

S  
O  
L  
U  
T  
I  
O  
N  
S  
C  
O  
M  
P  
A  
C  
T



# ADVANCED MICROPROCESSORS AND PERIPHERALS

**A K RAY | KM BHURCHANDI**

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.



**Tata McGraw-Hill**

Copyright © 2006, 2000 by Tata McGraw-Hill Publishing Company Limited.

Eleventh reprint 2009

RQXLCDDXRQLQD

No part of this publication may 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 Publishing Company Limited.

**ISBN-13: 978-0-07-014062-2**

**ISBN-10: 0-07-014062-6**

Published by Tata McGraw-Hill Publishing Company Limited,  
7 West Patel Nagar, New Delhi 110 008, typeset at Script Makers,  
19, A1-B, DDA Market, Pashchim Vihar, New Delhi 110 063  
and printed at Pashupati Printers Pvt. Ltd., Delhi 110 095

# Contents

<i>Preface to the Second Edition</i>	xiii
<i>Preface to the First Edition</i>	xv
<i>Acknowledgements</i>	xix
<b>1. The Processors: 8086/8088—Architectures, Pin Diagrams and Timing Diagrams</b>	1
1.1 Register Organisation of 8086	2
1.2 Architecture	3
1.3 Signal Descriptions of 8086	8
1.4 Physical Memory Organisation	14
1.5 General Bus Operation	16
1.6 I/O Addressing Capability	17
1.7 Special Processor Activities	18
1.8 Minimum Mode 8086 System and Timings	21
1.9 Maximum Mode 8086 System and Timings	25
1.10 The Processor 8088	28
Summary	36
Exercises	36
<b>2. 8086/8088 Instruction Set and Assembler Directives</b>	38
2.1 Machine Language Instruction Formats	38
2.2 Addressing Modes of 8086	41
2.3 Instruction Set of 8086/8088	46
2.4 Assembler Directives and Operators	74
Summary	82
Exercises	83
<b>3. The Art of Assembly Language Programming with 8086/8088</b>	84
3.1 A Few Machine Level Programs	85
3.2 Machine Coding the Programs	91
3.3 Programming with an Assembler	95
3.4 Assembly Language Example Programs	103
Summary	129
Exercises	129

**4. Special Architectural Features and Related Programming****131**

- 4.1 Introduction to Stack 131**
- 4.2 Stack structure of 8086/88 133**
- 4.3 Interrupts and Interrupt Service Routines 138**
- 4.4 Interrupt Cycle of 8086/8088 138**
- 4.5 Non Maskable Interrupt 141**
- 4.6 Maskable Interrupt (INTR) 141**
- 4.7 Interrupt Programming 142**
- 4.8 Passing Parameters to Procedures 145**
- 4.9 Handling Programs of Size More than 64K 148**
- 4.10 MACROS 150**
- 4.11 Timings and Delays 152**
  - Summary 155**
  - Exercises 155**

**5. Basic Peripherals and Their Interfacing with 8086/88****157**

- 5.1 Semiconductor Memory Interfacing 158**
- 5.2 Dynamic RAM Interfacing 167**
- 5.3 Interfacing I/O Ports 173**
- 5.4 PIO 8255 [Programmable Input-Output Port] 184**
- 5.5 Modes of Operation of 8255 187**
- 5.6 Interfacing Analog to Digital Data Converters 212**
- 5.7 Interfacing Digital to Analog Converters 224**
- 5.8 Stepper Motor Interfacing 228**
- 5.9 Control of High Power Devices Using 8255 231**
  - Summary 232**
  - Exercises 233**

**6. Special Purpose Programmable Peripheral Devices and Their Interfacing****235**

- 6.1 Programmable Interval Timer 8253 235**
- 6.2 Programmable Interrupt Controller 8259A 249**
- 6.3 The Keyboard/Display Controller 8279 266**
- 6.4 Programmable Communication Interface 8251 USART 278**
  - Summary 290**
  - Exercises 290**

**7. DMA, Floppy Disk and CRT Controllers****293**

- 7.1 DMA Controller 8257 294**
- 7.2 DMA Transfers and Operations 300**
- 7.3 Programmable DMA Interface 8237 306**

---

7.4 Floppy Disk Controller 8272 318

7.5 CRT Controller 8275 350

7.6 CRT Controller 6845 368

*Summary* 389

*Exercises* 389

---

## 8. Multimicroprocessor Systems

392

8.1 Interconnection Topologies 393

8.2 Software Aspects of Multimicroprocessor Systems 397

8.3 Numeric Processor 8087 399

8.4 I/O Processor 8089 420

8.5 Bus Arbitration and Control 423

8.6 Tightly Coupled and Loosely Coupled Systems 428

8.7 Design of a PC Based Multimicroprocessor System 430

*Summary* 442

*Exercises* 442

---

## 9. 80286-80287—A Microprocessor with Memory

### Management and Protection

444

9.1 Salient Features of 80286 444

9.2 Internal Architecture of 80286 446

9.3 Signal Descriptions of 80286 451

9.4 Real Addressing Mode 454

9.5 Protected Virtual Address Mode (PVAM) 455

9.6 Privilege 463

9.7 Protection 468

9.8 Special Operations 470

9.9 80286 Bus Interface 473

9.10 Basic Bus Operations 474

9.11 Fetch Cycles of 80286 475

9.12 80286 Minimum System Configuration 476

9.13 Interfacing Memory and I/O Devices with 80286 478

9.14 Priority of Bus Use by 80286 481

9.15 Bus Hold and HLDA Sequence 484

9.16 Interrupt Acknowledge Sequence 485

9.17 Instruction Set Features 486

9.18 80287 Math Coprocessor 492

*Summary* 503

*Exercises* 503

---

## 10. 80386-80387 and 80486 the 32-Bit Processors

505

10.1 Salient Features of 80386DX 506

10.2 Architecture and Signal Descriptions of 80386 506

<b>10.3 Register Organization of 80386</b>	<b>510</b>
10.4 Addressing Modes	512
10.5 Data Types of 80386	514
10.6 Real Address Mode of 80386	514
10.7 Protected Mode of 80386	515
10.8 Segmentation	516
<b>10.9 Paging</b>	<b>518</b>
10.10 Virtual 8086 Mode	520
10.11 Enhanced Instruction Set of 80386	523
10.12 The Coprocessor 80387	524
10.13 The CPU with a Numeric Coprocessor—80486DX	528
<i>Summary</i>	539
<i>Exercises</i>	540
<b>11. Recent Advances in Microprocessor Architectures— A Journey from Pentium Onwards</b>	<b>542</b>
11.1 Salient Features of 80586 (Pentium)	543
11.2 A Few Relevant Concepts of Computer Architecture	544
11.3 System Architecture	544
<b>11.4 Branch Prediction</b>	<b>548</b>
<b>11.5 Enhanced Instruction Set of Pentium</b>	<b>548</b>
11.6 What is MMX?	549
11.7 Intel MMX Architecture	549
11.8 MMX Data Types	550
11.9 Wraparound and Saturation Arithmetic	550
11.10 MMX Instruction Set	550
<b>11.11 Salient Points About Multimedia Application Programming</b>	<b>552</b>
<b>11.12 Journey to Pentium-Pro and Pentium-II</b>	<b>552</b>
11.13 Pentium III (P-III)—The CPU of the Next Millennium	554
<i>Summary</i>	554
<i>Exercises</i>	555
<b>12. Pentium 4—Processor of the New Millennium</b>	<b>556</b>
<b>12.1 Genesis of Birth of Pentium 4</b>	<b>556</b>
<b>12.2 Salient Features of Pentium 4</b>	<b>557</b>
<b>12.3 Netburst Microarchitecture for Pentium 4</b>	<b>558</b>
12.4 Instruction Translation Lookaside Buffer (ITLB) and Branch Prediction	562
12.5 Why Out of Order Execution	562
12.6 Rapid Execution Module	564
12.7 Memory Subsystem	564
12.8 Hyperthreading Technology	565
12.9 Hyperthreading in Pentium	566

12.10 Extended Instruction Set in Advanced Pentium Processors	568
12.11 Instruction Set Summary	573
12.12 Need for Formal Verification	584
<i>Summary</i>	584
<i>Exercises</i>	584
<b>13. RISC Architecture — An Overview</b>	<b>585</b>
13.1 A Short History of RISC Processors	586
13.2 Hybrid Architecture—RISC and CISC Convergence	586
13.3 The Advantages of RISC	587
13.4 Basic Features of RISC Processors	587
13.5 Design Issues of RISC Processors	588
13.6 Performance Issues in Pipelined Systems	589
13.7 Architecture of Some RISC Processors	591
13.8 Architecture of Some RISC Processors	596
<i>Summary</i>	596
<i>Exercises</i>	596
<b>14. Microprocessor Based Aluminium Smelter Control</b>	<b>598</b>
14.1 General Process Description of an Aluminium Smelter	598
14.2 Normal Control of Electrolysis Cell	599
14.3 Cell Abnormalities in an Aluminium Smelter	600
14.4 Brief Description of the Control Laws for Abnormal Cells	601
14.5 Salient Issues in Design	602
14.6 Smelter Controller Hardware	602
14.7 Control Algorithm	603
<i>Summary</i>	607
<b>15. Design of a Microprocessor Based Pattern Scanner System</b>	<b>608</b>
15.1 Organization of the Scanner System	609
15.2 Description of the Scanning System	611
15.3 Programmed Mode of Operation	613
15.4 Memory Read/Write System and Start-up Procedures	615
15.5 Result and Discussion	616
<i>Summary</i>	618
<b>16. Design of an Electronic Weighing Bridge</b>	<b>619</b>
16.1 Design Issues	620
16.2 Software Development	635
16.3 Calibration	645
<i>Summary</i>	646

<b>17. An Introduction to Microcontrollers 8051 and 80196</b>	<b>647</b>
<b>17.1 Intel's Family of 8-bit Microcontrollers</b>	<b>649</b>
<b>17.2 Architecture of 8051</b>	<b>649</b>
17.3 Signal Descriptions of 8051	652
17.4 Register Set of 8051	654
17.5 Important Operational Features of 8051	655
17.6 Memory and I/O Addressing by 8051	658
17.7 Interrupts of 8051	661
17.8 Instruction Set of 8051	662
17.9 Design of a Microcontroller 8051 Based Length Measurement System for Continuously Rolling Cloth or Paper	665
17.10 Intel's 16-bit Microcontroller Family MCS-96	668
<i>Summary</i>	678
<i>Exercises</i>	679
<i>Appendix A</i>	681
<i>Appendix B</i>	691
<i>Appendix C</i>	707
<b>Index</b>	<b>717</b>

# Preface to the Second Edition

Since the publication of the first edition of this book in 2000, a number of advanced microprocessors, such as Pentium 4, based on Net Burst micro-architecture have arrived on the scene. The recent advancements in the architecture of microprocessors have been quite extensive and the undergraduate and graduate students from diverse engineering as well as science disciplines have become keenly interested in the subject.

The first edition of the book has been extensively used by the students and teachers of diverse disciplines from all over India and also from abroad. We have received a number of valuable suggestions and comments towards the enhancement of the quality of the book. These suggestions along with our personal evaluation of the book primarily formed the basis for the revision of the contents and presentation of this edition.

Two new chapters have been introduced in this edition, while one chapter has been removed. The chapter on 'Pentium 4 – Processor of the new millennium' presents the architectural details of Pentium 4. A set of novel concepts like super scalar execution, execution trace cache, rapid execution engine, streaming SIMD Extension and so on have been presented in this chapter. With the advent of a host of processors, based on hybrid CISC-RISC architecture, it has become imperative for the students to understand the basic features and design issues of RISC processors. Keeping in view of this requirement we have introduced a new chapter 'RISC Architecture – An overview'. The chapter on 'System Software and Operating system' has been deleted from this edition. This chapter may however, be accessed through the Online Learning Centre version of the book. The rest of the chapters have been appropriately modified and recast, keeping in view the needs of the readers.

We hope that like the first edition, this edition of the book will also be immensely appreciated by the readers.

AJOY KUMAR RAY  
KISHOR M BHURCHANDI

# Preface to the First Edition

With the advent of the first 4-bit microprocessor 4004 from Intel Corporation in 1971, there has been a silent revolution in the domain of digital system design, which has shaken many facets of the current technological progress. In the last 28 years the world has seen an evolution of microprocessors, whose impact on today's technological scenario is phenomenal.

This evolution was possible because of the tremendous advances in the semiconductor process technology. The first microprocessor 4004 contained only ten thousand transistors while the component density increased more than threefold in less than a decade's time. Immediately after the introduction of the 4004, Intel introduced the first eight bit microprocessor 8008 in 1972; these processors were, however, not successful because of their inherent limitations. In 1974, Intel released the first general purpose 8-bit microprocessor 8080. This CPU also was not functionally complete and the first 8-bit functionally complete CPU 8085 was introduced in 1977.

The 8085 CPU is still the most popular one amongst all the 8-bit CPUs. The 8085 CPU houses an on-chip clock generator and provides good performance utilizing an optimum set of registers and a reasonably powerful ALU. The major limitations of these 8-bit microprocessors are their limited memory addressing capacity, slow speed of execution, limited number of scratchpad registers and non-availability of complex instruction set and addressing modes. Another important point to be mentioned here is that 8085 does not support adequate pipelining or parallelism which is so important for enhancing the speed of computation. For example, the non-availability of any instruction queue in an 8085 CPU leads to a situation where the fetching of opcode and operands along with the execution takes place in an absolutely sequential manner.

The first 16-bit CPU from Intel was a result of the designers' efforts to produce a more powerful and efficient computing machine. The designers of 8086 CPU had taken note of the major limitations of the previous generations of the 8-bit CPUs. The 8086 contains a set of 16-bit general purpose registers, supports a 16-bit ALU, a rich instruction set and provides segmented memory addressing scheme. The introduction of a set of segment registers for addressing the segmented memory in 8086 was indeed a major step in the process of evolution. All these features made this 16-bit processor a more efficient CPU.

The development of IBM PC started in July 1980, and precisely after one year, the first machine based on Intel 8088 CPU (which is functionally equivalent to 8086 but supports only 8-bit external data bus) with 1 or 2 floppy disk drives, a keyboard and a monochrome monitor was announced in August 1981. The machine operating system was an early version of operating system MS-DOS from Microsoft. In March 1983, a new version of IBM PC called PC-XT was introduced with ten megabyte hard disk, one double side double density floppy disk drive, keyboard, monitor and asynchronous communication adapter. In fact, the introduction of IBM PCs in 1980s had, to a large extent, produced a profound impact on the evolution of microprocessors. With the introduction of each new generation of microprocessors, the performance of the Personal Computers have also been enriched.

The major limitation in 8086 was that it did not have the memory management and protection capabilities, which was considered an extremely important feature deemed to be an integral part of a

CPU of the eighties. 80286 was the first CPU to possess the ability of memory management, privilege and protection. However, the 80286 CPU also had a limitation on the maximum segment size supported by it (only 64 Kb). Another limitation of 80286 was that, once it was switched into protected mode, it was difficult to get it back to real mode. The only way of reverting it to the real mode was to reset the system.

In the mid eighties the more computationally demanding problems necessitated the development of still faster CPUs. Thus appeared 80386 which was the first 32-bit CPU from Intel. The memory management capability of 80286 was enhanced to support virtual memory, paging and four levels of protection. The design of 80386 circumvented this problem. Moreover, the maximum segment size in 80386 was enhanced and this could be as large as 4 Gb with 80386 supporting as many as 16384 segments. The 80386 along with its math coprocessor 80387, provided a high speed environment. 80486 was designed with an integrated math coprocessor. After getting integrated, the speed of execution of mathematical operations enhanced three folds. Also for the first time an 8 Kb four-way set associative code and data cache was introduced in 80486. A five-stage instruction pipelining was also introduced.

The earlier generation CPUs supported rather crude instruction sets. It was not expected that the programmers those days would write large machine code programs. A single high level instruction might be compiled into ten or even hundred machine code operations. In the course of evolution from the early 8-bit CPUs, the trend was to design CPUs, which could support more and more complex instructions at the assembly language level. Designers of complex instruction set computers (CISC) wanted to reduce this gap.

Since the early days of microprocessor development the designers have tried to make them more powerful by designing more complex instructions. But then some of these powerful instructions and addressing modes were hardly used by the programmers. In fact some of these instructions' logic took up a large part of the microprocessors' silicon chip. The reduced instruction set computer (RISC) designers observed that the data movement type of machine instructions are frequently executed by the CPU. They have optimised the CPUs to execute these instructions rapidly. RISC provided a regular set of instructions having the same format with a lot of pipelining. To improve the processor's performance, the possible ways are suggested below.

- (a) Increasing the processor and system clock rate.
- (b) Optimising and improving the instruction set.
- (c) Executing multiple instructions in one cycle and incorporating parallelism in the CPU architecture.

The first option is applicable both to CISC and RISC processors. The second option is primarily for CISC but is applicable to RISC as well. The third option is more suited to RISC CPUs. Ever since the appearance of commercially available RISC CPUs, there has been a debate over the performance of RISC versus CISC. The RISC architects argue that their instructions may be executed in a single cycle and thus take less time than is taken by a CISC CPU. This is because of pipelining, reduction of instructions to a simple operation and synthesis of complex operations with compiler generated code sequences. When RISC machines first arrived in the market, CISC processors were performing at 6–10 cycles per instruction, while the RISC CPUs could execute a set of simpler instructions in one cycle and offer better performance. Many of the CISC processors have subsequently used many features of RISC.

This book is intended as a textbook on 'Advanced Microprocessors' which is a compulsory course at graduate and postgraduate levels in many science and engineering branches of studies, specially in

Electronics, Electrical, Instrumentation, Physics and Computer Science disciplines. The book is suitable for a one-semester course on advanced microprocessor—their architectures, programming, hardware interfacing and applications. The purpose of our book is to provide the readers with a good foundation on microprocessors, their principles and practices. We have tried to keep an appropriate balance between the basic concepts and practical applications related to microprocessors technology. Thus we have aimed at the following:

- To present the fundamental concepts of advanced microprocessors and their architectures.
- To enable the students to write efficient programs in assembly level language of the 8086 family of microprocessors.
- To make the students aware of the techniques of interfacing between the processors and peripheral devices so that they themselves can design and develop a complete microprocessor based system.
- To present in a lucid manner the basic concepts of systems programming, viz, operating systems, assemblers, compilers, etc. to enable the students to understand the entire space of microprocessor technology and specially the software aspects related to microprocessing.
- To present to the students the utility of faster modes of data transfer and techniques.
- To present a host of interesting applications involving microprocessors.

Some of the salient features of the book are listed below.

1. The book covers a wide range of microprocessors from 16-bit 8086 to Pentium in a lucid manner. The evolution from one processor architecture to another is evident as one goes through the chapters. A detailed description of each microprocessor has been presented in individual Chapters. Chapter 1 covers 8086/8088 architecture in adequate detail. Chapter 9 covers 80286 along with its coprocessor. Chapter 10 covers the microprocessor 80386 and its coprocessor 80387. This chapter also covers in sufficient detail 80486, the integrated CPU with built-in math coprocessor. Pentium, the latest in the Intel microprocessor family, has been briefly presented in Chapter 11.
2. An important feature of the book is the inclusion of a number of interesting applications of microprocessors. An adequate account of each one of these applications has been presented in the book. An interesting application of microprocessors for controlling an Aluminium Smelter has been presented in Chapter 13. Chapter 14 presents another interesting application in the area of Pattern Scanner Design. Design of a microprocessor-based Electronic Weighing Bridge has been elaborated in Chapter 15.
3. One of the major problems encountered by students is the difficulty in writing assembly language programs. In this book, a large number of assembly language programs have been presented. Which will enable the students to write efficient codes on 16-bit or 32-bit platforms. Chapter 2 covers the 8086 family instruction set and the assembler directives with necessary examples. The art of programming in 8086 Assembly language has been elaborated with a large number of program examples in Chapter 3. A very important spectrum of programs involving stacks, subroutines, interrupts, macros and time delays has been discussed in adequate detail in Chapter 4.
4. A good account of a number of general peripheral devices like I/O ports, keyboards, displays, ADCs, DACs, stepper motors, etc. has been elaborated in Chapter 5.

5. Some special dedicated peripherals like interrupt controllers, DMA controllers, CRT controllers, floppy disk controllers, etc. have been discussed elaborately along with interfacing examples and programs in Chapters 6 and 7. The detailed knowledge about these peripherals is extremely important for interfacing these devices with advanced CPUs and also for designing standalone microprocessor-based systems.
6. Usually the students look for separate books for understanding the system programs' concepts. In this book a complete chapter (Chapter 12) has been devoted to explaining the concepts of assemblers, loaders, linkers, compilers and the operating systems. The understanding of systems programming concepts are extremely important for an integrated understanding of a microcomputer system.
7. The importance of multiprocessor based system design cannot be underestimated in today's world. A full chapter has been devoted to presenting issues related to the multiprocessor based system design. The co-processor like 8087, 8089, etc. along with their interfacing strategies have been presented in Chapter 8 of the book. Design of an 8088 based multimicroprocessor system has been described in adequate detail in this chapter with an example.
8. Microcontrollers are being extensively used in today's industrial environments. An introductory chapter on microcontrollers has been included for the benefit of the students. Intel's 8-bit microcontroller 8051 and 16-bit microcontroller 80196 have been presented in sufficient detail in Chapter 16 along with an 8031 based application design example.  
On the whole, the book is intended to provide adequate support to the reader for acquiring an integrated understanding of the subject.

AJOY KUMAR RAY  
KISHOR M BHURCHANDI

# Acknowledgements

A number of individuals have contributed to the preparation of the manuscript of this book. The authors gratefully acknowledge the contributions of each one of them. The authors would like to express their thanks to the faculty members of IIT Kharagpur and Shri Ramdeobaba Kamla Nehru Engineering College, Nagpur. The authors convey their gratitude to the large number of teachers and students belonging to many science and engineering institutions from diverse parts of India and abroad for their keen interest in this book, which has prompted the authors to undertake the publication of the second edition of the book.

In particular, Prof. G S Sanyal, former Director and Advisor, IIT Kharagpur, Prof. A K Majumdar, Head, Computer Science and Engineering Department IIT Kharagpur, Dr. Tinku Acharya, CTO, Avisere Inc., USA have always shown their active interest in the publication of the book. We are thankful to them. The authors are thankful to Intel Corporation, USA for allowing them to use some of their product specifications for the book.

Some faculty members and students of IIT Kharagpur have contributed in various ways for the completion of this work. The authors would like to acknowledge the services rendered by them. In particular we thank Rajat Sethi of Computer Science and Engineering, who has contributed in shaping the chapter on Pentium 4. Anoop C V, Rahul Gupta, U V Srinadh, Maj. Nilesh Ingle, Maj. Vivek Vaidyanathan of Electronics & ECE department, IIT Kharagpur have reviewed some of the chapters in the second edition and we thank them all.

We thank Prof. D Sarkar, A K Ghosh, Ananda Mitra and. S S Biswas for providing valuable inputs for the chapter on Microprocessor Based Aluminium Smelter Design, Prof B P Sandilya, Tamalika Chakraborty, D S Deshpande, Amit Chatterji, S Verma, S Sukumar, Mainak Chowdhary, Rajat Subhra Mukherji, Arnab Chakraborty, Abha Jain, Smarajit Mishra, Animesh Khemka and S Dihidar deserve special mention for going through parts of the manuscript. We acknowledge the help rendered by Kaushik Mallick Arumoy Mukherjee, Arindam Mandal, Rana Pratap Ghoshal and many others for their help in preparation of the manuscript.

The authors are thankful to the Principal Dr S S Limaye and Management of Shri Ramdeobaba Kamla Nehru Engineering College, Nagpur for their appreciation and encouragement.

We also thank Dr P M Navghare, former Professor VNIT, Nagpur and Professor, Department of Engineering, University of Eritrea, South Africa for his review and valuable suggestions about organization and scope of this book.

Our thanks are also due to Prof. R V Kshirsagar, Chairman Electronics Engineering Board of Studies, Nagpur University and Prof. D P Dave, Head IT Dept, Tolani Maritime Institute, Pune for their suggestions towards finalization of this text. We also thank our colleagues Professors P T Karule, A S Khobragade, A G Kothari, A V Gokhale, P R Rothe, S Y Ambatkar and P M More for sharing their concepts and ideas during the preparation of the manuscript. We are grateful to Professors K K Bhoyar, S J Karale, Ms Padma Rao for compilation of the chapter on System Software and Operating Systems which is now going to be on the web site of the book.

We would like to express our thanks and appreciation to Professors R S Pande and Ms R R Harkare who have shown keen interest and encouragement for the publication of the second edition. We take this opportunity to express our gratitude to Prof M. Wasim Khanooni and Prof V Nitnaware for meticulously reading the proofs for second edition and also for preparation of solution manual of this book. We are also thankful to Prof Anil Srirao, Prof Nitin Narkhede, Prof Ms R S Asamwar, Prof Ms Rajshree Raut, Prof Ms P K Parlewar in Department of Electronics and Communication Engineering of Ramdeobaba Kamla Nehru Engineering College, Nagpur for their critical evaluation of the book. Our thanks are due to Professor P A Dwaramwar, R Khobragade, Rajesh Raut and V E Khetade, for their support during preparation of the manuscript.

Our heartfelt thanks are due to Manish Hote for typing and compiling considerable part of this text and Ms. Namita Gupta of Computer Complex, Nagpur for the dedicated and sincere efforts in the enormous DTP work.

AJOY KUMAR RAY  
KISHOR M BHURCHANDI

# The Processors: 8086/8088—Architectures, Pin Diagrams and Timing Diagrams

## INTRODUCTION

**I**ntel introduced its first 4-bit microprocessor 4004 in 1971 and its 8-bit microprocessor 8008 in 1972. These microprocessors could not survive as general purpose microprocessors due to their design and performance limitations. The launch of the first general purpose 8-bit microprocessor 8080 in 1974 by Intel is considered to be the first major stepping stone towards the development of advanced microprocessors. The microprocessor 8085 followed 8080, with a few more added features to its architecture, which resulted in a functionally complete microprocessor. The main limitations of the 8-bit microprocessors were their low speed, low memory addressing capability, limited number of general purpose registers and a less powerful instruction set. All these limitations of the 8-bit microprocessors pushed the designers to build more powerful processors in terms of advanced architecture, more processing capability, larger memory addressing capability and a more powerful instruction set. The 8086 was a result of such developmental design efforts.

In the family of 16-bit microprocessors, Intel's 8086 was the first one to be launched in 1978. The introduction of the 16-bit processor was a result of the increasing demand for more powerful and high speed computational resources. The 8086 microprocessor has a much more powerful instruction set along with the architectural developments which imparts substantial programming flexibility and improvement in speed over the 8-bit microprocessors.

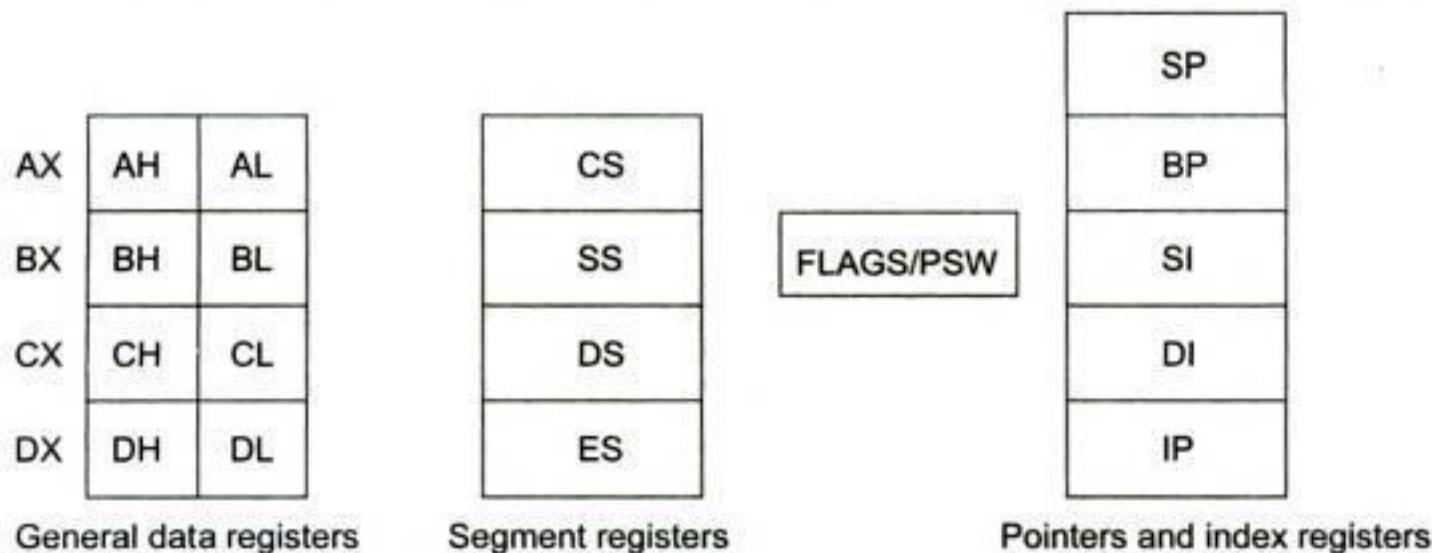
The peripheral chips designed earlier for 8085 were compatible with microprocessor 8086 with slight or no modifications. Though there is a considerable difference between the memory addressing techniques of 8085 and 8086, the memory interfacing technique is similar, but includes the use of a few additional signals. The clock requirements are also different as compared to 8085, but the overall minimal system organisation of 8086 is similar to that of a general 8-bit microprocessor. In this chapter, the architectures of 8086 and 8088 are discussed in adequate details along with the interfacing of the supporting chips with them to form a minimum system. The system organisation is also discussed in significant details for both the operating modes of 8086 and 8088, along with necessary timing diagrams.

## I.I REGISTER ORGANISATION OF 8086

8086 has a powerful set of registers known as *general purpose* and *special purpose registers*. All of them are 16-bit registers. The general purpose registers, can be used as either 8-bit registers or 16-bit registers. They may be either used for holding data, variables and intermediate results temporarily or for other purposes like a counter or for storing offset address for some particular addressing modes etc. The special purpose registers are used as segment registers, pointers, index registers or as offset storage registers for particular addressing modes. We will categorize the register set into four groups, as follows:

### I.I.I General Data Registers

Figure 1.1 shows the register organisation of 8086. The registers AX,BX,CX and DX are the general purpose 16-bit registers. AX is used as 16-bit *accumulator*, with the lower 8-bits of AX designated as AL and higher 8-bits as AH. AL can be used as an 8-bit accumulator for 8-bit operations. This is the most important general purpose register having multiple functions, which will be discussed later.



**Fig. I.1 Register Organisation of 8086**

Usually the letters L and H specify the lower and higher bytes of a particular register. For example, CH means the higher 8-bits of the CX register and CL means the lower 8-bits of the CX register. The letter X is used to specify the complete 16-bit register. The register CX is also used as a default counter in case of string and loop instructions. The register BX is used as an offset storage for forming physical addresses in case of certain addressing modes. DX register is a general purpose register which may be used as an implicit operand or destination in case of a few instructions. The detailed uses of these registers will be more clear when we discuss the addressing modes and the instruction set of 8086.

### 1.1.2 Segment Registers

Unlike 8085, the 8086 addresses a segmented memory. The complete 1 megabyte memory, which the 8086 addresses, is divided into 16 logical segments. Each segment thus contains 64 Kbytes of memory. There are four segment registers, viz. Code Segment Register (CS), Data Segment Register (DS), Extra Segment Register (ES) and Stack Segment Register (SS). The code segment register is used for addressing a memory location in the code segment of the memory, where the executable program is stored. Similarly, the data segment register points to the data segment of the memory, where the data is resided. The extra segment also refers to a segment which essentially is another data segment of the memory. Thus, the extra segment also contains data. The stack segment register is used for addressing stack segment of memory i.e. memory which is used to store stack data. The CPU uses the stack for temporarily storing important data, e.g. the contents of the CPU registers which will be required at a later stage. The stack grows down, i.e. the data is pushed onto the stack in the memory locations with decreasing addresses. When this information will be required by the CPU, they will be popped off from the stack. While addressing any location in the memory bank, the physical address is calculated from two parts, the first is *segment address* and the second is *offset*. The segment registers contain 16-bit segment base addresses, related to different segments. Any of the pointers and index registers or BX may contain the offset of the location to be addressed. The advantage of this scheme is that instead of maintaining a 20-bit register for a physical address, the processor just maintains two 16-bit registers which are within the word length capacity of the machine. Thus the CS, DS, SS and ES segment registers, respectively, contain the segment addresses for the code, data, stack and extra segments of memory. It may be noted that all these segments are the logical segments. They may or may not be physically separated. In other words, a single segment may require more than one memory chip or more than one segment may be accommodated in a single memory chip.

### 1.1.3 Pointers and Index Registers

The pointers contain offset within the particular segments. The pointers IP, BP and SP usually contain offsets within the code (JP), and stack (BP & SP) segments. The *index registers* are used as general purpose registers as well as for offset storage in case of indexed, based indexed and relative based indexed addressing modes. The register SI is generally used to store the offset of source data in data segment while the register DI is used to store the offset of destination in data or extra segment. The index registers are particularly useful for string manipulations.

### 1.1.4 Flag Register

The 8086 *flag register* contents indicate the results of computations in the ALU. It also contains some flag bits to control the CPU operations. Details of the flag register are discussed later in this chapter.

## 1.2 ARCHITECTURE

The architecture of 8086 provides a number of improvements over 8085 architecture. It supports a 16-bit ALU, a set of 16-bit registers and provides segmented memory addressing capability, a rich instruction set, powerful interrupt structure, fetched instruction queue for overlapped fetching and execution etc. The internal block diagram, shown in Fig.1.2, describes the overall organization of different units inside the chip.

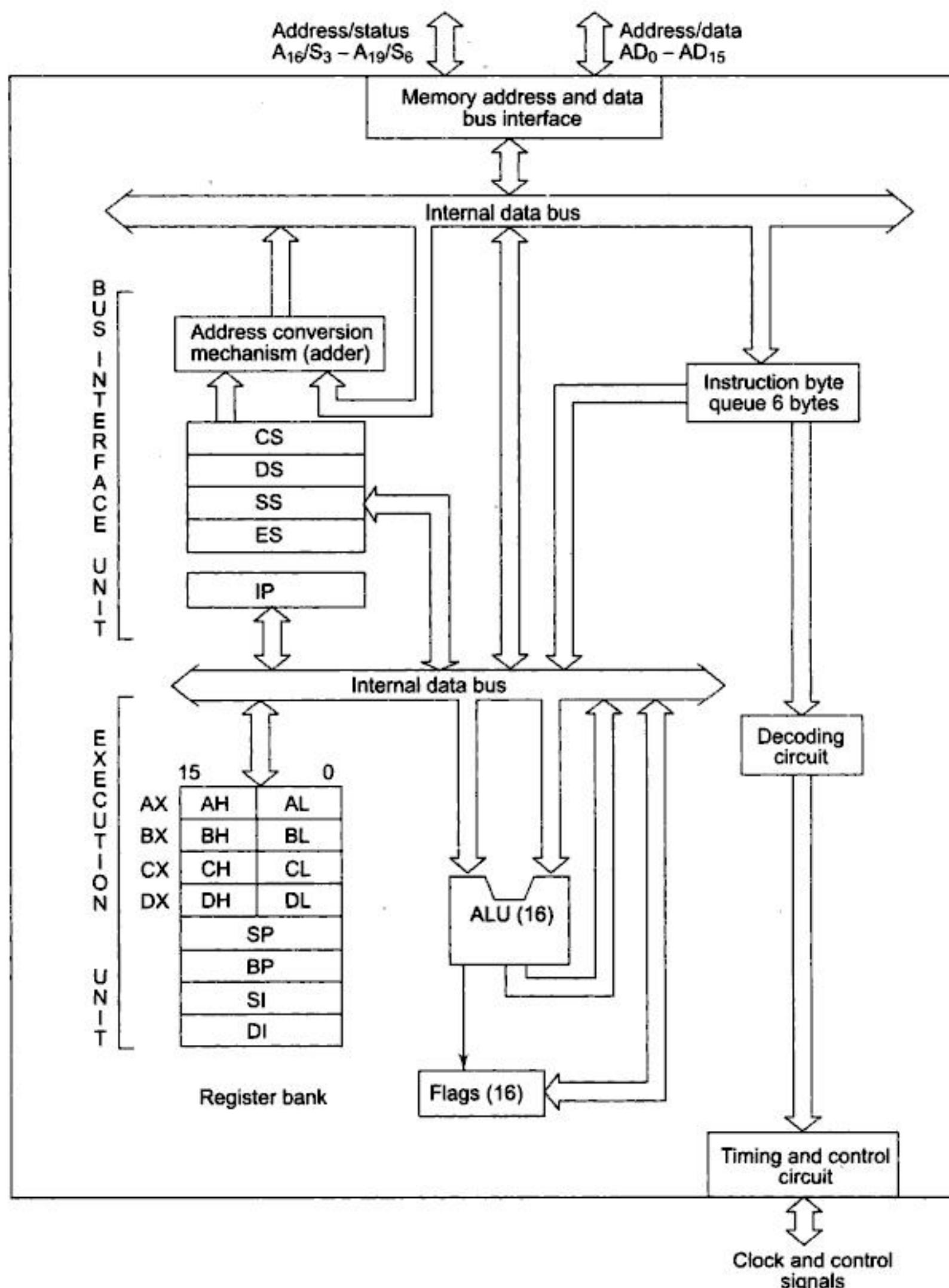


Fig. 1.2 8086 Architecture

The complete architecture of 8086 can be divided into two parts (a) Bus Interface Unit (BIU) and (b) Execution Unit (EU). *The bus interface unit contains the circuit for physical address calculations and*

a predecoding instruction byte queue (6 bytes long). The bus interface unit makes the system's bus signals available for external interfacing of the devices. In other words, this unit is responsible for establishing communications with external devices and peripherals including memory via the bus. As already stated, the 8086 addresses a segmented memory. The complete physical address which is 20-bits long is generated using segment and offset registers, each 16-bits long.

For generating a physical address from contents of these two registers, the content of a segment register also called as segment address is shifted left bit-wise four times and to this result, content of an offset register also called as offset address is added, to produce a 20-bit physical address. For example, if the segment address is 1005H and the offset is 5555H, then the physical address is calculated as below:

Segment address	→ 1005H
Offset address	→ 5555H
Segment address	→ 1005H → 0001 0000 0000 0101
Shifted by 4 bit positions	→ 0001 0000 0000 0101 0000
	+
Offset address	→ 0101 0101 0101 0101
Physical address	→ 0001 0101 0101 1010 0101 1    5    5    A    5

Thus, the segment addressed by the segment value 1005H can have offset values from 0000H to FFFFH within it, i.e. maximum 64K locations may be accommodated in the segment. Thus, the segment register indicates the base address of a particular segment, while the offset indicates the distance of the required memory location in the segment from the base address. Since the offset is a 16-bit number, each segment can have a maximum of 64K locations. The bus interface unit has a separate adder to perform this procedure for obtaining a physical address while addressing memory. The segment address value is to be taken from an appropriate segment register depending upon whether code, data or stack are to be accessed, while the offset may be the content of IP, BX, SI, DI, SP, BP or an immediate 16-bit value, depending upon the addressing mode.

In case of 8085, once the opcode is fetched and decoded, the external bus remains free for some time, while the processor internally executes the instruction. This time slot is utilised in 8086 to achieve the overlapped fetch and execution cycles. While the fetched instruction is executed internally, the external bus is used to fetch the machine code of the next instruction and arrange it in a queue known as predecoded instruction byte queue. It is a 6 bytes long, first-in first-out structure. The instructions from the queue are taken for decoding sequentially. Once a byte is decoded, the queue is rearranged by pushing it out and the queue status is checked for the possibility of the next opcode fetch cycle. While the opcode is fetched by the Bus Interface Unit (BIU), the Execution Unit (EU) executes the previously decoded instruction concurrently. The BIU along with the Execution Unit (EU) thus forms a pipeline. The bus interface unit, thus manages the complete interface of execution unit with memory and I/O devices, of course, under the control of the timing and control unit.

The execution unit contains the register set of 8086 except segment registers and IP. It has a 16-bit ALU, able to perform arithmetic and logic operations. The 16-bit flag register reflects the results of execution by the ALU. The decoding unit decodes the opcode bytes issued from the instruction byte queue. The timing and control unit derives the necessary control signals to execute the instruction

opcode received from the queue, depending upon the information made available by the decoding circuit. The execution unit may pass the results to the bus interface unit for storing them in memory.

### 1.2.1 Memory Segmentation

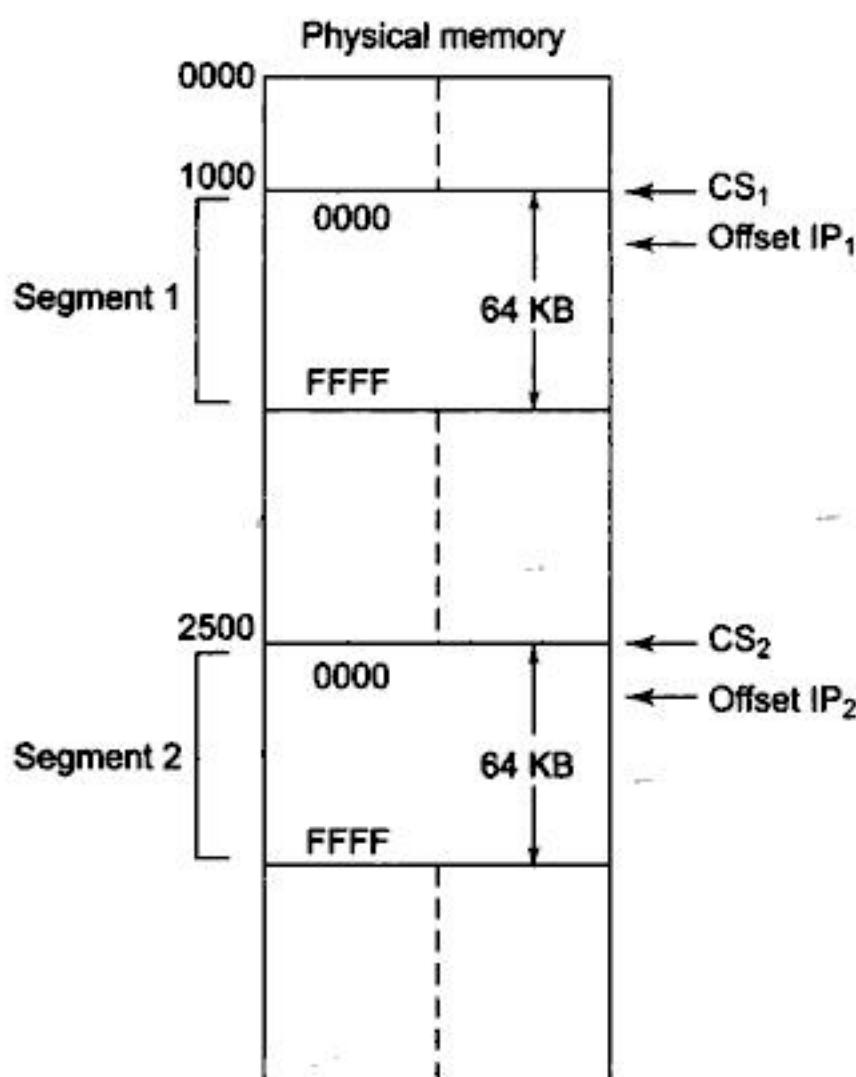
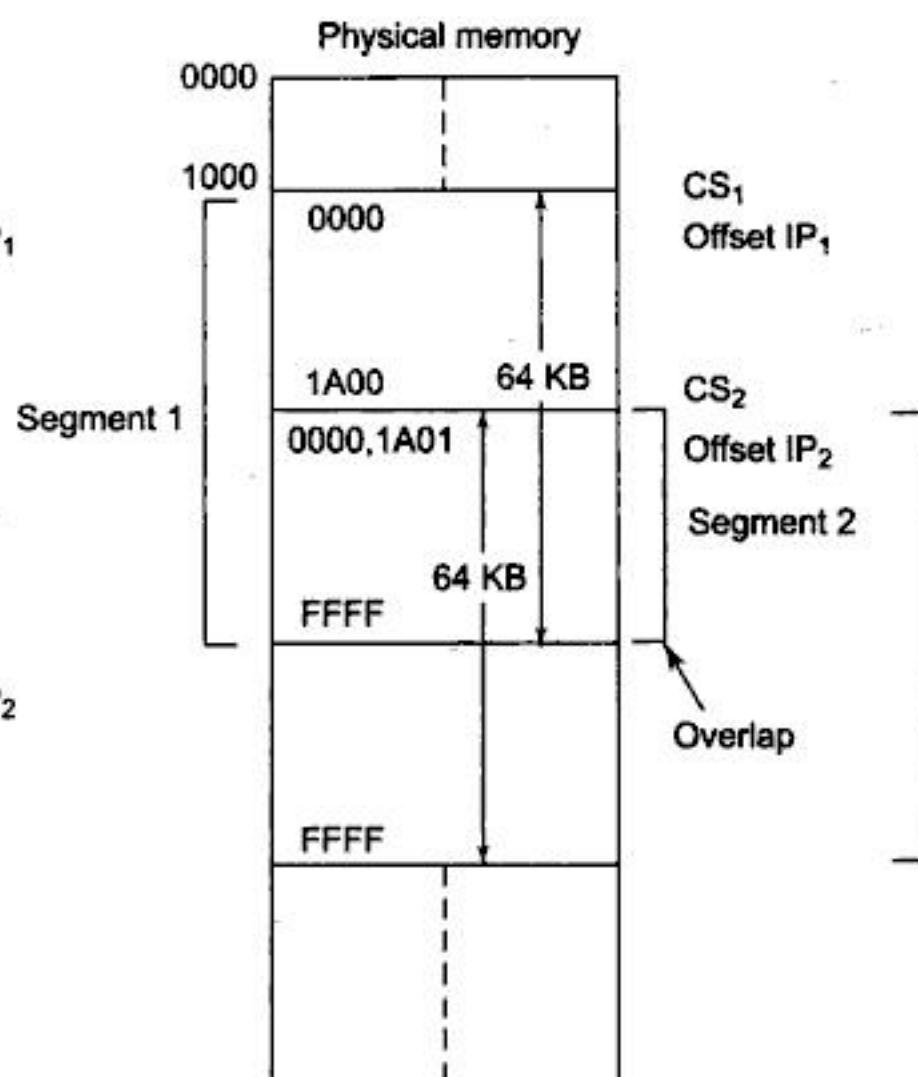
The memory in an 8086/8088 based system is organised as segmented memory. In this scheme, the complete physically available memory may be divided into a number of logical segments. Each segment is 64K bytes in size and is addressed by one of the segment registers. The 16-bit contents of the segment register actually point to the starting location of a particular segment. To address a specific memory location within a segment, we need an offset address. The offset address is also 16-bit long so that the maximum offset value can be FFFFH, and the maximum size of any segment is thus 64K locations. The physical address formation has been explained previously in Section 1.2.

To emphasize this segmented memory concept, we will consider an example of a housing colony containing say, 100 houses. The simplest method of numbering the houses will be to assign the numbers from 1 to 100 to each house sequentially. Suppose, now, if one wants to find out house number 67, then he will start from house number 1 and go on till he finds the house, numbered 67. Consider another case where the 100 houses are arranged in the  $10 \times 10$  (rows  $\times$  columns) pattern. In this case, to find out house number 67, one will directly go to the 6th row and then to the 7th column. In the second scheme, the efforts required for finding the same house will be too less. This second scheme in our example is analogous to the segmented memory scheme, where the addresses are specified in terms of segment addresses analogous to rows and offset addresses analogous to columns.

The CPU 8086 is able to address 1Mbytes of physical memory. The complete 1Mbytes memory can be divided into 16 segments, each of 64Kbytes size. The addresses of the segments may be assigned as 0000H to F000H respectively. The offset address values are from 0000H to FFFFH so that the physical addresses range from 00000H to FFFFFH. In the above said case, the segments are called non-overlapping segments. The non-overlapping segments are shown in Fig. 1.3(a). In some cases, however, the segments may be overlapping. Suppose a segment starts at a particular address and its maximum size can be 64Kbytes. But, if another segment starts before this 64Kbytes locations of the first segment, the two segments are said to be overlapping segments. The area of memory from the start of the second segment to the possible end of the first segment is called an overlapped segment area. Figure 1.3(b) explains the phenomenon more clearly. The locations lying in the overlapped area may be addressed by the same physical address generated from two different sets of segment and offset addresses. The main advantages of the segmented memory scheme are as follows:

1. Allows the memory capacity to be 1Mbytes although the actual addresses to be handled are of 16-bit size
2. Allows the placing of code, data and stack portions of the same program in different parts (segments) of memory, for data and code protection
3. Permits a program and/or its data to be put into different areas of memory each time the program is executed, i.e. provision for relocation is done.

In the Overlapped Area Locations Physical Address =  $CS_1 + IP_1 = CS_2 + IP_2$ , where '+' indicates the procedure of physical address formation.

**Fig. 1.3(a) Non-overlapping Segments****Fig. 1.3(b) Overlapping Segments**

### 1.2.2 Flag Register

8086 has a 16-bit flag register which is divided into two parts, viz. (a) *condition code or status flags* and (b) *machine control flags*. The **condition code flag register** is the lower byte of the 16-bit flag register along with the *overflow flag*. This flag is identical to the 8085 flag register, with an additional overflow flag, which is not present in 8085. This part of the flag register of 8086 reflects the results of the operations performed by ALU. The **control flag register** is the higher byte of the flag register of 8086. It contains three flags, viz. *direction flag* (D), *interrupt flag* (I) and *trap flag* (T).

The complete bit configuration of 8086 flag register is shown in Fig. 1.4.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	O	D	I	T	S	Z	X	Ac	X	P	X	Cy

- O — Overflow flag
- D — Direction flag
- I — Interrupt flag
- T — Trap flag
- S — Sign flag
- Z — Zero flag
- Ac — Auxiliary carry flag
- P — Parity flag
- Cy — Carry flag
- X — Not used

**Fig. 1.4 Flag Register of 8086**

The description of each flag bit is as follows:

**S-Sign Flag** This flag is set when the result of any computation is negative. For signed computations, the sign flag equals the MSB of the result.

**Z-Zero Flag** This flag is set if the result of the computation or comparison performed by the previous instruction/instructions is zero.

**P-Parity Flag** This flag is set to 1 if the lower byte of the result contains even number of 1s.

**C-Carry Flag** This flag is set when there is a carry out of MSB in case of addition or a borrow in case of subtraction. For example, when two numbers are added, a carry may be generated out of the most significant bit position. The carry flag, in this case, will be set to '1'. In case, no carry is generated, it will be '0'. Some other instructions also affect or use this flag and will be discussed later in this text.

**T-Trap Flag** If this flag is set, the processor enters the single step execution mode. In other words, a trap interrupt is generated after execution of each instruction. The processor executes the current instruction and the control is transferred to the Trap interrupt service routine.

**I-Interrupt Flag** If this flag is set, the maskable interrupts are recognised by the CPU, otherwise they are ignored.

**D-Direction Flag** This is used by string manipulation instructions. If this flag bit is '0', the string is processed beginning from the lowest address to the highest address, i.e. *autoincrementing mode*. Otherwise, the string is processed from the highest address towards the lowest address, i.e. *autodecrementing mode*. We will describe string manipulations later in chapter 2 in more detail.

**AC-Auxiliary Carry Flag** This is set if there is a carry from the lowest nibble, i.e. bit three, during addition or borrow for the lowest nibble, i.e. bit three, during subtraction.

**O-Overflow Flag** This flag is set if an overflow occurs, i.e. if the result of a signed operation is large enough to be accommodated in a destination register. For example, in case of the addition of two signed numbers, if the result overflows into the sign bit, i.e. the result is of more than 7-bits in size in case of 8-bit signed operations and more than 15-bits in size in case of 16-bit signed operations, then the overflow flag will be set.

### 1.3 SIGNAL DESCRIPTIONS OF 8086

The microprocessor 8086 is a 16-bit CPU available in three clock rates, i.e. 5, 8 and 10 MHz, packaged in a 40 pin CERDIP or plastic package. The 8086 operates in single processor or multiprocessor configurations to achieve high performance. The pin configuration is shown in Fig. 1.5. Some of the pins serve a particular function in minimum mode (single processor mode) and others function in maximum mode (multiprocessor mode) configuration.

The 8086 signals can be categorised in three groups. The first are the signals having common functions in minimum as well as maximum mode, the second are the signals which have special functions for minimum mode and the third are the signals having special functions for maximum mode.

		8086	Maximum mode	Minimum mode
GND	1		40	VCC
AD <sub>14</sub>	2		39	AD <sub>15</sub>
AD <sub>13</sub>	3		38	A <sub>16/S<sub>3</sub></sub>
AD <sub>12</sub>	4		37	A <sub>17/S<sub>4</sub></sub>
AD <sub>11</sub>	5		36	A <sub>18/S<sub>5</sub></sub>
AD <sub>10</sub>	6		35	A <sub>19/S<sub>6</sub></sub>
AD <sub>9</sub>	7		34	BHE/S <sub>7</sub>
AD <sub>8</sub>	8		33	MN/MX
AD <sub>7</sub>	9		32	RD
AD <sub>6</sub>	10		31	RQ/GT <sub>0</sub> (HOLD)
AD <sub>5</sub>	11		30	RQ/GT <sub>1</sub> (HLDA)
AD <sub>4</sub>	12		29	LOCK (WR)
AD <sub>3</sub>	13		28	S <sub>2</sub> (M/I <sub>O</sub> )
AD <sub>2</sub>	14		27	S <sub>1</sub> (DT/R)
AD <sub>1</sub>	15		26	S <sub>0</sub> (DEN)
AD <sub>0</sub>	16		25	QS <sub>0</sub> (ALE)
NMI	17		24	QS <sub>1</sub> (INTA)
INTR	18		23	TEST
CLK	19		22	READY
GND	20		21	RESET

**Fig. 1.5 Pin Configuration of 8086**

The following signal descriptions are common for both the minimum and maximum modes.

**AD<sub>15</sub>-AD<sub>0</sub>** These are the time multiplexed memory I/O address and data lines. Address remains on the lines during T<sub>1</sub> state, while the data is available on the data bus during T<sub>2</sub>, T<sub>3</sub>, T<sub>w</sub> and T<sub>4</sub>. Here T<sub>1</sub>, T<sub>2</sub>, T<sub>3</sub>, T<sub>4</sub> and T<sub>w</sub> are the clock states of a machine cycle. T<sub>w</sub> is a wait state. These lines are active high and float to a tristate during interrupt acknowledge and local bus hold acknowledge cycles.

**A<sub>19/S<sub>6</sub></sub>, A<sub>18/S<sub>5</sub></sub>, A<sub>17/S<sub>4</sub></sub>, A<sub>16/S<sub>3</sub></sub>** These are the time multiplexed address and status lines. During T<sub>1</sub>, these are the most significant address lines for memory operations. During I/O operations, these lines are low. During memory or I/O operations, status information is available on those lines for T<sub>2</sub>, T<sub>3</sub>, T<sub>w</sub> and T<sub>4</sub>. The status of the interrupt enable flag bit (displayed on S<sub>5</sub>) is updated at the beginning of each clock cycle. The S<sub>4</sub> and S<sub>3</sub> together indicate which segment register is presently being used for memory accesses, as shown in Table 1.1. These lines float to tri-state off (tristated) during the local bus hold acknowledge. The status line S<sub>6</sub> is always low (logical). The address bits are separated from the status bits using latches controlled by the ALE signal.

**Table 1.1 Bus High Enable/status**

<i>S<sub>4</sub></i>	<i>S<sub>3</sub></i>	<i>Indications</i>
0	0	Alternate Data
0	1	Stack
1	0	Code or none
1	1	Data

**BHE/S<sub>7</sub>-Bus High Enable/Status** The bus high enable signal is used to indicate the transfer of data over the higher order (D<sub>15</sub>-D<sub>8</sub>) data bus as shown in Table 1.2. It goes low for the data transfers over D<sub>15</sub>-D<sub>8</sub> and is used to derive chip selects of odd address memory bank or peripherals. BHE is low during T<sub>1</sub> for read, write and interrupt acknowledge cycles, whenever a byte is to be transferred on the higher byte of the data bus. The status information is available during T<sub>2</sub>, T<sub>3</sub> and T<sub>4</sub>. The signal is active low and is tristated during 'hold'. It is low during T<sub>1</sub> for the first pulse of the interrupt acknowledge cycle. S<sub>7</sub> is not currently used.

**Table 1.2**

<i>BHE</i>	<i>A<sub>0</sub></i>	<i>Indication</i>
0	0	Whole word
0	1	Upper byte from or to odd address.
1	0	Lower byte from or to even address
1	1	None

**RD-Read** Read signal, when low, indicates the peripherals that the processor is performing a memory or I/O read operation. RD is active low and shows the state for T<sub>2</sub>, T<sub>3</sub>, T<sub>w</sub> of any read cycle. The signal remains tristated during the 'hold acknowledge'.

**READY** This is the acknowledgement from the slow devices or memory that they have completed the data transfer. The signal made available by the devices is synchronized by the 8284A clock generator to provide ready input to the 8086. The signal is active high.

**INTR-Interrupt Request** This is a level triggered input. This is sampled during the last clock cycle of each instruction to determine the availability of the request. If any interrupt request is pending, the processor enters the interrupt acknowledge cycle. This can be internally masked by resetting the interrupt enable flag. This signal is active high and internally synchronized.

**TEST** This input is examined by a 'WAIT' instruction. If the TEST input goes low, execution will continue, else, the processor remains in an idle state. The input is synchronized internally during each clock cycle on leading edge of clock.

**NMI-Non-maskable Interrupt** This is an edge-triggered input which causes a Type2 interrupt. The NMI is not maskable internally by software. A transition from low to high initiates the interrupt response at the end of the current instruction. This input is internally synchronized.

**RESET** This input causes the processor to terminate the current activity and start execution from FFFF0H. The signal is active high and must be active for at least four clock cycles. It restarts execution when the RESET returns low. RESET is also internally synchronised.

**CLK-Clock Input** The clock input provides the basic timing for processor operation and bus control activity. It's an asymmetric square wave with 33% duty cycle. The range of frequency for different 8086 versions is from 5MHz to 10MHz.

**V<sub>CC</sub>** +5V power supply for the operation of the internal circuit.

**GND** ground for the internal circuit.

**MN/MX** The logic level at this pin decides whether the processor is to operate in either minimum (single processor) or maximum (multiprocessor) mode.

The following pin functions are for the minimum mode operation of 8086:

**M/I/O -Memory/IO** This is a status line logically equivalent to  $\bar{S}_2$  in the maximum mode. When it is low, it indicates the CPU is having an I/O operation, and when it is high, it indicates that the CPU is having a memory operation. This line becomes active in the previous  $T_4$  and remains active till final  $T_4$  of the current cycle. It is tristated during local bus "hold acknowledge".

**INTA -Interrupt Acknowledge** This signal is used as a read strobe for interrupt acknowledge cycles. In other words, when it goes low, it means that the processor has accepted the interrupt. It is active low during  $T_2$ ,  $T_3$  and  $T_w$  of each interrupt acknowledge cycle.

**ALE-Address Latch Enable** This output signal indicates the availability of the valid address on the address/data lines, and is connected to latch enable input of latches. This signal is active high and is never tristated.

**DT/R -Data Transmit/Receive** This output is used to decide the direction of data flow through the transreceivers (bidirectional buffers). When the processor sends out data, this signal is high and when the processor is receiving data, this signal is low. Logically, this is equivalent to  $\bar{S}_1$  in maximum mode. Its timing is the same as M/I/O. This is tristated during 'hold acknowledge'.

**DEN -Data Enable** This signal indicates the availability of valid data over the address/data lines. It is used to enable the transreceivers (bidirectional buffers) to separate the data from the multiplexed address/data signal. It is active from the middle of  $T_2$  until the middle of  $T_4$ . DEN is tristated during 'hold acknowledge' cycle.

**HOLD,HLDA-Hold/Hold Acknowledge** When the HOLD line goes high, it indicates to the processor that another master is requesting the bus access. The processor, after receiving the HOLD request, issues the hold acknowledge signal on HLDA pin, in the middle of the next clock cycle after completing the current bus (instruction) cycle. At the same time, the processor floats the local bus and control lines. When the processor detects the HOLD line low, it lowers the HLDA signal. HOLD is an asynchronous input, and it should be externally synchronized.

If the DMA request is made while the CPU is performing a memory or I/O cycle, it will release the local bus during  $T_4$  provided:

1. The request occurs on or before  $T_2$  state of the current cycle
2. The current cycle is not operating over the lower byte of a word (or operating on an odd address)

3. The current cycle is not the first acknowledge of an interrupt acknowledge sequence
4. A Lock instruction is not being executed.

So far we have presented the pin descriptions of 8086 in minimum mode.

The following pin functions are applicable for maximum mode operation of 8086.

**$\bar{S}_2$ ,  $\bar{S}_1$ ,  $\bar{S}_0$ -Status Lines** These are the status lines which indicate the type of operation, being carried out by the processor. These become active during  $T_4$  of the previous cycle and remain active during  $T_1$  and  $T_2$  of the current bus cycle. The status lines return to passive state during  $T_3$  of the current bus cycle so that they may again become active for the next bus cycle during  $T_4$ . Any change in these lines during  $T_3$  indicates the starting of a new cycle, and return to passive state indicates end of the bus cycle. These status lines are encoded in Table 1.3.

**Table 1.3**

$\bar{S}_2$	$\bar{S}_1$	$\bar{S}_0$	<i>Indication</i>
0	0	0	Interrupt acknowledge
0	0	1	Read I/O port
0	1	0	Write 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

**LOCK** This output pin indicates that other system bus masters will be prevented from gaining the system bus, while the LOCK signal is low. The LOCK signal is activated by the 'LOCK' prefix instruction and remains active until the completion of the next instruction. This floats to tri-state off during "hold acknowledge". When the CPU is executing a critical instruction which requires the system bus, the LOCK prefix instruction ensures that other processors connected in the system will not gain the control of the bus. The 8086, while executing the prefixed instruction, asserts the bus lock signal output, which may be connected to an external bus controller.

**$QS_1$ ,  $QS_0$ -Queue Status** These lines give information about the status of the code-prefetch queue. These are active during the CLK cycle after which the queue operation is performed. These are encoded as shown in Table 1.4.

**Table 1.4**

$QS_1$	$QS_0$	<i>Indication</i>
0	0	No operation
0	1	First byte of opcode from the queue
1	0	Empty queue
1	1	Subsequent byte from the queue

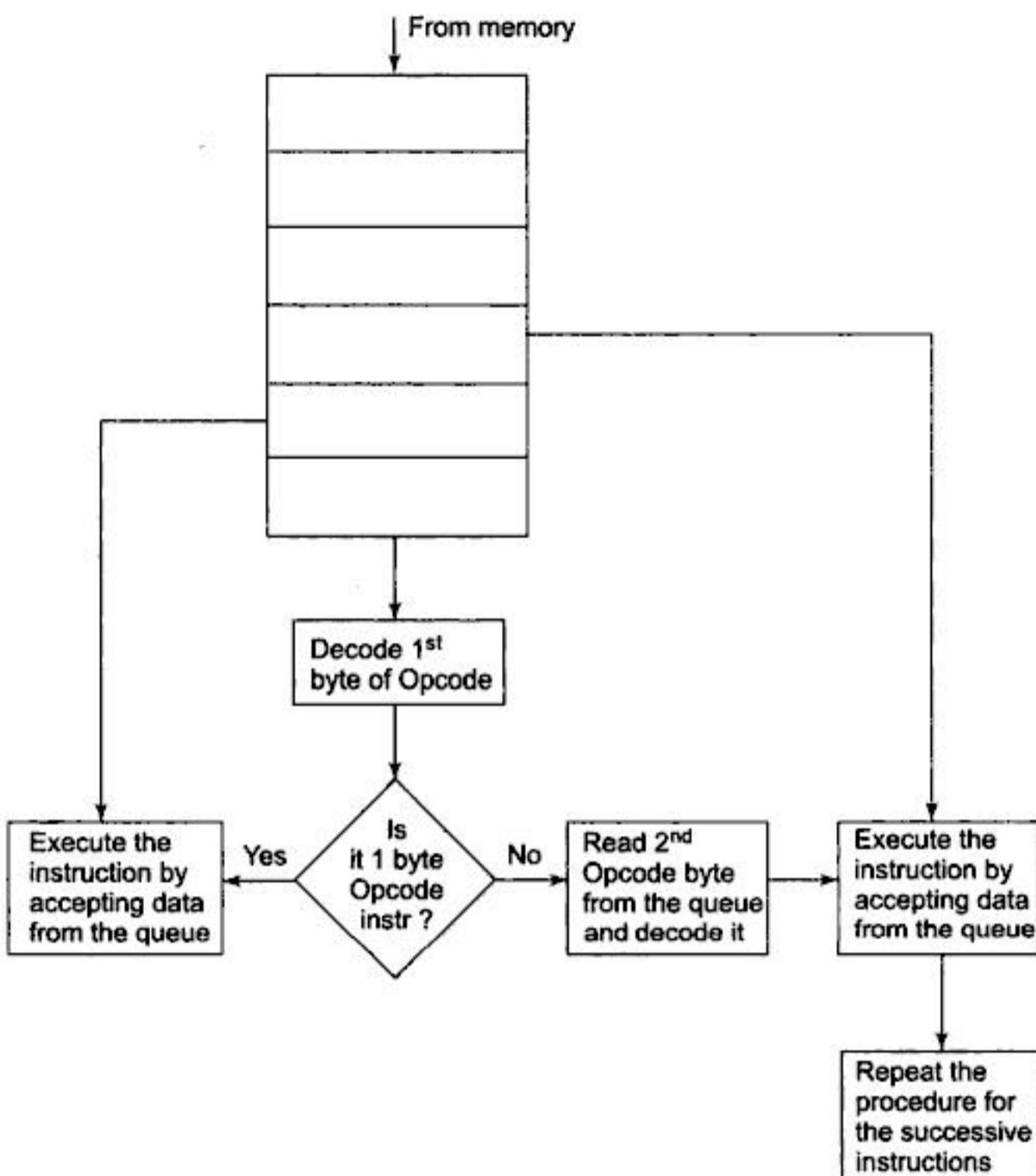
This modification in a simple fetch and execute architecture of a conventional microprocessor offers an added advantage of *pipelined processing* of the instructions. The 8086 architecture has a 6-byte instruction prefetch queue. Thus even the largest (6-bytes) instruction can be prefetched from the memory and stored in the prefetch queue. This results in a faster execution of the instructions. In 8085, an instruction (opcode and operand) is fetched, decoded and executed and only after the execution of this instruction, the next one is fetched. By prefetching the instruction, there is a considerable speeding up in instruction execution in 8086. This scheme is known as *instruction pipelining*.

In the beginning, the CS:IP is loaded with the required address from which the execution is to be started. Initially, the queue will be empty and the microprocessor starts a fetch operation to bring one byte (the first byte) of instruction code, if the CS:IP address is odd or two bytes at a time, if the CS:IP address is even. The first byte is a complete opcode in case of some instructions (one byte opcode instruction) and it is a part of opcode, in case of other instructions (two byte long opcode instructions), the remaining part of opcode may lie in the second byte. But invariably the first byte of an instruction is an opcode. These opcodes along with data are fetched and arranged in the queue. When the first byte from the queue goes for decoding and interpretation, one byte in the queue becomes empty and subsequently the queue is updated. The microprocessor does not perform the next fetch operation till at least two bytes of the instruction queue are emptied. The instruction execution cycle is never broken for fetch operation. After decoding the first byte, the decoding circuit decides whether the instruction is of single opcode byte or double opcode byte. If it is single opcode byte, the next bytes are treated as data bytes depending upon the decoded instruction length, otherwise, the next byte in the queue is treated as the second byte of the instruction opcode. The second byte is then decoded in continuation with the first byte to decide the instruction length and the number of subsequent bytes to be treated as instruction data. The queue is updated after every byte is read from the queue but the fetch cycle is initiated by BIU only if at least two bytes of the queue are empty and the EU may be concurrently executing the fetched instructions.

The next byte after the instruction is completed is again the first opcode byte of the next instruction. A similar procedure is repeated till the complete execution of the program. The main point to be noted here is, that the fetch operation of the next instruction is overlapped with the execution of the current instruction. As shown in the architecture, there are two separate units, namely, the execution unit and the bus interface unit. While the execution unit is busy in executing an instruction, after it is completely decoded, the bus interface unit may be fetching the bytes of the next instruction from memory, depending upon the queue status. Figure 1.6 explains the queue operation.

**RQ/GT<sub>0</sub>, RQ/GT<sub>1</sub>-Request/Grant** These pins are used by other local bus masters, in maximum mode, to force the processor to release the local bus at the end of the processor's current bus cycle. Each of the pins is bidirectional with RQ/GT<sub>0</sub> having higher priority than RQ/GT<sub>1</sub>. RQ/GT pins have internal pull-up resistors and may be left unconnected. The request/grant sequence is as follows:

1. A pulse one clock wide from another bus master requests the bus access to 8086.
2. During T<sub>4</sub> (current) or T<sub>1</sub> (next) clock cycle, a pulse one clock wide from 8086 to the requesting master, indicates that the 8086 has allowed the local bus to float and that it will enter the "hold acknowledge" state in the next clock cycle. The CPU's bus interface unit is likely to be disconnected from the local bus of the system.
3. A one clock wide pulse from the another master indicates to 8086 that the 'hold' request is about to end and the 8086 may regain control of the local bus at the next clock cycle.



**Fig. 1.6 The Queue Operation**

Thus, each master to master exchange of the local bus is a sequence of 3 pulses. There must be at least one dead clock cycle after each bus exchange. The request and grant pulses are active low. For the bus requests those are received while 8086 is performing memory or I/O cycle, the granting of the bus is governed by the rules as discussed in case of HOLD, and HLDA in the minimum mode.

Until now, we have described the architecture and pin configuration of 8086. In the next section, we will study some operational features of 8086 based systems.

#### 1.4 PHYSICAL MEMORY ORGANISATION

In an 8086 based system, the 1Mbytes memory is physically organised as an odd bank and an even bank, each of 512 Kbytes, addressed in parallel by the processor. Byte data with an even address is transferred on  $D_7 - D_0$ , while the byte data with an odd address is transferred on  $D_{15} - D_8$  buss lines. The processor provides two enable signals, BHE and  $A_0$  for selection of either even or odd or both the

banks. The instruction stream is fetched from memory as words and is addressed internally by the processor as necessary. In other words, if the processor fetches a word (consecutive two bytes) from memory, there are different possibilities, like:

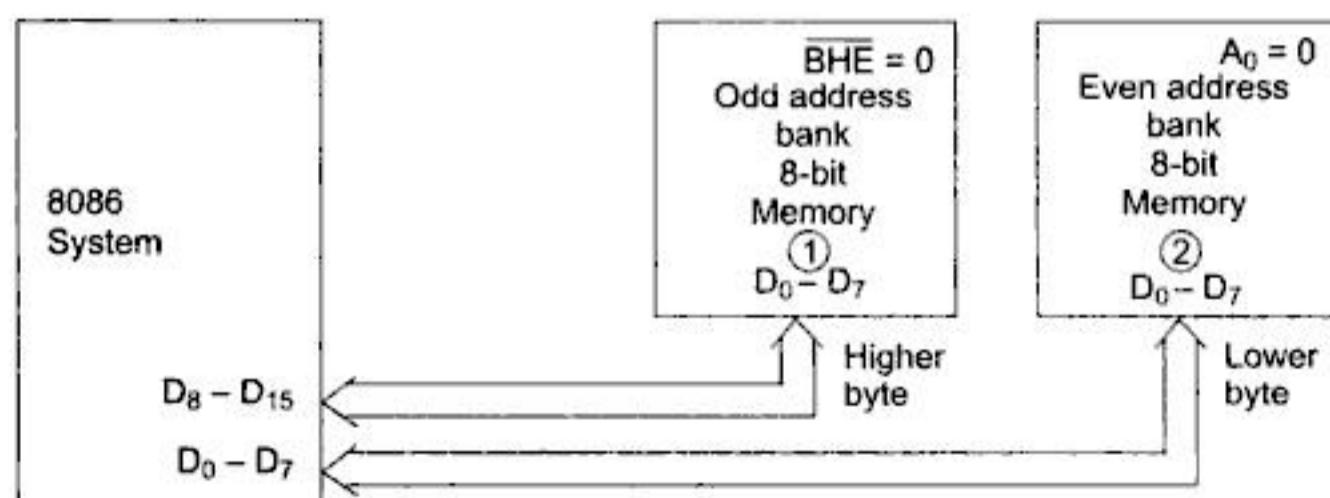
1. Both the bytes may be data operands
2. Both the bytes may contain opcode bits
3. One of the bytes may be opcode while the other may be data

All the above possibilities are taken care of by the internal decoder circuit of the microprocessor. The opcodes and operands are identified by the internal decoder circuit which further derives the signals those act as input to the timing and control unit. The timing and control unit then derives all the signals required for execution of the instruction.

While referring to word data, the BIU requires one or two memory cycles, depending upon whether the starting byte is located at an even or odd address. It is always better to locate the word data at an even address. To read or write a complete word from/to memory, if it is located at an even address, only one read or write cycle is required. If the word is located at an odd address, the first read or write cycle is required for accessing the lower byte while the second one is required for accessing the upper byte. Thus, two bus cycles are required, if a word is located at an odd address. It should be kept in mind that while initialising the structures like stack they should be initialised at an even address for efficient operation.

8086 is a 16-bit microprocessor and hence can access two bytes of data in one memory or I/O read or write operation. But the commercially available memory chips are only byte size, i.e. they can store only one byte in a memory location. Obviously, to store 16-bit data, two successive memory locations are used and the lower byte of 16-bit data can be stored in the first memory location while the second byte is stored in the next location. In a sixteen bit read or write operation both of these bytes will be read or written in a single machine cycle.

A map of an 8086 memory system starts at 00000H and ends at FFFFFH. 8086 being a 16-bit processor is expected to access 16-bit data to / from 8-bit commercially available memory chips in parallel, as shown below in Fig. 1.7.



**Fig. 1.7 Physical Memory Organization**

Thus, bits  $D_0 - D_7$  of a 16-bit data will be transferred over  $D_0 - D_7$  (lower byte) of 16-bit data bus to / from 8-bit memory (2) and bit  $D_8 - D_{15}$  of the 16-bit data will be transferred over  $D_8 - D_{15}$  (higher byte) of the 16-bit data bus of the microprocessor, to / from 8-bit memory (1). Thus to achieve 16-bit data transfer using 8-bit memories, in parallel, the map of the complete system byte memory addresses will obviously be divided into the two memory banks as shown in Fig. 1.7.

The lower byte of a 16-bit data is stored at the first address of the map 00000H and it is to be transferred over D<sub>0</sub> – D<sub>7</sub> of the microprocessor bus so 00000H must be in 8-bit memory (2). Higher byte of the 16-bit data is stored in the next address 00001H; it is to be transferred over D<sub>8</sub> – D<sub>15</sub> of the microprocessor bus so the address 00001H must be in 8-bit memory (1) of Fig. 1.7. On similar lines, for the next 16-bit data stored in the memory, immediately after the previous one, the lower byte will be stored at the next address 00002H and it must be in 8-bit memory (2) while the higher byte will be stored at the next address 00003H that must be in 8-bit memory (1). Thus, if it is imagined that the complete memory map of 8086 is filled with 16-bit data, all the lower bytes (D<sub>0</sub> – D<sub>7</sub>) will be stored in the 8-bit memory bank (2) and all the higher bytes (D<sub>8</sub> – D<sub>15</sub>) will be stored in the 8-bit memory bank (1). Consequently, it can be observed that all the lower bytes have to be stored at even addresses and all the higher bytes have to be stored at odd addresses. Thus, the 8-bit memory bank (1) will be called an odd address bank and the 8-bit memory bank (2) will be called an even address bank. The complete memory map of 8086 system is thus divided into even and odd address memory banks.

If 8086 transfers a 16-bit data to / from memory, both of these banks must be selected for the 16-bit operation. However, to maintain an upward compatibility with 8085, 8086 must be able to implement 8-bit operations. In which case, two possibilities arise; the first being 8-bit operation with even memory bank, i.e. with an even address and the second one is 8-bit operation with odd address memory bank, i.e. with an odd address. The two signals A<sub>0</sub> and  $\overline{\text{BHE}}$  solve the problem of selection of appropriate memory banks as presented in Table 1.2.

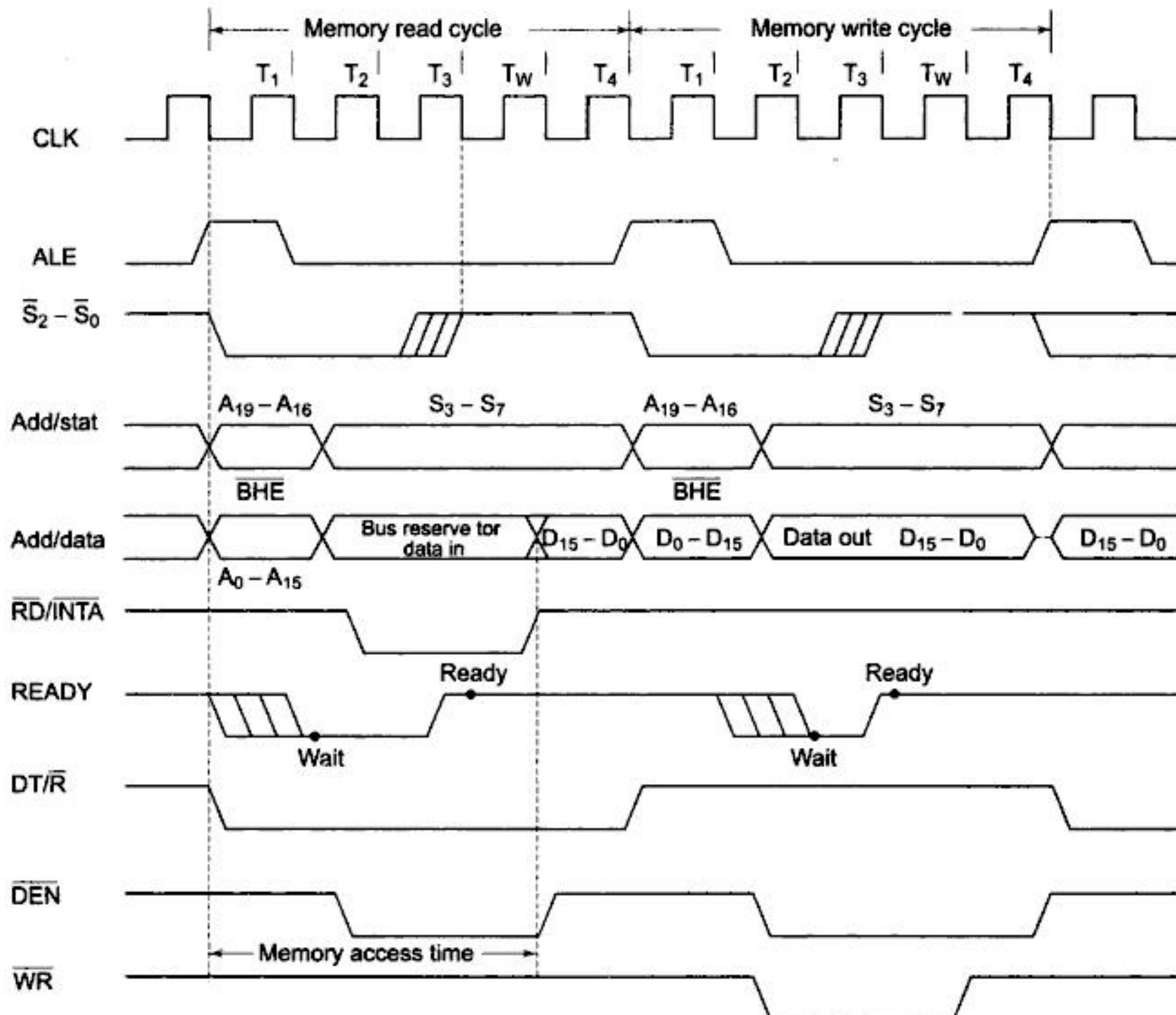
Certain locations in memory are reserved for specific CPU operations. The locations from FFFF0H to FFFFFH are reserved for operations including jump to initialisation programme and I/O-processor initialisation. The locations 00000H to 003FFH are reserved for *interrupt vector table*. The interrupt structure provides space for a total of 256 interrupt vectors. The vectors, i.e. CS and IP for each interrupt routine requires 4 bytes for storing it in the interrupt vector table. Hence, 256 types of interrupt require 256 × 4 = 03FFH (1Kbyte) locations for the complete interrupt vector table.

## I.5 GENERAL BUS OPERATION

The 8086 has a combined address and data bus commonly referred to as a time multiplexed address and data bus. The main reason behind multiplexing address and data over the same pins is the maximum utilisation of processor pins and it facilitates the use of 40 pin standard DIP package. The bus can be demultiplexed using a few latches and transreceivers, whenever required. In the following text, we will discuss a general bus operation cycle.

Basically, all the processor bus cycles consist of at least four clock cycles. These are referred to as T<sub>1</sub>, T<sub>2</sub>, T<sub>3</sub> and T<sub>4</sub>. The address is transmitted by the processor during T<sub>1</sub>. It is present on the bus only for one cycle. During T<sub>2</sub>, i.e. the next cycle, the bus is tristated for changing the direction of bus for the following data read cycle. The data transfer takes place during T<sub>3</sub> and T<sub>4</sub>. In case, an addressed device is slow and shows 'NOT READY' status the wait states T<sub>w</sub> are inserted between T<sub>3</sub> and T<sub>4</sub>. These clock states during wait period are called *idle states* (T<sub>i</sub>), *wait states* (T<sub>w</sub>) or *inactive states*. The processor uses these cycles for internal housekeeping. The Address Latch Enable (ALE) signal is emitted during T<sub>1</sub> by the processor (minimum mode) or the bus controller (maximum mode) depending upon the status of the MN/  $\overline{\text{MX}}$  input. The negative edge of this ALE pulse is used to separate the address and the data or status information. In maximum mode, the

status lines  $\bar{S}_0$ ,  $\bar{S}_1$  and  $\bar{S}_2$  are used to indicate the type of operation as discussed in the signal description section of this chapter. Status bits  $S_3$  to  $S_7$  are multiplexed with higher order address bits and the  $\overline{\text{BHE}}$  signal. Address is valid during  $T_1$  while the status bits  $S_3$  to  $S_7$  are valid during  $T_2$  through  $T_4$ . Figure 1.8 shows a general bus operation cycle of 8086.

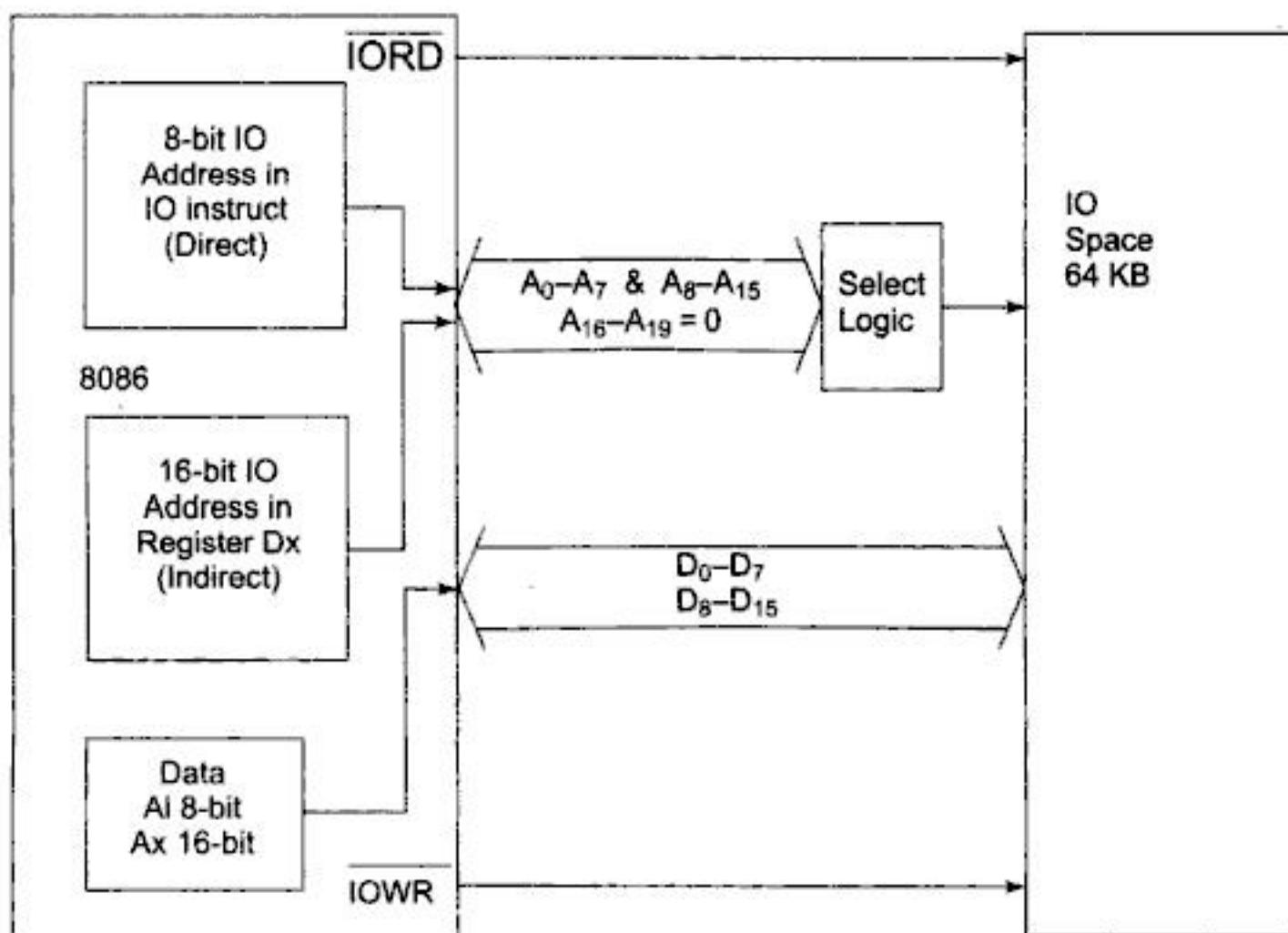


**Fig. 1.8 General Bus Operation Cycle of 8086**

## 1.6 I/O ADDRESSING CAPABILITY

The 8086/8088 processor can address up to 64K I/O byte registers or 32K word registers. The limitation is that the address of an I/O device must not be greater than 16 bits in size, this means that a maximum number of  $2^{16}$ , i.e. 64Kbyte I/O devices may be accessed by the CPU. The I/O address appears on the address lines  $A_0$  to  $A_{15}$  for one clock cycle ( $T_1$ ). It may then be latched using the ALE signal. The upper address lines ( $A_{16} - A_{19}$ ) are at logic 0 level during the I/O operations.

The 16-bit register DX is used as 16-bit I/O address pointer, with full capability to address up to 64K devices. In this case, the I/O ports are addressed in the same manner as memory locations in the based addressing mode using BX. In memory mapped I/O interfacing, the I/O device addresses are treated as memory locations in page 0, i.e. segment address 0000H. Even addressed bytes are transferred on D<sub>7</sub>-D<sub>0</sub> and odd addressed bytes are transferred on D<sub>8</sub>-D<sub>15</sub> lines. While designing any 8-bit I/O system around 8086, care must be taken that all the byte registers in the system should be even addressed. Figure 1.9 shows 8086 IO addressing scheme.



**Fig. 1.9 8086 IO Addressing**

## 1.7 SPECIAL PROCESSOR ACTIVITIES

### 1.7.1 Processor Reset and Initialisation

When logic 1 is applied to the RESET pin of the microprocessor, it is reset. It remains in this state till logic 0 is again applied to the RESET pin. The 8086 terminates the on-going operation on the positive edge of the reset signal. When the negative edge is detected, the reset sequence starts and is continued for nearly 10 clock cycles. During this period, all the internal register contents are set to 0000H except CS is set to value FFFFH . Thus, the execution starts again from the physical address FFFF0H. Due to this, the EPROM in an 8086 system is interfaced so as to have the physical memory locations FFFF0H to FFFFFH in it, i.e. at the end of the map.

For the reset signal to be accepted by 8086, it must be high for at least 4 clock cycles. From the instant the power is on, the reset pulse should not be applied to 8086 before 50  $\mu$ s to allow proper initialisation of 8086. In the reset state, all 3-state outputs are tristated. Status signals are active in idle

state for the first clock cycle after the reset becomes active, and then floats to tristate. The ALE and HLDA lines are driven low during the reset operation.

Non-maskable interrupt enable request, which appears before the second clock after the end of the reset operation, will not be served. For the NMI request to be served, it must appear after the second clock cycle during reset initialisation or later. If a HOLD request appears immediately after RESET, it will be immediately served after initialisation, before execution of any instruction.

### 1.7.2 HALT

When the processor executes a HLT instruction, it enters the 'halt' state. However, before doing so, it indicates that it is entering 'halt' state in two ways, depending upon whether it is in the minimum or maximum mode. When the processor is in minimum mode and wants to enter halt state, it issues an ALE pulse but does not issue any control signal. When the processor is in maximum mode and wants to enter the halt state, it puts the HALT status (011) on  $\bar{S}_2$ ,  $\bar{S}_1$  and  $\bar{S}_0$  pins and then the bus controller issues an ALE pulse but no qualifying signal, i.e. no appropriate address or control signals are issued to the bus. Only an interrupt request or reset will force the 8086 to come out of the 'halt' state. Even the HOLD request cannot force the 8086 out of 'halt' state.

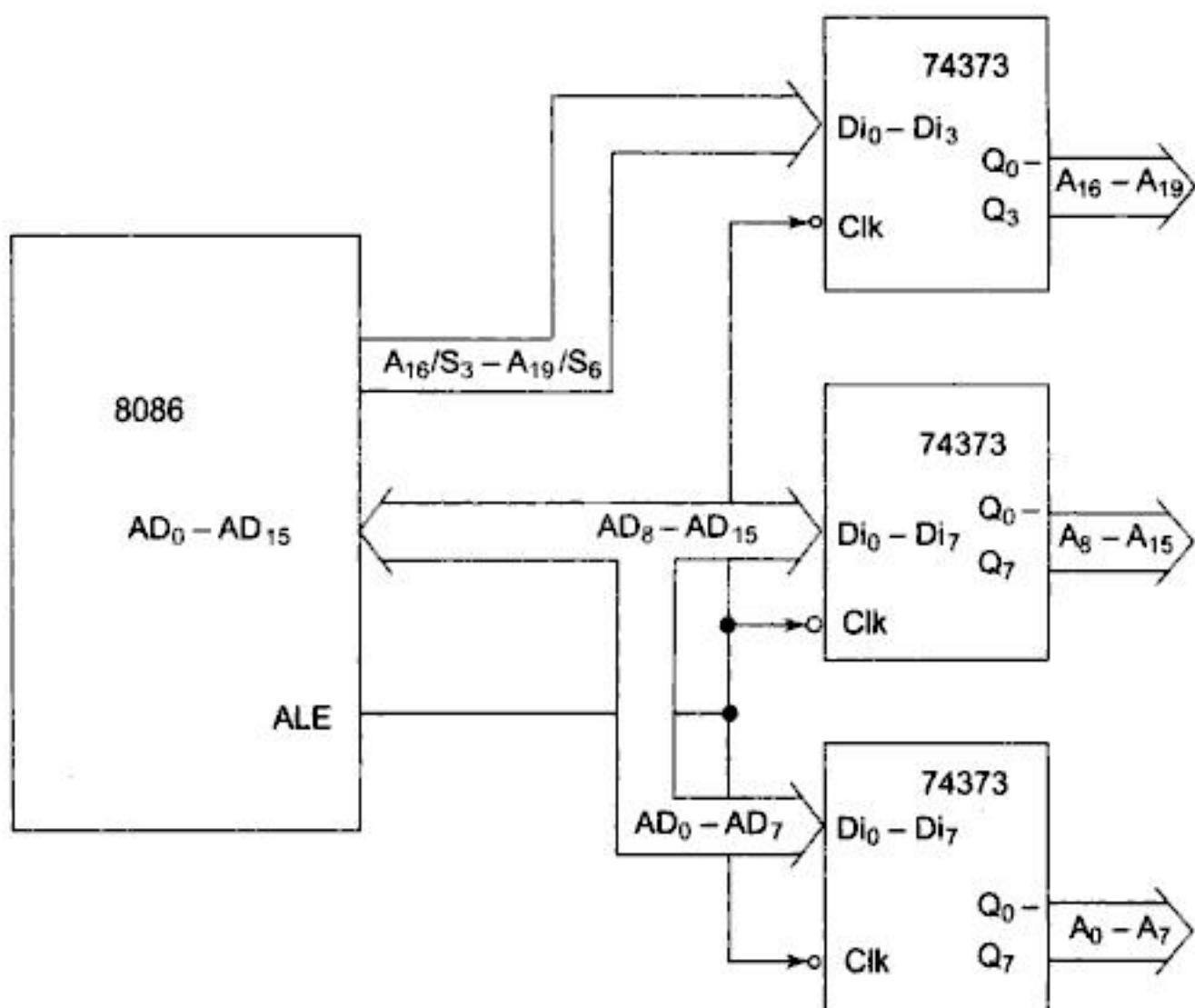
### 1.7.3 TEST and Synchronization with External Signals

Besides the interrupt, hold and general I/O capabilities, the 8086 has an extra facility of the TEST signal. When the CPU executes a WAIT instruction, the processor preserves the contents of the registers, before execution of the WAIT instruction, and the CPU waits for the TEST input pin to go low. If the TEST pin goes low, it continues further execution, otherwise, it keeps on waiting for the TEST pin to go low. For the TEST signal to be accepted, it must be low for at least 5 clock cycles. The activity of waiting does not consume any bus cycle. The processor remains in the idle state while waiting. While waiting, any 'HOLD' request from an external device may be served. If an interrupt occurs when the processor is waiting, it fetches the wait instruction once more, executes it, and then serves the interrupt. After returning from the interrupt, it fetches the wait instruction once more and continues in the 'wait' state.

Thus, the execution of the portion of a program which appears in the program after WAIT instruction can be synchronized with an external signal connected with the TEST input.

### 1.7.4 Deriving System Bus

The 8086 has a multiplexed 16-bit address / data bus ( $A_{D_0}$  -  $A_{D_{15}}$ ) and a multiplexed 4-bit address / status bus  $A_{16}$  /  $S_3$  -  $A_{19}$  /  $S_6$ . The address can be latched using signal ALE, as shown in Fig. 1.10. Commercially available latch chips contain eight latches. Thus for demultiplexing twenty address lines one requires three latch chips like 74373. While demultiplexing the address bus, two of the three latch chips will be fully used and four latches of the third chip will be used. Figure 1.10 shows arrangement for latching the twenty bit address. Di indicate D inputs of latches and Q indicate the respective latch outputs.



**Fig.1.10 Latching 20-Bit Address of 8086**

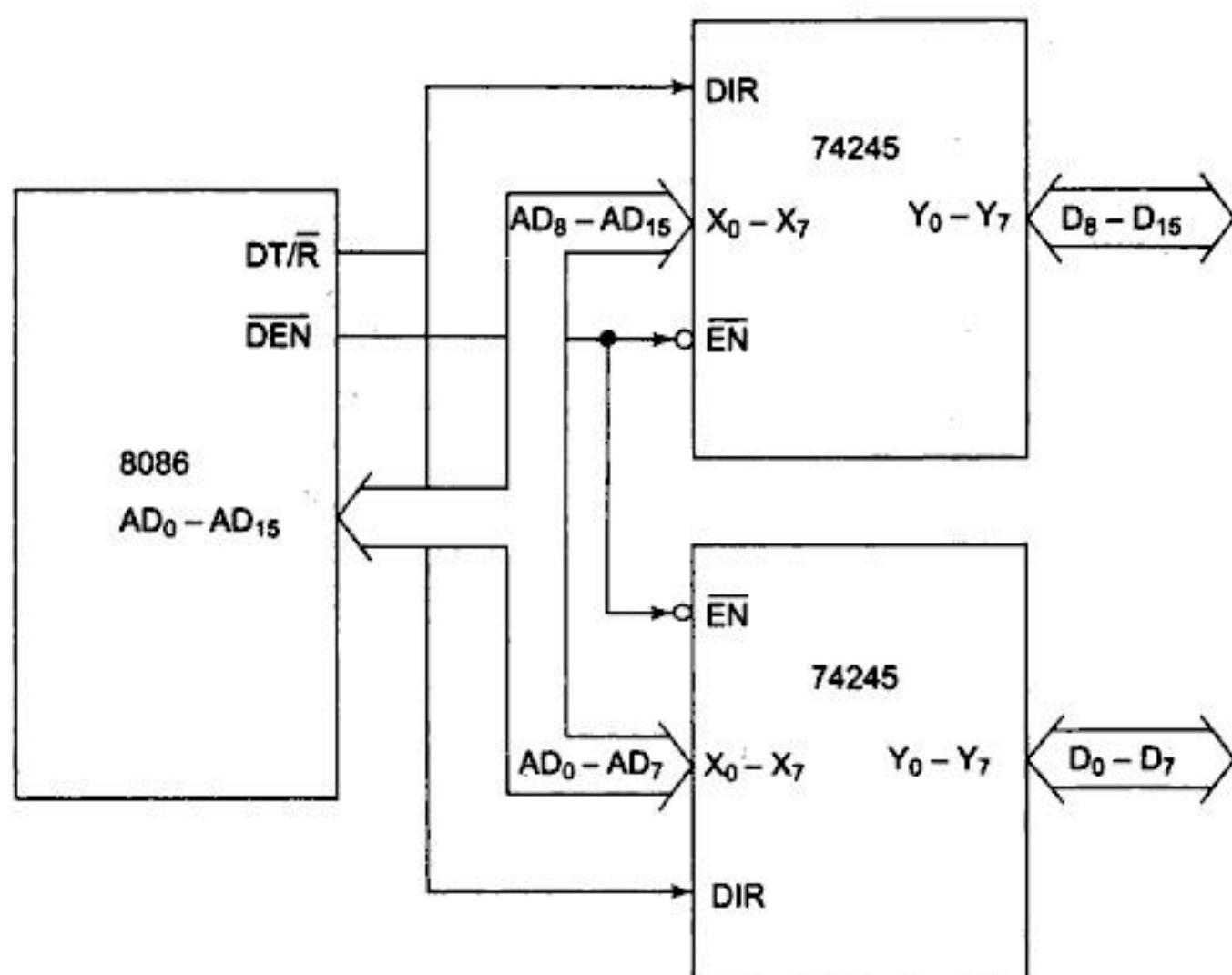
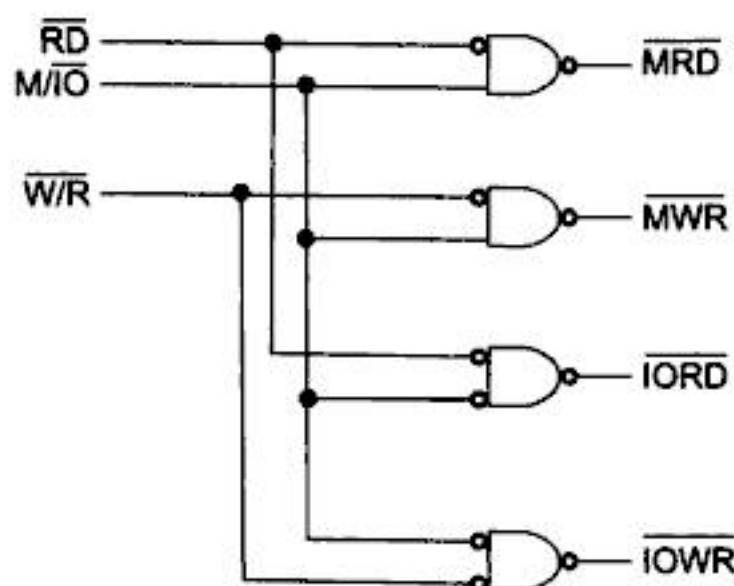
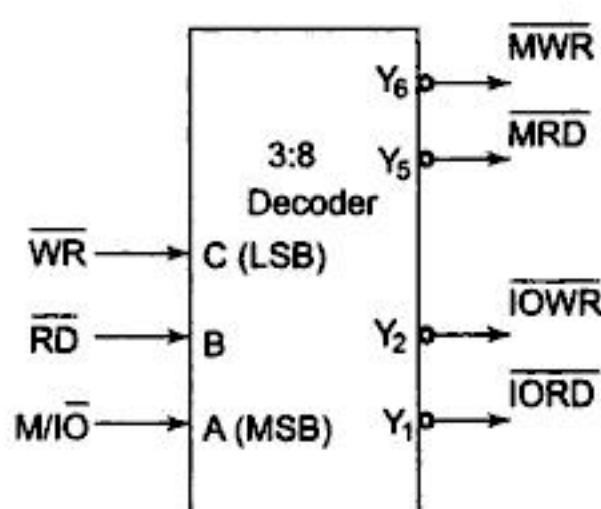
8086 has multiplexed 16-bit data bus in the form of  $AD_0 - AD_{15}$ . The data can be separated from the address and buffered using two bidirectional buffers 74245. It may be noted that the data can either be transferred from microprocessor to memory or from memory to microprocessor in case of write or read operations respectively hence bidirectional buffers are required for deriving the data bus. The signals

$\overline{DEN}$  and  $\overline{DT/R}$  indicate the presence of data on the bus and the direction of the data, i.e. to / from the microprocessor. They are used to drive the chip select (enable) and direction pins of the buffers as indicated below in Fig. 1.11.

If  $\overline{DEN}$  is low it indicates that the data is available on the multiplexed bus and both the buffers (74245) are enabled to transfer data. When DIR pin goes high the data available at X pins of 74245 are transferred to Y pins, i.e. data is transmitted from microprocessor to either memory or IO device (write operation). If DIR pin goes low the data available at Y pins of 74245 is transferred to X pins, i.e. data is received by microprocessor from memory or IO device (read operation).

For deriving control bus from the available control signals  $\overline{RD}$ ,  $\overline{WR}$  and  $M/IO$  in case of minimum mode of operation any combinational logic circuit may be used as shown in Fig. 1.12 (a) and Fig. 1.12 (b).

In case of maximum mode of operation a chip bus controller derives all the control signals using status signals  $\overline{S}_0, \overline{S}_1$  and  $\overline{S}_2$ .


**Fig.1.11** Buffering Data Bus of 8086

**Fig.1.12 (a)** Deriving 8086 Control Signals

**Fig.1.12 (b)** Deriving 8086 Control Signals

## 1.8 MINIMUM MODE 8086 SYSTEM AND TIMINGS

In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping its MN/MX pin to logic 1. In this mode, all the control signals are given out by the microprocessor chip itself. There is a single microprocessor in the minimum mode system. The remaining components in the system are latches, transreceivers, clock generator, memory and I/O devices. Some type of chip selection logic may be required for selecting memory or I/O devices, depending upon the address map of the system.

The latches are generally buffered output D-type flip-flops, like, 74LS373 or 8282. They are used for separating the valid address from the multiplexed address/data signals and are controlled by the ALE signal generated by 8086. Transreceivers are the bidirectional buffers and sometimes they are called data amplifiers. They are required to separate the valid data from the time multiplexed address/data signal. They are controlled by two signals, namely, DEN and DT/R. The DEN signal indicates that the valid data is available on the data bus, while DT/R indicates the direction of data, i.e. from / to the processor. The system contains memory for the monitor and users program storage. Usually, EPROMS are used for monitor storage, while RAMs for users' program storage. A system may contain I/O devices for communication with the processor as well as some special purpose I/O devices. The clock generator (IC8284) generates the clock from the crystal oscillator and then shapes it to make it more precise so that it can be used as an accurate timing reference for the system. The clock generator also synchronizes some external signals with the system clock. The general system organisation is shown in Fig. 1.13. Since it has 20 address lines and 16 data lines, the 8086 CPU requires three octal address latches and two octal data buffers for the complete address and data separation.

The working of the minimum mode configuration system can be better described in terms of the timing diagrams rather than qualitatively describing the operations. The opcode fetch and read cycles are similar. Hence, the timing diagram can be categorized in two parts, the first is the timing diagram for *read cycle* and the second is the timing diagram for *write cycle*.

The read cycle begins in T<sub>1</sub> with the assertion of the Address Latch Enable (ALE) signal and M/IO signal. During the negative going edge of this signal, the valid address is latched on the local bus. The BHE and A<sub>0</sub> signals address low, high or both bytes. From T<sub>1</sub> to T<sub>4</sub>, the M/IO signal indicates a memory or I/O operation. At T<sub>2</sub>, the address is removed from the local bus and is sent to the output. The bus is then tristated. The Read (RD) control signal is also activated in T<sub>2</sub>. This signal causes the addressed device to enable its data bus drivers. After RD goes low, the valid data is available on the data bus. The addressed device will drive the READY line high. When the processor returns the read signal to high level, the addressed device will again tristate its bus drivers. CS logic indicates chip select logic and 'e' and 'O' suffixes indicate even and odd address memory banks.

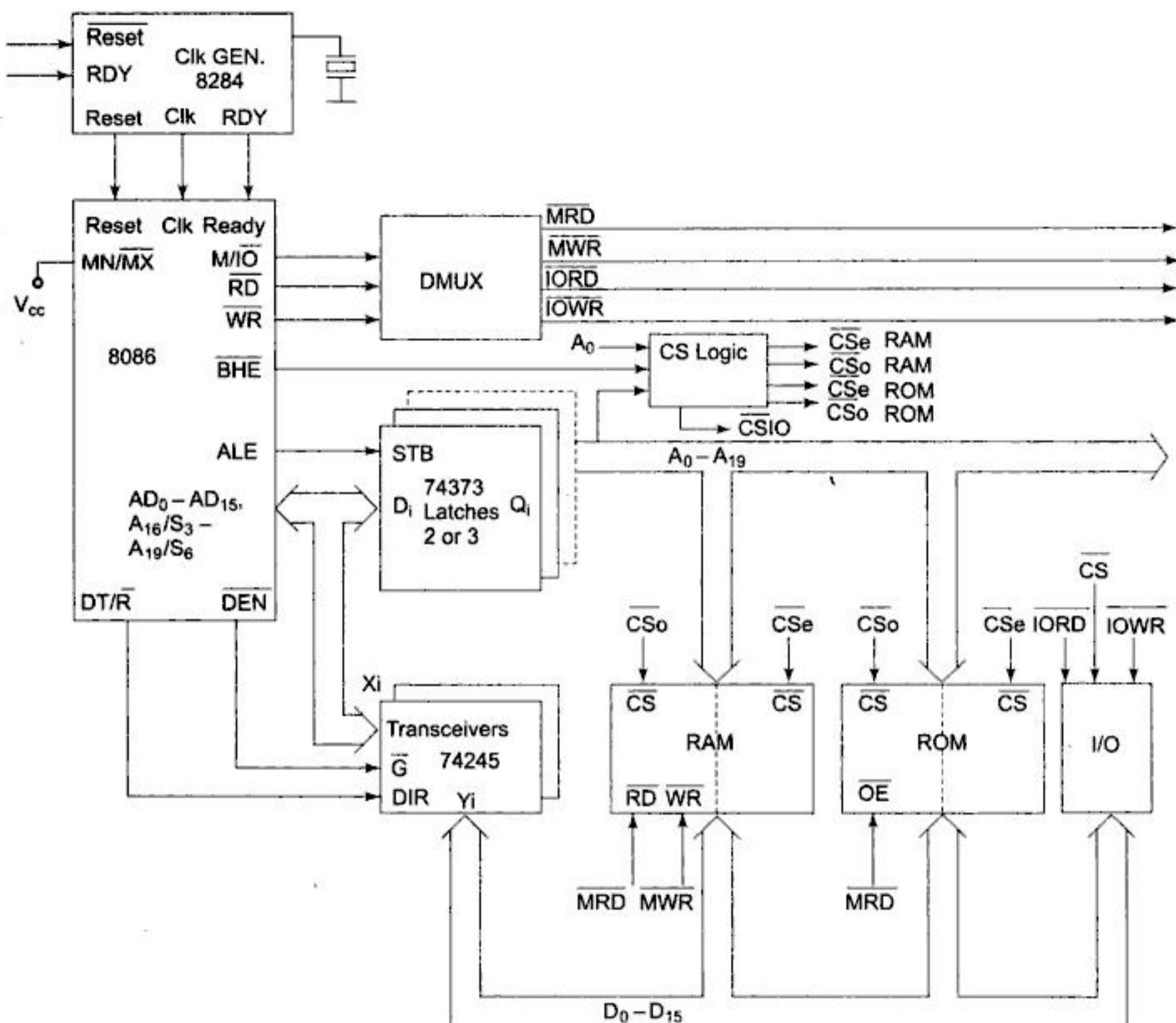
A write cycle also begins with the assertion of ALE and the emission of the address. The M/IO signal is again asserted to indicate a memory or I/O operation. In T<sub>2</sub>, after sending the address in T<sub>1</sub>, the processor sends the data to be written to the addressed location. The data remains on the bus until the middle of T<sub>4</sub> state. The WR becomes active at the beginning of T<sub>2</sub> (unlike RD is somewhat delayed in T<sub>2</sub> to provide time for floating).

The BHE and A<sub>0</sub> signals are used to select the proper byte or bytes of memory or I/O word to be read or written as already discussed in the signal description section of this chapter.

The M/IO, RD and WR signals indicate the types of data transfer as specified in Table 1.5.

**Table 1.5**

M/IO	RD	DEN	Transfer Type
0	0	1	I/O read
0	1	0	I/O write
1	0	1	Memory read
1	1	0	Memory write



**Fig. 1.13 Minimum Mode 8086 System**

Figure 1.14(a) shows the read cycle while Fig. 1.14(b) shows the write cycle.

### 1.8.1 HOLD Response Sequence

The **HOLD** pin is checked at the end of each bus cycle. If it is received active by the processor before  $T_4$  of the previous cycle or during  $T_1$  state of the current cycle, the CPU activates **HLDA** in the next clock cycle and for the succeeding bus cycles, the bus will be given to another requesting master. The control of the bus is not regained by the processor until the requesting master does not drop the **HOLD** pin low. When the request is dropped by the requesting master, the **HLDA** is dropped by the processor at the trailing edge of the next clock, as shown in Fig. 1.14 (c). The other conditions have already been discussed in the signal description section for the **HOLD** and **HLDA** signals.

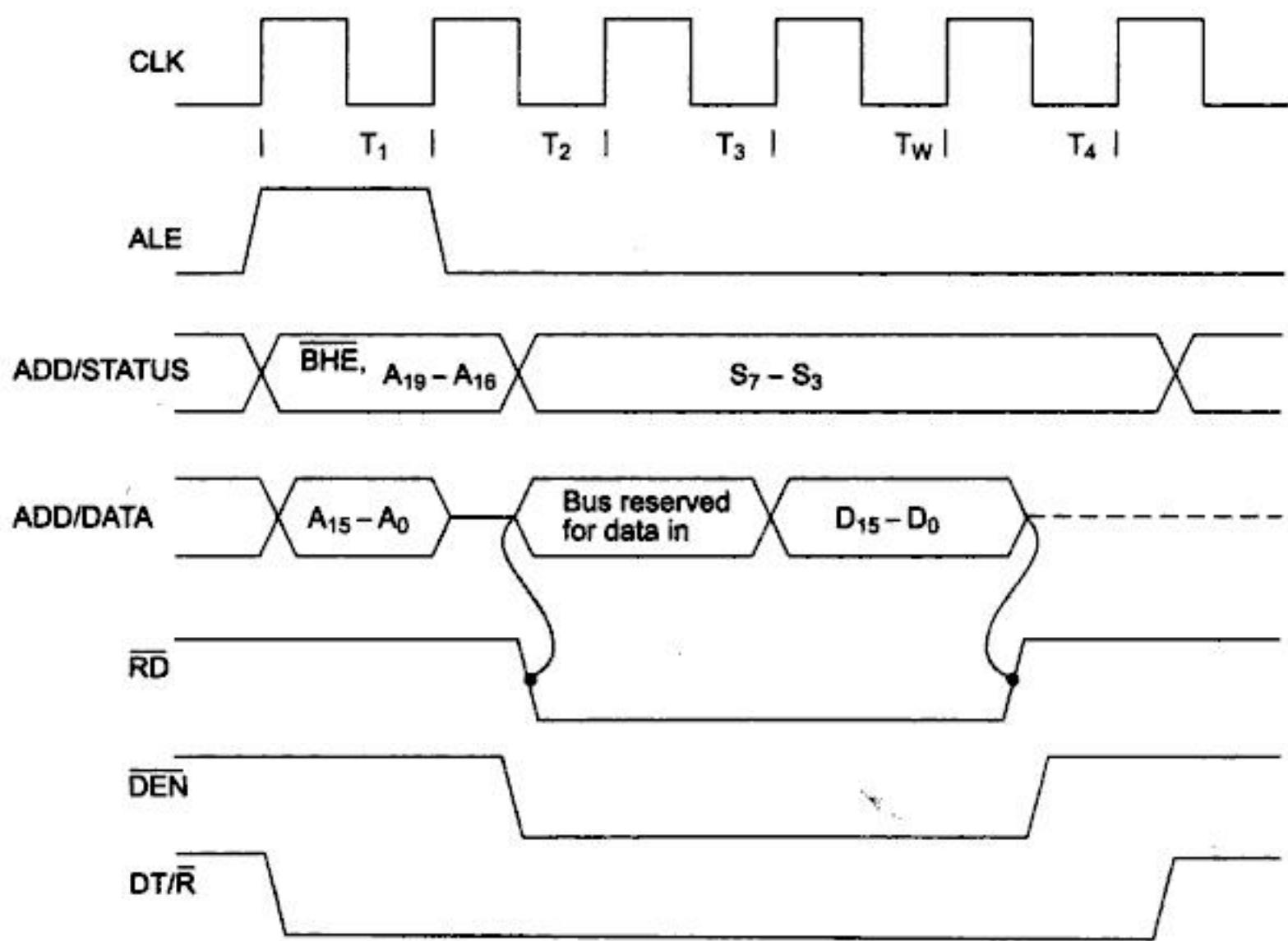


Fig. I.14(a) Read Cycle Timing Diagram for Minimum Mode

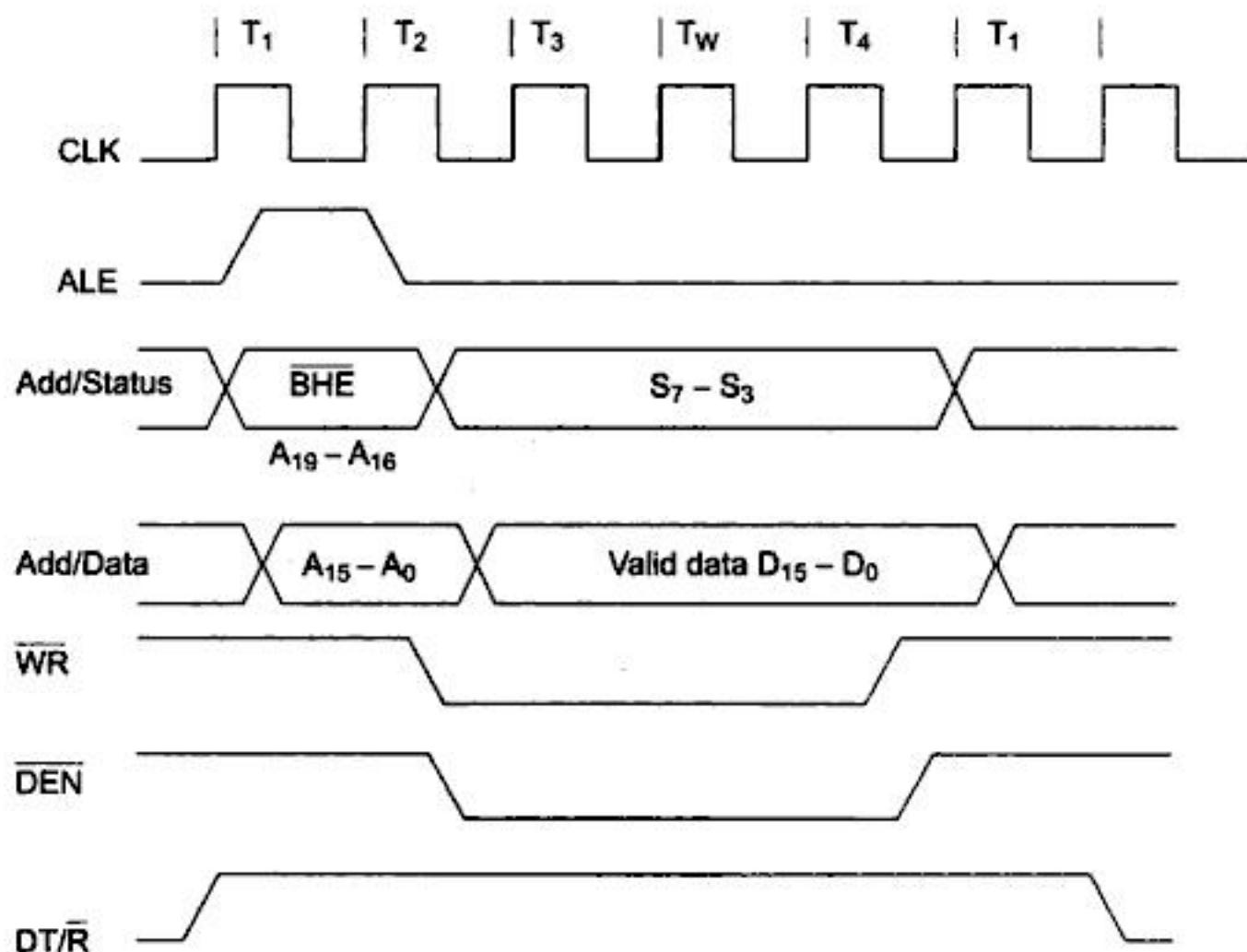
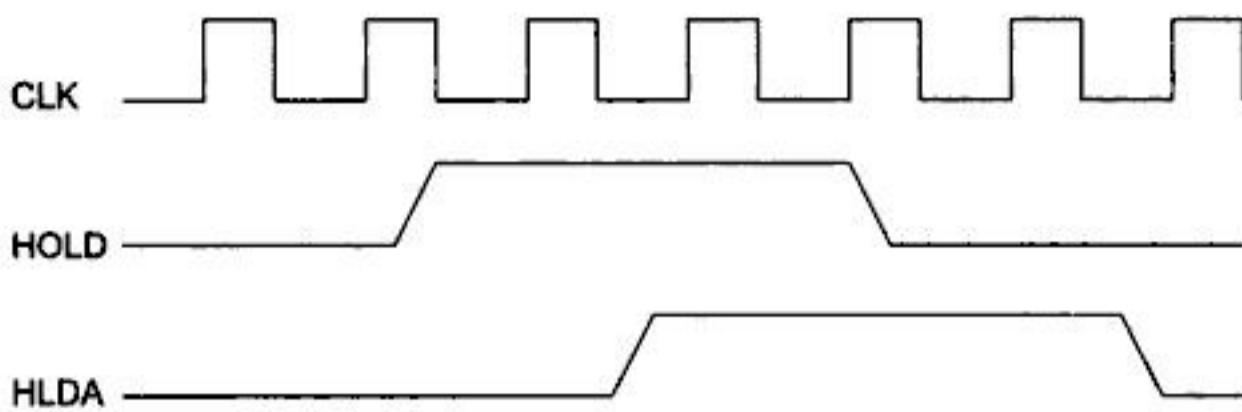


Fig. I.14(b) Write Cycle Timing Diagram for Minimum Mode Operation



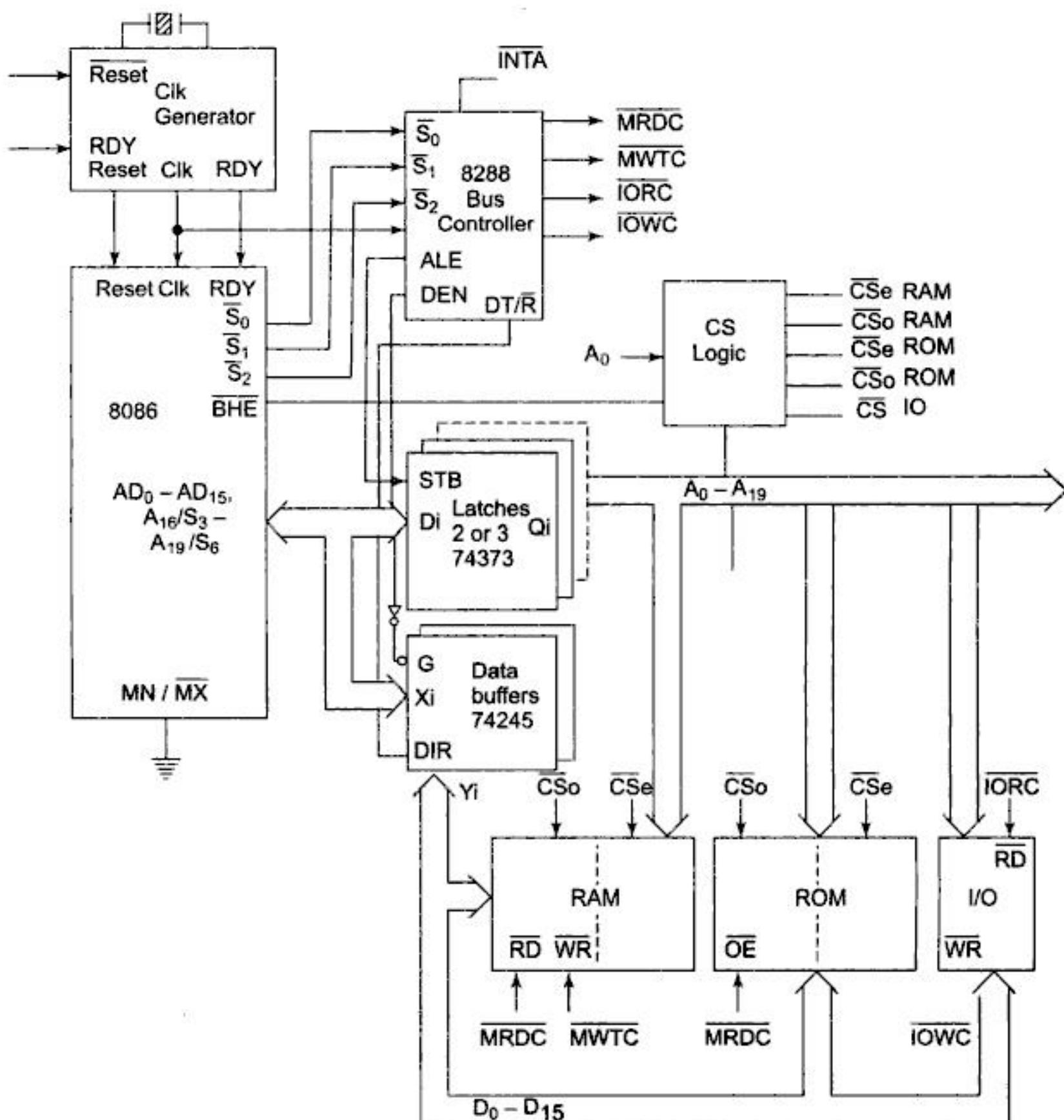
**Fig. 1.14(c) Bus Request and Bus Grant Timings in Minimum Mode System**

## 1.9 MAXIMUM MODE 8086 SYSTEM AND TIMINGS

In the maximum mode, the 8086 is operated by strapping the MN/  $\overline{MX}$  pin to ground. In this mode, the processor derives the status signals  $\overline{S}_2$ ,  $\overline{S}_1$ , and  $\overline{S}_0$ . Another chip called bus controller derives the control signals using this status information. In the maximum mode, there may be more than one microprocessor in the system configuration. The other components in the system are the same as in the minimum mode system. In this section, we will study the bus controller chip and its functions in brief. The functions of all the pins having special functions in maximum mode have already been discussed in the pin diagram section of this chapter.

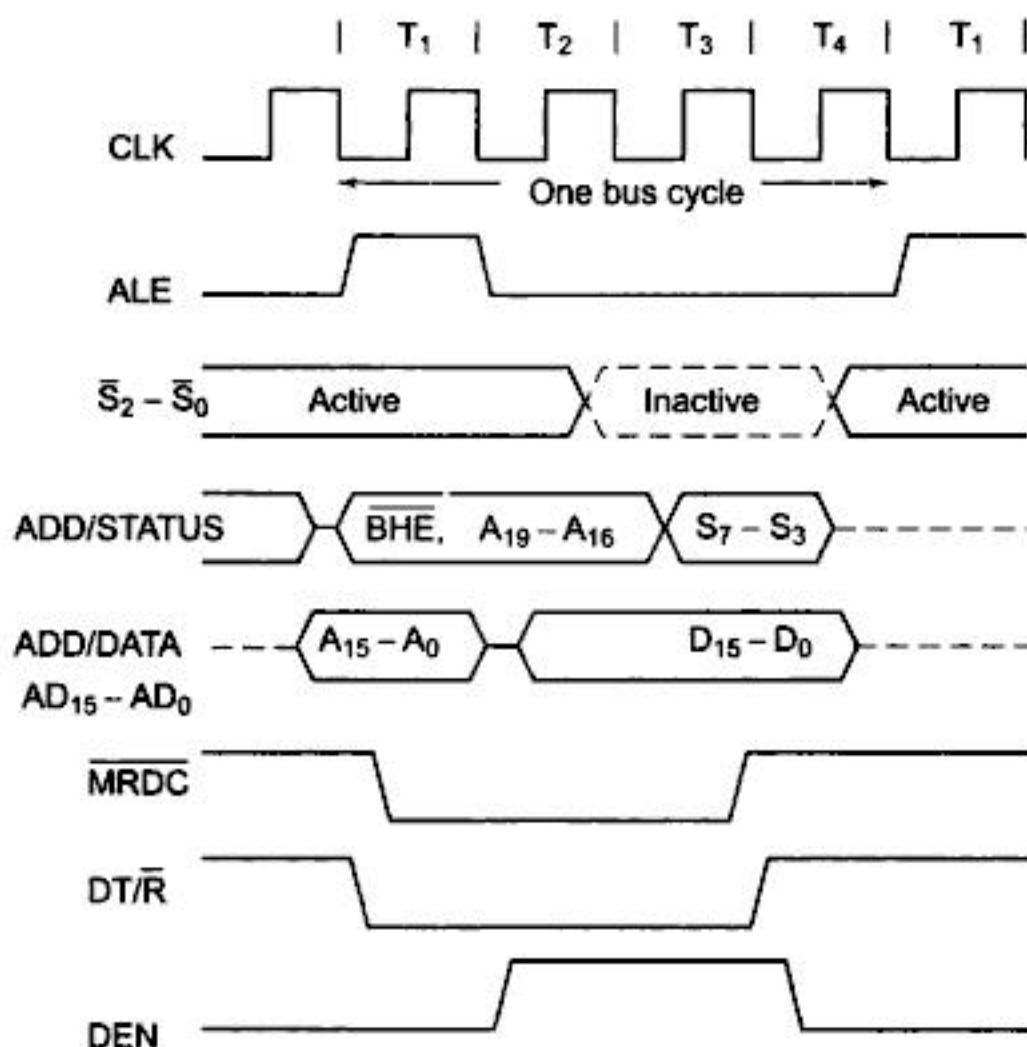
The basic functions of the bus controller chip IC8288, is to derive control signals like  $\overline{RD}$  and  $\overline{WR}$  (for memory and I/O devices),  $\overline{DEN}$ ,  $\overline{DT/R}$ ,  $\overline{ALE}$ , etc. using the information made available by the processor on the status lines. The bus controller chip has input lines  $\overline{S}_2$ ,  $\overline{S}_1$  and  $\overline{S}_0$  and CLK, which are driven by the CPU. It derives the outputs ALE, DEN, DT/ $\overline{R}$ ,  $\overline{MRDC}$ ,  $\overline{MWTC}$ ,  $\overline{AMWC}$ ,  $\overline{IORC}$ ,  $\overline{IOWC}$  and  $\overline{AIOWC}$ . The  $\overline{AEN}$ , IOB and CEN pins are specially useful for multiprocessor systems.  $\overline{AEN}$  and IOB are generally grounded. CEN pin is usually tied to +5V. The significance of the MCE/ $\overline{PDEN}$  output depends upon the status of the IOB pin. If IOB is grounded, it acts as master cascade enable to control cascaded 8259A, else it acts as peripheral data enable used in the multiple bus configurations. INTA pin is used to issue two interrupt acknowledge pulses to the interrupt controller or to an interrupting device.

$\overline{IORC}$ ,  $\overline{IOWC}$  are I/O read command and I/O write command signals respectively. These signals enable an IO interface to read or write the data from or to the addressed port. The  $\overline{MRDC}$ ,  $\overline{MWTC}$  are memory read command and memory write command signals respectively and may be used as memory read and write signals. All these command signals instruct the memory to accept or send data to or from the bus. For both of these write command signals, the advanced signals namely  $\overline{AIOWC}$  and  $\overline{AMWTC}$  are available. They also serve the same purpose, but are activated one clock cycle earlier than the  $\overline{IOWC}$  and  $\overline{MWTC}$  signals, respectively. The maximum mode system is shown in Fig. 1.15.

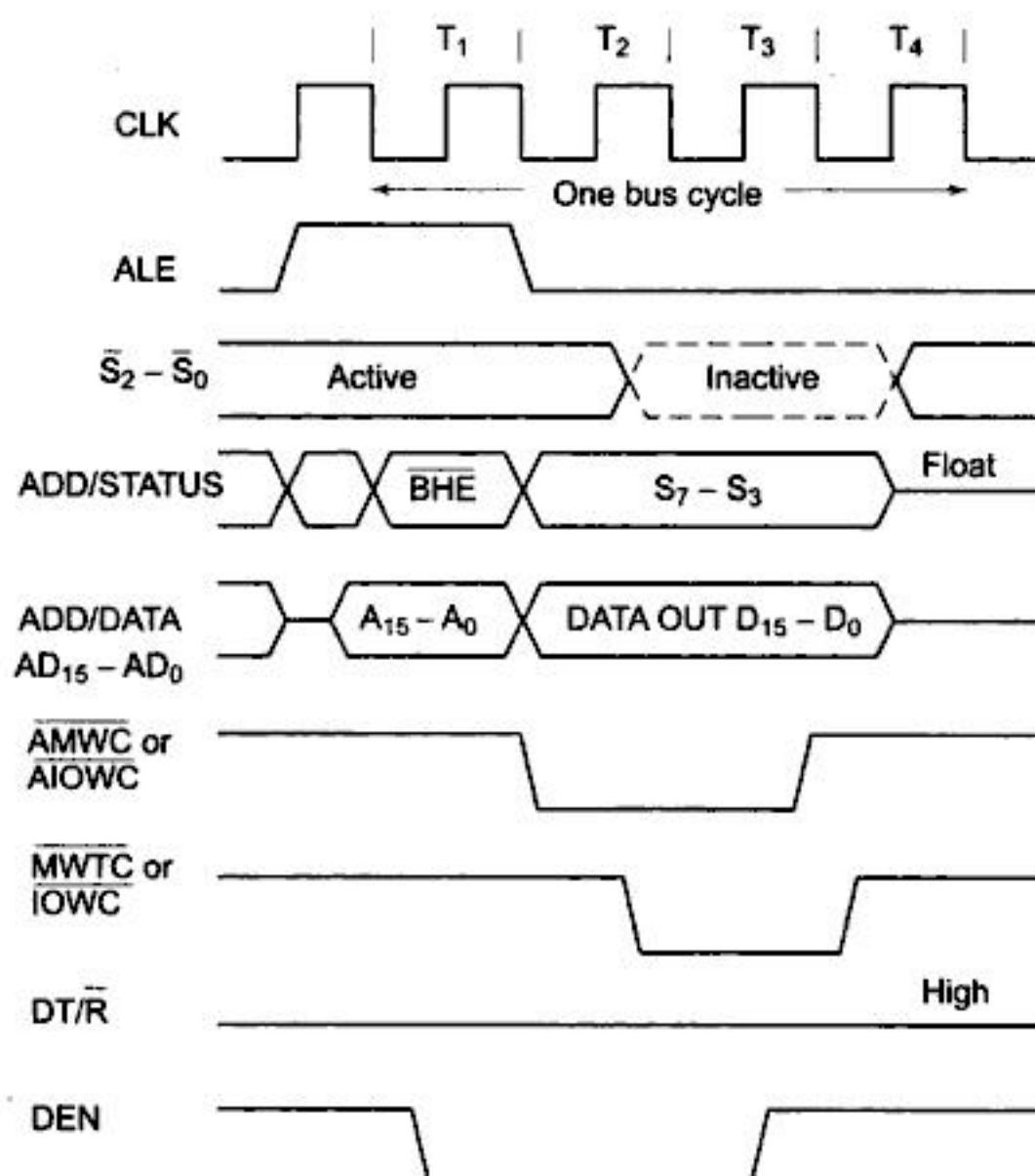


**Fig. 1.15 Maximum Mode 8086 System**

The maximum mode system timing diagrams are also divided in two portions as read (input) and write (output) timing diagrams. The address/data and address/status timings are similar to the minimum mode. ALE is asserted in T<sub>1</sub>, just like minimum mode. The only difference lies in the status signals used and the available control and advanced command signals. Figure 1.16 (a) shows the maximum mode timings for the read operation while the Fig. 1.16 (b) shows the same for the write operation. The CS Logic block represents chip select logic and the 'e' and 'O' suffixes indicate even and odd address memory bank.



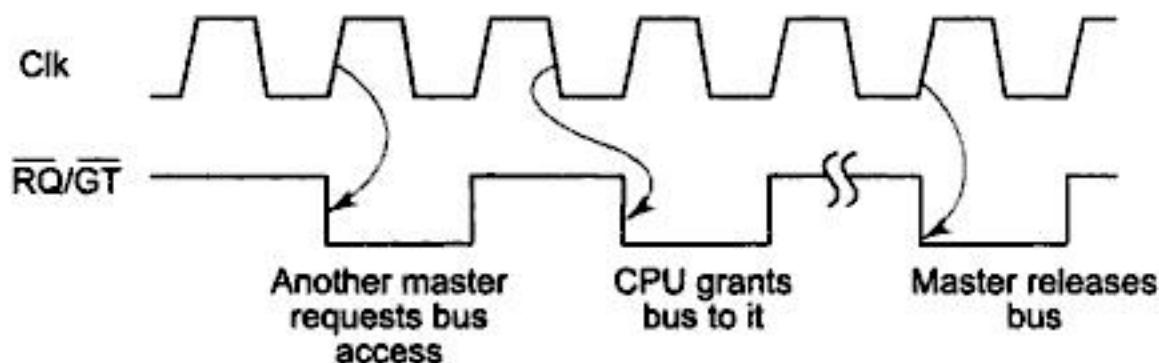
**Fig. 1.16 (a)** Memory Read Timing in Maximum Mode



**Fig. 1.16(b)** Memory Write Timing in Maximum Mode

### 1.9.1 Timings for $\overline{RQ}/\overline{GT}$ Signals

The request/grant response sequence contains a series of three pulses as shown in the timing diagram Fig.1.16 (c). The request/grant pins are checked at each rising pulse of clock input. When a request is detected and if the conditions discussed in pin diagram section of this chapter for valid HOLD request are satisfied, the processor issues a grant pulse over the  $\overline{RQ}/\overline{GT}$  pin immediately during the  $T_4$  (current) or  $T_1$  (next) state. When the requesting master receives this pulse, it accepts the control of the bus. The requesting master uses the bus till it requires. When it is ready to relinquish the bus, it sends a release pulse to the processor (host) using the  $\overline{RQ}/\overline{GT}$  pin. This sequence is shown in Fig. 1.16 (c).



**Fig. 1.16 (c)  $\overline{RQ}/\overline{GT}$  Timings in Maximum Mode**

## 1.10 THE PROCESSOR 8088

The launching of the processor 8086 is seen as a remarkable step in the development of high speed computing machines. Before the introduction of 8086, most of the circuits required for the different applications in computing and industrial control fields were already designed around the 8-bit processor 8085. The 8086 imparted tremendous flexibility in the programming as compared to 8085. So naturally, after the introduction of 8086, there was a search for a microprocessor chip which has the programming flexibility like 8086 and the external interface like 8085, so that all the existing circuits built around 8085 can work as before, with this new chip. The chip 8088 was a result of this demand. The microprocessor 8088 has all the programming facilities that 8086 has, along with some hardware features of 8086, like 1Mbyte memory addressing capability, operating modes (MN/MX), interrupt structure etc. However, 8088, unlike 8086, has 8-bit data bus. This feature of 8088 makes the circuits, designed around 8085, compatible with 8088, with little or no modification.

All the peripheral interfacing schemes with 8088 are the same as those for the 8-bit processors. The memory and I/O addressing schemes are now exactly similar to 8085 schemes except for the increased memory (1Mbyte) and I/O (64Kbyte) capabilities. The architecture shows the developments in 8088 over 8086. The abilities and limitations of 8088 are same as 8086. In this section, we will discuss those properties of 8088 which are different from that of 8086 in some respects.

### 1.10.1 Architecture and Signal Description of 8088

The register set of 8088 is exactly the same as that of 8086. The architecture of 8088 is also similar to 8086 except for two changes; a) 8088 has 4-byte instruction queue and b) 8088 has 8-bit data bus. The function of each block is the same as in 8086. Figure 1.17 shows the 8088 architecture.

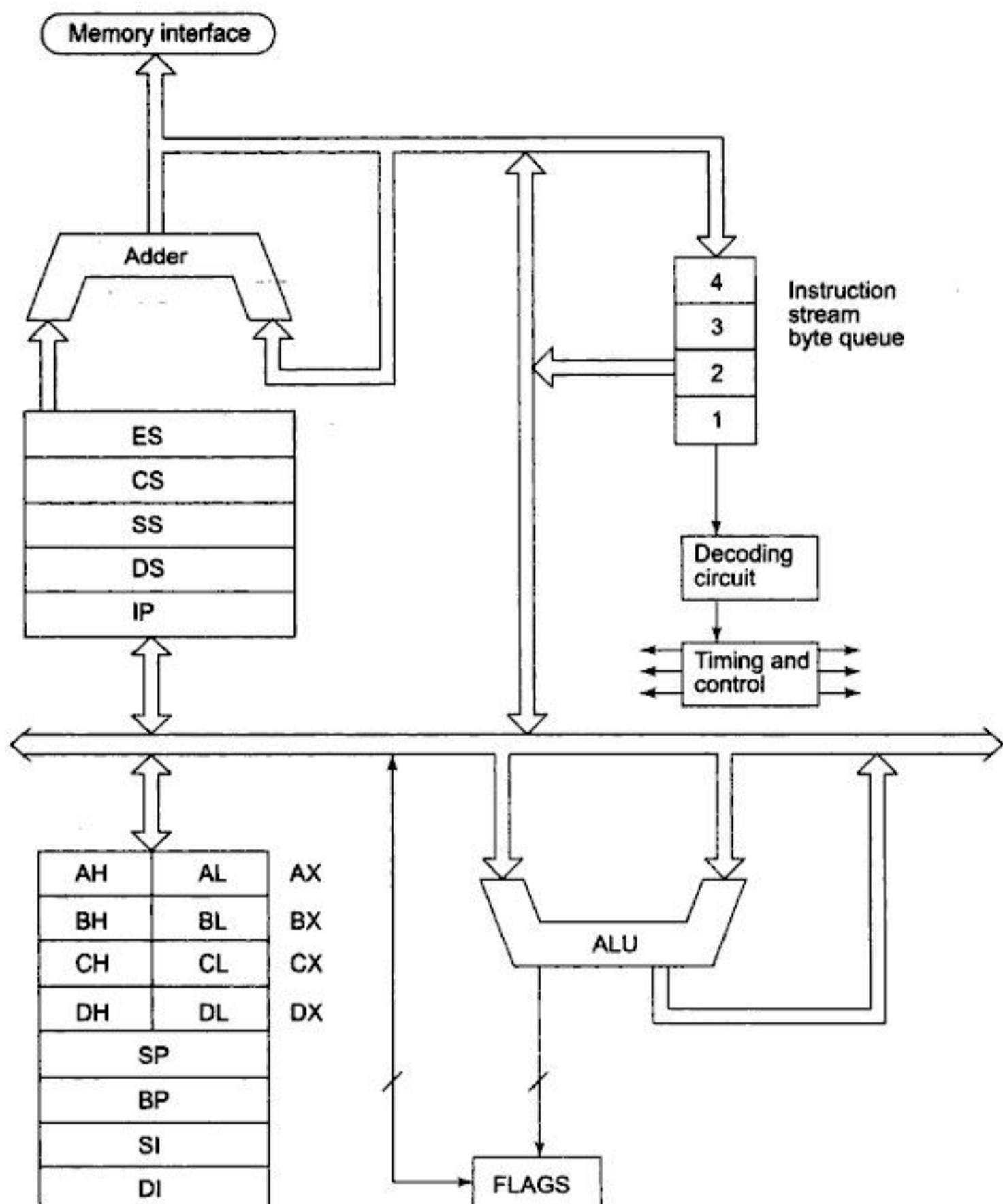


Fig. 1.17 Architecture of 8088

The addressing capability of 8088 is 1Mbyte, therefore, it needs 20 address bits, i.e. 20 addressing lines. While handling this 20-bit address, the segmented memory scheme is used and the complete physical address forming procedure is the same as explained in case of 8086. The memory organisation and addressing methods of 8088 and 8086 are similar. While physically interfacing memory to 8088, there is nothing like an even address bank or odd address bank. The complete memory is homogeneously addressed as a bank of 1Mbyte memory locations using the segmented memory scheme. This change in hardware is completely transparent to software. As a result of the modified data bus, the 8088 can access only a byte at a time. This fact reduces the speed of operation of 8088 as compared to 8086, but the 8088 can process the 16-bit data internally. On account of this change in bus structure, the 8088 has slightly different timing diagrams than 8086.

The pin diagram of 8088 is shown in Fig. 1.18. Most of the 8088 pins and their functions are exactly similar to the corresponding pins of 8086. Hence the pins that have different functions or timings are discussed in this section. Amongst them are the pins that have a common function in minimum and maximum mode.

	Minimum mode	Maximum mode
GND	1	40 VCC
A <sub>14</sub>	2	39 A <sub>15</sub>
A <sub>13</sub>	3	38 A <sub>16</sub> /S <sub>3</sub>
A <sub>12</sub>	4	37 A <sub>17</sub> /S <sub>4</sub>
A <sub>11</sub>	5	36 A <sub>18</sub> /S <sub>5</sub>
A <sub>10</sub>	6	35 A <sub>19</sub> /S <sub>6</sub>
A <sub>9</sub>	7	34 SS <sub>0</sub> (High)
A <sub>8</sub>	8	33 MN/MX
AD <sub>7</sub>	9	32 RD
AD <sub>6</sub>	10 8088	31 HOLD RQ/GT <sub>0</sub>
AD <sub>5</sub>	11	30 HLDA RQ/GT <sub>1</sub>
AD <sub>4</sub>	12	29 WR LOCK
AD <sub>3</sub>	13	28 IO/M S <sub>2</sub>
AD <sub>2</sub>	14	27 DT/R S <sub>1</sub>
AD <sub>1</sub>	15	26 DEN S <sub>0</sub>
AD <sub>0</sub>	16	25 ALE QS <sub>0</sub>
NMI	17	24 INTA QS <sub>1</sub>
INTR	18	23 TEST
CLK	19	22 READY
GND	20	21 RESET

Fig. 1.18 Pin Diagram of 8088

**AD<sub>7</sub>-AD<sub>0</sub> (Address/Data)** These lines constitute the address/data time multiplexed bus. During T<sub>1</sub> the bus is used for conducting addresses and during T<sub>2</sub>, T<sub>3</sub>, T<sub>w</sub> and T<sub>4</sub> states these lines are used for conducting data. These are tristated during 'hold acknowledge' and 'interrupt acknowledge' cycles.

**A<sub>15</sub>-A<sub>8</sub> (Address Bus)** These lines provide the address bits A<sub>8</sub> to A<sub>15</sub> in the entire bus cycle. These need not be latched for obtaining a stable valid address. These are active high and are tristated during the 'acknowledge' cycles. Note that as the 8088 data bus is only of 8 bits, there is no need of the BHE signal.

**SS<sub>0</sub>** A new pin  $\overline{SS}_0$  is introduced in 8088 instead of  $\overline{BHE}$  pin in 8086. In minimum mode, the pin  $\overline{SS}_0$  is logically equivalent to the  $\overline{S}_0$  in the maximum mode. In maximum mode it is always high.

**IO/M** This pin is similar to M/ $\overline{IO}$  pin of 8086, but it offers an 8085 compatible, memory/ IO bus interface.

The signals  $\overline{SS}_0$ , DT/ $\overline{R}$ , IO/M can be decoded to interpret the activities of the microprocessor as given in Table 1.6, in the minimum mode.

In the maximum mode, the pin  $\overline{SS}_0$  is permanently high. The functions and timings of other pins of 8088 are like that of 8086. Due to the difference in the bus structure, the timing diagrams are somewhat different.

**Table 1.6**

IO/M	DT/ $\overline{R}$	$\overline{SS}_0$	Operation/Interpretation
1	0	0	Interrupt Acknowledge
1	0	1	Read I/O port
1	1	0	Write I/O port
1	1	1	HALT
0	0	0	Code Access
0	0	1	Read memory
0	1	0	Write memory
0	1	1	Passive

### 1.10.2 Deriving 8088 Bus

As discussed earlier, 8088 is a microprocessor with an internal architecture, instruction set and memory addressing capability of 8086 but with the data bus of 8-bits like 8085. The 8-bit data bus makes 8088 compatible with the family of 8-bit peripherals designed around 8085.

For demultiplexing the bus, ALE signal is used to enable address latches. Usually three latch chips like 74373 are used for latching the twenty bit address while one bidirectional buffer chip (74245) is used for buffering the 8-bit data bus. The  $\overline{DEN}$  and DT/ $\overline{R}$  signals are used for buffering the data. The control bus may be derived exactly in the same way as that of 8086. The schematic diagram for deriving demultiplexed address/data bus is shown in Fig. 1.19, while the control bus can be derived as in Figs 1.20 (a) and 1.20 (b).

The minimum and maximum mode systems are also similar to the respective 8086 systems. The 8088 systems require only one data buffer due to the 8-bit data bus. The minimum and maximum mode systems of 8088 are shown in Figs 1.21 (a) and (b) respectively.

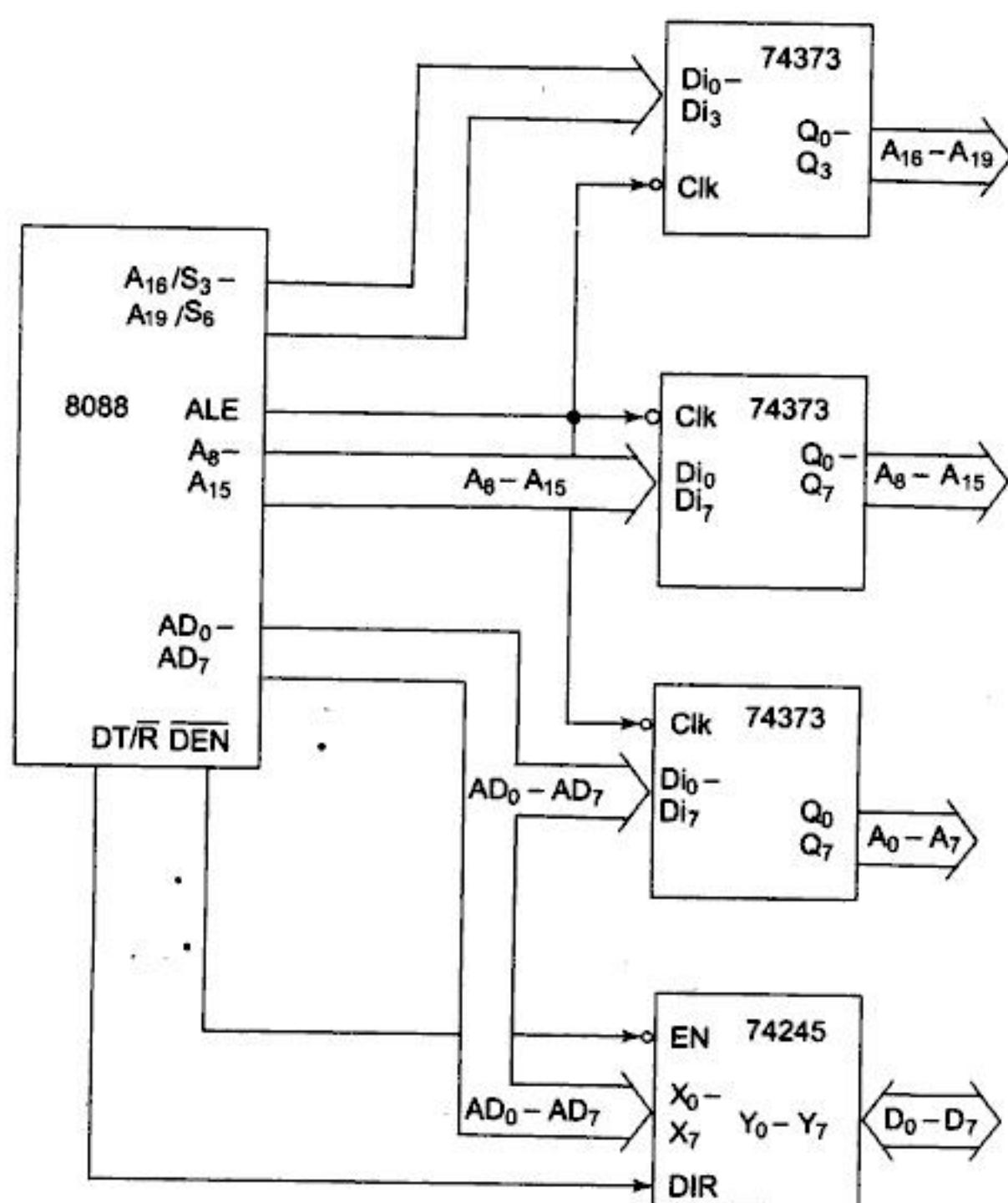


Fig 1.19 Deriving 8088 Address and Data Bus

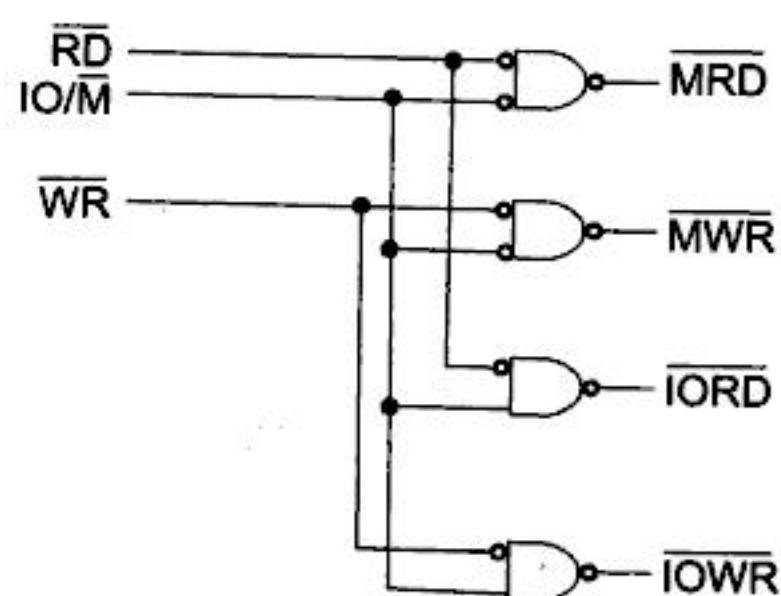


Fig 1.20 (a) Deriving 8088 Control Bus

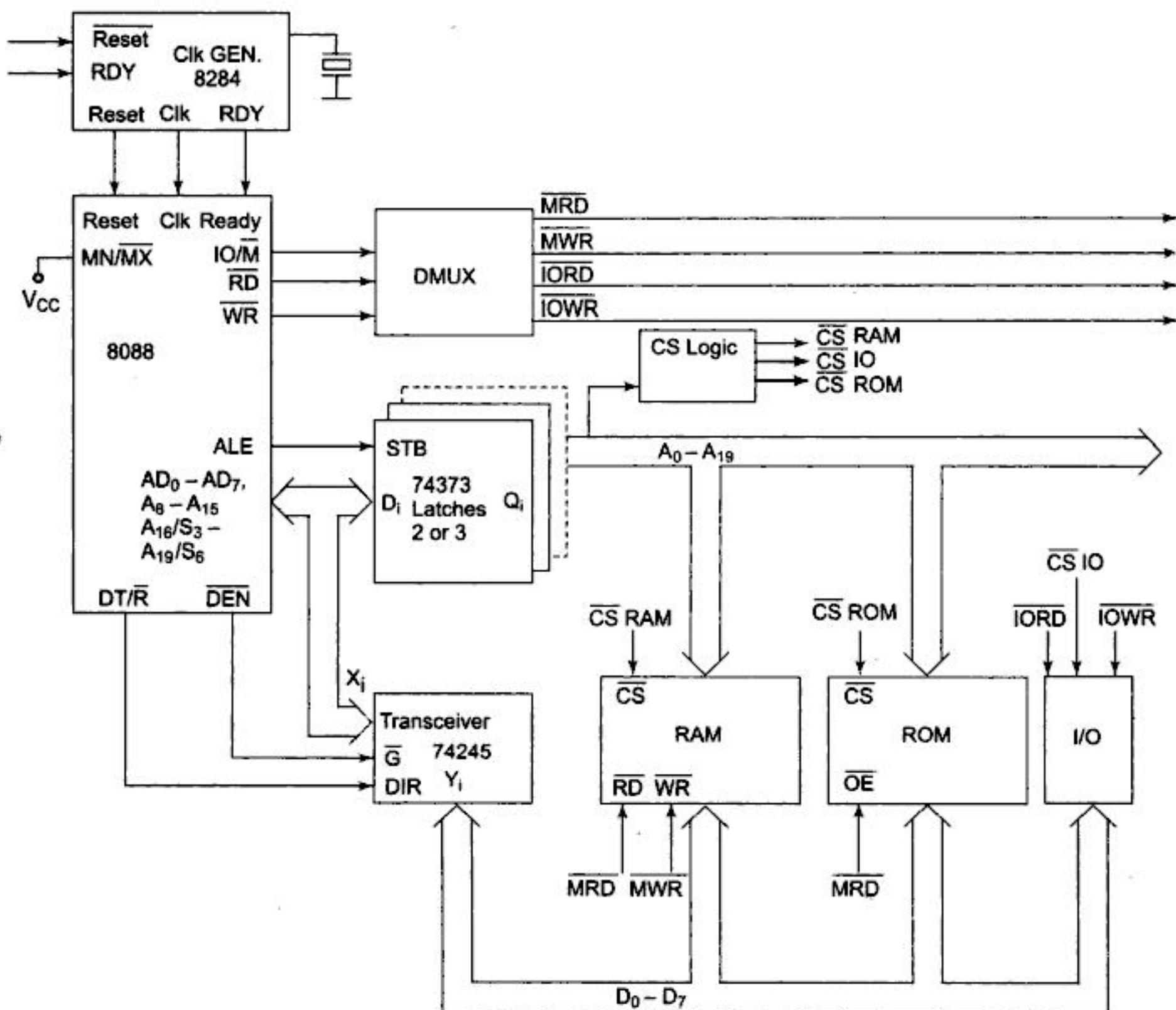
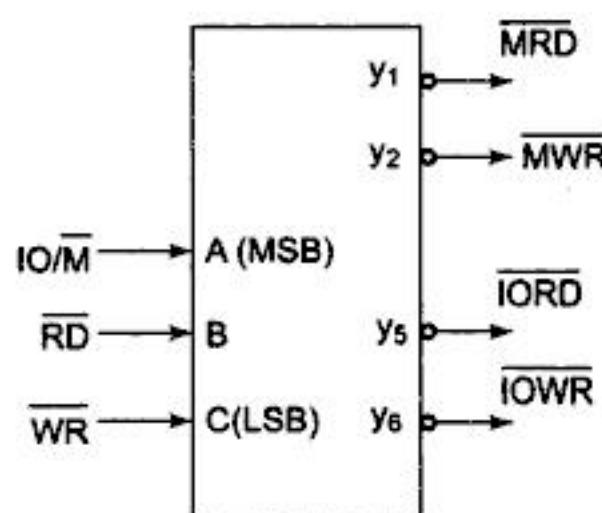


Fig 1.21 (a) Minimum Mode 8088 System

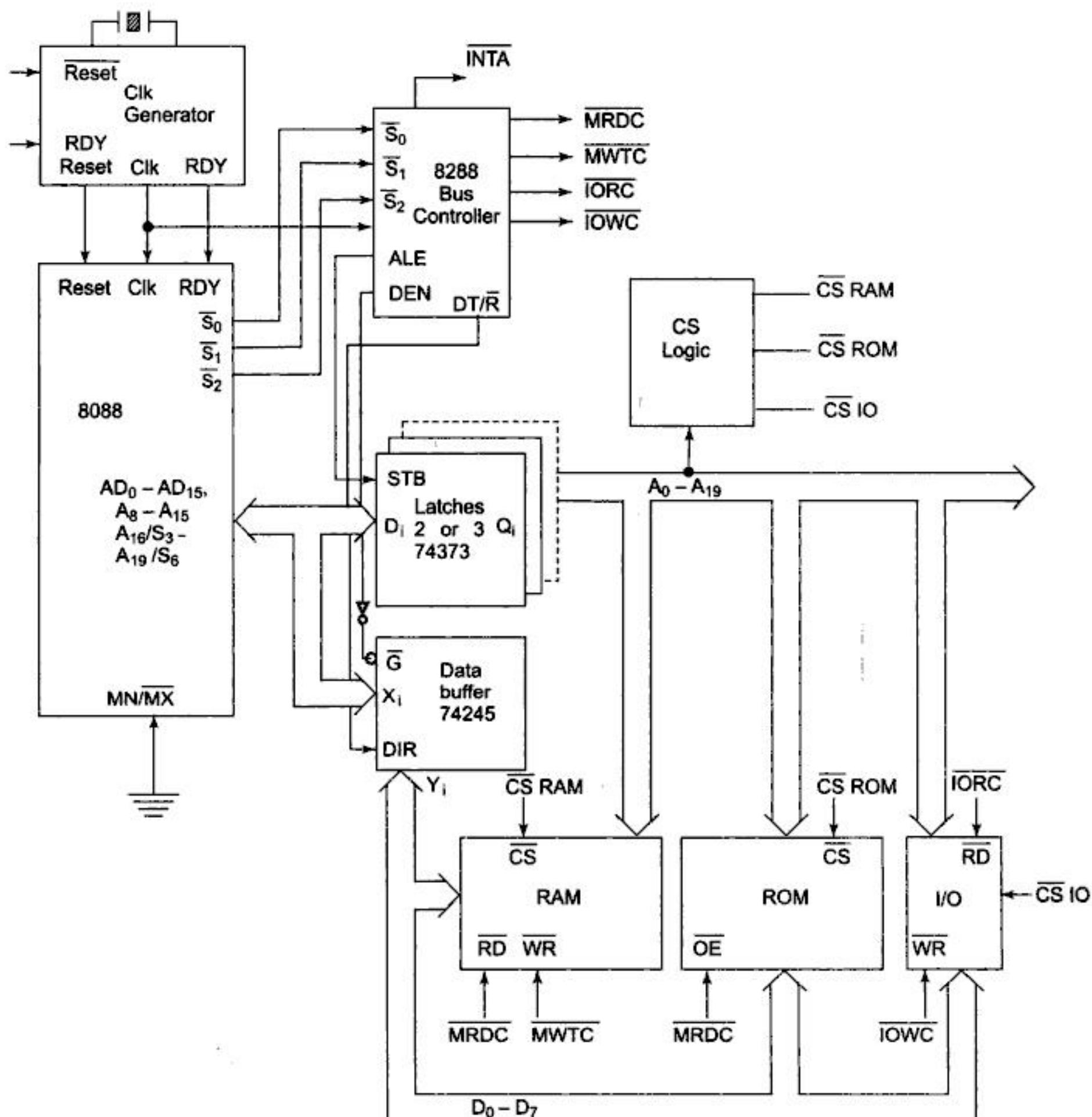


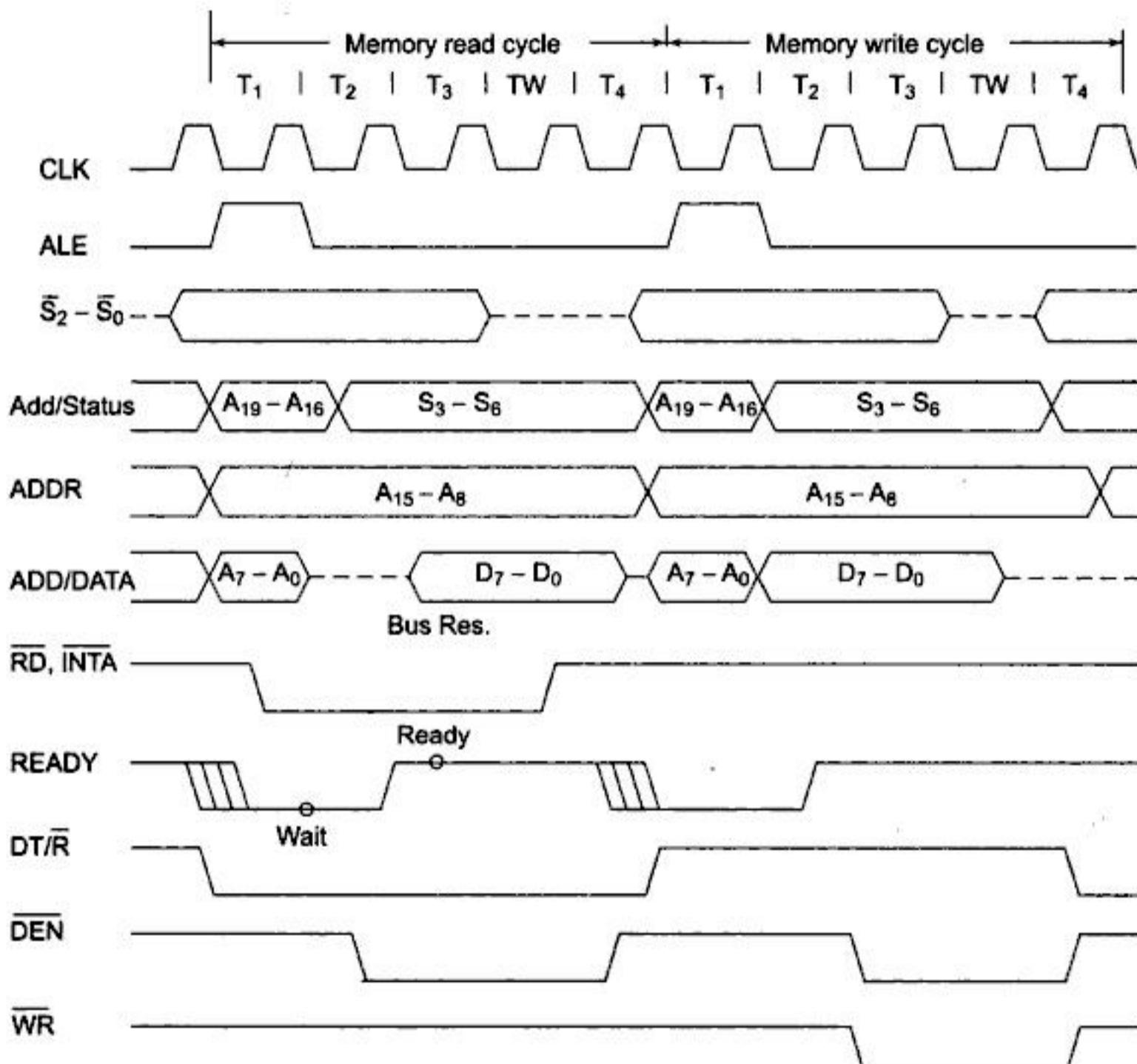
Fig 1.21 (b) Maximum Mode Minimum System of 8088

### 1.10.3 General 8088 System Timing Diagram

The 8088 address/data bus is divided into three parts (a) the lower 8 address/data bits, (b) the middle 8 address bits, and (c) the upper 4 address/status bits. The lower 8 lines are time multiplexed for address and data. The upper 4 lines are time multiplexed for address and status. Each of the bus cycles contains

$T_1$ ,  $T_2$ ,  $T_3$ ,  $T_w$  and  $T_4$  states. The ALE signal goes high for one clock cycle in  $T_1$ . The trailing edge of ALE is used to latch the valid addresses available on the multiplexed lines. They remain valid on the bus for the next cycle ( $T_2$ ). The middle 8 address bits are always present on the bus throughout the bus cycle. The lower order address bus is tristated after  $T_2$  to change its direction for read data operation. The actual data transfer takes place during  $T_3$  and  $T_4$ . Hence the data lines are valid in  $T_3$  or  $T_4$ . The multiplexed bus is again tristated to be ready for the next bus cycle. The status lines are valid over the multiplexed address/status bus for  $T_2$ ,  $T_3$  and  $T_4$  clock cycles.

In case of write cycle, the timing diagram is similar to the read cycle except for the validity of data. In write cycle, the data bits are available on the bus for  $T_2$ ,  $T_3$ ,  $T_w$ , and  $T_4$ . At the end of  $T_4$ , the bus is tristated. The other signals  $\overline{RD}$ ,  $\overline{WR}$ ,  $\overline{INTA}$ ,  $DT/\overline{R}$ ,  $\overline{DEN}$  and READY are similar to the 8086 timing diagram. Figure 1.22 shows the details of read and write bus cycles of 8088.



**Fig. 1.22 Read and Write Cycle Timing Diagram of 8088**

#### 1.10.4 Comparison between 8086 and 8088

The 8088, with an 8-bit external data bus, has been designed for internal 16-bit processing capability. Nearly all the internal functions of 8088 are identical to 8086. The 8088 uses the external bus in the same way as 8086, but only 8 bits of external data are accessed at a time. While fetching or writing the 16-bit data, the task is performed in two consecutive bus cycles. As far as the software is concerned, the chips are identical, except in case of timings. The 8088, thus may take more time for execution of a particular task as compared to 8086.

All the changes in 8088 over 8086 are directly or indirectly related to the 8-bit, 8085 compatible data and control bus interface.

1. The predecoded code queue length is reduced to 4 bytes in 8088, whereas the 8086 queue contains 6 bytes. This was done to avoid the unnecessary prefetch operations and optimize the use of the bus by BIU while prefetching the instructions.
2. The 8088 bus interface unit will fetch a byte from memory to load the queue each time, if at least 1 byte is free. In case of 8086, at least 2 bytes should be free for the next fetch operation.
3. The overall execution time of the instructions in 8088 is affected by the 8-bit external data bus. All the 16-bit operations now require additional 4 clock cycles. The CPU speed is also limited by the speed of instruction fetches.

The pin assignments of both the CPUs are nearly identical, however, they have the following functional changes.

1.  $A_8 - A_{15}$  already latched, all time valid address bus.
2.  $\overline{BHE}$  has no meaning as the data bus is of 8-bits only.
3.  $\overline{SS}_0$  provides the  $S_0$  status information in minimum mode.
4.  $\overline{IO/M}$  has been inverted to be compatible with 8085 bus structure.

#### SUMMARY

In this chapter, we have presented the internal architecture and signal descriptions of 8086. The functional details of the architecture, like register set, flags and segmented memory organisation are also discussed in significant details. Further, general bus cycle operations have been described with the help of timing diagrams. Then minimal 8086 systems have been presented for the minimum and maximum modes of operation. A software compatible processor-8088 has been discussed in the light of the modifications in it over 8086. To conclude with, the basic bus cycle operations and the timing diagrams of 8088 were discussed along with its comparison with 8086. This chapter has elaborated the architectural and functional concepts of the processors 8086 and 8088. The instruction set and programming techniques have been discussed in the following chapters.

#### Exercises

- 1.1 Draw and discuss the internal block diagram of 8086.
- 1.2 What do you mean by pipelined architecture? How is it implemented in 8086?

- 1.3 Explain the concept of segmented memory? What are its advantages?
- 1.4 Explain the physical address formation in 8086.
- 1.5 Draw the register organisation of 8086 and explain typical applications of each register.
- 1.6 Draw and discuss flag register of 8086 in brief.
- 1.7 Explain the function of the following signals of 8086.

(i) ALE	(ii) DT/R	(iii) DEN	(iv) LOCK
(v) TEST	(vi) MN/MX	(vii) BHE	(viii) M/IO
(ix) RQ/GT	(x) QS <sub>0</sub>	(xi) READY	(xii) NMI
(xiii) INTR	(xiv) HOLD	(xv) HLDA	
- 1.8 Explain the function of opcode prefetch queue in 8086.
- 1.9 How does 8086 differentiate between an opcode and instruction data?
- 1.10 Explain the physical memory organisation in an 8086 system.
- 1.11 What is the maximum memory addressing and I/O addressing capability of 8086?
- 1.12 Draw and discuss the read and write cycle timing diagrams of 8086 in minimum mode.
- 1.13 Draw and discuss the read and write cycle timing diagram of 8086 in maximum mode.
- 1.14 From which address the 8086 starts execution after reset?
- 1.15 How will you synchronise an external phenomenon like energising a relay with a program segment execution?
- 1.16 Draw and discuss a typical minimum mode 8086 system.
- 1.17 Draw and discuss a typical maximum mode 8086 system. What is the use of a bus controller in maximum mode?
- 1.18 Bring out the architectural and signal differences between 8086 and 8088.
- 1.19 What may be the reason for developing an externally 8-bit processor like 8088 after the 8086, when a 16-bit processor had already been introduced?
- 1.20 Explain the signal SS<sub>0</sub> of 8088.
- 1.21 Compare the bus interface of 8085 with 8088.
- 1.22 Draw and discuss a typical minimum mode 8088 system.
- 1.23 Draw and discuss a typical maximum mode 8088 system.
- 1.24 Draw and discuss a general 8088 system timing diagram.
- 1.25 What are the functions of the clock generator IC 8284, in the 8086/8088 systems?

# 8086/8088 Instruction Set and Assembler Directives

## INTRODUCTION

In Chapter 1, we have discussed the 8086/8088 architecture, pin diagrams and timing diagrams of read and write cycles. This chapter aims at introducing the readers with the general instruction formats, different addressing modes supported by 8086/8088 along with 8086/8088 instruction set. Further, a few important and frequently used assembler directives and operators have also been discussed. Thus this chapter creates a background for 'assembly language programming using 8086/8088'. A number of assemblers are available for programming with 8086/8088. Each of them has slightly different syntax, directives and operators. However, most of them work on similar principles. The directives and operators considered here are available with MASM (Microsoft MACRO ASSEMBLER).

### 2.1 MACHINE LANGUAGE INSTRUCTION FORMATS

A machine language instruction format has one or more number of fields associated with it. The first field is called as *operation code field* or *opcode field*, which indicates the type of the operation to be performed by the CPU. The instruction format also contains other fields known as *operand fields*. The CPU executes the instruction using the information which reside in these fields.

There are six general formats of instructions in 8086 instruction set. The length of an instruction may vary from one byte to six bytes. The instruction formats are described as follows:

**1. One byte Instruction** This format is only one byte long and may have the implied data or register operands. The least significant 3-bits of the opcode are used for specifying the register operand, if any. Otherwise, all the 8-bits form an opcode and the operands are implied.

**2. Register to Register** This format is 2 bytes long. The first byte of the code specifies the operation code and width of the operand specified by w bit. The second byte of the code shows the register operands and R/M field, as shown below.

D <sub>7</sub>	D <sub>1</sub>	D <sub>0</sub>	D7 D6	D5 D4 D3	D2 D1 D0
OP CODE	W		11	REG	R/M

The register represented by the REG field is one of the operands. The R/M field specifies another register or memory location, i.e. the other operand.

**3. Register to/from Memory with no Displacement** This format is also 2 bytes long and similar to the register to register format except for the MOD field as shown.

D <sub>7</sub>	D <sub>1</sub>	D <sub>0</sub>	D7 D6	D5 D4 D3	D2 D1 D0
OP CODE	W		MOD	REG	R/M

The MOD field shows the mode of addressing. The MOD, R/M, REG and the W fields are decided in Table 2.2.

**4. Register to/from Memory with Displacement** This type of instruction format contains one or two additional bytes for displacement along with 2-byte the format of the register to/from memory without displacement. The format is as shown below.

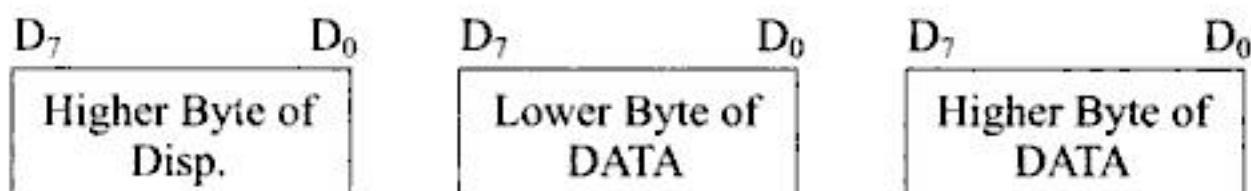
D <sub>7</sub>	D <sub>0</sub>	D7 D6	D5 D4 D3	D2 D1 D0	D <sub>7</sub>	D <sub>0</sub>	D <sub>7</sub>	D <sub>0</sub>
OP CODE		MOD	REG	R/M	Lower Byte of Disp.		Higher Byte of Disp.	

**5. Immediate Operand to Register** In this format, the first byte as well as the 3-bits from the second byte which are used for REG field in case of register to register format are used for opcode. It also contains one or two bytes of immediate data. The complete instruction format is as shown below.

D <sub>7</sub>	D <sub>0</sub>	D7 D6	D5 D4 D3	D2 D1 D0	D <sub>7</sub>	D <sub>0</sub>	D <sub>7</sub>	D <sub>0</sub>
OP CODE		11	OP-CODE	R/M	Lower Byte DATA		Higher Byte DATA	

**6. Immediate Operand to Memory with 16-bit Displacement** This type of instruction format requires 5 or 6 bytes for coding. The first 2 bytes contain the information regarding OPCODE, MOD, and R/M fields. The remaining 4 bytes contain 2 bytes of displacement and 2 bytes of data as shown.

D <sub>7</sub>	D <sub>0</sub>	D7 D6	D5 D4 D3	D2 D1 D0	D <sub>7</sub>	D <sub>0</sub>
OP CODE		MOD	OP-CODE	R/M	Lower Byte of Disp.	



The opcode usually appears in the first byte, but in a few instructions, a register destination is in the first byte and few other instructions may have their 3-bits of opcode in the second byte. The opcodes have the single bit indicators. Their definitions and significances are given as follows:

**W-bit** This indicates whether the instruction is to operate over an 8-bit or 16-bit data/operands. If W bit is 0, the operand is of 8-bits and if W is 1, the operand is of 16-bits.

**D-bit** This is valid in case of double operand instructions. One of the operands must be a register specified by the REG field. The register specified by REG is source operand if D=0, else, it is a destination operand.

**S-bit** This bit is called as sign extension bit. The S bit is used along with W-bit to show the type of the operation. For, example

8-bit operation with 8-bit immediate operand is indicated by S = 0, W = 0;

16-bit operation with 16-bit immediate operand is indicated by S = 0, W = 1 and

16-bit operation with a sign extended immediate data is given by S = 1, W = 1

**V-bit** This is used in case of shift and rotate instructions. This bit is set to 0, if shift count is 1 and is set to 1, if CL contains the shift count.

**Z-bit** This bit is used by REP instruction to control the loop. If Z bit is equal to 1, the instruction with REP prefix is executed until the zero flag matches the Z bit.

The REG code of the different registers (either as source or destination operands) in the opcode byte are assigned with binary codes. The segment registers are only 4 in number hence 2 binary bits will be sufficient to code them. The other registers are 8 in number, so at least 3-bits will be required for coding them. To allow the use of 16-bit registers as two 8-bit registers they are coded with W bit as shown in Table 2.1.

**Table 2.1 Assignment of Codes with Different Registers**

W bit	Register Address (code)	Registers	Segment 2 bit Register (code)	Segment Register
0	000	AL		
0	001	CL	00	ES
0	010	DL	01	CS
0	011	BL	10	SS
0	100	AH	11	DS
0	101	CH		
0	110	DH		
0	111	BH		
1	000	AX		
1	001	CX		
1	010	DX		
1	011	BX		
1	100	SP		
1	101	BP		
1	110	SI		
1	111	DI		

Please note that usually all the addressing modes have DS as the default data segment. However, the addressing modes using BP and SP have SS as the default segment register.

To find out the MOD and R/M fields of a particular instruction, one should first decide the addressing mode of the instruction. The addressing mode depends upon the operands and suggests how the effective address may be computed for locating the operand, if it lies in memory. The different addressing modes of the 8086 instructions are listed in Table 2.2. The R/M column and addressing mode row element specifies the R/M field, while the addressing mode column specifies the MOD field.

**Table 2.2 Addressing Modes and the Corresponding MOD, REG and R/M Fields**

Operands	Memory Operands			Register Operands	
	No Displacement	Displacement 8-bit	Displacement 16-bit		
MOD	00	01	10	11	
R/M				W = 0	W = 1
000	(BX) + (SI)	(BX) + (SI) + D8	(BX) + (SI) + D16	AL	AX
001	(BX) + (DI)	(BX) + (DI) + D8	(BX) + (DI) + D16	CL	CX
010	(BP) + (SI)	(BP) + (SI) + D8	(BP) + (SI) + D16	DL	DX
011	(BP) + (DI)	(BP) + (DI) + D8	(BP) + (DI) + D16	BL	BX
100	(SI)	(SI) + D8	(SI) + D16	AH	SP
101	(DI)	(DI) + D8	(DI) + D16	CH	BP
110	D16	(BP) + D8	(BP) + D16	DH	SI
111	(BX)	(BX) + D8	(BX) + D16	BH	DI

*Note:* 1. D8 and D16 represent 8 and 16 bit displacements respectively.

2. The default segment for the addressing modes using BP and SP is SS. For all other addressing modes the default segments are DS or ES.

DS is the default data segment register when a data is to be referred as an operand. CS is the default code segment register for storing program codes (executable codes). SS is the default segment register for the stack data accesses and operations. ES is the default segment register for the destination data storage. All the segments available (defined in a particular program) can be read or written as data segments by newly defining the data segment as required. There is no physical difference in the memory structure or no physical separation between the segment areas. They may or may not overlap with each other. Chapter 3 on 'Assembly Language Programming' explains the coding procedure of the instructions with suitable examples.

## 2.2 ADDRESSING MODES OF 8086

Addressing mode indicates a way of locating data or operands. Depending upon the data types used in the instruction and the memory addressing modes, any instruction may belong to one or more addressing modes, or some instruction may not belong to any of the addressing modes. Thus the addressing modes describe the types of operands and the way they are accessed for executing an instruction. Here, we will present the addressing modes of the instructions depending upon their types. According to the flow of

instruction execution, the instructions may be categorised as (i) Sequential control flow instructions and (ii) Control transfer instructions.

*Sequential control flow instructions are the instructions which after execution, transfer control to the next instruction appearing immediately after it (in the sequence) in the program.* For example, the arithmetic, logical, data transfer and processor control instructions are sequential control flow instructions. *The control transfer instructions, on the other hand, transfer control to some predefined address or the address somehow specified in the instruction, after their execution.* For example, INT, CALL, RET and JUMP instructions fall under this category.

The addressing modes for sequential and control transfer instructions are explained as follows:

**1. Immediate** In this type of addressing, immediate data is a part of instruction, and appears in the form of successive byte or bytes.

---

#### Example 2.1

MOV AX, 0005H

In the above example, 0005H is the immediate data. The immediate data may be 8-bit or 16-bit in size.

---

**2. Direct** In the direct addressing mode, a 16-bit memory address (offset) is directly specified in the instruction as a part of it.

---

#### Example 2.2

MOV AX, [5000H]

Here, data resides in a memory location in the data segment, whose effective address may be computed using 5000H as the offset address and content of DS as segment address. The effective address, here, is  $10H \cdot DS + 5000H$ .

---

**3. Register** In the register addressing mode, the data is stored in a register and it is referred using the particular register. All the registers, except IP, may be used in this mode.

---

#### Example 2.3

MOV BX, AX.

---

**4. Register Indirect** Sometimes, the address of the memory location which contains data or operand is determined in an indirect way, using the offset registers. This mode of addressing is known as register indirect mode. In this addressing mode, the offset address of data is in either BX or SI or DI register. The default segment is either DS or ES. The data is supposed to be available at the address pointed to by the content of any of the above registers in the default data segment.

---

#### Example 2.4

MOV AX, [BX]

Here, data is present in a memory location in DS whose offset address is in BX. The effective address of the data is given as  $10H \cdot DS + [BX]$ .

---

**5. Indexed** In this addressing mode, offset of the operand is stored in one of the index registers. DS is the default segment for index registers SI and DI. In case of string instructions DS and ES are default segments for SI and DI respectively. This mode is a special case of the above discussed register indirect addressing mode.

---

**Example 2.5**

MOV AX, [SI]

Here, data is available at an offset address stored in SI in DS. The effective address, in this case, is computed as  $10H \cdot DS + [SI]$ .

---

**6. Register Relative** In this addressing mode, the data is available at an effective address formed by adding an 8-bit or 16-bit displacement with the content of any one of the registers BX, BP, SI and DI in the default (either DS or ES) segment. The example given below explains this mode.

---

**Example 2.6**

MOV AX, 50H[BX]

Here, the effective address is given as  $10H \cdot DS + 50H + [BX]$ .

---

**7. Based Indexed** The effective address of data is formed, in this addressing mode, by adding content of a base register (any one of BX or BP) to the content of an index register (any one of SI or DI). The default segment register may be ES or DS.

---

**Example 2.7**

MOV AX, [BX] [SI]

Here, BX is the base register and SI is the index register. The effective address is computed as  $10H \cdot DS + [BX] + [SI]$ .

---

**8. Relative Based Indexed** The effective address is formed by adding an 8 or 16-bit displacement with the sum of contents of any one of the base registers (BX or BP) and any one of the index registers, in a default segment.

---

**Example 2.8**

MOV AX, 50H [BX] [SI]

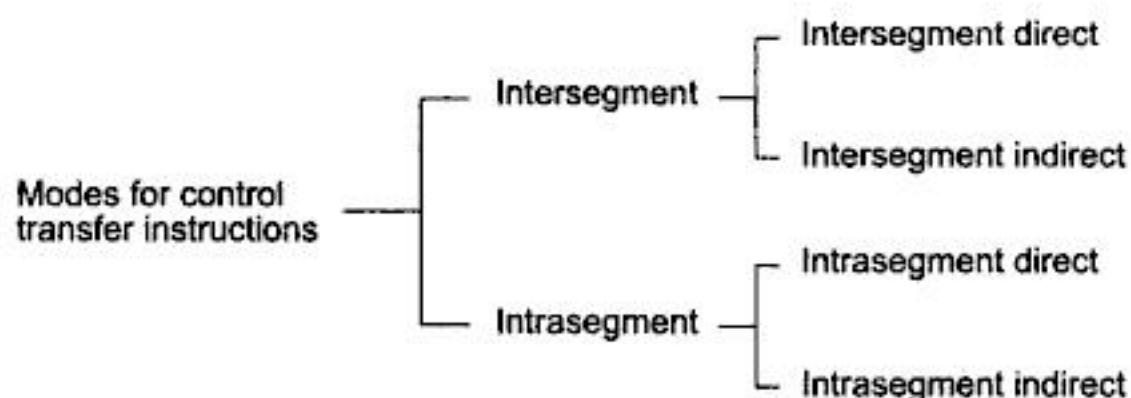
Here, 50H is an immediate displacement, BX is a base register and SI is an index register. The effective address of data is computed as  $10H \cdot DS + [BX] + [SI] + 50H$ .

---

For the control transfer instructions, the addressing modes depend upon whether the destination location is within the same segment or in a different one. It also depends upon the method of passing the destination address to the processor. Basically, there are two addressing modes for the control transfer instructions, viz. intersegment and intrasegment addressing modes.

*If the location to which the control is to be transferred lies in a different segment other than the current one, the mode is called intersegment mode. If the destination location lies in the same segment, the mode is called intrasegment mode.*

Figure 2.1 shows the modes for control transfer instructions.



**Fig. 2.1 Addressing Modes for Control Transfer Instructions**

**9. Intrasegment Direct Mode** In this mode, the address to which the control is to be transferred lies in the same segment in which the control transfer instruction lies and appears directly in the instruction as an immediate displacement value. In this addressing mode, the displacement is computed relative to the content of the instruction pointer IP.

The effective address to which the control will be transferred is given by the sum of 8 or 16 bit displacement and current content of IP. In case of jump instruction, if the signed displacement (d) is of 8 bits (i.e.  $-128 < d < +127$ ), we term it as *short jump* and if it is of 16 bits (i.e.  $-32768 < d < +32767$ ), it is termed as *long jump*.

#### Example 2.9

JMP SHORT LABEL; LABEL lies within -128 to +127 from the current IP content.  
Thus SHORT LABEL is 8-bit signed displacement.

A 16-bit target address of a label indicates that it lies within -32768 to +32767. But a problem arises when one requires a forward jump at a relative address greater than 32767 or backward jump at relative address -32768; in the same segment. Suppose current contents of IP are 5000H then a forward jump may be allowed at all the displacement DISP so that  $IP + DISP = FFFFH$  or  $DISP = FFFF - 5000 = AFFFH$ . Thus forward jumps may be allowed for all 16-bit displacement values from 0000H to AFFFH. If displacement exceeds AFFFH i.e. from B000H to FFFFH, then all such jumps will be treated as backward jumps. All such jumps are called NEAR PTR jumps and coded as below.

JMP NEAR PTR LABEL

**10. Intrasegment Indirect Mode** In this mode, the displacement to which the control is to be transferred, is in the same segment in which the control transfer instruction lies, but it is passed to the instruction indirectly. Here, the branch address is found as the content of a register or a memory location. This addressing mode may be used in unconditional branch instructions.

#### Example 2.10

JMP [BX]; Jump to effective address stored in BX.  
JMP [ BX + 5000H ]

**11. Intersegment Direct** In this mode, the address to which the control is to be transferred is in a different segment. This addressing mode provides a means of branching from one code segment to

another code segment. Here, the CS and IP of the destination address are specified directly in the instruction.

### Example 2.11

JMP 5000H : 2000H;  
Jump to effective address 2000H in segment 5000H.

**12. Intersegment Indirect** In this mode, the address to which the control is to be transferred lies in a different segment and it is passed to the instruction indirectly, i.e. contents of a memory block containing four bytes, i.e. IP(LSB), IP(MSB), CS(LSB) and CS(MSB) sequentially. The starting address of the memory block may be referred using any of the addressing modes, except immediate mode.

### Example 2.12

JMP [2000H];  
Jump to an address in the other segment specified at effective address 2000H in DS, that points to the memory block as said above.

**Forming the Effective Addresses** The following examples explain forming of the effective addresses in the different modes.

### Example 2.13

The contents of different registers are given below. Form effective addresses for different addressing modes.

Offset (displacement) = 5000H

[AX]-1000H, [BX]-2000H, [SI]-3000H, [DI]-4000H, [BP]-5000H,

[SP]-6000H, [CS]-0000H, [DS]-1000H, [SS]-2000H, [IP]-7000H.

Shifting a number four times is equivalent to multiplying it by  $16_D$  or  $10_H$ .

(i) Direct addressing mode

MOV AX, [5000H]

DS:OFFSET  $\Leftrightarrow$  1000H: 5000H

$10H \cdot DS \Leftrightarrow 10000$

Offset  $\Leftrightarrow +5000$

15000H - Effective address

(ii) Register indirect

MOV AX, [BX]

DS:BX  $\Leftrightarrow$  1000H:2000H

$10H \cdot DS \Leftrightarrow 10000$

[BX]  $\Leftrightarrow +2000$

12000H - Effective address

(iii) Register relative

MOV AX, 5000 [BX]

DS: [5000 + BX]

$10H \cdot DS \Leftrightarrow 10000$

Offset  $\Leftrightarrow +5000$   
 $[BX] \Leftrightarrow +2000$   
17000H - Effective address

(iv) Based indexed

MOV AX, [BX] [SI]

DS:[BX + SI]  
 $10H \cdot DS \Leftrightarrow 10000$   
 $[BX] \Leftrightarrow +2000$   
 $[SI] \Leftrightarrow +3000$   
15000H - Effective address

(v) Relative based indexed

MOV AX, 5000 [BX] [SI]

DS: [BX + SI + 5000]  
 $10H \cdot DS \Leftrightarrow 10000$   
 $[BX] \Leftrightarrow +2000$   
 $[SI] \Leftrightarrow +3000$   
 $Offset \Leftrightarrow +5000$   
1A000 - effective address

Below, we present examples of address formation in control transfer instructions.

#### Example 2.14

Suppose our main program resides in the code segment where CS = 1000H. The main program calls a subroutine which resides in the same code segment. The base register contains offset of the subroutine, i.e. BX = 0050H. Since the offset is specified indirectly, as the content of BX, this is indirect addressing. The instruction CALL [BX] calls the subroutine located at an address  $10H \cdot CS + [BX] = 10050H$ , i.e. in the same code segment. Since the control goes to the subroutine which resides in the same segment, this is an example of intrasegment indirect addressing mode.

#### Example 2.15

Let us now assume that the subroutine resides in another code segment, where CS = 2000H. Now CALL 2000H:0050H is an example of intersegment direct addressing mode, since the control now goes to different segment and the address is directly specified in the instruction. In this case, the address of the subroutine is 20050H.

### 2.3 INSTRUCTION SET OF 8086/8088

The 8086/8088 instructions are categorised into the following main types. This section explains the function of each of the instructions with suitable examples wherever necessary.

**(i) Data Copy/Transfer Instructions** These types of instructions are used to transfer data from source operand to destination operand. All the store, move, load, exchange, input and output instructions belong to this category.

- (ii) **Arithmetic and Logical Instructions** All the instructions performing arithmetic, logical, increment, decrement, compare and scan instructions belong to this category.
- (iii) **Branch Instructions** These instructions transfer control of execution to the specified address. All the call, jump, interrupt and return instructions belong to this class.
- (iv) **Loop Instructions** If these instructions have REP prefix with CX used as count register, they can be used to implement unconditional and conditional loops. The LOOP, LOOPNZ and LOOPZ instructions belong to this category. These are useful to implement different loop structures.
- (v) **Machine Control Instructions** These instructions control the machine status. NOP, HLT, WAIT and LOCK instructions belong to this class.
- (vi) **Flag Manipulation Instructions** All the instructions which directly affect the flag register, come under this group of instructions. Instructions like CLD, STD, CLI, STI, etc. belong to this category of instructions.
- (vii) **Shift and Rotate Instructions** These instructions involve the bitwise shifting or rotation in either direction with or without a count in CX.
- (viii) **String Instructions** These instructions involve various string manipulation operations like load, move, scan, compare, store, etc. These instructions are only to be operated upon the strings.

### 2.3.1 Data Copy/Transfer Instructions

**MOV: Move** This data transfer instruction transfers data from one register/memory location to another register/memory location. The source may be any one of the segment registers or other general or special purpose registers or a memory location and, another register or memory location may act as destination.

However, in case of immediate addressing mode, a segment register cannot be a destination register. In other words, 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 the data and then it will have to be moved to that particular segment register. The following example instructions explain the fact.

#### Example 2.16

Load DS with 5000H.

1. MOV DS, 5000H; Not permitted (invalid)

Thus to transfer an immediate data into the segment register, the correct procedure is given below.

2. MOV AX, 5000H  
MOV DS, AX

It may be noted here that both the source and destination operands cannot be memory locations (except for string instructions). Other MOV instruction examples are given below with the corresponding addressing modes.

3. MOV AX, 5000H;      Immediate
4. MOV AX, BX;      Register

- |                     |                                  |
|---------------------|----------------------------------|
| 5. MOV AX, [SI];    | Indirect                         |
| 6. MOV AX, [2000H]; | Direct                           |
| 7. MOV AX, 50H[BX]; | Based relative, 50H Displacement |

**PUSH: Push to Stack** This instruction pushes the contents of the specified register/memory location on to the stack. The stack pointer is decremented by 2, after each execution of the instruction. The actual current stack-top is always occupied by the previously pushed data. Hence, the push operation decrements SP by two and then stores the two byte contents of the operand onto the stack. The higher byte is pushed first and then the lower byte. Thus out of the two decremented stack addresses the higher byte occupies the higher address and the lower byte occupies the lower address.

The actual operation takes place as given belows. SS : SP points to the stack top of 8086 system as shown in Fig. 2.2 and AH, AL contains data to be pushed.

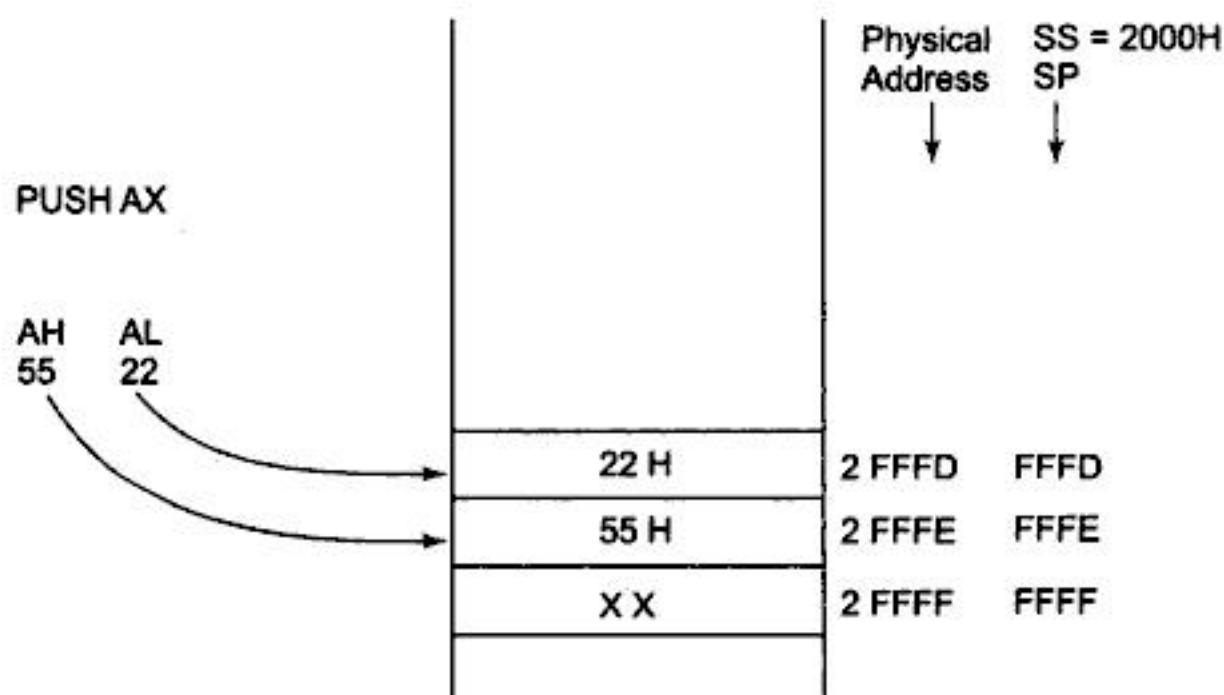


Fig. 2.2 Pushing data to stack memory

The sequence of operation as below:

1. Current stack top is already occupied so decrement SP by one then store AH into the address pointed to by SP.
2. Further decrement SP by one and store AL into the location pointed to by SP.

Thus SP is decremented by 2 and AH–AL contents are stored in stack memory as shown in Fig. 2.2. Contents of SP points to a new stack top.

The examples of these instructions are as follows:

#### Example 2.17

1. PUSH AX
2. PUSH DS
3. PUSH [5000H]; Content of location 5000H and 5001H in DS are pushed onto the stack

**POP: Pop from Stack** This instruction when executed, loads the specified register/memory location with the contents of the memory location of which the address is formed using the current stack segment

and stack pointer as usual. The stack pointer is incremented by 2. The POP instruction serves exactly opposite to the PUSH instruction.

16-bit contents of current stack top are popped into the specified operand as follows.

The sequence of operation is as below.

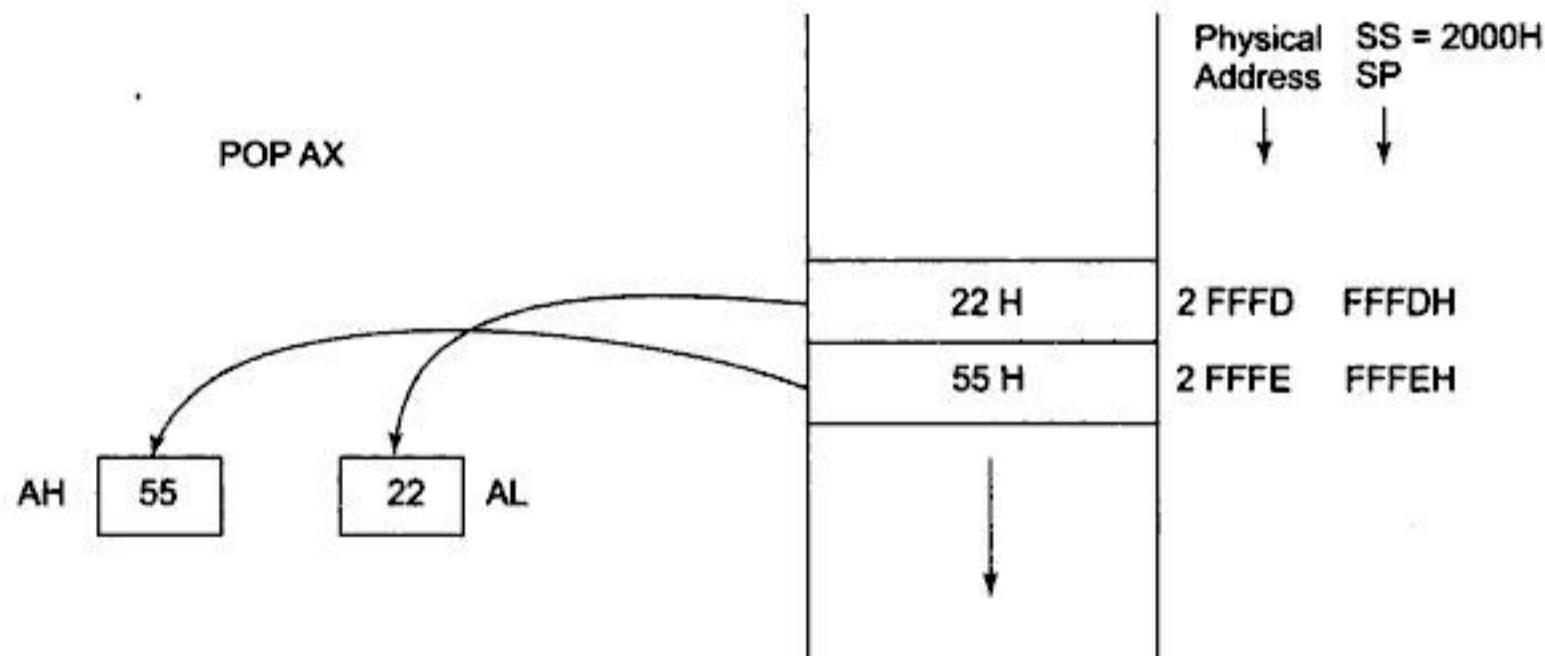
1. Contents of stack top memory location is stored in AL and SP is incremented by one
2. Further contents of memory location pointed to by SP are copied to AH and SP is again incremented by 1

Effectively SP is incremented by 2 and points to next stack top.

The examples of these instructions are shown as follows:

#### Example 2.18

1. POP AX
2. POP DS
3. POP [5000H]



**Fig. 2.3 Popping Register Contents from Stack Memory**

**XCHG: Exchange** This instruction exchanges the contents of the specified source and destination operands, which may be registers or one of them may be a memory location. However, exchange of contents of two memory locations is not permitted. Immediate data is also not allowed in these instructions. The examples are as follows:

#### Example 2.19

1. XCHG [5000H], AX ; This instruction exchanges data between AX and a memory location [5000H] in the data segment.
2. XCHG BX, AX ; This instruction exchanges data between AX and BX.

**IN: Input the Port** This instruction is used for reading an input port. The address of the input port may be specified in the instruction directly or indirectly. AL and AX are the allowed destinations for 8

and 16-bit input operations. DX is the only register (implicit) which is allowed to carry the port address. If the port address is of 16 bits it must be in DX. The examples are given as shown:

---

**Example 2.20**

1. IN AL, 03H ; This instruction reads data from an 8-bit port whose address is 03H and stores it in AL.
  2. IN AX, DX ; This instruction reads data from a 16-bit port whose address is in DX (implicit) and stores it in AX.
  3. MOV DX, 0800H ; The 16-bit address is taken in DX.  
IN AX, DX ; Read the content of the port in AX.
- 

**OUT: Output to the Port** This instruction is used for writing to an output port. The address of the output port may be specified in the instruction directly or implicitly in DX. Contents of AX or AL are transferred to a directly or indirectly addressed port after execution of this instruction. The data to an odd addressed port is transferred on D<sub>8</sub>-D<sub>15</sub> while that to an even addressed port is transferred on D<sub>0</sub>-D<sub>7</sub>. 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 bits it must be in DX. The examples are given as shown:

---

**Example 2.21**

1. OUT 03H, AL ; This sends data available in AL to a port whose address is 03H.
  2. OUT DX, AX ; This sends data available in AX to a port whose address is specified implicitly in DX.
  3. MOV DX, 0300H ; The 16-bit port address is taken in DX.  
OUT DX, AX ; Write the content of AX to a port of which address is in DX.
- 

**XLAT:Translate** The translate instruction is used for finding out the codes in case of code conversion problems, using look up table technique. We will explain this instruction with the aid of the following example.

Suppose, a hexadecimal key pad having 16 keys from 0 to F is interfaced with 8086 using 8255. Whenever a key is pressed, the code of that key (0 to F) is returned in AL. For displaying the number corresponding to the pressed key on the 7-segment display device, it is required that the 7-segment code corresponding to the key pressed is found out and sent to the display port. This translation from the code of the key pressed to the corresponding 7-segment code is performed using XLAT instruction.

For this purpose, one is required to prepare a look up table of codes, starting from an offset say 2000H, and store the 7-segment codes for 0 to F at the locations 2000H to 200FH sequentially. For executing the XLAT instruction, the code of the pressed key obtained from the keyboard ( i.e. the code to be translated) is moved in AL and the base address of the look up table containing the 7-segment codes is kept in BX. After the execution of the XLAT instruction, the 7-segment code corresponding to the pressed key is returned in AL, replacing the key code which was in AL prior to the execution of the XLAT instruction. To find out the exact address of the 7-segment code from the base address of look up table, the content of AL is added to BX internally, and the contents of the address pointed to by this new content of BX in DS are transferred to AL. The following sequence of instructions perform the task.

<i>Mnemonics &amp; Description</i>	<i>Instruction Code</i>			
<b>Data Transfer</b>				
<b>MOV = Move</b>	76543210	76543210	76543210	76543210
Register/Memory to/from Register	100010 dw	mod reg r/m		
Immediate to Register/Memory	1100011 w	mod 000 r/m	data	data if w = 1
Immediate to Register	1011 w reg	data	data if w = 1	
Memory to Accumulator	1010000 w	addr-low	addr-high	
Accumulator to Memory	1010001 w	addr-low	addr-high	
Register/Memory to Segment Register	10001110	mod 0 reg r/m		
Segment Register to Register/Memory	10001100	mod 0 reg r/m		
<b>PUSH = Push:</b>				
Register/Memory	11111111	mod 110 r/m		
Register	01010 reg			
Segment Register	000 reg 110			
<b>POP = Pop:</b>				
Register/Memory	10001111	mod 000 r/m		
Register	01011 reg			
Segment Register	000 reg 111			
<b>XCHG = Exchange</b>				
Register/Memory with Register	1000011 w	mod reg r/m		
Register with Accumulator	10010 reg			
<b>IN = Input from:</b>				
Fixed Port	1110010 w	port		
Variable Port	1110110 w			
<b>OUT = Output to</b>				
Fixed Port	1110011 w	port		
Variable Port	1110111 w			
<b>XLAT = Translate Byte to AL</b>				
<b>LEA = Load EA to Register</b>	10001101	mod reg r/m		
<b>LDS = Load Pointer to DS</b>	11000101	mod reg r/m		
<b>LES = Load Pointer to ES</b>	11000100	mod reg r/m		
<b>LAHF = Load AH with Flags</b>	10011111			
<b>SAHF = Store AH into Flags</b>	10011110			
<b>PUSHF = Push Flags</b>	10011100			
<b>POPF = Pop Flags</b>	10011101			
<b>ARITHMETIC</b>				
<b>ADD = Add:</b>	76543210	76543210	76543210	76543210
Reg/Memory with Register to Either	000000 dw	mod reg r/m		
Immediate to Register/Memory	100000 sw	mod 000 r/m	data	data if s w = 01
Immediate to Accumulator	0000010 w	data	data if w = 1	
<b>ADC = Add with Carry:</b>				
Reg/Memory with Register to Either	000100 dw	mod reg r/m		
Immediate to Register/Memory	100000 sw	mod 010 r/m	data	data if s w = 01
Immediate to Accumulator	0001010 w	data	data if w = 1	
<b>INC = Increment:</b>				

(Contd.)

Mnemonics & Description	Instruction Code			
Register/Memory	1111111 w	mod 000 r/m		
Register	01000 reg			
<b>AAA</b> = ASCII Adjust for Addition	00110111			
<b>DAA</b> = Decimal Adjust for Addition	00100111			
<b>SUB</b> = Subtract				
Reg/Memory and Register to Either	001010 dw	mod reg r/m		
Immediate from Register/Memory	100000 sw	mod 101 r/m	data	data if s w = 01
Immediate from Accumulator	0010110 w	data		data if w = 1
<b>SBB</b> = Subtract with Borrow				
Reg/Memory and Register to Either	000110 dw	mod reg r/m		
Immediate from Register/Memory	100000 sw	mod 011 r/m	data	data if s w = 01
Immediate from accumulator	0001110 w	data		data if w = 1
<b>DEC</b> = Decrement:				
Register/Memory	1111111 w	mod 001 r/m		
Register	01001 reg			
<b>NEG</b> = Change sign	1111011 w	mod 011 r/m		
<b>CMP</b> = Compare:				
Register/Memory and Register	001110 dw	mod reg r/m		
Immediate with Register/Memory	100000 sw	mod 111 r/m	data	data if s w = 01
Immediate with Accumulator	0011110 w	data		data if w = 1
<b>AAS</b> = ASCII Adjust for Subtract	00111111			
<b>DAS</b> = Decimal Adjust for Subtract	00101111			
<b>MUL</b> = Multiply (Unsigned)	1111011 w	mod 100 r/m		
<b>IMUL</b> = Integer Multiply (Signed)	1111011 w	mod 101 r/m		
<b>AAM</b> = ASCII Adjust Multiply	11010100	00001010		
<b>DIV</b> = Divide (Unsigned)	1111011 w	mod 110 r/m		
<b>IDIV</b> = Integer Divide (Signed)	1111011 w	mod 111 r/m		
<b>AAD</b> = ASCII Adjust for Divide	11010101	00001010		
<b>CBW</b> = Convert Byte to Word	10011000			
<b>CWD</b> = Convert Word to Double Word	10011001			
<b>LOGICAL</b>	76543210	76543210	76543210	76543210
<b>NOT</b> = Invert	1111011 w	mod 010 r/m		
<b>SHL/SAL</b> = Shift Logical/Arithmetic Left	110100 v w	mod 100 r/m		
<b>SHR</b> = Shift Logical Right	110100 v w	mod 101 r/m		
<b>SAR</b> = Shift Arithmetic Right	110100 v w	mod 111 r/m		
<b>ROL</b> = Rotate Left	110100 v w	mod 000 r/m		
<b>ROR</b> = Rotate Right	110100 v w	mod 001 r/m		
<b>RCL</b> = Rotate Through Carry Flag Left	110100 v w	mod 010 r/m		
<b>RCR</b> = Rotate Through Carry Right	110100 v w	mod 011 r/m		
<b>AND</b> = And:				
Reg/Memory and Register to Either	001000 dw	mod reg r/m		
Immediate to Register/Memory	1000000 w	mod 100 r/m	data	data if w = 1
Immediate to Accumulator	0010010 w	data		data if w = 1
<b>TEST</b> = And Function to Flags, No Result:				
Register/Memory and Register	1000010 w	mod reg r/m		

(Contd.)

<i>Mnemonics &amp; Description</i>	<i>Instruction Code</i>			
Immediate Data and Register/Memory	1111011 w	mod 000 r/m	data	data if w = 1
Immediate Data and Accumulator	1010100 w	data	data	data if w = 1
<b>OR = Or:</b>				
Reg/Memory and Register to Either	000010 dw	mod reg r/m		
Immediate to Register/Memory	1000000 w	mod 001 r/m	data	data if w = 1
Immediate to Accumulator	0000110 w	data	data	data if w = 1
<b>XOR = Exclusive or:</b>				
Reg/Memory and Register to Either	001100 dw	mod reg r/m		
Immediate to Register/Memory	1000000 w	mod 110 r/m	data	data if w = 1
Immediate to Accumulator	0011010 w	data	data	data if w = 1
<b>STRING MANIPULATIONS</b>				
<b>REP = Repeat</b>	1111001 z			
<b>MOVS = Move Byte/Word</b>	1010010 w			
<b>CMPS = Compare Byte/Word</b>	1010011 w			
<b>SCAS = Scan Byte/Word</b>	1010111 w			
<b>LODS = Load byte/Wd to AL/AX</b>	1010110 w			
<b>STOS = Stor Byte/Wd from AL/A</b>	1010101 w			
<b>CONTROL TRANSFER</b>				
<b>CALL = Call:</b>				
Direct Within Segment	11101000	disp-low	disp-high	
Indirect Within Segment	11111111	mod 010 r/m		
Direct Intersegment	10011010	offset-low	offset-high	
	76543210	seg-low	seg-high	
Indirect Intersegment	11111111	mod 011 r/m		
<b>JMP = Unconditional Jump:</b>				
Direct Within Segment	11101001	disp-low	disp-high	
Direct Within Segment-short	11101011	disp		
Indirect Within Segment	11111111	mod 100 r/m		
Direct Intersegment	11101010	offset-low	offset-high	
Indirect Intersegment	11111111	seg-low	seg-high	
		mod 101 r/m		
<b>RET = Return from CALL:</b>				
Within Segment	11000011			
Within Seg Adding Immediate to SP	11000010	data-low	data-high	
Intersegment	11001011			
Intersegment Adding Immediate to SP	11001010	data-low	data-high	
<b>JE/JZ = Jump on Equal/Zero</b>	01110100	disp		
<b>JL/JNGE = Jump on Less/Not Greater or Equal</b>	011111100	disp		
<b>JLE/JNG = Jump on Less or Equal/Not Greater</b>	01111110	disp		
<b>JB/JNAE = Jump on Below/Not Above or Equal</b>	01110010	disp		
<b>JBE/JNA = Jump on Below or Equal/Not Above</b>	01110110	disp		

(Contd.)

<i>Mnemonics &amp; Description</i>	<i>Instruction Code</i>	
<b>JP/JPE</b> = Jump on Parity/Parity Even	01111010	disp
<b>JO</b> = Jump on Overflow	01110000	disp
<b>JS</b> = Jump on Sign	01111000	disp
<b>JNE/JNZ</b> = Jump on Not Equal/Not Zero	01110101	disp
<b>JNL/JGE</b> = Jump on Not Less/Greater or Equal	01111101	disp
<b>JNLE/JG</b> = Jump on Not Less or Equal/Greater	01111111	disp
<b>JNB/JAE</b> = Jump on Not Below/Above or Equal	01110011	disp
<b>JNBE/JA</b> = Jump on Not Below or Equal/Above	01110111	disp
<b>JNP/JPO</b> = Jump on Not Par/Par Odd	01111011	disp
<b>JNO</b> = Jump on Not Overflow	01110001	disp
<b>JNS</b> = Jump on Not Sign	01111001	disp
<b>LOOP</b> = Loop CX Times	11100010	disp
<b>LOOPZ/LOOPE</b> = Loop While Zero/Equal	11100001	disp
<b>LOOPNZ/LOOPNE</b> = Loop While Not Zero/Equal	11100000	disp
<b>JCXZ</b> = Jump on CX Zero	11100011	disp
<b>INT</b> = Interrupt		
<b>Type Specified</b>	11001101	type
Type 3	11001100	
<b>INTO</b> = Interrupt on Overflow	11001110	
<b>IRET</b> = Interrupt Return	11001111	
	76543210	76543210
<b>PROCESSOR CONTROL</b>		
<b>CLC</b> = Clear Carry	11111000	
<b>CMC</b> = Complement Carry	11110101	
<b>STC</b> = Set Carry	11111001	
<b>CLD</b> = Clear Direction	11111100	
<b>STD</b> = Set Direction	11111101	
<b>CLI</b> = Clear Interrupt	11111010	
<b>STI</b> = Set Interrupt	11111011	
<b>HLT</b> = Halt	11110100	
<b>WAIT</b> = Wait	10011011	
<b>ESC</b> = Escape (to External Device)	11011xxx	mod xxx r/m
<b>LOCK</b> = Bus Lock Prefix	11110000	

\*The v, w, d, s and z bits and the mod, reg, r/m fields are discussed in the addressing modes' section.

**Fig. 2.4 8086/8088 Instruction Set Summary**

**Example 2.22**

```

MOV AX, SEG TABLE      ; Address of the segment containing look-up-table
MOV DS, AX             ; is transferred in DS
MOV AL, CODE           ; Code of the pressed key is transferred in AL
MOV BX, OFFSET TABLE   ; Offset of the code look-up-table in BX
XLAT                  ; Find the equivalent code and store in AL

```

**LEA: Load Effective Address** The load effective address instruction loads the effective address formed by destination operand into the specified source register. This instruction is more useful for assembly language rather than for machine language. The examples are given below.

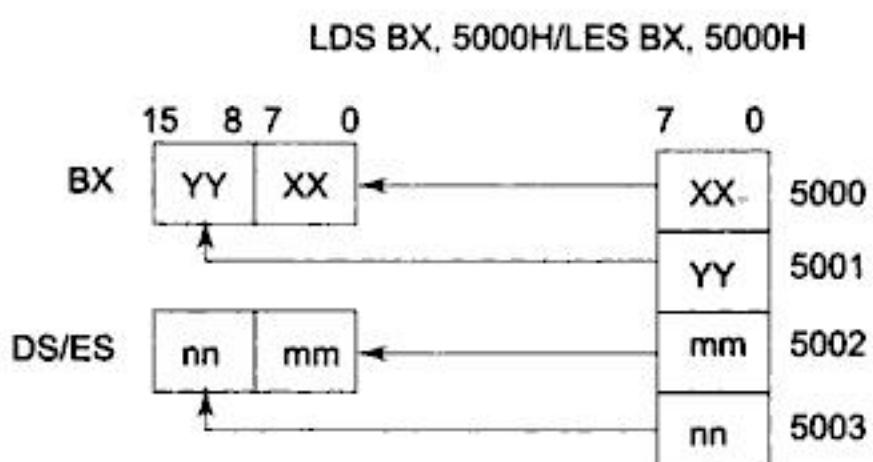
**Example 2.23**

```

LEA BX,ADR      ; Effective address of Label ADR i.e. offset of ADR will be transferred to Reg
                  ; BX.
LEA SI,ADR[Bx] ; offset of Label ADR will be added to content of Bx to form effective
                  ; address and it will be loaded in SI

```

**LDS/LES: Load Pointer to DS/ES** This instruction loads the DS or ES register and the specified destination register in the instruction with the content of memory location specified as source in the instruction. The example in Fig. 2.5 explains the operation.



**Fig. 2.5 LDS/LES Instruction Execution**

**LAHF : Load AH from Lower Byte of Flag** 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 (except overflow) at a time.

**SAHF: Store AH to Lower Byte of Flag Register** This instruction sets or resets the condition code flags (except overflow) in the lower byte of the flag register depending upon the corresponding bit positions in AH. If a bit in AH is 1, the flag corresponding to the bit position is set, else it is reset.

**PUSHF: Push Flags to Stack** The push flag instruction pushes the flag register on to the stack; first the upper byte and then the lower byte is pushed on to it. The SP is decremented by 2, for each push operation. The general operation of this instruction is similar to the PUSH operation.

**POPF: Pop Flags from Stack** The pop flags instruction loads the flag register completely (both bytes) from the word contents of the memory location currently addressed by SP and SS. The SP is incremented by 2 for each pop operation.

Figure 2.4 shows the data sheet for the hand coding of all the 8086 instructions. The MOD and R/M fields are to be decided as already described in this chapter. This type of instructions do not affect any flags.

### 2.3.2 Arithmetic Instructions

These instructions perform the arithmetic operations, like addition, subtraction, multiplication and division along with the respective ASCII and decimal adjust instructions. The increment and decrement operations also belong to this type of instructions. The 8086/8088 instructions falling under this category are discussed below in significant details. The arithmetic instructions affect all the condition code flags. The operands are either the registers or memory locations or immediate data depending upon the addressing mode.

**ADD: Add** This instruction adds an immediate data or contents of a memory location specified in the instruction or a register (source) to the contents of another register (destination) or memory location. The result is in the destination operand. However, both the source and destination operands cannot be memory operands. That means memory to memory addition is not possible. Also the contents of the segment registers cannot be added using this instruction. All the condition code flags are affected, depending upon the result. The examples of this instruction are given along with the corresponding modes.

---

#### Example 2.24

1. ADD AX, 0100H	Immediate
2. ADD AX, BX	Register
3. ADD AX, [SI]	Register indirect
4. ADD AX, [5000H]	Direct
5. ADD [5000H], 0100H	Immediate
6. ADD 0100H	Destination AX (implicit)

---

**ADC: Add with Carry** This instruction performs the same operation as ADD instruction, but adds the carry flag bit (which may be set as a result of the previous calculations) to the result. All the condition code flags are affected by this instruction. The examples of this instruction along with the modes are as follows:

---

#### Example 2.25

1. ADC 0100H	Immediate (AX implicit)
2. ADC AX, BX	Register
3. ADC AX, [SI]	Register indirect
4. ADC AX, [5000H]	Direct
5. ADC [5000H], 0100H	Immediate

---

**INC: Increment** This instruction increases the contents of the specified register or memory location by 1. All the condition code flags are affected except the carry flag CF. This instruction adds 1 to the

contents of the operand. Immediate data cannot be operand of this instruction. The examples of this instruction are as follows:

---

**Example 2.26**

1. INC AX Register
  2. INC [BX] Register indirect
  3. INC [5000H] Direct
- 

**DEC: Decrement** The decrement instruction subtracts 1 from the contents of the specified register or memory location. All the condition code flags, except the carry flag, are affected depending upon the result. Immediate data cannot be operand of the instruction. The examples of this instruction are as follows:

---

**Example 2.27**

1. DEC AX Register
  2. DEC [5000H] Direct
- 

**SUB: Subtract** The subtract instruction subtracts the source operand from the destination operand and the result is left in the destination operand. Source operand may be a register, memory location or immediate data and the destination operand may be a register or a memory location, but source and destination operands both must not be memory operands. Destination operand can not be an immediate data. All the condition code flags are affected by this instruction. The examples of this instruction along with the addressing modes are as follows:

---

**Example 2.28**

1. SUB AX, 0100H Immediate [destination AX]
  2. SUB AX, BX Register
  3. SUB AX, [5000H] Direct
  4. SUB [5000H], 0100 Immediate
- 

**SBB: Subtract with Borrow** The subtract with borrow instruction subtracts the source operand and the borrow flag (CF) which may reflect the result of the previous calculations, from the destination operand. Subtraction with borrow, here means subtracting 1 from the subtraction obtained by SUB, if carry (borrow) flag is set.

The result is stored in the destination operand. All the flags are affected (Condition code) by this instruction. The examples of this instruction are as follows:

---

**Example 2.29**

1. SBB AX, 0100H Immediate [destination AX]
  2. SBB AX, BX Register
  3. SBB AX, [5000H] Direct
  4. SBB [5000H], 0100 Immediate
-

**CMP: Compare** This instruction compares the source operand, which may be a register or an immediate data or a memory location, with a destination operand that may be a register or a memory location. For comparison, it subtracts the source operand from the destination operand but does not store the result anywhere. The flags are affected depending upon the result of the subtraction. If both of the operands are equal, zero flag is set. If the source operand is greater than the destination operand, carry flag is set or else, carry flag is reset. The examples of this instruction are as follows:

### Example 2.30

- |                      |                   |
|----------------------|-------------------|
| 1. CMP BX, 0100H     | Immediate         |
| 2. CMP AX, 0100H     | Immediate         |
| 3. CMP [5000H],0100H | Direct            |
| 4. CMP BX, [SI]      | Register indirect |
| 5. CMP BX, CX        | Register          |

**AAA: ASCII Adjust After Addition** The AAA instruction is executed after an ADD instruction that adds two ASCII coded operands to give a byte of result in AL. The AAA instruction converts the resulting contents of AL to unpacked decimal digits. After the addition, the AAA instruction examines the lower 4 bits of AL to check whether it contains a valid BCD number in the range 0 to 9. If it is between 0 to 9 and AF is zero, AAA sets the 4 high order bits of AL to 0. The AH must be cleared before addition. If the lower digit of AL is between 0 to 9 and AF is set, 06 is added to AL. The upper 4 bits of AL are cleared and AH is incremented by one. If the value in the lower nibble 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. The remaining flags are unaffected. The AH is modified as sum of previous contents (usually 00) and the carry from the adjustment, as shown in Fig. 2.6. This instruction does not give exact ASCII codes of the sum, but they can be obtained by adding 3030H to AX.

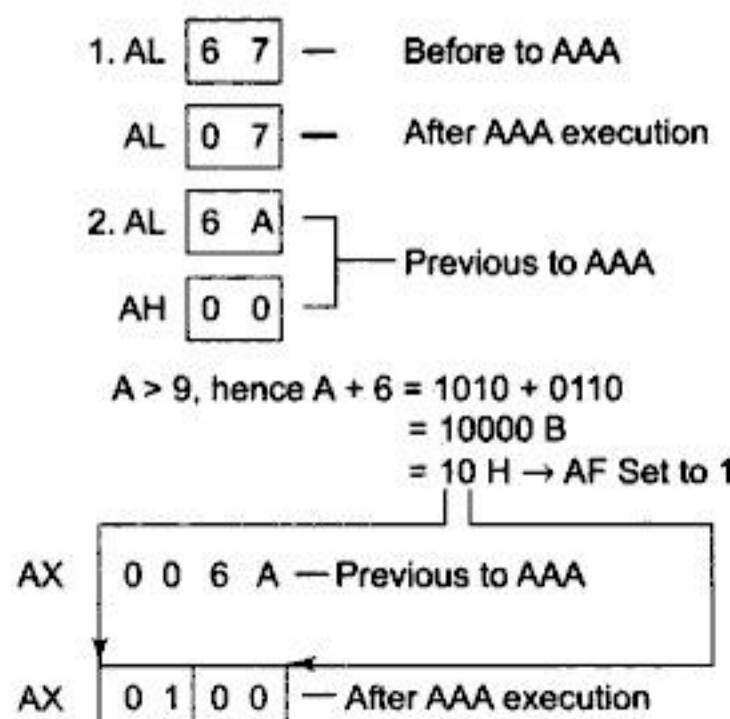


Fig. 2.6 ASCII Adjust after Addition Instruction

**AAS: ASCII Adjust AL after Subtraction** AAS instruction corrects the result in AL register after subtracting two unpacked ASCII operands. The result is in unpacked decimal format. If the lower 4 bits

of AL register are greater than 9 or if the AF flag is 1, the AL is decremented by 6 and AH register is decremented by 1, the CF and AF are set to 1. Otherwise, the CF and AF are set to 0, the result needs no correction. As a result, the upper nibble of AL is 00 and the lower nibble may be any number from 0 to 9. The procedure is similar to the AAA instruction except for the subtraction of 06 from AL. AH is modified as difference of the previous contents (usually zero) of AH and the borrow for adjustment.

**AAM : ASCII Adjust after Multiplication** This instruction, after execution, converts the product available in AL into unpacked BCD format. The AAM—ASCII Adjust After Multiplication—instruction follows a multiplication instruction that multiplies two unpacked BCD operands, i.e. higher nibbles of the multiplication operands should be 0. The multiplication of such operands is carried out using MUL instruction. Obviously the result of multiplication is available in AX. The following AAM instruction replaces content of AH by tens of the decimal multiplication and AL by singles of the decimal multiplication.

#### Example 2.31

```
MOV AL, 04 ; AL ← 04
MOV BL, 09 ; BL ← 09
MUL BL ; AH-AL ← 24H (9 × 4)
AAM ; AH ← 03
; AL ← 06
```

**AAD:ASCII Adjust before Division** Though the names of these two instructions (AAM and AAD) appear to be similar, there is a lot of difference between their functions. The AAD instruction converts two unpacked BCD digits in AH and AL to the equivalent binary number in AL. This adjustment must be made before dividing the two unpacked BCD digits in AX by an unpacked BCD byte. PF, SF, ZF are modified while AF, CF, OF are undefined, after the execution of the instruction AAD. The example explains the execution of the instruction. In the instruction sequence, this instruction appears before DIV instruction unlike AAM appears after MUL. Let AX contains 0508 unpacked BCD for 58 decimal, and DH contains 02H.

#### Example 2.32

AX	05	08
AAD result in AL	00	3A

58D = 3A H in AL

The result of AAD execution will give the hexadecimal number 3A in AL and 00 in AH. Note that 3A is the hexadecimal equivalent of 58 (decimal). Now, instruction DIV DH may be executed. So rather than ASCII adjust for division, it is ASCII adjust before division. All the ASCII adjust instructions are also called as unpacked BCD arithmetic instructions. Now, we will consider the two instructions related to packed BCD arithmetic.

**DAA: Decimal Adjust Accumulator** This instruction is used to convert the result of the addition of two packed BCD numbers to a valid BCD number. The result has to be only in AL. If the lower nibble is greater than 9, after addition or if AF is set, it will add 06 to the lower nibble in AL. After adding 06 in the lower nibble of AL, if the upper nibble of AL is greater than 9 or if carry flag is set, DAA instruction adds 60H to AL. The examples given below explain the instruction.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

quotient and the remainder, in this case, will be in AX and DX respectively. This instruction does not affect any flag.

**IDIV: Signed Division** This instruction performs the same operation as the DIV instruction, but with signed operands. The results are stored similarly as in case of DIV instruction in both cases of word and double word divisions. The results will also be signed numbers. The operands are also specified in the same way as DIV instruction. Divide by 0 interrupt is generated, if the result is too big to fit in AX (16-bit dividend operation) or AX and DX (32-bit dividend operation). All the flags are undefined after IDIV instruction.

### 2.3.3 Logical Instructions

These type of instructions are used for carrying out the bit by bit shift, rotate, or basic logical operations. All the condition code flags are affected depending upon the result. Basic logical operations available with 8086 instruction set are AND, OR, NOT, and XOR. The instruction for each of these operations are discussed as follows.

**AND: Logical AND** This instruction bit by bit ANDs the source operand that may be an immediate, a register or a memory location to the destination operand that may be a register or a memory location. The result is stored in the destination operand. At least one of the operands should be a register or a memory operand. Both the operands cannot be memory locations or immediate operands. An immediate operand cannot be a destination operand. The examples of this instruction are as follows:

---

#### Example 2.37

1. AND AX, 0008H
2. AND AX, BX
3. AND AX, [5000H]
4. AND [5000H], DX

If the content of AX is 3F0FH, the first example instruction will carry out the operation as given below. The result 3F9FH will be stored in the AX register.

0 0 1 1	1 1 1 1	0 0 0 0	1 1 1 1	= 3F0F H [AX]
↓ ↓ ↓ ↓	↓ ↓ ↓ ↓	↓ ↓ ↓ ↓	↓ ↓ ↓ ↓	AND
0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 0	= 0008 H
0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 0	= 0008 H [AX]

The result 0008H will be in AX.

---

**OR: Logical OR** The OR instruction carries out the OR operation in the same way as described in case of the AND operation. The limitations on source and destination operands are also the same as in case of AND operation. The examples are as follows:

---

#### Example 2.38

1. OR AX, 0098H
2. OR AX, BX
3. OR AX, [5000H]
4. OR [5000H], 0008H



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**RCR: Rotate Right through Carry** This instruction rotates the contents (bit-wise) of the destination operand right by the specified count through carry flag (CF). For each operation, the carry flag is pushed into the MSB of the operand, and the LSB is pushed into carry flag. The remaining bits are shifted right by the specified count positions. The SF, PF, ZF are left unchanged. The operand may be a register or a memory location. Figure 2.12 explains the operation.

BIT POSITIONS	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	CF (arbitrary)
OPERAND	1	0	1	0	1	1	1	1	0	1	0	1	1	1	0	1	0
Count = 1					0	1	0	1	0	1	1	1	1	0	1	1	0

Fig. 2.12 Execution of RCR Instruction

**RCL: Rotate Left through Carry** This instruction rotates (bit-wise) the contents of the destination operand left by the specified count through the carry flag (CF). For each operation, the carry flag is pushed into LSB, and the MSB of the operand is pushed into carry flag. The remaining bits are shifted left by the specified positions. The SF, PF, ZF are left unchanged. The operand may be a register or a memory location. Figure 2.13 explains the operation.

BIT POSITIONS	CF	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OPERAND		0	1	0	0	1	1	1	0	1	1	0	1	1	0	1	0
Count = 1			1	0	0	1	1	1	0	1	1	0	1	1	0	1	0

Fig. 2.13 Execution of RCL Instruction

The count for rotation or shifting is either 1 or is specified using register CL, in case of all the shift and rotate instructions.

### 2.3.4 String Manipulation Instructions

A series of data bytes or words available in memory at consecutive locations, to be referred to collectively or individually, are called as *byte strings* or *word strings*. For example, a string of characters may be located in consecutive memory locations, where each character may be represented by its ASCII equivalent. For referring to a string, two parameters are required, (a) starting or end address of the string and (b) length of the string. The length of a string is usually stored as count in the CX register. In case of 8085, similar structures can be set up by the pointer and counter arrangements which may be modified at each iteration, till the required condition for proceeding further is satisfied. On the other hand, the 8086 supports a set of more powerful instructions for string manipulations. The incrementing or decrementing of the pointer, in case of 8086 string instructions, depends upon the Direction Flag (DF) status. If it is a byte string operation, the index registers are updated by one. On the other hand, if it is a word string operation, the index registers are updated by two. The counter in both the cases, is decremented by one.



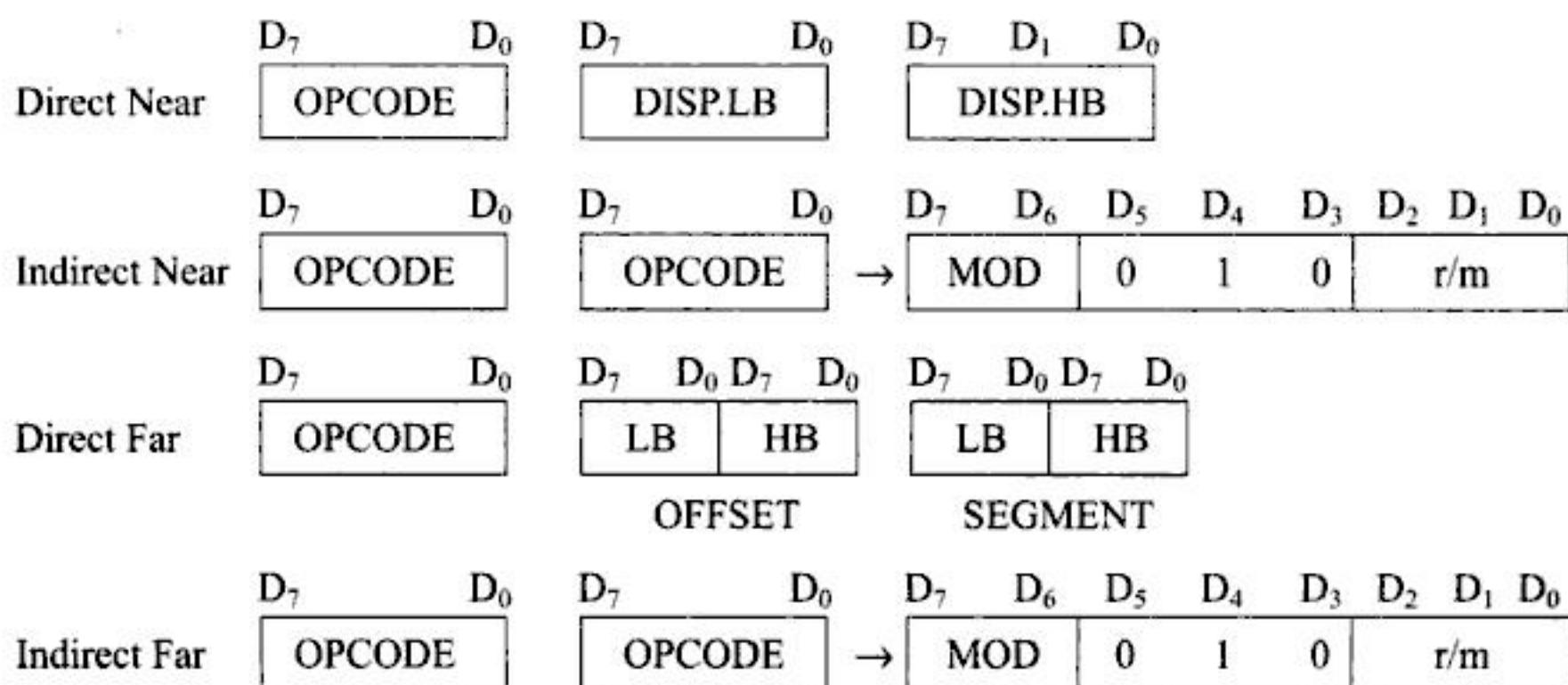
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



**RET: Return from the Procedure** At each CALL instruction, the IP and CS of the next instruction is pushed onto stack, before the control is transferred to the procedure. At the end of the procedure, the RET instruction must be executed. When it is executed, the previously stored content of IP and CS along with flags are retrieved into the CS, IP and flag registers from the stack and the execution of the main program continues further. The procedure may be a near or a far procedure . In case of a FAR procedure, the current contents of SP points to IP and CS at the time of return. While in case of a NEAR procedure, it points to only IP. Depending upon the type of procedure and the SP contents, the RET instruction is of four types.

1. Return within segment
2. Return within segment adding 16-bit immediate displacement to the SP contents.
3. Return intersegment
4. Return intersegment adding 16-bit immediate displacement to the SP contents.

**INT N: Interrupt Type N** In the interrupt structure of 8086/8088, 256 interrupts are defined corresponding to the types from 00H to FFH. 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'4) as offset address and 0000 as segment address. In other words, the multiplication of type N by 4 (offset) points to a memory block in 0000 segment, which contains the IP and CS values of the interrupt service routine. For the execution of this instruction, the IF must be enabled.

#### Example 2.45

Thus the instruction INT 20H will find out the address of the interrupt service routine as follows:

INT 20H

Type\* 4 = 20 \* 4 = 80H

Pointer to IP and CS of the ISR is 0000 : 0080 H



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

The machine control instructions supported by 8086 and 8088 are listed in Table 2.6 along with their functions. They do not require any operand.

**Table 2.6 Machine Control Instructions**

<b>WAIT</b>	-	Wait for Test input pin to go low
<b>HLT</b>	-	Halt the processor
<b>NOP</b>	-	No operation
<b>ESC</b>	-	Escape to external device like NDP (numeric co-processor)
<b>LOCK</b>	-	Bus lock instruction prefix.

As explained in Chapter 1, after executing the HLT instruction, the processor enters the halt state. The two ways to pull it out of the halt state are to reset the processor or to interrupt it. When NOP instruction is executed, the processor does not perform any operation till 4 clock cycles, except for incrementing the IP by one. It then continues with further execution after 4 clock cycles. ESC instruction when executed, frees the bus for an external master like a coprocessor or peripheral devices. The LOCK prefix may appear with another instruction. When it is executed, the bus access is not allowed for another master till the lock prefixed instruction is executed completely. This instruction is used in case of programming for multiprocessor systems. The WAIT instruction when executed, holds the operation of processor with the current status till the logic level on the TEST pin goes low. The processor goes on inserting WAIT states in the instruction cycle, till the TEST pin goes low. Once the TEST pin goes low, it continues further execution .

## 2.4 ASSEMBLER DIRECTIVES AND OPERATORS

The main advantage of machine language programming is that the memory control is directly in the hands of the programmer enabling him to manage the memory of the system more efficiently. However, there are more disadvantages. The programming, coding and resource management techniques are tedious. As the programmer has to consider all these functions, the chances of human errors are more. To understand the programs one has to have a thorough technical knowledge of the processor architecture and instruction set.

The assembly language programming is simpler as compared to the machine language programming. The instruction mnemonics are directly written in the assembly language programs. The programs are now more readable than that of machine language programs. The advantage that assembly language has over machine language is that now the address values and the constants can be identified by labels. If the labels are clear then certainly the program will become more understandable, and each time the programmer will not have to remember the different constants and the addresses at which they are stored, throughout the programs. Due to this facility, the tedious byte handling and manipulations are got rid of. Similarly, now 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 define facilities in assembly language programming. The documentation facility which was not possible with machine language programming is now available in assembly language. Readers will get a better glimpse of the different features of assembly language, when we discuss assembly language programming in the next chapter.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

result in the generation of wrong codes. If the EQU directive is used to assign the value with a label that can be used in place of each recurrence of that constant, only one change in the EQU statement will give the correct and modified code. The examples given below show the syntax.

#### Example 2.49

```
LABEL      EQU      0500H
ADDITION   EQU      ADD
```

The first statement assigns the constant 500H with the label LABEL, while the second statement assigns another label ADDITION with mnemonic ADD.

**EXTRN: External and PUBLIC: Public** The directive EXTRN informs the assembler that the names, procedures and labels declared after this directive have already been defined in some other assembly language modules. While in the other module, where the names, procedures and labels actually appear, they must be declared public, using the PUBLIC directive. If one wants to call a procedure FACTORIAL appearing in MODULE1 from MODULE 2; in MODULE1, it must be declared PUBLIC using the statement PUBLIC FACTORIAL and in module 2, it must be declared external using the declaration EXTRN FACTORIAL . The statement of declaration EXTRN must be accompanied by the SEGMENT and ENDS directives of the MODULE 1, before it is called in MOBULE 2. Thus the MODULE1 and MODULE 2 must have the following declarations.

MODULE1	SEGMENT
PUBLIC	FACTORIAL FAR
MODULE1	ENDS
MODULE2	SEGMENT
EXTRN	FACTORIAL FAR
MODULE2	ENDS

**GROUP: Group the Related Segments** This directive is used to form logical groups of segments with similar purpose or type. This directive is used to inform the assembler to form a logical group of the following segment names. The assembler passes an information to the linker/loader to form the code such that the group declared segments or operands must lie within a 64Kbyte memory segment. Thus all such segments and labels can be addressed using the same segment base.

```
PROGRAM GROUP CODE, DATA, STACK
```

The above statement directs the loader/linker to prepare an EXE file such that CODE, DATA and STACK segment must lie within a 64kbyte memory segment that is named as PROGRAM. Now, for the ASSUME statement, one can use the label PROGRAM rather than CODE, DATA and STACK as shown.

```
ASSUME CS: PROGRAM, DS: PROGRAM, SS: PROGRAM
```

**LABEL: Label** The Label directive is used to assign a name to the current content of the location counter. When the assembly process starts, the assembler initialises a location counter to keep track of memory locations assigned to the program. As the program assembly proceeds, the contents of the location counter are updated. During the assembly process, whenever the assembler comes across the LABEL directive, it assigns the declared label with the current contents of the location counter. The type of the label must be specified, i.e. whether it is a NEAR or a FAR label, BYTE or WORD label, etc.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**' + & -' Operators** These operators represent arithmetic addition and subtraction respectively and are typically used to add or subtract displacements (8 or 16 bit) to base or index registers or stack or base pointers as given in the example:

---

#### Example 2.54

```
MOV AL, [ SI +2 ]
MOV DX, [ BX - 5 ]
MOV BX, [ OFFSET LABEL + 10 H ]
MOV AX, [ BX + 9I ]
```

---

**FAR PTR** This directive indicates the assembler that the label following FAR PTR is not available within the same segment and the address of the label is of 32-bits i.e. 2 bytes offset followed by 2 bytes segment address.

---

#### Example 2.55

```
JMP FAR PTR LABEL
CALL FAR PTR ROUTINE
```

---

Both the above instructions indicate to the assembler that the target address is going to require four bytes; Lower byte of offset, higher byte of offset, lower byte of segment and higher byte of segment; indicating intersegment addressing mode.

**NEAR PTR** This directive indicates that the label following NEAR PTR is in the same segment and needs only 16 bit i.e. 2 byte offset to address it.

---

#### Example 2.56

```
JMP NEAR PTR LABEL
CALL NEAR PTR ROUTINE
```

---

If a label is not preceded by NEAR PTR or FAR PTR, then it is by default considered a NEAR PTR label and two bytes are reserved by the assembler for its address during the process of assembling.

## SUMMARY

This chapter is aimed at introducing the readers with the instruction set of 8086/88 and the most commonly used assembler directives and operators. To start with, the available instruction formats in 8086/88 instruction set are explained in details. Further, the addressing modes available in 8086/88 are discussed in significant details with necessary examples. The 8086/88 instructions can be broadly categorized in six types depending upon their functions, namely data transfer instructions, arithmetic instructions and logical instructions, shift and rotate instructions, string manipulation instructions, control transfer instructions and processor control instructions. All these instruction types have been discussed before proceeding with the assembler directives and operators. The



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

The above instruction sequence is quite straightforward. As the immediate data cannot be loaded into a segment register, the data is transferred to one of the general purpose registers, say AX, and then the register content is moved to the segment register DS. Thus the data segment register DS contains 2000H. The instruction MOV AX,[500H] signifies that the contents of the particular location, whose offset is specified in the brackets with the segment pointed to by DS as segment register, is to be moved to AX. The MOV [0700H],AX instruction moves the contents of the register AX to an offset 0700H in DS (DS = 2000H). Note that the code segment register CS gets automatically loaded by the code segment address of the program whenever it is executed. Actually it is the monitor program that accepts the CS:IP address of the program and passes it to the corresponding registers at the time of execution. Hence no instructions like DS or SS are required for loading the CS register.

### Example 3.2

Write a program to move the contents of the memory location 0500H to register BX and to CX. Add immediate byte 05H to the data residing in memory location, whose address is computed using DS=2000H and offset=0600H. Store the result of the addition in 0700H. Assume that the data is located in the segment specified by the data segment register DS which contain 2000H.

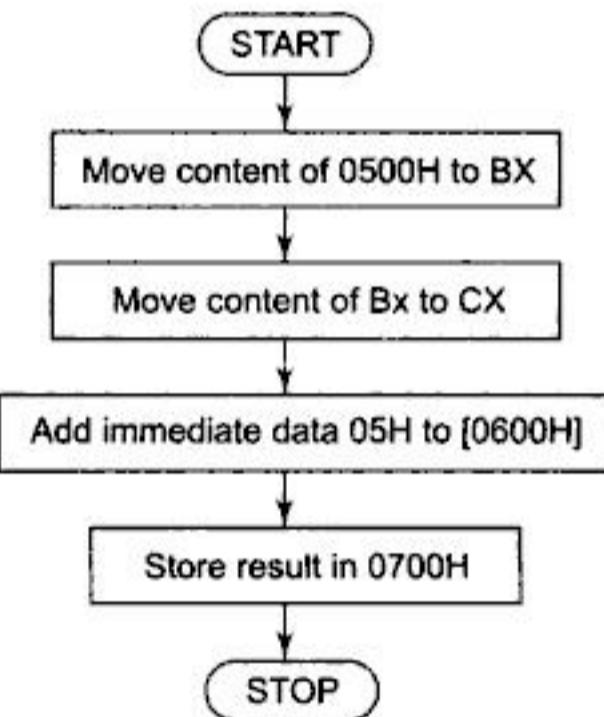
**Solution** The flow chart for the program is shown in Fig. 3.2.

```

MOV AX, 2000H
MOV DS, AX      ; Initialize data segment
                 ; register
MOV BX, [0500H] ; Get contents of 0500H in BX
MOV CX, BX      ; Copy the same contents in CX
ADD [0600H], 05H ; Add byte 05H to contents of
                  ; 0600H
MOV DX, [0600H] ; Store the result in DX
MOV [0700H], DX ; Store the result in 0700H
HLT             ; Stop
  
```

After initialising the data segment register, the content of location 0500H are moved to the BX register using MOV instruction. The same data is moved also to the CX register. For this data transfer, there may be two options as shown.

- (a) MOV CX, BX ; As the contents of BX will be  
; same as 0500H after execution  
; of MOV BX, [0500H].
- (b) MOV CX, [0500H] ; Move directly from 0500H to register CX



**Fig. 3.2 Flow Chart for Example 3.2**

The *opcode* in the first option is only of 2 bytes, while the second option will have 4 bytes of *opcode*. Thus the second option will require more memory and execution time. Due to these reasons, the first option is preferable.

The immediate data byte 05H is added to the content of 0600H using the ADD instruction. The result will be in the destination operand 0600H. This is next stored at the location 0700H. In case of the 8086/8088 instruction set, there is no instruction for the direct transfer of data from the memory source operand to the memory destination operand except, the string instructions. Hence the result of addition



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**Example 3.5**

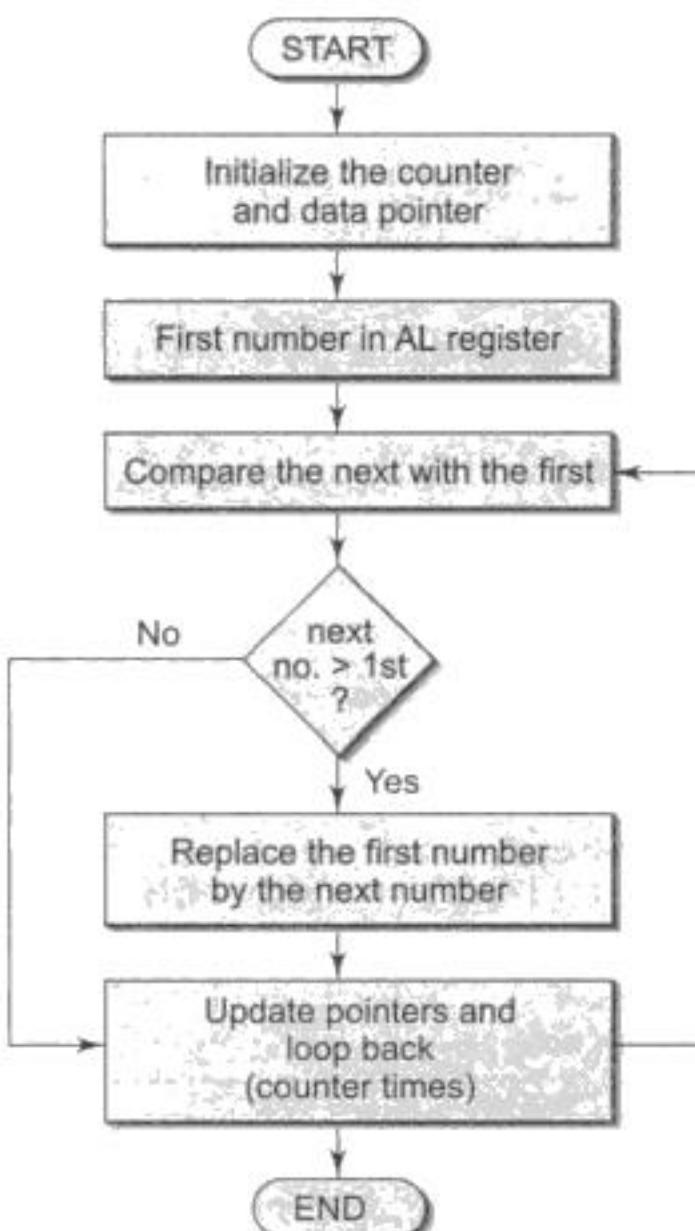
Find out the largest number from an unordered array of sixteen 8-bit numbers stored sequentially in the memory locations starting at offset 0500H in the segment 2000H.

**Solution** The logic for this procedure can be described as follows. The first number of the array is taken in a register, say AL. The second number of the array is then compared with the first one. If the first one is greater than the second one, it is left unchanged. However, if the second one is greater than the first, the second number replaces the first one in the AL register. The procedure is repeated for every number in the array and thus it requires 15 iterations. At the end of 15th iteration the largest number will reside in the register AL. This may be represented in terms of the flow chart as shown in Fig. 3.5. The listing is given below:

```

MOV CX, 0F H      ; Initialize counter for number of iterations
MOV AX, 2000H     ; Initialize data segment
MOV DS, AX         ;
MOV SI, 0500H     ; Initialize source pointer
MOV AL, [SI]       ; Take first number in AL
BACK : INC SI      ; Increment source pointer
            CMP AL,[SI]   ; Compare next number with the previous
            JNC NEXT      ; If the next number is larger
            MOV AL,[SI]   ; replace the previous one with the next
NEXT : LOOP BACK    ; Repeat the procedure 15 times
        HLT

```



**Fig. 3.5 Flow Chart for Example 3.5**



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**Finding out Machine Code for Conditional JUMP (Intrasegment) Instructions** The data sheet Fig. 2.4 shows that, corresponding to each of the conditional jump instructions, the first byte of the opcode is fixed and the jump displacement must be less than or equal to 127(D) bytes and greater than or equal to -128(D). This type of jump is called as short jump. The following conditional forward jump example explains how to find the displacement. The displacement is an 8-bit signed number. If it is positive, it indicates a forward jump, otherwise it indicates a backward jump. The following example is a sequence of instructions rather than a single instruction to elaborate the procedure of the calculation of positive displacement for a forward jump.

#### Example 3.14

2000 ,01	XOR AX, BX
2002 ,03	JNZ OK
2004	NOP
2005	NOP
2006 ,7,8,9	ADD BX, 05H
200A	OK : HLT

The above sequence shows that the programmer wants a conditional jump to label OK, if the zero flag is not set. For finding out the displacement corresponding to the label OK, subtract the address of the jump instruction (2002H), from the address of label (200AH). The required displacement is 200AH - 2002H = 08H. The 08H is the displacement for the forward jump.

Let us find out the displacement for a backward jump. Consider the following sequence of instructions.

#### Example 3.15

2000, 01, 02	MOV CL, 05H
2003              Repeat :	INC AX
2004	DEC CL
2005,2006	JNZ Repeat

For finding out the backward displacement, subtract the address of the label (repeat) from the address of the jump instruction. Complement the subtraction. The lower byte gives the displacement. In this example, the signed displacement for the JNZ instruction comes out to be (2005H-2003H=02, complement-FDH) The magnitude of the displacement must be less than or equal to 127(D). The MSB of the displacement decides whether it is a forward or backward jump. If it is 1, it is a backward jump or else it is a forward jump.

A similar procedure is used to find the displacement for intrasegment short calls.

**Finding out Machine Code for Unconditional JUMP Intrasegment** For this instruction there are again two types of jump, i.e short jump and long jump. The displacement calculation procedures are again the same as given in case of the conditional jump. The only new thing here is that, the displacement may be beyond  $\pm 127(D)$ . This type of jump is called the long jump. The method of calculation of the displacement is again similar to that for short jump.

**Finding out Machine Code for Intersegment Direct Jump** This type of instruction is used to make a jump directly to the address lying in another segment. The opcode itself specifies the new offset and the segment of jump address, directly.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Once the above procedure is completed, you may now focus on assembling the program. Note that all the commands and displays shown in this section are for Norton's Editor. Other editors may require some other commands and their display style may be somewhat different but the overall procedure is the same.

ASSUME DATA	CS:CODE, DS:DATA SEGMENT OPR1 DW 1234 H OPR2 DW 0002 H RESULT DW 01 H DUP(?)
DATA CODE START	ENDS SEGMENT MOV AX, DATA MOV DS, AX MOV AX, OPR 1 MOV BX, OPR 2 CLC ADD AX, BX MOV DI, OFFSET RESULT MOV [DI], AX MOV AH, 4CH INT 21 H
CODE	ENDS END START
KBM.ASM	

**Fig. 3.9 A Program KMB.ASM in Norton's Editor**

Note that before quitting the editor program the newly entered or modified file must be saved, otherwise it will be lost and will not be available for further assembling process.

### 3.3.2 Assembling a Program

Microsoft Assembler MASM is an easy to use and popular assemblers. This section deals with the MASM. As already discussed, the main task of any assembler program is to accept the `text_assembly` language program file as an input and prepare an object file. The `text_assembly` language program file is prepared by using any of the editor programs as discussed in Section 3.3.1. The MASM accepts the file names only with the extension `.ASM`. Even if a filename without any extension is given as input, it provides an `.ASM` extension to it. For example, to assemble the program in Fig. 3.9, one may enter the following command options:

C> MASM KMB

or

C> MASM KMB.ASM

If any of the command option is entered as above, the screen displays, as shown in Fig. 3.10.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**Table 3.1 DEBUG Commands**

<b>COMMAND CHARACTER</b>	<b>Format/Formats</b>	<b>Functions</b>
-R	<ENTER>	Display all Registers and flags
-R	reg<ENTER> old contents:New contents	Display specified register contents and modify with the entered new contents.
-D	<ENTER>	Display 128 memory locations of RAM starting from the current display pointer.
-D	SEG:OFFSET1 OFFSET2<ENTER>	Display memory contents in SEG from OFFSET1 to OFFSET2.
-E	<ENTER>	Enter Hex data at current display pointer SEG:OFFSET.
-E	SEG:OFFSET1 <ENTER>	Enter data at SEG:OFFSET1 byte by byte. The memory pointer is to be incremented by space key, data entry is to be completed by pressing the <ENTER> key.
-f	SEG:OFFSET1 OFFSET2 BYTE <ENTER>	Fill the memory area starting from SEG:OFFSET1 to OFFSET2 by the byte BYTE.
-f	SEG:OFFSET1 OFFSET2 BYTE1,BYTE2,BYTE3<ENTER>	Fill the memory area as above with the sequence BYTE1, BYTE2, BYTE3, etc.
-a	<ENTER>	Assemble from the current CS:IP.
-a	SEG:OFFSET <ENTER>	Assemble the entered instruction from SEG:OFFSET address.
-u	<ENTER>	Unassemble from the current CS:IP.
-u	SEG:OFFSET <ENTER>	Unassemble from the address SEG:OFFSET.
-g	<ENTER>	Execute from current CS:IP. By modifying CS and IP using R command this can be used for any address.
-g	=OFFSET <ENTER>	Execute from OFFSET in the current CS.
-s	SEG:OFFSET1 to OFFSET2 BYTE/BYTES <ENTER>	Searches a BYTE or string of BYTES separated by ',' in the memory block SEG:OFFSET1 to OFFSET2, and displays all the offsets at which the byte or string of bytes is found.

(contd.)



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

statement CODE ENDS marks the end of the CODE segment. The statement END START marks the end of the procedure that started with the label START. At the end of each file, the END statement is a must.

Until now, we have discussed Program 3.1(a) in significant detail. As we have already said, the program contains two logical segments CODE and DATA, but it is not at all necessary that all the programs must contain the two segments. A programmer may use a single segment to cover up data as well as instructions. Program 3.1(b) explains the fact.

```
ASSUME CS:CODE
        CODE      SEGMENT
        OPR1     DW      1234H
        OPR2     DW      0002H
        RESULT    DW      01 DUP(?)
START :   MOV AX, CODE
        MOV DS, AX
        MOV AX, OPR1
        MOV BX, OPR2
        CLC
        ADD AX, BX
        MOV DI, OFFSET RESULT
        MOV [DI], AX
        MOV AH, 4CH
        INT 21H
CODE      ENDS
END START
```

---

#### Program 3.1(b) Alternative listing for Program 3.1

---

We have discussed all the properties of this program in detail. For all the following programs, we will not explain the common things like forming segments using directives and operators, etc. Instead, just the logic of the program will be explained. The use of proper syntax of the 8086/8088 assembler MASM is self explanatory. The comments may help the reader in getting the ideas regarding the logic of the program.

Assemble the above written program using MASM after entering it into the computer using the procedure explained in Section 3.3.1. Once you get the EXE file as the output of the LINK program, Your listing is ready for execution. The Program 3.1 is prepared in the form of EXE file with the name KMB.EXE in the directory. Next, it can be executed with the command line as given below.

C> KMB

This method of execution will store the result of the program in memory but will not display it on the screen. To display the result on the screen the programmer will have to use DOS function calls, which will make the programs too lengthy. Hence, another method to check the results is to run the program under the control of DEBUG. To run the program under the control of debug and to observe the results one must prepare the LST file, that gives information about memory allotment to different labels and variables of the program while assembling it. The LST file can be displayed on the screen using NE-Norton's Editor.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
DEC CL  
JNZ AGAIN  
MOV AH,4CH  
INT 21H  
CODE ENDS  
END START
```

---

**Program 3.5 Listings**

---

**Program 3.6**

Write a program to find out the number of positive numbers and negative numbers from a given series of signed numbers.

**Solution** Take the *i*th number in any of the registers. Rotate it left through carry. The status of carry flag, i.e. the most significant bit of the number will give the information about the sign of the number. If CF is 1, the number is negative; otherwise, it is positive.

```
ASSUME CS:CODE,DS:DATA  
DATA SEGMENT  
LIST DW 2579H,0A500H,0C009H,0159H,0B900H  
COUNT EQU 05H  
DATA ENDS  
CODE SEGMENT  
START:      XOR BX,BX  
             XOR DX,DX  
             MOV AX,DATA  
             MOV DS,AX  
             MOV CL,COUNT  
             MOV SI,OFFSET LIST  
AGAIN:       MOV AX,[SI]  
             SHL AX,01  
             JC NEG  
             INC BX  
             JMP NEXT  
NEG:         INC DX  
NEXT:        ADD SI,02  
             DEC CL  
             JNZ AGAIN  
             MOV AH,4CH  
             INT 21H  
             CODE ENDS  
             END START
```

---

**Program 3.6 Listings**

---

The logic of Program 3.6 is similar to that of Program 3.5, hence comments are not given in Program 3.6 except for a few important ones.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

0110 is added to most significant nibble of the result if it is greater than 9 or AF is set.

$$\begin{array}{r}
 & 1 & \text{Carry from previous digit (AF)} \\
 \text{E} & \rightarrow & 1110 \\
 + & 0110 \\
 \hline
 \text{CF is set to 1} & 0101 & \text{next significant nibble of result}
 \end{array}$$

Result CF	Most significant	Least significant digit
1	5	1

```

ASSUME CS:CODE,DS:DATA
DATA SEGMENT
    OPR1 EQU 92H
    OPR2 EQU 52H
RESULT DB 02 DUP(00)
DATA ENDS
CODE SEGMENT
START:    MOV AX, DATA
          MOV DS, AX
          MOV BL, OPR1
          XOR AL, AL
          MOV AL, OPR2
          ADD AL, BL
          DAA
          MOV RESULT, AL
          JNC MSBO
          INC [RESULT+1]
MSBO:     MOV AH, 4CH
          INT 21H
CODE ENDS
END START

```

### Program 3.9 Listings

In this program, the instruction DAA is used after ADD. Similarly, DAS can be used after SUB instruction. The reader may try to write a program for BCD subtraction for practice.

### Program 3.10

Write a program that performs addition, subtraction, multiplication and division of the given operands. Perform BCD operation for addition and subtraction.

**Solution** Here we have directly given the routine for Program 3.10.

```

ASSUME CS:CODE,DS:DATA
DATA SEGMENT
    OPR1 EQU 98H
    OPR2 EQU 49H
    SUM DW 01 DUP(00)
    SUBT DW 01 DUP(00)
    PROD DW 01 DUP(00)

```



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**Solution** In the addition of two matrices, the corresponding elements are added to form the corresponding elements of the result matrix as shown:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11} + b_{11} + a_{12} + b_{12} + a_{13} + b_{13} \\ a_{21} + b_{21} + a_{22} + b_{22} + a_{23} + b_{23} \\ a_{31} + b_{31} + a_{32} + b_{32} + a_{33} + b_{33} \end{bmatrix}$$

$$[A] + [B] = [A + B]$$

The matrix A is stored in the memory at an offset MAT1, as given:

$a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{23}, a_{31}, a_{32}, a_{33}$ , etc.

A total of  $3 \times 3 = 9$  additions are to be done. The assembly language program is written as shown:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
    DIM EQU 09H
    MAT1 DB 01,02,03,04,05,06,07,08,09
    MAT2 DB 01,02,03,04,05,06,07,08,09
    RMAT3 DW 09H DUP(?)

DATA ENDS
CODE SEGMENT
START:    MOV AX,DATA
          MOV DS,AX
          MOV CX,DIM
          MOV SI,OFFSET MAT1
          MOV DI,OFFSET MAT2
          MOV BX,OFFSET RMAT3
NEXT:     XOR AX,AX
          MOV AL,[SI]
          ADD AL,[DI]
          MOV WORD PTR [BX],AX
          INC SI
          INC DI
          ADD BX,02
          LOOP NEXT
          MOV AH,4CH
          INT 21H
CODE      ENDS
END START
```

#### Program 3.14 Listings

#### Program 3.15

Write a program to find out the product of two matrices. Store the result in the third matrix. The matrices are specified as in the Program 3.14.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**Solution** The program to convert the binary number into equivalent BCD number is developed below :

```

ASSUME CS:CODE,DS:DATA
DATA SEGMENT
BIN EQU
RESULT DW (?)
DATA ENDS
CODE SEGMENT
START :    MOV AX, DATA      ; INITIALIZE DATA SEGMENT
            MOV DS, AX
            MOV BX,BIN
            MOV AX, 0      ; INITIALIZE TO 0
            MOV CX, 0      ; INITIALIZE TO 0
CONTINUE :   CMP BX, 0      ; COMPARISION FOR ZERO BINARY
             NUMBER.
             JZ ENDPORG    ; IF ZERO END THE PROGRAM
             DEC BX        ; DECREMENT BX BY 1
             MOV AL,CL
             ADD AL,1
             DAA
             MOV CL,AL      ; STORING RESULT IN CL REGISTER
             MOV AL,CH
             ADC AL,00H     ; ADD WITH CARRY
             DAA
             MOV CH,AL      ; STORING RESULT IN CH REGISTER
             JMP CONTINUE
ENDPROG :    MOV RESULT, CX ; STORING RESULT IN DATA SEGMENT
              MOV AH,4CH
              INT 21H
              CODE ENDS
              END START

```

#### Program 3.18 Listings

#### Program 3.19

Write a program to convert a BCD number into an equivalent binary number.

**Solution** A program to convert a BCD number into binary equivalent number is developed below.

```

ASSUME CS:CODE,DS:DATA
DATA SEGMENT
BCD_NUM EQU 4576H
BIN_NUM DW (?)
DATA ENDS
CODE SEGMENT
START :    MOV AX, DATA      ; INITIALIZE DATA SEGMENT
            MOV DS, AX
            MOV BX,BCD_NUM  ; BX IS NOW HAVING BCD NUMBER

```



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

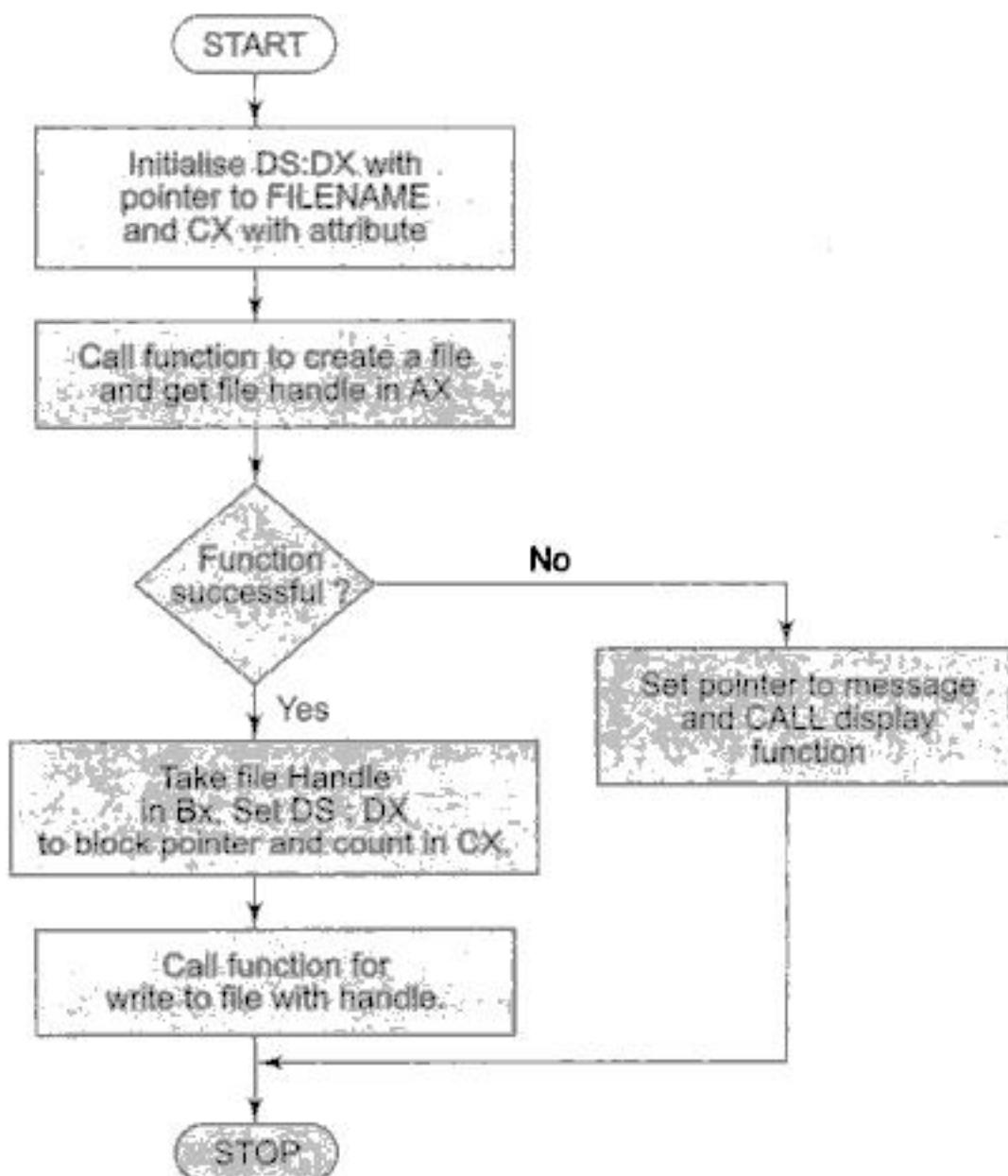


You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**Solution** This type of programs, written for the utilization of resources of a computer system does not require much of logic. These programs contain the specific function calls along with some instructions to load the registers to prepare the environment (required data) for the interrupt call. A flow chart for Program 3.23 is shown in Fig. 3.14. It is up to the application designer to combine these programs with the actual application programs.



**Fig. 3.14 Flow Chart for Program 3.23**

```

ASSUME CS:CODE,DS:DATA
DATA SEGMENT
    DATABLOCK DB 200H DUP (?)
    FILENAME DB "KMB.DAT","$"
    MESSAGE DB 0AH,0DH,"FILE NOT CREATED
    SUCCESSFULLY",0AH,0DH,"$"
DATA ENDS
CODE SEGMENT
START: MOV AX,DATA ;INITIALIZE DS
        MOV DS,AX
        MOV DX,OFFSET FILENAME ;OFFSET OF FILE NAME
        MOV CX,00H ;FILE ATTRIBUTE IN CX,REFER TO
                    ;APPENDIX B FUNCTION 3CH
        MOV AH,3CH ;t0 CREATE A FILE, FOR more DE-
        INT 21H
  
```



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

- 3.2 Describe the procedure for coding the intersegment and intrasegment jump and call instructions.
- 3.3 Enlist the advantages of assembly language programming over machine language.
- 3.4 What is an assembler?
- 3.5 What is a linker?
- 3.6 Explain various DEBUG commands for troubleshooting executable programs. (Refer to MSDOS Encyclopedia by Ray Duncan.)
- 3.7 What are the DOS function calls?
- 3.8 Write an ALP to convert a four digit hexadecimal number to decimal number.
- 3.9 Write an ALP to convert a four digit octal number to decimal number.
- 3.10 Write an ALP to find out ASCII codes of alphanumeric characters from a look up table.
- 3.11 Write an ALP to change an already available ascending order byte string to descending order.
- 3.12 Write an ALP to perform a 16-bit increment operation using 8-bit instructions.
- 3.13 Write an ALP to find out average of a given string of data bytes neglecting fractions.
- 3.14 Write an ALP to find out decimal addition of sixteen four digit decimal numbers.
- 3.15 Write an ALP to convert a four digit decimal number to its binary equivalent.
- 3.16 Write an ALP to convert a given sixteen bit binary number to its GRAY equivalent.
- 3.17 Write an ALP to find out transpose of a 3x3 matrix.
- 3.18 Write an ALP to find out cube of an 8-bit hexadecimal number.
- 3.19 Write an ALP to display message ‘Happy Birthday!’ on the screen after a key ‘A’ is pressed.
- 3.20 Write an ALP to open a file in the drive C of your hard disk , accept 256 byte entries each separated by comma and then store all these 256 bytes into the opened file. Issue suitable messages where-ever necessary.
- 3.21 Write an ALP to print a message ‘The Printer Is Busy’ on to a dot matrix printer.
- 3.22 Write an ALP to load a file from hard disk of your system into RAM system at segment address 5000H with zero relocation factor.
- 3.23 Write an ALP that goes on accepting the keyboard entries and displays them on line on the CRT display. The control escapes to DOS prompt if enter key is pressed.
- 3.24 Write an ALP to set interrupt vector of type 50H to an address of a routine ISR in segment CODE1.
- 3.25 Write an ALP to set and get the system time. Assume arbitrary time for setting the system time.
- 3.26 What is the function 4CH under INT 21H?



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



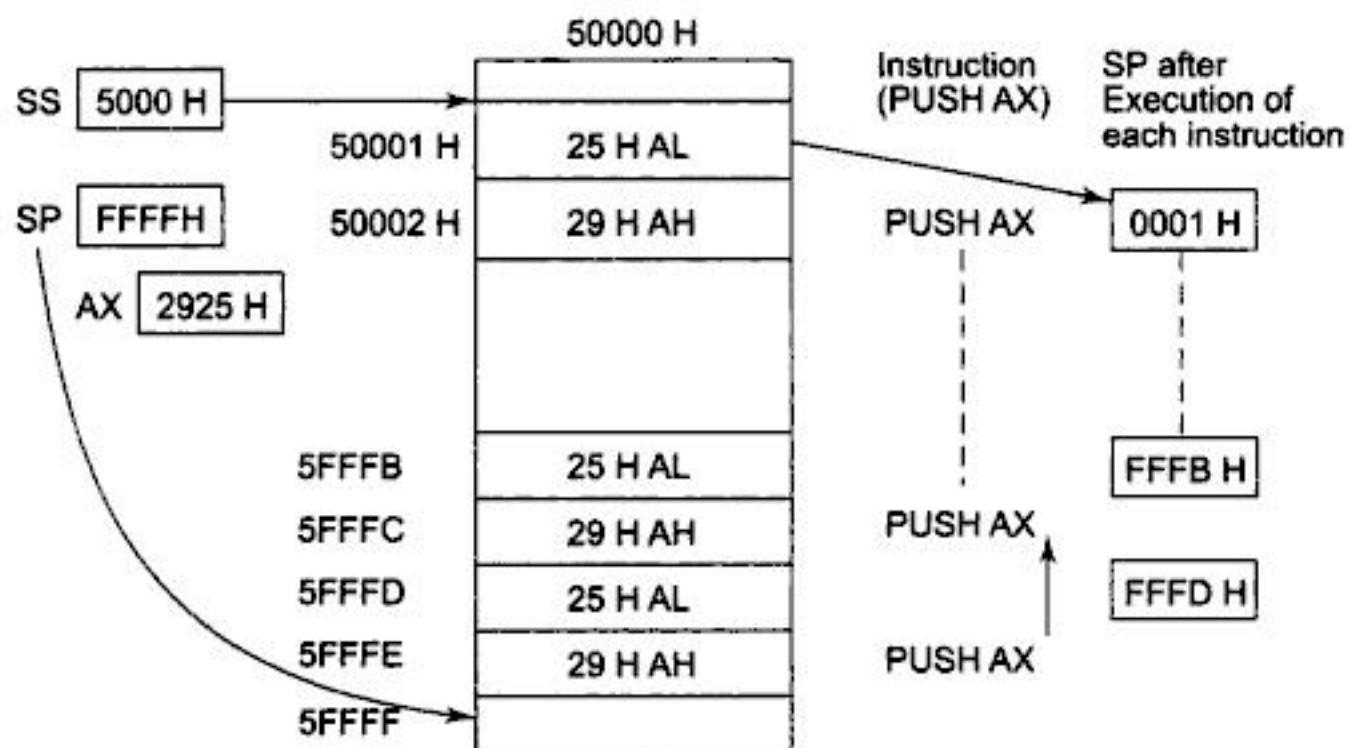
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



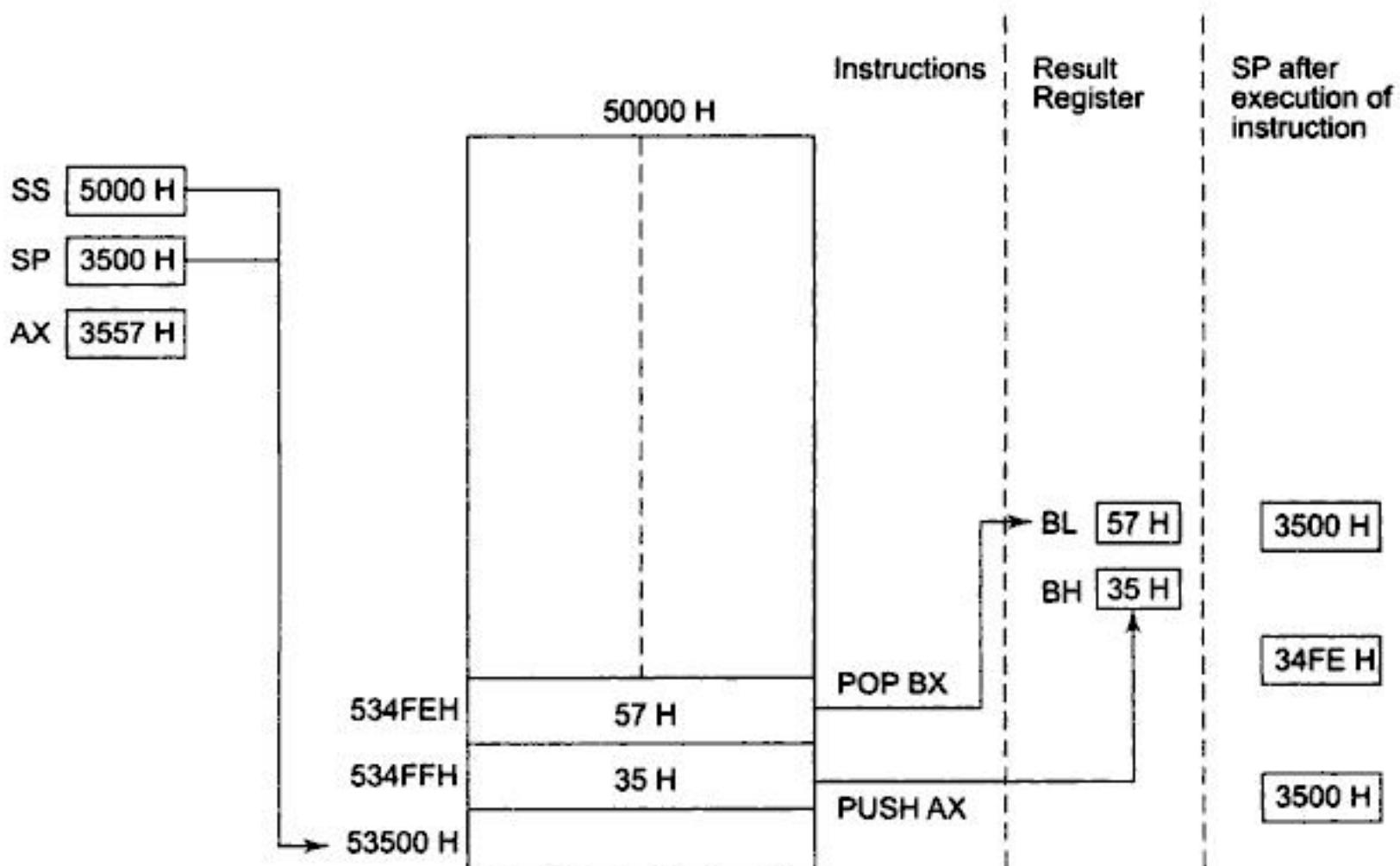
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Thus for a selected value of SS, the maximum value of SP = FFFF H and the segment can have maximum of 64K locations. Thus after starting with an initial value of FFFFH, the Stack Pointer (SP) is decremented by two, whenever a 16-bit data is pushed onto the stack. After successive push operations, when the Stack Pointer contains 0000 H, any attempt to further push the data to the stack will result in stack overflow.

Each PUSH operation decrements the SP as explained above, while each POP operation increments the SP. The POP operation is used to retrieve the data stored on to the stack. Figure 4.2 shows the stack overflow conditions, while Fig. 4.3 shows the effect of PUSH and POP operations on the stack memory block.



**Fig. 4.2** The Execution of Bracketed **PUSH AX** Instruction Results in Stack Overflow



**Fig. 4.3** Effect of **PUSH** and **POP** on SP



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

### 4.3 INTERRUPTS AND INTERRUPT SERVICE ROUTINES

The dictionary meaning of the word 'interrupt' is to break the sequence of operation. While the CPU is executing a program, an 'interrupt' breaks the normal sequence of execution of instructions, diverts its execution to some other program called *Interrupt Service Routine* (ISR). After executing ISR, the control is transferred back again to the main program which was being executed at the time of interruption.

Suppose you are reading a novel and have completed up to page 100. At this instant, your younger brother distracts you. You will somehow mark the line and the page you are reading, so that you may be able to continue after you attend to him. Say you have marked page number 101. You will now go to his room to solve his problem. While you are helping him a friend of yours comes and asks you for a textbook. Now, there are two options in front of you. The first is to make the friend wait till you complete serving your brother, and thereafter you serve his request. In this, you are giving less priority to your friend. The second option is to ask your brother to wait; remember the solution of his problem at the intermediate state; serve the friend; and after the friend is served, continue with the solution that was in the intermediate state. In this case, it may be said that you have given higher priority to your friend. After serving both of them, again you may continue reading from page 101 of the novel. Here, first you are interrupted by your brother. While you are serving your brother, you are again interrupted by a friend. This type of sequence of appearance of interrupts is called nested interrupt, i.e. interrupt within interrupt.

Whenever a number of devices interrupt a CPU at a time, and if the processor is able to handle them properly, it is said to have *multiple interrupt processing capability*. For example, 8085 has five hardware interrupt pins and it is able to handle the interrupts simultaneously under the control of software. In case of 8086, there are two interrupt pins, viz. NMI and INTR. The NMI is a *nonmaskable* interrupt input pin which means that any interrupt request at NMI input cannot be masked or disabled by any means. The INTR interrupt, however, may be masked using the Interrupt Flag (IF). The INTR, further, is of 256 types. The INTR types may be from 00 to FFH(or 00 to 255). If more than one type of INTR interrupt occurs at a time, then an external chip called programmable interrupt controller is required to handle them. The same is the case for INTR interrupt input of 8085. Interrupt Service Routines (ISRs) are the programs to be executed by interrupting the main program execution of the CPU, after an interrupt request appears. After the execution of ISR, the main program continues its execution further from the point at which it was interrupted.

### 4.4 INTERRUPT CYCLE OF 8086/8088

Broadly, there are two types of interrupts. The first out of them is *external interrupt* and the second is *internal interrupt*. In external interrupt, an external device or a signal interrupts the processor from outside or, in other words, the interrupt is generated outside the processor, for example, a keyboard interrupt. The internal interrupt, on the other hand, is generated internally by the processor circuit, or by the execution of an interrupt instruction. The examples of this type are divide by zero interrupt, overflow interrupt, interrupts due to INT instructions, etc.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Suppose an external signal interrupts the processor and the pin **LOCK** goes low at the trailing edge of the first ALE pulse that appears after the interrupt signal preventing the use of bus for any other purpose.

The pin **LOCK** remains low till the start of the next machine cycle.

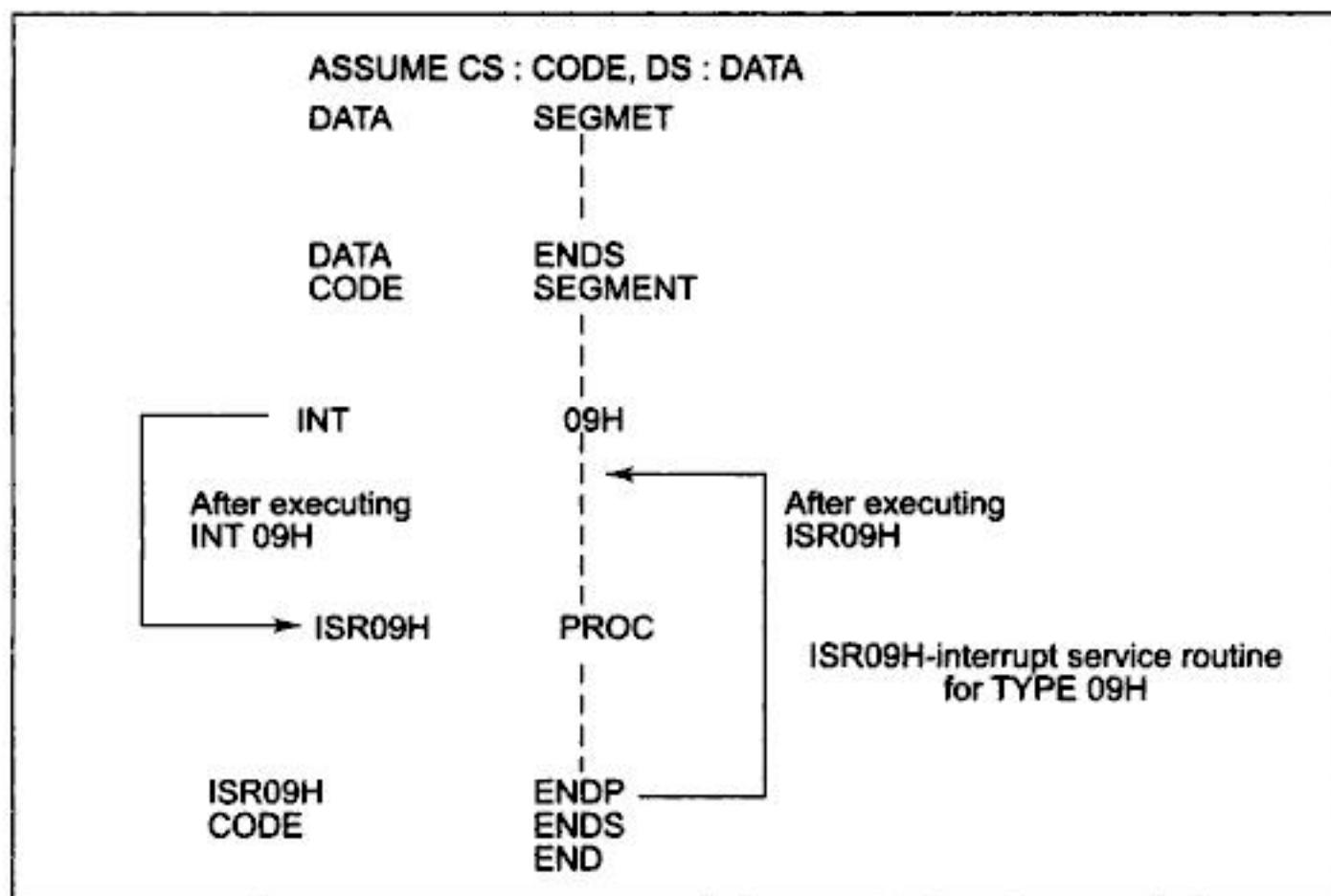
With the trailing edge of LOCK, the INTA goes low and remains low for two clock states before returning back to the high state.

It remains high till the start of the next machine cycle, i.e. next trailing edge of ALE.

Then INTA again goes low, remains low for two states before returning to the high state. The first trailing edge of ALE floats the bus AD<sub>0</sub>-AD<sub>7</sub>, while the second trailing edge prepares the bus to accept the type of the interrupt. The type of the interrupt remains on the bus for a period of two cycles.

## 4.7 INTERRUPT PROGRAMMING

While programming for any type of interrupt, the programmer must, either externally or through the program, set the interrupt vector table for that type preferably with the CS and IP addresses of the interrupt service routine. The method of defining the interrupt service routine for software as well as hardware interrupt is the same. Figure 4.7 shows the execution sequence in case of a software interrupt. It is assumed that the interrupt vector table is initialised suitably to point to the interrupt service routine. Figure 4.8 shows the transfer of control for the nested interrupts.



**Fig. 4.7 Transfer of Control during Execution of an Interrupt Service Routine**

### Program 4.3

Write a program to create a file RESULT and store in it 500H bytes from the memory block starting at 1000:1000, if either an interrupt appears at INTR pin with Type 0AH or an instruction equivalent to the above interrupt is executed.

Note: Pin IRQ<sub>2</sub> available at IO channel of PC is equivalent to Type 0AH interrupt.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
ASSUME CS:CODE2
CODE2 SEGMENT
    MOV AX,DATA
    MOV DS,AX
    MOV BX,NUMBER

CODE2    ENDS
END START
```

---

The CPU general purpose registers may be used to pass parameters to the procedures. The main program may store the parameters to be passed to the procedure in the available CPU registers and the procedure may use the same register contents for execution. The original contents of the used CPU register may change during execution of the procedure. This may be avoided by pushing all the register content to be used to the stack sequentially at the start of the procedure and by popping all the register contents at the end of the procedure in opposite sequence.

---

#### Example 4.2

```
ASSUME CS:CODE
CODE SEGMENT
    START :   MOV AX,5555H
               MOV BX,7272H
               •
               •
               CALL PROCEDURE1
               •
               •
               •
PROCEDURE  PROCEDURE1 NEAR
               •
               •
               ADD AX,BX
               •
               •
               RET
PROCEDURE1 ENDP
CODE ENDS
END START
```

---

Memory locations may also be used to pass parameters to a procedure in the same way as registers. A main program may store the parameter to be passed to a procedure at a known memory address location and the procedure may use the same location for accessing the parameter.

---

#### Example 4.3

```
ASSUME CS:CODE, DS:DATA
DATA SEGMENT
NUM DB (55H)
```



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
START :      MOV AX,DATA
              MOV DS,AX
              .
              .
              CALL FAR_PTR ROUTINE1
              .
              .
              CALL FAR_PTR ROUTINE2
              .
              .
CODE1        ENDS
PROCEDURE     ROUTINE1 FAR
              .
              .
ROUTINE1      ENDP
PROCEDURE     ROUTINE2 FAR
              .
              .
ROUTINE2      ENDP
END          START
```

---

#### 4.10 MACROS

Till now, we have studied the stack, subroutines, interrupts and interrupt service routines. It is a notable point that the control is transferred to a subroutine or an interrupt service routine whenever it is called or an interrupt signal appears at the interrupt pin of the processor. After executing these routines the control is again transferred back to the main calling program. Hence rather than writing a complete routine again and again, one may call it as many times as required. This imparts flexibility in programming as well as ease of troubleshooting. The concept of subroutine as well as interrupt service routine can be compared with an office where the main (calling) program acts as a head while the subroutines and interrupt service routines act as subordinates. The head may ask his subordinates to work out a particular task and be ready with the results. Here the main program calls subroutines and interrupt service routines and may refer the results of their execution for further processing. The subroutines and interrupt service routines are assigned labels for references.

The macro is also a similar concept. Suppose, a number of instructions are repeating through in the main program, the listings becomes lengthy. So a macro definition, i.e. a label, is assigned with the repeatedly appearing string of instructions. The process of assigning a label or macroname to the string is called defining a macro. A macro within a macro is called a nested macro. The macroname or macro definition is then used throughout the main program to refer to that string of instructions.

The difference between a macro and a subroutine is that in the macro the complete code of the instructions string is inserted at each place where the macro-name appears. Hence the EXE file becomes lengthy. Macro does not utilise the service of stack. There is no question of transfer of control as the program using the macro inserts the complete code of the macro at every reference of the macroname. On the other hand, subroutine is called whenever necessary, i.e. the control of execution is transferred to the subroutine, every time it is called. The executable code in case of the subroutines becomes



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```

RET          ; becomes zero and
DELAY ENDP    ; return to main
              ; program

```

#### Program 4.5 ALP to Generate 100 ms Delay

The exact delay obtained using the above routine can be calculated as shown:

$$\begin{aligned}
 T_d (\text{exact}) &= 0.1 * 4 + (2 + 3) * 47619 * 0.1 + 16 * 47618 * 0.1 + 4 * 0.1 + 8 * 0.1 \\
 &= 0.4 + 23809.5 + 76188.8 + 0.4 + .8 \\
 &= 99999.998 \mu\text{s} = 99.9999 \text{ ms} \\
 &\approx 100 \text{ ms}
 \end{aligned}$$

Note that if the zero condition is satisfied, the JNZ instruction takes only 4 clock states, otherwise, the instruction takes 16 clock states for execution. Also the instructions MOV CX, BA03 H and RET are executed only once during the execution of the delay loop.

It may be observed that in the above delay there is an error of 0.1 ms. The error is only of one clock state, and it cannot be corrected by adding further instructions to the ALP, after the JNZ instruction because the smallest execution time of an 8086 instruction is 2 states, i.e. 0.2  $\mu\text{s}$  for this system.

In case of a 16-bit count register, the maximum count value can be FFFFH. This may put a limitation on the maximum delay that can be generated using these instructions. Whenever large delays are required, more than one count register may be used to serve the purpose. Program 4.6 explains the use of another count register BX to obtain the required large delay.

#### Program 4.6

Using the ALP of Program 4.5 design a delay of ten minutes.

**Solution** Required delay  $T_d = 10 \text{ minutes} = 600 \text{ sec}$

Instructions selected	Clock states
MOV BX, Count 1	4
MOV CX, Count 2	4
DEC CX	2
DEC BX	2
JNZ 1able	16
NOP	3
RET	8

Clock frequency = 10 MHz

$T = 0.1 \mu\text{sec}$

There will be two nested counter loops for decrementing the two counting registers. Let the first loop has a count FFFF.

count2 = FFFFH

PROC DELAY LOCAL

ASSUME CS : CODE

CODE SEGMENT

    MOV BX, Count1 ; Load count1.

BBB : MOV CX, Count2 ; Load count2, i.e. FFFFH.

CCC : NOP ;



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

microprocessor. Thus each dedicated peripheral device needs suitable initialisation. However memory, unlike the peripheral devices, does not need any initialisation and does not directly participate in the process of communication between CPU and the user. Rather, it acts as a media for the communication between a peripheral with the microprocessor. While memory may be treated as a peripheral, on the other hand, peripheral devices are often treated as memory locations. Thus as far as the CPU is concerned, there is no special difference in the method of handling memory or peripherals, except for the use of respective control signals. In this chapter, we will present interfacing techniques of semiconductor memories, I/O ports and a few other peripherals with 8086/8088.

## 5.1 SEMICONDUCTOR MEMORY INTERFACING

Semiconductor memories are of two types, viz. RAM (Random Access Memory) and ROM (Read Only Memory).

### 5.1.1 Static RAM Interfacing

The semiconductor RAMs are of broadly two types—static RAM and dynamic RAM. In this section, we will consider the interfacing of static RAM and ROM with 8086/8088. The semiconductor memories are organised as two dimensional arrays of memory locations. For example,  $4K \times 8$  or  $4K$  byte memory contains 4096 locations, where each location contains 8-bit data and only one of the 4096 locations can be selected at a time. Once a location is selected all the bits in it are accessible using a group of conductors called ‘data bus’. Obviously, for addressing  $4K$  bytes of memory, twelve address lines are required. In general, to address a memory location out of  $N$  memory locations, we will require at least  $n$  bits of address, i.e.  $n$  address lines where  $n = \log_2 N$ . Thus if the microprocessor has  $n$  address lines, then it is able to address at the most  $N$  locations of memory, where  $2^n = N$ . However, if out of  $N$  locations only  $P$  memory locations are to be interfaced, then the least significant  $p$  address lines out of the available  $n$  lines can be directly connected from the microprocessor to the memory chip while the remaining  $(n-p)$  higher order address lines may be used for address decoding (as inputs to the chip selection logic). The memory address depends upon the hardware circuit used for decoding the chip select ( $\overline{CS}$ ). The output of the decoding circuit is connected with the  $\overline{CS}$  pin of the memory chip.

The general procedure of static memory interfacing with 8086 is briefly described as follows:

1. Arrange the available memory chips so as to obtain 16-bit data bus width. The upper 8-bit bank is called ‘odd address memory bank’ and the lower 8-bit bank is called ‘even address memory bank’, as described in memory organisation in Chapter 1.
2. Connect available memory address lines of memory chips with those of the microprocessor and also connect the memory  $\overline{RD}$  and  $\overline{WR}$  inputs to the corresponding processor control signals. Connect the 16-bit data bus of the memory bank with that of the microprocessor 8086.
3. The remaining address lines of the microprocessor,  $\overline{BHE}$  and  $A_0$  are used for decoding the required chip select signals for the odd and even memory banks. The  $\overline{CS}$  of memory is derived from the O/P of the decoding circuit.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Let us select a variable  $C$  for memory address pulse, i.e. output of 6-input NAND gate.  $\overline{BHE}$  is abbreviated as  $B$ . The chip selection logic can be designed as shown in Table 5.4.

**Table 5.4**

I/P		O/P	
$A_0$	$B(\overline{BHE})$	$C_1$	$C_2$
0	0	0	0
0	1	1	0
1	0	0	1
1	1	1	1

$$C_2 = A_0$$

$$C_1 = \overline{BHE}$$

To find out  $\overline{CS}_1$  and  $\overline{CS}_2$  we will have to combine  $C_1$  and  $C_2$  with  $C$ .

**Table 5.5**

I/P			O/P	
$C_1$	$C_2$	$C$	$\overline{CS}_1$	$\overline{CS}_2$
0	0	0	0	0
1	0	0	1	0
1	1	0	1	1
0	1	0	0	1

Table 5.5 shows that

$$\overline{CS}_1 = C + C_1 = C + \overline{BHE} \text{ and } \overline{CS}_2 = C + C_2 = C + A_0$$

Similarly we can find out  $CS_3$  and  $CS_4$ .

### Problem 5.3

It is required to interface two chips of  $32K \times 8$  ROM and four chips of  $32K \times 8$  RAM with 8086, according to the following map.

ROM 1 and 2 F0000H - FFFFFH, RAM 1 and 2 D0000H - DFFFFH

RAM 3 and 4 E0000H - EFFFFH

Show the implementation of this memory system.

**Solution** Let us write the memory map of the system as shown in Table 5.6.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

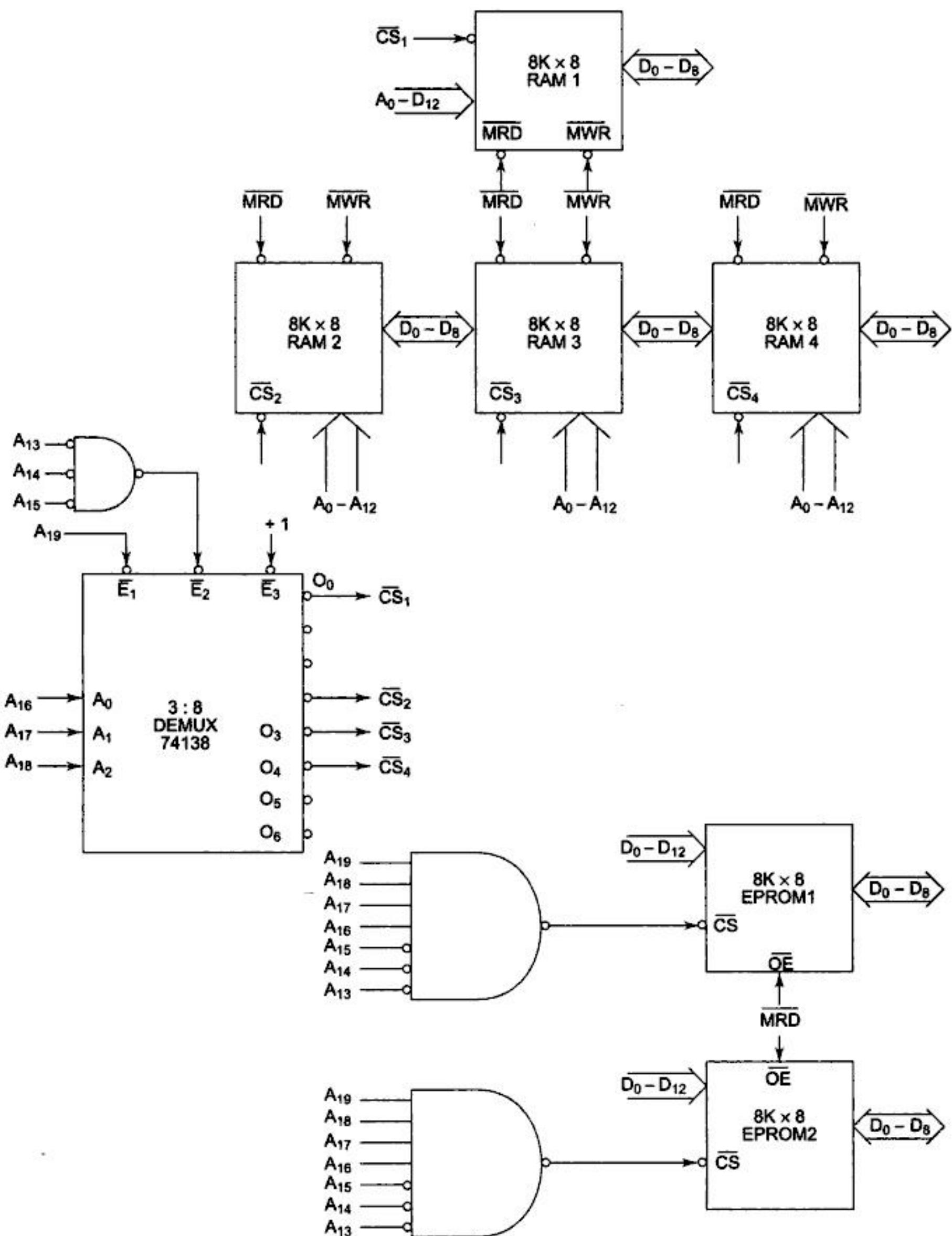


Fig. 5.4 Interfacing Problem 5.4



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

The refresh cycle is different from the memory read cycle in the following aspects.

1. The memory address is not provided by the CPU address bus, rather, it is generated by a refresh mechanism counter known as refresh counter.
2. Unlike memory read cycle, more than one memory chip may be enabled at a time so as to reduce the number of total memory refresh cycles.
3. The data enable control of the selected memory chip is deactivated, and data is not allowed to appear on the system data bus during refresh, as more than one memory units are refreshed simultaneously. This is to avoid the data from the different chips to appear on the bus simultaneously.
4. Memory read is either a processor initiated or an external bus master initiated operation while memory refresh is an independent regular activity, initiated and carried out by the refresh mechanism.

Generally, dynamic RAM is available in units of several kilobits to even megabits of memory (note that it is not in terms of bytes or nibbles as in a static RAM). This memory is arranged internally in a two dimensional matrix array so that it will have  $n$  rows and  $m$  columns. The row address  $n$  and column address  $m$  are important for the refreshing operation. For example, a typical 4K bit dynamic RAM chip has an internally arranged bit array of dimension  $64 \times 64$ , i.e. 64 rows and 64 columns. Thus the row address and column address will require 6 bits each. These 6 bits for each row address and column address will be generated by the refresh counter, during the refresh cycles. A complete row of 64 cells is refreshed at a time to minimize the refreshing time. Thus the refresh counter needs to generate only row addresses. The row addresses are multiplexed, over lower order address lines. The refresh signals act to control the multiplexer, i.e. when refresh cycle is in process the refresh counter puts the row address over the address bus for refreshing. Otherwise, the address bus of the processor is connected to the address bus of DRAM, during normal processor initiated activities. A timer, called *refresh timer*, derives a pulse for refreshing action after each refresh interval, which can be qualitatively defined as the time for which a dynamic RAM cell can hold data charge level practically constant, i.e. no data loss takes place. Suppose the typical dynamic RAM chip has 64 rows, then each row should be refreshed after each refresh interval, or in other words, all the 64 rows are to be refreshed in a single refresh interval. This refresh interval depends upon the manufacturing technology of the dynamic RAM cell. It may range anywhere from 1 ms to 3 ms. Let us consider 2 ms as a typical refresh time interval. Hence, the frequency of the refresh pulses will be calculated as shown:

$$\text{Refresh Time (per row)} \quad t_r = \frac{2 \times 10^{-3}}{64}$$

$$\text{Refresh Frequency} \quad f_r = \frac{64}{2 \times 10^{-3}} = 32 \times 10^3 \text{ Hz}$$

The block diagram in Fig. 5.7 explains the refreshing logic and 8086 interfacing with dynamic RAM. Each of the used chips is a  $16\text{K} \times 1$ -bit dynamic RAM cell array. The system contains two 16K byte dynamic RAM units. All the address and data lines are assumed to be available from an 8086 microcomputer system. The  $\overline{\text{OE}}$  pin controls output data buffers of the memory chips. The CE pins are active high chip selects of memory chips. The refresh cycle starts, if the refresh output of the refresh timer goes high,  $\overline{\text{OE}}$  and CE also tend to go high. The high CE enables the memory chip for refreshing,



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

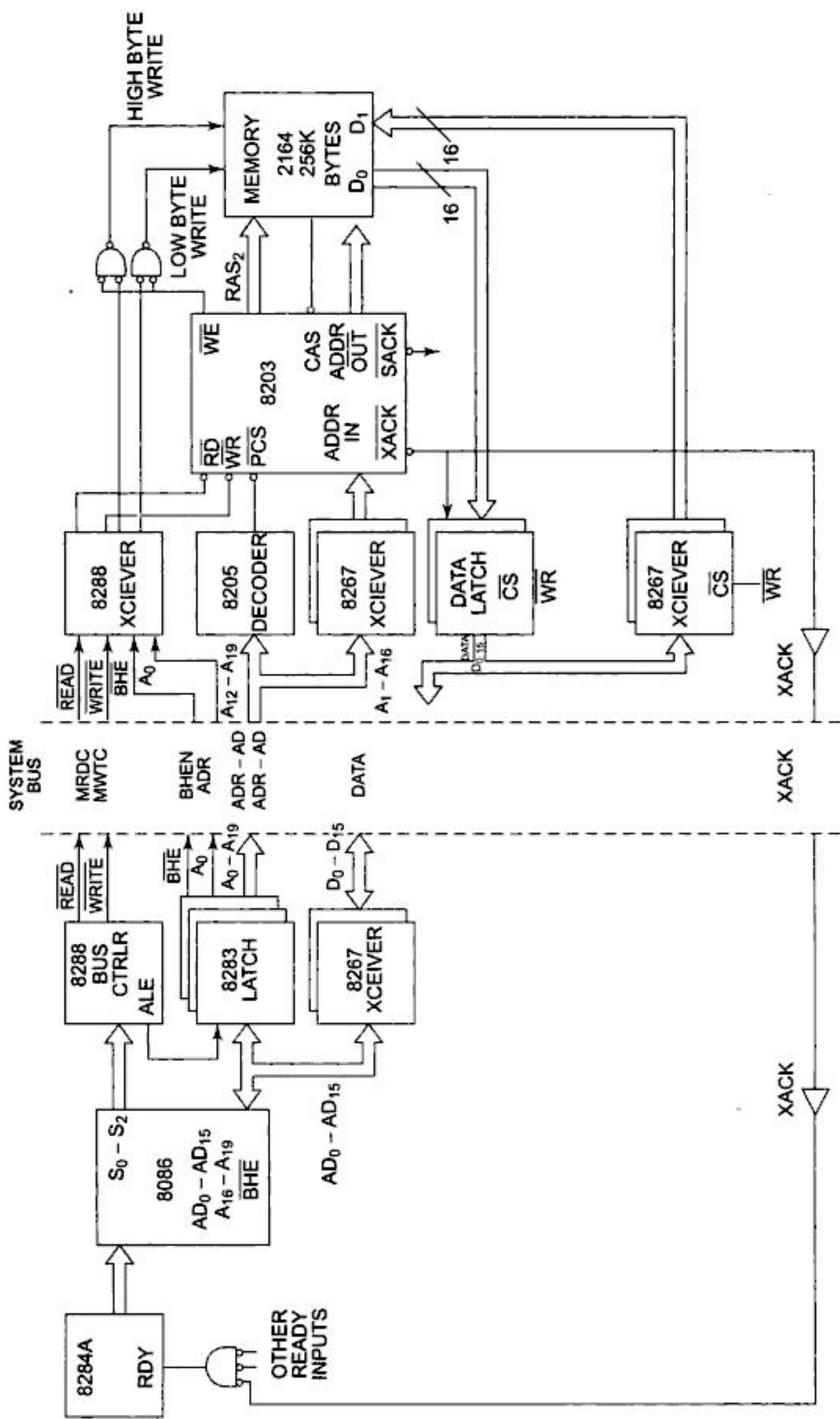


Fig. 5.9 Interfacing 2164 Using 8203



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**Problem 5.6**

Interface an input port 74LS245 to read the status of switches SW<sub>1</sub> to SW<sub>8</sub>. The switches, when shorted, input a '1' else input a '0' to the microprocessor system. Store the status in register BL. The address of the port is 0740H.

**Solution** The hardware interface circuit is shown in Fig. 5.12. The address, control and data lines are assumed to be readily available at the microprocessor system.

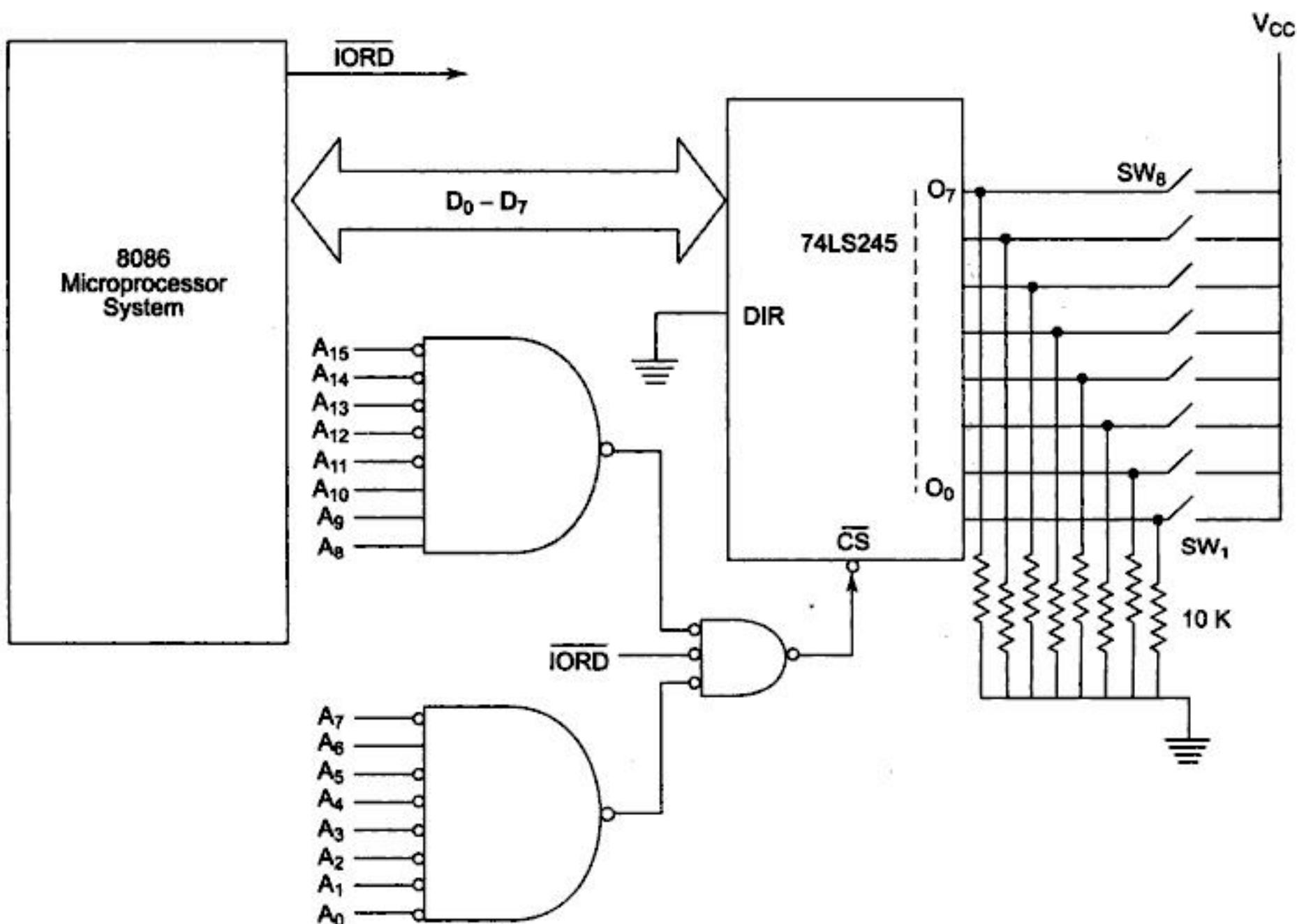


Fig. 5.12 Interfacing Input Port 74LS245

The ALP is given as follows:

```

MOV BL, 00H      ; Clear BL for status
MOV DX, 0740H    ; 16-bit port address in DX
IN AL, DX        ; Read port 0740H for switch positions
MOV BL, AL        ; Store status of switches from AL into BL
HLT              ; Stop

```

**Program 5.1**

ALP for Problem 5.6.

Here LSB bit of BL corresponds to the status of SW<sub>1</sub>, and likewise the MSB of BL corresponds to the



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**Solution** Let us select the two port addresses 0004H and 0008H for the output ports. The first port 0004H outputs 7-seg code while the second output port 0008H selects the display by grounding the common cathode. The hardware is given in Fig. 5.15.

The 7-seg codes for C.C. displays can be decided as given. For a LED to be 'on', that particular anode should be 1 and the common cathode line should be grounded, using a port line that drives a transistor. Thus for the numbers to be displayed the code is calculated as shown.

Decimal no.	a	b	c	d	e	f	g	dp
	A <sub>7</sub>	A <sub>6</sub>	A <sub>5</sub>	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>
1-	1	1	0	0	0	0	0	0 = C0
2-	1	1	0	1	1	0	1	0 = DA
3-	1	1	1	1	0	0	1	0 = F2
4-	0	1	1	0	0	1	1	0 = 66
5-	1	0	1	1	0	1	1	0 = B6

These codes are stored in a look up table starting from 2000H:0000, as shown below.

```
2000 : 0000 → C0 H
2000 : 0001 → DA H
2000 : 0002 → F2 H
2000 : 0003 → 66 H
2000 : 0004 → B6 H
```

Only one display should be selected at a time, i.e. only the corresponding bit of port 2 should be high for selecting a common cathode display. All the other bits should be low to keep the other displays disabled. Thus to enable the least significant display, the LSB of the 8-bit selected port should remain '1'. Hence AL should have 01 or E1H in it to select the least significant display. The codes for the selection of displays and 7-segment codes directly depend upon the hardwired connections between them.

```

MOV AX, 2000H      ; Initialize pointer to
MOV DS, AX          ; Code table DS:BX
MOV BX, 0000H
NEXT : MOV AL, 00H    ; Get 1st number from the table.
          MOV DH, AL
          MOV CL, 05H    ; Count for display
          MOV DL, E1H    ; Selection code for 1st display
AGAIN : XLAT           ; 
          OUT 04H, AL    ; Out the code for the first
                           ; number to port 04H.
          MOV AL, DL    ; Get to be enabled display code.
          OUT 08H, AL    ; Select 1st display.
          ROL DL         ; decide code for selecting next
          INC DH         ; display for next number
          MOV AL, DH    ; get next num. to be displayed.
          LOOP AGAIN     ; Repeat five times
          JMP NEXT       ; Continue the procedure

```

Program 5.4 ALP for Problem 5.9



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

PA <sub>3</sub>	1	40	PA <sub>4</sub>
PA <sub>2</sub>	2	39	PA <sub>5</sub>
PA <sub>1</sub>	3	38	PA <sub>6</sub>
PA <sub>0</sub>	4	37	PA <sub>7</sub>
RD	5	36	WR
CS	6	35	RESET
GND	7	34	D <sub>0</sub>
A <sub>1</sub>	8	33	D <sub>1</sub>
A <sub>0</sub>	9	32	D <sub>2</sub>
PC <sub>7</sub>	10	31	D <sub>3</sub>
PC <sub>6</sub>	11	30	D <sub>4</sub>
PC <sub>5</sub>	12	29	D <sub>5</sub>
PC <sub>4</sub>	13	28	D <sub>6</sub>
PC <sub>0</sub>	14	27	D <sub>7</sub>
PC <sub>1</sub>	15	26	V <sub>CC</sub>
PC <sub>2</sub>	16	25	PB <sub>7</sub>
PC <sub>3</sub>	17	24	PB <sub>6</sub>
PB <sub>0</sub>	18	23	PB <sub>5</sub>
PB <sub>1</sub>	19	22	PB <sub>4</sub>
PB <sub>2</sub>	20	21	PB <sub>3</sub>

**Fig. 5.17(b) 8255A Pin Configuration**

**PA<sub>7</sub>-PA<sub>0</sub>** These are eight port A lines that act as either latched output or buffered input lines depending upon the control word loaded into the control word register.

**PC<sub>7</sub>-PC<sub>4</sub>** Upper nibble of port C lines. They may act as either output latches or input buffers lines. This port also can be used for generation of handshake lines in mode 1 or mode 2.

**PC<sub>3</sub>-PC<sub>0</sub>** These are the lower port C lines, other details are the same as PC7-PC4 lines.

**PB<sub>0</sub>-PB<sub>7</sub>** These are the eight port B lines which are used as latched output lines or buffered input lines in the same way as port A.

**RD** This is the input line driven by the microprocessor and should be low to indicate read operation to 8255.

**WR** This is an input line driven by the microprocessor. A low on this line indicates write operation.

**CS** This is a chip select line. If this line goes low, it enables the 8255 to respond to RD and WR signals, otherwise RD and WR signals are neglected.

**A<sub>1</sub>-A<sub>0</sub>** These are the address input lines and are driven by the microprocessor. These lines (A<sub>1</sub> – A<sub>0</sub>) with RD, WR and CS form the following operations for 8255. These address lines are used for



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

(Contd.)

I/O Address lines														Hex. Port Addresses			
Ports	$A_{15}$	$A_{14}$	$A_{13}$	$A_{12}$	$A_{11}$	$A_{10}$	$A_{09}$	$A_{08}$	$A_{07}$	$A_{06}$	$A_{05}$	$A_{04}$	$A_{03}$	$A_{02}$	$A_{01}$	$A_{00}$	
Port C	0	0	0	0	0	1	1	1	0	1	0	0	0	1	0	0	0744H
CWR	0	0	0	0	0	1	1	1	0	1	0	0	0	1	1	0	0746H

Let us use absolute decoding scheme that uses all the 16 address lines for deriving the device address pulse. Out of  $A_0 - A_{15}$  lines, two address lines  $A_{02}$  and  $A_{01}$  are directly required by 8255 for the three port and CWR address decoding. Hence only  $A_3$  to  $A_{15}$  are used for decoding addresses. The complete hardware scheme is shown in Fig. 5.19. In the diagram, the 8086 is assumed to be in the maximum mode so that  $\overline{\text{IORD}}$  and  $\overline{\text{IOWR}}$  are readily available. If the 8086 is in minimum mode,  $\overline{\text{RD}}$  and  $\overline{\text{WR}}$  of 8086 are to be connected accordingly to 8255 and  $\text{M}/\overline{\text{IO}}$  pin is combined with the chip select of above hardware suitably so as to select the 8255 when  $\text{M}/\overline{\text{IO}}$  is low.

The ALP for the problem is developed as follows:

```

MOV DX, 0746 H ; Initialise CWR with
MOV AL, 82 H    ; control word 82H
OUT DX, AL      ;
SUB DX,04       ; Get address of port B in DX
IN AL, DX       ; Read port B for switch
SUB DX,02       ; positions in to AL and get port A address
                ; in DX.
OUT DX, AL      ; Display switch positions on port A
MOV BL, 00 H    ; Initialise BL for switch count
MOV CH, 08H    ; Initialise CH for total switch number
YY: ROL AL      ; Rotate AL through carry to check,
                ; whether the switches are on or
                ; off, i.e. either 1 or 0
INC BL          ;
XX : DEC CH      ; Check for next switch. If
                ; all switch are checked, the
                ; number of on switches are
                ; in BL. Display it on port C
ADD DX, 04      ;
OUT DX,AL       ; lower.
HLT             ; Stop

```

#### Program 5.5 ALP for Problem 5.10

#### Problem 5.11

Interface a 4\*4 Keyboard with 8086 using 8255. and write an ALP for detecting a key closure and return the key code in AL. The debouncing period for a key is 10 ms. Use software key debouncing technique. DEBOUNCE is an available 10 ms delay routine.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

hence only one segment is used for storing the program code, i.e. code segment (CS). This program is written in MASM syntax. The 8255 is again interfaced to the lower byte of the 8086 data bus. Absolute decoding scheme is not used here to implement the circuit using minimum hardware.

```

CODE --- SEGMENT
ASSUME CS : CODE
START: MOV AL, 82H      ; Load CWR with
        MOV DX, 8006H    ; control word
        OUT DX, AL       ; required
        MOV BL, 00H       ; Initialize BL for key code
        XOR AX, AX       ; Clear all flags
        MOV DX, 8000H    ; Port Address in AX.
        OUT DX, AL       ; Ground all rows.
        ADD DX, 02       ; Port B address in DX.
WAIT :  IN AL, DX      ; Read all columns.
        AND AL, 0F H     ; Mask data lines D7-D4.
        CMP AL, 0F H     ; Any key closed?
        JZ WAIT          ; If not, wait till key
        CALL DEBOUNCE   ; closure else wait for 10 ms
        MOV AL, 7FH      ; Load data byte to ground
        MOV BH, 04H      ; a row and set row counter.
NXTROW: ROL AL, 01     ; Rotate AL to ground next row.
        MOV CH, AL       ; Save data byte to ground next row.
        SUB DX, 02       ; Output port address is in DX.
        OUT DX, AL       ; Ground one of the rows.
        ADD DX, 02       ; Input port address is in DX.
        IN AL, DX        ; Read input port for key closure.
        AND AL, 0FH      ; Mask D4-D7.
        MOV CL, 04H      ; Set column counter.
NXTCOL: ROR AL, 01     ; Move D0 in CF.
        JNC CODEKY      ; Key closure is found, if CF=0.
        INC BL          ; Increment BL for next binary
        MOV CL, 04H      ; key code.
        DEC CL          ; Decrement column counter,
        JNZ NXTCOL      ; if no key closure found.
        MOV AL, CH      ; Check for key closure in next column
        DEC BH          ; Load data byte to ground next row.
        JNZ NXTROW      ; if no key closer found in column
        MOV AL, CH      ; get ready to ground next row.
NXTROW: JMP WAIT       ; Go back to ground next row.
        JNC CODEKY      ; Jump back to check for key
        MOV AL, CH      ; closure again.
CODEKY: MOV AL, BL     ; Key code is transferred to AL.
        MOV AH, 4CH      ; Return to DOS prompt.
        INT 21 H

```

This procedure generates 10 ms delay at 5 MHz operating frequency.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```

NXTDGT : MOV BX, 0000H ; Set pointer to look up table
      MOV AL, CH   ; First no to display
      XLAT       ; Store number to be displayed in AL.
      OUT 80H,AL  ; Find code from look up table
      MOV AL, DL   ; Display the code
      OUT 81H,AL  ; Enable the display
      OUT 81H,AL  ;
      ROL DL     ; Go for selecting the next display
      INC CH     ; Next number to display
      DEC CL     ; Decrement count.
      JNZ NXTDGT ; Go for next digit display
      JMP AGAIN  ; Repeat the procedure

```

Program 5.7 ALP for Problem 5.13

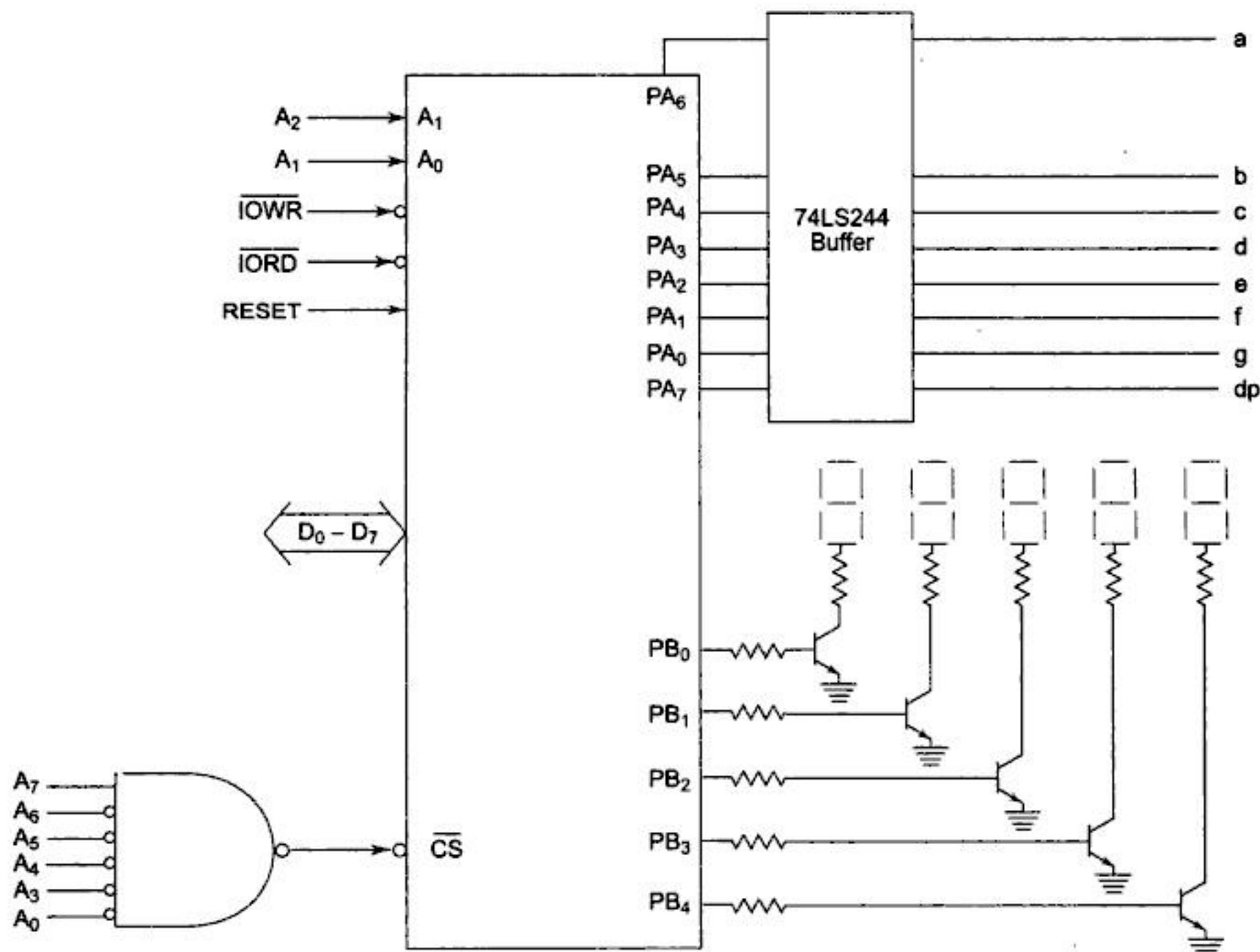


Fig. 5.26 Interfacing Multiplexed 7-Segment Display



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**Table 5.12** Pin Connections and Descriptions for Centronics-type Parallel Interface to IBM PC and EPSON FX-100 Printers

Printer Controller				
Signal Pin No.	Return Pin No.	Signal Name	Direction	Description
1	19	STROBE	IN	STROBE pulse to read data in. Pulse width must be more than 0.5 $\mu$ s at receiving terminal. The signal level is normally "high"; read-in of data is performed at the "low" level of this signal.
2	20	DATA 1	IN	These signals represent information of the 1st to 8th bits of parallel data respectively. Each signal is at "high" level when data is logical "1" and "low" when logical "0".
3	21	DATA 2	IN	
4	22	DATA 3	IN	
5	23	DATA 4	IN	
6	24	DATA 5	IN	
7	25	DATA 6	IN	
8	26	DATA 7	IN	
9	27	DATA 8	IN	
10	28	ACKNLG	OUT	Approximately 5 $\mu$ s pulse; "low" indicates the data has been received and the printer is ready to accept other data.
11	29	BUSY	OUT	A "high" signal indicates that the printer cannot receive data. The signal becomes "high" in the following cases. 1. During data entry. 2. During printing operation. 3. In "outline" state. 4. During printer error status.
12	30	PE	OUT	A "high" signal indicates that the printer is out of paper.
13	—	SLCT	OUT	This signal indicates that the printer is in the selected state.
14	—	AUTO FEED XT	IN	With this signal being at "low" level, the paper is automatically fed one line after printing. (The signal level can be fixed to "low" with DIPSW Pin 2-3 provided on the control circuit board.)
15	—	NC	—	Not used.
16	—	0V	—	Logic GND Level.
17	—	CHASIS GND	—	Printer chassis GND. In the printer, the chassis GND and the logic GND are isolated from each other.
18	—	NC	—	Not used.
19-30	—	GND	—	"Twisted-Pair Return" signal; GND level.

(Contd.)



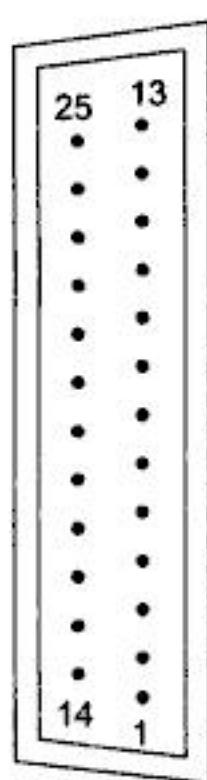
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



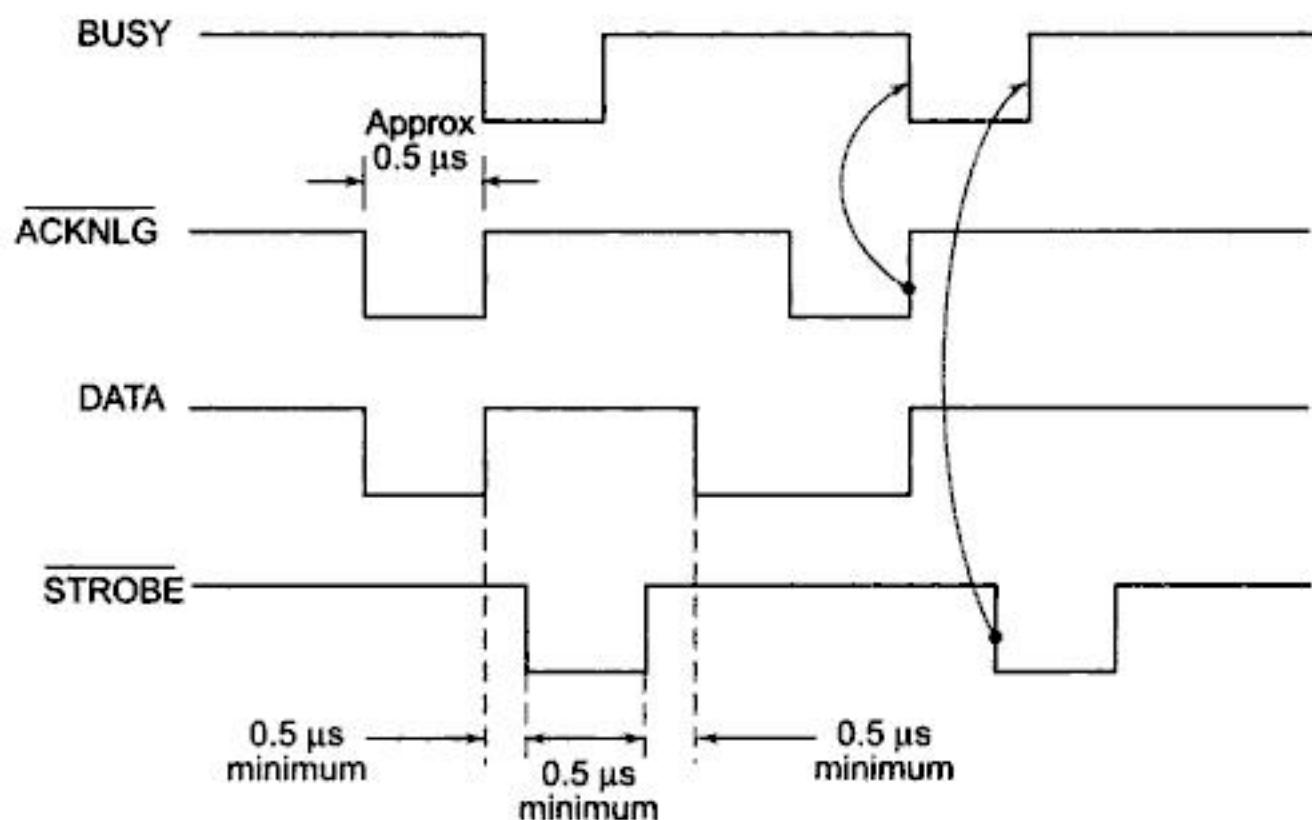
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



**Fig. 5.32** Centronics Printer Connector



**Fig. 5.33** Timing Waveforms of Data Transfer to a Centronix Compatible Parallel Printer

flow and synchronization between the data transmitter and receiver. The interrupt generation and other functions are similar to mode 1. Thus in this mode, 8255 is a bidirectional 8-bit port with handshake signals. The  $\overline{RD}$  and  $\overline{WR}$  signals decide whether the 8255 is going to operate as an input port or output port.

The salient features of mode 2 of 8255 are listed as follows:

1. The single 8-bit port in group A is available.
2. The 8-bit port is bidirectional and additionally a 5-bit control port is available.
3. Three I/O lines are available at port C, viz.  $PC_2-PC_0$ .
4. Inputs and outputs are both latched.
5. The 5-bit control port C ( $PC_3-PC_7$ ) is used for generating/accepting handshake signals for the 8-bit data transfer on port A.

#### ***Control signal definitions in mode 2***

**INTR (Interrupt request)** As in mode 1, this control signal is active high and is used to interrupt the microprocessor to ask for transfer of the next data byte to/from it. This signal is used for input (read) as well as output (write) operations.

#### ***Control signals for output operations***

**OBF (Output buffer full)** This signal, when falls to logic low level, indicates that the CPU has written data to port A.

**ACK (Acknowledge)** This control input, when falls to logic low level, acknowledges that the previous data byte is received by the destination and the next byte may be sent by the processor. This signal enables the internal tristate buffers to send out the next data byte on port A.

**INTE1 (A flag associated with OBF)** This can be controlled by bit set/reset mode with  $PC_6$ .



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```

MOV AX, DATA           ; Initialise data segment
MOV DS, AX
MOV AL, CW1            ; Initialise 8255 in
MOV DS, AX              ; mode 2
OUT F6H, AL
STI
MOV [SI], OFFSET BLOCK1-1
TRANS : INT 2
CALL FAR PTR SYNCHRO   ; Wait for synchronization
INC SI                 ; Pointer to block in SI
DEC CL                 ; Decrement count
JZ STOP                ; If = 0, then stop else
MOV AL,[SI]              ; Go for transfer of the next
OUT FOH,AL              ; byte and out it to port A
WAIT : JMP WAIT          ; Wait for acknowledgement
STOP : HLT                ; Stop if the complete block
CODE ENDS               ; is transferred
END

```

**Program 5.9(a) Transmitter ALP for Problem 5.15**

; The receiver program receives data bytes  
; transmitted by the other system and stores  
; them in the array as asked in the program.

```

STACK SEGMENT
STACKD DB 500H
STACK ENDS
DATA SEGMENT
CW2 EQU COH
BLOCK2 DB 100D DUP (?)
DATA ENDS
CODE SEGMENT
ASSUME CS : CODE, DS : DATA ,SS: STACK
MOV AX, STACK
MOV SS, AX
MOV AX, 0000H           ; Initialise interrupt
MOV DS, AX              ; vector table
MOV [0008H],OFFSET NEXT
MOV [000AH],SEG NEXT
MOV CL,101 D             ; count for bytes
MOV AX,DATA              ; Initialise data segment
MOV DS,AX
MOV AL,CW2                ; Initialise 8255 in mode 2
OUT 86H,AL                ; to receive data
MOV SI,OFFSET BLOCK2-1    ; Point to block 2
WAIT : JMP WAIT            ; to store received data and wait
NEXT : INC SI              ; Increment SI, point to start

```



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

<i>Pin</i>	<i>Symbol</i>	<i>Description</i>
5.	B12	Bit 12
6.	B11	Bit 11
7.	B10	Bit 10
8.	B9	Bit 9
9.	B8	Bit 8
10.	B7	Bit 7
11.	B6	Bit 6
12.	B5	Bit 5
13.		B4
14.		B3
15.		B2
16.		<b>B1 (Least Significant Bit)</b>
17.	TEST	<p>Input High—Normal Operation.</p> <p>Input Low—Forces all bit outputs high</p> <p>Note: This input is used for test purposes only.</p> <p>Tie high if not used.</p>
18.	LBEN	<p>Low Byte Enable—With Mode (Pin 21) low, and <u>CE/LOAD</u> (Pin 20) low, taking this pin low activates low order byte outputs B1–B8.</p> <p>— With Mode (Pin 21) high, this pin serves as a low byte flag output used in handshake mode.</p>
19.	HBEN	<p>High Byte Enable—With Mode (Pin 21) low, and <u>CE/LOAD</u> (Pin 20) low, taking this pin low activates high order byte outputs B9–B12.</p> <p><b>POL OR</b></p> <p>— With Mode (Pin 21) high. This pin serves as a high byte flag output used in handshake mode.</p>
20.	<u>CE/LOAD</u>	<p>Chip Enable Load—With Mode (Pin 21) low. <u>CE/LOAD</u> serves as a master output enable.</p> <p>When high, B1–B12, POL OR outputs are disabled.</p> <p>— With Mode (Pin 21) high, this pin serves as a load strobe used in handshake mode.</p>
21.	Mode	<p>Input Low—Direct output mode where <u>CE/LOAD</u> (Pin 20), <u>HBEN</u> (Pin 19) and <u>LBEN</u> (Pin 18) act as inputs directly controlling byte outputs.</p> <p>Input Pulsed High—Causes Immediate entry into handshake mode and outputs data are available accordingly.</p>

(Contd.)



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**De-integrate phase** This is the final phase of the analog to digital conversion. The input low is internally connected to analog common, and input high is connected across the previously charged reference capacitor. The capacitor then discharges through the internal circuit of the chip. Hence integrator output returns to zero crossing with a fixed slope. This time taken by the integrator output to return to zero is proportional to the input signal.

**Digital Section** The digital section includes the clock oscillator, 12-bit binary counter, output latches, TTL compatible output drivers, polarity, overrange and their control logics and UART handshake logic as organised in Fig. 5.42.

The digital section uses a positive logic system wherein logical ‘low’ corresponds to a low voltage and logical ‘high’ corresponds to a high voltage. The actual ranges for logical ‘low’ and ‘high’ can be obtained from the data sheets.

Intersil’s ‘Component Data Catalogue’ may be referred for the detailed information about the chip and its functioning. This text is only aimed at giving some brief introduction of the chip before we proceed for its interfacing with 8086.

**Typical component value selection** For the proper working of ICL 7109, it is necessary that the component values of the practical circuit are properly chosen. All the component values are recommended in the data manual but, the three components, viz.  $R_{int}$ ,  $C_{int}$  and  $CAz$  should be selected suitably as all of them directly determine the required input voltage range and the operating clock frequency. The full scale input voltage range is double the voltage between REF IN- and REF IN+. The clock frequency should be integral multiple of the power supply frequency (50Hz) to achieve the optimum power supply frequency rejection. The formulae for component values selection are given as follows:

$$R_{int} = \frac{\text{Full scale i/p voltage}}{20 \mu\text{A}}$$

$$C_{int} = \frac{2048 * (\text{Clock period}) * 20 \mu\text{A}}{\text{Integrator o/p voltage swing}}$$

The ADC 7109 can either be driven with a clock frequency, derived from a crystal or from an RC circuit. If the clock frequency is derived from the crystal then the actual operating frequency is given by  $f$ .

$$f = \frac{\text{Crystal freq}}{58}$$

Otherwise, if the ADC is driven by a clock frequency derived from an RC circuit it is given by the formula.

$$f = \frac{0.45}{RC}$$

### 5.6.3 Interfacing with 8086

Figure 5.43 shows the interfacing of ICL 7109 with 8086 using 8255. The assembly language program reads the digital equivalent output from ADC 7109 and stores it in the register CX. Note that the ADC gives only 12-bit output, hence the most significant nibble of CX must be masked.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

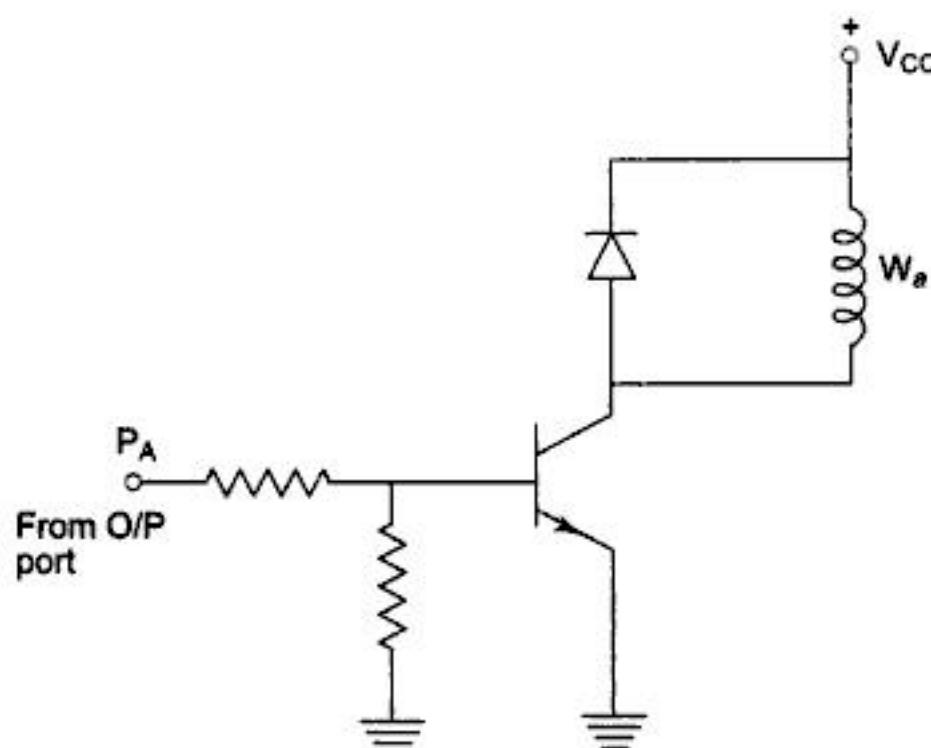


You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

have parameters like torque 3 kg-cm, operating voltage 12 V, current rating 0.2 A and a step angle 1.8°, i.e. 200 steps/revolution (number of rotor teeth).



**Fig. 5.50** Interfacing Stepper Motor Winding  $W_a$

A simple scheme for rotating the shaft of a stepper motor is called a wave scheme. In this scheme, the windings  $W_a$ ,  $W_b$ ,  $W_c$  and  $W_d$  are applied with the required voltage pulses, in a cyclic fashion. By reversing the sequence of excitation, the direction of rotation of the stepper motor shaft may be reversed. Table 5.15(a) shows the excitation sequences for clockwise and anticlockwise rotations. Another popular scheme for rotation of a stepper motor shaft applies pulses to two successive windings at a time but these are shifted only by one position at a time. This scheme for rotation of stepper motor shaft is shown in Table 5.15(b).

**Table 5.15(a)** Excitation Sequences of a Stepper Motor Using Wave Switching Scheme

Motion	Step	A	B	C	D
Clockwise	1	1	0	0	0
	2	0	1	0	0
	3	0	0	1	0
	4	0	0	0	1
	5	1	0	0	0
Anticlockwise	1	1	0	0	0
	2	0	0	0	1
	3	0	0	1	0
	4	0	1	0	0
	5	1	0	0	0



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

## Exercises

- 5.1 Interface four 8K chips of static RAM and four 4K chips of EPROM with 8086. Interface two of the RAM chips in the zeroth segment so as to accommodate IVT in them. The remaining two RAM chips are to be interfaced at the end of the fifth segment. Two EPROM chips should be interfaced so that the system is restartable as usual, and the remaining two EPROM chips should be interfaced at the starting of A000th segment. Use absolute decoding scheme.
- 5.2 Interface eight 8K chips of RAM and four 8K chips of EPROM with 8086. Interface the RAM bank at a segment address 0B00H and the EPROM bank at a physical address F8000H. Do not allow any fold back space.
- 5.3 Interface eight 8K chips of RAM and four 8K × 4 chips of EPROM with 8086 at the further specified address map. You may use linear decoding scheme for minimising the required hardware.

Chips	Segment Address	Starting Offset Address
RAM1 and RAM2	0000H	0000H
RAM3 and RAM4	0500H	5000H
RAM5 and RAM6	7000H	2000H
RAM7 and RAM8	E000H	A000H
EPROM1 and EPROM2	F000H	0000H
EPROM3 and EPROM4	D000H	7000H

Will this system be practically useful? Explain. If not then what minimum change do you suggest in this address map?

- 5.4 Interface two 8K RAM chips and two 4K EPROM chips with 8088 so as to form a completely working system configuration.
- 5.5 Describe the procedure of interfacing static memories with a CPU? Bring out the differences between interfacing the memories with 8086 and 8088.
- 5.6 Bring out the differences between static and dynamic RAM.
- 5.7 Design a 2-digit seconds counter using 74373 output ports.
- 5.8 Design a 3-digit pulse counter using 74373 output ports to count TTL compatible pulses using an input line of a 74245 input port.
- 5.9 Generate a square wave of period 1sec using 74373 output ports.
- 5.10 Design a multiplexed display scheme to display seconds, minutes and hours counter using 8255 ports. Assume that a standard delay of 1 second is available.
- 5.11 Design a one unit 14-segment alphanumeric display and write a program to display an alphanumeric character of which the code is in AX.
- 5.12 Interface an 8255 with 8086 so as to have port A address 00, port B address 02, port C address 01 and CWR address 03.
- 5.13 Explain the different modes of operation of 8255.
- 5.14 Explain the control word format of 8255 in I/O and BSR mode.
- 5.15 Write a program to print message—'This is a printer test routine.' to a printer using 8255 port initialised in mode1.
- 5.16 Interface an 8'8 keyboard using two 8255 ports and write a program to read the code of a pressed key.
- 5.17 Interface a typical 12-bit DAC with 8255 and write a program to generate a triangular waveform of period 10 ms. The CPU runs at 5 MHz clock frequency.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**Table 6.1** Selected Operations for Various Control Inputs of 8253

<b>CS</b>	<b>RD</b>	<b>WR</b>	<b>A<sub>1</sub></b>	<b>A<sub>0</sub></b>	<b>Selected Operation</b>
0	1	0	0	0	Write Counter 0
0	1	0	0	1	Write Counter 1
0	1	0	1	0	Write Counter 2
0	1	0	1	1	Write Control 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 (tristated)
0	1	1	x	x	No Operation (tristated)
1	x	x	x	x	Disabled (tristated)

A control word register accepts the 8-bit control word written by the microprocessor and stores it for controlling the complete operation of the specific counter. It may be noted that, the control word register can only be written and cannot be read as it is obvious from Table 6.1. The CLK, GATE and OUT pins are available for each of the three timer channels. Their functions will be clear when we study the different operating modes of 8253.

### 6.1.2 Control Word Register

The 8253 can operate in any one of the six different modes. A control word must be written in the respective control word register by the microprocessor to initialize each of the counters of 8253 to decide its operating mode. Each of the counter works independently depending upon the control word decided by the programmer as per the needs. In other words, all the counters can operate in any one of the modes or they may be even in different modes of operation, at a time. The control word format is presented, along with the definition of each bit, in Fig. 6.2.

While writing a count in the counter, it should be noted that, the count is written in the counter only after the data is put on the data bus and a falling edge appears at the clock pin of the peripheral thereafter. Any reading operation of the counter, before the falling edge appears may result in garbage data.

With this much information, on the general functioning of 8253, one may proceed further for the details of the operating modes of 8253. However, the concepts shall be clearer after one goes through the interfacing examples and the related assembly language programs.

### 6.1.3 Operating Modes of 8253

Each of the three counters of 8253 can be operated in one of the following six modes of operation:

1. Mode0 (Interrupt on terminal count)
2. Mode1 (Programmable monoshot)
3. Mode2 (Rate generator)
4. Mode3 (Square wave generator)
5. Mode4 (Software triggered strobe)
6. Mode5 (Hardware triggered strobe)

In this section, we will discuss all these modes of operation of 8253 in brief.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**MODE 4** This mode of operation of 8253 is named as *software triggered strobe*. After the mode is set, the output goes high. When a count is loaded, counting down starts. On terminal count, the output goes low for one clock cycle, and then it again goes high. This low pulse can be used as a strobe, while interfacing the microprocessor with other peripherals. The count is inhibited and the count value is latched, when the GATE signal goes low. If a new count is loaded in the count register while the previous counting is in progress, it is accepted from the next clock cycle. The counting then proceeds according to the new count. The related waveforms are shown in Fig. 6.7.

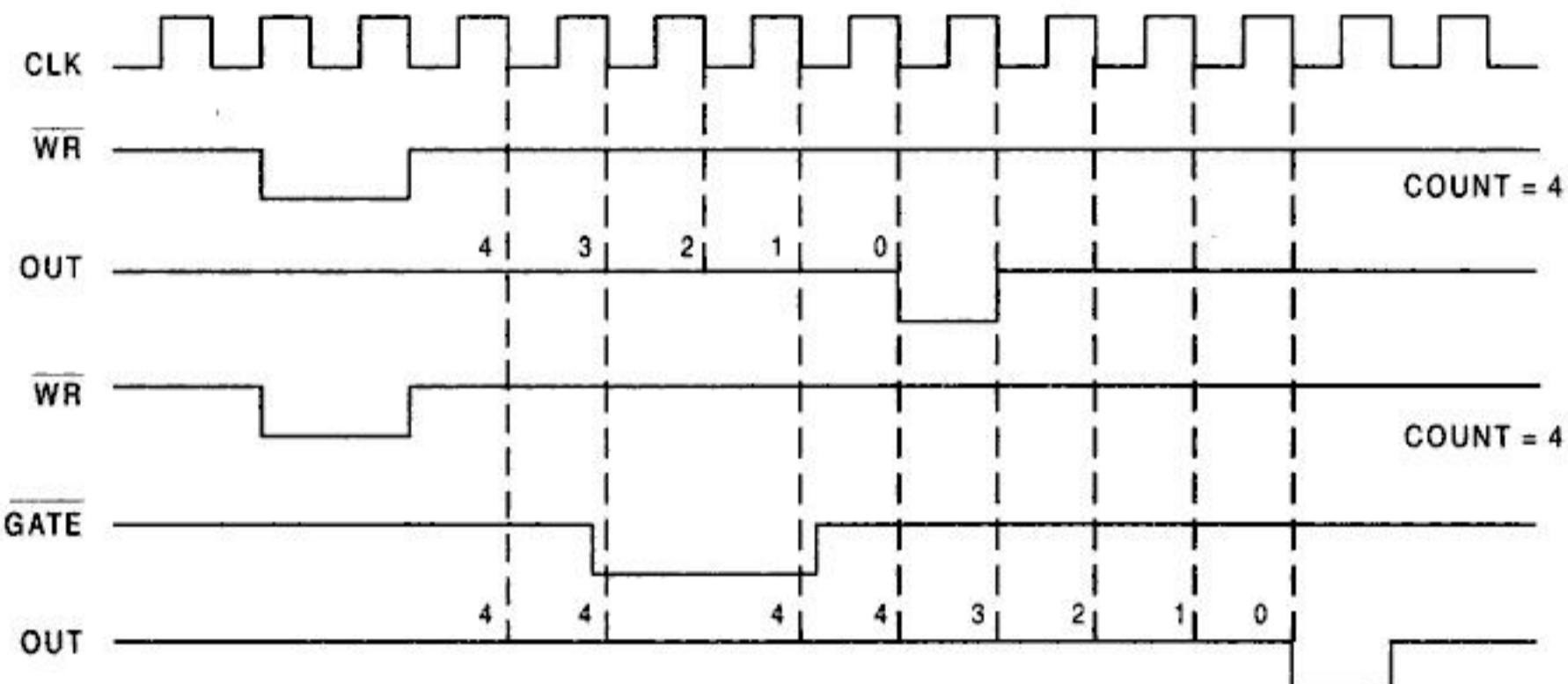


Fig. 6.7  $\overline{WR}$ , GATE and OUT Waveforms for Mode 4

**MODE 5** This mode of operation also generates a strobe in response to the rising edge at the trigger input. This mode may be used to *generate a delayed strobe* in response to an externally generated signal. Once this mode is programmed and the counter is loaded, the output goes high. The counter starts counting after the rising edge of the trigger input (GATE). The output goes low for one clock period, when the terminal count is reached. The output will not go low until the counter content becomes zero after the rising edge of any trigger. The GATE input in this mode is used as trigger input. The related waveforms are shown in Fig. 6.8.

#### 6.1.4 Programming and Interfacing 8253

As it is evident from the previous discussion, there may be two types of write operations in 8253, viz. (i) writing a control word into a control word register and (ii) writing a count value into a count register. The control word register, accepts data from the data buffer and initialises the counters, as required. The control word register contents are used for (a) initialising the operating modes (mode0-mode4) (b) selection of counters (counter0-counter2) (c) choosing binary/BCD counters (d) loading of the counter registers. The mode control register is a write only register and the CPU cannot read its contents.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

- (ii) For generating interrupt to the processor after 10 ms, the 8253 is to be used in mode 0. The OUT1 pin of 8253 is connected to interrupt input of the processor. Let us use counter 1 for this purpose, and operate the 8253 in HEX count mode.

$$\begin{aligned}\text{No. of } T \text{ states required for } 10 \text{ ms delay} &= \frac{10 \times 10^{-3}}{0.66 \times 10^{-6}} = 15 \times 10^3 \\ &= 15000 \\ &= 3A98 \text{ H}\end{aligned}$$

The Control word is written below:

SC <sub>1</sub>	SC <sub>0</sub>	RL <sub>1</sub>	RL <sub>0</sub>	M <sub>2</sub>	M <sub>1</sub>	M <sub>0</sub>	BCD	
0	1	1	1	0	0	0	0	= 70 H

The ALP is written in Program 6.2.

```

CODE      SEGMENT
ASSUME   CS : CODE
START:    MOV AL, 70 H      ; Initialize 8253 with
          OUT 46H, AL      ; Counter1 in mode 0.
          MOV AL, 98H      ; Load 98H as LSB of count
          OUT 42H, AL      ; in count reg of counter1
          MOV AL, 3AH      ; then load 3AH in MSB
          OUT 42H, AL      ; of counter1
          MOV AH, 4CH      ; Return to DOS
          INT 21H          ;
CODE      ENDS
END START

```

### Program 6.2 ALP for Problem 6.1(b)

- (iii) For generating a 5 ms quasistable state duration, the count required is calculated first. The counter 2 of 8253 is used in mode 1, to count in binary. The OUT2 signal normally remains high after the count is loaded, till the trigger is applied. After the application of a trigger signal, the output goes low in the next cycle, count down starts and whenever the count goes zero the output again goes high.

$$\begin{aligned}\text{Number of } T \text{ states required for } 05 \text{ ms} &= \frac{5 \times 10^{-3}}{0.66 \times 10^{-6}} = 7500 \text{ states} \\ &= 1 D4C \text{ H}\end{aligned}$$

The Control word is written below:

SC <sub>1</sub>	SC <sub>0</sub>	RL <sub>1</sub>	RL <sub>0</sub>	M <sub>2</sub>	M <sub>1</sub>	M <sub>0</sub>	BCD	
1	0	1	1	0	0	1	0	= B2 H

The ALP for the above purpose is written in Program 6.3.

```

CODE      SEGMENT
ASSUME   CS : CODE
START:    MOV AL, B2 H      ; Initialize 8253 with
          OUT 46H, AL      ; Counter 2 in mode 1
          MOV AL, 4CH       ; Load 4CH (LSB of count)
          OUT 44H, AL      ; into count register
          MOV AL, 1D         ; Load 1 DH (MSB of count)

```



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

In all the above programs, the counting down starts as soon as the writing operation to the count register is over, and the WR pin goes high after writing. In case of 'read on fly' operation, the READ ON FLY control word for the specific counter is to be loaded in the control word register followed by the successive read operations. The 8253 and 8254 are the most widely used peripherals to generate accurate delays for industrial or laboratory purposes. The 8254 is similar to 8253 in architecture, programming and operation, hence 8254 is not discussed in this text.

## 6.2 PROGRAMMABLE INTERRUPT CONTROLLER 8259A

The processor 8085 had five hardware interrupt pins. Out of these five interrupt pins, four pins were allotted fixed vector addresses but the pin INTR was not allotted any vector address, rather an external device was supposed to hand over the type of the interrupt, i.e. (Type 0 to 7 for RST0 to RST7), to the microprocessor. The microprocessor then gets this type and derives the interrupt vector address from that. Consider an application, where a number of I/O devices connected with a CPU desire to transfer data using interrupt driven data transfer mode. In these types of applications, more number of interrupt pins are required than available in a typical microprocessor. Moreover, in these multiple interrupt systems, the processor will have to take care of the priorities for the interrupts, simultaneously occurring at the interrupt request pins.

To overcome all these difficulties, we require a programmable interrupt controller which is able to handle a number of interrupts at a time. This controller takes care of a number of simultaneously appearing interrupt requests along with their types and priorities. This relieves the processor from all these tasks. The programmable interrupt controller 8259A from Intel is one such device. Its predecessor 8259 was designed to operate only with 8-bit processors like 8085. A modified version, 8259A was later introduced that is compatible with 8-bit as well as 16-bit processors.

### 6.2.1 Architecture and Signal Descriptions of 8259A

The architectural block diagram of 8259A is shown in Fig. 6.12. The functional explanation of each block is given in the following text in brief:

**Interrupt Request Register (IRR)** The interrupts at IRQ input lines are handled by Interrupt Request Register internally. IRR stores all the interrupt requests in it in order to serve them one by one on the priority basis.

**In-Service Register (ISR)** This stores all the interrupt requests those are being served, i.e. ISR keeps a track of the requests being served.

**Priority Resolver** This unit determines the priorities of the interrupt requests appearing simultaneously. The highest priority is selected and stored into the corresponding bit of ISR during INTA pulse. The IR<sub>0</sub> has the highest priority while the IR<sub>7</sub> has the lowest one, normally in fixed priority mode. The priorities however may be altered by programming the 8259A in rotating priority mode.

**Interrupt Mask Register (IMR)** This register stores the bits required to mask the interrupt inputs. IMR operates on IRR at the direction of the Priority Resolver.

**Interrupt Control Logic** This block manages the interrupt and the interrupt acknowledge signals to be sent to the CPU for serving one of the eight interrupt requests. This also accepts the interrupt



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

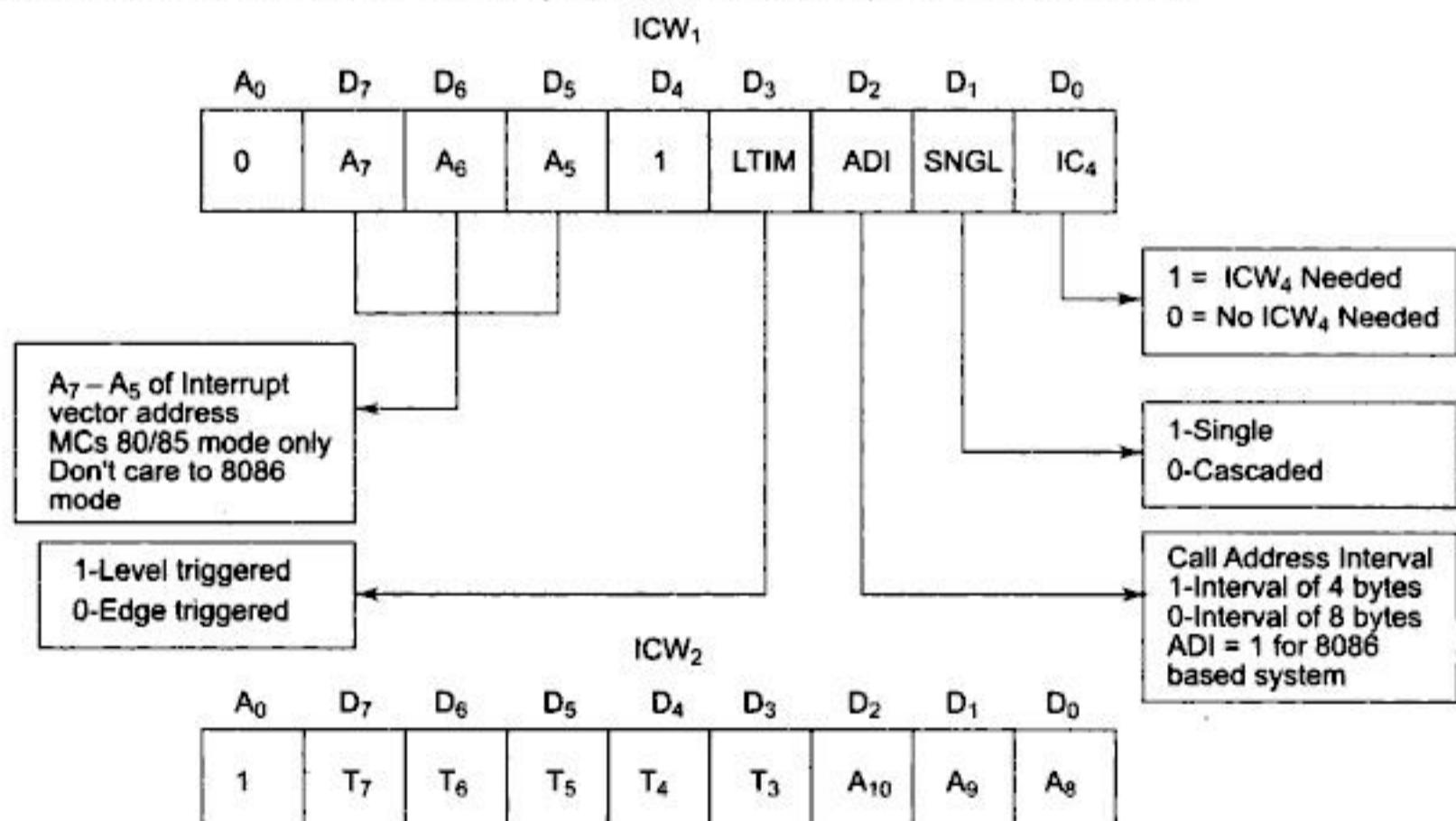
- (a) The edge sense circuit is reset, i.e. by default 8259A interrupts are edge sensitive
- (b) IMR is cleared
- (c) IR7 input is assigned the lowest priority
- (d) Slave mode address is set to 7
- (e) Special mask mode is cleared and the status read is set to IRR
- (f) If  $IC_4 = 0$ , all the functions of  $ICW_4$  are set to zero. Master/slave bit in  $ICW_4$  is used in the buffered mode only.

In an 8085 based system,  $A_{15} - A_8$  of the interrupt vector address are the respective bits of  $ICW_2$ . In 8086/88 based system, five most significant bits of the interrupt type byte are inserted in place of  $T_7 - T_3$  respectively and the remaining three bits ( $A_8$ ,  $A_9$  and  $A_{10}$ ) are inserted internally as 000 (as if they are pointing to  $IR_0$ ). Other seven interrupt levels' vector addresses are internally generated automatically by 8259 using  $IR_0$  vector. Address interval is always four in an 8086 based system.

$ICW_1$  and  $ICW_2$  are compulsory command words in initialization sequence of 8259A as is evident from Fig. 6.14, while  $ICW_3$  and  $ICW_4$  are optional. The  $ICW_3$  is read only when there are more than one 8259As in the system, i.e. cascading is used ( $SNGL = 0$ ). The  $SNGL$  bit in  $ICW_1$  indicates whether the 8259A is in the cascade mode or not. The  $ICW_3$  loads an 8-bit slave register. Its detailed functions are as follows:

In the master mode (i.e.  $\overline{SP} = 1$  or in buffer mode  $M/S = 1$  in  $ICW_4$ ), the 8-bit slave register will be set bit-wise to '1' for each slave in the system, as shown in Fig. 6.16. The requesting slave will then release the second byte of a CALL sequence.

In slave mode (i.e.  $\overline{SP} = 0$  or if  $BUF = 1$  and  $M/S = 0$  in  $ICW_4$ ) bits  $D_2$  to  $D_0$  identify the slave, i.e. 000 to 111 for slave1 to slave8. The slave compares the cascade inputs with these bits and if they are equal, the second byte of the CALL sequence is released by it on the data bus.



$T_7 - T_3$ —For 8085 system they are filled by  $A_{15} - A_{11}$  of the interrupt vector address and the least significant 3 bits are same as the respective bits of vector address. For 8086 system they are filled by most significant 5 bits of interrupt type and the least significant 3 bits are 0, pointing to  $IR_0$ .

Fig. 6.15 Initialization Command Words  $ICW_1$  and  $ICW_2$



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**Reading 8259 Status** The status of the internal registers of 8259A can be read using this mode. The  $OCW_3$  is used to read IRR and ISR while  $OCW_1$  is used to read IMR. Reading is possible only in no polled mode.

**Poll Command** In the polled mode of operation, the INT output of 8259A is neglected, though it functions normally, by not connecting INT output or by masking INT input of the microprocessor. The poll mode is entered by setting  $P = 1$  in  $OCW_3$ . The 8259A is polled by using software execution by microprocessor instead of the requests on INT input. The 8259A treats the next  $\overline{RD}$  pulse to the 8259A as an interrupt acknowledge. An appropriate ISR bit is set, if there is a request. The priority level is read and a data word is placed on to data bus, after  $\overline{RD}$  is activated. The data word is shown in Fig. 6.18.

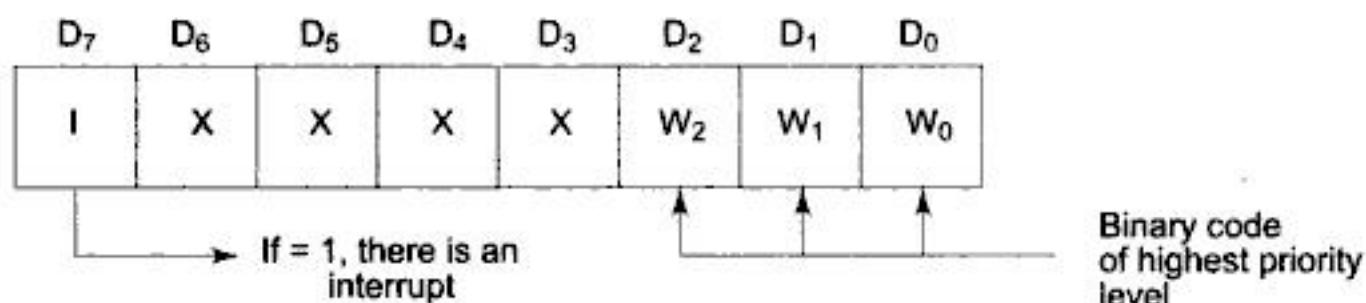


Fig. 6.19 Data Word of 8259

A poll command may give you more than 64 priority levels. Note that this has nothing to do with the 8086 interrupt structure and the interrupt priorities.

**Special Fully Nested Mode** This mode is used in more complicated systems, where cascading is used and the priority has to be programmed in the master using  $ICW_4$ . This is somewhat similar to the normal nested mode. In this mode, when an interrupt request from a certain slave is in service, this slave can further send requests to the master, if the requesting device connected to the slave has higher priority than the one being currently served. In this mode, the master interrupts the CPU only when the interrupting device has a higher or the same priority than the one currently being served. In normal mode, other requests than the one being served are masked out.

When entering the interrupt service routine the software has to check whether this is the only request from the slave. This is done by sending a non-specific EOI command to the slave and then reading its ISR and checking for zero. If its zero, a non-specific EOI can be sent to the master, otherwise no EOI should be sent. This mode is important, since in the absence of this mode, the slave would interrupt the master only once and hence the priorities of the slave inputs would have been disturbed.

**Buffered Mode** When the 8259A is used in the systems in which bus driving buffers are used on data buses (e.g. cascade systems), the problem of enabling the buffers arises. The 8259A sends a buffer enable signal on  $\overline{SP/EN}$  pin, whenever data is placed on the bus.

**Cascade Mode** The 8259A can be connected in a system containing one master and eight slaves (maximum) to handle upto 64 priority levels. The master controls the slaves using  $CAS_0-CAS_2$  which act as chip select inputs (encoded) for slaves. In this mode, the slave INT outputs are connected with master IR inputs. When a slave request line is activated and acknowledged, the master will enable the slave to release the vector address during the second pulse of  $\overline{INTA}$  sequence. The cascade lines are



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

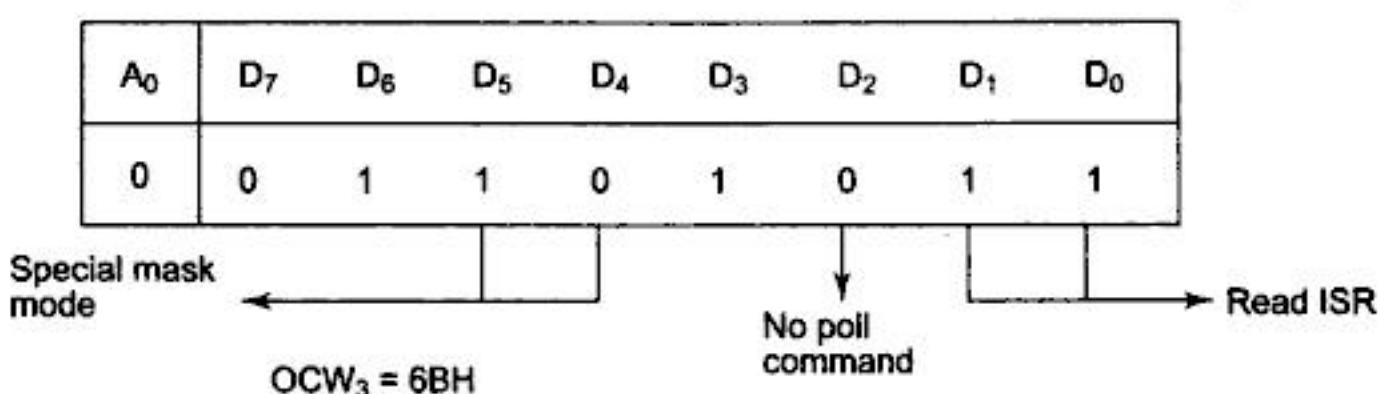


You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

For reading IRR



The following ALP writes these commands to initialize the operation of the 8259A as required in the problem. Program 6.4 gives an ALP for the required initialization of 8259A.

## CODE SEGMENT

```

ASSUME CS : CODE
START: MOV AL,1FH      ; Set the 8259A in single, level
        MOV DX,0740H
        OUT DX,AL      ; triggered mode with call address of interval of 4
        MOV DX,0742H
        MOV AL,80H      ; Select vector address 0000:0200H
        OUT DX,AL      ; for IR3 (ICW2)
        MOV AL,01H      ; ICW4 for 8086 system, normal
        OUT DX,AL      ; EOI, non-buffered, special fully nested mode masked
        MOV AL,40H      ;
        OUT DX,AL      ; OCW1 for IR6 masked,
        MOV AL,0E4H      ; Specific EOI with rotating
        MOV DX,0740H
        OUT DX,AL      ; Priority and bottom level of IR4 with OCW2
        MOV AL,6AH      ; Write OCW3 for reading
        OUT DX,AL      ; IRR and store in BH
        IN AL,DX      ;
        MOV BH,AL
        MOV AL,6BH      ; Write OCW3 to read
        OUT DX,AL      ; ISR and store in
        IN AL,DX      ; BL
        MOV BL,AL
        MOV AH,4CH      ; Return to DOS
        INT 21H
CODE ENDS
END START

```

Program 6.5 ALP to Initialize 8259A for Problem 6.2



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

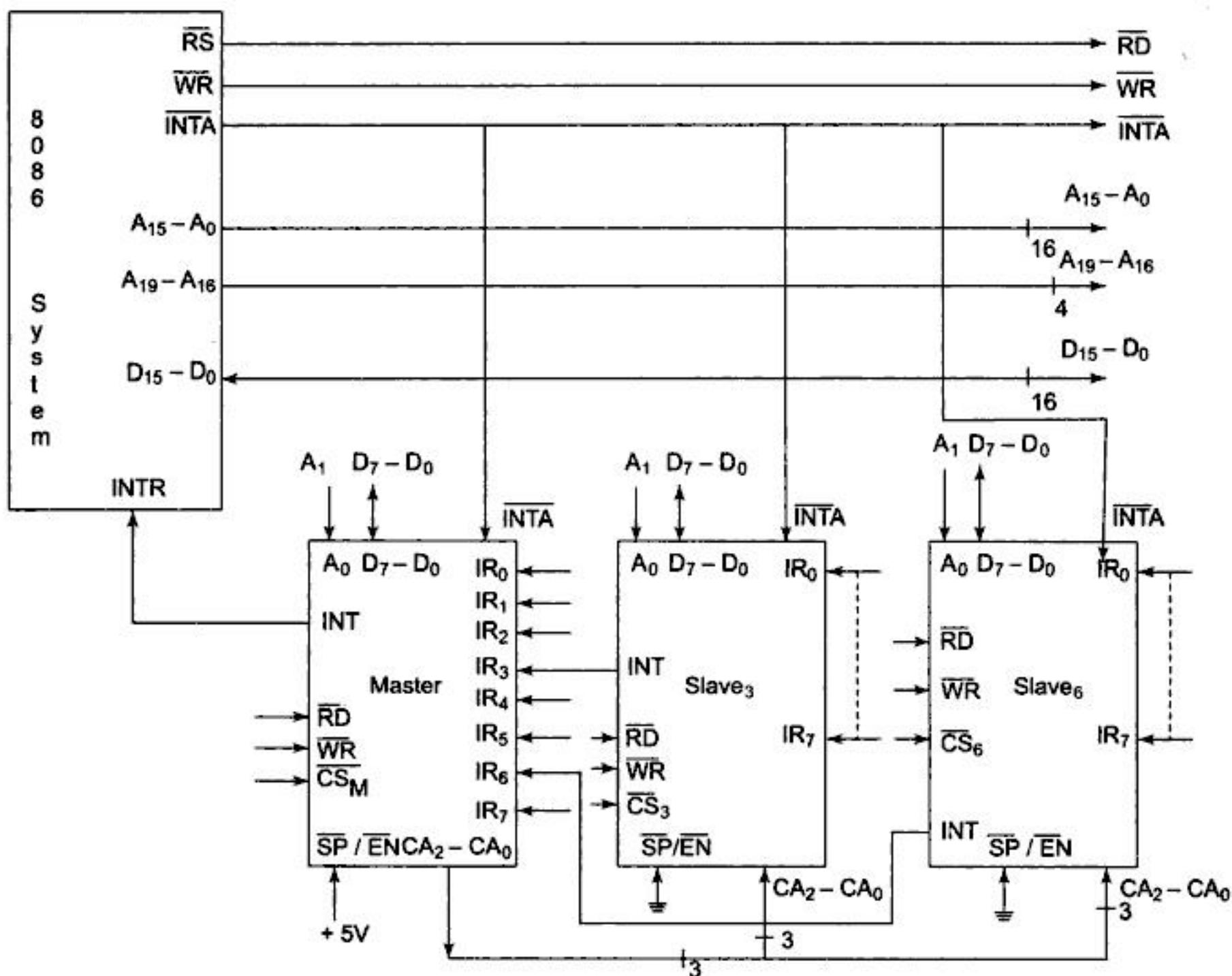
```

ADD DX, 02H          Port 1 address of Slave 6 in Dx
MOV AL, 80H          ICW2 for Slave 6
OUT DX, AL
MOV AL, 06H          ICW3 for Slave 6
OUT DX, AL
MOV AH, 01H          ICW4 for Slave 6
OUT DX, AL
MOV AH, 4CH
INT 21H
CODE ENDS
End start

```

**Program 6.6 ALP to Initialize Cascaded 8259A for Problem 6.4**

Interfacing circuit for this problem is presented below in Fig. 6.22.


**Fig. 6.22 Cascaded 8259 Interfacing for Problem 6.4**



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**CLK** This is a clock input used to generate internal timings required by 8279.

**RESET** This pin is used to reset 8279. A high on this line resets 8279. After resetting 8279, its in sixteen 8-bit display, left entry encoded scan, 2-key lock out mode. The clock prescaler is set to 31.

**CS** Chip Select: A low on this line enables 8279 for normal read or write operations. Otherwise, this pin should remain high.

**A<sub>0</sub>** A high on the A<sub>0</sub> line indicates the transfer of a command or status information. A low on this line indicates the transfer of data. This is used to select one of the internal registers of 8279.

**RD, WR** (Input/Output) READ/WRITE input pins enable the data buffers to receive or send data over the data bus.

**IRQ** This interrupt output line goes high when there is data in the FIFO sensor RAM. The interrupt line goes low with each FIFO RAM read operation. However, if the FIFO RAM further contains any key-code entry to be read by the CPU, this pin again goes high to generate an interrupt to the CPU.

**V<sub>ss</sub>, V<sub>cc</sub>** These are the ground and power supply lines for the circuit.

**SL<sub>0</sub>-SL<sub>3</sub>-Scan Lines** These lines are used to scan the keyboard matrix and display digits. These lines can be programmed as encoded or decoded, using the mode control register.

**RL<sub>0</sub>-RL<sub>7</sub>-Return Lines** These are the input lines which are connected to one terminal of keys, while the other terminal of the keys are connected to the decoded scan lines. These are normally high, but pulled low when a key is pressed.

**SHIFT** The status of the shift input line is stored along with each key code in FIFO in the scanned keyboard mode. Till it is pulled low with a key closure it is pulled up internally to keep it high.

**CNTL/STB-CONTROL/STROBED I/P Mode** In the keyboard mode, this line is used as a control input and stored in FIFO on a key closure. The line is a strobe line that enters the data into FIFO RAM, in the strobed input mode. It has an internal pull up. The line is pulled down with a key closure.

**BD-Blank Display** This output pin is used to blank the display during digit switching or by a blanking command.

**OUTA<sub>0</sub>-OUTA<sub>3</sub> and OUTB<sub>0</sub>-OUTB<sub>3</sub>** These are the output ports for two T6 × 4 (or one 16 × 8) internal display refresh registers. The data from these lines is synchronized with the scan lines to scan the display and keyboard. The two 4-bit ports may also be used as one 8-bit port.

### 6.3.2 Modes of Operation of 8279

The modes of operation of 8279 are next discussed briefly:

- (i) Input (Keyboard) modes
- (ii) Output (Display) modes

**Input (Keyboard) Modes** 8279 provides three input modes which are discussed below in brief:



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**(e) Write Display RAM**

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	A <sub>0</sub>
1	0	0	AI	A	A	A	A	1

AI — Auto Increment flag

AAAA — 4-bit address for 16-bit display RAM to be written

Other details of this command are similar to the 'Read Display RAM Command'.

**(f) Display Write Inhibit/Blanking**

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	A <sub>0</sub>
1	0	1	X	IW	IW	BL	BL	1

Output nibbles → A B A B

The IW (Inhibit Write flag) bits are used to mask the individual nibble as shown in the above command. The output lines are divided into two nibbles (OUTA<sub>0</sub>-OUTA<sub>3</sub> and OUTB<sub>0</sub>-OUTB<sub>3</sub>), those can be masked by setting the corresponding IW bit to 1. Once a nibble is masked by setting the corresponding IW bit to 1, the entry to display RAM does not affect the nibble even though it may change the unmasked nibble. The blank display bit flags (BL) are used for blanking A and B nibbles. Here D<sub>0</sub> and D<sub>2</sub> corresponds to OUTB<sub>0</sub>-OUTB<sub>3</sub> while D<sub>1</sub> and D<sub>3</sub> corresponds to OUTA<sub>0</sub>-OUTA<sub>3</sub> for blanking and masking respectively.

If the user wants to clear the display, blank (BL) bits are available for each nibble as shown in the format. Both BL bits will have to be cleared for blanking both the nibbles.

**(g) Clear Display RAM**

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	A <sub>0</sub>
1	1	0	CD <sub>2</sub>	CD <sub>1</sub>	CD <sub>0</sub>	CF	CA	1

The CD<sub>2</sub>, CD<sub>1</sub>, CD<sub>0</sub> is a selectable blanking code to clear all the rows of the display RAM as given below. The characters A and B represent the output nibbles.

CD <sub>2</sub>	CD <sub>1</sub>	CD <sub>0</sub>	
1	0	x	All zeros (x don't care) AB = 00
1	1	0	A <sub>3</sub> -A <sub>0</sub> = 2 (0010) and B <sub>3</sub> -B <sub>0</sub> = 00 (0000)
1	1	1	All ones (AB = FF), i.e. clear RAM

CD<sub>2</sub> must be 1 for enabling the clear display command. If CD<sub>2</sub> = 0, the clear display command is invoked by setting CA = 1 and maintaining CD<sub>1</sub>, CD<sub>0</sub> bits exactly same as above. If CF = 1, FIFO status is cleared and IRQ line is pulled down. Also the sensor RAM pointer is set to row 0. If CA = 1, this combines the effect of CD and CF bits. Here, CA represents Clear All and CF represents Clear FIFO RAM.

**End Interrupt/Error Mode Set**

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	A <sub>0</sub>
1	1	1	E	x	x	x	x	1

x—do not care

For the sensor matrix mode, this command lowers the IRQ line and enables further writing into the RAM. Otherwise, if a change in sensor value is detected, IRQ goes high that inhibits writing in the sensor RAM.



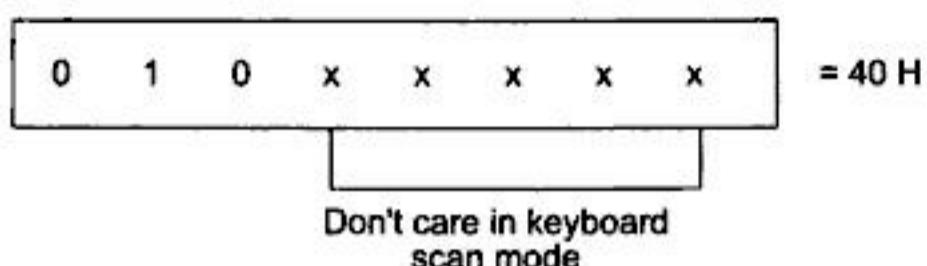
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



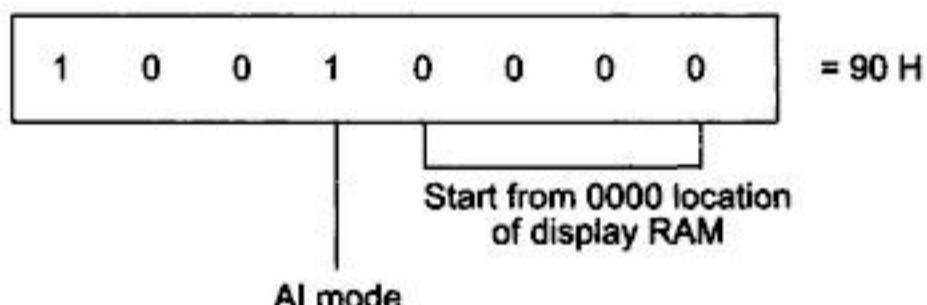
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



**Write Display RAM** This command enables the programmer to write the addressed display locations of the RAM as presented below.



Program 6.7 gives the ALP required to initialise the 8279 as required.

```

ASSUME CS : CODE
CODE SEGMENT
START: MOV AL, 1AH      ; Set 8279 in Encoded scan,
        OUT 82H, AL      ; N key rollover, 16 display, Right entry mode.
        MOV AL,34H      ; Set clock prescalar to
        OUT 82H, AL      ; 100 kHz
        MOV AL,0D3H      ; Clear display ram
        OUT 82H,AL       ; command
        MOV AL,40H      ; Read FIFO command
        OUT 82H,AL       ; for checking display RAM
WAIT:  IN AL,82H       ; Wait for clearing of
        AND AL,80H      ; Display RAM by reading
        CMP AL,80H      ; FIFO Du bit of the status word i.e.
        JNZ WAIT        ; If DU bit is not set wait, else proceed
        MOV AL,40H      ; Read FIFO command
        OUT 82H,AL       ; for checking key closure
        IN AL,82H       ; Read FIFO status
        AND AL,07H      ; Mask all bits except the
        CMP AL,00        ; number of characters bits
        JNZ KEYCODE    ; If any key is pressed, take
WRAM:  MOV AL,90H      ; required action, otherwise
        OUT 82H, AL      ; proceed to write display
        MOV AL,55H      ; RAM by using write display

```



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**RXD-Receive Data Input** This input pin of 8251A receives a composite stream of the data to be received by 8251A.

**RXRDY-Receiver Ready Output** This output indicates that the 8251A contains a character to be read by the CPU. The RXRDY signal may be used either to interrupt the CPU or may be polled by the CPU. To set the RXRDY signal in asynchronous mode, the receiver must be enabled to sense a start bit and a complete character must be assembled and then transferred to the data output register. In synchronous mode, to set the RXRDY signal, the receiver must be enabled and a character must finish assembly and then be transferred to the data output register. If the data is not successfully read 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.

**TXRDY-Transmitter Ready** This output signal indicates to the CPU that the internal circuit of the transmitter is ready to accept a new character for transmission from the CPU. The TXRDY signal is set by a leading edge of write signal if a data character is loaded into it from the CPU. In the polled operation, the TXRDY status bit will indicate the empty or full status of the transmitter data input register.

**DSR-Data Set Ready** This input may be used as a general purpose one bit inverting input port. Its status can be checked by the CPU using a status read operation. This is normally used to check if the data set is ready when communicating with a modem.

**DTR-Data Terminal Ready** This output may be used as a general purpose one bit inverting output port. This can be programmed low using the command word. This is used to indicate that the device is ready to accept data when the 8251 is communicating with a modem.

**RTS-Request to Send Data** This output also may be used as a general purpose one bit inverting output port that can be programmed low to indicate the modem that the receiver is ready to receive a data byte from the modem. This signal is used to communicate with a modem.

**CTS-Clear to Send** If the clear to send the input line is low, the 8251A is enabled to transmit the serial data, provided the enable bit in the command byte is set to '1'. If a Tx disable or CTS disable command occurs, while the 8251A is transmitting data, the transmitter transmits all the data written to the USART prior to disabling the CTS or Tx. If the CTS disable or Tx disable command occurs just before the last character appears in the serial bit string, the character will be retransmitted again whenever the CTS is enabled or the Tx enable occurs.

**TXE-Transmitter Empty** If the 8251A, while transmitting, has no characters to transmit, the TXE output goes high and it automatically goes low when a character is received from the CPU, for further transmission. In synchronous mode, a 'high' on this output line indicates that a character has not been loaded and the SYNC character or characters are about to be or are being transmitted automatically as 'fillers'. The TXE signal can be used to indicate the end of a transmission mode.

**SYNDET/BD-Synch Detect/Break Detect** This pin is used in the synchronous mode for detecting SYNC characters (SYNDET) and may be used as either input or output. This can be programmed using



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

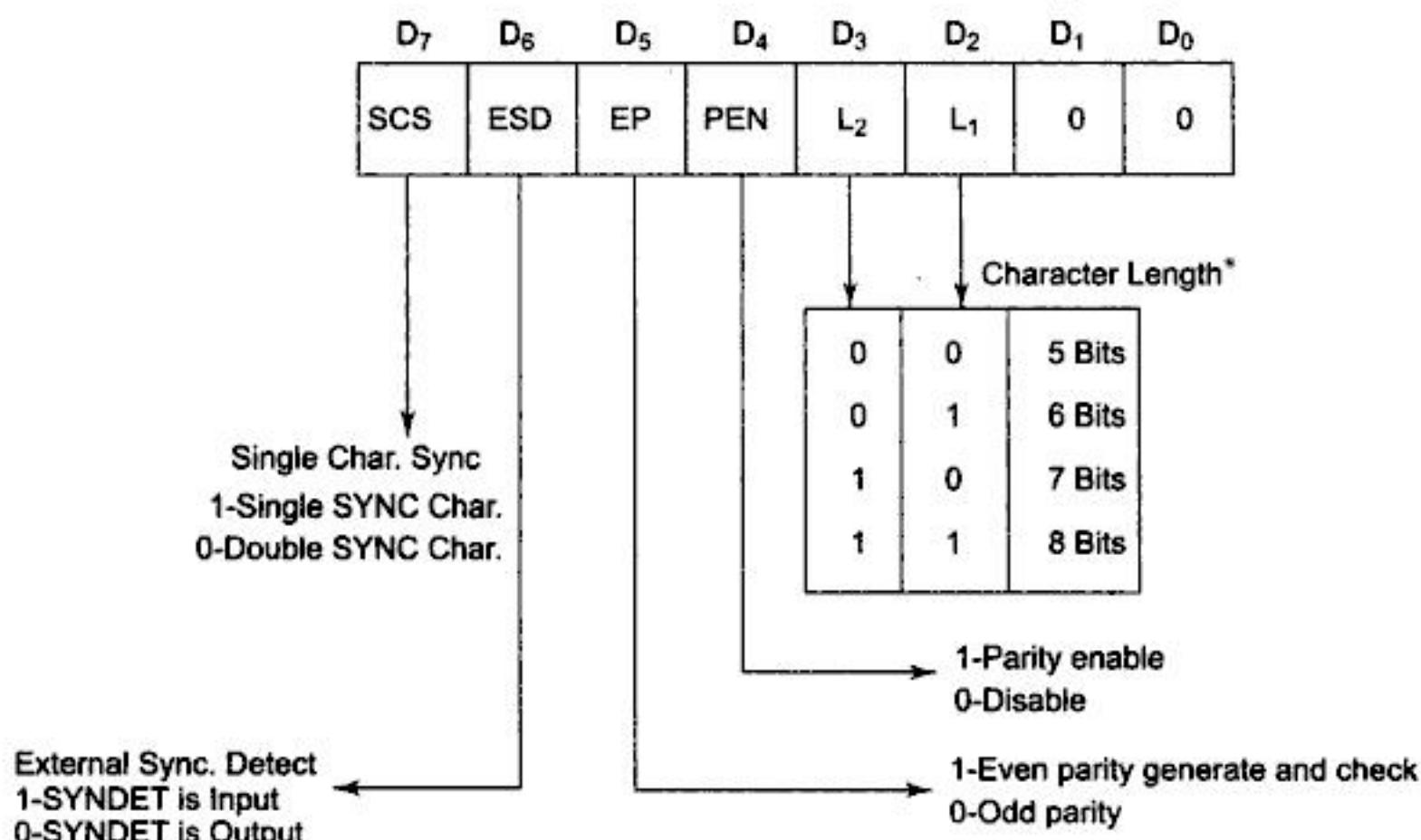


You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



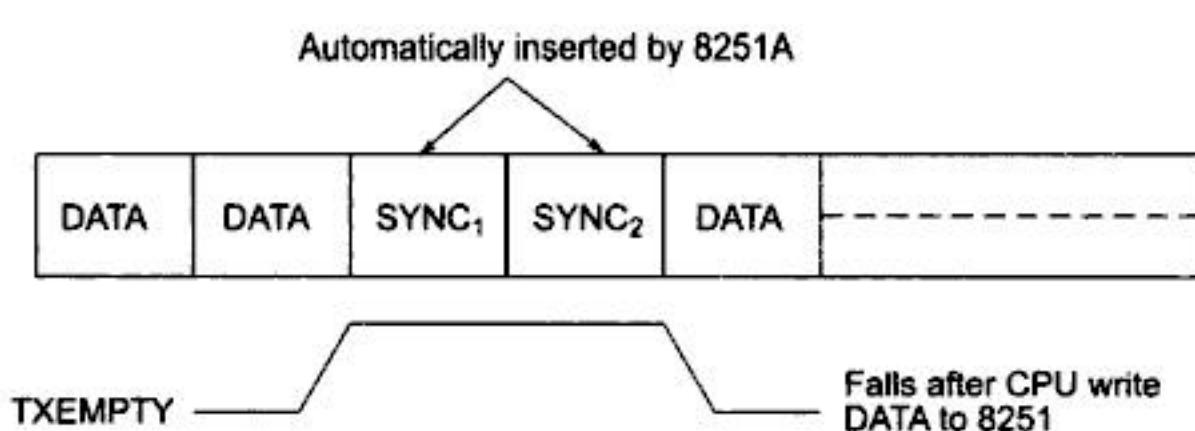
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

has been written into 8251A and the SYNC characters are inserted internally by 8251A, all further control words written with  $C/D = 1$  will load a command instruction. A reset operation returns 8251A back to mode instruction format. The command instruction format is shown in Fig. 6.33, with its bit definitions.



\* If the character size less than 8-bits, the remaining bits are set to '0'.

**Fig. 6.30 Synchronous Mode Instruction Format**



**Fig. 6.31 TXEMPTY Signal and SYNC Characters**

**Status Read Definition** This definition is used by the CPU to read the status of the active 8251A to confirm if any error condition or other conditions like the requirement of processor service has been detected, during the operation.

A read command is issued by processor with  $C/D = 1$  to accomplish this function. Some of the bits in the definition have the same significances as those of the pins of 8251A. These are used to interface the 8251A in a polled configuration, besides the interrupt controlled mode. The pin TXRDY is an exception. The status 'read format' is shown in Fig. 6.34, with its bit definitions.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

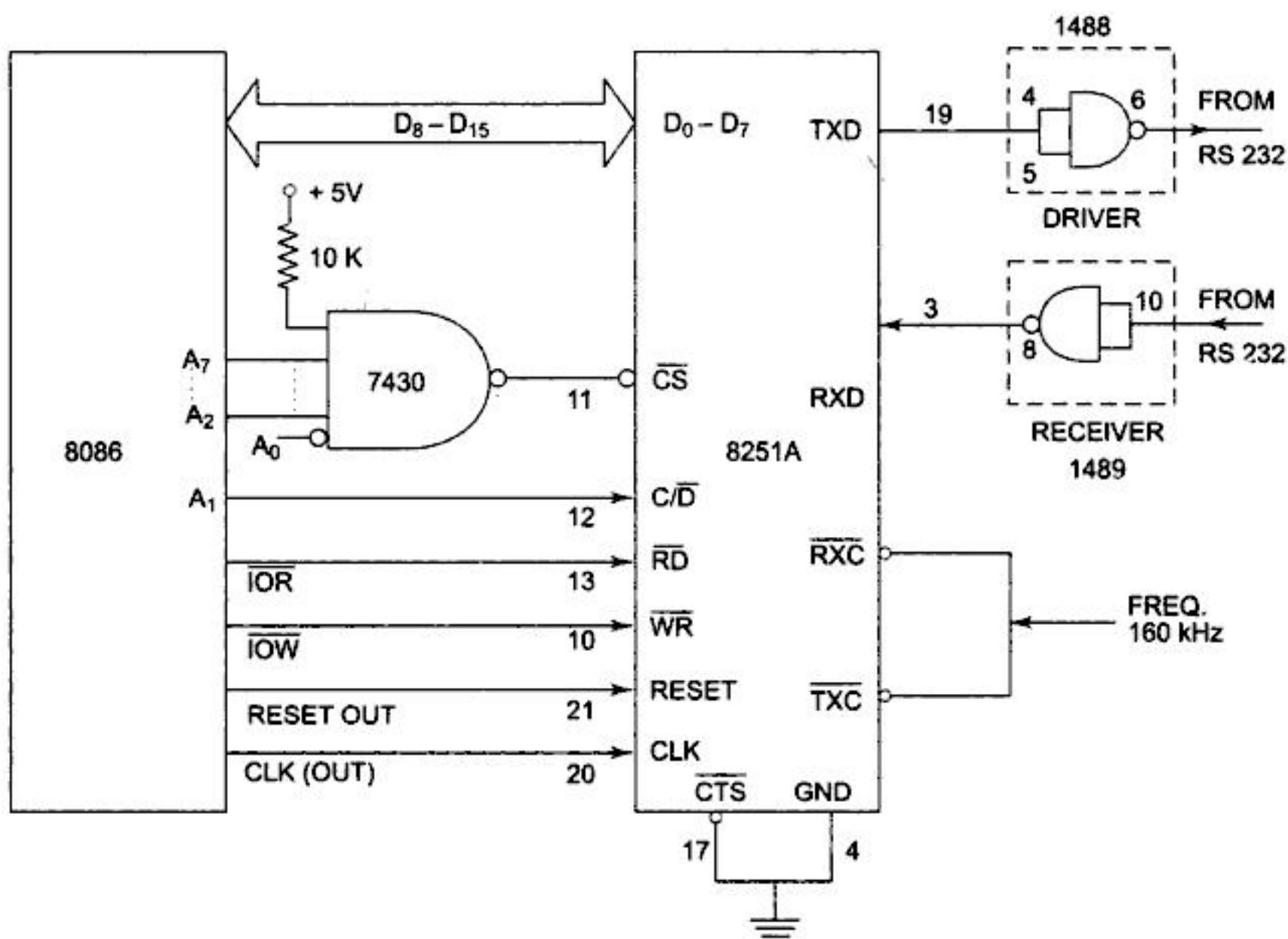
```

READY:    IN AL,0FEH      ; Check RXRDY. If the
          AND 02H        ; receiver is not ready,
          JZ READY       ; wait
          IN AL,0FCH      ; If it is ready,
          MOV [SI],AL      ; receive the character
          INC SI          ; Increment pointer to next byte
          DEC CL          ; Decrement counter
          JNZ NXTBT      ; Repeat, if CL is not zero
          MOV AH,4CH      ; If CL is 0, return to DOS
          INT 21H

CODE     ENDS

END START

```

**Program 6.9 ALP to Receive 100 Bytes****Fig. 6.35 Interfacing of 8251A with 8086**



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

# DMA, Floppy Disk and CRT Controllers

## INTRODUCTION

In the previous chapter, we studied some of the dedicated peripherals and their interfacing techniques with 8086. In this chapter, we will further discuss a few advanced peripherals and their interfacing techniques with 8086. In the applications where the CPU is to transfer bulk data, it may be a waste of time to transfer the data from source to destination using program controlled data transfer or interrupt driven data transfer. The alternate way of transferring the bulk data is the Direct Memory Access (DMA) technique in which the data is transferred under the control of a DMA controller, after it is properly initialised by the CPU. A DMA controller is designed to complete the bulk data transfer task much faster than the CPU. One such application which involves bulk data transfer is the storage of programs or data into secondary memory.

The floppy disk is one of the most popular forms of secondary memories. A Floppy Disk Controller (FDC) coordinates the complicated task of interfacing the floppy disk drive mechanism with the CPU, storing the parallel data onto the magnetic disk media in a serial bit string form and also reading the serially stored data and converting it into the parallel form. This data transfer between the CPU and the FDC may be controlled using a DMA controller.

A CRT controller derives all the control and timing signals required for controlling and interfacing a CRT with the CPU. A DMA controller may also be used to transfer data from the system memory to a video RAM of a CRT display.

In short, this chapter elaborates DMA controllers and other peripherals which may require DMA for their operation, viz. floppy disk controller and CRT controllers.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

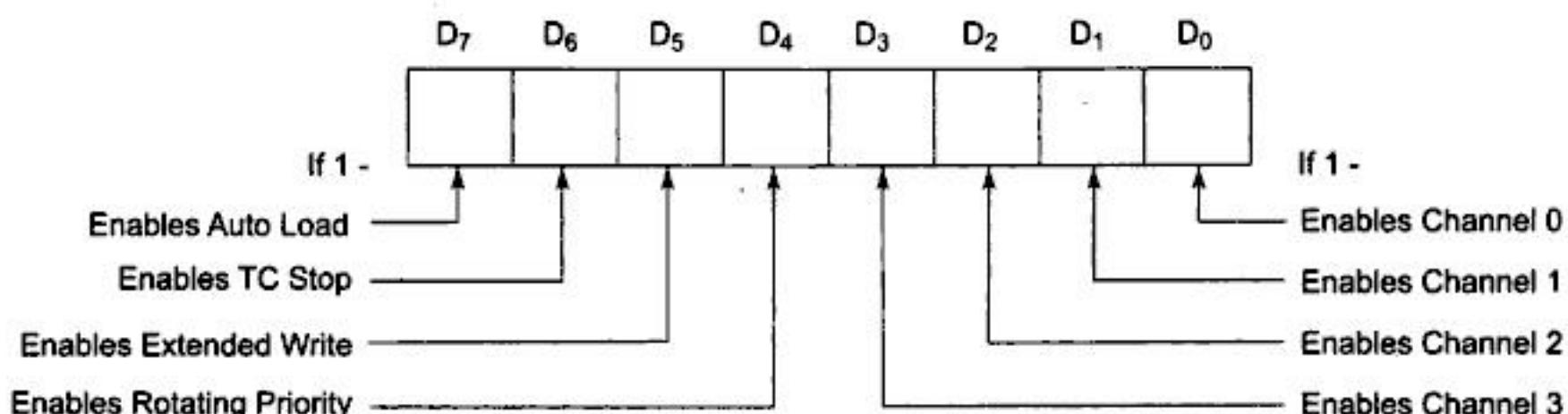


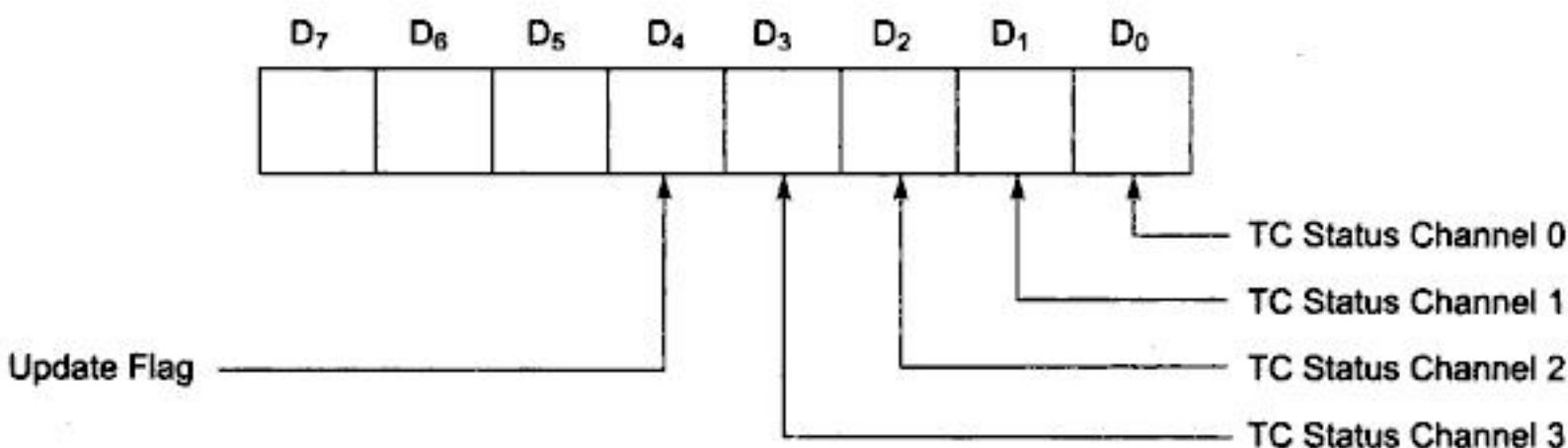
Fig. 7.2 Bit Definitions of the Mode Set Register

If the TC STOP bit is set, the selected channel is disabled after the *terminal count* condition is reached, and it further prevents any DMA cycle on the channel. To enable the channel again, this bit must be reprogrammed. If the TC STOP bit is programmed to be zero, the channel is not disabled, even after the count reaches zero and further requests are allowed on the same channel.

The auto load bit, if set, enables channel 2 for the repeat block chaining operations, without immediate software intervention between the two successive blocks. The channel 2 registers are used as usual, while the channel 3 registers are used to store the block reinitialisation parameters, i.e. the DMA starting address and the terminal count. After the first block is transferred using DMA, the channel 2 registers are reloaded with the corresponding channel 3 registers for the next block transfer, if the *Update* flag is set.

The extended write bit, if set to '1', extends the duration of **MEMW** and **IOW** signals by activating them earlier. This is useful in interfacing the peripherals with different access times. If the peripheral is not accessed within the stipulated time, it is expected to give the 'NOT READY' indication to 8257, to request it to add one or more wait states in the DMA cycle. The mode set register can only be written into.

**Status Register** The status register of 8257 is shown in Fig. 7.3. The lower order 4-bits of this register contain the terminal count status for the four individual channels. If any of these bits is set, it indicates that the specific channel has reached the terminal count condition. These bits remain set till either the status is read by the CPU or the 8257 is reset. The update flag is not affected by the read operation. This flag can only be cleared by resetting 8257 or by resetting the auto load bit of the mode set register. If the update flag is set, the contents of the channel 3 registers are reloaded to the corresponding registers of



If 1, the respective channel has reached the terminal count condition.

Fig. 7.3 Bit Definitions of Status Register of 8257



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

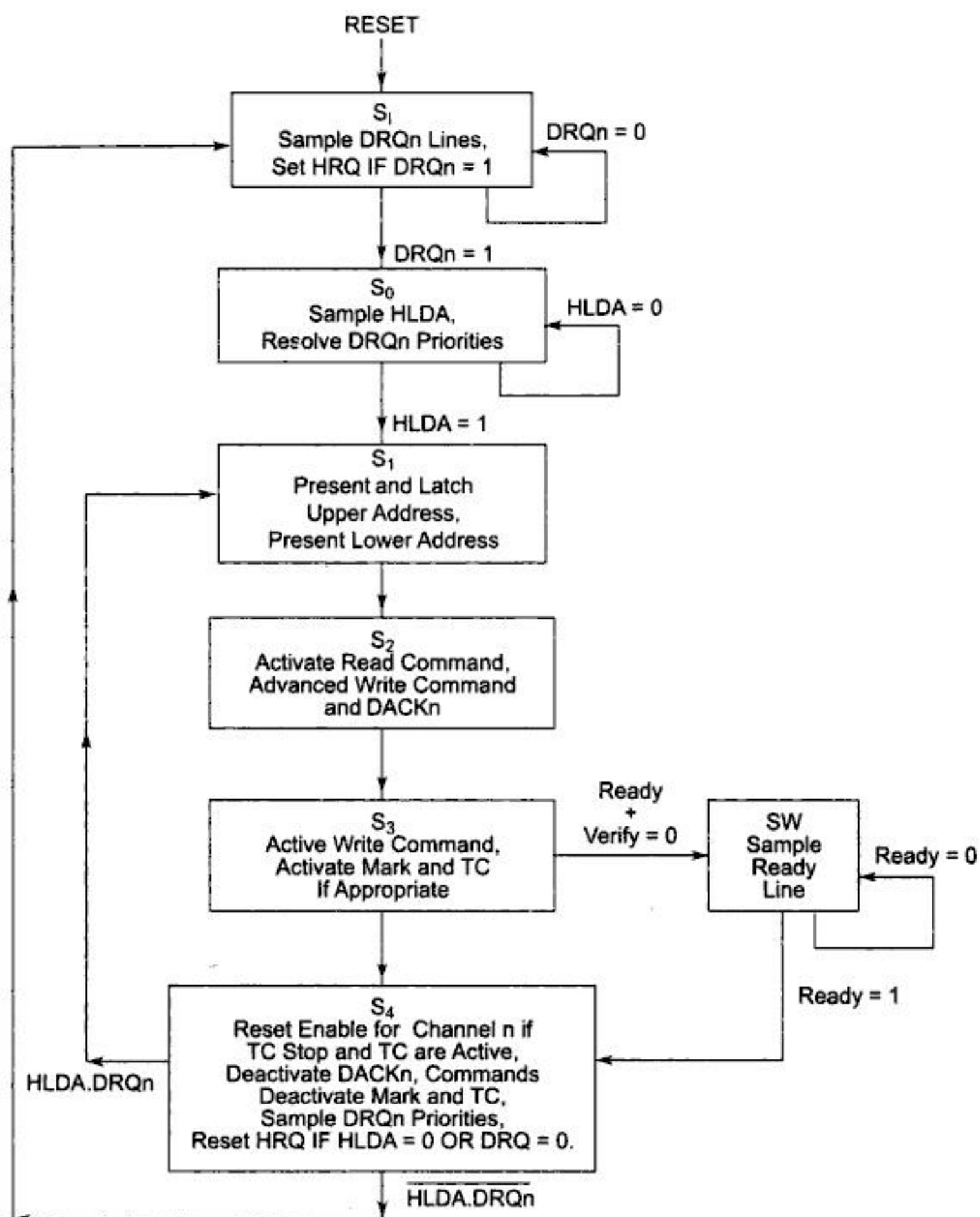


You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

8257 uses four clock cycles to complete a transfer. The 8257 has a READY input to interface it with low speed devices. The READY pin status is checked in  $S_3$  of the state diagram. If it is low, the 8257 enters a wait state. The status is sampled in every state till it goes high. Once the READY pin goes high, the 8257 continues from state  $S_4$  to complete the transfer. The 8257 can be interfaced as a memory mapped device or an I/O mapped device. If it is connected as memory mapped device, proper care must be taken while programming Rd/A<sub>15</sub> and Wr/A<sub>14</sub> bits in the terminal count register.



DRQn refers to any DRQ line of an enabled DMA channel

**Fig. 7.5 DMA Operation State Diagram**



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

the buses are in the control of the CPU, these latches are enabled, using the  $\overline{DS_1}$  signal. The data bus  $D_0-D_{15}$  is the general system data bus while the bus  $DD_0-DD_7$  is a data bus to be used by an 8-bit peripheral that works under the control of the DMA controller. The 8-bit data bus  $DD_0-DD_7$  is generated from the system data bus  $D_0-D_{15}$  using two additional data buffers which enable the 8-bit peripheral to access the even as well as odd addressed memory locations over the 8-bit data bus  $DD_0-DD_7$ . The **MEMRD** and **MEMWD** signals decide the direction of data flow under the control of the DMA controller. The  $A_0$  and  $DACK_0$  signals enable the two buffers only for the DMA controlled data transfer on channel 0. The DMA controller requires an additional latch to demultiplex the address bus  $A_8-A_{15}$  from the data bus  $D_0-D_7$ , generated by it. This latch is enabled by the **ADSTB** signal generated by the DMA controller. An additional 74245 is used to read and write the DMA controller registers under the control of the CPU. The inverted clock delays the DMA controller operation as compared to the CPU. Note that the upper data bus  $D_8-D_{15}$  is used for initialising the DMA controller in the slave mode. Also note the corresponding 16-bit instructions used to initialise the 8-bit peripheral using the upper 8-bit data bus  $D_8-D_{15}$ . Note the circuit arrangements made for accessing the even as well as odd addresses using  $D_0-D_7$ . All the initialisation command words should be derived before writing the program.

**Mode Set Register** As per the problem specification, we need the following: enable TC stop, enable channel 0, disable auto-load, Disable extended-write, disable rotating priority, disable all other channels. As already discussed, the individual bits of the mode set register are set or reset as shown below.

$D_7$	$D_6$	$D_5$	$D_4$	$D_3$	$D_2$	$D_1$	$D_0$	= 41 H
0	1	0	0	0	0	0	1	

**DMA Address Register** The DMA address register contains the starting address of the memory block which is to be accessed using DMA, i.e. 5000H.

**Terminal Count Register** As has been mentioned in Section 7.1.1, the last significant 14-bits of this register will contain the binary equivalent of the required number of the DMA cycles, i.e. number of bytes to be transferred minus one. As per the requirement, 2Kbytes of data have to be transferred from the device to memory. Therefore, the low order 14-bits of TC register will contain 7FFH. Moreover, the DMA operation in this case is going to be a memory write operation, hence  $A_{15}$  and  $A_{14}$  of this register should be 0 and 1. The TC register contents for these specifications are shown below.

$A_{15}$	$A_{14}$	$A_{13}$	$A_{12}$	$A_{11}$	$A_{10}$	$A_9$	$A_8$	$A_7$	$A_6$	$A_5$	$A_4$	$A_3$	$A_2$	$A_1$	$A_0$	= 47FFH
0	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	

The ALP for Problem 7.1 is given as follows.

```

ASSUME CS:CODE, DS:DATA
CODE SEGMENT
START: MOV AX, DATA ; Initialize Data Segment
       MOV DS, AX ;
       MOV AX, DMAL ; Load DMA address register with
       OUT 80H, AX ; lower byte of DMA address
       MOV AX, DMAH ; Load higher byte of DMA address
       OUT 80H, AX ; register of Channel 0
       MOV AX, TCL ; Load lower byte TC register of
       OUT 81H, AX ; channel 0
       MOV AX, TCH ; Load higher byte of TC register
       OUT 81H, AX ;

```



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**Request Register** Each channel has a request bit associated with it, in the request register. These are nonmaskable and subject to prioritization by the priority resolving network of 8237. Each bit is set or reset under program control or is cleared upon generation of a TC or an external EOP. This register is cleared by a reset. The bit definitions of the request register are shown in Fig. 7.11.

**Mask Register** Sometimes it may be required to disable a DMA request of a certain channel. Each of the four channels has a mask bit which can be set under program control to disable the incoming DREQ requests at the specific channel. This bit is set when the corresponding channel produces an EOP signal, if the channel is not programmed for auto-initialization. The register is set to FFH after a reset operation. This disables all the DMA requests till the mask register is cleared. The bit definitions of the mask register are shown in Figs 7.12(a) and (b) respectively. Interestingly, all these mask bits may be cleared using a software command to enable the devices at the respective channels to proceed further for DMA access.

**Temporary Register** The temporary register holds data during memory-to-memory data transfers. After the completion of the transfer operation, the last word transferred remains in the temporary register till it is cleared by a reset operation.

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
X	X	X	X	X			
Don't Care						0	0 Channel 0 Select
						0	1 Channel 1 Select
						1	0 Channel 2 Select
						1	1 Channel 3 Select
						→ 0 — Reset request bit	
						1 — Set request bit	

Fig. 7.11 Request Register Definition

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
Don't Care						0	0 Select Channel 0
						0	1 Select Channel 1
						1	0 Select Channel 2
						1	1 Select Channel 3
						→ 0 - Clear Mask bit	
						1 - Set Mask bit	

Fig. 7.12(a) Mask Register Definition to Program the Mask Bits Individually

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
Don't Care							
						→ 0 - Clear Channel 3 Mask bit	1 - Set Channel 0 Mask bit
						0 - Clear Channel 2 Mask bit	1 - Set Channel 1 Mask bit

Fig. 7.12(b) Mask Register Definition to Program all the Mask Bits Simultaneously



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

such transfer. The Terminal Count (TC) state is reached, when the count becomes zero. For each transfer, the DREQ must be active until the DACK is activated, in order to get recognized. After TC, the bus will be relinquished for the CPU. For a new DREQ to 8237, it will again activate the HRQ signal to the CPU and the HLDA signal from the CPU will push the 8237 again into the single transfer mode. This mode is also called as 'cycle stealing'.

**Block Transfer Mode** In this mode, the 8237 is activated by DREQ to continue the transfer until a TC is reached, i.e. a block of data is transferred. The transfer cycle may be terminated due to EOP (either internal or external) which forces Terminal Count (TC). The DREQ needs to be activated only till the DACK signal is activated by the DMA controller. Auto-initialization may be programmed in this mode.

**Demand Transfer Mode** In this mode, the device continues transfers until a TC is reached or an external EOP is detected or the DREQ signal goes inactive. Thus a transfer may exhaust the capacity of data transfer of an I/O device. After the I/O device is able to catch up, the service may be re-established activating the DREQ signal again. Only the EOP generated by TC or external EOP can cause the auto-initialization, and only if it is programmed for.

**Cascade Mode** In this mode, more than one 8237 can be connected together to provide more than four DMA channels. The HRQ and HLDA signals from additional 8237s are connected with DREQ and DACK pins of a channel of the host 8237 respectively. The priorities of the DMA requests may be preserved at each level. The first device is only used for prioritizing the additional devices (slave 8237s), and it does not generate any address or control signal of its own. The host 8237 responds to DREQ generated by slaves and generates the DACK and the HRQ signals to coordinate all the slaves. All other outputs of the host 8237 are disabled.

**Memory to Memory Transfer** To perform the transfer of a block of data from one set of memory address to another one, this transfer mode is used. Programming the corresponding mode bit in the command word, sets the channel 0 and 1 to operate as source and destination channels, respectively. The transfer is initialized by setting the  $DREQ_0$  using software commands. The 8237 sends HRQ (Hold Request) signal to the CPU as usual and when the HLDA signal is activated by the CPU, the device starts operating in block transfer mode to read the data from memory. The channel 0 current address register acts as a source pointer. The byte read from the memory is stored in an internal temporary register of 8237. The channel 1 current address register acts as a destination pointer to write the data from the temporary register to the destination memory location. The pointers are automatically incremented or decremented, depending upon the programming. The channel 1 word count register is used as a counter and is decremented after each transfer. When it reaches zero, a TC is generated, causing EOP to terminate the service.

The 8237 also responds to external EOP signals to terminate the service. This feature may be used to scan a block of data for a byte. When a match is found the process may be terminated using the external EOP.

Under all these transfer modes, the 8237 carries out three basic transfers namely, write transfer, read transfer and verify transfer. In write transfer, the 8237 reads from an I/O device and writes to memory under the control of IOR and MEMW signals. In read transfer, the 8237 reads from memory and writes to an I/O device by activating the MEMR and IOW signals. In verify transfers, the 8237 works in the same way as the read or write transfer but does not generate any control signal.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**Base and Current Word Count Register**

Channel 0	F1H
Channel 1	F3H
Channel 2	F5H
Channel 3	F7H

**Problem 7.3**

With the above hardware system of Problem 7.2, initialize the 8237 for memory-to-memory DMA transfer mode using channel 0, masking all other channels. Initialize the 8237 for normal timings, fixed priority, extended write with DREQ active high and DACK active-high. The 8237 should work in auto-initialization mode with address increment, block mode select with read transfer on channel 0. Further, write a program to transfer a data block of size 4KB available at 5000:0000H to 5000:1000H.

**Solution** Different programmable register contents for the initialization as per the problem specifications are given. Note that the upper data bus  $D_8-D_{15}$  drives the data lines of the DMA controller. The reader may refer to the bit patterns for each register presented in the register organisation section.

**Command Word Register**

$D_7$	$D_6$	$D_5$	$D_4$	$D_3$	$D_2$	$D_1$	$D_0$	= A1 H
1	0	1	0	0	0	0	1	

**Mode Register Word**

$D_7$	$D_6$	$D_5$	$D_4$	$D_3$	$D_2$	$D_1$	$D_0$	= 98 H
1	0	0	1	1	0	0	0	

**Request Register**

$D_7$	$D_6$	$D_5$	$D_4$	$D_3$	$D_2$	$D_1$	$D_0$	= 04 H
0	0	0	0	0	1	0	0	

**Mask Register**

$D_7$	$D_6$	$D_5$	$D_4$	$D_3$	$D_2$	$D_1$	$D_0$	= 0E H
0	0	0	0	1	1	1	0	

In memory to memory transfer, channel 0 acts as source, i.e. the channel 0 current address register contains source address and the channel 1 current address register contains destination address. The channel 1 current word count register contains the block length count. Program 7.2 gives the ALP for this problem.

```

ASSUME CS : CODE
CODE SEGMENT
START: MOV AX,0A100H      ; Out command word A1
        OUT 0F8H,AX      ; to port F8 for initialization
        MOV AX, 9800H     ; Out mode word to port FB
        OUT 0FBH,AX      ; for mode programming
        MOV AX,0E00H      ; Mask all requests
        OUT 0FFH,AX      ; except channel 0 (DREQ0).
        MOV AX, 00H       ; Clear byte pointer
        MOV 0FCH,AX       ; flip-flop
        MOV AX, 00H       ; Write base and current address
    
```



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

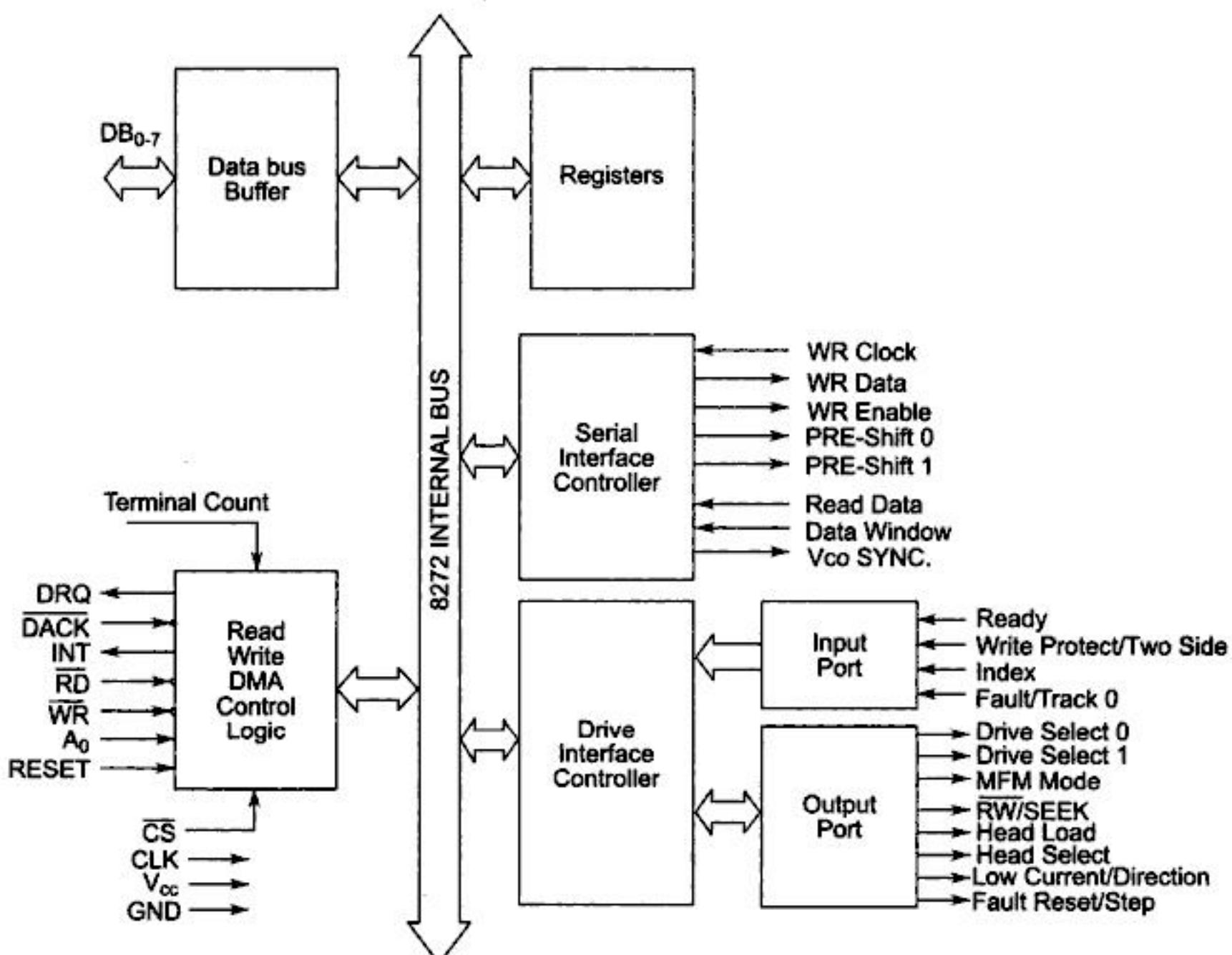


You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

has already been properly initialized. The track stepping rate, head load time and head unload time may be programmed by the user. The 8272 has features like multiple sector transfer in read as well as write modes.

#### 7.4.3 Internal Architecture of 8272

Figure 7.18 shows the internal architecture of 8272. The data bus buffer interfaces the internal bus of the controller with the system bus. The read/write and DMA control logic accepts the  $\overline{RD}$ ,  $\overline{WR}$ ,  $\overline{CS}$ ,  $A_0$ , RESET,  $\overline{DACK}$ , and TC inputs from CPU, FDD or DMA controller and generates DRQ and INT signals for the DMA controller and the CPU respectively. The data register and status register may be accessed by the CPU under program control. The data register may be read or written, but the status register can only be read. The serial interface controller gets the data from the CPU in parallel format and converts it into serial format for writing it onto the disk. Otherwise, it gets the already recorded serial data from the disk and converts it into the parallel format for reading by the CPU. Also it derives signals like  $PS_0$ - $PS_1$ , WE and VCO for FDD. This block works in synchronisation with WR clock.



**Fig. 7.18 Internal Architecture of 8272**

The *drive interface controller* interfaces the mechanism of the drive with the system via input and output ports. The input port accepts the READY, WP/TS, IDX and FLT/TRK<sub>0</sub> inputs from FDD. The



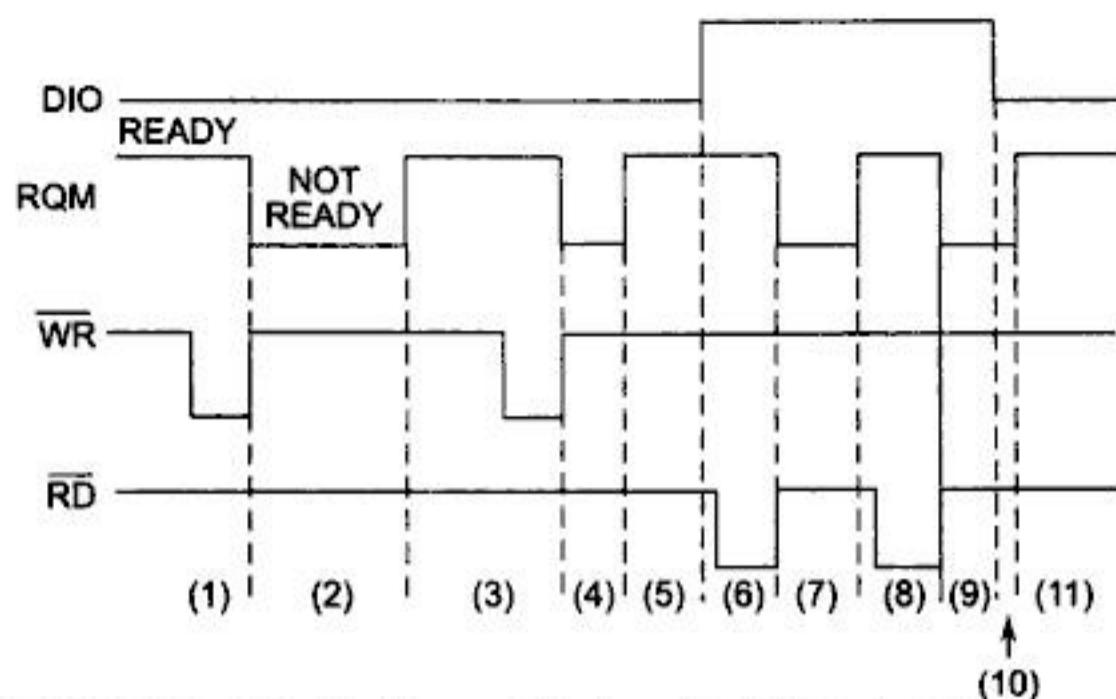
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



- (1), (3), (5), (11) - Data Register ready to be written into by the CPU.
- (2), (4), (10) - Data Register not ready to be written into by the CPU.
- (6), (8) - Data Register ready for the next byte to be read by the CPU.
- (7), (9) - Data Register not ready for the next byte to be read by the CPU.

**Fig. 7.20 Data Register Timings**

#### 7.4.6 Commands of 8272

The 8272 can execute 15 different commands. Each of which can be initiated by a multibyte transfer from the CPU to the FDC. The command may also result in a multibyte transfer from the FDC to the CPU. Each of the commands is supposed to have three phases of its execution.

**Command Phase** This is the first phase of execution of all the commands. In this phase, the FDC receives all the information including the command byte and parameters, required to execute a particular command from CPU.

**Execution Phase** This is the second phase of execution of a command and follows the first one. In this phase, The FDC performs the specific operation after decoding the command.

**Result Phase** After completion of the execution phase, the result, status and housekeeping information is made available to the processor, during this phase.

During the command or result phases the main status register must be read by the processor, before each byte of the information is written into or read from the data register. The bits  $D_6$  and  $D_7$  of the status register must be '0' and '1' respectively before each byte of the command word may be written into the 8272 data register. Hence in case of multiple byte commands, before each transfer the  $D_6$  and  $D_7$  bits must be checked for '0' and '1' state, respectively. During the result phase, the  $D_6$  and  $D_7$  both must be 1 before reading every byte from the data register. Thus the status register (main) must be read for every transfer during the command and result phase. During the execution phase, the FDC is busy with executing the command and hence the contents of the status register are invalid.

If the 8272 is in non-DMA mode, the receipt of each data byte from the FDD is indicated by 8272 to the processor using an interrupt signal. The generation of a Read Signal ( $\overline{RD} = 0$ ) resets the interrupt as well as clears the output data onto the data bus. If the processor cannot handle the interrupts fast enough (every 13  $\mu s$  for MFM mode), then it may poll the main status register. The bit  $D_7$  (RQM) of the



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**Table 7.4(b) (Contd.)**

Execution														
Result	R					ST <sub>0</sub>					Data transfer between the FDD and main system			
	R					ST <sub>1</sub>					Status information after Command execution			
	R					ST <sub>2</sub>								
	R					C								
	R					H					Sector ID information after command execution			
	R					R								
	R					N								
	READ DELETED DATA													
Command	W	MT	MFM	SK	0		I	1	0	0	Command Codes			
	W	0	0	0	0		0	HDS	DS <sub>1</sub>	DS <sub>0</sub>				
	W					C					Sector ID information prior to Command execution			
	W					H								
	W					R								
	W					N								
	W					EOT								
	W					GPL								
	W					DTL								
Execution										Data transfer between the FDD and main system				
Result	R					ST <sub>0</sub>								
	R					ST <sub>1</sub>					Status information after Command execution			
	R					ST <sub>2</sub>								
	R					C								
	R					H					Sector ID Information after command execution			
	R					R								
	R					N								
	WRITE DATA													
Command	W	MT	MFM	0	0		0	1	0	1	Command Codes			
	W	0	0	0	0		0	HDS	DS <sub>1</sub>	DS <sub>0</sub>				
	W					C					Sector ID information prior to Command			
	W					H								

(Contd.)



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**Table 7.4(b) (Contd.)**

R	ST <sub>2</sub>	execution
R	C	
R	H	Sector ID information
R	R	after Command
R	N	execution

**SCAN LOW OR EQUAL**

Command	W	MT	MFM	SK	I	1	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS <sub>1</sub>	DS <sub>0</sub>	
	W					C				Sector ID information
	W					H				prior to Command
	W					R				execution
	W					N				
	W					EOT				
	W					GPL				
	W					STP				
Execution										
Result										
	R					ST <sub>0</sub>				Status information
	R					ST <sub>1</sub>				after Command
	R					ST <sub>2</sub>				execution
	R					C				
	R					H				Sector ID information
	R					R				after Command
	R					N				execution

**SCAN HIGH OR EQUAL**

Command	W	MT	MFM	SK	I	1	1	0	0	Command Codes
	W	0	0	0	0	0	HDS	DS <sub>1</sub>	DS <sub>0</sub>	
	W					C				Sector ID information
	W					H				prior to Command
	W					R				execution
	W					N				
	W					EOT				
	W					GPL				
	W					STP				

(Contd.)



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**Table 7.6** (Contd.)

1	1A	Sectors 1 to 25 at Side 0	NC	NC	R + 1	NC
	0F	Sectors 1 to 14 at Side 0				
	08	Sectors 1 to 7 at Side 0				
	1A	Sector 26 at Side 0	NC	LSB	R = 01	NC
	0F	Sector 15 at Side 0				
	08	Sector 8 at Side 0				
	1A	Sectors 1 to 25 at Side 1	NC	NC	R + 1	NC
	0F	Sectors 1 to 14 at Side 1				
	08	Sectors 1 to 7 at Side 1				
	1A	Sector 26 at Side 1	C + 1	LSB	R = 01	NC
	0F	Sector 15 at Side 1				
	08	Sector 8 at Side 1				

**Notes:**

1. NC (No Change): The same value as the one at the beginning of command execution.
2. LSB (Least Significant Bit): The least significant bit of H is complemented.

**Write Data Command** This command requires a set of nine bytes to be written into 8272 to configure it in write data mode. After the write data command is written into 8272, the FDC loads the head, if it is unloaded, waits for the specified head settling time as defined by the specify command, and then begins reading the ID fields. If the current sector number 'R', stored in the ID register 'IDR', matches with the sector number read off the diskette, the FDC takes data from the processor byte-by-byte and outputs it to FDD.

After writing the data into the current sector, the sector number stored in 'R' is incremented by one and the next data field is written into the next sector. This operation is called as 'multi-sector write operation'. This is continued till the terminal count signal is received. If a terminal count signal is sent to FDC, it continues writing into the current sector to complete the data field. If the terminal count signal is received while the data field is being written, the remainder of the data field is filled with 00. The FDC reads the ID field of each sector and checks the CRC bytes. If the FDC detects a read error, i.e. incorrect CRC in one of the ID fields, it sets the DE flag of status register 1, and then terminates the write data command. The write command operates in the same way as the read command. In the write data mode, the data transfer between the processor and the FDC must occur every 31ms in the FM mode and every 15μs in MFM mode. If the time interval is longer than this, the FDC sets the OR (over run) flag in status register 1 and terminates the command.

**Write Deleted Data Command** This command is exactly the same as the write data command, except for the deleted data address mark, that is written at the beginning of the data field instead of the normal data address mark.

**Read Deleted Data Command** This command is exactly the same as the read data command, except when the FDC detects a data address mark at the beginning of the data field and if the SK bit is 0. If the SK bit is zero, it will read all the data in the sector, set the CM flag in the status register 2 and then terminate the command. If SK is 1, the FDC skips the sector with the data address mark and starts reading the next sector.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**Table 7.9 Cause of Interrupt as Decoded By  $D_5$ ,  $D_6$  and  $D_7$  of SR<sub>0</sub>**

<i>Seek End</i>	<i>Interrupt Code</i>			
<i>Bit 5</i>	<i>Bit 6</i>	<i>Bit 7</i>		<i>Cause</i>
0	1	1		Ready Line changed state, either polarity
1	0	0		Normal Termination of Seek or Recalibrate Command
1	1	0		Abnormal Termination of Seek or Recalibrate Command

The other sources of interrupt generation are the result phases of some commands and execution phases of the commands in non-DMA mode.

**Sense Drive status Command** This is used by the processor to get the status of all the drives by reading the status register 3.

**Specify Command** The specify command writes the initial time parameters to the three internal timers. The time from the end of the execution phase of one of the read/write commands to the head unload state is called as *Head Unload Time (HUT)*. This timer is programmable from 16 ms (i.e. BCD count 01) to 240 ms (i.e. BCD count 0F) in the steps of 16 ms. The time interval between the two successive step pulses is called as the *Step Rate Time (SRT)*. This can be programmed from 1 ms to 16 ms in steps of 1 ms. The binary count for this is in descending order, i.e. for 16 ms SRT the count is 0, for 15 ms it is 1, likewise for 3 ms it is D (1101) and for 1 ms it is F (1111). The time duration between the positive edge of the head load signal and the starting of the read/write operation is called as *Head Load Time (HLT)*. This timer is programmable from 2 to 254 ms in the steps of 2 ms, i.e. for 2 ms the count is 01, for 4 ms it is 02, for 6 ms it is 03 and likewise for 254 ms it is FEH. Note that,  $D_0$  bit in HLT parameter is not used for counter, rather the bits  $D_1$  to  $D_7$  specify the HLT count. All the above said time intervals directly depend upon the CLK input to the FDC. All the above time parameters are specified for an 8 MHz clock; all of them will be doubled, if a 4 MHz clock is used. The ND bit in HLT parameter specifies the DMA or non-DMA mode of operation.

**Invalid Command** If any of the commands, other than the above discussed ones, are written to the FDC, it will terminate the command without generating any interrupt. The DIO and RQM bits of the main status register are set to '1' to indicate to the CPU that the FDC is in result phase and it should read the status register 0. The status register 0 will contain 80 H to indicate that an invalid command was received by 8272. After every seek or recalibrate command, a sense interrupt command must be executed, otherwise, the following command is considered as invalid. This command may also be used as a no-operation command.

Tables 7.4(a) and (b) show the input parameters, and result parameters along with the command byte of the specific commands. The bit definitions of the status registers SR<sub>0</sub> to SR<sub>3</sub> are shown in Table 7.10.

#### 7.4.7 Interfacing 8272 with 8086

The general interconnections of 8272 with 8086 in the DMA mode (as it is mostly used) are presented in the block diagram in Fig. 7.21(a).



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

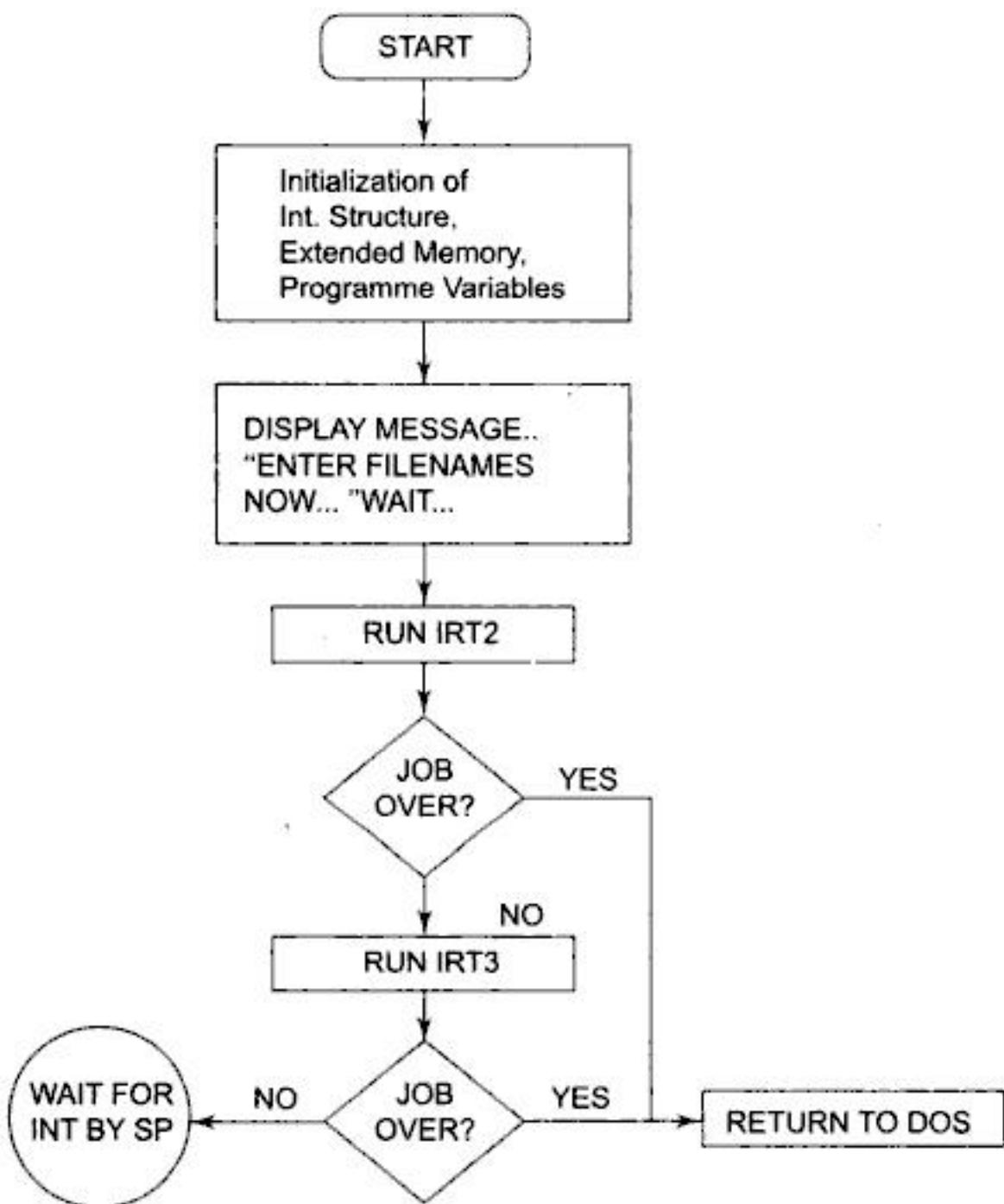


You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

them the next tasks. Accordingly, the main program consists of two interrupt service routines for the two slaves. Each of these interrupt service routines will have to perform the following functions:

1. Store the results of the previous program run by the subprocessor
2. Select the next program from the EXE filename sequence
3. Load it to the bus window of that particular subprocessor
4. Issue run message to the subprocessor

For each of these functions from (1) to (4) a separate routine is written. All of the routines are to be linked together to form a complete program. Each program is checked independently for the passed input parameters. Then the list of the required inputs to each routine is prepared and it is checked whether a particular routine prepares the expected output parameters which are to be passed as inputs to the next module. This is checked till the complete program runs successfully. The last step is now to run the complete program and search for bugs if any and to remove them. The program is then optimized by making use of the alternative better instructions and studying their effect on the required execution time and memory. Also, the program was optimized reducing the number of independent subroutines. The flow charts of the main program and the interrupt service routines are presented in Figs 8.35 and 8.36.



**Fig. 8.35 Main Program**



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

The supporting programs have been developed to load the specified program to the RAM of each card and to issue the run message and store the results of the programs back to the drive. If a continuous loop program is loaded to the memory of the card and a run message is issued, the subprocessor starts the execution of the program and the main processor becomes free. The operation of the subprocessors can be interrupted by the main processor whenever required, by resetting it.

## SUMMARY

This chapter has been devoted to the study of coprocessors and multimicroprocessor based systems. Starting with the need of multimicroprocessor systems, we have discussed different interconnection topologies and configurations. Further the software aspects of the multimicroprocessor systems are discussed briefly. A detailed account of the numeric coprocessor 8087 has been presented, starting from its architecture, pin diagram, instruction set, to the interfacing of 8087 with 8086 and programming examples. The architectural details of an I/O processor 8089 has been included to introduce the readers with the I/O processor concepts. Different bus arbitration and allotment schemes are then elaborated to highlight the bus sharing schemes used by the multimicroprocessor systems to avoid the resource contention problems. Multimicroprocessor systems based on their physical interconnections and the geographical placement of the processing nodes, viz. tightly coupled systems and loosely coupled systems have been briefly presented with circuit examples. The chapter concludes with a case study of 8088 based multimicroprocessor architecture designed by us.

## Exercises

- 8.1 Write short notes on the following multiprocessor configurations:
  - (i) Shared bus configuration
  - (ii) Multiport memory configuration
  - (iii) Linked I/O
  - (iv) Bus window
  - (v) Crossbar switching
- 8.2 Write short notes on the following interconnection topologies:
  - (i) Star interconnection
  - (ii) Loop interconnection
  - (iii) Complete interconnection
  - (iv) Regular Topology
  - (v) Irregular Topology
- 8.3 Discuss the software design of multimicroprocessor systems.
- 8.4 Draw and discuss the architecture of 8087.
- 8.5 Discuss the functions of following signals of 8087.
  - (i) BUSY
  - (ii)  $\overline{RQ}/\overline{GT}_0$
  - (iii)  $\overline{RQ}/\overline{GT}_1$
  - (iv)  $QS_0$  and  $QS_1$
  - (v) INT
- 8.6 Discuss the register organisation of 8087.
- 8.7 Discuss bit definitions of TAG word and status word of 8087.
- 8.8 Discuss bit definitions of control word register of 8087.
- 8.9 What are the different types of instructions available in the instruction set of 8087?



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

works with all of its memory management and protection capabilities with the advanced instruction set. In both the modes, 80286 is an upward object code compatible with 8086/8088.

## 9.2 INTERNAL ARCHITECTURE OF 80286

### 9.2.1 Register Organisation of 80286

The 80286 CPU contains almost the same set of registers, as in 8086, viz.

- (a) Eight 16-bit general purpose registers
- (b) Four 16-bit segment registers
- (c) Status and control register
- (d) Instruction pointer.

The register set of 80286 is shown in Fig. 9.1.

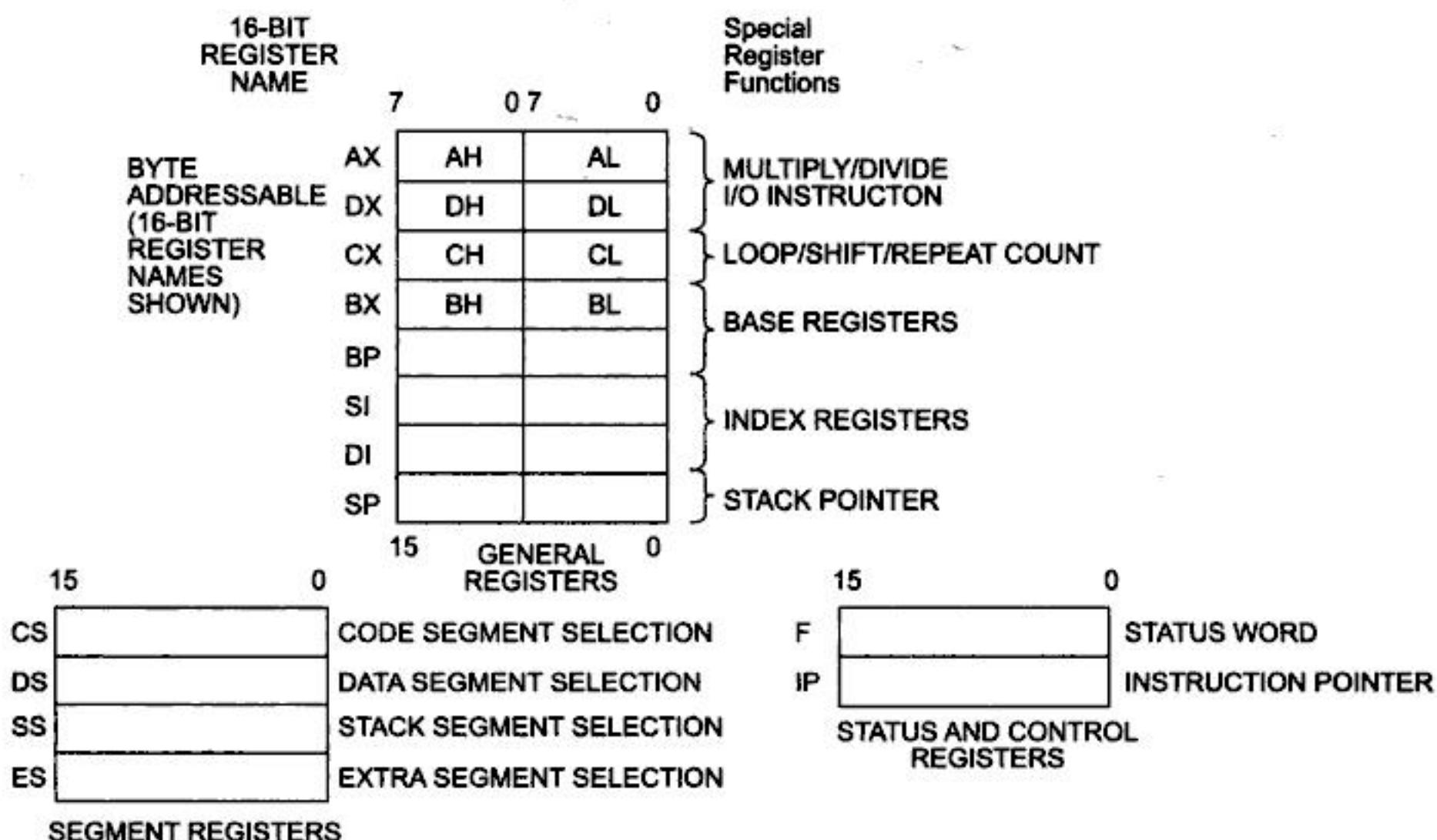


Fig. 9.1 Register Set of 80286 (Intel Corp.)

The flag register reflects the results of logical and arithmetic instructions. The flag register bits  $D_0$ ,  $D_2$ ,  $D_4$ ,  $D_6$ ,  $D_7$  and  $D_{11}$  are modified according to the result of the execution of logical and arithmetic instructions. These are called as status flag bits. The bits  $D_8$  and  $D_9$ , namely, Trap Flag (TF) and Interrupt Flag (IF) bits, are used for controlling machine operation and thus they are called control flags. All the above discussed flags are also available in 8086. Figure 9.2 shows the flag register of 80286 with the bit definitions, and the additional field definitions.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**Table 9.4 80C286 Cycle Status Definition (Intel Corp.)**

80C286 Bus Cycle Status Definition				
COD/INTA	M/I/O	S	$S_0$	Bus Cycle
0 (LOW)	0	0	0	Interrupt acknowledge
0	0	0	1	Will not occur
0	0	1	0	Will not occur
0	0	1	1	None; not a status cycle
0	1	0	0	IF $A_1 = 1$ 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 (HIGH)	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

**LOCK** This active-low output pin is used to prevent the other masters from gaining the control of the bus for the current and the following bus cycles. This pin is activated by a “LOCK” instruction prefix, or automatically by hardware during XCHG, interrupt acknowledge or descriptor table access.

**READY** This active-low input pin is used to insert wait states in a bus cycle, for interfacing low speed peripherals. This signal is neglected during HLDA cycle.

**HOLD and HLDA** This pair of pins is used by external bus masters to request for the control of the system bus (HOLD) and to check whether the main processor has granted the control (HLDA) or not, in the same way as it was in 8086.

**INTR** Through this active high input, an external device requests 80286 to suspend the current instruction execution and serve the interrupt request. Its function is like that of INTR pin of 8086.

**NMI** The Non-Maskable Interrupt request is an active-high, edge-triggered input that is equivalent to an INTR signal of type 2. No acknowledge cycles are needed to be carried out.

**PEREQ and PEACK (Processor Extension Request and Acknowledgement)** As has been mentioned earlier, processor extension refers to coprocessor (80287 in case of 80286 CPU). This pair of pins extend the memory management and protection capabilities of 80286 to the processor extension 80287. The PEREQ input requests the 80286 to perform a data operand transfer for a processor extension. The PEACK active-low output indicates to the processor extension that the requested operand is being transferred.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

# Pentium 4—Processor of the New Millennium

## INTRODUCTION

In Chapter 11, we have presented some of the salient features of the Pentium architecture. We have reviewed Pentium II and Pentium-Pro processors in moderate detail. One of the interesting features which has been introduced is the MMX instruction set, which is indeed a big step in advanced architecture. However, with passage of time designers found that this is not enough. The recent applications in three dimensional graphics, video processing, surveillance, gaming and multimedia technology demand faster performances from the processors. For example, some of the present day applications, like two and three dimensional static and moving image analysis, video surveillance, Internet audio streaming video, speech recognition and analysis etc. require speech, image and video encoding and processing in real time. The processors of 21st century should be able to perform encoding, such as JPEG or MPEG 4 in real time. Also it is important that the system should support more storage, i.e., RAM and cache (usually L1 (Level 1) cache is integrated in the chip while L2 (Level 2) cache is external to the chip). Thus at the end of the last century, there was a requirement to look for a high performance processor with novel architecture, which supported such high speed computations. This chapter provides glimpses of the features of such advanced processor like Pentium 4.

### 12.1 GENESIS OF BIRTH OF PENTIUM 4

For increasing the system performances in several such applications, duplicating or multiplying the number of processors or the number of execution units in a processor does not necessarily enhance the



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

### 12.3.3 Trace Cache (TC)

The basic function of the front end module is to fetch the instructions to be executed, decode them and feed decoded instructions to the next module, which is the out of order execution module. The instructions are first decoded into basic micro operations known as  $\mu$ ops, and the stream of decoded instructions are fed to a level-1 (L1) instruction cache. This special instruction cache is known as Trace Cache, which is a special feature of Pentium 4 microarchitecture. It is special because it does not store the instructions but the decoded stream of instructions, i.e., micro-operations or  $\mu$ ops, thus enhancing the execution speed considerably. The Trace cache can store up to 12 K  $\mu$ ops. The cache assembles the decoded  $\mu$ ops into ordered sequence of  $\mu$ ops called Traces. A single trace has many Trace lines and each Trace line has six  $\mu$ ops.

Usually the instructions are fetched and executed through Trace Cache (also known as Execution Trace Cache) only. In case there is a Trace Cache miss the microarchitecture allows the instructions to be fetched from Level 2 cache.

Two sets of next-instruction-pointers independently track the progress of the two software threads executing. We will discuss about threads later in the chapter. There are two logical processors in the CPU and when both want to access the Trace Cache every clock cycle simultaneously, then only one of them is granted the access, while the other is granted access in the alternating clock cycle. For example, if one cycle is used to fetch a line for one logical processor, the next cycle would be used to fetch a line for the other logical processor, provided that both logical processors requested access to the trace cache. If one logical processor is stalled or is unable to use the Trace Cache, the other logical processor is free to use the full bandwidth of the trace cache in every cycle.

### 12.3.4 Microcode ROM

When some complex instructions like interrupt handling or string manipulation etc. appear, the Trace Cache transfers the control to a Micro code ROM, which stores the  $\mu$ ops corresponding to these complex instructions. When the control is passed to the Microcode ROM, the corresponding  $\mu$ ops are issued. After the  $\mu$ ops are issued by the Microcode ROM, the control goes to the Trace Cache once again. The  $\mu$ ops delivered by the Trace cache and the Microcode ROM are buffered in a queue in an orderly fashion. The resultant flow of  $\mu$ ops is next fed to the execution engine.

As we have observed earlier, if both the logical processors want to execute complex IA-32 instructions simultaneously, then we need two microcode instruction pointers, which will access the microcode ROM. This is required for independent flow of control. In that case both the logical processors will be able to share the Microcode ROM entries. However, both the processors will not get the access concurrently. The access to the Microcode ROM will be alternately assigned to the two logical processors.

### 12.3.5 Front End Branch Predictor in Pentium 4

The other important unit in the front end is the Branch Prediction Logic unit.

This unit predicts the locations from where the next instruction bytes are fetched. The predictions are made based on past history of the program execution.

The earlier generation processors follow simple branching strategy. When the processor comes across a branch instruction, it evaluates the branch condition. The condition evaluation may involve a complex calculation, which may consume time and the processor has to wait till the condition is computed and thereafter it decides whether to take the branch or not. Let us look at the problem in more detail.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

As may be observed, these new set of extended instructions in SSE3 are mainly targeted towards enhancing three dimensional graphics, video and multimedia applications. Some of them will be useful for improving thread synchronization. In short they increase processor's ability to handle faster floating point computations in parallel, which are essential in applications in three dimensional graphics, multimedia and gaming. The register layouts and data types for streaming SIMD extension is given in Fig. 12.4.

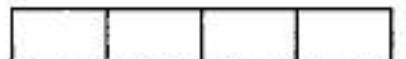
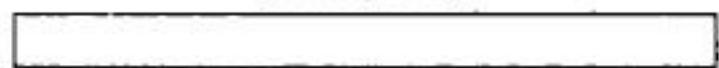
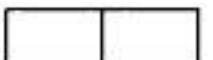
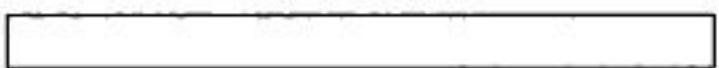
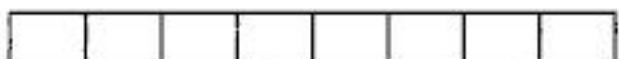
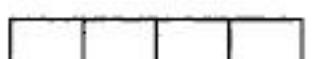
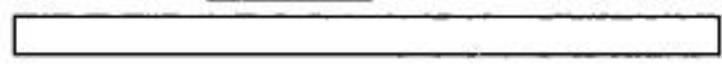
SIMD Extension	Register Layout	Data Type
	MMX Register	
MMX Technology	   	8-packed Byte Integers 4-packed word integers 2-packed double word integers Quad word
	MMX Register	
	   	8-packed byte integers 4-packed word integers 2-packed double word integers Quad word
	XMM Register	4-packed single precision floating point values
	MMX Register	2-packed double word integers
SSE2/SSE3		Quad word
	XMM Register	2-packed double-precision floating
		16-packed Byte integers
		8-packed word integers
		4-packed double word integers
		2-quad word integers
		Double quad word

Fig.12.4 Streaming SIMD Extension



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



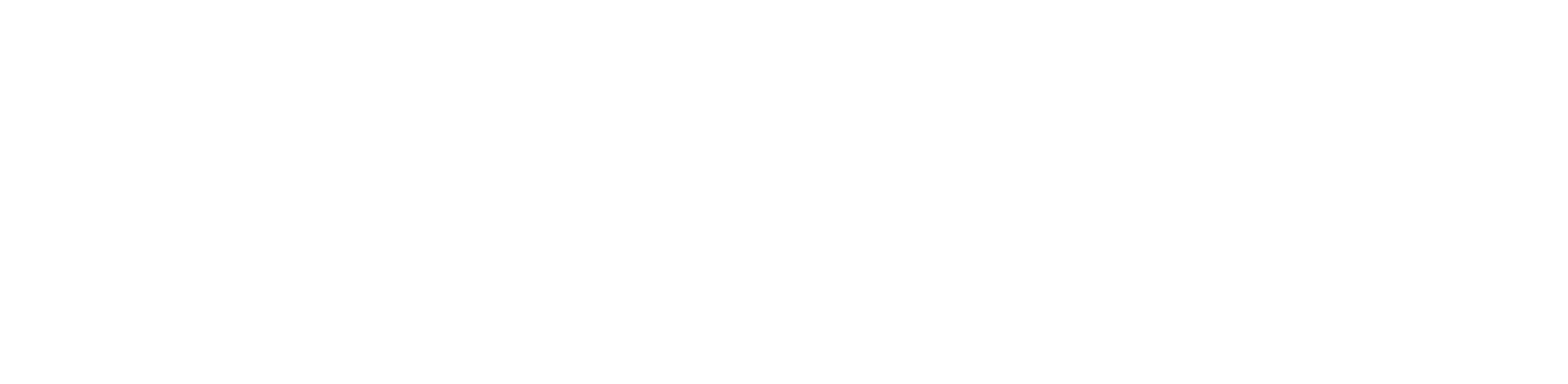
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

register by one. Thus if SP contains 07 H, the forthcoming PUSH operation will store the data at address 08H in the internal RAM. The SP content will be incremented to 08. The 8051 stack is not a top-down data structure, like other Intel processors. This register has also been allotted an address in the special function register bank.

**Data Pointer (DTPR)** This 16-bit register contains a higher byte (DPH) and the lower byte (DPL) of a 16-bit external data RAM address. It is accessed as a 16-bit register or two 8-bit registers as specified above. It has been allotted two addresses in the special function register bank, for its two bytes DPH and DPL.

**Port 0 to 3 Latches and Drivers** These four latches and driver pairs are allotted to each of the four on-chip I/O ports. These latches have been allotted addresses in the special function register bank. Using the allotted addresses, the user can communicate with these ports. These are identified as P0, P1, P2 and P3.

**Serial Data Buffer** The serial data buffer internally contains two independent registers. One of them is a transmit buffer which is necessarily a parallel-in serial-out register. The other is called receive buffer which is a serial-in parallel-out register. Loading a byte to the transmit buffer initiates serial transmission of that byte. The serial data buffer is identified as SBUF and is one of the special function registers. If a byte is written to SBUF, it initiates serial transmission and if the SBUF is read, it reads received serial data.

**Timer Registers** These two 16-bit registers can be accessed as their lower and upper bytes. For example, TL0 represents the lower byte of the timing register 0, while TH0 represents higher bytes of the timing register 0. Similarly, TL1 and TH1 represent lower and higher bytes of timing register 1. All these registers can be accessed using the four addresses allotted to them which lie in the special function registers SFR address range, i.e. 80 H to FF.

**Control Registers** The special function registers IP, IE, TMOD, TCON, SCON and PCON contain control and status information for interrupts, timers/counters and serial port. These registers will be described later in this chapter. All of these registers have been allotted addresses in the special function register bank of 8051.

**Timing and Control Unit** This unit derives all the necessary timing and control signals required for the internal operation of the circuit. It also derives control signals required for controlling the external system bus.

**Oscillator** This circuit generates the basic timing clock signal for the operation of the circuit using crystal oscillator.

**Instruction Register** This register decodes the opcode of an instruction to be executed and gives information to the timing and control unit to generate necessary signals for the execution of the instruction.

**EPROM and Program Address Register** These blocks provide an on-chip EPROM and a mechanism to internally address it. Note that EPROM is not available in all 8051 versions.

**RAM and RAM Address Register** These blocks provide internal 128 bytes of RAM and a mechanism to address it internally.

**ALU** The arithmetic and logic unit performs 8-bit arithmetic and logical operations over the operands held by the temporary registers TMP1 and TMP2. Users cannot access these temporary registers.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



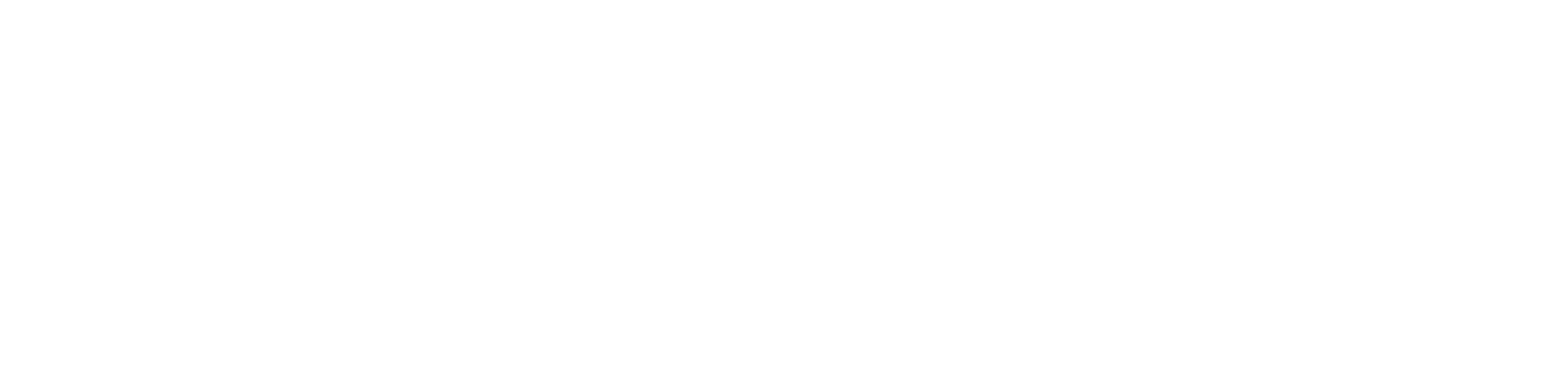
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



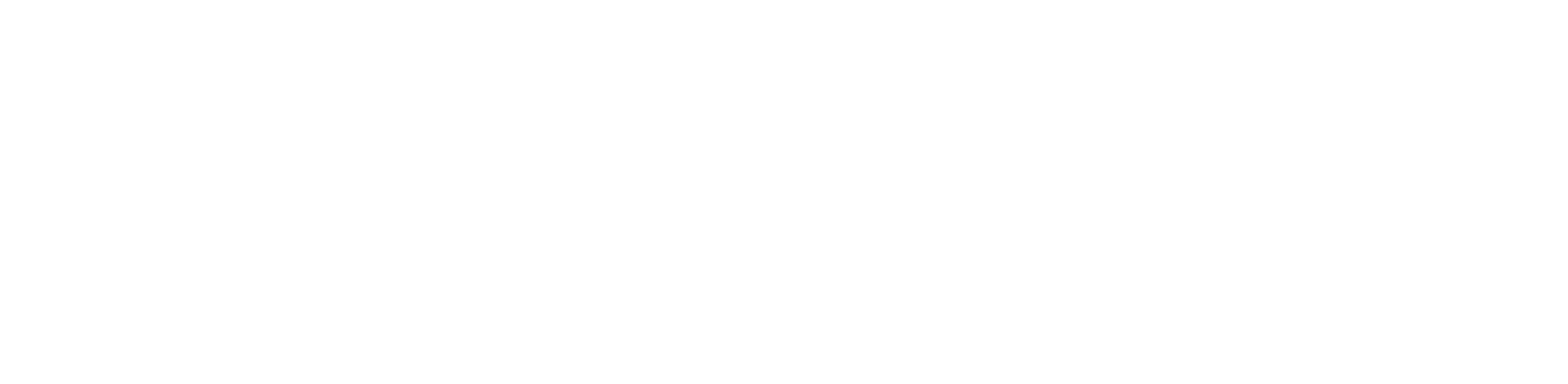
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



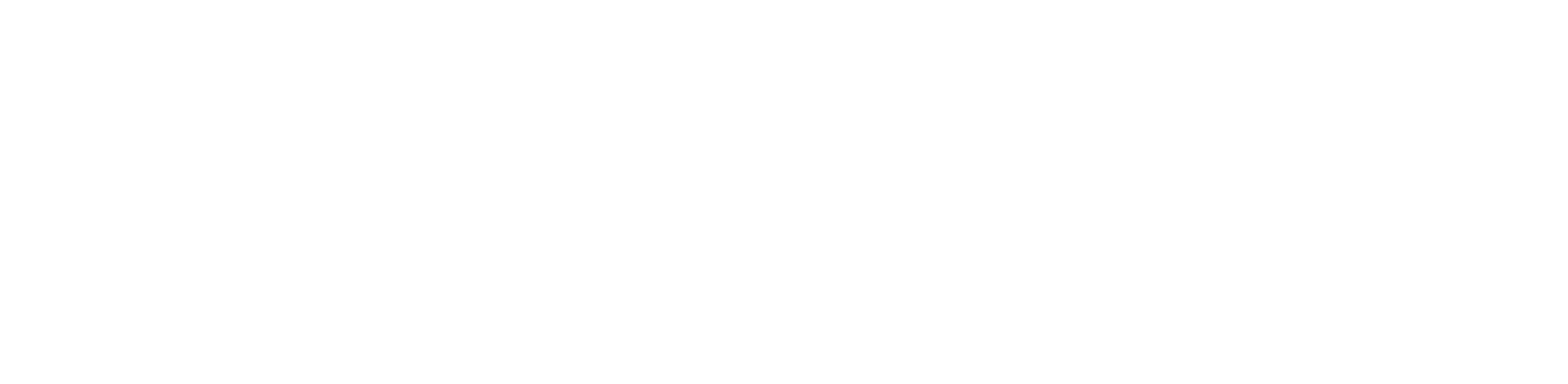
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



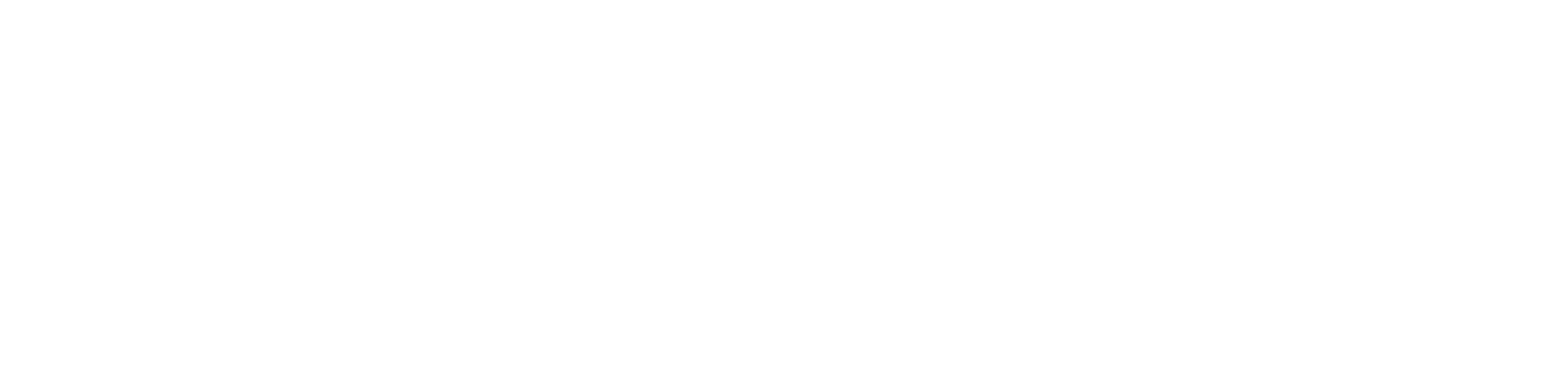
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

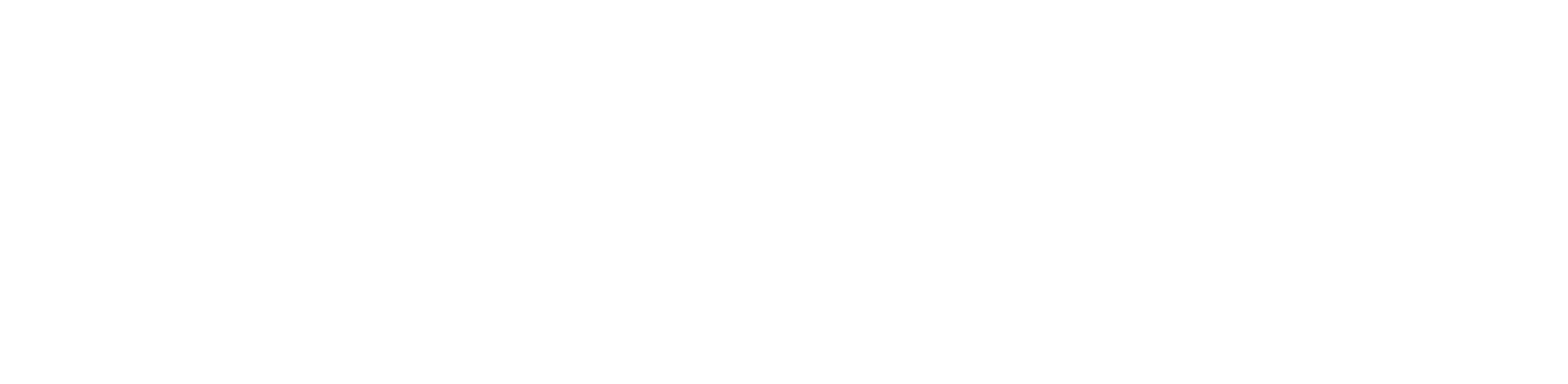


You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

- Internal block diagram 648  
 Interrupts of 8051 661  
   IE register 663  
   IP register 663  
**Memory**  
   Map 658, 659  
   On chip EPROM 658, 659  
   On chip RAM 658, 659  
   SFR addresses 658, 659  
**Microcontroller system** 667, 668  
**Mouse** 665, 666  
**Pin diagram** 652  
**Port 0-3** 653  
**Prog. status word** 655  
**PSEN** 653  
**Register set** 654  
**RXD** 653  
**Serial port control**  
   register 655, 657  
**SFR register bank** 652  
**T<sub>0</sub>** 653, 654  
**T<sub>1</sub>** 653, 654  
**Timer control register** 655, 657  
**Timer mode control register** 655, 656  
**TXD** 653  
**Waveshaping circuit** 667
- Microprocessor**
- 8086/8088**
- Architecture 3-6  
**BIU** 5  
**Bus controller** 25  
**EU** 5  
**Flag register** 3, 7, 8  
**General bus operation** 16, 17  
**General data registers** 2  
**HALT** 19  
**Index register** 3  
**Instruction byte queue** 5  
**Maximum mode** 25, 26  
   Read cycle 24  
   RQ/GT 28  
   Systems 25, 26  
   Timings 28  
   Write cycle 27  
**Memory & IO addressing** 17  
**Minimum mode** 8, 21
- Hold & acknowledge** 23  
**Read cycle** 24  
**Systems** 21  
**Timing** 24.  
**Write cycle** 24
- Physical memory organisation** 14  
**Pin diagram of 8086** 9  
**Pipelining** 13  
**Pointers** 3  
**Prefetching** 13, 35  
**Queue operations** 14  
**Register organisation** 2  
**Reset** 11, 18  
**Segment registers** 3  
**Segmentation** 6, 7  
   Advantages 6, 7  
   Non overlapped segments 6, 7  
   Overlapped segments 6, 7  
**Synchronization with external signals** 19  
**Microprocessor based weighing** 627, 628  
**Modules 1-4** 606, 607  
**Multiplexer driver** 612  
**Normal control law** 601  
**Offset** 625, 626
- 8088**
- Architecture 28, 29  
**Comparison between 8086 & 8088** 36  
**IO/M** 31  
**Maximum mode system** 28, 34  
**Minimum mode** 33  
**Pin diagram** 30  
**SS<sub>0</sub>** 31  
**System** 33, 34  
**Timing diagram** 35
- 8086 ALP**
- Addressing modes** 41  
**Assembly language** 84, 103, 104  
**Assembly language prog.** 49  
**Delays** 152  
**Direct Intersegment** 44  
**Divide by zero** 139  
**Editor**  
   Assembling a program 98, 99  
**Assembly language prog.** 104  
   Examples 104-106



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Condition code 404, 405  
 Control unit 400, 401  
 Control word register 407  
 Exception handling 406  
 Instruction set 409  
     Addressing modes 417  
     Arithmetic 411  
     Comparison 412, 413  
     Constant returning 413  
     Coprocessor control 414  
     Data transfer 410  
     Programming 417–419  
     Transcendental 412  
 Numeric extension unit 400, 401  
 Pin diagram 401  
 Register set of 8087 403  
 RQ/GT of 8087 403  
 Stack of registers 404  
 Status word 404  
 Tag word 404

### **Operating system**

Organization 609  
 Overall system 615  
 Pattern scanner 608  
 Power supply 624, 625

### **Programmable**

#### **8253**

Architecture 236  
 Control word registers 237  
 GATE 238  
 Hardware triggered strobe 241  
 Interrupt on terminal count 238  
 Modes 237–241  
     Mode 0 238  
     Mode 1 239  
     Mode 2 239  
     Mode 3 240  
     Mode 4 241  
     Mode 5 241  
 Pin diagram 236  
 Programmable monoshot 239  
 Programmable timer/Counter 235  
 Programming & interfacing 241

Rate generator 239  
 Software triggered strobe 241  
 Square wave generator 240  
 Programmed mode 613  
 Result and discussion 616, 617  
 Reticon array 616  
 Rodded cell 600  
 Salient issues in design 602  
 Shakiness of a cell 600, 603  
 Signal conditioning 625–627  
 System hardware 602, 603  
 System software 527, 528  
 Tapped cell 600

### **USART 8251**

Architecture  
     Modem control 278  
     Command instruction word 286  
     Instruction format 283  
     Interfacing & programming 287  
 Method of data common  
     Duplex 278  
     Half duplex 278  
     Simplex 278  
 Modes of operation 282  
     Asynchronous mode 282  
     control word 286  
     Mode instruction 283  
     Receiver 283  
     Transmission 283  
     Pin diagram 280  
     Status read instruction 287  
     Synchronous mode 283  
     SYNC characters 283  
     Wheatstone bridge 621

### **Pentium-4**

Salient features of Pentium-4 557  
 Architecture 558, 559  
 Instruction decoder 558  
 Trace cache 560, 562  
 Microcode ROM 560  
 Branch prediction 561  
 Branch history table 561  
 Branch target buffer 561

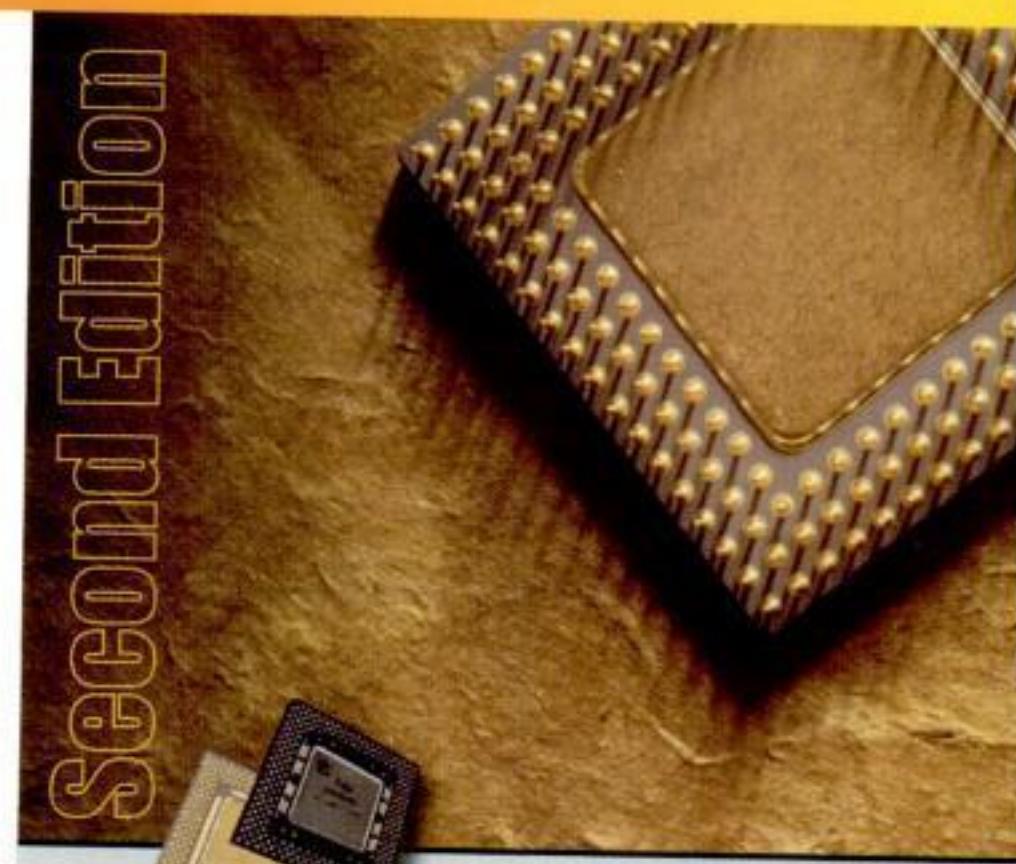


You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

This book maintains the appropriate balance between the basic concepts and practical applications related to microprocessors technology.

***Key Features***

- ❖ Advanced Microprocessors (8086/88, 80286 to 486, Pentium MMX through PIV)
- ❖ Real life Applications of Advanced Microprocessors
- ❖ Microcontrollers 8051 and 80196
- ❖ Chapter on RISC processor



**ADVANCED  
MICROPROCESSORS  
AND PERIPHERALS**

**Dedicated web site: <http://www.mhhe.com/ray/microprocessors>**

Visit us at : [www.tatamcgrawhill.com](http://www.tatamcgrawhill.com)

ISBN-13: 978-0-07-014062-2  
ISBN-10: 0-07-014062-6



9 780070 140622



**Tata McGraw-Hill**