read hits access the cache.
- (ii) Read misses cause line fills.
- (iii) Enter the Exclusive (E) or Shared (S) state under the control of the WB/WT# signal.
- (iv) Write hits update the cache.
- (v) Write to shared lines and write misses will appear externally.
- (vi) Write to shared lines will be changed to the Exclusive (E) state under the control of WB/WT# signal.
- (vi) Inquire or snooping cycles are performed.

To disable the cache, the following operations are required:

- (i) Set the CD and NW bits. Therefore CD = 1 and NW = 1.
- (ii) Flush the caches.

Page Cache-ability

Figure 12.52 shows the format of a page directory or page-table entry. To control page-cache ability, page directory and page table entries have two bits, namely, PWT (Page Write Through) and PCD (Page Cache Disable). The states of these bits are placed on the PWT and PCD pins during memory access cycles. The PWT bit controls write strategy for the external L2 caches. When PWT = 1, a write through strategy for the current page is used. If PWT = 0, a write-back strategy is used for the current page. The PCD bit controls cache ability on a page-by-page basis. The PCD bit is logically ANDed with the KEN# signal to control cache-ability on a cycle-by-cycle basis. When PCD = 0, this bit enables page caching operation. If PCD = 1, it disables page caching operation. The Cache line fills are enabled if PCD = 0 and KEN# = 0.

31	12	6	5	4	3	2	1	0
Address		D	A	PCD	PWT	U	W	P

Fig. 12.52 Format of a page directory or page table entry

Pentium L2 Cache

Intel has developed a 82496 cache controller for the Pentium processors to operate with the cache SRAM modules (82491) which work as an L2 cache in the system. The 82496 cache controller is used to implement the MESI Protocol for the L2 caches. This cache controller may be configured as an L2 cache with 256 Kbyte or 512 Kbyte size. The 82496 cache controller implements a two-way set-associative cache organization which can support cache line sizes of 32, 64 and 128 bytes.

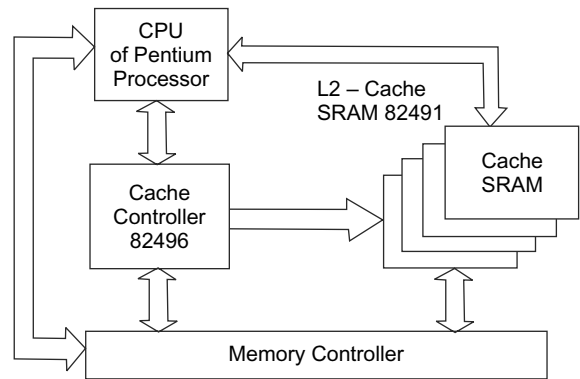


Fig. 12.53 L2 cache in a Pentium-based system

12.12 PENTIUM MMX

MMX stands for Multimedia Extension, Multiple Math Extension, or Matrix Math Extension. The MMX is a Single Instruction Multiple Data (SIMD) instruction and it was designed by Intel in 1996. The MMX TM technology has the following new extensions in Intel Architecture 32-bit Pentium processor:

- (i) It has eight MMX registers such as MM0–MM7.
- (ii) Four MMX data types such as packed bytes, packed words, packed double words and quad words. In packed bytes, eight bytes are packed into one 64-bit quantity. In packed words, four 16 bit words are packed into one 64-bit quantity. In packed double words, two 32-bit double words packed into one 64-bit quantity. The quad word is one 64-bit quantity.
- (iii) It has a 57 MMX instruction set.
- (iv) MMX technology supports saturating arithmetic and wraparound mode.

In 1997, Intel developed Pentium MMX processor incorporating the multimedia extension (MMX) technology for different multimedia applications such as 2D and 3D image processing. In multimedia applications, most of the operations involve pixels. During representation of a color image, one pixel consists of three components such as red, green and blue. Each component of a pixel is an 8-bit integer. The intensity of each component of a pixel can be varied from 0 to 255. The image processing and image compression operations require matrix multiplication and matrix convolution type computations and also require operations on multiple numbers of pixels simultaneously. Therefore, all multimedia applications need a Single Instruction Multiple Data (SIMD) type architecture.

The Pentium MMX processor has eight 64-bit MMX registers known as MM0 through MM7. The MMX instructions access the MMX registers directly. These registers can be used to perform computations with MMX data types. Intel provides a set of 57 MMX instructions which are used for improving graphic performance in image processing, image filtering, image enhancement and coding, etc. Pentium processors (P5) can operate on two pixels simultaneously, whereas MMX instructions of Pentium MMX processor can operate eight pixels at the same time. The Pentium MMX processor has the following drawbacks:

- (i) MMX instructions work with integers only, though 2D and 3D graphics often require floating point arithmetic operations.
- (ii) MMX instructions and floating-point unit instructions can share registers, but MMX and FPU instructions do not work simultaneously.

12.13 PENTIUM PRO, PENTIUM II, AND PENTIUM III: P6 FAMILY PROCESSORS

12.13.1 Pentium Pro

The Pentium Pro is a sixth-generation x86 microprocessor and it was introduced by Intel in 1995. This processor was developed based on the P6 micro-architecture. The Pentium and Pentium MMX processors have 3.1 and 4.5 million transistors respectively, but the Pentium Pro consists of 5.5 million transistors. The Pentium Pro is capable of both dual-processor and quad-processor configurations. The common features of Pentium Pro are given below:

- ◆ The Pentium Pro has a 12-stage decoupled super pipeline architecture which uses an instruction pool.
- ◆ It has extra decoded stages to dynamically translate IA-32 instructions into a sequence of buffered micro-operations.
- ◆ It has speculative execution through register renaming with 40 registers.
- ◆ It has dynamic program execution.
- ◆ Pentium Pro has a 36-bit address bus.
- ◆ It has an 8 KByte instruction cache. It has separate L1 code and data caches with write back strategy.
- ◆ It has data forwarding and dynamic branch prediction.
- ◆ The Pentium Pro has two integer units and one floating-point unit. One of the integer units shares the same ports as the FPU.
- ◆ It has an integrated L2 cache into processor core connected over a dedicated bus running at the CPU clock pulse (half or full).
- ◆ In the Pentium Pro processor, x86 instructions are decoded into 118-bit micro-operations (micro-ops).
- ◆ The Pentium Pro processor clock speeds are 150, 166, 180 or 200 MHz with a 60 or 66 MHz external bus clock.
- ◆ This processor is very popular in multiprocessing configuration.
- ◆ P6 architecture operates with 32-bit OS such as Windows NT 3.5, Unix and OS/2.
- ◆ It is packaged in ceramic multi-chip modules (MCM). The MCM has 387 pins which is approximately half of pin grid array (PGA) package. The MCM package was designed for Socket 8.
- ◆ Pentium Pro processor core operates at 3.1 V to 3.3 V.
- ◆ Pentium Pro is fabricated in a 0.6 μm BiCMOS process for 133 MHz operating frequency, in a 0.5 μm BiCMOS process for 150 MHz, and in a 0.35 μm BiCMOS process for 166, 180, and 200 MHz.

12.13.2 Pentium II

After the Pentium Pro processor, Pentium II was developed by Intel in 1997. Pentium II is manufactured based on P6 micro-architecture and it is a sixth-generation x86-compatible microprocessors. This processor core contains 7.5 million transistors and it is an improved version of the first P6-generation core of the Pentium Pro. All features of the Pentium Pro have been incorporated with Pentium II. This processor has a larger cache. As this processor can operate at 2.8 V, the power consumption is reduced significantly. Pentium II can support MMX instructions for enhanced floating-point operation. The features of Pentium II processor are given below:

- ◆ The L2 cache size is increased to 512 KB from the 256 KB on the Pentium Pro. The L2 cache operates at half of the processor's clock frequency, whereas the L2 cache of Pentium Pro operates at the same frequency as the processor.

- ◆ The Pentium II has a 32 KB L1 cache which is double of the Pentium Pro. This processor has separate 16 KB L1 data and 16 KB L1 instruction caches.
- ◆ The 16-bit code execution performance is available on the Pentium II processor.
- ◆ The Pentium II is a consumer-oriented version of the Pentium Pro. It is cheaper to manufacture because of the separate slower L2 cache memory.
- ◆ The improved 16-bit performance and MMX support make it suitable for Windows operating systems and multimedia applications.
- ◆ The Pentium II is packaged in a slot-based module rather than a CPU socket.
- ◆ The Pentium II processor clock speeds are 233, 266, 300, 350, 400 or 450 MHz with a 66 MHz front side bus.
- ◆ Pentium II *Klamath* (80522) processor core operates at 2.8 V, and Pentium II *Deschutes* (80523) processor core operates at 2.0 V.
- ◆ Pentium II *Klamath* is fabricated in a 0.35 μm CMOS process and Pentium II *Deschutes* is fabricated in a 0.25 μm CMOS process.

12.13.3 Pentium III

After the Pentium II, the next version of the Pentium processor is Pentium III. This processor was developed by Intel in 1999. Pentium III is a sixth-generation x86-compatible microprocessor and it was manufactured based on P6 micro-architecture. This processor core contains 9.5 to 28 million transistors. All features of the Pentium II have been added with Pentium III. This processor is used for high-performance desktop computers and servers which can operate Windows NT, Unix and Windows 98 operating systems. This processor is suitable for audio and video processing, image processing, and Internet and multimedia applications. The new features of the Pentium III processor are given below:

- ◆ Pentium III can operate in multiple branch prediction algorithms.
- ◆ 70 new instructions are added to the Pentium III for multimedia and advanced image processing applications. This processor has SSE (streaming SIMD) extensions.
- ◆ It can support dynamic execution technology.
- ◆ Pentium III has eight 64-bit Intel MMX registers and 57 MMX instructions for multimedia applications.
- ◆ It has an on-die 512 Kbyte L2 cache.
- ◆ This processor is available at operating frequencies of 450 MHz to 1.4 GHz with 100 MHz to 133 MHz front side bus.
- ◆ Pentium III can operate at 2.0 V to 1.45 V.
- ◆ Pentium III *Katmai* is fabricated in a 0.25 μm CMOS process, Pentium III *Coppermine* is fabricated in a 0.15 μm CMOS process and Pentium III *Tualatin* is fabricated in a 0.13 μm CMOS process.

In this section, the internal architecture, instruction pool, 36-bit address bus and paging mechanism of P6 family processors are discussed elaborately.

12.13.4 Architecture of P6 Family Processors

The block diagram representation of internal architecture P6 family processors is depicted in Fig. 12.54. It is clear from Fig. 12.54 that the P6 family processor has a Bus Interface Unit (BIU), L1 data cache, L1 code cache, instruction fetch unit, instruction decoder, Branch Target Buffer (BTB), Register Alias Table (RAT), Reservation Station (RS), retire unit and on-chip Advanced Programmable Interrupt Controller (APIC).

Bus Interface Unit (BIU) The bus interface unit is used to interface between system bus, 64-bit dedicated cache bus and L1 code and L1 data caches.

L1 Data Cache The L1 data cache is organized as two-way set-associative cache organization. Pentium Pro processor has 8 Kbyte data cache but Pentium II and Pentium III processors have 16 Kbyte data cache with two access ports. The write access (write port) and a read access (read port) can be occurred simultaneously. The data cache can support all 4 MESI protocol states.

L1 Code Cache The L1 cache is also organized in two-way set-associative cache organization. The pentium Pro processor an has 8 Kbyte code cache whereas Pentium II and Pentium III processors have 16 Kbyte code cache. The code cache supports only two states of 4 MESI protocol states, i.e., S and I states.

Instruction Fetch and Decoder In the instruction fetch stage, instructions are fetched from the instruction cache. After that instructions are decoded. There are 3 decode units in the instruction decoder

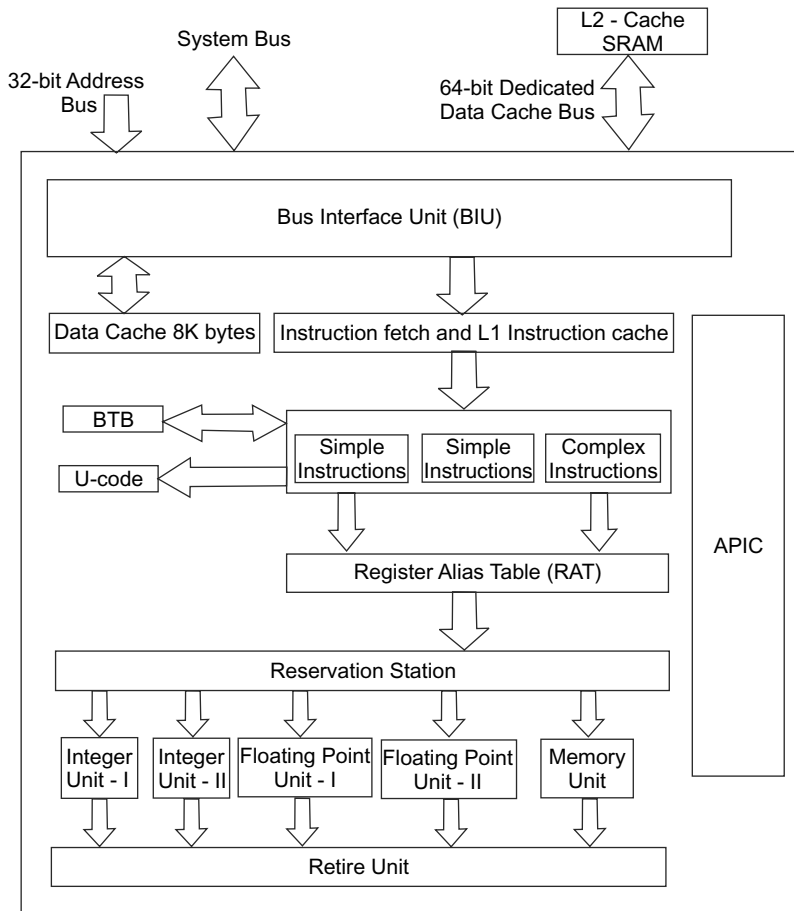


Fig. 12.54 Block diagram of P6 family processors

and they are operating in parallel. Two decode units are used to decode simple instructions that microcodes are not required for simple instructions and the third decoder is used to decode CISC or complex instructions that require a microcode. Usually, instructions are decoded into sequences of μ -ops.

Branch Target Buffer (BTB) The branch target buffer is used to store 512 branch targets along with information to predict branches correctly. The BTB is also required for speculative execution due to the length of the pipeline.

Internal Registers The P6 family, such as Pentium Pro, Pentium II and Pentium III processors have 40 internal registers. These processors use these registers in place of the x86 registers. Actually, the Register Alias Table (RAT) remaps the x86 registers from the instruction to the 40 internal registers by using the register-renaming technique.

Reservation Station and Execution Unit In P6 processors, there are five execution units which consist of two integer units INT-I and INT-II, two floating-point units FPU-I and FPU-II, and a Memory Unit (MU). The reservation station is used to send instructions to the above five execution units. All five execution units work independently. In best conditions, execution of five instructions can be completed in one clock pulse.

Retire Unit and APIC The retire unit resolves data dependencies. This unit can verify branches and writes to the x86 architectural registers. The on-chip advanced programmable interrupt controller supports multiprocessing with up to 4 processors.

12.13.5 Instruction Pool of P6 Family Processors

The P6 family Pentium processors have 3 pipelines and each pipeline has 12 stages. The 12 stages generate data-dependency problems and pipeline stalls. The P6 processors have decoupled pipelines and the simple execution stage is replaced by the two decoupled phases, namely, *dispatch/execute unit* and *retire unit*. In the decoupling method, the instructions in the pipelines are executed independently. Therefore, the instructions can be executed out-of-order in the Instruction Pool. Figure 12.55 shows the instruction pool of P6 family processors.

Instruction Fetch/Decode Unit The *instruction fetch (IF)/decode unit* reads the sequence of instructions from the instruction cache and decode them. The pre-fetching is performed in a speculative manner. Actually, IF/Decode unit reads 32 bytes or one cache line per clock from the L1 cache. This unit marks the start and end of the instructions and carries out branch prediction. After that, the IF unit transfers 16 byte to the decode unit

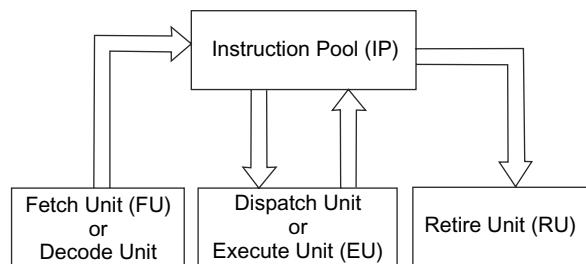


Fig. 12.55 Instruction pool of P6 family processors

Decoder Unit The decoder unit has three parallel decoders. Two decoders are used for simple instructions and one decoder is used for complex instructions. The decoders accept the stream of fetched instructions and decode them. Actually, the decoders translate $\times 86$ instructions into micro-operations (μ -ops). Each micro-operation consists of two logical sources and a logical destination. Simple instructions are translated into single μ -ops and complex instructions are converted into a string of up to 4 μ -ops. The

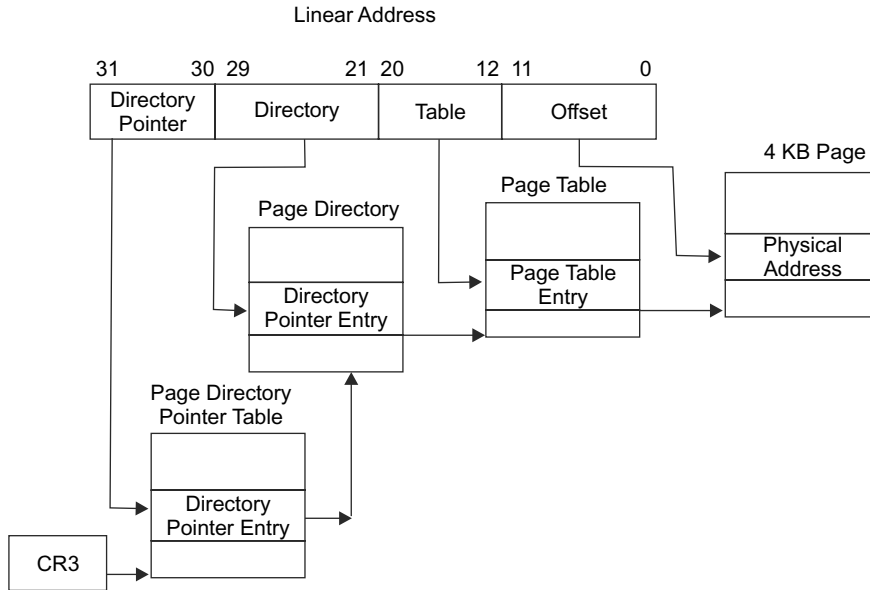


Fig. 12.58 The page-directory pointer, page directory and page-table hierarchy with 4 KB

(iv) The linear address translation has been changed to allow mapping 32-bit linear address into larger physical address space.

Figure 12.58 shows the page-directory pointer, page directory and page table during mapping linear address up to 4 Kbyte pages. $A_{31}-A_{30}$ of linear address are used for directory pointer to locate directory pointer entry in page directory pointer table. $A_{29}-A_{21}$ of linear address are used for directory to indicate directory entry of page directory. The page table entry is located by $A_{20}-A_{12}$ of the linear address and physical address is specified with the help of offset address $A_{11}-A_0$ with respect to page table entry. This paging technique can be used to address up to 2^{20} pages and linear address space of 4 GB or 2^{32} bytes. The 2^{20} pages are computed from the following expression:

$$4 \text{ PDPTE} \times 512 \text{ PDE} \times 512 \text{ PTE} = 2^{20} \text{ pages}$$

where, PDPTE = Page directory pointer table entry

PDE = Page directory entry

PTE = Page table entry

Figure 12.59 shows that the page-directory pointer and page directories can amplify linear address to 4 MByte or 2 MByte pages. It is clear from Fig. 12.59 that $A_{31}-A_{30}$ of linear address are used to locate directory pointer entry in the page-directory pointer table. Directory entry is addressed by $A_{29}-A_{21}$ of linear address and the physical address is located by the offset address of linear address $A_{20}-A_0$ with respect to directory entry. This paging method can be used to map up to 2048 pages and 4 GB linear address space. The computation of 2048 pages is given below:

$$4 \text{ PDPTE} \times 512 \text{ PDE} = 2048 \text{ pages}$$

where, PDPTE = Page directory pointer table entry and PDE = Page directory entry

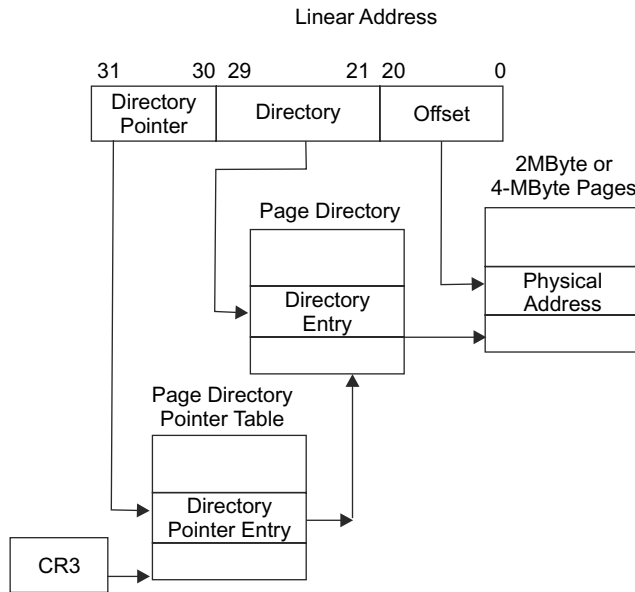


Fig. 12.59 The page-directory pointer and page directories with 2 MByte or 4 MByte pages

12.14 COMPARISON OF PENTIUM AND PENTIUM-PRO PROCESSOR

The comparison between Pentium and Pentium-Pro processor is given Table 12.2.

Table 12.2 Comparison between Pentium and Pentium-Pro processor

<i>Pentium Processor</i>	<i>Pentium-Pro processor</i>
The Pentium processor is the fifth-generation processor 80586 and it is represented by P5. This processor was developed in 1993 by Intel.	The Pentium-Pro processor is the sixth-generation processor and it is represented by the P6 family processors. This processor was developed in 1995 by Intel.
This is a 5 V processor and it was fabricated in 0.8-micron BiCMOS technology	Pentium-Pro processor core operates at 3.1 V to 3.3 V and it is fabricated in a 0.6 μm BiCMOS process.
Pentium runs at a clock frequency of 60 MHz or 66 MHz.	The Pentium-Pro processor's clock speeds are 150, 166, 180 or 200 MHz with a 60 or 66 MHz external bus clock.
Pentium has 3.1 million transistors.	Pentium-Pro consists of 5.5 million transistors.
Pentium has 2 × 8 Kbyte L1 cache, but there is no L2 cache.	Pentium-Pro has 2 × 8 Kbyte L1 cache, and a 256 Kbyte L2 cache.
Pentium has a 32-bit address bus.	Pentium-Pro has a 36-bit address bus.
It has two instruction units.	It has six instruction units.
It has superscalar pipeline. It has two independent integer pipelines and a floating point pipeline.	The Pentium-Pro has a 12-stage decoupled super pipeline architecture which uses an instruction pool. It has speculative execution through register renaming with 40 registers.

12.15 PENTIUM 4 PROCESSOR

The Pentium 4 processor was developed by Intel using Intel Netburst architecture in 2000. It was commonly used in desktop and laptop central processing units (CPUs). It is the 7th generation x 86 micro-architecture, called NetBurst. NetBurst differed from the preceding P6 family (6th generation x 86 micro-architecture) processors by featuring a very deep instruction pipeline to achieve very high clock speeds of up to 3.8 GHz. The Pentium 4 processor's NetBurst architecture incorporates all the features of the previous P6 architecture of Pentium II and Pentium III and some new features are added. The new features of Pentium 4 are given below:

- ◆ Pentium 4 was developed base on NetBurst micro-architecture.
- ◆ It consists of 42 million transistors.
- ◆ Clock speed of Pentium 4 varies from 1.3 GHz to 3.8 GHz.
- ◆ It operates in hyper-pipelined technology and it has a 20-stage pipeline.
- ◆ Pentium 4 has on-die 256 Kb non-blocking, 8-way set associative L2 cache. The L2 cache uses a 256-bit interface that transports data-transfer rates of 48 GB/s at 1.5 GHz.
- ◆ The instruction set of the Pentium 4 processor is compatible with x86 (i386), x 86-64, MMX, SSE, SSE2, and SSE3 instructions. These instructions include 128-bit SIMD integer arithmetic and 128-bit SIMD double-precision floating-point operations.
- ◆ It has 8 KB L1 data cache and an execution trace cache to store up to 12 K decoded micro-operations (μ -ops) in the order of program execution.
- ◆ It supports faster system bus at 400 MHz to 1066 MHz with 3.2 GB/s of bandwidth.
- ◆ The Pentium 4 processor has two arithmetic logic units (ALUs) which are operated at twice the core processor frequency.
- ◆ It is fabricated in 0.18 micron CMOS process.
- ◆ It has advanced dynamic execution.
- ◆ It has enhanced branch prediction.
- ◆ It has a rapid execution engine.
- ◆ It has enhanced floating point/multimedia applications.

In the next section, Pentium 4 NetBurst architecture, hyper-threading technology and SSE instructions are discussed.

12.15.1 Architecture of Pentium 4

The simplified block diagram of internal architecture of the Pentium 4 processor is shown in Fig. 12.60 and the detailed internal architecture of Pentium 4 is depicted in Fig. 12.61. Generally, the Pentium 4 architecture consists of a Bus Interface Unit (BIU), Instruction Fetch and Decoder Unit, Trace Cache (TC), Microcode ROM, Branch Target Buffer (BTB), Branch Prediction, Instruction Translation Look-aside Buffer (ITLB), Execution Unit, and Rapid Execution Module. It is clear from Fig. 12.60 that The Pentium 4 architecture has four different modules such as (i) memory subsystem module, (ii) front-end module, (iii) integer/floating point execution unit, and (iv) out-of-order execution unit. The memory subsystem module contains a Bus Interface Unit (BIU) and L3 cache (optional). The front-end module consists of instruction decoder, Trace Cache (TC), microcode ROM, Branch Target Buffer (BTB) and branch prediction. Integer/Floating point execution unit has the L1 data cache and execution unit. The out-of-order execution unit consists of execution unit and retirement. In this section, the detailed internal architecture of Pentium 4 has been discussed elaborately.

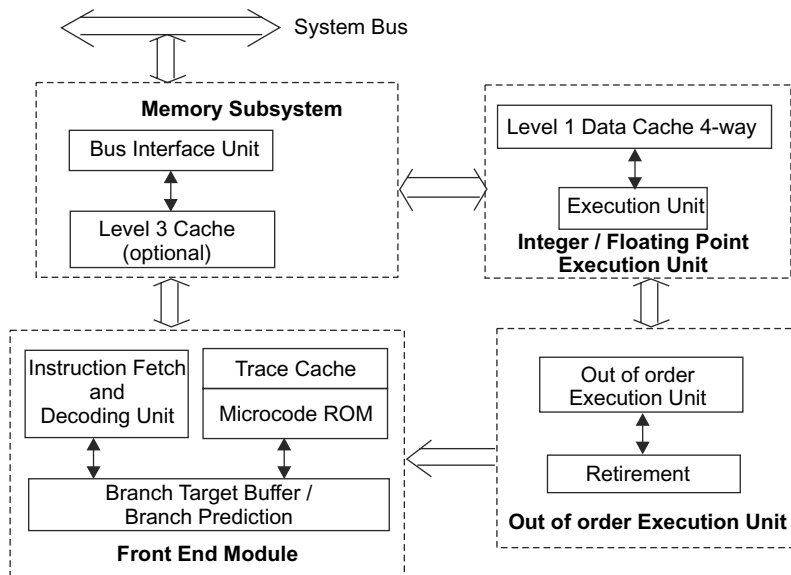


Fig. 12.60 Simplified block diagram of Pentium 4 processor

Bus Interface Unit (BIU) The Bus Interface Unit (BIU) is used to communicate with the system bus, cache bus, L2 cache, L1 data cache and L1 code cache.

Instruction Decoder The instruction decoder is used to decode all instructions of the Pentium 4 processor concurrently and translate them into micro-operations (μ -ops). One instruction decoder decodes one instruction per clock cycle. Simple instructions are translated into one μ -ops, but other instructions are translated into multiple numbers of μ -ops. Usually, a complex instruction requires more than four μ -ops. Therefore, the decoder cannot decode complex instructions and it transfers the task to a Microcode ROM.

Trace Cache (TC) After translation of instructions into micro-operations (μ -ops) by using an instruction decoder, the streams of decoded instructions are fed to an L1 instruction cache, which is known as trace cache. The L1 cache can store only the decoded stream of instructions, which are actually micro-operations (μ -ops). Hence, the speed of execution will be increased significantly. In a Pentium 4 processor, the trace cache can store up to 12 K μ -ops. Normally, the cache assembles the decoded μ -ops in order of sequence, called *traces*. A single trace contains many *trace lines* and each trace line has six μ -ops.

Microcode ROM As complex instructions perform string and interrupt operations, etc., the trace cache transfers the control operation of complex instructions to a micro-code ROM. Then microcode ROM is used to generate the micro-operations (μ -ops) of complex instructions. After the micro-operations (μ -ops) are issued by the microcode ROM, the control again returns back to the trace cache. Subsequently, μ -ops of complex instructions delivered by the trace cache as well as the microcode ROM will be buffered in a queue in order of sequence. Then the μ -ops are fed to the execution unit for execution.

Branch Prediction The branch prediction logic unit predicts the memory locations from where the next instruction will be fetched. Usually, the predictions are performed using the past information of the program execution. The sixth-generation Pentium processors use simple branching strategy. If the processors

come across a branch instruction, the branch condition will be evaluated by complex mathematical and logical computations which require some time and the processors have to wait till the branch condition is completely evaluated. To reduce the wait time, Pentium P4 processors use the speculative execution strategy.

The branch prediction can be done by static prediction and dynamic prediction. The static prediction is fast and simple, as it does not require any look-up tables or calculations. The dynamic predictions use two types of tables, namely, the Branch History Table (BHT) and the Branch Target Buffer (BTB), to record information about outcomes of branches, which have already been executed.

Instruction Translation Look aside Buffer (ITLB) When a trace cache miss occurs, instruction bytes are required to be fetched from the L2 cache. These instruction bytes are decoded into micro-operations (μ -ops) to be placed in the trace cache. If the Instruction Translation Look-aside Buffer (ITLB) receives any request from the trace cache to send new instructions, ITLB translates the next instruction pointer address to a physical address. After that, a request will be sent to the L2 cache and instruction bytes will be available from the L2 cache. These instruction bytes are stored in streaming buffers until they are decoded. As there are two logical processors in Pentium 4, there are two ITLBs. Each logical processor has its own ITLB and a set of two 64-byte streaming buffers, which store the instruction bytes. After that, these instruction bytes are sent to the instruction decode stage.

Execution Unit A superscalar processor has multiple parallel execution units, which can process the instructions simultaneously. Actually, the executions of instructions are sequentially dependant on

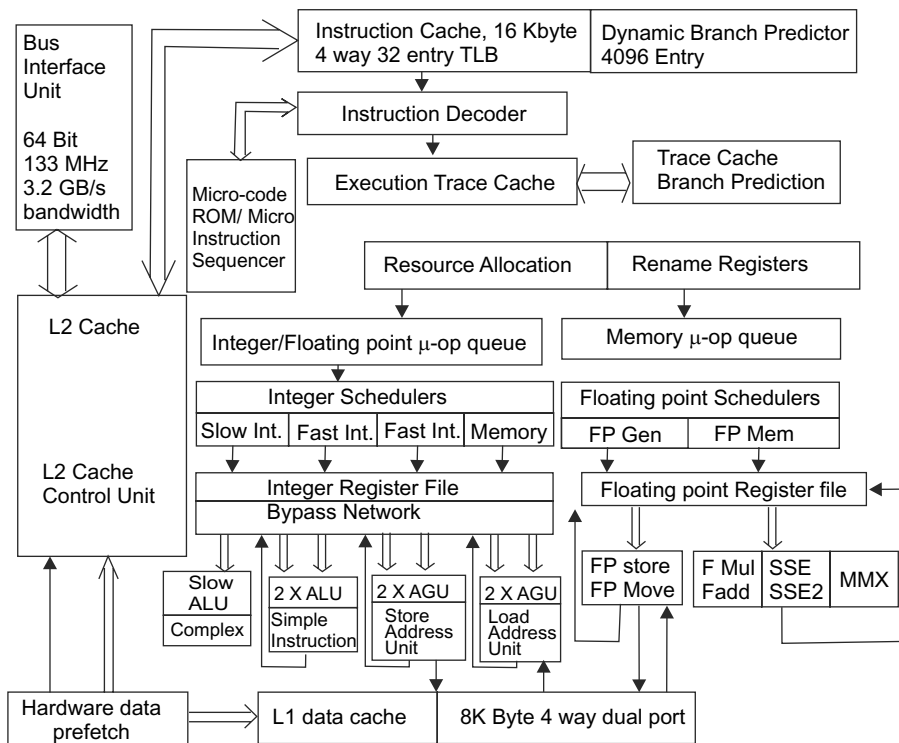


Fig. 12.61 Detail internal architecture of Pentium 4 processor

each other. Therefore, the result of one instruction depends on the result of its preceding instructions and the processor cannot execute instructions concurrently. The concept of *out-of-order execution* has been developed to solve the problem of parallel execution of instructions. The *out-of-order execution* consists of the allocation, register renaming, scheduling, and execution functions.

✓ **Allocator** The *allocator accepts* micro-operations (μ -ops) from the μ -ops queue and allocates the key machine buffers to execute micro-operations. The *allocator* has 126 re-order buffer entries, 128 integer and 128 floating-point physical registers, 48 load and 24 store buffer entries. Since there are two logical processors in Pentium 4, each logical processor can use at most half the entries that is 63 re-order buffer entries, 24 buffers, and 12 store buffer entries.

✓ **Register Rename** The *register rename* logic is used to rename the registers of Intel Architecture 32-bit Pentium processors onto the machine's physical registers. The register rename is possible in the 8 general-use IA-32 integer registers to be dynamically expanded to 128 physical registers. A Register Alias Table (RAT) is used to track the current status of registers and it is also used to inform the next instructions from where to get their input operands. As there are two logical processors in Pentium 4, there should be two RATs—one for each logical processor. Usually, the register renaming process is performed in parallel to the allocator.

✓ **Instruction Schedulers** The instruction scheduler is used to schedule micro-operations (μ -ops) to an appropriate execution unit. There are five instruction schedulers to schedule micro-operations in different execution units. Therefore, multiple numbers of μ -ops can be distributed in each clock cycle. Any micro-operation can be executed only whenever the operands of instruction are available and the specific execution unit must be available for execution of μ -ops. In this way, the scheduling strategy distributes all μ -ops whenever the operands are ready and the execution units are available for execution. Each scheduler should have its own scheduler queue of eight to twelve entries from which the scheduler selects μ -ops to transmit to the execution units.

Rapid Execution Module

There are two ALUs (Arithmetic Logic Unit) and two AGUs (Address Generation Unit) in a Pentium 4 processor. The ALU and AGU units operate at twice the processor speed. For example, if the processor works at 1.4 GHz, the ALUs can operate at 2.8 GHz. Hence, twice the numbers of instructions are executed per clock cycle. All integer calculations such as addition, subtraction, multiplication, division and logical operations are performed in the arithmetic and logic unit. AGUs are used to resolve indirect mode of memory addressing. The ALUs and AGUs are very useful for high-speed processing.

Memory Subsystem

The virtual memory and paging technique are used in memory-subsystem representation. The linear address space can be mapped into the processor's physical address space, either directly or using a paging technique. In direct mapping, paging is disabled and each linear address represents a physical address. Then linear address bits are sent out on the processor's address lines without translation. When the paging mechanism becomes enabled, the linear address space is divided into pages. Actually, pages are used to map into virtual memory. After that the pages of virtual memory are mapped into physical memory. The paging mechanism is transparent to programmers for any program. The Pentium 4 processor supports Intel Architecture 32-bit paging mechanism. The Page Address Extension (PAE) is used to address physical address space greater than 4 Gbytes. The Page Size Extension (PSE) is used to map a linear address to physical address in 4 M bytes pages.

12.15.2 Hyper-Threading (HT) Technology

Traditional P6 micro-architecture approaches to processor design have focused on higher clock speeds, super pipelining, Instruction-Level Parallelism (ILP) and caches. *Super pipelining* is a way to achieve higher clock speeds by having finer granularities. By using this technique, it is possible to execute more and more

instructions within a second. As there are far more instructions in-fly, handling such events like cache misses, interrupts and branch-miss prediction will be costly. The *instruction-level parallelism* technique is used to increase the number of instructions executed within a cycle. The superscalar processor has multiple parallel executions units for different instruction sets. The main problem is to find enough instructions for execution. Therefore, the out-of-order execution has been accepted where this new technique has additional burden on the system design. A *cache hierarchy* is used to reduce the latency originating from memory accesses where smaller and faster units are located closer to the processor than the bigger and slower ones. Although fast memories are used in the system, there will be always some events when instructions and data-cache misses occur.

For the last few decades, Internet and telecommunication industries have had an unprecedented growth. To fulfill the requirements of up coming telecommunication industries, the traditional micro-architecture is not sufficient for processor design. Therefore, processor designers are looking for another architecture where the ratio between cost and gain is more reasonable. Hyper-threading (HT) technology is one solution. This technology was first implemented in Pentium® 4 Xeon processor in 2002. The features of hyper-threading technology are given below:

- ◆ HT makes a single physical processor appear as multiple logical processors.
- ◆ Each logical processor has its own architecture state where a set of single execution units are shared between logical processors.
- ◆ HT allows a single processor to fetch and execute two separate code streams simultaneously.
- ◆ In most of the applications, the physical unit is shared by two logical units.

It is well known to us that each process has a context in which all the information related with the current state of execution of the process are described. In any process, the contents of the CPU registers, the program counter, the flag register are used as context. Each process should have at least one thread and sometimes more than one thread is present in a process. Each threads has its own local context. Sometimes the context of a process is shared by the other threads in that process. The common features of threads are as follows:

- ◆ The threads can be independent in a process.
- ◆ The threads can be bunched together into a process.
- ◆ The threads may be simple in structure and can be used to increase the speed of operation of the process.

Many processes may run on different processors in a multiprocessor system. Different threads of the same process can be shared and run on different processors. Therefore, multiple threads improve the performance of a multiprocessor system. In Intel's hyper-threading technology, the concepts of simultaneous multi-threading to the Intel architecture have been introduced. The hyper-threading technology makes a single processor appear as two logical processors; the physical execution resources are shared and the Architecture State (AS) is duplicated for two logical processors. HT means that the operating system and user programs can schedule processes or threads to logical processors as they would on multiple physical processors. On the other hand, micro-architecture perspective states that both processors execute simultaneously on shared resources.

Presently, the trend is to run multi-threaded applications on multi-processor systems. The most common multi-threaded applications are Symmetric Multi-Processor (SMP) and Chip Multi-Processing (CMP). Although a symmetric multi-processor has better performance, the die-size is still significantly large which causes higher costs and power consumptions. The chip multi-processing puts two processors on a single die. Each processor has a full set of execution and architectural resources. The processors can share an on-chip cache. CMP is orthogonal to conventional multiprocessor systems. The cost of a CMP processor is still high as the die size is larger than the size of a single core-chip and power consumption is also high.

A single processor with multi-processing/multi-threading or CMP can be supported in different ways such as Time-Sliced Multi-threading (TSM), Switch-on Event multi-threading (SEM) and Simultaneous Multi-Threading (SM).

Time-Sliced Multi-threading (TSM)

In time-sliced multi-threading, the processor switches from one task to another after a fixed amount of time has passed. This technique is also called *real multi-tasking*. As there is only one processor, there will be always some loss of execution cycles in the time-slice multi-threading. But each thread must be get the attention of the processor whenever its turn comes. When there is a cache miss, the processor will switch to another thread automatically.

Switch-on Event Multi-threading (SEM)

The processor could be designed to switch to another task whenever a cache miss occurs.

Simultaneous Multi-threading (SM)

In simultaneous multi-threading or hyper-threading, multiple threads may be executed on a single processor without switching. When multiple threads are executed simultaneously, it leads to better use of resources. Actually, hyper-threading (HT) technology brings the SM into life in Intel architecture.

Figure 12.62 shows a system in which there are two physical processors without hyper-threading technology. On the other hand, Fig. 12.63 shows a two-processor system where each unit is capable of hyper-threading technology. Therefore, the system in Fig. 12.63 can be implemented as 4 CPU systems. This scheme has the same performance gains but the cost increase is only 5% due to about 5% die-area size increase.

Architecture State (AS)

Hyper-threading technology was introduced on the Intel Pentium 4 Xeon™ processor. In this processor, there are two logical processors in a single physical processor. Each logical processor has a complete set of the architecture state. The architecture state has the following registers:

- ◆ Registers including the general-purpose registers
- ◆ Control register
- ◆ Advanced Programmable Interrupt Controller (APIC) registers
- ◆ Machine state registers

Since two architecture states are present in a single physical processor, the processor acts as two processors with respect to software perspective. There are three types of resources in hyper-threading technology such as *replicated* resources, *shared* resources, and *shared/replicated* resources.

Replicated Resources

Each processor has general-purpose registers, control registers, flags, time stamp counters, and APIC registers. The content of these registers are used as replicated resources.

✓ **Shared Resources** Memory and range registers can be independently read/write. Therefore, memory, range registers and data buses are used as shared resources.

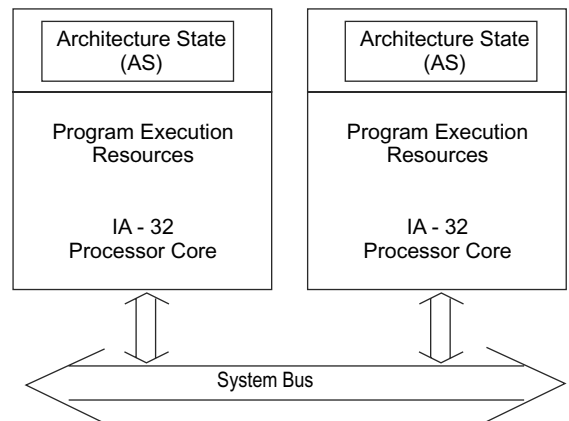


Fig. 12.62 Processors without hyper-threading technology

✓ **Shared/Replicated Resources** The caches and queues in the hyper-threading pipeline can be shared or not shared according to the situation.

Logical processors share resources on the physical processor, such as caches, execution units, branch predictors, control logic, and data buses. Each logical processor has its own advanced programmable interrupt controller. Usually, interrupts are sent to a specific logical processor for proper handling.

In most of the applications, multiple numbers of processes or threads may be executed in parallel. This kind of parallel execution is called *thread-level parallelism*, and these give better performance in online applications such as the server system and Internet applications.

When the current executing process is completed in time-sliced multi-threading, its context will be saved into the memory. Whenever the process starts execution again, the context of the process is again restored to exactly the same state. Therefore, this process consists of the following operations:

- ◆ Save the context of the currently executing process after the time slice is over.
- ◆ Flush the CPU of the same process.
- ◆ Load the context of the new next process, known as *context switch*.

When the process consists of m number of threads, the total time for context switching will be m times that of a single thread context-switching time. Hence, context switching requires a number of CPU cycles. If multiple numbers of threads are present, system performance will be improved. But a large number of threads consume more time in context switching.

To improve the system performances, the following methods are used:

- ◆ Reduce the number of context switches.
- ◆ Provide more CPU execution time to each process.
- ◆ Execute more than one process at the same time by increasing the number of CPUs.
- ◆ In multiple number of processor systems, the scheduler can schedule two processes to two different CPUs for execution simultaneously. Hence, the process will not be waiting for a long duration to get executed.

In hyper-threading, the concept of simultaneous multi-threading is used. Hence, there is an improvement in the Intel micro-architecture. With increasing the cost of less than 5% in the die area, the system performance is increased by about 25 per cent.

The major advantage of this architecture is appropriate resource sharing of each shared resource. The most commonly used sharing strategies are partitioned resources, threshold sharing, and full sharing. Usually, the sharing strategy is selected based on the traffic pattern, size of the resource, potential deadlock probabilities and other considerations.

To share the resources, there should be one copy of the architecture state for each logical processor, and the logical processors must share a single set of physical resources. Consequently, the operating systems and user programs can schedule processes or threads to logical processors as these processors behave as conventional physical processors in a multi-processor system. According to micro-architecture perspective, instructions from logical processors will carry on execution simultaneously after sharing resources.

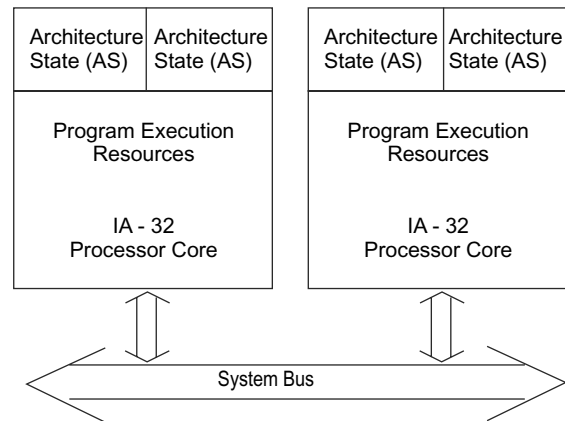


Fig. 12.63 Processors with hyper-threading technology

12.15.3 Streaming SIMD Extension (SSE), Extension 2 (SSE2) and Extension 3 (SSE3) Instructions

When the MMX instructions are extended incorporating floating-point instructions, the extended instructions are called *Streaming SIMD Extensions (SSE)* instructions. Initially, SSE instructions are used in Pentium III and then the SSE instruction set has further been enhanced in Pentium 4. The features of SSE instructions are given below:

Streaming SIMD Extensions (SSE) instruction

- ◆ SSE instructions are SIMD instructions for single-precision floating-point numbers.
- ◆ SSE instructions can be operate on four 32-bit floating points in parallel.
- ◆ A set of eight new SIMD floating-point registers are specifically defined for SSE. The SSE registers are named XMM0 through XMM7.
- ◆ Each register for SSE is 128 bits long allowing 4×32 bit numbers to be handled in parallel.
- ◆ As different registers have been allocated, it is possible to execute both fixed-point and floating-point operations simultaneously.
- ◆ The SSE instructions can execute non-SIMD floating-point and SIMD floating-point instructions concurrently.
- ◆ The SSE instructions can operate on packed data or on scalar data and increase the speed of manipulation of 128-bit SIMD integer operations.
- ◆ The SSE instructions can be grouped as data-transfer instructions, data-type conversion instructions, arithmetic, logic and comparison group of instructions, jump or branch group of instruction, data management and ordering instructions, shuffle instructions, cache-ability instructions and state-management instructions.

Streaming SIMD Extension 2 (SSE2) Instructions

In Pentium 4, the pipeline depth is increased significantly, and execution rate for all instructions are improved. About 144 new instructions are added with SSE instructions set which allow up to 4 Internet/multimedia based operations in the Pentium 4 processor and these will be executed simultaneously. These new instructions and the other improvements are called *Streaming SIMD Extension 2 (SSE2) instructions*. The SSE2 instructions support new data types, namely, double-precision floating points. The Intel NetBurst micro-architecture has extended the SIMD capabilities after adding SSE 2.

Streaming SIMD Extension 3 (SSE3) Instructions

The Streaming SIMD Extensions 3 (SSE3) instructions have been introduced in the next-generation Pentium 4 processor. This version was developed by Intel in 2004, when the latest version of Pentium 4, the Prescott was released. Actually, the SSE 2 instruction set was extended to SSE3 after adding 13 additional SIMD instructions over SSE2. The SSE3 instructions are used for the following operations:

- ◆ Complex arithmetic operations
- ◆ Floating-point-to-integer conversion
- ◆ Video encoding
- ◆ Thread synchronization
- ◆ SIMD floating-point operations using array-of-structures format

12.16 COMPARISON OF PENTIUM III AND PENTIUM 4 PROCESSORS

The comparison between Pentium III and Pentium 4 processor is given Table 12.3.

Table 12.3 Comparison between Pentium III and Pentium 4 processor

<i>Pentium III Processor</i>	<i>Pentium IV Processor</i>
Pentium III processor is a sixth-generation processor and it is represented by the P6 family processor. This processor was developed in 1999 by Intel.	Pentium 4 processor is the seventh generation processor. This processor was developed in 2000 by Intel.
Pentium III processor was manufactured based on the P6 micro-architecture.	Pentium 4 was manufactured base on NetBurst micro-architecture.
This processor core contains 9.5 to 28 million transistors.	This processor core contains 42 million transistors.
Pentium III operates at 2.0 V to 1.45 V.	Pentium 4 operates at 1.40 V to 1.25 V.
This processor is available at operating frequencies of 450 MHz to 1.4 GHz.	Clock speed of Pentium 4 varies from 1.3 GHz to 3.8 GHz.
It supports faster system bus at 100 MHz to 133 MHz.	It supports faster system bus at 400 MHz to 1066 MHz with 3.2 GB/s of bandwidth.
Pentium III Katmai was fabricated in a 0.25 μm CMOS process.	It is fabricated in 0.18 micron CMOS process.
Pentium III has 32 Kbyte L1 cache—16 Kb for instruction and 16 Kb for data.	Pentium 4 has 12 Kb L1 cache for code and 8 Kb for data.
It has on-die 0 to 512 Kb L2 cache	It has on-die 512 Kb L2 cache.
This processor has SSE (streaming SIMD) extensions. It can support dynamic execution technology. 70 new instructions are added to the Pentium III for multimedia and advanced image processing applications.	The instruction set of Pentium 4 processor is compatible with x86 (i386), x86-64, MMX, SSE, SSE2, and SSE3 instructions. These instructions include 128-bit SIMD integer arithmetic and 128-bit SIMD double-precision floating-point operations.

12.17 RISC PROCESSORS

In the previous sections, CISC processors were discussed elaborately. *CISC* is pronounced *sisk*, and it stands for Complex Instruction Set Computer. Most desktop computers and laptops use CPUs based on this architecture. Usually, CISC ICs have a large amount of different and complex instructions. The philosophy behind CISC processors is that hardware is always faster than software. Therefore, one should make a powerful instruction set, which provides programmers with assembly instructions to do a lot of tasks with very short programs. The example of CSIC microprocessors are Intel 80486, Pentium, Pentium Pro, Pentium II, Celeron and Pentium III, etc. The common features of CISC processors are as follows:

Features of CISC Processors

- ◆ A CISC processor supports extensive complex instructions.
- ◆ It has complex and efficient machine instructions.
- ◆ It supports micro-encoding of the machine instructions.

- ◆ It has extensive addressing capabilities of memory operations.
- ◆ The CISC processor has less number of very useful CPU registers.
- ◆ The CISC processors can operate at relatively slow speed.

With incorporating the more and more complex instructions in a CISC, more and more sophisticated processors were developed, manufactured and used as CPUs of personal computers (PCs) for marketing. As a result, the processor die size was increased to accommodate the large numbers of microcode of the complex instructions. Due to large die size, CISC consumed more silicon and the chip size was increased. Consequently, the power consumption also increased, more heat sink was required for better cooling arrangement and cost of the system became high.

When a processor supports a set of simple instructions, it does not require complex decoding and the design process of the processor becomes very simple. Accordingly, costs of the system as well as power consumption are reduced significantly. The execution of the simple instructions is also very fast. In the mid-1970s, John Cocke at IBM research demonstrated that on microcoded implementations of CISC architectures, complex operations using complex instructions tended to be slower than a sequence of simpler operations doing the same thing. Processor designers also realized that in many cases, most of the complex instructions of a CISC processor's instruction set are not actually used. In a typical CISC processor program, about 20% of the instruction set will perform about 80% operations of the program. Sometimes the execution of simple instructions is quicker than single complex machine instruction. Actually, the complex instructions will take a long time to decode, but simple instructions take less time to decode and execute faster. Therefore, simple instructions are most commonly used in programs and complex instructions are very rarely used.

Subsequently, *RISC* chips evolved around the mid-1980s as a reaction at CISC chips. RISC is pronounced *risk*, and it stands for *Reduced Instruction Set Computer*. The design philosophy behind RISC is that almost no one uses complex assembly-language instructions as used by CISC, and programmers mostly use compilers which never use complex instructions. RISC utilizes a set of fewer, simpler and faster instructions, rather than a more specialized set of instructions. However, more instructions are needed to accomplish a task in RISC. Generally, each instruction is executed within a single clock after it is fetched and decoded. To achieve this, RISC reduces decode and execute logic replacing microcode by hardwired logic gates and uses instruction pipelining techniques extensively. In a CISC, a lot of disc space is consumed by microcodes. As microcode is replaced by hardwired logic in RISC, the size of RISC processors becomes smaller than CISC and it consume less power.

The first attempt was taken to make a chip-based RISC CPU at IBM in 1975. After that, UC Berkeley and Stanford started work to design and develop RISC processors. After a long research, the IBM 801 was eventually developed in a single-chip form in 1981. After that Stanford MIPS (Microprocessor without interlocking Pipeline Stages), Berkeley RISC-I and RISC-II processors were developed. The examples of other microprocessors are DECs Alpha, power PC 601, 602, 603 and Ultra SPARC, etc. The common features of RISC processors are given below:

Features of RISC Processors

- ✓ **Simple Instruction Set** In a RISC processor, the instruction set consists of simple and basic instructions. The complex instructions can be composed using simple and basic instructions.
- ✓ **Reduction of Instruction Set** Less numbers of instructions are used to simplify instruction decoding.
- ✓ **Elimination of Microcoding** In RISC, microcode is replaced by hardwired logic gates. Hence all execution units are hardwired.
- ✓ **Same Length Instruction** Each instruction is of the same length.

- ✓ **Single Machine-Cycle Instructions** The execution of instructions complete in one machine cycle, which allows the processor to handle several instructions at the same time.
- ✓ **Pipelining** Usually, massive pipelining is used in a RISC processor. Due to pipelined instruction decoding and executing, more operations are performed in parallel. Therefore, the speed of RISC processors is increased.
- ✓ **Very Few Addressing Modes** RISC processors have very few addressing modes and it supports few formats.
- ✓ **Load and Store Architecture** In RISC processors, only the load and store instructions are used to access memory. Other instructions of the processor work with the internal registers of the processor.
- ✓ **Large Number of Registers** The RISC processors have a larger number of registers to reduce the interactions between the processor and memory.

The performance of a computer is computed by the following equation:

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Time}}{\text{Cycle}} \times \frac{\text{Cycle}}{\text{Instruction}} \times \frac{\text{Instruction}}{\text{Program}}$$

To get better performance, the CISC processors are designed to minimize the number of instructions per program sacrificing the number of cycles per instruction, but RISC processors are designed reducing the cycles per instruction at the cost of instructions per program. The comparison between RISC and CISC processors are given in Table 12.4.

Table 12.4 Comparison between RISC and CISC

<i>RISC Processor</i>	<i>CISC Processor</i>
In RISC, emphasis is given on software.	In CISC, emphasis is given on hardware.
Reduced and simple instructions are used.	Large number of instructions including complex instructions are used.
Each instruction is executed within a single clock.	Multi-clocks are required to execute complex instructions.
For register-to-register data movement, LOAD and STORE instructions are independently operated.	For memory-to-memory data movement, LOAD and STORE instructions are incorporated in instructions.
There are large code sizes for a program, but the number of cycles per instruction is low.	It has small code sizes for a program, but the number of cycles per instruction is high.
RISC spends more transistors on memory registers.	Transistors are used for storing complex instructions.

12.17.1 Architecture of RISC PROCESSOR: Power PC 601

The IBM POWER (Performance Optimized with Enhance RISC) architecture is used as the base of Power PC architecture and RISC single-chip vector processor is the base of the 601 microprocessor. The Power PC 601 microprocessor is the first implementation of Power PC architecture and it was developed jointly by IBM and Motorola in 1993. The Power PC 601 is a 32-bit Power PC architecture processor. This processor has three instruction units, namely, Branch Processor Unit (BPU), Fixed-Point Unit (FXU), and Floating-Point Unit (FPU). These three execution units are used to support superscalar dispatch of instructions. The simplified block diagram of a Power PC 601 is shown in Fig. 12.64. The branch processor unit is a subpart of the main instruction unit and is used for fetching and prefetching instructions. A maximum of 8 instructions can be stored in the instruction fetch queue. The instruction unit can fetch instructions in the branch processor unit, fixed-point unit and, floating-point unit simultaneously and execute instructions in parallel on each unit.

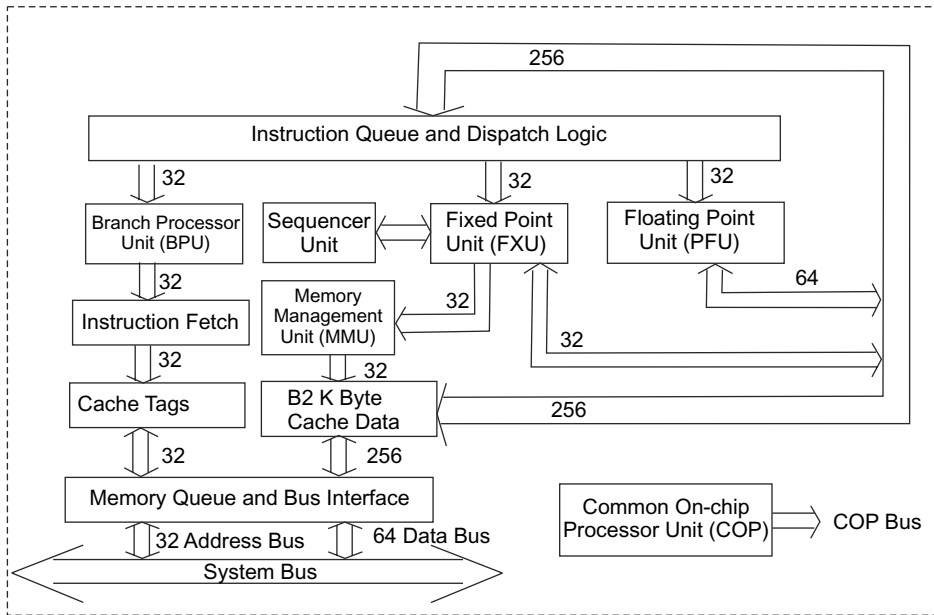


Fig. 12.64 Architecture of Power PC 601

The RISC processor pipeline operates in such a way that some operations may be carried out in parallel, although the stages in the pipeline are different. Though different processors have different number of steps to execute instructions, there are five steps that will be required when a processor executes an instruction. Figure 12.65 shows the pipeline structure of Power PC 601.

Step 1 Instruction fetch phase: Read the instruction from the memory or the prefetch queue.

Step 2 Decode phase: Read register and decode the instruction.

Step 3 Operand fetch phase: Fetch the operand whenever required for the instruction.

Step 4 Execute phase: Execute the instruction.

Step 5 Write-back phase: Write the result into the register.

Branch Instructions

Fetch	Dispatch, Decode, execute and Predict
-------	---------------------------------------

Integer Instructions

Fetch	Dispatch, Decode	Execute	Write Back
-------	------------------	---------	------------

Load/store Instructions

Fetch	Dispatch, Decode	Address Generation Cache	Write Back
-------	------------------	--------------------------	------------

Floating-point Instructions

Fetch	Dispatch	Decode	Execute 1	Execute 2	Write Back
-------	----------	--------	-----------	-----------	------------

Fig. 12.65 Pipeline structure of Power PC 601

The Power PC 601 has a 32 Kbyte unified L1 cache. The instruction queue and dispatch logic unit buffer instructions from 32 Kbyte unified cache and dispatch up to three instructions on each cycle: one each to the BPU, the FXU and the FPU. The fixed point unit communicates with the sequencer unit to control most frequently used instructions. The FXU interfaces with Memory Management Unit (MMU) for cache accesses. The 32 Kbyte unified cache provides a 256-bit interface with instruction queue and memory queue, a 64-bit interface to the FPU and a 32-bit interface to the FXU. The Power PC 601 has a 32-bit address bus and a 64-bit data bus. This processor supports a Common On-chip Processor (COP) unit through a COP bus and an asynchronous serial port for debugging and test features.

12.17.2 MIPS

MIPS stands for ‘Microprocessor without Interlocked Pipeline Stages’. MIPS is a RISC architecture processor. The MIPS processor has 32 registers and each register is 32 bits wide. The instruction set of a MIPS processor consists 111 instructions such as 15 load instructions, 10 store instructions, 21 arithmetic instructions, 8 logic instructions, 12 comparison instructions, 8 bit-manipulation instructions, 8 move instructions, 25 branch/jump instructions and 4 miscellaneous instructions. The addressing modes as well as instruction set of RISC microprocessors are simpler and much less in numbers compared to the CISC microprocessors. All arithmetic and logical instructions operate in register *mode* of addressing and the operands are stored in the set of 32-bit registers. The load and store instructions are used as data-transfer instructions to store data in the register from memory and vice versa. The *base displacement* addressing mode is also used for data-transfer instructions. In this case, the effective address is generated by addition of the content of a base register and a displacement which already exists in the instruction.

12.17.3 Sun Ultra Sparc

The SPARC (Scalable Processor Architecture) is a unique RISC instruction set architecture. It was developed by Sun Microsystems and introduced in mid-1987. This processor consists of an integer unit, a floating-point unit, and an optional co-processor. Usually, the SPARC processor contains about 160 general-purpose registers but only 32 of them are immediately visible to software at any point of time. Among the 32 registers, eight registers are a set of global registers and the other 24 registers are local registers and are used as the stack of registers. These 24 registers form a register window. During function call and return, this window is moved up and down the register stack. Each window consists of 8 local registers and shares 8 registers with each of the adjacent windows. Generally, the shared registers are used for passing function parameters and returning values, but the local registers are used for storing local values across function calls. The different versions of Sun Ultra SPARC microprocessors are Ultra SPARC T1, Ultra SPARC T2, SPARC V8, SPARC V9, SPARC64 VI, SPARC64 VII, and SPARC T3, etc. The common features of SPARC architecture are given below:

- ◆ The SPARC processor has a 14-stage nonstalling pipeline.
- ◆ It has six execution units which consist of two integer units, two floating-point units, one LOAD/STORE unit and one address generation unit.
- ◆ This processor has a large number of buffers.
- ◆ As there is only one LOAD/STORE unit, it dispatches one instruction at a time.
- ◆ The ULTRA SPARC 3 processor has a 32 KB L1 code cache, a 64 KB L1 data cache, a 1 MB on chip L2 cache, a 2 KB prefetch cache and a 2 KB write cache.
- ◆ The SPARC system operates at low speed compared to other processors.
- ◆ ULTRA SPARC supports a pipelined floating-point processor:

This processor supports multimedia instructions just like Pentium MMX. Hence, this can be used for multimedia and image-processing applications.

12.18 CORE PROCESSOR

The Chip-level Multiprocessor (CMP) or multi-core CPU merges two or more independent *cores* into a single package integrated circuit, called a *die*. A processor with two or more cores on a single die is called a *monolithic* processor. A multi-core microprocessor provides multiprocessing in a single physical package.

In this processor, each ‘core’ independently implements superscalar execution, pipelining, and multithreading.

A dual-core processor is a single chip that contains two distinct processors or cores in the same integrated circuit. Figure 12.66 shows an Intel Core 2 dual-core processor, with CPU-local Level 1 caches, and a shared, on-die Level 2 cache. This processor is introduced in 2006. The processor performance depends on the core and front-side bus clock frequency and amount of second-level cache. Core 2 Duo processors typically use the full L2 cache of 2, 3, 4 or 6 MB available in the chip. Core 2 Duo is widely used in embedded processors, network processors and digital signal processors, desktop computers, mobile Core 2 processor and in GPUs.

A *quad-core* processor contains four cores and it is represented by Core 2 Quad which is introduced by Intel in 2007. Actually, Core 2 Quad processors consist of two Core 2 Duo dies in a single die. Therefore, the performance of a Core 2 Quad is increased by two times from dual-core processors at the same clock frequency.

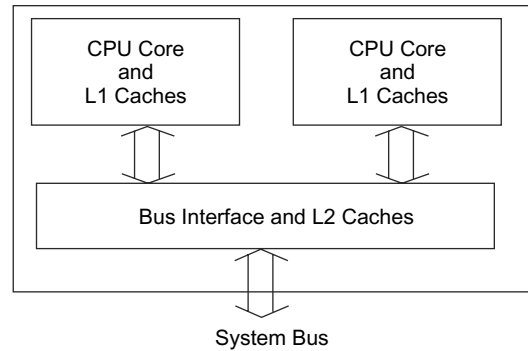


Fig. 12.66 *Intel Core 2 dual processor*

SUMMARY

- This chapter starts with an introduction of the Pentium microprocessor. The architecture, register set, operating modes—protected mode, real-address mode, system management mode (SMM) and virtual-8086 mode, addressing modes, physical address, linear address and logical address, bus interfacing and memory management—of Pentium processors have been discussed elaborately. The pin diagram of Pentium microprocessors and function of some specified pins are incorporated in this chapter.
- The function of cache memory, cache organization, cache consistency and bus snooping, MESI (modified, exclusive, shared, invalid) Protocol, L2 caches of Pentium and other higher version processors are discussed in detail.
- In this chapter, the features of the Pentium MMX processor are explained briefly. The features of advanced Pentium processors such as Pentium Pro, Pentium II and Pentium III: P6 family processors are included. The internal architecture, instruction pool, 36-bit address bus and paging mechanism of P6 family processors are elaborately discussed.
- Intel NetBurst micro-architecture of Pentium 4 supports some special features such as dynamic execution, trace cache, rapid execution engine and SSE instructions. The internal architecture of Pentium 4 processor, Hyper-Threading (HT) technology, Streaming SIMD Extension (SSE), Extension 2 (SSE2) and Extension 3 (SSE3) instructions are explained in detail.
- In this chapter, the features of CISC and RISC processors are presented. The comparison between CISC and RISC processors, internal architecture of RISC processors such as Power PC 601, MIPS and SUN Ultra SPARC and Core processors are also incorporated in this chapter.

MULTIPLE-CHOICE QUESTIONS

- 12.1 The Pentium processor has
 (a) 32-bit address bus and 64-bit data bus
 (b) 32-bit address bus and 32-bit data bus
 (c) 32-bit address bus and 16-bit data bus
 (d) 16-bit address bus and 24-bit data bus
- 12.2 The Pentium processor has
 (a) NetBurst architecture
 (b) Superscalar super-pipelined architecture
 (c) P6 architecture
 (d) 64-bit core architecture
- 12.3 The Pentium processor consists of
 (a) two independent integer pipelines and a floating-point pipeline
 (b) one integer pipeline and a floating-point pipeline
 (c) two integer pipelines
 (d) one floating-point pipeline
- 12.4 The Pentium processor has
 (a) six-stage pipelines
 (b) four-stage pipelines
 (c) five-stage pipelines
 (d) three-stage pipelines
- 12.5 The Pentium processor core consists of
 (a) 21 million transistors
 (b) 7.5 million transistors
 (c) 42 million transistors
 (d) 3.1 million transistors
- 12.6 The Pentium MMX processor has
 (a) 57 MMX instructions
 (b) 56 MMX instructions
 (c) 55 MMX instructions
 (d) 54 MMX instructions
- 12.7 SSE2 instructions are compatible with
 (a) Pentium processor
 (b) Pentium Pro processor
 (c) Pentium 4 processor
 (d) Pentium II processor
- 12.8 The P6 family Pentium processor has
 (a) 36-bit address bus
 (b) 32-bit address bus
 (c) 24-bit address bus
 (d) 20-bit address bus
- 12.9 The Pentium MMX processor has
 (a) seven MMX registers MM1 to MM7
 (b) eight MMX registers MM0 to MM7
 (c) six MMX registers MM1 to MM6
 (d) five MMX registers MM0 to MM4
- 12.10 SSE instructions use
 (a) 8 128-bit registers XMM0 to XMM7
 (b) 6 128-bit registers XMM0 to XMM5
 (c) 5 128-bit registers XMM0 to XMM4
 (d) 7 128-bit registers XMM1 to XMM7
- 12.11 In Hyper-Threading (HT) technology,
 (a) a single processor appears as one logical processors
 (b) a single processor appears as four logical processors
 (c) a single processor appears as three logical processors
 (d) a single processor appears as two logical processors
- 12.12 The _____ is a CISC processor.
 (a) IBM 801 (b) SUN SPARC
 (c) Pentium 4 (d) Power PC 601
- 12.13 The _____ is a RISC processor.
 (a) R600 (b) Pentium
 (c) Pentium III (d) Pentium MMX and Celeron
- 12.14 Pentium 4 operates at
 (a) 5 V (b) 1.25–1.40 V
 (c) 2.5 V (d) 1.6 V
- 12.15 The Pentium 4 has a
 (a) 20-stage pipeline
 (b) 14-stage pipeline
 (c) 10-stage pipeline
 (d) 6-stage pipeline
- 12.16 The front side bus speed of Pentium 4 is
 (a) 400 MHz (b) 50 MHz
 (c) 60 MHz (d) 133 MHz

SHORT-ANSWER-TYPE QUESTIONS

- 12.1 Write the length of address and data bus of the following processors:
(i) Pentium (ii) Power PC 601 (iii) Pentium 4 (iv) Pentium Pro and Pentium III
- 12.2 What are the different registers of the Pentium microprocessor?
- 12.3 What are the different types of operating modes of the Pentium microprocessor?
- 12.4 How many new instructions are available in Pentium MMX with respect to Pentium?
- 12.5 What are the different features of Pentium 4 processor?
- 12.6 What are the different addressing modes of Pentium?
- 12.7 Write the difference between RISC and CISC
- 12.8 What is superscalar technology?
- 12.9 What is register starvation in Pentium?
- 12.10 What is hyper-threading technology? What is the difference between Core 2 dual and Quad core?
- 12.11 What do you mean by paging? What are the advantages of paging in Pentium?
- 12.12 What are the new features of Pentium III over Pentium?
- 12.13 How advanced is Pentium 4 over Pentium III?
- 12.14 What is the name of the first processor to support multimedia applications?
- 12.15 What do you mean by MMX?

REVIEW QUESTIONS

- 12.1. Write the architectural difference between 80486 and Pentium processors.
- 12.2. Draw the internal architecture of the Pentium microprocessor and explain its operation.
- 12.3. Write the features of the following processors:
(i) Pentium (ii) Pentium Pro (iii) Pentium MMX (iv) Pentium 4
- 12.4. Explain superscalar organization of the Pentium processor.
- 12.5. Draw and explain the register set of the Pentium processor.
- 12.6. Discuss integer pipeline and floating-point pipeline in the Pentium processor.
- 12.7. Draw a block diagram to represent different operating modes of the Pentium processor and explain each operating mode. If the Pentium processor operates in real address mode, how can it operate in virtual 8086 mode?
- 12.8. Explain memory management of the Pentium microprocessor.
- 12.9. Define physical address, linear address and logical address. What are the difference between physical address, linear address and logical address?
- 12.10. Explain the paging mechanism of the Pentium processor in detail.
- 12.11. Write short notes on the following:
(i) Floating-Point Unit (FPU) of the Pentium processor
(ii) Instruction pairing of Pentium processor
(iii) Segment selector

- (iv) Segment descriptors
 - (v) System-management mode
 - (vi) Real mode
- 12.12 Draw the schematic pin diagram of the Pentium processor. Write the function of the following pins of the Pentium processor.
 (i) ADS# (ii) BE7#–BE0# (iii) APCHK# (iv) DP7–DP0 (v) EADS#
- 12.13 Discuss different addressing modes of the Pentium microprocessor with suitable examples.
- 12.14 Discuss bus interfacing of the Pentium processor with a suitable diagram.
- 12.15 Draw and explain the following timing diagram of the Pentium processor.
 (i) Single transfer cycle (ii) Burst read cycle (iii) Inquire cycle
- 12.16 Explain Intel architecture of caches in Pentium processors.
- 12.17 What are the different cache memories in microprocessors? Explain the advantages of separate code and data caches of the Pentium processor.
- 12.18 What are the different types of cache organization? Explain two-way set associative cache.
- 12.19 Write short notes on the following:
 (i) Cache consistency (ii) Cache controller (iii) MESI Protocol (iv) L2 cache in a Pentium
- 12.20 Define cache hit, cache miss, cache hit rate and cache miss rate.
- 12.21 Write a short note on the Pentium MMX processor.
- 12.22 Draw the internal architecture of Pentium Pro: P6 family processors and discuss in detail.
- 12.23 Discuss 36-bit address bus and paging mechanism of P6 family processors.
- 12.24 Write the difference between
 (i) Pentium and Pentium Pro: P6 family processors
 (ii) Pentium II and Pentium 4
- 12.25 Draw the internal architecture of Pentium 4 processors and explain in detail.
- 12.26 Write short notes on the following:
 (i) Translation Look aside Buffer (TLB) (ii) Trace cache
 (iii) Hyper-threading (iv) Branch prediction
 (v) Out-of-order execution (vi) Branch prediction
 (viii) SSE instructions
- 12.27 (a) Enlist the special features of CISC and RISC processors.
 (b) Write the difference between CISC and RISC processors.
 (c) Give a list of CISC and RISC processors.
- 12.28 Draw the internal architecture of Power PC 601 microprocessor and discuss its operation briefly.
- 12.29 Explain briefly the following RISC architecture:
 (i) MIPS (b) Sun Ultra SPARC
- 12.30 Discuss (a) Register rename (b) Instruction parallelism (c) Architecture state

Answers to Multiple-Choice Questions

-
- 12.1 (a) 12.2 (b) 12.3 (a) 12.4 (c) 12.5 (d) 12.6 (a) 12.7 (c) 12.8 (a) 12.9 (b)
 12.10 (a) 12.11 (d) 12.12 (c) 12.13 (a) 12.14 (b) 12.15 (a) 12.16 (a)

Chapter 13

Introduction to 8051 Microcontroller

13.1 INTRODUCTION

The microprocessor is the core of any computer system, but the microprocessor by itself is completely useless, until external peripheral devices are connected with it to help it interact with the outside world. The microcontroller is a single-chip microprocessor system which consists of CPU, data and program memory, serial and parallel I/O ports, timers and external as well as internal interrupts. Actually, a microcontroller is an entire computer manufactured on a single chip. Figure 13.1 shows the schematic block diagram of a microcontroller. The comparison between a microprocessor and microcontroller is given in Table 13.1. Single-chip microcontrollers are used in instrumentation and process control, automation, remote control, office automation such as printers, fax machines, intelligent telephones, CD players and some sophisticated communication equipments. Due to integration of all function blocks on a single-chip microcontroller IC, the sizes of control board and power consumption are reduced, system reliability is increased which also provides more flexibility. The other advantages of microcontroller based systems are easy troubleshooting and maintenance; ability to do interfacing for additional peripherals, and better software security.

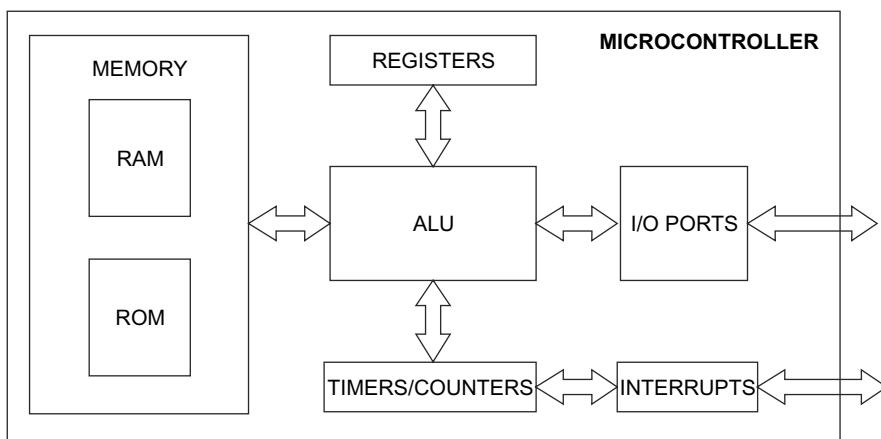
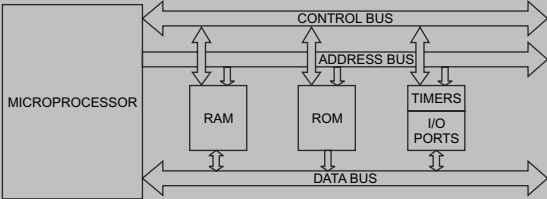
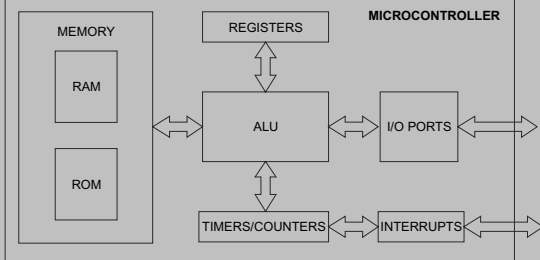


Fig. 13.1 Microcontroller

Table 13.1 Comparison between microprocessor and microcontroller

<i>Microprocessor</i>	<i>Microcontroller</i>
<p>The microprocessor is a single-chip CPU. The block diagram of microprocessor is given below.</p> 	<p>The microcontroller is a single-chip microcomputer system as given below.</p> 
<p>It consists of an ALU to perform arithmetic and logic manipulations, registers and a control unit.</p>	<p>It consists of a CPU, data and program memory, serial and parallel I/O, timers, external and internal interrupts.</p>
<p>It has address bus, data bus and control bus for interfacing with the outside world.</p>	<p>Microcontroller communicates with the outside world through P₀, P₁, P₂ and P₃ ports. Ports can be used as address and data bus depending upon control signals.</p>
<p>RAM and ROM are not incorporated within chip.</p>	<p>RAM is smaller, but it is enough for small applications. If it is not sufficient, then external memory may be added in the microcontroller-based system.</p>
<p>Microprocessors are used as the CPU in the microcomputer systems.</p>	<p>Microcontrollers are used in small embedded system products to perform control-oriented functions.</p>
<p>Microprocessor instructions perform operations based on nibbles and bytes.</p>	<p>Microcontroller instructions are able to perform bit-level operations and other operations such as based on nibbles, bytes, words, or even double words.</p>
<p>Microprocessors are available from 4-bit to 64-bit. 4-bit microprocessors are used for simple applications. 8-bit microprocessors are most commonly used in different applications. 16-bit, 32-bit and 64-bit microprocessors are used for personal computers and high-speed applications.</p>	<p>Microcontrollers are available from 4-bit to 32-bit. 4-bit microcontrollers are used for simple applications. 16-bit and 32-bit microcontrollers are used for high speed applications. 8-bit microcontrollers are most commonly used in different applications.</p>

4-bit to 32-bit microcontrollers are available in the market. Based on the number of bits, microcontrollers are classified into 4-bit microcontrollers, 8-bit microcontrollers, 16-bit microcontrollers and 32-bit microcontrollers. 4-bit microcontrollers are extensively used in electronic toys and examples of 4-bit microcontrollers are illustrated in Table 13.2. Generally, 8-bit microcontrollers are used in various control applications such as speed control, position control, and any process control system. Table 13.3 shows the different 8-bit microcontroller ICs with their features. The 16-bit microcontrollers are developed for high-speed control applications such as servo control systems, robotics etc. These microcontrollers can be programmed in high-level programming languages as well as in assembly-language programming. Some 16-bit microcontrollers are given in Table 13.4. The 32-bit microcontrollers are used for very high-speed operations in robotics, image processing, automobiles, intelligent control system, and telecommunications. Commonly used 32-bit microcontrollers are 80960 and MC683xx. Microcontrollers are most commonly used in consumer products,

automotive systems, different industrial applications and high-speed data processing. A list of microcontroller applications is given below:

- ✓ **Consumer Product** Washing machines, microwaveovens, printers, copiers, compressors, AC machines
- ✓ **Industrial Applications** dc and ac motor drives, control power electronics circuits, speed and position control, and motion control, etc.
- ✓ **Automation** Antilock braking systems, electronic power steering systems, etc.
- ✓ **High-speed data processing** Video conference, image processing, video processing, real-time compression systems and security, etc.

Table 13.2 4-bit microcontroller families

<i>IC</i>	<i>No. of Pins</i>	<i>No. of I/O Pins</i>	<i>On-chip data memory RAM</i>	<i>On-chip program memory, ROM</i>	<i>Counters</i>	<i>Extra features</i>
TLCS 47	42	35	128 bytes	2K ROM		Serial I/O
TMS1000	28	23	64 bytes	1K ROM		LED display
COP 420	28	23	64 bytes	1K ROM	1	Serial I/O
MSM6411	16	11	32 bytes	1K ROM		
HMCS 40	28	10	32 bytes	512 ROM		

Table 13.3 8-bit microcontroller families

<i>IC</i>	<i>No. of Pins</i>	<i>No. of I/O Pins</i>	<i>On-chip data memory, RAM</i>	<i>On-chip program memory, ROM</i>	<i>No. of 16-bit timers/counters</i>	<i>No. of vectored interrupts</i>	<i>Extra features</i>
8031	40	32	128 bytes	None	2	5	Full duplex serial I/O
8032	40	32	256 bytes	None	3	6	Full duplex serial I/O
8051	40	32	128 bytes	4K ROM	2	5	Full duplex serial I/O
8052	40	32	256 bytes	8K ROM	3	6	Full duplex serial I/O
8751	40	32	128 bytes	4K ROM	2	5	Full duplex serial I/O
8752	40	32	256 bytes	8K ROM	3	6	Full duplex serial I/O

Table 13.4 16-bit microcontroller families

<i>IC</i>	<i>No. of Pins</i>	<i>No. of I/O Pins</i>	<i>On-chip data memory, RAM</i>	<i>On-chip program memory, ROM</i>	<i>No. of 16-bit timers/counters</i>	<i>No. of vectored interrupts</i>	<i>Extra features</i>
HPC 46164	60	52	512 bytes	16 Kbytes ROM	4	8	External memory up to 64 K, full duplex UART, ADC
8096	68	40	256 bytes	8 Kbytes ROM	2	7	External memory up to 64 K

(Contd.)

(Contd.)

8094	48	24	256 bytes	–	2	7	External memory up to 64 K
8097	68	24	256 bytes	–	2	8	External memory up to 64 K
8095	48	20	256 bytes	–	2	8	External memory up to 64 K
8397	68	24	256 bytes	8 Kbytes ROM	2	8	External memory up to 64 K
8395	48	20	256 bytes	8 Kbytes ROM	2	8	External memory up to 64 K
80C196 EA	160	83	1K bytes register RAM 3 K bytes code RAM	8 Kbytes ROM	4	16	External memory up to 2MB, Serial I/O, ADC

13.2 ARCHITECTURE OF 8051 MICROCONTROLLER

Intel developed the 8051 microcontroller in 1980's. This microcontroller IC consists of standard on-chip peripherals, i.e., timers, counters, and UART, 4 Kbytes of on-chip program memory and 128 bytes of data memory. The 8051 has separate address spaces for program memory and data memory with the help of modified Harvard architecture. The program memory can be increased to 64 KB. Approximately, 4 Kbytes of program instructions can be stored in the internal memory of the 8051. The 8051 can address up to 64 KB of external data memory. The 8051 memory architecture includes 128 bytes of data memory that are accessible directly by its instructions. A 32-byte segment of this 128-byte memory block is bit addressable by a subset of the 8051 instructions, namely, the bit instructions. For this reason, the 8051 microcontroller is known as a Boolean processor and is used to deal with binary input and output conditions very efficiently.

In addition, the device has a low-power static design, which offers a wide range of operating frequencies down to zero. Two software-selectable modes of power reduction—*idle mode* and *power-down mode*—are available. The idle mode freezes the CPU while allowing the RAM, timers, serial port, and interrupt system to continue functioning. The power-down mode saves the RAM contents but freezes the oscillators.

This IC has the following features:

- Fabricated with Philips high-density CMOS technology with operation from 2.7 V to 5.5 V
- 8051 Central Processing Unit
 - 4K × 8 ROM
 - 128 × 8 RAM
 - Three 16-bit counter/timers
 - Full duplex serial channel
 - Boolean processor
 - Full static operation
 - Low voltage of 2.7 V to 5.5 V at 16 MHz operation
- Memory addressing capability
 - 64 K ROM and 64 K RAM
- Power control modes:
 - Idle mode
 - Power-down mode
- CMOS and TTL compatible
- Frequency range of 0 to 33 MHz
- 4-level priorities interrupt
- 6 interrupt sources
- Four 8-bit I/O ports
- Asynchronous port reset

Figure 13.2 shows the simplified block diagram of the 8051 microcontroller. The detailed architecture of the 8051 microcontroller has been depicted in Fig. 13.3 which consists of ALU, control and timing unit, RAM/EPROM/ROM, registers, latches and drivers for ports P_0 , P_1 , P_2 , and P_3 . The operation of each block has been explained in this section.

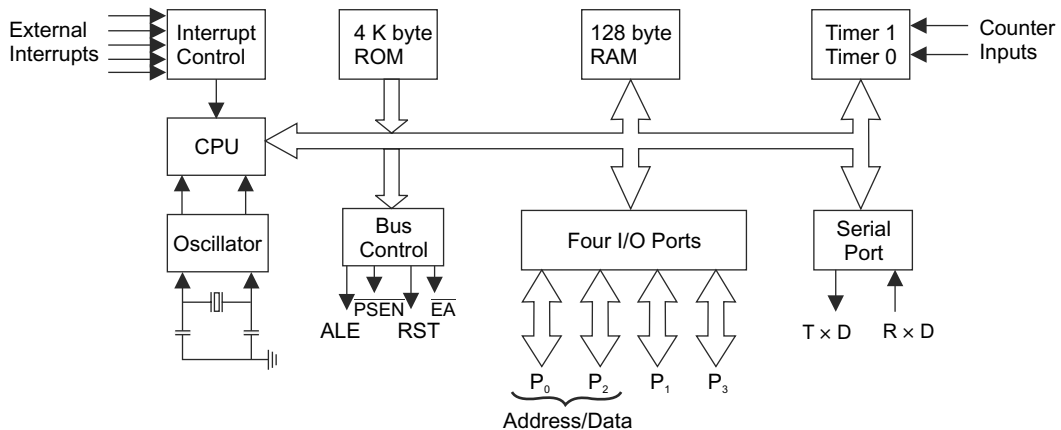


Fig. 13.2 Schematic block diagram of 8051 microcontroller

Accumulator (ACC)

The accumulator register (ACC) acts as an operand register. The accumulator may be referred as implicit or specified in the instruction by its SFR address 0E0H. ACC is commonly used for data transfer and arithmetic instructions. The accumulator is also bit addressable. ACC.2 states the bit 2 of ACC register. After any arithmetic operations, the result is stored in ACC.

B Register

The B register is used during multiply and divide operations to store the second operands for multiply and divide instructions MUL AB and DIV AB respectively. After multiplication and division, a part of the result such as upper 8 bits of multiplication result and remainder in case of division are stored in the B register. This register is commonly used as a temporary register and can also be accessed through its SFR address of 0F0H. This register is also bit addressable. It can be used as a general-purpose register except for MUL and IDIV instructions.

Program Status Word (PSW)

This is a special function register. This register consists of the different status bits that reflect the current state of microcontroller. Figure 13.4 shows the PSW which resides in the SFR space. It contains the Carry (CY), the Auxiliary Carry (AC), the two register bank select bits (RS1 and RS0), the Overflow flag (OV), a Parity bit (P), and two user-definable status flags. The carry bit serves the function of a carry bit in arithmetic operations and it also serves as the accumulator for a number of Boolean operations. The auxiliary carry bit is used in addition of BCD numbers. This bit is set if a carry is generated out of bit 3 into bit 4. F0 is a general-purpose flag bit available for user applications. The program status bits and their functions are given in Table 13.5. The bits RS1 and RS0 are used to select the register bank from four register banks as depicted in Table 13.6. The overflow flag is used for signed arithmetic operation to determine whether the result is out of range after a signed arithmetic operation. When the result is greater than +127 or less than -128, the OV flag bit is set.

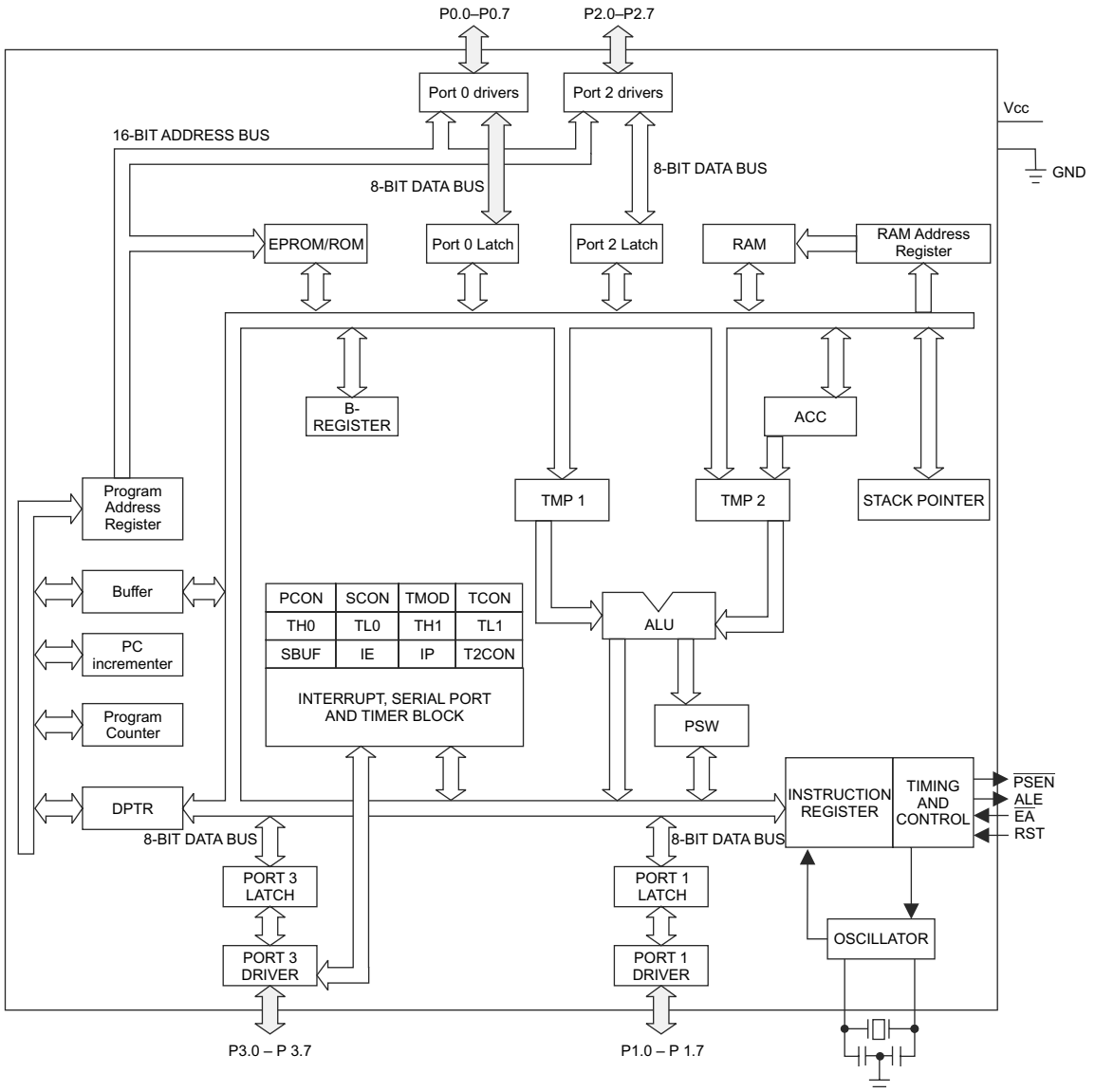


Fig. 13.3 Block diagram of 8051 microcontroller

The parity bit reflects the number of 1s in the accumulator. When the accumulator contains an odd number of 1s, P = 1. If the accumulator contains an even number of 1s, P = 0.

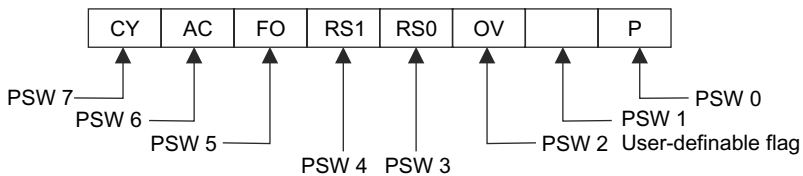


Fig. 13.4 Program status word (Bit Addressable)

Table 13.5 Program status bits and their functions

Symbol	Position	Address	Function
CY	PSW.7	D7H	Carry flag.
AC	PSW.6	D6H	Auxiliary carry flag.
F0	PSW.5	D5H	Flag 0. Available to the user for general purpose.
RS1	PSW.4	D4H	Register bank selector bit 1. Set by software to select the register bank which will be used.
RS0	PSW.3	D3H	Register bank selector bit 0. Set by software to select the register bank which will be used.
OV	PSW.2	D2H	Overflow flag.
–	PSW.1	D1H	Usable as a general-purpose flag.
P	PSW.0	D0H	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of '1' in the accumulator.

Table 13.6 Register bank selections

RS1	RS0	Register Bank	Address Range
0	0	0	00H – 07H
0	1	1	08H – 0FH
1	0	2	10H – 17H
1	1	3	18H – 1FH

Stack pointer (SP) This is an 8-bit register. SP is incremented before the data is stored onto the stack using PUSH/CALL instructions execution. During a PUSH operation, first increment SP and the copy data. In a POP operation, initially copy data and then decrement SP. The 8-bit address of the stack top is stored in this register. The stack can be located anywhere in the on-chip 128-byte RAM. Initially, the stack pointer is initialized to 07H after a reset operation. Hence, the stack begins at locations 08H. The stack can be relocated by setting SP to the upper memory area in 30H to 7FH.

Data Pointer (DPTR) DPTR is 16-bit register. It consists of a higher byte (DPH) and a lower byte (DPL) of a 16-bit external data RAM address. It can be accessed as two 8-bit registers or a 16-bit register. DPTR has been given two addresses in the special function register bank. DPTR is very useful for string operations and look-up table operations. With 16-bit DPTR, a maximum of 64 K of off-chip data memory and a maximum of 64 K of off-chip program memory can be addressed. For this, a 16-bit memory location address will be stored in DPTR through MOV DPTR, #XXXX instruction. To read data from the memory location and write data in the memory location, MOVX R0, @DPTR and MOVX @DPTR, #XX are used respectively. This can be used as a general-purpose register also. To increment the contents of DPTR, the INC DPTR instruction is executed. But, there is no such instruction in 8051 to decrement the DPTR.

Port 0, Port 1, Port 2, Port 3 Latches and Drivers Each latch and corresponding driver of ports 0–3 is allotted to the corresponding on-chip I/O port. All ports are bidirectional input/output ports of 8 bits each. The addresses of latches are stored in the special function register bank. Using these addresses, Ports 0–3 can communicate with other ICs and all input/output signals. The output drivers of ports 0 and 2, and the input buffers of Port 0, are used in accesses to external memory. In this case, Port 0 outputs the low

byte of the external memory address, and Port 2 outputs the higher byte of the external memory address. P1 can be used as input/output port and it acts an important role in programming of internal memory of 8051. P3 can also be used as an I/O port. The pins of Port 3 have alternate functions such as serial input line (P3.0), serial output line (P3.1), external interrupt lines (P3.2, P3.3), external timer input lines (P3.4, P3.5), external data memory write strobe (P3.6) and external data memory read strobe (P3.7).

The port registers specify the value to be output on the specific output port or read value from the specified input port. Ports are also used as bit addressable. The first bit of the port has the same address as the register. Suppose the port address of P1 is 90H in SFR. The address of P1.0 is 90H and the port address of P1.7 is 97H.

Serial Port Data Buffer

The serial port data buffer internally consists of two independent registers such as transmit buffer and receive buffer at the same location. The *transmit buffer* is a parallel-in serial-out register. The *serial data receive buffer* is a serial-in parallel-out register. The serial data buffer is identified as SBUF. If data is moved to SBUF, it goes to the transmit buffer and sets serial transmission. When data is moved from SBUF, it receives serial data from the receive buffer.

Timing Registers

There are two 16-bit timing registers in 8051. The 16-bit timer register can be accessed as their lower and upper bytes. The TH₀ and TL₀ represent the lower byte and higher byte of the timing register 0 respectively. In the same way, TH₁ and TL₁ represent the lower byte and higher byte of the timing register 1 respectively. All timing registers can be accessed by using the four different addresses allotted to them. The addresses of registers have been stored in the Special Function Registers (SFR) from 80H to FF. In the 8052, one more pair (TH₂ and TL₂) exists for Timer/Counter 2. The operation of the timing register may be timing or counting. There is a Timer Control (TCON) and a Timer Mode (TMOD) Registers to configure all timers/counters in various modes. These timers can be used to measure pulse width from 1ms to 65 ms, generate longer time delays and time intervals also.

Control Registers

The control register consists of special function registers such as Interrupt Priority (IP), Interrupt Enable (IE), Timer Mode (TMOD), Timer Control (TCON), Serial Port Control (SCON) and Power Control (PCON). All of these registers have allotted addresses in the special function register bank of the 8051 microcontroller.

Capture Registers

Register pairs RCAP2H and RCAP2L exist only in the 8052. Actually these are the capture registers for Timer-2. When Timer-2 operates in capture-mode operation, a transition at the 8052 T2EX pin causes TH₂ and TL₂ to be copied into RCAP2H and RCAP2L. Timer-2 has a 16-bit auto-reload mode and RCAP2H and RCAP2L hold the reload value for this mode operation.

Timing and Control Unit

The timing and control unit generates all the necessary timing and control signals required for the internal operation of the microcontroller. This unit also generates necessary control signals ALE, \overline{PSEN} , \overline{RD} and \overline{WR} to control the external system bus.

Oscillator

The oscillator circuit generates the basic timing clock signal for the operation of the microcontroller using a crystal oscillator. The 80C51 microcontroller operates at about 12 MHz frequency. In this microcontroller, only the quartz crystal oscillator is connected with microcontroller externally and the remaining oscillator circuit components are incorporated in the chip.

Instruction Register (IR) This register is used to decode the opcode of any instruction to be executed. After decoding, this register sends the decoded information to the timing and control unit to generate necessary signals for the execution of the specified instruction.

Program Address Register This is an on-chip EPROM and a basic circuit mechanism to internally address it. EPROM is available in 8051, 8052, 8751 and 8752 microcontrollers and it is not available in 8031 and 8032 microcontrollers.

RAM This block provides internal 128 bytes of RAM.

RAM Address Register The RAM address register is used to generate address of RAM internally.

ALU (Arithmetic and Logic Unit) The ALU performs 8-bit arithmetic and logical operations when the operands are held at the temporary registers TMP1 and TMP2. These temporary registers cannot be accessed by users. The output of the ALU is stored in the accumulator in most of the arithmetic and logical operations with few exceptions. Apart from addition and subtraction operations, the 8051 microcontroller also performs multiplication and division operations. The logical operations such as AND, OR, NOT, Ex-OR operations are also performed in ALU.

SFR (Special Function Registers) This register bank is a set of registers, which can be addressed using their respective addresses in the range of 80H to FFH.

13.3 MEMORY ORGANIZATION

The 80C51 microcontroller has separate address spaces for program memory as well as data memory. Figures 13.5 and 13.6 show the program and data memory respectively. The logical separation of program and data memory allows the data memory to be accessed by 8-bit addresses and this 8-bit memory access can be stored quickly and manipulated by an 8-bit CPU. The 16-bit data memory addresses can also be generated through the DPTR register and data can be read from external memory.

The program memory can be read only such as ROM and EPROM. There can be up to 64 Kbytes of program memory. In the 80C51, the lowest 4K bytes of program are exist on-chip and 60 K bytes program memory is external memory. In the ROMless versions of the microcontroller, program memory is external and its capacity is about 64 K. The read strobe for external program memory is the \overline{PSEN} (Program Store Enable). Depending on the instructions, the same address can refer to two logically and physically different memory locations.

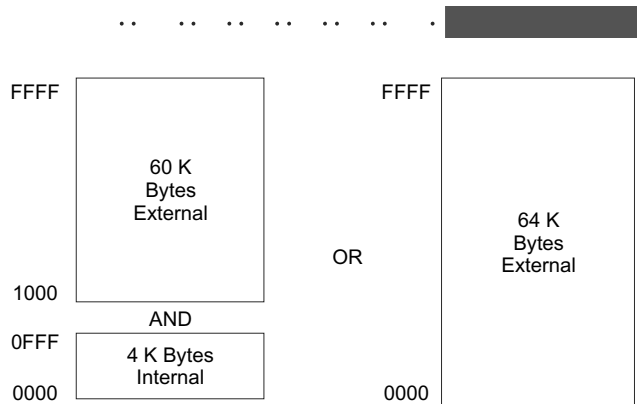


Fig. 13.5 Program memory (Read Only) structure

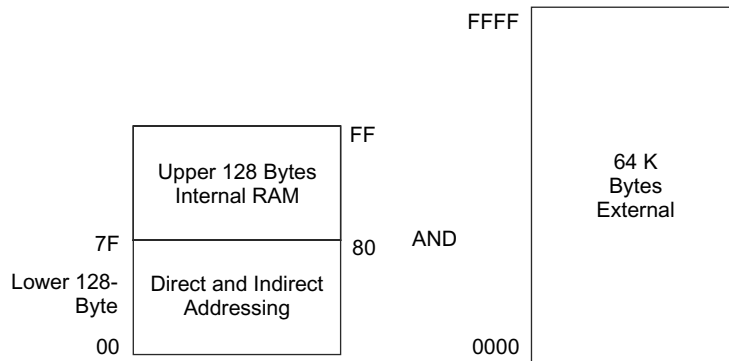


Fig. 13.6 Data memory (Read/Write) structure

The data memory has a separate address space from program memory. In the 80C51 microcontroller, the lowest 128 bytes of data memory are available on-chip and maximum 64 K bytes of external RAM can be addressed in the external data memory space. In ROMless versions of the microcontroller, the lowest 128 bytes are on-chip. The CPU of the microcontroller generates read and write signals during external data memory accesses.

Program Memory

A map of the lower part of the program memory is shown in Fig. 13.7. After reset, the microcontroller starts fetching instructions from 0000H and the CPU also starts execution from location 0000H. This can be either on-chip memory or external memory depending on the value of the \overline{EA} pin. When \overline{EA} is low, the program memory is external. If \overline{EA} is high, the address from 0000H to 0FFFH will refer to on-chip memory and the address from 1000H to FFFFH can refer to external memory as depicted in Fig. 13.5.

It is clear from Fig. 13.7 that each interrupt is assigned a fixed location in program memory. When any interrupt is executed, the CPU will jump to that specified location and it starts execution of the service routine. For example, external interrupt 0 is assigned to memory location 0003H. When external interrupt 0 is going to be used, its service routine must start at the location 0003H. If the interrupt is not going to be used, its service location is available as general-purpose program memory.

Usually, the interrupt service locations are available at 8-byte intervals: 0003H for external interrupt0, 000BH for Timer-0, 0013H for external interrupt1, and 001BH for Timer-1, etc. When the interrupt service routine is short, it can reside entirely within that 8-byte interval. But the longer service routines can use a jump instruction to skip over subsequent interrupt locations.

The lowest 4K bytes of program memory will be either in the on-chip ROM or in an external ROM. The internal or external program memory selection can be made by \overline{EA} (External Access) pin. In the 80C51, if

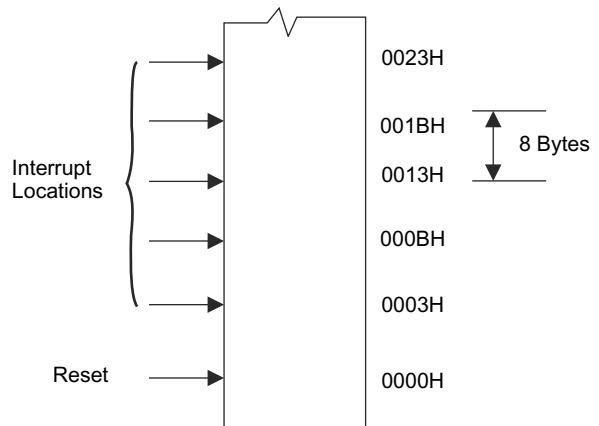


Fig. 13.7 8051 Program memory

the \overline{EA} pin is V_{CC} then the program fetches to internal ROM addresses 0000H through 0FFFH. In this case, program fetches to addresses 1000H through FFFFH are directed to external ROM.

When the \overline{EA} pin is grounded, all program fetches are directed to external ROM.

The read strobe to external ROM, \overline{PSEN} is used for all external program fetches. The \overline{PSEN} is not activated for internal program fetches.

Figure 13.8 shows the hardware configuration for external program execution. Program memory addresses are always 16 bits wide. Ports 0 and 2 are dedicated to bus functions during external program memory fetches. Port 0 serves as a time multiplexed address/data bus. When the ALE (Address Latch Enable) signal is high, Port 0 is used as the low byte of the Program Counter (PCL) as an address. Port 2 emits the high byte of the Program Counter (PCH). Then PSEN strobes the EPROM and the code byte is read into the microcontroller.

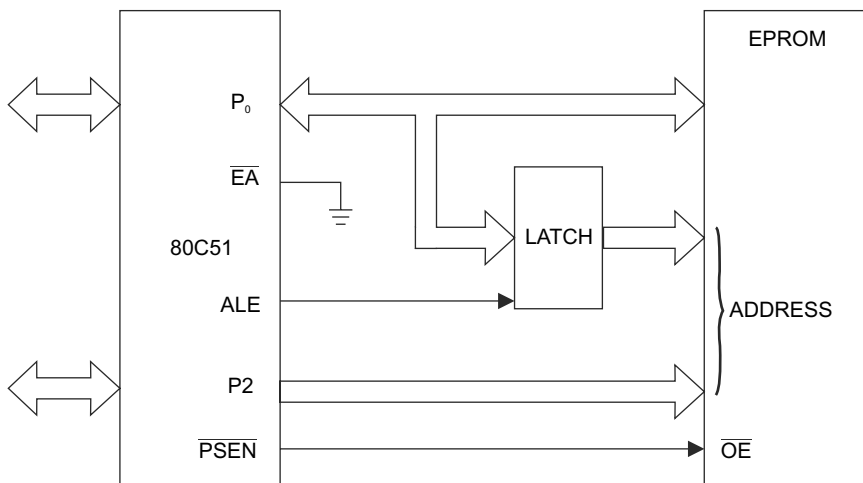


Fig. 13.8 Executing from External Program Memory

Data Memory

The hardware configuration for accessing up to 2 K bytes of external RAM is shown in Fig. 13.9. The MOVX instruction is used to access the external data memory. The CPU in this case is executing from internal ROM. Port 0 acts as a time multiplexed address/data bus to the RAM and 3 lines of Port 2 are being used to page the RAM. The CPU of the microcontroller generates \overline{RD} and \overline{RW} signals as required during external RAM accesses. The CPU can be used to access up to 64 K bytes of external data memory. The external data memory addresses can be either 1 or 2 bytes wide. One-byte addresses are frequently used in conjunction with one or more other I/O lines to page the RAM. Two-byte addresses can also be used, in which case the high-address byte is emitted at Port 2.

The internal and external data memory spaces available to the 80C51 user are given in Fig. 13.10. The 80C51 microcontroller has 256 bytes of RAM on the chip. Among them, only the lower 128 bytes are used for internal data storage. The upper 128 bytes are used as the special function registers (SFR). The detail of the lower 128 bytes is illustrated in Fig. 13.11. The lower 128 bytes of RAM which can be accessed by both direct and indirect addressing can be divided into three segments as listed below:

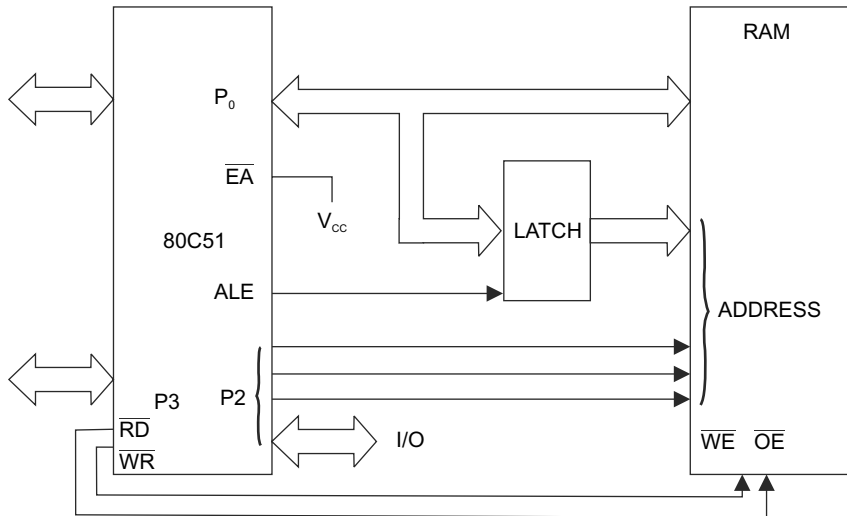


Fig. 13.9 Accessing external data memory

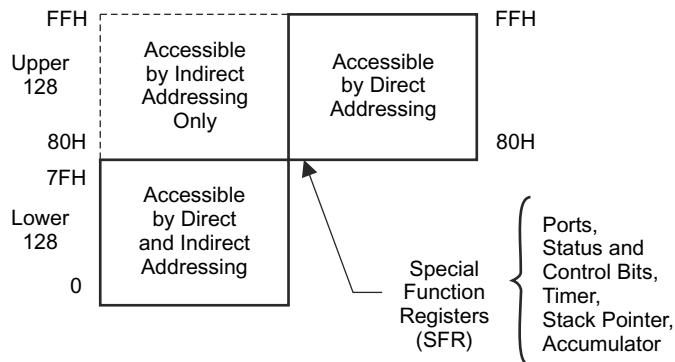


Fig. 13.10 Internal data memory

Register Banks 0–3

The lowest 32 bytes (00H to 1FH) of the on-chip RAM occupied as 4 banks of 8 registers each as depicted in Fig. 13.12. The bank is selected by setting 2 bits in PSW. Only one bank is active at a time. By default, Bank-0 is selected and its register addresses 00H to 07H. Each bank consists of eight 1-byte registers R0 to R7.

Reset initializes the stack pointer to location 07H and it is incremented once to start from location 08H, which is the first register (R0) of the second register bank. Therefore, in order to use more than one register bank, the SP should be initialized to a different location of the RAM.

Bit Addressable Area

The next 16 bytes contain 20H–2FH form a block that can be addressed as either bytes or individual bits. The bytes can be addressed from 20H to 2FH. The bits can be addressed from 00H to 7FH as depicted in Fig. 13.13. For accessing the bits, specific instructions are used. Hence, bits 0–7 can also be referred to as bits 20.0–20.7 and bits 8–FH are the same as 21.00–21.7, and so on. Each of the 16 bytes in this segment can also be addressed as a byte.

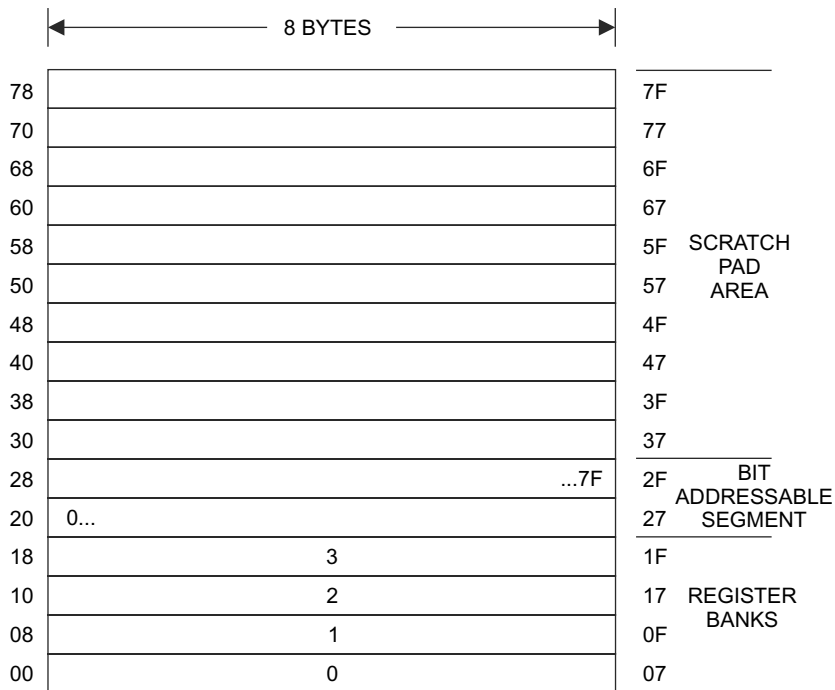


Fig. 13.11 128-bytes of RAM direct and indirect addressable

Scratch-Pad Area

The memory locations 30H–7FH are general-purpose RAM. On the other hand, if the stack pointer has been initialized to this area, enough bytes should be left aside to prevent SP data destruction.

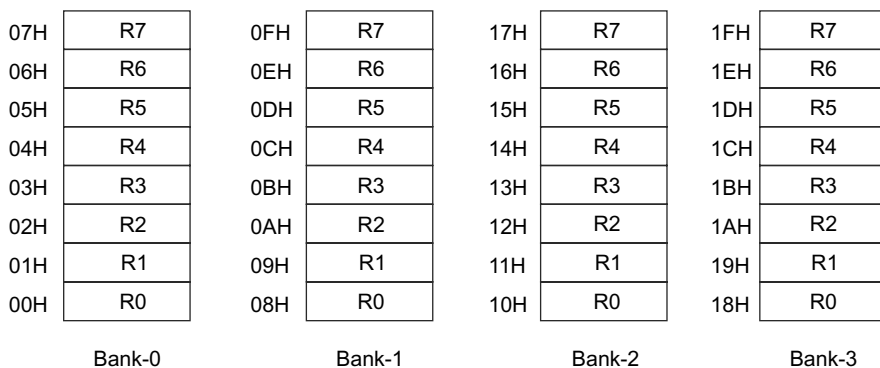


Fig. 13.12 Four banks of 8 registers R7-R0

The upper 128 bytes of the on-chip RAM are used for special function registers (SFR) as shown in Fig. 13.14. Actually, only 25 of these bytes are used. The other bytes are reserved for advanced versions of the microcontroller. These bytes are associated with registers which are used for different functions and operations of the microcontroller. Some of these registers are bit addressable and some of these byte addressable.

2F	7F	7E	7D	7C	7B	7A	79	78
2E	77	76	75	74	73	72	71	70
2D	6F	6E	6D	6C	6B	6A	69	68
2C	67	66	65	64	63	62	61	60
2B	5F	5E	5D	5C	5B	5A	59	58
2A	57	56	55	54	53	52	51	50
29	4F	4E	4D	4C	4B	4A	49	48
28	47	46	45	44	43	42	41	40
27	3F	3E	3D	3C	3B	3A	39	38
26	37	36	35	34	33	32	31	30
25	2F	2E	2D	2C	2B	2A	29	28
24	27	26	25	24	23	22	21	20
23	1F	1E	1D	1C	1B	1A	19	18
22	17	16	15	14	13	12	11	10
21	0F	0E	0D	0C	0B	0A	09	08
20	07	06	05	04	03	02	01	00

Fig. 13.13 Bit addressable RAM

13.4 PIN DIAGRAM OF 8051 MICROCONTROLLER

The 8051 microcontroller is available in DIP, QFP and LLC packages. This is a 40-pin IC. There are four 8-bit ports P0, P1, P2 and P4. Therefore, a total of 32 pins are covered for 4 ports. The remaining 8 pins are V_{CC} (power supply), V_{SS} (ground), crystal oscillator pins XTAL1 and XTAL2, RST (Reset), PSEN (Program Store Enable), ALE (Address Latch Enable), and EA (External Access) respectively. The pin diagram of 8051 microcontroller is depicted in Fig. 13.15 and schematic pin diagram is illustrated in Fig.13.16. The brief discussions of all these pins are explained in this section.

V_{CC} This pin is connected to +5 V supply voltage. A 125 mA current is drawn from supply for 8051/8031 microcontroller and the maximum power dissipation rating is about 1W.

V_{SS} This is a ground pin for supply. All the voltages are specified with respect to this pin.

F8								FF
F0	B							F7
E8								EF
E0	ACC							E7
D8								DF
D0	PSW							D7
C8								CF
C0								C7
B8	IP							BF
B0	P3							B7
A8	IE							AF
A0	P2							A7
98	SCON							9F
90	P1							97
88	TCON	TMOD	TL0	TL1	TH0	TH1		8F
80	P0	SP	DPL	DPH			PCON	87

Fig. 13.14 SFR memory map

RST

The RST input pin resets the 8051, only when the RST pin is high for two or more machine cycles. There are two ways to reset the 8051 microcontroller such as power-on reset and manual reset. When the microcontroller is reset, all values in the register are lost. So the reset values of PC, ACC, B, PSW and DPTR are 0000H and the content of SP is 0007H. The power on a reset circuit diagram is shown in Fig. 13.17. A pull-down resistance of 8.2K is connected between RST and ground terminals. A 10 μ F capacitance is also connected from V_{CC} to RST pin. These components provide a delay of about 24 clock cycles. Figure 13.18 shows the manual reset circuit. A push-button switch is added across the 10 μ F capacitor.

ALE (Address Latch Enable)

ALE is used for demultiplexing the address and data bus when the 8051 microcontroller is interfacing with external memory. Port 0 provides the low-byte of address bus A_7 to A_0 and data bus D_7 to D_0 . When $ALE = 1$, Port 0 used as address bus A_7 to A_0 . If $ALE = 0$, Port 0 is used as the data bus. Usually, ALE is activated periodically with a constant rate of one-sixth of the oscillator frequency.

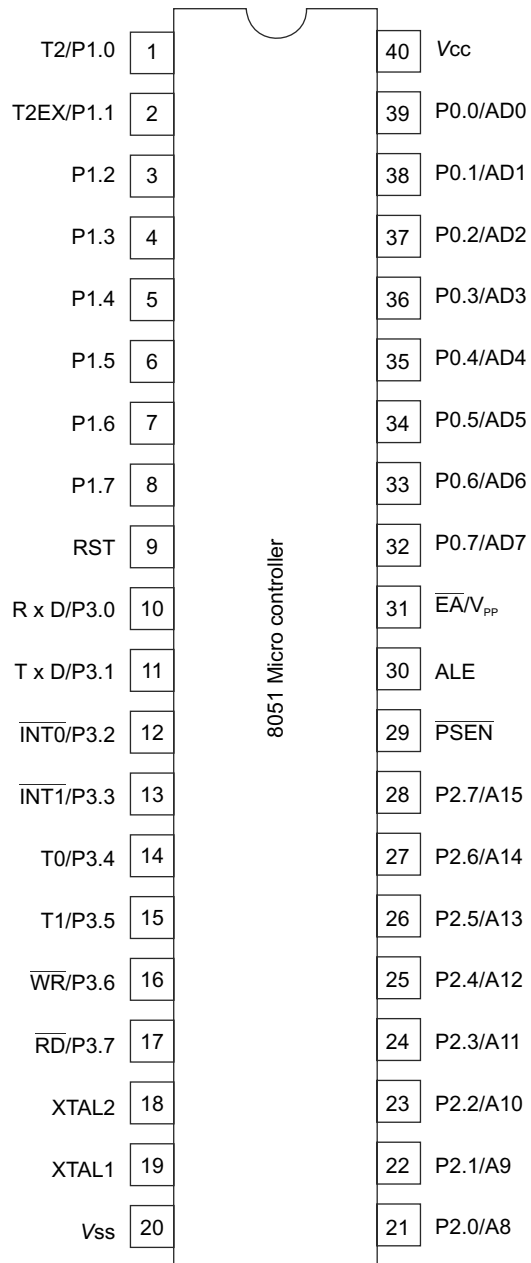


Fig. 13.15 Pin diagram of the 8051 microcontroller

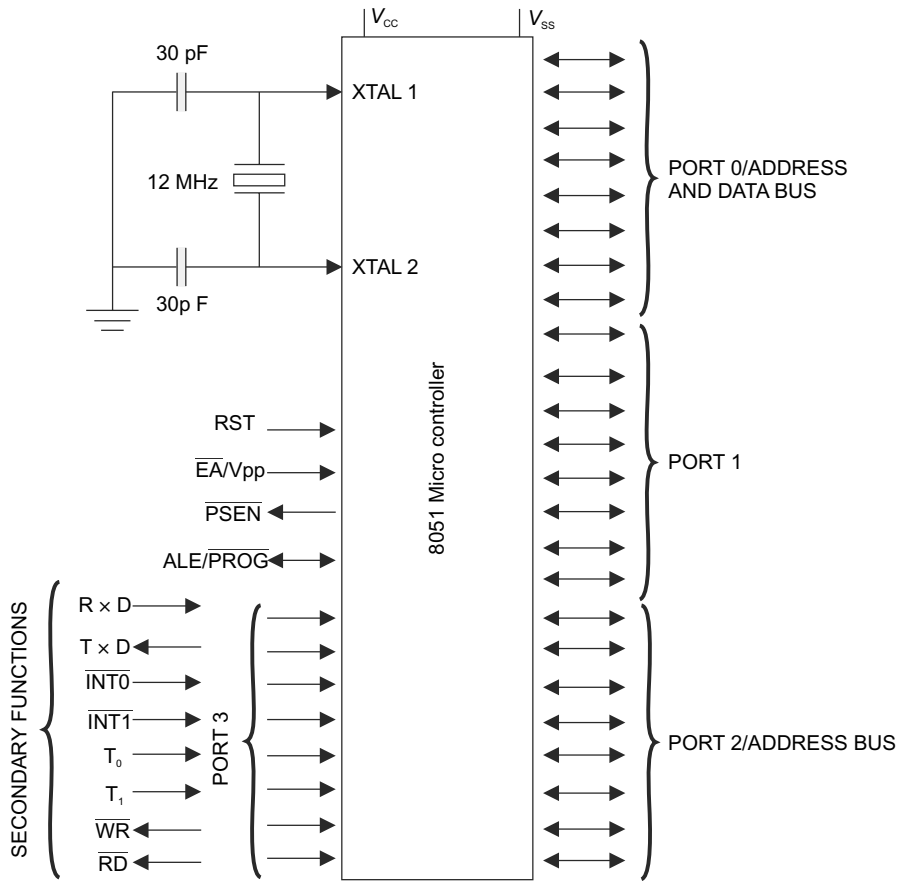


Fig. 13.16 Schematic pin diagram of the 8051 microcontroller

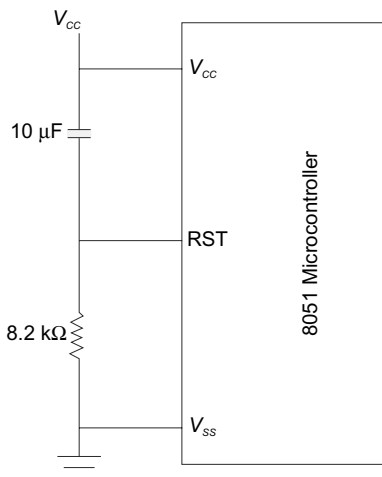


Fig. 13.17 Power on reset

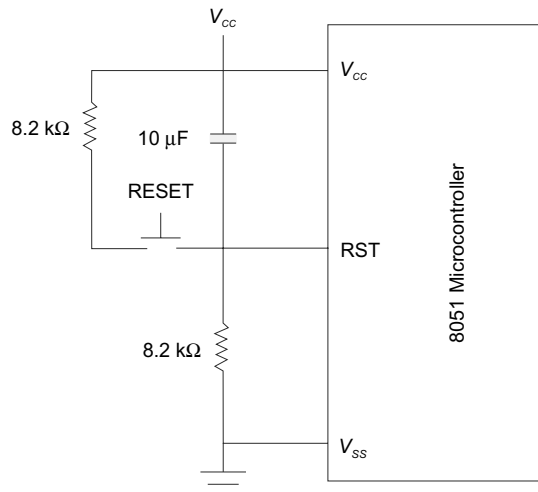


Fig. 13.18 Power on manual reset

\overline{EA} (External Access)

This pin is connected to either V_{cc} or ground. When this pin is connected to V_{cc} or high, 8051 can execute programs from internal program memory till address 0FFFH. If EA is low, the microcontroller can execute from external memory only.

 \overline{PSEN} (Program Store Enable)

This pin is an active-low output control signal. This is used as a read signal for reading data from the external program memory. This pin is activated after every six-clock cycles during fetching the data from external program memory. The \overline{PSEN} pin remains high during execution of a program from internal ROM.

$XTAL_1$ and $XTAL_2$ are the oscillator pins to connect the crystal oscillators of nominal frequency of 12 MHz or 11.059 MHz. $XTAL_1$ is input to the inverting oscillator amplifier and input to the internal clock generating circuits. $XTAL_2$ is output from the inverting oscillator amplifier. Usually, the quartz crystal oscillator is connected to $XTAL_1$ and $XTAL_2$ and it also needs two capacitors of 30 pF values. One side of each capacitor is connected to the grounds as shown in Fig. 13.19.

When the 8051 microcontroller is connected to a crystal oscillator and power supply is given, we can observe the frequency on the $XTAL_2$ pin. The 8051 microcontroller's operation is synchronizing with the crystal oscillator output signal. Effectively, the 8051 operates based on machine cycles. A machine cycle is the minimum amount of time in which a single 8051 instruction can be executed, but some instructions take multiple machine cycles. In 8051, a machine cycle consists of a sequence of 6 states numbered S_1 through S_6 (twelve clock cycles) as shown in Fig. 13.20. Each state time lasts for two oscillator periods. A machine cycle is also called an *instruction cycle*. Each instruction cycle has six states (S_1 – S_6) and each state has two pulses (P1 and P2). Hence a machine cycle takes 12 oscillator periods or 1 if the oscillator frequency is 12 MHz.

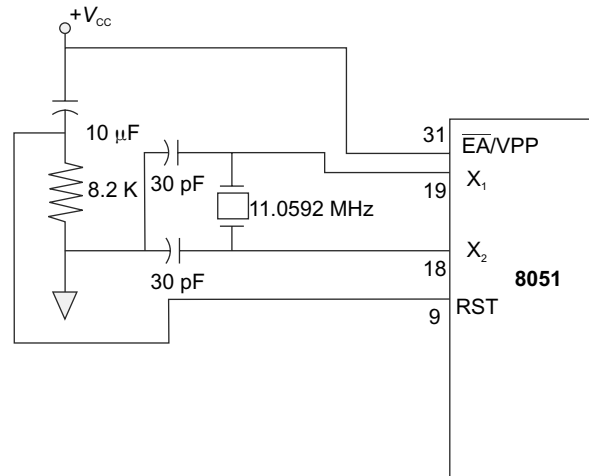


Fig. 13.19 Oscillator circuit

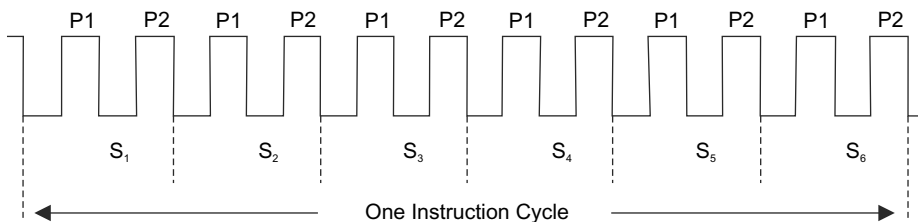


Fig. 13.20 Instruction cycle of 8051 microcontroller

Port 0 (P0.0–P0.7)

Port 0 consists of 8-bit bidirectional input/output port pins. These are bit addressable. This port has been given an address in the SFR address range. Port 0 acts as multiplexed low order address/data bus AD_7 – AD_0 . ALE is used for demultiplexing address and data bus. When ALE is 0, it provides data D_7 – D_0 , but when ALE is 1 it is used as address A_7 – A_0 . Port 0 pins are open drain I/O. To use the pins of Port 0, each pin must be connected externally to a 10 k-ohm pull-up resistor.

Port 1 (P1.0–P1.7) Port 1 pins can be used as either input or output. This port is an 8-bit quasi-bidirectional bit addressable port, and Port 1 pins are internally pulled high with fixed pull-up resistors. Hence, this port does not need any pull-up resistor as it already has a pull-up resistor internally. This port has been given an address in the SFR address range. The user should configure it either as an input or output port. This port acts as an input port when it writes 1 to all its 8 bits. This port acts as an output port when it writes 0 to all its 8 bits. Therefore Port 1 pins have no dual functions.

Port 2 (P2.0–P2.7) Port 2 is also an 8-bit quasi-bidirectional bit addressable I/O port and port pins are pulled high internally. It has also been given an address in the SFR address range. Port 2 generates higher eight bits of address (A_{15} – A_8) during external program and data memory accesses, if ALE is high and \overline{EA} is low. Port 2 receives higher order address bits during programming the internal ROM of 8051 microcontroller.

Port 3 (P3.0–P3.7) Port 3 is also an 8-bit bi-directional bit addressable I/O port with internal pull-up resistances. This port has been given an address in the SFR address range. There are other functions multiplexed with Port 3 pins as given in Table 13.7.

Table 13.7 Alternative functions of pin P3.0 to P3.7

Port 3	Alternative function
P3.0	Acts as serial input data pin ($R \times D$)
P3.1	Acts as serial output data pin ($T \times D$)
P3.2	Acts as external interrupt pin 0 ($\overline{INT_0}$)
P3.3	Acts as external interrupt input pin 1 ($\overline{INT_1}$)
P3.4	Acts as external input to timer 0 (T_0)
P3.5	Acts as external input to timer 1 (T_1)
P3.6	Acts as write control signal for external data memory (\overline{WR})
P3.7	Acts as read control signal for external data memory read operation (\overline{RD})

The port structures of 8051 microcontrollers are depicted in Fig. 13.21 and Fig. 13.22. Each port consists of a latch, an input buffer and an output driver. The D flip-flop is used as bit latch and it clocks from internal data bus in response to write to latch from internal CPU bus. The output of a flip-flop can be read onto the internal data bus in response to a read latch signal from the internal CPU bus. The operation of the read pin is different from read latch. The port pin can be read onto the internal data bus whenever CPU sends a read-pin command.

Ports 1, 2 and 3 are bi-directional ports with fixed internal pull-up resistors. When a port pin is used as input, 1 must be written to a port latch. The $Q = 1$, $\overline{Q} = 0$ FET is OFF and the pin is simply pulled high by the pull-up resistor. Thereafter, the pin status can be read onto the internal data bus. Writing '1' to output pin P1.X of Port 1 is shown in Fig. 13.23.

The port pin can be used as output while writing 0 onto the pin. Then $Q = 0$, $\overline{Q} = 1$ FET is ON. The port pins can sink more current than its source current. Sinking current is about 0.5 mA but source current is in the order of tens μA only. Figure 13.24 shows writing '0' to output pin P1.X of Port 1. Reading '1' and '0' at input pin P1.X of Port 1 using MOV A, P1 are depicted in Fig. 13.25 and Fig. 13.26 respectively.

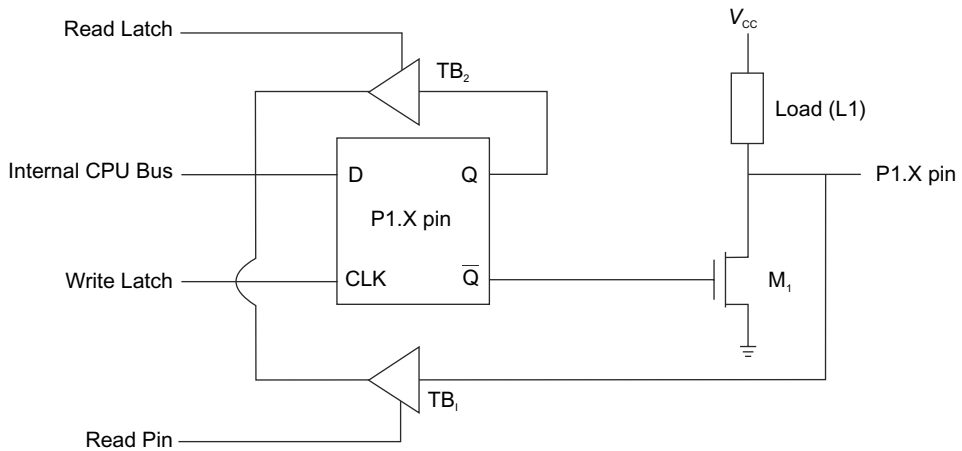


Fig. 13.21 A pin of Port 1

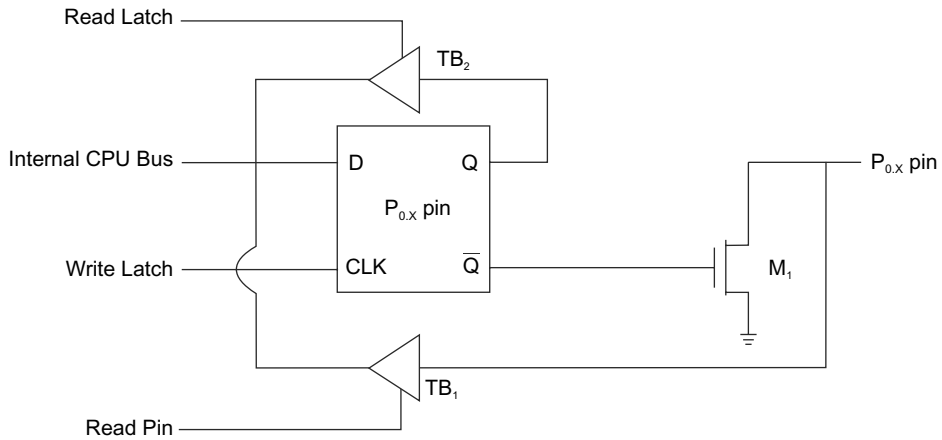


Fig. 13.22 A pin of Port 0

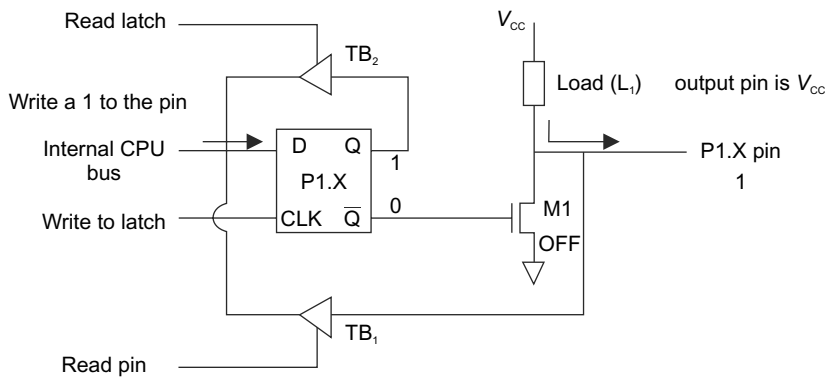


Fig. 13.23 Writing '1' to output pin P1.X of Port 1

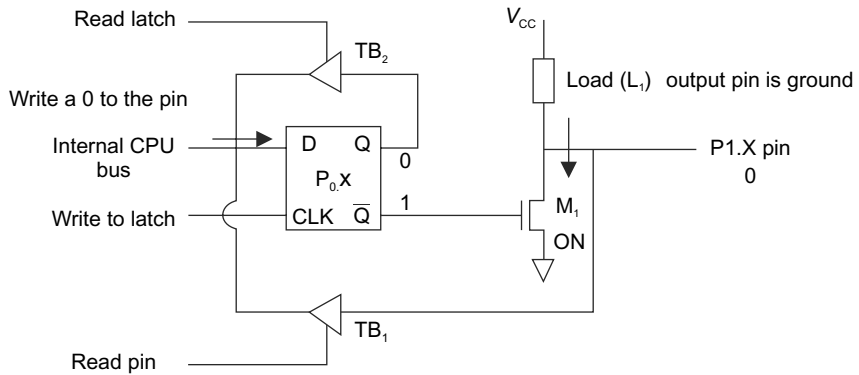


Fig. 13.24 Writing '0' to output pin P1.X of Port 1

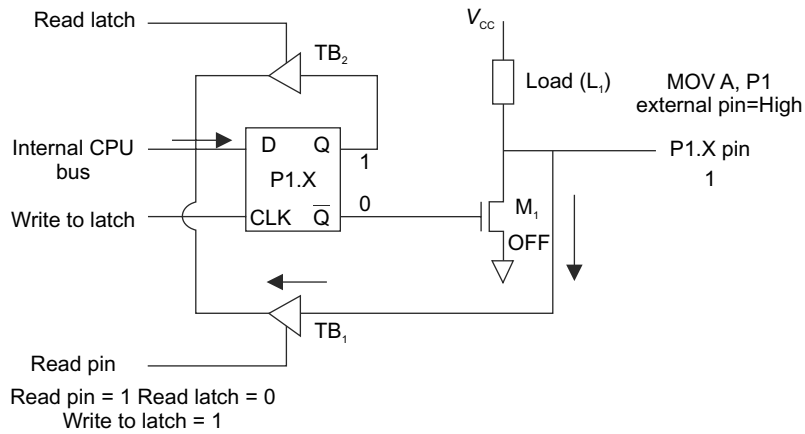


Fig. 13.25 Reading '1' at input pin P1.X of Port 1 using MOV A, P1

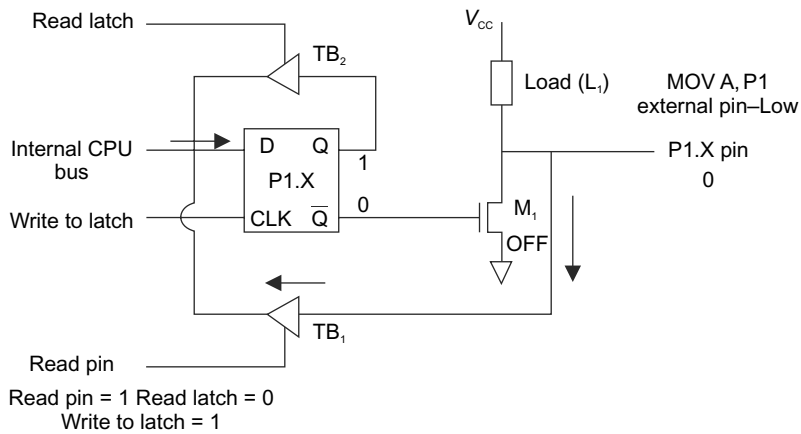


Fig. 13.26 Reading '0' at input pin P1.X of Port 1 using MOV A, P1

Port 0 is a bi-directional open drain I/O without internal pull-up resistors. When this port is configured as an input, it floats as depicted in Fig. 13.27. If '1' is written to a Port 0 latch, FET is OFF, the pin floats and can be used as high-impedance input. To get output of Port 0 pins external pull-up resistances are connected between Port 0 pins and +V_{cc} as shown in Fig. 13.28.

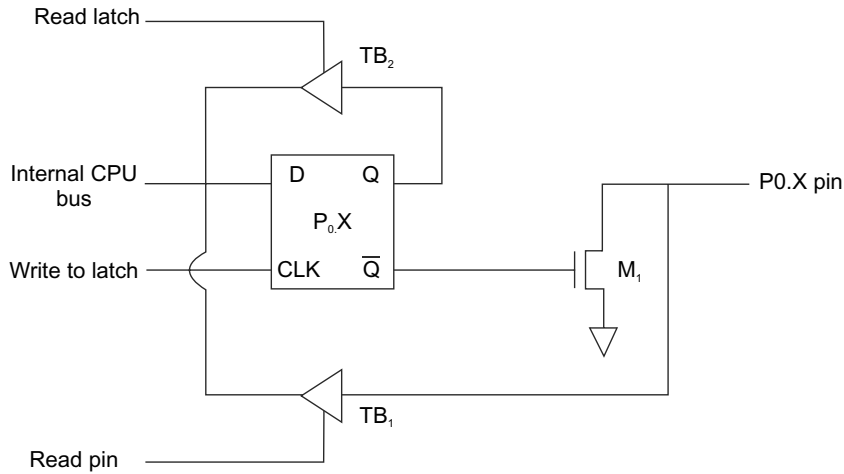


Fig. 13.27 A pin of Port 0

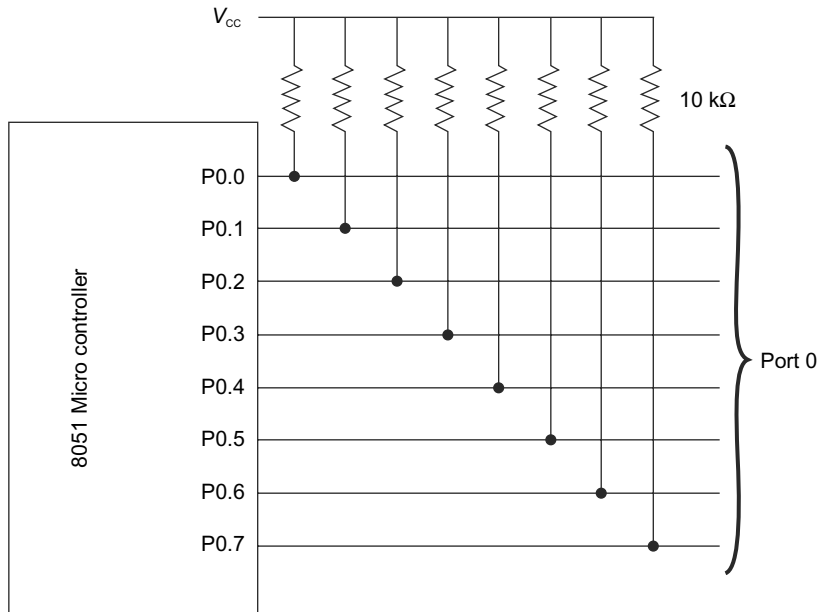


Fig. 13.28 Port 0 with pull-up resistors

13.5 TIMERS/COUNTERS

The 8051 microcontroller has two 16-bit timers/counters such as *TIMER0* (T0) and *TIMER1* (T1). Each timer can be programmed to count internal clock pulses of 8051 microcontroller. These timers are used for the following functions:

- ◆ Calculate time delay between two events
- ◆ Counting the number of events
- ◆ Generate baud rate for serial ports
- ◆ Frequency measurement
- ◆ Pulse width measurement

Generally a timer is used to count machine cycles and provides a specified time delay. Actually a machine cycle consists of 12 oscillator periods. Hence the counting rate is about

$$\frac{\text{Oscillator frequency}}{12}$$

When the oscillator frequency is 12 MHz, the time period of one clock cycle is 1µs. The counter of the 8051 microcontroller is incremented in response to a transition from 1 to 0 at external pin, either T0 or T1. The counter output is a count value which represents the occurrence of 1 to 0 transitions at the external pin. Usually, counters are used as up counter. Figure 13.29 shows a 3-bit counter which counts from 0 to 7 and the overflow flag is set after counting 7. The 3-bit counter is not used in the 8051 microcontroller, but 8-bit and 16-bit counters are commonly used. When the 16-bit counter overflows from 0000H to FFFFH, it can set a flag and generates an interrupt.

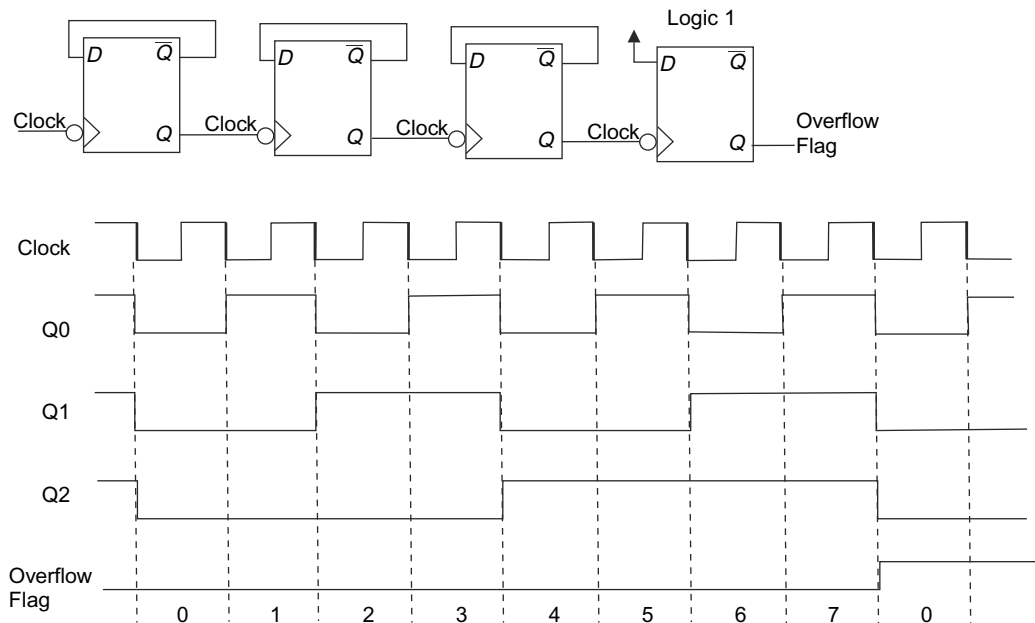


Fig. 13.29 3-bit counter circuit

The 16 bits of the timer consists of higher byte THx and the lower byte TLx, where x may be either 0 or 1. For TIMER1, TH1 is the higher byte of timer 1 and TL1 is the lower byte of timer 1 as shown in Fig. 13.30. Similarly, TH0 is the higher byte of timer 0 and TL0 is the lower byte of timer 0 as depicted in Fig. 13.31.

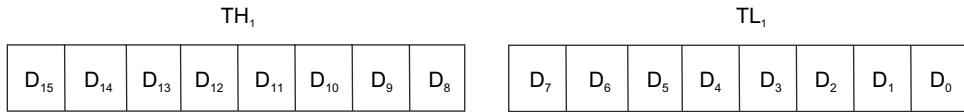


Fig. 13.30 Timer 1 registers



Fig. 13.31 Timer 0 registers

13.5.1 Operating Modes

The timer can be operating in four different modes, namely mode 0, mode 1, mode 2 and mode 3. The mode bits M₁ and M₀ in the TMOD register are used to select any one of the operating modes as given below:

M ₁	M ₀	Operating Modes	Functions
0	0	Mode 0	13-bit timer mode
0	1	Mode 1	16-bit timer mode
1	0	Mode 3	8-bit timer mode
1	1	Mode 4	Split timer mode

Timer Mode 0

In this mode, timer operates as a 13-bit timer. THx register is used as an 8-bit counter and TLx can be used as a 5-bit counter as shown in Fig. 13.32. The count value varies from 0000H to 1FFFH. Whenever the timer reaches its maximum value 1FFFH, it returns to 0000H and the overflow flag TF is set. The timer clock frequency is oscillator frequency/12. The clock frequency input to THx is

$$\frac{\text{Oscillator frequency}}{12 \times 2^5} = \frac{\text{Oscillator frequency}}{12 \times 32}$$

When oscillator frequency is 12MHz, the clock frequency input to THx is = $\frac{12 \text{ MHz}}{12 \times 32}$.

The overflow flag is set to zero after $32 \times 256 = 8192$ machine cycles.

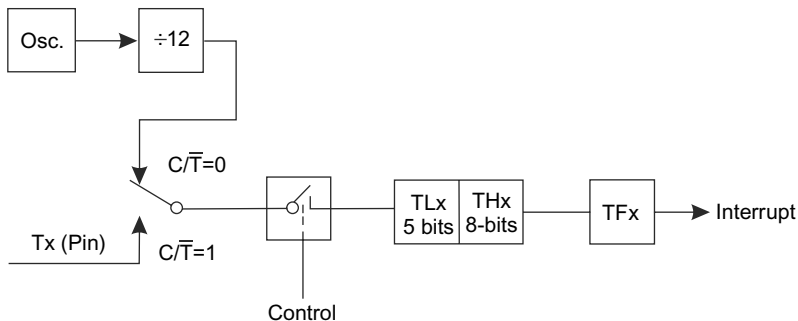


Fig. 13.32 Timer Mode 0 (13-bit timer)

Timer Mode 1

In timer mode-1, timer operates as a 16-bit timer where THx register is used as an 8-bit counter and TLx is used as an 8-bit counter. The timer higher byte THx is connected in cascade with the timer lower byte TLx as shown in Fig.13.33 where the timer counts from 0000H to FFFFH. TLx is incremented from 00H to FFH. After counting FFH, the timer resets to 0 and THx is incremented by 1. As TLx and THx operate as 16-bit counter, it can count up to 65536D. The overflow occurs after FFFFH and timer overflow flag is set. After overflow, the counter resets to 0000H when the timer starts counting from an initial value, the time delay will be

$$\frac{12 \times (65,536 - \text{Initial Value})}{\text{Frequency}}$$

where, initial value is equal to TLx + THx × 256

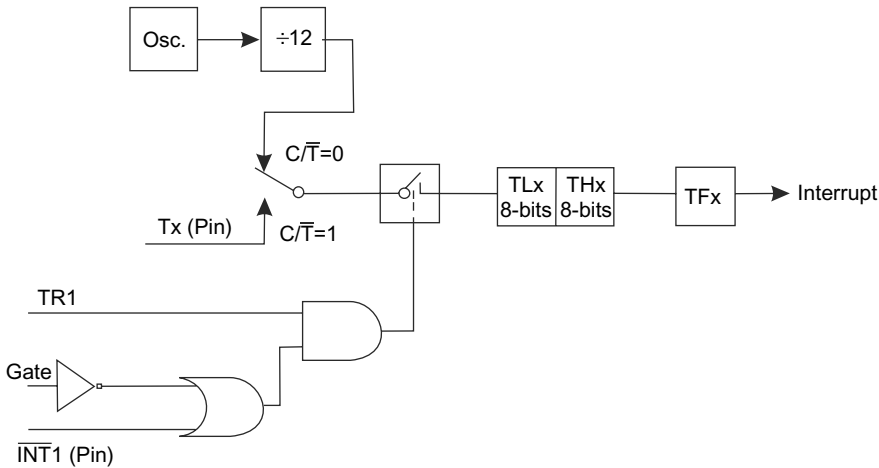


Fig. 13.33 Timer Mode 1 (16-bit timer)

Timer Mode 2

In mode 2, timer acts as an 8 bit timer, any values from 00H to FFH to be loaded into the timer's register THx. Initially THx will be loaded with the 8-bit value, after that the microcontroller copies the content of THx into TLx. Then timer starts counting. In this mode, time provides an auto-reload feature. TLx starts counting up, when TLx reaches FFH, subsequently it is incremented instead of resetting to 0, the TLx must be reset to the value which is stored in THx. Therefore, in timer mode 2, just after overflows of TLx, it is reloaded with the value i.e. the content of THx as depicted in Fig.13.34. Hence, the time delay between overflows is about

$$\frac{12 \times (256 - \text{THx})}{\text{Frequency}}$$

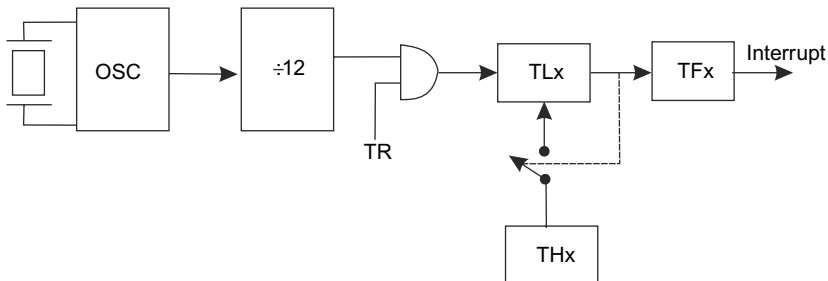


Fig. 13.34 Timer Mode 2 (8 bit timer) as auto-reload timer

Timer Mode 3 In this mode, the Timer 0 divides into two 8-bit counter/timers TL0 and TH0. TH0 and TL0 are two separate timers with overflow flags TF1 and TF0 respectively as shown in Fig.13.35. The first counter TL0 acts like mode 0 without pre-scalar. The second counter TH0 counts CPU cycles, uses TR1 (timer 1 run bit) as enable, uses TF1 (timer 1 overflow bit) as flag and uses timer 1 interrupt. When the timer 1 is in mode 3, Timer 1 works as counter stopped if it is in mode 3. Timer 1 operates in mode 0, 1, or 2 and it has gate (INT1) and external input (T1) but no flag or interrupt. Timer 1 can also be used as baud rate generator.

✓ **Counter** The timers T0 and T1 can be used as counters. The difference between the counter and timer is the source of the clock pulses to the counters. While it is used as a timer, the oscillator output pulse can be used as source of clock pulses after divide by 12. When it is used as a counter, pin T0 (P3.4) provide pulses to counter 0 and pin T1(P3.5) supplies pulses to counter 1. The C/\bar{T} bit in TMOD must be set to 1 to enable pulses from the Tx pin to reach the control circuit. Figure 13.36 shows the Timer/counter logic.

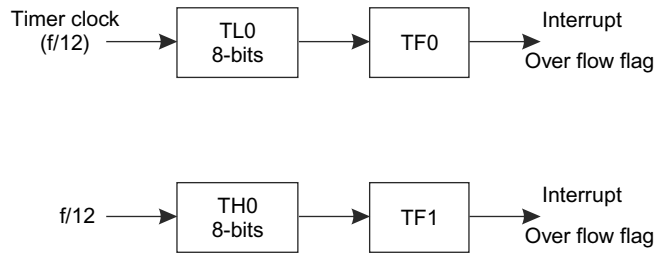


Fig. 13.35 Timer Mode 3

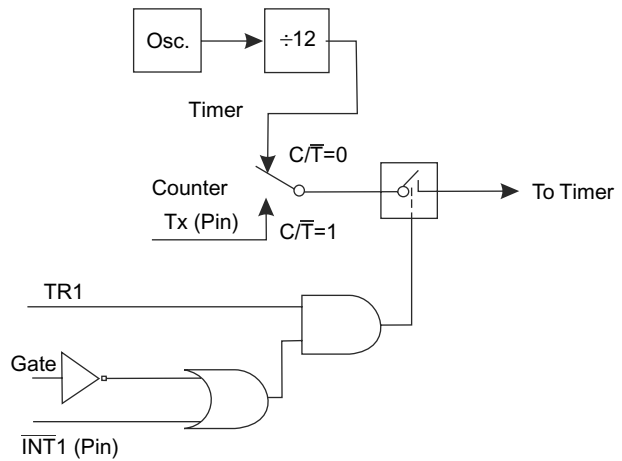


Fig. 13.36 Timer/counter logic

13.5.2 Control Registers

The timer/counter operation can be controlled by Timer Mode control (TMOD) register and Timer/counter Control (TCON) Register. In this section, the function of TMOD and TCON are explained briefly.

TCON Register The timer/counter control (TCON) register consists of control bits and flags for timers in the upper nibble and control bits and flags for the external interrupt in the lower nibble as depicted in Fig. 13.37.

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Fig. 13.37 Timer/ Counter control (TCON) register

✓ **TF1** Timer 1 overflows flag. It is set by the hardware when timer/counter 1 overflows. It is cleared by hardware as processor vectors to the interrupt service routine.

- ✓ **TR1** Timer 1 runs control bit. This is set to 1 by software program to enable timer 1 to count. It is cleared to 0 by program to halt timer.
- ✓ **TFO** Timer 0 overflows. It is set by the hardware when timer/counter 0 overflows. It is cleared when the processor vectors to execute interrupt service routine.
- ✓ **TRO** Timer 0 runs control bit. It is set to 1 by program software to enable timer 0 to count. It is cleared to 0 by software to halt timer.
- ✓ **IE1** External Interrupt 1 edge flag. This is set to 1 when a high to low (the falling) edge signal is received on port pin P3.3 (INT1). It is cleared when processor vectors to ISR.
- ✓ **IT1** Interrupt 1 type control bit. This bit is set to 1 by program to enable external interrupt 1 to be triggered by a falling edge signal. It is set to 0 by program to enable a low level signal on external interrupt 1 to generate an interrupt.
- ✓ **IE0** External Interrupt 0 edge flag. It is set to 1 by program to enable interrupt 0 to be triggered by a high to low (falling edge) signal. This is set to 0 by program to enable a low-level signal on external interrupt 0 to generate an interrupt.

TMOD Register The time node control (TMOD) register is used to set the various timer operating modes. Actually TMOD is related with the two times and can be considered to be two duplicate 4-bit registers as shown in Fig. 13.38.

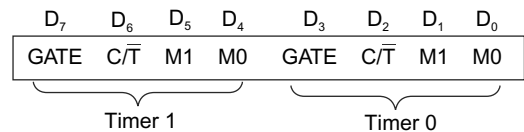


Fig. 13.38 Timer mode control (TMOD) register

Gate If TRx of TCON is set and GATE=1, timer/counter x will operate only when INTx pin is high for hardware control. When GATE=0, timer/counter x will run only if TRx =1 for software control.

C/T-bar (Clock/Timer) The C/T-bar bit in TMOD register is used to take decision whether the timer is used as a timer or an counter. If the timer or counter selector bit is cleared (C/T-bar=0), it is used as a timer to generate time delay. When C/T-bar=1, it is used as a counter by counting pulses from external input pin Tx (T1 and T0).

- ✓ **M1** Timer/counter operating mode selector bit. This bit is set or cleared by program to select mode.
- ✓ **M0** Timer/counter operating mode selector bit. This bit is set or cleared by program to select mode.

Table 13.8 Timer Operating Modes

M1	M0	Operating Modes	Functions
0	0	Mode 0	13-bit timer mode. THx as 8-bit timer/counter and TLx as 5 bit timer/counter (prescaler).
0	1	Mode 1	16-bit timer mode. THx and TLx are cascaded and there is no prescaler.
1	0	Mode 3	8-bit Auto reload timer/counter mode. THx hold a count value which is to be reloaded into TLx after each overflows.
1	1	Mode 4	Split timer mode. Timer 0 is used as two 8-bit timers. Timer 1 stopped counting and timing function is allowed. Timer 1 can be used as baud rate generator.

13.6 SERIAL COMMUNICATION

Serial communication is most commonly used either to control or to receive data from an embedded micro-processor. The advantage of serial communication is that the number of wires required is less as compared to that in parallel communication. Serial communication is a form of I/O in which the bits of a byte begin transferred appear one after the other in a timed sequence on a single wire. Figure 13.39 shows the serial communication through telephone line where P/S is parallel in serial out shift register, S/P Serial in parallel out shift register, D/A digital to analog converter and A/D is analog to digital converter.

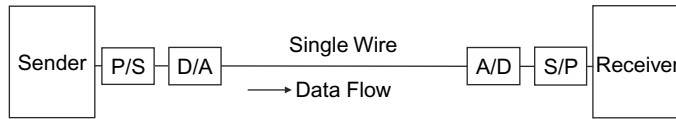


Fig. 13.39 Serial communication through single wire

There are two methods of serial communications, such as synchronous and asynchronous communications. In synchronous communication, transfer a block of data at a time, but in asynchronous communication transfer a single byte at a time. Software can be used for synchronous and asynchronous communications, but the programs can be tedious and long. Therefore hardware such as UART and USART are developed. Usually UART (Universal Asynchronous Receiver Transmitter) or USART (Universal Synchronous Asynchronous Receiver Transmitter) are used in serial communication. The 8051 microcontroller has a build in UART.

8051 support a full duplex serial port (UART). 8051 has T x D and R x D pins for transmission and receive serial data respectively. The function of serial port is to perform parallel to serial conversion for data output and serial to parallel conversion for data input. The block diagram of UART is shown in Fig. 13.40.

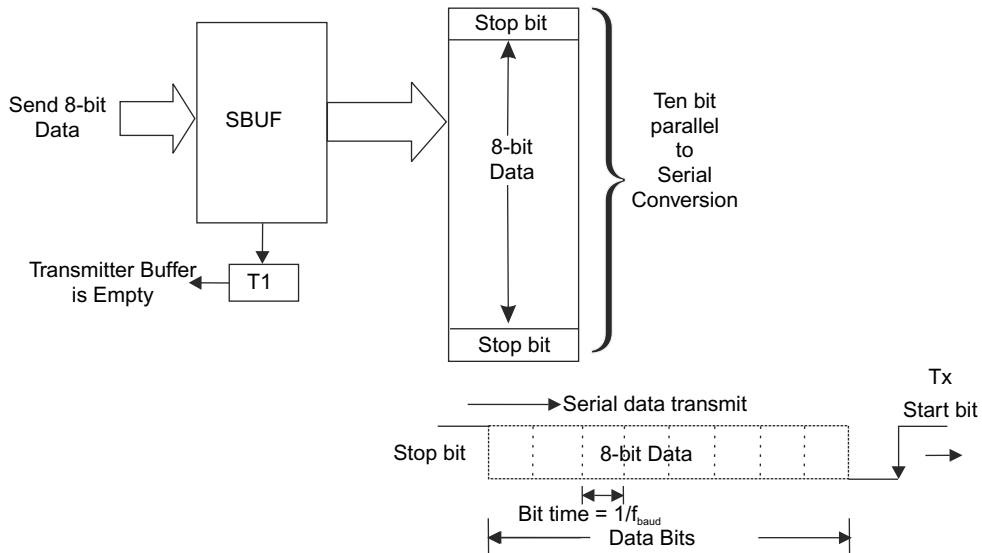


Fig. 13.40 (a) Transmitter half (contd.)

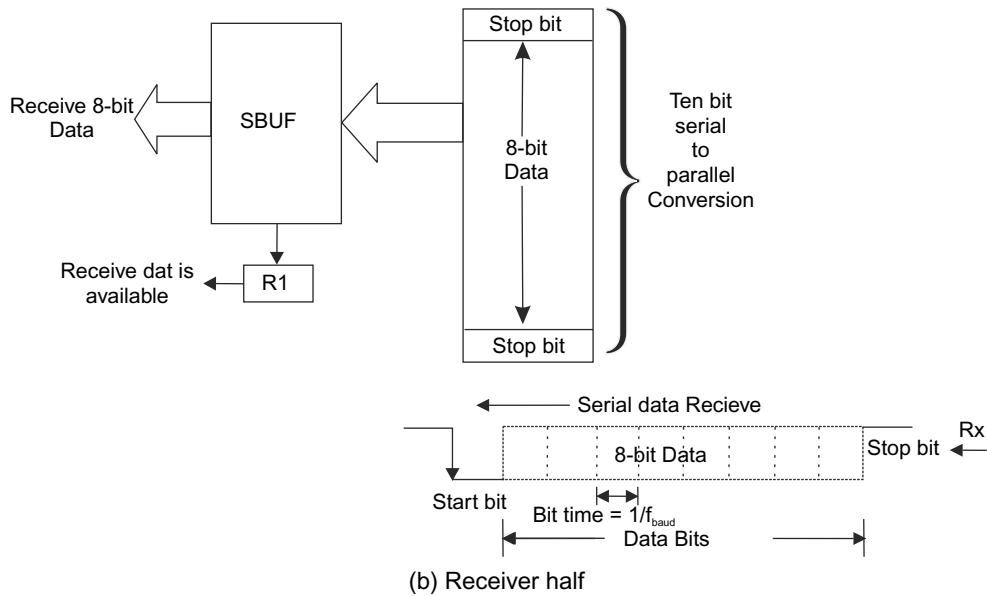


Fig. 13.40 Block diagram of UART (a) Transmitter half, and (b) Receiver half

The UART can be used for 9-bit data transmission and receive where 8-bits represent the data byte (information of character) and the 9th bit is the parity bit. A block diagram of UART transmitter is depicted in Fig. 13.41 where the 9th bit is used as the parity bit. Figure 13.42 shows the block diagram of UART receiver where the 9th bit is used as the parity bit.

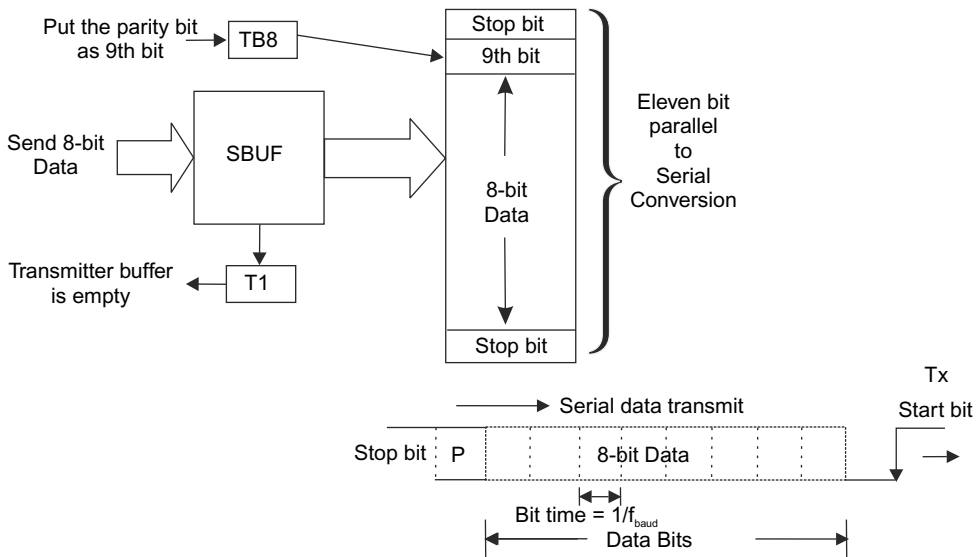


Fig. 13.41 Block diagram of UART transmitter for 11-bit transmission

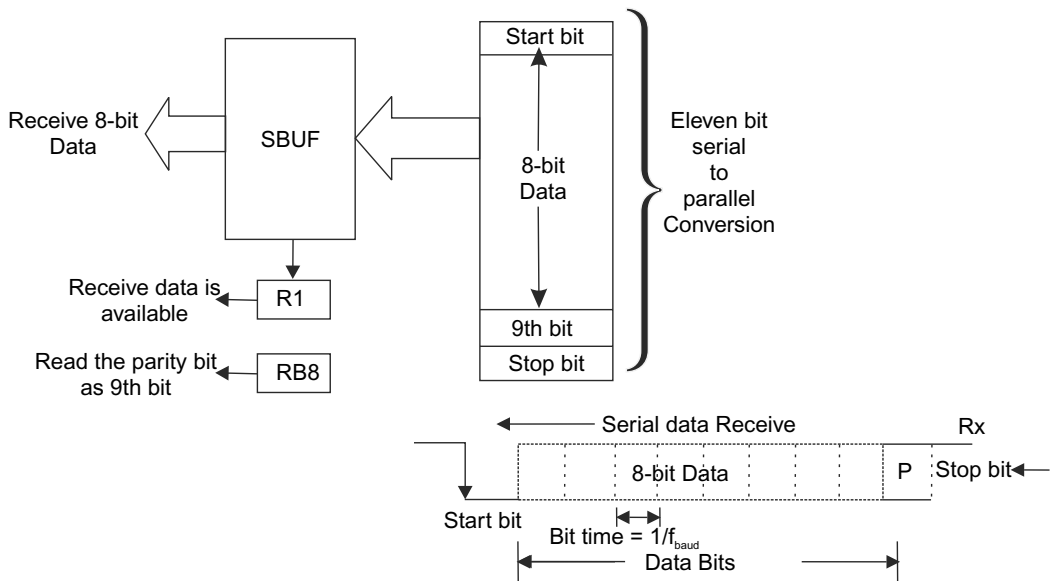


Fig. 13.42 Block diagram of UART receiver to receive 11 bits

The 8051 serial communication can support RS232. RS232 is not compatible to TTL. Therefore, to connect any RS232 to a microcontroller, we must use voltage converters such as MAX232 to convert TTL logic level to the RS232 voltage levels as shown in Fig.13.43. The MAX232 IC is also known as line driver. The 8051 microcontroller has two pins such as TxD and RxD which are used for transferring and receiving data serially. TxD and RxD pins are the part of Port 3 (P3.0 and P3.1). These pins are TTL compatible and a line driver is required to make these pins RS232 compatible. The serial communication between two microcontrollers and between microcontroller and microprocessor is also possible.

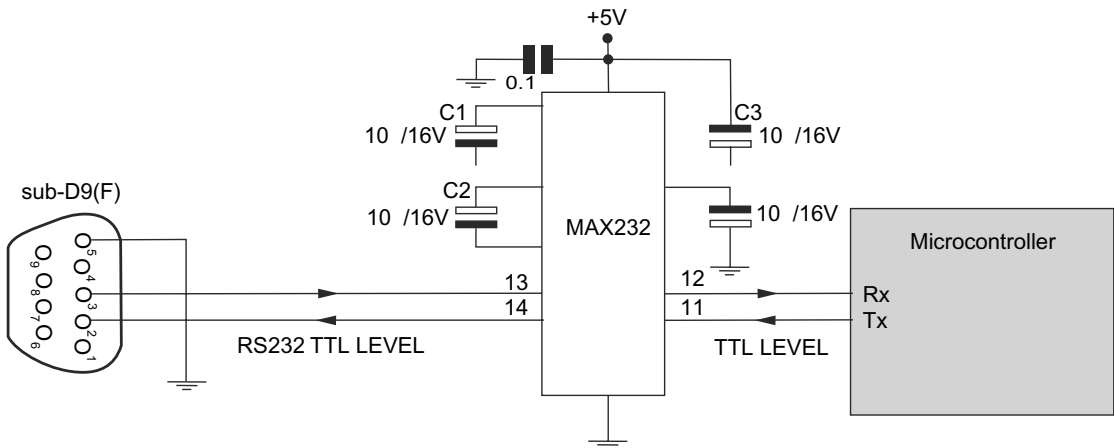


Fig. 13.43 MAX232 IC as a line driver

13.6.1 Registers for Serial Data Communication

The 8051 microcontroller use the following registers for serial data communication:

- ◆ SBUF(Serial Port data buffer)
- ◆ SCON(Serial port control) register
- ◆ PCON(Power control) register

SBUF (Serial Port data buffer) The serial port data buffer register has two registers. One register is used to hold data that to be transmitted through TxD of 8051 and it is write only type. The other register can able to hold data from external sources through RxD of 8051 and it is read only type.

SCON (Serial port control) Register The format of SCON (serial port control) register is shown in Fig. 13.44.

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Fig. 13.44 SCON register

Table 13.9 SCON Register

Bit	Name	Bit Address	Function
7	SM0	9FH	Serial port mode bit 0. This bit is set/cleared by program to select operating mode as shown in Table 1.1.
6	SM1	9EH	Serial port mode bit 1. This bit is set/cleared by program to select operating mode as shown in Table 1.1.
5	SM2	9DH	This pin enables the mutliprocessor communication feature in modes 2 and 3. In mode 2 and 3, if SM2=1, then RI will not be activated, if the received 9th data bit RB8 is 0. In Mode 0, SM2 must be 0. In mode 1, if SM=1, then RI will not be activated if a valid stop bit is not received.
4	REN	9CH	Receiver Enable bit. This bit must be set in order to receive characters. This bit must be cleared to disable reception.
3	TB8	9BH	Transmit bit 8. The 9th bit will be transmitted in mode 2 and 3. This bit can be set/cleared by software.
2	RB8	9AH	Receive bit 8. The 9th bit will be received in mode 2 and 3. In mode 1, if SM2=0, RB8 is the stop bit that was received. In mod 0, RB8 is not used.
1	TI	99H	Transmit Interrupt Flag. This bit is set by hardware at the end of the 8th bit time in mode 0. This can also be set by hardware at the beginning of the stop bit in other modes. This bit can be cleared by software.
0	RI	98H	Receive Interrupt Flag. This bit is set by hardware at the end of the 8th bit time in mode 0 or halfway through the stop bit in other modes. This bit can be cleared by software.

SM0	SM1	Serial Mode	Description	Baud Rate
0	0	0	8-bit Shift Register	Oscillator frequency/ 12
0	1	1	8-bit UART	Variable which is set by Timer 1
1	0	2	9-bit UART	Oscillator frequency/ 32 or Oscillator frequency / 64
1	1	3	9-bit UART	Variable which is set by Timer 1

PCON (Power control) Register

The format of power control register is shown in Fig.13.45

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
SMOD	-	-	-	GF1	GF0	PD	IDL

Fig. 13.45 PCON register

- ✓ **SMOD** Double baud-rate bit. When this bit is set to 1, timer 1 is used to generate baud rate and the baud rate is doubled while the serial port is used in modes 1, 2 or 3.
- ✓ **GF1** General-purpose flag bit
- ✓ **GF0** General-purpose flag bit
- ✓ **PD** Power down bit. When this bit is set, power down operation in 8051 is performed. This is available only in CHMOS processors.
- ✓ **IDL** Idle mode bit. If this bit is set, it activates idle mode operation in 8051. This is available only in CHMOS processors.

13.6.2 Serial Communication Modes

There are four serial communication modes such as

- Mode 0—Shift register mode
- Mode 1—Standard UART mode
- Mode 2—Multiprocessor mode
- Mode 3—9 bit UART mode

The above modes can be selected by the programmer by proper setting the mode bits SM0 and SM1 in SCON register.

Mode 0

This mode is known as shift register mode. When SM0 and SM1 are set to 00, the serial port data buffer (SBUF) receives and transmit data through the RxD pin. TxD pin outputs the shift clock only. RxD pin is connected to the internal shift frequency clock pulse to provide shift pulses to external circuits. In this mode eight data bits are transmitted or received. The shift frequency or baud rate is fixed and it can be determined from the system clock frequency. When the oscillator frequency is f_{osc} , the baud rate can be expressed as $\frac{f_{osc}}{12}$. For a 12 MHz crystal, the baud rate is 1 MHz. The transmission operation is

initiated by executing instructions to write data to SBUF. Then data can be shifted out on RxD line when the clock pulse is applied through TxD line. The receiving operation is initiated when REL=1 and RI=0. REN is set at the beginning of the program and then RI is cleared to start a data input operation. Figure 13.46 shows the data transmission/reception in mode 0.

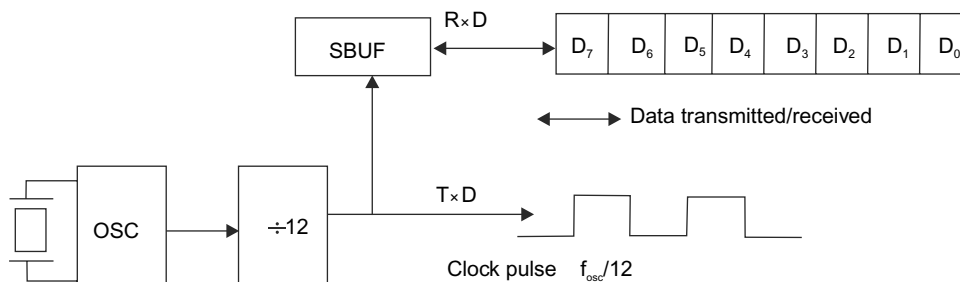


Fig. 13.46 Data transmission/receive in mode 0

Mode 1

When SM0 and SM1 are set to 01, mode 1 operation is performed. In this mode, 10 bits are transmitted through TxD pin or received through RxD pin. These 10 bits consists of one start bit (0), 8 data bits (LSB first) and a stop bit (1) as shown in Fig. 13.47. The transmit interrupt flag TI is set once after sending 10 bits. Each bit interval is the inverse of the baud rate frequency and each bit must be maintained high or low over this interval. After receiving, the stop bit goes to RB8 in SCON register. The baud rate is variable and it is computed by the timer 1 overflow rate. The baud rate can be expressed as

$$\text{Baud rate} = \frac{2^{\text{SOD}}}{32 \times \text{Timer 1 overflow rate}}$$

When the timer 1 operates in auto-reload mode or mode 2 with reload count value in TH1, after each over flow the content of TH1 must be loaded into TL1. In this mode of operation, the high nibble of TMOD is 0010B. The baud rate can be expressed as

$$\text{Baud rate} = \frac{2^{\text{SOD}} \times \text{oscillator frequency}}{32 \times 12 \times (256 - [\text{TH1}])}$$

For example, if the contents of TH1 are 230_D and the SOD bit in PCON is 0, the baud rate is 1201 baud or 1.2K(approx) for 12 MHz oscillator frequency.

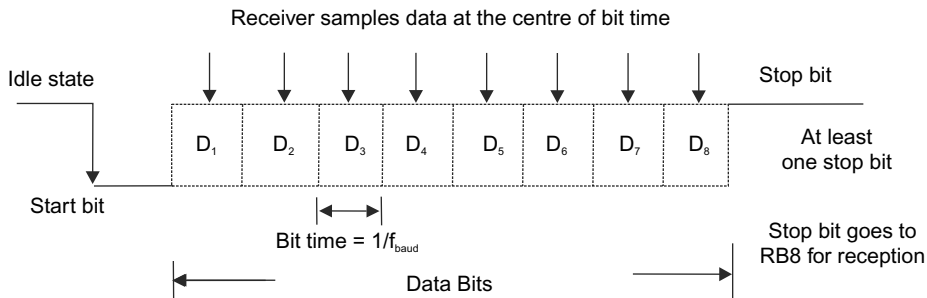


Fig. 13.47 Data transmission/receive in mode 1

Mode 2

In this mode, serial operates as a 9-bit UART and 11 bits are transmitted or received. These 11 bits are a start bit (always 0), 8 data bits (LSB first), a programmable 9th bit and a stop bit (always 1) as shown in Fig. 13.48. Programmer can define 9th bit as TB8 in SCON and it can be used as the parity bit of data byte. On reception, the 9th bit is placed into RB8 in SCON. In mode 2, the bit SMOD in PCON and oscillator frequency determine the baud rate and it can be expressed as

$$\text{Baud rate} = \frac{2^{\text{SOD}}}{64} \times (\text{oscillator frequency})$$

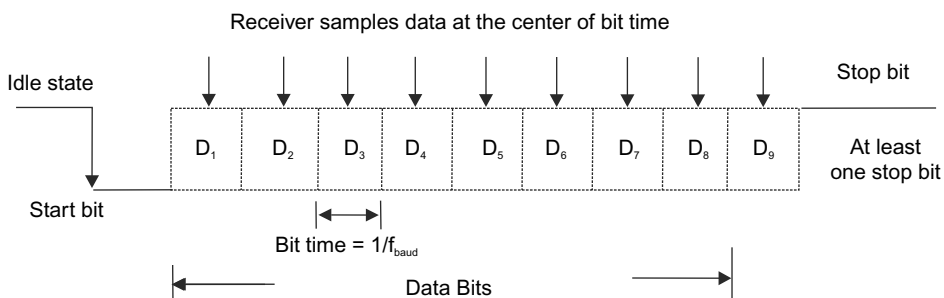


Fig. 13.48 Eleven bits transmitted in 8051 serial communication mode

$$\text{If SMOD} = 0, \text{ Baud rate} = \frac{2^0}{64} \times (\text{oscillator frequency}) = \frac{\text{Oscillator frequency}}{64}$$

$$\text{If SMOD} = 1, \text{ Baud rate} = \frac{2^1}{64} \times (\text{oscillator frequency}) = \frac{\text{Oscillator frequency}}{32}$$

Mode 3

In this mode, serial port operates as a 9-bit UART with variable baud rate and 11-bits are transmitted or received. This operating mode is same as mode 2 except baud rate is programmable through the timer 1 overflow rate. The baud rate calculations are same as that of mode 1 and it can be expressed as

$$\text{Baud rate} = \frac{2^{\text{SMOD}} \times \text{oscillator frequency}}{32 \times 12 \times (256 - [\text{TH1}])}$$

13.6.3 Multiprocessor Communication

8051 microcontrollers can be operating in multiprocessor mode for serial communication mode 2 and mode 3. In this mode, there is a master processor (master microcontroller) which can communicate with more than one slave processors (slave microcontrollers) as shown in Fig. 13.49. SM2 bit in SCON register is used as a flag for multiprocessor communication. Whenever a byte has been received, the 8051 will set the RI (Receive Interrupt) flag. Consequently, the program knows that a byte has been received and it is required to process data.

In multiprocessor mode, 9-bits are transferred or received. When SM2 is set, the RI flag will be triggered. If the 9th bit is cleared, The RI flag will never be set. Generally, the 9th bit is kept clear so that the RI flag is set after receiving any character. In serial communication Modes 2 and 3, the transmitting processor is used as a master

8051 which can control several slave 8051 microcontroller. The TxD outputs of the slave microcontrollers are joined together and connected to the RxD input of the master microcontroller. The RxD inputs of the slaves are tied together and connected to the TxD output of the master. Each slave microcontroller is assigned a specified address. When the master wants to transmit a block of data, it must send first the address byte of the slave. While transmitting address byte by the master, the 9th bit is '1' and 9th bit '0' during data bytes transfer.

Whenever the master 8051 transmits an address byte, all slave 8051 microcontrollers are interrupted. Then slave microcontrollers check to observe if they are being addressed or not. Subsequently, the addressed slave clear its SM2 bit and wait to receive the data bytes. Other slaves who are not addressed can continue their operations ignoring the incoming data bytes and they will be interrupted again while the next address byte is transmitted by master controller. Usually, the master communicates with one slave at a time and transmit 11 bit which consists of one start bit(0), 8-data bits(LSB first) TB8 and a stop bit(1). The TB8 is '1' for address byte and '0' for a data byte.

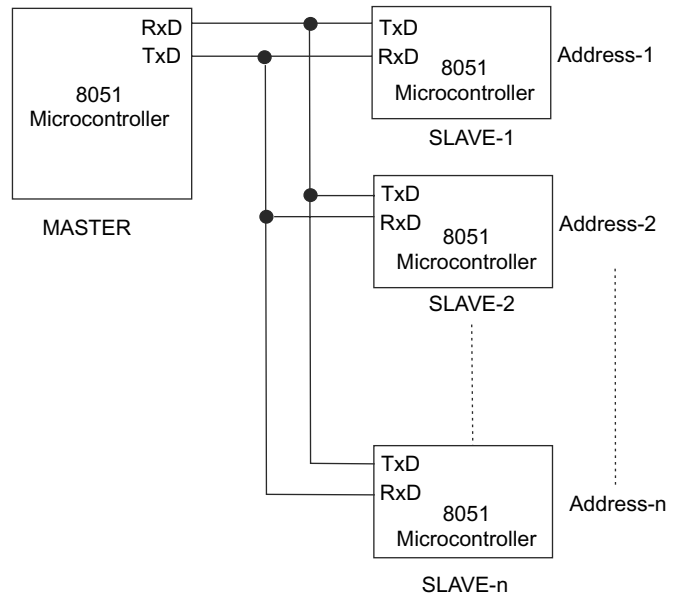


Fig. 13.49 Multiprocessor communication

When the master microcontroller wants to communicate with a slave microcontroller, it sends the address of the slave with TB8=1. Then all slave microcontrollers receive this address. Initially SM2 bit set to '1'. As all slave microcontrollers check the address to observe if they are being addressed or not. Then the selected slave microcontroller byte clear its SM2 bit to '0' so that data can be received. In mode 2 and 3, Receive Interrupt (RI) flag is set if REN=1 and RI=0, SM2=1 and RB8=1 and a valid stop bit is received. After proper communication between master microcontroller and a slave microcontroller, the data bytes will be send by the master with TB8=0.

13.7 INTERRUPTS

An interrupt is the occurrence of internal and external events that interrupts the micro-controller to provide service any device. In case of external events, the status of microprocessor pin is altered. In internal events, interrupts are generated due to timer overflow or transmission/reception of a byte through the serial port. After receiving an interrupt signal, the microcontroller interrupts the execution of main program. After saving the current status, the microprocessor jump to the memory location specified by interrupt and executes a subprogram called interrupt service routine (ISR). This memory location is called vector. Hence the interrupt is known as vector interrupt. After provide service to the interrupt, microprocessor restores the original status and continue to execute main program again.

13.7.1 Interrupts in 8051

There are five interrupt sources for the 8051 microcontroller. The priority wise five different interrupts of 8051 microcontroller are given below:

- ◆ External Interrupt 0
- ◆ Timer 0
- ◆ External Interrupt 1
- ◆ Timer 1
- ◆ Serial Port

These interrupts can recognize 5 different events that can interrupt regular program execution.

- ◆ Each interrupt can be enabled separately.
- ◆ Each interrupt type has a separate vector address.
- ◆ Each interrupt type can be programmed to one of two priority levels.
- ◆ External interrupts can be programmed for edge or level sensitivity.
- ◆ Each interrupt can be enabled or disabled by setting bits of the IE (interrupt enable) register. Likewise, the whole interrupt system can be disabled by clearing the EA bit of the same register as shown in Fig. 13.50.

In 8051 microcontroller, interrupts are generated by internal operations such as Timer flag 0 (TF0), Timer flag 1 (TF1), and serial port interrupt (RI or TI). When the timer/counter 0 overflows, the TF0 flag is set to 1. If the timer/counter 1 overflows, the TF1 flag is set to 1. The vector address of TF0 and TF1 are 000BH and 001BH respectively. The TF0 and TF1 flag will be cleared when the timer flag interrupt makes a program call from the timer subroutine. In serial port interrupt, a data byte will be received if RI=1 and a data byte will be transmitted if TI=1. The vector address of RI or TI is 0023H. Whenever RI or TI becomes 1, the 8051 microcontroller is interrupted and jumps to the memory location 0023H to execute the Interrupt Service Routine (ISR).

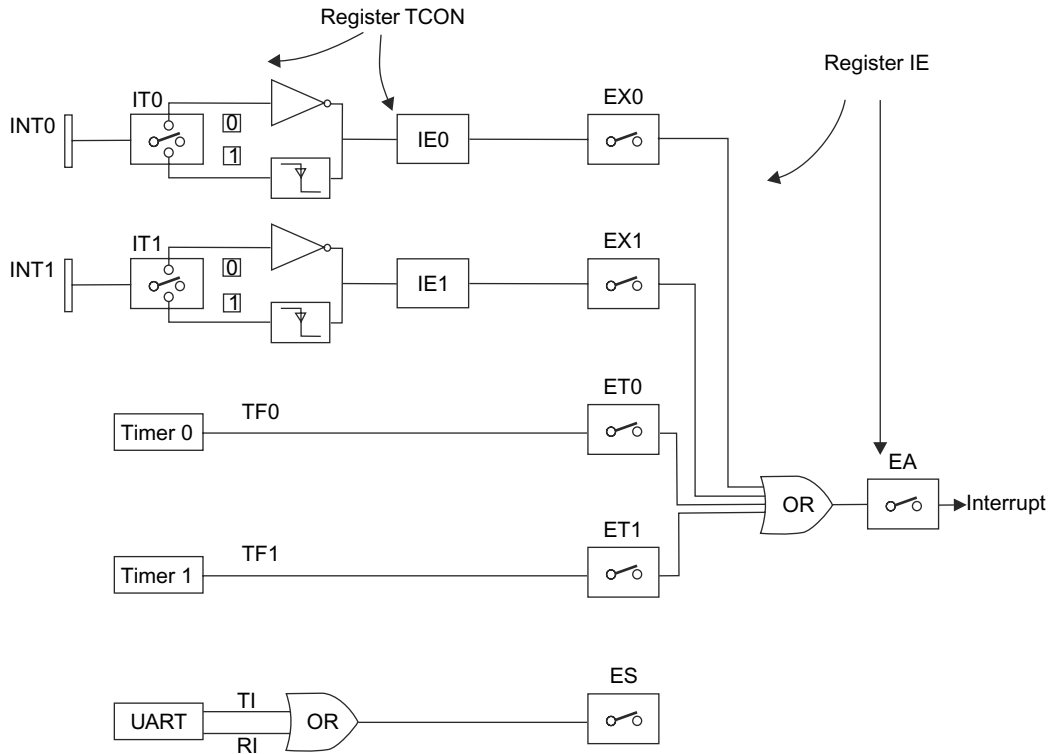


Fig. 13.50 Interrupts of 8051

Interrupts are also generated by external signals INT0 and INT1. The INT0 and INT1 are located on pins P3.2 and P3.3 respectively. External inputs on INT0 and INT1 pins set the interrupt flags IE0 and IE1 in the TCON register to 1 by level triggered or edge triggered. If the IT0 and IT1 bits of the TCON register are set, an interrupt will be generated on high to low transition, i.e. on the falling pulse edge. If these bits are cleared, an interrupt will be continuously executed as far as the pins are held low. The vector address of external interrupt 0 and external interrupt 1 are 0003H and 0013H respectively.

Table 13.10 Interrupt Vector Addresses

<i>Interrupt Source</i>	<i>Flag</i>	<i>Vector Address</i>
External Interrupt 0	IE0	0003H
Timer 0	TF0	000BH
External Interrupt 1	IE1	0013H
Timer 1	TF1	001BH
Serial Port	RI&TI	0023H

13.7.2 Interrupt Control Register

All interrupt operations are controlled by software. The programmer should program the control bits in following registers

- ◆ Interrupt Enable (IE) Register
- ◆ Interrupt Priority(IP) register and
- ◆ Timer Control Register(TCON)

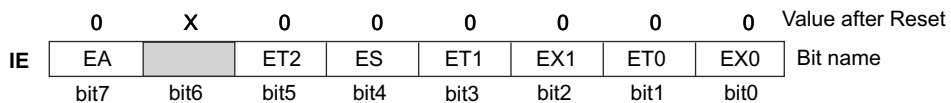
In this section IE and IP registers are explained.

Interrupt Enable (IE) Register

IE is Interrupt Enable Register which is shown in Fig. 13.51.

The function of EA, ES, ET1, EX1, ET0 and EX0 are given below:

- ✓ **EA** global interrupt enable/disable:
 - 0—disables all interrupt requests.
 - 1—enables all individual interrupt requests.
- ✓ **ES** enables or disables serial interrupt:
 - 0—UART system cannot generate an interrupt.
 - 1—UART system enables an interrupt.
- ✓ **ET1** bit enables or disables Timer 1 interrupt:
 - 0—Timer 1 cannot generate an interrupt.
 - 1—Timer 1 enables an interrupt.
- ✓ **EX1** bit enables or disables external 1 interrupt:
 - 0—change of the pin INT0 logic state cannot generate an interrupt.
 - 1—enables an external interrupt on the pin INT0 state change.
- ✓ **ET0** bit enables or disables timer 0 interrupt:
 - 0—Timer 0 cannot generate an interrupt.
 - 1—enables timer 0 interrupt.
- ✓ **EX0** bit enables or disables external 0 interrupt:
 - 0—change of the INT1 pin logic state cannot generate an interrupt.



Bit	Name	Bit Address	Function
7	EA	AFH	Global interrupt enable/disable
6	-	AH	Undefined
5	-	ADH	Undefined
4	ES	ACH	Enable serial interrupt
3	ET1	ABH	Enable Timer 1 interrupt
2	EX1	AAH	Enable External 1 interrupt
1	ET0	A9H	Enable Timer 0 interrupt
0	EX0	A8H	Enable External 0 interrupt

Fig. 13.51 Interrupt Enable (IE) Register

Interrupt Priority (IP) Register

The Interrupt Priority (IP) Register is used to determine the interrupt priority. Figure 13.52 shows the bit addressable IP register. When the bit is 0, the corresponding interrupt has lowest priority and if the bit is 1, the corresponding interrupt has the higher priority. When two interrupts occur at the same time, the higher priority interrupt gets service fast and then the next higher priority interrupt gets service. The priority of interrupts is given below:

- ◆ IE0 (External Interrupt 0)
- ◆ TF0 (Timer Flag 0)
- ◆ IE1 (External Interrupt 1)
- ◆ TF1 (Timer Flag 1)
- ◆ RI/TI (Serial Port)

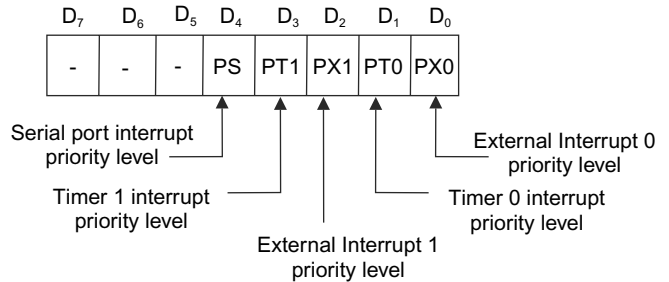


Fig. 13.52 IP register

13.7.3 Execution of Interrupt

Assume that the microcontroller is executing the main program and the external interrupt INT1 occurs. The 8051 microcontroller complete the execution of current instruction and save the address of the next instruction, i.e. the content of program counter (PC) to the stack. The current status of all the interrupts i.e. the content of IE register is also saved to the stack. The IE1 flag is disabled so that another INT1 interrupt will be inhibited.

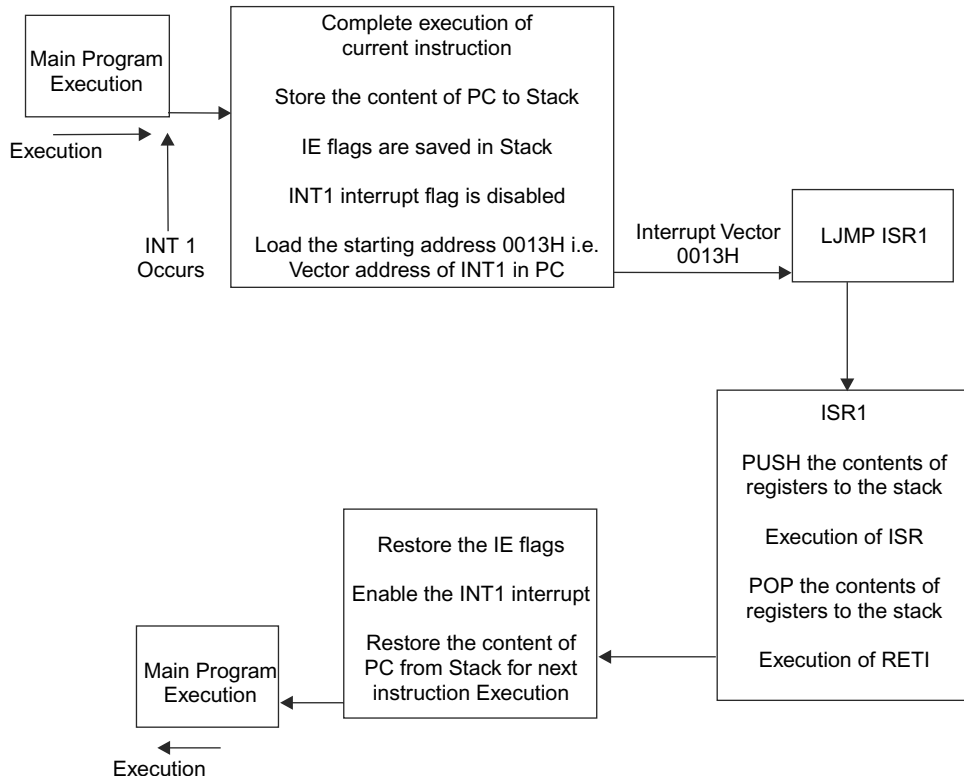


Fig. 13.53 The sequence of interrupt operation

Then the program counter is loaded with the vector location 0013H which is the predefined address of INT1. therefore the program execution has been transferred to the memory location 0013H. A LJMP instruction is programmed at the memory location. Consequently, the program execution jump to the specified starting address of Interrupt Service Routine (ISR).

The ISR is written by the programmer and this subprogram states what operation will be performed by the INT1 interrupt. During execution of ISR, initially PUSH the contents of registers to stack and execute the subprogram part. After execution of the subprogram part, it is required to restore or POP the contents of these registers. The last instruction in ISR is RETI (Return from interrupt) instruction. When RETI instruction is executed, 8051 should restore the content of IE register, enable INT1 flag and also restore the content of program counter (PC) from the stack. As the PC contains the address of next instruction, 8051 microcontroller starts to execute the next instruction of main program. Fig.13.53 shows the sequence of operations when the microcontroller receive an interrupt.

SUMMARY

- In this chapter, the architecture and features of 80C51 microcontrollers are described.
- All registers are discussed with their applications. Program status word, accumulator, B register, register banks are also discussed. The flags of microcontroller (PSW) are also explained.
- Special Function Registers (SFRs) related to timer/counters, I/O and serial operations are just introduced briefly. Pointer registers, stack pointer, DPTR and PC are also discussed.
- The schematic pin diagram and function of each pin and memory organization of the 80C51 microcontroller are incorporated in this chapter.

MULTIPLE-CHOICE QUESTIONS

- 13.1 The 8051 microcontroller has
- (a) 8-bit data bus and 16-bit address bus
 - (b) 16-bit data bus and 8-bit address bus
 - (c) 8-bit data bus and 8-bit address bus
 - (d) 16-bit data bus and 16-bit address bus
- 13.2 The 8051 microcontroller has
- (a) 4K bytes of on-chip ROM
 - (b) 8K bytes of on-chip ROM
 - (c) 16K bytes of on-chip ROM
 - (d) 32K bytes of on-chip ROM
- 13.3 The 8051 microcontroller has
- (a) three on-chip Timers
 - (b) two on-chip Timers
 - (c) one on-chip Timer
 - (d) four on-chip Timers
- 13.4 The 80C51 microcontroller family has
- (a) 32 pins for I/O
 - (b) 24 pins for I/O
 - (c) 16 pins for I/O
 - (d) 8 pins for I/O
- 13.5 The 8051 microcontroller can support
- (a) 5 interrupts
 - (b) 4 interrupts
 - (c) 3 interrupts
 - (d) 2 interrupts
- 13.6 A 80C51 microcontroller has
- (a) 128 bytes of on-chip RAM
 - (b) 256 bytes of on-chip RAM
 - (c) 228 bytes of on-chip RAM
 - (d) 556 bytes of on-chip RAM
- 13.7 A microcontroller has
- (a) 4 on-chip I/O ports
 - (b) 3 on-chip I/O ports

- (c) 2 on-chip I/O ports
(d) 1 on-chip I/O ports
- 13.8 The number of flags present in 8051 that respond to math operations are
(a) 2 (b) 3
(c) 4 (d) 5
- 13.9 Which of the following are 16 bit registers in the 80C51 microcontroller?
(a) DPTR (b) IE
(c) TMOD (d) PC
- 13.10 Which of the following registers can be used to hold the address of a byte in the memory of 80C51?
(a) DPTR (b) PCON
(c) SBUF (d) PSW
- 13.11 Which of the following registers can be used as two individual 8-bit registers?
(a) DPTR (b) PC
(c) SBUF (d) PSW
- 13.12 How many general-purpose registers exist in 80C51?
(a) 10 (b) 16
(c) 20 (d) 32
- 13.13 The operation of PSW is
(a) hold the status of register bank currently being used
(b) holding data during data transfer operation
(c) holding math flags
(d) hold address of a byte in memory
- 13.14 Which of the following registers is bit addressable?
(a) SBUF (b) TMOD
(c) PCON (d) PSW
- 13.15 Which port can only be used as I/O port?
(a) Port 0 (b) Port 1
(c) Port 2 (d) Port 3

SHORT-ANSWER QUESTIONS

- 13.1 What are microcontroller families?
- 13.2 What are the advantages of microcontroller-based systems over microprocessor-based systems?
- 13.3 Give a list of applications of microcontrollers.
- 13.4 What are the features of the Intel 80C51 microcontroller?
- 13.5 What are the general-purpose registers of 80C51?
- 13.6 What are the sizes of RAM in 8051, 8052 and 8031?
- 13.7 What are the sizes of ROM in 8051, 8052 and 8031?

REVIEW QUESTIONS

- 13.1 Define microcontroller. Write the differences between microprocessors and microcontrollers.
- 13.2 Draw the block diagram of the 8051 microcontroller and explain the operation of each block briefly.
- 13.3 Draw the schematic pin diagram of the 8051 microcontroller and explain the operations of the following pins:
(i) RST (ii) T×D (iii) R×D (iv) XTAL2 (v) ALE (vi) \overline{EA} (vii) \overline{PSEN} (viii) \overline{RD} (x) \overline{RW}
- 13.4 What is the difference between internal and external program memory? Why is external program memory used in a microcontroller? How can \overline{EA} be used to access internal and external program memory?

Chapter 14

Instruction Set and Programming of the 8051 Microcontroller

14.1 INTRODUCTION

In Chapter 13, the basic structure of 8051 microcontroller has been discussed elaborately. Like 8085 and 8080 microprocessors, the 8051 microcontroller has different addressing modes to locate operand in the instructions. In this section, all addressing modes of a microcontroller has been discussed with examples. This microcontroller has arithmetic and logical instructions, data transfer, Boolean operation instructions, bit operation instructions, branch control instructions and program control instructions. All these instructions are explained with appropriate examples. The programming format and some simple programs such as addition, subtraction, multiplication, division, ascending order, descending order, look-up table, keyboard interface, A/D converter interface, traffic control, washing machine control and stepper motor control have been incorporated in this chapter to understand the applications of instructions.

14.2 ADDRESSING MODES

An instruction is used to load or transfer data from a source to a destination. The *source* may be any register, internal memory, external memory, any one of four ports or any external I/O peripheral devices. Similarly, *destination* may be any register, memory (internal or external) and I/O devices. In any instruction of the 8051 microcontroller, the data is known as *operand*. The way in which an operand is specified is called an *addressing mode*. There are different ways to specify operands for instructions. The commonly used addressing modes of 8051 microcontroller are as follows:

- ◆ Immediate addressing
- ◆ Register addressing mode
- ◆ Direct addressing
- ◆ Indirect addressing
- ◆ Indexed addressing
- ◆ Relative addressing
- ◆ Absolute addressing
- ◆ Long addressing

14.2.1 Immediate Addressing

In immediate addressing mode, the source operand is a constant rather than a variable. The constant operand can be incorporated into the instruction as a byte of immediate address. The immediate operands are preceded by a # sign in assembly language. The operand may be a numeric constant (decimal or hexadecimal), a symbolic variable or an arithmetic expression.

Example MOV A, #FFH; This instruction is used to load the immediate data FF H to A register.

MOV R0, #26; This is used to load the immediate data byte 26H to register R0.

All instructions using immediate addressing use an 8-bit data field. But one exception is that a 16-bit constant is required for initialization of the Data Pointer Register (DPTR). For example, MOV DPTR, #9000H. After execution of the MOV DPTR, #9000H instruction, 9000H will be loaded into DPTR register. Table 14.1 shows some other examples of immediate addressing:

Table 14.1 Examples of immediate addressing instructions

<i>Instruction</i>	<i>Task</i>
ADD A, # data	Add immediate data to accumulator
SUBB A, # data	Subtract immediate data from accumulator with borrow
MOV R _n , # data	Move immediate data to register R _n
MOV DPTR, #data 16	Load data pointer register with a 16-bit constant

14.2.2 Register Addressing Mode

In register addressing mode, the selected register bank containing registers R0 through R7 can be accessed by certain instructions which carry a 3-bit register specification within the opcode of the instruction. As the three least significant bits of the instruction opcode are used to specify a register, this addressing mode eliminates an address byte. When the instruction is executed, one of the eight registers in the selected bank will be accessed. One of four banks is selected at execution time by the two bank select in the PSW.

Example MOV A, R0 Move the content of R0 register into accumulator

MOV R1, A Move the content of accumulator into R1 register

Some instructions are specific to a certain register. For example, some instructions always operate on the accumulator, or data pointer and no address byte is required to point to it. The opcode itself specifies the source of operand and an example is INC A. In this instruction, the accumulator itself is the operand. The examples of other register addressing instructions are given in Table 14.2.

Table 14.2 Examples of register addressing instructions

<i>Instruction</i>	<i>Task</i>
ADD A, R _n	Add the content of register R _n to accumulator.
SUBB A, R _n	Subtract the content of register from accumulator with borrow.
MOV R _n , A	Move data from accumulator to register R _n .
INC DPTR	Increment data pointer register by one.

14.2.3 Direct Addressing

In direct addressing mode, the operand is specified by an 8-bit address field in the instruction. Only the lower 128 bytes of internal data RAM and SFRs can be directly addressed by using a single-byte address.

Example MOV A, 33H This instruction is used to transfer the content of internal memory (RAM)

location 33H to accumulator as shown in Fig. 14.1.

MOV 32, R1 The content of register R1 moves to internal memory location 32H as depicted in Fig. 14.1.

Table 14.3 shows the some other examples of direct addressing.

Table 14.3 Examples of direct addressing

<i>Instruction</i>	<i>Task</i>
ADD A, R _n	Add the content of register R _n to accumulator
SUBB A, R _n	Subtract the content of register from accumulator with borrow
MOV R _n , A	Move data from accumulator to register R _n
INC DPTR	Increment data pointer register by one

14.2.4 Indirect Addressing

In indirect addressing mode, the instruction specifies a register which contains the address of the operand. Both internal and external RAM can be indirectly addressed. In this mode, R0 or R1 of selected bank or the stack pointer may operate as pointer registers for 8-bit addresses. Actually, the content of R0 or R1 indicates an address in internal RAM where data will be stored or read. In assembly-language programming, indirect addressing is presented by a @ symbol before R0 or R1.

Example MOV A, @R7

In this instruction, the content of R7 represents the internal memory address. As the R7 contains 33H, the internal memory location will be 33H. After the execution of this instruction, the value of the internal memory location 34H will be loaded into the accumulator as depicted in Fig. 14.2.

The data pointer register can also be used as the address register for a 16-bit address. Some examples of indirect addressing are illustrated in Table 14.4.

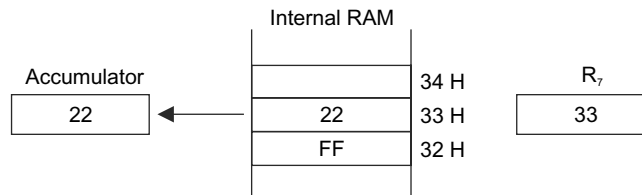


Fig. 14.2 Indirect addressing

Table 14.4 Examples of indirect addressing

<i>Instruction</i>	<i>Task</i>
ADD A, @R ₀	Add the contents of the address specified by the R ₀ register to accumulator
SUBB A, @R ₁	Subtract the contents of the address specified by the R ₁ register from accumulator with borrow
MOV @R _i , A	Move the content of accumulator to indirect RAM specified by R _i (R ₀ or R ₁)
MOV A, @R _i	Moves a byte of data from internal RAM at location whose address is in R _i (R ₀ or R ₁) to the accumulator
DEC @R _i	Decrement indirect RAM specified by R _i (R ₀ or R ₁)

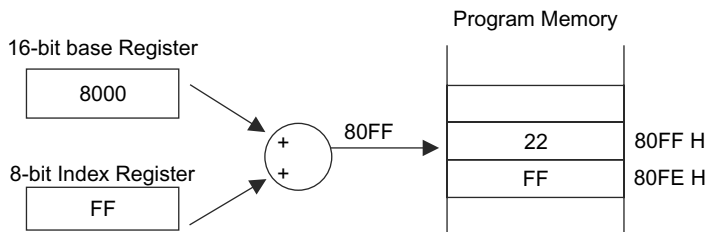
14.2.5 Indexed Addressing

In indexed addressing, only the program memory can be accessed and it can only be read. This addressing mode is used for reading look-up tables in program memory. The effective address of a program memory is calculated as the summation of the content of base register (program counter PC or data pointer DPTR) and an offset, i.e., the contents of accumulator. This addressing mode is intended for a JMP or MOVC instruction.

Example MOVC A, @A+DPTR

When this instruction is executed, move a byte of data from program memory, whose address can be obtained by adding the accumulator to the data pointer, to the accumulator as depicted in Fig. 14.3.

A list of examples of indexed addressing are given in Table 14.5.

**Fig. 14.3** Indexed addressing**Table 14.5** Examples of indexed addressing

<i>Instruction</i>	<i>Task</i>
MOVC A,@A+PC	Move a byte of data from program memory, whose address can be determined by sum of accumulator and program counter, to the accumulator.
MOVC A,@A+DPTR	Move a byte of data from program memory, whose address can be found by adding the accumulator and the data pointer, to the accumulator.
JMP @A+DPTR	Jump indirect relative to the data pointer; the address of a jump instruction is calculated as sum of the accumulator and the data pointer.

14.2.6 Relative Addressing

Generally, this addressing mode is used in certain jump instructions. The relative address is an 8-bit signed number (–128 to 127), which is added to the program counter to determine the address of the next instruction. Before addition, the program counter is incremented. Therefore, the new address relative to the next

instruction is determined before a jump to the new address instruction. For example, when the SJMP Level offset instruction is executed, the new address can be obtained from the sum of the PC and offset. Then, a jump to the new address occurs as depicted in Fig. 14.4. The advantage of relative addressing is that it has position-independent codes.

14.2.7 Absolute Addressing

The absolute addressing is only used with AJMP and ACALL instructions. The 11 least significant bits of the destination address come from the opcode and the upper five bits are the current upper five bits in the program counter. In this addressing mode, the destination address will be within 2K (2¹¹) memory. For example, ACALL addr11.

14.2.8 Long Addressing

The long addressing is used only with the LJMP and LCALL instructions. These instructions include a full 16-bit destination address. In this mode, the full 64K code space is available and the instruction is long and position dependent. For example, when LJMP, 9500H. instruction is executed, it jumps to memory location 9500H.

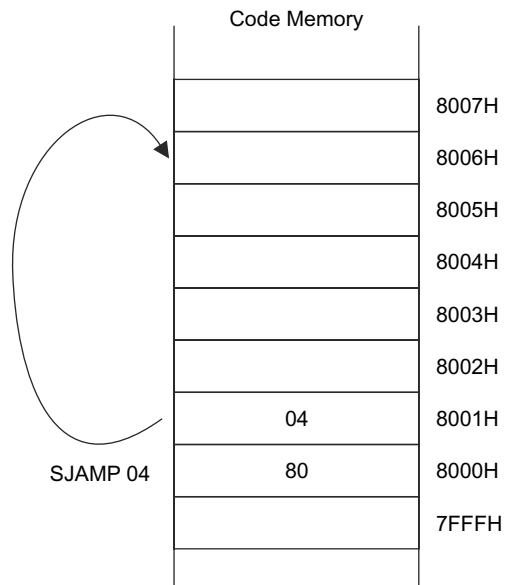


Fig. 14.4 Relative addressing

Example 14.1

Find the addressing modes of the following instructions:

- (i) ADD A, R7 (ii) ADD A, 55H (iii) MOV A, @R0 (iv) MUL AB
- (v) MOV A, #FF (vi) MOV DPTR, #8000 (vii) MOVC A, @A+DPTR

Solution

- (i) ADD A, R7 instruction is an example of register addressing.
- (ii) ADD A, 55H instruction is an example of direct addressing.
- (iii) MOV A, @R0 instruction is an example of indirect addressing.
- (iv) MUL AB instruction is an example of register addressing.
- (v) MOV A, #FF instruction is an example of immediate addressing.
- (vi) MOV DPTR, #8000 instruction is an example of immediate addressing.
- (vii) MOVC A, @A+DPTR instruction is an example of index addressing.

14.3 8051 INSTRUCTION SET

An instruction is a command applied to the microcontroller to perform a specified operation. There are 255 possible instructions in the 8051 microcontroller. Each instruction consists of an operation code (opcode) and an operand. Opcode states the specified operation which will be performed. The operand means data, which will be used in that operation, as no operation can be performed without data. The 8051 instructions have 8-bit opcodes. Data field may be of one-byte or two-bytes. Based on data (operand), the instructions are classified as one-byte, two-byte and three-byte instructions. 8051 instructions are divided into the following groups as given below:

- ◆ Arithmetic instructions
- ◆ Logical instructions
- ◆ Data-transfer instructions
- ◆ Boolean operations instructions
- ◆ Program-control instructions
- ◆ Branching instructions

The symbols and abbreviations, which have been used while explaining Intel 8051 microcontroller instructions, are illustrated in Table 14.6. In this section, all groups of instructions are explained elaborately with appropriate representations.

Table 14.6 Symbol/Abbreviations of instruction set

<i>Symbol/Abbreviations</i>	<i>Meaning</i>
addr 16	16-bit address
addr 11	11-bit address
#data	8-bit constant included in the instruction.
#data 16	16-bit constant included in the instruction
R _n , R _i	Register R ₇ –R ₀ of the currently selected register bank.
direct	8-bit internal data location's address. This could be an Internal Data RAM location (0–127) or a SFR [i.e., I/O port, control register, status register, etc. (128–255)].
@R _i	8-bit internal data RAM location (0–255) addressed indirectly through register R ₁ or R ₀ .
rel	Signed (two's complement) 8-bit offset byte. Used by SJMP and all conditional jumps. Range is –128 to +127 bytes relative to first byte of the instruction.
bit	Direct addressed bit in Internal Data RAM or Special Function Register.
DPTR	Data pointer register.
DPH, DPL	DPH—Data pointer register higher, DPL—Data pointer register lower.
SP	SP represents 16-bit stack pointer.
PC	16-bit program counter
PSW	Program Status Word
CS	Carry status
[]	Content of the memory location
←	Move data in the direction of the arrow
↔	Exchange contents
∧	Logical AND operation
∨	Logical OR operation
⊕	Logical EXCLUSIVE OR
	Complement

14.3.1 Arithmetic Instructions

The arithmetic instructions are used to perform arithmetic operations such as addition, subtraction, increment, decrement, multiplication, and division. Since different addressing modes are available, an arithmetic instruction can be written in different ways. All arithmetic instructions are executed in one machine cycle except INC, DPTR, MUL AB and DIV AB. INC and DPTR requires two machine cycles and MUL AB and DIV AB require four machine cycles. All arithmetic instructions are explained below:

ADD A, R_n (Add register to accumulator)

$$A \leftarrow A + R_n,$$

Machine cycles: 1; States: 12; Flags: all; Register Addressing; One-byte Instruction

The contents of the operand (register) are added to the contents of the accumulator and the result is stored in the accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from Bit 7 or Bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

✓ **Example** ADD A, R6 Add the content of register R6 to accumulator and result in accumulator.

ADD A, direct (Add direct byte to Accumulator)

$$A \leftarrow A + [\text{direct}],$$

Machine Cycles: 1; States: 12; Flags: all; Direct Addressing; Two-byte Instruction

The contents of the internal memory location specified by the 8-bit direct are added with accumulator. All flags are modified to reflect the result of the addition. For example, the instruction is ADD A, 44H.

ADD A, @R_i (Add indirect RAM to Accumulator)

$$A \leftarrow A + [R_i],$$

Machine Cycles: 1; States: 12; Flags: all; Register Indirect Addressing; One-byte Instruction

The contents of the internal RAM whose location is denoted by the content of register R_i (R0 or R1) are added to the contents of the accumulator and the result is stored in the accumulator.

✓ **Example** ADD A, @R0

ADD A, #data (Add immediate data to accumulator)

$$A \leftarrow A + \#data,$$

Machine Cycles: 1; States: 12; Flags: all; Immediate Addressing; Two-byte Instruction

Add the number specified by #data to accumulator and the result is stored in the accumulator.

✓ **Example** ADD A, 36H

ADDC A, R_n (Add register to accumulator with carry)

$$A \leftarrow A + R_n + C,$$

Machine Cycles: 1; States: 12. Flags: all; Register Addressing; One-byte Instruction

ADDC instruction simultaneously adds the contents of the register R_n and the carry flag to the contents of the accumulator and the result is stored in the accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from Bit 7 or Bit 3, and cleared otherwise. During adding unsigned integers, the carry flag indicates an overflow occurred. When adding signed integers, OV indicates a negative number

produced as the sum of two positive operands, or a positive sum from two negative operands. Therefore, flags are modified to reflect the result of the addition.

✓ **Example** ADDC A, R7. If accumulator content is C3H, R7 content is AAH with the carry flag set, the result in accumulator is 6E and AC cleared, the carry flag and OV set to 1.

ADDC A, direct (Add direct byte to accumulator with carry)

$$A \leftarrow A + [\text{direct}] + C,$$

Machine Cycles: 1; States: 12; Flags: all; Direct Addressing; Two-byte Instruction

The content of the memory location, which is specified by the direct address and the carry flag, are added to the contents of the accumulator. After addition, the result is stored in the accumulator. All flags are effected to reflect the result of the addition.

✓ **Example** ADDC A, 55H

ADDC A, @Ri (Add indirect RAM to Accumulator with carry)

$$A \leftarrow A + [Ri] + C,$$

Machine Cycles: 1; States: 12; Flags: all. Register Indirect Addressing; One-byte Instruction

The contents of the internal memory RAM located by R_i register (R0 or R1) are added to the contents of the accumulator with carry and the result is stored in the accumulator.

✓ **Example** ADDC A, @R1

ADDC A, #data (Add immediate data to ACC with carry)

$$A \leftarrow A + C + \# \text{direct},$$

Machine Cycles: 1; States: 12; Flags: all. Immediate Addressing; Two-byte Instruction

The 8-bit immediate data (operand) can be added to the contents of the accumulator and the result is stored in the accumulator.

✓ **Example** ADDC A, #FF

SUBB A, R_n (Subtract register from accumulator with borrow)

$$A \leftarrow A - R_n - C,$$

Machine Cycles: 1; States: 12; Flags: all; Register Addressing; One-byte Instruction

SUBB A, R_n states that the content of register R_n and the carry flag are subtracted from the content of the accumulator. After subtraction, the result is stored in the accumulator. This instruction sets the carry (borrow) flag if a borrow is needed for Bit 7, and clears C otherwise. AC is set if a borrow is needed for Bit 3, and cleared otherwise. OV is set if a borrow is needed into Bit 6, but not into Bit 7, or into Bit 7, but not Bit 6. During subtraction of signed integers, OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number. For example, SUBB A, R2. If the accumulator content is C9H, the content of R2 is 54H and the carry flag is set, the result 74H will be in the accumulator with the carry flag and AC cleared but OV set.

SUBB A, direct (Subtract direct byte from accumulator with borrow)

$$A \leftarrow A - [\text{direct}] - C$$

Machine Cycles: 1; States: 12; Flags all; Direct Addressing; two-byte Instructions

The contents of the 8-bit direct memory location are subtracted from the contents of the accumulator with borrow and the result is placed in the accumulator. All flags will be modified to reflect the result.

✓ **Example** SBBB A, 45H.

SUBB A, @Ri (Subtract indirect RAM from accumulator with borrow)

$$A \leftarrow A - [Ri] - C,$$

Machine Cycles: 1; States: 12; Flags: all; Register Indirect Addressing; One-byte instruction

The content of internal RAM whose location is specified by register *Ri* (R0 or R1) is subtracted from the content of the accumulator with borrow, and the result is stored in the accumulator.

✓ **Example** SUBB A, @R0

SUBB A, #data (Subtract immediate data from accumulator with borrow)

$$A \leftarrow A - \#data - C$$

Machine Cycles: 1; States: 12; Flags: all; Immediate Addressing; Two-byte Instructions

The 8-bit immediate data is subtracted from the contents of the accumulator with borrow and the result is placed in the accumulator.

✓ **Example** SBBB A, 78H

INC A (Increment accumulator by 1)

$$A \leftarrow A + 1.$$

Machine Cycles: 1; States: 12; No Flags are affected; Register Addressing; One-byte Instructions

When INC instruction is executed, the indicated variable is incremented by 1. Therefore, the contents of the accumulator are incremented by 1 and the result is stored in the accumulator.

✓ **Example** INC A

INC Rn (Increment register by 1)

$$Rn \leftarrow Rn + 1.$$

Machine Cycles: 1; States: 12; No flags are affected; Register Addressing; One-byte Instructions

The contents of the selected register *Rn* (R0 to R7) are incremented by 1 and the result is stored in the same register.

✓ **Example** INC R5

INC direct (Increment direct byte)

$$[\text{direct}] \leftarrow [\text{direct}] + 1.$$

Machine Cycles: 1; States: 12; No flags are affected; Direct Addressing; Two-byte Instructions

The contents of the 8-bit direct memory location are incremented by 1 and the result is stored in the same memory location.

✓ **Example** INC 44H

INC @Ri (Increment indirect RAM)

$$[Ri] \leftarrow [Ri] + 1.$$

Machine Cycles: 1; States: 12; No flags are affected; Register Indirect Addressing; One-byte instructions

The contents of the internal RAM location whose address can be selected by register R0 or R1 are incremented by 1 and the result is stored in the same RAM location.

✓ **Example** INC @R0

INC DPTR (Increment data pointer register by 1)

$DPTR \leftarrow DPTR + 1.$

Machine Cycles: 2; States: 24; No flags are affected; Register Addressing; One-byte instruction

The contents of the 16-bit data pointer register are incremented by 1 and the result is stored in the same register. A 16-bit increment is performed; an overflow of the low-order byte of the data pointer (DPL) from FFH to 00H will increment the high-order byte (DPH). No flags are affected.

✓ **Example** INC DPTR

DEC A (Decrement accumulator by 1)

$A \leftarrow A - 1.$

Machine Cycles: 1; States: 12; Flags: No flags are affected; Register Addressing; One-byte Instruction

The contents of the accumulator are decremented by 1 and the result is stored in the accumulator.

✓ **Example:** DEC A.

DEC Rn (Decrement register by 1)

$R_n \leftarrow R_n - 1.$

Machine Cycles: 1; States: 12; Flags: No flags are affected; Register Addressing; One-byte Instruction

The contents of the selected register R0 to R7 are decremented by 1 and the result is stored in the same register.

✓ **Example** DEC R6.

DEC direct (Decrement direct byte)

$[\text{direct}] \leftarrow [\text{direct}] - 1.$

Machine Cycles: 1; States: 12; Flags: No flags are affected; Direct Addressing; Two-byte Instruction

The contents of the 8-bit direct memory location is decremented by 1 and the result is stored in the same memory location.

✓ **Example** DEC 34H.

DEC @Ri (Decrement indirect RAM)

$[[R_i]] \leftarrow [[R_i]] - 1.$

Machine Cycles: 1; States: 12; Flags: No flags are affected; Register Indirect Addressing; One-byte Instruction.

The contents of the internal RAM location specified by registers R0 or R1 are decremented by 1 and the result is stored in the same place.

✓ **Example** DEC @Ri.

MUL AB (Multiply A by B)

$$A_{7-0} \leftarrow A \times B$$

$$B_{15-8}$$

Machine Cycles: 4; States: 48; Flags: Flags are affected; Register Addressing; One-byte Instruction.

MUL AB multiplies the unsigned eight-bit integers in the accumulator and register B. The low-order byte of the sixteen-bit product will be stored in the accumulator, and the high-order byte will be stored in B. If the product is greater than 255 (FFH) the overflow flag is set; otherwise it is cleared. The carry flag is always cleared.

✓ **Example** MUL AB.

DIV AB (Divide A by B)

$$A_{15-8} \leftarrow A/B$$

$$B_{7-0}$$

Machine Cycles: 4; States: 48; Flags: Flags are affected; Register Addressing; One-byte Instruction

DIV AB divides the unsigned eight-bit integer in the accumulator by the unsigned eight-bit integer in Register B. After execution of DIV AB, the Accumulator receives the integer part of the quotient and register B receives the integer remainder. The carry and OV flags will be cleared.

✓ **Example** DIV AB.

DAA (Decimal adjust accumulator for addition)

Machine Cycles: 1; States: 12; Flags: all; One-byte Instruction

If $[(A_{3-0}) > 9] \vee [(AC) = 1]$, then $(A_{3-0}) \leftarrow (A_{3-0}) + 6$

AND If $[(A_{7-4}) > 9] \vee [(C) = 1]$, then $(A_{7-4}) \leftarrow (A_{7-4}) + 6$

The contents of accumulator are transferred from a binary code to two 4-bit binary coded decimal (BCD) digits. This is the only instruction which uses the auxiliary flag to perform the binary to BCD conversion. The conversion procedure is as follows:

When the value of the low-order 4-bits/nibble in the accumulator is greater than 9 or AC flag is set, the instruction adds 6 to the low-order four bits.

If the value of the high-order 4-bits/nibble in the accumulator is greater than 9 or the carry flag is set, the instruction adds 6 to the high-order four bits. In this instruction S, Z, AC, P, CY flags are altered to reflect the results of the operation.

✓ **Example** ADC A, R3

DAA

If the accumulator holds 56H, i.e., the packed BCD digits of decimal number 56 and the content of register R3 is 67H, i.e., the packed BCD digits of decimal number 67. The carry flag is set. After execution of the above instructions, 24H will be stored in the accumulator as the true sum of 56, 67 and 1 is 124.

Example 14.2

Write instructions for the following operations:

- (i) Add 23H to the contents of accumulator.
- (ii) Add the content of the memory location specified by R0 with accumulator.
- (iii) Subtract the content of R1 register from accumulator with borrow.
- (iv) Subtract immediately 45H from accumulator register with borrow.
- (v) Increment the content of internal memory location specified by R0.

Solution

- (i) ADD A, #23; Add 23H to the contents of accumulator.
- (ii) ADD A, @R0; Add the content of the memory location specified by R0 with accumulator.
- (iii) SUBB A, R1; Subtract the content of R1 register from accumulator with borrow.
- (iv) SUBB A, #45; Subtract immediately 45 from accumulator register with borrow.
- (v) INC @R0; Increment the content of internal memory location specified by R0.

Example 14.3

Write instructions for the following operations:

- (i) Multiply the content of accumulator by B register.
- (ii) Divide the content of accumulator by B register.
- (iii) Increment data pointer register by one.

Solution

- (i) MUL A B; Multiply the content of accumulator by B register.
- (ii) DIV A B; Divide the content of accumulator by B register.
- (iii) INC DPTR; Increment data pointer register by one.

14.3.2 Logical Instructions

The logical instructions perform AND, OR, EX-OR, operations; compare, rotate or complement of data in register or memory. All logical instructions are discussed in this section.

ANL performs the bitwise logical AND operation between the variables indicated in the instruction and stores the results in the destination variable. No flags are affected. The two operands allow six addressing mode combinations. If the destination is the accumulator, the source can use register, direct, register-indirect, or immediate addressing. When the destination is a direct address, the source can be the accumulator or immediate data.

ANL A, R_n (Logical AND register to accumulator)

$$A \leftarrow A \wedge R_n$$

Machine Cycles: 1; States: 12; Flags: No flags are affected; Register Addressing; One-byte Instruction

The contents of the accumulator are logically ANDed with the contents of the register R_n (R0–R7). The result is stored in the accumulator. No flags are affected.

✓ **Example** ANL A, R5

ANL A, direct (Logical AND direct byte to Accumulator)

$$A \leftarrow A \wedge [\text{direct}].$$

Machine Cycles: 1; States: 12; Flags: No flags are affected; Direct Addressing; Two-byte Instructions

The content of the 8-bit memory location whose address is specified by the direct address is ANDed with the accumulator. The result is placed in the accumulator. No flags are affected.

✓ **Example** ANL A, direct

ANL A, @R_i (Logical AND direct byte to Accumulator)

$$A \leftarrow A \wedge [R_i]$$

Machine Cycles: 1; States: 12; Flags: No flags are affected; Register Indirect Addressing; One-byte Instruction

The content of the memory location whose address is specified by the register R0 or R1 is ANDed with the accumulator. After ANDing, the result is stored in the accumulator. No flags are affected.

✓ **Example** ANL A, @R₁.

ANL A, #data (Logical AND immediate data to accumulator)

$A \leftarrow A \wedge \#data.$

Machine Cycles: 1; States: 12; Flags: No flags are affected; Immediate Addressing; Two-byte Instruction

The contents of the accumulator are logically ANDed with the 8-bit data (#data). After ANDing, the result is stored in the accumulator. No flags are affected.

✓ **Example** ANL A, #45H.

ANL direct, A (Logical AND immediate data to direct byte)

$A \leftarrow [direct] \wedge A.$

Machine Cycles: 1; States: 12; Flags: No flags are affected; One-byte Instruction

The content of the memory location whose address is specified by the 8-bit direct address is ANDed with the accumulator. The result will be stored in the 8-bit direct memory address. No flags are affected.

✓ **Example** ANL direct, A.

ANL direct, #data (Logical AND memory with accumulator)

$[direct] \leftarrow [direct] \wedge \#data.$

Machine Cycles: 1; States: 12; Flags: No flags are affected; Immediate Addressing; One-byte Instruction

The content of the memory location whose address is specified by the 8-bit direct address is ANDed with the 8-bit immediate data. The result will be stored in the 8-bit direct memory address. No flags are affected.

✓ **Example** ANL direct, #A.

ORL performs the bitwise logical OR operation between the indicated variables in instruction and store the results in the destination byte. No flags are affected. Six different addressing-mode combinations are available for this instruction. When the destination is the accumulator, the source can use register, direct, register-indirect, or immediate addressing. While the destination is a direct address, the source can be the accumulator or immediate data.

ORL A, R_n (Logical OR Register to Accumulator)

$A \leftarrow A \vee R_n$

Machine Cycles: 1; States: 12; Flags: No flags are affected; Register Addressing; One-byte Instruction

The content of register R_n (R0–R7) is logically ORed with the content of the accumulator. The result is stored in the accumulator. No flags are affected.

✓ **Example** ORL A, R_n

ORL A, direct (Logical OR direct byte to accumulator)

$A \leftarrow A \vee [direct].$

Machine Cycles: 1; States: 12; Flags: No flags are affected; Direct Addressing; Two-byte Instruction

The contents of the accumulator are logically ORed with the contents of the memory location, whose address is specified by the 8-bit direct address and the result is placed in the accumulator. No flags are affected.

✓ **Example** ORA A, direct.

ORL A, @R_i (Logical OR indirect RAM to Accumulator)

$$A \leftarrow A \vee [R_i]$$

Machine Cycles: 1; States: 12; Flags: No flags are affected; Register Indirect Addressing; One-byte Instruction

The contents of the accumulator are logically ORed with the contents of the memory location, whose address is specified by the content of register R0 or R1. The result is placed in the accumulator. No flags are affected. ORA

✓ **Example** ORAA, @R0.

ORL A, #data (Logical OR immediate 8-bit data with accumulator)

$$A \leftarrow A \vee \#8\text{-bit data.}$$

Machine Cycles: 1; States: 12; Flags: No flags are affected; Immediate Addressing; Two-byte Instruction

In this instruction, 8-bit data is ORed with the content of the accumulator and the result is placed in the accumulator. No flags are affected.

✓ **Example** ORA A, #45H.

ORL direct, A (Logical OR Accumulator to direct byte)

$$[\text{direct}] \leftarrow [\text{direct}] \vee A.$$

Machine Cycles: 1; States: 12; Flags: No flags are affected; Two-byte Instruction

The contents of the accumulator are logically ORed with the contents of the memory location, whose address is specified by the 8-bit direct address and the result is stored in the 8-bit direct address. No flags are affected.

✓ **Example** ORA direct, A

ORL direct, #data (Logical OR immediate data to direct byte)

$$[\text{direct}] \leftarrow [\text{direct}] \vee \#data.$$

Machine Cycles: 2; States: 24; Flags: No flags are affected; Immediate Addressing; Three-byte Instruction

The 8-bit immediate data is logically ORed with the contents of the memory location, whose address is specified by the 8-bit direct address and the result is placed in the 8 bit direct address. No flags are affected.

✓ **Example** ORA direct, #data

XRL performs the bitwise logical Exclusive-OR operation between the indicated variables in instruction and stores the results in the destination. No flags are affected. Different addressing-mode combinations are possible for this instruction. When the destination is the accumulator, the source can use register, direct, register-indirect, or immediate addressing. If the destination is a direct address, the source can be the accumulator or immediate data.

XRL A, R_n (Exclusive-OR register with accumulator)

$$A \leftarrow A \oplus R_n$$

Machine Cycles: 1; States: 12; Flags: No flags are affected; Register Addressing; One-byte Instruction

The contents of the accumulator are Exclusive ORed with the contents of the register R_n (R0–R7) and the result is placed in the accumulator. No flags are affected.

✓ **Example** XRL A, R7.

XRL A, direct (Exclusive-OR direct byte to Accumulator)

$A \leftarrow A \oplus [\text{direct}]$

Machine Cycles: 1; States: 12; Flags: No flags are affected; Direct Addressing; Two-byte Instruction

The contents of the accumulator are Exclusive ORed with the contents of the memory location, which is specified by 8-bit direct address and the result is placed in the accumulator. No flags are affected.

✓ **Example** XRL A, direct.

XRL A, @R_i (Exclusive-OR indirect RAM to Accumulator)

$A \leftarrow A \oplus [R_i]$

Machine Cycles: 1; States: 12; Flags: No flags are affected; Register Indirect Addressing; One-byte Instruction

The contents of the accumulator are Exclusive ORed with the contents of the memory location, which is specified by the register R_i (R0 or R1) and the result is placed in the accumulator. No flags are affected.

✓ **Example** XRL A, @R0.

XRL A, #data (Exclusive-OR immediate data to Accumulator)

$A \leftarrow A \oplus \#data$

Machine Cycles: 1; States: 12; Flags: No flags are affected; Immediate Addressing; Two-byte Instruction

The contents of the accumulator are Exclusive ORed with the 8-bit data. The result is stored in the accumulator. No flags are affected.

✓ **Example** XRL A, #78H.

XRL direct, A (Exclusive-OR Accumulator to direct byte)

$[\text{direct}] \leftarrow [\text{direct}] \oplus A$

Machine Cycles: 1; States: 12; Flags: No flags are affected; One-byte Instruction

The contents of the accumulator are Exclusive ORed with the contents of the memory location, which is specified by the 8-bit direct address and the result is placed in the same address. No flags are affected.

✓ **Example** XRL direct, A.

XRL direct, #data (Exclusive-OR immediate data to direct byte)

$[\text{direct}] \leftarrow [\text{direct}] \oplus \#data$

Machine Cycles: 2; States: 24; Flags: No flags are affected; Three-byte Instruction

The contents of the 8-bit direct address memory location are Exclusive ORed with 8-bit immediate data and the result is placed in the 8-bit direct memory address. No flags are affected.

✓ **Example** XRL direct, #data.

CLR A (Clear Accumulator)

$A \leftarrow 0$; Machine Cycles: 1; States: 12

The accumulator is cleared (all bits reset to zero). No flags are affected.

CPL A (Complement Accumulator)

$A \leftarrow \overline{A}$; Machine Cycles: 1; States: 12,

Each bit of the accumulator is logically complemented, i.e., one's complement. Bits which previously contained a one are changed to a zero and vice-versa. No flags are affected.

RL A (Rotate accumulator left)

$A_{n+1} \leftarrow A_n, A_0 \leftarrow A_7$,

Machine Cycles: 1; States: 12; Flags: No Flags are affected; Implicit Addressing; One-byte Instruction

The eight bits in the accumulator are rotated one bit to the left. Bit 7 is rotated into the Bit 0 position as shown in Fig. 14.5. No flags are affected.

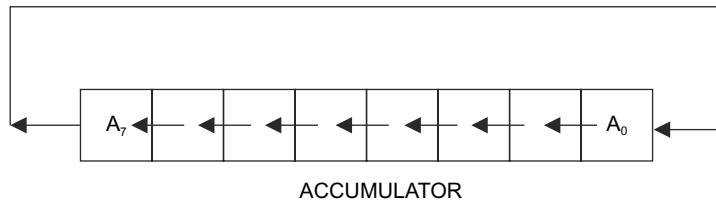


Fig. 14.5 Diagram for RLA

✓ Example RLA

The accumulator holds the value C5H (11000101) and after execution of RLA instruction, the accumulator holds the value 8BH (10001011) with the carry unaffected.

RLC A (Rotate accumulator left through the carry)

$A_{n+1} \leftarrow A_n, A_0 \leftarrow C, C \leftarrow A_7$

Machine Cycles: 1; States: 12; Flags: CS; Implicit Addressing; One-byte Instruction

The eight bits in the accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag and the original state of the carry flag moves into the Bit 0 position. Each bit of the accumulator is rotated left by one bit. The seventh bit of the accumulator is placed in the position of carry and carry flag moves to A0 as shown in Fig. 14.6. No other flags are affected.

✓ Example RLC A

Assume the accumulator holds the value C5H (11000101), and the carry is zero. After execution of RLC A, the accumulator holds the value 8AH (10001010) with the carry set.

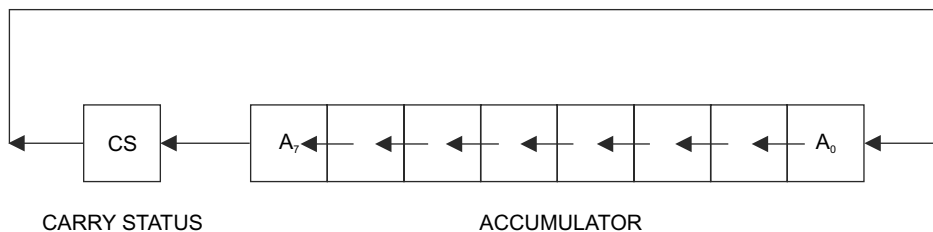


Fig. 14.6 Diagram for RLC A

RR A (Rotate accumulator right)

$$A_n \leftarrow A_{n+1}, A_7 \leftarrow A_0,$$

Machine Cycles: 1; States: 4 Flags: No flags are effected; Implicit Addressing; One-byte Instruction

The eight bits in the accumulator are rotated one bit to the right. Bit 0 is rotated into the Bit 7 position. Each binary bit of the accumulator is shifted right by one position as depicted in Fig. 14.7. No flags are affected.

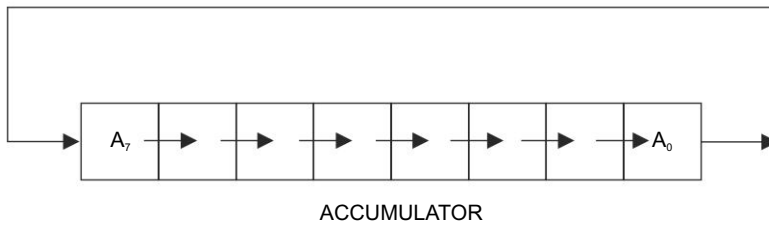


Fig. 14.7 Diagram for RR A

✓ **Example RR A**

If the accumulator holds the value C5H (11000101). After execution RR A instruction, the accumulator holds the value E2H (11100010) with the carry unaffected.

RRC A (Rotate accumulator right through the carry)

$$A_n \leftarrow A_{n+1}, A_7 \leftarrow C, C \leftarrow A_0$$

Machine Cycles: 1; States: 4; Flags: CS; Implicit Addressing; One-byte Instruction

The eight bits in the accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original state of the carry flag moves into the Bit 7 position as shown in Fig. 14.8. No other flags are affected.

✓ **Example RRC A**

When the accumulator holds the value C5H (11000101) and the carry is zero. After execution of RRC A instruction, the accumulator holds the value 62H (01100010) with the carry set.

SWAP (Swap nibbles within the Accumulator)

$$(A_{3-0}) \leftrightarrow (A_{7-4})$$

Machine Cycles: 1; States 4. Flags: No flags are affected

SWAP A instruction interchanges the low-order and high-order nibbles of the accumulator (bits 3–0 and bits 7–4). The operation can also be thought of as a four-bit rotate instruction. No flags are affected.

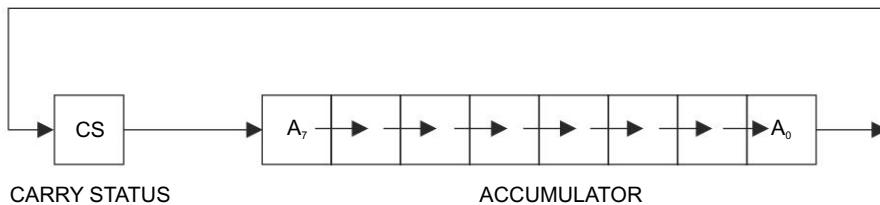


Fig. 14.8 Diagram for RRC A

✓ **Example** SWAP A.

If the accumulator holds the value C5H (11000101), after execution of SWAP A instruction, the accumulator holds the value 5CH (01011100).

14.3.3 Data-Transfer Instructions

Data-transfer instructions are used to transfer data between registers, register pairs, memory and registers, etc. The byte variable indicated by the second operand is copied into the location specified by the first operand. After execution of MOV <destination-byte>, <source-byte>, the source byte is not affected. No other register or flag is affected. This is by far the most flexible operation. Fifteen combinations of source and destination addressing modes are allowed. In this section, all data-transfer instructions are described in detail.

MOV A, Rn (Move register to Accumulator)

$A \leftarrow R_n$,

Machine Cycles: 1; States: 12; Flags none; Register Addressing Mode; One-byte Instruction

This instruction copies the contents of the source register into the accumulator but the contents of the source register are not changed. Flags and other registers are not affected.

✓ **Example** MOV A, R2.

MOV A, direct (Move direct byte to Accumulator)

$A \leftarrow [\text{direct}]$,

Machine Cycles: 1; States: 12; Flag none; Direct Addressing; Two-byte Instruction

The content of the memory location moves to the accumulator.

✓ **Example** The instruction MOV A, 44H will move the content 44H memory location to the accumulator.

MOV A, @Ri (Move indirect RAM to Accumulator)

$A \leftarrow [R_i]$,

Machine Cycles: 1; States: 12; Flag none; Register Indirect Addressing; One-byte Instruction

The content of memory location whose address is specified by register R0 or R1 moves to accumulator.

✓ **Example** The instruction MOV A, @R0 will move the content of the memory location specified by R0 register to accumulator.

MOV A, #data (Move immediate data to Accumulator)

$A \leftarrow \#data$,

Machine Cycles: 1; States: 12; Flags: none; Immediate Addressing Modes; Two-byte Instruction

The 8-bit data can be stored in the accumulator immediately.

✓ **Example** The instruction MOV A, #44H moves 44H to accumulator.

MOV Rn, A (Move Accumulator to register)

$R_n \leftarrow A$,

Machine Cycles: 1; States: 12; Flags: none; Register Addressing; One-byte Instruction

The contents of accumulator will be stored in the register R_n (R0–R7).

✓ **Example** MOV R_n, A.

MOV Rn, direct (Move direct byte to register)

$R_n \leftarrow [\text{direct}],$

Machine Cycles: 2; States: 12; Flags: none; Direct Addressing; Two-byte Instruction

The content of 8-bit direct memory location will be stored in register R_n .

✓ **Example** MOV R2, 22H. When this instruction is executed, the content of 22H memory location moves to Register R2.

MOV Rn, #data (Move immediate data to register)

$R_n \leftarrow \text{data},$

Machine Cycles: 1; States: 12; Flags: none; Immediate Addressing; Two-byte Instruction

The 8-bit immediate data will be stored in the register R_n .

Example: MOV R4, #67H. When this instruction is executed, 67H data move to the register R4.

MOV direct, A (Move Accumulator to direct byte)

$[\text{direct}] \leftarrow A,$

Machine Cycles: 1; States: 12; Flags: none; Two-byte Instruction

The content of the accumulator will be copied in the 8-bit direct address memory location.

✓ **Example** MOV 25, A. After execution of this instruction, accumulator content moves to 45H memory location.

MOV direct, Rn (Move register to direct byte)

$[\text{direct}] \leftarrow R_n,$

Machine Cycles: 2; States: 12; Flags: none; Two-byte Instruction

The content of Register R_n (R_0 – R_7) will be stored in the memory location, specified by the 8-bit direct address. Example: MOV 45H, R7. If this instruction is executed, R7 register content moves to 45H memory location.

MOV direct, direct (Move direct byte to direct)

$[\text{direct}] \leftarrow [\text{direct}],$

Machine Cycles: 2; States: 24; Flags: none; Three-byte Instruction

The data will be stored in an 8-bit direct memory location from an 8-bit direct memory location.

✓ **Example** MOV 23, 22H. The content of 22H memory location is copied to 23H memory location.

MOV direct, @Ri (Move indirect RAM to direct byte)

$[\text{direct}] \leftarrow [R_i],$

Machine Cycles: 3; States: 10; Flags none; Indirect Addressing; Two-byte Instruction

The content of internal RAM whose address is specified by the contents of the register R_i (R_0 or R_1) will be stored in 8-bit direct address memory location.

✓ **Example** MOV 44H, @R0. After execution of this instruction, the content of the memory location specified by R0 register will be stored in the 8-bit direct address memory location.

MOV direct, #data (Move immediate data to direct byte)

[direct] ← data,

Machine Cycles: 2; States: 24; Flags: none; Immediate Addressing; Two-byte Instruction

The 8-bit immediate data will be stored in the 8-bit direct memory location.

✓ **Example** MOV 45H, #22H In this instruction, MOV 45H, #22H, 22H data move to 8-bit direct address memory location 45 H.

MOV @Ri, A (Move Accumulator to indirect RAM)

[Ri] ← A,

Machine Cycles: 1; States: 12; Flags: none; One-byte Instruction

The content of accumulator will be stored in the memory location, specified by the contents of the Ri (R0 or R1) register.

✓ **Example** MOV @R0, 35H. In this instruction MOV @R0, 35H, 35H data moves to the memory location specified by R0 register.

MOV @Ri, direct (Move direct byte to indirect RAM)

[Ri] ← [direct],

Machine Cycles: 2; States: 24; Flags: none; Direct Addressing;

The data of 8-bit direct memory location will be stored in the memory location which is specified by the contents of the Ri (R1 or R0) register.

✓ **Example** MOV @R0, 33H. When this instruction is executed, the content of 33H memory location moves to memory location specified by R0.

MOV @ Ri, #data (Move immediate data to indirect RAM)

[Ri] ← #data,

Machine Cycles: 1; States: 12; Flags none; Immediate Addressing; Two-byte Instruction

The 8-bit immediate data will be stored in memory location, which is specified by the contents of the Ri register.

✓ **Example** MOV @R1, #FFH. After execution of MOV @R1, #FFH instruction, FFH data move to memory location specified by the R1 register.

MOV DPTR, #data 16 (Load data pointer with a 16-bit constant)DPTR ← data₁₅₋₀DPH ← data₁₅₋₈, DPL ← data₇₋₀

Machine Cycles: 3; States: 10; Flags: none; Immediate Addressing; Two-byte Instruction

The data pointer register is loaded with the 16-bit constant indicated in the instruction. The 16-bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte and the third byte (DPL) holds the low-order byte. No flags are affected. This is the only instruction, which moves 16 bits of data at once.

✓ **Example** MOV DPTR, #8000H.

When this instruction is executed, load the value 8000H into the Data Pointer. Hence DPH will hold 80H and DPL will hold 00H.

The $\text{MOVC A, @A + \langle\text{base-register}\rangle}$ instructions load the accumulator with a code byte from program memory. The address of the byte fetched is the sum of the unsigned eight-bit accumulator contents and the contents of a 16-bit base register. The base register may be either the data pointer or the PC. The PC is incremented before being added with the accumulator. As 16-bit addition is performed, a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected. Examples of MOV C instructions are explained below:

MOVC A, @A+DPTR (Move Code byte relative to DPTR to ACC)

$A \leftarrow [A+DPTR]$,

Machine Cycles: 2; States: 12; Flags: none; Index Addressing; One-byte Instruction

The contents of memory location specified by the contents of accumulator and the DPTR register, move to accumulator.

✓ **Example** MOVC A, @A+DPTR .

MOVC A, @A+PC (Move Code byte relative to PC to ACC)

$PC \leftarrow PC+1, A \leftarrow [A+PC]$,

Machine Cycles: 2; States: 24; Flags none; Index Addressing; One-byte Instruction

The contents of memory location specified by the contents of the accumulator and the PC register, move to the accumulator.

✓ **Example** MOVC A, @A+PC .

The $\text{MOVX } \langle\text{destination-byte}\rangle, \langle\text{source-byte}\rangle$ instruction is used to transfer data between the accumulator and a byte of external memory. There are two types of instructions differing in whether they provide eight bits or sixteen bits in direct address to the external data RAM.

In the first case, the contents of R0 or R1 of the selected register bank provide an eight-bit address multiplexed with data on P0. In the second case, the data pointer registers provide the sixteen-bit address, P2 outputs the content of DPH, i.e., high-order eight address bits, but P0 multiplexes the low-order eight bits (DPL) with data. All types of MOVX instruction are explained in this section.

MOVX A, @R_i (Move external RAM (8-bit addr) to ACC)

$A \leftarrow [R_i]$,

Machine Cycles: 2; States: 24; Flags: none; Indirect Addressing; One -byte Instruction

The contents of external RAM location (8-bit address), which is specified by the contents of R0 or R1, moves to accumulator.

✓ **Example** MOVX A, @R0 .

MOVX A, @DPTR (Move external RAM (16-bit addr) to ACC)

$A \leftarrow [DPTR]$,

Machine Cycles: 2; States: 24; Flags: none; Indirect Addressing, One-byte Instruction

The contents of external RAM location (16-bit address), which is specified by the contents of DPTR, moves to accumulator.

✓ **Example** MOVX A, @DPTR .

MOVX @R_i, A (Move ACC to external RAM (8-bit addr))

$[R_i] \leftarrow A$,

Machine Cycles: 2; States: 24; Flags: none; One -byte Instruction

The contents of accumulator move to external RAM location (8-bit address), which is specified by the contents of R0 or R1.

✓ **Example** MOVX @R0,A.

MOVX @DPTR, A (Move ACC to external RAM (16-bit addr))

[DPTR] ←A,

Machine Cycles: 2; States: 24; Flags: none; One -byte Instruction

The contents of accumulator move to external RAM location (16-bit address), which is specified by the contents of DPTR.

✓ **Example** MOVX @DPTR, A.

Example 14.4

Write instructions to perform the following operations:

- (i) Move the content of accumulator into R0 register.
- (ii) Load immediate 8-bit data (FFH) into accumulator.
- (iii) Load Data pointer with 9000H.
- (iv) Move the content of accumulator to external RAM location 8000H.

Solution

- (i) MOV A, R0; Move the content of accumulator into R0 register.
- (ii) MOV A, #FFH; Load immediate 8-bit data (FFH) into accumulator.
- (iii) MOV DPTR, #9000H; Load Data pointer with 9000H.
- (iv) MOV DPTR, #8000; Load Data Pointer with 8000H.
MOVX @DPTR, A; Move the content of accumulator to external RAM location 8000H.

14.3.4 Boolean Variable Manipulation

The 8051 controller contains a complete Boolean processor for single-bit operations. In these instructions, all bit accesses use direct addressing and bits may be set or cleared using a single instruction. All Boolean instructions are explained below:

CLR C (Clear carry)

C ←0,

Machine Cycles: 1; States: 12; Flags: none; Direct Addressing Mode; One-byte Instruction

This instruction clears the carry flag. No other flags are affected.

✓ **Example** CLR C.

CLR bit (Clear direct bit)

bit ←0,

Machine Cycles: 1; States: 12; Flags: none; Direct Addressing Mode; Two-byte Instruction

The indicated bit is cleared (reset to zero). No other flags are affected. CLR can operate on the carry flag or any directly addressable bit.

✓ **Example** Port 1 has previously been written with 5DH (01011101B). The instruction CLR P1.2 will leave the port set to 59H (01011001B).

SETB C (Clear carry)

$C \leftarrow 1$,

Machine Cycles: 1; States: 12; Flags: none; Direct Addressing Mode; One-byte Instruction

This instruction set the carry flag. No other flags are affected.

✓ **Example** SETB C.

SETB bit (Clear direct bit)

$\text{bit} \leftarrow 1$,

Machine Cycles: 1; States: 12; Flags: none; Direct Addressing Mode; Two-byte Instruction

SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly addressable bit.

No other flags are affected.

✓ **Example** The carry flag is cleared. Output Port 1 has been written with the value 34H (00110100B). The instructions, SETB P1.0 will leave the carry flag set to 1 and change the data output on Port 1 to 35H (00110101B).

CPL C (Complement carry)

$C \leftarrow \overline{C}$,

Machine Cycles: 1; States: 12; Flags: none; Direct Addressing Mode; One-byte instruction

This instruction complements the carry flag. No other flags are affected.

✓ **Example** CPL C.

CPL bit (Complement bit)

$\text{bit} \leftarrow \overline{\text{bit}}$,

Machine Cycles: 1; States: 12; Flags: none; Direct Addressing Mode; One-byte Instruction

The bit variable specified is complemented. When a bit is one, it is changed to zero and vice-versa. No other flags are affected. CPL can operate on the carry or any directly addressable bit.

✓ **Example** Port 1 has previously been written with 5DH (01011101B). The instruction sequence, CPL P1.1 and CPL P1.2 will leave the port set to 5BH (01011011B).

ANL C, bit (AND direct bit to carry)

$C \leftarrow C \wedge \text{bit}$

Machine Cycles: 2; States: 24; Flags: none; Direct Addressing Mode; Two-byte Instruction

This instruction performs logical AND operation between the source bit and the carry flag. No other flags are affected.

✓ **Example** ANL C, ACC.7; AND operation between the accumulator bit 7 and the carry.

ANL C, /bit (Complement bit)

$C \leftarrow C \wedge \overline{\text{bit}}$

Machine Cycles: 2; States: 24; Flags: none; Direct Addressing Mode; Two-byte Instruction

The slash (/) preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.

✓ **Example** ANL C, /OV; AND with inverse of overflow flag

ORL C, bit (OR direct bit to carry)

$$C \leftarrow C \vee \text{bit}$$

Machine Cycles: 2; States: 24; Flags: none; Direct Addressing Mode; Two-byte Instruction

This instruction performs logical-OR operation between source bit and the carry. No other flags are affected Example: ORL C, ACC.7; OR carry with the ACC. BIT 7.

ORL C, /bit (OR complement of direct bit to carry)

$$C \leftarrow C \vee \overline{\text{bit}}$$

Machine Cycles: 2; States: 24; Flags: none; Direct Addressing Mode; Two-byte Instruction

A slash (/) preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.

✓ **Example** ORL C, /OV; OR carry with the inverse of OV.

MOV C, bit (Move direct bit to carry)

$$C \leftarrow \text{bit}$$

Machine Cycles: 1; States: 12; Flags: none; Direct Addressing Mode; Two-byte Instruction

This instruction is used to copy the Boolean variable indicated by the second operand into the location specified by the first operand. No other flags are affected.

✓ **Example** MOV C, P3.3.

MOV bit, C (Move carry to direct bit)

$$\text{bit} \leftarrow C$$

Machine Cycles: 2; States: 24; Flags none; Direct Addressing Mode; Two-byte Instruction

The Boolean variable indicated by the second operand must be copied into the location specified by the first operand. One of the operands is the carry flag and the other is any directly addressable bit. No other register or flag is affected.

✓ **Example** MOV P1.3, C; Assume the carry flag is set and the data present at output Port 1 is 35H (0011 0101B). After execution of MOV P1.3, C; Port 1 changes to 3DH (0011 1101B).

JC rel (Jump if carry is set)

$$PC \leftarrow PC+2, \text{ If } C=1, \text{ Then } PC \leftarrow PC + \text{rel}$$

Machine Cycles: 2; States: 24; Flags: none; Two-byte Instruction

When the carry flag is set, jump to the address indicated in instruction; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement to the PC, after incrementing the PC twice. No flags are affected.

✓ **Example** JC LABEL-1

JNC rel (Jump if carry is not set)

$PC \leftarrow PC+2$, If $C=0$, Then $PC \leftarrow PC + rel$

Machine Cycles: 2; States: 24; Flags: none; Two-byte Instruction

If the carry flag is a zero, jump to the address indicated in the instruction; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified.

✓ **Example** JNC LABEL-1

JB bit, rel (Jump if direct bit is set)

$PC \leftarrow PC+3$, If $bit=1$, Then $PC \leftarrow PC + rel$

Machine Cycles: 2; States: 24; Flags: none, Three-byte Instruction

If the indicated bit is '1', jump to the address indicated in the instruction; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement to the PC, after incrementing the PC. The bit tested is not modified. No flags are affected.

✓ **Example** JB P1.2, LABEL-1

JNB bit, rel (Jump if direct bit is not set)

$PC \leftarrow PC+2$, If $C=1$, Then $PC \leftarrow PC + rel$

Machine Cycles: 2; States: 24; Flags: none; Three-byte Instruction

If the indicated bit is a zero, jump to the indicated address in the instruction; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement to the PC, after incrementing the PC. The bit tested is not modified. No flags are affected.

✓ **Example** JNB P1.3, LABEL-1

JB C bit, rel (Jump if direct bit is set and clear bit)

$PC \leftarrow PC+3$, If $bit=1$, Then $bit \leftarrow 0$, $PC \leftarrow PC + rel$

Machine Cycles: 2; States: 24; Flags: none; Three-byte Instruction

If the indicated bit is '1', jump to the address indicated in instruction; otherwise proceed with the next instruction. The bit will not be cleared if it is already a zero. The branch destination is computed by adding the signed relative-displacement to the PC, after incrementing the PC. No flags are affected.

✓ **Example** JBC ACC.3, LABEL-1

14.3.5 Branch Group

The branch-group instructions are generally used to change the sequence of the program execution. There are two types of branch instructions, namely, conditional and unconditional. The *conditional branch instructions* transfer the program to the specified address only when the condition is satisfied. The *unconditional branch instructions* transfer the program to the specified address unconditionally. All conditional and unconditional branch instructions are explained in this section.

ACALL 11-bit address (Absolute subroutine CALL)

$PC \leftarrow PC+2$, $SP \leftarrow SP+1$, $SP \leftarrow PC_{7-0}$,

$SP \leftarrow SP+1$, $SP \leftarrow PC_{15-8}$, $PC_{10-0} \leftarrow$ page address

Machine Cycles: 2; States: 24 ; Flags: none; Two-byte Instruction

ACALL instruction unconditionally calls a subroutine located at the indicated address. This instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the stack pointer twice. The destination address can be obtained by successively concatenating the five high-order bits of the incremented PC, opcode bits 7–5, and the second byte of the instruction. This instruction can be used to call a subroutine within the same 2K block of the program memory as the first byte of the instruction following ACALL. No flags are affected.

✓ **Example** ACALL address 11

LCALL 16-bit address (Long subroutine CALL)

$PC \leftarrow PC+3$, $SP \leftarrow SP+1$, $[SP] \leftarrow PC_{7-0}$,

$SP \leftarrow SP+1$, $[SP] \leftarrow PC_{15-8}$, $PC \leftarrow address_{15-0}$

Machine Cycles: 2; States: 24 ; Flags: none; Three-byte Instruction

LCALL instruction calls a subroutine located at the indicated address. This instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low byte first), incrementing the stack pointer by two. The high-order and low-order bytes of the PC are then loaded, respectively, with the second and third bytes of the LCALL instruction. Therefore, the subroutine may start anywhere in the full 64Kbyte program memory address space. No flags are affected.

✓ **Example** LCALL SUBRTN

RET (Return from subroutine)

$PC_{15-8} \leftarrow [SP]$, $SP \leftarrow SP-1$, $PC_{7-0} \leftarrow [SP]$, $SP \leftarrow SP-1$

Machine Cycles: 2; States: 24; Flags: none; One-byte Instruction

RET pops the high-order and low-order bytes of the PC successively from the stack, decrementing the Stack Pointer by two. Generally this instruction immediately follows ACALL or LCALL. No flags are affected.

✓ **Example** RET

RETI (Return from interrupt)

$PC_{15-8} \leftarrow SP$

$SP \leftarrow SP-1$, $PC_{7-0} \leftarrow SP$

$SP \leftarrow SP-1$

Machine Cycles: 2; States: 24; Flags: none; One-byte Instruction

RETI pops the high-order and low-order bytes of the PC successively from the stack, and restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. The stack pointer is left decremented by two. No other registers are affected and the PSW is not automatically restored to its pre-interrupt status. Usually, this instruction immediately after the point at which the interrupt request was detected. If a lower- or same-level interrupt has been pending when the RETI instruction is executed, that one instruction will be executed before the pending interrupt is processed.

✓ **Example** RETI

AJMP 11-bit address (Absolute jump)

$PC \leftarrow PC+2$, $PC_{10-0} \leftarrow page\ address$

Machine Cycles: 2; States: 24; Flags: none; Two-bytes Instruction

AJMP instruction transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC after incrementing the PC twice, opcode bits 7–5, and the second byte of the instruction. The destination address must be within the same 2 K block of program memory.

✓ **Example** AJMP 11-bit address

LJMP 16-bit address (Long Jump)

$PC \leftarrow \text{addr}_{15-0}$,

Machine Cycles: 2; States: 24; Flags: none; Three-byte Instruction

LJMP instruction is an unconditional branch to the indicated address, by loading the high-order and low-order bytes of the PC with the second and third instruction bytes. The destination address will be anywhere in the full 64K program memory address space. No flags are affected.

✓ **Example** LJMP JMPADR

SJMP rel (Short Jump)

$PC \leftarrow PC+2, PC \leftarrow PC + \text{rel}$,

Machine Cycles: 2; States: 24; Flags: none; Two-byte Instruction

If SJMP rel instruction is executed, program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. The range of destinations allowed is from 128 bytes preceding this instruction to 127 bytes following it.

✓ **Example** SJMP RELADR

JMP @A+DPTR (Jump indirect relative to the DPTR)

$PC \leftarrow A+DPTR$,

Machine Cycles: 2; States: 24; Flags: none; One-byte Instruction

The eight-bit unsigned contents of the accumulator is added with the sixteen-bit data pointer, and load the result to the program counter. This will be the address for subsequent instruction fetches. No flags are affected.

✓ **Example** MOV DPTR,#8000H; JMP @A+DPTR. If the accumulator is equal to 04H, execution will jump to label 8004H memory location.

JZ rel (Jump if accumulator is zero)

$PC \leftarrow PC+2$, If $A=0$, then $PC \leftarrow PC + \text{rel}$,

Machine Cycles: 2; States: 24; Flags: none; Two-byte Instruction

If all bits of the accumulator are zero, jump to the indicated address; otherwise proceed with the next instruction. The branch destination address is computed by adding the signed relative-displacement to the PC, after incrementing the PC twice. The accumulator is not modified. No flags are affected.

✓ **Example** DEC A; JZ LABEL2

Assume the accumulator holds 01H. After execution of above instructions, the accumulator will change to 00H and cause jump to the label LABEL2.

JNZ rel (Jump if accumulator is not zero)

PC \leftarrow PC+2, If A \neq 0 then PC \leftarrow PC+rel,

Machine Cycles: 2; States: 24; Flags: none; Two-byte Instruction

If any bit of the accumulator is '1', jump to the indicated address; otherwise proceed with the next instruction. The branch destination address is computed by adding the signed relative-displacement to the PC, after incrementing the PC twice. The accumulator is not modified. No flags are affected.

✓ **Example** INC A; JNZ LABEL2

Assume the accumulator holds 00H. After execution of above instructions the accumulator will set to 01H and continue at label LABEL2.

CJNE <dest-byte, <src-byte>, rel instruction is used to compare the magnitudes of the first two operands, and branches if their values are not equal. The branch destination address is computed by adding the signed relative displacement to the PC, after incrementing the PC. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. All addressing mode combinations of CJNE instructions are explained below:

CJNE A, direct, rel (Compare direct byte to ACC and jump if not equal)

PC \leftarrow PC+3, If A<> direct, then PC \leftarrow PC + relative offset; If A<direct, then C \leftarrow 1, Else C \leftarrow 0.

Machine Cycles: 2; States: 24; Flags: none; Three-byte Instruction

Compare the content of data which is specified by direct memory location and accumulator, thereafter jump to destination address if not equal. The destination address is computed by addition of PC and relative offset address after incrementing PC by 3.

CJNE A, #data, rel (Compare immediate to ACC and jump if not equal)

PC \leftarrow PC+3, If A<> data, then PC \leftarrow PC + relative offset; If A<data, then C \leftarrow 1, Else C \leftarrow 0

Machine Cycles: 2; States: 24; Flags: none; Three-byte Instruction

Compare the contents of accumulator with 8-bit immediate data, thereafter jump to destination address if not equal. The destination address is computed by addition of PC and relative offset address after incrementing PC by 3.

CJNE R_n, #data, rel (Compare immediate to register and jump if not equal)

PC \leftarrow PC+3, If R_n<> data, then PC \leftarrow PC + relative offset; If R_n < data, then C \leftarrow 1, Else C \leftarrow 0

Machine Cycles: 2; States: 24; Flags: none; Three-byte Instruction

Compare the contents of register R_n with 8-bit immediate data; thereafter jump to destination address if not equal. The destination address is computed by addition of PC and relative offset address after incrementing PC by 3.

CJNE @R_i, #data, rel (Compare immediate to indirect and jump if not equal)

PC \leftarrow PC+3, If [R_i]<> data, then PC \leftarrow PC + relative offset; If R_i<data, then C \leftarrow 1, Else C \leftarrow 0

Machine Cycles: 2; States: 24; Flags: none; Three-byte Instruction

Compare the content of memory location which is specified by R0 or R1 and 8-bit immediate data, thereafter jump to destination address if not equal. The destination address is computed by addition of PC and relative offset address after incrementing PC by 3.

DJNZ <byte>, <rel-addr> instruction decrements the first operand by 1 and jump to the address indicated by the second operand if the resulting value is not zero. An original value of 00H will underflow to FFH. No flags are affected. The branch destination would be computed by adding the signed relative displacement

value to the PC, after incrementing the PC. The location of first operand may be a register or a directly addressed byte. Two types of DJNZ instructions are explained below:

DJNZ Rn, rel (Decrement register and jump if not zero)

$PC \leftarrow PC+2$, $R_n \leftarrow R_n-1$, If $R_n > 0$ or $R_n < 0$, then $PC \leftarrow PC + rel$

Machine Cycles: 2; States: 24; Flags: none; Two byte Instruction

Decrements the contents of register R_n by 1 and jump to the address indicated by the instruction if the resulting value is not zero. The branch destination would be computed by adding the signed relative-displacement value to the PC, after incrementing the PC by two.

✓ **Example** DJNZ R2, 8-bit offset address.

DJNZ direct,rel (Decrement direct byte and jump if not zero)

$PC \leftarrow PC+2$, direct \leftarrow direct-1, If direct >0 or direct <0 , then $PC \leftarrow PC + rel$

Machine Cycles: 2; States: 24; Flags: none; Three-byte Instruction

Decrements the contents of memory location which is specified by direct address, by 1 and jump to the address indicated by the instruction if the resulting value is not zero. The branch destination would be computed by adding the signed relative-displacement value to the PC, after incrementing the PC by two.

✓ **Example** DJNZ 40, 8-bit offset address.

NOP (No operation)

$PC \leftarrow PC+1$,

Machine Cycles: 1; States: 12; Flags: none; One-byte Instruction

Execution continues at the following instruction. Other than the program counter PC, no registers or flags are affected.

✓ **Example** NOP

14.3.6 PUSH, POP and EXCHANGE Instructions

The PUSH and POP instructions are used to manipulate stack related operations. All stack and exchange instructions are given Table 14.12 and their function are discussed as follows:

PUSH direct (Push direct byte onto stack)

$SP \leftarrow SP+1$; $[SP] \leftarrow direct$,

Machine Cycles 2; States: 24; Flags: none; Two-byte Instruction

The stack pointer is incremented by one. The contents of the indicated variable is then copied into the internal RAM location addressed by the stack pointer. No flags are affected.

✓ **Example** PUSH DPL

POP Direct (Pop direct byte from stack)

Direct $\leftarrow [SP]$; $SP \leftarrow SP -1$

Machine Cycles: 2; States: 24; Flags: none; Two-byte Instruction

The contents of the internal RAM location addressed by the stack pointer are read, and the stack pointer is decremented by one. The value read is then transferred to the directly addressed byte indicated. No flags are affected.

✓ **Example** POP DPH and POP SP

XCH A, <byte> loads the accumulator with the contents of the indicated variable, at the same time writing the original Accumulator contents to the indicated variable. The source/destination operand can use register, direct, or register-indirect addressing. All types of instructions are explained below:

XCH A, R_n (Exchange register with Accumulator)

$$A \leftrightarrow R_n$$

Machine Cycles: 1; States: 12; Flags: none; One-byte Instruction

Exchange the contents of specified register R_n with accumulator.

✓ **Example** XCH A, R3**XCH A, direct (Exchange direct byte with Accumulator)**

$$A \leftrightarrow \text{direct}$$

Machine Cycles: 1; States: 12; Flags: none; One-byte Instruction

Exchange the contents of memory location specified by direct address with accumulator.

✓ **Example** XCH A, 40H**XCH A, @R_i (Exchange indirect RAM with Accumulator)**

$$A \leftrightarrow [R_i]$$

Machine Cycles: 1; States: 12; Flags: none; One-byte Instruction

Exchange the contents of RAM location which is specified by R0 or R1 with accumulator.

✓ **Example** XCH A, @R0; Assume R0 contains the address 40H, the internal RAM location 40H holds 25H and accumulator holds 2FH. After execution of XCH A, @R0, the accumulator contains 25H and internal memory location content is 2FH.

XCHD A, @R_i (Exchange low-order digit indirect RAM with ACC)

$$A_{3-0} \leftrightarrow R_{i3-0}$$

Machine Cycles: 1; States: 12; Flags: none; One-byte instruction

XCHD instruction exchanges the low-order nibble of the accumulator (bits 3–0) with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7–4) of each register are not affected. No flags are affected.

✓ **Example** XCHD A, @R0

14.4 SIMPLE EXAMPLES IN ASSEMBLY-LANGUAGE PROGRAMS OF 8051 MICROCONTROLLER

Example 14.4

Store 8-bit immediate data (65H) into accumulator.

<i>Mnemonics</i>	<i>Opcode</i>	<i>Comments</i>
MOV	A, #65H	Store 65H into accumulator immediately

Example 14.5 Transfer the contents of B register into accumulator.

<i>Mnemonics</i>	<i>Opcode</i>	<i>Comments</i>
MOV	A, B	Copy the content of B register into accumulator

Example 14.6 Load 42H and 55H in registers R0 and R1 respectively.

<i>Mnemonics</i>	<i>Opcode</i>	<i>Comments</i>
MOV	R0, #42H	Load 42H in R0 register
MOV	R1, #55H	Load 55 in R1 register

Example 14.7 Place the contents of external memory location 8000H into accumulator.

<i>Mnemonics</i>	<i>Opcode</i>	<i>Comments</i>
MOV	DPTR, #8000H	Load 8000H in data pointer register immediately
MOVB	A, @DPTR	Copy the content of external memory location 8000H into accumulator

Example 14.8 Read the contents of external RAM locations 2000H and 2001H. Place values in R5 and R6 respectively.

<i>Mnemonics</i>	<i>Opcode</i>	<i>Comments</i>
MOV	DPTR, #2000H	Load 2000H in data pointer register immediately
MOVB	A, @DPTR	Copy the content of external memory location 2000H into accumulator
MOV	R5, A	Copy the content of accumulator in R5 register
INC	DPTR	Increment data pointer register
MOVB	A, @DPTR	Copy the content of external memory location 2001H into accumulator
MOV	R6, A	Copy the content of accumulator in R6 register

Example 14.9 Load 45H in external memory location 8000 H.

<i>Mnemonics</i>	<i>Opcode</i>	<i>Comments</i>
MOV	DPTR, #8000H	Load 8000H in data pointer register immediately
MOV	A, #45H	Load 45H into accumulator
MOVB	@DPTR, A	Copy the content of accumulator (45H) into external memory location 8000H

Example 14.10 Load 89 (HEX) in internal memory location 40 H.

<i>Mnemonics</i>	<i>Opcode</i>	<i>Comments</i>
MOV	40, #89H	Load 89H in internal memory location 40H

Example 14.11

Write program instructions to load a byte in memory location 9000H and increment the contents of the memory location.

<i>Mnemonics</i>	<i>Opcode</i>	<i>Comments</i>
MOV	DPTR, #9000H	Load 9000H in data pointer register immediately
MOV	A, #48H	Load 48H into accumulator
MOVB	@DPTR, A	Copy the content of accumulator 48H into external memory location 9000H
INC	A	Increment accumulator
MOVB	@DPTR, A	Load 49H, i.e., the content of accumulator into external memory location 9000H

Example 14.12

Write program instructions to load 44H in internal memory location 45H and decrement the contents of the memory.

<i>Mnemonics</i>	<i>Opcode</i>	<i>Comments</i>
MOV	45, #44H	Load 44H in internal memory location 45H immediately
MOV	A, 45H	Copy the content of internal memory location 45H into accumulator
DEC	A	Increment accumulator
MOV	45,A	Copy the content of internal memory location 45H into accumulator

Example 14.13

Load ABH in Register B. Then transfer the data to memory location 9050H.

<i>Mnemonics</i>	<i>Opcode</i>	<i>Comments</i>
MOV	B, #ABH	Load ABH in B register
MOV	DPTR, #9050H	Load 9050H in data pointer register immediately
MOV	A, B	Move B register to accumulator
MOVB	@DPTR,A	Load ABH i.e. the content of Accumulator into external memory location 9050H

Example 14.14

Store 01H, 02H, 03H and 04H in register R0, R1, R2 and R3 respectively and exchange data stored in Reg. R0 with R1 and data in Reg. R2 with R3.

<i>Mnemonics</i>	<i>Opcode</i>	<i>Comments</i>
MOV	R0, #01H	Load 01H in R0 register immediately
MOV	R1, #02H	Load 02H in R1 register immediately
MOV	R2, #03H	Load 03H in R2 register immediately
MOV	R3, #04H	Load 04H in R3 register immediately
MOV	A, R0H	Copy the content of R0 into accumulator
XCH	A,R1	The content of accumulator and register R1 are exchanged
MOV	R0,A	The content of accumulator into R0 register
MOV	A, R2	Copy the content of R2 into accumulator
XCH	A, R3	The content of accumulator and register R3 are exchanged
MOV	R2,A	The content of accumulator into R2 register

Example 14.15 Load data 12H and 34H in memory location 8500 and 8501. Transfer the contents of 8500 and 8501 to Register R0 and R1 respectively.

<i>Mnemonics</i>	<i>Opcode</i>	<i>Comments</i>
MOV	DPTR, #8500H	Load 8500H in data pointer register immediately
MOV	A, #12H	Load 12H into accumulator
MOVX	@DPTR, A	Copy the content of accumulator into external memory location 8500H
MOVX	A, @DPTR	Copy the content of external memory location 8500H into accumulator
MOV	R0, A	Move the content of accumulator into R0 register
INC	DPTR	Increment data pointer register
MOV	A, #34H	Load 34H into accumulator
MOVX	@DPTR, A	Copy the content of accumulator into external memory location 8501H
MOVX	A, @DPTR	Copy the content of external memory location 8501H into accumulator
MOV	R1, A	Move the content of accumulator into R1 register

Example 14.16 A 8-bit data is stored in 40H Memory Location. Find its one's complement and store it in 41H memory location.

<i>Mnemonics</i>	<i>Opcode</i>	<i>Comments</i>
MOV	R0, #40H	Load 40H in R0 register immediately
MOV	A, @R0	Load the content of 40H location into accumulator
CPL	A	Complement accumulator
INC	R0	Increment R0
MOV	@R0, A	Move the content of accumulator into 41H memory location

14.5 ASSEMBLY-LANGUAGE PROGRAMS

14.5.1 Addition of Two 8-bit Numbers whose Sum is 8 bits

Add 49 H and 56 H. The first number 49 H is in the external memory location 9001H. The second number 56 H is in the external memory location 9002 H. The result is to be stored in the external memory location 9003H.

<i>Memory address</i>	<i>Machine Codes</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	90 90 01	MOV	DPTR, #9001	Load 16-bit constant 9001 into DPTR
8003	E0	MOVX	A, @DPTR	Move the content of external memory 9001 into accumulator
8004	F5 0B	MOV	B, A	Move accumulator to B
8006	A3	INC	DPTR	Increment DPTR

(Contd.)

(Contd.)

8007	E0	MOVX	A, @DPTR	Move second data into accumulator
8008	25 0B	ADD	A ,B	Add B register with accumulator
800A	A3	INC	DPTR	Increment DPTR
800B	F0	MOVX	@DPTR,A	Store result into 9003H
800C	02 00 00	LJMP	0000	

DATA

9001–49 H

9002–56 H

RESULT

9003–9F H. The sum is stored in the memory location 9003 H

14.5.2 Program to Add Two Numbers and Store the Sum

Two data 24 H and 23 H are Stored in RAM locations 40H and 41H write a program to find the sum and Store at 42 H.

<i>Memory address</i>	<i>Machine Codes</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	78 40	MOV	R0, #40H	Load 40H in R0 register
8002	A6 24	MOV	@R0, #24H	Store 24H in 40H memory location
8004	E6	MOV	A, @R0	Content of 40H location in accumulator
8005	08	INC	R0	Increment R0
8006	76 23	MOV	@R0, #23H	Load 23H into 41 memory location
8008	26	ADD	A, @R0	Content of 41H location in accumulator
8009	08	INC	R0	Increment R0
800A	F6	MOV	@R0, A	Move the content of accumulator into 42H memory location
800B	02 00 00	LJMP	0000	Addition of two 8-bit numbers whose sum is 16 bits

14.5.3 Addition of Two 8-bit Numbers whose Sum is 16 Bits

Add 98 H and 9A H. The first number 98H is in the memory location 9001 H. The second number 9A H is in the memory location 9002H. The results are to be stored in 9003 and 9004H.

<i>Memory address</i>	<i>Machine Codes</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	90 90 01	MOV	DPTR, #9001	Load 16-bit constant 9001 into DPTR
8003	E0	MOVX	A, @DPTR	Move the content of external memory 9001 into accumulator
8004	F5 0B	MOV	B, A	Move accumulator to B
8006	A3	INC	DPTR	Increment DPTR

(Contd.)

(Contd.)

8007	E0	MOVX	A, @DPTR	Move second data into accumulator
8008	25 0B	ADD	A, B	Add B register with accumulator
800A	A3	INC	DPTR	Increment DPTR
800B	F0	MOVX	@DPTR, A	Store result into 9003H
800C	40 04	JC	8012	If carry flag is set, jump to 8012
800E	74 00	MOV	A, #00	Load 00H in accumulator
8010	A3	INC	DPTR	Increment DPTR
8011	F0	MOVX	@DPTR, A	Store 00H into 9004H
8012	74 01	MOV	A, #01	Load 01H in accumulator
8014	A3	MOVX	@DPTR, A	Store 01H into 9004H
8015	02 00 00	LJMP	0	

Addition of 98 H and 9A H is SUM = 01, 32 H. In this case, the sum is to be stored in two consecutive memory locations. The LSBs of the sum is 32H and it will be stored in the memory location 9003 H. The MSB of the sum is 01 which will be stored in 9004 H.

14.5.4 Addition of Ten 8-bit Numbers whose Sum is 16 Bits

Assume ten 8-bit numbers are stored in the internal RAM locations from 31H to 3A. After addition, MSD will be stored in R2 and LSD will be in R3.

PROGRAM 14.1

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	78 31		MOV	R0,#31	Load 31H in R0 register
8002	79 0A		MOV	R1, #0A	The number of data in R1 register
8004	E4		CLR	A	Clear accumulator and A becomes 00H
8005	FA		MOV	R2, A	Move the accumulator content into R2 register
8006	E6		MOV	A, @R0	Move the content of internal RAM location into accumulator
8007	08	LOOP	INC	R0	Increment R0 register to read next data
8008	26		ADD	A, @R0	Add next data with Accumulator
8009	50 01		JNC	Level_1	Jump no carry to Level-1
800B	0A		INC	R2	Increment R2 register
800C	D9 F9	Level_1	DJNZ	R1, LOOP	Repeat until R1 becomes 0
800E	FB		MOV	R3, A	Move the accumulator content into R3 register
800F	02 00 00		LJMP	00	

14.5.5 Addition of Ten 8-bit Numbers

The numbers are stored in the external RAM locations starting from 8000H. Sum will be 16 bit and Result will be stored in the memory location 8100H and 8101H.

PROGRAM 14.2

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	90 80 00		MOV	DPTR, #8000H	Load 8000H in data pointer register
8003	79 0A		MOV	R1, #0A	The number of data in R1 register
8005	E4		CLR	A	Clear accumulator and A becomes 00H
8006	FA		MOV	R2, A	Initialize R2 register
8007	E0		MOVX	A, @DPTR	Load first data in accumulator
8008	FB	LOOP	MOV	R3, A	Move data from A to R3
8009	A3		INC	DPTR	Increment DPTR register to MOVE next data in accumulator
800A	E0		MOVX	A, @DPTR	
800B	2B		ADD	A, R3	Add R3 register with accumulator
800C	50 01		JNC	Level_1	Jump no carry to Level_1
800E	0A		INC	R2	Increment R2 register
800F	D9 F7	Level_1	DJNZ	R1, LOOP	Repeat until R1 becomes 00H
8011	90 81 00		MOV	DPTR,#8100	Load 8100H in data pointer register
8014	F0		MOVX	@DPTR,A	Move ACC to external memory location 8100H
8015	A3		INC	DPTR	Increment DPTR
8016	E3		MOV	A,R2	Move R2 register to ACC
8017	F0		MOVX	@DPTR,A	Move ACC to external memory location 8101H
8018	02 00 00		LJMP	00	

14.5.6 Addition of Two 16-Bit Numbers whose Sum is more than 16 Bits

The first number is 2498H and the second number is FE4CH. After addition result will be stored in R2, R1 and R0 registers.

PROGRAM 14.3

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	C3		CLR	C	Clear carry C = 0
8001	7A 00		MOV	R2, #00H	R2 register is initialized
8003	74 98		MOV	A, #98	Least significant byte of first number in accumulator
8005	24 4C		ADD	A, #4CH	Add least significant byte of second number with accumulator

(Contd.)

(Contd.)

8007	F8		MOV	R0,A	Store result of lower byte in R0 register
8008	E5 24		MOV	A,24H	Most significant byte of first number in accumulator
800A	35 FE		ADDC	A,FEH	Add most significant byte of second number with accumulator
800C	50 01		JNC	Level_1	Jump no carry to Level_1
800E	0A		INC R2		Increment R2 register
800F	F9	Level_1	MOV	R1,A	Move the content of accumulator into R1 register
8010	02 00 00		LJMP	0000	

✓ **Result** After addition the content of R2, R1 and R0 registers as follows: R2 = 01H, R1 = 22H and R0 = E4H

	2	4	9	8	H	1 st number
	+F	E	4	C	H	2 nd number
Sum: 1	2	2	E	4	H	

14.5.7 Decimal Addition of Five 8-Bit Numbers

Assume five BCD numbers are stored from 41H to 45H. The result must be BCD and stored in memory locations 46H and 47H.

PROGRAM 14.4

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	79 05		MOV	R1, #05H	Number of data in R1 register
8002	78 41		MOV	R0, #41H	Load 41H in R0 register
8004	E4		CLR	A	Clear accumulator
8005	FA		MOV	R2, A	Content of accumulator in R2 register
8006	26	LOOP	ADD	A, @R0	The content of indirect RAM with accumulator
8007	D4		DAA		Decimal Adjustment
8008	50 00		JNC	Level_1	Jump no carry to Level_1
800A	0A		INC	R2	Increment R2
800B	08	Level_1	INC	R0	Increment R0
800C	D9 F8		DJNZ	R1, LOOP	Repeat until R1 become zero
800E	F5 46		MOV	46, A	Move LSD into 46H memory location
8010	8A 47		MOV	47, R2	Move MSD into 46H memory location
8012	02 00 00		LJMP	0000	

14.5.8 Program to Convert packed BCD to two ASCII Numbers and save them in R3 and R2

PROGRAM 14.5

<i>Memory address</i>	<i>Machine Codes</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	74 45	MOV	A, #45H	Load 45H in accumulator
8002	F8	MOV	R0, A	Move content of accumulator in R0 register
8003	54 0F	ANL	A, #0FH	Mask the upper nibble A = 05H
8005	44 30	ORL	A, #30H	Make it an ASCII, A = 35H
8007	FA	MOV	R2, A	Save ASCII equivalent into R2 register
8008	E8	MOV	A,R0	Move content of R0 register into accumulator
8009	54 F0	ANL	A, #F0H	Mask lower nibble, A = 40H
800B	03	RR	A	Rotate right accumulator
800C	03	RR	A	Rotate right accumulator
800D	03	RR	A	Rotate right accumulator
800E	03	RR	A	Rotate right accumulator, A = 04H
800F	44 30	ORL	A, #30H	Make it an ASCII, A = 34H
8011	FB	MOV	R3, A	Save ASCII equivalent into R3 register
8012	02 00 00	LJMP	00	

14.5.9 Subtraction of Two 8-Bit Numbers

Assume one number EFH is in accumulator and other number 45H in R0 register. After subtraction result will be stored in R1 register.

PROGRAM 14.6

<i>Memory address</i>	<i>Machine Codes</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	74 EF	MOV	A, #EFH	Load EFH into accumulator
8002	78 45	MOV	R0, #45H	Load 45H into R0 register
8004	98	SUBB	A, R0	Subtract the content of R0 from accumulator
8005	F9	MOV	R1, A	Move accumulator content into R1 register
8006	02 00 00	LJMP	0000	

14.5.10 One's Complement of an 8-bit Number

Assume 45H data is immediately loaded into accumulator. Complement accumulator and Store result in 9001H memory location.

PROGRAM 14.7

<i>Memory address</i>	<i>Machine Codes</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	74 45	MOV	A, #45	Load 45H into accumulator
8002	F4	CPL	A	Complement accumulator
8003	90 90 01	MOV	DPTR, #9001	Load 9001H in DPTR
8006	F0	MOVX	@DPTR,A	Store accumulator content in 9001H memory location
8007	02 00 00	LJMP	00	

14.5.11 Two's Complement of an 8-bit Number

Store the result in 9001H memory location.

PROGRAM 14.8

<i>Memory address</i>	<i>Machine Codes</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	C3	CLR	C	Clear carry
8001	74 4F	MOV	A, #4F	Move 4FH into accumulator
8003	F4	CPL	A	1's complement accumulator
8004	24 01	ADD	A, #01	1's complement + 1
8006	90 90 01	MOV	DPTR, #9001	Load 9001 in DPTR
8009	F0	MOVX	@DPTR, A	Store result in 9001H memory location
800A	02 00 00	LJMP	0000	

14.5.12 Shift an 8-bit Number Left by Two Bits and Store at 40H Memory Location

PROGRAM 14.9

<i>Memory address</i>	<i>Machine Codes</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	74 68	MOV	A, #68H	Load 68H into accumulator
8002	23	RL	A	Rotate accumulator left by one bit
8003	23	RL	A	Rotate accumulator left by one bit
8004	F5 40	MOV	40, A	Store accumulator content into 40H memory location

14.5.13 SWAP 4-MSBs with 4-LSBs in Accumulator and store in 9100H Memory Location

PROGRAM 14.10

<i>Memory address</i>	<i>Machine Codes</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	74 65	MOV	A, #65H	Load 65H into accumulator
8002	C4	SWAP	A	Swap command interchanges the low and high order nibbles
8003	90 91 00	MOV	DPTR, #9100	Load 9100 in DPTR
8006	F0	MOVX	@DPTR, A	Store accumulator content into 9100H memory location
8007	02 00 00	LJMP	00	

14.5.14 Move One Section of Data in Internal Memory to Another Section Internal Memory

Assume a section of data is stored in internal RAM starting from 40H. These data will be shifted to memory locations starting from 80H.

PROGRAM 14.11

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	78 40		MOV	R0, #40H	Store 40H in R0 register immediately
8002	79 80		MOV	R1, #80H	Store 80H in R1 register immediately
8004	AA 20		MOV	R2, #20	Store no of data, 20H in R2 register immediately
8006	E6	LOOP	MOV	A, @R0	Move register indirect memory location into accumulator
8007	F7		MOV	@R1, A	Move accumulator content into register indirect memory location
8008	08		INC	R0	Increment R0 register
8009	09		INC	R1	Increment R1 register
800A	DA FA		DJNZ	R2, LOOP	Repeat until R2 becomes zero
800C	02 00 00		LJMP	00	

14.5.15 Finding Largest Number from an Array of Numbers

Assume number of data is stored in 9000H and array of numbers is stored in external data memory starting from 9001H. Store the largest number in 9100H.

PROGRAM 14.12

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	90 90 00		MOV	DPTR, #9000H	Load 9000H in data pointer register
8003	E0		MOVX	A, @DPTR	Move content of data pointer into accumulator
8004	F8		MOV	R0, A	Move accumulator content into R0 register
8005	A3		INC	DPTR	Increment data pointer register
8006	E0		MOVX	A, @DPTR	Move content of data pointer into accumulator
8007	F9		MOV	R1, A	Copy accumulator content into R1 register
8008	18		DEC	R0	Decrement R0 register
8009	A3	Loop	INC	DPTR	Increment data pointer register
800A	E0		MOVX	A, @DPTR	Move content of data pointer into accumulator
800B	FA		MOV	R2, A	Move accumulator content into R0 register
800C	99		SUBB	A, R1	Subtract the content of R1 register from accumulator
800D	40 02		JC	Level	Jump no carry to shift Level
800F	EA		MOV	A, R2	Move R2 register to accumulator

(Contd.)

(Contd.)

8010	F9		MOV	R1,A	Move accumulator content into R0 register
8011	D8 F6	Level	DJNZ	R0, Loop	If R0 is not equal to zero, jump to Loop
8013	E9		MOV	A,R1	Move R1 to accumulator
8014	90 91 00		MOV	DPTR,#9100H	Load 9100H in DPTR register
8017	F0		MOVX	@DPTR,A	Move content of accumulator into memory location
8018	02 00 00		LJMP	0000	

14.5.16 Finding Smallest Number from an Array of Numbers

Assume number of data is stored in 9000H and array of numbers is stored in external data memory starting from 9001H.

PROGRAM 14.13

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	90 90 00		MOV	DPTR,#9000	Load 9000H in data pointer register
8003	E0		MOVX	A,@DPTR	Move content of data pointer into accumulator
8004	F8		MOV	R0,A	Move accumulator content into R0 register
8005	A3		INC	DPTR	Increment data pointer register
8006	E0		MOVX	A,@DPTR	Move content of data pointer into accumulator
8007	F9		MOV	R1,A	Copy accumulator content into R1 register
8008	18		DEC	R0	Decrement R0 register
8009	A3	LOOP	INC	DPTR	Increment data pointer register
800A	E0		MOVX	A,@DPTR	Move content of data pointer into accumulator
800B	FA		MOV	R2,A	Move accumulator content into R0 register
800C	99		SUBB	A,R1	Subtract the content of R1 register from accumulator
8000D	50 02		JNC	Level	Jump no carry to Level
800F	EA		MOV	A,R2	Move R2 register to accumulator
8010	F9		MOV	R1,A	Move accumulator content into R0 register
8011	D8 F6	Level	DJNZ	R0, LOOP	If R0 is not equal to zero, jump to LOOP
8013	E9		MOV	A,R1	Move R1 to accumulator
8014	90 91 00		MOV	DPTR,#9100	Load 9100H in DPTR register

(Contd.)

(Contd.)

8017	F0		MOVX	@DPTR,A	Move content of accumulator into memory location
8018	02 00 00		LJMP	0000	

14.5.17 Arranging a Series of Numbers in Descending Order

Assume number of data is stored in R0 register and array of numbers is stored in external data memory starting from 9000H.

PROGRAM 14.14

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	78 08		MOV	R0,#0A	Number of data bytes, 0A is stored in R0
8002	18		DEC	R0	Decrement R0 by 1
8003	90 90 00	LOOP1	MOV	DPTR,#9000	Load 9000H in DPTR
8006	E8		MOV	A,R0	Content of R0 in accumulator
8007	F9		MOV	R1,A	Content of accumulator in R1
8008	E0	LOOP2	MOVX	A,@DPTR	Move data into accumulator
8009	FA		MOV	R2,A	Copy data in R2
800A	A3		INC	DPTR	Increment DPTR
800B	E0		MOVX	A,@DPTR	Move next data in accumulator
800C	9A		SUBB	A,R2	Compare above two data
800D	50 08		JC	8017 LOOP3	Jump to LOOP 3 if carry flag is 0
800F	E0		MOVX	A,@DPTR	
8010	CA		XCH	A,R2	Exchange data in accumulator and R2 if carry flag is 1
8011	F0		MOVX	@DPTR,A	Replace current memory data by accumulator content
8012	15 82		DEC	82	Decrement DPL by1 DPL = DPL-1
8014	EA		MOV	A,R2	Move R2 content into accumulator
8015	F0		MOVX	@DPTR,A	Replace previous memory data by R2
8016	A3		INC	DPTR	Increment DPTR
8017	D9 EF	LOOP3	DJNZ	R1,8008 LOOP2	Decrement R1, if not zero, Jump to LOOP2
8019	D8 E8		DJNZ	R0,8003 LOOP1	Decrement R0, if not zero, Jump to LOOP1
801B	02 00 00		LJMP	0000	

14.5.18 Arrange a Data Array in Ascending Order

Assume number of data is stored in R0 register and array of numbers is stored in external data memory starting from 9000H.

PROGRAM 14.15

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	78 08		MOV	R0,#08	Number of data bytes, 08 is stored in R0
8002	18		DEC	R0	Decrement R0 by 1
8003	90 90 00	LOOP1	MOV	DPTR,#9000	Load 9000H in DPTR
8006	E8		MOV	A,R0	Content of R0 in accumulator
8007	F9		MOV	R1,A	Content of accumulator in R1
8008	E0	LOOP2	MOVX	A,@DPTR	Move data into accumulator
8009	FA		MOV	R2,A	Copy data in R2
800A	A3		INC	DPTR	Increment DPTR
800B	E0		MOVX	A,@DPTR	Move next data in accumulator
800C	9A		SUBB	A,R2	Compare above two data
800D	50 08		JNC	8017	Jump to LOOP 3 if carry flag is 0
800F	E0		MOVX	A,@DPTR	
8010	CA		XCH	A,R2	Exchange data in accumulator and R2 if carry flag is 1
8011	F0		MOVX	@DPTR,A	Replace current memory data by accumulator content
8012	15 82		DEC	82	Decrement DPL by1 DPL=DPL-1
8014	EA		MOV	A,R2	Move R2 content into accumulator
8015	F0		MOVX	@DPTR,A	Replace previous memory data by R2
8016	A3		INC	DPTR	Increment DPTR
8017	D9 EF	LOOP3	DJNZ	R1,8008	Decrement R1, if not zero, Jump to LOOP2
8019	D8 E8		DJNZ	R0,8003	Decrement R0, if not zero, Jump to LOOP2
801B	02 00 00		LJMP	0000	

14.5.19 Finding Square of a Number Using Look-Up Table**PROGRAM 14.16**

<i>Memory address</i>	<i>Machine Codes</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	90 90 00	MOV	DPTR,#9000H	Lookup table address
8003	78 05	MOV	R0,#05	Load the number 05 in R0
8005	E8	MOV	A,R0	Move data R0 to accumulator
8006	93	MOVC	A,@A+DPTR	Get square of 05 from table and stored in accumulator
8007	90 91 00	MOV	DPTR,#9100	Load 9100 in DPTR
800A	F0	MOVX	@DPTR, A	Send result to 9100H memory location
800B	02 00 00	LJMP	00	

<i>ADDRESS</i>	<i>SQUARE</i>
9000	00
9001	01
9002	04
9003	09
9004	16
9005	25
9006	36
9007	49
9008	64
9009	81

14.5.20 Program to Perform Multiplication of Two Numbers

PROGRAM 14.17

<i>Memory address</i>	<i>Machine Codes</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	74 24	MOV	A, #24	Load accumulator with first number (24H)
8002	75 0B 12	MOV	B, #12	Load register B with Second number (12H)
8005	A4	MUL	AB	Multiplication accumulator with B register and store result in accumulator
8006	F8	MOV	R0,A	Store LSB in R0
8007	A9 0B	MOV	R1,B	Store MSB in R1

14.5.21 Division of Two 8-bit Numbers

PROGRAM 14.18

<i>Memory address</i>	<i>Machine Codes</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	74 60	MOV	A, #60H	Load accumulator with first number (60H)
8002	75 0B 12	MOV	B, #12H	Load register B with Second number (12H)
8005	84	DIV	AB	Division accumulator with B register and store result in accumulator
8006	F8	MOV	R0,A	Store result in R0 register
8007	A9 0B	MOV	R1,B	Store remainder in R1 register

14.5.22 Write a Program for Logical AND of two 8-bit Data

PROGRAM 14.19

<i>Memory address</i>	<i>Machine Codes</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
8000	79 45	MOV	R1, #45H	Load first data byte, 45H in R0 register
8002	74 7F	MOV	A, #7FH	Load second data byte, 7FH in accumulator
8004	59	ANL	A,R1	Logical AND operation of accumulator and R1 register
8005	FA	MOV	R2,A	Store result in R2 register

14.6 APPLICATIONS OF MICROCONTROLLERS

14.6.1 Time Delay in Terms of Number of T States for the following Program

PROGRAM 14.20

<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
DELAY	MOV	R1,#FF	Outer loop counter = FFH = 256
	MOV	R0,#FF	Inner loop counter = FFH = 256
LOOP	DJNZ	R0,LOOP	Loop 256 times
	DJNZ	R1,LOOP	Loop 256 times
	RET		Return

The DJNZ instruction takes 24 clock periods or *T* states. Initially, the inner loop is executed 256 times. After that R1 is decremented by 1, again the inner loop is executed 256 times until R1 is not zero. As the outer loop is also executed for 256 times, total *T* states = $24 \times 256 \times 256 + 24 \times 256$ *T* states required to execute DJNZ R0, LOOP and DJNZ R0, LOOP instructions. Twelve *T* states are required to execute MOV R1, #FF and MOV R0, #FF instructions. Hence, the total *T* states for the DELAY loop is $12 + 12 + 24 \times 256 \times 256 + 24 \times 256$ *T* states = 1579032 *T* states. If the microcontroller operating frequency is 12 MHz, 24 *T* states is equal to 2 μ s. So that the time delay is equal to 1579032 *T* states = 131586 μ s = 0.13s

14.6.2 Program to Read the Port 0 and turn on LEDs for 1 Second and OFF for 1 second if Port 0 is not zero

The content of Port 0 states number of times LEDs ON. Assume that input switches are connected to Port 0 and output connected to Port 1 of 8051.

PROGRAM 14.21

<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
READ	MOV	A,P0	Read Port 0 switches
	JZ	READ	If no switch is on, jump to READ
	MOV	R0,A	Move Port 0 value in R0 register
ON	MOV	P1,#0FF	Turn ON all LEDs connected to Port 1
	CALL	DELAY	Call Delay Loop
OFF	MOV	P1,#00	Turn ON all LEDs connected to Port 1
	CALL	DELAY	Call Delay Loop
	DJNZ	R0,ON	Decrement R0 by 1, if R0 is not zero jump to ON
DELAY	MOV	R1,#FF	Outer loop counter = FH=256
	MOV	R2,#FF	Inner loop counter = FFH=256
	MOV	R3,#08	Middle loop counter for 9 times
LOOP	DJNZ	R2,LOOP	Loop 256 times
	DJNZ	R1,LOOP	Loop 256 times
	DJNZ	R3,LOOP	Loop 9 times
	RET		Return

To execute the delay program given in example 14.6.1, time required is about 0.13 s. If the said program is repeated 8 times, about 1 s delay will be generated. Therefore, a middle loop must be incorporated in the program as given above.

14.6.3 Program for Keyboard Interfacing with 8051 Microcontroller

Figure 14.9 shows the keyboard interfacing with 8051 microcontroller. It is clear from Fig. 14.9 that the keyboard is wired as a 4×4 row-column matrix. The low-order nibble of Port 0 is connected to the rows and the high-order nibble of Port 0 is connected to the columns. All rows and columns are connected with the 10 K pull-up resistors. As the I/O ports of the 8051 microcontroller can be used as bidirectional port to perform both read and write operations, therefore, the status of Port 0 can be read to scan the keyboard. Three different subroutines such as ROW_READ, COLUMN_READ and CONVERT are used to scan the key which is actually pressed.

To find out the row of the depressed key, assume all of the columns are LOW and all of the rows HIGH. This is possible by executing the instruction `MOV P0, #0FH`. The HIGH on the rows is actually a floating state and the rows to be read. The other instructions of ROW_READ subroutine are executed to read each row and to determine the row number that is LOW and the other three rows will be high.

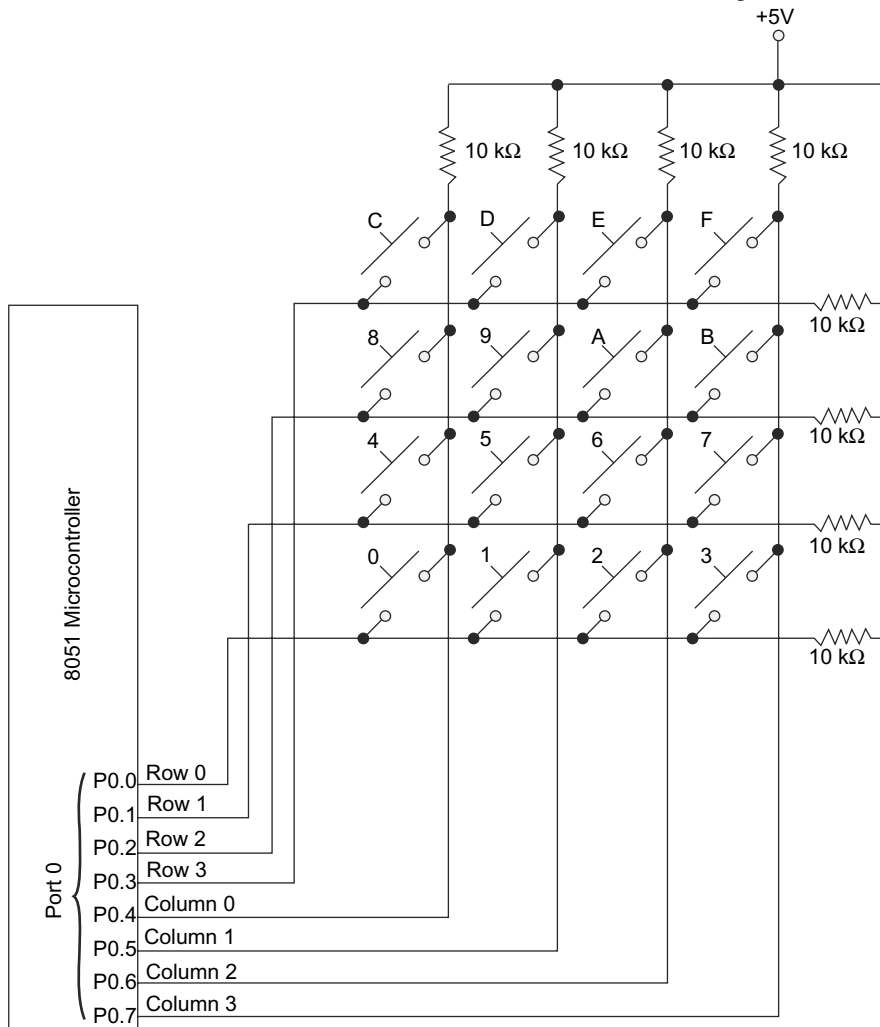


Fig. 14.9 Keyboard interfacing with 8051 microcontroller

After the row read operation, a column must be read to determine the column number by executing COLUMN_READ subroutine. When the instruction MOV P0, #F0 is executed, all of the rows are LOW and all of the columns are HIGH that float the columns. When any key is still depressed, the column of the key will be LOW. After execution of COLUMN_READ subroutine, the column number will be stored in R1 register.

Lastly, the CONVERT subroutine converts the row–column combination to the numeric value of the key pressed. Rows 0, 1, 2 and 3 have weighting factors of 0, 4, 8 and 12 respectively. Similarly, columns 0, 1, 2 and 3 have weighted factors 0, 1, 2 and 3 respectively. The CONVERT subroutine determines the numeric value of the key pressed by using formula: Key pressed = row \times 4 + column. The program for keyboard interfacing with 8051 microcontroller is given below:

PROGRAM 14.22(a)

<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
	CALL	ROW_READ	Find out row of key pressed
	CALL	COLUMN_READ	Find out column of key pressed
	CALL	CONVERT	Convert row/column to key value
	LJMP	0000	
ROW_READ	MOV	P0, #0F	Output 0s to all columns
	MOV	R0, #00	ROW=0
	JNB	P0.0, RET_1	If row 0 is LOW, return to RET_1
	MOV	R0, #01	ROW=1
	JNB	P0.1, RET_1	If row 1 is LOW, return to RET_1
	MOV	R0, #02	ROW=2
	JNB	P0.2, RET_1	If row 2 is LOW, return to RET_1
	MOV	R0, #03	ROW=3
	JNB	P0.3, RET_1	If row 3 is LOW, return to RET_1
	JMP	ROW_READ	Jump to ROW_READ
RET_1	RET		Return

PROGRAM 14.22(b)

<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
COLUMN_READ	MOV	P0,#0F0H	Output 0s to all rows
	MOV	R1,#00	COLUMN=0
	JNB	P0.4,RET_2	If column 0 is LOW, return to RET_2
	MOV	R1,#01	COLUMN = 1
	JNB	P0.4,RET_2	If column 1 is LOW, return to RET_2
	MOV	R1,#02	COLUMN =2
	JNB	P0.5,RET_2	If column 2 is LOW, return to RET_2
	MOV	R1,#03	COLUMN =3
	JNB	P0.6,RET_2	If column 3 is LOW, return to RET_2
	JMP	COLUMN_READ	Jump to COLUMN_READ
RET_2	RET		Return

PROGRAM 14.22(c)

Labels	Mnemonics	Operands	Comments
CONVERT	MOV	B,#04	Move multiplication factor = 04 to B register
	MOV	A,R0	Move row number to A
	MUL	AB	A = row × 4
	ADD	A, R1	A = row × 4 + column which is the key value
	RET		Return

14.6.4 Program for A/D Converter Interfacing with 8051 Microcontroller

Figure 14.10 shows the ADC0804 is interfaced with the 8051 microcontroller. The clock input for ADC is taken from the crystal oscillator of the microcontroller. As frequency is very high, two flip-flops are used to divide the frequency by 4. The connection start-of-conversion \overline{SC} and end-of-conversion \overline{EOC} signals are shown in Fig. 14.10. The steps of A/D converter is as follows:

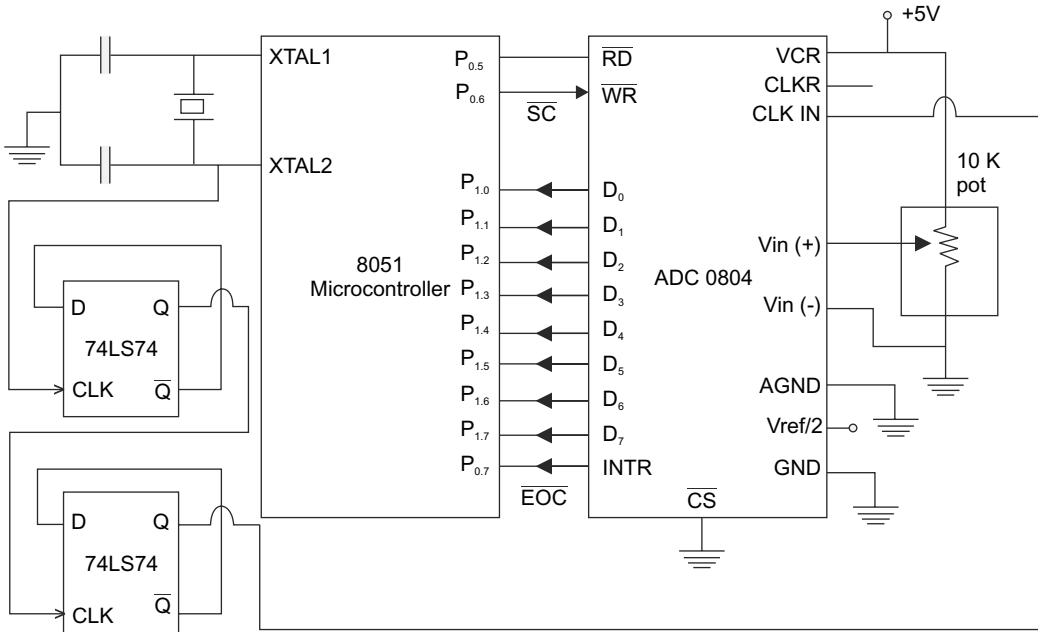


Fig. 14.10 Interfacing an ADC 0804 with 8051 microcontroller

- Step 1** The start of conversion \overline{SC} signal is sent to pin \overline{WR} to start the conversion.
- Step 2** INTR pin is connected with end-of-conversion \overline{EOC} signal. Keep monitoring the INTR pin to check end of conversion. If INTR is high, keep polling until it becomes low.
- Step 3** When the INTR is low, the A/D conversion is completed and the ADC0804 sent a high-to-low pulse to the \overline{RD} pin.

After the initial reset of the 8051 microcontroller, all I/O ports are in the floating condition. The first two instructions in this program, write ones to ports 1 and 0, which make them float. To start the DATA conversion, bit 6 of Port0 is pushed LOW then HIGH using the CLR and set bit SETB instructions. The conversion

process will continue and remains in a WAIT loop until bit 7 of Port 0 becomes LOW. Then the microcontroller reads the ADC output data through Port 1 and stores it in Register R 0. The program of A/D converter is given below:

PROGRAM 14.23

Labels	Mnemonics	Operands	Comments
	MOV	P1,#0FFH	All pins of Port 1 become 1
	MOV	P0,#0FFH	All pins of Port 0 become 1
	CLR	P0.6	Make P0.6 LOW to HIGH for \overline{SC}
	SETB	P0.6	
WAIT	JB	P0.7, WAIT	Wait until \overline{EOC} becomes low
	CLR	P0.5	Conversion is completed and RD enable
	MOV	R0,P1	Read the data from port P1 and store it in R0 register

14.6.5 Microcontroller-Based Traffic Control

Nowadays microcontrollers are used to implement traffic-control systems. Figure 14.11 shows the simple model of a microcontroller-based traffic control system. The various control signals such as red, green, orange,

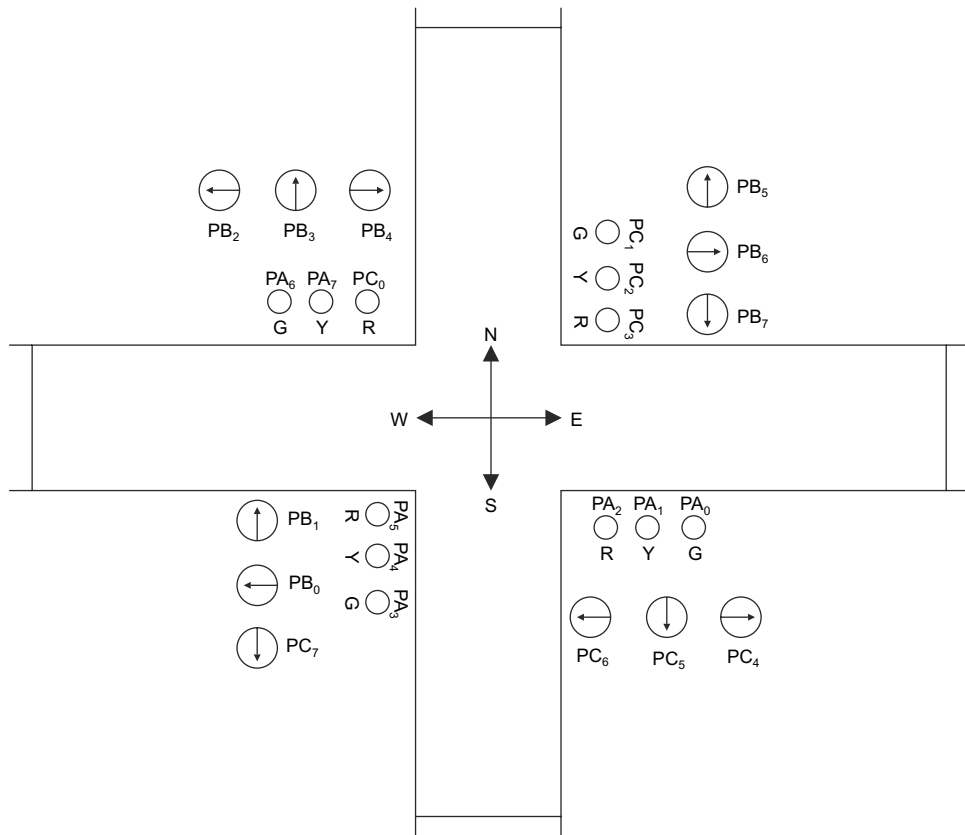


Fig. 14.11 Traffic-light control

forward arrow, right arrow and left arrow are used in this scheme. The forward, right and left arrows are used to indicate forward, right and left movement respectively. The red (R) signal is used to stop the traffic in the required lane and the yellow (Y) signal is used as standby, which indicates that the traffic must wait for the next signal. The green (G) light for a particular lane remains on for DELAY-1 seconds followed by the standby signal for DELAY-2 seconds. However, at a time, 3 out of the 4 roads, the left signal or the left arrow remains ON even though that lane may have a red signal. The traffic light control is implemented using an 8051 microcontroller kit having 8255 on board, and the interfacing circuit is illustrated in Fig. 14.12. Each signal is controlled by a separate pin of I/O ports. The total number of logic signals required for this arrangement is twenty-four. The programmable peripheral interface device 8255 is used to interface these 24 logic signals with the lamps. The logic '0' and '1' represent the state of the lamp. Logic '1' represents ON and '0' represents OFF. All ports of 8255 are used as output ports. The control word to make all ports as output ports for Mode 0 operation is 80H. The traffic light control program can be written by the following steps:

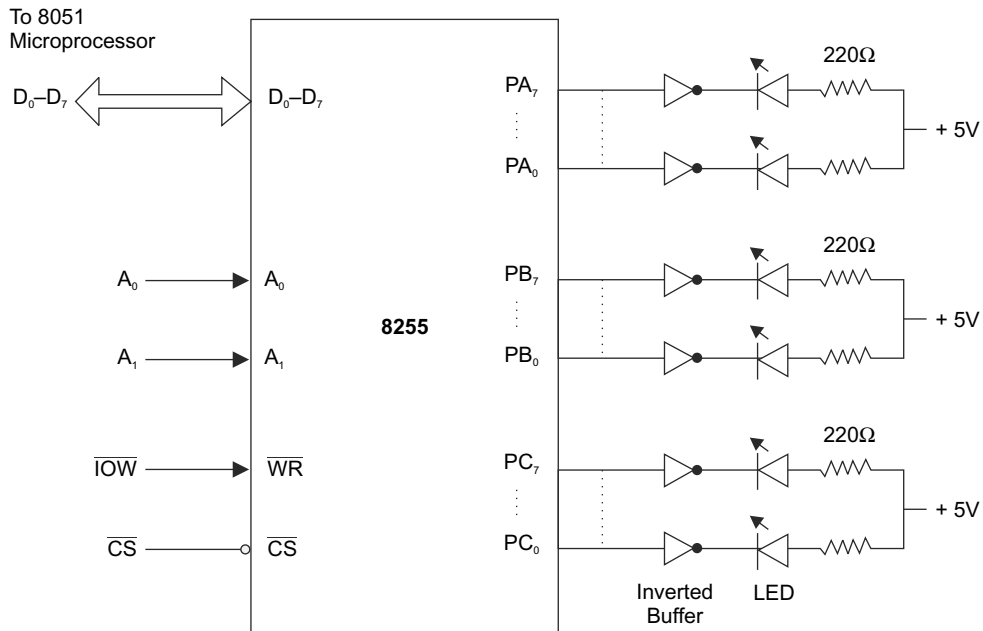


Fig. 14.12 The interfacing circuit for traffic-light control

Step 1 Initialize all ports of the 8255 as output ports.

Step 2 Determine the required status of Port A, Port B and Port C of 8255 for north to south traffic movement. Load data into accumulator and send to Port A, Port B and Port C for north to south traffic movement.

Step 3 Call delay subroutine -1.

Step 4 Before starting east to west traffic movement, north-to-south traffic movement will be ready to stop and east to west traffic must be ready for movement. Therefore, determine the required status of Port A, Port B and Port C for this operation. Then load data into accumulator and send to Port A, Port B and Port C for north-to-south traffic movement will be ready to stop and east-to-west traffic must be ready for movement.

Step 5 Call delay subroutine-2.

Step 6 For east-to-west traffic movement, determine the required status of Port A, Port B and Port C of 8255. Load data into accumulator and send to Port A, Port B and Port C for east-to-west traffic movement.

Step 7 Call delay subroutine-1.

Step 8 Prior to starting south-to-north traffic movement, east-to-west traffic will be ready to stop and south-to-north traffic must be ready for movement. For this operation, determine the status of Port A, Port B and Port C of 8255. Load required data into accumulator and send to Port A, Port B and Port C for east-to-west traffic will be ready to stop and south-to-north traffic must be ready for movement.

Step 9 Call delay subroutine-2.

Step 10 Determine the status of Port A, Port B and Port C for south-to-north traffic movement. Load require data into accumulator and send to Port A, Port B and Port C for south-to-north movement.

Step 11 Call delay subroutine-1.

Step 12 Before starting west-to-east traffic movement, south-to-north traffic movement will be ready to stop and west-to-east traffic must be ready for movement. Find out the status of Port A, Port B and Port C for this operation. Load required data into accumulator and send to Port A, Port B and Port C for south-to-north traffic movement will be ready to stop and west-to-east traffic must be ready for movement.

Step 13 Call delay subroutine-2.

Step 14 For west-to-east traffic movement, determine the status of Port A, Port B and port C of 8255. Load necessary data into accumulator and send to Port A, Port B and Port C for west-to-east traffic movement.

Step 15 Call delay subroutine-1.

Step 16 Subsequently west-to-east traffic movement will be ready to stop and north-to-south traffic must be ready for movement. Determine the status of Port A, Port B and Port C for this operation. Load needed data into accumulator and send to Port A, Port B and Port C. Then west-to-east traffic movement will be ready to stop and north-to-south traffic must be ready for movement.

Step 17 Call delay subroutine-2.

Step 18 Jump to Step 2.

The Chart below shows the bit assignment of ports. Putting 0s and 1s in required positions, the data byte for each port can be derived. For example, during north-to-south traffic movement, the statuses of Port A, Port B and Port C are as follows:

PA ₇	PA ₆	PA ₅	PA ₄	PA ₃	PA ₂	PA ₁	PA ₀
0	0	1	0	0	0	0	1
PB ₇	PB ₆	PB ₅	PB ₄	PB ₃	PB ₂	PB ₁	PB ₀
0	0	0	0	0	1	0	0
PC ₇	PC ₆	PC ₅	PC ₄	PC ₃	PC ₂	PC ₁	PC ₀
1	1	1	1	1	0	0	1

When north-to-south traffic movement will be ready to stop and east-to-west traffic must be ready for movement, the statuses of Port A, Port B and Port C are as follows:

PA ₇	PA ₆	PA ₅	PA ₄	PA ₃	PA ₂	PA ₁	PA ₀
0	0	0	1	0	0	1	0
PB ₇	PB ₆	PB ₅	PB ₄	PB ₃	PB ₂	PB ₁	PB ₀
0	0	0	0	0	1	0	0
PC ₇	PC ₆	PC ₅	PC ₄	PC ₃	PC ₂	PC ₁	PC ₀
0	0	0	0	1	0	0	1

The calculated necessary data bytes of Port A, Port B and Port C for all types of traffic movement are illustrated in Table 14.7 as given below:

Table 14.7 Traffic movement and status of ports

<i>Traffic movement</i>	<i>Status of Port A</i>	<i>Status of Port B</i>	<i>Status of Port C</i>
North-to-south traffic movement	21H	04H	F9H
North-to-south traffic movement be ready to stop and east-to-west traffic be ready for start	12H	04H	09H
East-to-west traffic movement	0CH	27H	89H
East-to-west traffic movement be ready to stop and south-to-north traffic are ready for start	94H	20H	08H
South-to-north traffic movement	64H	3CH	18H
South-to-north traffic movement be ready to stop and west-to-east traffic be ready for start	A4H	00H	14H
West-to-east traffic movement	24H	D0H	93H
West-to-east traffic movement be ready to stop and north-to-south traffic be ready for start	22H	00H	85H

The green light is provided for the traffic flowing from north to south. The arrows indicate the deviations in which traffic is allowed to move. The arrows with a cross indicate that the traffic is not allowed to move in that particular direction. Each signal is controlled by a separate port. The various signals used are red, green, orange, forward arrow, right and left arrows. The forward arrow, right and left arrows are used to indicate forward, right and left movement. The red signal is used to stop traffic in the required lane and the orange signal is used as standby which indicates that the traffic must wait for the next signal. The green lights for a particular lane remain on for 10 seconds followed by the standby signal for 4 seconds. However, at a time, for 3 out of the 4 roads, the left signal or the left arrow remains ON even though that lane may have a red signal. This system is implemented using 8051 trainer having 8255 on board. The 8051 is interfaced with the 8255 and output pins are used to control the various signals. The program for traffic-light control is given below.

PROGRAM 14.24

<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
	MOV	0A0,#0E8	Load control word of 8255 into Control word register whose address is E803H. Control word is 80H
	MOV	R0,#03	
	MOV	A,#80	
	MOVX	@R0,A	
START	MOV	R0,#00	Send 21H in Port A, F9H in Port C and 04H in Port B for north-to south traffic movement.
	MOV	A,#21	
	MOVX	@R0,A	

<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
	MOV	R0,#02	
	MOV	A,#F9	
	MOVX	@R0,A	
	MOV	R0,#01	
	MOV	A,#04	
	MOVX	@R0,A	
	LCALL	DEDAY-1	Call Delay-1 subroutine.
	MOV	R0,#00	Send 12H in Port A, 09H in Port C and 04H in Port B for north-to-south traffic movement will be ready to stop and east-to-west traffic movement is ready to start.
	MOV	A,#12	
	MOVX	@R0,A	
	MOV	R0,#02	
	MOV	A,#09	
	MOVX	@R0,A	
	MOV	R0,#01	
	MOV	A,#04	
	MOVX	@R0,A	
	LCALL	DELAY-2	Call Delay-2 subroutine
	MOV	R0,#00	Send 0CH in Port A, 89H in Port C and 27H in Port B for east-to-west traffic movement.
	MOV	A,#0C	
	MOVX	@R0,A	
	MOV	R0,#02	
	MOV	A,#89	
	MOVX	@R0,A	
	MOV	R0,#01	
	MOV	A,#27	
	MOVX	@R0,A	
	LCALL	DEDAY-1	Call Delay-1 subroutine.
	MOV	R0,#00	Send 94H in Port A, 08H in Port C and 20H in Port B for east-to-west traffic movement will be ready to stop and south-to-north traffic movement is ready to start.
	MOV	A,#94	
	MOVX	@R0,A	
	MOV	R0,#02	
	MOV	A,#08	
	MOVX	@R0,A	
	MOV	R0,#01	
	MOV	A,#20	
	MOVX	@R0,A	
	LCALL	DELAY-2	Call Delay-2 subroutine.
	MOV	R0,#00	Send 64H in Port A, 18H in Port C and 3CH in Port B for south-to North traffic movement.
	MOV	A,#64	
	MOVX	@R0,A	

<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
	MOV	R0,#02	
	MOV	A,#18	
	MOVX	@R0,A	
	MOV	R0,#01	
	MOV	A,#3C	
	MOVX	@R0,A	
	LCALL	DELAY-1	Call Delay-1 subroutine.
	MOV	R0,#00	
	MOV	A,#A4	Send A4H in Port A, 14H in Port C and 00H in Port B for south-to-north traffic movement is ready to stop and west-to-east traffic movement will be ready to start.
	MOVX	@R0,A	
	MOV	R0,#02	
	MOV	A,#14	
	MOVX	@R0,A	
	MOV	R0,#01	
	MOV	A,#00	
	MOVX	@R0,A	
	LCALL	DEDAY-2	Call Delay-2 subroutine.
	MOV	R0,#00	Send 24H in Port A, 93H in Port C and D0H in Port B for west-to-east traffic movement.
	MOV	A,#24	
	MOVX	@R0,A	
	MOV	R0,#02	
	MOV	A,#93	
	MOVX	@R0,A	
	MOV	R0,#01	
	MOV	A,#D0	
	MOVX	@R0,A	
	LCALL	DELAY-1	Call Delay-1 subroutine.
	MOV	R0,#00	Send 22H in Port A, 85H in Port C and 00H in Port B for west-to-east traffic movement is ready to stop and north-to-south traffic movement will be ready to to start.
	MOV	A,#22	
	MOVX	@R0,A	
	MOV	R0,#02	
	MOV	A,#85	
	MOVX	@R0,A	
	MOV	R0,#01	
	MOV	A,#00	
	MOVX	@R0,A	
	LCALL	DELAY-2	Call Delay-2 subroutine.
	LJMP	START	Jump to START.

DELAY-1 SUBROUTINE for 10 SECONDS

Labels	Mnemonics	Operands
	MOV	R4,#0A
LOOP-3	MOV	R7,#08
	MOV	R5,#00
LOOP-2	MOV	R6,#F3
LOOP-1	DJNZ	R5,LOOP_1
	DJNZ	R6,LOOP_1
	DJNZ	R7,LOOP_2
	DJNZ	R4,LOOP_3
	RET	

DELAY-1 SUBROUTINE for 4 SECONDS

Labels	Mnemonics	Operands
	MOV	R4,#04
LOOP-3	MOV	R7,#08
	MOV	R5,#00
LOOP_5	MOV	R6,#F3
LOOP_4	DJNZ	R5,LOOP_4
	DJNZ	R6,LOOP_4
	DJNZ	R7,LOOP_5
	DJNZ	R4,LOOP_6
	RET	

14.6.6 Microcontroller-Based Stepper Motor Control

Stepper motors are electromechanical devices, which convert electrical pulses into proportionate discrete mechanical rotational movement. To rotate the stepper motor's shaft, a sequence of pulses is required to be applied to stator windings of a stepper motor. When a given number of command pulses are supplied to the motor, the shaft will have turned through a known angle. Therefore, the motor can be used to control position by keeping count of the number of command pulses. Each revolution of the stepper motor's shaft is made up of a series of discrete individual steps. A step is defined as the angular rotation produced by the shaft each time when the motor receives a step pulse. Due to each step, the shaft can rotate a specified angle in degrees. The rotation of the shaft due to each step is called *step angle*. The stepper motors are usually used in position control of robot arms, paper-drive mechanism

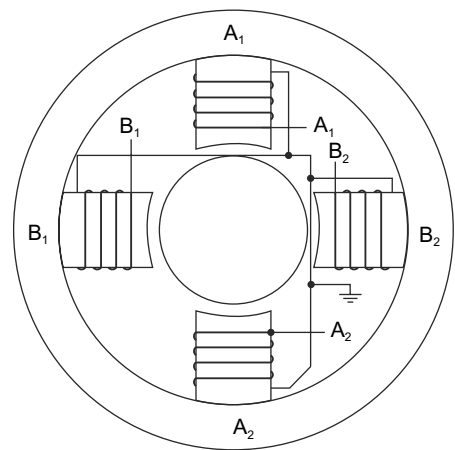


Fig. 14.13 Four-phase stepper motor windings

in a printer, machine-tools control, process-control system, textile industry, integrated circuit fabrication, electric watches, tapes as well as disk drive systems, etc. Further, the average motor speed is proportional to rate at which the pulse command is delivered. At low-command pulse rate, the rotor moves in steps, but when the pulse rate is made sufficiently high, because of the inertia, the rotor moves smoothly, as in case of dc motors. As motor speed is proportional to rate of command pulses, it can be used for speed control.

Figure 14.13 shows four-phase stepper motor windings and its interfacing is depicted in Fig. 14.14. The four windings A_1 , A_2 , B_1 and B_2 are connected to PA_3 , PA_2 , PA_1 and PA_0 respectively. When PA_3 is level '1' and PA_1 is level '1', the coils A_1 and B_1 are energized and the motor will rotate by one step clockwise. Similarly, coils A_1 and B_2 will be energized when PA_3 is level '1' and PA_0 is in level '1' and again the motor rotates by one step. In the same way, the other phases are energized sequentially as per Table 14.8 and switching sequence waveform of windings is illustrated in Fig. 14.15. The assembly-language program for stepper-motor control in clockwise as well as anti-clockwise rotation is illustrated below.

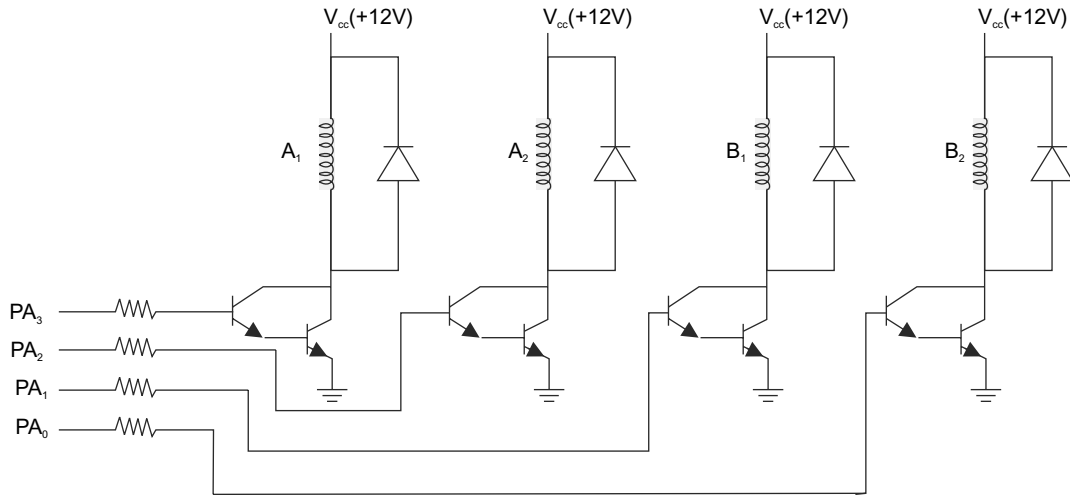


Fig. 14.14 Driver circuit of stepper motor

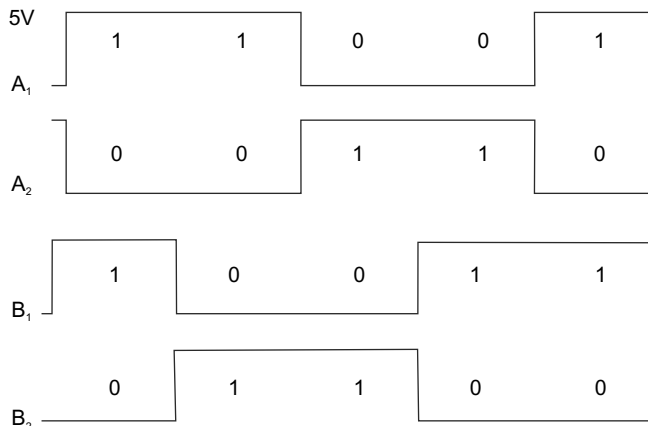


Fig. 14.15 Switching sequence waveform of windings

Table 14.8 Sequence of switching of windings

A_1	A_2	B_1	B_2	Clockwise CW	Counter Clockwise	CCW
1	0	1	0	A	↓	A
1	0	0	1	9		9
0	1	0	1	5		5
0	1	1	0	6		6

PROGRAM 14.25

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	74 80		MOV	A,#80	Load control word 80H in 8255
8002	90 E8 03		MOV	DPTR,#E803	
8005	F0		MOVX	@DPTR,A	
8006	74 00	START	MOV	A,#00	
8008	12 01 61		LCALL	0161	Call subroutine to read character
800B	B4 55 03		CJNE	A,#55,8011	Compare if U is pressed
800E	12 D0 00		LCALL	D000	Call clockwise rotation subroutine
8011	B4 44 03		CJNE	A,#44,8017	Compare if D is pressed
8014	12 90 00		LCALL	9000	Call Counter clockwise rotation subroutine
8017	02 80 06		LJMP	8006	Jump to 8006

Subroutine for Clockwise Rotation

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
D000	74 0A	LOOP_1	MOV	A, #0A	Load 0A in accumulator
D002	12 C0 00		LCALL	C000	Call subroutine to send data, 0AH in Port A
D005	12 B0 00		LCALL	B000	Call delay subroutine
D008	74 09		MOV	A,#09	Load 09 in accumulator
D00A	12 C0 00		LCALL	C000	Call subroutine to send data, 09H in Port A
D00D	12 B0 00		LCALL	B000	Call delay subroutine
D010	74 05		MOV	A,#05	Load 05 in accumulator
D012	12 C0 00		LCALL	C000	Call subroutine to send data, 05H in Port A
D015	12 B0 00		LCALL	B000	Call delay subroutine
D018	74 06		MOV	A,#06	Load 06 in accumulator
D01A	12 C0 00		LCALL	C000	Call subroutine to send data, 06H in Port A
D01D	12 B0 00		LCALL	B000	Call delay subroutine
D020	02 D0 00		LJMP	D000	Jump to LOOP_1

Subroutine for Counter Clockwise Rotation

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
9000	74 06	LOOP_2	MOV	A, #06	Load 0A in accumulator
9002	12 C0 00		LCALL	C000	Call subroutine to send data, 0AH in Port A
9005	12 B0 00		LCALL	B000	Call delay subroutine
D008	74 05		MOV	A, #05	Load 09 in accumulator
900A	12 C0 00		LCALL	C000	Call subroutine to send data, 09H in Port A
900D	12 B0 00		LCALL	B000	Call delay subroutine
9010	74 09		MOV	A, #09	Load 05 in accumulator
9012	12 C0 00		LCALL	C000	Call subroutine to send data, 05H in Port A
9015	12 B0 00		LCALL	B000	Call delay subroutine
9018	74 0A		MOV	A, #0A	Load 06 in accumulator
901A	12 C0 00		LCALL	C000	Call subroutine to send data, 06H in Port A
901D	12 B0 00		LCALL	B000	Call delay subroutine
9020	02 90 00		LJMP	9000	Jump to LOOP_2

Subroutine to Send data in Port A

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
C000	90 EB 00		MOV	DPTR, #E800	Load Port A address E800H in DPTR
C003	F0		MOVX	@DPTR, A	Send accumulator content into Port A
C004	22		RET		Return

Delay Subroutine

<i>Memory address</i>	<i>Machine Codes</i>	<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
B000	7B 10		MOV	R3, #10	Move 10H into R3
B002	7C FF		MOV	R4, #FF	Move FFH into R4
B004	1B		DEC	R3	Decrement R3
B005	BB 00 FC		CJNE	R3, #00, B004	Compare R3 with 00. if R3≠0, jump to B004
B008	1C		DEC	R4	Decrement R4
B009	BC 00 FC		CJNE	R4, #00, B008	Compare R4 with 00. if R4≠0, jump to B008
B00C	22		RET		Return

14.6.7 Microcontroller-Based Washing-Machine Control

Since early days, many efforts have been made to overcome the difficulties associated with washing clothes manually. It was very painful, wearisome and time consuming to constantly wash bulky clothes, especially the very dirty ones. Due to research for easier and highly efficient ways of washing cloths, electrical washing machines evolved. Presently, washing machines are incorporated with timers. The timer is a device which is subject to the effects of wear and tear and high cost of maintenance. Nowadays, microcontroller-controlled

washing machines are available in the market and these machines are highly efficient, reliable and durable with less running cost. Such a washing machine is controlled by a controller based on a software program. Usually, the controller is designed to control the pumping of water in and out of the machine and the washing and the rinsing of the clothes. The controller is very flexible and user friendly.

Figure 14.16 shows an automatic washing machine with top loading capability. On the top of the machine, there are four knobs for controller input settings. The functions of knobs are as follows:

✓ **Load Select Knob** The load of a washing machine is the number of clothes to be washed. There are three settings such as *Low*, *Medium* and *High*. According to load, the controller decides the amount of water which to be filled by the electric pump.

✓ **Water Inlet Select Knob** This knob is used to select hot, normal or mixed water. There are two inlet pipes for hot and normal tap water at the rear of the machine. The hot and cold lines are hooked up to the body of a solenoid valve as shown in Fig. 14.17. There are two valves, but they feed into a single house. Therefore, depending on the specified temperature, the hot valve, the cold valve or both valves will open.



Fig. 14.16 Automatic washing machine

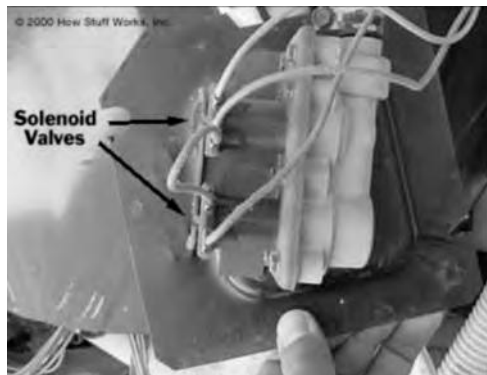


Fig. 14.17 Water-control valves

✓ **Mode Select Knob** The washing machine can be operated in *Save mode* and *Normal mode*. In *Save mode*, the machine can be used to save detergent. After washing a lot of clothes, the user should take out the clothes from the machine and put another lot of clothes and restart the machine. During *Normal mode*, the machine operates on the following steps:

Step 1 The clothes are washed for a specified time which is set by the controller.

Step 2 The detergent water is drained.

Step 3 The fresh water is put through pump.

Step 4 The clothes are rinsed for a specified time which is set by the controller.

Step 5 The water is drained and the moisture is absorbed from clothes

✓ **Program Select Knob** Usually, the machine is programmed to wash clothes in four different settings such as *Heavy*, *Normal*, *Light* and *Delicate*. When the clothes are very dirty, the heavy knob will be selected. For normal dirty clothes, the normal knob is selected. Delicate is used for silk clothes. To wash clothes completely, the controller should select the following parameters based on the Program Select Knob.

✓ **Fill** Water will be filled by the pump as per Load Select Knob. During first fill, the water temperature will be selected by proper tap setting. During second fill after drain, only tap water is filled for rinsing the clothes. The filling time will be controlled by relay setting.

✓ **Agitate** The wash basket will rotate in a clockwise direction for ten revolutions. After that, the wash basket will stop for 2 seconds. Then the wash basket will rotate in anticlockwise direction for ten revolutions. This process will continue for specified minutes as shown in Table 14.9. Figure. 14.18 shows the inner tub when it has been removed from the outer tub. It is resting on the gearbox, and the plastic agitator is visible in the center of the tub. Figure. 14.18(b) shows the gearbox when the inner tub is removed. The inner tub bolts to the three holes in the flange of the gearbox.

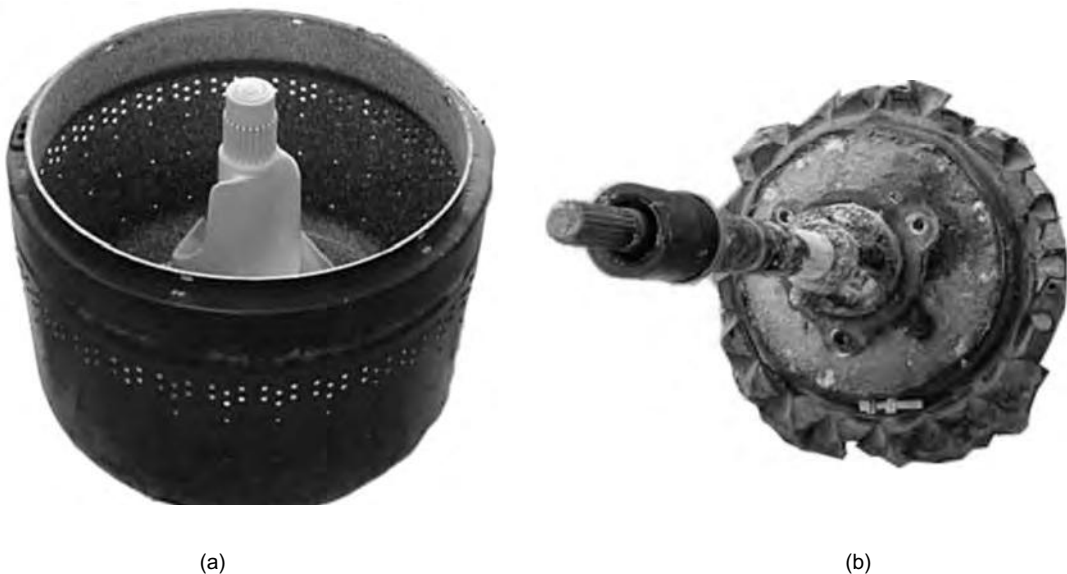


Fig. 14.18 (a) Inner tub and plastic agitator, and (b) gear box

✓ **Drain**

After agitation, the water and detergent are drained.

✓ **Spin**

During spin, the agitator does not move and the wash basket will rotate at very high speed. Then the moisture of clothes are removed through holes in the inner metallic basket. Table 14.9 shows a complete washing cycle.

Table 14.9 Washing cycle for Heavy, Normal, Light and Delicate setting

Operation	Heavy	Normal	Light	Delicate
Fill water	Set by Load Select Knob	Set by Load Select Knob	Set by Load Select Knob	Set by Load Select Knob
Agitate	20 minutes	15 minutes	10 minutes	5 minutes
Drain	5 minutes	5 minutes	5 minutes	5 minutes
Fill water	Set by Load Select Knob	Set by Load Select Knob	Set by Load Select Knob	Set by Load Select Knob
Agitate	10 minutes	10 minutes	5 minutes	5 minutes
Drain	5 minutes	5 minutes	5 minutes	5 minutes
Spin	10 minutes	10 minutes	5 minutes	5 minutes

✓ **Indicators** When the machine is ON, a LED will be ON to indicate its state. After completion of washing cycle, a buzzer sound will be generated.

Circuit Diagram of Washing-Machine Controller

Figure 14.19 shows the circuit diagram of washing-machine-controller. The input signals of the controller are *Load/Water Level Select*, *Water Inlet* and *Program Select*. The level measurement transducers are used to measure level of water of the machine. the transducers outputs are connected to P0.0 to P0.2, Hot and Normal water knobs are connected to pins P0.3 to P0.4 respectively, and Program Select Knobs are attached to pins P1.0 to P1.3 as shown in Table 14.10.

The output signals of washing-machine controller are Machine ON, Fill water, Agitation control, Drain, Spin and Washing complete. These output signals are available from P2.0, P2.1, P2.2, P2.3, P2.4, P2.5, P2.6 and P2.7 respectively. The flowchart for program of a washing-machine controller is depicted in Fig. 14.20.

Table 14.10 Operation, signals, I/O and port numbers

Operation	Signal	Input/Output	Port Pin No.
Load/Water level select	Water level low	Input	P0.0
	Water level medium	Input	P0.1
	Water level high	Input	P0.2
Water Inlet	Hot water knob	Input	P0.3
	Normal water knob	Input	P0.4
Program select	Heavy	Input	P1.0
	Normal	Input	P1.1
	Light	Input	P1.2
	Dedicate	Input	P1.3
Machine ON	Machine ON indication	Output	P2.0
Fill water	Hot water inlet	Output	P2.1
	Normal water inlet	Output	P2.2
Agitation control	Motor rotate in clockwise direction	Output	P2.3
	Motor rotate in anticlockwise direction	Output	P2.4
Drain	Drain valve open	Output	P2.5
Spin	Spin motor ON/OFF	Output	P2.6
Washing complete	Washing complete indication	Output	P2.7

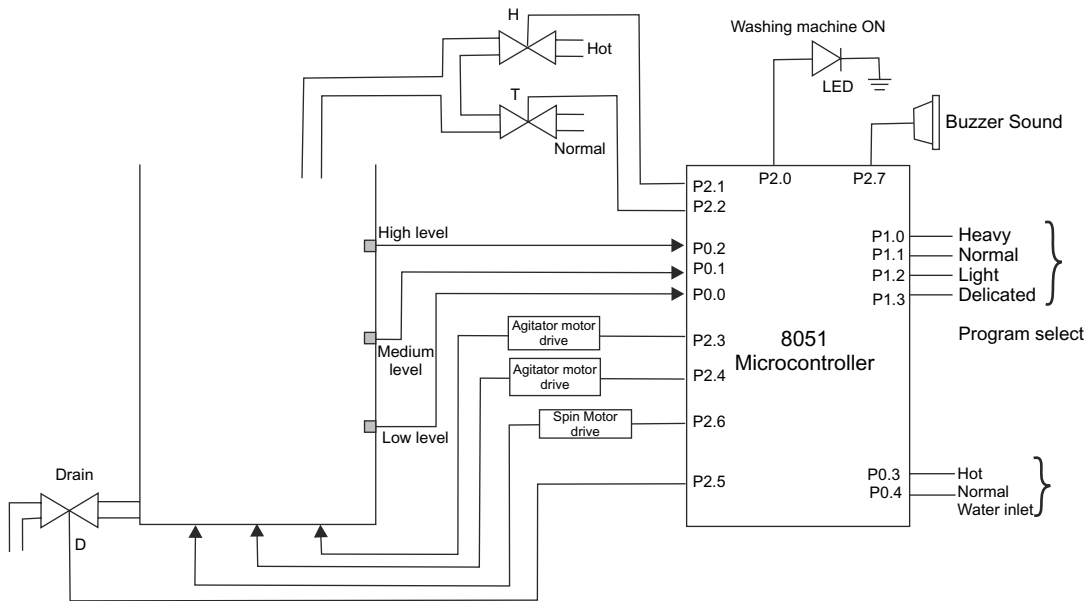


Fig. 14.19 Washing machine control circuit

Software of Washing-Machine Controller

The flowchart for sequence of operations of a washing machine is shown in Fig. 14.20. The integer number in parentheses states the minutes of operation. For example, agitation (10) means that agitation will continue for 10 minutes. The washing-machine controller software consists of a main program and subroutine programs for the following operations:

- ✓ **Fill Operation** Fill machine with water.
- ✓ **Agitation Operation** The agitator rotates 10 revolutions in clockwise direction, stops for one second followed by 10 revolutions in anticlockwise direction. This process will run for specified minutes as indicated in parentheses.
- ✓ **Drain Operation** Remove water from tank. This operation is also performed for specified minutes.
- ✓ **Spin Operation** In this operation, the moisture of clothes is removed.

In this section, only the main program of the washing-machine controller is given and the programmer should write the subroutine programs so that the machine can operate as per washing cycle.

Table 14.11 Commands for washing-machine controller

Labels	Mnemonics	Operands	Comments
	SETB	P2.0	Machine ON indication
	LCALL	FILL_1	Machine fill with water first time
	JNB	P1.0, LOOP_1	Check program setting knob for HEAVY. If P1.0 is not set, Jump to LOOP_1
	SJMP	HEAVY	If P1.0 is set, Jump to HEAVY
LOOP_1	JNB	P1.1, LOOP_2	Check program setting knob for NORMAL. If

LOOP_2	SJMP JNB	NORMAL P1.2, LOOP_3	P1.1 is not set, Jump to LOOP_2 If P1.1 is set, Jump to NORMAL
LOOP_3	SJMP JNB	LIGHT P1.3, LOOP_4	Check program setting knob for LIGHT. If P1.2 is not set, Jump to LOOP_3 If P1.2 is set, Jump to LIGHT
DISPLAY	SJMP SETB	DELICATE P2.7	Check program setting knob for DELICATE. If P1.3 is not set, Jump to LOOP_4 If P1.2 is set, Jump to DELICATE
LOOP_4	NOB LJMP	0000	Indicate the completion of wash cycle End of program

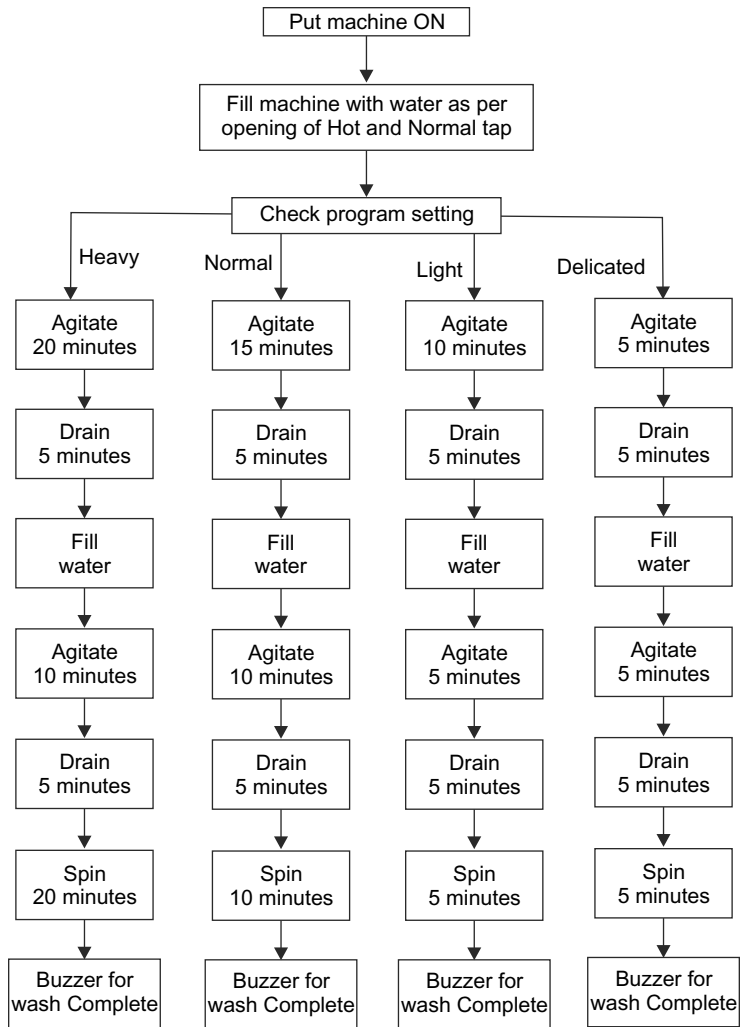


Fig. 14.20 Flowchart for washing-machine controller

PROGRAM 14.26(a)

<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
HEAVY	LCALL	AGITATE(20)	Agitate for 15 minutes
	LCALL	DRAIN(5)	Drain operation for 5 minutes
	LCALL	FILL_2	Machine fill with water second time
	LCALL	AGITATE(10)	Agitate for 10 minutes
	LCALL	DRAIN(5)	Drain operation for 5 minutes
	LCALL	SPIN(10)	Spin for 10 minutes
	SJMP	DISPLAY	Jump to DISPLAY

PROGRAM 14.26(b)

<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
NORMAL	LCALL	AGITATE(15)	Agitate for 15 minutes
	LCALL	DRAIN(5)	Drain operation for 5 minutes
	LCALL	FILL_2	Machine fill with water second time
	LCALL	AGITATE(10)	Agitate for 10 minutes
	LCALL	DRAIN(5)	Drain operation for 5 minutes
	LCALL	SPIN(10)	Spin for 10 minutes
	SJMP	DISPLAY	Jump to DISPLAY

PROGRAM 14.26(c)

<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
LIGHT	LCALL	AGITATE(10)	Agitate for 15 minutes
	LCALL	DRAIN(5)	Drain operation for 5 minutes
	LCALL	FILL_2	Machine fill with water second time
	LCALL	AGITATE(5)	Agitate for 10 minutes
	LCALL	DRAIN(5)	Drain operation for 5 minutes
	LCALL	SPIN(5)	Spin for 10 minutes
	SJMP	DISPLAY	Jump to DISPLAY

PROGRAM 14.26(d)

<i>Labels</i>	<i>Mnemonics</i>	<i>Operands</i>	<i>Comments</i>
DELICATE	LCALL	AGITATE(5)	Agitate for 15 minutes
	LCALL	DRAIN(5)	Drain operation for 5 minutes
	LCALL	FILL_2	Machine fill with water second time
	LCALL	AGITATE(5)	Agitate for 10 minutes
	LCALL	DRAIN(5)	Drain operation for 5 minutes
	LCALL	SPIN(5)	Spin for 10 minutes
	SJMP	DISPLAY	Jump to DISPLAY

SUMMARY

- In this chapter, all types of addressing modes of the 8051 microcontroller such as direct addressing, register addressing, register indirect addressing, immediate addressing, index addressing are discussed with examples.
- The classification of instruction set of 8051 microcontroller has been explained and detailed operation of each instruction has been described with examples.
- Assembly-language programs such as addition, subtraction, multiplication, division, 1's complement, 2's complement, largest and smallest value of an array, descending order, ascending order of an array are given in this chapter.
- Applications of microcontrollers in stepper-motor control, traffic-light control, display, A/D converter interfacing, keyboard interface, and washing-machine control are also incorporated in this chapter.

MULTIPLE-CHOICE QUESTIONS

- 14.1 What is the addressing mode of MOV A, 40?
- (a) Direct addressing
 (b) Indirect addressing
 (c) Index addressing
 (d) Register addressing
- 14.2 Which instruction does not belong to register addressing mode?
- (a) MOV A,R7 (b) MOV R0,R1
 (c) MOV A,@R3 (d) MOV R5,A
- 14.3 Which of the following instructions is index addressing?
- (a) MOVC A,@A+DPTR
 (b) MOVX @DPTR,A
 (c) MOVX A,@DPTR
 (d) MOVX A,@R0
- 14.4 The MOVX A, @R0 instruction performs
- (a) data transfer from external RAM 8-bit address specified by R0 to accumulator
 (b) data transfer from internal RAM 8-bit address specified by R0 to accumulator
 (c) data transfer from external ROM 8-bit address specified by R0 to accumulator
 (d) data transfer from internal ROM 8-bit address specified by R0 to accumulator
- 14.5 Which of the following instructions is incorrect?
- (a) CPL A (b) SWAP A
 (c) CLR C (d) RL B
- 14.6 Which of the following flags are affected by the instruction INC A and INC @R0?
- (a) Carry flag (b) Auxiliary carry flag
 (c) Overflow flag (d) No flags are affected
- 14.7 What will be the output after execution of the following instructions?
- MOV A, #55
 ANL A, #67
- (a) 54 (b) 45
 (c) 55 (d) 67
- 14.8 To exchange the content of A and R0 which instruction is used?
- (a) XCH A, R0 (b) XCH A, @R0
 (c) XCHD A, @R0 (d) XCH R0, A
- 14.9 Which of the following instructions is not a logical instruction?
- (a) ANL A, #FF (b) CPL A
 (c) INC A (d) SWAP A
- 14.10 Which of the following instructions is not an arithmetic instruction?

- | | | | |
|------------|-----------------|-----------|-------------------|
| (a) MUL AB | (b) ADD A, #66H | (a) RLC A | (b) SWAP B |
| (c) DIV AB | (d) CPL A | (c) CPL C | (d) MOVC A, @A+PC |

- 14.11 Which of the following instructions do not perform the increment of the content of memory location 50H by 1?
 (a) INC 50
 (b) MOV R0, #50; INC @R0
 (c) MOV A, #50; INC A
 (d) MOV R0, #50, INC R0
- 14.12 Which of the following instructions are used to swap nibbles inside the accumulator?
 (a) SWAP A
 (b) RR A; RR A
 (c) RR A; RR A; RR A;
 (d) RRC A RRC A; RRC A; RRC A;
- 14.13 Which of the following instructions is incorrect?
 (a) 62 (b) 26
 (c) 66 (d) 22
- 14.14 Which of the following instructions is indirect addressing?
 (a) MOV A, R0 (b) MOV A, 40H
 (c) MOV R7, #55 (d) MOV A,@R0
- 14.15 The operation "end the content of RAM whose address is specified by R3 to port3" is performed by
 (a) MOV P3, @R3 (b) MOV P3, R3
 (c) MOV @R3, P3 (d) MOV R3, P3
- 14.16 What will be the contents of A register after execution of instruction RRC A. Assume the contents of A before execution is C5H and carry is zero.
 (a) 62 (b) 26
 (c) 66 (d) 22

SHORT-ANSWER-TYPE QUESTIONS

- 14.1 What is the difference between CY and OV flags ?
- 14.2 What is the addressing mode of MOV A, @Ri instruction?
- 14.3 What will be the content of the accumulator after execution of the following instructions?
 MOV A, #FFH ADD A, #23H
- 14.4 What is the difference between the following instructions?
 (a) LJMP and SJMP (ii) RET and RETI (iii) MOV and MOVX
- 14.5 Which addressing mode is suitable for look-up table access?
- 14.6 What are the instructions to access the program memory?

REVIEW QUESTIONS

- 14.1 What are the addressing modes of 8051 microcontroller? Explain each addressing mode with an example.
- 14.2 Write the addressing modes of the following instructions:
 (i) MOV A, @R0 (ii) MOVX @DPTR, A (iii) MOV A, @A+DPTR (iv) MOA R0,#45H
- 14.3 State different types of instructions of 8051 and explain any three instructions from each group of the instructions.
- 14.4 Write instructions to perform the following operations:
 (i) Move the content of accumulator to register 7(R7).
 (ii) Move the contents of RAM memory location 55H to Port 1
 (iii) Send 22H to Port 0

- (iv) Move the value at Port 2 to Register 1(R1)
 - (v) Clear bit 7 of the accumulator.
- 14.5 Write the following programs in assembly language.
- (i) Add two 8-Bit numbers
 - (ii) Add two 16-bit numbers
 - (iii) Add a series of 8-bit numbers
 - (iv) Subtract two 8-Bit numbers
 - (v) Two's complement of an 8-bit number
 - (vi) Find the largest number in a data array
 - (viii) Find smallest number in a data array
 - (ix) Perform division of two numbers
 - (x) Perform multiplication two numbers
 - (xi) SWAP 4 MSBs with 4 LSBs in the accumulator
 - (xii) Arrange a series of numbers in descending order
 - (xiii) Compare two 8-bit numbers
 - (xiv) Arrange a data array in ascending order
- 14.6 Explain a microcontroller-based traffic-light control system with assembly-language program.
- 14.7 Explain the 8051 microcontroller-based position control system using a stepper motor.
- 14.8 Draw a circuit diagram for keyboard interface with 8051 microcontroller and write a program for reading any key.
- 14.9 Write a program for A/D converter interface with the 8051 microcontroller.
- 14.10 N 8-bit numbers whose sum is 8bits.
- 14.11 Add two 16-bit numbers whose Sum is 16 bits.
- 14.12 Find one's complement of a 16-bit number.
- 14.13 Shift an 8-bit number right by one bit and store in 50H memory location.
- 14.14 Find out the largest of two numbers and store in 9000H memory location.
- 14.15 Find out the smallest of two numbers.
- 14.16 Write a program for logical OR of two 8-bit data.
- 14.17 Write a program to display '8051 microcontroller' on the screen.

Answers to Multiple-Choice Questions

-
- 14.1 (a) 14.2 (c) 14.3 (a) 14.4 (a) 14.5 (d) 14.6 (d) 14.7 (b) 14.8 (a)
 - 14.9 (c) 14.10 (d) 14.11 (c)&(d) 14.12 (a) 14.13 (b) 14.14 (d) 14.15 (a) 14.16 (a)

Appendix A

OPCODE of the 8085 Instruction Set

<i>Mnemonic</i>	<i>Hex Code</i>	<i>Mnemonic</i>	<i>Hex Code</i>	<i>Mnemonic</i>	<i>Hex Code</i>
MOV A,A	7F	MOV H,B	60	POP B	C1
MOV A,B	78	MOV H,C	61	POP D	D1
MOV A,C	79	MOV H,D	62	POP H	E1
MOV A,D	7A	MOV H,E	63	POP PSW	F1
MOV A,E	7B	MOV H,H	64	PUSH B	C5
MOV A,H	7C	MOV H,L	65	PUSH D	D5
MOV A,L	7D	MOV H,M	66	PUSH H	E5
MOV A,M	7E	MOV L,A	6F	PUSH PSW	F5
MOV B,A	47	MOV L,B	68	PCHL	E9
MOV B,B	40	MOV L,C	69	RAL	17
MOV B,C	41	MOV L,D	6A	RAR	1F
MOV B,D	42	MOV L,E	6B	RC	D8
MOV B,E	43	MOV L,H	6C	RET	C9
MOV B,H	44	MOV L,L	6D	RIM	20
MOV B,L	45	MOV L,M	6E	RLC	07
MOV B,M	46	MOV M,A	77	RM	F8
MOV C,A	4F	MOV M,B	70	RNC	D0
MOV C,B	48	MOV M,C	71	RNZ	C0
MOV C,C	49	MOV M,D	72	RP	F0
MOV C,D	4A	MOV M,E	73	RPE	E8
MOV C,E	4B	MOV M,H	74	RPO	E0
MOV C,H	4C	MOV M,L	75	RRC	0F
MOV C,L	4D	MVI A,8-bit	3E	RST 0	C7
MOV C,M	4E	MVI B,8-bit	06	RST 1	CF
MOV D,A	57	MVI C,8-bit	0E	RST 2	D7
MOV D,B	50	MVI D,8-bit	16	RST 3	DF

(Contd.)

(Contd.)

<i>Mnemonic</i>		<i>Hex Code</i>	<i>Mnemonic</i>		<i>Hex Code</i>	<i>Mnemonic</i>		<i>Hex Code</i>
MOV	D,C	51	MVI	E,8-bit	1E	RST	4	E7
MOV	D,D	52	MVI	H,8-bit	26	RST	5	EF
MOV	D,E	53	MVI	L,8-bit	2E	RST	6	F7
MOV	D,H	54	MVI	M,8-bit	36	RST	7	FF
MOV	D,L	55	NOP		00	RZ		C8
MOV	D,M	56	ORA	A	B7	SBB	A	9F
MOV	E,A	5F	ORA	B	B0	SBB	B	98
MOV	E,B	58	ORA	C	B1	SBB	C	99
MOV	E,C	59	ORA	D	B2	SBB	D	9A
MOV	E,D	5A	ORA	E	B3	SBB	E	9B
MOV	E,E	5B	ORA	H	B4	SBB	H	9C
MOV	E,H	5C	ORA	L	B5	SBB	L	9D
MOV	E,L	5D	ORA	M	B6	SBB	M	9E
MOV	E,M	5E	ORI	8-bit	F6	SBI	8-Bit	DE
MOV	H,A	67	OUT	8-bit	D3	SHLD	16-Bit	22
SIM		30	ACI	8-Bit	CE	CP	16-Bit	F4
SPHL		F9	ADC	A	8F	CPE	16-Bit	EC
STA	16-Bit	32	ADC	B	88	CPI	8-Bit	FE
STAX	B	02	ADC	C	89	CPO	16-Bit	E4
STAX	D	12	ADC	D	8A	CZ	16-Bit	CC
STC		37	ADC	E	8B	DAA		27
SUB	A	97	ADC	H	8C	DAD	B	09
SUB	B	90	ADC	L	8D	DAD	D	19
SUB	C	91	ADC	M	8E	DAD	H	29
SUB	D	92	ADD	A	87	DAD	SP	39
SUB	E	93	ADD	B	80	DCR	A	3D
SUB	H	94	ADD	C	81	DCR	B	05
SUB	L	95	ADD	D	82	DCR	C	0D
SUB	M	96	ADD	E	83	DCR	D	15
SUI	8-Bit	D6	ADD	H	84	DCR	E	1D
XCHG		EB	ADD	L	85	DCR	H	25
XRA	A	AF	ADD	M	86	DCR	L	2D
XRA	B	A8	ADI	8-Bit	C6	DCR	M	35
XRA	C	A9	ANA	A	A7	DCX	B	0B
XRA	D	AA	ANA	B	A0	DCX	D	1B
XRA	E	AB	ANA	C	A1	DCX	H	2B
XRA	H	AC	ANA	D	A2	DCX	SP	3B

(Contd.)

(Contd.)

<i>Mnemonic</i>		<i>Hex Code</i>	<i>Mnemonic</i>		<i>Hex Code</i>	<i>Mnemonic</i>		<i>Hex Cod</i>
XRA	L	AD	ANA	E	A3	DI		F3
XRA	M	AE	ANA	H	A4	EI		FB
XRI	8-Bit	EE	ANA	L	A5	HLT		76
XTHL		E3	ANA	M	A6	IN	8-Bit	DB
			ANI	8-Bit	E6	INR	A	3C
			CALL	16-Bit	CD	INR	B	04
JNC	16-Bit	D2	CC	16-Bit	DC	INR	C	0C
JNZ	16-Bit.	C2	CM	16-Bit	FC	INR	D	14
JP	16-Bit	F2	CMA		2F	INR	E	1C
JPE	16-Bit	EA	CMC		3F	INR	H	24
JPO	16-Bit	E2	CMP	A	BF	INR	L	2C
JZ	16-Bit	CA	CMP	B	B8	INR	M	34
LDA	16-Bit	3A	CMP	C	B9	INX	B	03
LDAX	B	0A	CMP	D	BA	INX	D	13
LDAX	D	1A	CMP	E	BB	INX	H	23
LHLD	16-Bit	2A	CMP	H	BC	INX	SP	33
LXI	B, 16-Bit	01	CMP	L	BD	JC	16-Bit	DA
LXI	D, 16-Bii	11	CMP	M	BE	JM	16-Bit	FA
LXI	H, 16-Bit	21	CNC	16-Bit	D4	JMP	16-Bit	C3
LXI	SP. 16-Bit	31	CNZ	16-Bit	C4			

Appendix B

Some Important Tables for 8051

Table 1 8051 Data-transfer Instructions Set Summary

Opcode	Operand	Functions	Clock cycle	Number of bytes	Instruction code
MOV	A, Rn	Move register to accumulator	12	1	1 1 1 0 1 r r r
MOV	A, direct	Move direct byte to accumulator	12	2	1 1 1 0 0 1 0 1
MOV	A, @Ri	Move indirect RAM to accumulator	12	1	1 1 1 0 0 1 1 i
MOV	A, #data	Move immediate data to accumulator	12	2	0 1 1 1 0 1 0 0
MOV	Rn, A	Move accumulator to register	12	1	1 1 1 1 1 r r r
MOV	Rn, direct	Move direct byte to register	24	2	1 0 1 0 1 r r r
MOV	Rn, #data	Move immediate data to register	12	2	0 1 1 1 1 r r r
MOV	direct, A	Move accumulator to direct byte	12	2	1 1 1 1 0 1 0 1
MOV	direct, Rn	Move register to direct byte	24	2	1 0 0 0 1 r r r
MOV	direct, direct	Move direct byte to direct	24	3	1 0 0 0 0 1 0 1
MOV	direct, @Ri	Move indirect RAM to direct byte	24	2	1 0 0 0 0 1 1 i
MOV	direct, #data	Move immediate data to direct byte	24	3	0 1 1 1 0 1 0 1
MOV	@Ri, A	Move accumulator to indirect RAM	12	1	1 1 1 1 0 1 1 i
MOV	@Ri, direct	Move direct byte to indirect RAM	24	2	1 0 1 0 0 1 1 i
MOV	@Ri, #data	Move immediate data to indirect RAM	12	2	0 1 1 1 0 1 1 i
MOV	DPTR, #data16	Load data pointer with a 16-bit constant	24	3	1 0 0 1 0 0 0 0
MOVC	A,@A+DPTR	Move code byte relative to DPTR to ACC	24	1	1 0 0 1 0 0 1 1
MOVC	A,@A+PC	Move code byte relative to PC to ACC	24	1	1 0 0 0 0 0 1 1
MOVX	A, @Ri	Move external RAM (8-bit addr) to ACC	24	1	1 1 1 0 0 0 1 i
MOVX	A,@DPTR	Move external RAM (16-bit addr) to ACC	24	1	1 1 1 0 0 0 0 0
MOVX	A, @Ri	Move ACC to external RAM (8-bit addr)	24	1	1 1 1 1 0 0 1 i
MOVX	A,@DPTR	Move ACC to external RAM (16-bit addr)	24	1	1 1 1 1 0 0 0 0

Table 2 8051 Arithmetic Instruction Set Summary

Opcode	Operand	Functions	Clock cycle	Number of bytes	Instruction code
ADD	A, Rn	Add register to accumulator	12	1	0 0 1 0 1 r r r
ADD	A, direct	Add direct byte to accumulator	12	2	0 0 1 0 0 1 0 1
ADD	A, @Ri	Add indirect RAM to accumulator	12	1	0 0 1 0 0 1 1 i
ADD	A, #data	Add immediate data to accumulator	12	2	0 0 1 0 0 1 0 0
ADDC	A, Rn	Add register to accumulator with carry	12	1	0 0 1 1 1 r r r
ADDC	A, direct	Add direct byte to accumulator with carry	12	2	0 0 1 1 0 1 0 1
ADDC	A, @Ri	Add indirect RAM to Accumulator with carry	12	1	0 0 1 1 0 1 1 i
ADDC	A, #data	Add immediate data to ACC with carry	12	2	0 0 1 1 0 1 0 0
SUBB	A, Rn	Subtract register from ACC with borrow	12	1	1 0 0 1 1 r r r
SUBB	A, direct	Subtract direct byte from ACC with borrow	12	2	1 0 0 1 0 1 0 1
SUBB	A, @Ri	Subtract indirect RAM from ACC with borrow	12	1	1 0 0 1 0 1 1 i
SUBB	A, #data	Subtract immediate data from ACC with borrow	12	2	1 0 0 1 0 1 0 0
INC	A	Increment accumulator	12	1	0 0 0 0 0 1 0 0
INC	Rn	Increment register	12	1	0 0 0 0 1 r r r
INC	direct	Increment direct byte	12	2	0 0 0 0 0 1 0 1
INC	@Ri	Increment indirect RAM	12	1	0 0 0 0 0 1 1 i
INC	DPTR	Increment data pointer	12	1	0 0 0 0 0 0 1 1
DEC	A	Decrement accumulator	12	1	0 0 0 1 0 1 0 0
DEC	Rn	Decrement register	12	1	0 0 0 1 1 r r r
DEC	direct	Decrement direct byte	12	2	0 0 0 1 0 1 0 1
DEC	@Ri	Decrement indirect RAM	12	1	0 0 0 1 0 1 1 i
MUL	AB	Multiply A and B	48	1	1 0 1 0 0 1 0 0
DIV	AB	Divide A by B	48	1	1 0 0 0 0 1 0 0
DAA		Decimal adjust accumulator	12	1	1 1 0 1 0 1 0 0

Table 3 8051 Logical instruction set summary

Opcode	Operand	Functions	Clock cycle	Number of bytes	Instruction code
ANL	A, Rn	AND register to accumulator	12	1	0 1 0 1 1 r r r
ANL	A, direct	AND direct byte to accumulator	12	2	0 1 0 1 0 1 0 1
ANL	A,@Ri	AND indirect RAM to accumulator	12	1	0 1 0 1 0 1 1 i
ANL	A,#data	AND immediate data to accumulator	12	2	0 1 0 1 0 1 0 0
ANL	direct, A	AND accumulator to direct byte	12	2	0 1 0 1 0 0 1 0
ANL	direct, #data	AND immediate data to direct byte	24	3	0 1 0 1 0 0 1 1
ORL	A, Rn	OR register to accumulator	12	1	0 1 0 0 1 r r r
ORL	A, direct	OR direct byte to accumulator	12	2	0 1 0 0 0 1 0 1
ORL	A, @ Ri	OR indirect RAM to accumulator	12	1	0 1 0 0 0 1 1 i
ORL	A, #data	OR immediate data to accumulator	12	2	0 1 0 0 0 1 0 0
ORL	direct, A	OR accumulator to direct byte	12	2	0 1 0 0 0 0 1 0
ORL	direct, #data	OR immediate data to direct byte	24	3	0 1 0 0 0 0 1 1
XRL	A, Rn	Exclusive-OR register to accumulator	12	1	0 1 1 0 1 r r r
XRL	A, direct	Exclusive-OR direct byte to accumulator	12	2	0 1 1 0 0 1 0 1
XRL	A, @ Ri	Exclusive-OR indirect RAM to accumulator	12	1	0 1 1 0 0 1 1 i
XRL	A, # data	Exclusive-OR immediate data to accumulator	12	2	0 1 1 0 0 1 0 0
XRL	direct,A	Exclusive-OR accumulator to direct byte	12	2	0 1 1 0 0 0 1 0
XRL	direct, #data	Exclusive-OR immediate data to direct byte	24	3	0 1 1 0 0 0 1 1
CLR	A	Clear accumulator	12	1	1 1 1 0 0 1 0 0
CPL	A	Complement accumulator	12	1	1 1 1 1 0 1 0 0
RL	A	Rotate accumulator left	12	1	0 0 1 0 0 0 1 1
RLC	A	Rotate accumulator left through the carry	12	1	0 0 1 1 0 0 1 1
RR	A	Rotate accumulator right	12	1	0 0 0 0 0 0 1 1
RRC	A	Rotate accumulator right through the carry	12	1	0 0 0 1 0 0 1 1
SWAP	A	Swap nibbles within the accumulator	12	1	1 1 0 0 0 1 0 0

Table 4 8051 Boolean instruction set summary

Opcode	Operand	Functions	Clock cycle	Number of bytes	Instruction code
CLR	C	Clear carry	12	1	1 1 0 0 0 0 1 1
CLR	bit	Clear direct bit	12	2	1 1 0 0 0 0 1 0
SETB	C	Set carry	12	1	1 1 0 1 0 0 1 1
SETB	bit	Set direct bit	12	2	1 1 0 1 0 0 1 0
CPL	C	Complement carry	12	1	1 0 1 1 0 0 1 1
CPL	bit	Complement direct bit	12	2	1 0 1 1 0 0 1 0
ANL	C, bit	AND direct bit to carry	24	2	1 0 0 0 0 0 1 0
ANL	C,/bit	AND complement of direct bit to carry	24	2	1 0 1 1 0 0 0 0
ORL	C, bit	OR direct bit to carry	24	2	0 1 1 1 0 0 1 0
ORL	C,/bit	OR complement of direct bit to carry	24	2	1 0 1 0 0 0 0 0
MOV	C,bit	Move direct bit to carry	12	2	1 0 1 0 0 0 1 0
MOV	bit,C	Move carry to direct bit	24	2	1 0 0 1 0 0 1 0
JC	rel	Jump if carry is set	24	2	0 1 0 0 0 0 0 0
JNC	rel	Jump if carry is not set	24	2	0 1 0 1 0 0 0 0
JB	Bit, rel	Jump if direct bit is set	24	3	0 0 1 0 0 0 0 0
JNB	Bit,rel	Jump if direct bit is not set	24	3	0 0 1 1 0 0 0 0
JBC	bit,rel	Jump if direct bit is set and clear bit	24	3	0 0 0 1 0 0 0 0

Table 5 8051 Branch group instructions set summary

Opcode	Operand	Functions	Clock cycle	Number of bytes	Instruction code
ACALL	addr11	Absolute subroutine call	24	2	a10 a9 1 0 0 0 1
LCALL	addr16	Long subroutine call	24	3	0 0 0 1 0 0 1 0
RET		Return from subroutine	24	1	0 0 1 0 0 0 1 0
RETI		Return from interrupt	24	1	0 0 1 1 0 0 1 0
AJMP	addr11	Absolute jump	24	2	a10 a9 1 a8 0 0 0 1
LJMP	addr16	Long jump	24	3	0 0 0 0 0 0 1 0
SJMP	rel	Short jump (relative addr)	24	2	1 0 0 0 0 0 0 0
JMP	@A+ DPTR	Jump indirect relative to the DPTR	24	1	0 1 1 1 0 0 1 1
JZ	rel	Jump if accumulator is zero	24	2	0 1 1 0 0 0 0 0
JNZ	rel	Jump if accumulator is not zero	24	2	0 1 1 1 0 0 0 0
CJNE	A, direct, rel	Compare direct byte to ACC and jump if not equal	24	3	1 0 1 1 0 1 0 1
CJNE	A, #data, rel	Compare immediate to ACC 24 and jump if not equal	3	1	0 1 1 0 1 0 0

Contd.

CJNE	Rn, #data, rel	Compare immediate to register and jump if not equal	24	3	1	0	1	1	1	r	r	r
CJNE	@Ri, #data, rel	Compare immediate to indirect and jump if not equal	24	3	1	0	1	1	0	1	1	i
DJNZ	Rn, rel	Decrement register and jump if not zero	24	2	1	1	0	1	1	r	r	r
DJNZ	direct, rel	Decrement direct byte and jump if not zero	24	3	1	1	0	1	0	1	0	1
NOP		No operation	12	1	0	0	0	0	0	0	0	0

Table 6 8051 12 PUSH, POP and Exchange instructions set summary

<i>Opcode</i>	<i>Operand</i>	<i>Functions</i>	<i>Clock cycle</i>	<i>Number of bytes</i>	<i>Instruction code</i>							
PUSH	direct	Push direct byte onto stack	24	2	1	1	0	0	0	0	0	0
POP	direct	Pop direct byte from stack	24	2	1	1	0	1	0	0	0	0
XCH	A, Rn	Exchange register with accumulator	12	1	1	1	0	0	1	r	r	r
XCH	A, direct	Exchange direct byte with accumulator	12	2	1	1	0	0	0	1	0	1
XCH	A, @ Ri	Exchange indirect RAM with accumulator	12	1	1	1	0	0	0	1	1	i
XCHD	A, @ Ri	Exchange low-order digit indirect RAM with ACC	12	1	1	1	0	1	0	1	1	i

Appendix C

Some Important Tables for 8085

Table 1 8085 Data-Transfer Instructions Set Summary

Opcode	Operand	Functions	Clock cycle	Number of bytes	Instruction code
MOV	Rd, Rs	Move register to register	4	1	0 1 D D D S S S
MOV	M, Rs	Move register to memory	7	1	0 1 1 1 0 S S S
MOV	Rd, M	Move memory to register	7	1	0 1 D D 1 1 0
MVI	Rd, data	Move immediate register	7	2	0 0 D D D 1 1 0
MVI	M, data	Move immediate memory	10	2	0 0 1 1 0 1 1 0
LDA	16 bit address	Load A direct	13	3	0 0 1 1 1 0 1 0
LDAX	B	Load A indirect	7	1	0 0 0 0 1 0 1 0
LDAX	D	Load A direct	7	1	0 0 0 1 1 0 1 0
LXI	B	Load immediate register pair B and C	10	3	0 0 0 0 0 0 0 1
LXI	D	Load immediate register pair D and E	10	3	0 0 0 1 0 0 0 1
LXI	H	Load immediate register pair H and L	10	3	0 0 1 0 0 0 0 1
LXI	SP	Load immediate stack pointer	10	3	0 0 1 1 0 0 0 1
LHLD	16 bit address	Load H & L direct	16	3	0 0 1 0 1 0 1 0
STA	16 bit address	Store A direct	13	3	0 0 1 1 0 0 1 0
STAX	B	Store A indirect	7	1	0 0 0 0 0 0 1 0
STAX	D	Store A indirect	7	1	0 0 0 1 0 0 1 0
SHLD		Store H & L direct	16	3	0 0 1 0 0 0 1 0
XCHG		Exchange D & E and H & L registers	4	1	1 1 1 0 1 0 1 1

Table 2 8085 Arithmetic Instruction Set Summary

Opcode	Operand	Functions	Clock cycle	Number of bytes	Instruction code
ADD	R	Add register to A	4	1	1 0 0 0 0 0 S S S
ADD	M	Add memory to A	7	1	1 0 0 0 0 1 1 0
ADC	R	Add register to A with carry	4	1	1 0 0 0 1 S S S
ADC	M	Add memory to A with carry	7	1	1 0 0 0 1 1 1 0
ADI	8-bit data	Add immediate to A	7	2	1 1 0 0 0 1 1 0
ACI	8-bit data	Add immediate to A with carry	7	2	1 1 0 0 1 1 1 0
DAD	B	Add B & C to H & L	10	1	0 0 0 0 1 0 0 1

DAD	D	Add D & E to H & L	10	1	0	0	0	1	1	0	0	1
DAD	H	Add H & L to H & L	10	1	0	0	1	0	1	0	0	1
DAD	SP	Add stack pointer to H & L	10	1	0	0	1	1	1	0	0	1
SUB	R	Subtract register from A	4	1	1	0	0	1	0	S	S	S
SUB	M	Subtract memory from A	7	1	1	0	0	1	0	1	1	0
SBB	R	Subtract register from A with borrow	4	1	1	0	0	1	1	S	S	S
SBB	M	Subtract memory from A with borrow	7	1	1	0	0	1	1	1	1	0
SUI	8-bit data	Subtract immediate from A	7	2	1	1	0	1	0	1	1	0
SBI	8-bit data	Subtract immediate from A with borrow	7	2	1	1	0	1	1	1	1	0
INR	R	Increment register	4	1	0	0	D	D	D	1	0	0
INR	M	Increment memory	10	1	0	0	1	1	0	1	0	0
INX	B	Increment B & C registers	6	1	0	0	0	0	0	0	1	1
INX	D	Increment D & E registers	6	1	0	0	0	1	0	0	1	1
INX	H	Increment H & L registers	6	1	0	0	1	0	0	0	1	1
INX	SP	Increment stack pointer	6	1	0	0	1	1	0	0	1	1
DCR	R	Decrement register	4	1	0	0	D	D	D	1	0	1
DCR	M	Decrement memory	10	1	0	0	1	1	0	1	0	1
DCX	B	Decrement B & C registers	6	1	0	0	0	0	1	0	1	1
DCX	D	Decrement D & E registers	6	1	0	0	0	1	1	0	1	1
DCX	H	Decrement H & L registers	6	1	0	0	1	0	1	0	1	1
DCX	SP	Decrement stack pointer	6	1	0	0	1	1	1	0	1	1
DAA		Decimal adjustment	4	1	0	0	1	0	0	1	1	1

Table 3 8085 Logical instruction set summary

Opcode	Operand	Functions	Clock cycle	Number of bytes	Instruction code
CMP	R	Compare register with A	4	1	1 0 1 1 1 S S S
CMP	M	Compare memory with A	7	1	1 0 0 0 0 1 1 0
CPI	8 bit data	Compare immediate with A	7	2	1 1 1 1 1 1 1 0
ANA	R	AND register with A	4	1	1 0 1 0 0 S S S
ANA	M	AND memory with A	7	1	1 0 1 0 0 1 1 0
ANI	8 bit data	AND immediate with A	7	2	1 1 1 0 0 1 1 0
ORA	R	OR register with A	4	1	1 0 1 1 0 S S S
ORA	M	OR memory with A	7	1	1 0 1 1 0 1 1 0
ORI	8 bit data	OR immediate with A	7	2	1 1 1 1 0 1 1 0
XRA	R	Exclusive OR register with A	4	1	1 0 1 0 1 S S S
XRA	M	Exclusive OR memory with A	7	1	1 0 1 0 1 1 1 0
XRI	8 bit data	Exclusive OR immediate with A	7	2	1 1 1 0 1 1 1 0
RLC		Rotate A left	4	1	0 0 0 0 0 1 1 1
RRC		Rotate A right	4	1	0 0 0 0 1 1 1 1
RAL		Rotate A left with carry	4	1	0 0 0 1 0 1 1 1

RAR	Rotate A right with carry	4	1	0	0	0	1	1	1	1	1	1
CMA	Complement A	4	1	0	0	1	0	1	1	1	1	1
CMC	Complement carry	4	1	0	0	1	1	1	1	1	1	1
STC	Set carry	4	1	0	0	1	1	0	1	1	1	1

Table 4 8085 JUMP Instruction Set Summary

Opcode	Operand	Functions	Clock cycle	Number of bytes	Instruction code
JMP	16-bit address	Jump unconditional	10	3	1 1 0 0 0 0 0 1 1
JC	16-bit address	Jump on carry	7/10	3	1 1 0 1 1 0 1 0
JNC	16-bit address	Jump on no carry	7/10	3	1 1 0 1 0 0 1 0
JP	16-bit address	Jump on positive	7/10	3	1 1 1 1 0 0 1 0
JM	16-bit address	Jump on minus	7/10	3	1 1 1 1 1 0 1 0
JZ	16-bit address	Jump on zero	7/10	3	1 1 0 0 1 0 1 0
JNZ	16-bit address	Jump on no zero	7/10	3	1 1 0 0 0 0 1 0
JPE	16-bit address	Jump on parity even	7/10	3	1 1 1 0 1 0 1 0
JPO	16-bit address	Jump on parity odd	7/10	3	1 1 1 0 0 0 1 0

Table 5 8085 CALL and Return Instruction Set Summary

Opcode	Operand	Functions	Clock cycle	Number of bytes	Instruction code
CALL	16-bit address	Call unconditional	18	3	1 1 0 0 1 1 0 1
CC	16-bit address	Call on carry	9/18	3	1 1 0 1 1 1 0 0
CNC	16-bit address	Call on no carry	9/18	3	1 1 0 1 0 1 0 0
CP	16-bit address	Call on positive	9/18	3	1 1 1 1 0 1 0 0
CM	16-bit address	Call on minus	9/18	3	1 1 1 1 1 1 0 0
CZ	16-bit address	Call on zero	9/18	3	1 1 0 0 1 1 0 0
CNZ	16-bit address	Call on no zero	9/18	3	1 1 0 0 0 1 0 0
CPE	16-bit address	Call on parity even	9/18	3	1 1 1 0 1 1 0 0
CPO	16-bit address	Call on parity odd	9/18	3	1 1 1 0 0 1 0 0
RET		Return unconditional	10	1	1 1 0 0 1 0 0 1
RC		Return on Carry	6/12	1	1 1 0 1 1 0 0 0
RNC		Return on no Carry	6/12	1	1 1 0 1 0 0 0 0
RP		Return on positive	6/12	1	1 1 1 1 0 0 0 0
RM		Return on minus	6/12	1	1 1 1 1 1 0 0 0
RZ		Return on zero	6/12	1	1 1 0 0 1 0 0 0
RNZ		Return on no zero	6/12	1	1 1 0 0 0 0 0 0
RPE		Return on parity even	6/12	1	1 1 1 0 1 0 0 0
RPO		Return on parity odd	6/12	1	1 1 1 0 0 0 0 0

Table 6 8085 Stack/PUSH and POP Instructions Set Summary

Opcode	Operand	Functions	Clock cycle	Number of bytes	Instruction code
PUSH	B	Push register pair Band C on stack	12	1	1 1 0 0 0 1 0 1
PUSH	D	Push register pair D and E on stack	12	1	1 1 0 1 0 1 0 1
PUSH	H	Push register pair H and L on stack	12	1	1 1 1 0 0 1 0 1
PUSH	PSW	Push accumulator A and Flags on stack	12	1	1 1 1 1 0 1 0 1
POP	B	Pop register pair Band C off stack	10	1	1 1 0 0 0 0 0 1
POP	D	Pop register pair D and E off stack	10	1	1 1 0 1 0 0 0 1
POP	H	Pop register pair H and L off stack	10	1	1 1 1 0 0 0 0 1
POP	PSW	Pop accumulator A and Flags off stack	10	1	1 1 1 1 0 0 0 1
XTHL		Exchange top of stack H and L	16	1	1 1 1 0 0 0 1 1
SPLH		H and L to stack pointer	6	1	1 1 1 1 1 0 0 1

Table 7 8085 I/O and Machine Control Instructions Set Summary

Opcode	Operand	Functions	Clock cycle	Number of bytes	Instruction code
EI		Enable interrupts	4	1	1 1 1 1 1 0 1 1
DI		Disable interrupts	4	1	1 1 1 1 1 0 0 1
NOP		No-operation	4	1	0 0 0 0 0 0 0 0
HLT		Halt(Power Down)	5	1	0 1 1 1 0 1 1 0
RIM		Read interrupt mask	4	1	0 0 1 0 0 0 0 0
SIM		Set interrupt musk	4	1	0 0 1 1 0 0 0 0
IN		Input	10	1	1 1 0 1 1 0 1 1
OUT		output	10	1	1 1 0 1 0 0 1 1

Appendix D

Some Important Tables for 8086

Table 1a 8085 Data-transfer Instructions Set Summary

<i>Opcode</i>	<i>Operand</i>	<i>Mnemonics</i>	<i>Symbolic Operation</i>	<i>Comments</i>
MOV	destination, source	MOV AX, BX	AX ← BX	Register to register
		MOV AL, BL	AL ← BL	Register to register
		MOV AX, MEMW	AL ← [0100H]; AH ← [0101H]	Memory to register
		MOV AL, MEMB	AL ← [0100H]	Memory to register
		MOV MEMW, AX	[0100H] ← AL; [0101H] ← AH	Register to memory
		MOV MEMB, BL	[0100H] ← BL	Register to memory
		MOV MEMW, 2244H	[0100H] ← 44H; [0101H] ← 22H	Immediate data to memory
		MOV MEMB, 44H	[0100H] ← 44H	Immediate data to memory
		MOV AL, 22H	AL ← 22H	Immediate data to register
		MOV AX, 2000H	AL ← 00H; AH ← 20H	Immediate data to register
		MOV DS, AX	DS ← AX	General register to segment register
		MOV DX, ES	DX ← ES	Segment register to general register
MOV ES, MEMW	ES ← [0101H : 0100H]	Memory to segment register		
MOV MEMW, CS	[0101H : 0100H] ← CS	Segment register to memory		
XCHG	destination, source	XCHG AX, BX	AX ↔ BX	Exchange the contents of the word or byte source operand with the destination operand; none of the flags are affected.
		XCHG AL, BL	AL ↔ BL	
		XCHG [SI], BX	[SI] ↔ BL; [SI+1] ↔ BH	
LAHF		LAHF	AH ← Flags _L	Copy the low order flag byte into AH
SAHF		SAHF	Flags _L ← AH	Copy AH into the low order flag byte
IN	Accumulator, port	IN AL, 01H	AL ← Port 01H	Input a byte or word from direct I/O ports 00H to FFH.
		IN AX, 02H	AL ← Port 02H; AH ← 03H	
		IN AL, DX	AL ← Port DX	Input a byte or word from indirect I/O ports 0000H
		IN AX, DX	AL ← Port DX;	

Contd.

OUT	Port, accumulator	OUT 01H, AL OUT 02H, AX OUT DX, AL OUT DX, AX	AH ← Port DX + 1 Port 01H ← AL Port 02H ← AL; Port 03H ← AH Port DX ← AL Port DX ← AL; Port DX+1 ← AH	to FFFFH; the port address is in DX; None of the flags are affected Output a byte or word to direct I/O ports 00H to FFH Output a byte or word to indirect I/O ports 0000H to FFFFH; the port address is in DX; The flags are not affected.
-----	-------------------	--	---	---

Table 1b 8086 Data Transfer Instruction Set Summary

Opcode	Operand	Mnemonics	Symbolic Operation	Comments
LEA	destination, source	LEA DX, MEMB	BL ← 00; BH ← 01H	The effective address of the source operand is transferred to the destination operand; the flags are not affected
LDS	destination, source	LDS BX, DWORD PTR[SI]	BL ← [SI]; BH[SI+1]; DS ← [SI+3:SI+2]	Transferred 32-bit pointer variable from the source operand in memory to the destination register and register DS or ES; none of the flags are affected
LES	destination, source	LES BX, DWORD PTR[SI]	BL ← [SI]; BH ← [SI+1]; ES ← [SI+3:SI+2]	
XLAT		XLAT	AL ← [BX+AL]	Replace the byte in AL with the byte from the 256 byte lookup table beginning at [BX]; AL is used as an offset into this table; The flags are not affected

MEMB = 0100 is used to locate a byte in data segment, MEMW = 0100 is used to locate a word in data segment

Table 2 8086 PUSH and POP Instructions Set Summary

Opcode	Operand	Mnemonics	Symbolic Operation	Comments
PUSH	Source	PUSH BX	SP ← SP - 2; [SP + 1] ← BH; [SP] ← BL	Decrement SP by 2 and transfer the word from the source operand to the top of the stack pointed by SP and SS
		PUSH DS	SP ← SP - 2; [SP + 1: SP] ← DS;	
		PUSH [DI + 5]	SP ← SP - 2; [SP + 1] ← [DI + 6]; [SP] ← [DI + 5]	
POP	Destination	POP BX	BL ← [SP]; BH ← [SP + 1]; SP ← SP + 2	Increment SP by 2 and transfer the word from the top of the stack pointed by SP and SS to the destination
POP DS	DS ← [SP + 1: SP]; SP ← SP + 2			

Contd.

		POP [DI + 5]	$[DI + 6] \leftarrow [SP + 1];$ $[DI + 5] \leftarrow [SP];$ $SP \leftarrow SP + 2$	tion operand
PUSHF	None	PUSHF	$SP \leftarrow SP - 2;$ $[SP + 1: SP] \leftarrow \text{Flags};$	Push the 16-bit flag word onto the top of stack
POPF	None	POPF	$\text{Flags} \leftarrow [SP + 1: SP];$ $SP \leftarrow SP + 2$	Pop the top of the stack into the 16-bit flag word

Table 3 8086 Arithmetic Instructions Set Summary

<i>Opcode</i>	<i>Operand</i>	<i>Mnemonics</i>	<i>Symbolic Operation</i>	<i>Comments</i>
ADD	destination, source	ADD SI, AX ADD [BX], CL ADD DI, 4000H ADD MEMW, 4000H	$SI \leftarrow SI + AX$ $[BX] \leftarrow [BX] + CH$ $DI \leftarrow DI + 4000H$ $[0101H:0100H] \leftarrow [0101H:0100H] + 4000H$	Substitute the destination byte or word with the sum of the source and destination operands; all flags are updated
ADC	destination, source	ADC SI, AX ADC [BX], CL ADC DI, 4000H ADC MEMW, 4000H	$SI \leftarrow SI + AX + CF$ $[BX] \leftarrow [BX] + CL + CF$ $DI \leftarrow DI + 4000H + CF$ $[0101H:0100H] \leftarrow [0101H:0100H] + 4000H + CF$	Replace the destination byte or word with the sum of the source and destination operands plus the carry; all flags are updated
SUB	destination, source	SUB SI, AX SUB [BX], CL SUB DI, 4000H SUB MEMW, 8000H	$SI \leftarrow SI - AX$ $[BX] \leftarrow [BX] - CL$ $DI \leftarrow DI - 4000H$ $[0101H:0100H] \leftarrow [0101H:0100H] - 4000H$	Substitute the destination byte or word with the difference between of destination operands and source operand; all flags are updated
SBB	destination, source	SBB SI, AX SBB [BX], CL SBB DI, 4000H SBB MEMW, 8000H	$SI \leftarrow SI - AX - CF$ $[BX] \leftarrow [BX] - CL - CF$ $DI \leftarrow DI - 4000H - CF$ $[0101H:0100H] \leftarrow [0101H:0100H] - 4000H - CF$	Replace the destination byte or word with the difference between of destination operands and source operand plus the carry; all flags are updated
INC	destination	INC CL INC WORD [DI] INC MEMBS	$CL \leftarrow CL + 1$ $[DI + 1:DI] \leftarrow [DI + 1:DI] + 1$ $[0100H] \leftarrow [0100H] + 1$	Increment by one or Add one the byte or word destination operand; store the result in the destination operand; all flags except CF are updated.
DEC	destination	DEC CL DEC WORD [DI] DEC MEMB	$CL \leftarrow CL - 1$ $[DI + 1:DI] \leftarrow [DI + 1:DI] - 1$ $[0100H] \leftarrow [0100H] - 1$	Subtract one from byte or word destination operand; store the result in the destination operand; all flags except CF are updated.
NEG	destination	NEG AL NEG WORD [DI] NEG MEMB	$AL \leftarrow 0 - AL$ $[DI + 1:DI] \leftarrow 0 - [DI + 1:DI]$ $[0100H] \leftarrow 0 - [0100H]$	Find the 2's complement of the byte or word destination operand; all flags except CF are updated.

Contd.

CMP	destination	CMP AL, BL CMP [DI],BX CMP MEMW, 4000H CMP DI,4000H	AL-BL; update flags [DI+1:DI]-BX; update flags [0101H:0100H]-4000H; update flags DI-4000H; update flags	Subtract the byte or word source operand from the similar destination operand; the operands remain unchanged; all flags are updated.
-----	-------------	--	---	--

Table 4 8086 Multiplication and Division Instructions Set Summary

<i>Opcode</i>	<i>Operand</i>	<i>Mnemonics</i>	<i>Symbolic Operation</i>	<i>Comments</i>
MUL	source	MUL BL MUL BX MUL [BX] MUL MEMW	AX ← AL × BL (Unsigned multiplication) DX: AX ← AX × BX (Unsigned multiplication) AX ← AL × [BX] (Unsigned multiplication) DX:AX ← AX × [0101H:0100H] (Unsigned multiplication)	Unsigned multiplication of the source operand byte or word and the accumulator; results are stored in AX; Double word results are stored in DX: AX, if the result cannot be stored in a single word CF and OF are set; all other flag are undefined
IMUL	source	IMUL BL IMUL BX IMUL [BX] IMUL MEMW	AX ← AL × BL (signed multiplication) DX: AX ← AX × BX (signed multiplication) AX ← AL × [BX] (signed multiplication) DX:AX ← AX × [0101H:0100H] (signed multiplication)	Its operation is same as MUL. The source operand is limited to -128 to +127 for byte multiplication and -32768 to +32767 for word multiplication. The CF and OF are set if the result cannot be represented in the low order register; then the sign bit is extended to the high order register and the other flags are undefined
DIV	source	DIV BL DIV BX DIV [BX] DIVL MEMW	AX ← AL / BL (Unsigned division) DX: AX ← AX / BX (Unsigned division) AX ← AL / [BX] (Unsigned division) DX:AX ← AX / [0101H:0100H] (Unsigned division)	Unsigned division of the accumulator and the source operand byte or word; the result is stored in AL and the remainder is stored in AH; for word divisors the result is stored in AX with remainder in DX; when the quotient exceeds the capacity of its destination register (AL or AX), a type 0 interrupt is generated; and all flags are not affected.

Contd.

IDIV	source	DIV BL DIV BX DIV [BX] DIVL MEMW	AX ← AL / BL (signed division) DX: AX ← AX / BX (signed division) AX ← AL / [BX] (signed division) DX:AX ← AX / [0101H:0100H] (signed division)	Its operation is same as DIV; the source operand is limited to -128 to +127 for byte division and -32768 to +32767 for word division
------	--------	---	--	--

Table 5 8086 Arithmetic Adjust Instructions Set Summary

Opcode	Operand	Mnemonics	Symbolic Operation	Comments
DAA	none	DAA	If AL.0F > 09 or AF = 1, then AL ← AL + 6; AF ← 1 If AL.F0 > 90 or CF = 1, then AL ← AL + 60H; CF ← 1	Adjust the content of AL to a pair of valid packed decimal digits though the addition of two valid packed or unpacked decimal operands; all flags except of are affected
DAS	none	DAS	If AL . 0F > 9 or AF = 1, then AL ← AL - 6; AF ← 1 If AL. F0 > 90 or CF = 1, then AL ← AL - 60H; CF ← 1	Adjust the content of AL to a pair of valid packed decimal digits after the subtraction of two valid packed or unpacked decimal operands; all flags except OF are affected
AAA	none	AAA	If AL. 0F > 9 or AF = 1, then AL ← AL + 6; AF ← AH + 1 AF ← 1; CF ← AF; AL ← AL. 0F	Adjust the content of AL to a single unpacked decimal number following the addition of two valid unpacked decimal operands. The high order half-byte of AL is zeroed and AH is incremented by 1; all flags except AF and CF are not affected
AAS	none	AAS	If AL . 0F > 9 or AF = 1, then AL ← AL - 6; AF ← AH - 1 AF ← 1; CF ← AF; AL ← AL . 0F	Adjust the content of AL to a single unpacked decimal number following the subtraction of two valid unpacked decimal operands. The high order half-byte of AL is zeroed and AH is decremented by 1; all flags except AF and CF are not affected

Contd.

AAM	None	AAM	$AH \leftarrow AL/0AH$ $AL \leftarrow \text{Remainder}$	After the multiplication of two valid unpacked decimal operands, AAM converts the result in AL to two valid unpacked decimal digits in AH and AL. PF, SF, and ZF are affected
AAD	None	AAD	$AL \leftarrow (AH \times 0AH) + AL$ $AL \leftarrow 0$	Before dividing AX by a single-digit unpacked, decimal operand, AAD converts the two-digit unpacked decimal number in AX to a binary number in AL and 0 in AH. The quotient will be a valid unpacked decimal number in AL and remainder in AH. PF, SF, and ZF flags are affected
CBW	None	CBW	If $AL > 80H$, then $AH \leftarrow 0$ If $AL < 7FH$, then $AH \leftarrow FFH$	Before dividing AX by a byte operand, CBW extends the sign of a byte dividend in AL into AH, thus converting AL into a valid signed word in AX; flags are not affected
CWD	None	CWD	If $AX < 8000H$, then $DX \leftarrow 0$ If $AX > 7FFFH$, then $DX \leftarrow FFFFH$	It works as CBW but extends the sign of a word dividend in AX into double word in DX:AX; flags are not affected

Table 6 8086 Logical Instructions Set Summary

<i>Opcode</i>	<i>Operand</i>	<i>Mnemonics</i>	<i>Symbolic Operation</i>	<i>Comments</i>
NOT	Destination	NOT AX NOT [SI]	$AX \leftarrow \overline{AX}$ $[SI] \leftarrow \overline{[SI]}$	Complement all bits of the byte or word operand; flags are not affected
AND	Destination, source	AND AX, BX AND AL, [SI] AND AX, 0200H	$AX \leftarrow AX \cdot BX$ $AL \leftarrow AL \cdot [SI]$ $AX \leftarrow AX \cdot 0200H$	Perform logical AND operation of the source and destination byte or word operands bit by bit; the result is stored in the destination operand; AF is undefined, all other flags are updated

Contd.

OR	Destination, source	OR AX, BX OR AL, [SI] OR AX, 0200H	$AX \leftarrow AX + BX$ $AL \leftarrow AL + [SI]$ $AX \leftarrow AX + 0200H$	Perform logical OR operation of the source and destination byte or word operands bit by bit; the result is stored in the destination operand. AF is undefined, all other flags are updated
XOR	Destination, source	XOR AX, BX XOR AL, [SI] XOR AX, 0200H	$AX \leftarrow AX \oplus BX$ $AL \leftarrow AL \oplus [SI]$ $AX \leftarrow AX \oplus 0200H$	Perform logical exclusive-OR operation of the source and destination byte or word operands bit by bit; the result is stored in the destination operand. AF is undefined, all other flags are updated
TEST	Destination, source	TEST AX, BX TEST AL, [SI] TEST AX, 0200H	AX, BX; update flags AL, [SI]; update flags AX, 0200H; update flags	Perform logical AND operation of the source and destination byte or word operands bit by bit; the operands remain unchanged; AF is undefined, all other flags are updated

Table 7 8086 Shift and Rotate Instructions Set Summary

Opcode	Operand	Mnemonics	Symbolic Operation	Comments
SAL/SHL	Destination, count	SAL AX, 1 SAL AX, CL	$A_{n+1} \leftarrow A_n, A_{15} \leftarrow A_{14},$ $A_0 \leftarrow 0, CF \leftarrow A_{15}$	Shift word or byte operand left or right once or CL times
SAR	Destination, count	SAR AX, 1 SAR AX, CL	$A_n \leftarrow A_{n+1}, CF \leftarrow A_0, A_{15} \leftarrow A_{15}$	AF is undefined, all other flags are updated;
SHR	Destination, count	SHR AX, 1 SHR AX, CL	$A_n \leftarrow A_{n+1}, CF \leftarrow A_0, A_{15} \leftarrow 0$	For single-bit shift operation, OF is set if the sign of the operand changes
RCL	Destination, count	RCL AX, 1 RCL AX, CL	$A_{n+1} \leftarrow A_n, CF \leftarrow A_{15}, A_0 \leftarrow CF$	Rotate word or byte operand left or right once or CL times; CF and OF are affected; For single-bit shift operation, OF is set if the sign of the operand changes.
RCR	Destination, count	RCR AX, 1 RCR AX, CL	$A_n \leftarrow A_{n+1}, A_{15} \leftarrow CF, CF \leftarrow A_0$	
ROL	Destination, count	ROL AX, 1 ROL AX, CL	$A_{n+1} \leftarrow A_n, A_0 \leftarrow A_{15}, CF \leftarrow A_{15}$	
ROR	Destination, count	ROR AX, 1 ROR AX, CL	$A_n \leftarrow A_{n+1}, A_{15} \leftarrow A_0, CF \leftarrow A_0$	

Table 8 8086 Jump Instructions Set Summary

<i>Opcode</i>	<i>Operand</i>	<i>Mnemonics</i>	<i>Symbolic Operation</i>	<i>Comments</i>
JMP	Near target	JMP MEM JMP [MEMW] JMP [BX] JMP AX	$IP \leftarrow MEM$ $IP \leftarrow [MEMW+1:MEMW]$ $IP \leftarrow [BX+1:BX]$ $IP \leftarrow AX$	After execution of JMP instruction, transfer control to near target location within the segment; the addressing mode will be direct, memory indirect or register indirect
JMP	Short target	JMP SHORT MEM	$IP \leftarrow MEM$	After execution of this instruction transfer control to short target location; the addressing mode will be direct only
JMP	Far target	JMP FAR MEMF JMP[MEMW] JMP DWORD [BX]	$IP \leftarrow 0003H$; $CS \leftarrow 9000H$ $IP \leftarrow [0102H:1001H]$; $CS \leftarrow [0104H:0103H]$ $IP \leftarrow [BX+1:BX]$; $CS \leftarrow [BX+3:BX+2]$	After execution of this instruction transfer control to far target location within the segment
Jcond	Short target	JNC MEM	If $CF=0$, then $IP \leftarrow MEMS$	After execution of this instruction transfer control to the short target address if the condition is true. Conditional jumps are possible only for short targets
JCXZ	Short target	JCXZ MEM	If $CX=0$, then $IP \leftarrow MEMS$	If $CX=0$, transfer control to the short target address

Table 9 8086 Loop Instructions Set Summary

<i>Opcode</i>	<i>Operand</i>	<i>Mnemonics</i>	<i>Symbolic Operation</i>	<i>Comments</i>
LOOP	Short target	LOOP MEM	$CX \leftarrow CX - 1$ If $CX \neq 0$, then $IP \leftarrow MEM$	Decrement CX register and transfer control to the short target address if $CX \neq 0$
LOOPE/ LOOPZ	Short target	LOOPZ MEM	$CX \leftarrow CX - 1$ If $(CX \neq 0) \cdot (ZF = 1)$, then $IP \leftarrow MEM$	Decrement CX register and transfer control to the short target address if $(CX \neq 0) \cdot (ZF=1)$; this instruction affect the flag $ZF = 1$
LOOPNE/ LOOPNZ	Short target	LOOPNZ MEM	$CX \leftarrow CX - 1$ If $(CX \neq 0) \cdot (ZF = 0)$, then $IP \leftarrow MEM$	Decrement CX register and transfer control to the short target address if $(CX \neq 0) \cdot (ZF = 0)$; this instruction affect the flag $ZF = 0$

Contd.

Table 10 8086 CALL and RETRUN Instructions Set Summary

<i>Opcode</i>	<i>Operand</i>	<i>Mnemonics</i>	<i>Symbolic Operation</i>	<i>Comments</i>
Call	Near target	CALL MEM CALL [MEMW] CALL [DI] CALL DI	$SP \leftarrow SP - 2;$ $[SP + 1: SP] \leftarrow IP;$ $IP \leftarrow MEM$ $SP \leftarrow SP - 2;$ $[SP + 1: SP] \leftarrow IP;$ $IP \leftarrow [0101H; 0100H]$ $SP \leftarrow SP - 2;$ $[SP + 1: SP] \leftarrow IP;$ $IP \leftarrow [DI + 1: DI]$ $SP \leftarrow SP - 2;$ $[SP + 1: SP] \leftarrow IP;$ $IP \leftarrow DI$	IP is pushed onto the top of the stack and control is transferred within the segment to the near target address
CALL	Far target	CALL FAR MEMF CALL [MEMW] CALL DWORD [DI]	$SP \leftarrow SP - 2;$ $[SP + 1: SP] \leftarrow CS;$ $SP \leftarrow SP - 2;$ $[SP + 1: SP] \leftarrow IP;$ $IP \leftarrow 0100H$ Same as above except: $CS \leftarrow [0103H : 0102H];$ $IP \leftarrow [0101H; 0100H]$ Same as above except: $CS \leftarrow [DI + 3: DI + 2];$ $IP \leftarrow [DI + 1: DI]$	CS and IP are pushed onto the top of the stack and control is transferred to the new segment and far target address
RET	n(near)	RET RET 8	$IP \leftarrow [SP + 1: SP];$ $SP \leftarrow SP + 2$ $IP \leftarrow [SP + 1: SP];$ $SP \leftarrow SP + 2 + 8$	The word at the top of the stack is popped into IP transferring control to this new address; RET normally used to return control to the instruction following a near subroutine call; if included, the optional pop value (n) is added to SP
RET	n(far)	RET RET 8	$IP \leftarrow [SP + 1: SP];$ $SP \leftarrow SP + 2;$ $CS \leftarrow [SP + 1: SP];$ $SP \leftarrow SP + 2;$ $IP \leftarrow [SP + 1: SP];$ $SP \leftarrow SP + 2;$ $CS \leftarrow [SP + 1: SP];$ $SP \leftarrow SP + 2 + 8 ;$	As the above except that double word at the top of the stack is popped into IP and CS transferring control to this new far address

Table 11 8086 String Instructions Set Summary

<i>Opcode</i>	<i>Operand</i>	<i>Mnemonics</i>	<i>Symbolic Operation</i>	<i>Comments</i>
STOSB	None	STOSB	ES:[DI] ←AL If DF = 0, DI ←DI+1. If DF = 1, DI ←DI-1.	Transfer a byte or word from register AL to the string element addressed by DI in the extra segment; When DF = 0, increment DI, otherwise decrement DI; Flags are not affected
STOSW	None	STOSW	ES:[DI] ←AL ES:[DI+1] ←AH. If DF = 0, DI ←DI+2. If DF = 1, DI ←DI-2.	Transfer a word from register AX to the string element addressed by DI in the extra segment; If DF = 0, increment DI, else decrement DI; Flags are not affected
STOS	Destination	STOS MEMB STOS MEMW	ES:[MEMB] ←AL If DF=0, MEMB ← MEMB + 1. If DF=1, MEMB ← MEMB -1. ES:[MEMW] ←AL ES:[MEMW + 1] ←AH. If DF=0, MEMW ← MEMW +2. If DF=1, MEMW ← MEMW -2.	Transfer a byte from register AL to the string element addressed by DI in the extra segment; when DF = 0, increment MEMB, otherwise decrement MEMB. Flags are not affected Transfer a word from register AX to the string element addressed by DI in the extra segment; if DF=0, increment MEMW, else decrement MEMW. Flags are not affected
LODSB		LODSB	AL←DS:[SI]. If DF=0, SI←SI+1. If DF=1, SI← SI-1.	Transfer a byte from the string element addressed by DS:SI to register AL; If DF=0, increment SI, else decrement SI. Flags are not affected
LODSW		LODSW	AL←DS:[SI]. AH←DS:[SI+1]. If DF=0, SI←SI+2. If DF=1, SI←SI-2.	Transfer a word from the string element addressed by DS:SI to register AX; If DF=0, increment SI, else decrement SI; Flags are not affected
LODS	Source	LODS MEMB	AL←DS:[MEMB]. If DF=0, MEMB←MEMB+1. If DF=1, MEMB←MEMB-1.	Transfer a byte from the string element addressed by DS:MEMB to register AL; When DF=0, increment MEMB, else decrement MEMB; flags are not affected

Contd.

		LODS MEMW	$AL \leftarrow DS:[MEMW]$. $AH \leftarrow DS:[MEMW+1]$. If $DF=0$, $MEMWDS \leftarrow MEMW+2$. If $DF=1$, $MEMW \leftarrow MEMW-2$.	Transfer a word from the string element addressed by $DS:MEMW$ to register AX . when $DF=0$, increment $MEMW$, otherwise decrement $MEMW$; flags are not affected
MOVSB	None	MOVSB	$ES:[DI] \leftarrow DS:[SI]$. If $DF=0$, $DI \leftarrow DI+1$, $SI \leftarrow SI+1$. If $DF=1$, $DI \leftarrow DI-1$, $SI \leftarrow SI-1$.	Transfer a byte from the string element addressed by $DS:SI$ to the string element addressed by $ES:DI$; if $DF=0$, increment SI and DI , else decrement SI and DI . Flags are not affected
MOVSW	None	MOVSW	$ES:[DI] \leftarrow DS:[SI]$ $ES:[DI+1] \leftarrow DS:[SI+1]$ If $DF=0$, $DI \leftarrow DI+2$ $SI \leftarrow SI+2$ If $DF=1$, $DI \leftarrow DI-2$ $SI \leftarrow SI-2$.	Transfer a word from the string element addressed by $DS:SI$ to the string element addressed by $ES:DI$; if $DF=0$, increment SI and DI , else decrement SI and DI . Flags are not affected

Table 12 8086 String Instructions Set Summary

Opcode	Operand	Mnemonics	Symbolic Operation	Comments
MOVS	Destination,Source	MOVS MEMBES ,MEMBDS	$ES:[MEMBE] \leftarrow DS:[MEMBD]$. If $DF=0$, $MEMBE \leftarrow MEMBE+1$ $MEMBD \leftarrow MEMBD+1$. If $DF=1$, $MEMBE \leftarrow MEMBE-1$ $MEMBD \leftarrow MEMBD-1$.	Transfer a byte from the string element addressed by $DS:MEMBD$ to the string element addressed by $ES:MEMBE$; if $DF=0$, increment $MEMBD$ and $MEMBE$, else decrement $MEMBD$ and $MEMBE$. Flags are not affected
		MOVS MEMWE, MEMWD	$ES:[MEMWE] \leftarrow DS:[MEMWD]$ $ES:[MEMWE+1] \leftarrow DS:[MEMWD+1]$ If $DF=0$, $MEMWE \leftarrow MEMWE+2$ $MEMWD \leftarrow MEMWD+2$ If $DF=1$, $MEMWE \leftarrow MEMWE-2$ $MEMWD \leftarrow MEMWD-2$.	Transfer a word from the string element addressed by $DS:MEMWD$ to the string element addressed by $ES:MEMWE$ in the extra segment; if $DF=0$, increment $MEMWD$ and $MEMWE$, else decrement $MEMWD$ and $MEMWE$; flags are not affected

Contd.

SCASB		SCASB	AL - ES:[DI]; If DF=0, DI←DI+1. If DF=1, DI←DI-1.	Subtract the byte of the string element addressed by ES:DI from AL. if DF=0, increment DI, else decrement DI; flags are updated
SCASW		SCASW	AX - ES:[DI+1:DI]; If DF = 0, DI←DI+2 If DF = 1, DI←DI-2.	Subtract the word of the string element addressed by ES:DI from AX; if DF=0, increment DI, else decrement DI; flags are updated
SCAS	Destination	SCAS MEMBE	AL - ES:[MEMBE]; If DF = 0, MEMBE ← MEMBE +1. If DF=1, MEMBE ← MEMBE -1.	Subtract the byte of the string element addressed by ES: MEMBE from AL; if DF=0, increment MEMBE, else decrement MEMBE. Flags are updated
		SCAS MEMWE	AX - ES:[MEMWE +1: MEMWE]; If DF=0, MEMWE I← MEMWE +2 If DF=1, MEMWE ← MEMWE -2.	Subtract the word of the string element addressed by ES: MEMWES from AX; if DF=0, increment MEMWE, else decrement MEMWE; flags are updated
CMPSB		CMPSB	DS:[SI] - ES:[DI]; If DF=0, DI←DI+1 SI←SI+1 If DF=1, DI←DI-1 SI←SI-1.	Subtract the byte of the destination string element addressed by ES:DI in the extra segment from byte of the source string element addressed by DS:SI; if DF = 0, increment DI and SI, else decrement SI and DI. Flags are updated
CMPSW		CMPSW	DS:[SI+1:SI] - ES:[DI+1 :DI] If DF=0, DI←DI+2 SI←SI+2. If DF=1, DI←DI-2 SI←SI-2	Subtract the word of the destination string element addressed by ES:DI from word of the source string element addressed by DS:SI; if DF = 0, increment DI and SI, else decrement SI and DI; flags are updated

Table 13 8086 String Instructions Set Summary

<i>Opcode</i>	<i>Operand</i>	<i>Mnemonics</i>	<i>Symbolic Operation</i>	<i>Comments</i>
CMPS	Dest, source	CMPS MEMBE, MEMBD	DS:[MEMBD] – ES: [MEMBDS]; If DF=0 MEMBE← MEMBE +1 MEMBD ← MEMBD +1 If DF=1, MEMBE ← MEMBE -1 MEMBD ← MEMBD -1.	Subtract the byte of the destination string element addressed by ES:MEMBE from byte of the source string element addressed by DS:MEMBD; if DF=0, increment MEMBE and MEMBD, else decrement MEMBD and MEMBE; flags are updated
		CMPS MEMWES, MEMWDS	DS:[MEMWD+1: MEMWD] – ES: [MEMWE+1 : MEMWE]; If DF=0, MEMWE ← MEMWE +2 MEMWD ← MEMWDS +2. If DF=1, MEMWE ← MEMWE -2 MEMWD ← MEMWDS	Subtract the word of the destination string element addressed by ES:MEMBE from word of the source string element addressed by DS:MEMBD; if DF=0, increment MEMBE and MEMBD, else decrement MEMBD and MEMBE; flags are updated
REP		REP STOSB	CX←CX-1. Repeat until CX=0	The string instruction following the REP prefix is repeated until CX becomes to 0
		REP STOSW	CX←CX-1. Repeat until CX=0	
		REP MOVSB	CX←CX-1. Repeat until CX=0	
		REP MOVSW	CX←CX-1. Repeat until CX=0	
REPE/ REPZ		REPZ SCASB	CX←CX-1. Repeat if (ZF=1) and CX≠0	Repeat the string operation if (ZF = 1) and CX ≠ 0
		REPZ SCASW	CX←CX-1. Repeat if ZF=1 and CX≠0.	
		REPZ CMPSB	CX←CX-1. Repeat if ZF=1 and CX≠0.	
		REPZ CMPSW	CX←CX-1. Repeat if ZF=1 and CX≠0	
REPNE/ EPNZ		REPNE SCASB	CX←CX-1. Repeat if ZF=0 and CX≠0.	Repeat the string operation if (ZF=0) and CX≠0
		REPNE SCASW	CX←CX-1. . Repeat if ZF=0 and CX≠0	
		REPNE CMPSB	CX←CX-1. . Repeat if ZF=0 and CX≠0	
		REPNE CMPSW	CX←CX-1. Repeat if ZF=0 and CX≠0	

Table 14 8086 Processor Control Instructions Set Summary

<i>Opcode</i>	<i>Operand</i>	<i>Mnemonics</i>	<i>Symbolic Operation</i>	<i>Comments</i>
STC	None	STC	$CF \leftarrow 1$	Set carry flag
CLC	None	CLC	$CF \leftarrow 0$	Clear carry flag
CMC	None	CMC	$CF \leftarrow \overline{CF}$	Complement carry flag
STD	None	STD	$DF \leftarrow 1$	Set direction flag
CLD	None	CLD	$DF \leftarrow 0$	Clear direction flag
STI	None	STI	$IF \leftarrow 1$	Set interrupt flag
CLI	None	CLI	$IF \leftarrow 0$	Clear interrupt flag
HLT	None	HLT	None	Halt
WAIT	None	WAIT	None	Wait state when $\overline{TEST} = 1$
LOCK	Instruction	LOCK MOV AX,BX	None	$\overline{LOCK} = 0$ used to prevent coprocessors from accessing the bus during execution of instruction
NOP	None	NOP	None	No operation
ESC	Number, source	ESC FF, MEMW	Data bus \leftarrow [MEMW]	Put the contents of the memory source operand on the data bus and execute NOP instruction

Model Question Paper - 1

Full Marks : 100

Duration: 3 hours

GROUP-A (Multiple-Choice Questions)

1. Choose the appropriate answer for the following questions from the given options. (20×1=20)
- (i) The 64-bit processor is
 - (a) Pentium
 - (b) Pentium II
 - (c) Pentium III
 - (d) Pentium 4
 - (ii) The program counter (PC) in a microprocessor
 - (a) keeps the address of the next instruction to be fetched
 - (b) counts the number of instructions being executed on the microprocessor
 - (c) counts the number of programs being executed on the microprocessor
 - (d) counts the number of interrupts handled by the microprocessor
 - (iii) CALL 8000H is an instruction of
 - (a) direct addressing mode
 - (b) indirect addressing mode
 - (c) register addressing mode
 - (d) immediate addressing mode
 - (iv) OUT 02H is executed by
 - (a) one machine cycle
 - (b) two machine cycles
 - (c) three machine cycles
 - (d) Four machine cycles
 - (v) When the RET instruction is executed at the end of a subroutine,
 - (a) the memory address of the RET instruction is transferred to the program counter
 - (b) two data bytes stored in the top locations of the stack are transferred to the stack pointer
 - (c) the data where the stack is initialized is transferred to the stack pointer
 - (d) two data bytes stored in the top two locations of the stack are transferred to the program counter
 - (vi) The number of address lines required to access 2 Mbytes of data from the microprocessor
 - (a) 16-bit address lines
 - (b) 8-bit address lines
 - (c) 20-bit address lines
 - (d) 21-bit address lines
 - (vii) The RIM instruction is used to
 - (a) enable RST 7.5, 6.5 and 5.5
 - (b) disable RST 7.5, 6.5 and 5.5
 - (c) enable or disable RST 7.5, 6.5 and 5.5
 - (d) none of these
 - (viii) When Port A is used as input, Port B and Port C are used as output, the control word of 8255 is
 - (a) 80H
 - (b) 90H
 - (c) 85H
 - (d) 86H

- (ix) Which pin is used to control the output of counters 2 of 8253 in Mode 2?
 (a) GATE 0 (b) GATE 1 (c) GATE 2 (d) GATE 3
- (x) The UART performs
 (a) a serial-to-parallel conversion (b) a parallel-to serial conversion
 (c) control and monitoring functions (d) all
- (xi) 8279 is known as
 (a) DMA controller (b) programmable keyboard display interface
 (c) counter (d) interrupt controller
- (xii) The resolution of a D/A converter is 0.4 per cent of full scale range. It is a/an
 (a) 8-bit converter (b) 10-bit converter
 (c) 12-bit converter (d) 16-bit converter
- (xiii) 80386 can be operated in
 (a) real mode only (b) protected virtual mode only
 (c) real and protected virtual mode only
 (d) real, protected virtual mode and virtual 8086 mode
- (xiv) 80486 is the combination of
 (a) 80386 and 80387 (b) 80386 and 80287
 (c) 80286 and 80387 (d) 80286 and 80287
- (xv) Physical memory of 8086 is (a) 1 MB (b) 64 KB (c) 2 MB (d) 4 MB
- (xvi) What are the control signals of the 8085 microprocessor used to interface I/O devices?
 (a) IO / \overline{M} , \overline{RD} , \overline{WR} (b) IO / \overline{M} (c) \overline{RD} (d) \overline{WR}
- (xvii) Which of the following instructions is indirect addressing?
 (a) MOV A, R0 (b) MOV A, 40H (c) MOV R7, #55 (d) MOV A, @R0
- (xviii) Which of the following signals indicates a/an upper 8-bit data transfer?
 (a) A0 = 0 and BHE = 0 (b) A0 = 1 and BHE = 1
 (c) A0 = 0 and BHE = 1 (d) A0 = 1 and BHE = 0
- (xix) SSE2 instructions are compatible with
 (a) Pentium processor (b) Pentium Pro processor
 (c) Pentium 4 processor (d) Pentium II processor
- (xx) The Pentium processor is a
 (a) net burst architecture (b) superscalar super-pipelined architecture
 (c) P6 architecture (d) 64-bit core architecture

GROUP-B

(Short-Answer-Type Questions)

Answer any four questions.

(4×5=20)

- Draw and explain the time multiplexing of AD_0-AD_7 in the 8085 microprocessor.
- What are the different addressing modes of 80286? Discuss any one addressing mode with examples.
- Write the difference between Real Address Mode and Protected Virtual Address Mode (PVAM).

12. (a) Explain how data can be transferred using 8251 USART at different baud rates. Write the features of 8251.
- (b) What are the software interrupts of the 8085 microprocessor? Mention interrupts instructions with their hex code and vector address. How is the vector address for a software interrupt determined?
- (c) Write the control word format for I/O mode operation of 8255.
- (d) Write a program to read the count value of the counter while counting is going on. Assume Counter 0 in Mode 0 with a count value of 80FFH.
13. (a) How is the physical address computed in real address mode 80286.
- (b) Explain the concept of virtual memory.
- (c) What do you mean by a descriptor? Discuss different types of descriptor supported by the 80286 and their applications.
- (d) What are the different cache memories in microprocessors?
Explain the advantages of separate code and data caches of the Pentium processor.
14. (a) Draw the block diagram of the 8051 microcontroller and explain the operation of each block briefly.
- (b) Explain the operation of the following pins of the 8051 microcontroller:
- (i) RST (ii) T × D (iii) R × D (iv) \overline{EA}
- (v) \overline{PSEN}
- (c) What is the difference between internal and external program memory? Why is external program memory used in the microcontroller? How can \overline{EA} be used to access internal and external program memory?
- (d) Explain a microcontroller-based traffic-light control system with an assembly-language program.

Model Question Paper - 2

Full Marks :100

Duration: 3 hours

GROUP-A

(Multiple-Choice Questions)

1. Choose the appropriate answer for the following questions from the given options. (20×1=20)
- (i) If a microprocessor is capable of addressing 64K bytes of memory, its address bus width is
 - (a) 16 bits
 - (b) 20 bits
 - (c) 8 bits
 - (d) none of these
 - (ii) Which of the following processors has in-built math processor?
 - (a) 8086
 - (b) Pentium-4
 - (c) 8085
 - (d) 8088
 - (iii) Three-byte instructions should have
 - (a) opcode and an operand
 - (b) opcode only
 - (c) opcode and two operand
 - (d) operand only
 - (iv) The number of flags of the 8086 microprocessor is
 - (a) 7
 - (b) 8
 - (c) 9
 - (d) 10
 - (v) The SUB A instruction in 8085 microprocessor
 - (a) resets the carry and sign flag
 - (b) sets the zero and parity flag
 - (c) sets the zero and carry flag
 - (d) resets the zero and sign flag
 - (vi) Opcode is
 - (a) the part of the construction which tells the computer what operation to perform
 - (b) an auxiliary register that stores the data to be added or subtracted from the accumulator
 - (c) the register that receives the constructions from memory
 - (d) the data which will be used in data manipulation of instruction
 - (vii) To design a 4 KB RAM with 1024-byte RAM ICs, how many ICs are required?
 - (a) 4
 - (b) 8
 - (c) 2
 - (d) none of these
 - (viii) 8237 is a
 - (a) DMA controller
 - (b) programmable keyboard display interface
 - (c) counter
 - (d) interrupt controller
 - (ix) The concept of memory management, privilege and protection are incorporated in
 - (a) 8088
 - (b) 8086
 - (c) 80186
 - (d) 80286
 - (x) 80386 has
 - (a) 24-bit address bus and 16-bit data bus
 - (b) 32-bit address bus and 32-bit data bus
 - (c) 24-bit address bus and 32-bit data bus
 - (d) 16-bit address bus and 32-bit data bus.

- (xi) The N -bit successive approximation ADC requires
 (a) $2^N - 1$ clock pulses (b) 2^N clock pulses
 (c) N clock pulses (d) none of these
- (xii) The physical address when DS = 2345H and IP = 1000H is
 (a) 23450H (b) 24450H (c) 12345H (d) 2345H
- (xiii) 2's complement instruction is (a) NEG (b) NOT (c) CMP (d) CMC
- (xiv) Pentium MMX processor has
 (a) 57 MMX instructions (b) 56 MMX instructions
 (c) 55 MMX instructions (d) 54 MMX instructions
- (xv) 8251 is a
 (a) USART IC (b) counter
 (c) interrupt controller (d) programmable peripheral interface
- (xvi) What will be the output of A after execution of the following instructions?
 MOV A, #55 ANL A, #67
 (a) 54 (b) 45 (c) 55 (d) 67
- (xvii) The resolution of a 12-bit D/A converter with a full scale output of 10 V is
 (a) $\frac{10}{2^{12} - 1}$ (b) $\frac{10}{2^{12} + 1}$ (c) $\frac{10}{2^{12}}$ (d) none of these
- (xviii) In 8259, which is the lowest priority interrupt?
 (a) IR₀ (b) IR₃ (c) IR₄ (d) IR₇
- (xix) The bit set reset mode in 8255 is used with which one of the following?
 (a) Port A (b) Port B (c) Port C (d) none of these
- (xx) The 8051 microcontroller has
 (a) 4K bytes of on-chip ROM (b) 8K bytes of on-chip ROM
 (c) 16K bytes of on-chip ROM (d) 32K bytes of on-chip ROM

GROUP-B

(Short-Answer-Type Questions)

Answer any four questions.

(4 × 5 = 20)

2. What is hyper-threading technology? What is the difference between Core 2 dual and Quad core?
3. Mention the purpose of SID and SOD lines. Explain the functions of SIM and RIM instructions in 8088 μ P.
4. Explain memory mapped I/O and peripheral mapped I/O. Compare the memory mapped I/O with peripheral mapped I/O.
5. What are the new features of 80486 over 80386?
6. What is translation look-aside buffer? How can it increase the speed of execution of programs?
7. Define stack. Explain function of PUSH and POP instructions.
8. What do you mean by paging? What are the advantages of paging in Pentium?

GROUP-C

(Long-Answer-Type Questions)

Answer any four questions.

(4×15 = 60)

9. (a) Draw the schematic diagram of all the functional blocks of the 8085 microprocessor and explain the 8085 microprocessor architecture briefly.
- (b) Explain the generation of \overline{MEMR} , \overline{MEMW} , \overline{IOR} and \overline{IOW} control signals from IO / \overline{M} , \overline{RD} , \overline{WR} signals.
- (c) Explain time-delay loop using register and register pair. Write some applications of time-delay loop.
- (d) Calculate the time required to execute the following two instructions if the system clock frequency is 1 MHz.

LOOP: MOV A, B	4 T-states
DEC B	4 T-states
JMP LOOP	10 T-states
10. (a) Draw the timing diagram for execution of the instruction LXI H,8000H.
- (b) Specify the contents of registers A and B and the status of flags S, Z and CY as the following instructions are executed.

XRA A;	MVI B, 4BH;	SUI 42;	ANA B;	HLT
--------	-------------	---------	--------	-----
- (c) Write an assembly-language program in 8085 to add two sixteen-bit data. Store the result and carry in two different register pairs.
- (d) Write a BSR control word to set bits PC_7 and PC_0 and to reset them after a 1-second delay.
11. (a) Explain memory organization of 80C51 microcontroller.
- (b) What is a SFR? How you can identify the bit addressable SFRs from their addresses?
- (c) What are the ports used for external memory access? How can an I/O pin be used as both input and output?
- (d) Draw a circuit diagram for keyboard interface with 8051 microcontroller and write a program for reading any key.
12. (a) Draw a circuit diagram of ZCD and discuss its operating principle with waveforms.
- (b) Discuss a microprocessor-based frequency measurement and display scheme. Draw the flowchart for frequency measurement. Give the assembly language program for frequency measurement.
- (c) Determine the control word for the following configuration of the ports of 8255.
 - (i) Port A-output and mode of port A is mode 1
 - (ii) Port B-output and mode of port B is mode 1
 - (iii) Remaining pins of Port C are used as input
- (d) Write the interfacing procedure to interface 8253 with the 8085 microprocessor.
13. (a) Define physical address, linear address and logical address.
What are the differences between physical address, linear address and logical address?
- (b) Draw the internal architecture of Pentium microprocessor and explain its operation.
- (c) Write the architectural difference between 80486 and Pentium processors.
- (d) Enlist the special features of CISC and RISC processors.
14. Write short notes on of the following (any three)
 - (a) Power PC 601 microprocessor
 - (b) Pentium MMX processor
 - (c) Superscalar organization of Pentium processor
 - (d) MESI protocol

Model Question Paper - 3

Full Marks: 100

Duration: 3 hours

GROUP-A (Multiple-Choice Questions)

1. Choose the appropriate answer for the following questions from the given options. (20×1=20)
- (i) The 16-bit processor is
(a) 8085 (b) 8086 (c) 80486 (d) Pentium
 - (ii) Address bus of a microprocessor is
(a) unidirectional (b) bi-directional (c) unidirectional as well as bi-directional
(d) none-of these
 - (iii) The number of flags of the 8085 microprocessor is
(a) 6 (b) 5 (c) 4 (d) 3
 - (iv) MOV A, C is executed by
(a) 1 machine cycle (b) 2 machine cycle (c) 3 machine cycle (d) 4 machine cycle
 - (v) When a CALL instruction is executed, the stack pointer register is
(a) decremented by two (b) incremented by two
(c) decremented by one (d) incremented by one
 - (vi) The PC contains 8452H and SP contains 88D6H. What will be the content of PC and SP following a CALL to subroutine at the location 82AFH?
(a) 82AF, 88D4 (b) 82AF, 8450 (c) 8450, 88D4 (d) 82AF, 8452
 - (vii) Which is the highest priority interrupt in the 8085 microprocessor?
(a) TRAP (b) RST 6.5 (c) RST 5.5 (d) RST 7.5
 - (viii) 8259 is a
(a) programmable interrupt controller (b) DMA controller
(c) programmable keyboard display interface (d) programmable counter
 - (ix) If A_0 and A_1 pins of 8255 are 00, which port will be selected?
(a) Port A (b) Port B (c) Port C (d) None of these
 - (x) 8253 has
(a) 6 modes of operation (b) 5 modes of operation
(c) 4 modes of operation (d) 3 modes of operation
 - (xi) 8279 displays can operate in
(a) 8 8-bit character display-left entry only
(b) 16 16-bit character display-left entry only
(c) 8 8-bit character display-right entry only

- (d) 8 8-bit character display-left and right entry and 16 16-bit character display-left and right entry
- (xii) The signals used for DMA operation are
 (a) HRQ (b) HLDA (c) HRQ and HLDA (d) none of these
- (xiii) In hyper-threading technology
 (a) a single processor appears as one logical processors
 (b) a single processor appears as four logical processors
 (c) a single processor appears as three logical processors
 (d) a single processor appears as two logical processors
- (xiv) 8086 has
 (a) 16-bit data bus and 20 bit address bus
 (b) 8-bit data bus and 20 bit address bus
 (c) 16-bit data bus and 16 bit address bus
 (d) 8-bit data bus and 16 bit address bus
- (xv) What is the addressing mode of instruction MOV AX, [BX + SI + 06]?
 (a) Index addressing (b) Base addressing
 (c) Base index addressing (d) Base index displacement addressing
- (xvi) The difference between the 80386 and 80486 flag register is
 (a) alignment check flag (b) virtual mode flag
 (c) resume flag (d) trap flag
- (xvii) The instruction prefetcher of 80386 processor consists of
 (a) 16-byte instruction code queue (b) 12-byte instruction code queue
 (c) 10-byte instruction code queue (d) 6-byte instruction code queue
- (xviii) P6 family Pentium processor has
 (a) 36-bit address bus (b) 32-bit address bus
 (c) 24-bit address bus (d) 20-bit address bus
- (xix) Pentium processor consists of
 (a) two independent integer pipelines and a floating-point pipeline
 (b) one integer pipeline and a floating-point pipeline
 (c) two integer pipelines
 (d) one floating-point pipeline
- (xx) The 80C51 microcontroller family has
 (a) 32 pins for I/O (b) 24 pins for I/O
 (c) 16 pins for I/O (d) 8 pins for I/O

GROUP-B
(Short-Answer-Type Questions)

Answer any four questions.

(4 × 5 = 25)

2. What do you mean by paging? What are the advantages of paging?

3. Draw the schematic diagram to generate Read/Write control signals for memory and I/O devices in the 8085 microprocessor.
4. What are RISC and CISC? Write the difference between RISC and CISC. Give a list of examples of RISC and CISC.
5. Mention the different modes of operation of 8253 IC. Explain rate generator mode of 8253.
6. Write the functions of the following pins of 8259A:
 - (i) $T \times D$
 - (ii) $R \times D$
 - (iii) $T \times RDY$
 - (iv) C / \overline{D}
 - (v) \overline{DSR}
7. What is serial data transfer? Write the difference between synchronous and asynchronous data transfer.
8. What are the advantages of DMA controlled data transfer over interrupt driven data transfer? Write the difference between 8057 and 8037 DMA controllers.

GROUP-C

(Long-Answer-Type Questions)

Answer any four questions.

(4 × 15 = 60)

9. (a) What are the flags in 8085? Discuss the flag register of 8085 with some examples.
 (b) An 8085 program subtracts the hex number 22H from FFH and places the result in its accumulator. What would be the status of the 8085 flags *CY, P, AC, Z, S* on completion of this subtraction?
 (c) A block of 10 bytes of data is stored at the memory location starting from 9000H. Write a program to move this block to the memory location starting from 9050H.
 (d) Draw the RIM instruction format and discuss with an example. "A RIM instruction should be performed immediately after TRAP occurs". Why?
10. (a) What are the advantages and disadvantages of I/O mapped I/O over CPU initiated data transfer? Explain why I/O mapped I/O data transfer technique is limited to 256 input and 256 out peripherals.
 (b) What do you mean by priority interrupts? Explain the operation of different interrupts available in 8085 and 8086 microprocessors.
 (c) Draw an interface circuit of an A/D converter with 8085 μP and write a program to convert the analog input signal to digital output signal.
 (d) Explain the seven-segment display interfacing with 8279. How are sixteen-digit display interfaced with 8279?
11. (a) What are the different operating modes of 8255? Explain any one operating mode of 8255.
 (b) Write the control word in Mode 0 operation for the following cases:
 (i) Port A = Input port, Port B = output port, Port C = output port
 (ii) Port A = Input port, Port B = output port, Port C_U = output port, Port C_L = input port
 (c) Write a program to generate a square wave using 8255.
 (d) Describe the flowchart of DMA mode of data transfer. What do you mean by DMA cycle?
12. (a) What are the different addressing modes of 8086 microprocessors? Explain each addressing mode with examples.

- (b) Write the procedure to determine physical address for the following instructions as given below:
- (i) MOV AX, [SI+00FF]
 - (ii) MOV AL, CS:[BX+0200]
 - (iii) MOV AX, [3000]
 - (iv) MOV AL, [BX+SI+2F]
- Assume CS =5000H, IP =3000, SI=1000 and DS =6000.
- (c) Write the difference between the following instructions:
- (i) MUL and IMUL
 - (ii) DIV and IDIV
 - (iii) JUMP and LOOP
 - (iv) Shift and Rotate
- (d) Write instructions to perform the following operations:
- (i) Copy content of BX to a memory location in the data segment with offset 0100H
 - (ii) Increment content of CX by 2
 - (iii) Multiply AX with 16 bit data 4000H
 - (iv) Rotate left the content of AL by two bits
13. (a) What are the electrical quantities measured by the microprocessor? Draw the schematic block diagram of dc voltage (5 V) measurement and discuss briefly. Write an assembly-language program for this measurement. What modification is required in hardware and software to measure very high voltage ac and dc?
- (b) Draw the schematic pin diagram of Pentium processor. Write the function of the following pins of the Pentium processor.
- (i) ADS#
 - (ii) BE7#–BE0#
 - (iii) APCHK#
 - (iv) DP₇–DP₀
 - (v) EADS#
- (c) Draw a circuit diagram for keyboard interface with the 8051 microcontroller and write a program for reading any key.
- (d) Write an assembly-language program for 2 ms time delay. Assume the system clock time period is equal to 0.33 μs.
14. Write short notes on the following (any three):
- (a) 8259 Interrupt Controller
 - (b) Minimum/Maximum mode operation of 8086 μP
 - (c) Segment memory
 - (d) Architecture of 8051 microcontroller

Index

Chapter 1

Input devices 1, 20
Output devices 1, 2, 5
Memory 1, 5
Primary memory 2
Secondary memory 2
CPU 2
ALU 2, 3, 5
Registers 2,5
Timing and control unit 2...,3,5
Control unit 3
Microprocessor 3,6
Microcomputer 4,5
System bus 6
Address bus 6,7
Data bus 6,7
Control Bus 6
Von-Neumann architecture 6,7
Harvard architecture 6,7
Super Harvard architecture 6,7
ENIAC 8
EDSAC 8
EDVAC 8
SSI 9
MSI 9
LSI 9
VLSI 9
4004 9
8085 9
8080 9
8086 9
80386 10
80486 10
Pentium 10
Comparisons of
microprocessors 11
Evolution of
microprocessors 12
Microprocessor applications 12

Evolution of
microcontrollers 13
Applications of
microcontrollers 15

Chapter 2

8085 Microprocessor 18
8085 features 18
Architecture of 8085
Microprocessor 19
Bus Architecture of 8085 20
Input devices 20
Memory 20
Output devices 20
Operation of 8085 20
Memory read 21
Data flow 21
ALU 22
Accumulator 22,23
Temporary register 22
Flags 22
Timing and Control unit 23
Registers 23
Program counter 23,24
Stack pointer 23,24
PSW 23,25
General purpose registers 23
Special purpose registers 24
Status register 24
Parity flag 24,25
Carry flag 24
Auxiliary carry flag 24,25
Zero flag 24,25
Sign flag 24,25
Flag register 24
Instruction register 26
Memory address register 26
Temporary register 26
System bus 26

Address bus 26
Data bus 26,27
Control Bus 26,28
Multiplexing of address
bus 27,28
Pin diagram of 8085 29
A15-A8 30
AD7-AD0 30
ALE 30
READY 31
HOLD 31
HLDA 31
INTR 31
Memory and I/O read/write con-
trol signals 31
RST 5.5, RST 6.5 RST 7.5 31,32
TRAP 32
RESET 32
SID 32
SOD 32
Comparisons of 8085 and
8080 32

Chapter 3

Addressing modes 37
Direct addressing 38
Register addressing 38
Register indirect addressing 38
Immediate addressing 39
Implicit addressing 39
Instruction set 37,40
Operation code 37
Operand 37
Data transfer group 40
Arithmetic group 41
Logical group 41
Branch control group 42
Stack 43
Machine control group 43

Instruction format 43
 One byte Instruction 44
 Two byte Instruction 44
 Three byte Instruction 44,45
 Symbol and abbreviations 45
 MOV 46,47
 MVI 47
 LDA 47
 LDAX 47
 LXI 47
 LHLD 48
 STA 48
 SHLD 48
 XCHG 48
 ADD 48
 ADC 49
 ADI 49
 ACI
 DAD 49
 SUB 49
 SBB 50
 SUI 50
 SBI 50
 INR 50
 INX 51
 DCR 51
 DCX 51
 DAA 51
 CMP 51,52
 CPI 52
 ANA 52
 ANI 52
 ORA 52,53
 ORI 53
 XRA 53
 XRI 53
 RLC 53,54
 RRC 54
 RAL 54
 RAR 54,55
 CMA 55
 CMC 55
 STC 55
 JMP 55
 JC 56
 JNC 56

JP 56
 JNZ 57
 JZ 56
 CALL 57
 RET 58
 PCHL 58
 RST 0-7 59
 TRAP 59
 PUSH 59,60
 POP 60, 61
 XTHL 61
 SPHL 61
 EI 62
 DI 62
 NOP 62
 HLT 62
 SIM 62
 RIM 62
 IN 63
 OUT 63
 Timing Diagram 63,66
 Fetch operation 63
 Execute operation 64
 Machine cycle 64
 Instruction cycle 63,64
 Fetch cycle 64,66
 Memory read 67
 I/O read 69,70
 Memory write 71,72
 I/O write 72
 Flow of opcode and data 65
 Memory data register 65
 Instruction decoder 65

Chapter 4

Program 81
 Machine language 82
 Assembly language 82
 Assembler 83
 High level language 83,84
 Translator 83
 Compiler 83
 Interpreter 83
 Fortran 84
 Basic 85
 Cobol 85

Pascal 85
 C 86
 C++ and OOP 86
 LISP 86
 APL 86
 ADA 86
 PROLOG 87
 Stack 87
 Subroutines 89
 Nested Subroutines 90
 Time delay loop 91,93
 Modular programming 94,95
 Macro 95
 Instruction format 96
 Memory address 96
 Machine codes 96
 Labels 96
 Mnemonics 96
 Operands 97
 Comments 97
 Addition 98-103
 Decimal Addition 104-107
 Subtraction 107-111
 Decimal Subtraction 110
 One's complement 111
 Two's complement 112
 Shift 113,114
 Largest 114-117
 Smallest 117-120
 Descending order 120
 Ascending order 123
 Look-up table 126
 Square root 126
 Square 136
 Multiplication 127-129
 Division 130-132
 Hexadecimal to Binary 132
 Transfer block of data 135
 BCD to Binary 138
 Binary to Decimal 139

Chapter 5

Architecture of 8086 147, 149
 8086 147
 8088 147
 Compare 8085 and 8086 148

- Compare 8086 and 8088 148
- Bus interface unit 150
- Execution unit 150
- Fetch and Execute 150
- Process of fetching 151
- Instruction fetch 151
- Instruction queue 152
- Registers 152
 - Data registers 153
 - AX register 153
 - BX register 153
 - CX register 153
 - DX register 154
 - Segment registers 154
 - Code segment 154
 - Data segment 154
 - Stack segment 154
 - Extra segment 154
 - Pointer and index registers 154
 - Stack pointer 155
 - Base pointer 155
 - Source index 155
 - Destination index 155
 - Instruction pointer 155
 - Flag register 155
 - Carry flag, CF 155,156
 - Parity flag, PF 155,156
 - Auxiliary Carry flag, AF 155,156
 - Zero flag, ZF 155,156
 - Sign Flag, SF 155,156
 - Overflow flag, OF 155,156
 - Direction flag, DF 155,156
 - Interrupt flag, IF 155,156
 - Trap flag, TF 155,156
 - Logical address 156,157
 - Physical address 156,157
 - Effective address 157
 - Segment address 157
 - Address bus 158,161-164
 - Data bus 158,161-164
 - Control bus 158
 - Memory segmentation 158
 - Memory addressing 160
 - Memory map 160
 - Odd address bank 161-164
 - Even address bank 161-164
 - Pin description of 8086 164
 - AD15-AD0 Address/data bus 164
 - BHE Bus high enable 165
 - Ready 166
 - INTR 166
 - NMI 166
 - RESET 167
 - CLK 166A
 - MN/MX 166
 - Operating modes of 8086 167
 - Maximum mode 167, 174
 - Minimum mode 167, 171
 - HLDA 169
 - HOLD 169
 - QS1-QS0 Instruction queue status 169
 - S2,S1,S0 status signals 169
 - Lock 170
 - Memory read and write 170
 - Timing diagram of 8086 171
 - Memory read bus cycle 172, 175
 - Memory write bus cycle 173, 176
 - I/O read and write bus cycle 174, 176
 - 8088 processor 176-180
 - AD7-AD0 Address/data bus 178
 - A15-A8 Address bus 178
 - Timing diagram of 8088 179
 - Demultiplexing of system bus 180
 - 8284A Clock generator 183
 - 8286 Bi-directional bus 186
 - 8287 Bi-directional bus 186
 - 8282 input-output port 186
 - 8283 input-output port 186
 - 8288 bus controller 187

Chapter 6

 - Opcode 192
 - Operand 192
 - Assembly language 192
 - Addressing modes 192
 - Instruction format 192
 - Immediate addressing 193
 - Register addressing 193
 - Memory addressing 193
 - Direct addressing 194
 - Register indirect addressing 194
 - Based addressing 195
 - Indexed addressing 195
 - Based indexed addressing 196
 - Based indexed and displacement addressing 197
 - String addressing 197
 - Branch addressing 199
 - Intrasegment direct 200
 - Intrasegment indirect 200
 - Intersegment direct 201
 - Intersegment indirect 201
 - 8086 Instruction set 203
 - Classification of instructions 204
 - Instruction format 192
 - One byte instruction 204
 - Two byte instruction 205
 - Three and Four byte instruction 205
 - Five and Six byte instruction 206
 - Data transfer instructions 206
 - Arithmetic and logical instructions 206,216
 - Branch instructions 206
 - Loop instructions 206
 - Processor control instructions 206,238
 - Flag manipulation instructions 206
 - Shift and rotate instructions 206
 - String instructions 206,235
 - MOV 206-210
 - XCHG 211
 - LAHF 211
 - SAHF 211
 - IN 212
 - OUT 212

LEA 213
 LDS 213
 LES 214
 XLAT 214
 PUSH, PUSHF 214,215
 POP, POPF 214,216
 ADD 216
 ADC 217
 SUB 217
 SBB 218
 INC 218
 DEC 219
 NEG 219
 CMP 220
 Multiplication 220
 Division 220
 MUL, IMUL 220,221
 DIV, IDIV 221,222
 DAA 223
 DAS 223
 AAA 223
 AAS 224
 AAM 224
 CBW 224
 CWD 224
 NOT 225
 AND 225
 OR 225
 XOR 226
 TEST 226
 SHL 226
 SAL 227
 SHR 227
 SAR 228
 ROR 229
 ROL 229
 RCR 230
 RCL 230
 JMP 231
 JCXZ 231
 JCOND 231
 LOOP 232, 233
 CALL 233
 RETURN 233
 RET 234
 INT, IRET, INTO 236

MOVSB, MOVSW 236
 STOSB, STOSW 236
 LODSB, LODSW 236
 CMPSB, CMPSW 236
 SCASB, SCASW 237
 REPEAT, REP 237
 CLC 238
 CMC 238
 STC 238
 CLD 238
 STD 238
 CLI 238
 STI 239
 HLT 239
 WAIT 239
 LOCK 239
 NOP 239
 TEST 239
 ESC 239

Chapter 7

Machine language 243
 Assembly language 243
 Assembler 244
 Directives 244
 Norton's editor 244
 MASM Editor 245
 LINK 246
 Assembly language
 commands 246
 A Assemble command 246
 U Un-assemble command 247
 R Register command 248
 Status flags 249
 G Go command 250
 T Trace command 251
 D display command 252
 E enter command 253
 F Fill command 254
 M Move command 254
 S Search command 254
 Addition 255-257
 Subtraction 257
 2's complement 258,259
 Multiply 260, 271
 Divide 261, 272

Decimal addition 261-262
 Rotating 264, 265
 Shift left 266
 Largest number 266-268
 Transfer a block of data 269
 ASCII 270, 271
 Descending order 272
 Ascending order 274
 Square of a number 277
 Look-up table 277
 Square root of a number 287
 Addition of matrix 278
 Multiplication of matrix 280
 Gray code 282
 BCD 282
 Factorial 283
 Positive and negative
 number 284
 Even and odd number 286
 Binary to BCD 288
 8087 289
 80287 289
 80387 289
 Numeric Data Processors 289
 Numeric Execution Unit 290
 Registers of 8087 291
 Floating Point Data
 Registers 291
 Registers 292
 Control Word Register 292
 Status register 293
 Tag word 294
 Tag Register 294
 Instruction and Data
 Pointer 294
 Pin Description of 8087 295
 INT 296
 BUSY 296
 RESET 296
 S2, S1 and S0 296
 Interfacing of 8087 with
 8086 297
 Data format of 8087 297
 Instruction Set of 8087 298
 Floating Point Data Transfer
 Instructions 299

- Integer Point Data Transfer
 - Instructions 299
- BCD Data Transfer
 - Instructions 300
- FADD, FADDP 300
- FSUB, FSUBP, FSUBR 300
- FMUL, FMULP 300
- FDIV, FDIVP, FDIVR and
 - FDIVRP 300
- FSQRT 301
- FABS 301
- Comparison Instruction 301
- Transcendental Instruction 301
- FPTAN 301
- FPATAN 301
- FLY2X 301
- FLY2XP1 301
- Co-processor control
 - instructions 301
- FINT 301
- FENI 301
- FDISI 302
- FWAIT 302
- Volume of sphere 302
- 80287 303
- Bus Control Logic 303
- Data Interface and Control
 - Unit 303
- Floating point unit 304
- Status word 304
- B Flag 304
- TOP 304
- ES 305
- SF 305
- Exception Flags 305
 - Control word 305
 - Masking Bits 305
 - Precision control bits 306
 - Rounding control bits 306
 - Infinity control bit 306
 - Pin description of 80287 306
 - 80387 307
- Chapter 8**
- Memory interfacing 313
- Types of memory 313
- ROM 314
- RAM 314
- Static RAM 315
- Dynamic RAM 315
- Memory organisation 316, 318
- Address lines 317
- Data lines 317
- Chip select signal 317
- Read or write 317
- Output enable 317
- Memory map 321
- Address decoding 322
- Memory interfacing to
 - microprocessor 323
- Memory mapped I/O 325,326
- I/O mapped I/O 324,326
- EPROM interfacing with
 - 8086 328, 329
- Timing diagram 330
- Memory read cycle 330
- Memory write cycle 331
- I/O read cycle 331
- I/O write cycle 334
- Interrupts 335
 - data transfer 335
- Interrupt Service Routine
 - (ISR) 335,336
- 8259 interrupt controller
 - Classification of interrupts 336
 - 8085 interrupts 336
 - INTR 336,337,338
 - RST 5.5, RST 6.5, RST
 - 7.5 336,337,338
 - TRAP 337
- Interrupt vector 339
- Vectored interrupts 339
- Non-Vectored Interrupt 340
- Triggering levels 343
- Interrupt instructions 343
- Enable interrupts (EI) 343
- Disable interrupts (DI) 343
- Read Interrupt Mask
 - (RIM) 344, 345
- Set Interrupt Mask (SIM) 344
- Serial Input Data (SID) 344
- Serial Output Data (SOD) 344
- Pending interrupts 348
- Interrupts of 8086 348
- Interrupts of 8088 348
- Software interrupts 349
- Non-maskable interrupts 349
- Internal interrupts 349
- Hardware interrupts 349, 352
- Reset 349
- INT 00H 349
- INT 01H 350
- INT 02H 351
- INT 03H 351
- INT 04H 351
- INTR interrupts 351
- 8259A 352
- Programmable Interrupt
 - Controller (PIC) 355
- Priority of 8086/8088
 - interrupts 353
- Interrupt cycle of
 - 8086/8088 353
- Interrupt operation 354
- Pin diagram of 8259A 356
- IR₀-IR₇ (interrupt
 - requests) 356, 357
- Interrupt Request Register
 - (IRR) 357
- In-Service register (ISR). 358
- Priority resolver 358
- Interrupt mask register
 - (IMR) 358
- Interrupt control logic 358
- Data bus buffer 358
- Read/write control logic 358
- Cascade buffer/comparator 358
- Interrupt sequence 359
- Interfacing of 8259 with
 - 8085 359
- Programming of 8259A 360
- Initialization Command Word 1
 - (ICW₁) 360, 361
- Initialization Command Word 2
 - (ICW₂) 361
- Initialization Command Word 3
 - (ICW₃) 362
- Initialization Command Word 4

- (ICW₄) 362
 - Fully nested mode (FNM) 363
 - Operation command words (OCWs) 364
 - Programming Sequence of 8259 363
 - Operation command words OCW₁, OCW₂ and OCW₃ 364
 - Non-specific EOI command 365
 - Specific EOI Command 365
 - Automatic EOI Mode (AEOI) 365
 - Automatic Rotation Equal Priority 366
 - Rotate on non-specific EOI Command 366
 - Rotate on automatic EOI Mode 367
 - Set priority command 367
 - Rotate on specific EOI command 367
 - Pull Mode 368
 - Special Mask Mode (SMM) 368
 - 8255A 368
 - Architecture of 8255A 369
 - Pin diagram of 8255A 371
 - Group A and Group B controls 372
 - Control word 379
 - Operating modes 373
 - Mode 0 Basic input/output 373, 375
 - Mode 1 Strobed input/output 373, 375
 - Mode 2 Bidirectional 373, 377
 - Input Control Signals 375
 - Input Control Signals 377
 - Bit set/reset (BSR) mode 379
 - Applications of 8255 381
 - Programmable interval timer/counter 382
 - 8253 382
 - Operating modes 388
 - Mode 0 Interrupt on terminal count. 382, 389
 - Mode 1 Programmable one-shot 382, 391
 - Mode 2 Rate generator 382, 392
 - Mode 3 Square wave generator 382, 393
 - Mode 4 Software triggered mode 382, 395
 - Mode 5 Hardware triggered mode 382, 396
 - Pin diagram of 8253 383
 - Read/write logic 384
 - CLK 385
 - GATE 385
 - OUT 385
 - Counter section 385
 - Systems Interface 385
 - Control word register 386
 - Control word format 386
- ## Chapter 9
- Serial communication interface 404
 - Interfacing 404
 - simplex 404
 - half-duplex 404
 - full-duplex 404
 - 8251 404
 - Serial data transfer 404
 - Asynchronous data transfer 405
 - Synchronous data transfer 405
 - Transmitter 407,411
 - Receiver 407, 411
 - Modem control 408
 - Pin diagram of 8251 409
 - Read/Write Control Logic 409
 - TXD (Transmit Data Output) 410,411
 - TxC (Transmitter Clock Input) 410,411
 - TXRDY (Transmitter Ready) 410, 411
 - TXE (Transmitter Empty) 410, 411
 - Control logic and registers 410
 - Modem Control pins 411
 - SYNDET/BD 412
 - 8251 interface with microprocessor 412
 - Operating modes of 8251 412
 - Mode Instruction Control word 413
 - Command Instruction Control word 413, 416
 - Asynchronous mode 413,414,415
 - Synchronous mode 415
 - Instruction format 413
 - Status Word Register Format 417
 - Direct memory access (DMA) 417
 - DMA operation 417, 418,425
 - DMA controller 417,418
 - DMA request (DRQ) 418
 - Hold acknowledge(HLDA) 418, 420
 - Hold request(HRQ) 418, 420
 - Pin diagram of 8257 418
 - Data bus buffer 421
 - DMA address register 421
 - MARK 420
 - READY 420
 - AEN (Address Enable) 420
 - ADSTB (Address Strobe) 420
 - TC(Terminal Count) 420
 - Architecture of 8257 420
 - Read/write logic 421
 - Priority Resolver 425
 - 8257 register selection 422
 - Terminal count register 422
 - Mode set register 423
 - Priority of DMA channels 423
 - Status register 424
 - Data bus buffer 424
 - Read/write logic 424
 - Control unit 424
 - Single byte Data transfer 425

- Reset 425
- Burst mode 426
- Control override mode 426
- Not ready mode 426
- Interfacing 8257 427
- Slave mode operation 427
- Master mode operation 428
- Disk controller using 8257 429
- Programmable key board 430
- 8279 430
- Display I/O interface 430
- Features of 8279 431
- Keyboard section 430
- Display section 430
- Scanned keyboard 431
- Scanned sensor 431
- Pin diagram of 8279 431
- Scan Lines 432
- Blank display
- I/O control and Data buffer 433
- Control and timing registers 433
- Scan Counter 433
- Return buffers and Keyboard debounce and control 434
- Display address registers 434
- Display RAM 434
- Strobed input 434
- Scanned Keyboard 434
- Scanned Sensor matrix 434
- Display scan 435
- Display entry 435
- Display modes 435
- Keyboard modes 434
- Software operation 435
- Program clock 435
- Read Display RAM 436
- Write Display RAM 436
- Display write inhibit/Blanking 436
- Clear display RAM 437
- End Interrupt/Error Mode Set 437
- Data format 437
- Display 438
- Left Entry 438
- Right Entry 438
- Interfacing 8279 439
- Keyboard interface 439
- Sixteen digit display interface with 8279 439
- CRT Controller 441
- 8275 441
- Data Bus Buffer 441
- Character Counter 441
- Line Counter 442, 443
- Row Counter 442
- Light Pen register 442
- Raster Timing and Video control 442
- ROW Buffers 443
- FIFOs 443
- Buffer Input/Output Controllers 443
- DMA Request, DRQ 443
- DMA Acknowledge, DACK 443
- Horizontal Retrace, HRTC 443
- Vertical Retrace, VRTC 443
- Light Pen 443
- Video Suppression, VSP 444
- Reverse Video, RVV 444
- Light Enable 444
- Character Clock, CCLK 444
- Highlight, HLGHT 444
- Attribute Codes 444
- Line Attribute Codes 444
- System operation 444
- Interfacing 8275 with 8257 445
- Display of Characters 445
- Analog to Digital (ADC) conversion 445
- Digital to Analog (DAC) conversion 445, 458
- Counting type ADC 446
- Single slope serial ADC 446
- Successive approximation ADC 447
- Parallel converter 449
- Flash converter 449
- Specification of ADC 449
- Input impedance 449
- Quantisation Error 450
- Accuracy 450, 463
- Resolution 450, 463
- Conversion Time 450
- Temperature Stability 450
- ADC ICs 450
- ADC0800 451
- ADC80 452
- Interfacing of ADC 453, 456
- Binary weighted or $R/2^N$ R DAC 458
- R-2R ladder circuit 460
- Specification of DAC 463
- Linearity 463
- Settling time 464
- Temperature Sensitivity 464
- Interfacing of DAC ICs 464
- DAC0800 465
- Bus interface 467
- System Bus 467
- CPU 468
- Memory 468
- Input and output signals 468
- Bus structure 468
- Data lines 468
- Address lines 468
- Chip set 469
- PCI slots 469
- AGP slots 469
- Mother Board 469
- Cache bus 469, 470
- Memory bus 469, 470
- Expansion Bus 469
- ISA 469, 471
- EISA 469, 472
- MCA 469, 472
- PCI bus 469, 472
- VL bus 469
- USB 469
- SCSI bus 470, 473
- PC card bus 470
- USB bus 470
- Bus architecture 470
- CPU Bus 470
- Processor Bus 470
- Local I/O bus 470

- Standard I/O bus 471
 - Expansion Bus 471
 - VESA 472
 - Single processor system 473
 - AGP 473
 - Parallel Printer Interface 474, 475
 - Daisy chain 474
 - RS-232C 475, 477, 478, 479, 480
 - Data Packet 476
 - TTL interfacing 478
 - DTE, DCE 479
 - RS-422A, RS-423A 479
 - IEEE-488 bus 479, 480
 - Computer network 479
 - 8250 UART 480, 481
 - 16550 481, 485
 - Registers 481
 - Line status registers 482
 - Interrupt Enable register 482
 - Interrupt identification register 483
 - Modem control register 483
 - Modem status register 483
 - Divisor Register 483
 - Baud rate 483
 - Transmitter 484
 - Receiver 484
 - UART application 484
 - FIFO control register 486
 - Line control register 486
 - Line status register 487
 - Programmable baud rate generator 487
 - 8089 I/O processor 487
 - Peripheral devices 488
 - Pin description of 8989 488
 - 8089 architecture 490
 - Register model 492
- ## Chapter 10
- Seven segment display 500
 - Single digit display 502
 - Two digit display 502
 - For digit display 503
- Measurement of voltage 504
 - Half wave precision rectifier 509
 - Full wave precision rectifier 509
 - Measurement of current 509
 - Frequency measurement 511, 514
 - ZCD 513
 - Phase angle measurement 516, 517
 - Power factor measurement 518
 - Impedance measurement 520
 - VA measurement 523
 - Power measurement 526
 - VAR measurement 526
 - Energy measurement 527
 - Displacement measurement 529-534
 - Strain measurement 534-537
 - Force measure 537
 - Torque measurement 539
 - Pressure measurement 541
 - Temperature measurement 541-545
 - Water level measurement 545
 - Measurement and display of speed 547
 - Microprocessor based protection 549
 - Over voltage protection 549
 - Microprocessor based traffic light control 552
 - Microprocessor based firing circuit of a thyristor 558
 - Speed control 562
 - DC motor 562
 - Stepper motor 569
- ## Chapter 11
- 80186 581
 - 80186 microprocessor architecture 581
 - Clock generator 580
 - Bus interface unit 580
 - Programmable timer 580
 - Programmable interrupt controller 580
 - Chip select unit 580
 - Register set 581
 - General Registers 581
 - Base and index registers 581
 - Status and Control Registers 581
 - Status word 582
 - Carry flag 582
 - Parity flag 582
 - Auxiliary Flag 582
 - Sign Flag 582
 - Zero Flag 582
 - Sign Flag 582
 - Single step flag 582
 - Interrupt enable flag 582
 - Direction Flag 582
 - Overflow flag 583
 - Clock generator 583
 - Oscillator 583
 - DMA Channels 583
 - DMA operation 583
 - DMA control 584
 - DMA channel control word register 585
 - DMA control word bit descriptions 585
 - Timers 586
 - Timer operation 587
 - Timer mode/ control register 587
 - Interrupt controller 589
 - Non-iRMX(Master) Mode operation 590
 - Fully Nested Mode 590
 - Cascade Mode 590
 - Special Fully Nested Mode 590
 - Interrupt Controller Registers 590
 - iRMX 86 mode 591
 - Interrupt control register 591
 - Memory organization 592
 - Memory chip select 592
 - Peripheral chip selects 593
 - Pin description of 80186 594
 - RESET 594

- CLKOUT 594
- TEST 595
- TMR IN0, TMR IN1 595
- TMR OUT0, TMR OUT1 595
- ARDY 596
- SRDY 596
- Addressing modes of
 - 80186 598
- Register operand addressing mode 598
- Immediate operand addressing mode 598
- Direct addressing mode 598
- Register indirect addressing mode 598
- Based addressing mode 598
- Indexed addressing mode 598
- Based indexed addressing mode 598
- Based indexed with displacement addressing mode 598
- Data types of 80186 599
- Instruction set of 80186 599
- Data transfer Instructions 599
- Arithmetic instructions 599
- Logical instructions 600
- String Instructions 600
- PUSHA 599
- POPA 599
- PUSH immediate 599
- IMUL 599
- INS 600
- OUTS 600
- BOUND 600
- ENTER 600
- Comparison between 8086 and 80186 601
- 80286 601
- Architecture of 80286 602
- Address unit 603
- Bus unit 603
- Instruction unit 603
- Execution Unit 604
- General-purpose registers 604
- Segment registers 605
- Base and Index registers 605
- Status and control registers 605
- Flags word description 605
- Flag registers 605
- Task switch(TS) 605
- Machine status word 606
- Pin diagram of 80286 606
- PEREQ 608
- PEACK 608
- BUSY 609
- ERROR 609
- CAP 609
- RESET 609
- Addressing modes of 80286 609
- Data types of 80286 610
- 80286 instruction set 610
- ARPL 611
- CLTS 611
- LAR 611
- LGDT 611
- LIDT 611
- LLDT 612
- LMSW 612
- LSL 612
- LTR 612
- SGDT 612
- SIDT 612
- SMSW 612
- STR 612
- VERR 612
- VERW 612
- ENTER 613
- LEAVE 613
- BOUND 613
- LAHF 613
- PUSH IMD 613
- PUSH A 613
- POP A 613
- IMUL 613
- Rotate source, count 614
- INS 614
- OUTS 614
- 80286 addressing modes 614
- Real addressing modes 614
- Protected virtual Address mode(PVAM) 615
- Physical address calculations 615
- Descriptors 616
- Code and Data Segment Descriptors 616
- System Segment Descriptors 616
- Gate Descriptors 617
- Segment Descriptor Cache Registers 618
- Selector fields 618
- Local and Global Descriptor Tables 618
- GDT 618
- LDT 618
- LGDT 619
- LLDT 619
- Interrupt description table(IDT) 619, 620
- Privilege 619
- Four level privilege 620
- Comparison 8086 and 80286 620
- Comparison 80186 and 80286 621
- 80386 622
- Architecture of 80386 623
- CPU 623
- Memory management unit 623
- Segmentation unit 623
- Paging unit 623
- Bus interface unit 624
- Registers of 80386 624
- 32-bit EIP 625
- Stack Segment and Stack Pointer 625
- General purpose registers 625
- Segment registers 625
- Control register 626
- Debugging registers 626
- Test Registers 626
- System Address Registers 626
- Flag Registers 626
- RF (Resume Flag) 627
- VM(Virtual Mode Flag) 628
- Pin description of 80386 628

- Addressing modes of
 - 80386 630
 - Scaled indexed mode 630
 - Based scaled indexed mode 630
 - Based scaled indexed mode with displacement 630
 - Data type of 80386 631
 - Operating modes of 80386 631
 - Real addressing mode 631
 - Protected mode addressing 632
 - Memory management 633
 - Segmentation 633
 - Descriptors 634
 - Paging 635
 - Paging operation 635, 637
 - Paging unit 635
 - Page descriptor base register 635
 - Page directory 635
 - Page table 636
 - Linear address 636
 - Physical address 636
 - Virtual 8086 mode 637
 - Instruction set of 80386 638
 - Bit scan instructions 638
 - Bit test instructions 638
 - Conditional set byte instructions 638
 - Shift double instructions 639
 - Control transfer instructions 639
 - Comparison 80286 and 80386 639
 - 80486 640
 - Architecture of 80486 640
 - Bus interface unit 641
 - Execution unit 641
 - Control unit 641
 - Floating point unit(FPU) 641
 - Registers of 80486 643
 - Pin description of 80486 643
 - Data Bus 644
 - Data parity group 644
 - Bus cycle definition group 645
 - Burst Control group 646
 - Bus Arbitration Group 646
 - Interrupts 646
 - Cache Invalidation group 646
 - Page cache ability group 647
 - Bus size control group 647
 - FPU Error group 647
 - Test Access Port Group 647
 - Comparison 80386 and 80486 648
- ## Chapter 12
- Pentium 653
 - RISC processors 653
 - Code and data cache 653
 - Pentium architecture 653
 - Branch prediction 654
 - Branch trace buffer 654
 - Control unit 654
 - Integer pipelines U and V 655
 - Prefetch(PF) 655, 656
 - Decode-1(D1) 655, 656
 - Decode-2(D2) 655, 657
 - Execute(E) 656
 - Write back 656
 - Superscalar 656
 - Floating –Point unit 656
 - Operand fetch 657
 - Floating point
 - execute-1(X1) 657
 - Floating point
 - execute-2(X2) 657
 - Write Float(WF) 657
 - Error Reporting(ER) 657
 - FADD 658
 - FMUL 658
 - FDIV 658
 - FEXP 658
 - FRD 658
 - Floating point exceptions 658
 - Instruction pairing 659
 - Pentium register set 660
 - Registers of Pentium
 - processor 661
 - Pentium operating modes 662
 - Protected mode 662
 - Real address mode 662
 - System management
 - mode(SMM) 662
 - Virtual -8086 mode 662
 - Real mode 662
 - Protected mode 663
 - Memory management of Pentium 663
 - Segmentation 663, 665
 - Paging 663
 - Basic flat model 665
 - Protected flat model 666
 - Multi segment model 666
 - Physical address 666
 - Linear address 666, 667
 - Logical address 666, 667
 - Segment registers 668
 - Segment selectors 668
 - Segment descriptors 668
 - Global and Local Descriptor tables 669
 - Cache registers 669
 - Interrupts in Protected Mode 669
 - Task stat segment(TSS) 670
 - Virtual 8086 mode 670
 - Page tables and directories 671
 - Translation look-aside buffers 672
 - Pin description of Pentium 673
 - Addressing modes of Pentium 676
 - Register mode 676
 - Register direct mode 676
 - Immediate mode 676
 - Direct mode 676
 - Base displacement mode 676
 - PC relative mode 676
 - Pentium bus interfacing 677
 - Single transfer cycles 677
 - Burst cycles 679
 - Burst read cycles 680
 - Burst read cycles 681
 - Pentium address pipelining 681
 - Special cycles 682
 - Inquiry cycles 683
 - System management mode 684
 - Dual processing 684

- Bus arbitration 684
- On chip advanced programmable interrupt controllers 685
- Performance of Pentium 686
- Cache memories 686
- SRM cache 686
- Write through strategy 688
- Write back strategy 688
- Write allocate 689
- Cache organization 689
- Direct mapped cache 689
- Two-way set associative cache 690
- Cache consistency 691
- Bus snooping 692
- MESI 692
- L2 caches 694
- MESI protocol 694
- Pentium on chip caches 695
- Physical address 696
- Cache operating modes 696
- Page cache ability 696
- Pentium L2 cache 697
- Pentium MMX 697
- Pentium pro 698
- Pentium II 698
- Pentium III 699
- Architecture of P6 699
- Bus interface unit 700
- L1 data cache 700
- L1 code cache 700
- Instruction fetch and decoder 700
- Branch target buffer 701
- Internal registers 701
- Reservation station 701
- Retire unit 701
- Instruction pool 701
- Instruction fetch 701
- Decode unit 701
- Dispatch unit 702
- Execution unit 702
- Instruction pool 702
- Retire unit 702
- P6 family processors 702
- 36-bit address bus 702
- Comparison Pentium and Pentium-pro 704
- Pentium 4 processors 705
- Architecture of Pentium 4 705
- Bus interface unit 706
- Instruction decoder 706
- Trace cache 706
- Microcode ROM 706
- Branch Prediction 706
- ITLB 707
- Execution unit 707
- Allocator 708
- Register rename 708
- Instruction schedulers 708
- Rapid Execution Module 708
- Memory Subsystem 708
- Hyper-threading technology 708, 711
- Time sliced multi-threading 710
- Switch on event multi-threading 710
- Simultaneous multi-threading 710
- Architecture state(AS) 710
- Replicated resources 710, 711
- Shared resources 710, 711
- Streaming SIMD extension (SSE) 712
- SSE2, SSE3 712
- Comparison Pentium III and Pentium 4 713
- RISC processors 713, 714
- CISC 713
- Comparison RISC and CISC 715
- Power PC 601 715
- MIPS 717
- Sun Ultra Sparc 717
- Core processor 717
- Core 2 dual processor 718
- Chapter 13**
- Microcontroller 722, 723
- Microprocessor 723
- Microcontroller applications 724
- 4-bit Microcontroller 723, 724
- 8-bit Microcontroller 723, 724
- 16-bit Microcontroller 723, 724
- 8051 724, 725
- Features of 8051 725
- Accumulator ACC 726
- B register 726
- Program status word PSW 726, 727
- Stack pointer(SP) 728
- Data pointer(DPTR) 728
- Port 0, Port 1, Port 2, Port 3 728
- Serial port data buffer 729
- Timing registers 729
- Control registers 729
- Capture registers 729
- Timing and control unit 729
- Oscillator 729
- Instruction register 730
- Program address register 730
- ALU 730
- SFR (Special function registers) 730
- Memory organisation 730
- Data memory 731, 732
- Program memory 730, 731
- Register Banks 0-3 733
- Bit addressable area 733
- Scratch pad area 734
- RST 376
- ALE Address latch enable 376
- Pin diagram of 8051 375
- External Access 739
- Program Store Enable 739
- XTAL1 , XTAL2 739
- Instruction cycle 739
- Timers 744
- Counters 744
- Oscillator frequency 744
- Operating modes 745
- Timer mode 0 745
- Timer mode 1 746
- Timer mode 2 746
- Timer mode 3 747
- Control registers 747

TCON register 748
 TCON register 748
 Timer operating modes 749
 Serial communication 749
 Transmitter half 749
 Receiver half 750
 UART transmitter 750
 UART receiver 751
 MAX232 751
 SBUF 752
 SCON 752
 PCON 752
 Serial communication
 modes 753
 Multiprocessor
 communication 755
 Interrupts in 8051 756
 Interrupt control register 757
 Interrupt Enable Register 758
 Execution of Interrupts 759

Chapter 14

Addressing modes 763
 Immediate addressing 764
 Register addressing 764
 Direct addressing 765
 Indirect addressing 765
 Indexed addressing 766
 Relative addressing 766
 Absolute addressing 767
 Long addressing 767
 8051 Instruction set 767
 Arithmetic instructions 769
 Logical instructions 774
 Data transfer instructions 780
 Boolean instructions 784

Branch group instructions 787
 PUSH, POP instructions 791
 Exchange instructions 791
 ADD 769
 ADDC 769, 770
 SUBB 770
 INC 771
 DEC 772
 MUL 773
 DIV 773
 DAA 773
 ANL 774
 ORL 775
 XRL 776
 CLR 778
 CPL 778
 RL 778
 RLC 778
 RR 779
 RRC 779
 SWAP 779
 MOV 780
 MOVC 783
 MOVX 783
 CLR 784
 SETB 785
 CPL 785
 ANL 785
 ORL 786
 JC 786
 JNC 787
 JB 787
 JNB 787
 ACALL 787
 LCALL 788
 RET 788

RETI 788
 AJMP 788
 LJMP 789
 SJMP 789
 JMP 789
 JZ 789
 JNZ 790
 CJNE 790
 DJNZ 791
 NOP 791
 PUSH 791
 POP 791
 XCH 792
 XCHD 792
 Addition 795-798
 Decimal addition 799
 ASCII 800
 Subtraction 800
 One's complement 800
 Two's complement 801
 Shift 801
 Swap 801
 Largest 802
 Smallest 803
 Descending order 804
 Ascending order 804
 Square 805
 Multiplication 806
 Division 806
 Time delay 807
 Key board interfacing 808
 A/D converter interfacing 810
 Traffic control 811
 Stepper motor control 817
 Washing machine control 820