Microprocessors and Microcontrollers

Third Edition

WBUT–2014

About the Author

Soumitra Kumar Mandal obtained a BE (Electrical Engineering) from Bengal Engineering College, Shibpur, Calcutta University, and an MTech (Electrical Engineering) with specialisation in Power Electronics from Institute of Technology, Banaras Hindu University, Varanasi. Thereafter, he obtained a PhD degree from Punjab University, Chandigarh. He started his career as a lecturer of electrical engineering at SSGM College of Engineering, Shegaon. Later he joined Punjab Engineering College, Chandigarh, as a lecturer, and served there from March 1999 to January 2004. In February 2004, he joined National Institute of Technical Teachers' Training and Research, Kolkata, and is presently Associate Professor of Electrical Engineering at this institute. Prof. Mandal is also a life member of ISTE and a member of IE. Throughout his academic career, he has published twenty research papers in national and international journals and presented many papers in national and international conferences. His research interests are in the field of computer-controlled drives, microprocessor-and-microcontroller-based system design, embedded system design and neuro-fuzzy computing.

Microprocessors and Microcontrollers

Third Edition

WBUT–2014

Soumitra Kumar Mandal
Associate Professor
National Institute of Technical Teachers' Training and Research
Kolkata

McGraw Hill Education (India) Private Limited
NEW DELHI

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

McGraw Hill Education (India) Private Limited

Published by McGraw Hill Education (India) Private Limited
P-24, Green Park Extension, New Delhi 110016

Microprocessors and Microcontrollers (WBUT), 3e

ISBN (13 digits) : 978-93-392-1425-8
ISBN (10 digits) : 93-392-1425-0

Managing Director: *Kaushik Bellani*

Head—Higher Education Publishing and Marketing: *Vibha Mahajan*
Senior Publishing Manager—SEM & Tech Ed: *Shalini Jha*
Editorial Executive: *Koyel Ghosh*
Manager—Production Systems: *Satinder S Baveja*
Assistant Manager—Editorial Services: *Sohini Mukherjee*
Assistant Manager—Production: *Anjali Razdan*

Assistant General Manager: Marketing—Higher Education: *Vijay Sarathi*
Asst. Product Manager—SEM & Tech Ed: *Tina Jajoriya*
Senior Graphic Designer (Cover): *Meenu Raghav*

General Manager—Production: *Rajender P Ghansela*
Production Manager: *Reji Kumar*

Typeset at Print-O-World, 2579, Mandir Lane, Shadipur, New Delhi 110 008, and printed at Magic International Pvt. Ltd., Plot No. 26E, Sector-31(INDUSTRIAL), Site-IV, Greater Noida - 201306

Cover Printer : Magic International Pvt. Ltd.

RAZLCRAORYXYY

In the memory of
My youngest son, the late ***Gajanan Mandal***
and to
My parents, ***Smt. Arati Mandal*** *and* ***Shri Prokash Mandal***
My wife, ***Malvika****, and*
My children, ***Om*** *and* ***Puja***

Contents

Preface

Introduction

Though the progress and advancement in microprocessor technology has been very fast, the study of the basic principles of the digital building blocks of 8085 and 8086 microprocessors and 8051 microcontrollers is still in vogue. This subject has been incorporated as part of the syllabus of undergraduate engineering courses, namely, Computer Science Engineering, Electronics and Communication Engineering, Electrical Engineering, Electrical and Electronics Engineering, and Instrumentation and Control Engineering.

Target Reader

This book is based on the latest revised syllabi of Microprocessors and Microcontrollers as taught in the West Bengal University of Technology (WBUT). Although a large number of books on microprocessors are available in the market, most of these cover either the 8085 microprocessor and its interfacing or advanced microprocessors, the 8086 to Pentium processors or the 8051 microcontroller. Consequently, there is no one book which covers all the topics starting from the 8085 microprocessor to the 8086 microprocessor and the 8051 microcontroller and PIC controller. In this book, I have attempted to fill this gap by covering all these topics in detail.

This book is written for the third-year students of CSE, ECE, EE, EEE, and ICE branches at WBUT. It will also be useful for fourth-semester students of Applied Electronics and Instrumentation Engineering. Though this course mitigates a definite percentage in every competitive examination of engineering professionals, namely, IES, UPSC, GATE, etc., it is basically written to help students acquire a strong foundation in the concepts of microprocessors and microcontrollers, their architecture, programming, and applications.

Salient Features

- Coverage and chapter organisation as per the latest WBUT syllabus
- Each topic is covered in depth from basic concepts to industrial applications of microprocessors and microcontrollers
- Hands-on practice with Hardware, ICs, Assembler, EPROM Programmers, Design of Small-Scale Embedded Systems
- Dedicated chapter on PIC Microcontroller (16F877)
- Model Question Papers at the end for self-assessment
- Latest Solved WBUT Question Papers
- Pedagogy
  - *Illustrations: 500*
  - *Multiple-Choice Questions: 230*
  - *Review Questions: 340*

Chapter Organisation

This book comprises 19 chapters. **Chapter 1** presents the introduction to microprocessors and microcontrollers. This chapter covers the basic concepts of microprocessors and microcontrollers, history of microprocessors, evolution of microprocessors and their applications. **Chapter 2** deals with the architecture of the 8085 microprocessor in a generalised way. This chapter covers the block diagram of the 8085 microprocessor and its operating principles. The pin diagram of 8085 microprocessors and function of pins are also explained elaborately. **Chapter 3** describes the different addressing modes, instruction set and instruction timing diagrams of the 8085 microprocessor.

Chapter 4 deals with machine language, assembly language, and high-level language. The operation of stack, subroutines and time-delay loops are discussed here. Modular programming, macro, instruction format and assembly-language programs are also incorporated in this chapter. In **Chapter 5**, memory ICs and their interfacing with the 8085 microprocessor are discussed elaborately. Interrupts of 8085 microprocessors and serial mode operation using SID and SOD pins of 8085 microprocessor are explained in **Chapter 6**. **Chapter 7** describes the architecture of the 8051 microcontroller. This chapter also describes the special-function registers and memory organisation, power management, timer/counters, interrupts and serial communication of the 8051 microcontroller.

Chapter 8 deals with the addressing modes and instruction set of the 8051 microcontroller. Assembly-language programs for 8051 microcontroller are discussed elaborately in this chapter. The applications of microcontrollers for keyboard interfacing, A/D converter interfacing, traffic-light control and stepper-motor control are elucidated here. The architectures of 8086 and 8088 microprocessors are introduced in **Chapter 9**. The minimum and maximum mode configurations, memory addressing, pin description of 8086 and 8088 and other supporting ICs such as 8284A, 8286/8287, 8282/8283 and 8288 are incorporated in this chapter.

Chapter 10 covers the different addressing modes and instruction set of the 8086 in detail. A brief introduction to assembly-language commands and assembly-language programs of 8086 are described in **Chapter 11**.

In **Chapters 12 to 16**, 8255A Programmable Peripheral Interface, 8253 Programmable Timer Interface, 8259 Programmable Interrupt Controller Interface, 8279 Programmable Keyboard and Display Interface, 8251 Serial Communication Interface with 8085 and 8086 microprocessors and 8051 Microcontrollers are discussed in a lucid manner.

Chapter 17 deals with the 8257 Direct Memory Access (DMA) Controller. In **Chapter 18**, ADC as well as DAC ICs and their interfacing with 8085 and 8086 microprocessors and 8051 microcontrollers, keyboard and multiplexed display, LCD and keyboard interfacing with 8051 microcontrollers are discussed in detail. **Chapter 19** introduces the PIC microcontroller (16F877).

Acknowledgements

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

I am also indebted to the following external reviewers who assessed various chapters of the book and contributed with their constructive criticism and suggestions.

Anil Kumar Sharma	*Abacus Institute of Engineering and Management, 24 Parganas, West Bengal*
Debarshi Datta	*SDET Brainware Group of Institutions, Kolkata, West Bengal*
Paulami Basu Malik	*Techno India College of Technology, Kolkata, West Bengal*

Criticism and suggestions for improvement shall be gratefully acknowledged. Readers may contact me through email at *mandal_soumitra@yahoo.com*.

Soumitra Kumar Mandal

Publisher's Note

Remember to write to us. We look forward to receiving your feedback, comments and ideas to enhance the quality of this book. You can reach us at *tmh.corefeedback@gmail.com.* Please mention the title and author's name as the subject. In case you spot piracy of this book, please do let us know.

ROADMAP TO THE SYLLABUS

This text is useful for subject codes:
CS502 and EC502—CSE, ECE
Microprocessors and Microcontrollers

Module 1:

Introduction to microcomputer-based system. History of evolution of microprocessor and microcontrollers and their advantages and disadvantages.

Architecture of 8085 microprocessor. Address/data bus demultiplexeing, status signals and the control signal generation. Instruction set of 8085 microprocessor. Classification of instruction, addressing modes, timing diagram of the instructions (a few examples).

Module 2:

Assembly language programming with examples, interrupts of 8085 processor, programming using interrupts.

Serial and parallel data transfer – programmed I/O, interrupts driven I/O, DMA, asynchronous and synchronous serial transmission using SID and SOD pins of 8085 processor.

Module 3:

Introduction to MCS-51 microcontroller – Architecture, pin details, memory organization, Hardware features of MCS-51, external memory interfacing, timers, interrupts, power management, serial port, addressing modes, assembly language programming.

The 8086 microprocessor–Architecture, pin details, addressing modes, instruction set, assembly language programming interrupts.

Support IC chips–8255, 8253, 8259, 8279 and 8251 and their interfacing with 8085, 8086 and microcontroller 8051.

CHAPTER 7	8051 MICROCONTROLLER ARCHITECTURE
CHAPTER 8	INSTRUCTION SET & PROGRAMMING OF 8051 MICROCONTROLLER
CHAPTER 9	ARCHITECTURE OF 8086 & 8088 MICROPROCESSORS
CHAPTER 10	INSTRUCTION SET OF 8086 MICROPROCESSORS
CHAPTER 11	ASSEMBLY LANGUAGE PROGRAM OF 8086 MICROPROCESSORS
CHAPTER 12	8255 INTERFACING WITH 8085, 8086 AND 8051 MICROCONTROLLER
CHAPTER 13	8253 INTERFACING WITH 8085, 8086 AND 8051 MICROCONTROLLER
CHAPTER 14	8259 INTERFACING WITH 8085, 8086 AND 8051 MICROCONTROLLER
CHAPTER 15	8279 INTERFACING WITH 8085, 8086 AND 8051 MICROCONTROLLER
CHAPTER 16	8251 INTERFACING WITH 8085, 8086 AND 8051 MICROCONTROLLER

Module 4:

Keyboard and multiplexed display, LCD interfacing, with 8085, 8086, and 8051. Memory interfacing with 8085, 8086, and 8051–ADC and DAC interfacing with the processor 8085, 8086 and 8051.

Brief introduction to PIC microcontroller (16F877).

CHAPTER 18	ADC, DAC, KEYBOARD, MULTIPLEX DISPLAY AND LCD INTERFACING WITH 8085, 8086, 8051
CHAPTER 17	DIRECT MEMORY ACCESS (DMA) CONTROLLER 8257
CHAPTER 5	MEMORY & INTERFACING WITH 8085 MICROPROCESSOR
CHAPTER 19	INTRODUCTION TO PIC MICROCONTROLLER (16F877)

This text is useful for subject codes:
EE 504C, EEE504C, ICE504C—EE, EEE, ICE
Microprocessors and Microcontrollers

UNIT 1:

Introduction to computer architecture: Architecture of a typical microprocessor, Bus configuration, The CPU module, ROM and RAM families, Introduction to assembly language and machine language programming, Instruction set of typical microprocessor (e.g. 8085), Subroutine and stack, Timing diagram, Memory Interfacing, Interfacing input output-port, Interrupt and interrupt handling, Serial and parallel data transfer scheme, Programmed and interrupt driven data transfer, Direct memory access, Programmable peripheral devices, Programmable interval timer, Analog input-output using AD and DA converter.

UNIT 2:

Assembly language program of a typical microprocessor: Use of compilers, assembler, linker and debugger.

UNIT 3:

Basic 16 bit microprocessor (e.g. 8086): Architecture, Min-max mode.

UNIT 4:

Introduction to microcontroller: Architecture and instruction set of a typical microcontroller (e.g. PIC16F84 device), Feature of popular controller (processor 8031/8051), Its programming and interfacing.

This text is useful for subject code:
EI402—AEIE
Microprocessors and Computer Architecture

Module I:

Introduction to microprocessors: Overview of 8085, Internal architecture, Pin Diagram description. Software instruction set and Assembly Language Programming, Addressing Modes.

CHAPTER 1	INTRODUCTION TO MICROPROCESSORS & MICROCONTROLLERS
CHAPTER 2	ARCHITECTURE OF 8085 MICROPROCESSOR
CHAPTER 3	INSTRUCTION SET OF 8085 MICROPROCESSOR
CHAPTER 4	ASSEMBLY LANGUAGE PROGRAMMING USING 8085

Module II:

Instruction cycle, Machine cycle, Timing diagrams. Interrupts: Introduction, Interrupt vector table, Interrupt service routine, Design of programs using interrupts. DMA operation. Stack and stack handling, Call and subroutine, Counter, Time delay generation.

CHAPTER 3	INSTRUCTION SET OF 8085 MICROPROCESSOR
CHAPTER 4	ASSEMBLY LANGUAGE PROGRAMMING USING 8085
CHAPTER 5	MEMORY & INTERFACING WITH 8085 MICROPROCESSOR
CHAPTER 6	INTERRUPTS OF 8085 MICROPROCESSOR
CHAPTER 17	DIRECT MEMORY ACCESS (DMA) CONTROLLER 8257

Module III:

Hardware Interfacing: Interfacing memory, Interfacing I/O devices. Programmable peripheral devices (PPI) – Intel 8255, Programmable interval timer – Intel 8254, Programmable keyboard/display controller–Intel 8279, A/D and D/A converters and interfacing of the same.

CHAPTER 12	8255 INTERFACING WITH 8085, 8086 AND 8051 MICROCONTROLLER
CHAPTER 13	8253 INTERFACING WITH 8085, 8086 AND 8051 MICROCONTROLLER
CHAPTER 14	8259 INTERFACING WITH 8085, 8086 AND 8051 MICROCONTROLLER
CHAPTER 15	8279 INTERFACING WITH 8085, 8086 AND 8051 MICROCONTROLLER
CHAPTER 16	8251 INTERFACING WITH 8085, 8086 AND 8051 MICROCONTROLLER
CHAPTER 17	DIRECT MEMORY ACCESS (DMA) CONTROLLER 8257
CHAPTER 18	ADC, DAC, KEYBOARD, MULTIPLEX DISPLAY AND LCD INTERFACING WITH 8085, 8086, 8051

CHAPTER

1

Introduction to Microprocessors and Microcontrollers

1.1 INTRODUCTION TO MICROPROCESSORS AND MICROCONTROLLERS

Initially, standard logic gates, digital and analog ICs were used to measure any physical and electrical quantity in all electronics products. A product using standard logic gates can be replaced by interconnections of standard hardware with the logic stored in a ROM. When the logic circuit is concentrated in only a few components, a high degree of design flexibility is possible. This type of system has limitations on size, weight, power consumption and price. The microprocessor makes it possible to improve old products in all directions and develop more sophisticated new industrial products incorporating new features. Microprocessor technology has been used to replace hardware designs, which were formally implemented with logic devices. Actually, microprocessor applications are limited only by the technology rather than by the imagination of the designers.

The microprocessor is a VLSI IC in which large numbers of transistors are placed. As microprocessors are relatively new devices, these devices should be used to implement various functions such as measurement of electrical and physical quantities, monitoring, controlling and protection of any process control system, motion control, servo control system and power system, etc. These devices are programmable and can substitute program logic for hardwired logic. Initially, microprocessor cost was too high, but due to rapid decrease in the microprocessor-based system cost, enormous logic power can be added with some additional integrated circuits in a microcomputer. The advantages of *microprocessor-based design* of a system are given below:

- The manufacturing costs of the electronic products are generally lower, but the typical microprocessor-based designs cost 20 to 60 per cent of their TTL implementation costs.
- The time and cost for the original development can be substantially lowered. Due to applications of microprocessors, the design time can be reduced by about two thirds. Presently, numbers of software are available to design a prototype system before implementation of the final product. Therefore, the design cycle will continue to decrease.
- Consequently, microprocessor-based products can be brought to the market very early as per consumer requirement.
- Microprocessor-based products have many complex functional capabilities and these products can be provided at reasonable cost. Therefore, the realization of better products for the same or lower prices are possible.
- The smaller number of components in a microprocessor system increase the reliability of the final product.

- Sometimes microprocessor-based products fail. The computational capability of a microprocessor can be used to perform self-diagnosing of the product to find error and help to remove faults. These devices also provide substantial reductions in service charges.

In industry, there are a variety of microprocessor and microcontroller applications such as instrumentation, industrial control, and aerospace, etc. Some of the actual applications come across industrial boundaries and these are more informative to about the type of function to be performed. Microprocessors are used in data-collection terminals, office equipment, business machines, calculators, point-of-sale terminals, and various kinds of data-communication equipments. As the incremental cost for additional functions is very small in a microprocessor-based system, always there is an increasing tendency to add greater functional capability. This tendency is most noticeable in the area of instrumentation, where increasingly sophisticated products are finding their way to the market in growing numbers. Presently, modern instruments have the additional features such as remote control, programmability, improved readout, and peripheral interfaces.

Generally, microprocessors and microcontroller are also used to control traffic lights, appliances, motion control, position control, servo control, elevators, automation, electric car, and control of AC/DC machines, measurement and display of electrical and physical quantities such as voltage, current, frequency, phase angle, power factor, power, energy, force, displacement, speed, acceleration, temperature, pressure, stress, strain, deflection, water level, traffic-light control, overvoltage and overcurrent protection, speed control of dc and induction motors.

1.2 HISTORY OF MICROPROCESSORS

In 1643, Blaise Pascal, the French mathematician and philosopher, invented the first mechanical calculator which could perform addition as well as subtraction. In the 17th century, the multiplication and division actions were added by the German mathematician Gottfried Leibriz. The difference engine was developed in 1832 by Charles Babbage, professor of mathematics at Cambridge University. This machine could add, subtract, multiply, divide and perform a sequence of steps automatically. In 1887, Herman Hollerith invented a device for automatic census tabulation.

The first large-scale electronic digital computer was designed and constructed at the Moore School of Electrical Engineering of the University of Pennsylvania. In 1943, J W Mauchly and J Presper Eckert prepared a proposal for the US army to build an Electronic Numerical Integrator and Computer (ENIAC) and subsequently started construction of the ENIAC. In 1944, the ENIAC team members started work on stored-program computers. Then ENIAC was innovated in 1946. It occupied a room of approximately 12 m × 6 m and contained nearly 18000 vacuum tubes. Its power consumption was about 150 kW and it operated on numbers with ten decimal digits. Addition could be carried out at the rate of 5000 calculations per second, multiplication at 350 per second and division at 166 per second. It was able to store up to 20 different numbers and recall them immediately whenever required. After that John Von Neumann developed an improved version of the ENIAC machine with the help of all ENIAC team members.

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

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

common and inexpensive. Then in the late 1950s computer designers started developing ways to use the transistor in place of vacuum tubes.

In 1960s, the semiconductor industry developed a way to integrate a number of transistors on one silicon wafer. The transistors were connected together with small metal traces. When the transistors were connected together, they became a circuit which performed different functions such as gate, flip-flop, register, counter or adder. This new technology created the basic semiconductor building blocks. The building blocks or circuit modules made this way are known as an Integrated Circuits (ICs). Thereafter integrated circuits (ICs) became feasible and the integration has been developed with time. There are three different stages of development from the period 1961 to 1972, namely, Small-Scale Integration (SSI), Medium-Scale Integration (MSI) and Large-Scale Integration (LSI). In general, an SSI chip has dozens of transistors with their associated circuit components, but an MSI chip has hundreds of transistors and an LSI chip has thousands of transistors.

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

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

1.3 MICROPROCESSOR

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

The microprocessor is the Central Processing Unit (CPU) of digital computers and it is constructed with IC technology. Figure 1.1 shows the block diagram of a microprocessor. The microprocessor has a digital circuit for data handling and computation under program control. The microprocessor is a data-processing unit. Data processing includes both computation and data handling. Computation is performed by logic circuits called the Arithmetic Logic Unit (ALU). The ALU is used to perform add, subtract, AND, OR, XOR, compare, increment, and decrement functions. The ALU cannot perform any functions without control signals. In order to process data, the microprocessor must have control logic which instructs the microprocessor how to decode and execute the program.

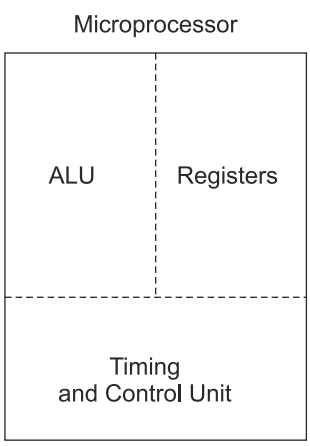


Fig. 1.1 Architecture of a microprocessor

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

The microprocessor always operates in binary digits: 0 and 1, known as *bits*. Bit is an abbreviation for 'binary digit' which can be represented in terms

of voltages. The microprocessor recognises and processes a group of bits, called the *word*. Microprocessors are classified according to their word length such as 8-bit, 16-bit, 32-bit and 64-bit microprocessors. The microprocessor ICs are programmable so that instructions can be executed by the microprocessor to perform given tasks within its capability. The instructions are stored in a storage device which is called the *memory*, and the microprocessor can read instructions from memory.

1.4 MICROCOMPUTER

Generally, the words 'microprocessor' and 'microcomputer' are used to correspond to the same thing, but in fact these words have different meanings. The microprocessor is an integrated circuit (IC) developed based on LSI or VLSI technology. It is the core of any computer system, but the microprocessor by itself is completely useless, until external peripheral devices are connected with it to interact with the outside world. The microcomputer is a complete computing system and it is built with a microprocessor, input/output devices and memory (RAM and ROM). The schematic block diagram of a microcomputer is shown in Fig. 1.2. The detailed architecture of a microcomputer is illustrated in Fig. 1.3.

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

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

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

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

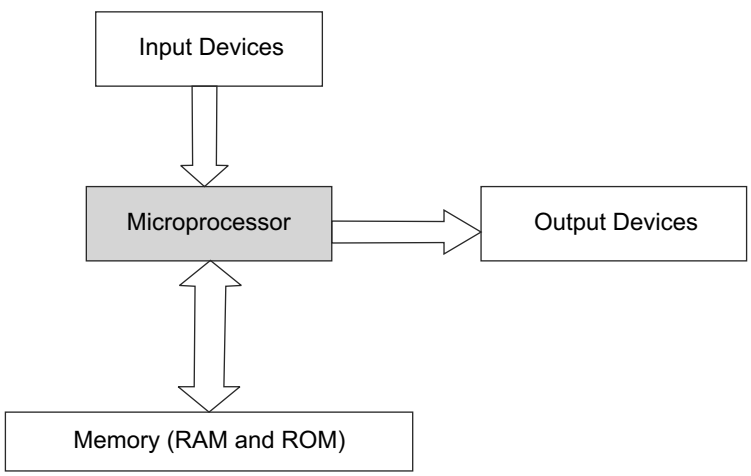


Fig. 1.2 Schematic block diagram of a microcomputer

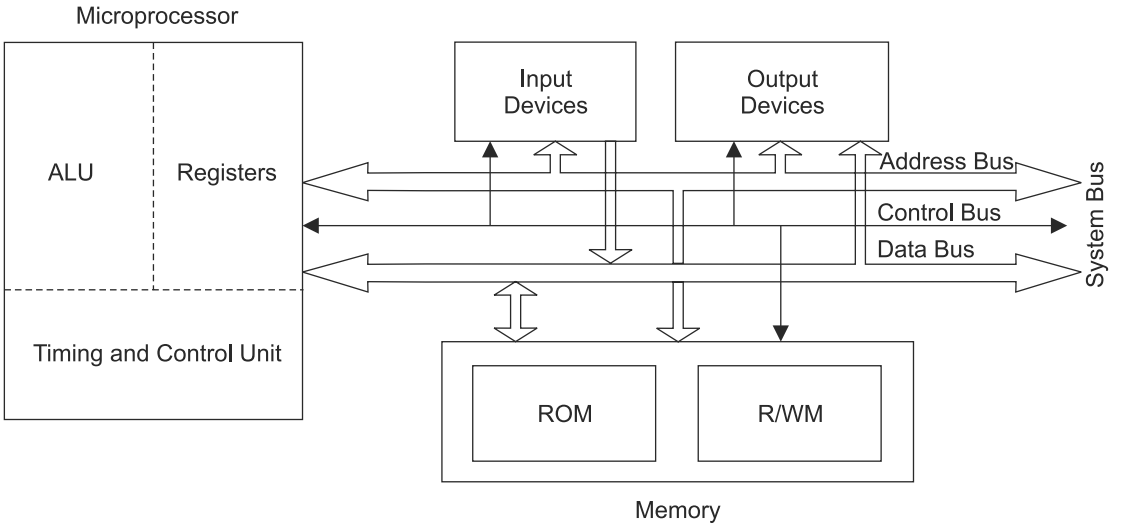


Fig. 1.3 Architecture of a microcomputer

Output The output devices transfer data from the microprocessor to any output device such as a printer, plotter, monitor, or magnetic tape.

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

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

1.5 EVOLUTION OF MICROPROCESSORS

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

Thereafter, in 1982, the 80186 processor was designed with few more instructions and additional on-chip circuits such as clock generators, timers, DMA controller and interrupt controller, but its addressing capability is the same as the 8086 microprocessor. But due to the need of large memory in advanced applications, processor designers put efforts in designing advanced microprocessors. The 80286 microprocessor is the first advanced microprocessor with proper memory management and protection abilities. It was developed by Intel in 1982 and it has an address capability 16 Mbyte and an operating frequency of 12.5 MHz.

The semiconductor technology could support the fabrication of a CPU with a 32-bit word size and higher operating frequency. Hence, the 32-bit processor 80386 was developed. The first 32-bit processor is 80386. The numerical processor 80387 is compatible with 80386. In 1989, the 80486 was developed by Intel which combines all the features of 80386 after incorporating a math processor 80387 inside the processor.

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

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

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

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

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

The Pentium III was eventually superseded by the Pentium 4. The Pentium 4 brand refers to Intel's line of single-core mainstream desktop and laptop central processing units developed in November 2000. This processor had the 7th-generation microarchitecture, called *NetBurst*. The original Pentium 4, codenamed 'Willamette', ran at 1.4 and 1.5 GHz and was released in November 2000 on the Socket 423 platform. In 2004, the initial 32-bit x86 instruction set of the Pentium 4 microprocessors was extended by the 64-bit x86-64 set. Pentium 4 CPUs introduced the SSE2 and, in later versions, SSE3 instruction sets to accelerate

calculations, transactions, media processing, 3D graphics, and games. In 2005, the Pentium 4 was complemented by the Pentium D and Pentium Extreme Edition dual-core CPUs.

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

The Intel Core 2 Extreme Quad-Core Processor QX6000 was introduced by Intel in 2007. This processor is designed to deliver performance across applications and usages in the Internet, image processing, video content creation, 3D, CAD, games, speech, multimedia and multitasking user environments. The Intel 64 architecture enables the processor to execute operating systems and applications written to take advantage of the Intel 64 architecture. Quad-core processors are available in the FC-LGA6 package with a 2x4 MB L2 cache.

The Intel Core 2 Duo processor uses architecture to create two cores on a single die or, in other words, there are two chips. It has better performance than dual-core processors in almost all benchmarking tests. They can be easily overclocked up to 4.0 GHz with suitable coolers. The Intel Core 2 Duo processor E8000 and E7000 series are 64-bit processors that maintain compatibility with IA-32 software and are based on the Enhanced Intel Core microarchitecture. These processors use Flip-Chip Land Grid Array (FC-LGA8) package technology, and plug into a 775-land surface mount, Land Grid Array (LGA) socket. These processors are based on a 45-nm process technology. The Intel Core 2 Duo processor E8000 series features a 1333 MHz Front Side Bus (FSB) and 6 MB of L2 cache. The Intel Core™2 Duo processor E8300 and E7200 were released on April 2008 and the Intel Core 2 Duo processor E7600 was developed in June 2009. These processors are used in Internet audio and streaming video, image processing, multimedia, and multitasking user environments. The differences between microprocessors are word length, size of the memory and speed at which the microprocessor can execute instructions. The comparison between difference microprocessors is shown in Table 1.1. Figure1.4 shows the evolution of processors with respect to year of development and number of transistors in the processor.

Table 1.1 Comparison of different microprocessors

Microprocessor	No. of Transistors	Data bus/ Word length	Address bus	Memory address range	Clock frequency	Pin	Year of development
4004	2300	4-bit	10-bit	640 B/1 kB	750 kHz	16	1971
8008	3500	8-bit	14-bit	16 kB	0.5 – .8 MHz	18	1972
8080	6000	8-bit	16-bit	64 kB	2 MHz	40	1974
8085	6500	8-bit	16-bit	64 kB	3 – 6 MHz	40	1976
8088	29K	8-bit/16-bit	20-bit	1 MB	5 – 10 MHz	40	1980
8086	29K	16-bit	20-bit	1 MB	5 – 10 MHz	40	1978
80186	29K	16-bit	20-bit	1 MB	5 – 16 MHz	68	1982
80286	134K	16-bit	24-bit	16 MB real 4 GB virtual	6 – 12.5 MHz	68	1982
80386	275K	32-bit	24/32-bit	4 GB real 64 TB virtual	20 – 33 MHz	132	1985
80486	3200K	32-bit	32-bit	4 GB real 64 TB virtual	25 – 100 MHz	168	1989

Contd.

Table 1.1 *(Contd.)*

Microprocessor	No. of Transistors length	Data bus/ Word	Address bus range	Memory address	Clock frequency	Pin	Year of development
Pentium	3200K	32-bit	32-bit	4 GB real	60 – 200 MHz	264	1993
Pentium Pro	5500K	32-bit	36-bit	64 GB	150 – 200 MHz	387	1995
Pentium II	7500K	32-bit	36-bit	64 GB	233 – 400 MHz	387	1997
Pentium III	9500K	32-bit	36-bit	64 GB	600 – 1000 MHz	387	1999
Pentium 4	55000K	32-bit	36-bit	64 GB	1.3 – 2 GHz	478	2001
Dual-Core Processor (Athlon)	1.72 billion	64-bit	40-bit	1 TB	2.93 GHz		2007
Core 2 Duo processor E8500	410 million transistors	64-bit	40-bit	1 TB	3.16 GHz	775	2008

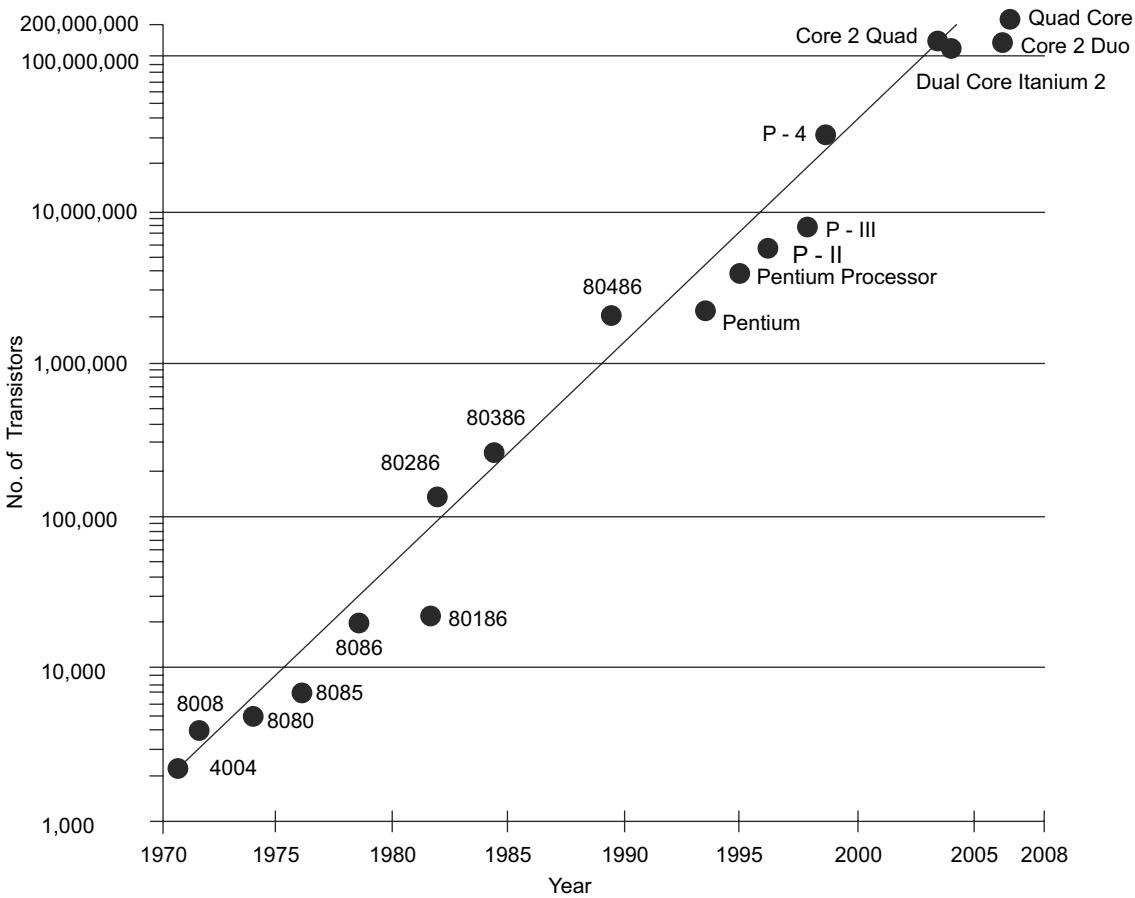


Fig. 1.4 Evolution of processors

1.6 MICROPROCESSOR APPLICATIONS

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

Toys Robots, remote-controlled cars, and hand held games.

Simple Applications Microwave ovens, telephone diallers, smart thermostats, shortwave scanners, and TV remote controls.

Complex Intelligent Product Controllers VCR control and programming, security systems, and lighting system controllers

Computer Peripherals Video display, higher-speed printers, modems, plotters, and communication controllers.

Industrial Controllers Robotics, processing control, sequence control, and machine tool control.

Instruments Logic analysers, communication analysers, disk drive testers, digital oscilloscopes, and smart voltmeters.

Communications Data, voice, mobile, electronic switching, and routing.

Automatic Test Equipment Automatic test equipment at all levels from development, fabrication, component testing assembly, PCB, module and system testing.

Electrical Power System Data acquisition, logging, protection, metering, control and processing, automatic control of generators voltage and fuel control of furnaces in a power plant.

Industrial Process Control Instrumentation, monitoring and control, data acquisition, logging and processing.

Household Appliances Cooking ovens, and washing machines.

Medical Electronics Quick patient check up, diagnosis, blood analysis, ECG, etc.

Database Management Word processing, database management, and storing information.

1.7 EVOLUTION OF MICROCONTROLLERS

A microcontroller is a small computer on a single Integrated Circuit (IC) containing a processor, memory, and programmable input/output ports. The program memory, in the form of flash or ROM, is also incorporated on a chip and a small amount of RAM is also included on a single chip. Microcontrollers are specially designed for embedded applications.

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

In 1980, Intel had developed an 8-bit microcontroller named the 8051 microcontroller. It had 128 bytes of RAM, 4 K bytes of on-chip ROM, two timers, four parallel ports with each port 8-bits wide and a serial port. This microcontroller had 2-byte instructions. Thereafter, the 8052 microcontroller was developed. This

microcontroller had all the standard features of 8051 with an extra 128 bytes of RAM, 4K bytes of ROM and an extra timer. Therefore 8052 had 256 bytes of RAM, 8 K bytes of ROM and three timers. The 8031 microcontroller is also a member of the 8051 family. This microcontroller has all features of 8051 microcontroller except 0 K bytes on-chip ROM. Table 1.2 shows the salient features of 8051, 8052, 8031 and 8032 microcontrollers.

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

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

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

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

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

1.8 ADVANTAGES AND DISADVANTAGES OF MICROCONTROLLERS

Advantages The advantages of microcontrollers are as follows:

Flexibility Microcontrollers are special types of processor chips that are very small and somewhat flexible, due to their programmable nature.

Faster Speed of Execution Since microcontrollers are fully integrated inside the processor, i.e., a "computer on a chip," these devices operate at faster speeds to execute instructions compared to general purpose microprocessors.

Inexpensive As microcontrollers are fully integrated onto one chip, these devices are cheap to manufacture. Usually, microcontrollers have much lower specifications than low-power consumer-grade general-purpose microprocessors, making them even easier to mass produce.

Rigid Once microcontrollers are programmed, typically they cannot be reprogrammed, if microcontrollers are controlled by Read-Only Memory (ROM) only rather than Random Access Memory (RAM) .

Labor Saving Many tasks can be performed by microcontrollers repetitively and human efforts can be saved. The programmable nature of these devices also allows manufacturing robots to reproduce these motions very quickly and consistently, increasing productivity.

Disadvantages The advantages of microcontrollers are as follows:

Complex Architecture Microcontrollers have more complex architecture than microprocessors. Therefore, understanding their functionality is quite difficult.

Development Time Due to complexity of the circuit board, the development time of a microcontroller increases and cost increases.

1.9 APPLICATIONS OF MICROCONTROLLERS

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

- Measurement of any physical quantity such as pressure, force, velocity, acceleration, displacement, force, stress, strain, and water level
- Microcontroller-based laboratory instruments to measure voltage, current, phase angle, power factor, frequency, resistance, power, and energy, etc.
- Robot-arm position control
- Angular speed measurement
- Temperature measurement
- dc motor and stepper motor control
- Induction motor control
- Traffic light control system
- Automobile applications
- Household appliances such as washing machine, light control, camera, TV, VCR and video games, etc.
- Office equipments such as photocopying machines, telephones, fax machines, printers, and security system, etc.

Review Questions

1.1 List the components of a microprocessor and microcomputer.

1.2 Write the difference between microprocessors and microcomputers.

1.3 What was the first microprocessor? Which company built that microprocessor?

1.4 Define SSI, MSI, LSI and VLSI.
1.5 What is ALU? Explain the following terms: registers, control unit, and input and output devices.
1.6 Draw the architecture of a microcomputer and explain it briefly.
1.7 Compare between the following processors:
(i) 8085 and 8086 (ii) 80286 and 80486 (iii) Pentium II and Pentium 4
1.8 Give a list of applications of microprocessors.
1.9 Define microprocessor, microcomputer and microcontroller
1.10 Write the features of 8051 microcontroller
1.11 List the components of a microprocessor and microcontroller
1.12 What are the advantages and disadvantages of microcontrollers?
1.13 Write the application of microcontrollers.
1.14 What is the major difference between 8051 and 8052 microcontrollers?
1.15 What are the advantages of microprocessor based system design?

Multiple-Choice Questions

1.1 The first microprocessor was
(a) 4001 (b) 4002 (c) 4003 (d) 4004
1.2 The 64-bit processor is
(a) Pentium (b) Pentium II (c) Pentium III (d) Pentium 4
1.3 The memory capacity of the 8085 microprocessor is
(a) 64 k (b) 1 MB (c) 16 MB (d) 640 B
1.4 The address bus 80186 microprocessor is
(a) 16 bit (b) 20 bit (c) 24 bit (d) 32 bit
1.5 The operating frequency of the 8086 microprocessor is about
(a) 750 kHz (b) 3 – 6 MHz (c) 5 – 10 MHz (d) 3 – 6 GHz
1.6 The first electronic computer was
(a) ENIAC (b) EDVAC (c) EDSAC (d) Difference Engine
1.7 Ten thousand and more transistors exist in
(a) LSI ICs (b) MSI ICs (c) SSI ICs (d) VLSI ICs
1.8 The memory capacity of a Pentium Pro microprocessor is
(a) 64 KB (b) 64 MB (c) 64 GB (d) 640 B
1.9 A microcontroller has
(a) ROM (b) RAM (c) I/o Ports (d) All of these
1.10 8051 microcontroller is a _____ processor
(a) 4-bit (b) 8-bit (c) 16-bit (d) 32-bit
1.11 The 8052 microcontroller has
(a) 20 pins for I/o (b) 32 pins for I/o (c) 35 pins for I/o (d) 40 pins for I/o

1.12 8051 microcontroller has _____ bytes on chip program memory.
(a) 2 K (b) 4 K (c) 8 K (d) 16 K

Answers to Multiple-Choice Questions

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

CHAPTER

2

Architecture of 8085 Microprocessor

2.1 INTRODUCTION

The computer is a machine that processes data and generates information with speed and accuracy. Electronic, electromechanical devices and software make this machine. It is a programmable machine. The basic block diagram of a computer is shown in Fig. 2.1. The computer consists of four basic units, namely, input (I/P), memory, output (O/P), and central processing unit.

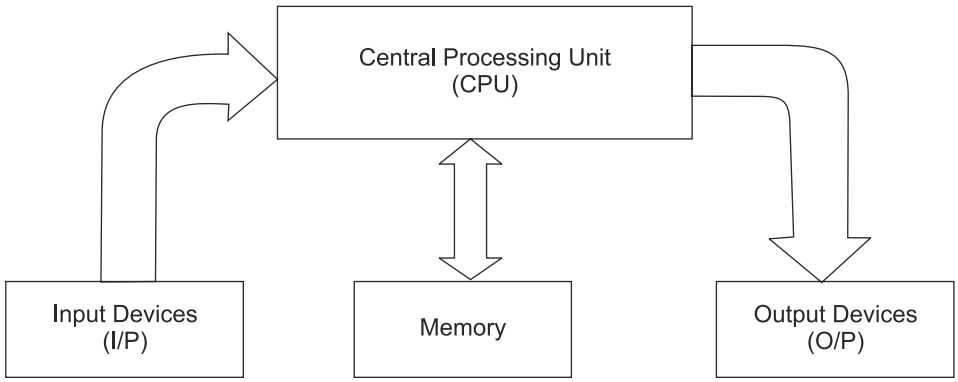
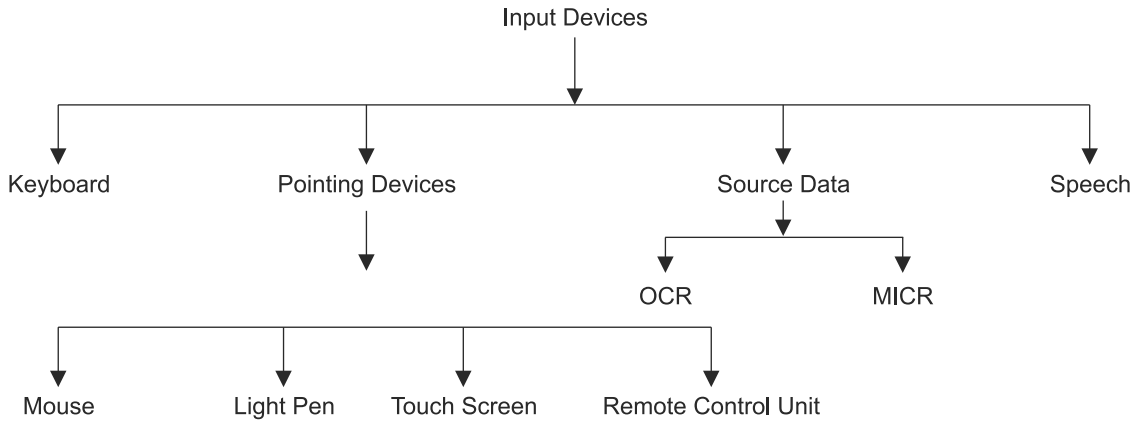


Fig. 2.1 Basic block diagram of a computer

2.1.1 Input Devices

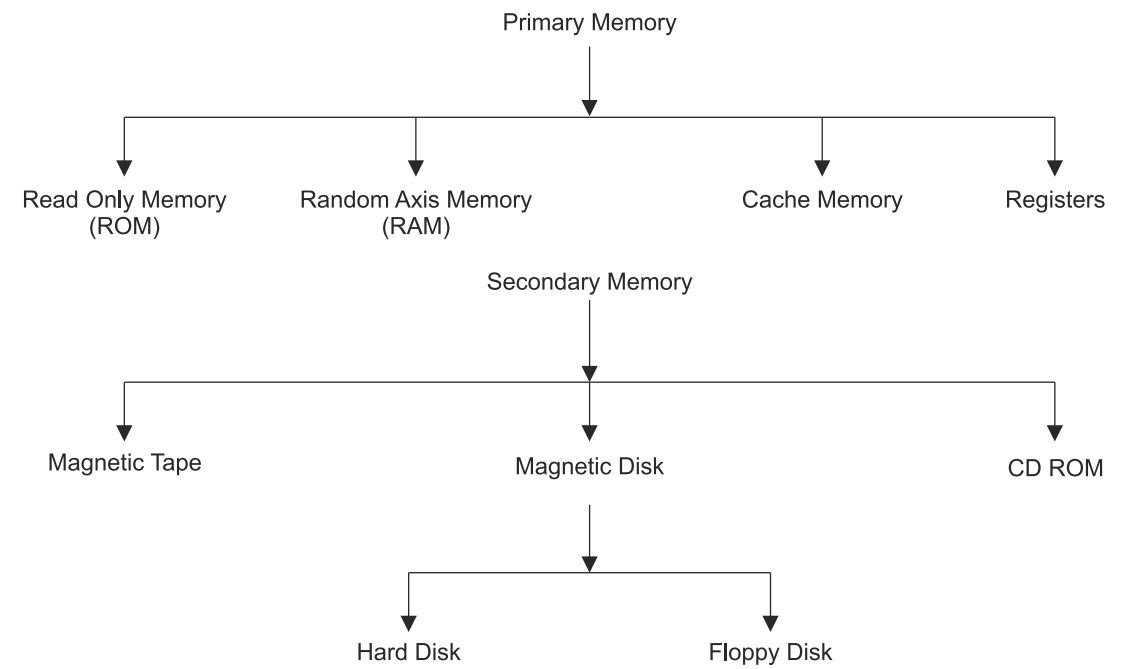
An input device accepts data from the environment, converts it into digital form and sends it to the memory for storing in the computer. Commonly used input devices are punched cards, paper tapes, magnetic tapes, floppy disks and magnetic disks. Card readers, paper tape readers, magnetic tape readers, disk drives read data transmitted by input devices. A keyboard terminal can be used as input to the computer. In optical mark readers and optical character readers, the input data are scanned by an array of photocells, converted into machine code and transmitted into the memory of the computer for processing. On identical principles, bar-code readers read the information prepared in bar-code for application in computers. In a magnetic ink reader,

information written or printed in magnetic ink is read and transmitted directly to the memory for processing. Electronic mouse, touch screens and light pens are also used as input devices.



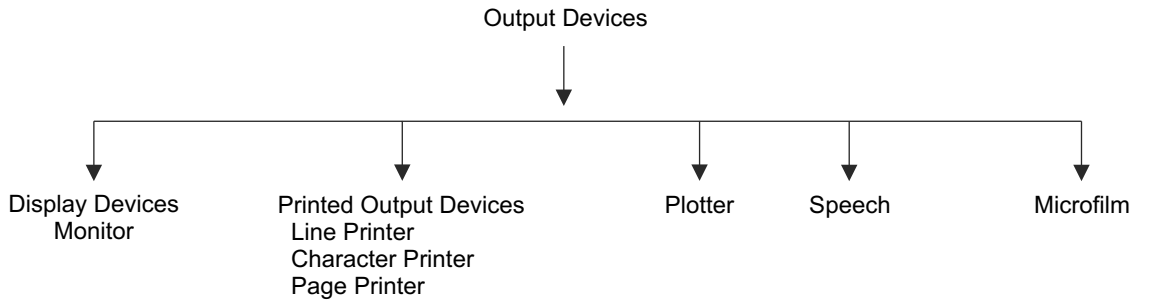
2.1.2 Memory

A computer system also has storage areas, often referred to as memory. The memory unit stores the information to be processed by the CPU. This information consists of the program as well as data. The memory can receive data, hold them and deliver them when instructed to do so. The storage available in the memory is also known as *main storage* or *primary storage*. The data can be processed only when it is available in the main memory, which is finite. It may be increased by adding auxiliary or secondary storage, such as magnetic tapes or magnetic disks. The information stored in the auxiliary storage can be transferred to the main memory for processing at a high speed.



2.1.3 Output Devices

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



2.1.4 Central Processing Unit

The central processing unit is the brain of the computer. It executes the programmer's software and control memory, input and output devices. The program is stored in the memory. The CPU fetches instructions of a program from the memory sequentially. It fetches one instruction at a time, decodes it and then executes it. After decoding an instruction, the CPU comes to know what operations are to be performed. It also comes to know whether the data to be processed are in the memory; general-purpose registers of the microprocessor or at input/output ports. If data are in the general-purpose registers, the CPU executes the program. The CPU controls memory, input and output devices to receive, store and send data/result of the program under execution. Under its control programs, data and results are displayed on the CRT, stored in the memory or printed by the printer. The major components of a CPU are ALU, timing and control unit and registers as depicted in Fig. 2.2.

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

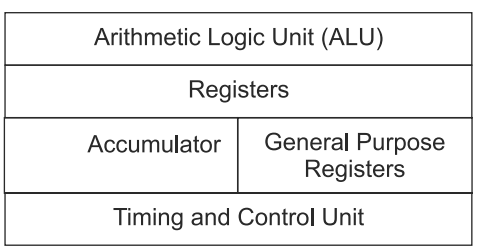


Fig. 2.2 Block diagram of Central Processing Unit (CPU)

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

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

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

The central processing unit built on a single IC is called a *microprocessor*. In a microcomputer, the microprocessor acts as the central processing unit. Figure 2.3 shows the block diagram of a microcomputer. In this chapter, the architecture of a microprocessor is explained.

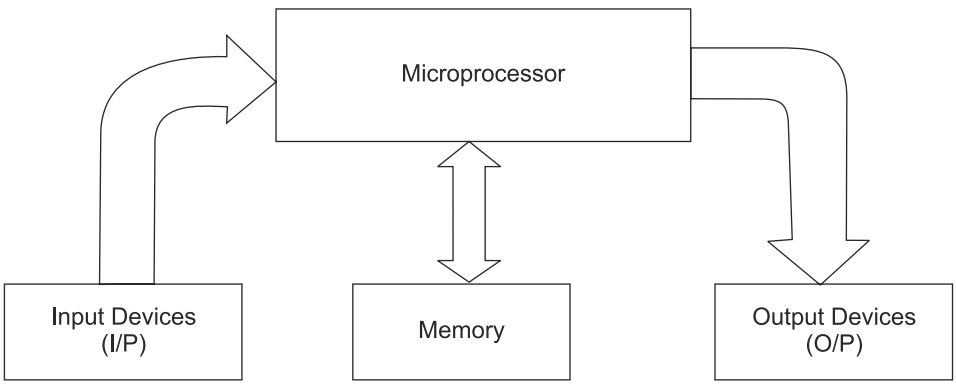


Fig. 2.3 Basic block diagram of a microcomputer

2.2 ARCHITECTURE OF THE 8085 MICROPROCESSOR

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

Features

- Power-down mode (HALT-HOLD)
- Low Power Dissipation of about 50 mW
- Single +3 to +6 V power supply
- Operating temperature from –40 to +85°C
- On-chip clock generator incorporating external crystal oscillators
- On-chip system controller
- Four vectored interrupt including one non-maskable
- Serial input/Serial output port
- Addressing capability to 64 kBytes of memory
- TTL compatible
- Available in 40-pin Plastic DIP package

2.2.1 Block Diagram of the 8085 Microprocessor

The functional block diagram of the Intel 8085 is depicted in Fig. 2.4. It consists of three main sections; an arithmetic and logic unit, timing and control unit and a set of registers. These important sections are described in the subsequent sections.

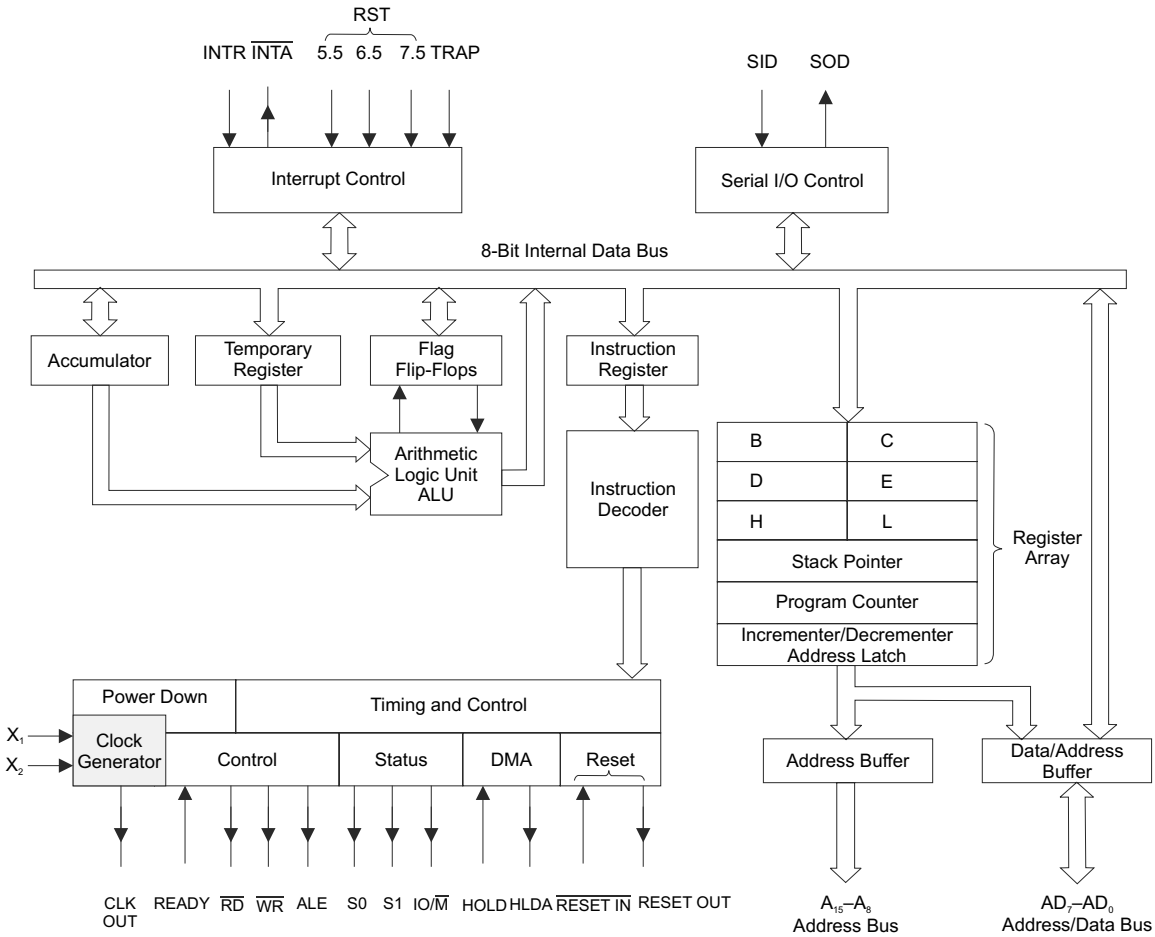


Fig. 2.4 Architecture of the 8085 microprocessor

2.2.2 Operation of the 8085 Microprocessor

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

The memory read/write and input/output read and write operations are performed as part of communication between the microprocessor and memory or input/output devices. The microprocessor communicates with the memory, I/O devices through address bus, data bus and control bus as depicted in Fig. 2.5. For this

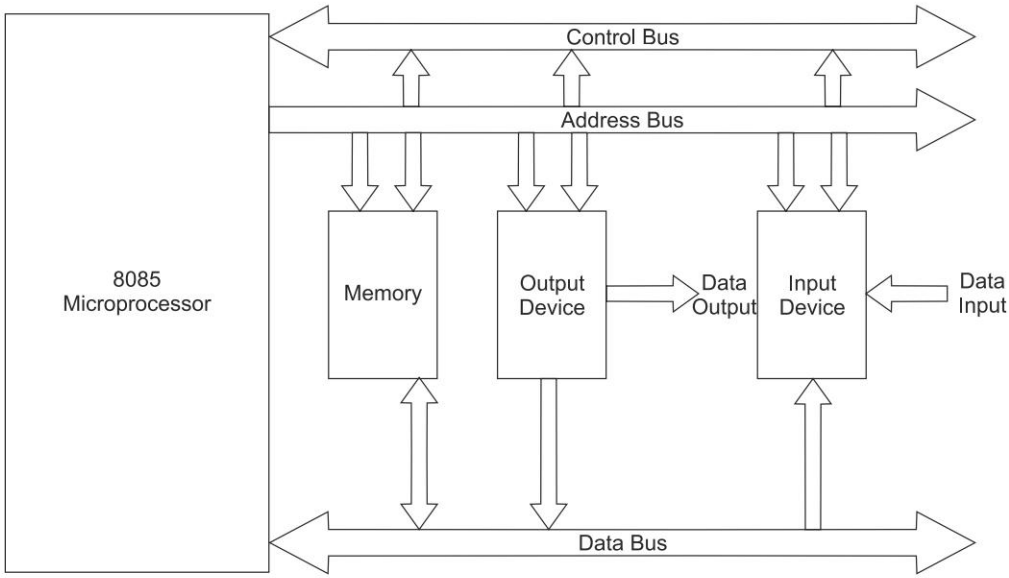


Fig. 2.5 Bus structure of the 8085 microprocessor

communication, firstly the microprocessor identifies the peripheral devices by proper addressing. Then it sends data and provides control signal for synchronisation.

Figure 2.6 shows the memory read operation. Initially, the microprocessor places a 16-bit address on the address bus. Then the external decoders logic circuit decodes the 16-bit address on the address bus and the memory location is identified. Thereafter, the microprocessor sends $\overline{MEMR}$ control signal which enables the memory IC. After that, the content of memory location is placed on the data bus and also sent to the microprocessor. Figure 2.7 shows the data flow diagram for data transfer from the memory to the microprocessor. The step-by-step procedure of data flow is given below:

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

2.2.3 Arithmetic Logic Unit (ALU)

All arithmetic and logical operations are performed in the Arithmetic Logic Unit (ALU). The functioning of the ALU are given in Fig. 2.8. The ALU functioning consists of Accumulator (A), Temporary Register (TR), Flag Register (FR) and arithmetic logic unit. The temporary register is not accessible to the user. Therefore, the user cannot read the content of TR. Actually, this register is used to store or load the operand during arithmetic and logical operations. Accumulator, TR and flag registers are explained in Section 2.1 in detail. The ALU always operates with one or two operands. Generally, operands are available in general-purpose

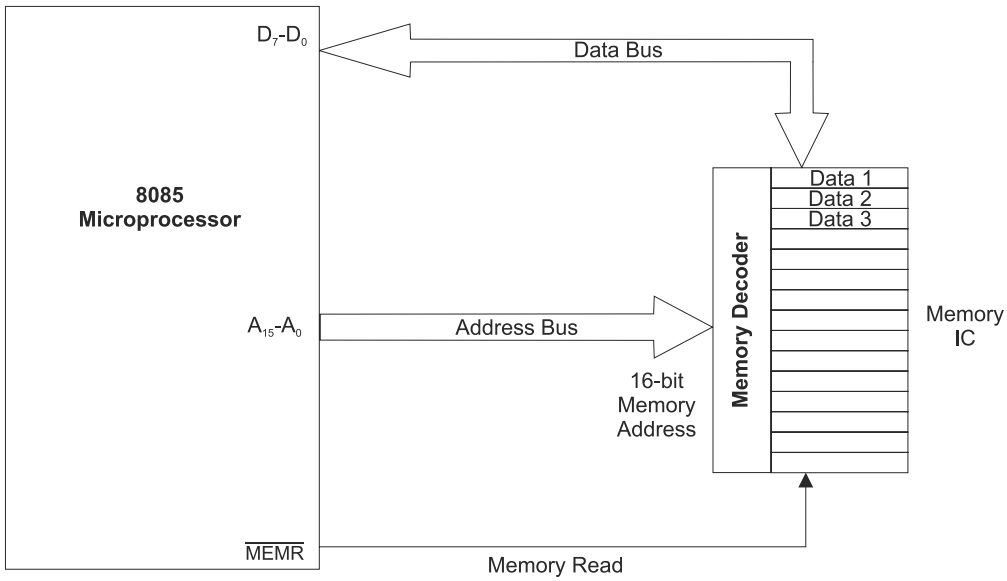


Fig. 2.6 Memory read operation

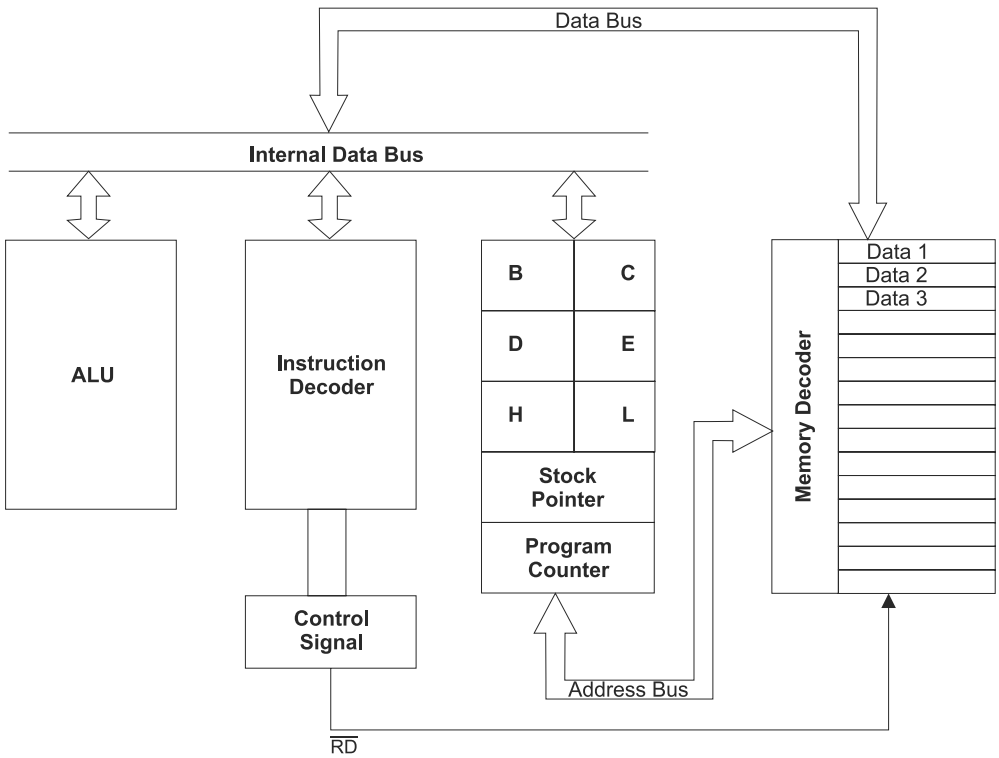


Fig. 2.7 Data flow from memory to microprocessor

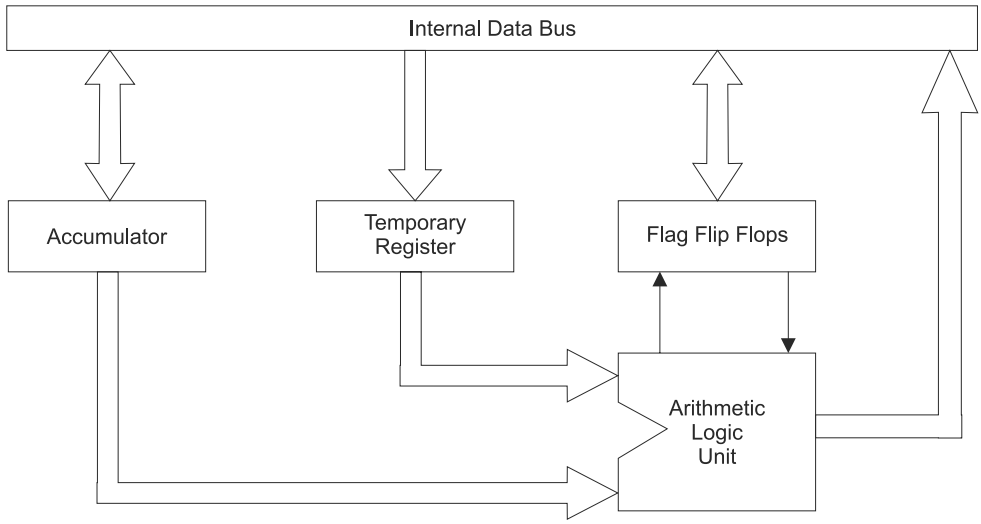


Fig. 2.8 ALU functioning

registers or memory locations. The result after arithmetic and logical operations are stored in accumulator. The sequence of operations in ALU are given below:

(i) One operand is in the A register.

(ii) The other operand may be in the general-purpose register or memory location, which will be transferred to the temporary register.

(iii) Then content of accumulator and temporary register are considered as inputs of ALU and the specified operation is carried out in the ALU.

(iv) The result of ALU operation is transferred in the A register through the internal data bus.

(v) The content of the flag register will be changed depending on the result.

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

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

2.2.4 Timing and Control Unit

The control unit controls the operations of different units with the CPU. This unit generates timing sequence signals for the execution of instructions. This unit controls the data flow between CPU and memory and CPU

and peripheral devices. This unit provides—control, status, DMA and reset signals to perform any memory and input–output related operations. Actually, it controls the entire operation of microprocessors. Therefore, the timing and control unit acts as the brain of the microprocessor.

2.2.5 Registers

The Intel 8085 has six general-purpose registers to store 8-bit data and these registers are identified as B, C, D, E, H and L. When two registers are combined, 16-bit data can be stored in a register pair. Only possible combinations of register pairs are BC, DE and HL. These register pairs are used to perform 16-bit operations. There is an accumulator register and one flag register. The accumulator is an 8-bit register. Arithmetic and logical operations are performed in the accumulator, and after operation, the result is stored in the accumulator. In addition with the above registers, there are two 16-bit registers, namely, the Stack Pointer (SP) and Program Counter (PC). The following registers of the Intel 8085 microprocessor are depicted in Fig. 2.4.

- One 8-bit accumulator (ACC) known as register A
- Six 8-bit general-purpose registers: B, C, D, E, H and L
- One 16-bit stack pointer (SP)
- One 16-bit program counter (PC)
- Instruction register
- Temporary register
- Program status word register (PSW)

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

General-Purpose Registers The general-purpose registers of the 8085 microprocessor are B, C, D, E, H and L registers as shown in Fig. 2.9. These registers are used to store 8-bit operands. To hold 16-bit data or a 16-bit memory address location, two 8-bit registers can be combined. The combination of two 8-bit registers is known as a register pair. The only possible combination register pairs of the 8085 microprocessor are B–C, D–E and H–L. The programmer cannot form a register pair by selecting any two registers of his choice. The H–L register pair can be used for the address of a memory location whereas B–C and D–E register pairs are used to store 16-bit data. During the execution of the program, all general-purpose registers can be accessed by program instructions and also used for data manipulation.

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

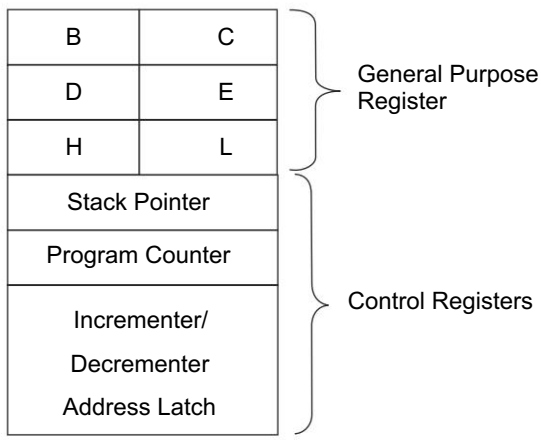


Fig. 2.9 Registers of the 8085 microprocessor

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

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

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

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

Figure 2.10 shows an 8-bit register, which indicates bit positions of different flags. This register is known as flag register and it is adjacent to the accumulator. Though it is an eight bit register, only five bit positions out of eight are used to store the outputs of the five flip-flops. The flags are stored in the 8-bit register so that the programmer can check these flags through an instruction. These flags are used in the decision-making process of the microprocessor.

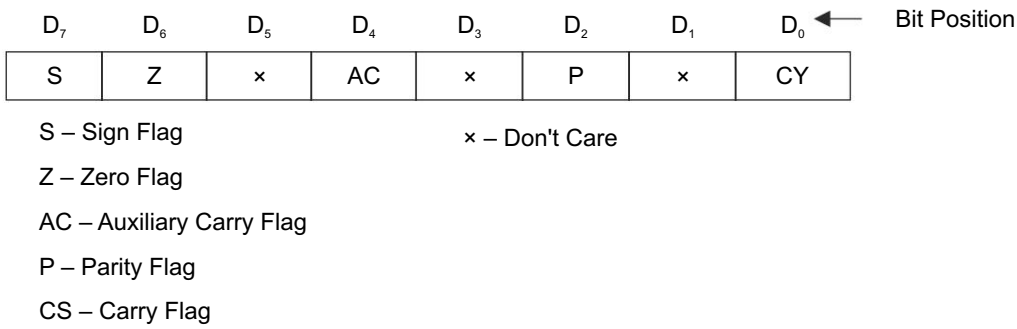


Fig. 2.10 Flag register

(a) Carry Flag (CY) The arithmetic operation generates a carry in case of addition or a borrow in case of subtraction after execution of an arithmetic instruction and the Carry Flag (CY) is set to 1. When the two 8-bit numbers are added and the sum is larger than 8 bits, a carry is produced and the Carry Flag (CY) is set to 1. During subtraction, if borrow is generated, the carry flag is also set to 1. The position of the carry flag is D_0 as depicted in Fig. 2.10.

(b) Parity Flag (P) After an arithmetic or logical operation, if the number of 1_s in the result is even (even parity), this parity status flag (P) is set, and if the number of 1_s is odd (odd parity), this flag is reset. For example, if the data byte is 1 1 1 1 1 1 1 1, the number of 1_s in the data byte is eight (even parity) and the parity flag (P) is set to 1. The position of the parity flag is D_2 as shown in Fig. 2.10.

(c) Auxiliary Carry Flag (AC) In arithmetic operations of numbers, if a carry is generated by bit D_3 and

passed on to D_4, the auxiliary carry flag (AC) is set. Actually, this flag is used for internally Binary Coded Decimal (BCD) operations and this is not available for the programmer to change the sequence of operations through jump instructions. The position of the auxiliary carry flag is D_4 as given in Fig. 2.10.

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

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

The most significant bit of an 8-bit data is the sign bit. When a number is negative, the sign bit is 1. If the number is positive, the sign bit is 0. For an 8-bit signed operation, the remaining 7 bits are used to represent the magnitude of a number. After execution of a signed arithmetic operation, the MSB of the result also represents its sign. The position of a sign flag is D_7 as depicted in Fig. 2.10.

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

4. Instruction Register (IR) The instruction register holds the operation code (opcode) of the current instruction of a program during an arithmetic/logical operation. The instruction is fetched from memory prior to execution. The decoder takes instruction and decodes the instruction. The decoded instruction is then passed to the next stage for execution.

5. Memory Address Register (MAR) The Memory Address Register (MAR) holds the address of the next program instruction. Then MAR feeds the address bus with addresses of the memory location of the program instruction which will be executed.

6. Temporary Register This is an 8-bit register, which is associated with the ALU. This register holds data during arithmetic and logical operation. This register can be used by the microprocessor but this is not accessible to a programmer.

2.2.6 System Bus: Address Bus, Data and Control Bus

The system bus is collection of wires, which are used to transfer binary numbers, one bit per wire. The 8085 microprocessor communicates with memory and input and output devices using three buses, namely, address bus, data bus and control bus as depicted in Fig. 2.11.

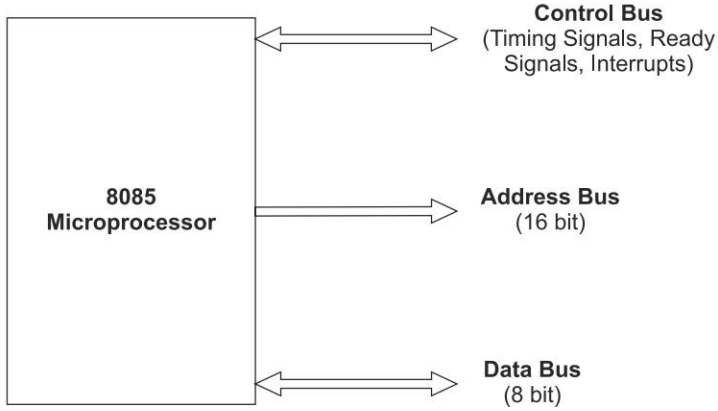


Fig. 2.11 Microprocessor and its buses

Address Bus Each memory location has a unique address. The address bus consists of 16 wires, therefore, the address bus has 16 bits. Its "width" is 16-bits. A 16-bit binary number allows 2^{16} different numbers, or 65536 different numbers, i.e., 0000 0000 0000 0000 up to 1111 1111 1111 1111. Therefore, the Intel 8085 microprocessor has 65536 = 64 K (where 1K = 1024) memory for locations and each memory location contains 1 byte of data. The address bus is unidirectional which means numbers are only sent from microprocessor to memory, and not the other way. The 16-bit address bus consists of the 8 most significant bits of the address A_{15}–A_8 and the 8 least significant bits of the address/data AD_7–AD_0. A_7–A_0 is multiplexed with the data lines D_0–D_7. During the first clock period of the machine cycle, the microprocessor sends 8 MSBs of the address on the A bus and 8 LSBs of the address on the AD bus. If the data was sent during the first clock period, it will be latched. After the first clock pulse, AD lines as data bus are shown in Fig. 2.12 and the timing diagram is depicted in Fig. 2.13.

Data Bus The data bus as 8-bit data is stored in each memory location. The data bus is used to move or transfer data in binary form. The data is transferred between the microprocessor and external devices. In

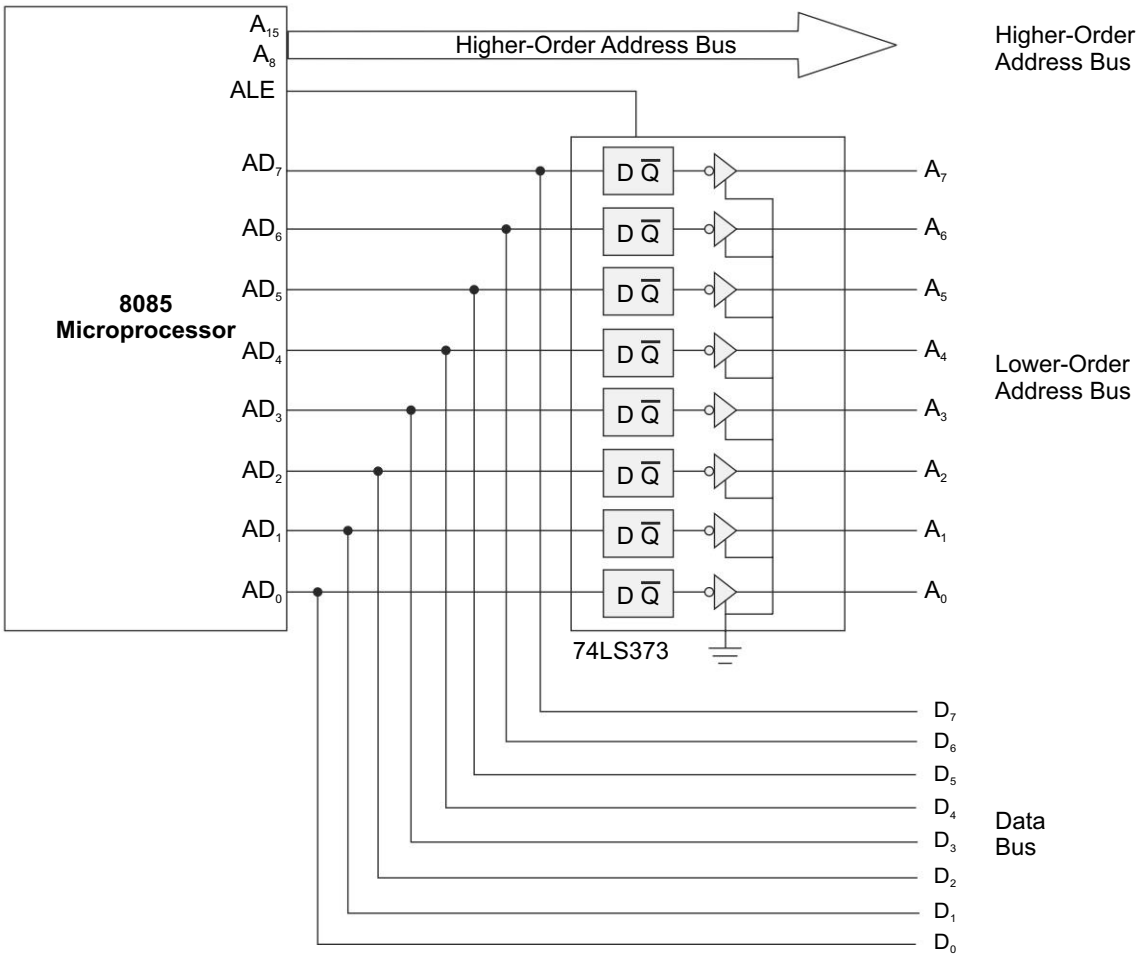


Fig. 2.12 Multiplexing of lower-order address bus

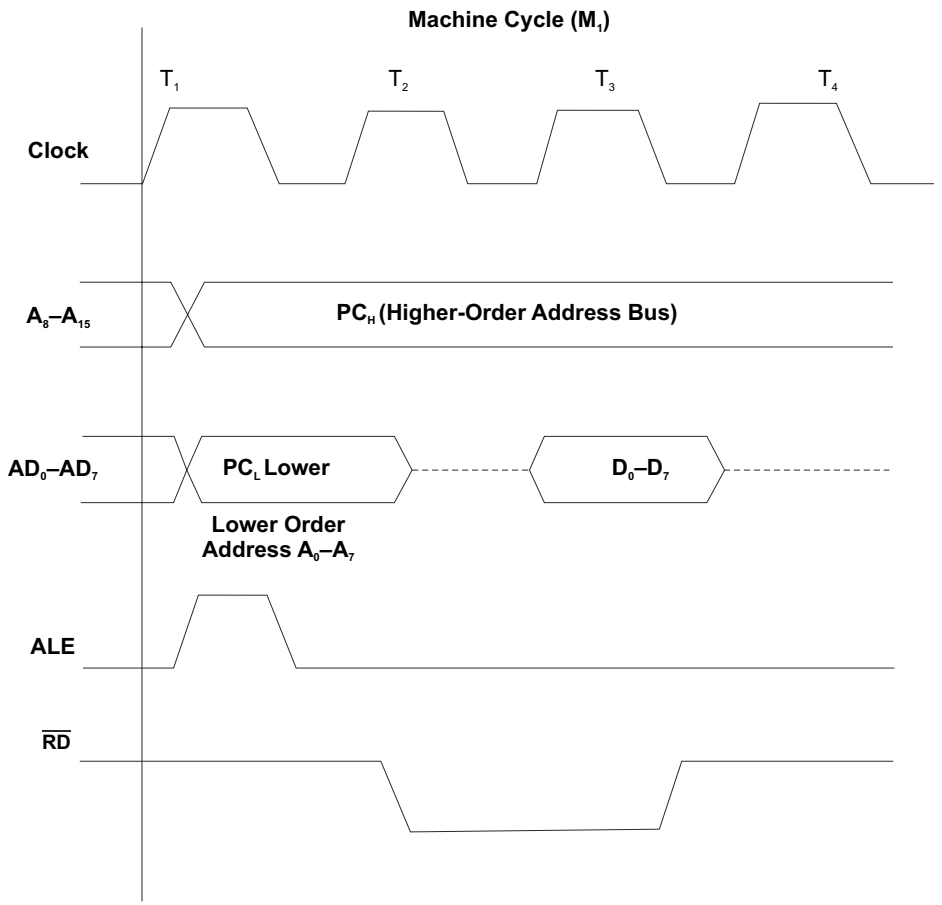


Fig. 2.13 Time multiplexing of address bus

the 8085 microprocessor, the data size is 8 bits. Consequently, the data bus typically consists of 8 wires. Therefore, there are 2^8 combinations of binary digits. The data bus is used to transmit 'data', i.e., information, results of arithmetic, etc., between memory and the microprocessor. This bus is bi-directional. The size of the data bus determines what arithmetic can be done. As a data bus is 8 bits wide, the largest number is 11111111 (255 in decimal).

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

Control Bus The control bus has various lines, which have specific functions for coordinating and controlling microprocessor operations. For example, $RD/\overline{WR}$ line is a control signal and this is also a single binary digit. This signal can differentiate the read and write operations. When RD is logically '0', memory and other input–output devices are read. If $\overline{WR}$ is logically '0', data can be written in memory and any other devices. Various other control signals are used to control and coordinate the operation of the system. Typically, the 8085 microprocessor has the following control lines: S_0, S_1, $\overline{RD}$, $\overline{WR}$ and $IO/\overline{M}$. The microprocessor cannot function correctly without these vital control signals. The Control Bus carries control signals—partly unidirectional, and partly bi-directional.

2.3 PIN DIAGRAM OF THE 8085 MICROPROCESSOR

Figure 2.14 shows the schematic diagram of Intel 8085. The PIN diagram of the 8085 microprocessor is illustrated in Fig. 2.15. The descriptions of various pins are as follows:

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

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

The address on the higher order bus remains on the bus for the entire machine cycle. But the lower-order address is changed after the first clock cycle. Actually, this address is latched and used for identifying the memory address. The AD_7–AD_0 is used to identify the memory location. Figure 2.12 shows the address bus A_{15} to A_0 after the latching operation. When the ALE signal is low, the data is latched till the next ALE signal. The output of the latch represents the low-order address bus A_7–A_0. If ALE is high, the latch is transparent, which means that the output changes according to input data.

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

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

Table 2.1 (a) Status codes and states of 8085

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

Table 2.1 (b) Status codes and states of 8085

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

*High impedance state, x — Unspecified

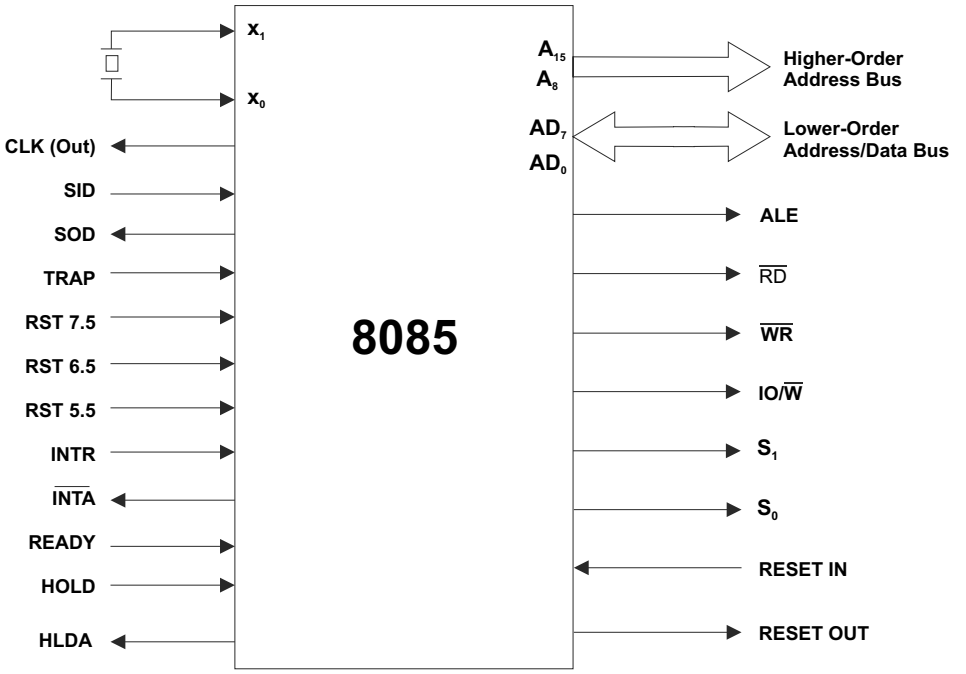


Fig. 2.14 The schematic diagram of Intel 8085

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

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

Figure 2.16 shows the generation of four different control signals by combining $\overline{RD}$, $\overline{WR}$ and $IO/\overline{M}$ signals. The signal $IO/\overline{M}$ is low for any memory-related operation. The $IO/\overline{M}$ is logically ANDed with $\overline{RD}$, $\overline{WR}$ signals and generates memory read $\overline{MEMR}$ and memory write $\overline{MEMW}$ control signals. If $IO/\overline{M}$ becomes high, then input/output peripheral operations are carried out.

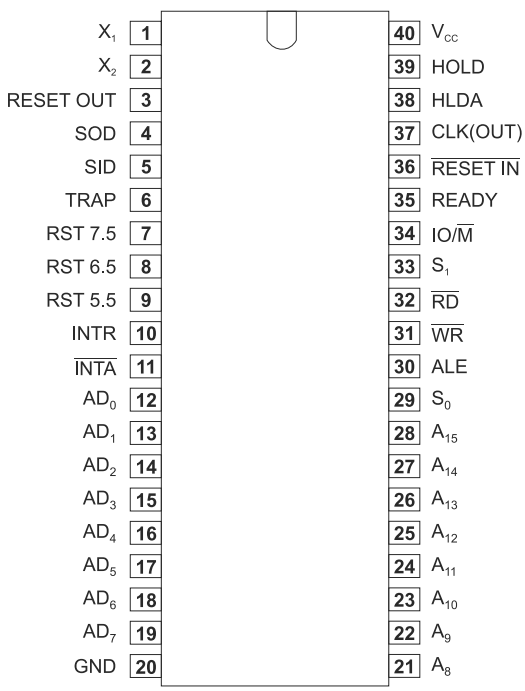


Fig. 2.15 Pin diagram of Intel 8085

It is depicted in Fig. 2.16 that the signal is ANDed with $\overline{RD}$, $\overline{WR}$ signals and generate I/O read and I/O write control signals for any I/O related operation.

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

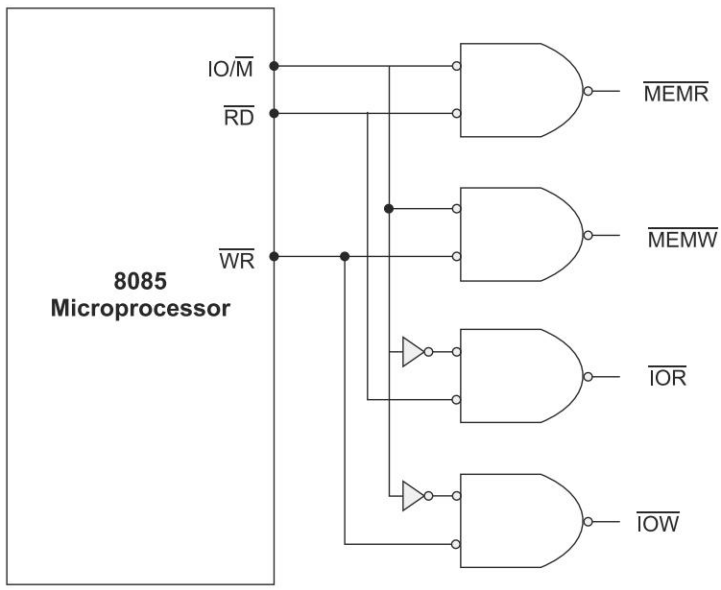


Fig. 2.16 Generation of Memory and I/O Read/Write control signals

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

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

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

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

$\overline{INTA}$ (Output) It is an interrupt acknowledge signal. This is used instead of $\overline{RD}$ during the instruction

cycle after an INTR is accepted. This signal is sent by the microprocessor after INTR is received. It can be used to activate the 8259 interrupt IC.

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

Table 2.2 Interrupt priorities and restart address

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

TRAP (Inputs) Trap interrupt is a non-maskable restart interrupt. It has the highest priority of any interrupt as depicted in Table 2.1. It is recognised at the same timing as INTR or RST 5.5 or RST 6.5 or RST 7.5. It is unaffected by any mask or interrupt enable.

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

RESET OUT (Output) RESET OUT indicates that the CPU is in reset condition. This can be used as a system reset. This signal is also synchronised to the processor clock and lasts an integral number of clock periods.

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

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

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

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

VCC (+ 5 Volt supply) The 8085 microprocessor operates on a 5-V supply, which is connected with V_{cc} terminal at pin number 40.

GND (Ground Reference) The power supply ground is connected to GND at pin number 20.

2.4 COMPARISON OF 8085 AND 8080A

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

Table 2.3 Comparison between 8085 and 8080A

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

Example 2.1 Determine the status of different flags after addition of 07H and CFH.

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

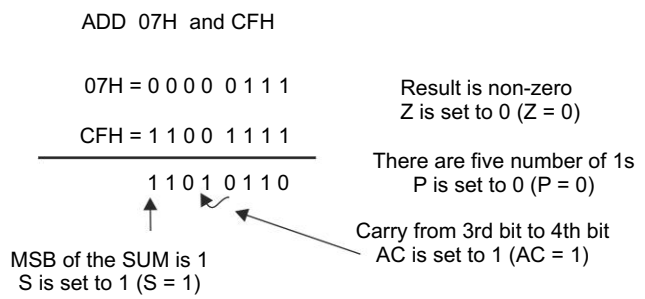


Fig. 2.8 Status of different flags after addition of 07H and CFH

Example 2.2 Find out the status of different flags after addition of CEH and 9BH.

Sol. If CEH and 9BH are added, the result is non zero. Hence, the *Z* flag is set to 0. There is a carry from 3rd bit to 4th bit. As a result, the auxiliary carry (AC) flag is set to 1. Since the MSB of the sum is 0, the *S* flag is

set to 0. As there are four numbers of 1s in the result, the parity flag (*P*) is set to 1. The carry is generated after addition of CEH and 9BH as the sum is greater than 8 bits. Therefore, CS is set to 1. Figure 2.17 shows the status of different flags after addition of CEH and 9BH.

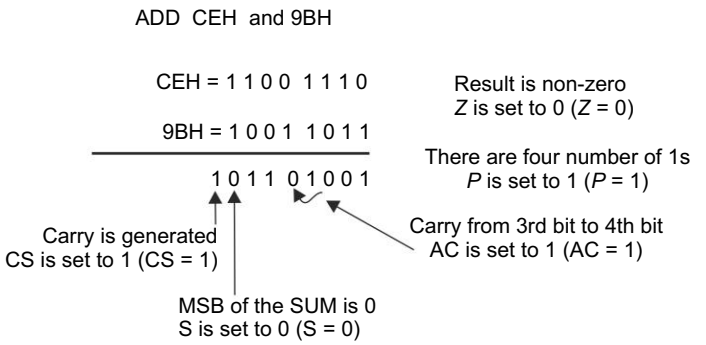


Fig. 2.17 Status of different flags after addition of CEH and 9BH

Review Questions

2.1 Draw the schematic diagram of all the functional blocks of the 8085 microprocessor along with their necessary interconnection. Explain in brief this architecture.

2.2 List the various registers of 8085 and explain their function.

2.3 What are the flags in 8085? Discuss the flag register with some example.

2.4 Explain the need of a program counter, stack pointer and status flags in the architecture of Intel 8085 microprocessor.

2.5 What is the clock frequency of 8085 microprocessor if crystal frequency is 5 MHz?

2.6 An 8085 program adds the hex numbers 2FH and 32H and places the result in its accumulator. What would be the status of the 8085 flags *CY, P, AC, Z, S* on completion of this addition?

2.7 Draw the schematic diagram to generate Read/Write control signals ($\overline{MEMR}$, $\overline{MEMW}$, $\overline{IOR}$, $\overline{IOW}$) for memory and I/O devices in the 8085-microprocessor.

2.8 Explain the control and status signals of the microprocessor in memory read and write operations.

2.9 Discuss the function of the following signals of 8085 microprocessor:

(i) ALE (ii) INTR (iii) INTA

(iv) HOLD (v) HLDA (vi) READY

2.10 Explain the functions of following interrupt signal lines of 8085A

(i) TRAP (ii) RST 7.5 (iii) RST 6.5 (iv) RST 5.5

2.11 Draw and explain the time multiplexing of $AD_0 - AD_7$.

2.12 Mention the purpose of SID and SOD lines.

2.13 Write the difference between 8085 and 8080 microprocessor.

2.14 What is the difference between microprocessor and microcontroller? Describe how data can flow between microprocessor, memory and I/O devices.

2.15 An 8085 program subtracts the hex number 23H and FFH and places the result in its accumulator.

What would be the status of the 8085 flags CY, P, AC, Z, S after completion of this subtraction.

2.16 What would be the status of the 8085 flags after addition of 69H and 72H.

2.17 Find out the status of different flags after addition of CBH and E8H.

2.18 Determine the status of different flags after subtraction 25H from FFH.

2.19 Explain the need to demultiplex the bus AD_0–AD_7.

2.20 Mention the purpose of SID and SOD lines.

Multiple-Choice Questions

2.1 The microprocessor was introduced in the year
(a) 1940 (b) 1971 (c) 1973 (d) 1980

2.2 Which semiconductor technology is used for fabrication of the 8085 microprocessor?
(a) ECL (b) NMOS (c) NMOS and HMOS (d) NMOS and CMOS

2.3 Which of the following microprocessors is a 4-bit microprocessor?
(a) 4004 (b) 8080 (c) 8085 (d) Z80

2.4 Which of the following microprocessors is an 8-bit microprocessor?
(a) 4004 (b) 8080 (c) 8085 (d) Z80

2.5 Which of the following microprocessors has a 16-bit address bus?
(a) 4004 (b) 8088 (c) 8085 (d) 8086

2.6 An 8-bit microprocessor has an
(a) 8-bit data bus (b) 8-bit address bus
(c) 8-bit control bus (d) 8 interrupt lines

2.7 If a microprocessor is capable of addressing 64 kbytes of memory, its address bus width is
(a) 16 bits (b) 20 bits (c) 8 bits (d) none of these

2.8 If a microprocessor is capable of addressing 1MB memory, its address bus width is
(a) 16 bits (b) 20 bits (c) 8 bits (d) none of these

2.9 A microprocessor performs as
(a) CPU of a computer (b) memory of a computer
(c) output device of a computer (d) input device of a computer

2.10 A microprocessor is an
(a) SSI device (b) MSI device (c) LSI device (d) VLSI device

2.11 The program counter in a microprocessor
(a) keeps the address of the next instruction to be fetched
(b) counts the number of instructions being executed on the microprocessor
(c) counts the number of programs being executed on the microprocessor
(d) counts the number of interrupts handled by the microprocessor

2.12 The number of flags of the 8085 microprocessor is
(a) 6 (b) 5 (c) 4 (d) 3

2.13 The word size of the 8085 microprocessor is
(a) 8-bits (b) 16-bits (c) 20-bits (d) 4-bits

2.14 The 8085 microprocessor is a
(a) 40 pin IC (b) 32 pin IC (c) 28 pin IC (d) 24 pin IC

2.15 The address bus of microprocessor is
(a) unidirectional (b) bi-directional
(c) unidirectional as well as bi-directional (d) none of these

2.16 The data bus of microprocessor is
(a) unidirectional (b) bi-directional
(c) unidirectional as well as bi-directional (d) none of these

2.17 Flip-flops are used in a microprocessor to indicate.
(a) Shift register (b) latches (c) country (d) flags

2.18 For using a microprocessor-based system,
(a) a program is required
(b) the program must be stored in memory before the system can be used
(c) the program need to be stored in memory
(d) the program is stored in the internal resistors of the microprocessor.

Answers to Multiple-Choice Questions

2.1 (b)	2.2 (b)	2.3 (a)	2.4 (b) (c)
2.5 (c)	2.6 (a)	2.7 (a)	2.8 (b)
2.9 (a)	2.10 (c)	2.11 (a)	2.12 (b)
2.13 (a)	2.14 (a)	2.15 (a)	2.16 (b)
2.17 (d)	2.18 (b)		

CHAPTER

3

Instruction Set of 8085 Microprocessor

3.1 INTRODUCTION

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

3.2 ADDRESSING MODES

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

- Immediate addressing
- Register addressing
- Direct addressing
- Indirect addressing

3.2.1 Direct Addressing

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

Table 3.1 Direct addressing

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

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

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

3.2.2 Register Addressing

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

Table 3.2 Register addressing

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

3.2.3 Register Indirect Addressing

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

Table 3.3 Register indirect addressing

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

3.2.4 Immediate Addressing

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

Table 3.4 Immediate addressing

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

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

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

3.2.5 Implicit Addressing

The addressing mode of certain instructions can be implied by the instruction's function. Actually, these instructions work on the content of the accumulator and there is no need of the address of the operand. Examples of implicit addressing are given in Table 3.5.

Table 3.5 Implicit addressing

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

3.3 INSTRUCTION SET

An instruction is a command applied to the microprocessor to perform a specific function. The instruction set

of a microprocessor means the entire group of instructions. Generally, instructions have been classified into the following five functional groups.

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

3.3.1 Data Transfer Group

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

Table 3.6 Types of data transfer

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

Examples

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

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

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

3.3.2 Arithmetic Group

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

Addition The contents of a register or the contents of a memory location or any 8-bit number can be added to the contents of the accumulator. After addition, the sum is stored in the accumulator.

Subtraction An 8-bit number or the contents of a register or the contents of a memory location can be subtracted from the contents of the accumulator. After subtraction, the results will be stored in the accumulator.

Increment/Decrement The content of a register or a memory location, 8-bit data can be incremented or decremented by 1. In the same way, the contents of a register pair H–L or B–C or D–E (16 bit data) can be incremented or decremented by 1. The increment and decrement operations can also be performed in a memory location.

Examples

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

3.3.3 Logical Group

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

AND, OR, Exclusive-OR The content of a register or content of a memory location or content of any 8-bit data can be logically ANDed, Ored, or Exclusive-ORed with the contents of the accumulator. Then results must be stored in the accumulator.

Rotate Each bit of the accumulator can be shifted either left or right by one bit.

Compare An 8-bit number or the content of a register or content of a memory location be compared with the contents of the accumulator to check greater than or equal or less than.

Complement The contents of the accumulator can be complemented. Therefore, all 0s are replaced by 1s and all 1s are replaced by 0s.

Examples

ANA	Logical AND with Accumulator
ANI	Logical AND with Accumulator using Immediate Data
ORA	Logical OR with Accumulator
OR	Logical OR with Accumulator using Immediate Data

XRA	Exclusive Logical OR with Accumulator
XRI	Exclusive OR using Immediate Data

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

CMP	Compare
CPI	Compare using Immediate Data

The rotate instructions shift the contents of the accumulator one bit to the left or right:

RLC	Rotate Accumulator Left
RRC	Rotate Accumulator Right
RAL	Rotate Left through Carry
RAR	Rotate Right through Carry

Complement and carry flag instructions are

CMA	Complement Accumulator
CMC	Complement Carry Flag
STC	Set Carry Flag

3.3.4 Branch Control Group

This group includes the instruction changes in the sequence of program execution using conditional and unconditional jumps, subroutine call and return, and restart.

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

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

The *unconditional branch control* instructions are as follows:

JMP	Jump
CALL	Call
RET	Return

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

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

Thus, the conditional branching instructions are specified as follows:

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

3.3.5 Stack, I/O and Machine Control Group

These instructions are performed by various functions related with stack and input/output ports and machine control.

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

PUSH	Push Two bytes of Data onto the Stack
POP	Pop Two Bytes of Data off the Stack
XTHL	Exchange Top of Stack with H and L
SPHL	Move content of H and L to Stack Pointer

The I/O instructions are given below:

IN	Initiate Input Operation
OUT	Initiate Output Operation

The Machine Control instructions are as follows:

EI	Enable Interrupt System
DI	Disable Interrupt System
HLT	Halt
NOP	No Operation

3.4 INSTRUCTION AND DATA FORMATS

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

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

- One-Byte Instructions
- Two-Byte Instructions
- Three-Byte Instructions

3.4.1 One-Byte Instructions

A one-byte instruction consists of the opcode and operand in the same byte. Operand(s) are internal registers and are coded into the instruction. Some examples are given here.

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

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

3.4.2 Two-Byte Instructions

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

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

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

3.4.3 Three-Byte Instructions

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

Op Code	Operand	Binary code	Hex code	Operations
LXI	H, 8000H	0010 0001	21H First byte operation code	Load immediately 8000H in the HL
		0000 0000	00H Second byte data	registers pair
		1000 0000	80H Third byte data	

JMP	8085H	1100 0011	C3H First byte operation code	Jump to the memory location 8085H
		1000 0101	85H Second byte data	
		1000 0000	80H Third byte data	
LDA	8000H	0011 0100	34H First byte operation code	The content of memory location 8000H
		0000 0000	00H Second byte data	is copied to the accumulator
		1000 0000	80H Third byte data	

3.4.4 Symbols and Abbreviations

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

Table 3.7 Symbols and abbreviations of 8085

Symbol/Abbreviations	Meaning
16-bit address	16-bit address of the memory location
data	8-bit data
16-bit data	16-bit data
R, R_d, R_s	One of the registers A, B, C, D, E, H , L
A, B, C, D, H, L	8-bit register
A	Accumulator
H–L	Register pair H–L
B–C	Register pair B–C
D–E	Register pair D–E
RP	One of the register pairs. The representation of a register pair is given below: B represents B–C pair, B is high order register and C low order register D represents D–E pair, D is high order register and E is low order register. H represents H–L pair, H is high order register and L low order register.
Rh	The high order register of a register pair.
Rl	The low order register of a register pair.
SP, SPH, SPL	SP represents 16-bit stack pointer. SPH is high order 8 bits and SPL low order 8 bits of register SP.
PC, PCH, PCL	16-bit program counter. PCH is high order 8 bits and PCL low order 8 bits of register PC.
PSW	Program Status Word
M	Memory whose address is in H–L pair
CS	Carry status
[ ]	The content of the memory location
←	Move data in the direction of arrow
⇔	Exchange contents
∧	Logical AND operation
∨	Logical OR operation
⊕	Logical EXCLUSIVE OR
-	One's complement

3.5 8085 INSTRUCTIONS

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

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

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

3.5.1 Data Transfer Instructions

Data transfer instructions are used to transfer data between registers, register pairs, memory and registers, etc. All data transfer instructions are described below:

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

Rd ← Rs,

Machine cycles: 1, States: 4, Flags none, Register Addressing mode, one byte instruction

This instruction copies the contents of the source register into the destination register but the contents of the source register are not altered. For example, the instruction MOV B, C moves the content of register C to register B. To execute this instruction, 4-clock period are required and flags are not affected.

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

[M] ← Rs,

Machine cycles: 2, States: 7, Flag none, Register indirect Addressing, one byte instruction

The content of the source register moves to the memory location, its location is specified by the contents of the HL registers. For example, the instruction MOV M, B will move the content B register to the memory location 8000H if the content of the HL register pair is 8000H.

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

Rd ← [M],

Machine cycles: 2, States: 7, Flag none, Register indirect addressing, one byte instruction

The content of memory location moves to a register. For example, the instruction MOV C, M will move the content of the memory location 8000H to the C register when the content of the HL register pair is 8000H.

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

Rd ← data,

Machine cycles: 2, States: 7, Flags none, Immediate addressing modes, two-byte instruction

The 8-bit data is stored in the destination register. For example, the instruction MVI B, FFH moves FF to B register. The code of this instruction is 3E, FF. The 1st byte of the instruction is the opcode and the 2nd byte of the instruction is the operand, FFH that is to be moved to the register B.

MVI M, data (Move immediate data to memory)

[M] ← data,

Machine cycles: 3, States: 10, Flags: none, immediate addressing, two-byte instruction

The data will be stored in the memory location, which is specified by the contents of the H-L register pair.

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

LDA 16-bit address (Load Accumulator direct)

A ← [16-bit Address].

Machine cycles: 4, States: 13, Flags: none, Direct addressing: three byte instructions

The content of a memory location, specified by a 16-bit address in the operand, is copied to the accumulator and the contents of the source are not altered.

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

LDAX B/D Register pair (Load Accumulator Indirect)

A ← [BC] or A ← [DE]

Machine cycles: 2, States: 7, Flags: none, Register indirect Addressing, three byte instructions

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

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

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

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

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

\ This instruction loads 16-bit immediate data in the register pair designated in the operand.

Example LXI H, 9050H (16-bits data). In the code form, it can be written as 3A, 50, 90. H ← 90H MSBs and L ← 50H LSBs of data.

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

L ← [address], H ← [address+1].

Machine cycles: 5, States: 16, Flags: none, direct Addressing: three-byte instructions .

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

STA 16-bit address (Store accumulator direct)

16-bit Address ← A,

Machine cycles: 4, States: 13, Flags: none, direct Addressing, three-byte instructions

The content of the accumulator is stored into the memory location specified by the operand. This is a 3-byte instruction. The second byte specifies the low-order address and the third byte specifies the high-order

address. Example is STA 9050H. This instruction stores the content of the accumulator in the memory location 9050H.

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

[address] ← L, [address +1] ← [H].

Machine cycles: 5, States: 16, Flags: none, direct addressing, three-byte instructions

The content of L register is stored into the memory location specified by the 16-bit address in the operand and the content of the H register is also stored into the next memory location by increasing the operand. The contents of registers HL will not be changed. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address. For example, SHLD 8000H the content of register L in the memory location 8000 H. The content of register H is stored in the memory location 8001 H.

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

H ↔ D, L ↔ E

Machine cycles: 1, States: 4, Flags: none, register Addressing.

The content of register H is exchanged with the content of register D, and the content of register L will be exchanged with the content of register E. For example, XCHG

Table 3.8 show the 8085 data transfer instruction set summary.

Table 3.8 8085 Data-transfer instructions set summary

Opcode	Operand	Functions	Clock cycle	Number of bytes	Instruction code							
MOV	Rd, Rs	Move register to register	4	1	0	1	D	D	D	S	S	S
MOV	M, Rs	Move register to memory	7	1	0	1	1	1	0	S	S	S
MOV	Rd, M	Move memory to register	7	1	0	1	D	D	1	1	0	
MVI	Rd, data	Move immediate register	7	2	0	0	D	D	D	1	1	0
MVI	M, data	Move immediate memory	10	2	0	0	1	1	0	1	1	0
LDA	16 bit address	Load A direct	13	3	0	0	1	1	1	0	1	0
LDAX	B	Load A indirect	7	1	0	0	0	0	1	0	1	0
LDAX	D	Load A direct	7	1	0	0	0	1	1	0	1	0
LXI	B	Load immediate register pair B and C	10	3	0	0	0	0	0	0	0	1
LXI	D	Load immediate register pair D and E	10	3	0	0	0	1	0	0	0	1
LXI	H	Load immediate register pair H and L	10	3	0	0	1	0	0	0	0	1
LXI	SP	Load immediate stack pointer	10	3	0	0	1	1	0	0	0	1
LHLD	16 bit address	Load H & L direct	16	3	0	0	1	0	1	0	1	0
STA	16 bit address	Load A direct	13	3	0	0	1	1	0	0	1	0
STAX	B	Store A indirect	7	1	0	0	0	0	0	0	1	0
STAX	D	Store A indirect	7	1	0	0	0	1	0	0	1	0
SHLD		Store H & L direct	16	1	0	0	1	0	0	0	1	0
XCHG		Exchange D & E and H & L registers	4	1	1	1	1	0	1	0	1	1

3.5.2 Arithmetic Instructions

The arithmetic instructions are used to perform arithmetic operations. All arithmetic instructions are explained below:

ADD R (Add register to accumulator)

A ← A + R,

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

The contents of the operand (register) are added to the contents of the accumulator and the result is stored in the accumulator. For example, ADD B.

ADD M (Add memory to accumulator)

A ← A+ [M],

Machine cycles: 2, States: 4, Flags: all, Register indirect Addressing, one-byte instruction

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

ADC R (Add register to accumulator with carry)

A ← A + R+CS,

Machine cycles: 1, States: 7, Flags : all, Register Addressing, one-byte instruction

The contents of the register and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. For example, ADC C.

ADC M (Add register to accumulator with carry)

A ← A + M+CS,

Machine cycles: 2, States: 7, Flags : all, Register indirect Addressing, one-byte instruction

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

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

A ← A + data,

Machine cycles: 2, States: 7, Flags : all, immediate addressing, Two-byte instruction

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

ACI 8-bit data (Add immediate 8-bit data to accumulator with carry)

A ← A + data + CS,

Machine cycles: 2, States: 7, Flags : all, immediate addressing, Two byte instruction

The 8-bit data and the Carry flag are added with the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. For example, the instruction is ACI 80H.

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

H–L ← H–L + Register pair.

Machine cycles: 3, States: 10, Flags. CS, register Addressing, one byte instruction

The 16-bit contents of the specified register pair can be added to the contents of the H-L register pair and the sum is stored in the H and L registers. The contents of the source register pair cannot be modified. When the result is greater than 16 bits, the CY flag will be set and no other flags are affected. For example, DAD B. The instruction DAD B states that the contents of the B-C register pair will be added with the content of the H-L register pair

SUB R (Subtract register from accumulator)

A ← A – R,

Machine cycles: 1, States: 4, Flags: all, Register Addressing, One byte instructions

The content of register R is subtracted from the content of the accumulator, and the result is stored in the accumulator. For example, SUB C.

SUB M (Subtract memory from accumulator)

A ← A – [M]

Machine cycles: 2, States: 7, Flags all, Register indirect addressing, one-byte instructions

The contents of the memory are subtracted from the contents of the accumulator and the result is placed in the accumulator. The memory location is specified by the contents of the H–L register pair. All flags will be modified to reflect the result. For example, SBB M.

SBB R (Subtract register from accumulator with borrow)

A ← A – R – CS.

Machine cycles: 2, States: 4, Flags all, register addressing, one-byte instructions

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

SBB M (Subtract memory and borrow from accumulator)

A ← A – [M] – CS.

Machine cycles: 2, States: 7, Flags : all, register indirect addressing, one-byte instructions

The content of the memory location specified by H–L pair and borrow flag are subtracted from the content of the accumulator. The result is placed in the accumulator. For example, SBB M.

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

A ← A – 8-bit data.

Machine cycles: 2, States: 7, Flags: all, immediate addressing, two-byte instructions.

The 8-bit data is subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are affected to reflect the result of the subtraction. For example, SUI 34H. This instruction will subtract 34 from the content of the accumulator and store the result in the accumulator.

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

A ← A – 8-bit data -CS.

Machine cycles: 2, States: 7, Flags: all, immediate addressing, two-byte instructions.

The 8-bit data and the borrow flag is subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are affected to reflect the result of the subtraction. For example, SBI 34H. This instruction will subtract 34 and the borrow flag from the content of the accumulator and store the result in the accumulator.

INR R (Increment register by 1)

R ← R + 1

Machine cycles: 1, States: 4, Flags all except carry flag, Register Addressing, one-byte instructions.

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

INR M (Increment memory by 1)

[M] ← [M] +1

Machine cycles: 3, States: 10, Flags: all except carry flag, Register indirect addressing, one-byte instructions.

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

INX RP (Increment register pair)

RP ← RP+1

Machine cycles: 1, States: 6, Flags are not effected, Register Addressing, one-byte instructions

The content of the specified register pair is incremented by one and result will be stored in the same register pair. For example, INX H.

DCR R (Decrement register by 1)

R ← R – 1

Machine cycles: 1, States: 4, Flags: all flags except carry flag, register addressing, one-byte instruction

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

DCR M (Decrement memory by 1)

[M] ← [M] – 1

Machine cycles: 3, States:10, Flags: all except carry flag, Register indirect addressing, one byte instruction

The content of the memory location addressed by H–L pair is decremented by one. For example, DCR M.

DCX RP (Decrement register pair by 1)

RP ← RP-1,

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

The contents of the specified register pair are decremented by 1 and the result is stored in the same place. For example, DCX D

DAA (Decimal adjust accumulator)

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

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

When the value of the low-order 4-bits in the accumulator is greater than 9 or AC flag is set, the instruction adds 6 to the low-order four bits. If the value of the high-order 4-bits in the accumulator is greater than 9 or the Carry flag is set, this instruction adds 6 to the high-order four bits. In this instruction S, Z, AC, P, CY flags are altered to reflect the results of the operation. For example, DAA.

Table 3.9 shows the 8085 Arithmetic Instruction Set Summary.

Table 3.9 8085 arithmetic instruction set summary

Opcode	Operand	Functions	Clock cycle	Number of bytes	Instruction code							
ADD	R	Add register to A	4	1	1	0	0	0	0	S	S	S
ADD	M	Add memory to A	7	1	1	0	0	0	0	1	1	0
ADC	R	Add register to A with carry	4	1	1	0	0	0	1	S	S	S
ADC	M	Add memory to A with carry	7	1	1	0	0	0	1	1	1	0
ADI	8-bit data	Add immediate to A	7	2	1	1	0	0	0	1	1	0
ACI	8-bit data	Add immediate to A with carry	7	2	1	1	0	0	1	1	1	0
DAD	B	Add B & C to H & L	10	1	0	0	0	0	1	0	0	1
DAD	D	Add D & E to H & L	10	1	0	0	0	1	1	0	0	1
DAD	H	Add H & L to H & L	10	1	0	0	1	0	1	0	0	1
DAD	SP	Add stack pointer to H & L	10	1	0	0	1	1	1	0	0	1
SUB	R	Subtract register from A	4	1	1	0	0	1	0	S	S	S
SUB	M	Subtract memory from A	7	1	1	0	0	1	0	1	1	0
SBB	R	Subtract register from A with borrow	4	1	1	0	0	1	1	S	S	S
SBB	M	Subtract memory from A with borrow	7	1	1	0	0	1	1	1	1	0
SUI	8-bit data	Subtract immediate from A	7	2	1	1	0	1	0	1	1	0
SBI	8-bit data	Subtract immediate from A with borrow	7	2	1	1	0	1	1	1	1	0
INR	R	Increment register	4	1	0	0	D	D	D	1	0	0
INR	M	Increment memory	10	1	0	0	1	1	0	1	0	0
INX	B	Increment B & C registers	6	1	0	0	0	0	0	0	1	1
INX	D	Increment D & E registers	6	1	0	0	0	1	0	0	1	1
INX	H	Increment H & L registers	6	1	0	0	1	0	0	0	1	1
INX	SP	Increment stack pointer	6	1	0	0	1	1	0	0	1	1
DCR	R	Decrement register	4	1	0	0	D	D	D	1	0	1
DCR	M	Decrement memory	10	1	0	0	1	1	0	1	0	1
DCX	B	Decrement B & C registers	6	1	0	0	0	0	1	0	1	1
DCX	D	Decrement D & E registers	6	1	0	0	0	1	1	0	1	1
DCX	H	Decrement H & L registers	6	1	0	0	1	0	1	0	1	1
DCX	SP	Decrement stack pointer	6	1	0	0	1	1	1	0	1	1
DAA		Decimal adjustment	4	1	0	0	1	0	0	1	1	1\

3.5.3 Logical Instructions

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

CMP R (Compare register with accumulator)

A – R.

Machine cycles: 1, States 4, Flags all, Register Addressing, One-byte instructions

The contents of the register are compared with the contents of the accumulator. Both contents are conserved. The result of the comparison can be reflected by setting the flags in PSW. When A < register, carry flag is set. If A = register, zero flag is set. While A > register, carry and zero flags are reset. For example, CMP B.

CMP M (Compare memory with accumulator)

A – [M]

Machine cycles: 2, States 7, Flags all, Register indirect addressing, One-byte instructions

The content of the memory location specified by H – L pair is compared with the content of the accumulator, and status flags are set according to the result of the comparison. The content of the accumulator remains uncharged. For example, CMP M.

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

A – 8-bit data.

Machine cycles: 2, States: 7, Flags: all, immediate addressing, Two-byte instructions

The second byte of the instruction or 8-bit data is compared with the contents of the accumulator. The values being compared remain unchanged. The result of the comparison will be reflected by setting the flags of the PSW. If A < 8-bit data, carry flag is set. If A = data, zero flag is set. When A > 8-bit data, carry and zero flags are reset. For example, the instruction is CPI 46H.

ANA R (Logical AND register with accumulator)

$A \leftarrow A \wedge R$

Machine cycles: 1, States: 4, Flags: all, Register indirect addressing, One-byte instructions

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

ANA M (Logical AND memory with accumulator)

$A \leftarrow A \wedge [M]$.

Machine cycles: 2, States: 7, Flags: all, register indirect addressing, One-byte instructions

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

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

$A \leftarrow A \wedge$ 8-bit data

Machine cycles: 2, States: 7, Flags: all, Register indirect addressing, One-byte instructions

The contents of the accumulator are logically ANDed with the 8-bit data. After ANDing the result is stored in the accumulator. All flags are modified to reflect the result of the operation. The CY flag is reset and AC flag is set. For example, ANI 24H.

ORA R (Logical OR register with accumulator)

$A \leftarrow A \vee R$

Machine cycles: 1, States: 4. Flags: all, Register addressing, One-byte instructions

The content of register R is logically ORed with the content of the accumulator. The result is stored in the accumulator. All flags are modified to reflect the result of the operation. Carry flag (CY) and auxiliary carry (AC) are reset.

ORA M (Logical OR memory with accumulator)

$A \leftarrow A \vee [M]$.

Machine cycles: 2, States: 7, Flags: all, Register addressing, One-byte instructions

The contents of the accumulator are logically ORed with the contents of the memory location, whose address

is specified by the contents of H and L registers and the result is placed in the accumulator. All flags are modified to reflect the result of the operation. CY and AC are reset. The example is ORA M.

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

$A \leftarrow A \vee 8\text{-bit data}$.

Machine cycles: 2, States: 7, Flags: all, Immediate addressing,Two-byte instructions

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

XRA R (EXCLUSIVE-OR register with accumulator)

$A \leftarrow A \oplus R$

Machine cycles: 1, States: 4, Flags: all, Register addressing, One-byte instructions

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

XRA M (EXCLUSIVE – OR memory with accumulator)

$A \leftarrow A \oplus [M]$

Machine cycles: 2, States: 4, Flags: all, Register Addressing, One-byte instructions

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

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

$A \leftarrow A \oplus \text{data}$

Machine cycles: 2, States: 7, Flags: all, immediate Addressing, One-byte instructions

The contents of the accumulator are Exclusive ORed with the 8-bit data. The result is stored in the accumulator. Flags are modified to reflect the result of the operation. The CY and AC flag become 1. For example, XRI 86H.

RLC (Rotate accumulator left)

$A_{n+1} \leftarrow A_n$, $A_0 \leftarrow A_7$, $C_S \leftarrow A_7$

Machine cycles: 1, States: 4, Flags: CS, Addressing: implicit, One-byte instructions

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

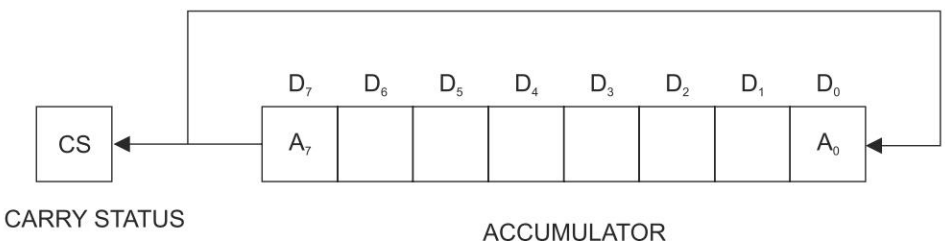


Fig. 3.1 Diagram for RLC

RRC (Rotate accumulator right)

$A_7 \leftarrow A_0$, CS $\leftarrow A_0$, $A_n \leftarrow A_{n+1}$

Machine cycles: 1, States: 4, Flags: CS, implicit Addressing, One-byte instructions

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

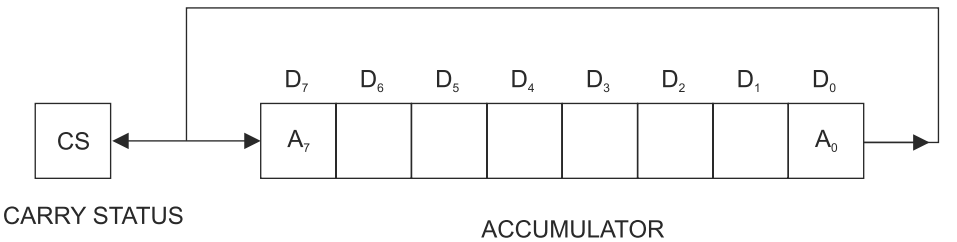


Fig. 3.2 Diagram for RRC

RAL (Rotate accumulator left through carry)

$A_{n+1} \leftarrow A_n$, CS $\leftarrow A_7$, $A_0 \leftarrow$ CS

Machine cycles: 1, States: 4, Flags: CS, Implicit Addressing , One-byte instructions

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

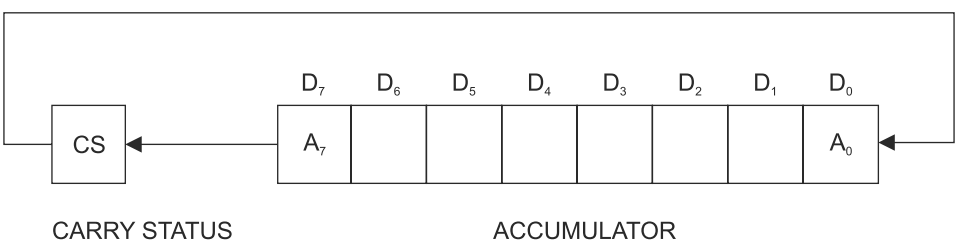


Fig. 3.3 Diagram for RAL

RAR (Rotate accumulator right through carry)

An $\leftarrow$ An+1, CS $\leftarrow A_0$, $A_7 \leftarrow$ CS

Machine cycles: 1, States: 4, Flags: CS, Addressing implicit, One-byte instructions

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

CMA (Complement the accumulator)

$A \leftarrow \overline{A}$

Machine cycles: 1, States: 4, Flags: none, Implicit Addressing, One-byte instruction

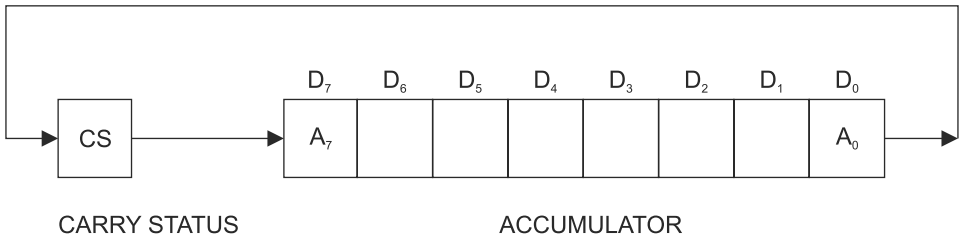


Fig. 3.4 Diagram for RAR

The content of the accumulator is complemented, and the result is placed in the accumulator. No flags are affected. For example, CMA determines one's complement of 0000 1100 is 1111 0011. Assume A = 00001100.

CMC (Complement the carry)

CS ← $\overline{CS}$ Machine cycles: 1, States: 4, Flags: CS, One-byte instruction

The carry flag CS is complemented. No other flags are affected. For example, CMC.

STC (Set the carry)

CS ← 1 Machine cycles: 1, States: 4, Flags: CS, One-byte instruction

The carry flag is set to 1. No other flags are affected. For example, STC.

Table 3.10 shows the 8085 logical instruction set summary.

Table 3.10 8085 logical instruction set summary

Opcode	Operand	Functions	Clock cycle	Number of bytes	Instruction code							
CMP	R	Compare register with A	4	1	1	0	1	1	1	S	S	S
CMP	M	Compare memory with A	7	1	1	0	0	0	0	1	1	0
CPI	8 bit data	Compare immediate with A	7	2	1	1	1	1	1	1	1	0
ANA	R	AND register with A	4	1	1	0	1	0	0	S	S	S
ANA	M	AND memory with A	7	1	1	0	1	0	0	1	1	0
ANI	8 bit data	AND immediate with A	7	2	1	1	1	0	0	1	1	0
ORA	R	OR register with A	4	1	1	0	1	1	0	S	S	S
ORA	M	OR memory with A	7	1	1	0	1	1	0	1	1	0
ORI	8 bit data	OR immediate with A	7	2	1	1	1	1	0	1	1	0
XRA	R	Exclusive OR register with A	4	1	1	0	1	0	1	S	S	S
XRA	M	Exclusive OR memory with A	7	1	1	0	1	0	1	1	1	0
XRI	8 bit data	Exclusive OR immediate with A	7	2	1	1	1	0	1	1	1	0
RLC		Rotate A left	4	1	0	0	0	0	0	1	1	1
RRC		Rotate A right	4	1	0	0	0	0	1	1	1	1
RAL		Rotate A left with carry	4	1	0	0	0	1	0	1	1	1
RAR		Rotate A right with carry	4	1	0	0	0	1	1	1	1	1
CMA		Complement A	4	1	0	0	1	0	1	1	1	1
CMC		Complement carry	4	1	0	0	1	1	1	1	1	1
STC		Set carry	4	1	0	0	1	1	0	1	1	1

3.5.4 Branch Group

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

JMP 16-bit address (Jump Unconditionally)

PC ← Label (16-bit address)

Machine cycles: 3, States: 10, Flags: none, Immediate Addressing.

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

Conditional Jump 16-bit address (Jump Conditionally)

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

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

JC 16-bit address (Jump on carry)

PC ← 16-bit address, jump if CY = 0.

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

The program jumps to the memory location specified by the 16-bit address when the carry flag CY = 1. For example, JC 9000H.

JNC 16-bit address (Jump on no carry)

PC ← 16-bit address, jump if CY = 0.

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

The program is transferred to the memory location specified by the 16-bit address when there is no carry. For example, JNC 9000H.

JP 16-bit address (Jump on positive)

PC ← 16-bit address, jump is S = 0.

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

The program jumps to the memory location specified by the 16-bit address if the result is plus or positive. For example, JP 8000H.

JM 16-bit address (Jump on minus)

PC ← 16-bit address jump if S = 1.

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

When the result is minus, the program jumps to the memory location specified by the 16-bit address. For example, JM 9060H.

JZ 16-bit address (Jump on zero)

PC ← 16-bit address jump if Z=1.

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

The program jumps to the memory location specified by the 16-bit address while the result is zero or the zero flag is set. For example, JZ 9500H.

JNZ 16-bit address (Jump on no zero)

PC ← 16-bit address jump if CS = 1.

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

The program jumps to the memory location specified by the 16-bit when the result is not zero or the zero flag is reset. For example, JNZ 9500H.

JPE 16-bit address (Jump on even parity)

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

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

When the result contains even number of 1s, the program jumps to the memory location specified by the 16-bit address. For example, JPE 85000H.

JPO 16-bit address (Jump on odd parity)

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

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

The program jumps to the memory location specified by the 16-bit address when the result contains odd number of 1s. For example, JPO 8000H.

CALL 16-bit address (Unconditional subroutine CALL)

([SP] – 1) ← PCH , ([SP]-2) ← PCL,

[SP] ← [SP]-2, PC ← 16 bit address

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

The program sequence is transferred to the memory location specified by the 16-bit address given in the instruction. Before the transfer, the contents of the program counter (the address of the next instruction after CALL) are pushed onto the stack. For example, CALL 8700H.

CALL 16-bit address (CALL Conditionally)

([SP] – 1) ← PCH , ([SP]-2) ← PCL,

[SP] ← [SP]-2, PC ← 16 bit address

Machine cycles: 2/5, States: 9/18, Flags: none, Immediate/register.

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

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

RET (Return from subroutine unconditionally)

PCL ← [SP]

PCH ← [SP]+1

[SP] ← [SP]+2

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

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

Conditional Return (Return from subroutine conditionally)

PCL ← [SP]

PCH ← [SP]+1

[SP] ← [SP]+2

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

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

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

PCHL (Load program counter with H–L contents)

PC ← H–L, PCH ← H, PCL ← L

Machine cycles: 1, States: 6, Flags: none, Register Addressing.

The contents of H and L registers are transferred into program counter. The contents of register H are placed as the high-order byte of PC register and the contents of register L as the low order byte. For example, PCHL.

RST 0-7 (Restart)

[SP] – 1 ←PCH, [SP] – 2←PCL

[SP] ←[SP] – 2], [PC] ←8 times n

Machine cycles: 3, States: 12, Flags: none, Register indirect Addressing,

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

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

Interrupt Instructions The 8085 microprocessor has four additional interrupts. The interrupt instructions and their Restart addresses are

Interrupt	Restart Address
TRAP	0024H
RST 5.5	002CH
RST 6.5	0034H
RST 7.5	003CH

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

Table 3.11 shows the 8085 Jump instruction Set summary. The 8085 CALL and Return instruction set summary is given in Table 3.12

Table 3.11 8085 JUMP instruction set summary

Opcode	Operand	Functions	Clock cycle	Number of bytes	Instruction code							
JMP	16-bit address	Jump unconditional	10	3	1	1	0	0	0	0	1	1
JC	16-bit address	Jump on carry	7/10	3	1	1	0	1	1	0	1	0
JNC	16-bit address	Jump on no carry	7/10	3	1	1	0	1	0	0	1	0
JP	16-bit address	Jump on positive	7/10	3	1	1	1	1	0	0	1	0
JM	16-bit address	Jump on minus	7/10	3	1	1	1	1	1	0	1	0
JZ	16-bit address	Jump on zero	7/10	3	1	1	0	0	1	0	1	0
JNZ	16-bit address	Jump on no zero	7/10	3	1	1	0	0	0	0	1	0
JPE	16-bit address	Jump on parity even	7/10	3	1	1	1	0	1	0	1	0
JPO	16-bit address	Jump on parity odd	7/10	3	1	1	1	0	0	0	1	0

Table 3.12 8085 CALL and Return instruction set summary

Opcode	Operand	Functions	Clock cycle	Number of bytes	Instruction code							
CALL	16-bit address	Call unconditional	18	3	1	1	0	0	1	1	0	1
CC	16-bit address	Call on carry	9/18	3	1	1	0	1	1	1	0	0
CNC	16-bit address	Call on no carry	9/18	3	1	1	0	1	0	1	0	0
CP	16-bit address	Call on positive	9/18	3	1	1	1	1	0	1	0	0
CM	16-bit address	Call on minus	9/18	3	1	1	1	1	1	1	0	0
CZ	16-bit address	Call on zero	9/18	3	1	1	0	0	1	1	0	0
CNZ	16-bit address	Call on no zero	9/18	3	1	1	0	0	0	1	0	0
CPE	16-bit address	Call on parity even	9/18	3	1	1	1	0	1	1	0	0
CPO	16-bit address	Call on parity odd	9/18	3	1	1	1	0	0	1	0	0
RET		Return unconditional	10	1	1	1	0	0	1	0	0	1
RC		Return on Carry	6/12	1	1	1	0	1	1	0	0	0
RNC		Return on no Carry	6/12	1	1	1	0	1	0	0	0	0
RP		Return on positive	6/12	1	1	1	1	1	0	0	0	0
RM		Return on minus	6/12	1	1	1	1	1	1	0	0	0
RZ		Return on zero	6/12	1	1	1	0	0	1	0	0	0
RNZ		Return on no zero	6/12	1	1	1	0	0	0	0	0	0
RPE		Return on parity even	6/12	1	1	1	1	0	1	0	0	0
RPO		Return on parity odd	6/12	1	1	1	1	0	0	0	0	0

3.5.5 Stack/PUSH and POP Instructions

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

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

[SP] – 1 ←B

[SP] –2 ←C,

[SP]←[SP] –2

Machine cycles: 3, States: 12, Flags: none, Register indirect addressing

The contents of the register pair BC are copied onto the stack in the following sequence. The stack pointer register is decremented and the contents of the high-order register B are copied into that location. Then stack pointer register is decremented again and the contents of the low-order register C are copied to that location.

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

[SP] – 1 ←D

[SP] –2 ← E,

[SP]←[SP] –2

Machine cycles: 3, States: 12, Flags none, Register indirect addressing.

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

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

[SP] – 1 ←H

[SP] –2 ← L

[SP]←[SP] –2

Machine cycles: 3, States: 12. Flags none, Register indirect addressing.

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

PUSH PSW (PUSH accumulator content and flags on stack)

[SP] – 1 ←A

[SP]– 2←PSW (Program Status Word)

[SP] ←[SP] –2

Machine cycles: 3, States: 12, Flags: none, register indirect addressing,

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

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

C←[SP]

B←[SP] +1

[SP] ←[SP] +2

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

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

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

E←[SP]

D←[SP] + 1

[SP] ←[SP] + 2

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

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

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

L←[SP]

H←[SP] +1

[SP]←[SP] +2

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

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

POP PSW (Pop off stack to accumulator and flags)

PSW←[SP]

A←[SP]+1

[SP]←[SP]+2

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

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

XTHL (Exchange H and L with top of stack)

L→[SP]

H↔[SP]+1

Machine cycles: 5, States: 16, Flags: none, Register indirect addressing.

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

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

[H–L]→[SP].

Machine cycles: 1, States: 6, Flags none, Addressing: register.

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

The 8085 stack/PUSH and POP instruction set summary is depicted in Table 3.13

Table 3.13 8085 Stack/PUSH and POP instructions set summary

Opcode	Operand	Functions	Clock cycle	Number of bytes	Instruction code							
PUSH	B	Push register pair Band C on stack	12	1	1	1	0	0	0	1	0	1
PUSH	D	Push register pair D and E on stack	12	1	1	1	0	1	0	1	0	1
PUSH	H	Push register pair H and L on stack	12	1	1	1	1	0	0	1	0	1
PUSH	PSW	Push accumulator A and Flags on stack	12	1	1	1	1	1	0	1	0	1
POP	B	Pop register pair Band C off stack	10	1	1	1	0	0	0	0	0	1
POP	D	Pop register pair D and E off stack	10	1	1	1	0	1	0	0	0	1
POP	H	Pop register pair H and L off stack	10	1	1	1	1	0	0	0	0	1
POP	PSW	Pop accumulator A and Flags off stack	10	1	1	1	1	1	0	0	0	1
XTHL		Exchange top of stack H and L	16	1	1	1	1	0	0	0	1	1
SPHL		H and L to stack pointer	6	1	1	1	1	1	1	0	0	1

3.5.6 I/O and Machine Control Instructions

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

EI (Enable Interrupts)

Machine cycle: 1, States: 4, Flags: none.

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

DI (Disable Interrupts)

Machine cycle: 1, States: 4, Flags: none

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

NOP (No operations)

Machine cycle: 1, States: 4, Flags: none.

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

For example, NOP.

HLT (Halt and enter wait state)

Machine cycle: 1, States: 5, Flags: none.

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

SIM (Set Interrupt Mask)

Machine cycle: 1, States: 4, Flags: none

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

RIM (Read Interrupt Mask)

Machine cycle: 1, States: 4, Flags: none.

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

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

A ← [Port]

Machine cycle: 3, States: 10, Flags: none, Direct addressing.

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

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

[Port] ← A

Machine cycles: 3, States: 10. Flags: none, Direct addressing.

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

Table 3.14 shows the 8085 I/o and machine control instruction set summary.

Table 3.14 8085 I/O and machine control instructions set summary

Opcode	Operand	Functions	Clock cycle	Number of bytes	Instruction code
EI		Enable interrupts	4	1	1 1 1 1 1 0 1 1
DI		Disable interrupts	4	1	1 1 1 1 0 0 1 1
NOP		No-operation	4	1	0 0 0 0 0 0 0 0

HLT	Halt(Power Down)	5	1	0	1	1	1	0	1	1	0
RIM	Read interrupt mask	4	1	0	0	1	0	0	0	0	0
SIM	Set interrupt musk	4	1	0	0	1	1	0	0	0	0
IN	Input	10	1	1	1	0	1	1	0	1	1
OUT	output	10	1	1	1	0	1	0	0	1	1

3.6 INSTRUCTION TIMING DIAGRAM

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

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

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

T–State This is defined as one subdivision of the operation performed in one clock period. Each subdivision is considered as internal state of operation, which is synchronised with the system clock.

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

$$IC = FC + EC$$

3.6.1 Fetch Operation

The first byte of an instruction is known as its opcode. An instruction may be one byte or two bytes or three bytes long. When an instruction is more than one byte, the other bytes are data or operand. In the Program Counter (PC), the memory address of the next instruction to be executed is stored. At the starting of a fetch cycle, the content of the program counter, which is the address of the memory location where the opcode is stored, will be sent to the memory. Then the memory gets the opcode from memory location on the data bus. To perform this operation, two consecutive clock pulses are required. In the next clock cycle, data will be transferred to the microprocessor. To complete the entire fetching operation, four clock cycles are required as depicted in Fig. 3.6. In slow memory, the microprocessor has to wait till the memory sends the opcode. The clock cycle for which the CPU waits is known as *wait cycle*. Therefore, more than four clock cycles are required for opcode fetch operation in case of a slow memory system.

3.6.2 Execute Operation

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

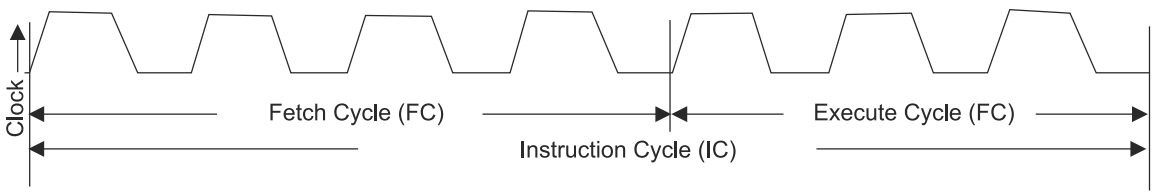


Fig. 3.5 Instruction cycle

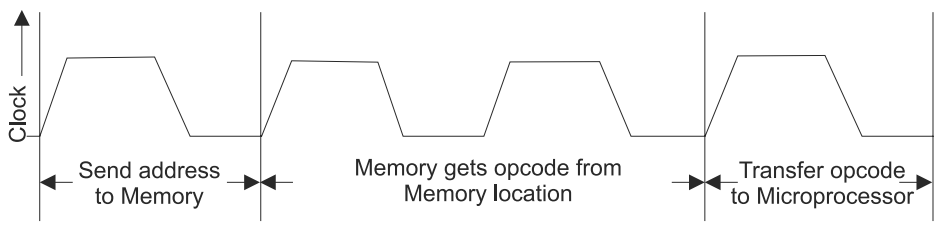


Fig. 3.6 Fetch cycle

3.6.3 Machine Cycle

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

Opcode fetch, memory read, memory write, I/O read and I/O write operations. In other words, the operation of accessing either memory or I/O device is called a *machine cycle*. In the 8085 microprocessor, the machine cycle consists of three to six clock cycles. An instruction cycle consists of several machine cycles. In the first machine cycle of an instruction cycle, the opcode of an instruction is fetched. The 1-byte instructions require only one machine cycle to fetch the opcode and execute the instruction. Two-byte and three-byte instructions require more than one machine cycle. The additional machine cycles are required to read data from memory or I/O devices or to write data into the memory or I/O devices. For example, instruction cycle for MVI B, data is depicted in Fig. 3.7. This instruction has two machine cycles. The first machine cycle (M_1) for fetching opcode, and the other machine cycle (M_2) for reading data from the memory and execute the instruction.

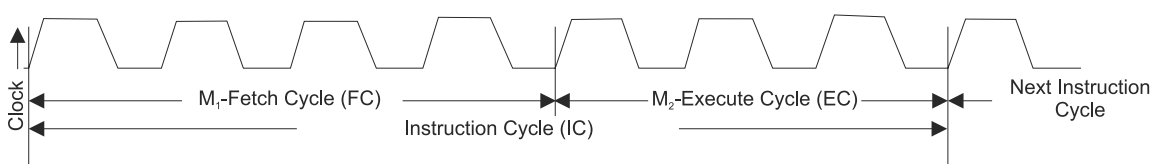


Fig. 3.7 Instruction cycle

3.7 TIMING DIAGRAM

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

3.7.1 Timing Diagram of a Fetch Cycle

In a fetch cycle, the microprocessor fetches the opcode of an instruction from the memory. Figure 3.8 shows the timing diagram for an opcode fetch cycle of an instruction MOV A, B. Assume that the opcode of instruction MOV A, B is stored in 8000H and the content of register B is 4FH.

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

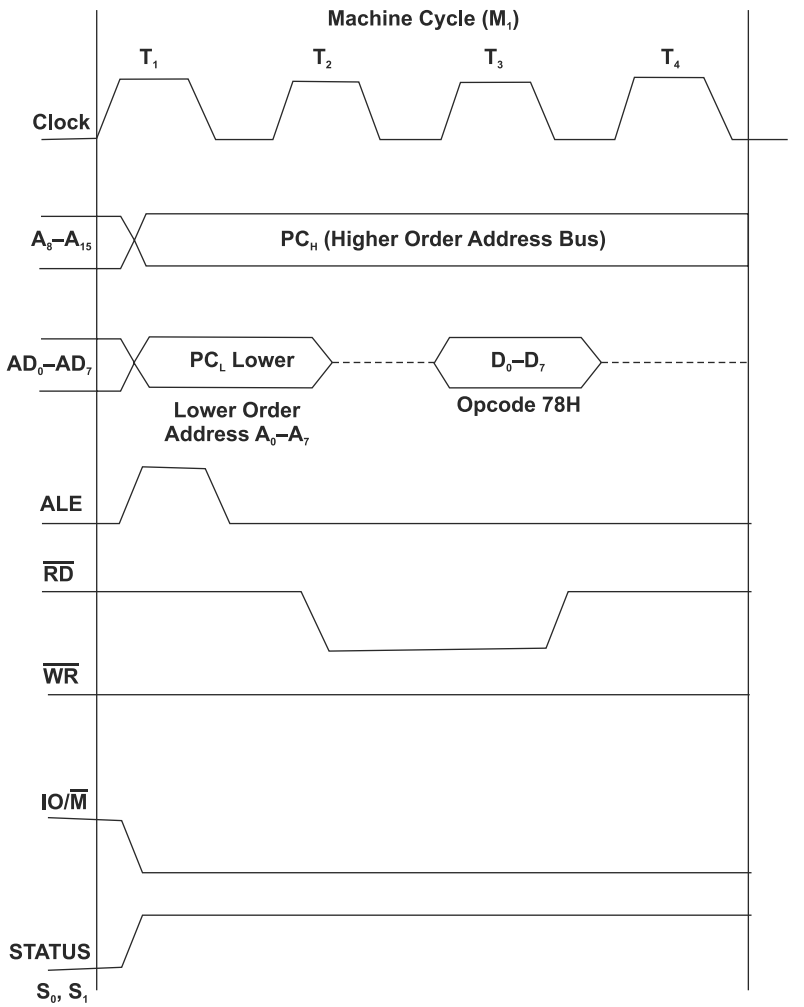


Fig. 3.8 Timing diagram of opcode fetch operation (MOV A, B)

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

First Clock Cycle

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

Second Clock Cycle During T_2, the low-order bus AD_7–AD_0 is ready to carry data from memory location. The microprocessor sends the control signal $\overline{RD} = 0$ to enable memory and the program counter is incremented by 1 to 8001H. Now the opcode from the specified memory location, 8000H places on the data bus.

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

Fourth Clock Cycle The microprocessor decodes the instruction opcode in T_4. It also places the content of B register in the temporary register. After that, it transfers to the accumulator.

3.7.2 Timing Diagram of Memory Read

When an instruction is one byte long, only one machine cycle is required as depicted in Fig. 3.8. For example, MOV, SUB, ADD, RAL are one-byte long instructions. The operands are in the general-purpose registers; the decoding of the operation code and its execution takes only one clock cycle, T_4. When an instruction is two or three bytes long, more than one machine cycle are required. In the first machine cycle, M_1 the opcode is fetched from the memory. The subsequent machine cycles M_2, and M_3 are required to read the operand from the memory or I/O devices or to write data into the memory or I/O devices. The timing diagram for a two-byte instruction MVI C, data is illustrated in Fig. 3.9.

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

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

This instruction requires two machine cycles, M_1 and M_2. The first machine cycle M_1 is known as the fetch cycle to fetch the operation code 3E from the memory. The timing diagram for opcode fetch operation has already been explained in Section 3.7.1. The second machine cycle M_2 is used to read the operand (FFH) from the memory. Actually this is a memory read cycle. Figure 3.10 shows the machine cycle M_2 and its operation is explained below.

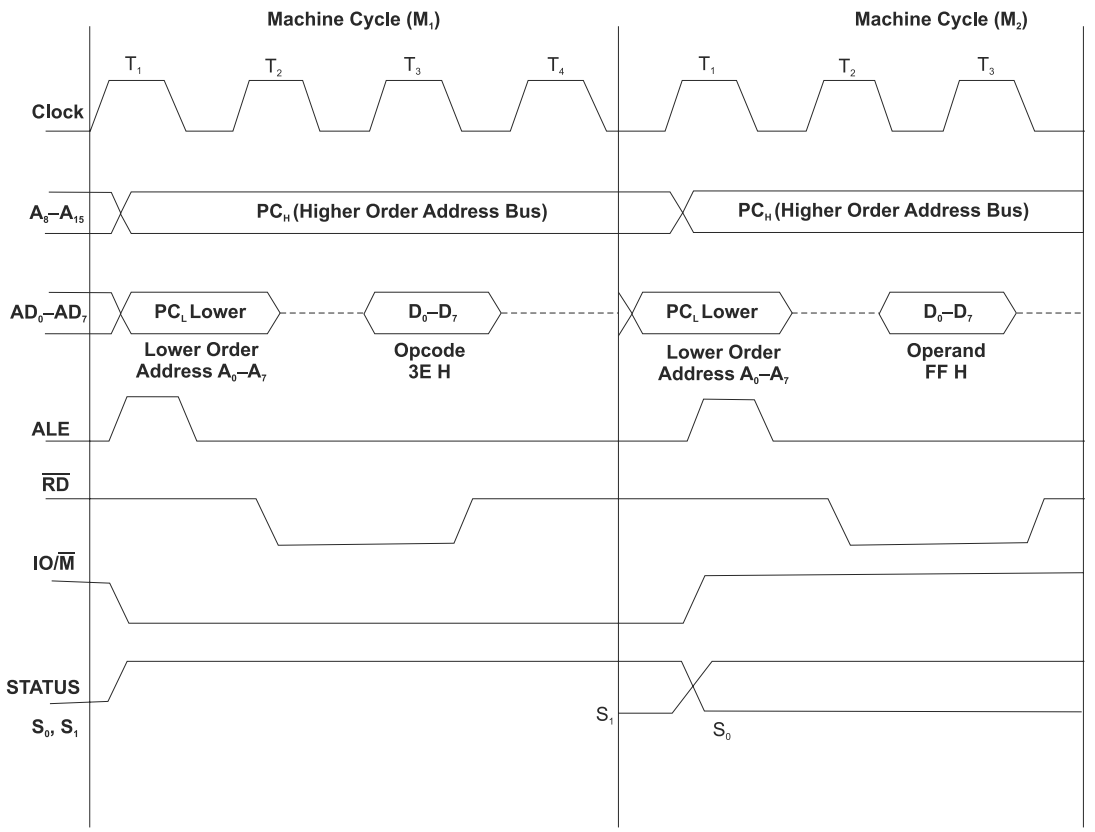


Fig. 3.9 Timing diagram of MVI R, data (MVI C, FFH)

First Clock Cycle of M_2

- In the first clock cycle (T_1) the microprocessor places the content of program counter, 8001H, which is the address of operand on the 16-bit address bus. The 8 MSBs of the memory address, 80H are placed on the high-order address bus, A_{15}–A_8 and 8 LSBs of the memory address, 01H are placed on the low-order address bus, AD_7–AD_0.
- The microprocessor sends an address latch enable (ALE) signal to go high and latch the 8 LSBs of the memory address. Then low-order address bus is demultiplexed and the complete 16-bit memory address is available in the subsequent clock cycles to get the operand from memory location, 8001H.
- The status signals IO/$\overline{M}$ = 0, S_0 = 0 and S_1 = 1 to identify the memory read operation.

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

Third Clock Cycle of M_2

- During T_3, the microprocessor reads the operand. $\overline{RD}$ becomes high during T_3 and the memory is disabled.
- The microprocessor also places the operand in the C register.

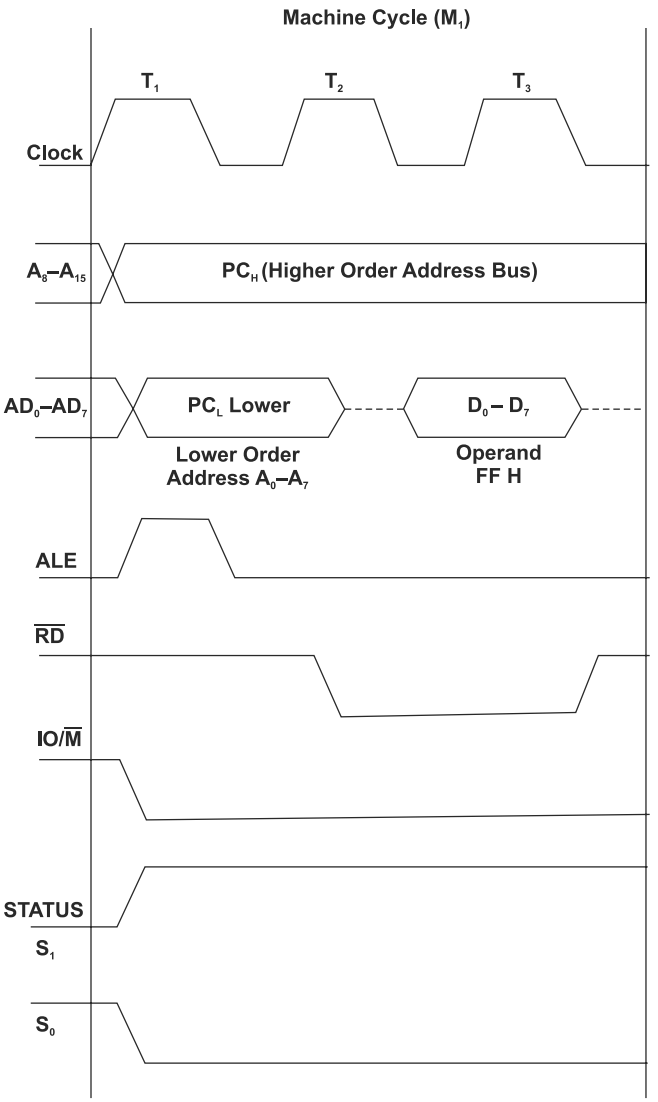


Fig. 3.10 Timing diagram of memory read operation

3.7.3 Timing Diagram of I/O Read

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

The timing diagram of I/O read operation is shown in Fig. 3.11. In this case, the address on the A-bus is for an input device. As I/O device or I/O port addresses is only 8 bits long, the address of I/O device or I/O port is duplicated on both high-order address bus A_8–A_{15} and low-order address bus AD_0–AD_7.

For I/O read operation, the IN instruction is used. One example is IN 00. This is two-byte instruction. The

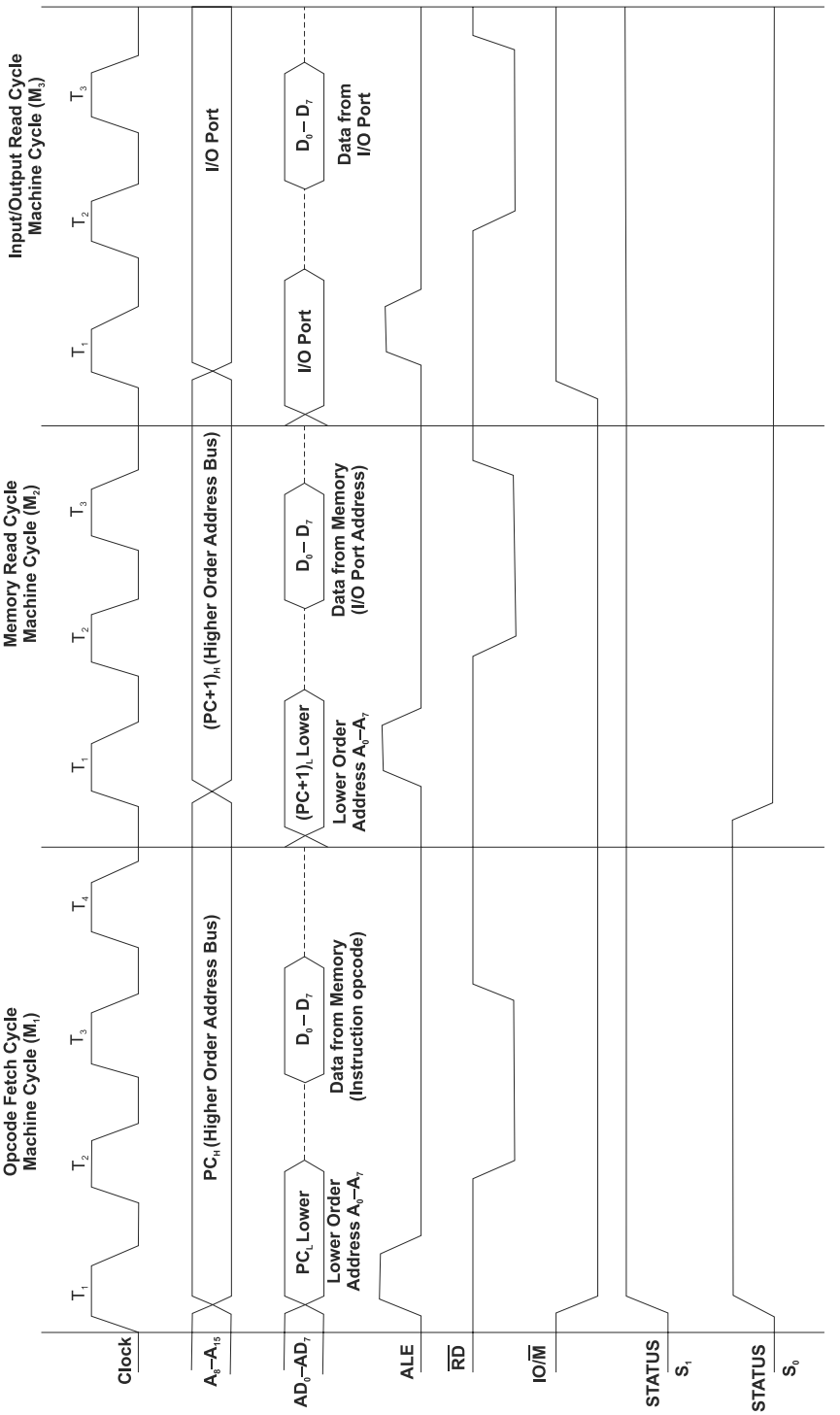


Fig. 3.11 Timing diagram of IN Port Address (IN 00H)

code of this instruction is DB, 00 where DB is for IN and 00 is the input port address.

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

This instruction requires three machine cycles for execution. The first machine cycle is opcode fetch cycle, and the second machine cycle is a memory read cycle to read the address of input device or input port. In the third machine cycle, I/O read operation is performed means the data to be read from the input device or input port. After execution of this instruction, the data is placed in the accumulator. The opcode fetch cycle and memory read cycle are exactly similar to MVI C, FF H instruction. Figure 3.12 shows the machine cycle M_3 of I/O read operation and it is explained below.

T_1 State of M_3

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

T_2 State of M_3

- $\overline{RD}$ is low for read operation

T_3 State of M_3

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

3.7.4 Timing Diagram of Memory Write

In a memory write operation, the microprocessor sends data from the accumulator or any general-purpose register to the memory. The timing diagrams of a memory write cycle is depicted in Fig. 3.13. The memory writes cycle is similar with memory read cycle, but there are differences on status signals. The status signals $S_0 = 1$ and $S_1 = 0$ and write $\overline{WR}$ is low during T_2 of machine cycle M_2 which indicates that the memory write operation is to be performed.

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

3.7.5 Timing Diagram of I/O Write

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

The OUT instruction is used for I/O write operation. This is a two-byte instruction and it requires three machine cycles as depicted in Fig. 3.13. The first machine cycle is for opcode fetch operation and the second

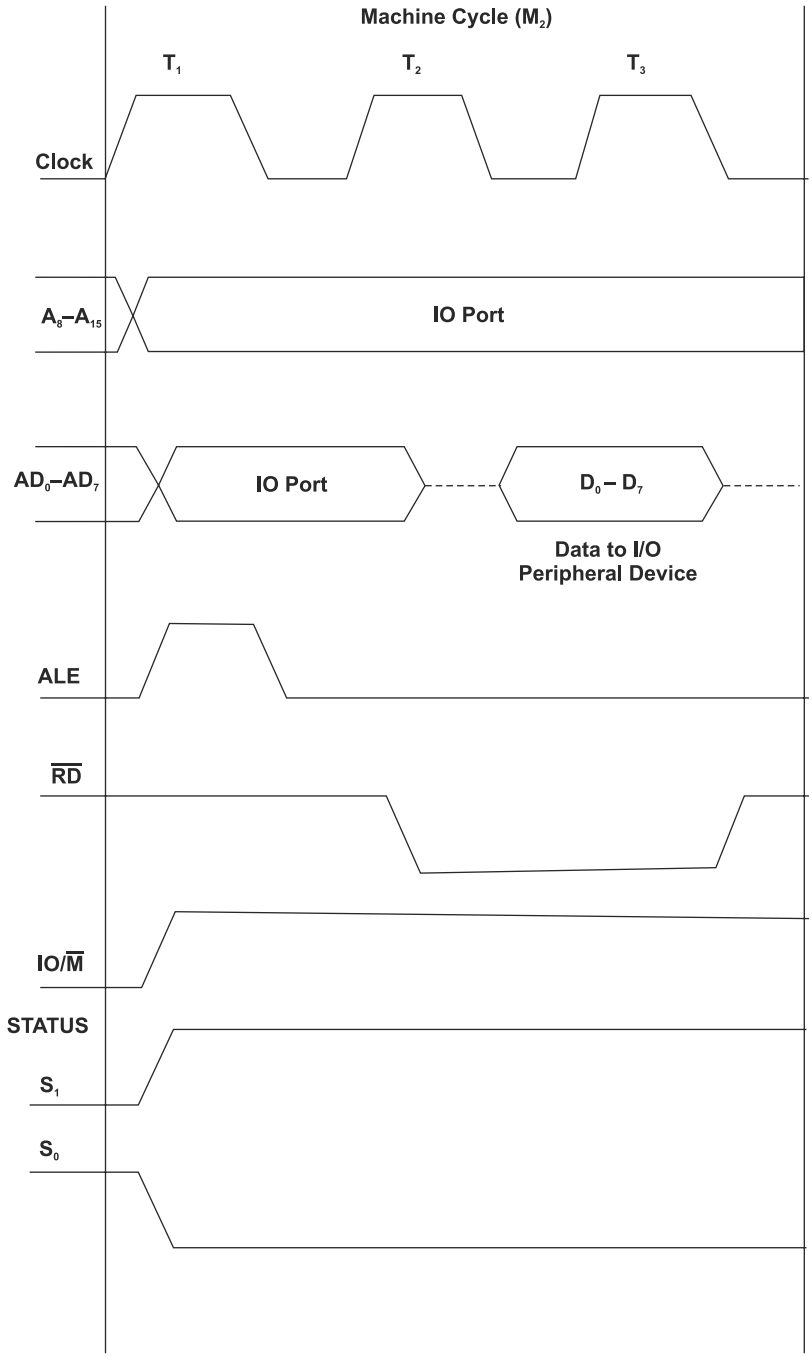


Fig. 3.12 Timing diagram of I/O read in machine cycle M_3

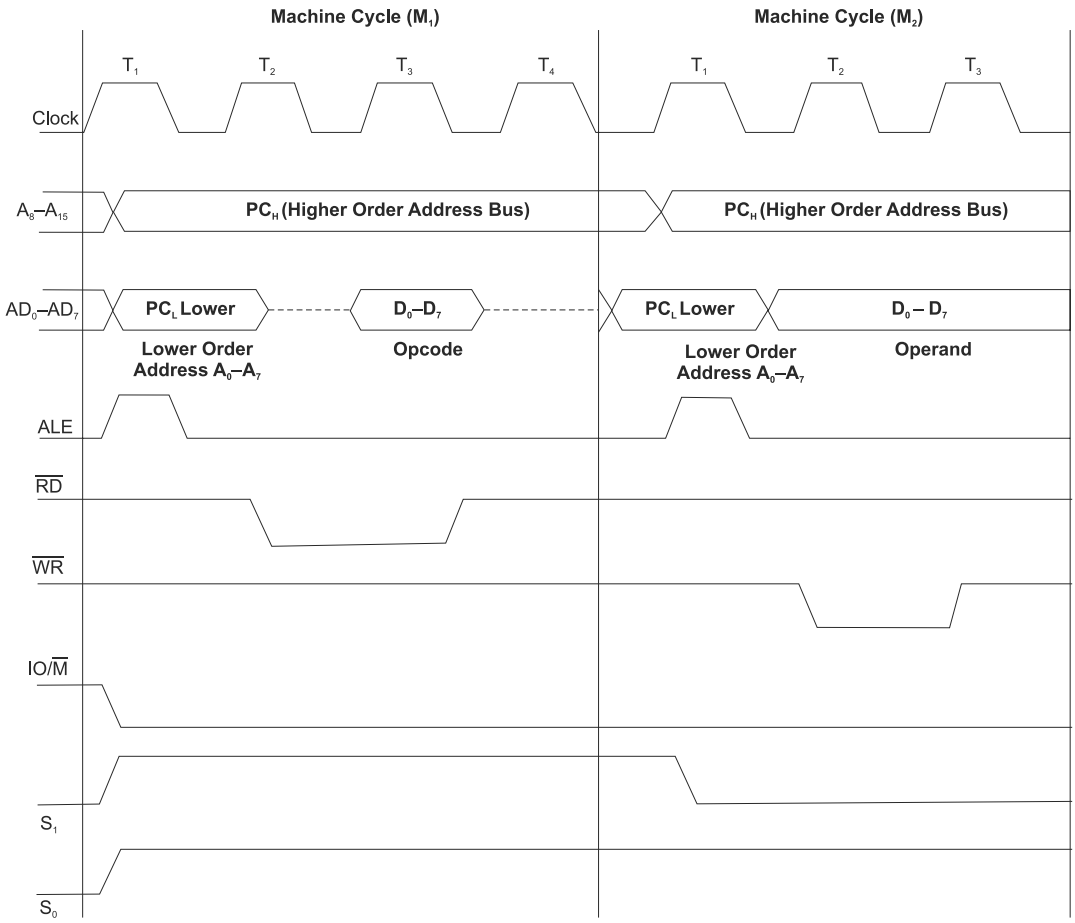


Fig. 3.13 Timing diagram for memory write operation

machine cycle is a memory read cycle for reading the address output device or output port from the memory. In the next third machine cycle data will be written in output device or output port. In other words, data is to be sent to the I/O device. The third machine cycle is explained below:

T_1 State of M_3

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

T_2 State of M_3

- $\overline{WR}$ becomes low for write operation

T_3 State of M_3

- CPU places the content of A register in data bus

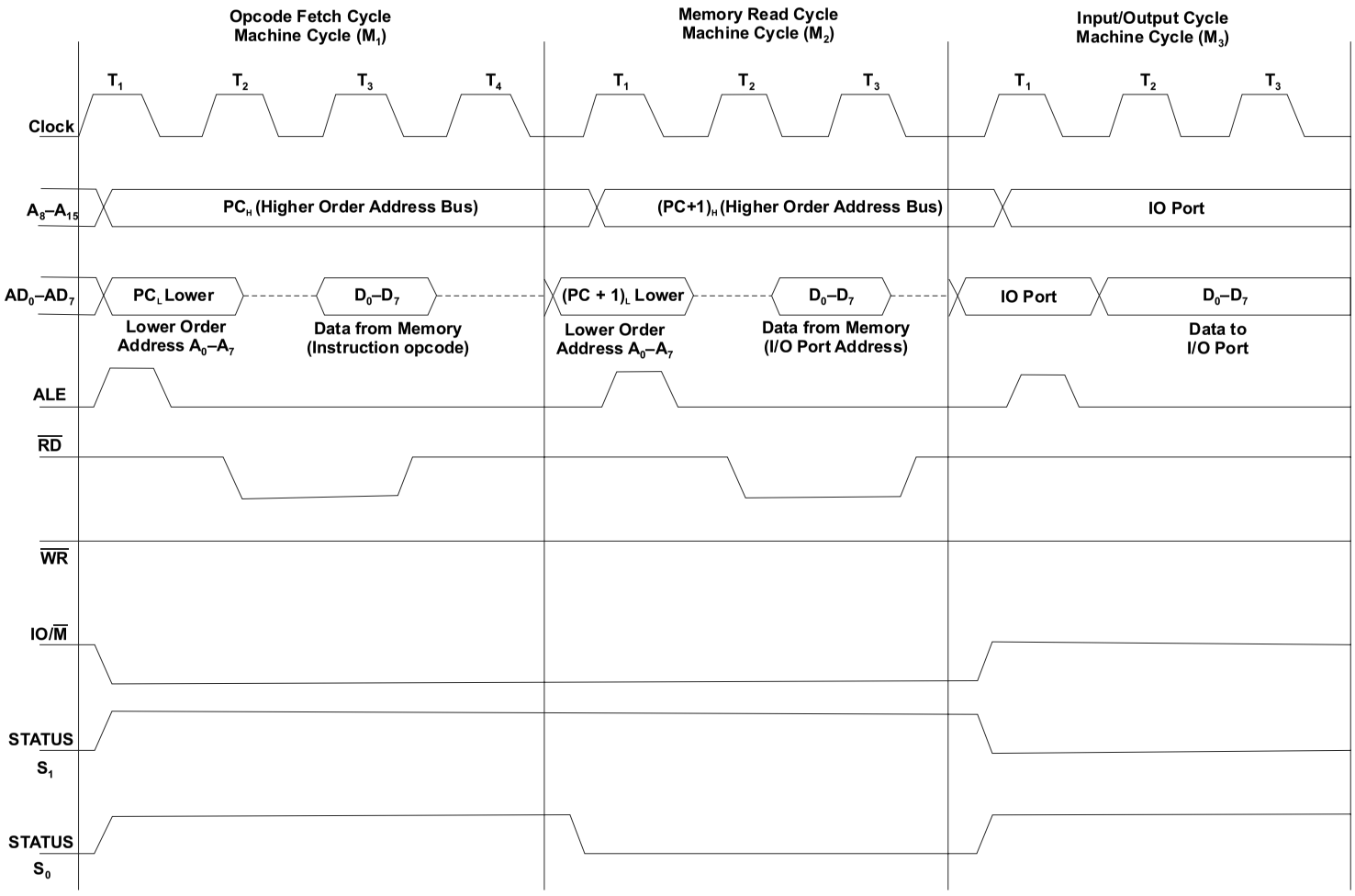


Fig. 3.14 Timing diagram of OUT Port Address

- Then write data to I/O port
- $\overline{WR}$ Signal becomes high as I/O read operation has been completed

Example 3.1 Draw and explain the timing diagram for the instruction STA 8000H

Sol. Consider that instruction STA 8000H is stored at 9000H, 9001H and 9002H memory location as given below:

Memory location	Opcode	Mnemonics
9000 H	32 H	STA 8000H
9001 H	00 H	
9002 H	80 H	

STA 8000H is a three-byte instruction. Figure 3.15 shows the timing diagram of STA 8000H. This instruction requires four machine cycles as depicted in Fig. 3.15. The first machine cycle M_1 is opcode fetch cycle to read the opcode from 9000H memory location. The memory read cycle machine cycle M_2 is used to read the lower order address from memory location 9001H. The machine cycle M_3 is also a memory read cycle to read the higher order address from 9002H. The last machine cycle M_4 can store the contents of A register at the specified memory location 8000H. Therefore machine cycle M_4 is memory write cycle.

The opcode fetch cycle and first memory read cycle is same as the timing diagram of MVI A, FFH. In STA 8000H instruction, the second memory read cycle used to read higher order address. In this section, the operation of M_4 has been explained below:

T_1 State of M_4

- CPU places the content of program counter on the address bus during T_1 of machine cycle M_4; here PCL content 00H and PCH content is 80H.
- ALE signal is high
- IO/$\overline{M}$ signal is low so that program counter content locates the memory location

T_2 State of M_4

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

T_3 State of M_4

- The content of data bus will be written in the specified memory location 8000H
- Then $\overline{WR}$ signal will be changed from low to high as I/O read operation has been completed

Example 3.2 Draw and explain the timing diagram for the instruction LXI H, 8050H.

Sol. The LXI H, 8050H is a three-byte long instruction and is stored at the memory location 8000H, 8001H and 8002H. The opcode for LXI H is 21H and the address is 8050H. Timing diagram of this instruction is depicted in Fig. 3.16. It is clear for Fig. 3.16 that this instruction requires a three-machine cycle. The first machine cycle M_1 is fetch cycle and other two consecutive machine cycles, M_2 and M_3 are memory read cycles.

In the fetch cycle, the opcode for LXI H, 21H is fetched from memory location 8000H and program

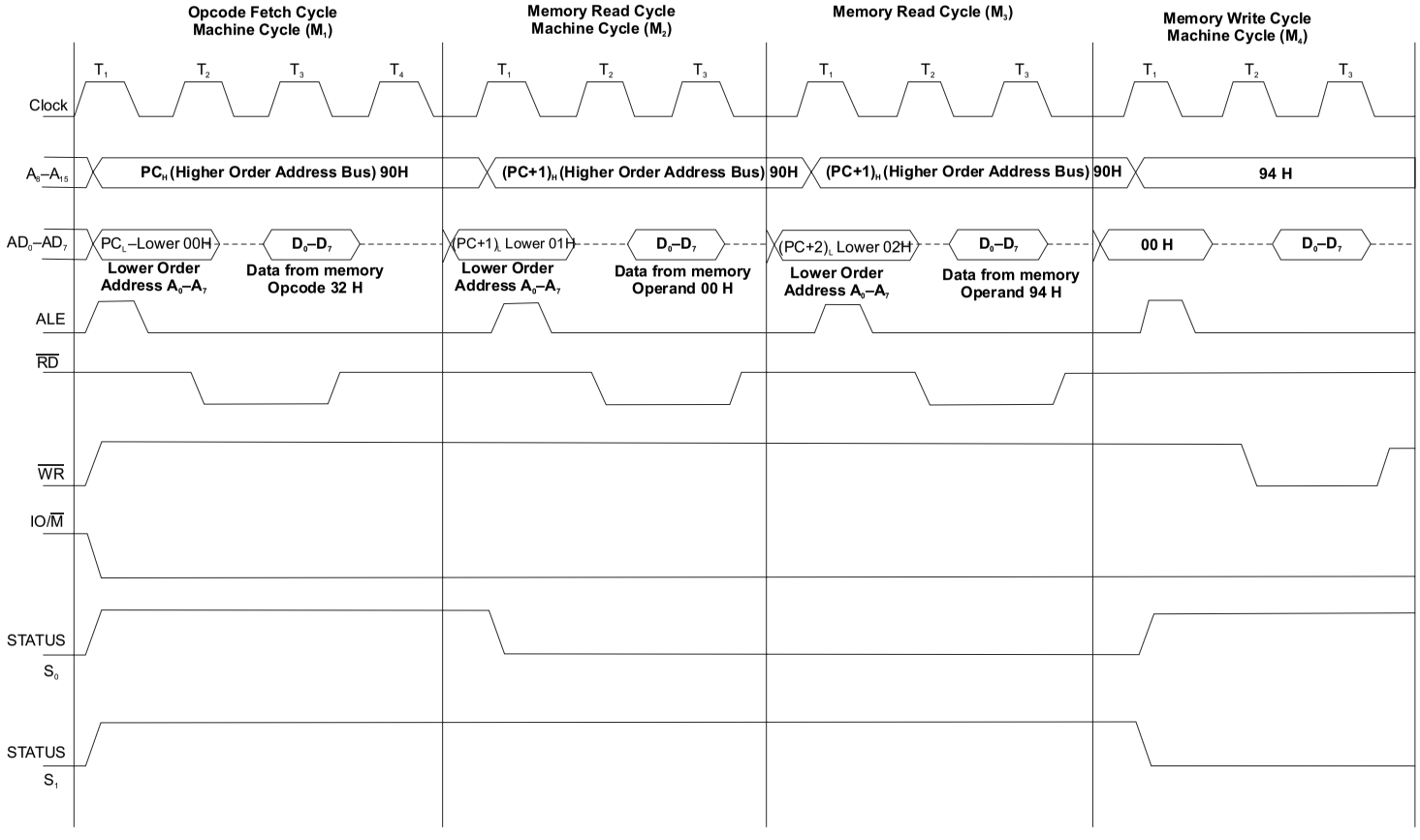


Fig. 3.15 Timing diagram of STA 8000H

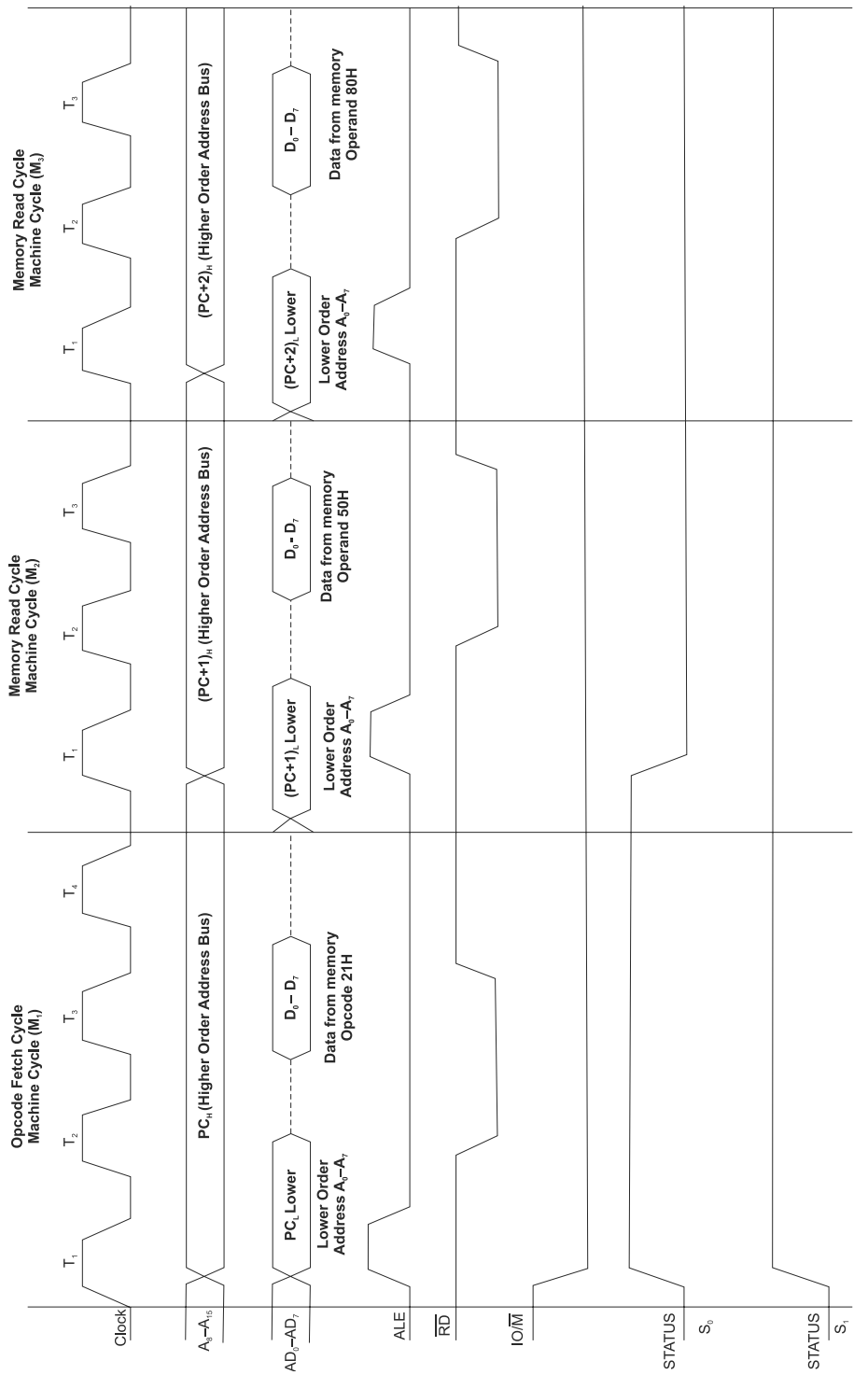


Fig. 3.16 Timing diagram of LXI H, 8050H

counter is incremented by 1 to 8001H. In the second machine cycle M_2, 8 LSBs of the 16-bit data (8050H) to be read. Similarly, in the third machine cycle M_3, 8 MSBs of the 16-bit data (8050) will be read. After execution of this instruction, 50H is stored in L register and 80H is also stored in the H register.

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

Review Questions

3.1 What are the types of addressing modes of Intel 8085? Explain any one addressing mode with suitable examples.

3.2 Classify 8085 instructions in various groups. Give a list of examples of instructions for each group.

3.3 What are the various types of data formats for 8085 instructions? Give a list of examples for each type of data format.

3.4 Write the addressing modes of the following instructions:
(a) MOV A, B (b) MVI C, FFH (c) LXI H, 2500H (d) RAL

3.5 Explain the operation of the following instructions when they are executed:
(a) LXI rp, data (b) STA addr (c) DAD rp (d) DAA
(e) CMP M (f) RAR (g) PUSH rp (h) POP rp.

3.6 Define opcode and operand, and specify the opcode and the operand in the instruction MVI C, 45H.

3.7 Write instructions for the following operations:
(a) Clear accumulator
(b) Exchange the content of DE and HL register pair
(c) Logical AND memory with accumulator
(d) Decrement DE register pair by 1

3.8 Write the difference between
(a) RAL and RLC
(b) RAC and RAR
(c) PUSH and POP

3.9 Find the machine code for the instruction MOV H, A if the opcode = 01, the register code for H = 100_2, and the register code for A = 111.

3.10 Define instruction cycle, machine cycle, and T-state.

3.11 Draw and explain the timing diagram for opcode fetch operation.

3.12 Draw and explain the timing diagram for memory read operation.

3.13 Draw and explain the timing diagram for memory write operation.

3.14 Draw and explain the timing diagram for I/O read operation.

3.15 Draw and explain the timing diagram for I/O write operation.

3.16 Draw timing diagram for the execution of the following instructions:
(a) MVI A, FFH (b) SUB C (c) LXI H, 5000H (d) MOV A, B

3.17 Draw I/O read and write machine cycles. Compare the two machine cycles.

3.18 Distinguish between JMP and CALL instructions.

3.19 Compare the following instructions:
(a) MOV A, M and LDAX D (b) CMP B and SUB B
(c) XCHG and XTHL (d) DAD and DAA

3.20 Draw and explain the timing diagram of STA 8000H.

3.21 Determine the time required to execute the following two instructions if the microprocessor clock frequency is 1 MHz
MOV A, B 5T states
MOV B, C 5T states

Multiple-Choice Questions

3.1 The instructions of 8085 microprocessor has been classified into
(a) four groups of instructions (b) five groups of instructions
(c) six groups of instructions (d) seven groups of instructions

3.2 XCHG is an
(a) data transfer instruction (b) arithmetic instruction
(c) logical instruction (d) I/O and stack pointer instruction

3.3 When the PUSH B instruction is executed,
(a) the content of B register to be copied in the stack
(b) the content of B register and C register to be copied in the stack
(c) the content of B register and C register to be stored in the stack and the registers are cleared
(d) B register and C register are cleared Data transfer instructions

3.4 ORA is an
(a) data transfer instruction (b) arithmetic instruction
(c) logical instruction (d) I/O and stack pointer instruction

3.5 CALL and RET are used in
(a) data transfer instructions (b) branch control
(c) logical instructions (d) I/O and stack pointer instruction

3.6 A one-byte instruction has
(a) opcode and an operand (b) opcode only
(c) opcode and two operands (d) operand only

3.7 A two-byte instruction consists of
(a) opcode and an operand (b) opcode only
(c) opcode and two operands (d) operand only

3.8 A three-byte instruction should have
(a) opcode and an operand (b) opcode only
(c) opcode and two operand (d) operand only

3.9 IN 00H is an instruction of
(a) direct addressing mode (b) indirect addressing mode
(c) register addressing mode (d) immediate-addressing mode

3.10 STA 9000H is an instruction of
(a) one byte (b) two bytes (c) three bytes (d) four bytes

3.11 MOV is an instruction of
(a) direct addressing mode (b) indirect addressing mode
(c) register addressing mode (d) immediate-addressing mode

3.12 CMA is an instruction of
(a) direct addressing mode (b) implicit addressing mode
(c) register addressing mode (d) immediate-addressing mode

3.13 SUB A instruction in the 8085 microprocessor
(a) resets the zero flag (b) sets the zero flag
(c) sets the carry flag (d) resets the auxiliary carry flag

3.14 LXI H, 2500H is an instruction of
(a) direct addressing mode (b) indirect addressing mode
(c) register addressing mode (d) immediate-addressing mode

3.15 CALL 8000H is an instruction of
(a) direct addressing mode (b) indirect addressing mode
(c) register addressing mode (d) immediate-addressing mode

3.16 MOV A, C is executed by
(a) one machine cycle (b) two machine cycles
(c) three machine cycles (d) four machine cycles

3.17 HLT is a
(a) data transfer instruction (b) machine control instruction
(c) arithmetic instruction (d) logical instruction

3.18 LDA 9500H is executed by
(a) one machine cycle (b) two machine cycles
(c) three machine cycles (d) four machine cycles

3.19 STA 8000H is executed by
(a) one machine cycle (b) two machine cycles
(c) three machine cycles (d) four machine cycles

3.20 OUT 02H is executed by
(a) one machine cycle (b) two machine cycles
(c) three machine cycles (d) four machine cycles

3.21 ADI FFH is executed by
(a) one machine cycle (b) two machine cycles
(c) three machine cycles (d) four machine cycles

3.22 DAD is a/an
(a) data transfer instruction (b) arithmetic instruction
(c) logical instruction (d) I/O and stack-pointer instruction

Answers to Multiple-Choice Questions

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

CHAPTER

4

Assembly Language Programming Using 8085

4.1 INTRODUCTION

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

Step 1 Define the problem Before starting, it is required to understand the problem completely and assume all initical conditions.

Step 2 Plan the solution Break the problem into a modular form and determine how the modules are logically linked.

Step 3 Code the program Translate the logical solution of each module in an assembly or any programming language which the microcomputer can understand.

Step 4 Test the program After writing the program, implement/test the program in a microcomputer system.

Step 5 Documentation All related matters must be documented, as we do not always remember the most important steps that we took during development of the program.

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

4.2 MACHINE LANGUAGE

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

- Machine languages
- Assembly languages
- High-level languages

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

The Machine language has the following advantages:

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

The disadvantages of machine-language programs are the following:

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

4.3 ASSEMBLY LANGUAGE

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

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

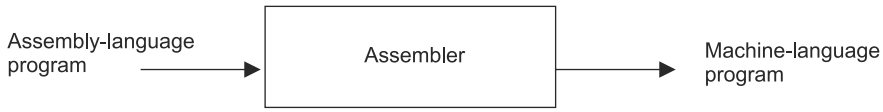


Fig. 4.1 Assembler

The assembly language has the following advantages:

1. It is easy to write.
2. It is easy to understand.
3. Assembly-language programs produce faster results.
4. This is suitable for real-time control and industrial applications.
5. It requires less computations time.

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

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

4.4 HIGH-LEVEL LANGUAGE

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

Fig. 4.2 Compiler

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

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

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

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

Advantages of High-Level Language High-level languages have the following advantages:

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

Disadvantages of High-Level Languages High-level languages have the following disadvantages:

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

4.5 STACK

The stack is a group of memory locations in Read/Write (R/W) memory of any microcomputer and is used to store the contents of a register, operand and memory address. The starting location of the stack is defined

by loading a 16-bit address into the stack pointer, that space is reserved, usually at the top of the memory map. Theoretically, the size of the stack is unlimited, but it is restricted only by the available R/W memory in a microcomputer system. The stack can be initialized anywhere in the user memory map, but the stack is initialized at the highest user memory location so that there will not be any interface with the program.

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

This method of information storage looks like the process of stacking books one above another. Therefore, data is always retrieved from the top of the stack. So data are stored in the stack on last-in-first-out (LIFO) principle.

The syntax of stack instructions to store data on and retrieve data from the stack are given below:

Opcode	Operand	Description
LXI	SP, 16-bit	Load 16-bit address into the stack pointer register. This is a load instruction, similar to other 16-bit load instructions discussed previously.
PUSH PUSH and	RP R	This is a 1-byte instruction and copies the contents of the specified register pair or index register onto the stack as described in this section. Instructions for four register pairs index registers are listed here.
PUSH	PSW	The instruction first decrements the stack pointer (register) and copies the high-order
PUSH	BC	byte of the register pair or the index register on the stack location SP – 1.
PUSH	DE	Then it again decrements the stack pointer and copies the low-order byte of the register
PUSH	HL	pair or the index register onto the stack location SP – 2.

Opcode	Operand	Description
POP	RP	This is a 1-byte instruction and copies the contents of the top two locations of the
POP	R	stack into the specified register pair or the index register.
POP	PSW	First, the instruction copies the contents of the stack indicated by SP into the
POP	BC	low-order register (for example, register C of the BC pair) or as a low-order
POP	DE	byte into the index register and then increments the stack pointer to SP + 1.
POP	HL	It copies the contents of the SP + 1 location into the high order register (for example, register B of the BC pair) or as a high-order byte into the index register and increments the stack pointer to SP + 2.

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

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

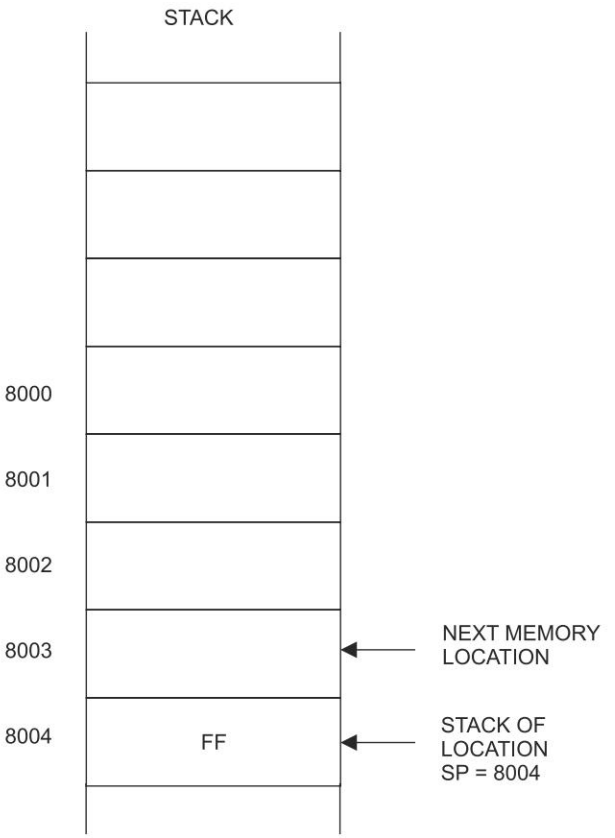


Fig. 4.3 Stack and stack top location

Program

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

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

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

1. During the execution of a program, a 16-bit address (8004H) is given to the stack pointer register. The stack space grows upward in the numerically decreasing order of memory addresses. The contents of *DE* register pairs can be stored beginning from the next location (SP – 1).
2. The PUSH instructions are used to store contents of register pairs on the stack, and the POP

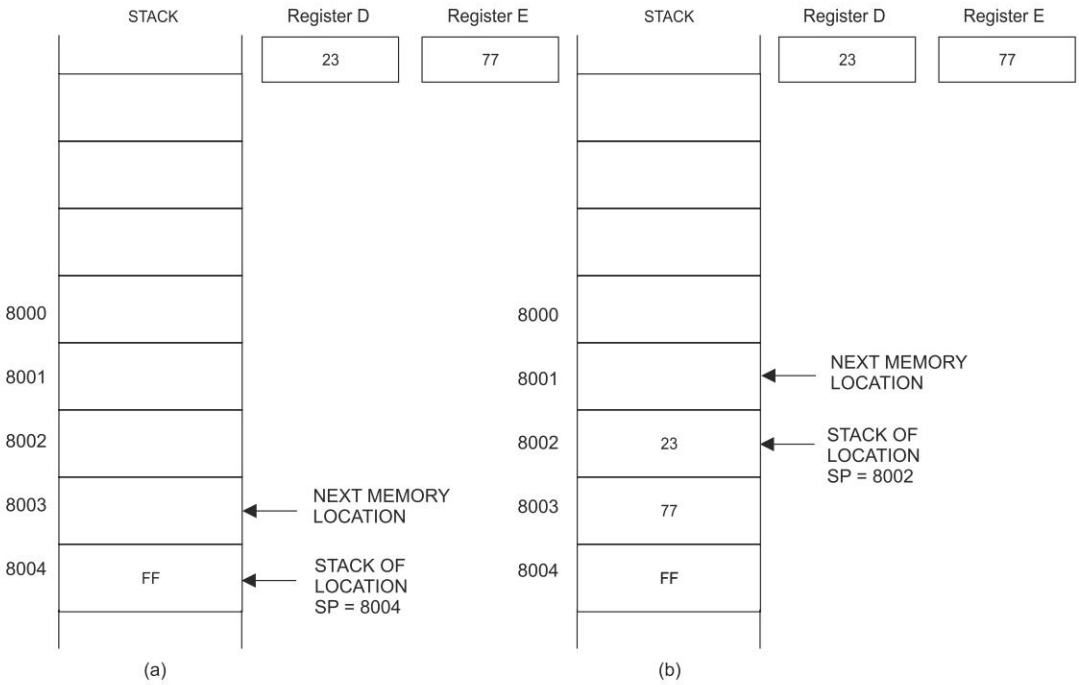


Fig. 4.4 (a) Stack before PUSH operation (b) Stack after PUSH operation

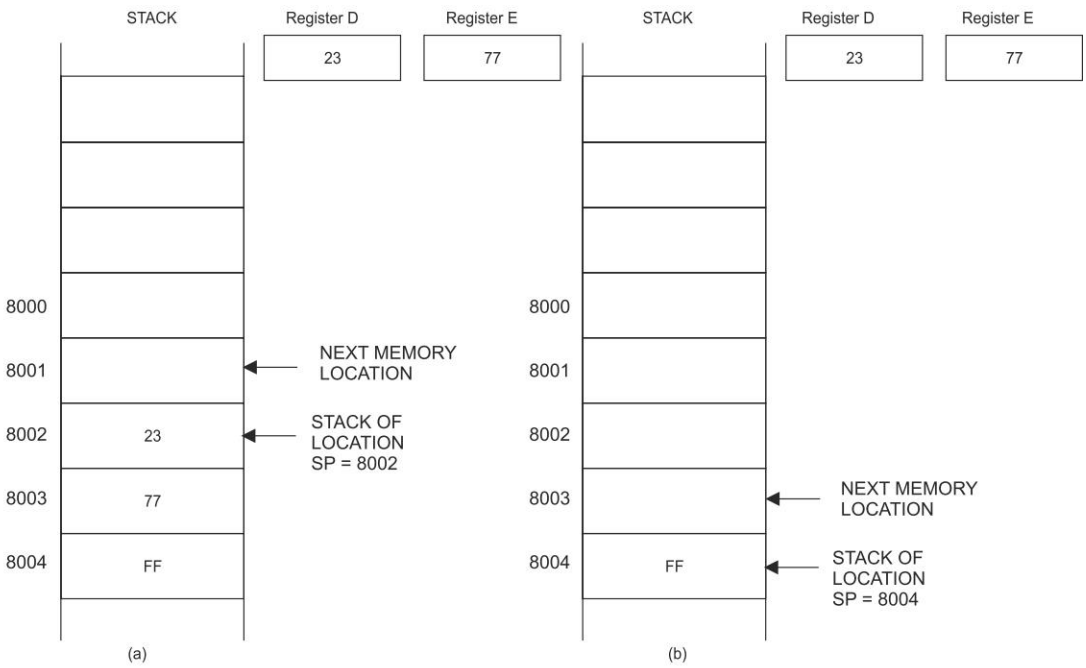


Fig. 4.5 (a) Stack before POP operation (b) Stack after POP operation

instructions are used to retrieve the information from the stack. The address in the stack pointer register always points to the top of the stack, and the address is decremented as information is stored or retrieved, respectively.

3. The storage and retrieval of the content of registers on the stack should follow the LIFO (Last-In-First-Out) sequence.
4. Information in the stack locations may not be destroyed until new information is stored in that memory location.

4.6 SUBROUTINES

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

The subroutine is implemented with two associated instructions, namely, Call and Return. *Call* is used to call a subroutine and the Call instruction is written in the main program. *Return* is used to return from the subroutine and the Return instruction is written in the subroutine to return to the main program. When a subroutine is called, the contents of the program counter are stored on the stack, and the program execution is transferred to the subroutine address. When the Return instruction is executed at the end of the subroutine, the memory address stored in the stack is retrieved and the sequence of execution is resumed in the main program. All types of CALL and RET instructions are explained in this chapter. The syntax of CALL and RET are given below:

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

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

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

The following types of subroutines generally are used in microprocessors:

1. Multiple call subroutines

2. Nested subroutines
3. Multiple Ending subroutines

4.6.1 Multiple Call Subroutines

Figure 4.6 shows the basic concept of multiple CALL subroutines. This is a subroutine called from many locations in the main program. For example, the DELAY routine is a multiple call subroutine. These types of routines are easy to trace and need minimal stack space. Initially, stack pointer content is XX55 H so that the return address can be stored on the stack. When the CALL instruction starts to execute, the subroutine is called from the 8050 memory location. The return address is stored on the stack and the stack pointer is decremented by two locations to XX53H.

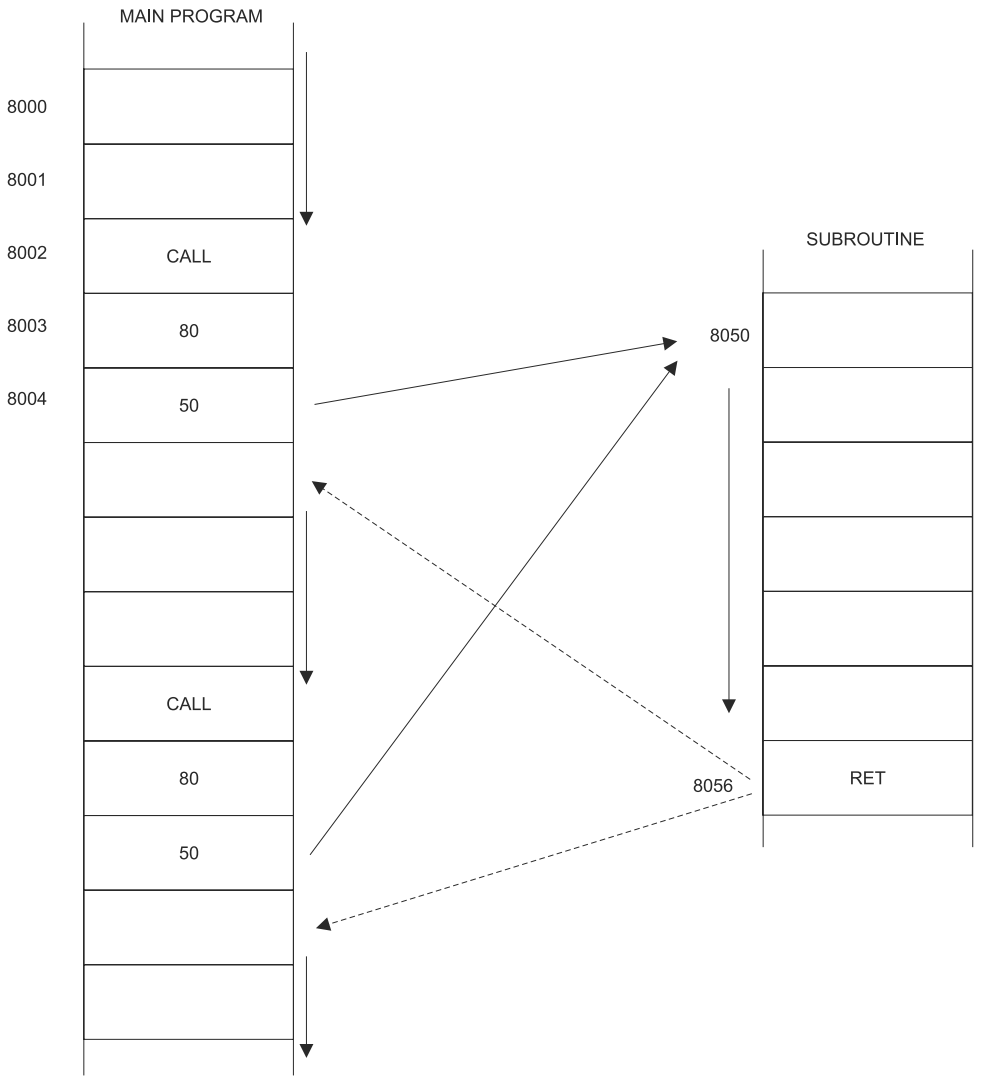


Fig. 4.6 Multiple call subroutines

4.6.2 Nested Subroutines

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

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

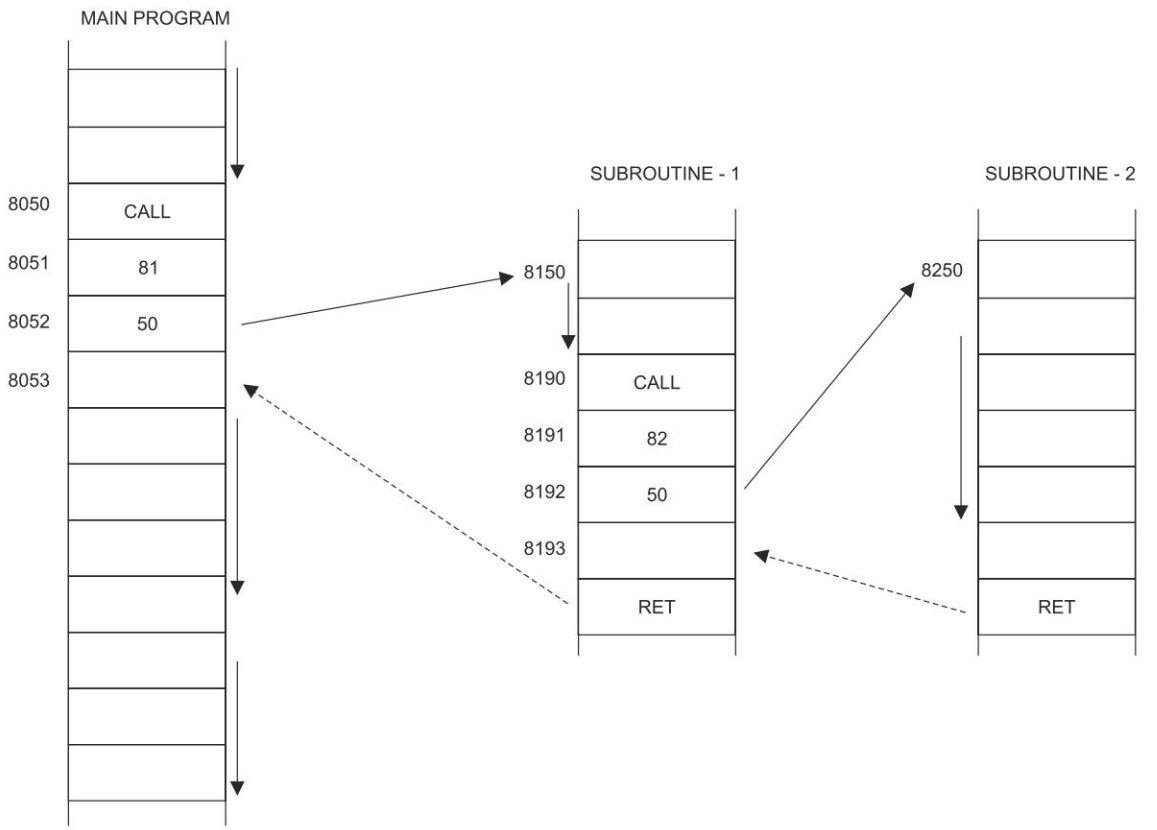


Fig. 4.7 Nested subroutines

4.6.3 Multiple Ending Subroutines

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

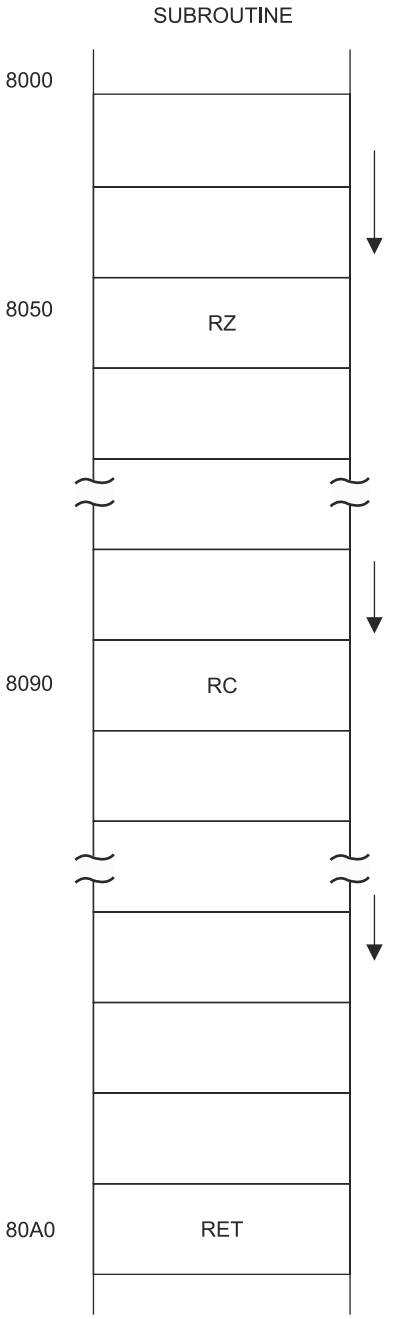


Fig. 4.8 Multiple ending subroutine

4.7 TIME DELAY LOOPS

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

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

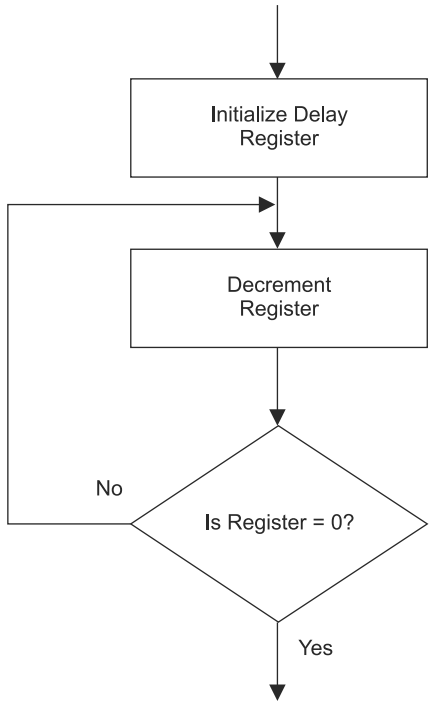


Fig. 4.9 Flow chart for time delay using register

4.7.1 Calculation of Time Delay Using One Register

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

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments	*T* states
8000	06, 80		MVI	B, 80	Initialise the B register	7
8002	05	LOOP	DCR	B	Decrement B register	5
8003	C2, 02, 80		JNZ	LOOP	Jump not zero to LOOP	10/7

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

Hence, DCR B is executed 128 times. The JNZ is also executed for 128 times, out of which (N-1)=127 times the program jumps to the level LOOP as the content of B register is not equal to zero and takes 10 T states each time. When the content of B register becomes zero, JNZ is executed and the program will be out from LOOP. For the last execution of JNZ instruction, only 7 T states are required. The detailed execution of instructions and T states are given below:

Instructions	Number of times instruction executed	*T* states
MVI B,80	1	7
DCR B	128	128×5
JNZ	128	(128-1)×10+7
		Total T states = 1924

The number of T states for execution of LOOP is
$= N \times T$ states for DCR B + $(N - 1) \times$ T states for JNZ +7
$=128 \times 5 + (128–1) \times 10 + 7$ T states $= 1917$ T states

The delay time to execute the LOOP instruction is $T_L = T \times$ number of T states for execution of LOOP, where T is the system clock period. When the microprocessor operate in 5 MHz clock frequency, $T_L = \frac{1}{5} \times 10^{-6} \times 1917$ s = 383.4 μs.

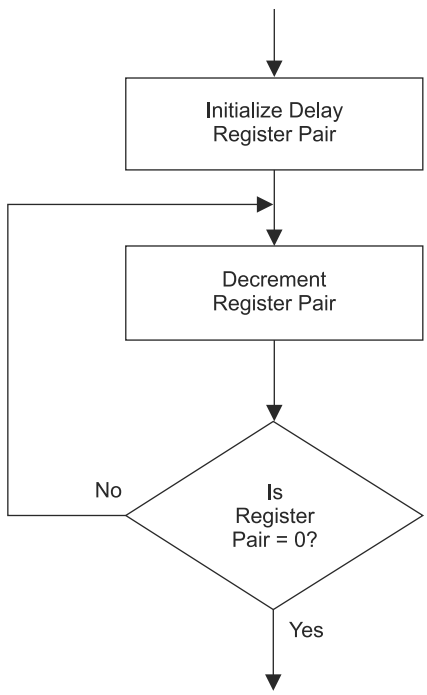


Fig. 4.10 Flow chart for time delay using register pair

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

$T_D = T_{OL} + T_L = \frac{1}{5} \times 10^{-6} \times 7 + \frac{1}{5} \times 10^{-6} \times 1917$ s = 384.8 μs

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

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

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments	T states
8100	11, 00, 80		LXI	D, 8000	Initialise the DE register pair	10
8103	1B	LOOP	DCX	D	Decrement DE register pair	5
8104	7B		MOV	A, E	Copy content of E register in accumulator	5
8105	B2		ORA	D	OR D with accumulator	4
8106	C2, 03, 81		JNZ	LOOP	Jump not zero to LOOP	10/7

4.7.2 Calculation of Time Delay Using Register Pair

In time delay loop using register pair, LXI D, 8000 is executed once and the other instructions DCX D, MOV A, E, ORA D and JNZ are executed for many times.

The detail execution of instructions and *T* states are given below:

Instructions	Number of times instruction executed	T states
LXI D,8000	1	10
DCX D	32768	32768 × 5
MOV A, E	32768	32768 × 5
ORA D	32768	32768 × 4
JNZ	32768	(32768 – 1) × 10 + 7
		Total *T* states = 786439

The *number* of *T* states for execution of LOOP is

= (*N* × *T* states for DCX D + *N* × *T* states for MOV A, E + *N* × ORA D + (*N* – 1 × *T* states for JNZ) + 7

where, *N* = 8000H = 32768 D

= 32768 × (5 + 5 + 4) + (32768 –1) × 10 + 7 *T* states

= 786429 *T* states

If the microprocessor clock frequency is 5 MHz, time delay in LOOP is equal to T_L.

$T_L = T \times$ number of *T* states for execution of LOOP

$= \frac{1}{5} \times 10^{-6} \times 786429$ s = 157.2858 ms (approx).

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

$$T_D = T_{OL} + T_L = \frac{1}{5} \times 10^{-6} \times 10 + \frac{1}{5} \times 10^{-6} \times 786429 \text{ s}$$

=157.2878 ms

4.7.3 Time Delay Using Two LOOPs

The time delay can also be generated by using two loops as depicted in Fig. 4.11. The C register is used in the inner loop (LOOP-I) and B register is used in the external loop (LOOP-II). Here, both B and C registers are loaded with numbers. The C register is decremented until it becomes zero. When the content of C register

is zero, decrement B register. If the content of B register is not zero, load the C register with initial value and repeat the process.

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

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments	T state
8200	06, 80		MVI	B, 80	Initialise the B register	7
8202	0E, FF	LOOP-II	MVI	C, FF	Initialise the C register	7
8204	0D	LOOP-I	DCR	C	Decrement C register	5
8205	C2, 04, 82		JNZ	LOOP-I	Jump not zero to LOOP-I	10/7
8208	05		DCR	B	Decrement B register	5
8209	C2, 02, 82		JNZ	LOOP-II	Jump not zero to LOOP-II	10/7
820C	C9		RET		Return to main program	10

In time delay generation using two loops as given above, MVI B, 80 is executed once and the other instructions MVI C, FF, DCR C, JNZ, DCR B and JNZ are executed for many times. The detail execution of instructions and *T* states are given below:

Instructions	Number of times instruction executed	*T* states
MVI B,80	1	7×1
MVI C, FF	128	128×7
DCR C	255×128	$255 \times 128 \times 5$
JNZ LOOP - I	255×128	$(255 - 1) \times 128 \times 10 + 7$
DCR B	128	128×5
JNZ LOOP - II	128	$(128 - 1) \times 10 + 7$
RET	1	10×1
		Total *T* states = 491157

Time delay will be calculated based on the time delay for LOOP-I and LOOPII. Total number of *T* states for LOOP-I and LOOP-II are equal to

$= 128 \times 7 + 255 \times 128 \times 5 + (255 - 1) \times 128 \times 10 + 7 + 128 \times 5 + (128 - 1) \times 10 + 7$ *T* states

$= 491140$ *T* states

If the microprocessor clock frequency is 5 MHz, time delay in LOOP-I and LOOP-II is equal to T_L,

$T_L = T \times$ number of *T* states for execution of LOOP -I and LOOP-II

$= \frac{1}{5} \times 10^{-6} \times 491140$ s = 98.228 ms.

Total delay time (T_D)

= Time to execute instruction of outside LOOP (T_{OL}) + Time to execute LOOP- I and LOOP-II

$= \frac{1}{5} \times 10^{-6} \times 7 + \frac{1}{5} \times 10^{-6} \times 491140 + \frac{1}{5} \times 10^{-6} \times 10$ s

= 98.2314 ms

4.8 MODULAR PROGRAMMING

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

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

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

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

Advantages of Modular Programming The advantages of modular programming are the following:

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

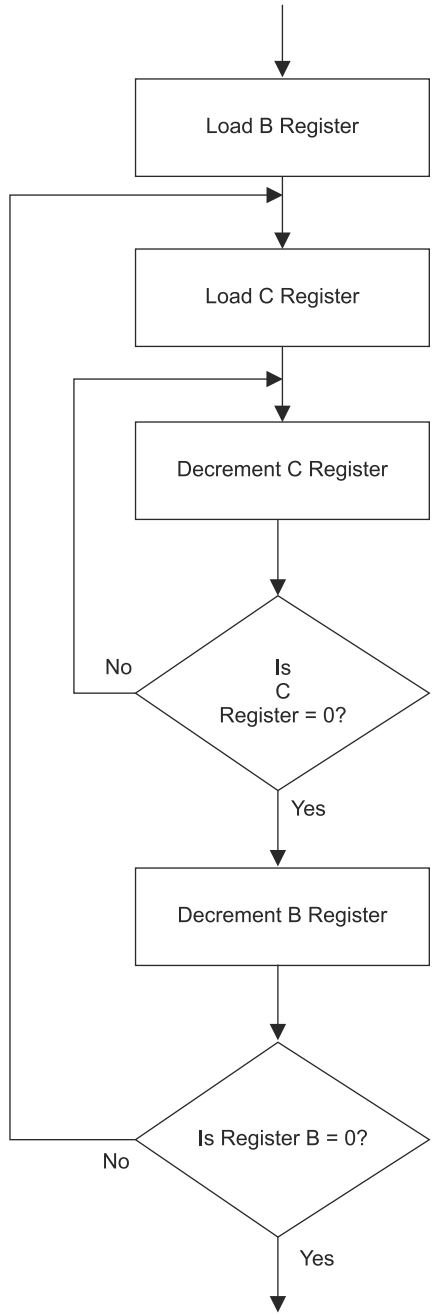


Fig. 4.11 Flow chart for time delay using two loops

8. *Manageable* Reduces problem to smaller, simpler, humanly comprehensible problems.
9. *Divisible* Modules can be assigned to different teams/programmers. Enables parallel work, reducing program development time. Facilitates programming, debugging, testing, maintenance.
10. *Portable* Individual modules can be modified to run on other platforms.
11. *Re-usable* Modules can be re-used within a program and across programs.

Disadvantages of Modular Programming The disadvantages of modular programming are the following:

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

4.9 MACRO

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

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

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

The example of a DELAY MACRO is

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

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

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

Another example of MACRO is ADDER as given below:

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

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

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

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

4.10 INSTRUCTION FORMAT

Each statement in an assembly-language program consists of the following fields: memory address, machine codes, labels, mnemonics, operands and comments. The commonly used format of an instruction in assembly language is given below:

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments

Memory Address This is the address of the memory location in which a program or a series of instructions are stored.

Machine Codes Every instruction has a unique one-byte code called *operation code*. Instructions are operated using data. Data may be of one byte or two bytes. Machine codes are the hexadecimal representation of operation codes.

Labels It is assigned for the instruction in which it appears. The presence of a label in an instruction is optional. When a label is present, it provides a symbolic name that can be used in branch instructions of the instruction. If there is no label, then the colon must not be entered. A label may be of any length, from 1 to 35 characters. This appears in a program to identify the name of a memory location for storing data and other purposes. This is used for conditional/unconditional jumping.

Mnemonics Each instruction has a specific mnemonic. The mnemonic states the operation which will be executed.

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

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

4.11 ASSEMBLY-LANGUAGE PROGRAMS

4.11.1 Simple Examples of Assembly-Language Programs

Example 1 Transfer data from accumulator to B register.

Mnemonics	*Opcode*	*Comments*
MOV B, A	47	Copy the content of accumulator to B register

Example 2 Load FFH in C register

Mnemonics	*Opcode*	*Comments*
MVI C, FFH	0E, FF	Load FFH in C register immediately

Example 3 Load 22H and 67H in Registers B and C respectively

Mnemonics	*Opcode*	*Comments*
LXI B 22 67	01, 67, 22	Load Register C with 67H and Register B with 22H.

Example 4 Load H-L register pair by the data 8150H

Mnemonics	*Opcode*	*Comments*
LXI H,8150H	21, 50, 81	Load H-L register pair with 8150H

Example 5 Load the content of memory location 8100H in the accumulator

Mnemonics	*Opcode*	*Comments*
LXI H, 8100H	21, 00, 81	Load memory location address 8100H in H-L register pair
MOV A, M	7E	Copy content of memory location in accumulator

Example 6 Store the content of the accumulator in 8001H location

Mnemonics	*Opcode*	*Comments*
STA 8001H	32, 01, 80	Content of accumulator is stored in 8001 location

Example 7 Transfer data stored in memory location 9950H to the ACC

Mnemonics	*Opcode*	*Comments*
LDA 9950H	3A 50 99	Move data to accumulator from memory location 9950H

Example 8 Load 45H data in the memory location 8500H and increment the content of memory location

Mnemonics	*Opcode*	*Comments*
LXI H, 8500H	21, 00, 85	Load memory location address 8500H in H-L register pair
MVI M, 45H	36, 45	45H is stored in 8500H location
INR M	34	Content of memory location incremented by one

4.11.2 Detect Even and Odd Numbers

Assume data is stored in the memory location 8000. When data is even, 00H will be stored in 8001H. When data is odd, 01H will be stored in 8002H. Assume the program is written from memory location 8500H.

Algorithm

1. Consider the number is stored in 8000H.
2. Move data into the accumulator.
3. AND 01H with content of accumulator.
4. After AND operation if LSB is 0 then the number is even, otherwise number is odd.
5. To check LSB, rotate the accumulator right with carry. If the carry flag is set, the number is odd, otherwise it is even.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8500	21, 00, 80		LXI	H, 8000	Initialise the memory location of number, 8000H in HL register pair.
8503	36, Number		MVI	M, NUMBER	Store number in memory.
8505	7E		MOV	A, M	Transfer data from memory into accumulator.
8506	E6, 01		ANI	01H	AND 01H with accumulator
8508	0F		RRC		Rotate accumulator right once so that LSB moves to carry bit.
8509	DA, 11, 85		JC	LOOP	If carry flag is set, number is odd and is stored in location 8102.
850C	23		INX	H	Increment register pair.
850D	3E, 00		MVI	A, 00H	Load 00H in accumulator.
850F	77		MOV	M, A	If carry flag is reset, number is even and 00H is stored in location 8001.
8510	76		HLT		Halt
8511	23	LOOP	INX	H	Increment register pair.
8512	3E, 01		MVI	A, 01H	Load 01H in accumulator.
8514	77		MOV	M, A	If carry flag is set, number is even and 01H is stored in location 8002.
8515	76		HLT		Halt

4.11.3 Addition of two 8-bit Numbers and getting the Sum as 16-bit

The first number F2H is stored in the memory location 8501H, and the second number 2FH is stored in the memory location 8502H. The result after addition will be stored in the memory location 8503H and 8504H. Consider that the program is written from memory location 8000H.

Algorithm

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

Program

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

Example

DATA		RESULT	
Memory location	**Data**	**Memory location**	**Data**
8501	F2H	8503	82H LSBs of sum
8502	2FH	8504	01H MSBs of sum

4.11.4 Addition of *N* 8-bit Numbers

Write a program for addition of a series of 8-bit numbers with carry. The *N* number of hexadecimal numbers lie from F101 onwards. F100 has the number of hexadecimal bytes to be added. The result is stored at F200 and F201 memory location. Assume program is written in memory location F000H. Figure 4.12 shows the flow chart for addition of *N* 8-bit numbers.

Algorithm

1. Load the number of bytes to be added in F100 memory location.
2. Initialise accumulator, as LSBs of the result will be stored in the accumulator.
3. B register is also initialised to store MSBs of sum.
4. Let the memory point the number of the bytes to be added and stored in C register.
5. Move the next memory location for data and data with accumulator.
6. If carry is generated, B register will be incremented by one.
7. Decrement the counter having number of bytes.
8. Check if zero — no repetition from point 5.
9. Store the result at F200 and F201 location.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
F000	21, 00, F1		LXI	H,F100H	Address of number of bytes in H-L register pair.
F003	4E		MOV	C, M	Transfer number of bytes from memory location to C register.
F004	AF		XRA	A	Clear accumulator register.
F005	06, 00		MVI	B,00	Initialise B register with 00H to store MSBs of sum.
F007	23	LOOP	INX	H	Address of 1st number in H-L pair.
F008	66		ADD	M	Add memory to accumulator.
F009	D2, 0D, F0		JNC	LEVEL_1	If carry is not generated, jump to LEVEL_1.
F00C	04		INR	B	If carry is generated, increment B register.
F00D	0D	LEVEL_1	DCR	C	Decrement count by one.
F00E	C2, 07, F0		JNZ	LOOP	Test to check whether addition of all numbers are done.
F011	32, 00, F2		STA	F200	Store LSBs of sum in memory location 8503H.
F014	78		MOV	A,B	Copy content of B in accumulator.
F015	32, 01, F2		STA	F201	Store MSBs of sum in memory location 8503H.
F018	76		HLT		Stop

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

DATA		RESULT	
Memory location	**Data**	**Memory location**	**Data**
F100	05H	F 200	0FH LSBs of sum
F101	01H	F 201	00H MSBs of sum
F102	02H		
F103	03H		
F104	04H		
F105	05H		

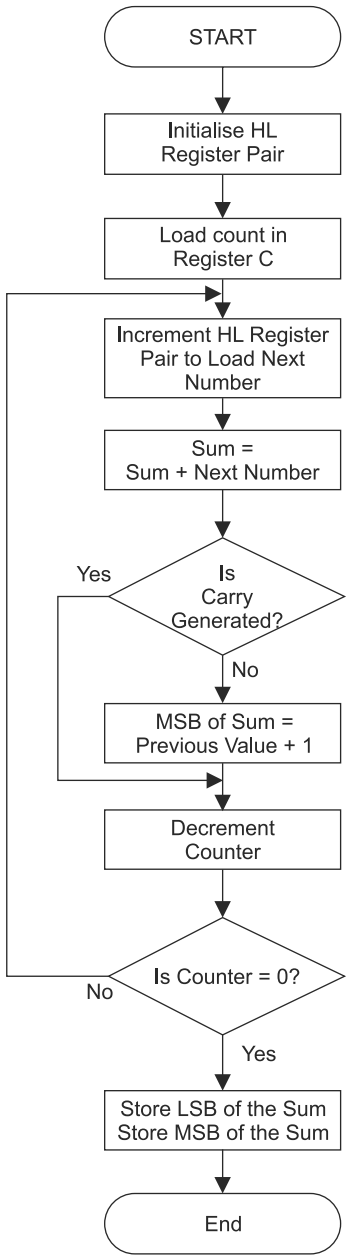


Fig. 4.12 Flow chart for addition of *N* 8-bit numbers

4.11.5 Addition of Two 16-bit Numbers and Sum is more than 16-bit

Assume the first 16-bit number is stored in 8001 and 8002 memory locations. The second 16-bit number is stored in 8003 and 8004 memory locations. After addition, the result will be stored from 8005 to 8007 memory locations. Consider that the program is written from memory location 8500H.

Algorithm

1. Load the content of memory location 8001H in L register and 8002 in register H.
2. Exchange the contents of D-E pair and H-L pair and first number is stored in D-E register pair.
3. Store the second 16-bit number in H-L register pair.
4. Addition of 1st and 2nd number using DAD instruction.
5. Clear accumulator.
6. Add accumulator with carry and store result in the accumulator.
7. Store the results in 8005H to 8007H locations.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8500	2A, 01, 80		LHLD	8001 H	Store first 16-bit number in H-L pair.
8503	EB		XCHG		Exchange the contents of D-E pair and H-L pair to store first number in D-E registers pair.
8504	2A, 03, 80		LHLD	8003 H	Store second 16-bit number in H-L register pair.
8507	3E, 00		MVI	A, 00	Load 00H in accumulator.
8509	19		DAD	D	Add the content of D-E and H-L register pair as well as store the result in H-L registers pair.
8500					
850A	D2, 0E, 85		JNC	LEVEL_1	If carry is not generated, jump to the LEVEL_1.
850D	8F		ADC	A	Add carry and accumulator.
850E	EB	LEVEL_1	XCHG		Exchange the contents of D-E pair and H-L pair.
850F	21, 05, 80		LXI	H, 8005H	Initial memory location 8005H.
8502	73		MOV	M, E	Copy the LSBs of sum in 8005H location.
8503	23		INX	H	Increment H-L register pair.
8504	72		MOV	M, D	Copy the LSBs of sum in 8006H location.
8505	23		INX	H	Increment H-L register pair.
8506	77		MOV	M, A	Store MSBs of the sum in 8007 H.
8507	76		HLT		Halt.

Example

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

	F	0	F	0	H	1st number
	1	F	0	F	H	2nd number
Sum: 01	0	F	F	F	H	

4.11.6 Addition of *N* 16-bit Numbers

The number of 16-bit data is loaded in the 8000H location and the numbers are stored from 8001H onwards. Assume 05H is loaded in the 8000H location and during addition, if result is more than 16 bit, 8400H can be used to store carry. After addition of all 16-bit numbers, the results will be stored in 8500H to 8502H locations. Consider the stack pointer register is initialised with 9000H. Assume program starts from memory location 8600H.

Algorithm

1. Initialise stack pointer register with 9000H.
2. Load 00H in accumulator and store in memory location 8400H.
3. Number of data will be stored in 8000H location and *N* number 16 bit data are stored from 8001H.
4. HL register pair content is initialised by loading 0000H.
5. D-E register pair is loaded with first data.
6. Content of D-E added with content of H-L.
7. If carry is generated, the content of memory location 8400H is incremented by one.
8. Load accumulator from memory location 8000H and decrement accumulator. If accumulator content is not zero, load next number in DE and perform addition.
9. If accumulator content is zero, store the results in 8500H to 8502H locations.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8600	31, 00, 90		LXI	SP, 9000H	Initialise stack pointer register with 9000H.
8603	3E, 00		MVI	A, 00H	Store 00H in accumulator.
8605	32, 00, 84		STA	8400H	Store the accumulator content in 8400 H.
8608	21, 00, 80		LXI	H, 8000	Address of number of 16-bit data is loaded in H-L pair.
860B	36, 05		MVI	M, 05H	Load 05H, no. of data in 8000H location.
860D	21, 00, 00		LXI	H, 0000H	Clear H-L register pair.
8610	E5		PUSH	H	The content of H-L register pair pushed into stack.
8611	21, 01, 80		LXI	H, 8001	Load address of first 16-bit data.
8614	5E	LOOP	MOV	E, M	Copy the content of memory in E register.
8615	23		INX	H	Increment H-L register pair.
8616	56		MOV	D, M	Copy the content of memory in D register.
8617	4D		MOV	C, L	Move the content of L register to C register.
8618	44		MOV	B, H	Move the content of H register to C register.
8619	E1		POP	H	Saved data in stack pointer register to H-L register pair.

861A	19		DAD	D	Addition of the contents of D-E pair and H-L pair.
861B	E5		PUSH	H	After addition, the content of H-L register pair is pushed into stack.
861C	DA, 2F, 86		JC	LEVEL_1	
861F	3A, 00, 80	LEVEL_3	LDA	8000H	Load accumulator from memory location 8000H.
8622	3D		DCR	A	Decrement accumulator.
8623	CA, 3A, 86		JZ	LEVEL_2	
8626	32, 00, 80		STA	8000H	Store the content of accumulator in memory location 8000H.
8629	69		MOV	L, C	Move the content of C register to L register.
862A	60		MOV	H, B	Move the content of B register to H register.
862B	23		INX	H	Increment H-L register pair.
862C	C3, 14, 86		JMP	LOOP	Jump to LOOP.
862F	3A, 00, 84	LEVEL_1	LDA	8400H	Load the content of 8400 to accumulator.
8632	3C		INR	A	Increment accumulator.
8633	32, 00, 84		STA	8400H	Store the accumulator content in 8400H.
8636	3F		CMC		Complement the carry status or reset carry.
8637	C3, 1F, 86		JMP	LEVEL_3	Jump to LEVEL_3.
863A	E1	LEVEL_2	POP	H	Retrieve data of H-L register pair from stack pointer register.
863B	EB		XCHG		Exchange the content of D-E and H-L register pair.
863C	21, 00, 85		LXI	H, 8500H	Load address 8500H.
863F	73		MOV	M, E	Copy the content of E register in memory location.
8640	23		INX	H	Increment H-L register pair.
8641	72		MOV	M, D	Copy the content of D register in memory location.
8642	3A, 00, 84		LDA	8400H	Load the content of 8400 to accumulator.
8645	32, 03, 85		STA	8503	Load the content of accumulator to 8503.
8648	76		HLT		Stop.

4.11.7 Decimal Addition of Two 8-bit Numbers and Sum is 8-bit

Two decimal numbers are stored in F000H and F001H. The result is to be stored in the F002H location. Consider program starts from memory location F100H.

Algorithm

1. Initialise the memory location of the first number in HL register pair.
2. Load the first number in the accumulator.
3. Increment the content of HL register pair to initialise the memory location of second data.
4. Addition of the content of second memory location with first data.
5. Decimally adjust the result.
6. Store the result in memory location F002H.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
F100	21, 50, F0		LXI	H, F050	Load the address of first number in H-L register pair.
F103	7E		MOV	A, M	Store the first number in accumulator.
F104	23		INX	H	Increment H-L register pair to locate second number.
F105	86		ADD	M	Addition of 1st and 2nd number.
F106	27		DAA		Decimal adjust.
F107	32, 02, F0		STA	F002	Store result in F002H location.
F10A	76		HLT		Halt.

Example

DATA		RESULT	
Memory location	**Data**	**Memory location**	**Data**
F000	22H	F002	89H
F001	67H		

4.11.8 Subtraction of Two 8-bit Numbers

Data are stored in memory location 8000H and 8001H. The result after subtraction will be stored in 8002. Consider the program is written from memory location 8500H.

Algorithm

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

Program

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

Example

DATA		RESULT	
Memory location	**Data**	**Memory location**	**Data**
8000	89H	8002	47H
8001	42H		

4.11.9 Subtraction of Two 16-bit Numbers

Consider 1st number in DE registers pair and the 2nd number is in BC register pair. After subtraction result will be stored in 9000H onwards. Assume MSBs of 1st number is greater than 2nd number. Assume program starts from 8500H memory location.

Algorithm

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

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8500	21, 00, 80		LXI	H, 8000	Load address of first number in H-L register pair.
8503	5E		MOV	E, M	Load first two byte number in the DE register
8504	23		INX	H	pair. LSBs in E register and MSBs in D register.
8505	56		MOV	D, M	
8506	23		INX	H	Increment content of H-L pair for address of second number.
8507	4E		MOV	C, M	Load second two byte number in BC register pair.
8508	23		INX	H	LSBs in C register and MSBs in B register.
8509	46		MOV	B, M	
850A	7B		MOV	A, E	Transfer LSBs of first number in accumulator.
850B	B9		CMP	C	Compare between LSBs of second number and LSBs of first number.
850C	DA, 00, 85		JC	LEVEL_1	If carry is generated, jump to LEVEL_1.
850F	7B		MOV	A, E	Transfer LSBs of first number in accumulator.
8510	91		SUB	C	Find E-C.
8511	32, 00, 90		STA	9000	Store LSBs, the result of (E-C) in 9000H location.
8514	7A		MOV	A, D	MSBs of second number in accumulator
8515	98		SUB	B	Find D-B.
8516	32, 01, 90		STA	9001	Store MSBs , D-B in 9001 location.
8519	76		HLT		Halt.

851A	79	LEVEL_1	MOV	A, C	Transfer LSBs of first number in accumulator.
851B	2F		CMA		Get the complement of C = $\overline{C}$.
851C	83		ADD	E	
851D	C6, 01		ADI	01H	Determine E + $\overline{C}$ + 1 pair.
851F	32, 00, 90		STA	9000	Store LSBs the result of E+ $\overline{C}$ +1 in 9000H location.
8522	7A		MOV	A, D	Transfer MSBs of first number in accumulator.
8523	90		SUB	B	Subtract B from accumulator.
8524	D6, 01		SUI	01H	Subtract 01H from accumulator. Find D-B-1.
8526	32, 01, 90		STA	9001	Store MSBs, the result of D-B-1 in 9001H location.
8529	76		HLT		

Example

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

4.11.10 One's Complement of an 8-bit Number

The number is stored in memory location 8050H and one's complement of the number will be stored in location 8051H. Assume the program memory starts from 8000H.

Algorithm

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

Program

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

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

Example

DATA		RESULT	
Memory location	**Data**	**Memory location**	**Data**
8050	F0H	8051	0FH

4.11.11 One's Complement of a 16-bit Number

A 16-bit or 2-byte number is stored in the memory location 9001 and 9002 locations. The result will be stored in the memory location 9003H and 9004H. Consider the program memory starts from 9010H.

Algorithm

1. Load memory location of LSBs of data, 9001H in H-L registers pair.
2. Move LSBs data into accumulator.
3. Complement accumulator.
4. Store one's complement of LSBs in memory location 9003H.
5. Load memory location of MSBs of data, 9001H in H-L registers pair.
6. Move MSBs data into accumulator.
7. Complement accumulator.
8. Store one's complement of MSBs in the memory location 9004H.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
9010	21, 01,90		LXI	H, 9001H	Load the address of LSBs of data.
9013	7E		MOV	A, M	Move LSBs to accumulator.
9014	2F		CMA	CMA	Complement of LSBs of data.
9015	32, 03, 90		STA	9003H	Complement of LSBs of data in 9003H.
9018	23		INX	H	Increment H-L register pair.
9019	7E		MOV	A, M	Move MSBs to accumulator.
901A	2F		CMA	CMA	Complement of MSBs of data.
901B	32, 04, 90		STA	9004H	Complement of MSBs of data in 9004 H.
901E	76		HLT		Halt.

Example

DATA		RESULT	
Memory location	**Data**	**Memory location**	**Data.**
9001	52H	9003	89H LSBs of result.
9002	85H	9004	01H MSBs of result.

4.11.12 Two's Complement of an 8-bit Number

The number is stored in memory location 8500H. The two's complement will be stored in 8501H. The program is written from memory location 8510H.

Algorithm

1. Transfer the content of memory location 8500H to accumulator.

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

Program

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

Example

DATA		RESULT	
Memory location	**Data**	**Memory location**	**Data**
8500	F0H	8501	10H

4.11.13 Shift an 8-bit Number Left by One Bit

The number is stored in 8000H and its one bit left shift will be stored in 8001H. Assume the program is written from memory location 8010H.

Algorithm

1. Load memory location of data 8050H in H-L registers pair.
2. Move data from memory to accumulator.
3. Content of accumulator rotate left by one bit.
4. Store the result in memory location 8051H.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8010	21, 50, 80		LXI	H, 8050	Load memory address of data 8050H in H-L pair.
8013	7E		MOV	A, M	Move data in accumulator.
8014	07		RLC		Content of accumulator rotate left by one bit.
8015	23		INX	H	Increment HL register pair.
8016	77		MOV	M, A	Store the result in memory location 8051H.
8017	76		HLT		Halt.

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

Example

DATA		RESULT	
Memory location	**Data**	**Memory location**	**Data**
8050	04H	8051	08H

4.11.14 Find out the Largest of Two Numbers

The first number is stored in memory location 8050H and the second number is placed in the 8051H location. The result will be stored in the memory location 8052. Assume the program memory starts from 8100H.

Algorithm

1. Load the first number in the accumulator from memory location 8050H.
2. Compare 2nd number with 1st number.
3. If 2nd number is greater than 1st number, copy 2nd number in accumulator from memory.
4. Store result in 8052H location.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8100	21, 50, 80		LXI	H, 8050	Load the 1st number in accumulator from
8103	7E		MOV	A, M	memory location 8050H.
8104	23		INX	H	Address of 2nd number in HL pair.
8105	BE		CMP	M	Compare between 2nd number and 1st number.
8106	D2, 0A, 81		JNC	LEVEL	If borrow (carry) is not generated, jump LEVEL.
8109	7E		MOV	A,M	Move 2nd number in accumulator.
810A	32	LEVEL	STA	8052	Store largest number in 8052H.
810B	76		HLT		Halt.

Example

DATA		RESULT	
Memory location	**Data**	**Memory location**	**Data**
8050	78H	8052	FFH
8051	FFH		

4.11.15 Find out the Largest Number from an Array of Numbers

The count value of numbers 05H is stored in C register directly and the numbers are stored in the memory locations from 9001 to 9005. The largest number will be stored in the 9006 location. Assume the program memory starts from 9100H. The flow chart to find out the largest number from an array is depicted in Fig. 4.13.

Algorithm

1. Load count value of numbers 05H in C register immediately.
2. Load the first number in accumulator from memory location 9001H.
3. Move first number in accumulator.

4. Decrement the count value by one.
5. Move to next memory location for next data.
6. Compare the content of memory with content of accumulator.
7. If carry is generated, copy content of memory in accumulator.
8. Decrement the count value by one.
9. If count value does not equal to zero, repeat step 5 to step 8.
10. Store result in 9006H location.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
9100	0E, 05		MVI	C, 05	Load cout value in C register.
9102	21, 01, 90		LXI	H, 9001	Load addressof first data in HL register pair.
9105	7E		MOV	A, M	Copy 1st data in accumulator.
9106	0D		DCR	C	Decrement C register.
9107	23	LOOP	INX	H	Increment HL register for address of next data.
9108	BE		CMP	M	Compare next data with the content of accumulator.
9109	D2, 0D, 91		JNC	LEVEL	If carry is not generated, jump to LEVEL.
910C	7E		MOV	A, M	Copy large number in accumulator from memory.
910D	0D	LEVEL	DCR	C	Decrement C register.
910E	C2, 07, 91		JNZ	LOOP	Jump not zero to LOOP.
9111	32, 06, 90		STA	9006	Store largest number in 9006H location.
9114	76		HLT		

Example

DATA		RESULT	
Memory location	**Data**	**Memory location**	**Data**
9001	23H	9006	FFH
9002	FFH		
9003	47H		
9004	92H		
9005	10H		

4.11.16 Find out the Smallest Number from an Array of Numbers

A series of five numbers: 01H, FFH, 27H, 44H, 65H are stored in memory locations from 8001 to 8005. The smallest number will be stored in 8006 location. Assume the program memory starts from 8100H.

Algorithm

1. Store count of numbers 05H in C register immediately.
2. Load the 1st number in accumulator from memory location 8001H.
3. Decrement the count value by one.
4. Move to next memory location for next data.
5. Compare the content of memory with content of accumulator.
6. If carry is not generated, copy the content of memory in accumulator.

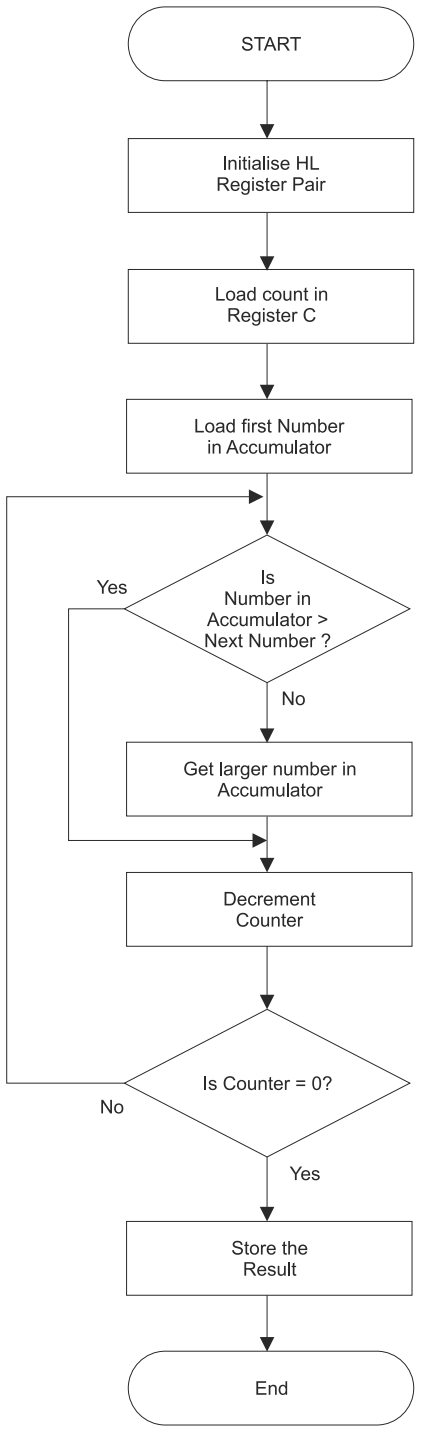


Fig. 4.13 Flow chart to find out the largest number from an array

7. Decrement the count value by one.
8. If count value does not equal to zero, repeat steps 4 to 8.
9. Store result in 8006H location.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8100	0E, 05		MVI	C, 05	Load cout value in C register.
8102	21, 01, 80		LXI	H, 8001	Load address of 1st number in HL register pair.
8105	7E		MOV	A, M	Copy 1st number in accumulator.
8106	0D		DCR	C	Decrement C register.
8107	23	LOOP	INX	H	Increment HL register for address of next number.
8108	BE		CMP	M	Compare next number with the content of accumulator.
8109	DA, 0D, 81		JC	LEVEL	If carry is generated, jump to LEVEL.
810C	7E		MOV	A, M	Copy large number in accumulator from memory.
810D	0D	LEVEL	DCR	C	Decrement C register.
810E	C2, 07, 81		JNZ	LOOP	Jump not zero to LOOP.
8111	32, 06, 80		STA	8006	Store smallest number in 9006H location.
8104	76		HLT		

Example

DATA		RESULT	
Memory location	**Data**	**Memory location**	**Data**
8001	01H	8006	01H
8002	FFH		
8003	27H		
8004	44H		
8005	65H		

4.11.17 Arrange a Series of Numbers in Descending Order

A series of five numbers: 11H, 05H, 46H, 23H, 65H are stored in memory locations from 9001H to 9005 H. Arrange the above numbers in descending order and to be stored in 9001H to 9005H locations. Assume the program memory starts from 9100H.

Algorithm

1. Store 05H, number of data to be arranged in C register from memory and store number of comparisons in D register.
2. Initialise the memory location 9001H of 1st data.
3. Load the 1st data in accumulator from memory.
4. Increment H-L register pair for addressing next data.
5. Load the next data in B register from memory.
6. Compare next data with accumulator. Store the smallest number in accumulator and largest number in memory.

7. Then next number is compared with accumulator and store the largest number in memory and smallest number in accumulator.
8. This process will continue, till comparison of all numbers have been completed. After completion of comparison of all numbers, the smallest number in accumulator is stored it in memory. In this way, the first process will be completed.
9. At the starting of second process, C register is decremented and store number of comparisons in D register. Then repeat step-2 to step-8. After completion of this process, smallest number is stored in 9005H and second smallest number is stored in 9004H.
10. C register is decremented and the next process starts if the content of C register is not zero.

The flow chart for arranging a series of numbers in descending order is depicted in Fig. 4.14.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
9100	0E, 05		MVI	C, 05H	Load count value of number of data in register C register.
9102	16, 05	START	MVI	D, 05	Load count for number of comparisons in D register.
9104	21, 01, 90		LXI	H, 9001H	Load memory location of 1st number.
9107	7E		MOV	A, M	1st number in accumulator.
9108	23	LOOP	INX	H	Increment HL register pair for addressing next number.
9109	46		MOV	B, M	Copy next number in B register from memory.
910A	B8		CMP	B	Compare next number with accumulator.
910B	DA, 14, 91		JC	LEVEL_1	If the content of accumulator > next number, Jump to LEVEL_1.
910E	2B		DCX	H	Decrement HL register pair to locate the addressing for storing largest number.
910F	77		MOV	M, A	Store largest of the two numbers in memory.
9110	78		MOV	A, B	Move smallest of the two numbers in accumulator from B register.
9111	C3, 16, 91		JMP	LEVEL_2	Jump to LEVEL_2.
9114	2B	LEVEL_1	DCX	H	Store largest of the two numbers in memory.
9115	70		MOV	M, B	
9116	23	LEVEL_2	INX	H	
9117	15		DCR	D	Decrement D register to count for number of comparisons.
9118	C2, 08, 91		JNZ	LOOP	Jump zero.
911B	77		MOV	M, A	Place smallest number in memory.
911C	0D		DCR	C	Decrement count value.
911D	C2, 02, 91		JNZ	START	Jump not zero to START.
9120	76		HLT		Halt.

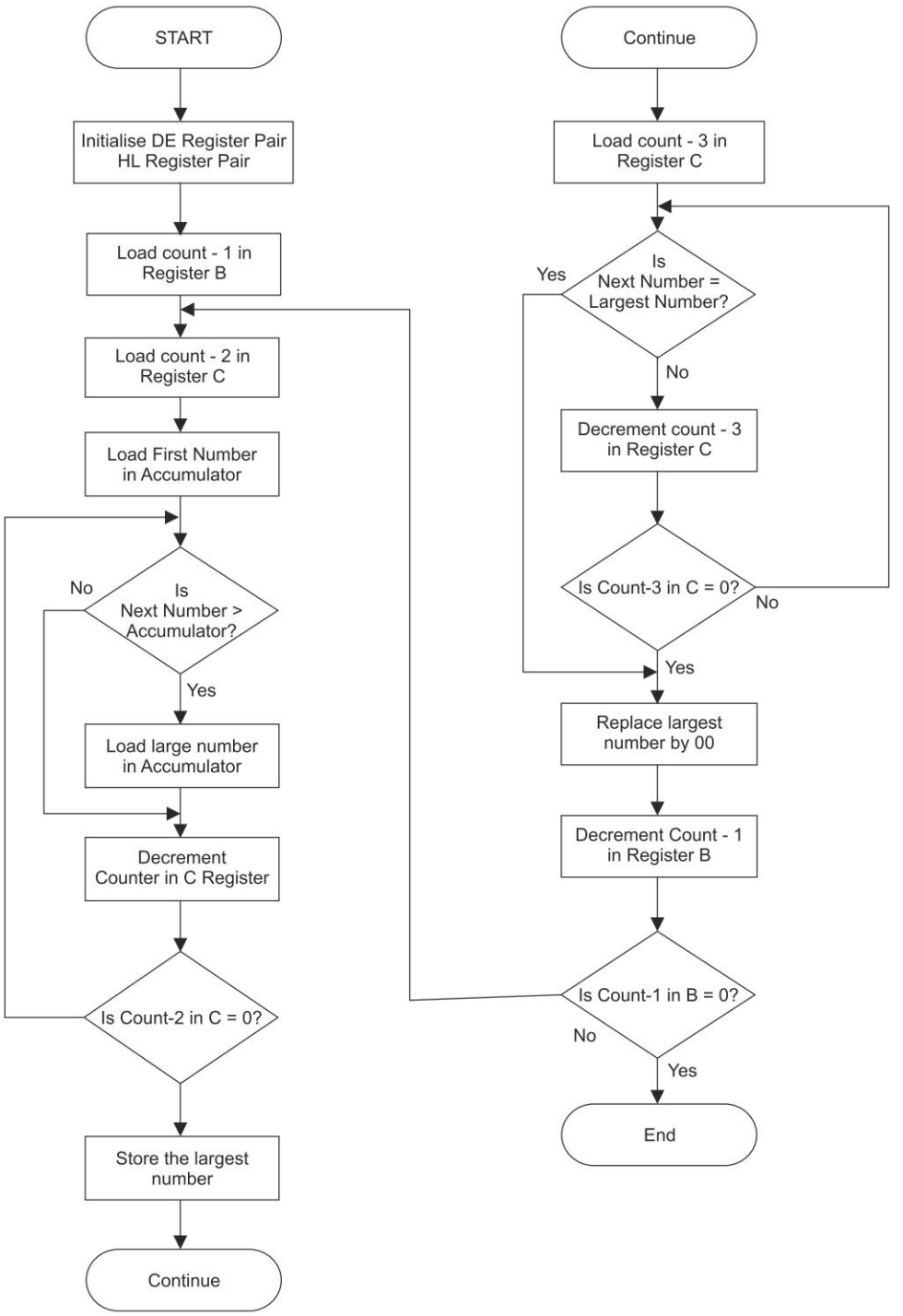


Fig. 4.14 Flow chart for arranging a series of numbers in descending order

Example

DATA	RESULT					
Memory location	**Data**	**Memory location**	**After 1st process**	**After 2nd process**	**After 3rd process**	**After 4th process**
9001	11H	9001	11H	46H	46H	65H
9002	05H	9002	46H	23H	65H	46H
9003	46H	9003	23H	65H	23H	23H
9004	23H	9004	65H	11H	11H	11H
9005	65H	9005	05H	05H	05H	05H

4.11.18 Arrange a Series of Numbers in Ascending Order

A series of five numbers: F2H, 05H, 88H, 23H, 65H are stored in memory locations from 9001H to 9005H. Arrange the above numbers in ascending order and to be stored in 9001H to 9005H locations. Assume the program memory starts from 9100H.

Algorithm

1. Store 05H, number of data to be arranged in C register from memory and store number of comparisons in D register.
2. Initialise the memory location 9001H of 1st data.
3. Load the 1st data in accumulator from memory.
4. Increment H-L register pair for addressing next data.
5. Load the next data in B register from memory.
6. Compare next data with accumulator. Store the largest number in accumulator and smallest number in memory.
7. Then next number is compared with accumulator and store the smallest number in memory and largest number in accumulator.
8. This process will continue, till comparison of all numbers have been completed. After completion of comparison of all numbers, the largest number in accumulator and store it in memory. In this way, first process will be completed.
9. At the starting of second process, C register is decremented and store number of comparisons in D register. Then repeat step-2 to step-8. After completion of this process, largest number in 9005H and second largest number in 9004H.
10. C register is decremented and the next process starts, if the content of C register is not zero.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
9100	0E, 05		MVI	C, 05H	Load count value of number of data in register C register.
9102	16, 05	START	MVI	D, 05	Load count for number of comparisons in D register.
9104	21, 01, 90		LXI	H, 9001H	Load memory location of 1st number.
9107	7E		MOV	A, M	1st number in accumulator.
9108	23	LOOP	INX	H	Increment HL register pair for addressing next number.

9109	46		MOV	B, M	Copy next number in B register from memory.
910A	B8		CMP	B	Compare next number with accumulator.
910B	D2, 14, 91		JNC	LEVEL_1	If the content of accumulator > next number, Jump to LEVEL_1.
910E	2B		DCX	H	Decrement HL register pair to locate the addressing for storing smallest number.
910F	77		MOV	M, A	Store smallest of the two numbers in memory.
9110	78		MOV	A, B	Move largest of the two numbers in accumulator from B register.
9111	C3, 16, 91		JMP	LEVEL_2	Jump to LEVEL_2.
9114	2B	LEVEL_1	DCX	H	Store smallest of the two numbers in memory.
9115	70		MOV	M, B	
9116	23	LEVEL_2	INX	H	
9117	15		DCR	D	Decrement D register to count for number of comparisons.
9118	C2, 08, 91		JNZ	LOOP	Jump zero to zero.
911B	77		MOV	M, A	Place largest number in memory.
911C	0D		DCR	C	Decrement count value.
912D	C2, 02, 91		JNZ	START	Jump not zero to START.
9120	76		HLT		Halt.

Example

DATA		**RESULT**				
Memory location	**Data**	**Memory location**	**After 1st process**	**After 2nd process**	**After 3rd process**	**After 4th process**
9001	F2H	9001	05H	05H	05H	05H
9002	05H	9002	88H	23H	23H	23H
9003	88H	9003	23H	65H	65H	65H
9004	23H	9004	65H	88H	88H	88H
9005	65H	9005	F2H	F2H	F2H	F2H

4.11.19 Find out Square of a Decimal Number using Look-Up Table

Load the decimal number in memory location F000H and the square of decimal number will be stored in the memory location F001H. The square values of decimal numbers from 0 to 9 are stored in F110 to F119H in tabular form as depicted in Look-up table. Assume the program is written from memory location F150H.

Algorithm

1. Store the decimal number in accumulator from memory location F000H.
2. Move the content of accumulator in L register and Load F1H in H register.
3. If the decimal number is 05, the content of H and L register are F1 and 05H respectively. Then the memory location F105H will be denoted by H-L register pair.

4. Move square of decimal number in accumulator from memory location represented by H-L register pair.
5. Store the result, square value in F001H.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
F150	3A, 00, F0		LDA	F0000H	Load the decimal number in accumulator from memory location F000H.
F153	6F		MOV	L, A	Copy the content of accumulator in L register.
F154	26, F1		MVI	H, F1	Load F1H in H register.
F156	7E		MOV	A, M	Move square of decimal number in accumulator from memory.
F157	32, 01, F0		STA	F001	Store square value in F001H.
F15A	76		HLT		Halt.

Look-up Table

ADDRESS	SQUARE (decimal)
F100 H	00
F101 H	01
F102 H	04
F103 H	09
F104 H	16
F105 H	25
F106 H	36
F107 H	49
F108 H	64
F109 H	81

Example

ADDRESS	SQUARE (decimal)	ADDRESS	Result (decimal)
F000 H	05	F001 H	25

4.11.20 Find out Square Root of 0, 1, 4, 9, 16, 25, 36, 49, 64 and 81 using Look-Up Table

Load the number in memory location 9000H and the square root of the number will be stored in the memory location 9001H. The square root of numbers 0, 1, 4, 9, 16, 25, 36, 49, 64 and 81 are stored in 8500H, 8501H, 8504H, 8509H, 8516H, 8525H, 8536H, 8549H, 8564H and 8581H locations respectively as given in tabular form. Assume the program is written from memory location 9100H.

Algorithm

1. Store the number in accumulator from memory location 9000H.
2. Move the content of accumulator in L register and store 85H in H register.
3. When the number is 16, the content of H and L register are 85 and 16H respectively. Then the H-L register pair represents 8516H memory location.

4. Copy square root of number in accumulator from memory location which is represented by H-L register pair.
5. Store the result in 9001H.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
9100	3A, 00, 90		LDA	9000H	Load the number in accumulator from memory location 9000H.
9103	6F		MOV	L, A	Copy the content of Accumulator in L register
9104	26, 85		MVI	H, 85	Load 85H in H register.
9106	7E		MOV	A, M	Move square root of decimal number in accumulator from memory.
9107	32, 01, 90		STA	9001H	Store square root value in 9001H.
910A	76		HLT		

Look-up Table

ADDRESS	SQUARE (decimal)
8500	00
8501	01
8504	02
8509	03
8516	04
8525	05
8536	06
8549	07
8564	08
8581	09

Example

ADDRESS	SQUARE	ADDRESS	Result
9000 H	16	9001 H	04

4.11.21 Multiplication of Two 8-bit Numbers

Two eight-bit data are stored in 8000H and 8001H memory locations. After multiplication, the result will be stored in 9000H and 9001H memory location. Assume the program is written from memory location 9100H.

Repetitive-Addition Algorithm

1. Store multiplicand in B register and multiplier in E register.
2. Clear D register and clear H-L register pair.
3. Content of D-E register will be added with content of H-L register.
4. Decrement B register.
5. If content of B register is not equal to zero, repeat Step 3 to 5.
6. When B register is equal to zero, the content of H-L will be stored in memory location 9000H and 9001H.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
9100	21, 00, 80		LXI	H, 8000H	Address of multiplier in H-L pair.
9103	4E		MOV	C, M	Store multiplier in C register from memory.
9104	24		INX	H	Address of multiplicand in H-L pair.
9105	5E		MOV	E, M	Multiplicand in E register.
9106	16, 00		MVI	D, 00H	Load 00H in D register.
9108	21, 00, 00		LXI	H, 0000	Initial value of product = 00 in H-L pair.
910B	19	LOOP	DAD	D	Add content of DE with content of HL.
910C	0D		DCR	C	Decrement C register.
910D	C2, 0B, 91		JNZ	LOOP	If not zero, jump to LOOP.
9110	EB		XCHG		The content of DE register pair and H-L register pair exchanged. Result is stored in DE register.
9111	21, 00, 90		LXI	H, 9000H	Load 9000H in H-L pair.
9114	73		MOV	M, E	Store content of E register in 9000H location.
9115	23		INX	H	Address of next memory location in H-L pair.
9116	72		MOV	M, D	Store content of E register in 9001H location.
9117	76		HLT		Stop.

Example

ADDRESS	DATA	ADDRESS	Result
8000 H	45H Multiplicand	9000 H	05H
8001 H	13H Multiplier	9001 H	FFH

Binary Multiplication

The other method of multiplication is binary multiplication. If the multiplicand is multiplied by 1, the product will be equal to the multiplicand. If the multiplicand is multiplied by zero, the product is zero.

For binary multiplication, the following procedure is followed:

Example

```
    45 H Multiplicand
  × 13 H Multiplier
  ----------------
    05 1FH Product

      0 1 0 0 0 1 0 1
    × 0 0 0 1 0 0 1 1
  ------------------
      0 1 0 0 0 1 0 1
    0 1 0 0 0 1 0 1 x
  0 0 0 0 0 0 0 0 x x
0 0 0 0 0 0 0 0 x x x
0 1 0 0 0 1 0 1 x x x x
----------------------
0 1 0 1 0 0 0 1 1 1 1 1     (05 1F)H
```

The multiplicand and multiplier are represented in binary form. The product is also in the binary form. When a multiplicand is multiplied by 1, the product is equal to the multiplicand. When a multiplicand is multiplied by zero, the product is zero. The procedure for multiplication is given below:

Step-1 The multiplicand is multiplied by the LSB of the multiplier and the partial product is stored. Then the multiplicand is shifted right.

Step-2 Again the shifted multiplicand is multiplied by the 2nd bit and then added with the previous result. Then the shifted multiplicand is shifted right. If the bit is a '0' bit, nothing will be added with the partial product but the multiplicand is simply shifted right by one bit.

Step-3 The step will be repeated till the completion of multiplication of all bits of the multiplier.

In binary multiplication, the multiplicand is shifted right and shift multiplier left to check the LSB bit whether it is '1' or '0'. Flow chart for multiplication of two numbers is shown in Fig. 4.15.

Algorithm

1. Load multiplicand and multiplier.
2. Initialize product value = 0.
3. Load number of bits of multiplier in C register.
4. Shift multiplier right by one bit.
5. If carry flag is set, multiplicand adds with initial value 0000H + multiplicand. Then product is equal to 0000H +multiplicand. This result is also called as partial product. Then partial product shifted left by one bit.
6. DCR counts value.
7. If the content of C register not zero, modified multiplier again shifted one bit right.
8. If carry flag is set, shifted multiplicand adds with partial product. Then one again shifts modified multiplicand left.
9. Repeat step 6, 7 and 8 till the content of C register becomes zero.
10. Store the result in 9000H and 9001H.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
9100	21, 00, 80		LXI	H, 8000H	Address of multiplicand in H-L pair.
9103	5E		MOV	E, M	Store Multiplicand in E register from memory.
9104	23		INX	H	Address of multiplicand in H-L pair.
9105	56		MOV	D, M	Multiplicand in D register.
9106	0E, 08		MVI	C, 08H	Load 08H in C register.
9108	3A, 02, 80		LDA	8002H	Load multiplier in accumulator.
910B	21, 00, 00		LXI	H, 0000H	Initial value of product = 00 in H-L pair.
910E	0F	LOOP	RRC		Rotate accumulation left.
910F	D2, 13, 91		JNC	LEVEL	If there is no carry, jump to level.
9112	19		DAD	D	Add content of DE with content of HL.
9113	EB	LEVEL	XCHG		The content of DE register pair and HL register pair exchanged. Result is stored in DE register.

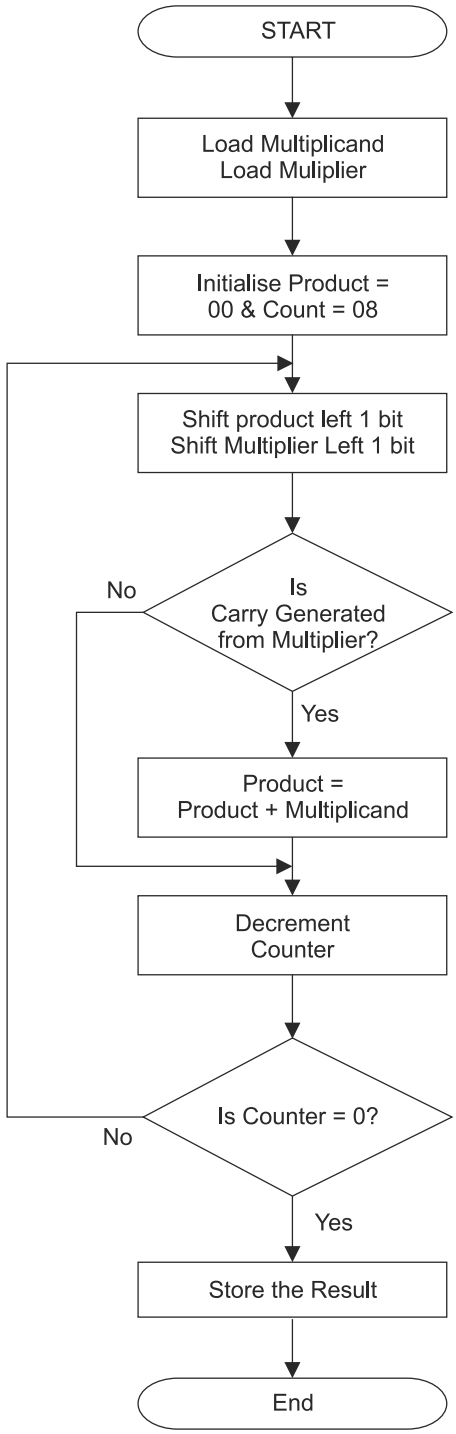


Fig. 4.15 Flow chart for multiplication of two numbers

9114	19	DAD	H	Multiplicand shifted one bit right.
9115	EB	XCHG		The content of DE register pair and HL register pair exchanged. Result is stored in DE register.
9116	37	STC		Clear the carry flag using set carry status and then complement the carry status.
9117	3F	CMC		
9118	0D	DCR	C	Decrement C register.
9119	C2, 0E, 91	JNZ	LOOP	If content of C register is not zero, jump to LOOP.
911C	22, 00, 90	SHLD	9000H	Store the content of H-L register pair in 9000H and 9001 memory location.
911F	76	HLT		Stop.

Example

ADDRESS	DATA	ADDRESS	Result
8000 H	45H Multiplicand	9000 H	05H
8001 H	13H Multiplier	9001 H	FFH

4.11.22 Division of Two 8-bit Numbers

Repetitive Subtractions The division can be performed by repetitive subtractions. The divisor is subtracted from the dividend. When there is no borrow, the quotient is incremented by one. If there is borrow, the quotient and reminder are stored in a specified memory location. Assume dividend and divisor are of nonzero quantity. Assume the program starts from memory location 9100H.

Algorithm

1. Store dividend in memory location 8000H and divisor in memory location 8001H.
2. Clear C register by store 00H within it.
3. Move dividend in accumulator and copy it in D register.
4. Subtract divisor from dividend.
5. If carry is not generated, Increment C register. Repeat Step 3 to 5.
6. When carry is generated, store quotient, content of C register and remainder content of D register in memory location.
7. If zero flag is set, C register incremented by one. Then store quotient, content of C register and remainder content of D register in memory location.

The flow chart for division of two numbers is illustrated in Fig. 4.16.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
9100	21, 00, 80		LXI	H, 8000H	Address of dividend in H-L pair.
9103	36, 22 (Dividend)		MVI	M, Dividend	Store dividend in memory location.
9105	23		INX	H	Address of divisor in H-L pair.
9106	36, 21 (Divisor)		MVI	M, Divisor	Store divisor in memory location.
9108	0E, 00		MVI	C, 00H	Load 00H in C register for initial value of quotient.

910A	2B		DCX	H	Decrement H-L register pair.
910B	7E	LOOP	MOV	A,M	Load dividend in accumulator from memory.
910C	57		MOV	D,A	Copy dividend in D register.
910D	23		INX	H	Increment H-L register pair.
910E	96		SUB	M	Subtract divisor from dividend.
910F	DA, 1B, 91		JC	LEVEL_1	If there is carry, jump to LEVEL_1.
9112	CA, 20, 91		JZ	LEVEL_2	If there is zero, jump to LEVEL_2.
9115	2B		DCX	H	Decrement H-L register pair.
9116	77		MOV	M,A	Store modified dividend in memory location from accumulator.
9117	0C		INR	C	Increment C register.
9118	C3, 0B, 91		JMP	LOOP	
911B	37	LEVEL_1	STC		Clear the carry flag using set carry status and then complement the carry status.
911C	3F		CMC		
911D	C3, 21, 91		JMP	LEVEL_3	Jump to LEVEL_3.
9120	0C	LEVEL_2	INR	C	
9121	21, 00, 90	LEVEL_3	LXI	H, 9000H	
9124	71		MOV	M, C	Store quotient in 9000H from C register.
9125	23		INX	H	Increment H-L register pair.
9126	72		MOV	M, D	Store remainder in 9000H from C register.
9127	76		HLT		Stop.

Binary Division Binary division is also performed by trial subtractions. The divisor is subtracted from the 8 most significant bits of the dividend. When there is no borrow, the bit of the quotient is set to 1; otherwise 0. Then the dividend and quotient are shifted left by one bit before the next subtraction. The dividend and quotient can use a 16-bit register. As dividend is shifted, one bit of the register falls vacant in each step and the quotient is stored in unoccupied bit positions.

The dividend is a 16-bit number and the divisor is an 8-bit number. When the dividend is an 8-bit number, place 00H in MSBs positions. The dividend is stored in the memory locations 8000H and 8001 H. The divisor is placed in the memory location 8002 H. The results will be stored in the memory locations 8003 and 8004 H.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
9100	2A, 00, 80		LHLD	8000 H	Store dividend in H-L pair.
9103	3A, 02, 80		LDA	8002H	Store divisor in accumulator from memory location 8002H.
9106	47		MOV	B, A	Copy the content of accumulator in B register.
9107	0E, 08		MVI	C, 08	Load 08H in C register.
9109	29	LOOP	DAD	H	Shift dividend and quotient right by one bit.
910A	7C		MOV	A, H	Move MSBs of dividend in accumulator.
910B	90		SUB	B	Subtract divisor from dividend.
910C	DA, 11, 91		JC	LEVEL_1	If there is carry, jump to LEVEL_1.
910F	67		MOV	H, A	After subtraction, store dividend in H register from accumulator.

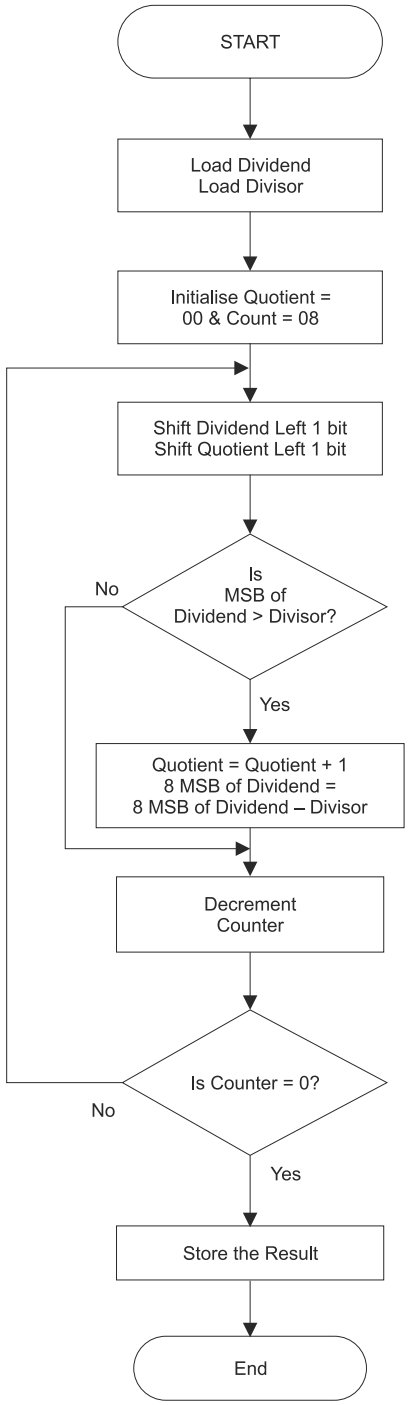


Fig. 4.16 Flow chart for division of two numbers

9110	2C		INR	L	Increment L register.
9111	0D	LEVEL_1	DCR	C	Decrement C register.
9112	C2, 12, 91		JNZ	LOOP	If there is no zero, jump to LOOP.
9115	22, 03, 80		SHLD	8003H	Store results in 8003 and 8004 H.
9118	76		HLT		Stop.

Example

ADDRESS	DATA	ADDRESS	Result
8000 H	9A H LSBs of dividend	9003H	F2 Quotient
8001 H	48 H MSBs of dividend	9004 H	06 Remainder
8002 H	1A H Divisor		

4.11.23 Convert an 8-bit Hexadecimal Number to a Binary Number

Store an 8-bit data in 8000H location and load 08H in C register. Transfer data from memory to accumulator. The content of accumulator is shifted right with carry and store carry bit in 8050H location. After that clear carry, and again content of accumulator is shifted right with carry and content of carry will be stored in previous memory location. In this way, the operation will be repeated till C register becomes zero.

Assume the program starts from the memory location 8100H.

Algorithm

1. Initialise memory location 8000H. Load 8-bit hexadecimal number in the memory.
2. Initialise the memory location to store result.
3. Load 8-bit data in accumulator.
4. Load 08H in C register.
5. Rotate accumulator right with carry.
6. Copy content of accumulator in E register.
7. Save carry in accumulator.
8. Store in memory.
9. Transfer E register to accumulator.
10. Clear carry.
11. Decrement H-L pair.
12. Decrement C register.
13. Repeat steps 6 to 12 till the content of C register becomes zero.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8100	21, 00, 80		LXI	H, 8000H	Initialise the memory location 8000 H through HL register pair.
8103	36, DATA		MVI	M, DATA	Load DATA in the memory.
8105	21, 50, 80		LXI	H, 8050H	Initialise the memory location 8050 H to store binary.
8108	3A, 00, 80		LDA	8000 H	Load data in accumulator from the memory location 8000 H.

810B	0E, 08		MVI	C, 08H	Store 08 H in C register.
810D	1F	LOOP	RAR		Rotate the content of accumulator right with carry.
810E	5F		MOV	E, A	Copy the content of accumulator in register E.
810F	3E, 00		MVI	A, 00H	Store 00 H in accumulator.
8111	8F		ADC	A	Add the contents of accumulator and carry.
8112	77		MOV	M, A	Move the content of accumulator in memory.
8113	7B		MOV	A, E	Transfer the content of accumulator in register E.
8114	B7		ORA	A	Clear carry.
8115	2B		DCX	H	Decrement H-L register pair.
8116	0D		DCR	C	Decrement C register.
8117	C2, 0D, 81		JNZ	LOOP	If content of C is not zero, Jump to LOOP.
811A	76		HLT		Halt.

Example

8 Bit DATA	Binary number in Memory Location							
	8051H	**8052H**	**8053H**	**8054H**	**8055H**	**8056H**	**8057H**	**8058H**
27	0	0	1	0	0	1	1	1
9F	1	0	0	1	1	1	1	1

4.11.24 Transfer a Block of Data from One Section of Memory to the Other Section of Memory

A block of data is available starting from 9051H. Tansfer the block so that it can be stored from 9100H. The number of bytes in the block is stored in 9050H. Assume program starts from 8000H.

Algorithm

1. Store the address of number of data in H-L register pair.
2. Load number of data in C register from memory.
3. Store the starting address of destination in DE register pair.
4. Increment H-L register pair to get data from source.
5. Copy data from source to accumulator.
6. Exchange H-L and DE register pair, store the content of accumulator in destination address.
7. Exchange H-L and DE register pair.
8. Increment H-L and DE register.
9. Decrement C register.
10. If C register is not zero, repeat steps 5 to 9.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	21, 50, 90		LXI	H, 9050	Store the address of number of data, 9050H in HL register pair.
8003	46		MOV	B, M	Load number of data in B register from memory.
8004	21, 00, 91		LXI	D, 9100	Store the destination address in DE register pair.

8007	23		INX	H	Increment H-L register pair.
8008	7E	LOOP	MOV	A, M	Move data from source to accumulator.
8009	EB		XCHG		Exchange the content of HL and DE.
800A	77		MOV	M, A	Store the content of accumulator, data in destination address.
800B	23		INX	H	Increment source address.
800C	13		INX	D	Increment destination address.
800D	05		DCR	B	Decrement B register.
800E	C2, 08, 80		JNZ	LOOP	If B is not zero, Jump to LOOP.
8011	76		HLT		

Example

Input		Result	
ADDRESS	**DATA**	**ADDRESS**	**DATA**
9050	05 H	9100	48 H
9051	48 H	9101	1A H
9052	1A H	9102	F2 H
9053	F2 H	9103	06 H
9054	06 H	9104	33H
9055	33H		

4.11.25 Display Digits 0 1 2 3 4 5 6 7 8 9 A B C D E F on the Data Field of Screen

This program will display a flashing 0 1 2 3 4 5 6 7 8 9 A B C D E F on the data field. The flashing rate is 500 ms. Consider the program is written from memory location 8000H and data is stored from 8C50H.

Algorithm

1. Initialise the stack pointer.
2. Clear the display.
3. Point to the data which will be displayed in the data field.
4. Wait for 50 milliseconds.
5. Clear the display and wait for 500 ms.
6. Jump to step-3.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	31, FF, 8C		LXI	SP, 8CFF	Initialise the stack pointer.
8003	0E, 0F		MVI	C, 0F	Load the counter, C register with 0FH.
8005	21, 50, 8C		LXI	H, 8C50	Starting address where the display data is stored.
8008	3E, 01		MVI	A, 01	A is 01 and B is 00 to display character in the data field.
800A	06, 00		MVI	B,00	

800C	E5	LOOP_I	PUSH	H	Move the content of HL, BC, PSW and A registers into stack.
800D	C5		PUSH	B	
800E	F5		PUSH	PSW	
800F	CD, 47, 03		CALL	CLEAR	Clear the display by using a subroutine whose address is 0347H.
8012	11, 00, 00		LXI	D,0000H	Generate 500 ms delay.
8015	CD, BC, 03		CALL	DELAY	
8018	CD, D0, 05		CALL	OUTPUT	Display character in data field.
801B	CD, 47, 03		LXI	D,0000H	To display the number for 500 ms.
801E	11, 00, 00		CALL	DELAY	
8021	F1		POP	PSW	POP the content of PSW, A, BC and HL from stack.
8022	C1		POP	B	
8023	E1		POP	H	
8024	23		INX	H	Increment HL register pair to display next number.
8025	0D		DCR	C	Decrement C register.
8026	C2, 0C, 80		JNZ	LOOP_I	Jump to LOOP_I if the content of C is not equal to zero.

Input

ADDRESS	DATA	ADDRESS	DATA
8C50	00 H	8C58	08 H
8C51	01 H	8C59	09H
8C52	02 H	8C5A	0AH
8C53	03 H	8C5B	0B H
8C54	04 H	8C5C	0C H
8C55	05 H	8C5D	0D H
8C56	06 H	8C5E	0E H
8C57	07 H	8C5F	0F H

4.11.26 Rolling Display "HELP 85" on the Address and Data Field of Screen

This program will display a flashing "HELP 85 up" in the address and data field. The flashing rate is 500 ms. Assume the program is written from memory location 9000H and data is stored from 9C50H.

Algorithm

1. Initialise the stack pointer.
2. Clear the display.
3. Point to the data of "HELP 85" and display "HELP 85" in the address and data field.
4. Wait for 500 ms.
5. Clear the display and wait for 500 ms.
6. Jump to start.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
9000	31, FF, 9C		LXI	SP, 9CFF	Initialise the stack pointer.
9003	CD, 47, 03		CALL	CLEAR	Clear the display by using a subroutine whose address is 0347H.
9006	0E, 0A	START	MVI	C, 06	Load the counter, C register with 06.
9008	AF		XRA	A	A is 00 to display character in the address field.
9009	47		MOV	B, A	
900A	21, 50, 9C		LXI	H, 9C50	Starting address where the display is to be started.
900D	E5	LOOP_ I	PUSH	H	Move the content of HL, BC, PSW and A registers into stack.
900E	C5		PUSH	B	
900F	F5		PUSH	PSW	
9010	CD, D0, 05		CALL	OUTPUT	Call OUTPUT subroutine to display characters in address field.
9013	3E, 01		MVI	A, 01	Initialise A, B for data to be displayed in data field.
9015	06, 00		MVI	B, 00	
9017	CD, D0, 05		CALL	OUTPUT	Call OUTPUT subroutine to display characters in data field.
901A	21, 00, 00		LXI	D, 0000H	Generate 500 ms delay.
901D	CD, BC, 03		CALL	DELAY	
9020	F1		POP	PSW	POP the content of PSW, A, BC and HL from stack.
9021	C1		POP	B	
9022	E1		POP	H	
9023	23		INX	H	Increment H-L register pair to display next number
9024	0D		DCR	C	Decrement C register.
9025	C2, 0D, 90		JNZ	LOOP_ I	Jump to LOOP_I if the content of C is not equal to zero.
	C3, 06, 90		JMP	START	Jump to START.

Input

ADDRESS	DATA	ADDRESS	DATA
9C50	16 H	9C55	10H (H)
9C51	16 H	9C56	0E H (E)
9C52	16 H	9C57	11H (L)
9C53	16 H	9C58	12H (P)
9C54	16 H	9C59	08 H (8)
		9C5A	05 H (5)

Review Questions

4.1. Explain the following terms:
(a) assembly language (b) machine language
(c) high-level language (d) compiler

4.2. What are the disadvantages of machine languages? What the advantages of assembly languages?

4.3. What are translators? Write the differences between compiler and interpreter.

4.4. Define stack. Explain function of PUSH and POP instructions.

4.5. Define macro and explain operation of macro with examples. What is the difference between macro and subroutine?

4.6. Explain time delay loop using register and register pair. Write some applications of time delay loop. Calculate the time required to execute the following two instructions if the system clock frequency is 1MHz.

```
LOOP:  MOV A, B    5 T-states
       JMP LOOP    10 T-states
```

4.7. What is modular programming? Write advantages and disadvantages of modular programming.

4.8. Write an assembly language program to load memory locations 8100H and 8200H with 58H and 42H. Add the contents and store it in memory location 8300H.

4.9. Data byte 28H is stored in register B and data byte 97H is stored in the accumulator. Show the contents of registers B, C and the accumulator after the execution of the following instructions:
(a) MOV C, A (b) MOV A, B (c) ADD B

4.10. Find the sum of 10 numbers stored in successive memory locations starting from 2000H and store the result in two bytes 8100H and 8101H.

4.11. A series of sixteen bytes of data are stored in memory locations from 9000 H to 900F H. Write an assembly-language program to transfer the entire block of data bytes to new memory locations starting from 9100H.

4.12. Calculate the l's and 2's complement of the contents of two successive memory locations 8500H and 8501H and store them in four consecutive memory locations starting from 8600H.

4.13. *N* numbers are stored in consecutive memory locations starting from 8001H and the value of *N* is available in memory location 8000H. Find the maximum of *N* numbers and stored in 9001.

4.14. *N* numbers are stored in consecutive memory locations starting from 8001H and the value of *N* is available in memory location 8000H. Sort the numbers in ascending order and stored in memory location starting from 9001.

4.15. Calculate the square of the contents of memory location 9100H using lookup table and place the result in memory location 9501.

4.16. A block of 32 bytes data is stored at the memory location starting from 8000H. Move this block to the memory location starting from 9000H.

4.17 Write an assembly language program to detect a even and odd numbers.

4.18 Write an assembly-language program for converting temperature from F to C degree.

4.19 Write an assembly-language program for shifting an 16-bit number left by two bit.

4.20 Write an assembly-language program to arrange a series of numbers in descending order using a subroutine.

Multiple-Choice Questions

4.1 The assembler is
(a) a program that translates mnemonics into binary code
(b) a program that translates mnemonics into octal code
(c) an operating system which manages all the programs in the system
(d) a compiler that translates statements from assembly language program into machine language

4.2 The compiler is
(a) faster than the interpreter (b) slower than the interpreter
(c) an interpreter (d) a single-step process

4.3 When a subroutine is called by the CALL instruction, the microprocessor stores the 16-bit address of the instruction next to CALL on the
(a) stack pointer (b) accumulator
(c) program counter (d) stack

4.4 When a CALL instruction is executed, the stack pointer register is
(a) decremented by two (b) incremented by two
(c) decremented by one (d) incremented by one

4.5 When the RET instruction is executed at the end of a subroutine,
(a) the memory address of the RET instruction is transferred to the program counter
(b) two data bytes stored in the top locations of the stack are transferred to the stack pointer
(c) the data where the stack is initialised is transferred to the stack pointer
(d) two data bytes stored in the top two locations of the stack are transferred to the program counter

4.6 Whenever the PUSH H instruction is executed,
(a) data bytes in the H-L pair are stored on the stack
(b) two data bytes at the top of the stack are transferred to the H-L register pair
(c) two data bytes at the top of the stack are transferred to the program counter
(d) two data bytes from H-L register that were previously stored on the stack are transferred back to the H-L register

4.7 Whenever the POP H instruction is executed,
(a) two data bytes from H-L register which were previously stored on the stack, are transferred back to the H-L register
(b) data bytes in the H-L pair are stored on the stack
(c) two data bytes at the top of the stack are transferred to the program counter
(d) data bytes in the H-L pair are stored on the program counter

4.8 Opcode is
(a) the part of the Instruction which tells the computer what operation to perform
(b) an auxiliary register that stores the data to be added or subtracted from the accumulator
(c) the register that receives the constructions from memory
(d) is the data which will be used in data manipulation of instruction

Register A contains 5FH, B contains 4FH, C contains 26H, H contains 80H, L contains FFH and the

memory location 80EE and 80FF contains 2D and 4E respectively. The following program begins at memory location 8110H

```
ADD C
MOV B, M
MOV M, A
DAD B
```

4.9 What will A contain after the program?
(a) 85 H (b) 5 FH (c) FFH (d) 4 AH

4.10 What will B contain after the program?
(a) 4E H (b) AA H (c) DD H (d) 4 AH

4.11 What will H contain after the program?
(a) CF H (b) AB H (c) DA H (d) 4 AH

4.12 What will L contain after the program?
(a) 25 H (b) AA H (c) DD H (d) 4 AH

4.13 What will be the content of memory location 80EE H after the program?
(a) 2D H (b) AA H (c) DD H (d) 85 H

4.14 What will be the content of memory location 80FF after the program?
(a) 85 H (b) AD H (c) D 0H (d) 4 FH

4.15 The PC contains 8452h and SP contains 88D6H. What will be the content of PC and SP following a CALL to subroutine at location 82AFH?
(a) 82AF, 88D4 (b) 82AF, 8450 (c) 8450, 88D4 (d) 82AF, 8452

4.16 What will be the delay generated by the following instructions?

```
        MOV C, FF     10 T
LOOP,   DCR C         5 T
        JNZ LOOP      10 / 7T
```

(a) 3832 T States (b) 3850 T States
(c) 3857 T States (d) 3835 T States

Answers to Multiple-Choice Questions

4.1 (d)	4.2 (a)	4.3 (d)	4.4 (a)
4.5 (d)	4.6 (a)	4.7 (a)	4.8 (a)
4.9 (a)	4.10 (a)	4.11 (a)	4.12 (a)
4.13 (a)	4.14 (b)	4.15 (a)	4.16 (c)

CHAPTER

5

Memory and Interfacing with 8085 Microprocessor

5.1 INTRODUCTION

The microprocessor is a very powerful IC, which is used to perform various ALU functions with the help of data from the environment. For this, the microprocessor is connected with memory and input/output devices to form a microcomputer. The technique of connection between input/output devices and a microprocessor is known as interfacing. Special attention must always be given during the connection of pins of peripheral devices and microprocessor pins, as ICs cannot be simply connected. In the development of a microprocessor-based system, all memory ICs and input/output devices are selected as per requirement of the system and then interfaced with the microprocessor. Actually address, data and control lines are used for connecting peripherals. After connecting properly, programs are written in the microprocessor. Programs will be different for different applications. When the program is executed, the microprocessor communicates with input/output devices and performs system operations. In this chapter, the interfacing of memory devices are explained.

5.2 MEMORY INTERFACING

Memory devices are used to store digital information. The simplest type of digital memory device is the flip-flop, which is capable for storing single bit data, and is volatile and very fast. This device is generally used to store data in the form of registers. Registers are also used as main memory of computers for internal computational operations. The basic goal of digital memory is to store and access binary data, which is a sequence of 1's and 0's. In this section, different types of memory and its interfacing with a microprocessor is explained.

5.3 TYPES OF MEMORY

There are two types of semiconductor memories, namely, ROM and RAM. The ROM stands for read only memory. Data are permanently stored in memory cells from where we are able to read data. ROM cannot be reprogrammed. This memory is nonvolatile and data is retained when power is switched off. But the data contents of ROM are accessed randomly just like the volatile memory circuits. Vinyl records and compact audio disks are typically referred as read-only memory, or ROM, in the digital system.

5.3.1 Classification of ROM

ROMs are manufactured with bipolar technology and MOS technology. Figure 5.1 shows the classification of ROM. The different features of ROM, PROM, EPROM and EEPROM are explained below:

ROM (Read Only Memory) The data is permanently stored in the memory and these devices are mask programmed during manufacturing. ROMs cannot be reprogrammed and are of nonvolatile type. These devices are cheaper than programmable memory devices. The applications of ROM are fixed programmed instructions, look-up tables, conversions, and some specific operations.

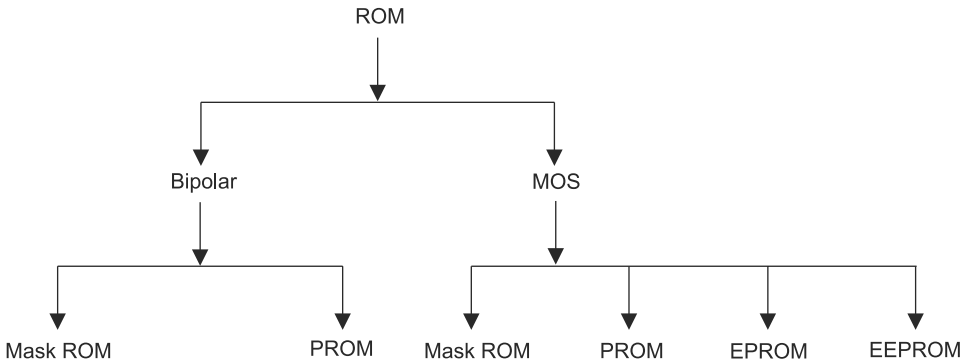


Fig. 5.1 Classification of ROM

PROM (Programmable Read Only Memory) The data can be electrically stored. It can be programmed by blowing built-in fuses. PROM can be reprogrammed and is of nonvolatile type. These memory devices are of very low memory density and occupy more space.

EPROM (Erasable Programmable Read Only Memory) These are strictly a MOS device and programmed by storing charge on insulated gates. These devices are erasable with ultraviolet rays and reprogrammable after erasing. These memory devices are of nonvolatile type.

EEPROM (Electrically Erasable Programmable Read Only Memory) These memory devices are electrically programmable by the programmer and the stored data can be erased by ultraviolet light. This is of nonvolatile type. This is also called Electrically Alterable Programmable Read Only Memory (EAPROM).

5.3.2 Classification of RAM

Read only memory is used only for reading data stored in the memory. ROMs can be programmed only once and data once recorded cannot be erased. In a RAM, data can be written into its memory as often as desired and the data stored in a RAM can be read without destroying the contents of the memory.

Data can be written into and read from a RAM at any selected address in any sequence. When data are written into a given address in the RAM, the data previously stored at that address are destroyed and replaced by the new data. When data are read from a given address in the RAM, the data at that address are not destroyed. The non-destructive read operation can be thought of as copying the contents of an address while leaving the content intact. A vinyl record platter is an example of a random-access device. RAM memory is typically randomly accessed; it is actually virtually/volatile memory.

There are two types of RAMs, static and dynamic. The basic memory cell in a *static RAM* is a flip-flop, bipolar or MOS. After a bit has been stored in the flip-flop of a memory cell, it will remain there while power

is available. *Dynamic RAMs* called DRAMs are based on charge, which is stored by using MOS devices. Since this charge is dissipated by passage of time, DRAMs need periodical recharging or refreshing. RAMs and Dynamic RAMs are volatile devices. The comparison between different memories based on category, erasing property, writing mechanism and volatility is illustrated in Table 5.1.

Table 5.1 Comparison of memories

Types of Memory	Category	Erasing property	Writing mechanism	Volatility
Read only Memory (ROM)	Read only Memory	Not possible	Masks	Nonvolatile
Programmable ROM (PROM)		Not possible	Electrically	Nonvolatile
Erasable PROM (EPROM)		Ultra-violet light and chip level	Electrically	Nonvolatile
Electrically Erasable PROM (EPROM)		Electrically and byte level	Electrically	Nonvolatile
Flash memory		Electrically and block level	Electrically	Nonvolatile
Random access memory (RAM)	Read and write memory	Electrically and byte level	Electrically	Volatile

RAMs are also manufactured with bipolar technology and MOS technology. The bipolar RAMs are static RAMs but MOS RAMs are static and dynamic type. Figure 5.2 shows the classification of RAM. The different features of static and dynamic RAM are explained below.

Static RAMs These RAMs are built with static or dynamic cells. Five or six transistors are used to store a single bit. Data can be written and read in nanoseconds. Usually TTL, ECL, NMOS and CMOS technology are used to manufacture static RAMs. When the power is shut off, data stored in cells can be lost.

Dynamic RAMs In a dynamic memory, data can be stored on capacitors and to retain data every cell has to be refreshed periodically. One transistor is used to build a memory cell and requires less space. These memories consume less power compared to static RAMs. The comparison between SRAM and DRAM is given below:

Table 5.2 Comparison of Static RAM and Dynamic RAM

Static RAM	Dynamic RAM
• Stored data is retained as long as power is ON.	• Stored data will be erased and repeated refreshing is required to store data.
• Stored data will not be changed with time.	• Stored data will be changed with time.
• Consume more power.	• Consume less power than static RAM.
• SRAM is expensive.	• SRAM is less expensive.
• SRAM has less packing density.	• DRAM has high packing density.
• These memories are not easy to construct.	• These memories construction are simple.
• No refreshing is required.	• As refreshing is required, additional circuit is incorporated with memory.
• No maintenance is required.	• Maintenance is required.

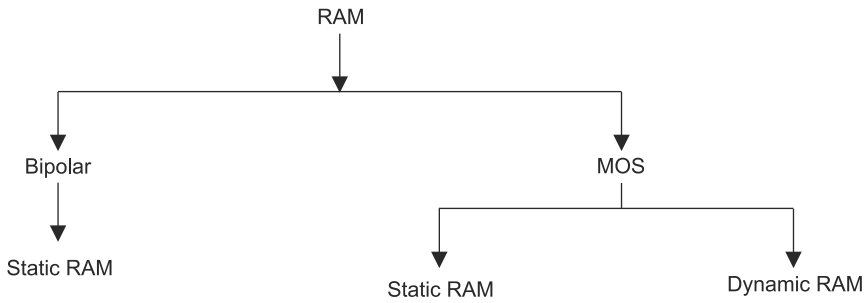


Fig. 5.2 Classification of RAM

5.4 MEMORY ORGANIZATION

The block diagram of a $M \times K$ bit memory structure is shown in Fig. 5.3. It has N bit input lines to locate a address of memory and each address line can store K bits. So the total number of bits in the memory is $M \times K$ bits. Each memory location is represented by address lines to locate M address locations. Here N bits inputs are required to locate M address locations. The relationship between address locations and input lines is $2^N = M$. To generate an address line, an N lines to M lines decoder is used. Actually, the decoder decodes M locations depending upon inputs. The number of locations and number of bits may be varied for different memories. When M is the number of locations and K is the number of bits in each location, the size of the memory is $M \times K$ bits. The size of commonly used memory devices are 64, 256, 512, 1024 (1K), 2048 (2K), 4096 (4K), 16384 (16K) but the common values of word size are 1, 2, 4, 8, 12, 16, etc. The Chip Enable (CE) signal is used to enable the address lines for selecting a bit or a group of bits.

Nowadays all the semiconductor memory devices are now available in integrated circuit (IC) form. Each memory IC can store a large number of words. Memory ICs are available in various sizes. The examples of ICs are 64 × 4 (64 words of 4 bits each), 256 × 8 (256 words of 8 bits each), 1 K × 8 (1024 words of 8 bits each), 1 M × 8 (1,048,576 words of 8 bits each).

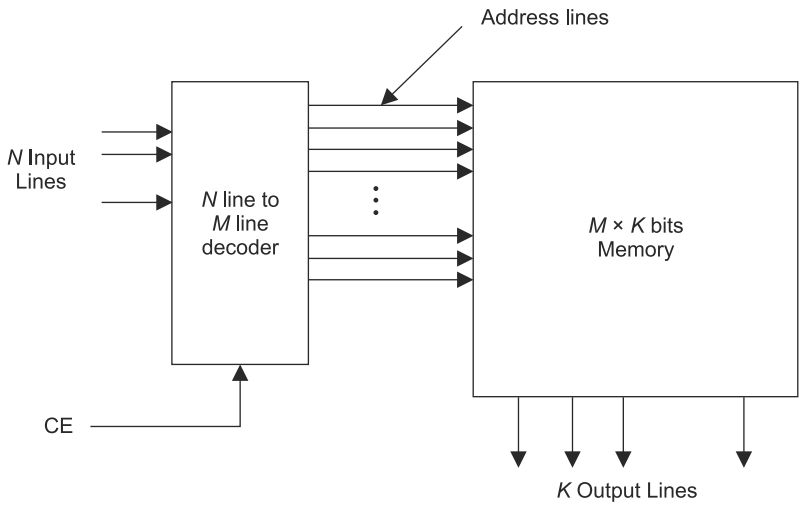


Fig. 5.3 Block diagram of $M \times K$ bits Memory

Each memory IC should have address and data lines including chip select $\overline{CS}$, output enable $\overline{OE}$ and Read/Write $R/\overline{W}$ control signals. Figure 5.4 shows the memory organization of an IC. This chip has address lines, data lines, chip select signals and read and write control signals which are explained below:

Address Lines The memory ICs should have address lines to receive the address values. For a 1K-byte memory, ten address lines A_0–A_9 exist. The relationship between number of address lines and size of memory is 2^n, where n is the number of address lines. Similarly, for a 64K byte memory, number of address line is 16, A_0 to A_{15}. Address line A_0–A_{n-1} be used to select one of the 2^n memory locations.

Data Lines Data lines provide for data input to the IC during write operation and data will be output from IC during read operation. M data lines $D_0 - D_{m-1}$ are used for data transfer between microprocessor and memory IC.

Chip Select Signal $\overline{CS}$ The chip select signal $\overline{CS}$ can enable the chip. When the $\overline{CS}$ is low, memory access within the chip is possible.

Read or Write $R/\overline{W}$ The read or write operation can be performed based on $R/\overline{W}$ control signal. If $R/\overline{W}$ =1, data will be read from memory. When $R/\overline{W}$ = 0, data will be stored in the memory IC.

Output Enable $\overline{OE}$ The output enable signal is used to connect the output with the data bus.

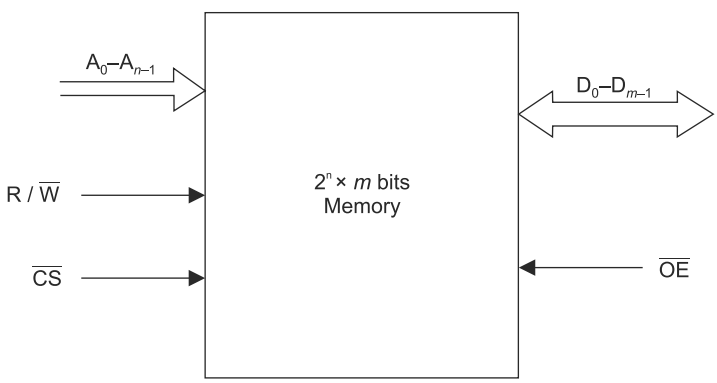


Fig. 5.4 Memory organization of $2^n \times m$ bits memory IC

For example, the memory organization of 256 × 4 memory IC is depicted in Fig. 5.5. This memory IC has 8 address lines A_0–A_7 to select 256 memory locations. As there are four data lines, 4 data bits can be stored in each location. Therefore, the size of the memory is 256 × 4 bits.

Memory ICs are available in four-bit and eight-bit word configurations. In some applications, sixteen bits and more than sixteen bits are also used. The memory capacity of each IC is limited. Therefore, memory expansion is required. The memory size can be expanded by increasing the word size and address locations. The memory expansion can also be possible by proper interconnections of decoder and memory ICs. Figure 5.6 shows 2K × 8 bits memory using two 1K × 8 bits.

A 2K-byte RAM can be developed using two 1K byte RAM ICs. In this case, $\overline{CS}$ is directly connected with IC_1 and the complement of $\overline{CS}$ is connected to IC_2. $R/\overline{W}$ and $\overline{OE}$ control signals of both ICs are directly interconnected as given in Fig. 5.6. The address lines A_0–A_9 of the ICs are connected in parallel. The chip-1 provides the 1K addresses from 0 to 1023 and the chip-2 provides the next 1K addresses from 1024 to 2047.

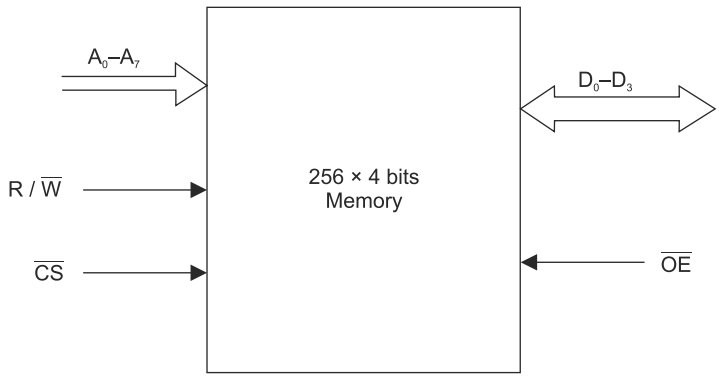


Fig. 5.5 Memory organization of 256 × 4 bits memory IC

As for the first 1K addresses, chip-I is activated and for the next 1K addresses, chip-2 is activated. The chip select signal is connected with the address line A_{10}. Each chip also provides 8 data lines D_0–D_7. So that memory size is increased from 1K byte to 2K byte.

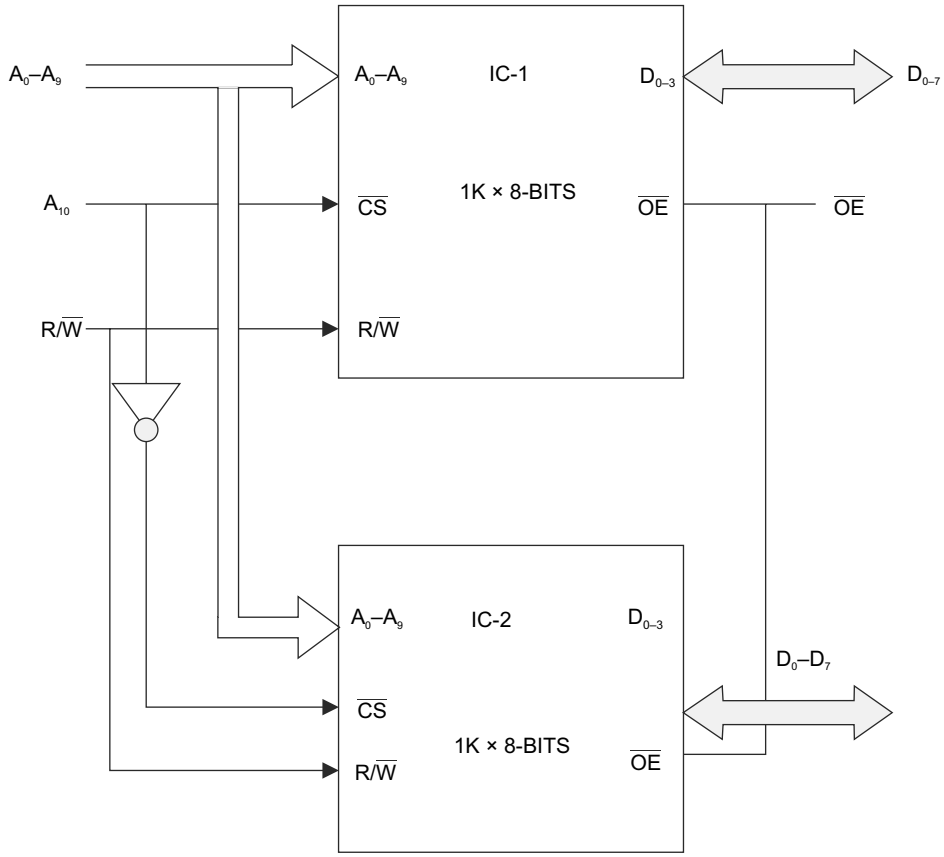


Fig. 5.6 Memory organization of 2K × 8 bits memory using two 1K × 8 bits

Another example is that two 1K × 4 bits RAMs can be combined to develop 1K bytes RAM as depicted in Fig. 5.7. The IC-1 and IC-2 have ten address lines, which are connected in parallel. The chip select $\overline{CS}$, read/write $R / \overline{W}$ and output enable $\overline{OE}$ are also connected together. In this case, memory size is fixed, but word size is increased from 4 bit to 8 bit. IC-1 and IC-2 are selected at a time for 8-bit data storage or data read operation.

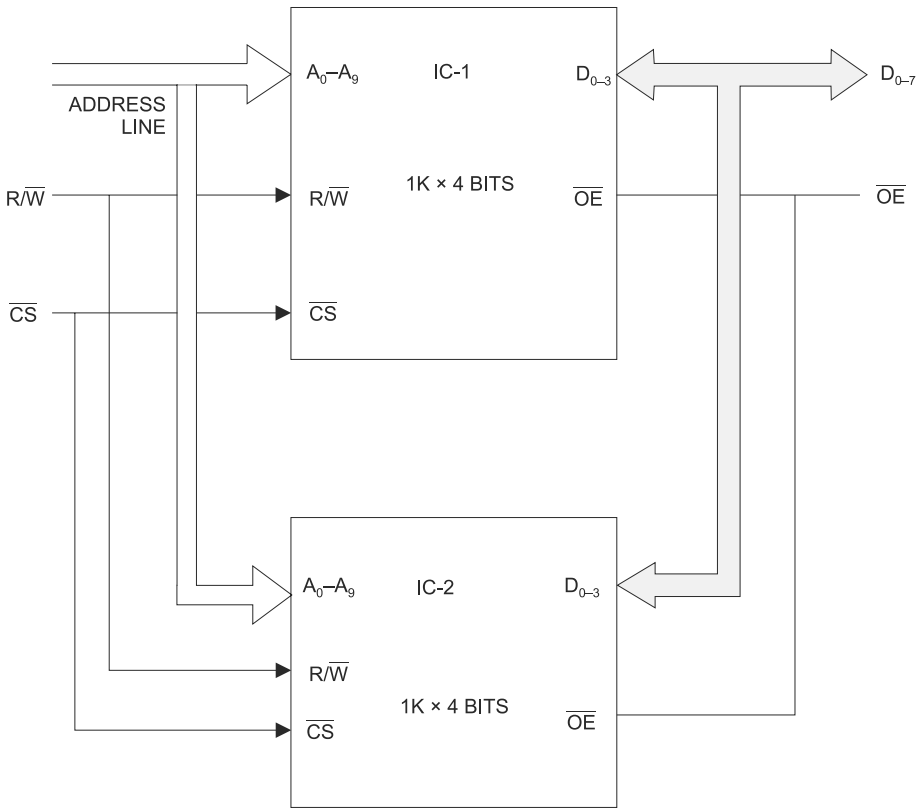


Fig. 5.7 Memory organization of 1K × 8 bits memory using two 1K × 4 bits

The memory organization of 2K bytes using four chips of 1K × 4 bits is illustrated in Fig. 5.8. In this case the memory sizes as well as word size are increased. The memory size is increased from 1K to 2K, and the word size is also increased from 4 bits to 8 bits. IC-1 and IC-2 are selected at a time for 8-bit data storage or data read operation. Similarly, IC-3 and IC-4 are used for 1K × 8 bits memory read/write operations.

5.5 ROM AND RAM ICs

The organization and operation of memory is already explained briefly in Section 5.4. Most commonly used ROM and RAM ICs are given in Tables 5.3 and 5.4 respectively with their category, organization, package, access time, technology and power dissipation, etc.

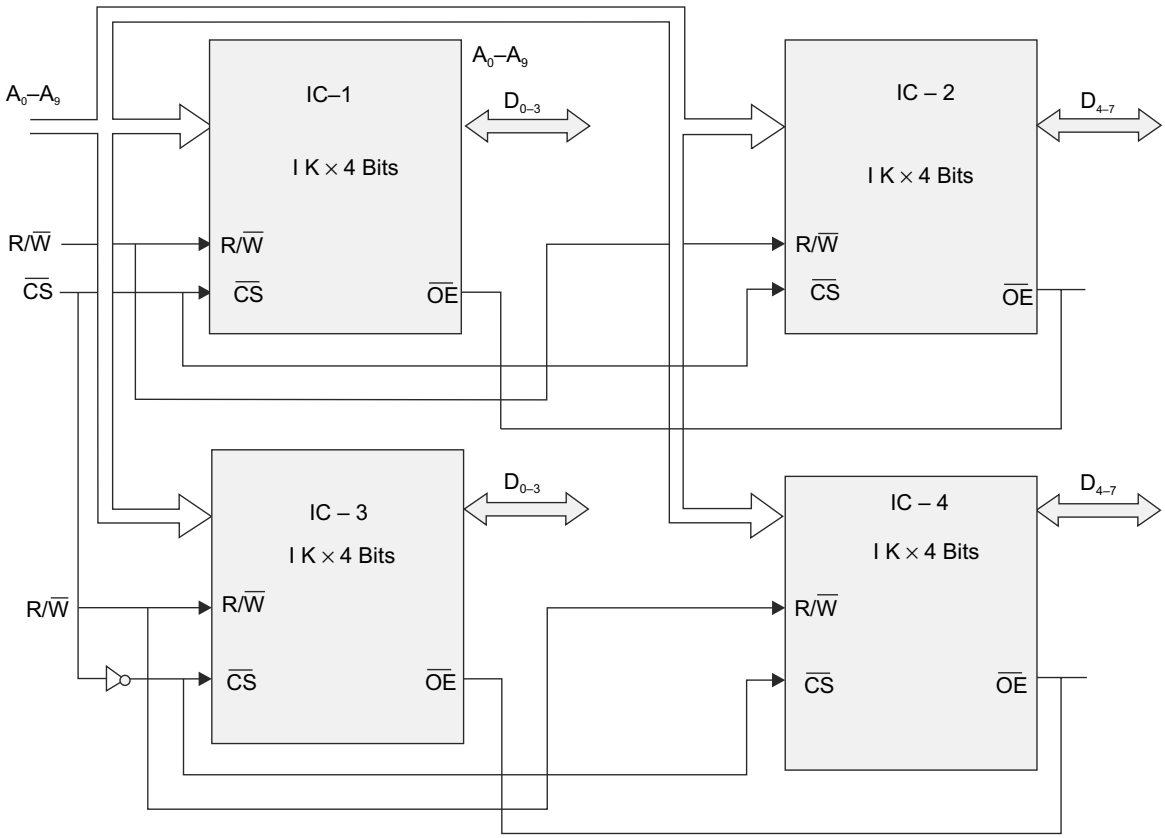


Fig. 5.8 Memory organization of 2K × 8 bits memory using four 1K × 4 bits.

Table 5.3 Commonly used ROM ICs

IC No.	Category	Package	Organization	Technology	Access time	Power dissipation
6206D	Mask PROM	16-pin DIP Package	512 × 4	Bipolar	60 ns	625 mW
23C1010	MASK ROM	32-PIN DIP/PDIP/SOP/ PLCC/TSOP Package	128K × 8	MOS	45 ns	250 mW
23C2000	MASK ROM	32-pin PDIP/PLCC/SOP/ TSOP Package	256K × 8	MOS	70 ns	250 mW
23C4000	MASK ROM	32-pin PDIP/PLCC/ SOP/ TSOP Package	512K × 8	MOS	90 ns	210 mW
23C6410	Mask ROM	44-pin SOP and 48 pin TSOP Package	8M × 8 4M × 16	MOS	100 ns	420 mW
Am1702A	PROM	24-pin duel in-line hermetic cerdip package	256 × 8	MOS	550 ns	1000 mW
3602A	PROM	16-pin DIP package	512 × 4	Bipolar	70 ns	750 mW
3605	PROM	18-pin DIP package	1024 × 4	Bipolar	70 ns	800 mW
27C256	EPROM	28-pin DIP package and a 32-pin windowed LCC	32,768 × 8	Low-power CMOS	250 ns	55 mW

Contd.

Table 5.3 (*Contd.*)

2716	EPROM	24-pin DIP package	2048 × 8	MOS	450 ns	525 mW
2732A	EPROM	24-pin DIP package	4096 × 8	MOS	250 ns	790 mW
2764	EPROM	28-pin DIP package	8192 × 8	MOS	250 ns	790 mW
24AA00/ 24LC00/ 24C00	Serial EEPROM	8L DIP, SOIC, TSSOP and 5L SOT-23 packages	16 bytes × 8 bits	Low-power CMOS	1000 ns	10 mW

Table 5.4 Commonly used RAM ICs

IC No.	Category	Package	Organization	Technology	Access time	Power dissipation
7489	Static RAM	16-pin DIP Package	16 × 4	Bipolar	33 ns	500 mW
2114	Static RAM	18-PIN DIP Package	2K × 4	MOS	200 ns	300 mW
74189	Static RAM	16-pin	16 × 4	Bipolar	50 ns	550 mW
74289	Static RAM	DIP Package	16 × 4	Bipolar	35 ns	250 mW
6116	Dynamic RAM	24-pin DIP, Thin Dip, SOIC and SOJ package	2K × 4	CMOS	15 ns	4 mW
4166	Dynamic RAM	16-pin DIP Package	16384 × 1	NMOS	200 ns	460 mW
2104A	Dynamic RAM	16-pin DIP Package	4096 × 1	MOS	150 ns	420 mW
2164		16-pin DIP Package	64K × 1	MOS	450 ns	330 mW

Fig. 5.9(a) EPROM

Fig. 5.9(b) RAM

5.6 MEMORY MAP

As 8085 microprocessor has 16 address lines, it has an address capability of 64K (2^{16} = 65,536) from 0000H to FFFFH. This 64K memory will be used by EPROMs, and RAMs ICs. The assignment of address to various memory ICs is known as memory map. The memory map of 2K EPROM and a 2K static RAM is depicted

in Fig. 5.10. The address of EPROM is from 0000H to 07FFH and the address of RAM is 8000H to 87FFH. If 2K RAM is not sufficient for programming; another 2K RAM may be connected from 8800H to 9000H as shown in Fig. 5.10(b). Figure 5.10(c) shows the memory map of a 4K EPROM and a 4K RAM. The address 0000H to 1000H are specified for EPROM and 4K RAM occupies addresses from 8000H to 9000H.

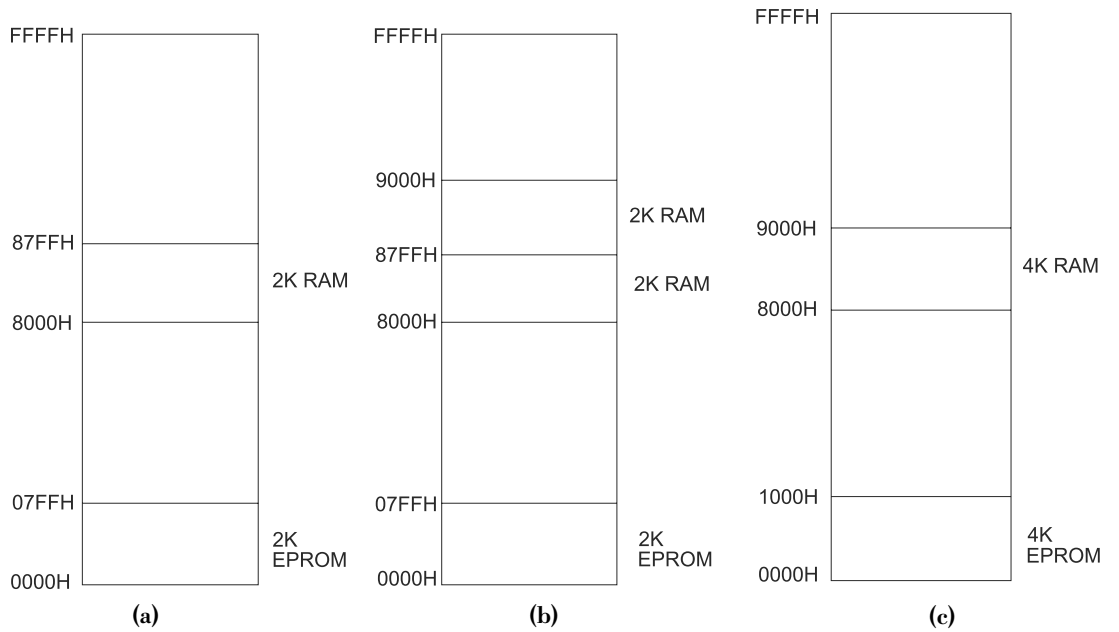


Fig. 5.10(a) Memory map of 2K EPROM and 2K RAM, (b) Memory map of 2K EPROM and 2 × 2K RAM, and (c) Memory map of 4K EPROM and 4K RAM

The 2K memories IC have 11 address lines A_{10}–A_0, which are used to locate the memory location where data will be stored or read. The other address lines A_{12}–A_{15} of the microprocessor can be used for the chip select signal. The memory map of 2K memories is given below:

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 = 0000H
⋮													⋮			
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1 = 07FFH

The memory map from 0000H to 07FFH can be expressed in terms of page analogy as given below:

Memory map	Page No.	Memory map	Page No
0000 – 00FF	Page 0	0400 – 04FF	Page 4
0100 – 01FF	Page 1	0500 – 05FF	Page 5
0200 – 02FF	Page 2	0600 – 06FF	Page 6
0300 – 03FF	Page 3	0700 – 07FF	Page 7

5.7 ADDRESS DECODING

Figure 5.11 shows the address decoding technique of 8085 microprocessor. $A_0 - A_{10}$ are used for addressing the EPROM IC. $A_{11} - A_{15}$ address lines are applied to a NAND gate to generate the chip select signal $\overline{CS}$. The memory map of EPROM is given below:

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 = 0000H
⋮													⋮		
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1 = 03FFH

Figure 5.12 shows the complete memory and address decoder circuit. In this case, a 3-line to 8-line decoder can be used to select any one output. Based on inputs at A_{11}, A_{12}, A_{13} any one output of O_0–O_7 will be active low and other output lines remain high. The output lines are connected to the chip select of memory ICs. It is depicted in Fig. 5.12 that O_0 is connected with the chip select of 2K EPROM and O_7 is connected with the 2 K RAM. The output lines and corresponding memory address capability is given in Table 5.5.

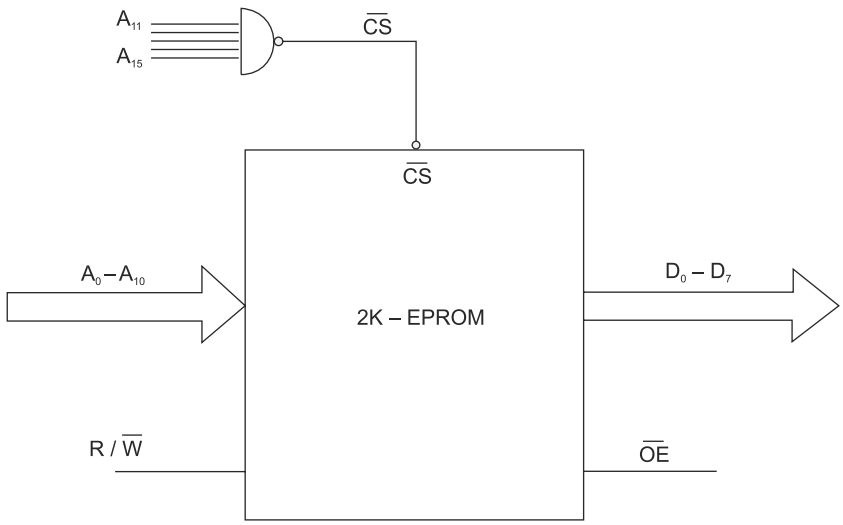


Fig. 5.11 Address decoding of 2K EPROM

Table 5.5 Memory address selected by the decoder

Output lines	Memory address
O_0	0000H-07FFH
O_1	0800H-0FFFH
O_2	1000H-17FFH
O_3	1800H-1FFFH
O_4	2000H-27FFH
O_5	2800H-2FFFH
O_6	3000H-37FFH
O_7	3800H-3FFFH

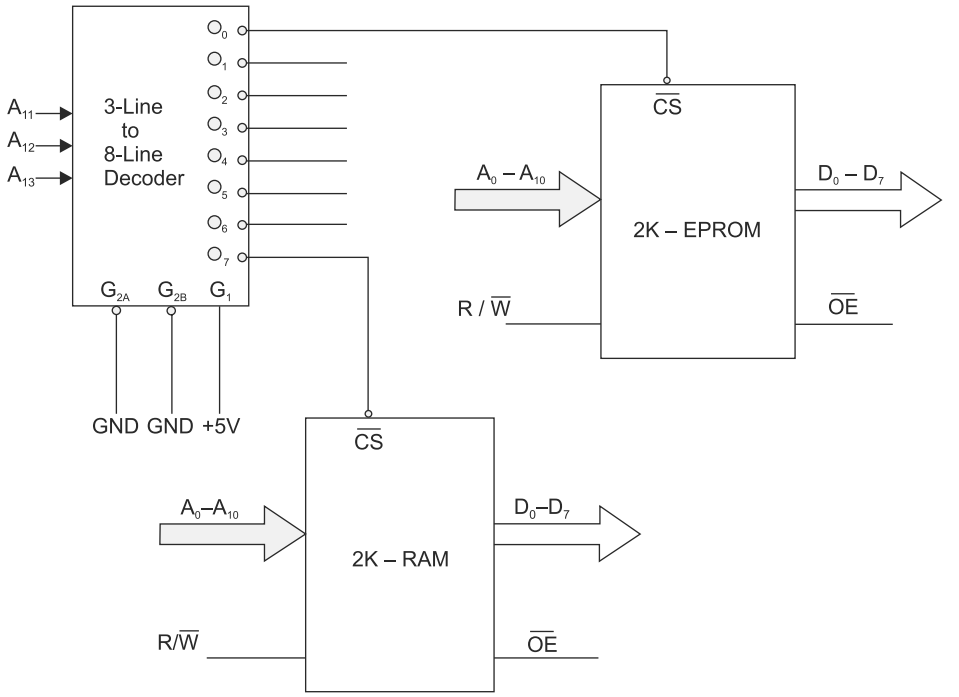


Fig. 5.12 Address decoding EPROM using decode

5.8 MEMORY INTERFACING TO MICROPROCESSOR

The 8085 microprocessor has higher order address bus $A_8 - A_{15}$ and a lower order address/data bus $AD_0 - AD_7$. The lower order address/data bus is multiplexed as *address bus* and *data bus*. During the first clock pulse of a machine cycle, the program counter releases the lower order address in $AD_0 - AD_7$ and higher order address $A_8 - A_{15}$. Then ALE signal is high; the $AD_0 - AD_7$ can be used as lower order address bus and not a data bus. The external latch circuit makes the difference between the address and data bus as shown in Fig. 5.13.

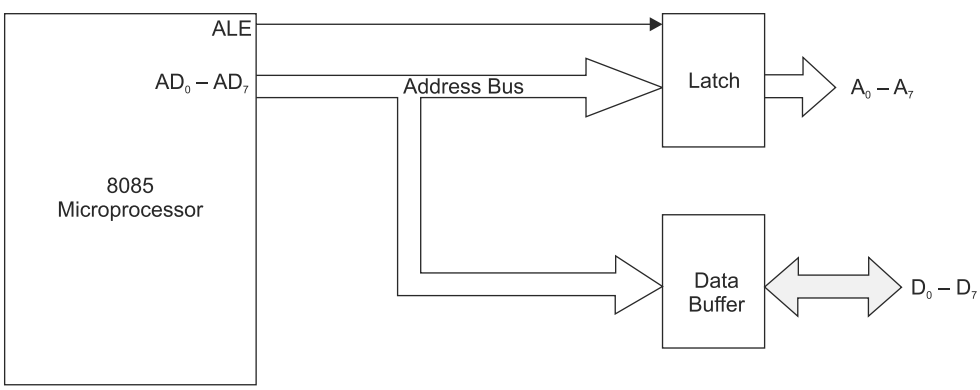


Fig. 5.13 Multiplexing of lower order address and data bus

The microprocessor communicates with various memory ICs. The interfacing between a microprocessor, memory and I/O devices through address bus, data bus and control bus is depicted in Fig. 5.14. Usually the address decoder is used to select proper memory and I/O devices.

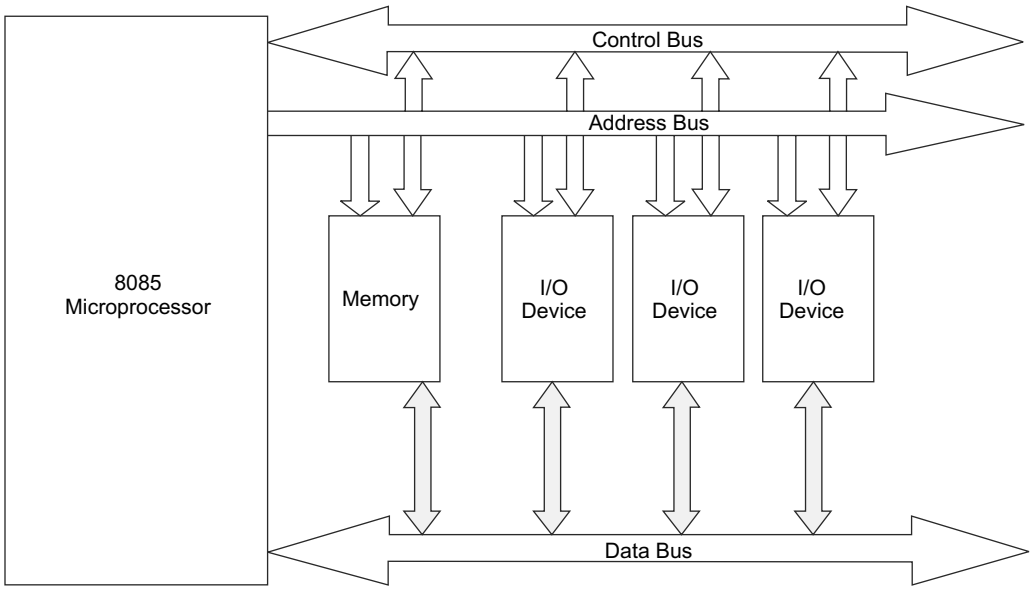


Fig. 5.14 Schematic block diagram for memory and I/O interfacing with microprocessor

When the address decoder is enabled and chip select signals are applied to decoder, RAM or EPROM or I/O devices are selected. Data will be stored or read from memory devices or I/O devices. The total 64K addresses are to be assigned to memories and I/O devices. There are two types of address mapping: memory mapped I/O and I/O mapped I/O.

In some microprocessors, memory and I/O operation can be differentiated by control signals. The control signal IO/$\overline{M}$ is available to distinguish between memory and I/O operations. When the control signal IO/$\overline{M}$ is high, I/O operation can be performed. If the control signal IO/$\overline{M}$ is low, memory operations will be performed. In this case, the same address can be assigned to I/O devices as well as memory location. Generally, two separate address spaces are exist and each address space can be entirely assigned to either memory or I/O devices. This technique is known as I/O-mapped-I/O.

In I/O-mapped-I/O scheme, I/O device cannot be considered as memory location. I/O-mapped-I/O scheme requires special instruction like IN/OUT to access I/O devices and special signals IO/$\overline{M}$. In this scheme, 8085 can access 256 I/O ports. In the 8085 microprocessor, this scheme requires 8-bit address lines. It requires less hardware to decode 8-bit address. Arithmetical or logical operations cannot be directly performed with the input data.

Figure 5.15 shows the connection between microprocessor and I/O devices. The I/O devices are identified by port addresses. The I/O read and write operations are performed by using software instructions such as IN and OUT. The I/O read and write operations are controlled by control signals — IOR and IOW respectively. In this scheme, port addresses are varied from 00H to FFH. Therefore, 256 I/O devices may be connected with the microprocessor in I/O mapped I/O devices.

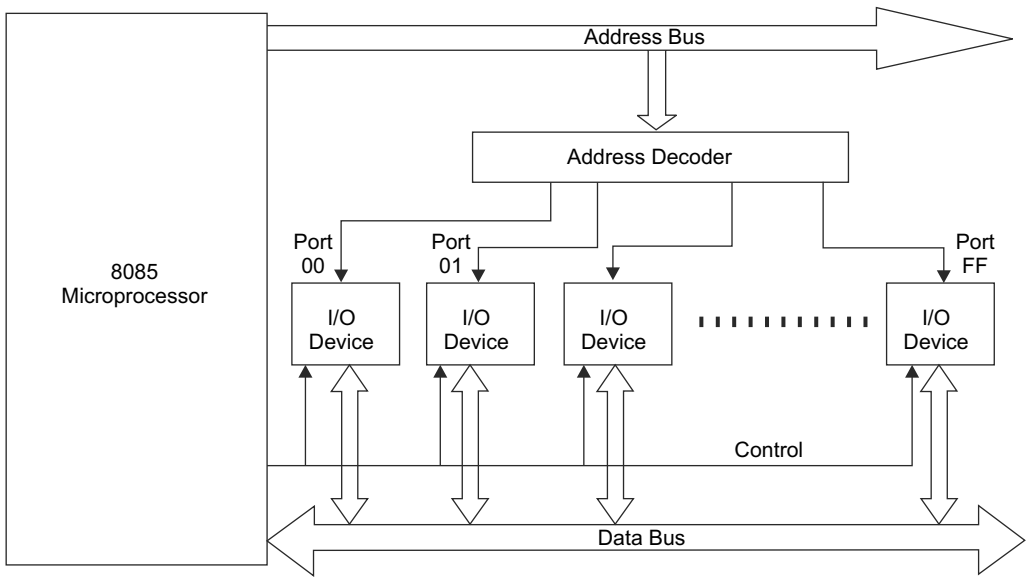


Fig. 5.15 I/O mapped I/O devices

The advantages and disadvantages of I/O mapped I/O are given below.

I/O mapped I/O has the following advantages:

- The total 256-address spaces are available for I/O devices.
- Program writing will be easy as special instructions are used for I/O operations. In 8085 microprocessor, IN and OUT instructions are usually used for data transfer with I/O devices.
- In I/O mapped I/O scheme, the I/O address length is usually one byte and instructions are two byte long. Therefore the program requires less memory and shorter execution time compared to memory mapped I/O.
- The memory reference instructions can be easily distinguished from I/O reference instructions, which make program debugging easier.

I/O mapped I/O has the following disadvantages:

- One microprocessor pin must be used to distinguish between memory and I/O operations. The additional control signals, IOR and IOW must be generated for read and write operations.
- In data transfer with I/O devices and microprocessor, the data has to be transferred to the accumulator only to perform arithmetic or logical operations. Different addressing modes are not used in I/O mapped I/O.

Figure 5.16 shows the connection between microprocessor and memory. The memory location can be identified by I/O devices. The memory read and write operations are also performed by using software instructions such as STA 8000H. In memory read and write operations, $\overline{MEMR}$ and $\overline{MEMW}$ control signals are used.

When memory location address is the same as port address of I/O devices. An I/O device will be selected for read and write operations. The memories read and write instructions employ various addressing modes. As the I/O device is considered as memory location, this type of interfacing is known as memory-mapped I/O. The advantages and disadvantages of memory mapped I/O are given below.

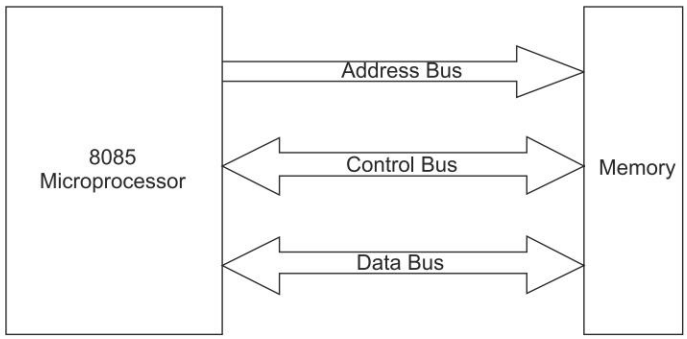


Fig. 5.16 Memory mapped I/O scheme

Memory mapped I/O has the following advantages:

- The Memory mapped I/O scheme can provide more than 256 input-output ports, as the port Addresses are 16-bit.
- All the memory related instructions can be used in read and write operations of memory mapped I/O devices. The arithmetic and logical operations can be performed on available I/O data directly.
- CPU registers can exchange transfer of data with I/O devices directly without accumulator.
- Therefore memory mapped I/O simplifies and increases speed of data transfer.

Memory mapped I/O has the following disadvantages:

- Memory mapped I/O scheme utilizes memory reference instructions, which are three byte instructions and is longer than I/O instructions.
- Due to wider port address, the interface of hardware is also complicated.
- The complexity of the program is large.

The method of memory map is a memory mapped I/O. In a memory mapped I/O scheme, actually a part of the memory space is allocated to the I/O devices. In this scheme, all the memory reference instructions can be used in the case of I/O devices and the arithmetic and logical operations are directly performed on I/O data.

In memory mapped I/O schemes, I/O device is treated as a memory location. This scheme does not require any special instruction. The microprocessor can access I/O device by memory instruction. It does not require special signals. $\overline{MEMR}$, $\overline{MEMW}$ signal can be used to access I/O devices. In this scheme, 8085 can access 64K memory locations or 64K I/O ports. In the 8085 microprocessor, this scheme requires 16 address lines. More hardware is required to decode 16-bit address. Arithmetical or logical operations can be directly performed with the input/output data. The comparison between memory mapped I/O and I/O mapped I/O is given in Table 5.6.

Table 5.6 Comparison between memory mapped I/O and I/O mapped I/O

Memory-Mapped I/O	I/O Mapped I/O
16-bit address	8-bit address
$\overline{MEMR}$ memory read, $\overline{MEMW}$ memory write	$\overline{IOR}$ I/O read, $\overline{IOW}$ I/O write
Memory related instructions are MOV M,R, MOV R,M, ADDM, ANA M, SUB M, STA, LDA, LDAX, STAX	I/O related instruction are IN and OUT

Contd.

Table 5.6 (*Contd.*)

Data transfer between any register and I/O	Data transfer between accumulator and I/O
The memory map 64K is shared between I/Os and system memory	The I/O map is independent of the memory map; 256 input devices and 256 output devices
13 T states (STA, LDA)	The IN and OUT instructions are required 10 T–states for execution
7 T states (MOV M,R)	Less hardware is needed to decode 8-bit address.
More hardware is needed to decode 16 bit address	Not available
Arithmetic or logical operations can be directly performed with I/O data	

In any microprocessor based system, the design of interface is very important. In this section the memory interface of memory and microprocessor is explained below.

The first step is to determine the number of address lines required for memory interface. Find the available memory address. Design the logic circuit for interfacing.

Figure 5.17 shows the 8K × 8 RAM interface to 8085 microprocessor. For 4K-memory interface, 13-address lines are required as 2^{13} = 8K. The 8K memory has starting address in 8000H. Then the end address will be 9FFFH. The following pins are used for interface between microprocessor and memory $\overline{WR}$, $\overline{RD}$, $\overline{CS}$ and A_0–A_{12}.

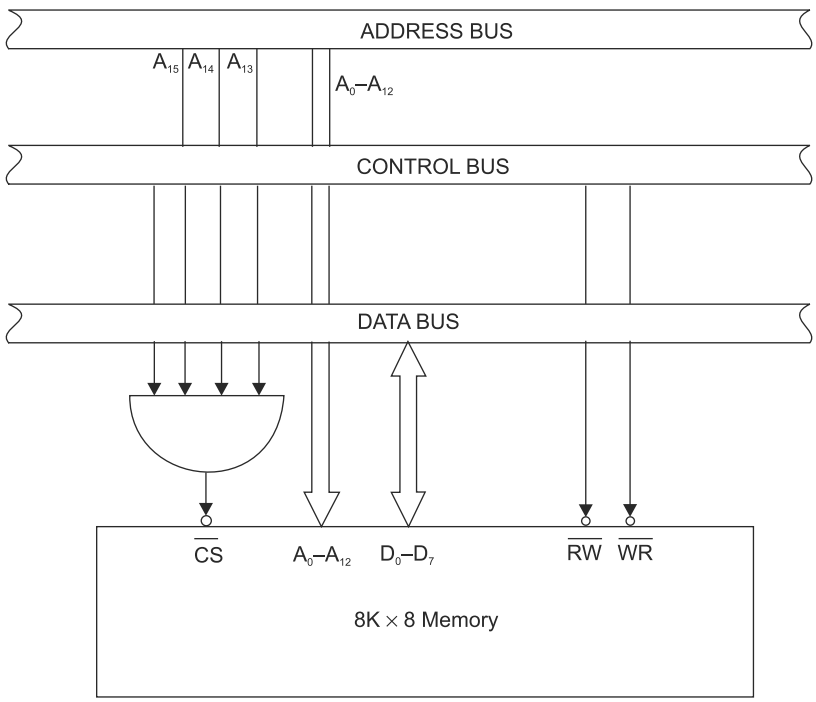


Fig. 5.17 Interfacing 8K byte RAM with microprocessor

But the pins available for memory interface on microprocessor are $\overline{WR}$, $\overline{RD}$, IO/$\overline{M}$ and A_0–A_{15}. A_0–A_{12} lines are directly connected with microprocessor and the chip select signal is generated from remaining address lines A_{13}–A_{15} and IO/$\overline{M}$.

Interfacing 4K byte RAM with microprocessor is illustrated in Fig. 5.18.

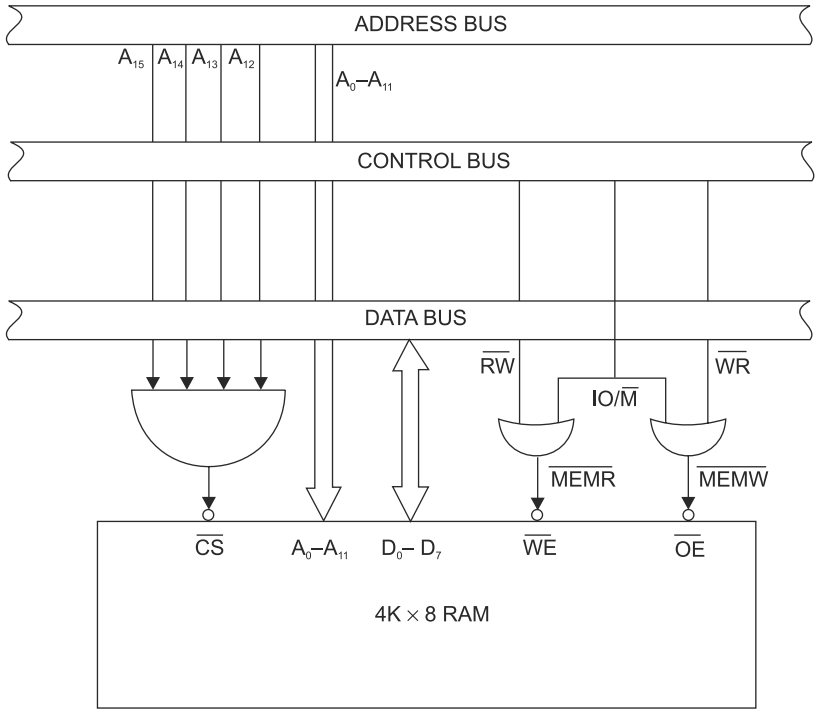


Fig. 5.18 Interfacing 4K byte RAM with microprocessor

5.8.1 Control Signals for Memory and I/O Devices

The 8085 microprocessor has control signals $\overline{RD}$, $\overline{WR}$ for read and write operation of memory and I/O devices. This IC also has a status signal IO/$\overline{M}$ to distinguish the read/write operation of memory or I/O devices. The memory and I/O devices require the following control signals:

$\overline{MEMR}$ (memory read), $\overline{MEMW}$ (memory write), $\overline{IOR}$ (I/O read) and $\overline{IOW}$ (I/O write)

The above control signals are developed from $\overline{RD}$, $\overline{WR}$ and IO/$\overline{M}$ using gates as depicted in Fig. 5.19.

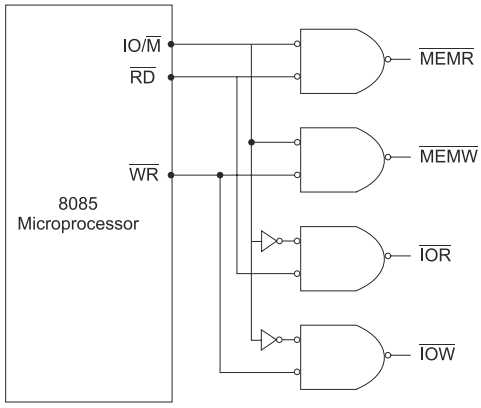


Fig. 5.19 Control Signal of memory and I/O read and write operations

5.8.2 Timing Diagram of Memory Read Cycle and Access Timing

Figure 5.20 shows the typical memory read cycle. The memory must respond with valid data after t_{ACC} seconds after placing the memory address on the address bus. t_{ACC} is known as address access time. It represents the maximum amount of time that the memory requires to decode the address and place the data byte on the data bus. The address access time is in the range of 450 ns to 575 ns. t_{RD} is called the *memory read time* which is the maximum amount of time after $\overline{MEMR}$ becomes low and the valid data will be placed on the data bus. This time is approximately 300 ns. t_{CA} is the minimum amount of time after $\overline{MEMR}$ becomes high before a new address is placed in the address bus. When minimum t_{CA} is not given in the system, a new memory address will not be placed in the address bus and the previous address output data will be in the data bus. This is known as *bus contention*. The t_{CA} is of approximately 20 ns. The detailed operation of memory read is explained below:

First Clock Cycle of Memory Read

- In the first clock cycle (T_1) the microprocessor places the content of program counter, 8001H, which is the address of operand on the 16-bit address bus. The 8 MSBs of the memory address, 80H are placed on the high-order address bus, A_{15}–A_8 and 8 LSBs of the memory address, 00H are placed on the low-order address bus, AD_7–AD_0.

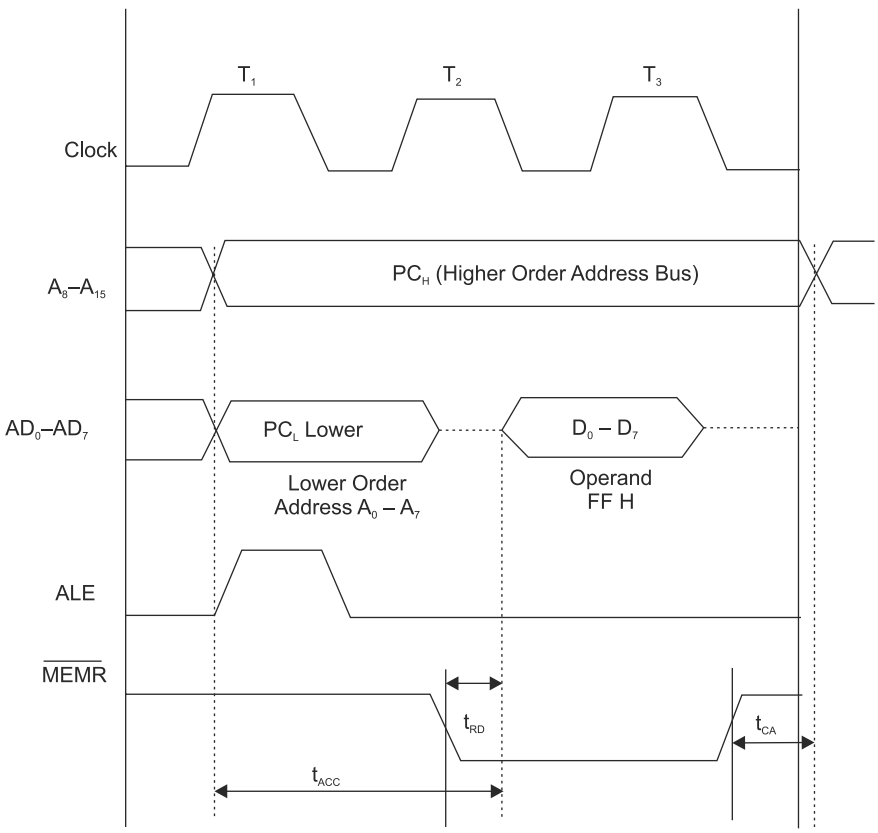


Fig. 5.20 Memory read cycle

- The microprocessor sends an Address Latch Enable (ALE) signal to go high and latch the 8 LSBs of the memory address. Then the low-order address bus is demultiplexed and the complete 16-bit memory address is available in the subsequent clock cycles to get the operand from memory location, 8001H.
- The status signals IO/$\overline{M}$ = 0, S_0 = 0 and S_1 = 1 to identify the memory read operation.

Second Clock Cycle of Memory Read

- The low-order bus AD_7–AD_0 is ready to accept operand from memory. The microprocessor sends the control signal $\overline{MEMR}$ = 0 to enable memory and the program counter is incremented by 1 to 8002H. After that, the operand from the memory location, 8001H is placed on the data bus.

Third Clock Cycle of Memory Read

- During T_3, the microprocessor reads the operand. $\overline{MEMR}$ signal becomes high during T_3 and the memory is disabled.
- The microprocessor also places the operand in any register.

5.8.3 Timing Diagram of Memory Write

In a memory write operation, the microprocessor sends data from the accumulator or any general-purpose register to the memory. The timing diagrams of a memory write cycle are depicted in Fig. 5.21. The memory write cycle is similar to the memory read cycle, but there are differences on status signals. The status signals S_0 = 1 and S_1 = 0 and write $\overline{WR}$ is low during T_2 of machine cycle which indicates that the memory write operation is to be performed.

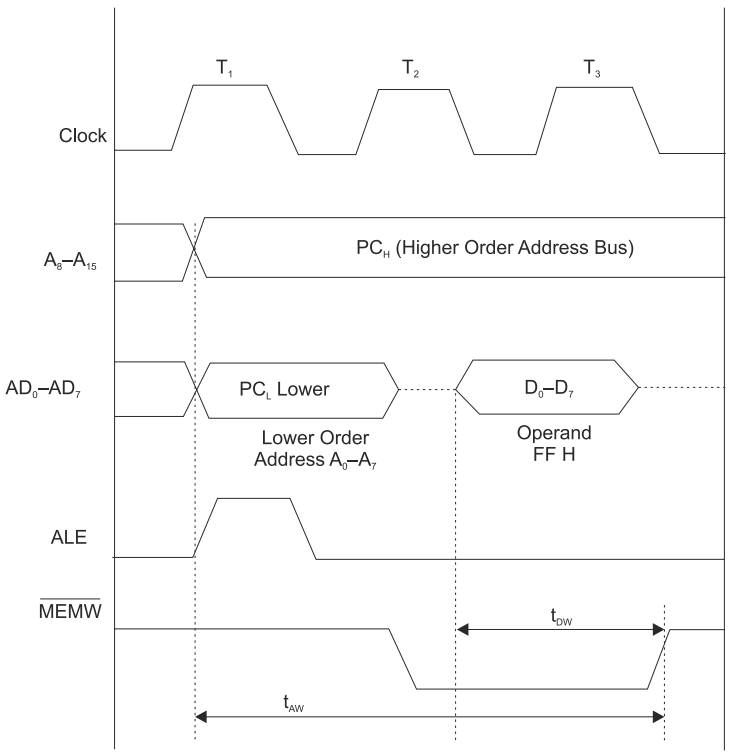


Fig. 5.21 Memory write cycle

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

The memory write cycle begins by placing a valid address on the address bus A_0–A_{15} and valid data is also placed on the data bus D_0–D_7 early in the memory write cycle which is known as data write set-up time. When $\overline{MEMW}$ becomes low, the memory cycle will be started and $\overline{MEMW}$ will be low until writing operation has been completed.

t_{AW} is the minimum amount of time that valid address will be held on the data bus before $\overline{MEMW}$ becomes high. Generally, t_{AW} is 450 ns. t_{DW} is the minimum amount of time that valid data will be held on the bus before $\overline{MEMW}$ becomes high and it is approximately 200 ns.

5.8.4 Timing Diagram of I/O Read Cycle

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

The timing diagram of an I/O read operation is shown in Fig. 5.22. In this case, the address on the A-bus is for an input device. As in an I/O device or I/O port the address is only 8 bits long, the address of I/O device or I/O port is duplicated on both high-order address bus A_8–A_{15} and low-order address bus AD_0–AD_7.

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

This instruction requires three machine cycles for execution. The first machine cycle is *opcode fetch cycle*, and the second machine cycle is a *memory read cycle* to read the address of input device or input port. In the third machine cycle, I/O read operation is performed means the data to be read from the input device or input port. After execution of this instruction, the data is placed in the accumulator. The opcode fetch cycle and memory read cycle are exactly similar to MVI C, FF H instruction. Figure 5.22 shows the machine cycle M_3 of I/O read operation and it is explained below:

T_1 State of M_3

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

T_2 State of M_3

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

T_3 State of M_3

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

5.8.5 Timing Diagram of I/O Write Cycle

The microprocessor sends the content of the accumulator to an I/O port or I/O device in an I/O write cycle. The operations of an I/O write cycle is similar to a memory write cycle. But the difference between memory write and I/O write cycle is that IO/$\overline{M}$ becomes high in case of I/O write cycle. When IO/$\overline{M}$ is high, the

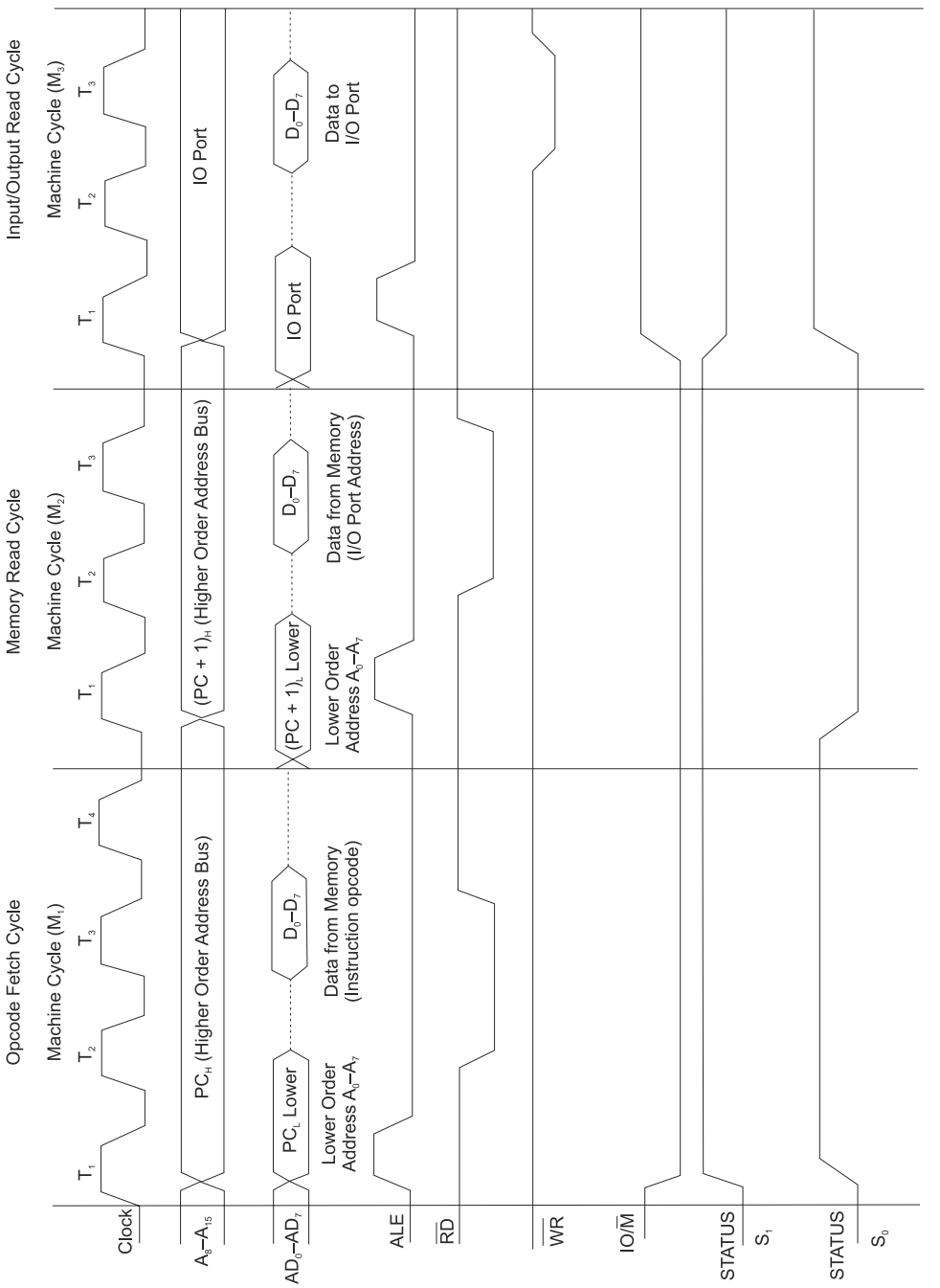


Fig. 5.22 Timing diagram of IN Port Address

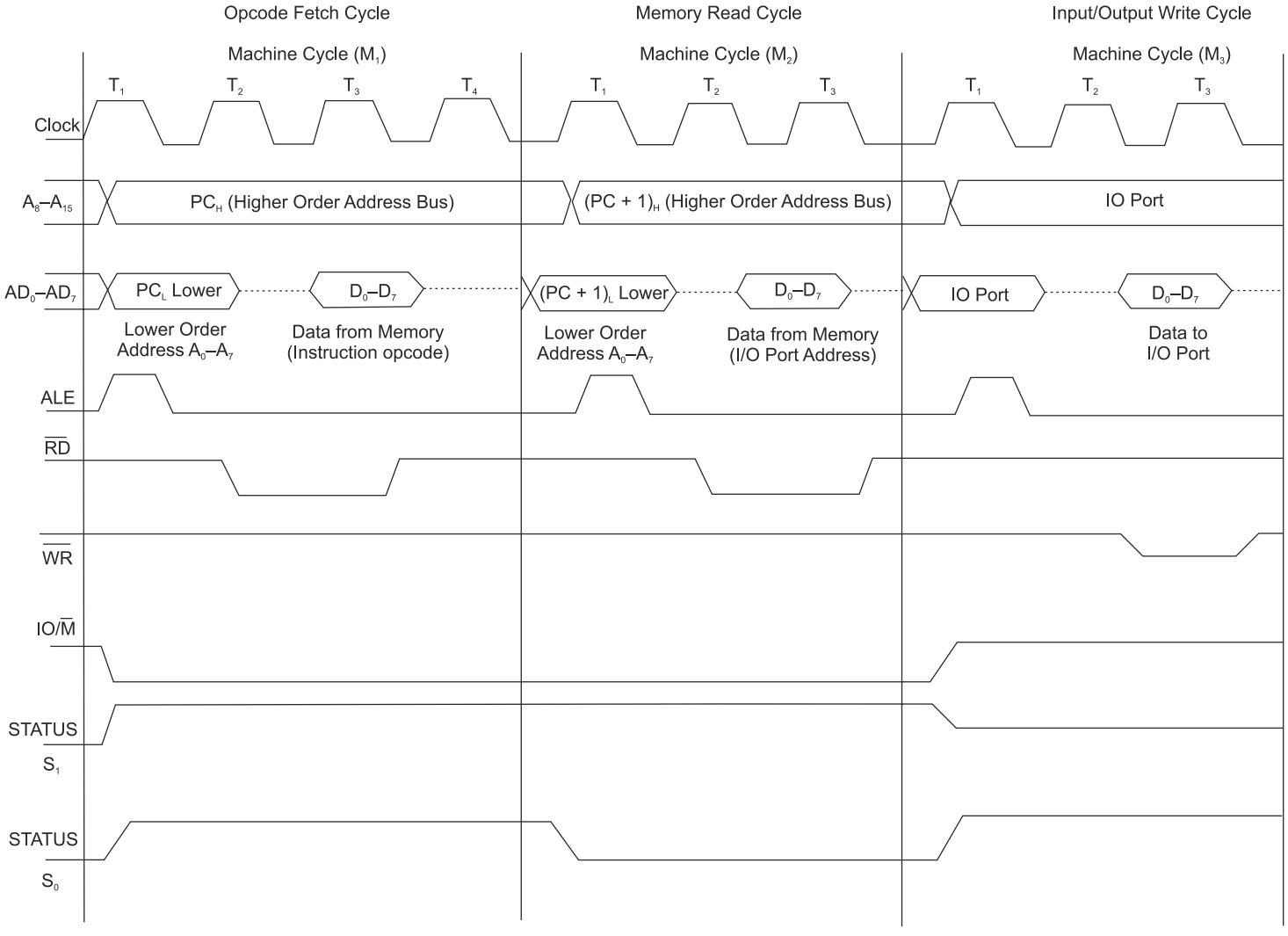


Fig. 5.23 Timing diagram of OUT Port Address

microprocessor locates the address of any output device or an output port. The address of an output device or an output port is duplicated on both higher-order address bus A_8–A_{15} and lower-order address bus AD_0–AD_7.

The OUT instruction is used for I/O write operation. This is a two-byte instruction and it requires three machine cycles as depicted in Fig. 5.23. The first machine cycle is for opcode fetch operation, and the second machine cycle is a memory read cycle for reading the address output device or output port from the memory. In the next third machine cycle data will be written in output device or output port. In other words, data is to be sent to the I/O device. The third machine cycle is explained below:

T_1 State of M_3

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

T_2 State of M_3

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

T_3 State of M_3

- CPU places the content of register A in data bus.
- Then write data to I/O port.
- $\overline{WR}$ Signal becomes high as I/O read operation has been completed.

Review Questions

5.1 What are the types of memory? Write the comparison between different types of memory.

5.2 Explain memory mapped I/O and I/O mapped I/O. Write the comparison between memory mapped I/O and I/O mapped I/O. What are the instructions available in memory mapped I/O and I/O mapped I/O scheme?

5.3 What are the advantages and disadvantages of memory mapped I/O over I/O mapped I/O?

5.4 What are the advantages and disadvantages of I/O mapped I/O over CPU initiated data transfer? Explain why I/O mapped I/O data transfer technique is limited to 256 input and 256 out peripherals.

5.5 A semiconductor memory is specified as 4K × 8. Mention the number of words, word size and total capacity of this memory.

5.6 Design a memory 16 × 8 from 16 × 4 memory IC.

5.7 Develop a 32 × 4 memory using 16 × 4 memory ICs.

5.8 Draw a memory read cycle and explain briefly.

5.9 Draw a memory write cycle and explain briefly.

5.10 Draw the timing diagram of I/O read cycle and explain briefly.

5.11 Draw the timing diagram of I/O write cycle and explain briefly.

5.12 What are the control signals used for memory and I/O read and write operations?

5.13 Explain the generation of $\overline{MEMR}$, $\overline{MEMW}$, $\overline{IOR}$ and $\overline{IOW}$ control signals from $IO/\overline{M}$, $\overline{RD}$, $\overline{WR}$ signals.

5.14 Compare the memory mapped I/O with peripheral mapped I/O.

5.15 Design to interface 2K × 8 RAM and 4K × 8 RAM to an 8085 microprocessor.

5.16 Discuss address decoding with a suitable example.

5.17 Explain memory interfacing with 8085 microprocessor. Design a memory interfacing circuit to interface the following memory ICs

(i) 2K × 8-bit EPROM 2716. Assume starting address is 8000H.

(ii) 2K × 4-bit RAM 6116. Consider starting address is 9000H.

Write the memory map of the above ICs.

5.18 Compare static RAM and dynamic RAM.

5.19 Design the interfacing of 8K × 8 bit RAM with the 8085 microprocessor. Assume the starting address is 7000H. Show the memory map.

5.20 Design the interfacing circuit to interface two 8K-byte RAM and two 4K-byte EPROM with 8085 microprocessor. Assume the starting address is 8000H. Show the memory map.

5.21 Draw the memory organization of 2K × 8-bit memory using two 1K × 8 bit memory IC and explain briefly.

5.22 Design a memory organization of 1K × 8 bit memory using two 1K × 4 bit memory IC.

5.23 Draw the memory map of the following

(a) 2K EPROM with starting address 4000H

(b) 4K RAM with starting address 8F00H

5.24 Draw the interfacing circuit to interface 4K byte RAM with 8085 microprocessor and explain briefly.

Multiple-Choice Questions

5.1 How many address lines are required to access 1MB RAM using microprocessor?
(a) 16 (b) 8 (c) 20 (d) 12

5.2 What are the control signals of 8085 microprocessor used to interface I/O devices.
(a) $IO/\overline{M}$, $\overline{RD}$, $\overline{WR}$ (b) $IO/\overline{M}$ (c) $\overline{RD}$ (d) $\overline{WR}$

5.3 To design a 4KB RAM with 1024 byte RAM ICs, how many ICs are required?
(a) 4 (b) 8 (c) 2 (d) None of these

5.4 In I/O mapped I/O device interfacing, the device has
(a)16-bit address lines (b) 8-bit address lines
(c) 20-bit address lines (d) 12-bit address lines

5.5 In memory mapped I/O device interfacing, the device has
(a) 16-bit address lines (b) 8-bit address lines
(c) 20-bit address lines (d) 12-bit address lines

5.6 When the starting address of 4K RAM is 8000H, the memory map will be
(a) 8000H–9000H (b) 8000H–8500H
(c) 8000H–9500H (d) 8000H–A000H

5.7 If the starting address of 2K RAM is 4000H, the end address of memory map will be
(a) 4800H (b) 8400H (c) 8000H (d) 4000H

5.8 A 64K bit memory device can be organized as
(a) 64K × 1 (b) 16K × 4 (c) 8K × 8 (d) all of these

5.9 How many address lines are used to identify an I/O port in I/O mapped I/O and memory mapped I/O?
(a) 16-bit and 8-bit address lines (b) 8-bit and 16-bit address lines
(c) 8-bit and 20-bit address lines (d) 16-bit and 12-bit address lines

5.10 The number of 4K × 4 memory devices are required for 16K × 8 memory
(a) 2 (b) 3 (c) 4 (d) 8

5.11 The memory map of a 4K-byte memory chip begins at the location 3000H. The last location of memory address and number of pages in the chip are
(a) 2000, 2 (b) 3000, 4 (c) 4000, 16 (d) 5000, 9

5.12 The number of address lines are required to access 2M byte of data from microprocessor
(a) 16-bit address lines (b) 8-bit address lines
(c) 20-bit address lines (d) None of these

5.13 To design a 2KB RAM with 1024 byte RAM ICs, how many ICs are required
(a) 4 (b) 8 (c) 2 (d) None of these

5.14 The address lines required for 16 K byte memory chip are
(a) 13 (b) 14 (c) 15 (d) 16

5.15 The number of address lines are required to access 1M byte of data from microprocessor
(a) 16-bit address lines (b) 20-bit address lines
(c) 24-bit address lines (d) 32-bit address lines

5.16 The timing diagram of I/O write Cycle has
(a) One machine cycle (M1) (b) Two machine cycles (M1 & M2)
(c) Three machine cycles (M1, M2 & M3) (d) None of these

Answers to Multiple-Choice Questions

5.1 (c)	5.2 (a)	5.3 (a)	5.4 (b)
5.5 (a)	5.6 (a)	5.7 (a)	5.8 (d)
5.9 (b)	5.10 (d)	5.11 (c)	5.12 (d)
5.13 (c)	5.14 (b)	5.15 (b)	5.16 (c)

CHAPTER

6

Interrupts of 8085 Microprocessor

6.1 INTRODUCTION

An interrupt is the facility provided by the microprocessor to communicate with the outside environment to let the microprocessor divert its operation based on priority. The interrupts can be used for various applications in different environments.

An interrupt is a process where an external device can get the attention of the microprocessor. The process starts from the I/O device and it is asynchronous type data transfer. Figure 6.1 shows the interrupt-driven data transfer. The microprocessor can initiate the data transfer after getting an interrupts signal from the I/O device. The microprocessor can scan the interrupt pin on every machine cycle. When the interrupt signal is present, microprocessor suspended its present operation after storing the current status in the microprocessor so that microprocessor can restart the suspended work again from where it left. Therefore, stack is used to store the current status. Then the microprocessor provides services the interrupt request by executing an *interrupt service routine*.

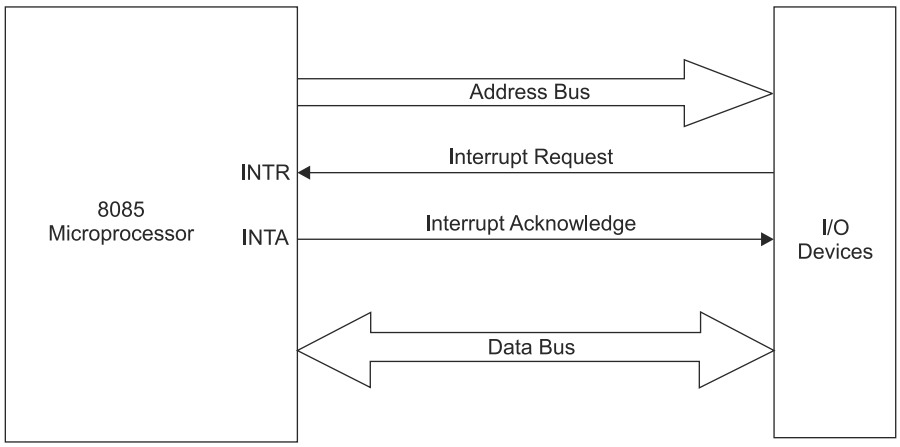


Fig. 6.1 Interrupt-driven data transfer for an I/O device

An interrupt is considered to be an emergency signal, which may be serviced. The microprocessor may respond to it as soon as possible. When the microprocessor receives an interrupt signal, it suspends the currently executing program and jumps to an Interrupt Service Routine (ISR) to respond to the incoming interrupt. Each interrupt will most probably have its own ISR. Figure 6.2 shows the interrupt execution. Responding to an interrupt may be immediate or delayed depending on whether the interrupt is maskable or non-maskable and whether interrupts are being masked or not. There are two different ways of redirecting the execution to the ISR depending on whether the interrupt is vectored or non-vectored. In a vectored interrupt, the address of the subroutine is already known to the microprocessor. In case of a non-vectored interrupt, the I/O devices will have to supply the address of the subroutine to the microprocessor.

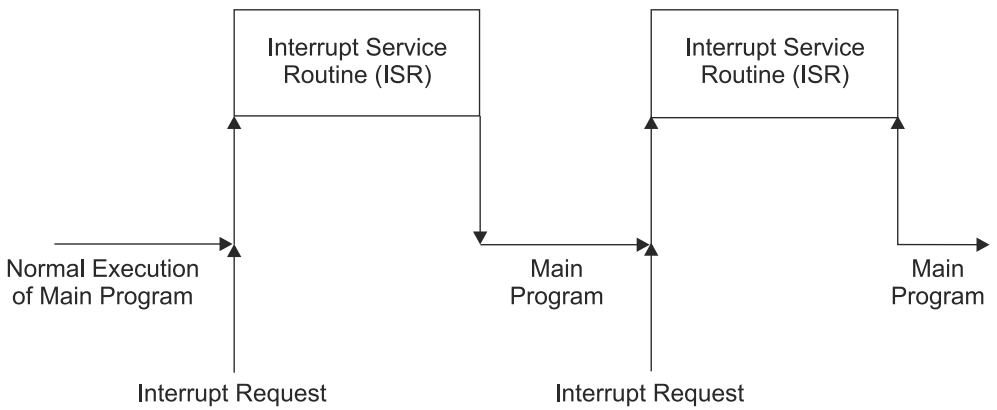


Fig. 6.2 Interrupt execution

In any microprocessor-based system, I/O devices can use interrupt-driven data transfer. A microprocessor may have one interrupt level and more than one I/O devices share the interrupt level. The microprocessor may have many interrupt levels and several I/O devices. Each device can be connected to a interrupt level. When several I/O devices are connected with a single interrupt level, these devices are reconnected with an 8259 interrupt controller. Using an 8259 interrupt controller, only eight I/O devices can be connected. When more than eight I/O devices are connected to the 8085 microprocessor, more than 8259 interrupt controllers are connected in cascade. If each I/O device is connected to an independent interrupt level, the microprocessor should have several interrupt levels. In this case, the number of I/O devices must be less than the number of interrupt levels.

6.2 CLASSIFICATION OF INTERRUPTS

Interrupts can be classified into two types: maskable interrupts and non-maskable interrupts. The *maskable interrupts* can be delayed or rejected but the *non-maskable* interrupts cannot be delayed or rejected. Interrupts can also be classified into vectored and non-vectored interrupts. In *vectored interrupts*, the address of the service routine is hard wired; but in *non-vectored interrupts*, the address of the service routine needs to be supplied externally by the device. All types of interrupts are explained in the 8085 interrupts section.

6.3 THE 8085 INTERRUPTS

When a device interrupts, it actually wants the microprocessor to give a service, which is equivalent to asking

the microprocessor to call a subroutine. This subroutine is known as Interrupt Service Routine (ISR). Figure 6.3 shows the 8085 microprocessor interrupts and their detailed operations are depicted in Fig. 6.4. The 'EI' instruction is a one-byte instruction and is used to enable the non-maskable interrupts. The 'DI' instruction is a one-byte instruction and is used to disable the non-maskable interrupts.

The 8085 microprocessor has a single non-maskable interrupt and the non-maskable interrupt is not affected by the value of the interrupt enable flip-flop. The processor has five hardware interrupts such as INTR, RST 5.5, RST 6.5, RST 7.5, and TRAP. They are presented below in the order of their priority from lowest priority to highest priority:

INTR is a maskable interrupt. When the interrupt occurs, the processor fetches from the bus one instruction, usually one of EI and DI instructions.

The syntax for the interrupt instruction is RST *n*, where *n* is equal to 0 to 7. Any one of the 8 RST instructions (RST0 – RST7) can be executed at a time. During execution, the processor saves the current program counter into the stack and branches to the memory location. The vector address of this software interrupt is calculated from $N \times 8$, where N is a 3-bit number from 0 to 7 supplied with the RST instruction. For example, the vector address of RST 3 is $3 \times 8 = 24_{10} = 0018$H. Table 6.1 shows the vector address of RST instructions. The 8085 recognize 8 RESTART instructions: RST0 – RST7. Each of these would send the execution to a predetermined hard-wired memory location.

Table 6.1 RST *n* interrupts vector address

Restart instruction	Hex code	Equivalent Vector Address
RST0	C7	CALL 0000H
RST1	CF	CALL 0008H
RST2	D7	CALL 0010H
RST3	DF	CALL 0018H
RST4	E7	CALL 0020H
RST5	EF	CALL 0028H
RST6	F7	CALL 0030H
RST7	FF	CALL 0038H

When any of the above instructions are executed, a CALL instruction to the specified address is executed. The content of the program counter is saved in the stack and the control moves to the specified address. The vector addresses of the instructions are 8 bytes apart. Therefore, 8 bytes of instructions can be stored at any vector address. Generally, a 3-byte JMP instruction is stored for the corresponding ISR and program control is transferred to the desired ISR. The CALL instruction is a 3-byte instruction. The processor calls the subroutine, the address of which is specified in the second and third bytes of the instruction. The INTR input is the only non-vectored interrupt. INTR is maskable using the EI/DI instruction pair.

RST 5.5 is a maskable interrupt. When this interrupt is received, the processor saves the contents of the program counter register into the stack and the branches to 002CH address.

RST 6.5 is a maskable interrupt. When this interrupt is received, the processor saves the contents of the program counter register into the stack and branches to 0034H address.

RST 7.5 is a maskable interrupt. When this interrupt is received the processor saves the contents of the program counter register into the stack and branches to 003CH address.

TRAP is a non-maskable interrupt. It does not need to be enabled, as it cannot be disabled. It has the highest priority amongst all interrupts. This is edge and level sensitive. This TRAP signal needs to be high and stay

high for recognisation. Once it is recognised, it does not need to be recognised again until it becomes low and then high again. Generally, TRAP is used for power failure and emergency shutdown. When this interrupt is received, the processor saves the contents of the PC register into the stack and branches to 0024H address. Figure 6.4 shows the TRAP interrupts circuit with other interrupts. The positive edge of TRAP input signal sets the D flip-flop and Q becomes '1'. Then AND gate output will be '1' for the duration of high level of TRAP input. Jump to the vector memory location 0024H as the starting address of an interrupt service routine for TRAP interrupt is 0024H.

RST 5.5, RST 6.5, RST 7.5 are all automatically vectored. RST 5.5, RST 6.5, and RST 7.5 are all maskable and TRAP is the only non-maskable interrupt in the 8085. TRAP is also automatically vectored. All maskable interrupts can be enabled or disabled using EI and DI instructions. RST 5.5, RST6.5 and RST7.5 interrupts can be enabled or disabled individually using SIM instruction RST5.5, a maskable interrupt. When this interrupt is received, the processor saves the contents of the PC register into the stack and branches to 002CH (hexadecimal) address.

Table 6.2 8085 interrupts

Interrupt	Maskable	Vectored
INTR	Yes	No
RST 5.5	Yes	Yes
RST 6.5	Yes	Yes
RST 7.5	Yes	Yes
TRAP	No	Yes

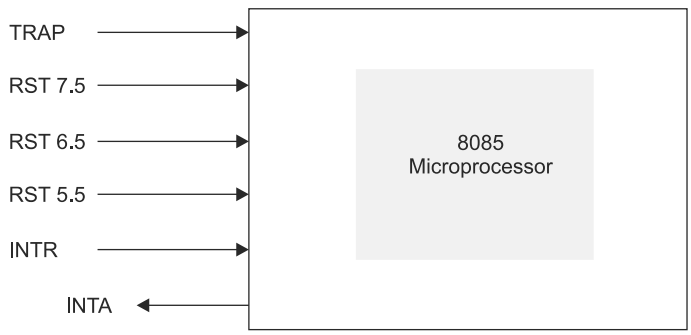


Fig. 6.3 8085 microprocessor interrupts

6.4 INTERRUPT VECTORS AND VECTOR TABLE

An interrupt vector is a pointer in which the Interrupt Service Routine (ISR) is stored in memory. All vectored interrupts are mapped onto a memory area called the Interrupt Vector Table as given in Table 6.1 and Table 6.3. The Interrupt Vector Table is generally located in memory page 00 (0000H – 00FFH). The purpose of the Interrupt Vector Table is to hold the vectors that redirect the microprocessor to the right place when an interrupt appears.

For example, assume a device interrupts the microprocessor using the RST 7.5 interrupt line. As the RST 7.5 interrupt is a vectored-type interrupt, microprocessor should know, in which memory location it jumps

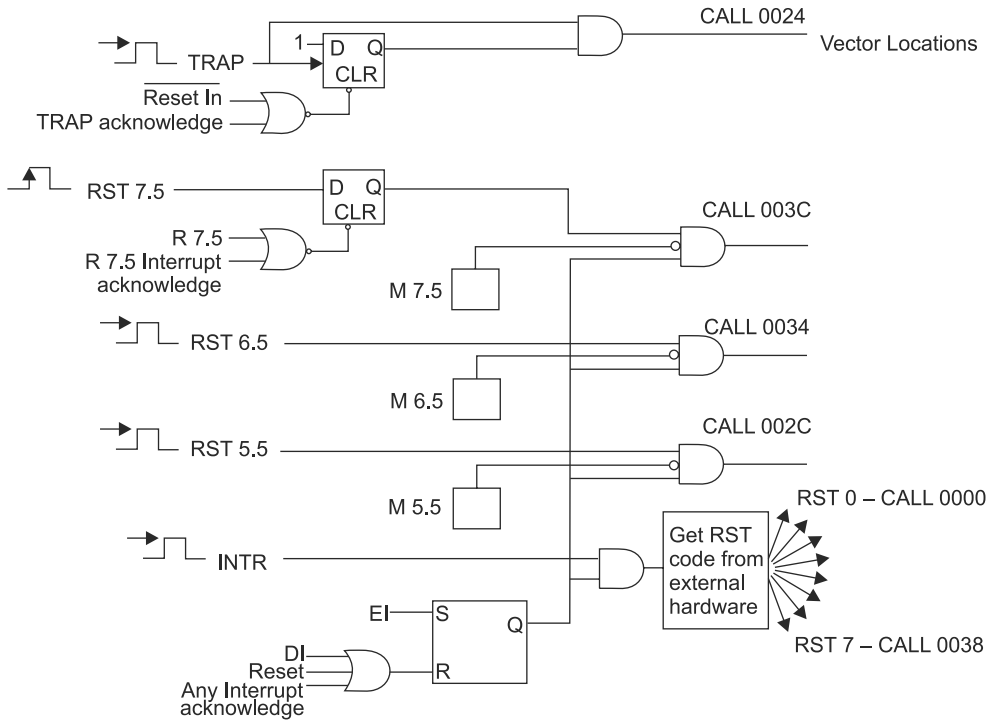


Fig. 6.4 Interrupts of 8085 microprocessor

using a call instruction to get the ISR address. RST7.5 is known as call 003CH to microprocessor. The microprocessor jumps to 003CH memory location and it also gets a JMP instruction to the actual ISR address. After that the microprocessor jumps to the ISR location.

6.4.1 Maskable/Vectored Interrupts of 8085

The 8085 has 4 masked/vectored interrupt inputs. RST 5.5, and RST 6.5 and RST 7.5 are all maskable. They are automatically vectored according to Table 6.3. The vectors for these interrupts fall in between the vectors for the RST instructions. For this, they have names like RST 5.5 (RST 5 and a half).

Table 6.3 Maskable interrupts and vector locations

Interrupt	Vector Address
RST 5.5	002CH
RST 6.5	0034H
RST 7.5	003CH

The spaces between software interrupt RST 5 and hardware interrupt RST 5.5 is 4 bytes. In this space, a 3-byte JMP instruction is written. The vector address of the hardware interrupts is spaced 8 bytes. Usually, a three-byte jump instruction is written in this space. The hardware interrupts signal are directly vectored to the address specified in the interrupt vector table. These interrupts are called *vector interrupts*.

6.4.2 Masking RST 5.5, RST 6.5 and RST 7.5

RST 5.5, RST 6.5 and RST 7.5 interrupts are masked at two levels through the interrupt enable flip-flop and the EI/DI instructions. The interrupt enable flip-flop controls the whole maskable interrupt process through individual mask flip-flops that control the availability of the individual interrupts. These flip-flops control the interrupts individually. The 8085 maskable/vectored interrupt should process the following steps.

Step-1 The interrupt process must be enabled using the EI instruction.

Step-2 The 8085 should check for an interrupt during the execution of every instruction.

Step-3 When there is an interrupt and the interrupt is enabled using the interrupt mask, the microprocessor will complete the executing instruction, and then reset the interrupt flip flop.

Step-4 Thereafter, the microprocessor executes the CALL instruction which sends the execution to the appropriate memory location according to the interrupt vector table.

Step-5 When the microprocessor executes the call instruction, it saves the address of the next instruction on the stack.

Step-6 The microprocessor jumps to the specific service routine. The Interrupt Service Routine (ISR) must incorporate the instruction EI to re-enable the interrupt process.

Step-7 At the end of the service routine, the RET instruction returns the execution to where the program was interrupted.

6.4.3 Non-Vectored Interrupt

The 8085 non-vectored interrupt processes are completed by the following steps:

Step-1 The interrupt process should be enabled using the EI instruction.

Step-2 The 8085 checks for an interrupt during the execution of every instruction.

Step-3 If INTR is high, MP completes current instruction, disables the interrupt and sends INTA (Interrupt acknowledge) signal to the device that interrupted.

Step-4 INTA allows the I/O device to send an RST instruction through data bus.

Step-5 After receiving the INTA signal, the microprocessor saves the memory location of the next instruction on the stack and the program is transferred to 'call' location (ISR Call) specified by the RST instruction.

Step-6 The microprocessor performs the ISR. ISR must include the 'EI' instruction to enable further interrupt within the program.

Step-7 RET instruction at the end of the ISR allows the microprocessor to retrieve the return address from the stack and the program is transferred back to where the program was interrupted.

The 8085 recognises 8 RESTART instructions: RST0 – RST7. Each of these would send the execution to a predetermined hard-wired memory location given in Table 6.1. The syntax for the interrupt instruction is RST *n*, where *n* is equal to 0 to 7. The restart sequence is made up of three machine cycles.

In the 1st Machine Cycle

- The microprocessor sends the INTA signal
- When INTA is active low, the microprocessor reads the data lines expecting to receive, from the interrupting device, the opcode for the specific RST instruction

In the 2nd and 3rd Machine Cycles

- The 16-bit address of the next instruction is saved on the stack
- Then the microprocessor jumps to the address associated with the specified RST instruction

The opcode is simply a collection of bits. The external device produces the opcode for the appropriate RST instruction. So, the device needs to set the bits of the data bus to the appropriate value in response to an INTA signal.

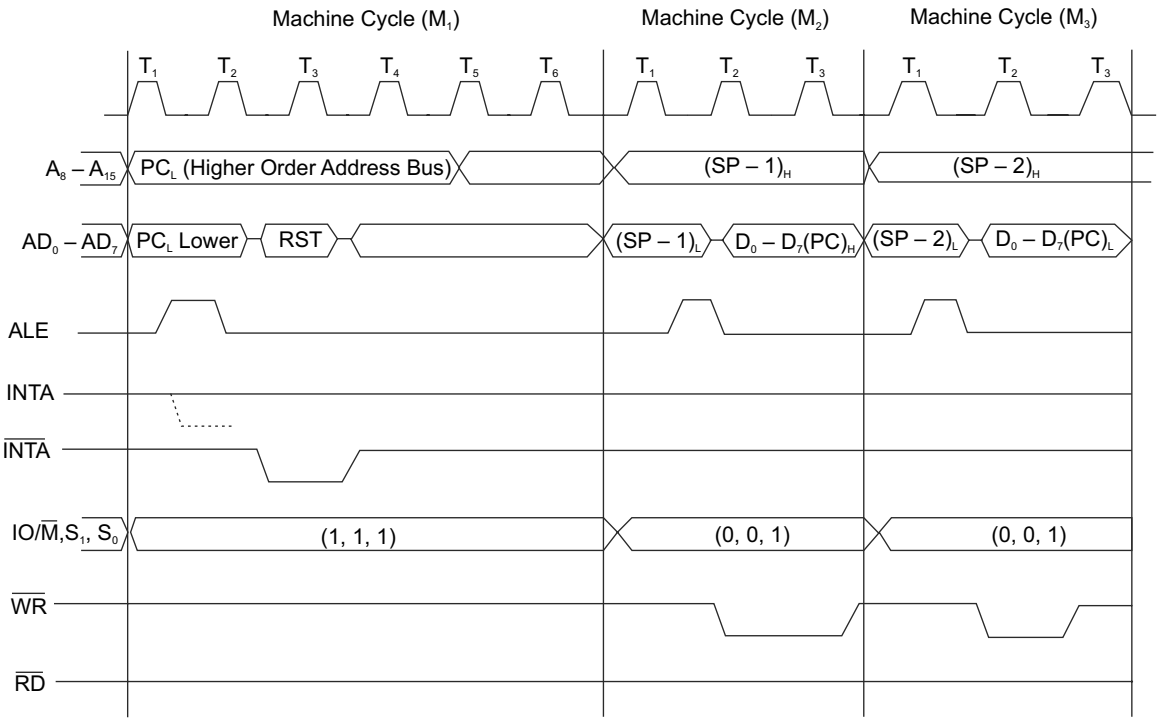


Fig. 6.5 Timing diagram of RST 5 instruction

The timing diagram of RST 5 is shown in Fig. 6.5. Consider that the RST 5 is stored at 8000H memory location. This instruction has three machine cycles. In the machine cycle M_1, the opcode of RST 5 instruction is fetched. In the opcode fetch cycle, opcode will be read and decoded. As RST is an interrupt instruction, it needs to execute its service routine and after execution it must return back to the next memory location of RST5 instruction. This return address must be stored in the stack.

The second machine cycle is M_2, which is called a *memory write cycle*. In this machine cycle, the higher order byte of the the program counter will be stored in the stack. For this, the content of the stack pointer is decremented by one and a 16-bit content is placed on the address bus. Then the higher order byte of the program counter is stored in that memory location.

In the third machine cycle, the content of the stack pointer is also decremented by one and again placed on the address bus. Thereafter, the lower order byte of the program counter can be stored in that memory location.

Figure 6.6 shows an interrupt acknowledge cycle for CALL instruction. M_2 and M_3 machine cycles are needed to call the 2-byte address of CALL instruction. Then the content of the program counter is stored in memory write cycles M_4 and M_5. After that a new instruction cycle starts.

Figure 6.7 shows the generation of RST 5 opcode. RST 5 opcode is 11101111 (EFH). If INTR is acknowledged by the microprocessor, $\overline{INTA}$ signal becomes low. The RST 5 is gated into the system bus. Then the

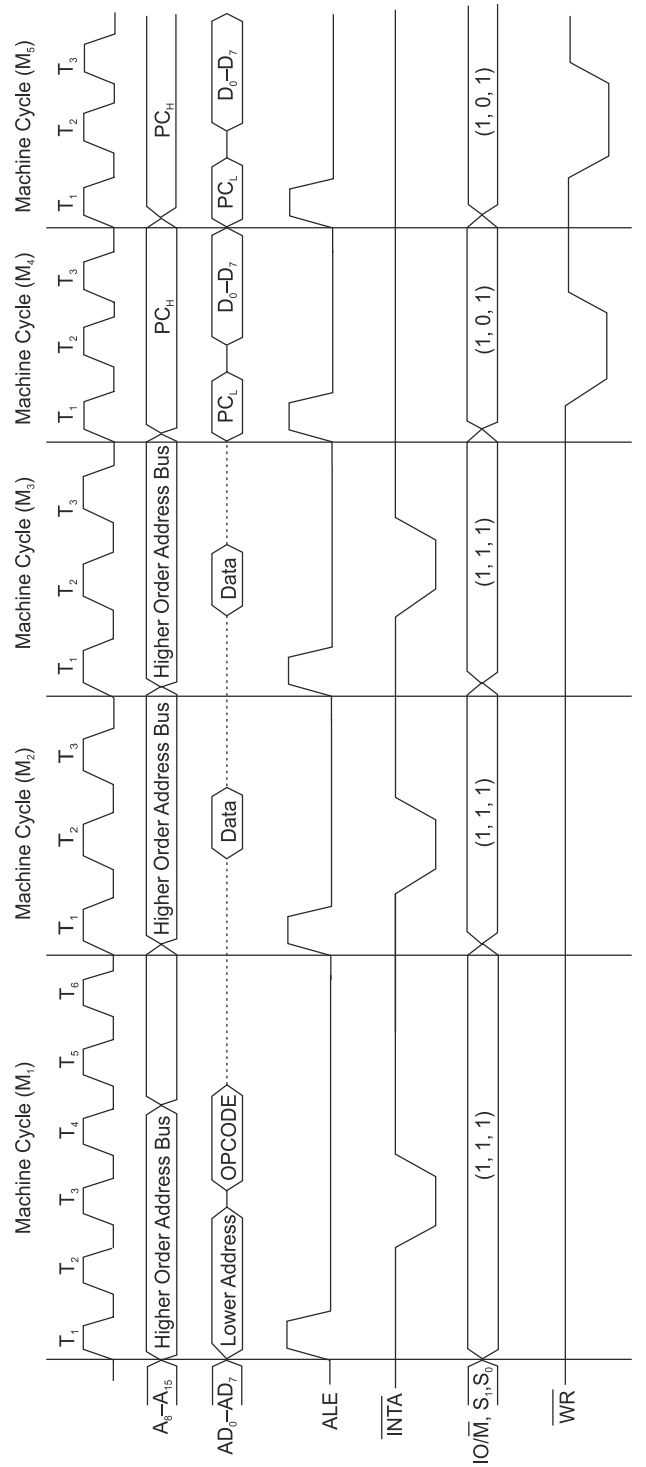


Fig. 6.6 Timing Diagram of INTA machine cycle and execution of CALL instruction

microprocessor saves the content of the program counter in the STACK and jumps to the memory location 0028H. In this address, the interrupt service starts and ends with RET instruction. After execution of RET instruction, the processor restores the saved address in the stack to the program counter so that normal execution of the main program can be started.

In the 1st machine cycle of the RST operation, the microprocessor activates the INTA signal. This signal enables the tri-state buffers, which place the value EFH on the data bus. Therefore, it sends the microprocessor the RST 5 instruction. The RST 5 instruction is exactly equivalent to CALL 0028H.

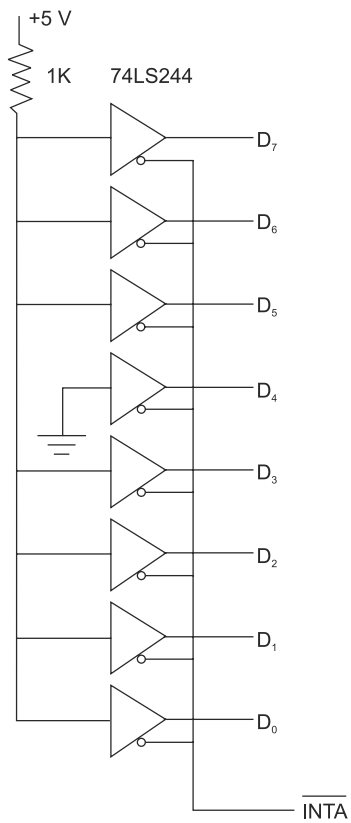


Fig. 6.7 Generation of RST 5 opcode

6.4.4 Triggering Levels

RST 7.5 is positive-edge sensitive. When a positive edge appears on the RST 7.5 line, logic '1' is stored in the flip-flop as a 'pending' interrupt. Since the value has been stored in the flip-flop, the line does not have to be high when the microprocessor checks for the interrupt to be recognised. The line must go to zero and back to one before a new interrupt is recognised.

RST 6.5 and RST 5.5 are level-sensitive. The interrupting signal must remain present until the microprocessor checks for interrupts.

TRAP is edge triggered as well as level triggered. Therefore, TRAP must be high until this is acknowledged. Figure 6.4 shows the TRAP interrupt. When the interrupt is acknowledged, the flip-flop of TRAP interrupt will be cleared so that the next new TRAP interrupt can be entertained. The summary of all 8085 interrupts is given in Table 6.4.

Table 6.4 8085 interrupts

Interrupt	Maskable	Masking method	Vectored	Memory	Triggering method
INTR	Yes	DI/EI	No	No	Level sensitive
RST 5.5	Yes	DI/EI SIM	Yes	No	Level sensitive
RST 6.5	Yes	DI/EI SIM	Yes	No	Level sensitive
RST 7.5	Yes	DI/EI SIM	Yes	Yes	Edge sensitive
TRAP	No	None	Yes	No	Level and edge sensitive

6.5 INTERRUPT INSTRUCTIONS

The Enable Interrupts (EI) and Disable Interrupts (DI) instructions authorise the microprocessor to allow or reject interrupts. In case of EI, the interrupts will be enabled following the completion of the next instruction following the EI. This allows at least one more instruction like JMP or RET, to be executed before the microprocessor allows itself to be again interrupted. In the DI, the interrupts are disabled immediately and no flags are affected.

The Read Interrupt Mask (RIM) and Set Interrupt Mask (SIM) instructions are used to provide interrupt

services of the 8085 microprocessor with the help of the Serial Input Data (SID) and Serial Output Data (SOD) pins on the device. The discussion of the above two instructions are as follows.

6.5.1 SIM Instruction

Sometimes it is required to enable some selected interrupts and disable some other interrupts. The selected interrupts are enabling through the Set Interrupt Mask. The Accumulator (A) is loaded with the specified mask bits. The SIM instruction reads the accumulator content and enables and disables the interrupts.

The individual masks for RST 5.5, RST 6.5 and RST 7.5 are manipulated using the SIM instruction. This instruction takes the bit pattern in the accumulator. The SIM instruction reads the accumulator content and enables or disables the specific interrupts. Figure 6.8 shows the accumulator content for SIM instruction.

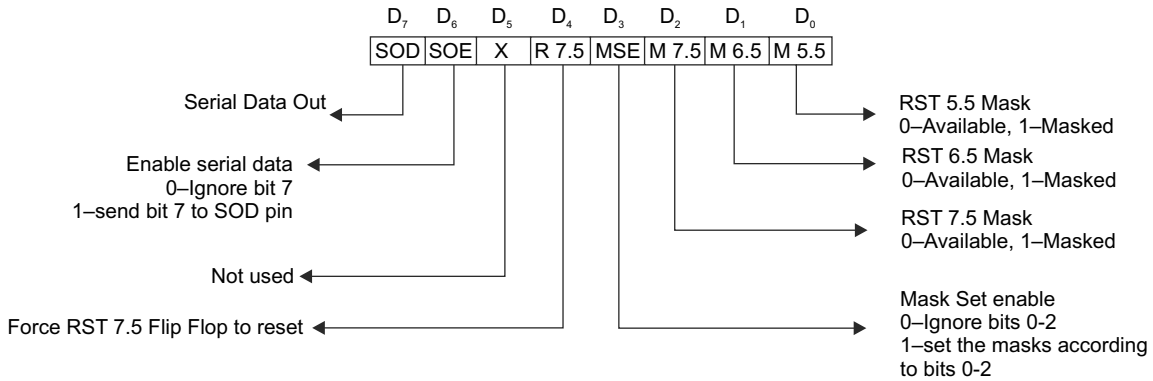


Fig. 6.8 Accumulator content for SIM

RST Masks Bits D_0, D_1, and D_2 Bit D_0 is the mask for RST 5.5, bit D_1 is the mask for RST 6.5 and bit D_2 is the mask for RST 7.5. If the mask bit is 0, the interrupt is available. If the mask bit is 1, the interrupt is masked. If bits D_0 or D_1 are set to 1, a signal applied to their respective pins causes no action. When D_0 or D_1 are set to 0, their respective bits will be visible through the RIM instruction, and the call to the interrupt vector will occur. In the case of bit D_2, the RIM instruction can indicate that RST 7.5 interrupt is pending, and an automatic call will not occur.

Mask Set Enable Bit D_3 Bit D_3 is Mask Set Enable (MSE) and this is an enable for setting the mask. If it is set to 0, the mask is ignored and the old settings remain. If it is set to 1, the new settings are applied. The SIM instruction is used for multiple purposes and not only for setting interrupt masks. It is also used to control functionality such as serial data transmission. Therefore, bit D_3 is necessary to tell the microprocessor whether or not the interrupt masks should be modified.

RST 7.5 Reset Bit D_4 Bit D_4 is RST 7.5. The RST 7.5 interrupt is the only 8085 interrupt that has memory. If a signal on RST 7.5 arrives while it is masked, a flip-flop will remember the signal. When RST 7.5 is unmasked, the microprocessor will be interrupted even if the device has removed the interrupt signal. This flip-flop will be automatically reset when the microprocessor responds to an RST 7.5 interrupt. Bit 4 of the accumulator in the SIM instruction allows explicitly resetting the RST 7.5 memory even if the microprocessor did not respond to it.

Undefined Bit D_5 Bit D_5 is not used by the SIM instruction

SOD Enable Bit D_6 Bit D_6 is used for serial output data enable.

Serial Output Data Bit D_7 Bit D_7 is used for serial output data. The SIM instruction is used for serial data transmission. When the SIM instruction is executed, the content of bit D_7 of accumulator will be output on the SOD line.

Example 6.1 Set the interrupt masks so that RST 5.5 is enabled, RST 6.5 is masked, and RST 7.5 is enabled.

Sol. Initially determine the contents of the accumulator

Enable 5.5	bit $D_0 = 0$
Disable 6.5	bit $D_1 = 1$
Enable 7.5	bit $D_2 = 0$
Allow setting the masks	bit $D_3 = 1$
Don't reset the flip flop	bit $D_4 = 0$
Bit 5 is not used	bit $D_5 = 0$
Don't use serial data	bit $D_6 = 0$
Serial data is ignored	bit $D_7 = 0$

SOD	SOE	X	R7.5	MSE	M 7.5	M 6.5	M 5.5
0	0	0	0	1	0	1	0

Content of accumulator is 0AH. The program for the above operation is given below:

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	EB		EI		Enable all interrupts
8001	3E, 0A		MVI	A, 0AH	Mask to enable RST 7.5, and 5.5, disable 6.5
8003	30		SIM		Apply the settings RST masks

6.5.2 RIM Instruction

The RIM instruction loads the accumulator with 8 bits, which consists of the status of the interrupt mask, the interrupt, enable, the pending interrupts and one bit of serial input data. Figure 6.9 shows the accumulator content for RIM instruction

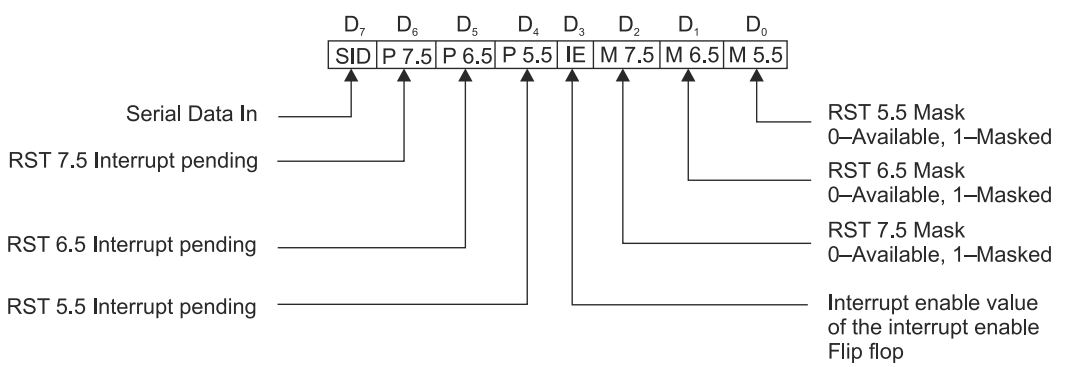


Fig. 6.9 Accumulator content for RIM instruction

Interrupt Mask Bits D_0, D_1, and D_2 Bits D_0 D_1 and D_2 represent the current setting of the mask for each of RST 7.5, RST 6.5 and RST 5.5. A high level shows that interrupt is masked and low level means that interrupt is not masked. Bits D_0 D_1 and D_2 return the contents of the three mask flip flops. These bits can be used by a program to read the mask settings in order to modify only the right mask.

Interrupt Enable Bit D_3 Bit D_3 is the interrupt enable flag. This bit shows whether the maskable interrupt process is enabled or not. When it is high, interrupt is enabled. If it is low, interrupt is disabled. It returns the contents of the interrupt enable flip-flop. It can be used by a program to determine whether or not interrupts are enabled.

Interrupts Pending Bits D_4, D_5, D_6 Bits D_4, D_5 and D_6 represent the pending interrupts. Bits D_4 and D_5 return the current value of the RST 5.5 and RST 6.5 pins. Bit D_6 returns the current value of the RST 7.5 memory flip-flop. A high level on bits D_4, D_5 and D_6 states that interrupt are pending. When a low level on bits D_4, D_5 and D_6 states that interrupts are not pending.

Serial Input Data Bit D_7 Bit D_7 is used for Serial Data Input. The RIM instruction reads the value of the SID pin on the microprocessor and returns it in this bit.

Example 6.2 Write instructions to call interrupt service subroutine 003CH corresponding to RST7.5 if it is pending. Assume the content of accumulator is 20H on executing of RIM instruction.

Sol. The program for above operation is given below:

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
9000	20		RIM		Accumulator content is 20H on executing RIM instruction
9001	E6, 40		ANI	40H	And immediate with 40H
9003	CD, 3C, 00		CALL	003CH	Call interrupts service routine for RST 7.5, when RST 7.5 is pending.

Initially, the content of the accumulator is 20H.

SID	P 7.5	P 6.5	P 5.5	IE	M 7.5	M 6.5	M 5.5
0	0	1	0	0	0	0	0

After immediate ANDing with 40H, the content of the accumulator is given below.

SID	P 7.5	P 6.5	P 5.5	IE	M 7.5	M 6.5	M 5.5
0	1	0	0	0	0	0	0

The 8085 microprocessor has two additional instructions such as Enable Interrupt (EI) and Disable Interrupt (DI). These instructions can enable or disable all the interrupts except TRAP interrupt. The interrupt enable flip-flop is manipulated using the EI/DI instructions. Actually, EI and DI instruction generates internally EI and DI signals. The connections of EI and DI are depicted in Fig. 6.4. The EI signal sets the SR flip-flop and generates an interrupt output signal. The interrupt enable signal enables the AND gates at the RST 7.5, RST 6.5 and RST 5.5 inputs. The DI signal can reset the SR flip-flop and makes the interrupt enable output becomes low. Then all maskable interrupts are disabled. The application of EI and DI is shown in Fig. 6.10.

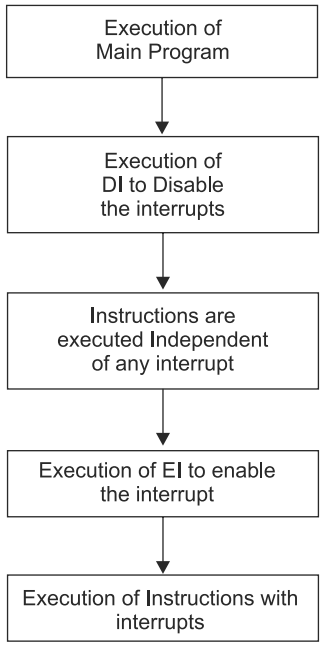


Fig. 6.10

Example 6.3 Write instructions to enable interrupt RST 7.5 and disable RST 6.5 and RST 5.5.

Sol. The content of accumulator for the SIM instruction to enable interrupt RST 7.5 and disable RST 6.5 and RST 5.5 are

SOD	SOE	X	P 7.5	MSE	M 7.5	M 6.5	M 5.5
0	0	0	0	1	0	1	1

Bit D_2 is set to 0 and Bit D_1 and D_0 are reset to 1 to enable interrupt RST 7.5 and disable RST 6.5 and RST 5.5 respectively. The content of the accumulator is 0BH. The instructions for the above operation are given below:

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8500	FB		EI		Enable all interrupts
8501	3E, 0B		MVI	A, 0BH	Load 0BH to enable RST 7.5, and disable RST 5.5 and RST 6.5
8502	30		SIM		Apply the settings to enable RST 7.5, and disable RST 5.5 and RST 6.5

Example 6.4 Write instructions for the following operations:

The microprocessor has completed the RST 7.5 interrupt. Check if RST 5.5 is pending. When RST 5.5 is pending, enable RST 5.5 interrupt.

Sol. The content of the accumulator for the RIM instruction for RST 5.5 pending is 10H.

SOD	P 7.5	P 6.5	P 5.5	IE	M 7.5	M 6.5	M 5.5
0	0	0	1	0	0	0	0

The content of the accumulator for the SIM instruction to enable interrupt RST 5.5 and disable RST 6.5 and RST 7.5 are

SOD	SOE	x	P 7.5	MSE	M 7.5	M 6.5	M 5.5
0	0	0	0	1	1	1	0

Bit D_0 is set to 0 and bits D_1 and D_2 are reset to 1 to enable interrupt RST 5.5 and disable RST 6.5 and RST 7.5 respectively. Then content of the accumulator is 0EH. The instructions for above operation are as follows:

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
9000H	20		RIM		Read interrupt mask
9001H	47		MOV	B, A	Store mask information into B register
9002H	E6, 10		ANI	10H	Check whether RST 5.5 is pending or not
9004H	C2, Level		JNZ	Level	
9007H	FB		EI		
9008H	C9		RET		RST 5.5 is not pending, and return to main program
9009H	78		MOV	A, B	Read bit pattern and RST 5.5 is pending
900AH	E6, 0E		ANI	0EH	Enable RST 5.5
900CH	F6, 08		ORI	08H	Enable SIM by setting D_3
900EH	30		SIM		
900FH	CD, 2C, 00		CALL	002CH	Call service routine from 002CH

The RIM instructions check whether any interrupts are pending. The ANI 10H masks all bits except D_4 to check pending RST 5.5 interrupt. Bit D_4 indicates that RST 5.5 is pending. The ANI 0E and ORI 08H instructions enable RST 5.5 interrupt. The CALL instruction transfers the execution of the program to the service routine of RST 5.5.

6.6 PENDING INTERRUPTS

The 8085 microprocessor has five interrupt lines. Only one interrupt may occur during an ISR and other interrupts remain pending. If more than one interrupt wants service simultaneously, the microprocessor can only respond to one interrupt at a time. Therefore, some priority has been assigned to different interrupt lines which allows their signals to reach the microprocessor according to the priority. This problem can be solved by one additional circuit known as the *priority encoder, 74LS148*.

This circuit has 8 inputs and 3 outputs. The inputs are assigned increasing priorities according to the increasing index of the input, so that Input 7 has highest priority and Input 0 has the lowest. Using the RIM instruction, the programmer can read the status of the interrupt lines and find if there are any pending interrupts.

6.7 SERIAL MODE OPERATION USING SID AND SOD PINS OF 8085 MICRO-PROCESSOR

For serial mode of operation, 8085 microprocessor has two serial input and output pins such as SID and SOD. SID represents serial input data and SOD represents serial output data. The SID and SOD pins are used to read/write one-bit data to and from peripheral devices. There are two software instructions such as RIM and SIM which are associated with SID and SOD lines.

The Serial Input Data (SID) line exists inside the 8085 microprocessor as Pin 5. One bit data can be externally read and stored using the SID line. The data which is read is stored in the A_7 bit of the accumulator whenever the RIM instruction is executed. Fig. 6.11 shows the serial input data through SID pin and RIM instruction.

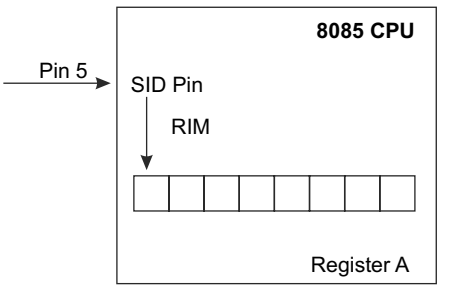


Fig. 6.11 Execution of RIM instruction

When the SID line is connected with +5 V and RIM instruction is executed, the accumulator's MSB bit will be loaded with a logic 1 and the content of the accumulator after the execution of RIM instruction is 80 H as depicted in Fig. 6.12.

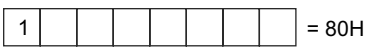


Fig. 6.12 Accumulator content after the execution of RIM instruction with SID = 1

If the SID is connected with 0 V (Ground) and RIM is executed, the accumulator's MSB bit will be loaded with a logic 0. Then the content of the accumulator after the execution of RIM instruction is 00H as shown in Fig. 6.13.

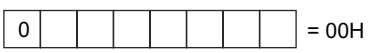


Fig. 6.13 Accumulator content after the execution of RIM instruction with SID = 0

The Serial Output Data (SOD) line exists inside the 8085 microprocessor as Pin 4. One bit data can be externally written in this port. To write data into this port, the SIM instruction must be executed. The data which is to be written in this port must be stored in the A_7 bit of the accumulator. The A_6 bit of the accumulator is called SOE (serial Output Enable) and this bit should be 1 to enable serial data output. Figure 6.14 shows the serial output data through SOD pin and SIM instruction.

To write 1 in the SOD line, load the accumulator with C0H and execute the SIM instruction as shown in Fig. 6.15.

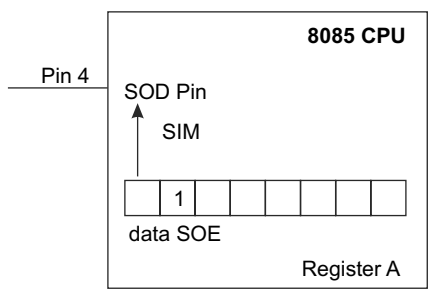


Fig. 6.14 Serial output data through SOD pin and SIM instruction

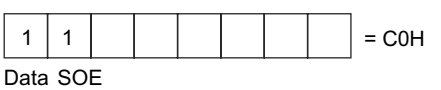


Fig. 6.15 Accumulator content after the execution of SIM instruction

To write 0 in the SOD line, load the accumulator with 40H and execute the SIM instruction as shown in Fig. 6.16.

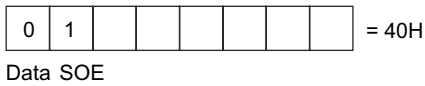


Fig. 6.16 Accumulator content after the execution of SIM instruction

Review Questions

6.1 What are the interrupt pins of 8085 microprocessor? Explain briefly maskable and non-maskable interrupts. What is meant by priority interrupts?

6.2 What are the software interrupts of the 8085 microprocessor? Mention interrupts instructions with their Hex code and vector address. How is the vector address for a software interrupt determined?

6.3 Draw the TRAP interrupt and explain briefly. Why is the TRAP input edge and level sensitive? Can the TRAP interrupt be disabled by a combination of hardware and software? Write some applications of TRAP interrupt.

6.4 Draw the SIM instruction format and discuss with an example.

6.5 Draw the RIM instruction format and discuss with an example. "A RIM instruction should be performed immediately after TRAP occurs." Why?

6.6 Explain the software instructions EI and DI. "When returning back to the main program for Interrupt Service Subroutine (ISS), the software instruction EI is inserted at the end of the ISS". Why?

6.7 What do you mean by priority interrupts? Explain the operation of different interrupts available in 8085, with the help of a circuit diagram.

6.8 Distinguish between

(i) Vectored and non-vectored interrupt

(ii) Maskable and non-maskable interrupt

(iii) Internal and external interrupt

(iv) Software and hardware interrupt

6.9 Draw the timing diagram of RST 5 instruction and explain briefly.

6.10 Explain interrupt driven I/O technique. How does an 8085 microprocessor respond to INTR interrupt signal?

6.11 After the execution of RIM instruction, the accumulator contains 49H. Explain the accumulator contents.

6.12 Which interrupts are marked after the execution of the following instructions

MVI A, 1DH, SIM

6.13 Discuss the serial mode operation using SID and SOD pins of 8085 microprocessor.

6.14 What are vectored and non-vectored interrupts? Explain the instructions RIM and SIM. Write an instruction to enable the RST 7.5, RST 6.5 and disable RST 5.5

Multiple-Choice Questions

6.1 What is the vector address of a software interrupt?
(a) vector address = interrupt number × 8 (b) vector address = interrupt number × 16
(c) vector address = interrupt number × 12 (d) vector address = interrupt number × 4

6.2 If interrupt instruction RST 7 is executed, the program will jump to memory location
(a) 2000H (b) 0020H (c) 0038H (d) 0016H

6.3 Which is the highest priority interrupt?
(a) TRAP (b) RST 6.5 (c) RST 5.5 (d) RST 7.5

6.4 SIM instruction is used to
(a) enable RST 7.5, 6.5 and 5.5 (b) disable RST 7.5, 6.5 and 5.5
(c) enable or disable RST 7.5, 6.5 and 5.5 (d) none of these

6.5 RST 7.5 interrupt is
(a) Vectored and maskable (b) Non-vectored and maskable
(c) Non-vectored and non-maskable (d) Vectored and Non-maskable

6.6 Which one of the following is the software interrupt of 8085A microprocessor?
(a) RST 7.5 (b) EI (c) RST0 (d) None of these

6.7 In order to enable TRAP interrupt, which of the following instructions is/are needed?
(a) EI only (b) SIM only (c) EI and SIM (d) None of these

6.8 The call location for TRAP interrupt is
(a) 0000H (b) 0020H (c) 0024H (d) 0034H

6.9 The interrupt pin available in the 8085A microprocessor chip is
(a) ALE (b) HLDA (c) INTER (d) SOD

6.10 The Call location for RST 7.5 interrupt is
(a) 003CH (b) 0034H (c) 002CH (d) 0000H

Answers to Multiple-Choice Questions

6.1 (a)	6.2 (c)	6.3 (a)	6.4 (c)
6.5 (a)	6.6 (c)	6.7 (a)	6.8 (c)
6.9 (c)	6.10 (a)		

CHAPTER

7

8051 Microcontroller Architecture

7.1 INTRODUCTION

The microprocessor is the core of any computer system, but the microprocessor by itself is completely useless, until external peripheral devices are connected with it to interact with the outside world. The microcontroller is a single-chip microprocessor system which consists of CPU, data and program memory, serial and parallel I/O ports, timers and external as well as internal interrupts. Actually, a microcontroller is an entire computer manufactured on a single chip. Figure 7.1 shows the schematic block diagram of a microcontroller. The comparison between a microprocessor and microcontroller is given in Table 7.1. The single-chip microcontrollers are used in instrumentation and process control, automation, remote control, office automation such as printers, fax machines, intelligent telephones, CD players and some sophisticated communication equipments. Due to integration of all function blocks on a single-chip microcontroller IC, the sizes of control board and power consumption are reduced, system reliability increased and also provides more flexibility. The other advantages of microcontroller-based systems are easy troubleshooting and maintenance; interfacing can be done for additional peripherals, and better software security.

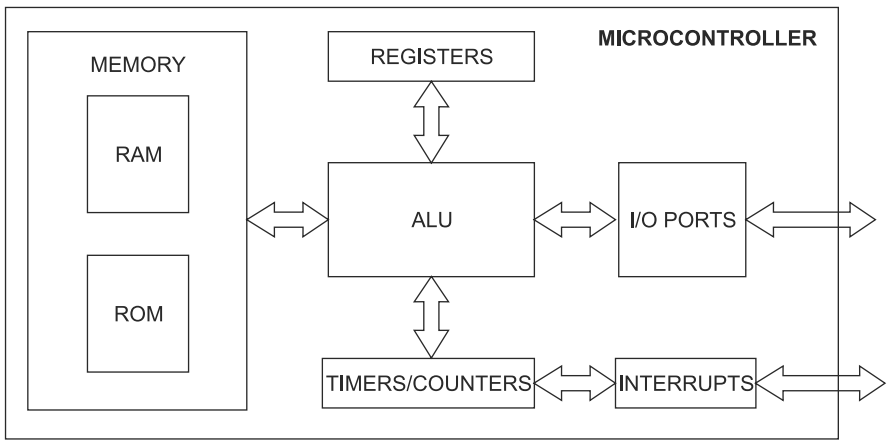
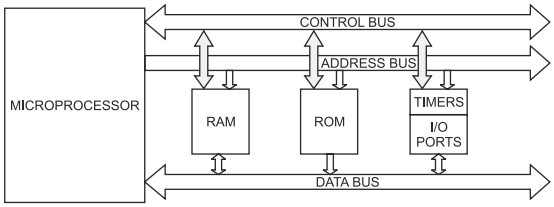
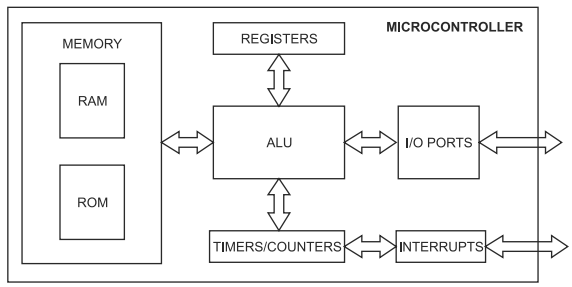


Fig. 7.1 Microcontroller

Table 7.1 Comparison between microprocessor and microcontroller

Microprocessor	Microcontroller
Microprocessor is a single-chip CPU. The block diagram of microprocessor is given below.	The microcontroller is a single-chip microcomputer system as given below.
	
It consists of an ALU to perform arithmetic and logic manipulations, registers and a control unit.	It consists of CPU, data and program memory, serial and parallel I/O, timers, external and internal interrupts.
It has address bus, data bus and control bus for interfacing with the outside world.	Microcontroller communicates with outside world through P_0, P_1, P_2 and P_3 ports. Ports can be used as address and data bus depending upon control signals.
RAM and ROM are not incorporated within chip.	RAM is smaller, but it is enough for small applications. If it is not sufficient, then external memory may be added in the microcontroller-based system.
Microprocessors are used as the CPU in the microcomputer systems.	Microcontrollers are used in small embedded system products to perform control-oriented functions.
Microprocessor instructions perform operations based on nibbles and bytes.	Microcontroller instructions are able to perform bit-level operations and other operations such as based on nibbles, bytes, words, or even double words.
Microprocessors are available from 4-bit to 64-bit. 4-bit microprocessors are used for simple applications. 8-bit microprocessors are most commonly used in different applications. 16-bit, 32-bit and 64-bit microprocessors are used for personal computers and high-speed applications.	Microcontrollers are available from 4-bit to 32-bit. 4-bit microcontrollers are used for simple applications. 16-bit and 32-bit microcontrollers are used for high speed applications. 8-bit microcontrollers are most commonly used in different applications.

4-bit to 32-bit microcontrollers are available in the market. Based on the number of bits, microcontrollers are classified into 4-bit microcontrollers, 8-bit microcontrollers, 16-bit microcontrollers and 32-bit microcontrollers. Four-bit microcontrollers are extensively used in electronics toys and example of 4-bit microcontrollers are illustrated in Table 7.2. Generally, 8-bit microcontrollers are used in various control applications such as speed control, position control, and any process control system. Table 7.3 shows the different 8-bit microcontroller ICs with their features. The 16-bit microcontrollers are developed for high speed control applications such as servo control system, robotics, etc. These microcontrollers can be programmed in high-level programming languages as well as assembly language programming. Some 16-bit microcontrollers are given in Table 7.4. The 32-bit microcontrollers are used for very high speed operations in robotics, image processing, automobiles, intelligent control system, and telecommunications. Commonly used 32-bit microcontrollers are 80960 and MC683xx. The microcontrollers are most commonly used in consumer products, automotive systems, different industrial applications and high speed data processing. A list of microcontroller

applications is given below:

Consumer Products Washing machines, micro-ovens, printers, copiers, compressors, AC machines.

Industrial Applications Control power electronics circuits, DC and AC motor drives, speed and position control, and motion control, etc.

Automation Antilock braking systems, electronic power steering systems, etc.

High-speed data processing Video conference, image processing, video processing, real-time compression systems and security, etc.

Table 7.2 4-bit microcontroller families

IC	No. of Pins	No. of I/O Pins	On-chip data memory RAM	On-chip program memory, ROM	Counters	Extra features
TLCS 47	42	35	128 bytes	2K ROM		Serial I/O
TMS1000	28	23	64 bytes	1K ROM		LED display
COP 420	28	23	64 bytes	1K ROM	1	Serial I/O
MSM6411	16	11	32 bytes	1K ROM		
HMCS 40	28	10	32 bytes	512 ROM		

Table 7.3 8-bit microcontroller families

IC	No. of Pins	No. of I/O Pins	On-chip data memory, RAM	On-chip program memory, ROM	No. of 16-bit timers/counters	No. of vectored interrupts	Extra features
8031	40	32	128 bytes	None	2	5	Full duplex serial I/O
8032	40	32	256 bytes	None	3	6	Full duplex serial I/O
8051	40	32	128 bytes	4K ROM	2	5	Full duplex serial I/O
8052	40	32	256 bytes	8K ROM	3	6	Full duplex serial I/O
8751	40	32	128 bytes	4K ROM	2	5	Full duplex serial I/O
8752	40	32	256 bytes	8K ROM	3	6	Full duplex serial I/O

Table 7.4 16-bit microcontroller families

IC	No. of Pins	No. of I/O Pins	On-chip data memory, RAM	On-chip program memory, ROM	No. of 16-bit timers/counters	No. of vectored interrupts	Extra features
HPC 46164	60	52	512 bytes	16K bytes ROM	4	8	External memory up to 64K, full duplex UART, ADC
8096	68	40	256 bytes	8K bytes ROM	2	7	External memory up to 64K
8094	48	24	256 bytes	–	2	7	External memory up to 64K

Contd.

Table 7.4 *(Contd.)*

8097	68	24	256 bytes	–	2	8	External memory up to 64K
8095	48	20	256 bytes	–	2	8	External memory up to 64K
8397	68	24	256 bytes	8K bytes ROM	2	8	External memory up to 64K
8395	48	20	256 bytes	8K bytes ROM	2	8	External memory up to 64K
80C196 EA	160	83	1K bytes register RAM 3K bytes code RAM	8K bytes ROM	4	16	External memory up to 2MB, Serial I/O, ADC

7.2 ARCHITECTURE OF 8051 MICROCONTROLLER

Intel developed the 8051 microcontroller in 1980s. This microcontroller IC consists of standard on-chip peripherals, i.e., timers, counters, and UART, 4K bytes of on-chip program memory and 128 bytes of data memory. The 8051 have separate address spaces for program memory and data memory with the help of modified Harvard architecture. The program memory can be increased to 64 KB. Approximately, 4K bytes of program instructions can be stored in the internal memory of the 8051. The 8051 can address up to 64 KB of external data memory. The 8051 memory architecture includes 128 bytes of data memory that are accessible directly by its instructions. A 32-byte segment of this 128-byte memory block is bit addressable by a subset of the 8051 instructions, namely, the bit-instructions. For this reason, 8051 microcontroller is known as Boolean processor and used to deal with binary input and output conditions very efficiently.

In addition, the device is a low power static design, which offers a wide range of operating frequencies down to zero. Two software selectable modes of power reduction—idle mode and power-down mode are available. The idle mode freezes the CPU while allowing the RAM, timers, serial port, and interrupt system to continue functioning. The power-down mode saves the RAM contents but freezes the oscillators.

This IC has the following features:

- Fabricated with Philips high-density CMOS technology with operation from 2.7 V to 5.5 V
- 8051 Central Processing Unit
  - 4K × 8 ROM
  - 128 × 8 RAM
  - Three 16-bit counter/timers
  - Full duplex serial channel
  - Boolean processor
  - Full static operation
  - Low voltage 2.7V to 5.5V at 16 MHz operation
- *Memory addressing capability*
  - 64K ROM and 64K RAM

- *Power control modes*
  - Idle mode
  - Power-down mode
- CMOS and TTL compatible
- Frequency range 0 to 33MHz
- 4 level priorities interrupt
- 6 interrupt sources
- Four 8-bit I/O ports
- Programmable clock out
- Asynchronous port reset

Figure 7.2 shows the simplified block diagram of 8051 microcontroller. The detailed architecture of 8051 microcontroller has been depicted in Fig. 7.3 which consists of ALU, control and timing unit, RAM/EPROM/ ROM, registers, latches and drivers for ports P_0, P_1, P_2, and P_3. The operation of each block has been explained in this section.

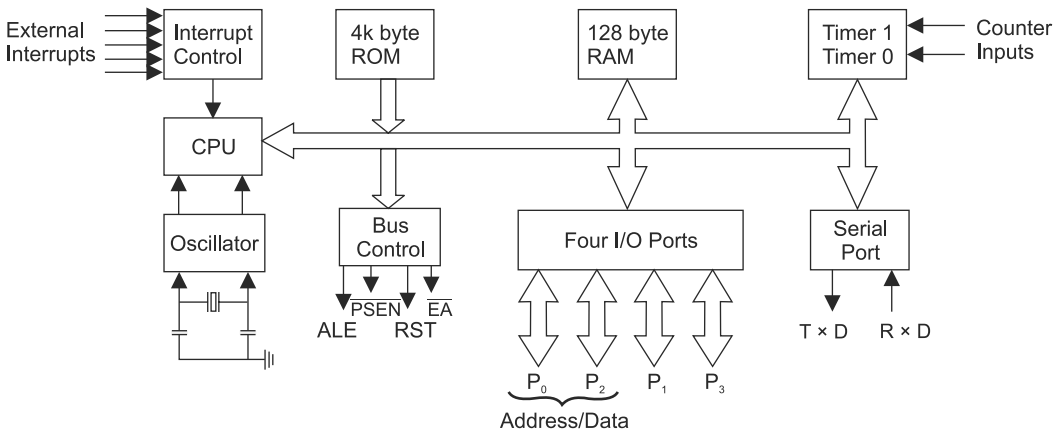


Fig. 7.2 Schematic block diagram of 8051 microcontroller

Accumulator (ACC) The Accumulator register (ACC) acts as an operand register. The accumulator may be referred as implicit or specified in the instruction by its SFR address 0E0H. ACC commonly used for data transfer and arithmetic instructions. Accumulator is also bit addressable. ACC.2 states the bit 2 of ACC register. After any arithmetic operations, the result is stored in ACC.

B Register The B register is used during multiply and divide operations to store the second operands for multiply and divide instructions MUL AB and DIV AB respectively. After multiplication and division, part of the result such as upper 8-bits of multiplication result and remainder in case of division are stored in B register. This register is commonly used as temporary register and can also be accessed through its SFR address of 0F0H. This register is also bit addressable. It can be used as general purpose register except MUL and IDIV instructions.

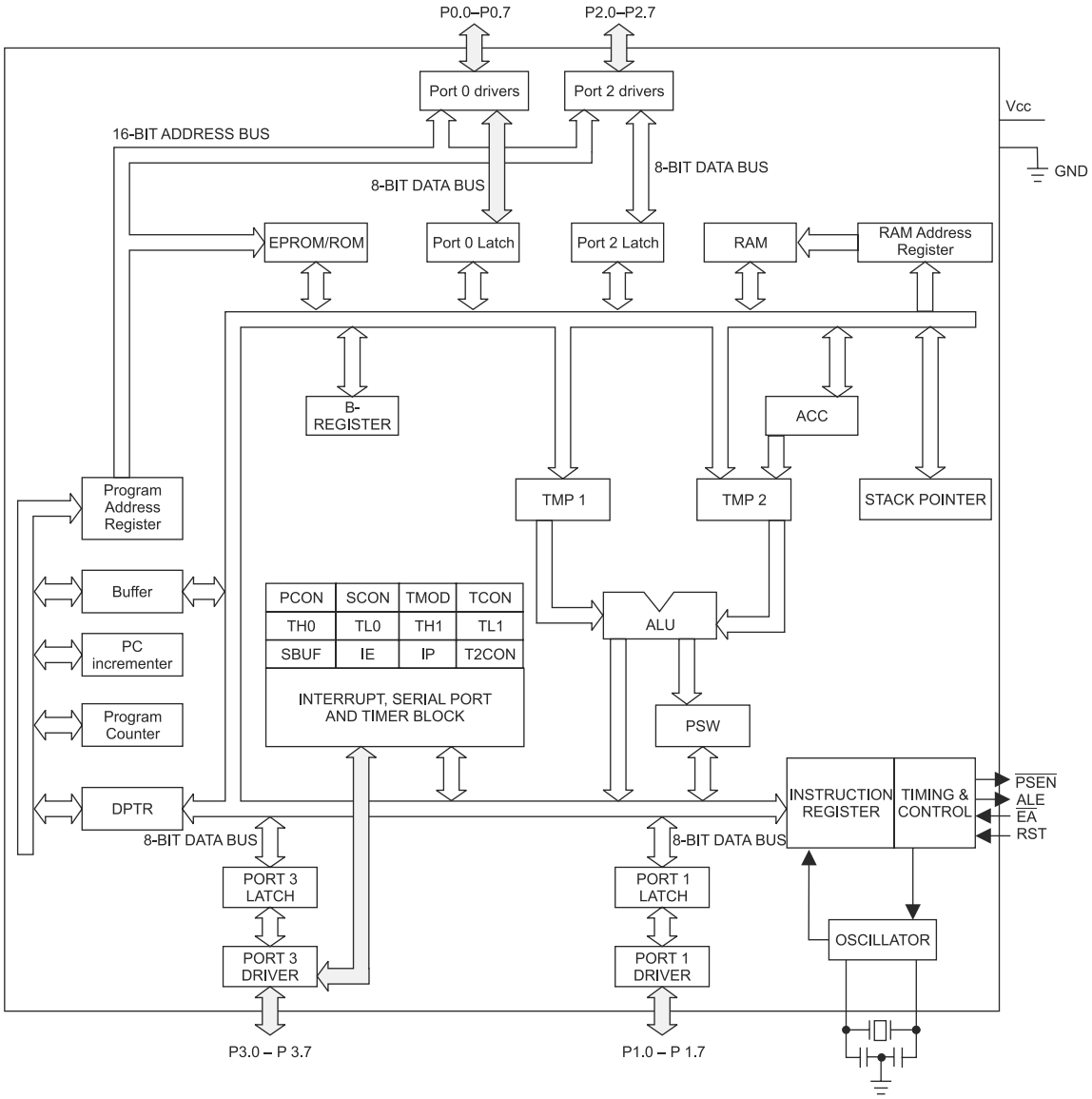


Fig. 7.3 Block diagram of 8051 microcontroller

Program Status Word (PSW) This is a special-function register. This register consists of the different status bits that reflect the current state of microcontroller. Figure 7.4 shows the PSW which resides in the SFR space. It contains the Carry (CY), the Auxiliary Carry (AC), the two register bank select bits (RS1 and RS0), the Overflow flag (OV), a Parity bit (P), and two user-definable status flags. The carry bit can serve the function of a carry bit in arithmetic operations and it also serves as the accumulator for a number of Boolean operations. The auxiliary carry bit is used in addition of BCD numbers. This bit is set if a carry is generated out of bit 3 into bit 4. F0 is a general-purpose flag bit available for user applications. The program status bits and their functions are given in Table 7.5. The bits RS1 and RS0 are used to select the register bank from four

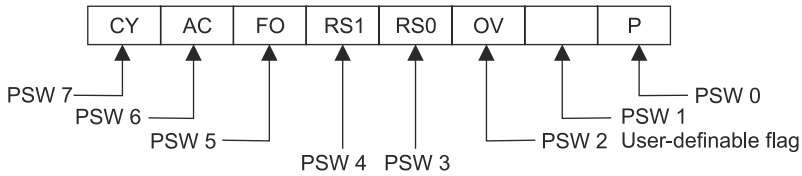


Fig. 7.4 Program Status Word (Bit Addressable)

register banks as depicted in Table 7.6. The overflow flag is used for signed arithmetic operation to determine whether the result is out of range after signed arithmetic operation. When the result is greater than +127 or less than –128, OV flag bit is set.

The parity bit reflects the number of 1s in the accumulator. When the accumulator contains an odd number of 1s, P = 1. If the accumulator contains an even number of 1s, P = 0.

Table 7.5 Program status bits and their functions

Symbol	Position	Address	Function
CY	PSW.7	D7H	Carry flag
AC	PSW.6	D6H	Auxiliary carry flag.
F0	PSW.5	D5H	Flag 0. Available to the user for general purpose.
RS1	PSW.4	D4H	Register bank selector bit 1. Set by software to select the register bank which will be used.
RS0	PSW.3	D3H	Register bank selector bit 0. Set by software to select the register bank which will be used.
OV	PSW.2	D2H	Overflow flag.
–	PSW.1	D1H	Usable as a general-purpose flag.
P	PSW.0	D0H	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of '1' in the accumulator.

Table 7.6 Register Bank Selections

RS1	RS0	Register Bank	Address Range
0	0	0	00H – 07H
0	1	1	08H – 0FH
1	0	2	10H – 17H
1	1	3	18H – 1FH

Stack Pointer (SP) This is an 8-bit register. SP is incremented before the data is stored onto the stack using PUSH/CALL instructions execution. During PUSH operation, first increment SP and then copy the data. In the POP operation, initially copy the data and then decrement SP. The 8-bit address of the stack top is stored in this register. The stack can be located anywhere in the on-chip 128-byte RAM. Initially, the stack pointer is initialised to 07H after reset operation. Hence the stack begins at locations 08H. The stack can be relocated by setting SP to the upper memory area in 30H to 7FH.

Data Pointer (DTPR) DTPR is 16-bit register. It consists of a higher byte (DPH) and a lower byte (DPL) of a 16-bit external data RAM address. It can be accessed as two 8-bit registers or a 16-bit register. DTPR has been given two addresses in the special-function register bank. DPTR is very useful for string operations and

look-up table operations. With 16-bit DPTR, a maximum of 64 K of off-chip data memory and a maximum of 64 K of off-chip program memory can be addressed. For this, 16-bit memory location address will be stored in DPTR through MOV DPTR,#XXXX instruction. To read data from memory location and write data in memory location, MOVX R0,@DPTR and MOVX @DPTR,#XX are used respectively. This can also be used as a general-purpose register. To increment the contents of DPTR, INC DPTR instruction is executed. But, there is no such instruction in 8051 to decrement the DPTR.

Port 0, Port 1, Port 2, Port 3 Latches and Drivers Each latch and corresponding driver of ports 0–3 is allotted to the corresponding on-chip I/O port. All ports are bidirectional input/output ports of 8 bits each. The addresses of latches are stored in the special function register bank. Using these addresses, ports 0–3 can be communicated with other ICs and all input/output signals. The output drivers of Ports 0 and 2, and the input buffers of Port 0 and Port 2, are used in accesses to external memory. In this case, Port 0 outputs the low byte of the external memory address and Port 2 outputs the higher byte of the external memory address. P1 can be used as input/output port and it acts an important role in programming of internal memory of 8051. P3 can also be used as I/O port. The pins of Port 3 have alternate functions such as serial input line (P3.0), serial output line (P3.1), external Interrupt lines (P3.2, P3.3), external timer input lines (P3.4, P3.5), external data memory write strobe (P3.6) and external data memory read strobe (P3.7).

The port registers specify the value to be output on the specific output port or read value from the specified input port. Ports are also used as bit addressable. The first bit of port has the same address as the register. Suppose the port address of P1 is 90H in SFR. The address of P1.0 is 90H and Port address of P1.7 is 97H.

Serial Port Data Buffer The serial port data buffer internally consists of two independent registers such as transmit buffer and receive buffer at the same location. The transmit buffer is a parallel-in serial-out register. The serial data receive buffer is a serial-in parallel-out register. The serial data buffer is identified as SBUF. If data is moved to SBUF, it goes to the transmit buffer and sets serial transmission. When data is moved from SBUF, it receives serial data from the receive buffer.

Timing Registers There are two 16-bit timing registers in 8051. The 16-bit timer register can be accessed as their lower and upper bytes. The TH_0 and TL_0 represent the higher byte and lower byte of the timing register 0 respectively. In the same way TH_1 and TL_1 represent the higher byte and lower byte of the timing register 1 respectively. All timing registers can be accessed by using the four different addresses allotted to them. The addresses of registers have been stored in the special function registers (SFR) from 80H to FF. In the 8052, one more pair (TH_2 and TL_2) exists for Timer/Counter 2. The operation of timing register may be timing or counting. There is a timer control (TCON) and a timer mode registers (TMOD) to configure all timers/counters in various modes. These timers can be used to measure pulse width from one microsecond to 65 ms, generate longer time delay and time interval.

Control Registers The control register consists of special function registers such as Interrupt Priority (IP), Interrupt Enable (IE), Timer Mode (TMOD), Timer Control (TCON), Serial Port Control (SCON) and Power Control (PCON). All of these registers have allotted addresses in the special function register bank of 8051 microcontroller.

Capture Registers Register pair RCAP2H and RCAP2L exist only in the 8052. Actually these are the capture registers for the Timer-2. When Timer-2 operates in capture mode operation, a transition at the 8052 T2EX pin causes TH_2 and TL_2 to be copied into RCAP2H and RCAP2L. Timer-2 has a 16-bit auto-reload mode and RCAP2H and RCAP2L hold the reload value for this mode operation.

Timing and Control Unit The timing and control unit generates all the necessary timing and control

signals required for the internal operation of the microcontroller. This unit also generates necessary control signals ALE, $\overline{PSEN}$, $\overline{RD}$ and $\overline{WR}$ to control the external system bus.

Oscillator The oscillator circuit generates the basic timing clock signal for the operation of the microcontroller using crystal oscillator. The 80C51 microcontroller operates at about 12 MHz frequency. In this microcontroller, only the quartz crystal oscillator is connected with microcontroller externally and remaining oscillator circuit components are incorporated in the chip.

Instruction Register (IR) This register is used to decode the opcode of any instruction to be executed. After decoding, this register sends the decoded information to the timing and control unit to generate necessary signals for the execution of the specified instruction.

Program Address Register This is an on-chip EPROM and a basic circuit mechanism to internally address it. EPROM is available in 8051, 8052, 8751 and 8752 microcontrollers and it is not available in 8031 and 8031 microcontrollers.

RAM This block provides internal 128 bytes of RAM.

RAM Address Register The RAM address register is used to generate address of RAM internally.

ALU (Arithmetic and Logic Unit) The ALU performs 8-bit arithmetic and logical operations when the operands are held at the temporary registers TMP1 and TMP2. These temporary registers cannot be accessed by users. The output of the ALU is stored in accumulator in most of the arithmetic and logical operations with few exceptions. Apart from addition and subtraction operations, the 8051 microcontroller also performs multiplication and division operations. The logical operations such as AND, OR, NOT, Ex-OR operations are also performed in ALU.

SFR (Special Function Registers) This register bank is a set of registers, which can be addressed using their respective addresses in the range of 80H to FFH.

7.3 MEMORY ORGANIZATION

The 80C51 microcontroller has separate address spaces for program memory as well as data memory. Figures 7.5 and 7.6 show the program and data memory respectively. The logical separation of program and data memory allows the data memory to be accessed by 8-bit addresses and this 8-bit memory access can be stored quickly and manipulated by an 8-bit CPU. The 16-bit data memory addresses can also be generated through the DPTR register and data can be read from external memory.

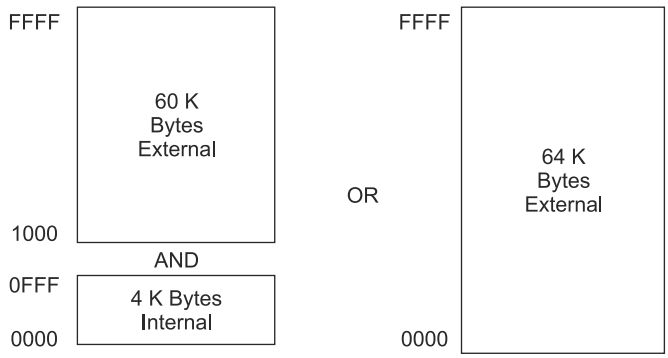


Fig. 7.5 Program memory (Read Only) structure

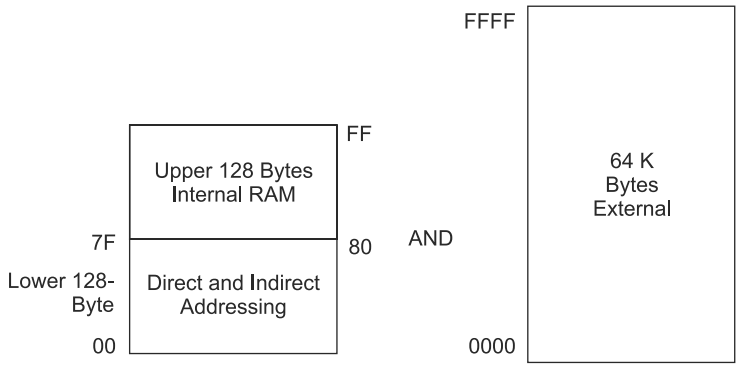


Fig. 7.6 Data memory (Read/Write) structure

The program memory can be read only such as ROM and EPROM. There can be up to 64K bytes of program memory. In the 80C51, the lowest 4K bytes of program exist on-chip and 60K bytes program memory is external memory. In the ROM less versions of microcontrollers program memory is external and its capacity is about 64K. The read strobe for external program memory is the $\overline{PSEN}$ (Program Store Enable). Depending on the instructions, the same address can refer to two logically and physically differed memory locations.

The data memory has a separate address space from program memory. In the 80C51 microcontroller, the lowest 128 bytes of data memory are available on-chip and maximum 64K bytes of external RAM can be addressed in the external data memory space. In ROM less version of microcontroller, the lowest 128 bytes are on-chip. The CPU of microcontroller generates read and write signals during external data memory accesses.

7.3.1 Program Memory

A map of the lower part of the program memory is shown in Fig. 7.7. After reset, the microcontroller starts fetching instructions from 0000H and the CPU also starts execution from location 0000H. This can be either on-chip memory or external memory depending on the value of the $\overline{EA}$ pin. When $\overline{EA}$ is low, the program memory is external. If $\overline{EA}$ is high, the address from 0000H to 0FFFH will refer to on-chip memory and the address from 1000H to FFFFH can refer to external memory as depicted in Fig. 7.5.

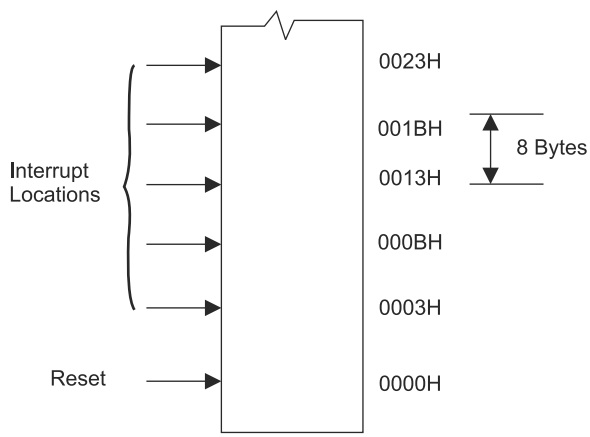


Fig. 7.7 8051 Program memory

It is clear from Fig. 7.7 that each interrupt is assigned a fixed location in program memory. When any interrupt is executed, the CPU will jump to that specified location and it starts execution of the service routine. For example, external interrupt-0 is assigned to memory location 0003H. When external interrupt 0 is going to be used, its service routine must start at location 0003H. If the interrupt is not going to be used, its service location is available as general purpose program memory.

Usually the interrupt service locations are available at 8-byte intervals: 0003H for external interrupt-0, 000BH for timer-0, 0013H for external interrupt-1, and 001BH for timer-1, etc. When the interrupt service routine is short, it can reside entirely within that 8-byte interval. But the longer service routines can use a jump instruction to skip over subsequent interrupt locations.

The lowest 4K bytes of program memory will be either in the on-chip ROM or in an external ROM. The internal or external program memory selection can be made by $\overline{EA}$ (External Access) pin. In the 80C51, if the $\overline{EA}$ pin is V_{CC}, then the program fetches to internal ROM addresses 0000H through 0FFFH. In this case, program fetches to addresses 1000H through FFFFH are directed to external ROM.

When the $\overline{EA}$ pin is grounded, all program fetches are directed to external ROM.

The read strobe to external ROM, $\overline{PSEN}$ is used for all external program fetches. The $\overline{PSEN}$ is not activated for internal program fetches.

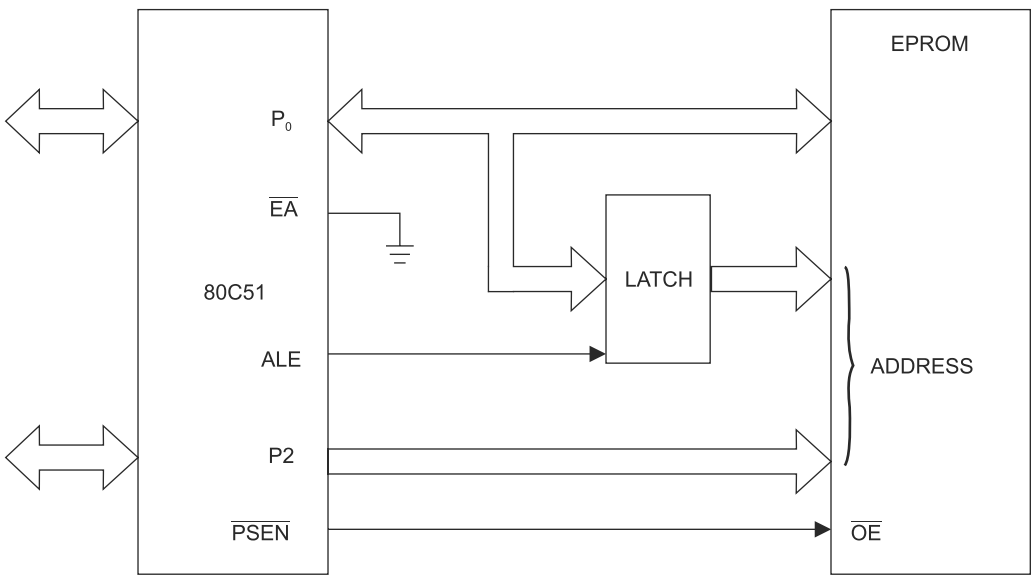


Fig. 7.8 Executing from external program memory

Figure 7.8 shows the hardware configuration for external program execution. Program memory addresses are always 16 bits wide. Ports 0 and 2 are dedicated to bus functions during external Program Memory fetches. Port 0 serves as a time multiplexed address/data bus. When the ALE (Address Latch Enable) signal is high, Port 0 is used as the low byte of the Program Counter (PCL) as an address. Port 2 is used as the high byte of the Program Counter (PCH). Then PSEN strobes the EPROM and the code byte is read into the microcontroller.

7.3.2 Data Memory

The hardware configuration for accessing up to 2K bytes of external RAM is shown in Fig. 7.9. The MOVX

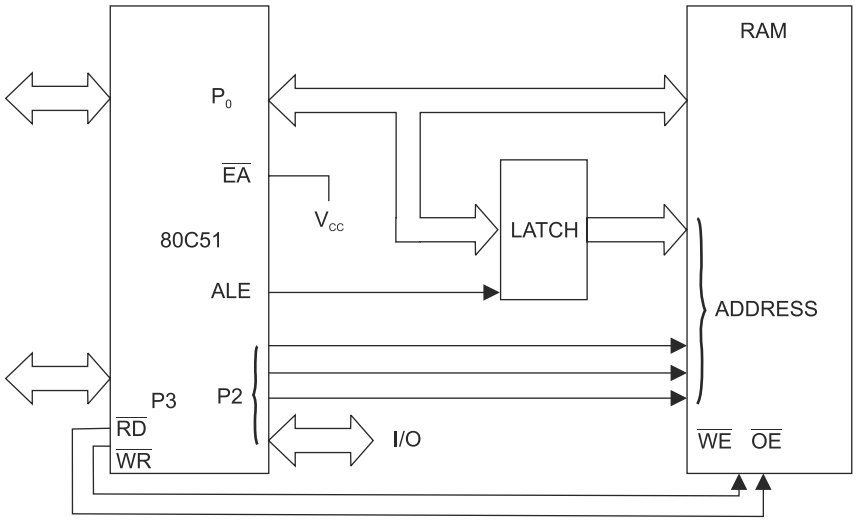


Fig. 7.9 Accessing external data memory

instruction is used to access the external data memory. The CPU in this case is executing from internal ROM. Port 0 acts as a time-multiplexed address/data bus to the RAM and 3 lines of Port 2 are being used to page the RAM. The CPU of microcontroller generates $\overline{RD}$ and $\overline{WR}$ signals as required during external RAM accesses. The CPU can be used to access up to 64K bytes of external Data Memory. The external data memory addresses can be either 1 or 2 bytes wide. One-byte addresses are frequently used in conjunction with one or more other I/O lines to page the RAM. Two-byte addresses can also be used, in which case the high address byte is emitted at Port 2.

The internal and external data memory spaces available to the 80C51 user are given in Fig. 7.10. The 80C51 microcontroller has 256 bytes of RAM on the chip. Among them, only the lower 128 bytes are used for internal data storage. The upper 128 bytes are used as the special function registers (SFR). The detail of the lower 128 bytes is illustrated in Fig. 7.11. The lower 128 bytes of RAM which can be accessed by both direct and indirect addressing can be divided into three segments as listed below:

Register Banks 0-3 The lowest 32-bytes (00H to 1FH) of the on-chip RAM occupied as 4 banks of 8 registers each as depicted in Fig. 7.12. The bank is selected by setting 2 bit in PSW. Only one bank is active

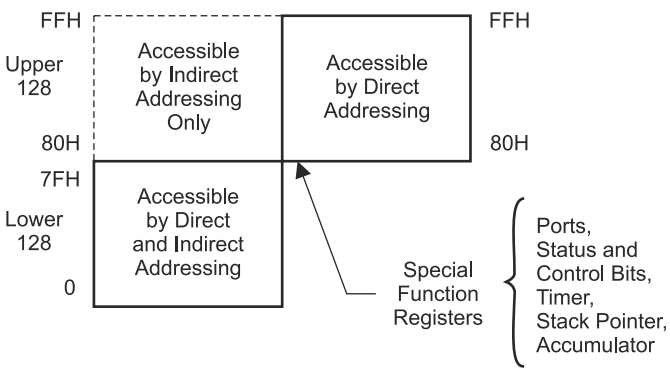


Fig. 7.10 Internal data memory

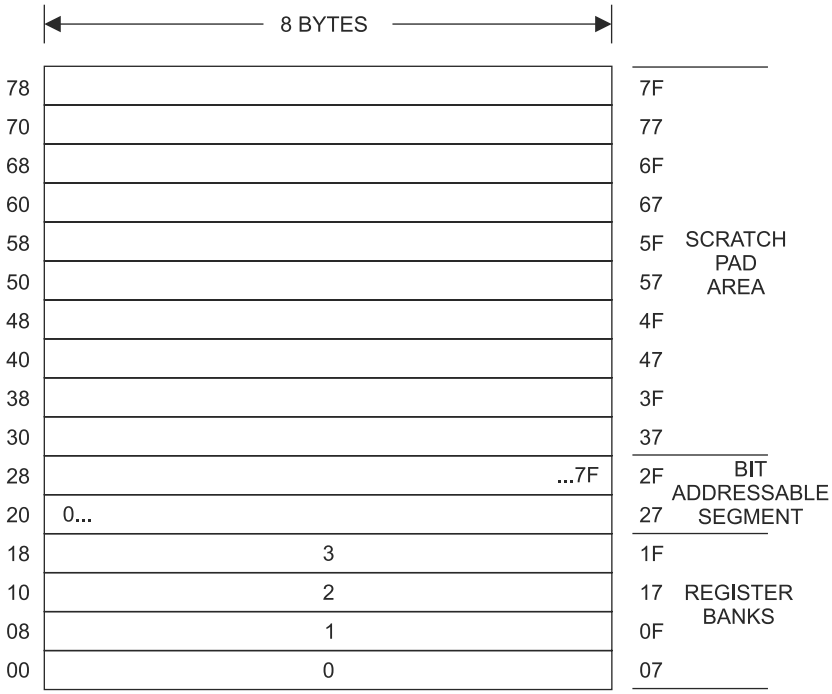


Fig. 7.11 128-bytes of RAM direct and indirect addressable

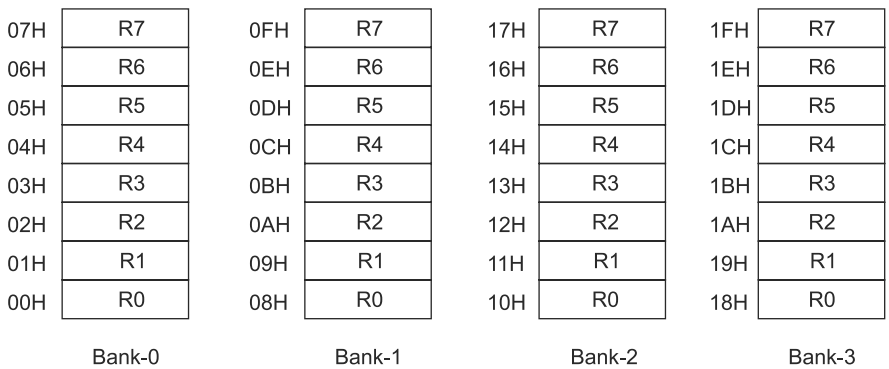


Fig. 7.12 Four banks of 8-registers R7–R0

at a time. By default Bank-0 is selected and its register addresses 00H to 07H. Each bank consists of eight 1-byte registers R_0 to R_7.

Reset initializes the stack pointer to location 07H and it is incremented once to start from location 08H, which is the first register (R_0) of the second register bank. Therefore, in order to use more than one register bank, the SP should be initialized to a different location of the RAM.

Bit Addressable Area The next 16-bytes contain 20H–2FH form a block that can be addressed as either bytes or individual bits. The bytes can be addressed 20H to 2FH. The bits can be addressed 00H to 7FH as depicted in Fig.7.13. For accessing the bits, specific instructions are used. Hence, bits 0–7 can also be referred

2F	7F	7E	7D	7C	7B	7A	79	78
2E	77	76	75	74	73	72	71	70
2D	6F	6E	6D	6C	6B	6A	69	68
2C	67	66	65	64	63	62	61	60
2B	5F	5E	5D	5C	5B	5A	59	58
2A	57	56	55	54	53	52	51	50
29	4F	4E	4D	4C	4B	4A	49	48
28	47	46	45	44	43	42	41	40
27	3F	3E	3D	3C	3B	3A	39	38
26	37	36	35	34	33	32	31	30
25	2F	2E	2D	2C	2B	2A	29	28
24	27	26	25	24	23	22	21	20
23	1F	1E	1D	1C	1B	1A	19	18
22	17	16	15	14	13	12	11	10
21	0F	0E	0D	0C	0B	0A	09	08
20	07	06	05	04	03	02	01	00

Fig. 7.13 Bit addressable RAM

to as bits 20.0–20.7 and bits 8-FH are the same as 21.0–21.7, and so on. Each of the 16-bytes in this segment can also be addressed as a byte.

Scratch Pad Area The memory locations 30H–7FH are general purpose RAM. On the other hand, if the stack pointer has been initialized to this area, enough bytes should be left aside to prevent SP data destruction.

The upper 128 bytes of the on-chip RAM are used for special function registers as shown in Fig. 7.14. Actually, only 25 of these bytes are used. The other bytes are reserved for advanced versions of the microcontroller. These bytes are associated with registers which are used for different functions and operation of microcontroller. Some of these registers are bit addressable and some of these bytes addressable.

7.4 PIN DIAGRAM OF 8051 MICROCONTROLLER

The 8051 microcontroller is available in DIP, QFP and LLC packages. This is a 40 pin IC. There are four 8-bit ports P_0, P_1, P_2 and P_4. Therefore, total 32 pins are covered for 4-ports. The remaining 8 pins are V_{CC} (power supply), V_{SS} (ground), crystal oscillator pins XTAL1 and XTAL2, RST (Reset), PSEN (Program Store Enable), ALE (Address Latch Enable), and EA (External Access) respectively. The Pin diagram of 8051 microcontroller is depicted in Fig. 7.15 and schematic pin diagram is illustrated in Fig.7.16. The brief discussions of all these pins are explained in this section.

F8								FF
F0	B							F7
E8								EF
E0	ACC							E7
D8								DF
D0	PSW							D7
C8								CF
C0								C7
B8	IP							BF
B0	P3							B7
A8	IE							AF
A0	P2							A7
98	SCON							9F
90	P1							97
88	TCON	TMOD	TL0	TL1	TH0	TH1		8F
80	P0	SP	DPL	DPH			PCON	87

Fig. 7.14 SFR memory map

V_{CC} This pin is connected to +5 V supply voltage. A 125-mA current is drawn from supply for 8051/8031 microcontroller and the maximum power dissipation rating is about 1W.

V_{SS} This is a ground pin for supply. All the voltages are specified with respect to this pin.

RST The RST input pin resets the 8051, only when RST pin is high for two or more machine cycles. There are two ways to reset 8051 microcontroller such as power-on reset and manual reset. When the microcontroller is reset, all values in the register to will be lost. So the reset values of PC, ACC, B, PSW and DPTR are 0000H and the content of SP is 0007H. The power-on reset circuit diagram is shown in Fig. 7.17. A pull-down resistance of 8.2K is connected between RST and ground terminals. A10 μF capacitance is also connected from V_{CC} to RST pin. These components provide delay about 24 clock cycles. Figure 7.18 shows the manual reset circuit. A push-button switch is added across the 10 μF capacitor.

ALE (Address Latch Enable) ALE is used for demultiplexing the address and data bus when 8051 microcontroller is interfacing with external memory. Port 0 provides the low-byte of address bus A_7 to A_0 and data bus D_7 to D_0. When ALE = 1, port 0 is used as address bus A_7 to A_0. If ALE = 0, port 0 is used as the data bus. Usually, ALE is activated periodically with a constant rate of one-sixth of the oscillator frequency.

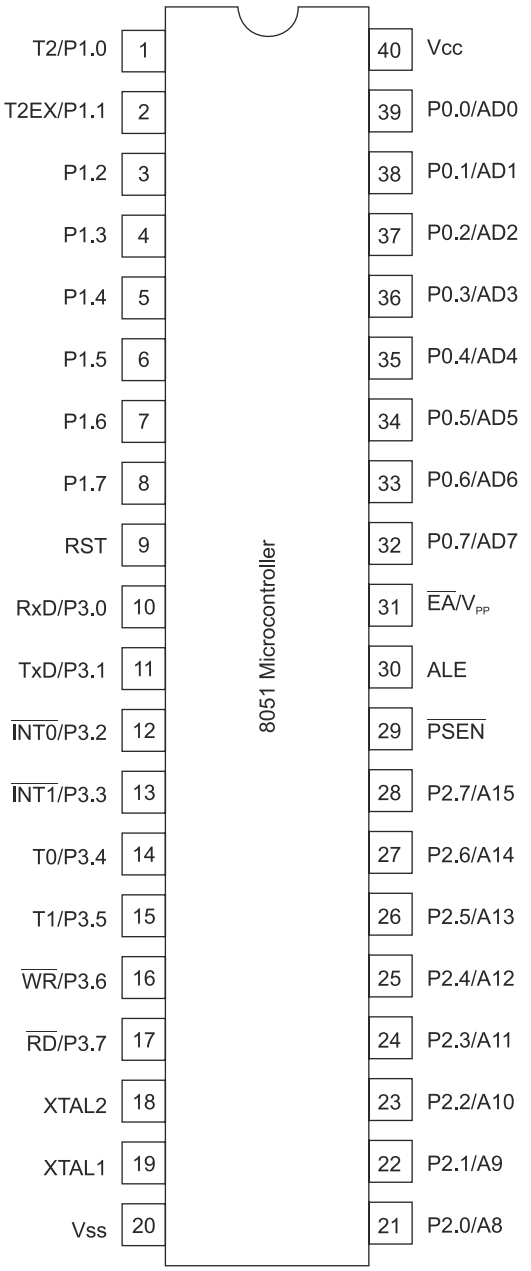


Fig. 7.15 Pin diagram of 8051 Microcontroller

***EA* (External Access)** This pin is connected to either V_{CC} or ground. When this pin is connected to V_{CC} or high, 8051 is to execute programs from internal program memory till address 0FFFH. If EA is low, microcontroller can execute from external memory only.

***PSEN* (Program Store Enable)** This pin is an active-low output control signal. This is used as a read signal for reading data from the external program memory. This pin is activated after every six-clock cycles during fetching the data from external program memory. The *PSEN* pin is remaining high during execution a program from internal ROM.

***XTAL*$_1$ and *XTAL*$_2$** These are the oscillator pins to connect the crystal oscillators of nominal frequency 12 MHz or 11.059 MHz. $XTAL_1$ is input to the inverting oscillator amplifier and input to the internal clock generating circuits. $XTAL_2$ is output from the inverting oscillator amplifier. Usually the quartz crystal oscillator is connected to $XTAL_1$ and $XTAL_2$ and it also needs two capacitors of 30-pf values. One side of each capacitor is connected to the ground as shown in Fig. 7.19.

When the 8051 microcontroller is connected to a crystal oscillator and power supply is given, we can observe the frequency on the $XTAL_2$ pin. The 8051 microcontroller's operation is synchronising with crystal oscillator output signal. Effectively, the 8051 operates based on machine cycles. A machine cycle is the minimum amount of time in which a single 8051 instruction can be executed, but some instructions take multiple machine cycles. In 8051, a machine cycle consists of a sequence of 6 states numbered S_1 through S_6 (twelve clock cycles) as shown in Fig. 7.20. Each state time lasts for two oscillator periods. Machine cycle is also called as instruction cycle. Each instruction cycle has six states (S_1–S_6) and each state has two pulses (P_1 and P_2). Hence a machine cycle takes 12 oscillator periods or 1μs if the oscillator frequency is 12 MHz.

Port 0 (P0.0–P0.7) Port 0 consists of 8-bit bidirectional input/output port pins. These are bit addressable. This port has been given an address in the SFR address range. Port 0 acts as multiplexed low order address/data bus AD_7–AD_0. ALE is used for demultiplexing address and data bus. When ALE is 0, it provides data D_7–D_0, but when ALE is 1, it is used as address A_7–A_0. The Port 0 pins are open drain I/O. To use the pins of port 0, each pin must be connected externally to a 10 K ohm pull-up resistor.

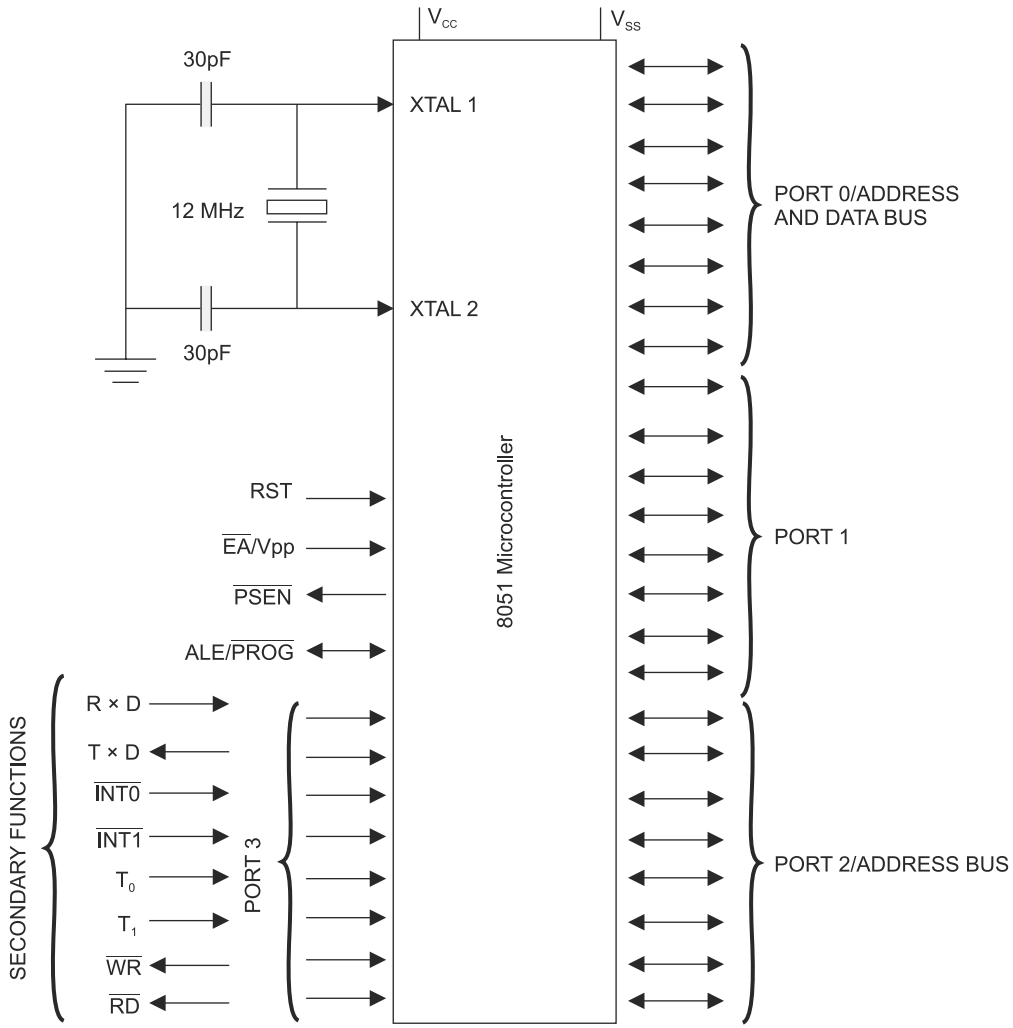


Fig. 7.16 Schematic pin diagram of 8051 Microcontroller

Port 1 (P1.0–P1.7) Port 1 pins can be used as either input or output. This port is an 8-bit quasi-bidirectional bit addressable port and Port 1 pins are internally pulled high with fixed pull-up resistors. Hence, this port does not need any pull-up resistor as it already has internal pull-up resistance. This port has been given an address in the SFR address range. User should configure it either as input or output port. This port acts as input port when write 1 to all its 8 bits. This port acts as output port when write 0 to all its 8 bits. Therefore, port 1 pins have no dual functions.

Port 2 (P2.0–P2.7) Port 2 is also an 8 quasi-bidirectional bit addressable I/O port and port pins are pulled high internally. It has also been given an address in the SFR address range. Port 2 generates higher eight bits of address (A_{15}–A_8) during external program and data memory accesses, if ALE is high and $\overline{EA}$ is low. Port 2 receives higher order address bits during programming the internal ROM of 8051 microcontroller.

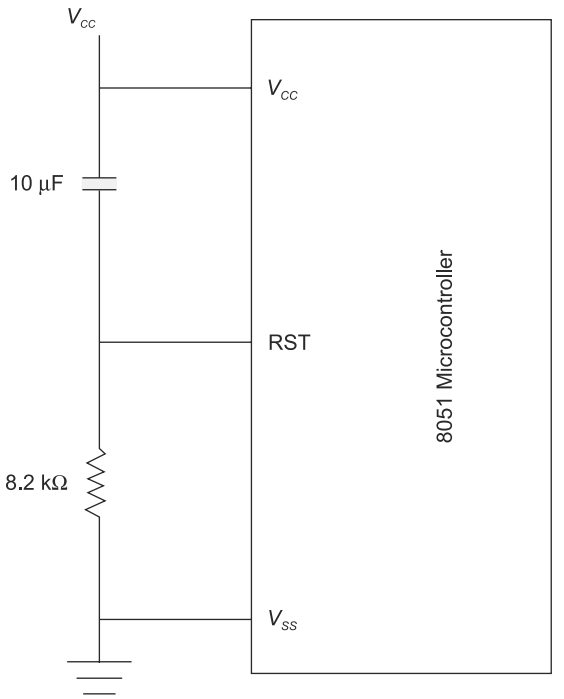


Fig. 7.17 Power on reset

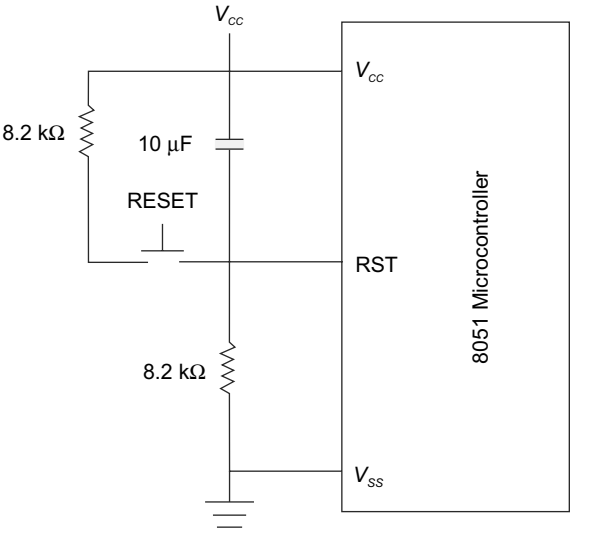


Fig. 7.18 Power on manual reset

Port 3 (P3.0–P3.7) Port 3 is also an 8-bit bi-directional bit-addressable I/O port with internal pull-up resistances. This port has been given an address in the SFR address range. There are other functions multiplexed with Port 3 pins as given in Table 7.7.

Table 7.7 Alternative functions of pins P3.0 to P3.7

Port 3	Alternative function
P3.0	Acts as serial input data pin (R×D)
P3.1	Acts as serial output data pin (T×D)
P3.2	Acts as external interrupt pin 0 ($\overline{INT_0}$)
P3.3	Acts as external interrupt input pin 1 ($\overline{INT_1}$)
P3.4	Acts as external input to timer 0 (T_0)
P3.5	Acts as external input to timer 1 (T_1)
P3.6	Acts as write control signal for external data memory ($\overline{WR}$)
P3.7	Acts as read control signal for external data memory read operation ($\overline{RD}$)

The port structures of 8051 microcontrollers are depicted in Fig. 7.21 and Fig. 7.22. Each port consists of a latch, an input buffer and an output driver. The D-flip-flop is used as bit latch and it clocks from internal data bus in response to write to latch from internal CPU bus. The output of flip-flop can be read onto the internal data bus in response to a Read Latch signal from the internal CPU bus. The operation of Read Pin is different from Read Latch. The port pin can be read onto the internal data bus whenever CPU sends a read-pin command.

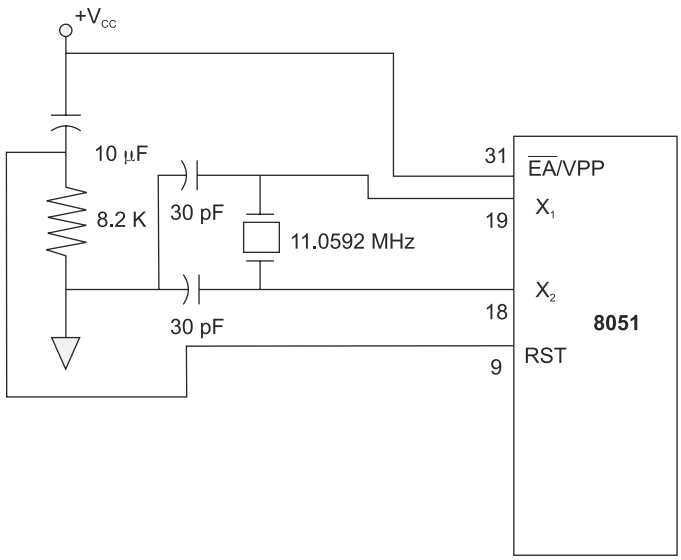


Fig. 7.19 Oscillator circuit

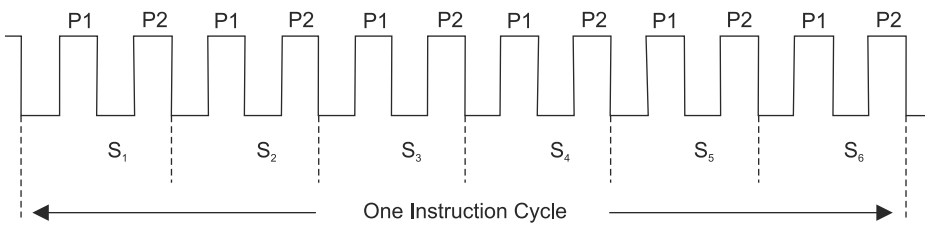


Fig. 7.20 Instruction cycle of 8051 microcontroller

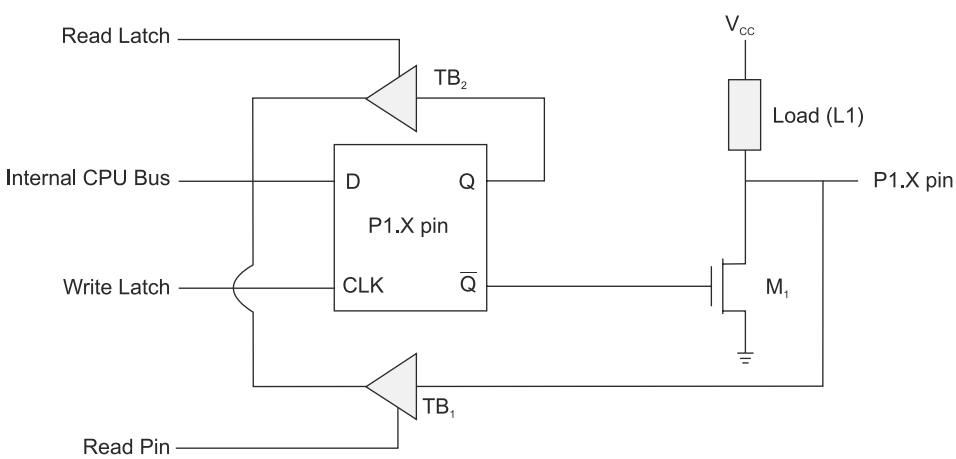


Fig. 7.21 A pin of Port 1

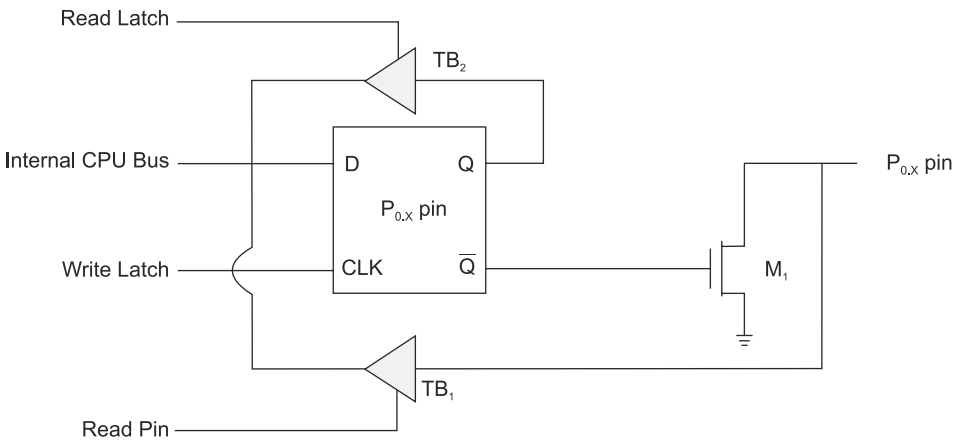


Fig. 7.22 A pin of Port 0

Port 1, 2 and 3 are bi-directional ports with fixed internal pull-up resistors. When a port pin is used as input, 1 must be written to a port latch. The $Q = 1, \overline{Q} = 0,$ FET is OFF and the pin is simply pulled high by the pull-up resistor. Thereafter the pin status can be read onto the internal data bus. Writing '1' to output pin P1.X of Port 1 is shown in Fig. 7.23.

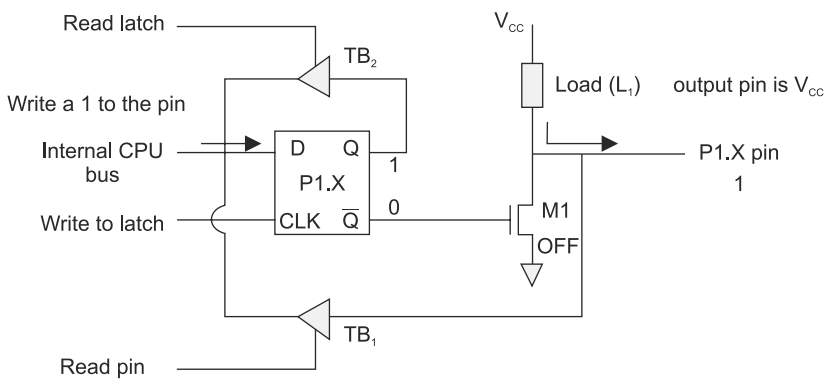


Fig. 7.23 Writing '1' to output pin P1.X of Port 1

The port pin can be used as output while write 0 onto the pin. Then $Q = 0, \overline{Q} = 1$ and FET is ON. The port pins can sink more current than its source current. Sinking current is about 0.5 mA but source current is in the order of tens μA only. Figure 7.24 shows writing '0' to output pin P1.X of Port 1. Reading '1' and '0' at input pin P1.X of Port 1 using MOV A,P1 are depicted in Fig. 7.25 and Fig. 7.26 respectively.

Port 0 is a bi-directional open drain I/O without internal pull-up resistors. When this port is configured as an input, it floats as depicted in Fig. 7.27. If '1' is written to a port 0 latch, FET is OFF, the pin floats and can be used as high-impedance input. To get output of port 0 pins external pull up resistances are connected between Port 0 pins and $+V_{cc}$ as shown in Fig. 7.28.

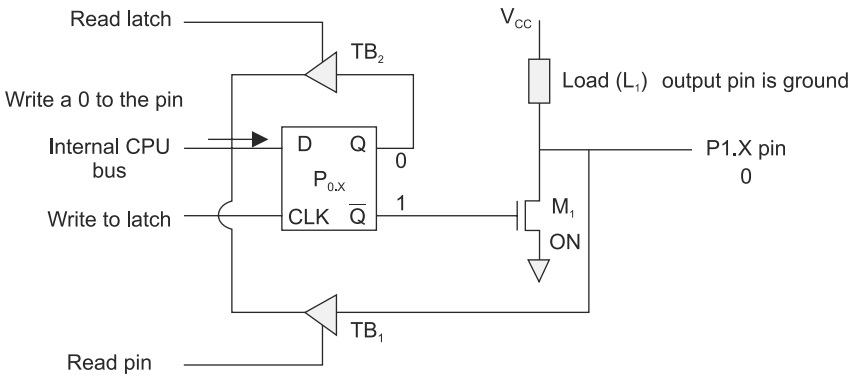


Fig. 7.24 Writing '0' to output pin P1.X of Port 1

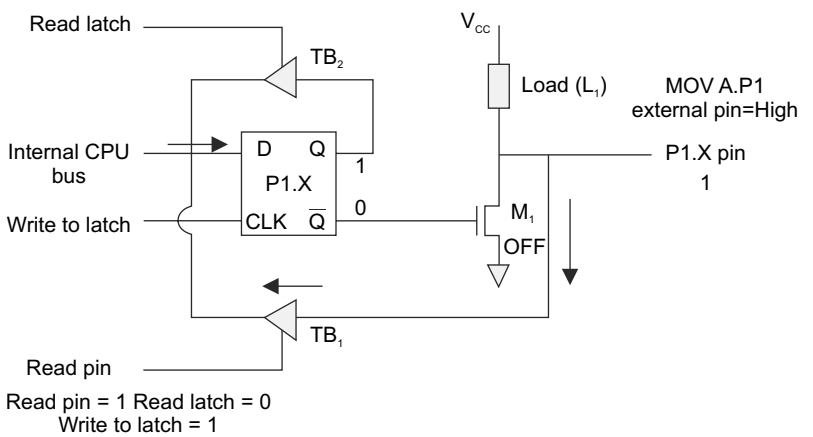


Fig. 7.25 Reading '1' at input pin P1.X of Port 1 using MOV A, P_1

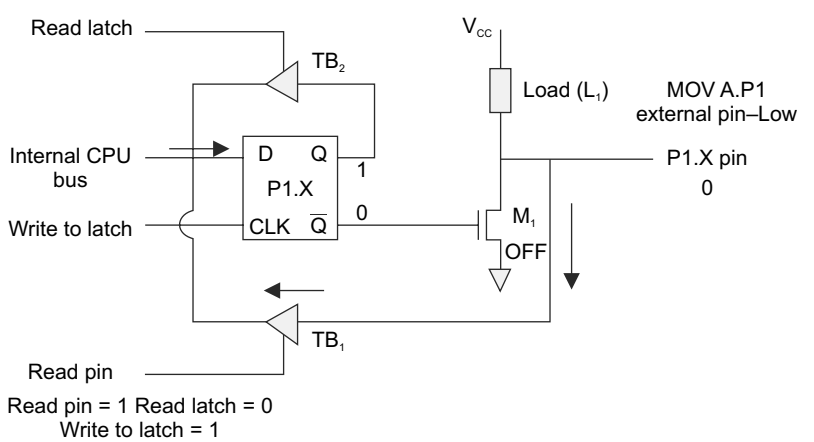


Fig. 7.26 Reading '0' at input pin P1.X of Port 1 using MOV A, P_1

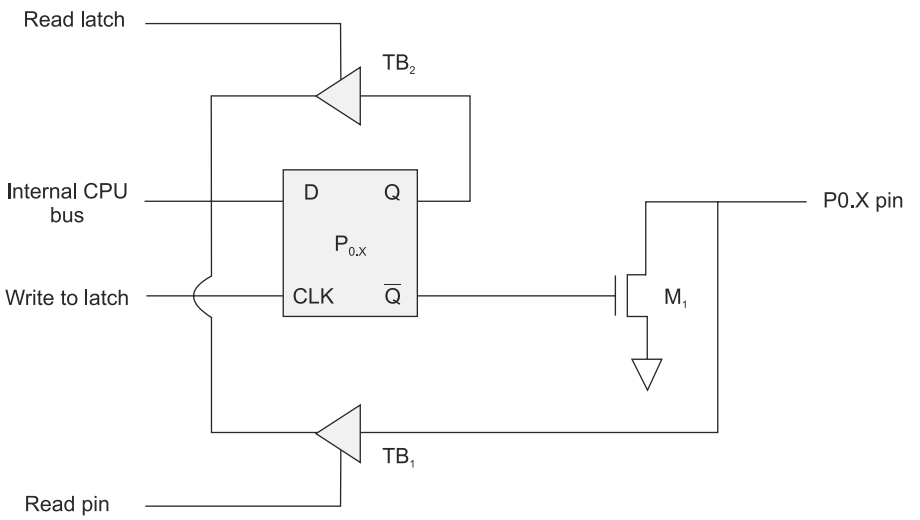


Fig. 7.27 A pin of Port 0

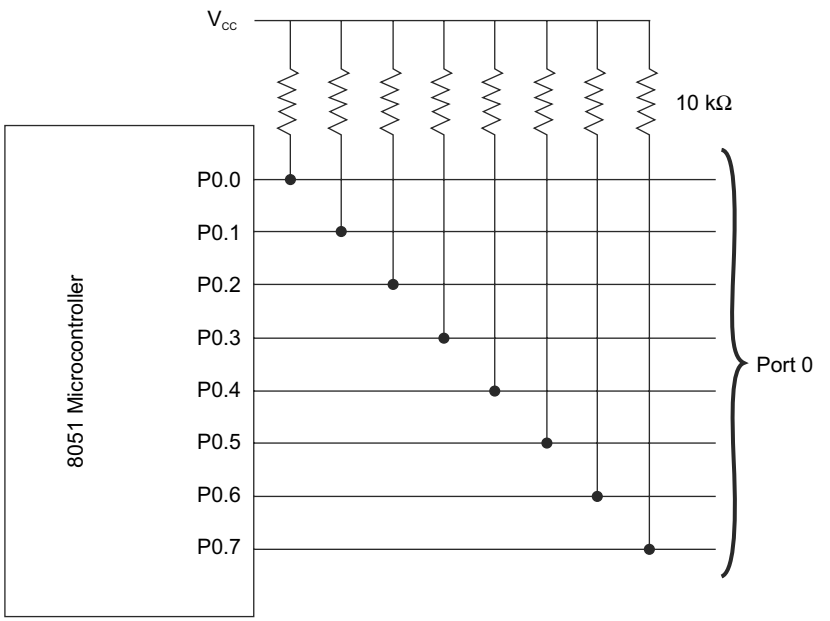


Fig. 7.28 Port 0 with pull-up resistors

7.5 POWER MANAGEMENT

In stop-clock mode, the static design of microcontroller enables the clock speed to be reduced down to 0 MHz. Since the oscillator is stopped, the RAM and Special Function Registers retain their values. Actually, this mode allows step-by-step utilization and permits reduced system power consumption by reducing the clock frequency down to any value. For lowest power consumption, the Power Down mode is used.

The 8051 microcontroller has two power-saving modes, namely idle mode and power-down mode. These operating modes are activated by software via the PCON (power control) special function registers as shown in Fig.7.29.

D7	D6	D5	D4	D3	D2	D1	D0
SMOD	–	–	–	GF1	GF0	PD	IDL

Bit	Name	Bit Address	Function
0	IDL	PCON.0	Idle-mode bit
1	PD	PCON.1	Power-down bit
2	GF0	PCON.2	General purpose Flag bit 0
3	GF1	PCON.3	General purpose Flag bit 1
4		PCON.4	
5		PCON.5	
6		PCON.6	
7	SMOD	PCON.7	Serial baud rate modify bit

Fig. 7.29 PCON register

7.5.1 Idle Mode

In the idle mode, the CPU puts itself to sleep but all the on-chip peripherals stay active. The idle mode is activated by the IDL bit. When the IDL bit is high, the microcontroller operates in idle mode. Actually, an instruction sets the PCON.0 bit. The instruction to invoke the idle mode is the last instruction executed in the normal operating mode before the idle mode is activated. During this mode, all program execution stops, the CPU contents, the on-chip

RAM contents are preserved and all of the special function registers remain intact. The oscillator continues to run but it is blocked from the CPU. The timers and UART continue to operate normally. The idle mode can be terminated either by any enabled interrupt or by a hardware reset which starts the processor in the same manner as a power-on reset. Due to activation of any enabled interrupt, the PCON.0 bit will be cleared, idle mode will be terminated and microcontroller executes the interrupt service routine. After completion of the interrupt service routine, execution will continue from the instruction following the instruction which set the idle bit.

7.5.2 Power-Down Mode

To save even more Power, a Power-down mode is used in the microcontroller. The Power-Down mode is activated by setting the PDWN bit high. This operating mode can be achieved by software. In this mode, actually the oscillator is stopped and the instruction that invoked Power Down is the last instruction executed. The timers and the UART and software execution are halted. During this mode, the on-chip RAM and Special Function Registers retain their values as long as a minimum 2.0 V is applied to the microcontroller chip.

Either a hardware reset or external interrupt can be used to exit from Power-Down mode of the microcontroller. Reset redefines all the SFRs but does not change the on-chip RAM. External interrupts allow both the SFRs and the on-chip RAM to retain their values.

To terminate Power-Down mode, the Reset or external interrupt should not be executed before V_{CC} is restored to its normal operating level and must be held active long enough (less than 10 ms) for the oscillator to restart and stabilize.

7.6 TIMERS/COUNTERS

The 8051 microcontroller has two 16-bit timers/counters such as *TIMER0* (T0) and *TIMER1* (T1). Each timer can be programmed to count internal clock pulses of 8051 microcontroller. These timers are used for the following functions:

- Calculate time delay between two events
- Counting the number of events
- Generate baud rate for serial ports
- Frequency measurement
- Pulse width measurement

Generally a timer is used to count machine cycles and provides a specified time delay. Actually a machine cycle consists of 12 oscillator periods. Hence the counting rate is about

$$\frac{\text{Oscillator frequency}}{12}.$$

When the oscillator frequency is 12 MHz, the time period of one clock cycle is 1μs. The counter of the 8051 microcontroller is incremented in response to a transition from 1 to 0 at external pin, either T0 or T1. The counter output is a count value which represents the occurrence of 1 to 0 transitions at the external pin. Usually, counters are used as up counter. Figure 7.30 shows a 3-bit counter which counts from 0 to 7 and the overflow flag is set after counting 7. The 3-bit counter is not used in the 8051 microcontroller, but 8-bit and

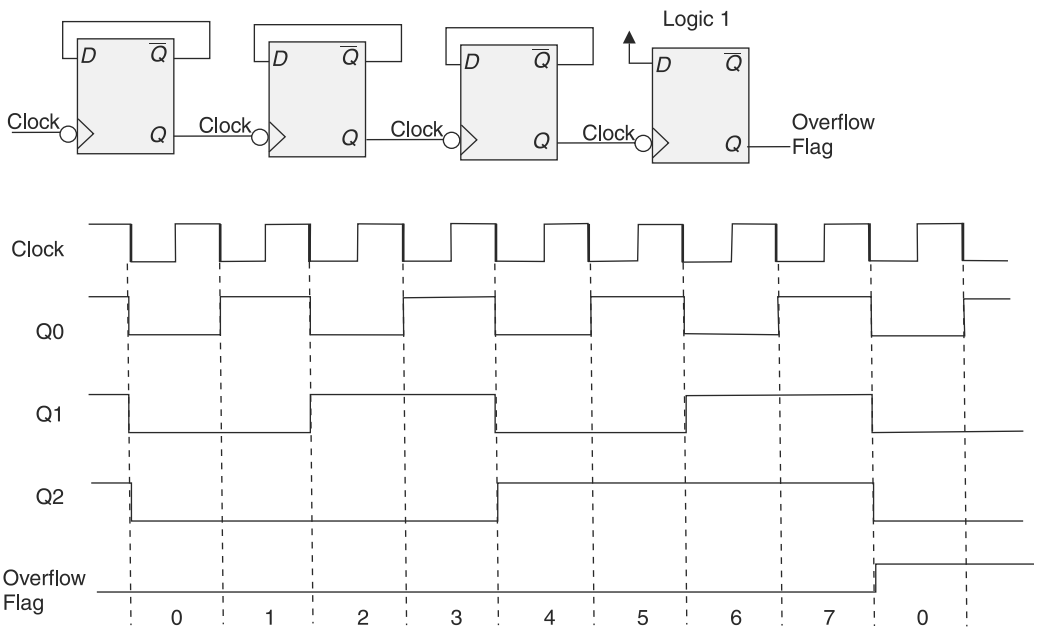


Fig. 7.30 3-bit counter circuit

16-bit counters are commonly used. When the 16-bit counter overflows from 0000H to FFFFH, it can set a flag and generates an interrupt.

The 16 bits of the timer consists of higher byte THx and the lower byte TLx, where x may be either 0 or 1. For TIMER1, TH1 is the higher byte of timer 1 and TL1 is the lower byte of timer 1 as shown in Fig. 7.31(a). Similarly, TH0 is the higher byte of timer 0 and TL0 is the lower byte of timer 0 as depicted in Fig. 7.31(b).

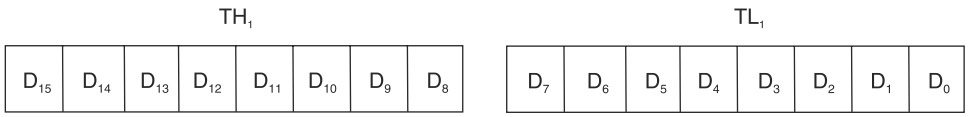


Fig. 7.31(a) Timer 1 registers

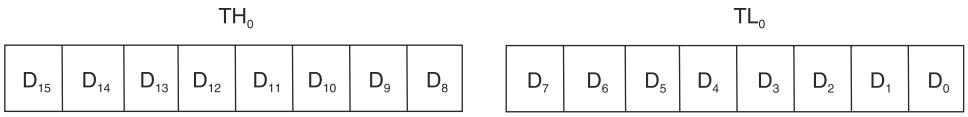


Fig. 7.31(b) Timer 0 registers

7.6.1 Operating Modes

The timer can be operating in four different modes, namely mode 0, mode 1, mode 2 and mode 3. The mode bits M1 and M0 in the TMOD register are used to select any one of the operating modes as given below:

M_1	M_0	*Operating Modes*	*Functions*
0	0	Mode 0	13-bit timer mode
0	1	Mode 1	16-bit timer mode
1	0	Mode 3	8-bit timer mode
1	1	Mode 4	Split timer mode

Timer Mode 0 In this mode, timer operates as a 13-bit timer. THx register is used as an 8-bit counter and TLx can be used as a 5-bit counter as shown in Fig. 7.32. The count value varies from 0000H to 1FFFH. Whenever the timer reaches its maximum value 1FFFH, it returns to 0000H and the overflow flag TF is set. The timer clock frequency is oscillator frequency/12. The clock frequency input to THx is

$$\frac{\text{Oscillator frequency}}{12 \times 2^5} = \frac{\text{Oscillator frequency}}{12 \times 32}$$

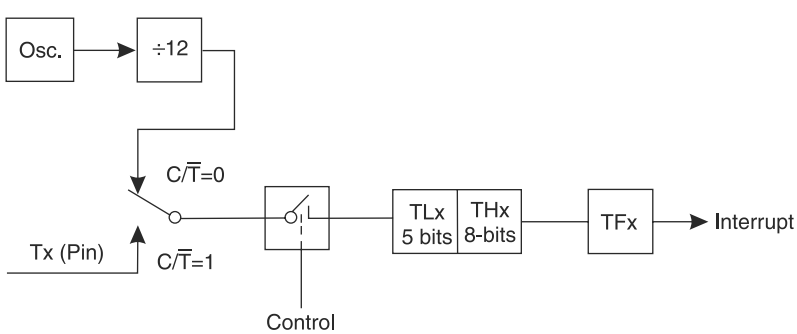


Fig. 7.32 Timer Mode 0 (13-bit timer)

When oscillator frequency is 12MHz, the clock frequency input to THx is = $\frac{12\text{ MHz}}{12 \times 32}$.

The overflow flag is set to zero after 32 × 256 = 8192 machine cycles.

Timer Mode 1 In timer mode-1, timer operates as a 16-bit timer where THx register is used as an 8-bit counter and TLx is used an 8-bit counter. The timer higher byte THx is connected in cascade with the timer lower byte TLx as shown in Fig. 7.33 where the timer counts from 0000H to FFFFH. TLx is incremented from 00H to FFH. After counting FFH, the timer resets to 0 and THx is incremented by 1. As TLx and THx operate as 16-bit counter, it can count up to 65536D. The overflow occurs after FFFFH and timer overflow flag is set. After overflow, the counter reset at 0000H when the timer starts counting from a initial value, the time delay will be

$$\frac{12 \times (65,536 - \text{Initial Value})}{\text{Frequency}}$$

where, initial value is equal to TLx + THx × 256

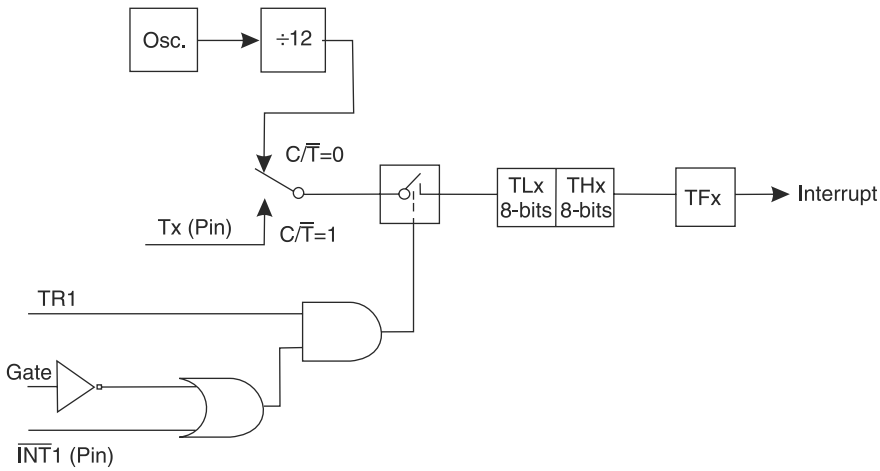


Fig. 7.33 Timer Mode 1 (16-bit timer)

Timer Mode 2 In mode 2, timer acts as an 8 bit timer, any values from 00H to FFH to be loaded into the timer's register THx. Initially THx will be loaded with the 8-bit value, after that the microcontroller copies the content of THx into TLx. Then timer starts counting. In this mode, time provides an auto-reload feature. TLx starts counting up, when TLx reaches FFH, subsequently it is incremented instead of resetting to 0, the

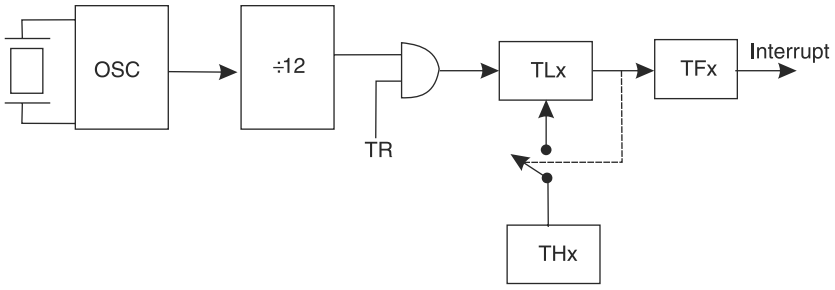


Fig. 7.34 Timer Mode 2 (8 bit timer) as auto-reload timer

TLx must be reset to the value which is stored in THx. Therefore, in timer mode 2, just after overflows of TLx, it is reloaded with the value, i.e. the content of THx as depicted in Fig.7.34. Hence, the time delay between overflows is about

$$\frac{12 \times (256 - \text{THx})}{\text{Frequency}}$$

Timer Mode 3 In this mode, the Timer 0 divides into two 8-bit counter/timers TL0 and TH0. TH0 and TL0 are two separate timers with overflow flags TF1 and TF0 respectively as shown in Fig.7.35. The first counter TL0 acts like mode 0 without pre-scalar. The second counter TH0 counts CPU cycles, uses TR1 (timer 1 run bit) as enable, uses TF1 (timer 1 overflow bit) as flag and uses timer 1 interrupt. When the timer 1 is in mode 3, Timer 1 works as counter stopped if it is in mode 3. Timer 1 operates in mode 0, 1, or 2 and it has gate (INT1) and external input (T1) but no flag or interrupt. Timer 1 can also be used as baud rate generator.

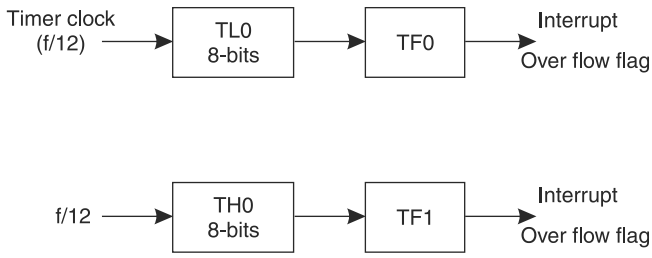


Fig. 7.35 Timer Mode 3

Counter The timers T0 and T1 can be used as counters. The difference between the counter and timer is the source of the clock pulses to the counters. While it is used as a timer, the oscillator output pulse can be used as source of clock pulses after divide by 12. When it is used as a counter, pin T0 (P3.4) provide pulses to counter 0 and pin T1(P3.5) supplies pulses to counter 1. The $C/\overline{T}$ bit in TMOD must be set to 1 to enable pulses from the Tx pin to reach the control circuit. Figure 7.36 shows the Timer/counter logic.

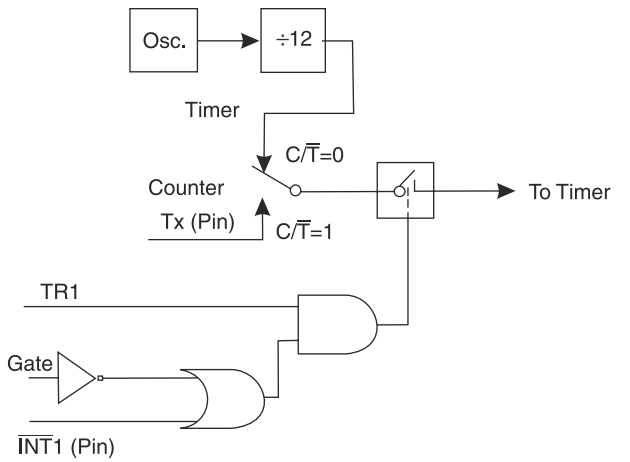


Fig. 7.36 Timer/counter logic

7.6.2 Control Registers

The timer/counter operation can be controlled by Timer Mode Control (TMOD) register and Timer/Counter Control (TCON) Register. In this section, the function of TMOD and TCON are explained briefly.

TCON Register The timer/counter control (TCON) register consists of control bits and flags for timers in the upper nibble and control bits and flags for the external interrupt in the lower nibble as depicted in Fig. 7.37.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Fig. 7.37 Timer/Counter Control (TCON) register

TF1 Timer 1 overflows flag. It is set by the hardware when timer/counter 1 overflows. It is cleared by hardware as processor vectors to the interrupt service routine.

TR1 Timer 1 runs control bit. This is set to 1 by software program to enable timer 1 to count. It is cleared to 0 by program to halt timer.

TF0 Timer 0 overflows. It is set by the hardware when timer/counter 0 overflows. It is cleared when the processor vectors to execute interrupt service routine.

TR0 Timer 0 runs control bit. It is set to 1 by program software to enable timer 0 to count. It is cleared to 0 by software to halt timer.

IE1 External Interrupt 1 edge flag. This is set to 1 when a high to low (the falling) edge signal is received on port pin P3.3 (INT1). It is cleared when processor vectors to ISR.

IT1 Interrupt 1 type control bit. This bit is set to 1 by program to enable external interrupt 1 to be triggered by a falling edge signal. It is set to 0 by program to enable a low level signal on external interrupt 1 to generate an interrupt.

IE0 External Interrupt 0 edge flag. It is set to 1 by program to enable interrupt 0 to be triggered by a high to low (falling edge) signal. This is set to 0 by program to enable a low-level signal on external interrupt 0 to generate an interrupt.

TMOD Register The time node control (TMOD) register is used to set the various timer operating modes. Actually TMOD is related with the two times and can be considered to be two duplicate 4-bit registers as shown in Fig. 7.38.

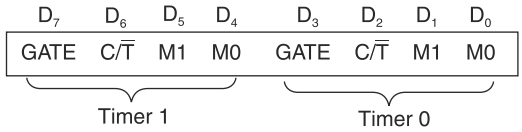


Fig. 7.38 Timer mode control (TMOD) register

Gate If TRx of TCON is set and GATE=1, timer/counter x will operate only when INTx pin is high for hardware control. When GATE=0, timer/counter x will run only if TRx =1 for software control.

C/$\overline{T}$ (Clock/Timer) The C/$\overline{T}$ bit in TMOD register is used to take decision whether the timer is used as a timer or an counter. If the timer or counter selector bit is cleared (C/$\overline{T}$=0), it is used as a timer to generate time delay. When C/$\overline{T}$ =1, it is used as a counter by counting pulses from external input pin Tx (T1 and T0).

M1 Timer/counter operating mode selector bit. This bit is set or cleared by program to select mode.

M0 Timer/counter operating mode selector bit. This bit is set or cleared by program to select mode.

Table 7.8 Timer operating modes

M1	*M0*	*Operating Modes*	*Functions*
0	0	Mode 0	13-bit timer mode. THx as 8-bit timer/counter and TLx as 5 bit timer/counter (prescalar).
0	1	Mode 1	16-bit timer mode. THx and TLx are cascaded and there is no prescalar.
1	0	Mode 3	8-bit Auto reload timer/counter mode. THx hold a count value which is to be reloaded into TLx after each overflows.
1	1	Mode 4	Split timer mode. Timer 0 is used as two 8-bit timers. Timer 1 stopped counting and timing function is allowed. Timer 1 can be used as baud rate generator.

7.7 INTERRUPTS

An interrupt is the occurrence of internal and external events that interrupts the micro-controller to provide service any device. In case of external events, the status of microprocessor pin is altered. In internal events, interrupts are generated due to timer overflow or transmission/reception of a byte through the serial port. After receiving an interrupt signal, the microcontroller interrupts the execution of main program. After saving the current status, the microprocessor jump to the memory location specified by interrupt and executes a subprogram called interrupt service routine (ISR). This memory location is called vector. Hence the interrupt is known as vector interrupt. After provide service to the interrupt, microprocessor restores the original status and continue to execute main program again.

7.7.1 Interrupts in 8051

There are five interrupt sources for the 8051 microcontroller. The prioritywise five different interrupts of 8051 microcontroller are given below:

- External Interrupt 0
- Timer 0
- External Interrupt 1
- Timer 1
- Serial Port

These interrupts can recognize 5 different events that can interrupt regular program execution.

- Each interrupt can be enabled separately.
- Each interrupt type has a separate vector address.
- Each interrupt type can be programmed to one of two priority levels.
- External interrupts can be programmed for edge or level sensitivity.
- Each interrupt can be enabled or disabled by setting bits of the IE (interrupt enable) register. Likewise, the whole interrupt system can be disabled by clearing the EA bit of the same register as shown in Fig. 7.39.

In 8051 microcontroller, interrupts are generated by internal operations such as Timer flag 0 (TF0), Timer flag 1 (TF1), and serial port interrupt (RI or TI). When the timer/counter 0 overflows, the TF0 flag is set to 1. If the timer/counter 1 overflows, the TF1 flag is set to 1. The vector address of TF0 and TF1 are 000BH and 001BH respectively. The TF0 and TF1 flag will be cleared when the timer flag interrupt makes a program call from the timer subroutine. In serial port interrupt, a data byte will be received if RI=1 and a data byte will be transmitted if TI=1. The vector address of RI or TI is 0023H. Whenever RI or TI becomes 1, the 8051 microcontroller is interrupted and jumps to the memory location 0023H to execute the Interrupt Service Routine (ISR).

Interrupts are also generated by external signals INT0 and INT1. The INT0 and INT1 are located on pins P3.2 and P3.3 respectively. External inputs on INT0 and INT1 pins set the interrupt flags IE0 and IE1 in the TCON register to 1 by level triggered or edge triggered. If the IT0 and IT1 bits of the TCON register are set, an interrupt will be generated on high to low transition, i.e. on the falling pulse edge. If these bits are cleared, an interrupt will be continuously executed as far as the pins are held low. The vector address of external interrupt 0 and external interrupt 1 are 0003H and 0013H respectively as shown in Table 7.9.

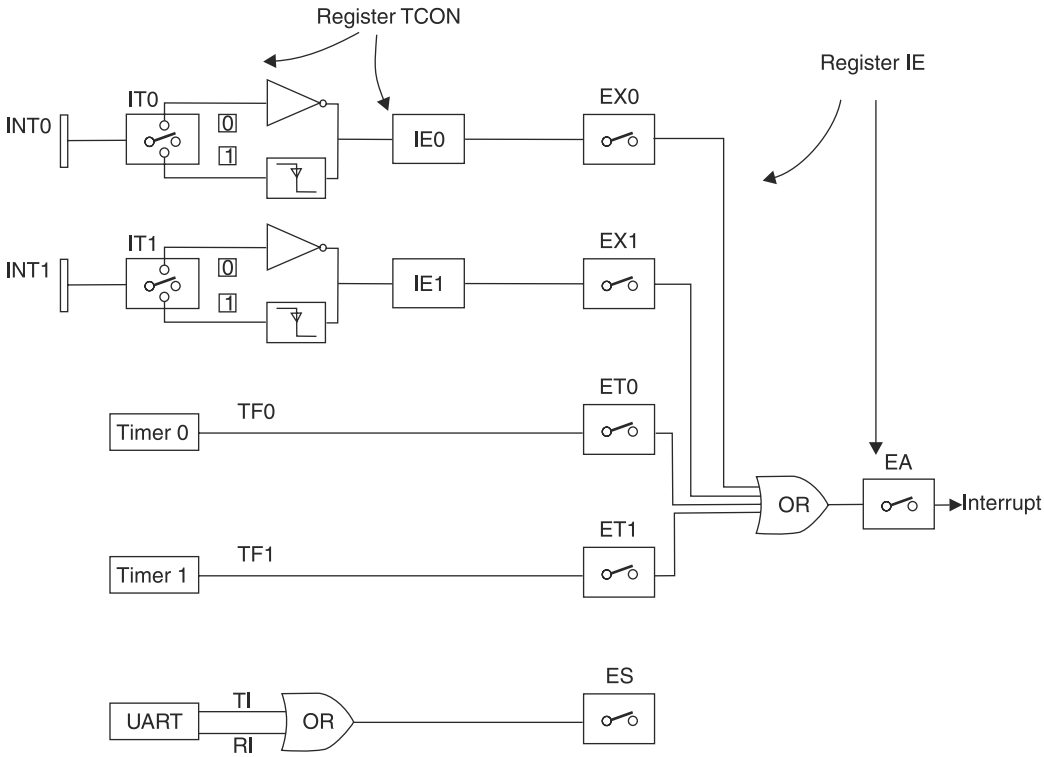


Fig. 7.39 Interrupts of 8051

Table 7.9 Interrupt vector addresses

Interrupt Source	Flag	Vector Address
External Interrupt 0	IE0	0003H
Timer 0	TF0	000BH
External Interrupt 1	IE1	0013H
Timer 1	TF1	001BH
Serial Port	RI&TI	0023H

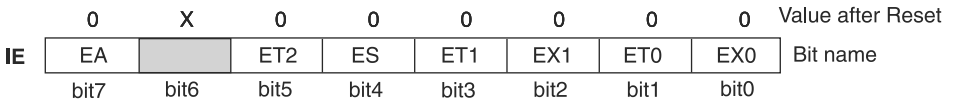
7.7.2 Interrupt Control Register

All interrupt operations are controlled by software. The programmer should program the control bits in following registers

- Interrupt Enable (IE) Register
- Interrupt Priority (IP) register and
- Timer Control Register (TCON)

In this section IE and IP registers are explained.

Interrupt Enable (IE) Register IE is Interrupt Enable Register which is shown in Fig. 7.40. The function of EA, ES, ET1, EX1, ET0 and EX0 are given below:



Bit	*Name*	*Bit Address*	*Function*
7	EA	AFH	Global interrupt enable/disable
6	-	AH	Undefined
5	-	ADH	Undefined
4	ES	ACH	Enable serial interrupt
3	ET1	ABH	Enable Timer 1 interrupt
2	EX1	AAH	Enable External 1 interrupt
1	ET0	A9H	Enable Timer 0 interrupt
0	EX0	A8H	Enable External 0 interrupt

Fig. 7.40 Interrupt Enable (IE) Register

EA global interrupt enable/disable:
0—disables all interrupt requests.
1—enables all individual interrupt requests.

ES enables or disables serial interrupt:
0—UART system cannot generate an interrupt.
1—UART system enables an interrupt.

ET1 bit enables or disables Timer 1 interrupt:
0—Timer 1 cannot generate an interrupt.
1—Timer 1 enables an interrupt.

EX1 bit enables or disables external 1 interrupt:
0—change of the pin INT0 logic state cannot generate an interrupt.
1—enables an external interrupt on the pin INT0 state change.

ET0 bit enables or disables timer 0 interrupt:
0—Timer 0 cannot generate an interrupt.
1—enables timer 0 interrupt.

EX0 bit enables or disables external 0 interrupt:
0—change of the INT1 pin logic state cannot generate an interrupt.
1—enables an external interrupt on the pin INT1 state change and the interrupt vector addresses are given in Table 7.9

Interrupt Priority (IP) Register The Interrupt Priority (IP) Register is used to determine the interrupt priority. Figure 7.41 shows the bit addressable IP register. When the bit is 0, the corresponding interrupt has lowest priority and if the bit is 1, the corresponding interrupt has the higher priority. When two interrupts occur at the same time, the higher priority interrupt gets service fast and then the next higher priority

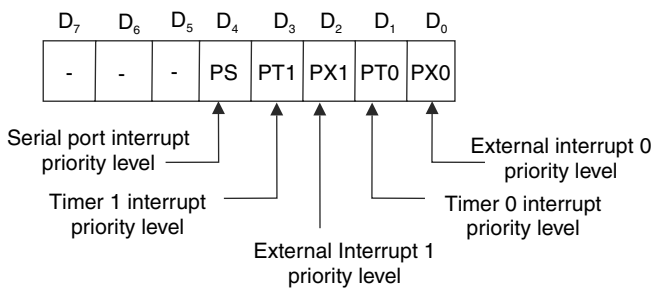


Fig. 7.41 IP register

interrupt gets service. The priority of interrupts is given below:

- IE0 (External Interrupt 0)
- TF0 (Timer Flag 0)
- IE1 (External Interrupt 1)
- TF1 (Timer Flag 1)
- RI/TI (Serial Port)

7.7.3 Execution of Interrupt

Assume that the microcontroller is executing the main program and the external interrupt INT1 occurs. The 8051 microcontroller completes the execution of current instruction and save the address of the next instruction, i.e. the content of program counter (PC) to the stack. The current status of all the interrupts, i.e. the content of IE register is also saved to the stack. The IE1 flag is disabled so that another INT1 interrupt will be inhibited.

Then the program counter is loaded with the vector location 0013H which is the predefined address of INT1. Therefore the program execution has been transferred to the memory location 0013H. A LJMP instruction is programmed at the memory location. Consequently, the program execution jump to the specified starting address of Interrupt Service Routine (ISR).

The ISR is written by the programmer and this subprogram states what operation will be performed by the

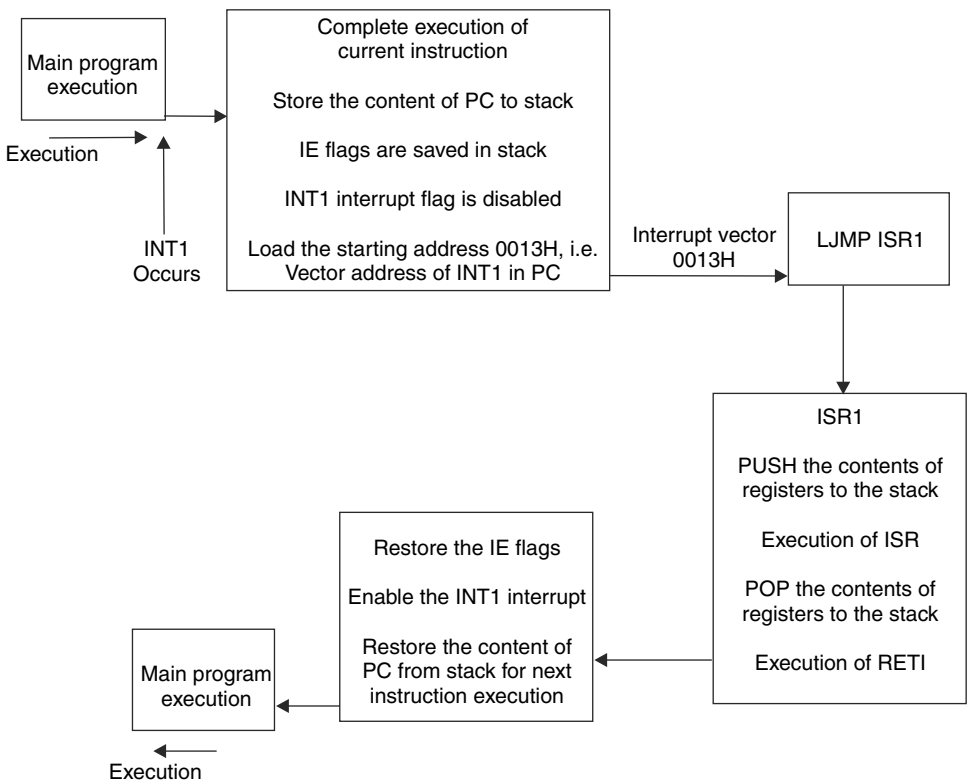


Fig. 7.42 The sequence of interrupt operation

INT1 interrupt. During execution of ISR, initially PUSH the contents of registers to stack and execute the subprogram part. After execution of the subprogram part, it is required to restore or POP the contents of these registers. The last instruction in ISR is RETI (Return from interrupt) instruction. When RETI instruction is executed, 8051 should restore the content of IE register, enable INT1 flag and also restore the content of program counter (PC) from the stack. As the PC contains the address of next instruction, 8051 microcontroller stars to execute the next instruction of main program. Figure 7.42 shows the sequence of operations when the microcontroller receives an interrupt.

7.8 SERIAL COMMUNICATION

Serial communication is most commonly used either to control or to receive data from an embedded microprocessor. The advantage of serial communication is that the number of wires required is less as compared to that in parallel communication. Serial communication is a form of I/O in which the bits of a byte begin transferred appear one after the other in a timed sequence on a single wire. Figure 7.43 shows the serial communication through telephone line where P/S is parallel in serial out shift register, S/P Serial in parallel out shift register, D/A digital to analog converter and A/D is analog to digital converter.

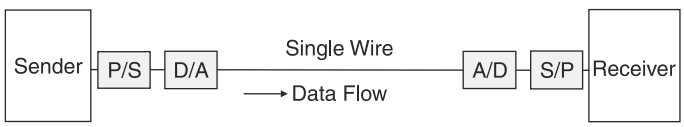


Fig. 7.43 Serial communication through single wire

There are two methods of serial communications, such as synchronous and asynchronous communications. In synchronous communication, transfer a block of data at a time, but in asynchronous communication transfer a single byte at a time. Software can be used for synchronous and asynchronous communications, but the

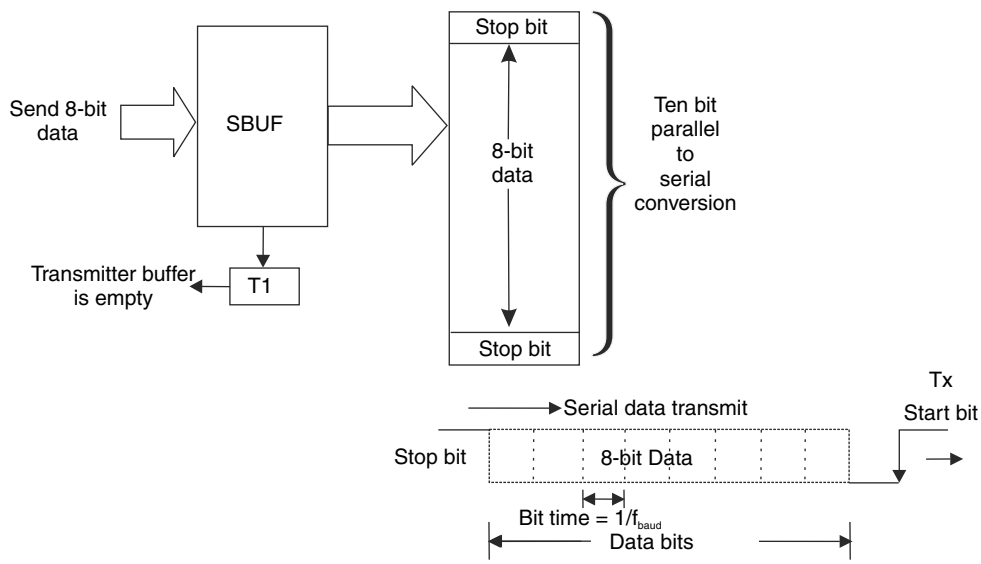


Fig. 7.44 Block diagram of UART (a) Transmitter half

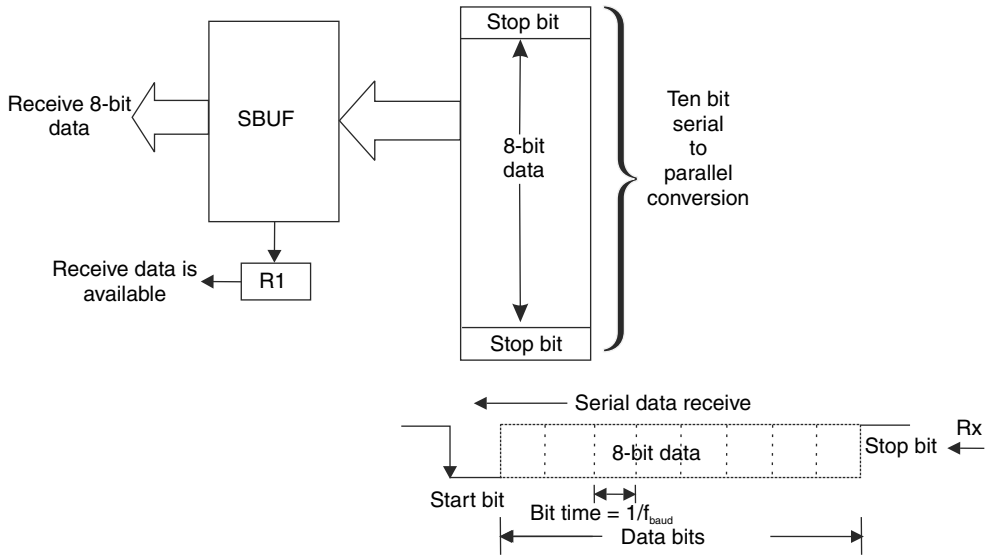


Fig. 7.44 (Contd.) Block diagram of UART (b) Receiver half

programs can be tedious and long. Therefore hardware such as UART and USART are developed. Usually UART (Universal Asynchronous Receiver Transmitter) or USART (Universal Synchronous Asynchronous Receiver Transmitter) are used in serial communication. The 8051 microcontroller has a build in UART.

8051 support a full duplex serial port (UART). 8051 has T x D and R x D pins for transmission and receive serial data respectively. The function of serial port is to perform parallel to serial conversion for data output and serial to parallel conversion for data input. The block diagram of UART is shown in Fig. 7.44 above.

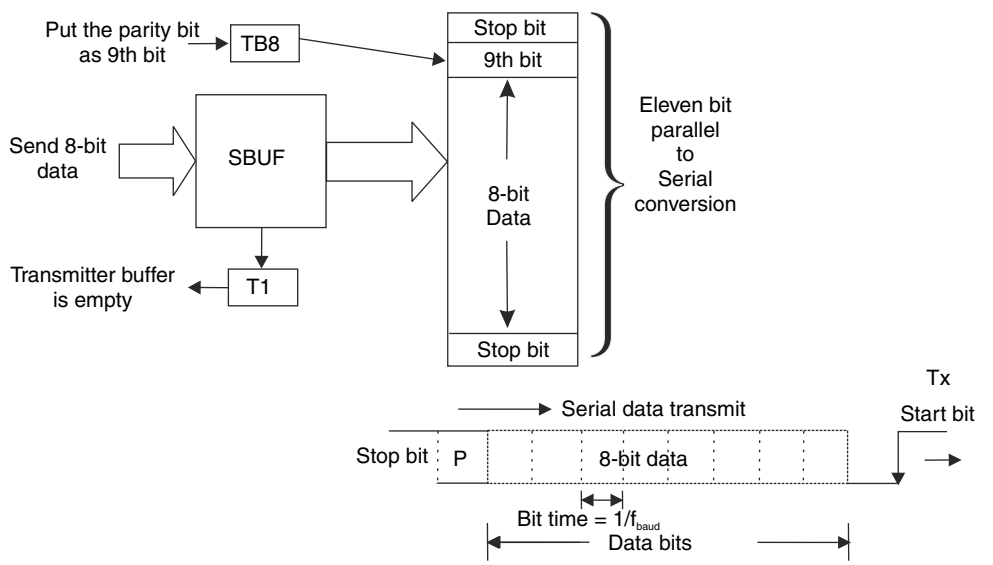


Fig. 7.45 Block diagram of UART transmitter for 11-bit transmission

The UART can be used for 9-bit data transmission and receive where 8-bits represent the data byte (information of character) and the 9th bit is the parity bit. A block diagram of UART transmitter is depicted in Fig. 7.45 (page 7.34) where the 9th bit is used as the parity bit. Figure 7.46 shows the block diagram of UART receiver where the 9th bit is used as the parity bit.

The 8051 serial communication can support RS232. RS232 is not compatible to TTL. Therefore, to connect any RS232 to a microcontroller, we must use voltage converters such as MAX232 to convert TTL logic level to the RS232 voltage levels as shown in Fig.7.47. The MAX232 IC is also known as line driver. The 8051 microcontroller has two pins such as TxD and RxD which are used for transferring and receiving data serially. TxD and RxD pins are the part of Port 3 (P3.0 and P3.1). These pins are TTL compatible and a line driver is required to make these pins RS232 compatible. The serial communication between two microcontrollers and between microcontroller and microprocessor is also possible.

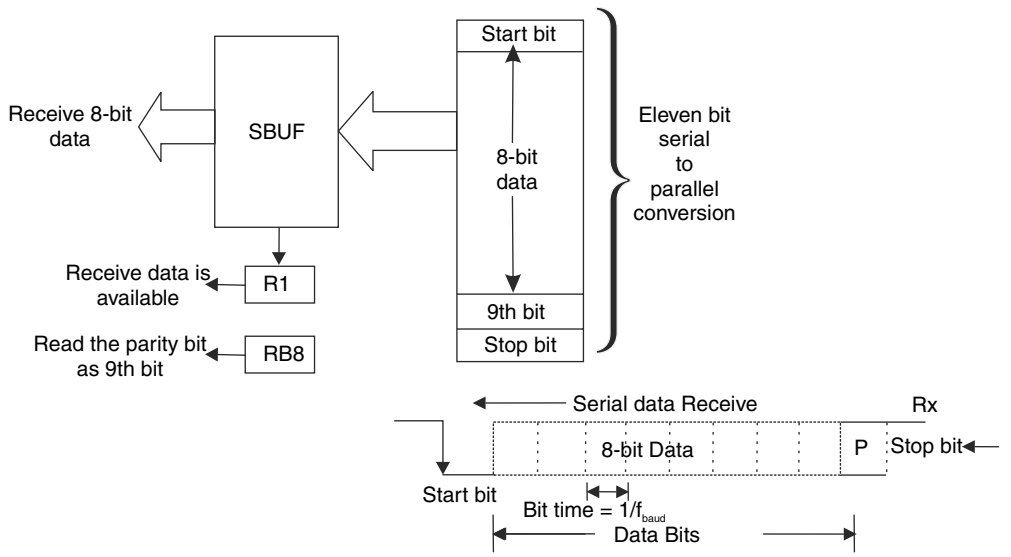


Fig. 7.46 Block diagram of UART receiver to receive 11 bits

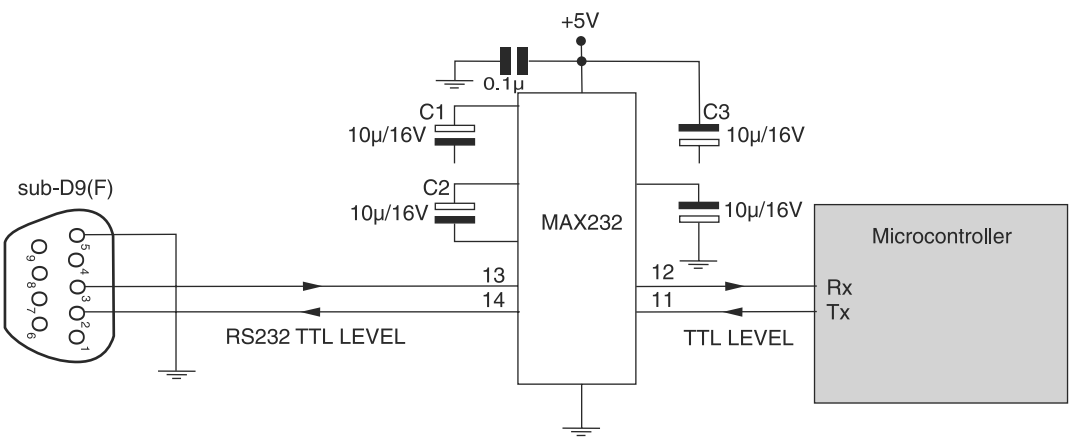


Fig. 7.47 MAX232 IC as a line driver

7.8.1 Registers for Serial Data Communication

The 8051 microcontroller use the following registers for serial data communication:

- SBUF(Serial port data buffer)
- SCON(Serial port control) register
- PCON(Power control) register

SBUF (Serial port data buffer) The serial port data buffer register has two registers. One register is used to hold data that to be transmitted through TxD of 8051 and it is write only type. The other register can be able to hold data from external sources through RxD of 8051 and it is read only type.

SCON (Serial port control) Register The format of SCON (serial port control) register is shown in Fig. 7.48.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Fig. 7.48 SCON register

Table 7.10 SCON Register

Bit	Name	Bit Address	Function
7	**SM0**	9FH	Serial port mode bit 0. This bit is set/cleared by program to select operating mode as shown in Table 1.1.
6	**SM1**	9EH	Serial port mode bit 1. This bit is set/cleared by program to select operating mode as shown in Table 1.1.
5	**SM2**	9DH	This pin enables the mutliprocessor communication feature in modes 2 and 3. In mode 2 and 3, if SM2=1, then RI will not be activated, if the received 9th data bit RB8 is 0. In Mode 0, SM2 must be 0. In mode 1, if SM=1, then RI will not be activated if a valid stop bit is not received.
4	**REN**	9CH	Receiver Enable bit. This bit must be set in order to receive characters. This bit must be cleared to disable reception.
3	**TB8**	9BH	Transmit bit 8. The 9th bit will be transmitted in mode 2 and 3. This bit can be set/ cleared by software.
2	**RB8**	9AH	Receive bit 8. The 9th bit will be received in mode 2 and 3. In mode 1, if SM2=0, RB8 is the stop bit that was received. In mod 0, RB8 is not used.
1	**TI**	99H	Transmit Interrupt Flag. This bit is set by hardware at the end of the 8th bit time in mode 0. This can also be set by hardware at the beginning of the stop bit in other modes. This bit can be cleared by software.
0	**RI**	98H	Receive Interrupt Flag. This bit is set by hardware at the end of the 8th bit time in mode 0 or halfway through the stop bit in other modes. This bit can be cleared by software.

SM0	SM1	Serial Mode	Description	Baud Rate
0	0	0	8-bit Shift Register	Oscillator frequency/ 12
0	1	1	8-bit UART	Variable which is set by Timer 1
1	0	2	9-bit UART	Oscillator frequency/ 32 or Oscillator frequency/ 64
1	1	3	9-bit UART	Variable which is set by Timer 1

PCON (Power control) Register The format of power control register is shown in Fig.7.49

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SMOD	-	-	-	GF1	GF0	PD	IDL

Fig. 7.49 PCON register

SMOD Double baud-rate bit. When this bit is set to 1, timer 1 is used to generate baud rate and the baud rate is doubled while the serial port is used in modes 1, 2 or 3.

GF1 General-purpose flag bit

GF0 General-purpose flag bit

PD Power down bit. When this bit is set, power down operation in 8051 is performed. This is available only in CHMOS processors.

IDL Idle mode bit. If this bit is set, it activates idle mode operation in 8051. This is available only in CHMOS processors.

7.8.2 Serial Communication Modes

There are four serial communication modes such as

Mode 0—Shift register mode
Mode 1—Standard UART mode
Mode 2—Multiprocessor mode
Mode 3—9 bit UART mode

The above modes can be selected by the programmer by proper setting the mode bits SM0 and SM1 in SCON register.

Mode 0 This mode is known as shift register mode. When SM0 and SM1 are set to 00, the serial port data buffer (SBUF) receives and transmit data through the RxD pin. TxD pin outputs the shift clock only. RxD pin is connected to the internal shift frequency clock pulse to provide shift pulses to external circuits. In this mode eight data bits are transmitted or received. The shit frequency or baud rate is fixed and it can be determined from the system clock frequency. When the oscillator frequency is f_{osc}, the baud rate can be expressed as $\frac{f_{osc}}{12}$. For a 12 MHz crystal, the baud rate is 1 MHz. The transmission operation is initiated by executing instructions to write data to SBUF. Then data can be shifted out on RxD line when the clock pulse is applied through TxD line. The receiving operation is initiated when REL=1 and RI=0. REN is set at the beginning of the program and then RI is cleared to start a data input operation. Figure 7.50 shows the data transmission/reception in mode 0.

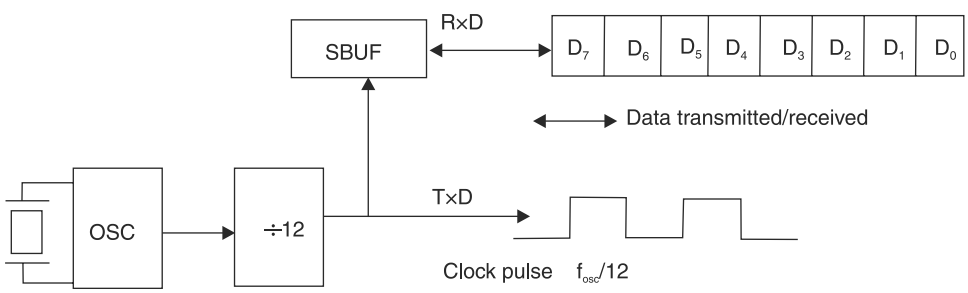


Fig. 7.50 Data transmission/receive in mode 0

Mode 1 When SM0 and SM1 are set to 01, mode 1 operation is performed. In this mode, 10 bits are transmitted through TxD pin or received through RxD pin. These 10 bits consists of one start bit (0), 8 data bits (LSB first) and a stop bit (1) as shown in Fig. 7.51. The transmit interrupt flag TI is set once after sending 10 bits. Each bit interval is the inverse of the baud rate frequency and each bit must be maintained high or low over this interval. After receiving, the stop bit goes to RB8 in SCON register. The baud rate is variable and it is computed by the timer 1 overflow rate. The baud rate can be expressed as

$$\text{Baud rate} = \frac{2^{\text{SMOD}}}{32 \times \text{Timer 1 overflow rate}}$$

When the timer 1 operates in auto-reload mode or mode 2 with reload count value in TH1, after each overflow the content of TH1 must be loaded into TL1. In this mode of operation, the high nibble of TMOD is 0010B. The baud rate can be expressed as

$$\text{Baud rate} = \frac{2^{\text{SMOD}} \times \text{oscillator frequency}}{32 \times 12 \times (256 - [\text{TH1}])}$$

For example, if the contents of TH1 are 230_D and the SMOD bit in PCON is 0, the baud rate is 1201 baud or 1.2K (approx) for 12 MHz oscillator frequency.

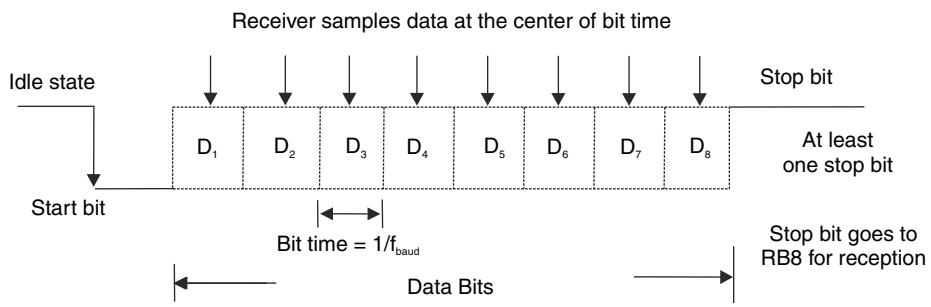


Fig. 7.51 Data transmission/receive in mode 1

Mode 2 In this mode, serial operates as a 9-bit UART and 11 bits are transmitted or received. These 11 bits are a start bit (always 0), 8 data bits (LSB first), a programmable 9th bit and a stop bit (always 1) as shown in Fig. 7.52. Programmer can define 9th bit as TB8 in SCON and it can be used as the parity bit of data byte. On reception, the 9th bit is placed into RB8 in SCON. In mode 2, the bit SMOD in PCON and oscillator frequency determine the baud rate and it can be expressed as

$$\text{Baud rate} = \frac{2^{\text{SMOD}}}{64} \times (\text{oscillator frequency})$$

$$\text{If SMOD = 0, Baud rate} = \frac{2^0}{64} \times (\text{oscillator frequency}) = \frac{\text{Oscillator frequency}}{64}$$

$$\text{If SMOD = 1, Baud rate} = \frac{2^1}{64} \times (\text{oscillator frequency}) = \frac{\text{Oscillator frequency}}{32}$$

Mode 3 In this mode, serial port operates as a 9-bit UART with variable baud rate and 11–bits are transmitted or received. This operating mode is same as mode 2 except baud rate is programmable through the timer 1 overflow rate. The baud rate calculations are same as that of mode 1 and it can be expressed as

$$\text{Baud rate} = \frac{2^{\text{SMOD}} \times \text{oscillator frequency}}{32 \times 12 \times (256 - [\text{TH1}])}.$$

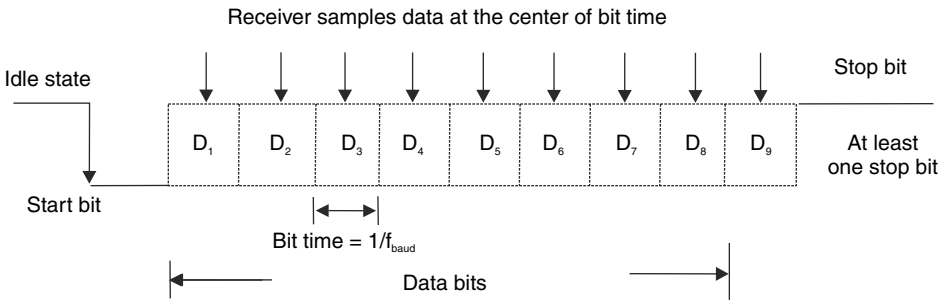


Fig. 7.52 Eleven bits transmitted in 8051 serial communication mode

7.8.3 Multiprocessor Communication

8051 microcontrollers can be operating in multiprocessor mode for serial communication mode 2 and mode 3. In this mode, there is a master processor (master microcontroller) which can communicate with more than one slave processors (slave microcontrollers) as shown in Fig. 7.53. SM2 bit in SCON register is used as a flag for multiprocessor communication. Whenever a byte has been received, the 8051 will set the RI (Receive Interrupt) flag. Consequently, the program knows that a byte has been received and it is required to process data.

In multiprocessor mode, 9-bits are transferred or received. When SM2 is set, the RI flag will be triggered. If the 9th bit is cleared, The RI flag will never be set. Generally, the 9th bit is kept clear so that the RI flag is set after receiving any character. In serial communication Modes 2 and 3, the transmitting processor is used as a master 8051 which can control several slave 8051 microcontroller. The TxD outputs of the slave microcontrollers are joined together and connected to the RxD input of the master microcontroller. The RxD inputs of the slaves are tired together and connected to the TxD output of the master. Each slave microcontroller is assigned a specified address. When the master wants to transmit a block of data, it must send first the address byte of the slave. While transmitting address byte by the master, the 9th bit is '1' and 9th bit '0' during data bytes transfer.

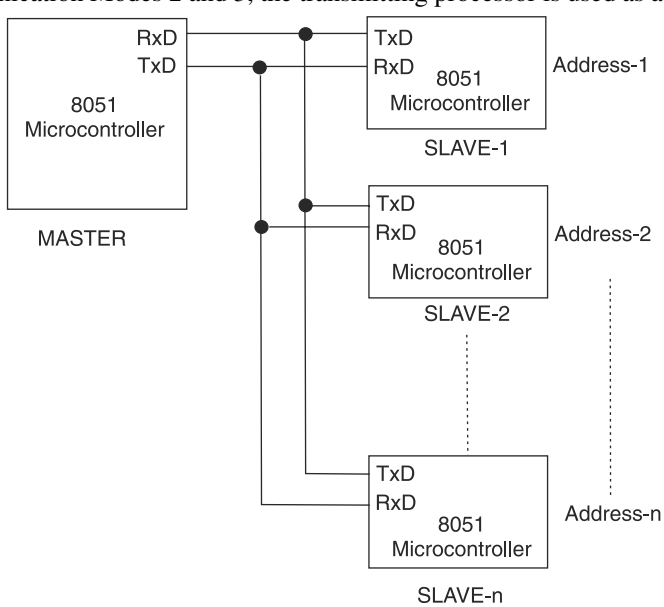


Fig. 7.53 Multiprocessor communication

Whenever the master 8051 transmits an address byte, all slave 8051 microcontrollers are interrupted. Then slave microcontrollers check to observe if they are being addressed or not. Subsequently, the addressed slave clear its SM2 bit and wait

to receive the data bytes. Other slaves who are not addressed can continue their operations ignoring the incoming data bytes and they will be interrupted again while the next address byte is transmitted by master controller. Usually, the master communicates with one slave at a time and transmit 11 bit which consists of one start bit (0), 8-data bits (LSB first) TB8 and a stop bit(1). The TB8 is '1' for address byte and '0' for a data byte.

When the master microcontroller wants to communicate with a slave microcontroller, it sends the address of the slave with TB8=1. Then all slave microcontrollers receive this address. Initially SM2 bit set to '1'. As all slave microcontrollers check the address to observe if they are being addressed or not. Then the selected slave microcontroller byte clear its SM2 bit to '0' so that data can be received. In mode 2 and 3, Receive Interrupt (RI) flag is set if REN=1 and RI=0, SM2=1 and RB8=1 and a valid stop bit is received. After proper communication between master microcontroller and a slave microcontroller, the data bytes will be sent by the master with TB8=0.

Review Questions

7.1 Define microcontroller. Write the differences between microprocessors and microcontrollers.

7.2 What are microcontroller families? What are the advantages of microcontroller-based systems over microprocessor-based systems?

7.3 Give a list of applications of microcontrollers.

7.4 What are the features of Intel 80C51 microcontroller? What are the general purposes registers of 80C51?

7.5 Draw the block diagram of the 8051 microcontroller and explain operation of each block briefly.

7.6 Draw the schematic pin diagram of the 8051 microcontroller and explain operation of the following pins:
(i) RST (ii) T×D (iii) R×D (iv) XTAL2 (v) ALE (vi) $\overline{EA}$ (vii) $\overline{PSEN}$ (viii) $\overline{RD}$ (x) $\overline{WR}$

7.7 What is the difference between internal and external program memory? Why is external program memory used in microcontroller? How $\overline{EA}$ can be used to access internal and external program memory?

7.8 What is an SFR? How can you identify the bit addressable SFRs from their addresses?

7.9 What are the ports used for external memory access? How can an I/O pin be used as both input and output?

7.10 What do you mean by "bi-directional port" and "quasi-bi-directional port"? How can Port 0 be used as bi-directional port?

7.11 Discuss power management in 8051 microcontroller.

7.12 Explain the various timer modes of 8051 microcontroller. What is the auto-reload mode?

7.13 Write short notes on the following:
(i) Serial data communication (ii) interrupts of 8051 (iii) Idle mode of 8051 (iv) Power-down mode of 8051.

7.14 Draw the schematic block diagram of multiprocessor communication using 8051 microcontroller and explain briefly.

7.15 Draw the block diagram of UART (a) Transmitter half (b) Receiver half and explain their operation briefly.

7.16 Explain interrupts in 8051 microcontroller.

Multiple-Choice Questions

7.1 The 8051 microcontroller has
(a) 8-bit data bus and 16-bit address bus (b) 16-bit data bus and 8-bit address bus
(c) 8-bit data bus and 8-bit address bus (d) 16-bit data bus and 16-bit address bus

7.2 The 8051 microcontroller has
(a) 4K bytes of on-chip ROM (b) 8K bytes of on-chip ROM
(c) 16K bytes of on-chip ROM (d) 32K bytes of on-chip ROM

7.3 The 80C51 microcontroller family has
(a) 32 pins for I/O (b) 24 pins for I/O (c) 16 pins for I/O (d) 8 pins for I/O

7.4 The 8051 microcontroller can support
(a) 5 interrupts (b) 4 interrupts (c) 3 interrupts (d) 2 interrupts

7.5 A 80C51 microcontroller has
(a) 128 bytes of on-chip RAM (b) 256 bytes of on-chip RAM
(c) 228 bytes of on-chip RAM (d) 556 bytes of on-chip RAM

7.6 A microcontroller has
(a) 4 on-chip I/O ports (b) 3 on-chip I/O ports
(c) 2 on-chip I/O ports (d) 1 on-chip I/O ports

7.7 The number of flags present in 8051 that respond to math operations are
(a) 2 (b) 3 (c) 4 (d) 5

7.8 Which of the following are 16-bit registers in the 80C51 microcontroller?
(a) DPTR (b) IE (c) TMOD (d) PC

7.9 Which of the following registers can be used to hold address of byte in memory of 80C51?
(a) DPTR (b) PCON (c) SBUF (d) PSW

7.10 Which of the following registers can be used as two individual 8-bit registers?
(a) DPTR (b) PC (c) SBUF (d) PSW

7.11 Which port can only be used as I/O port
(a) Port 0 (b) Port 1 (c) Port 2 (d) Port 3

7.12 Which of the following registers is bit addressable?
(a) SBUF (b) TMOD (c) PCON (d) PSW

7.13 The operation of PSW is
(a) hold the status of register bank currently being used
(b) holding data during data transfer operation
(c) holding math flags
(d) hold address of a byte in memory.

7.14 How many general-purpose registers exit in 8051?
(a) 10 (b) 16 (c) 20 (d) 32

7.15 The 8051 microcontroller has
(a) One-on-chip timer (b) Two-on-chip timers
(c) Three-on-chip timers (d) Four-on-chip timers

7.16 There are _____ interrupt sources for 8051 microcontroller
(a) 3 (b) 4 (c) 5 (d) 6

7.17 The power saving mode of 8051 microcontroller is activated by software via
(a) PCON (b) SCON (c) TCON (d) T2CON

7.18 The vector address of External interrupt 1 (IE1) is
(a) 0003H (b) 000BH (c) 0013H (d) 0018H

Answers to Multiple-Choice Questions

7.1 (b)	7.2 (a)	7.3 (a)	7.4 (a)
7.5 (a)	7.6 (a)	7.7 (c)	7.8 (a) & (d)
7.9 (a)	7.10 (a)	7.11 (b)	7.12 (d)
7.13 (c)	7.14 (a)	7.15 (c)	7.16 (c)
7.17 (a)	7.18 (c)		

CHAPTER

8

Instruction Set and Programming of 8051 Microcontroller

8.1 INTRODUCTION

In Chapter 7, the basic structure of 8051 microcontroller has been discussed elaborately. Like 8085 and 8080 microprocessors, the 8051 microcontroller has different addressing modes to locate operand in the instructions. In this section all addressing modes of the microcontroller has been discussed with examples. This microcontroller has arithmetic and logical instructions, data transfer, Boolean operation instructions, bit operation instructions, branch control instructions and program control instructions. All these instructions are explained with appropriate examples. The programming format and some simple programs such as addition, subtraction, multiplication, division, ascending order, descending order, look-up table, key board interface, A/D converter interface, and stepper motor control have been incorporated in this chapter to understand the applications of instructions.

8.2 ADDRESSING MODES

An instruction is used to load or transfer data from a source to a destination. The source may be any register, internal memory, external memory, any one of four ports or any external I/O peripheral devices. Similarly, destination may be any registers, memory (internal or external) and I/O devices. In any instruction of the 8051 microcontroller, the data is known as operand. The way in which an operand is specified is called addressing mode. There are different ways to specify operands for instructions. The commonly used addressing modes of 8051 microcontroller are as follows:

- Immediate Addressing
- Register Addressing Mode
- Direct Addressing
- Indirect Addressing
- Indexed Addressing
- Relative Addressing

- Absolute Addressing
- Long Addressing

8.2.1 Immediate Addressing

In immediate addressing mode, the source operand is a constant rather than a variable. The constant operand can be incorporated into the instruction as a byte of immediate address. The immediate operands are preceded by # sign in assembly language. The operand may be a numeric constant (decimal or hexadecimal), a symbolic variable or an arithmetic expression, e.g., **MOV A, #FFH** This instruction is used to load the immediate data FF H to A register.

MOV R0, #26 This is used to load the immediate data byte 26H to register R0.

All immediate addressing instructions use 8-bit data field. But one exception is that a 16-bit constant is required for initialisation of the data pointer register (DPTR). For example, MOV DPTR, #9000H. After execution of MOV DPTR, #9000H instruction, 9000H will be loaded into DPTR register. Table 8.1 shows some other examples of immediate addressing:

Table 8.1 Examples of immediate addressing

Instruction	Task
ADD A, # data	Add immediate data to accumulator
SUBB A, # data	Subtract immediate data from accumulator with borrow
MOV Rn, # data	Move immediate data to register R*n*
MOV DPTR, #data 16	Load data pointer register with a 16-bit constant

8.2.2 Register Addressing Mode

In the register addressing mode, the selected register bank containing registers R0 through R7 can be accessed by certain instructions which carry a 3-bit register specification within the opcode of the instruction. As the three least significant bits of the instruction opcode is used to specify a register, this addressing mode eliminates an address byte. When the instruction is executed, one of the eight registers in the selected bank will be accessed. One of four banks is selected at execution time by the two banks select bits RS_1 and RS_0 in the PSW. e.g., **MOV A, R0** Move the content of R0 register into accumulator

MOV R1, A Move the content of accumulator into R1 register

Some instructions are specific to a certain register. For example, some instructions always operate on the accumulator, or data pointer and no address byte is required to point it. The opcode itself specifies the source of operand and an example is INC A. In this instruction, accumulator itself is the operand. The examples of other register addressing instructions are given in Table 8.2.

Table 8.2 Examples of register addressing

Instruction	Task
ADD A, R*n*	Add the content of register R*n* to accumulator.
SUBB A, R*n*	Subtract the content of register from accumulator with borrow.
MOV R*n*, A	Move data from accumulator to register R*n*.
INC DPTR	Increment data pointer register by one.

8.2.3 Direct Addressing

In direct addressing mode, the operand is specified by an 8-bit address field in the instruction. Only lower 128 bytes of internal data RAM and SFRs can be directly addressed by using single byte address. e.g., **MOV A, 33H** This instruction is used to transfer the content of internal memory (RAM) location 33H to accumulator as shown in Fig. 8.1.

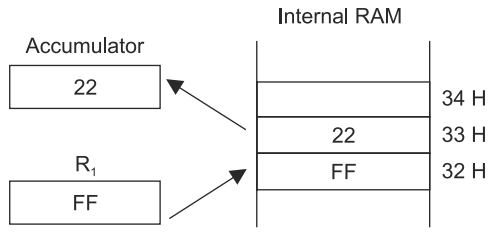


Fig. 8.1 Direct addressing

MOV 32, R1 The content of register R1 moves to internal memory location 32H as depicted in Fig. 8.1.

Table 8.3 shows some other examples of direct addressing.

Table 8.3 Examples of direct addressing

Instruction	Task
MOV A, direct	Copy the Contents of memory location specified by 'direct' into accumulator.
MOV A, SBUF	Copy the contents of special function register SBUF into accumulator.

8.2.4 Indirect Addressing

In indirect addressing mode, the instruction specifies a register which contains the address of the operand. Both internal and external RAM can be indirectly addressed. In this mode, R_0 or R_1 of selected bank or the stack pointer may operate as pointer registers for 8-bit addresses. Actually the content of R_0 or R_1 indicates an address in internal RAM where data will be stored or read. In assembly language programming, indirect addressing is presented by an @ symbol before R_0 or R_1 e.g., **MOV A,@R7.** In this instruction the content of R_7 represents the internal memory address. As the R_7 contain 33H, the internal memory location will be 33H. After the execution of this instruction, the value of internal memory location 33H will be loaded into accumulator as depicted in Fig. 8.2.

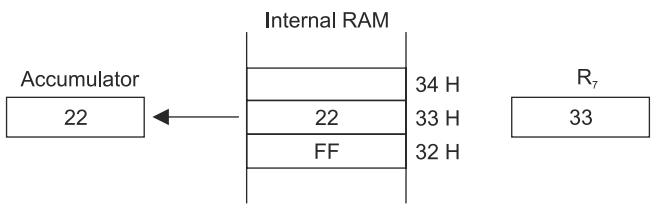


Fig. 8.2 Indirect addressing

The Data Pointer Register (DPTR) can also be used as the address register for 16-bit address. Some examples of indirect addressing are illustrated in Table 8.4.

Table 8.4 Examples of indirect addressing

Instruction	Task
ADD A, @R_0	Add the contents of the address specified by the R_0 register to accumulator
SUBB A, @R_1	Subtract the contents of the address specified by the R_1 register from accumulator with borrow

Contd.

Contd.

MOV @R_i, A	Move the content of accumulator to indirect RAM specified by R_i (R0 or R_1)
MOV A, @R_i	Moves a byte of data from internal RAM at location whose address is in R_i (R_0 or R_1) to the accumulator
DEC @R_i	Decrement indirect RAM specified by R_i (R_0 or R_1)

8.2.5 Indexed Addressing

In indexed addressing, only program memory can be accessed and it can only be read. This addressing mode is used for reading look-up tables in program memory. The effective address of program memory is calculated as the summation of the content of base register (program counter PC or data pointer DPTR) and an offset, i.e. the contents of accumulator. This addressing mode is intended for a JMP or MOVC instruction. e.g. **MOVC A, @A+DPTR** When this instruction is executed, move a byte of data from program memory whose address can be obtained by adding the accumulator to the data pointer, to the accumulator as depicted in Fig. 8.3.

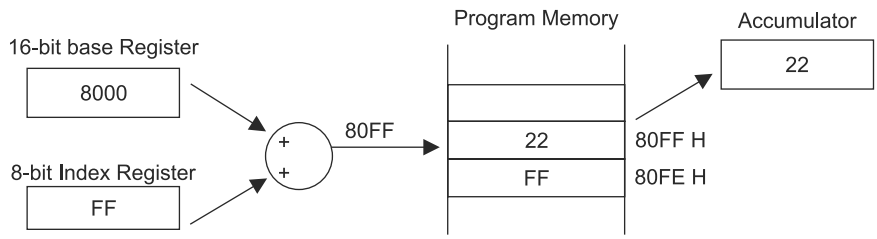


Fig. 8.3 Indexed addressing

A list of examples of indexed addressing are given in Table 8.5.

Table 8.5 Examples of indexed addressing

Instruction	Task
MOVC A,@A+PC	Move a byte of data from program memory whose address can be determined by sum of accumulator and program counter, to the accumulator.
MOVC A,@A+DPTR	Move a byte of data from program memory whose address can be found by adding the accumulator and the data pointer, to the accumulator.
JMP @A+DPTR	Jump indirect relative to the data pointer. The address of a jump instruction is calculated as sum of the accumulator and the data pointer.

8.2.6 Relative Addressing

Generally, this addressing mode is used in certain jump instructions. The relative address is an 8-bit signed number (–128 to 127), which is added to the program counter to determine the address of next instruction. Before addition, the program counter is incremented. Therefore, the new address is relative to the next instruction is determined and then jump to the new address instruction. For example, SJMP Level offset. When this instruction is executed, the new address can be obtained from the sum of the PC and offset. Then jump to the new address as depicted in Fig. 8.4. The advantage of relative addressing is that it has position independent codes.

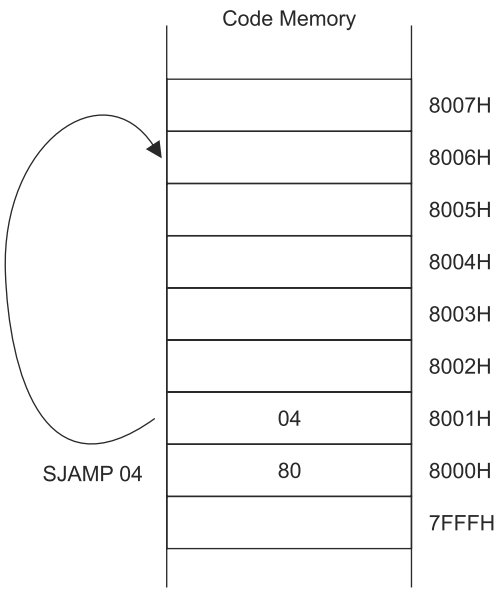


Fig. 8.4 Relative addressing

8.2.7 Absolute Addressing

The absolute addressing is only used with AJMP and ACALL instructions. The 11 least significant bits of the destination address comes from the opcode and the upper five bits are the current upper five bits in the program counter (PC). In this addressing mode, the destination address will be within 2K (2^{11}) memory. For example, ACALL addr11.

8.2.8 Long Addressing

The long addressing is used only with the LJMP and LCALL instructions. These instructions include a full 16-bit destination address. In this mode, the full 64K code space is available and the instruction is long and position dependent. For example LJMP 9500H. When this instruction is executed, it jumps to memory location 9500H.

Example 8.1 Find the addressing modes of the following instructions:

(i) ADD A, R_7
(ii) ADD A, 55H
(iii) MOV A, @R_0
(iv) MUL AB
(v) MOV A, #FF
(vi) MOV DPTR, #8000
(vii) MOVC A, @A+DPTR

Sol. (i) ADD A, R_7 instruction is an example of register addressing
(ii) ADD A, 55H instruction is an example of direct addressing
(iii) MOV A, @R_0 instruction is an example of indirect addressing
(iv) MUL AB instruction is an example of register addressing
(v) MOV A, #FF instruction is an example of immediate addressing
(vi) MOV DPTR, #8000 instruction is an example of immediate addressing
(vii) MOVC A, @A+DPTR instruction is an example of index addressing

8.3 8051 INSTRUCTION SET

An instruction is a command applied to a microcontroller to perform specified operation. There are 255 possible instructions in the 8051 microcontroller. Each instruction consists of operation code (opcode) and operand. Opcode states the specified operation, which will be performed. The operand means data, which will be used in that operation as no operation can be performed without data. The 8051 instructions have 8-bit opcode. Data field may be of one byte or two bytes. Based on data (operand) the instructions are classified as one byte, two-byte and three-byte instructions. 8051 instructions are divided into following groups as given below:

- Arithmetic instructions
- Logical instructions
- Data transfer instructions
- Boolean operations instructions
- Program control instructions
- Branching instructions

The symbols and abbreviations, which have been used while explaining Intel 8051 microcontroller instructions, are illustrated in Table 8.6. In this section, all groups of instructions are explained elaborately with appropriate representations.

Table 8.6 Symbol/Abbreviations of instruction set

Symbol/Abbreviations	Meaning
addr 16 16-bit address	16-bit destination address. Used by LCALL and LJMP. A branch can be anywhere within the 64K-byte program memory address space.
addr 11 11-bit address	11-bit destination address. Used by ACALL and AJMP. The branch will be within the same 2K-byte page of program memory as the first byte of the following instruction.
#data	8-bit constant included in the instruction.
#data 16	16-bit constant included in the instruction
R*n*, R*i*	Register R_7–R_0 of the currently selected register bank.
direct	8-bit internal data location's address. This could be an Internal Data RAM location (0–127) or a SFR [i.e., I/O port, control register, status register, etc. (128–255)].
@R*i*	8-bit internal data RAM location (0–255) addressed indirectly through register R_1 or R_0.
rel	Signed (two's complement) 8-bit offset byte. Used by SJMP and all conditional jumps. Range is –128 to +127 bytes relative to first byte of the instruction
bit	Direct addressed bit in Internal Data RAM or Special Function Register.
DPTR	Data pointer register
DPH, DPL	DPH–Data pointer register higher, DPL—Data pointer register lower
SP	SP represents 16-bit stack pointer
PC	16-bit program counter
PSW	Program Status Word
CS	Carry status
[ ]	The content of the memory location
←	Move data in the direction of arrow
⇔	Exchange contents
∧	Logical AND operation
∨	Logical OR operation
⊕	Logical EXCLUSIVE OR
|	Complement

8.3.1 Arithmetic Instructions

The arithmetic instructions are used to perform arithmetic operations such as addition, subtraction, increment, decrement, multiplication, and division. Since different addressing modes are available, an arithmetic instruction can be written in different ways. All arithmetic instructions are executed in one machine cycle except INC DPTR, MUL AB and DIV AB. INC DPTR requires two machine cycles and MUL AB and DIV AB require four-machine cycle. All arithmetic instructions are given below:

ADD A, Rn (Add register to accumulator)

A ← A+ R*n*,

Machine cycles: 1, States: 12, Flags: all, Register Addressing, One byte instruction

The contents of the operand (register) are added to the contents of the accumulator and the result is stored in the accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands. For example **ADD A, R6** Add the content of register R6 to accumulator and result in accumulator.

ADD A, direct (Add direct byte to accumulator)

A ← A+ [direct]

Machine cycles: 1, States: 12, Flags: all, Direct Addressing, Two byte instruction

The contents of the internal memory location specified by the 8-bit direct are added with accumulator. All flags are modified to reflect the result of the addition. For example, the instruction is ADD A, 44H.

ADD A, @R_i (Add indirect RAM to accumulator)

A ← A+ [R_i]

Machine cycles: 1, States: 12, Flags: all, Register indirect Addressing, One byte instruction

The contents of the internal RAM whose location is denoted by the content of register R*i* (R_0 or R_1) are added to the contents of the accumulator and the result is stored in the accumulator, for example, ADD A,@R_0.

ADD A, #data (Add immediate data to accumulator)

A ← A + #data

Machine cycles: 1, States: 12, Flags: all, Immediate Addressing, Two-byte instruction

Add the number specified by #data to accumulator and the result is stored in the accumulator, for example ADD A, 36H.

ADDC A, R_n (Add register to accumulator with carry)

A ← A + R*n* + C

Machine cycles: 1, States: 12. Flags: all, Register Addressing, One-byte instruction.

ADDC instruction simultaneously adds the contents of the register R*n* and the Carry flag to the contents of the accumulator and the result is stored in the accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. During adding unsigned integers, the carry flag indicates an overflow occurred. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands. Therefore, flags

are modified to reflect the result of the addition, e.g., ADDC A, R_7. If accumulator content is C_3H, R_7 content is AAH with the carry flag set, the result in accumulator is 6E and AC cleared, the carry flag and OV set to 1.

ADDC A, direct (Add direct byte to accumulator with carry)

A ← A + [direct] + C

Machine cycles: 1, States: 12, Flags: all, direct Addressing, Two-byte instruction

The content of the memory location, which is specified by the direct address and the Carry flag are added to the contents of the accumulator. After addition, the result is stored in the accumulator. All flags are effected to reflect the result of the addition, e.g., ADDC A, 55H.

ADDC A, @Ri (Add indirect RAM to accumulator with carry)

A ← A + [R*i*] + C

Machine cycles: 1, States: 12, Flags: all, Register indirect Addressing, one byte instruction

The contents of the internal memory RAM located by R*i* register (R_0 or R_1) are added to the contents of the accumulator with carry and the result is stored in the accumulator, e.g., ADDC A, @R_1

ADDC A, #data (Add immediate data to ACC with carry)

A ← A + C + #data

Machine cycles: 1, States: 12, Flags: all, Immediate Addressing, Two-byte instruction

The 8-bit immediate data (operand) can be added to the contents of the accumulator with carry and the result is stored in the accumulator, e.g. ADDC A, #FF

SUBB A, Rn (Subtract register from accumulator with borrow)

A ← A – R*n* – C

Machine cycles: 1, States: 12, Flags: all, Register Addressing, One-byte instructions

SUBB A, R*n* states that the content of register *Rn* and the carry flag are subtracted from the content of the accumulator. After subtraction, the result is stored in the accumulator. This instruction sets the carry (borrow) flag if a borrow is needed for bit 7, and clears C otherwise. AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6. During subtraction of signed integers, OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number. For example, SUBB A, R_2. If accumulator content is C_9H, content of R_2 is 54H and the carry flag is set, the result 74H will be in accumulator with the carry flag and AC cleared but OV set.

SUBB A, direct (Subtract direct byte from accumulator with borrow)

A ← A – [direct] – C

Machine cycles: 1, States: 12, Flags all, Direct Addressing, two-byte instructions

The contents of the 8-bit direct memory location are subtracted from the contents of the accumulator with borrow and the result is placed in the accumulator. All flags will be modified to reflect the result, e.g., SBBB A, 45H.

SUBB A, @Ri (Subtract indirect RAM from accumulator with borrow)

A ← A – [Ri] – C

Machine cycles: 1, States: 12, Flags: all, Register indirect Addressing, One-byte instructions

The content of internal RAM whose location is specified by register R*i* (R_0 or R_1) is subtracted from the

content of the accumulator with borrow, and the result is stored in the accumulator, e.g., SUBB A, @R0.

SUBB A, #data (Subtract immediate data from accumulator with borrow)

A ← A – #data – C

Machine cycles: 1, States: 12, Flags all, Immediate Addressing, Two-byte instructions

The 8-bit immediate data is subtracted from the contents of the accumulator with borrow and the result is placed in the accumulator, e.g., SBBB A, 78H.

INC A (Increment accumulator by 1)

A ← A + 1.

Machine cycles: 1, States: 12, No Flags are affected, Register Addressing, One-byte instructions

When INC instruction is executed, the indicated variable is incremented by 1. Therefore, the contents of the accumulator are incremented by 1 and the result is stored in the accumulator, e.g., INC A

INC Rn (Increment register by 1)

R*n* ← R*n* + 1.

Machine cycles: 1, States: 12, No Flags are affected, Register Addressing, One-byte instructions

The contents of the selected register R*n* (R_0 to R_7) are incremented by 1 and the result is stored in the same register, e.g., INC R_5.

INC direct (Increment direct byte)

[direct] ← [direct] + 1.

Machine cycles: 1, States: 12, No Flags are affected, Direct Addressing, two-byte instructions

The contents of the 8-bit direct memory location are incremented by 1 and the result is stored in the same memory location, e.g., INC 44H.

INC @Ri (Incrément indirect RAM)

[R*i*] ← [R*i*] + 1.

Machine cycles: 1, States: 12, No Flags are affected, Register indirect Addressing, One-byte instructions

The contents of the internal RAM location whose address can be selected by register R_0 or R_1 are incremented by 1 and the result is stored in the same RAM location, e.g., INC @R_0.

INC DPTR (Increment data pointer register by 1)

DPTR ← DPTR + 1.

Machine cycles: 2, States: 24, No Flags are affected, Register Addressing, one-byte instructions

The contents of the 16-bit data pointer register are incremented by 1 and the result is stored in the same register. A 16-bit increment is performed; an overflow of the low-order byte of the data pointer (DPL) from FFH to 00H will increment the high-order byte (DPH). No flags are affected, e.g., INC DPTR

DEC A (Decrement accumulator by 1)

A ← A – 1.

Machine cycles: 1, States: 12, Flags: No flags are affected, Register Addressing, One-byte instruction

The contents of the accumulator are decremented by 1 and the result is stored in the accumulator, e.g., DEC A.

DEC Rn (Decrement register by 1)

R*n* ← R*n* – 1.

Machine cycles: 1, States: 12, Flags: No flags are affected, Register Addressing, One-byte instruction

The contents of the selected register R_0 to R_7 are decremented by 1 and the result is stored in the same register, e.g., DEC R_6.

DEC direct (Decrement direct byte)

[direct] ← [direct] – 1.

Machine cycles: 1, States: 12, Flags: No flags are affected, Direct Addressing, Two-byte instruction.

The contents of the 8-bit direct memory location is decremented by 1 and the result is stored in the same memory location, e.g., DEC 34H.

DEC @Ri (Decrement indirect RAM)

[R*i*] ← [R*i*] – 1.

Machine cycles: 1, States: 12, Flags: No flags are affected, Register indirect Addressing, One-byte instruction

The contents of the internal RAM location specified by register R_0 or R_1 are decremented by 1 and the result is stored in the same place, e.g., DEC @R*i*.

MUL AB (Multiply A by B)

$A_{7\text{-}0} \leftarrow A \times B$

$B_{15\text{-}8}$

Machine cycles: 4, States: 48, Flags: Flags are affected, Register Addressing, One-byte instruction

MUL AB multiplies the unsigned eight-bit integers in the Accumulator and register B. The low-order byte of the sixteen-bit product will be stored in the accumulator, and the high-order byte will be stored in B. If the product is greater than 255 (FFH) the overflow flag is set; otherwise it is cleared. The carry flag is always cleared, e.g., MUL AB.

DIV AB (Divide A by B)

$A_{15\text{-}8} \leftarrow A/B$

$B_{7\text{-}0}$

Machine cycles: 4, States: 48, Flags: Flags are affected, Register Addressing, One-byte instruction

DIV AB divides the unsigned eight-bit integer in the accumulator by the unsigned eight-bit integer in register B. After execution of DIV AB, the accumulator receives the integer part of the quotient and register B receives the integer remainder. The carry and OV flags will be cleared, e.g., DIV AB.

DAA (Decimal adjust accumulator for addition)

Machine cycles: 1, States: 12, Flags : all, One byte instruction

If $[[(A_{3\text{-}0}) > 9] \vee [(AC)=1]]$, then $(A_{3\text{-}0}) \leftarrow (A_{3\text{-}0}) + 6$

AND If $[[(A_{7\text{-}4}) > 9] \vee [(C)=1]]$, then $(A_{7\text{-}4}) \leftarrow (A_{7\text{-}4}) + 6$

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

- When the value of the low-order 4-bits/nibble in the accumulator is greater than 9 or AC flag is set, the instruction adds 6 to the low-order four bits.
- If the value of the high-order 4-bits/nibble in the accumulator is greater than 9 or the Carry flag is set, the instruction adds 6 to the high-order four bits. In this instruction S, Z, AC, P, CY flags are altered to reflect the results of the operation, e.g.,

ADC A, R_3
DAA

If the accumulator holds 56H, i.e., the packed BCD digits of decimal number 56 and the content of register R_3 is 67H, i.e., the packed BCD digits of decimal number 67. The carry flag is set. After execution of above instructions, 24 H will be stored in accumulator as the true sum of 56 and 67 is 124.

Table 8.7 shows the 8051 arithmetic instruction set summary.

Table 8.7 8051 Arithmetic instruction set summary

Opcode	Operand	Functions	Clock cycle	Number of bytes	Instruction code							
ADD	A, Rn	Add register to accumulator	12	1	0	0	1	0	1	r	r	r
ADD	A, direct	Add direct byte to accumulator	12	2	0	0	1	0	0	1	0	1
ADD	A, @Ri	Add indirect RAM to accumulator	12	1	0	0	1	0	0	1	1	i
ADD	A, #data	Add immediate data to accumulator	12	2	0	0	1	0	0	1	0	0
ADDC	A, Rn	Add register to accumulator with carry	12	1	0	0	1	1	1	r	r	r
ADDC	A, direct	Add direct byte to accumulator with carry	12	2	0	0	1	1	0	1	0	1
ADDC	A, @Ri	Add indirect RAM to Accumulator with carry	12	1	0	0	1	1	0	1	1	i
ADDC	A, #data	Add immediate data to ACC with carry	12	2	0	0	1	1	0	1	0	0
SUBB	A, Rn	Subtract register from ACC with borrow	12	1	1	0	0	1	1	r	r	r
SUBB	A, direct	Subtract direct byte from ACC with borrow	12	2	1	0	0	1	0	1	0	1
SUBB	A, @Ri	Subtract indirect RAM from ACC with borrow	12	1	1	0	0	1	0	1	1	i
SUBB	A, #data	Subtract immediate data from ACC with borrow	12	2	1	0	0	1	0	1	0	0
INC	A	Increment accumulator	12	1	0	0	0	0	0	1	0	0
INC	Rn	Increment register	12	1	0	0	0	0	1	r	r	r
INC	direct	Increment direct byte	12	2	0	0	0	0	0	1	0	1
INC	@Ri	Increment indirect RAM	12	1	0	0	0	0	0	1	1	i
INC	DPTR	Increment data pointer	12	1	0	0	0	0	0	0	1	1
DEC	A	Decrement accumulator	12	1	0	0	0	1	0	1	0	0
DEC	Rn	Decrement register	12	1	0	0	0	1	1	r	r	r
DEC	direct	Decrement direct byte	12	2	0	0	0	1	0	1	0	1
DEC	@Ri	Decrement indirect RAM	12	1	0	0	0	1	0	1	1	i
MUL	AB	Multiply A and B	48	1	1	0	1	0	0	1	0	0
DIV	AB	Divide A by B	48	1	1	0	0	0	0	1	0	0
DAA		Decimal adjust accumulator	12	1	1	1	0	1	0	1	0	0

Example 8.2 Write instructions for the following operations:

(i) Add 23H to the contents of accumulator.
(ii) Add the content of the memory location specified by R_0 with accumulator.
(iii) Subtract the content of R_1 register from accumulator with borrow.

(iv) Subtract immediately 45 from accumulator register with borrow.

(v) Increment the content of internal memory location specified by R_0.

Sol.

(i) ADD A, #23; Add 23H to the contents of accumulator.

(ii) ADD A, @R_0; Add the content of the memory location specified by R_0 with accumulator.

(iii) SUBB A, R_1; Subtract the content of R_1 register from accumulator with borrow.

(iv) SUBB A, #45 ; Subtract immediately 45 from accumulator register with borrow.

(v) SBB A, 45; Subtract immediately 45 from ACC register with borrow.

(vi) INC @R_0; Increment the content of internal memory location specified by R_0.

Example 8.3 Write instructions for the following operations:

(i) Multiply the content of accumulator by B register.

(i) Divide the content of accumulator by B register.

(ii) Increment data pointer register by one.

Sol.

(i) MUL A B ; Multiply the content of accumulator by B register.

(ii) DIV A B ; Divide the content of accumulator by B register.

(iii) INC DPTR ; Increment data pointer register by one.

8.3.2 Logical Instructions

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

ANL performs the bitwise logical-AND operation between the variables indicated in instruction and stores the results in the destination variable. No flags are affected. The two operands allow six addressing-mode combinations. If the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing. When the destination is a direct address, the source can be the accumulator or immediate data.

ANL A, Rn (Logical AND register to accumulator)

$A \leftarrow A \wedge R_n$

Machine cycles: 1, States: 12, Flags: No flags are affected, Register Addressing, One-byte instructions

The contents of the accumulator are logically ANDed with the contents of the register R*n* (R_0–R_7). The result is stored in the accumulator. No flags are affected, e.g., ANL A, R_5.

ANL A, direct (Logical AND direct byte to accumulator)

$A \leftarrow A \wedge$ [direct].

Machine cycles: 1, States: 12, Flags: No flags are affected, Direct Addressing, Two-byte instructions

The content of the 8-bit memory location whose address is specified by the direct address is ANDed with the accumulator. The result is placed in the accumulator. No flags are affected, e.g., ANL A, direct

ANL A, @Ri (Logical AND direct byte to Accumulator)

$A \leftarrow A \wedge [Ri]$

Machine cycles: 1, States : 12, Flags: No flags are affected, Register indirect Addressing, One-byte instructions
The content of the memory location whose address is specified by the register R_0 or R_1 is ANDed with the accumulator. After ANDing, the result is stored in the accumulator. No flags are affected, e.g., ANL A,@R_i.

ANL A, #data (Logical AND immediate data to accumulator)

A ← A ∧ #data.

Machine cycles: 1, States: 12, Flags: No flags are affected, Immediate Addressing, Two-byte instructions

The contents of the accumulator are logically ANDed with the 8-bit data (#data). After ANDing the result is stored in the accumulator. No flags are affected, e.g., ANL A, #45H.

ANL direct, A (Logical AND immediate data to direct byte)

A ← [direct] ∧ A.

Machine cycles: 1, States: 12, Flags: No flags are affected, One-byte instructions

The content of the memory location whose address is specified by the 8-bit direct address is ANDed with the accumulator. The result will be stored in the 8-bit direct memory address. No flags are affected, e.g., ANL direct, A.

ANL direct, #data (Logical AND memory with accumulator)

[direct] ← [direct] ∧ #data.

Machine cycles: 1, States: 12, Flags: No flags are affected, Immediate Addressing, One-byte instructions

The content of the memory location whose address is specified by the 8 bit direct address is ANDed with the 8-bit immediate data. The result will be stored in the 8-bit direct memory address. No flags are affected, e.g., ANL direct, #A.

ORL performs the bitwise logical-OR operation between the indicated variables in instruction and store the results in the destination byte. No flags are affected. Six different addressing mode combinations are available for this instruction. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing. While the destination is a direct address, the source can be the Accumulator or immediate data.

ORL A, Rn (Logical OR register to accumulator)

A ← A ∨ R*n*

Machine cycles: 1, States: 12, Flags: No flags are affected, Register Addressing, One-byte instructions

The content of register R*n* (R_0–R_7) is logically ORed with the content of the accumulator. The result is stored in the accumulator. No flags are affected, e.g., ORL A, R*n*

ORL A, direct (Logical OR direct byte to accumulator)

A ← A ∨ [direct].

Machine cycles: 1, States: 12, Flags: No flags are affected, Direct Addressing, Two-byte instructions

The contents of the accumulator are logically ORed with the contents of the memory location, whose address is specified by the 8-bit direct address and the result is placed in the accumulator. No flags are affected. The example is ORA A, direct.

ORL A, @Ri (Logical OR indirect RAM to accumulator)

A ← A ∨ [R*i*].

Machine cycles: 1, States: 12, Flags: No flags are affected, Register indirect Addressing, One-byte instructions

The contents of the accumulator are logically ORed with the contents of the memory location, whose address is specified by the content of register R_0 or R_1. The result is placed in the accumulator. No flags are affected.

The example is ORA A, @R_0.

ORL A, #data (Logical OR immediate 8-bit data with accumulator)

A ← A ∨ #8-bit data.

Machine cycles: 1, States: 12, Flags: No flags are affected, Immediate Addressing, Two-byte instructions

In this instruction, 8-bit data is ORed with the content of the accumulator and the result is placed in the accumulator. No flags are affected. The example is ORA A, #45H.

ORL direct, A (Logical OR accumulator to direct byte)

[direct] ← [direct] ∨ A.

Machine cycles: 1, States: 12, Flags: No flags are affected, Two-byte instructions

The contents of the accumulator are logically ORed with the contents of the memory location, whose address is specified by the 8 bit direct address and the result is stored in the 8 bit direct address. No flags are affected. The example is ORA direct, A.

ORL direct, #data (Logical OR immediate data to direct byte)

[direct] ← [direct] ∨ #data.

Machine cycles: 2, States: 24, Flags: No flags are affected, Immediate Addressing, Three-byte instructions

The 8-bit immediate data is logically ORed with the contents of the memory location, whose address is specified by the 8 bit direct address and the result is placed in the 8 bit direct address. No flags are affected. The example is ORA direct, #data.

XRL performs the bitwise logical Exclusive-OR operation between the indicated variables in instruction and, store the results in the destination. No flags are affected. Different addressing mode combinations are possible for this instruction. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing. If the destination is a direct address, the source can be the accumulator or immediate data.

XRL A, Rn (EXCLUSIVE-OR register with accumulator)

A ← A ⊕ R*n*

Machine cycles: 1, States: 12, Flags: No flags are affected, Register Addressing, One-byte instructions

The contents of the accumulator are Exclusive ORed with the contents of the register R*n* (R_0–R_7) and the result is placed in the accumulator. No flags are affected. Example: XRL A, R7 .

XRL A, direct (Exclusive-OR direct byte to accumulator)

A ← A ⊕ [direct]

Machine cycles: 1, States: 12, Flags: No flags are affected, Direct Addressing, Two-byte instructions

The contents of the accumulator are Exclusive ORed with the contents of the memory location, which is specified by 8-bit direct address and the result is placed in the accumulator. No flags are affected, e.g., XRL A, direct.

XRL A, @Ri (Exclusive-OR indirect RAM to accumulator)

A ← A ⊕ [R*i*]

Machine cycles: 1, States: 12, Flags: No flags are affected, Register indirect Addressing, One-byte instructions

The contents of the accumulator are Exclusive ORed with the contents of the memory location, which is specified by the register R*i* (R_0 or R_1) and the result is placed in the accumulator. No flags are affected, e.g., XRL A, @R_0.

XRL A, #data (Exclusive-OR immediate data to accumulator)

A ← A ⊕ #data

Machine cycles: 1, States: 12, Flags: No flags are affected, Immediate Addressing, Two-byte instructions

The contents of the accumulator are Exclusive ORed with the 8-bit data. The result is stored in the accumulator. No flags are affected., e.g., XRL A, #78H.

XRL direct, A (Exclusive-OR accumulator to direct byte)

[direct] ← [direct] ⊕ A

Machine cycles: 1, States: 12, Flags: No flags are affected, One-byte instructions

The contents of the accumulator are Exclusive ORed with the contents of the memory location, which is specified by 8-bit direct address and the result is placed in the same address. No flags are affected, e.g., XRL direct, A.

XRL direct, #data (Exclusive-OR immediate data to direct byte)

[direct] ← [direct] ⊕ #data

Machine cycles: 2, States: 24, Flags: No flags are affected, Three-byte instructions

The contents of the 8-bit direct address memory location are Exclusive ORed with 8-bit immediate data and the result is placed in the 8-bit direct memory address. No flags are affected, e.g., XRL direct, #data.

CLR A (Clear accumulator)

A ← 0, Machine cycles: 1, States 12, One-byte instruction.

The accumulator is cleared (all bits reset to zero). No flags are affected.

CPL A (Complement accumulator)

A ← $\overline{A}$, Machine cycles: 1, States 12, One-byte instruction

Each bit of the accumulator is logically complemented, i.e., one's complement. Bits which previously contained a one are changed to a zero and vice-versa. No flags are affected.

RL A (Rotate accumulator left)

An+1 ← An, A_0 ← A_7

Machine cycles: 1, States: 12, Flags: No Flags are affected, Implicit Addressing, One-byte instructions

The eight bits in the accumulator are rotated one bit to the left. Bit 7 is rotated into the bit 0 position as shown in Fig. 8.5. No flags are affected.

For Example, RL A. The accumulator holds the value C5H (11000101) and after execution of RL A instruction, the accumulator holding the value 8BH (10001011) with the carry unaffected.

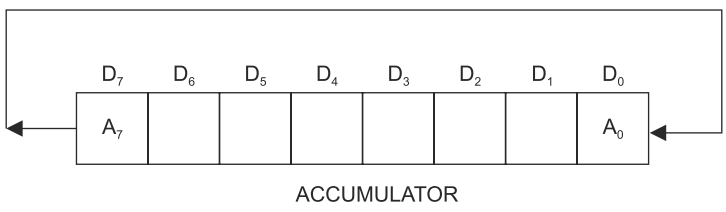


Fig. 8.5 Diagram for RL A

RLC A (Rotate accumulator left through the carry)

An+1 ← An, A_0 ← C, C ← A_7

Machine cycles: 1, States: 12, Flags: CS, Implicit Addressing, One-byte instructions

The eight bits in the accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag and the original state of the carry flag moves into the bit 0 position. Each bit of the accumulator is rotated left by one bit. The seventh bit of the accumulator is placed in the position of carry and carry flag moves to A_0 as shown in Fig. 8.6. No other flags are affected, e.g., RLC A. Assume the accumulator holds the value C5H (11000101), and the carry is zero. After execution of RLC A, the Accumulator holds the value 8AH (10001010) with the carry set.

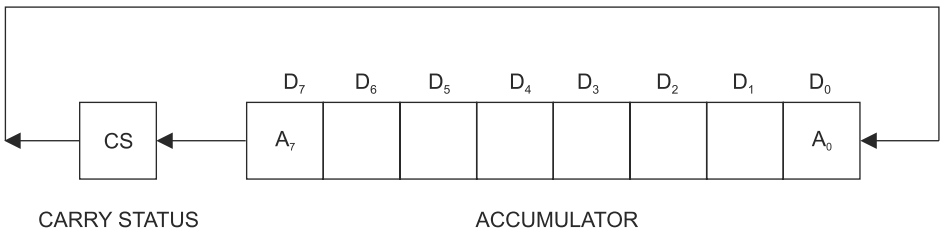


Fig. 8.6 Diagram for RLC A

RR A (Rotate accumulator right)

$An \leftarrow An+1,\ A_7 \leftarrow A_0$

Machine cycles: 1, States: 4, Flags: No flags are affected, Implicit Addressing, One-byte instructions

The eight bits in the accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 positions. Each binary bit of the accumulator is shifted right by one position as depicted in Fig. 8.7. No flags are affected, e.g., RR A. If the Accumulator holds the value C5H (11000101), after execution RR A instruction, the Accumulator holds the value E2H (11100010) with the carry unaffected.

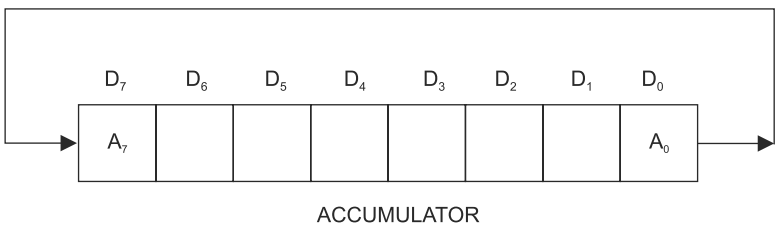


Fig. 8.7 Diagram for RR A

RRC A (Rotate Accumulator right through the carry)

$A_n \leftarrow A_{n+1}, A_7 \leftarrow C, C \leftarrow A_0$

Machine cycles: 1, States: 4, Flags: CS, Implicit Addressing, One-byte instructions

The eight bits in the Accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original state of the carry flag moves into the bit 7 position as shown in Fig. 8.8. No other flags are affected, e.g., RRC A. When the accumulator holds the value C5H (11000101) and the carry is zero, after execution of RRC A instruction, the accumulator holds the value 62H (01100010) with the carry set.

SWAP (Swap nibbles within the accumulator)

$(A_{3-0}) \leftrightarrow (A_{7-4})$

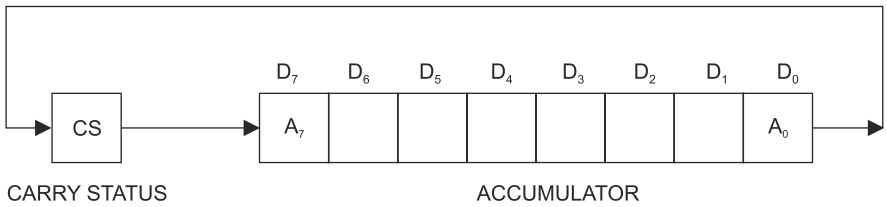


Fig. 8.8 Diagram for RRC A

Machine cycles: 1, States 4, Flags: No flags are affected,

SWAP A instruction interchanges the low-order and high-order nibbles of the Accumulator (bits 3-0 and bits 7-4). The SWAP operation can also be thought of as a four-bit rotate instruction. No flags are affected, e.g., SWAP A. If the accumulator holds the value C5H (11000101), after execution of SWAP A instruction, the accumulator holds the value 5CH (01011100).

Table 8.8 shows the 8051 logical instruction set summary

Table 8.8 8051 Logical instruction set summary

Opcode	Operand	Functions	Clock cycle	Number of bytes	Instruction code							
ANL	A, Rn	AND register to accumulator	12	1	0	1	0	1	1	r	r	r
ANL	A, direct	AND direct byte to accumulator	12	2	0	1	0	1	0	1	0	1
ANL	A,@Ri	AND indirect RAM to accumulator	12	1	0	1	0	1	0	1	1	i
ANL	A,#data	AND immediate data to accumulator	12	2	0	1	0	1	0	1	0	0
ANL	direct, A	AND accumulator to direct byte	12	2	0	1	0	1	0	0	1	0
ANL	direct, #data	AND immediate data to direct byte	24	3	0	1	0	1	0	0	1	1
ORL	A, Rn	OR register to accumulator	12	1	0	1	0	0	1	r	r	r
ORL	A, direct	OR direct byte to accumulator	12	2	0	1	0	0	0	1	0	1
ORL	A, @ Ri	OR indirect RAM to accumulator	12	1	0	1	0	0	0	1	1	i
ORL	A, #data	OR immediate data to accumulator	12	2	0	1	0	0	0	1	0	0
ORL	direct ,A	OR accumulator to direct byte	12	2	0	1	0	0	0	0	1	0
ORL	direct, #data	OR immediate data to direct byte	24	3	0	1	0	0	0	0	1	1
XRL	A, Rn	Exclusive-OR register to accumulator	12	1	0	1	1	0	1	r	r	r
XRL	A, direct	Exclusive-OR direct byte to accumulator	12	2	0	1	1	0	0	1	0	1
XRL	A, @ Ri	Exclusive-OR indirect RAM to accumulator	12	1	0	1	1	0	0	1	1	i
XRL	A, # data	Exclusive-OR immediate data to accumulator	12	2	0	1	1	0	0	1	0	0
XRL	direct,A	Exclusive-OR accumulator to direct byte	12	2	0	1	1	0	0	0	1	0
XRL	direct, #data	Exclusive-OR immediate data to direct byte	24	3	0	1	1	0	0	0	1	1
CLR	A	Clear accumulator	12	1	1	1	1	0	0	1	0	0

CPL	A	Complement accumulator	12	1	1	1	1	1	0	1	0	0
RL	A	Rotate accumulator left	12	1	0	0	1	0	0	0	1	1
RLC	A	Rotate accumulator left through the carry	12	1	0	0	1	1	0	0	1	1
RR	A	Rotate accumulator right	12	1	0	0	0	0	0	0	1	1
RRC	A	Rotate accumulator right through the carry	12	1	0	0	0	1	0	0	1	1
SWAP	A	Swap nibbles within the accumulator	12	1	1	1	0	0	0	1	0	0

8.3.3 Data Transfer Instructions

Data transfer instructions are used to transfer data between registers, register pairs, memory and registers, etc. The byte variable indicated by the second operand is copied into the location specified by the first operand. After execution of MOV <destination-byte>, <source-byte>, the source byte is not affected. No other register or flag is affected. This is the most flexible operation. Fifteen combinations of source and destination addressing modes are allowed. All data transfer instructions are described below:

MOV A, R_n (Move register to accumulator)

$A \leftarrow R_n$

Machine cycles: 1, States: 12, Flags none, Register Addressing mode, one-byte instruction

This instruction copies the contents of the source register into the accumulator but the contents of the source register are not changed. Flags and other registers are not affected. For example, MOV A, R_2.

MOV A, direct (Move direct byte to accumulator)

A ←[direct]

Machine cycles: 1, States: 12, Flag none, Direct Addressing, Two-byte instruction

The content of the memory location moves to accumulator. For example, the instruction MOV A, 44H will move the content 44H memory location to accumulator.

MOV A, @R_i (Move indirect RAM to accumulator)

$A \leftarrow [R_i]$

Machine cycles: 1, States: 12, Flag none, Register Indirect Addressing, one-byte instruction

The content of memory location whose address specified by register R_0 or R_1 moves to accumulator. For example, the instruction MOV A, @R_0 will move the content of the memory location specified by R_0 register to accumulator.

MOV A, #data (Move immediate data to accumulator)

A← #data

Machine cycles: 1, States: 12, Flags none, immediate addressing modes, two-byte instruction

The 8-bit data can be stored in the accumulator immediately. For example, the instruction MOV A, #44H moves 44H to accumulator.

MOV Rn, A (Move accumulator to register)

R*n* ← A

Machine cycles: 1, States: 12, Flags none, Register Addressing, One-byte instruction

The contents of accumulator will be stored in the register R*n* (R_0–R_7). For example, MOV R*n*, A.

MOV Rn, direct (Move accumulator to register)

Rn ← [direct]

Machine cycles: 2, States: 12, Flags none, Direct Addressing, Two-byte instruction

The content of 8 bit direct memory location will be stored in register R*n*. For example, MOV R_2, 22H. When this instruction is executed, the content of 22H memory location move to register R_2.

MOV Rn, #data (Move immediate data to register)

R*n* ← data

Machine cycles: 1, States: 12, Flags none, Immediate Addressing, two-byte instruction

The 8-bit immediate data will be stored in the register R*n*. For example, MOV R4, #67H. When this instruction is executed, 67H data move to the register R_4.

MOV direct, A (Move accumulator to direct byte)

[direct] ← A

Machine cycles: 1, States: 12, Flags none, Two-byte instruction

The content of accumulator will be copied in 8-bit direct address memory location. For example, MOV 25, A. After execution of this instruction, accumulator content move to 25H memory location.

MOV direct, Rn (Move register to direct byte)

[direct] ← R*n*

Machine cycles: 2, States: 12, Flags none, two-byte instruction

The content of Register R*n* (R_0–R_7) will be stored in memory location, which is specified by the 8-bit direct address. For example, MOV 45H, R7. If this instruction is executed, R_7 register content move to 25H memory location.

MOV direct, direct (Move direct byte to direct)

[direct] ← [direct]

Machine cycles: 2, States: 24, Flags none, three-byte instruction

The data will be stored in a 8-bit direct memory location from an 8-bit direct memory location. For example, MOV 23, 22H. The content of 22H memory location is copied to 23H memory location.

MOV direct, @Ri (Move indirect RAM to direct byte)

[direct] ← [R*i*]

Machine cycles: 3, States: 10, Flags none, Indirect Addressing, two-byte instruction

The content of internal RAM whose address is specified by the contents of the register R*i* (R_0 or R_1) will be stored in 8-bit direct address memory location. For example, MOV 44H, @R_0. After execution of this instruction, the content of the memory location specified by R_0 register will be stored in the 8-bit direct address memory location.

MOV direct, #data (Move immediate data to direct byte)

[direct] ← data

Machine cycles: 2, States: 24, Flags none, Immediate Addressing, Two-byte instruction

The 8-bit immediate data will be stored in the 8-bit direct memory location. For example, MOV 45H, #22H. In this instruction MOV 45H, #22H, 22H data move to 8-bit direct address memory location 45 H.

MOV @Ri, A (Move accumulator to indirect RAM)

[R*i*] ← A

Machine cycles: 1, States: 12, Flags none, One-byte instruction

The content of accumulator will be stored in memory location, which is specified by the contents of the R*i* (R_0 or R_1) register. For example, MOV @R_0, A. In this instruction Accumulator Content data move to memory location specified by R_0 register.

MOV @Ri, direct (Move direct byte to indirect RAM)

[R*i*] ←[direct]

Machine cycles: 2, States: 24, Flags none, Direct Addressing, two-byte instruction

The data of 8-bit direct memory location will be stored in the memory location, which is specified by the contents of the R*i* (R_1 or R_0) register, For example, MOV @R_0, 33H. When this instruction is executed, the content of 33H memory location move to memory location specified by R_0.

MOV @ Ri, #data (Move immediate data to indirect RAM)

[R*i*]←#data

Machine cycles: 1, States: 12, Flags none, Immediate Addressing, two-byte instruction

The 8-bit immediate data will be stored in memory location, which is specified by the contents of the *Ri* register. Example: MOV @R_1, #FFH. After execution of MOV @R_1, #FFH instruction, FFH data move to memory location specified by the R_1 register.

MOV DPTR, #data 16 (Load data pointer with a 16-bit constant)

DPTR← $data_{15\text{-}0}$

DPH← $data_{15\text{-}8}$, DPL ← $data_{7\text{-}0}$

Machine cycles: 3, States: 10, Flags none, Immediate Addressing, three-byte instruction

The data pointer register is loaded with the 16-bit constant indicated in the instruction. The 16-bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte and the third byte (DPL) holds the low-order byte. No flags are affected. This is the only instruction, which moves 16 bits of data at once. For example, MOV DPTR, #8000H. When this instruction is executed, load the value 8000H into the data pointer. Hence DPH will hold 80H and DPL will hold 00H.

The MOVC A, @A + <base-register> instructions load the Accumulator with a code byte from program memory. The address of the byte fetched is the sum of the unsigned eight-bit accumulator contents and the contents of a sixteen-bit base register. The base register may be either the data pointer or the PC. The PC is incremented before being added with the accumulator. As sixteen-bit addition is performed, a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected. The example of MOV C instructions are explained below:

MOVC A,@A+DPTR (Move code byte relative to DPTR to ACC)

A←[A+DPTR]

Machine cycles: 2, States: 12, Flags none, Index Addressing, one-byte instruction

The contents of memory location, which is specified by the contents of accumulator and the DPTR register, move to accumulator. For example, MOVC A, @A+DPTR.

MOVC A, @A+PC (Move code byte relative to PC to ACC)

PC←PC+1, A←[A + PC]

Machine cycles: 2, States: 24, Flags none, Index Addressing, one-byte instruction

The contents of memory location, which is specified by the contents of accumulator and the PC register, move to accumulator. For example, MOVC A, @A+PC.

The MOVX <destination-byte>, <source-byte> instruction is used to transfer data between accumulator and a byte of external memory. There are two types of instructions differing in whether they provide an eight bit or sixteen-bit in direct address to the external data RAM.

In the first case, the contents of R_0 or R_1 of the selected register bank provide an eight-bit address multiplexed with data on P_0. In the second case, the data pointer registers provides the sixteen bit address, P_2 outputs the content of DPH, i.e., high-order eight address bits, but P_0 multiplexes the low-order eight bits (DPL) with data. All types of MOVX instruction are explained in this section.

MOVX A, @Ri (Move external RAM (8-bit addr) to ACC)

A ← [R*i*]

Machine cycles: 2, States: 24, Flags none, Indirect Addressing, one-byte instruction

The contents of external RAM location (8-bit address), which is specified by the contents of R_0 or R_1, move to accumulator. For example, MOVX A, @R_0.

MOVX A, @DPTR (Move external RAM (16-bit addr) to ACC)

A ← [DPTR]

Machine cycles: 2, States: 24, Flags none, Indirect Addressing, One-byte instruction

The contents of external RAM location (16-bit address), which is specified by the contents of DPTR, move to accumulator. For example, MOVX A, @DPTR.

MOVX @Ri, A (Move ACC to external RAM (8-bit addr))

[R*i*] ← A

Machine cycles: 2, States: 24, Flags none, One-byte instruction

The contents of accumulator move to external RAM location (8-bit address), which is specified by the contents of R_0 or R_1. For example, MOVX @R_0, A.

MOVX @DPTR, A (Move ACC to external RAM (16-bit addr))

[DPTR] ← A

Machine cycles: 2, States: 24, Flags none, one-byte instruction

The contents of accumulator move to external RAM location (16-bit address), which is specified by the contents of DPTR. For example, MOVX @DPTR, A.

Table 8.9 shows the 8051 data-transfer instruction set summary.

Table 8.9 8051 Data-transfer instructions set summary

Opcode	Operand	Functions	Clock cycle	Number of bytes	Instruction code							
MOV	A, Rn	Move register to accumulator	12	1	1	1	1	0	1	r	r	r
MOV	A, direct	Move direct byte to accumulator	12	2	1	1	1	0	0	1	0	1
MOV	A, @Ri	Move indirect RAM to accumulator	12	1	1	1	1	0	0	1	1	i
MOV	A, #data	Move immediate data to accumulator	12	2	0	1	1	1	0	1	0	0
MOV	Rn, A	Move accumulator to register	12	1	1	1	1	1	1	r	r	r
MOV	Rn, direct	Move direct byte to register	24	2	1	0	1	0	1	r	r	r
MOV	Rn, #data	Move immediate data to register	12	2	0	1	1	1	1	r	r	r
MOV	direct, A	Move accumulator to direct byte	12	2	1	1	1	1	0	1	0	1
MOV	direct, Rn	Move register to direct byte	24	2	1	0	0	0	1	r	r	r

MOV	direct, direct	Move direct byte to direct	24	3	1	0	0	0	0	1	0	1
MOV	direct, @Ri	Move indirect RAM to direct byte	24	2	1	0	0	0	0	1	1	i
MOV	direct, #data	Move immediate data to direct byte	24	3	0	1	1	1	0	1	0	1
MOV	@Ri, A	Move accumulator to indirect RAM	12	1	1	1	1	1	0	1	1	i
MOV	@Ri, direct	Move direct byte to indirect RAM	24	2	1	0	1	0	0	1	1	i
MOV	@Ri, #data	Move immediate data to indirect RAM	12	2	0	1	1	1	0	1	1	i
MOV	DPTR, #data16	Load data pointer with a 16-bit constant	24	3	1	0	0	1	0	0	0	0
MOVC	A,@A +DPTR	Move code byte relative to DPTR to ACC	24	1	1	0	0	1	0	0	1	1
MOVC	A,@A+PC	Move code byte relative to PC to ACC	24	1	1	0	0	0	0	0	1	1
MOVX	A, @Ri	Move external RAM (8-bit addr) to ACC	24	1	1	1	1	0	0	0	1	i
MOVX	A,@ DPTR	Move external RAM (16-bit addr) to ACC	24	1	1	1	1	0	0	0	0	0
MOVX	@Ri, A	Move ACC to external RAM (8-bit addr)	24	1	1	1	1	1	0	0	1	i
MOVX	@DPTR, A	Move ACC to external RAM (16-bit addr)	24	1	1	1	1	1	0	0	0	0

Example 8.4 Write instructions to perform the following operations:

(i) Move the content of accumulator into R_0 register

(ii) Load immediate 8-bit data (FFH) into accumulator

(iii) Load Data pointer with 9000H

(iv) Move the content of accumulator to external RAM location 8000H

Sol.

(i) MOV A, R_0 ; Move the content of accumulator into R_0 register

(ii) MOV A, #FFH; Load immediate 8-bit data (FFH) into accumulator

(iii) MOV DPTR, #9000H ; Load Data pointer with 9000H

(iv) MOV DPTR, #8000 ;
MOVX @DPTR, A; Move the content of accumulator to external RAM location 8000H

8.3.4 Boolean Variable Manipulation

The 8051 controller contains a complete Boolean processor for single-bit operations. In these instructions, all bit accesses use direct addressing and bits may be set or cleared using single instruction. All Boolean instructions are explained below:

CLR C (Clear carry)

C←0

Machine cycles: 1, States: 12, Flags none, Direct Addressing mode, one byte instruction

This instruction clears the carry flag. No other flags are affected. For example, CLR C.

CLR bit (Clear direct bit)

bit ← 0

Machine cycles: 1, States: 12, Flags none, Direct Addressing mode, Two-byte instruction

The indicated bit is cleared (reset or zero). No other flags are affected. CLR can operate on the carry flag or any directly addressable bit, e.g., Port 1 has previously been written with 5DH (01011101B). The instruction, CLR P1.2 will leave the port set to 59H (01011001B).

SETB C (Clear carry)

C ← 1

Machine cycles: 1, States: 12, Flags none, Direct Addressing mode, one byte instruction

This instruction set the carry flag. No other flags are affected. For example, SETB C.

SETB bit (Clear direct bit)

bit ← 1

Machine cycles: 1, States: 12, Flags none, Direct Addressing mode, Two-byte instruction

SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly addressable bit. No other flags are affected, e.g., the carry flag is cleared. Output Port 1 has been written with the value 34H (00110100B). The instructions, SETB P1.0 will leave the carry flag set to 1 and change the data output on Port 1 to 35H (00110101B).

CPL C (Complement carry)

C ← $\overline{\text{C}}$

Machine cycles: 1, States: 12, Flags none, Direct Addressing mode, one byte instruction

This instruction complement the carry flag. No other flags are affected. For example, CPL C.

CPL bit (Complement bit)

bit ← $\overline{\text{bit}}$

Machine cycles: 1, States: 12, Flags none, Direct Addressing mode, one byte instruction

The bit variable specified is complemented. When a bit is one, it is changed to zero and vice-versa. No other flags are affected. CPL can operate on the carry or any directly addressable bit, e.g., Port 1 has previously been written with 5DH (01011101B). The instruction sequence, CPL P1.1 and CPL P1.2 will leave the port set to 5BH (01011011B).

ANL C, bit (AND direct bit to carry)

C ← C ∧ bit

Machine cycles: 2, States: 24, Flags none, Direct Addressing mode, two-byte instruction

This instruction performs logical AND operation between the source bit and the carry flag. No other flags are affected. For example, ANL C, ACC.7. AND operation between the accumulator bit 7 and the carry.

ANL C, /bit (Complement bit)

C ← C ∧ $\overline{\text{bit}}$

Machine cycles: 2, States: 24, Flags none, Direct Addressing mode, Two-byte instruction

The slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected. For example, ANL C, /OV ; AND with inverse of overflow flag.

ORL C, bit (OR direct bit to carry)

C ← C ∨ bit

Machine cycles: 2, States: 24, Flags none, Direct Addressing mode, two-byte instruction

This instruction performs logical-OR operation between source bit and the carry. No other flags are affected, e.g., ORL C, ACC.7 ; OR carry with the ACC. BIT 7.

ORL C, /bit (OR complement of direct bit to carry)

C ← C ∨ $\overline{bit}$

Machine cycles: 2, States: 24, Flags none, Direct Addressing mode, two-byte instruction

A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected, e.g., ORL C, /OV ;OR carry with the inverse of OV.

MOV C, bit (Move direct bit to carry)

C ← bit

Machine cycles: 1, States: 12, Flags none, Direct Addressing mode, two-byte instruction

This instruction is used to copy the Boolean variable indicated by the second operand into the location specified by the first operand. No other flags are affected. For example, MOV C, P3.3.

MOV bit, C (Move carry to direct bit)

bit ← C

Machine cycles: 2, States: 24, Flags none, Direct Addressing mode, Two-byte instruction

The Boolean variable indicated by the second operand must be copied into the location specified by the first operand. One of the operands is the carry flag and the other is any directly addressable bit. No other register or flag is affected, e.g., MOV P1.3, C Assume the carry flag is set and the data present at output Port 1 is 35H (0011 0101B). After execution of MOV P1.3, C the Port 1 change to 3DH (0011 1101B).

JC rel (Jump if carry is set)

PC ← PC + 2, If C = 1, then PC ← PC + rel

Machine cycles: 2 States: 24, Flags none, Two-byte instruction

When the carry flag is set, jump to the address indicated in instruction; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement to the PC, after incrementing the PC twice. No flags are affected, e.g., JC LABEL-1.

JNC rel (Jump if carry is not set)

PC ← PC + 2 , If C = 0, then PC ← PC + rel

Machine cycles: 2, States: 24, Flags none, Two-byte instruction

If the carry flag is a zero, jump to the address indicated in the instruction; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified, e.g., JNC LABEL-1.

JB bit, rel (Jump if direct bit is set)

PC ← PC + 3 , If bit = 1, then PC ← PC + rel

Machine cycles: 2 States: 24, Flags none, Three-byte instruction

If the indicated bit is '1', jump to the address indicated in the instruction; otherwise proceed with the next

instruction. The branch destination is computed by adding the signed relative-displacement to the PC, after incrementing the PC. The bit tested is not modified. No flags are affected, e.g., JB P1.2, LABEL-1.

JNB bit, rel (Jump if direct bit is not set)

PC←PC + 2 , If C = 1, then PC←PC + rel

Machine cycles: 2, States: 24, Flags none, Three-byte instruction

If the indicated bit is a zero, jump to the indicated address in the instruction; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement to the PC, after incrementing the PC. The bit tested is not modified. No flags are affected, e.g., JNB P1.3, LABEL-1.

JB C bit, rel (Jump if direct bit is set and clear bit)

PC←PC + 3 , If bit = 1, then bit←0, PC←PC + rel

Machine cycles: 2 States: 24, Flags none, Three-byte instruction

If the indicated bit is '1', jump to the address indicated in instruction; otherwise proceed with the next instruction. The bit will not be cleared if it is already a zero. The branch destination is computed by adding the signed relative-displacement to the PC, after incrementing the PC. No flags are affected, e.g., JBC ACC.3, LABEL-1.

Table 8.10 shows the 8051 Boolean instruction set summary

Table 8.10 8051 Boolean instruction set summary

Opcode	Operand	Functions	Clock cycle	Number of bytes	Instruction code							
CLR	C	Clear carry	12	1	1	1	0	0	0	0	1	1
CLR	bit	Clear direct bit	12	2	1	1	0	0	0	0	1	0
SETB	C	Set carry	12	1	1	1	0	1	0	0	1	1
SETB	bit	Set direct bit	12	2	1	1	0	1	0	0	1	0
CPL	C	Complement carry	12	1	1	0	1	1	0	0	1	1
CPL	bit	Complement direct bit	12	2	1	0	1	1	0	0	1	0
ANL	C, bit	AND direct bit to carry	24	2	1	0	0	0	0	0	1	0
ANL	C,/bit	AND complement of direct bit to carry	24	2	1	0	1	1	0	0	0	0
ORL	C, bit	OR direct bit to carry	24	2	0	1	1	1	0	0	1	0
ORL	C,/bit	OR complement of direct bit to carry	24	2	1	0	1	0	0	0	0	0
MOV	C,bit	Move direct bit to carry	12	2	1	0	1	0	0	0	1	0
MOV	bit,C	Move carry to direct bit	24	2	1	0	0	1	0	0	1	0
JC	rel	Jump if carry is set	24	2	0	1	0	0	0	0	0	0
JNC	rel	Jump if carry is not set	24	2	0	1	0	1	0	0	0	0
JB	Bit, rel	Jump if direct bit is set	24	3	0	0	1	0	0	0	0	0
JNB	Bit,rel	Jump if direct bit is not set	24	3	0	0	1	1	0	0	0	0
JBC	bit,rel	Jump if direct bit is set and clear bit	24	3	0	0	0	1	0	0	0	0

8.3.5 Branch Group

The branch group instructions are generally used to change the sequence of the program execution. There are two types of branch instructions 'namely' conditional and unconditional. The conditional branch instructions

transfer the program to the specified address when condition is satisfied only. The unconditional branch instructions transfer the program to the specified address unconditionally. All conditional and unconditional branch instructions are explained in this section.

ACALL 11-bit address (Absolute subroutine CALL)

$PC \leftarrow PC + 2$, $SP \leftarrow SP + 1$, $SP \leftarrow PC_{7\text{-}0}$,

$SP \leftarrow SP+1$, $SP \leftarrow PC_{15\text{-}8}$, $PC_{10\text{-}0} \leftarrow$ page address

Machine cycles: 2, States: 24, Flags: none, Two-bytes instruction

ACALL instruction unconditionally calls a subroutine located at the indicated address. This instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the stack pointer twice. The destination address can be obtained by successively concatenating the five high-order bits of the incremented PC, opcode bits 7–5, and the second byte of the instruction. This instruction can be used to call a subroutine within the same 2K block of the program memory. The first byte of the instruction following ACALL. No flags are affected, e.g., ACALL address 11.

LCALL 16-bit address (Long subroutine CALL)

$PC \leftarrow PC + 3$, $SP \leftarrow SP + 1$, $[SP] \leftarrow PC_{7\text{-}0}$,

$SP \leftarrow SP + 1$, $[SP] \leftarrow PC_{15\text{-}8}$, $PC \leftarrow address_{15\text{-}0}$

Machine cycles: 2, States: 24, Flags: none, Three-byte instruction

LCALL instruction calls a subroutine located at the indicated address. This instruction adds three to the program counter to generate the address of the next instruction and then push the 16-bit result onto the stack (low byte first), and increment the stack pointer by two. Then high-order and low-order bytes of the PC are loaded with the second and third bytes of the LCALL instruction. Therefore the subroutine may start anywhere in the full 64K-byte program memory address space. No flags are affected, e.g., LCALL 2000.

RET (Return from subroutine)

$PC_{15\text{-}8} \leftarrow [SP]$, $SP \leftarrow SP\text{-}1$, $PC_{7\text{-}0} \leftarrow [SP]$, $SP \leftarrow SP\text{-}1$

Machine cycles: 2, States: 24, Flags: none, One-byte instruction

The RET instruction pops the high-order and low-order bytes of the PC successively from the stack, and decrement the stack pointer by two. Generally, this instruction immediately follows ACALL or LCALL. No flags are affected, e.g., RET.

RETI (Return from interrupt)

$PC_{15\text{-}8} \leftarrow SP$

$SP \leftarrow SP\text{-}1$, $PC_{7\text{-}0} \leftarrow SP$

$SP \leftarrow SP\text{-}1$

Machine cycles: 2, States: 24, Flags: none, One-byte instruction

RETI instruction pops the high-order and low-order bytes of the PC successively from the stack, and restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. The Stack Pointer is decremented by two. No other registers are affected and the PSW is not automatically restored to its pre-interrupt status. Usually, this instruction is executed immediately after the point at which the interrupt request is detected. If a lower- or same-level interrupt has been pending when the RETI instruction is executed, some instructions will be executed before processing the pending interrupt, e.g., RETI.

AJMP 11-bit address (Absolute jump)

$PC \leftarrow PC + 2$, $PC_{10\text{-}0} \leftarrow$ page address

Machine cycles: 2, States: 24, Flags: none, Two-byte instruction

AJMP instruction transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC after incrementing the PC by 2, opcode bits 7-5, and the second byte of the instruction. The destination address must be within the same 2K block of program memory, e.g., AJMP 11-bit address.

LJMP 16-bit address (Long Jump)

PC ← $addr_{15-0}$

Machine cycles: 2, States: 24, Flags: none, Three-byte instruction.

LJMP instruction is an unconditional jump to the indicated address, by loading the high-order and low-order bytes of the PC with the second and third instruction bytes. The destination address will be anywhere in the full 64K program memory address space. No flags are affected, e.g., LJMP 4000H.

SJMP rel (Short Jump)

PC ← PC + 2, PC ← PC + rel

Machine cycles: 2, States: 24, Flags: none, Two-byte instruction.

If SJMP rel instruction is executed, program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC by 2. The range of destinations is from 128 bytes preceding this instruction to 127 bytes following it, e.g., SJMP 70H.

JMP @A+DPTR (Jump indirect relative to the DPTR)

PC ← A + DPTR

Machine cycles: 2, States: 24, Flags: none, One-byte instruction.

The eight-bit unsigned contents of the accumulator is added with the sixteen-bit data pointer, and load the result to the program counter. This will be the address for subsequent instruction fetches. No flags are affected, e.g., MOV DPTR,#8000H; JMP @A+DPTR. If the Accumulator is equal to 04H, execution will jump to label 8004H memory location.

JZ rel (Jump if accumulator is zero)

PC ← PC + 2, If A = 0, then PC ← PC + rel

Machine cycles: 2, States: 24, Flags: none, Two-byte instruction.

If all bits of the Accumulator are zero, jump to the indicated address; otherwise proceed with the next instruction. The branch destination address is computed by adding the signed relative-displacement to the PC, after incrementing the PC by 2. The accumulator is not modified. No flags are affected, e.g., DEC A ; JZ LABEL2. Assume the accumulator holds 01H. After execution of above instructions, the accumulator will change to 00H and cause jump to the label LABEL2.

JNZ rel (Jump if accumulator is not zero)

PC ← PC + 2, If A ≠ 0, then PC ← PC + rel

Machine cycles: 2, States: 24, Flags: none, Two-byte instruction.

If any bit of the accumulator is '1', jump to the indicated address; otherwise proceed with the next instruction. The branch destination address is computed by adding the signed relative-displacement to the PC, after incrementing the PC by 2. The accumulator is not modified. No flags are affected, e.g., INC A ; JNZ LABEL2. Assume the accumulator holds 00H. After execution of above instructions the accumulator will set to 01H and continue at label LABEL2.

CJNE <dest-byte, <src-byte>, rel instruction is used to compare the magnitudes of the first two operands, and branches if their values are not equal. The branch destination address is computed by adding the signed relative-displacement to the PC, after incrementing the PC. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. All addressing mode combinations of CJNE instructions are explained below:

CJNE A, direct, rel (Compare direct byte to ACC and jump if not equal)

PC ← PC + 3, If A< > direct, then PC ← PC + relative offset; If A<direct, then C ← 1, Else C ← 0.

Machine cycles: 2, States: 24, Flags: none, Three-byte instruction.

Compare the content of data which is specified by direct memory location and accumulator, thereafter jump to destination address if not equal. The destination address is computed by addition of PC and relative offset address after incrementing PC by 3.

CJNE A, #data, rel (Compare immediate to ACC and jump if not equal)

PC ← PC + 3, If A< > data, then PC ← PC + relative offset; If A<data, then C ← 1, Else C ← 0

Machine cycles: 2, States: 24, Flags: none, Three-byte instruction.

Compare the contents of accumulator with 8-bit immediate data, thereafter jump to destination address if not equal. The destination address is computed by addition of PC and relative offset address after incrementing PC by 3.

CJNE Rn, #data, rel (Compare immediate to register and jump if not equal)

PC ← PC + 3, If R*n*< > data, then PC ← PC + relative offset; If R*n* < data, then C ← 1, Else C ← 0

Machine cycles: 2, States: 24, Flags: none, Three-byte instruction.

Compare the contents of register R*n* with 8-bit immediate data; thereafter jump to destination address if not equal. The destination address is computed by addition of PC and relative offset address after incrementing PC by 3.

CJNE @Ri, #data, rel (Compare immediate to indirect and jump if not equal)

PC ← PC + 3, If [R*i*]< > data, then PC ← PC + relative offset; If R*i* < data, then C ← 1, Else C ← 0

Machine cycles: 2, States: 24, Flags: none, Three-byte instruction.

Compare the content of memory location which is specified by R_0 or R_1 and 8-bit immediate data, thereafter jump to destination address if not equal. The destination address is computed by addition of PC and relative offset address after incrementing PC by 3.

DJNZ <byte>, <rel-addr> instruction decrements the first operand by 1 and jump to the address indicated by the second operand if the resulting value is not zero. An original value of 00H will underflow to FFH. No flags are affected. The branch destination address will be computed by adding the signed relative-displacement value to the PC, after incrementing the PC. The location of first operand may be a register or directly addressed byte. Two types of DJNZ instructions are explained below:

DJNZ Rn, rel (Decrement register and jump if not zero)

PC ← PC + 2, R*n* ← R*n*-1, If R*n* > 0 or R*n* < 0, then PC ← PC + rel

Machine cycles: 2, States: 24, Flags: none, Three-bytes instruction

Decrements the contents of register R*n* by 1 and jump to the address indicated by the instruction if the resulting value is not zero. The branch destination address would be computed by adding the signed relative-displacement value to the PC, after incrementing the PC by two, e.g., DJNZ R_2, 8-bit offset address.

DJNZ direct, rel (Decrement direct byte and jump if not zero)

PC←PC + 2, direct←direct-1, If direct > 0 or direct < 0, then PC←PC + rel

Machine cycles: 2, States: 24, Flags: none, Three-byte instruction

Decrements the contents of memory location which is specified by direct address, by 1 and jump to the address indicated by the instruction if the resulting value is not zero. The branch destination would be computed by adding the signed relative-displacement value to the PC, after incrementing the PC by two, e.g., DJNZ 40, 8-bit offset address.

NOP (No operation)

PC←PC+1

Machine cycles: 1, States: 12, Flags: none, One-byte instruction

Execution continues at the following instruction. Other than the program counter PC, no registers or flags are affected, e.g., NOP.

Table 8.11 shows the 8051 Branch group instruction set summary

Table 8.11 8051 Branch group instructions set summary

Opcode	Operand	Functions	Clock cycle	Number of bytes	Instruction code							
ACALL	addr11	Absolute subroutine call	24	2	a10	a9	a8	1	0	0	0	1
LCALL	addr16	Long subroutine call	24	3	0	0	0	1	0	0	1	0
RET		Return from subroutine	24	1	0	0	1	0	0	0	1	0
RETI		Return from interrupt	24	1	0	0	1	1	0	0	1	0
AJMP	addr11	Absolute jump	24	2	a10	a9	a8	0	0	0	0	1
LJMP	addr16	Long jump	24	3	0	0	0	0	0	0	1	0
SJMP	rel	Short jump (relative addr)	24	2	1	0	0	0	0	0	0	0
JMP	@A+ DPTR	Jump indirect relative to the DPTR	24	1	0	1	1	1	0	0	1	1
JZ	rel	Jump if accumulator is zero	24	2	0	1	1	0	0	0	0	0
JNZ	rel	Jump if accumulator is not zero	24	2	0	1	1	1	0	0	0	0
CJNE	A, direct, rel	Compare direct byte to ACC and jump if not equal	24	3	1	0	1	1	0	1	0	1
CJNE	A, #data, rel	Compare immediate to ACC 24 and jump if not equal	3	1	0	1	1	0	1	0	0	
CJNE	RN, #data, rel	Compare immediate to register and jump if not equal	24	3	1	0	1	1	1	r	r	r
CJNE	@Ri, # data, rel	Compare immediate to indirect and jump if not equal	24	3	1	0	1	1	0	1	1	i
DJNZ	Rn, rel	Decrement register and jump if not zero	24	2	1	1	0	1	1	r	r	r
DJNZ	direct,rel	Decrement direct byte and jump if not zero	24	3	1	1	0	1	0	1	0	1
NOP		No operation	12	1	0	0	0	0	0	0	0	0

8.3.6 PUSH, POP and EXCHANGE Instructions

The PUSH and POP instructions are used to manipulate stack related operations. All stack and exchange instructions discussed as follows:

PUSH direct (Push direct byte onto stack)

SP ← SP+1; [SP] ← direct

Machine cycles 2, States: 24, Flags none, Two-bytes instruction

The Stack Pointer is incremented by one. The contents of the indicated variable is then copied into the internal RAM location addressed by the Stack Pointer. No flags are affected, e.g., PUSH DPL.

POP Direct (Pop direct byte from stack)

Direct ← [SP]; SP ← SP –1

Machine cycles: 2, States: 24, Flags: none, Two-bytes instruction

The contents of the internal RAM location addressed by the stack pointer are read, and the Stack Pointer is decremented by one. The value read is then transferred to the directly addressed byte indicated. No flags are affected, e.g., POP DPH and POP SP.

XCH A,<byte> loads the accumulator with the contents of the indicated variable, at the same time writing the original accumulator contents to the indicated variable. The source/destination operand can use register, direct, or register-indirect addressing. All types of XCH instructions are explained below:

XCH A, R_n (Exchange register with accumulator)

A ↔ R_n

Machine cycles: 1, States: 12, Flags: none, One byte instruction

Exchange the contents of specified register R*n* with accumulator, e.g., XCH A,R_3

XCH A, direct (Exchange direct byte with accumulator)

A ↔ direct

Machine cycles: 1, States: 12, Flags: none, One-byte instruction

Exchange the contents of memory location specified by direct address with accumulator, e.g, XCH A, 40H.

XCH A, @R_i (Exchange indirect RAM with accumulator)

A ↔ [R_i]

Machine cycles: 1, States: 12, Flags: none, One byte instruction.

Exchange the contents of RAM location which is specified by R_0 or R_1 with accumulator, e.g., XCH A, @R_0. Assume R_0 contains the address 40H, the internal RAM location 40H holds 25H and Accumulator holds 2FH. After execution of XCH A, @R_0, Accumulator contains 25H and internal memory location content is 2FH.

XCHD A, @Ri (Exchange low-order digit indirect RAM with ACC)

$A_{3\text{-}0}$ ↔ $Ri_{3\text{-}0}$

Machine cycles: 1, States: 12, Flags: none, One-byte instruction.

XCHD instruction exchanges the low-order nibble of the Accumulator (bits 3-0) with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7-4) of each register are not affected. No flags are affected, e.g., XCHD A, @R_0.

Table 8.12 shows the PUSH, POP and exchange instruction set summary

Table 8.12 8051 12 PUSH, POP and Exchange instructions set summary

Opcode	Operand	Functions	Clock cycle	Number of bytes	Instruction code							
PUSH	direct	Push direct byte onto stack	24	2	1	1	0	0	0	0	0	0
POP	direct	Pop direct byte from stack	24	2	1	1	0	1	0	0	0	0
XCH	A, Rn	Exchange register with accumulator	12	1	1	1	0	0	1	r	r	r
XCH	A, direct	Exchange direct byte with accumulator	12	2	1	1	0	0	0	1	0	1
XCH	A, @ Ri	Exchange indirect RAM with accumulator	12	1	1	1	0	0	0	1	1	i
XCHD	A, @ Ri	Exchange low-order digit indirect RAM with ACC	12	1	1	1	0	1	0	1	1	i

8.4 SIMPLE EXAMPLES IN ASSEMBLY LANGUAGE PROGRAMS OF 8051 MICROCONTROLLER

Example 8.4.1 Store 8-bit immediate data (65H) into accumulator.

Mnemonics	Opcode	Comments
MOV	A, #65H	Store 65H into accumulator immediately

Example 8.4.2 Load 42H and 55H in Registers R_0 and R_1 respectively.

Mnemonics	Opcode	Comments
MOV	R_0, #42H	Load 42H in R_0 register
MOV	R_1, #55H	Load 55H in R_1 register

Example 8.4.3 Place the contents of external memory location 8000H into accumulator.

Mnemonics	Opcode	Comments
MOV	DPTR, #8000H	Load 8000H in data pointer register immediately
MOVX	A, @DPTR	Copy the content of external memory location 8000H into accumulator

Example 8.4.4 Load 45H in external memory location 8000 H.

Mnemonics	Opcode	Comments
MOV	DPTR, #8000H	Load 8000H in data pointer register immediately
MOV	A, #45H	Load 45H into accumulator
MOVX	@DPTR,A	Copy the content of Accumulator (45H) into external memory location 8000H

Example 8.4.5 Write program instructions to load a byte in memory location 9000H and increment the contents of the memory location.

Mnemonics	Opcode	Comments
MOV	DPTR, #9000H	Load 9000H in data pointer register immediately
MOV	A, #48H	Load 48H into accumulator
MOVX	@DPTR, A	Copy the content of Accumulator 48H into external memory location 9000H

INC	A	Increment accumulator
MOVX	@DPTR, A	Load 49H, i.e., the content of Accumulator into external memory location 9000H

Example 8.4.6 Store 01H, 02H, 03H and 04H in register R_0, R_1, R_2 and R_3 respectively and exchange data stored in Reg. R_0 with R_1 and data in Reg. R_2 with R_3.

Mnemonics	Opcode	Comments
MOV	R0, #01H	Load 01H in R_0 register immediately.
MOV	R1, #02H	Load 02H in R_1 register immediately.
MOV	R2, #03H	Load 03H in R_2 register immediately.
MOV	R3, #04H	Load 04H in R_3 register immediately.
MOV	A, R0H	Copy the content of R_0 into accumulator.
XCH	A, R1	The content of accumulator and register R_1 are exchanged.
MOV	R0, A	The content of accumulator into R_0 register.
MOV	A, R2	Copy the content of R_2 into accumulator.
XCH	A, R3	The content of accumulator and register R_3 are exchanged.
MOV	R2, A	The content of accumulator into R_2 register.

8.5 ASSEMBLY-LANGUAGE PROGRAMS

8.5.1 Addition of Two 8-bit Numbers and Sum is 8-bit

Add 49 H and 56 H. The 1st number 49 H is in the external memory location 9001H. The 2nd number 56 H is in the external memory location 9002 H. The result is to be stored in the external memory location 9003H.

Memory address	Machine Codes	Mnemonics	Operands	Comments
8000	90 90 01	MOV	DPTR, #9001	Load 16-bit constant 9001 into DPTR
8003	E0	MOVX	A, @DPTR	Move the content of external memory 9001 into accumulator
8004	F5 0B	MOV	B, A	Move accumulator to B
8006	A3	INC	DPTR	Increment DPTR
8007	E0	MOVX	A, @DPTR	Move second data into Accumulator
8008	25 0B	ADD	A, B	Add B register with accumulator
800A	A3	INC	DPTR	Increment DPTR
800B	F0	MOVX	@DPTR, A	Store result into 9003H
800C	02 00 00	LJMP	0000	

DATA

9001–49 H

9002–56 H

RESULT

9003–9F H. The sum is stored in the memory location 9003 H

8.5.2 Two Data 24H and 23H are Stored in RAM Locations 40H and 41H. Write a Program to find Sum and Store at 42H

Memory address	Machine Codes	Mnemonics	Operands	Comments
8000	78 40	MOV	R_0, #40H	Load 40H in R_0 register.
8002	A6 24	MOV	@R_0, #24H	Store 24H in 40H memory location.
8004	E6	MOV	A, @R_0	Content of 40H location in accumulator.
8005	08	INC	R_0	Increment R_0.
8006	76 23	MOV	@R_0, #23H	Load 23H into 41H memory location.
8008	26	ADD	A, @R_0	Content of 41H location is add with accumulator.
8009	08	INC	R_0	Increment R_0.
800A	F6	MOV	@R_0, A	Move the content of accumulator into 42H memory location.
800B	02 00 00	LJMP	0000	

8.5.3 Addition of Ten 8-bit Numbers and Sum is 16 bit

Assume ten 8-bit numbers are stored in the internal RAM locations from 31H to 3A. After addition MSD will be stored in R_2 and LSD will be in R_3.

PROGRAM

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	78 31		MOV	R_0, #31	Load 31H in R_0 register.
8002	79 0A		MOV	R_1, #0A	The no. of data in R_1 register.
8004	E4		CLR	A	Clear accumulator and A becomes 00H.
8005	FA		MOV	R_2, A	Move the accumulator content into R_2 register and R_2 becomes OOH.
8006	E6		MOV	A, @R_0	Move the content of internal RAM location into accumulator.
8007	08	LOOP	INC	R_0	Increment R_0 register to read next data.
8008	26		ADD	A, @R_0	Add next data with Accumulator.
8009	50 01		JNC	Level_1	Jump no carry to Level-1.
8008	0A		INC	R_2	Increment R_2 register.
800C	D9 F9	Level_1	DJNZ	R_1, LOOP	Repeat until R_1 becomes 0.
800E	FB		MOV	R_3, A	Move the accumulator content into R_3 register.
800F	02 00 00		LJMP	00	

8.5.4 Addition of Ten 8-bit Numbers Stored in the External RAM Locations Starting from 8000H. Sum is 16 bit and Result is Stored in Memory Locations 8100H and 8101H

PROGRAM

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	90 80 00		MOV	DPTR, #8000H	Load 8000H in data pointer register.
8003	79 0A		MOV	R_1, #0A	The no. of data in R1 register.
8005	E_4		CLR	A	Clear accumulator and A becomes 00H.
8006	FA		MOV	R_2, A	Initialize R_2 register.
8007	E_0		MOVX	A, @DPTR	Load first data in accumulator.
8008	FB	LOOP	MOV	R_3, A	Move data from A to R_3.
8009	A_3		INC	DPTR	Increment DPTR register to MOVE
800A	E_0		MOVX	A, @DPTR	next data in accumulator.
800B	2B		ADD	A, R_3	Add R_3 register with Accumulator.
800C	50 01		JNC	Level_1	Jump no carry to Level_1.
800E	0A		INC	R_2	Increment R_2 register.
800F	D_9 F_7	Level_1	DJNZ	R_1, LOOP	Repeat until R_1 becomes 00H.
8011	90 81 00		MOV	DPTR, #8100	Load 8100H in data pointer register.
8014	F_0		MOVX	@DPTR, A	Move ACC to external memory location 8100H.
8015	A_3		INC	DPTR	Increment DPTR.
8016	E_3		MOV	A, R_2	Move R_2 register to ACC.
8017	F_0		MOVX	@DPTR, A	Move ACC to external memory location 8101H.
8018	02 00 00		LJMP	00	

8.5.5 Addition of Two 16-bit Numbers and Sum is 16-bit

Assume the first 16-bit number is stored in the internal memory 40H and 41H. The second 16-bit number is in the internal memory location 42H and 43H. The results are to be stored in 40H and 41H.

Memory address	Machine Codes	Mnemonics	Operands	Comments
8000	E_5 40	MOV	A, 40H	Move content of internal RAM 40H into accumulator.
8002	A_8 42	MOV	R_0, 42H	Move content of internal RAM 42H into R_0 register.
8004	28	ADD	A, R_0	Add the content of R_0 with accumulator.
8005	F_5 40	MOV	40H, A	Copy the Addition of Least Significant Bytes into 40H internal RAM location.
8007	E_5 41	MOV	A, 41H	Move content of internal RAM 41H into accumulator.
8009	A_9 43	MOV	R_1, 43H	Copy the content of 43H memory location into R_1 register.

Contd.

Contd.

800B	39	ADDC	A, R_1	Add the content of R_1 with accumulator with carry.
800C	F_5 41	MOV	41H, A	Copy the Most Significant Bytes into 41H internal RAM location.
800E	02 00 00	LJMP	0000	

8.5.6 To Convert Packed BCD to two ASCII Numbers and save them in R_3 and R_2

Memory address	Machine Codes	Mnemonics	Operands	Comments
8000	74 45	MOV	A, #45H	Load 45H in accumulator.
8002	F8	MOV	R_0, A	Move content of accumulator in R_0 register.
8003	54 0F	ANL	A, #0FH	Mask the upper nibble A = 05H.
8005	44 30	ORL	A, #30H	Make it an ASCII, A = 35H.
8007	FA	MOV	R_2, A	Save ASCII equivalent into R_2 register.
8008	E8	MOV	A, R_0	Move content of R_0 register into accumulator.
8009	54 F0	ANL	A, #F0H	Mask lower nibble, A = 40H.
800B	03	RR	A	Rotate right accumulator.
800C	03	RR	A	Rotate right accumulator.
800D	03	RR	A	Rotate right accumulator .
800E	03	RR	A	Rotate right accumulator, A = 04H.
800F	44 30	ORL	A, #30H	Make it an ASCII, A = 34H.
8011	FB	MOV	R_3, A	Save ASCII equivalent into R_3 register.
8012	02 00 00	LJMP	00	

8.5.7 Subtraction of Two 8-Bit Numbers—One Number EFH is in the Accumulator and Other Number 45H in R_0 register. After subtraction, Result to be Stored in R_1 register

Memory address	Machine Codes	Mnemonics	Operands	Comments
8000	74 EF	MOV	A, #EFH	Load EFH into accumulator
8002	78 45	MOV	R_0, #45H	Load 45H into R_0 register
8004	98	SUBB	A, R_0	Subtract the content of R_0 from accumulator
8005	F9	MOV	R_1, A	Move accumulator content into R_1 register
8006	02 00 00	LJMP	0000	

8.5.8 One's Complement of an 8-bit Number

Assume 45H data is immediately loaded into accumulator. Complement accumulator and store result in 9001H memory location.

Memory address	Machine Codes	Mnemonics	Operands	Comments
8000	74 45	MOV	A, #45	Load 45H into accumulator.
8002	F4	CPL	A	Complement accumulator.
8003	90 90 01	MOV	DPTR, #9001	Load 9001H in DPTR.
8006	F0	MOVX	@DPTR,A	Store accumulator content in 9001H memory location.
8007	02 00 00	LJMP	00	

8.5.9 One's Complement of an 16-bit Number

Assume a 16-bit number is stored in 40H and 41H. Find 1's complement and store in 42H and 43H.

PROGRAM

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	78 40		MOV	R_0, #40H	Load 40H in R_0 register directly.
8002	E6		MOV	A, @R_0	Load LSB in accumulator.
8003	F4		CPL	A	Complement of accumulator.
8004	F5 42		MOV	42, A	Store in 42H memory location.
8006	08		INC	R_0	Increment R_0.
8007	E6		MOV	A, @R_0	Load MSB into accumulator.
8008	F4		CPL	A	Complement of accumulator.
8009	F5 43		MOV	43, A	Store in 42H memory location.

8.5.10 Two's Complement of an 8-bit Number and Store Result in 9001H Memory Location

Memory address	Machine Codes	Mnemonics	Operands	Comments
8000	C_3	CLR	C	Clear carry.
8001	74 4F	MOV	A, #4F	Move 4FH into accumulator.
8003	F_4	CPL	A	1's complement accumulator.
8004	24 01	ADD	A, #01	1's complement +1.
8006	90 90 01	MOV	DPTR, #9001	Load 9001 in DPTR.
8009	F_0	MOVX	@DPTR, A	Store result in 9001H memory location.
800A	02 00 00	LJMP	0000	

8.5.11 Shift an 8-bit Number Right by One Bit and Stored in 50H Memory Location

Memory address	Machine Codes	Mnemonics	Operands	Comments
8000	74 66	MOV	A, #66H	Load 66H into accumulator.
8002	03	RR	A	Rotate right by one bit.
8003	F5 50	MOV	50,A	Move accumulator content into 50H memory location.

8.5.12 SWAP 4-MSBs with 4-LSBs in Accumulator and Store in 9100H Memory Location

Memory address	Machine Codes	Mnemonics	Operands	Comments
8000	74 65	MOV	A, #65H	Load 65H into accumulator.
8002	C4	SWAP	A	Swap command interchanges the low and high order nibbles.
8003	90 91 00	MOV	DPTR, #9100	Load 9100 in DPTR.
8006	F_0	MOVX	@DPTR, A	Store accumulator content into 9100H memory location.
8007	02 00 00	LJMP	00	

8.5.13 Move One Section of Data in Internal Memory to Another Section in Internal Memory

Assume a section of data is stored in internal RAM starting from 40H. These data will be shifted to memory location starting from 80H.

PROGRAM

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	78 40		MOV	R_0, #40H	Store 40H in R_0 register immediately.
8002	79 80		MOV	R_1, #80H	Store 80H in R_1 register immediately.
8004	AA 20		MOV	R_2, #20H	Store no of data, 20H in R_2 register immediately.
8006	E_6	LOOP	MOV	A, @R_0	Move register indirect memory location into accumulator.
8007	F7		MOV	@R_1, A	Move accumulator content into register indirect memory location.
8008	08		INC	R_0	Increment R_0 register.
8009	09		INC	R_1	Increment R_1 register.
800A	DA FA		DJNZ	R_2, LOOP	Repeat until R_2 becomes zero.
800C	02 00 00		LJMP	00	

8.5.14 Finding the Largest Number from an Array of Numbers

Assume number of data is stored in 9000H and array of numbers is stored in external data memory starting from 9001H.

PROGRAM

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	80 90 00		MOV	DPTR, #9000H	Load 9000H in data pointer register.
8003	E_0		MOVX	A,@DPTR	Move content of data pointer into accumulator.
8004	F_8		MOV	R_0, A	Move accumulator content into R_0 register.
8005	A_3		INC	DPTR	Increment data pointer register.
8006	E_0		MOVX	A,@DPTR	Move content of data pointer into accumulator.
8007	F_9		MOV	R_1, A	Copy accumulator content into R_1 register.
8008	18		DEC	R_0	Decrement R_0 register.
8009	A_3	Loop	INC	DPTR	Increment data pointer register.
800A	E_0		MOVX	A,@DPTR	Move content of data pointer into accumulator.
800B	FA		MOV	R_2, A	Move accumulator content into R_2 register.
800C	99		SUBB	A, R_1	Subtract the content of R_1 register from accumulator.
800D	40 02		JC	Level	Jump no carry to shift Level.
800F	EA		MOV	A, R_2	Move R_2 register to accumulator.
8010	F_9		MOV	R_1, A	Move accumulator content into R_1 register.
8011	D_8 F_6	Level	DJNZ	R_0, Loop	If R_0 is not equal to zero, jump to Loop.
8013	E_9		MOV	A, R_1	Move R_1 to accumulator.
8014	90 91 00		MOV	DPTR, #9100H	Load 9100H in DPTR register.
8017	F_0		MOVX	@DPTR, A	Move content of accumulator into memory location.
8018	02 00 00		LJMP	0000	

8.5.15 Finding the Smallest Number from an Array of Numbers

Assume number of data is stored in 9000H and array of numbers is stored in external data memory starting from 9001H.

PROGRAM

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	90 90 00		MOV	DPTR, #9000	Load 9000H in Data pointer register.
8003	E_0		MOVX	A, @DPTR	Move content of data pointer into accumulator.
8004	F8		MOV	R_0, A	Move accumulator content into R_0 register.

Contd.

Contd.

8005	A_3		INC	DPTR	Increment data pointer register.
8006	E_0		MOVX	A,@DPTR	Move content of data pointer into accumulator.
8007	F_9		MOV	R_1, A	Copy accumulator content into R_1 register.
8008	18		DEC	R_0	Decrement R_0 register.
8009	A_3	LOOP	INC	DPTR	Increment data pointer register.
800A	E_0		MOVX	A,@DPTR	Move content of data pointer into accumulator.
800B	FA		MOV	R_2, A	Move accumulator content into R_0 register.
800C	99		SUBB	A, R_1	Subtract the content of R_1 register from accumulator.
800D	50 02		JNC	Level	Jump no carry to Level.
800F	EA		MOV	A, R_2	Move R_2 register to accumulator.
8010	F_9		MOV	R_1, A	Move accumulator content into R_1 register.
8011	D_8 F_6	Level	DJNZ	R_0, LOOP	If R_0 is not equal to zero, jump to LOOP.
8013	E_9		MOV	A, R_1	Move R_1 to accumulator .
8014	90 90 10		MOV	DPTR, #9100	Load 9100H in DPTR register.
8017	F_0		MOVX	@DPTR, A	Move content of accumulator into memory location.
8018	02 00 00		LJMP	0000	

8.5.16 Arranging a series of Numbers in Descending Order

Assume number of data is stored in R0 register and array of numbers is stored in external data memory starting from 9000H.

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	78 08		MOV	R_0,#0A	Number of data bytes, 0A is stored in R_0.
8002	18		DEC	R_0	Decrement R_0 by 1.
8003	90 90 00	LOOP1	MOV	DPTR, #9000	Load 9000H in DPTR.
8006	E_8		MOV	A, R_0	Content of R_0 in Accumulator.
8007	F_9		MOV	R_1, A	Content of Accumulator in R_1.
8008	E_0	LOOP2	MOVX	A, @DPTR	Move data into Accumulator.
8009	FA		MOV	R_2, A	Copy data in R_2.
800A	A_3		INC	DPTR	Increment DPTR.
800B	E_0		MOVX	A, @DPTR	Move next data in Accumulator.
800C	9A		SUBB	A, R_2	Compare above two data.
800D	50 08		JC	8017	Jump to LOOP 3 if carry flag is 0.
800F	E_0		MOVX	A, @DPTR	

Contd.

Contd.

8010	CA		XCH	A, R_2	Exchange data in Accumulator and R_2 if carry flag is 1.
8011	F_0		MOVX	@DPTR, A	Replace current memory data by Accumulator content.
8012	15 82		DEC	82	Decrement DPL by 1, DPL=DPL-1.
8014	EA		MOV	A, R_2	Move R_2 content into Accumulator.
8015	F_0		MOVX	@DPTR, A	Replace previous memory data by R_2.
8016	A_3		INC	DPTR	Increment DPTR .
8017	D_9 EF	LOOP3	DJNZ	R_1, 8008	Decrement R_1, if not zero, Jump to LOOP2.
8019	D_8 E_8		DJNZ	R_0, 8003	Decrement R_0, if not zero, Jump to LOOP1.
801B	02 00 00		LJMP	0000	

8.5.17 Arranging a Data Array in Ascending Order

Assume number of data is stored in R_0 register and array of numbers is stored in external data memory starting from 9000H.

PROGRAM

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	78 08		MOV	R_0, #08	Number of data bytes, 08 is stored in R0.
8002	18		DEC	R_0	Decrement R_0 by 1
8003	90 90 00	LOOP1	MOV	DPTR, #9000	Load 9000H in DPTR
8006	E_8		MOV	A, R_0	Content of R_0 in Accumulator
8007	F_9		MOV	R_1, A	Content of Accumulator in R_1
8008	E_0	LOOP2	MOVX	A,@DPTR	Move data into Accumulator
8009	FA		MOV	R_2, A	Copy data in R_2
800A	A_3		INC	DPTR	Increment DPTR
800B	E_0		MOVX	A,@DPTR	Move next data in Accumulator
800C	9A		SUBB	A, R_2	Compare above two data
800D	50 08		JNC	8017	Jump to LOOP 3 if carry flag is 0
800F	E_0		MOVX	A,@DPTR	
8010	CA		XCH	A,R_2	Exchange data in Accumulator and R_2 if carry flag is 1
8011	F_0		MOVX	@DPTR, A	Replace current memory data by Accumulator content
8012	15 82		DEC	82	Decrement DPL by1, DPL = DPL-1
8014	E_A		MOV	A, R_2	Move R_2 content into Accumulator
8015	F_0		MOVX	@DPTR,A	Replace previous memory data by R_2
8016	A_3		INC	DPTR	Increment DPTR
8017	D_9 EF	LOOP3	DJNZ	R_1, 8008	Decrement R_1, if not zero, Jump to LOOP2
8019	D_8 E_8		DJNZ	R_0, 8003	Decrement R_0, if not zero, Jump to LOOP1
801B	02 00 00		LJMP	0000	

8.5.18 Finding Square of a Number Using Look-Up Table

Memory address	Machine Codes	Mnemonics	Operands	Comments
8000	90 90 00	MOV	DPTR, #9000H	Load look-up table address 9000H.
8003	78 05	MOV	R_0, #05	Load the number 05 in R_0.
8005	E_8	MOV	A, R_0	Move data R_0 to Accumulator.
8006	93	MOVC	A, @A+DPTR	Get square of 05 from look up table and stored in Accumulator.
8007	90 91 00	MOV	DPTR, #9100	Load 9100 in DPTR.
800A	F_0	MOVX	@DPTR, A	Send result to 9100H memory location.
800B	02 00 00	LJMP	00	

ADDRESS	SQUARE
9000	00
9001	01
9002	04
9003	09
9004	16
9005	25
9006	36
9007	49
9008	64
9009	81

8.5.19 Program to Perform Multiplication of Two Numbers

Memory address	Machine Codes	Mnemonics	Operands	Comments
8000	74 24	MOV	A, #24	Load accumulator with 1st number (24H).
8002	75 0B 12	MOV	B, #12	Load register B with 2nd number (12H).
8005	A_4	MUL	AB	Multiplication accumulator with B register and store result in accumulator.
8006	F_8	MOV	R_0, A	Store LSB in R_0.
8007	A_9 0B	MOV	R_1, B	Store MSB in R_1.

8.5.20 Division of Two 8-bit Numbers

Memory address	Machine Codes	Mnemonics	Operands	Comments
8000	74 60	MOV	A, #60H	Load accumulator with 1st number (60H).
8002	75 0B 12	MOV	B, #12H	Load register B with 2nd number (12H).
8005	84	DIV	AB	Division accumulator with B register and store result in accumulator.
8006	F_8	MOV	R_0, A	Store result in R_0 register.
8007	A_9 0B	MOV	R_1, B	Store reminder in R_1 register.

8.5.21 Time Delay in Terms of Number of T States for the given Program

Labels	Mnemonics	Operands	Comments
DELAY	MOV	R_1, #FF	Outer loop counter = FFH = 256.
	MOV	R_0, #FF	Inner loop counter = FFH = 256.
LOOP	DJNZ	R_0, LOOP	Loop 256 times.
	DJNZ	R_1, LOOP	Loop 256 times.
	RET		Return.

Sol. The DJNZ instruction takes 24 clock periods or T states. Initially the inner loop is executed 256 times. After that R_1 is decremented by 1, again the inner loop is executed 256 times until R_1 is not zero. As the outer loop is also executed for 256 times, total T states = $24 \times 256 \times 256 + 24 \times 256$ T states required to execute DJNZ R_0, LOOP and DJNZ R_0, LOOP instructions. Twelve T states are required to execute MOV R_1, #FF and MOV R_0, #FF instructions. Hence, the total T states for the DELAY loop is $12 + 12 + 24 \times 256 \times 256 + 24 \times 256$ T states = 1579032 T states. If the microcontroller operating frequency is 12 MHZ, 24 T states is equal to 2μs. So that the time delay is equal to 1579032 T states = 131586 μs = 0.13s.

8.5.22 Program to read Port 0 and turn on the LEDs for one second and OFF for one second if Port 0 is not zero.

The content of port 0 states number of times LEDs ON. Assume that input switches are connected to port 0 and output LEDs are connected to port 1 of 8051.

Labels	Mnemonics	Operands	Comments
READ	MOV	A, P_0	Read the port 0 switches.
	JZ	READ	If no switch is on, jump to READ.
	MOV	R_0, A	Move the port 0 value in R_0 register.
ON	MOV	P_1, #0FF	Turn ON all LEDs connected to port 1.
	CALL	DELAY	Call Delay Loop.
	MOV	P_1, #00	Turn OFF all LEDs connected to port 1.
	CALL	DELAY	Call Delay Loop.
	DJNZ	R_0, ON	Decrement R_0 by 1, if R_0 is not zero jump to ON.

Contd.

Labels	Mnemonics	Operands	Comments
DELAY	MOV	R_1, #FF	Middle loop counter = FH = 256.
	MOV	R_2, #FF	Inner loop counter = FFH = 256.
	MOV	R_3, #08	Outer loop counter for 9 times.
LOOP	DJNZ	R_2, LOOP	Loop 256 times.
	DJNZ	R_1, LOOP	Loop 256 times.
	DJNZ	R_3, LOOP	Loop 8 times.
	RET		Return.

To execute the delay program given in Section 8.5.21, time required is about 0.13s. If the said program is repeated 8 times, about 1s delay will be generated. Therefore, a middle loop must be incorporated in the program as given above.

8.5.23 Program for A/D Converter Interfacing with 8051 Microcontroller

Figure 8.9 shows the ADC0804 which is interface with 8051 microcontroller. The clock input for ADC is taken from the crystal oscillator of the microcontroller. As frequency is very high, two flip-flops are used to divide the frequency by 4. The connection for start of conversion, $\overline{SC}$ and end of conversion, $\overline{EOC}$ signals are shown in Fig. 8.9. The step of A/D converter is as follows:

Step-1 The start of conversion, $\overline{SC}$ signal send to pin $\overline{WR}$ to start the conversion.

Step-2 INTR pin is connected with end of conversion $\overline{EOC}$ signal. Keep monitoring the INTR pin to check end of conversion. If INTR is high, keep polling until it becomes low.

Step-3 When the INTR is low, the A/D conversion is completed and the ADC0804 send a high-to-low pulse to the $\overline{RD}$ pin.

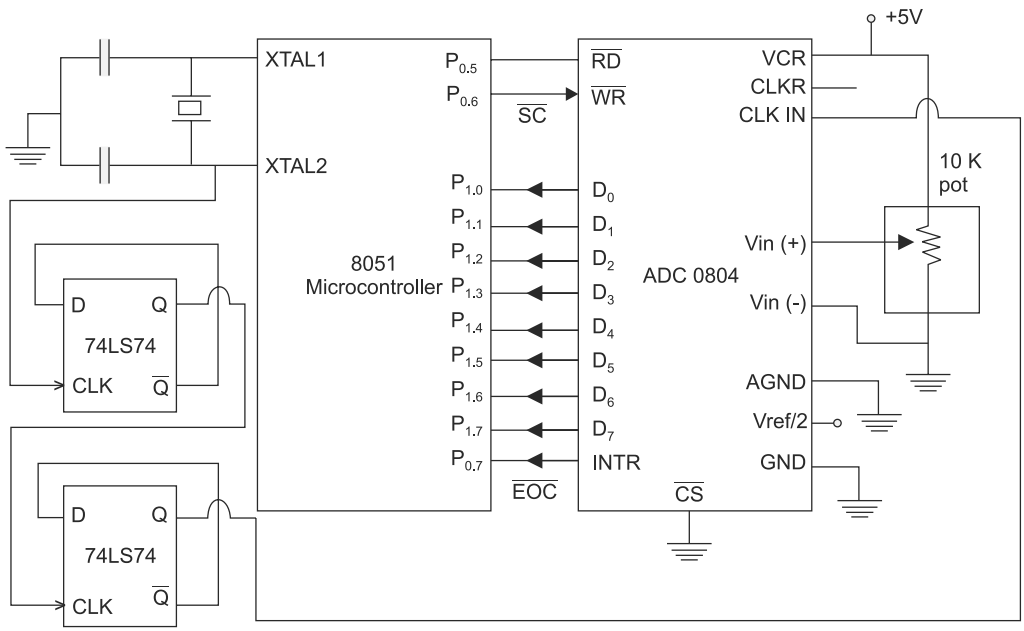


Fig. 8.9 Interfacing an ADC 0804 with 8051 microcontroller

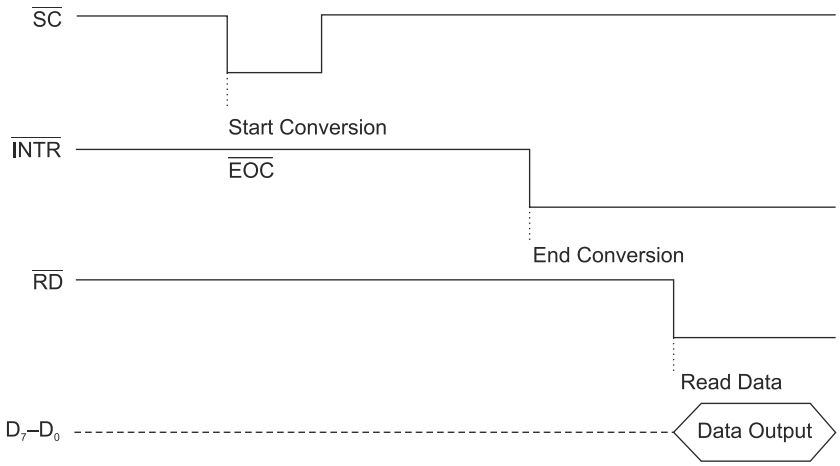


Fig. 8.9a Timing diagram of Analog to Digital Conversion

After the initial reset of 8051 microcontroller, all I/O ports are in the floating condition. The first two instructions in this program, write ones to ports 1 and 0, which make them float. To start the DATA conversion, bit 6 of port 0 is pushed LOW then HIGH using the CLR and set bit SETB instructions. The conversion process will continue and remains in a WAIT loop until bit 7 of port 0 becomes LOW. Then microcontroller read the ADC output data through port 1 and stored in register R_0. Figure 8.9a shows the timing diagram of analog to digital conversion using ADC 0804. The program of A/D converter is given below:

Labels	Mnemonics	Operands	Comments
	MOV	P_1, #0FF	All pins of Port 1 become 1.
	MOV	P_0, #0FF	All pins of Port 0 become 1.
	CLR	$P_{0.6}$	Make $P_{0.6}$ LOW from HIGH for $\overline{SC}$. (start conversion)
	SETB	$P_{0.6}$	
WAIT	JB	$P_{0.7}$, WAIT	Wait until $\overline{EOC}$ becomes low.
	CLR	$P_{0.5}$	Conversion is completed and RD enable.
	MOV	R_0, P_1	Read the data from port P_1 and store it in R_0 register.

8.5.24 Microcontroller Based Traffic Control

Nowadays microcontrollers are used to implement the traffic control system. Figure 8.10 shows the simple model of microcontroller-based traffic control system. The various control signals such as red, green, orange, forward arrow, right arrow and left arrow are used in this scheme. The forward, right and left arrows are used to indicate forward, right and left movement respectively. The red (R) signal is used to stop the traffic in the required lane and the yellow (Y) signal is used as standby, which indicates that the traffic must wait for the next signal. The green (G) light for a particular lane remains ON for DELAY-1 seconds followed by the stand- by signal for DELAY-2 seconds. However at a time 3 out of the four roads, the left signal or the left arrow remains ON even though that lane may have a red signal. The traffic light control is implemented using a 8051 microcontroller kit having 8255 on board and the interfacing circuit is illustrated in Fig. 8.11. Each signal is controlled by separate pin of I/O ports. The total number of logic signals required for this

arrangement is twenty-four. The programmable peripheral interface device 8255 is used to interface these 24 logic signals with the lamps. The logic '0' and '1' represent the state of the lamp. Logic '1' represents ON and '0' represents OFF. All ports of 8255 are used as output ports. The control word to make all ports as output ports for Mode 0 operation is 80H. The traffic light control program can be written by the following steps:

Step-1 Initialize all ports of the 8255 as output ports.

Step-2 Determine the required status of port A, port B and port C of 8255 for North to South traffic movement. Load data into Accumulator and send to port A, port B and port C for North to South traffic movement.

Step-3 Call delay subroutine-1.

Step-4 Before starting East to West traffic movement, North to south traffic movement will be ready to stop and East to West traffic must be ready for movement. Therefore determine the required status of port A, port B and port C for this operation. Then load data into Accumulator and send to port A, port B and port C so that North to South traffic movement will be ready to stop and East to West traffic must be ready for movement.

Step-5 Call delay subroutine-2.

Step-6 For East to West traffic movement, determine the required status of port A, port B and port C of 8255. Load data into Accumulator and send to port A, port B and port C for East to West traffic movement.

Step-7 Call delay subroutine-1.

Step-8 Prior to starting South to North traffic movement, East to West traffic will be ready to stop and South to North traffic must be ready for movement. For this operation determine the status of port A, port B and port C of 8255. Load required data into accumulator and send to port A, port B and port C so that East to West traffic will be ready to stop and South to North traffic must be ready for movement.

Step-9 Call delay subroutine-2.

Step-10 Determine the status of port A, port B and port C for South to North traffic movement. Load require data into accumulator and send to port A, port B and port C for South to North movement.

Step-11 Call delay subroutine-1.

Step-12 Before starting West to East traffic movement, South to North traffic movement will be ready to stop and West to East traffic must be ready for movement. Find out the status of port A, port B and port C for this operation. Load required data into Accumulator and send to port A, port B and port C so that South to North traffic movement will be ready to stop and West to East traffic must be ready for movement.

Step-13 Call delay subroutine-2.

Step-14 For West to East traffic movement, determine the status of port A, port B and port C of 8255. Load necessary data into Accumulator and send to port A, port B and port C for West to East traffic movement.

Step-15 Call delay subroutine-1.

Step-16 Subsequently West to East traffic movement will be ready to stop and North to South traffic must be ready for movement. Determine the status of port A, port B and port C for this operation. Load needed data into Accumulator and send to port A, port B and port C. Then West to East traffic movement will be ready to stop and North to South traffic must be ready for movement.

Step -17 Call delay subroutine-2.

Step-18 Jump to step-2.

Figure 8.10 shows the bit assignment of ports. Putting 0s and 1s in required position the data byte for each

port can be derived. For example, during North to South traffic movement, the status of port A, port B and port C are as follows:

PA_7	PA_6	PA_5	PA_4	PA_3	PA_2	PA_1	PA_0
0	0	1	0	0	0	0	1
PB_7	PB_6	PB_5	PB_4	PB_3	PB_2	PB_1	PB_0
0	0	0	0	0	1	0	0
PC_7	PC_6	PC_5	PC_4	PC_3	PC_2	PC_1	PC_0
1	1	1	1	1	0	0	1

When North to South traffic movement will be ready to stop and East to West traffic must be ready for movement, the status of port A, port B and port C are as follows:

PA_7	PA_6	PA_5	PA_4	PA_3	PA_2	PA_1	PA_0
0	0	0	1	0	0	1	0
PB_7	PB_6	PB_5	PB_4	PB_3	PB_2	PB_1	PB_0
0	0	0	0	0	1	0	0
PC_7	PC_6	PC_5	PC_4	PC_3	PC_2	PC_1	PC_0
0	0	0	0	1	0	0	1

The calculated necessary data bytes of port A, port B and port C for all types of traffic movement are illustrated in a Table 8.13 as given below:

Table 8.13 Traffic movement and status of ports

Traffic movement	Status of Port A	Status of Port B	Status of Port C
North to South traffic movement	21H	04H	F9H
North to South traffic movement be ready to stop and East to West traffic be ready for start	12H	04H	09H
East to West traffic movement	0CH	27H	89H
East to West traffic movement be ready to stop and South to North traffic be ready for start	94H	20H	08H
South to North traffic movement	64H	3CH	18H
South to North traffic movement be ready to stop and West to East traffic be ready for start	A4H	00H	14H
West to East traffic movement	24H	D0H	93H
West to East traffic movement be ready to stop and North to South traffic be ready for start	22H	00H	85H

The green light is provided for the traffic flowing from north to south. The arrows indicate the deviations in which traffic is allowed to move. The arrows with a cross indicate that the traffic is not allowed to move in that particular direction. Each signal is controlled by a separate port. The various signals used are red, green,

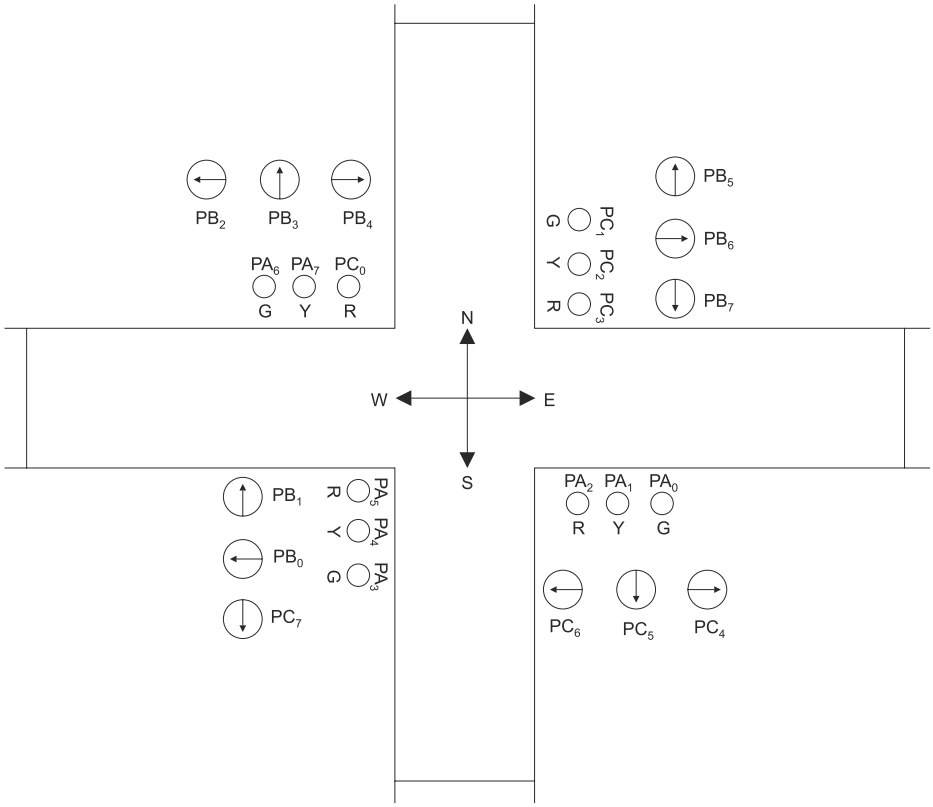


Fig. 8.10 Traffic light control

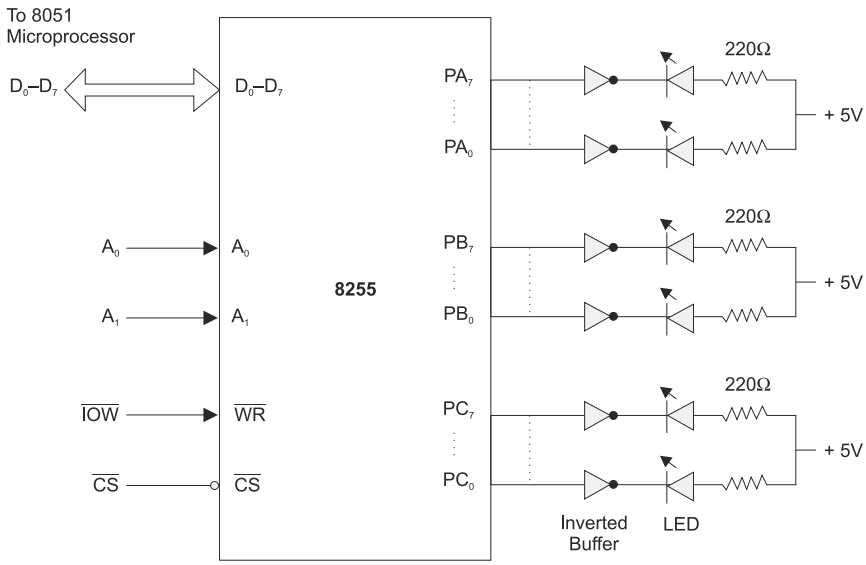


Fig. 8.11 The interfacing circuit for traffic light control

orange, forward arrow, right and left arrows. The forward arrow, right and left arrows are used to indicate forward, right and left movement. The red signal is used to stop traffic in the required lane and the orange signal is used as standby which indicates that the traffic must wait for the next signal. The green lights for a particular lane remain on for 10 seconds followed by the stand by signal for 4 seconds. However at a time 3 out of the four roads the left signal or the left arrow remains on even though that lane may have a red signal. This system is implemented using 8051 trainer having 8255 on board. The 8051 is interfaced with the 8255 and output pins are used to control the various signals. The program for Traffic light control is as follows.

Labels	Mnemonics	Operands	Comments
	MOV	$0A_0$, $\#0E_8$	Load control word of 8255 into Control word register whose address is E803H. Control word is 80H.
	MOV	R_0, #03	
	MOV	A, #80	
	MOVX	@R_0, A	
START	MOV	R_0, #00	Send 21H in Port A, F9H in Port C and 04H in Port B
	MOV	A, #21	for North to South traffic movement.
	MOVX	@R_0, A	
	MOV	R_0, #02	
	MOV	A, $\#F_9$	
	MOVX	@R_0, A	
	MOV	R_0, #01	
	MOV	A, #04	
	MOVX	@R_0, A	
	LCALL	DEDAY-1	Call Delay-1 subroutine
	MOV	R_0, #00	Send 12H in Port A, 09H in Port C and 04H in Port B
	MOV	A, #12	for North to South traffic movement will be ready to stop
	MOVX	@R_0, A	and East to West traffic movement is ready to start.
	MOV	R_0, #02	
	MOV	A, #09	
	MOVX	@R_0, A	
	MOV	R_0, #01	
	MOV	A, #04	
	MOVX	@R_0, A	
	LCALL	DELAY-2	Call Delay-2 subroutine
	MOV	R_0, #00	Send 0CH in Port A, 89H in Port C and 27H in Port B
	MOV	A, #0C	for East to West traffic movement.
	MOVX	@R_0, A	
	MOV	R_0, #02	
	MOV	A, #89	
	MOVX	@R_0, A	
	MOV	R_0, #01	
	MOV	A, #27	
	MOVX	@R_0, A	

Contd.

Contd.

LCALL	DEDAY-1	Call Delay-1 subroutine.
MOV	R_0, #00	Send 94H in Port A, 08H in Port C and 20H in Port B for
MOV	A, #94	East to West traffic movement will be ready to stop and
MOVX	@R_0, A	South to North traffic movement is ready to start.
MOV	R_0, #02	
MOV	A, #08	
MOVX	@R_0, A	
MOV	R_0, #01	
MOV	A, #20	
MOVX	@R_0, A	
LCALL	DELAY-2	Call Delay-2 subroutine
MOV	R_0, #00	Send 64H in Port A, 18H in Port C and 3CH in Port B for South to North traffic movement
MOV	A, #64	
MOVX	@R_0, A	
MOV	R_0, #02	
MOV	A, #18	
MOVX	@R_0, A	
MOV	R_0, #01	
MOV	A, #3C	
MOVX	@R_0, A	
LCALL	DELAY-1	Call Delay-1 subroutine
MOV	R_0, #00	
MOV	A, #A4	Send A4H in Port A, 14H in Port C and 00H in Port B
MOVX	@R_0, A	for South to North traffic movement is ready to stop and
MOV	R_0, #02	West to East traffic movement will be ready to start
MOV	A, #14	
MOVX	@R_0, A	
MOV	R_0, #01	
MOV	A, #00	
MOVX	@R_0, A	
LCALL	DEDAY-2	Call Delay-2 subroutine
MOV	R_0, #00	Send 24H in Port A, 93H in Port C and D_0H in Port B
MOV	A, #24	for West to East traffic movement
MOVX	@R_0, A	
MOV	R_0, #02	
MOV	A, #93	
MOVX	@R_0, A	
MOV	R_0, #01	

MOV	A, #D_0	
MOVX	@R_0, A	
LCALL	DELAY-1	Call Delay-1 subroutine
MOV	R_0, #00	Send 22H in Port A, 85H in Port C and 00H in Port B for
MOV	A, #22	West to East traffic movement is ready to stop and North
MOVX	@R_0, A	to South traffic movement will be ready to start
MOV	R_0, #02	
MOV	A, #85	
MOVX	@R_0, A	
MOV	R_0, #01	
MOV	A, #00	
MOVX	@R_0, A	
LCALL	DELAY-2	Call Delay-2 subroutine
LJMP	START	Jump to START

DELAY-1 SUBROUTINE for 10 SECONDS

Labels	Mnemonics	Operands
	MOV	R_4, #0A
LOOP-3	MOV	R_7, #08
	MOV	R_5, #00
LOOP-2	MOV	R_6, #F_3
LOOP-1	DJNZ	R_5, LOOP_1
	DJNZ	R_6, LOOP_1
	DJNZ	R_7, LOOP_2
	DJNZ	R_4, LOOP_3
	RET	

DELAY-1 SUBROUTINE for 4 SECONDS

Labels	Mnemonics	Operands
	MOV	R_4, #04
LOOP-6	MOV	R_7, #08
	MOV	R_5, #00
LOOP_5	MOV	R_6, #F_3
LOOP_4	DJNZ	R_5, LOOP_4
	DJNZ	R_6, LOOP_4
	DJNZ	R_7, LOOP_5
	DJNZ	R_4, LOOP_6
	RET	

8.5.25 Microcontroller-Based Stepper Motor Control

Stepper motors are electro-mechanical devices, which convert electrical pulses into proportionate discrete mechanical rotational movement. To rotate the stepper motor's shaft, a sequence of pulses is required to be

applied to stator windings of stepper motor. When a given number of command pulses are supplied to the motor, shaft will have turned through a known angle. Therefore, motor can be used to control position by keeping count of the number of command pulses. Each revolution of the stepper motor's shaft is made up of a series of discrete individual steps. A step is defined as the angular rotation produced by the shaft each time when the motor receives a step pulse. Due to each step, the shaft can rotate a specified angle in degree. The rotation of the shaft due to each step is called as step angle. The stepper motors are usually used in position control of robot arms, paper drive mechanism in a printer, machine tools control, process control system, textile industry, integrated circuit fabrication, electric watches, tape as well as disk drive systems, etc. Further, the average motor speed is proportional to rate at which the pulse command is delivered. At low command pulse rate, rotor moves in steps, but when the pulse rate is made sufficiently high, because of the inertia, the rotor moves smoothly, as in case of dc motors. As motor speed is proportional to rate of command pulses, it can be used for speed control.

Figure 8.12 shows four phase stepper motor windings and its interfacing is depicted in Fig. 8.13. The four windings A_1, A_2, B_1 and B_2 are connected to PA_3, PA_2, PA_1 and PA_0 respectively. When PA_3 is level '1' and PA_1 is level '1', the coil A_1 and B_1 are energized and motor will rotate by one step clockwise. Similarly coil A_1 and B_2 will be energized when PA_3 is level '1' and PA_0 is in level '1' and again motor rotate by one step. In the same way the other phases are energised sequentially as per Table 8.14 and switching sequence waveform of windings is illustrated in Fig. 8.14. The assembly language program for stepper motor control in clockwise as well as anti-clockwise rotation is illustrated below.

Table 8.14 Sequence of switching of windings

A_1	A_2	B_1	B_2	Clockwise CW	Counterclockwise CCW
1	0	1	0	A	A
1	0	0	1	9	9
0	1	0	1	5	5
0	1	1	0	6	6

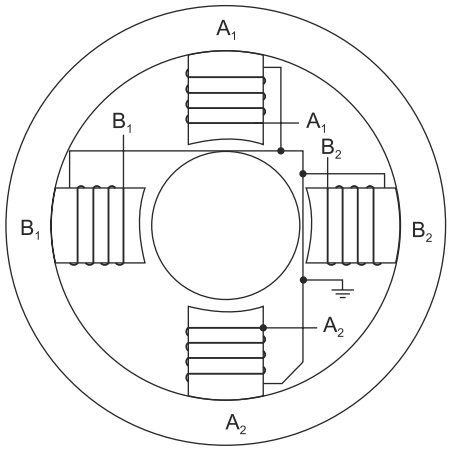


Fig. 8.12 Four-phase stepper motor windings

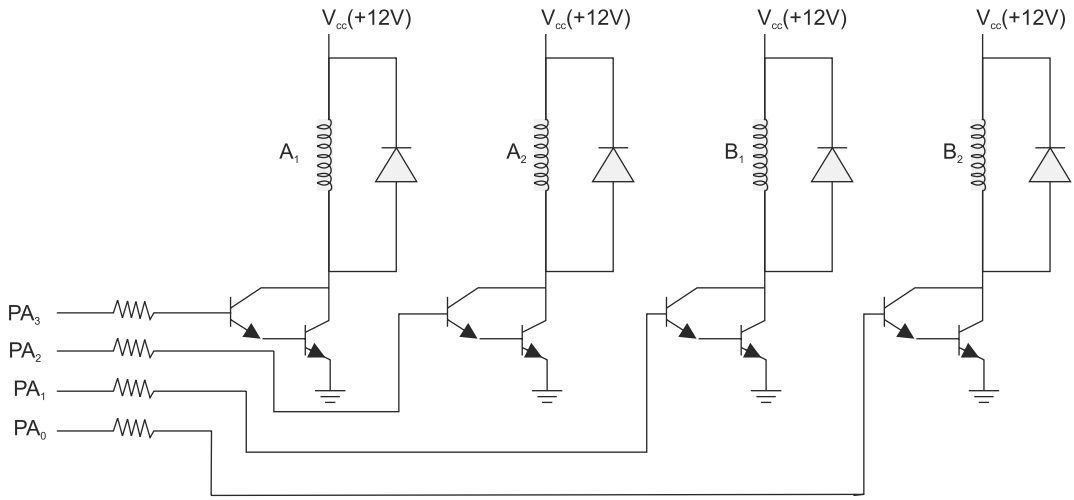


Fig. 8.13 Driver circuit of stepper motor

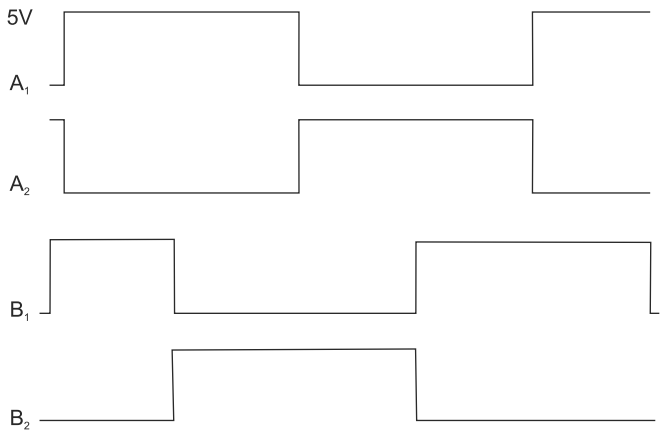


Fig. 8.14 Switching sequence waveform of windings

PROGRAM

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	74 80		MOV	A, #80	Load control word 80H in 8255.
8002	90 E_8 03		MOV	DPTR, #E803	
8005	F_0		MOVX	@DPTR, A	
8006	74 00	START	MOV	A, #00	
8008	12 01 61		LCALL	0161	Call subroutine to read character.
800B	B_4 55 03		CJNE	A,#55, 8011	Compare if U is pressed.
800E	12 D_0 00		LCALL	D000	Call Clockwise rotation subroutine.
8011	B_4 44 03		CJNE	A,#44, 8017	Compare if D is pressed.

Contd.

8014	12 90 00		LCALL	9000	Call Counterclockwise rotation subroutine.
8017	02 80 06		LJMP	8006	Jump to 8006.

Subroutine for Clockwise rotation

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
D000	74 0A	LOOP_1	MOV	A, #0A	Load 0A in accumulator.
D002	12 C_0 00		LCALL	C000	Call subroutine to send data, 0AH in Port A.
D005	12 B_0 00		LCALL	B000	Call delay subroutine.
D008	74 09		MOV	A, #09	Load 09 in accumulator.
D00A	12 C_0 00		LCALL	C000	Call subroutine to send data, 09H in Port A.
D00D	12 B_0 00		LCALL	B000	Call delay subroutine.
D010	74 05		MOV	A, #05	Load 05 in accumulator.
D012	12 C_0 00		LCALL	C000	Call subroutine to send data, 05H in Port A.
D015	12 B_0 00		LCALL	B000	Call delay subroutine.
D018	74 06		MOV	A,#06	Load 06 in accumulator.
D01A	12 C_0 00		LCALL	C000	Call subroutine to send data, 06H in Port A.
D01D	12 B_0 00		LCALL	B000	Call delay subroutine.
D020	02 D_0 00		LJMP	D000	Jump to LOOP_1.

Subroutine for Counter Clockwise rotation

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
9000	74 06	LOOP_2	MOV	A, #06	Load 0A in accumulator.
9002	12 C_0 00		LCALL	C000	Call subroutine to send data, 0AH in Port A.
9005	12 B_0 00		LCALL	B000	Call delay subroutine.
D008	74 05		MOV	A,#05	Load 09 in accumulator.
900A	12 C_0 00		LCALL	C000	Call subroutine to send data, 09H in Port A.
900D	12 B_0 00		LCALL	B000	Call delay subroutine.
9010	74 09		MOV	A,#09	Load 05 in accumulator.
9012	12 C_0 00		LCALL	C000	Call subroutine to send data, 05H in Port A.
9015	12 B_0 00		LCALL	B000	Call delay subroutine.
9018	74 0A		MOV	A,#0A	Load 06 in accumulator.
901A	12 C_0 00		LCALL	C000	Call subroutine to send data, 06H in Port A.
901D	12 B_0 00		LCALL	B000	Call delay subroutine.
9020	02 90 00		LJMP	9000	Jump to LOOP_2.

Subroutine to Send data in Port A					
Memory address	**Machine Codes**	**Labels**	**Mnemonics**	**Operands**	**Comments**
C000	90 EB 00		MOV	DPTR, #E800	Load Port A address E800H in DPTR.
C003	F_0		MOVX	@DPTR,A	Send accumulator content into Port A.
C004	22		RET		Return.

Delay Subroutine					
Memory address	**Machine Codes**	**Labels**	**Mnemonics**	**Operands**	**Comments**
B000	7B 10		MOV	R_3,#10	Move 10H into R_3.
B002	7C FF		MOV	R_4,#FF	Move FFH into R_4.
B004	1B		DEC	R_3	Decrement R_3.
B005	BB 00 FC		CJNE	R_3, #00, B004	Compare R_3 with 00. if $R_3 \neq 0$, jump to B004.
B008	1C		DEC	R_4	Decrement R_4.
B009	BC 00 FC		CJNE	R_4,#00, B008	Compare R_4 with 00. if $R_4 \neq 0$, jump to B008.
B00C	22		RET		Return.

8.5.26 Program to Display "8051 Microcontroller" on Screen

Delay Subroutine					
Memory address	**Machine Codes**	**Labels**	**Mnemonics**	**Operands**	**Comments**
8000	90 80 09		MOV	DPTR, #8009	Data corresponding to "8051 MICROCONTROLLER" is stored in program memory starting from 8009.
8003	12 12 00		LCALL	1200	Call subroutine to display string of characters on screen.
8006	02 00 00		LJMP	0000	

Memory address	Data
8009	20 20 20 20 30 30 35 31 20 20 20 20
8015	4D 49 43 52 4F 43 4F 4E 54 52 4F 4C 4C 45 52 20 20 20

The main program to display characters on screen is written from 8000H to 8008. Data corresponding to characters "8051 MICROCONTROLLER" are stored in program memory starting from 8009 as given above. After executing the above program, "8051 MICROCONTROLLER" will be displayed on screen.

Review Questions

8.1 What are the addressing modes of 8051 microcontroller? Explain each addressing mode with example.

8.2 Write the addressing modes of the following instructions
(i) MOV A,@R_0 (ii) MOVX @DPTR, A (iii) MOV A, @A+DPTR (iv) MOA R_0, #45H

8.3 State different types of instructions of 8051 and explain any three instructions from each group of the instructions.

8.4 Write instructions to perform the following operations
(i) Move the content of accumulator to register 7(R_7)
(ii) Move the contents of RAM memory location 55H to port 1
(iii) Send 22H to port 0
(iv) Move the value at port 2 to register 1(R_1)
(v) Clear bit 7 of the accumulator

8.5 Write the difference between the following instructions
(i) LJMP and SJMP (ii) RET and RETI (iii) MOV and MOVX

8.6 Write the following programs in assembly language
(i) Add Two 8-Bit Numbers
(ii) Add Two 16-Bit Numbers
(iii) Add a Series of 8-bit Numbers
(iv) Subtract Two 8-Bit Numbers
(v) Two's Complement of an 8-bit Number
(vi) Find the Largest Number in a Data Array
(viii) Find Smallest Number in a Data Array
(ix) Perform Division of Two Numbers
(x) Perform Multiplication of Two Numbers
(xi) SWAP 4 MSBs with 4 LSBs in Accumulator
(xii) Arrange a Series of Numbers in Descending Order
(xiii) Compare two 8-bit Numbers
(xiv) Arrange a Data Array in Ascending Order

8.7 Explain a microcontroller based traffic light control system with assembly language program.

8.8 Explain 8051 microcontroller based position control system using stepper motor.

8.9 Draw a circuit diagram for key board interface with 8051 microcontroller and write a program for reading any key.

8.10 Write a program for A/D converter interface with 8051 microcontroller.

8.11 What is the difference between CY and OV flags?

8.12 What is the addressing mode of MOV A, @ R_i instruction?

8.13 What is the content of the accumulator after execution of the following instructions?
MOV A, HFFH and ADD A, #23H

8.14 Which addressing mode is suitable for look-up table access?

8.15 What are the instructions used to access the program memory?

Multiple-Choice Questions

8.1 What is the addressing mode of MOV A, 40
(a) direct addressing (b) indirect addressing
(c) index addressing (d) register addressing

8.2 Which instruction does not belong to register addressing mode
(a) MOV A, R_7 (b) MOV R_0, R_1 (c) MOV A, @R_3 (d) MOV R_5, A

8.3 Which of the following instructions is index addressing
(a) MOVC A, @A+DPTR
(b) MOVX @DPTR, A
(c) MOVX A, @DPTR
(d) MOVX A, @R_0

8.4 MOVX A, @R_0 instruction performs
(a) data transfer from external RAM 8-bit address specified by R_0 to accumulator
(b) data transfer from internal RAM 8-bit address specified by R_0 to accumulator
(c) data transfer from external ROM 8-bit address specified by R_0 to accumulator
(d) data transfer from internal ROM 8-bit address specified by R_0 to accumulator

8.5 Which of the following instructions is incorrect
(a) CPL A (b) SWAP A (c) CLR C (d) RL B

8.6 Which of the following flags are affected by the instruction INC A and INC @R_0
(a) carry flag (b) auxiliary carry flag
(c) overflow flag (d) No flags are affected

8.7 What will be the output after execution of the following instructions

```
MOV A, #55
ANL A, #67
```

(a) 54 (b) 45 (c) 55 (d) 67

8.8 To exchange the content of A and R_0 which instruction is used
(a) XCH A,R_0 (b) XCH A,@R_0 (c) XCHD A,@R_0 (d) XCH R_0,A

8.9 Which of the following instructions is not a logical instruction
(a) ANL A, #FF (b) CPL A (c) INC A (d) SWAP A

8.10 Which of the following instructions is not an arithmetic instruction
(a) MUL AB (b) ADD A,#66H (c) DIV AB (d) CPL A

8.11 Which of the following instructions do not perform the increment of the content of memory location 50H by 1
(a) INC 50 (b) MOV R_0, #50; INC @R_0
(c) MOV A, #50; INC A (d) MOV R_0, #50, INC R_0

8.12 Which of the following instructions are used to swap nibbles inside accumulator
(a) SWAP A (b) RR A; RR A
(c) RR A; RR A; RR A; (d) RRC A RRC A; RRC A; RRC A;

8.13 Which of the following instructions is incorrect
(a) RLC A (b) SWAP B (c) CPL C (d) MOVC A,@A+PC

8.14 Which of the following instructions is indirect addressing
(a) MOV A,R_0 (b) MOV A, 40H (c) MOV R_7, #55 (d) MOV A,@R_0

8.15 The operation "send the content of RAM whose address is specified by R_3 to port 3" is performed by
(a) MOV P_3,@R_3 (b) MOV P_3, R_3 (c) MOV @R_3, P_3 (d) MOV R_3, P_3

8.16 What will be the contents of A register after execution of instruction RRC A. Assume the contents of A before execution is C5H and carry is zero
(a) 62 (b) 26 (c) 66 (d) 22

Answers to Multiple-Choice Questions

8.1 (a)	8.2 (c)	8.3 (a)	8.4 (a)
8.5 (d)	8.6 (d)	8.7 (b)	8.8 (a)
8.9 (c)	8.10 (d)	8.11 (c)	8.12 (a)
8.13 (b)	8.14 (d)	8.15 (a)	8.16 (a)

CHAPTER

9

Architecture of 8086 and 8088 Microprocessors

9.1 INTRODUCTION

The Intel 8086 is a high performance 16-bit, N-channel, HMOS microprocessor which is available in three clock rates: 5, 8, and 10 MHz. The term HMOS stands for 'High-Speed MOS'. 8086 is Intel's first 16-bit microprocessor. This processor was introduced in 1978, due to the demand for more powerful and high-speed computers. This processor has a more powerful instruction set and more programming flexibility and its speed is more than the 8085 microprocessor. The CPU of the 8086 processor is implemented in N-channel, depletion load, and silicon gate technology. This processor has the following features:

- The CPU has direct addressing capability of 1 MB memory.
- Bit, byte, word and block operations are available.
- 8-bit and 16-bit signed and unsigned arithmetic in binary and decimal operations are performed.
- Available in 40-pin lead CERDIP and plastic DIP package (Dual In-Line Package).
- Architectures designed for assembly language as well as high-level language.

The 8086 is manufactured for standard temperature range (32°F to 180°F) and extended temperature range (40°F to +225°F). It contains an electronic circuitry of 29000 transistors. The 8086 has 20 address lines and 16 data lines. This CPU can directly address up to 2^{20} = 1 Mbytes of memory. The 16-bit data word can be divided into a low-order byte and a high-order byte. The 20-bit address lines are time multiplexed to select lines of low-order byte and a high-order byte data separately. The 8088 is an 8-bit processor designed around 8086 architecture. Internal functions of 8088 are same as the 8086 processor functions. The 8088 processor has a 20-bit address bus and an 8-bit data bus. The comparison between 8085, 8086 and 8088 microprocessors are illustrated in Table 9.1 and Table 9.2. In this chapter, architecture of 8088 and 8088 are discussed in detail.

9.2 ARCHITECTURE OF 8086

The 8086 architecture has been implemented using two-stage pipelining in instruction execution. The processor logic unit has been divided into Bus Interface Unit (BIU) and Execution Unit (EU). These units are

always operating asynchronously. The Bus Interface Unit (BIU) provides interface with external memory and I/O device addresses and data bus, and executes all bus operations. The BIU has a 6-byte instruction queue. On the other hand, the execution unit takes the instruction from 6-byte instruction queue of BIU and executes it. Thus, the instruction-fetch time has been drastically reduced.

Table 9.1 Comparison between 8085 and 8086 microprocessors

8085 Microprocessor	8086 Microprocessor
8085 is an 8-bit processor created in 1977 and it has 8-bit data bus.	8086 is a 16-bit processor developed in 1978 and it has a 16-bit data bus.
8085 is manufactured using NMOS technology and this processor IC consists of about 6200 transistors.	8086 is fabricated based on HMOS technology and this processor IC consists of approximately 29000 transistors.
8085 has a 16-bit address bus and can be able to access 2^{16} = 64 KB memory locations.	8086 has a 20-bit address bus and is able to access 2^{20} = 1 MB memory locations.
Number of flags are 5.	Number of flags are 9.
Pipelining concept is not used in 8085.	8086 uses pipelining.
Instruction queue does not exist in 8085 and sequentially execute instructions.	8086 has a 6-byte instruction queue in BIU.
No segment registers exist in 8085.	There are four segment registers, CS, DE, ES, SS in 8086.
Only four types of addressing modes are available.	Eight types of addressing modes are available.
8085 has less instruction than 8086. Direct multiplication, divide, string-byte block movement and loop instructions are not available in 8085.	8086 has more instructions than 8085. Direct multiplication, divide, string-byte block movement and loop instructions are available in 8085.

Table 9.2 Comparison between 8086 and 8088

8086 Microprocessor	8088 Microprocessor
8086 is a 16-bit processor developed in 1978 and it has a 16-bit data bus.	8088 is an 8-bit processor developed in 1979 and it has 8-bit data bus.
8086 has a 6-byte instruction queue in BIU.	8086 has a 4-byte instruction queue in BIU.
The 8086 BIU fills the queue when its queue has an empty space of 2 bytes.	The 8088 BIU fetches a new instruction byte to load into the queue, whenever there is one byte hole in the queue.
As 8086 has a 16-bit data bus, 8-bit or 16-bit memory read/write operation is possible in a single operation.	As 8088 has an 8-bit data bus, it can read 8 bits data from memory or I/O devices and write 8-bits of data to memory or I/O devices. To read 16-bit data, the 8088 requires two memory read operations.
AD_{15}–AD_8 pins are used as time multiplexed address/data bus in 8086.	AD_7–AD_0 pins are used as time multiplexed address/data bus and A_{15}–A_8 pins are used as address bus only in 8088.
$\overline{BHE}$ is present in 8086 and the external memory interfaces have even or odd address banks.	$\overline{BHE}$ is not present in 8088. Therefore, the external memory interfaced will not have been even or odd address banks. The external memory will therefore be byte oriented as 8085.
In 8086 I/O and memory pin is represented as $IO/\overline{M}$.	In 8088 I/O and memory pin has been inverted and represented as $IO/\overline{M}$.
The status signals of 8086 are $\overline{S}_2$, $\overline{S}_1$ and $\overline{S}_0$.	The status signals of 8086 are $IO/\overline{M}$, $DT/\overline{R}$ and $\overline{SS}_0$.
The overall execution time of the instructions in 8086 is less compared to 8088 as 8086 has 16-bit data bus and only 4 clock cycles are required to execute.	The overall execution time of the instructions in 8088 is more due to the 8-bit data bus and the 16-bit operations require additional 4 clock cycles.

The 8086 is a 16-bit microprocessor and it has a 20-bit address bus and a 16-bit data bus. Therefore, this processor can directly access 2^{20} = 1,048,567 (1 MB) memory locations. It can read/write 8 bits data or 16 bits data from/to memory or input/output (I/O) devices. The 8086 has time multiplexed address and data buses. Hence, the number of pins can be reduced, but it slows down the data transfer rate. The block diagram of internal architecture of 8086 processor is shown in Fig. 9.1. It is divided into two separate functional units

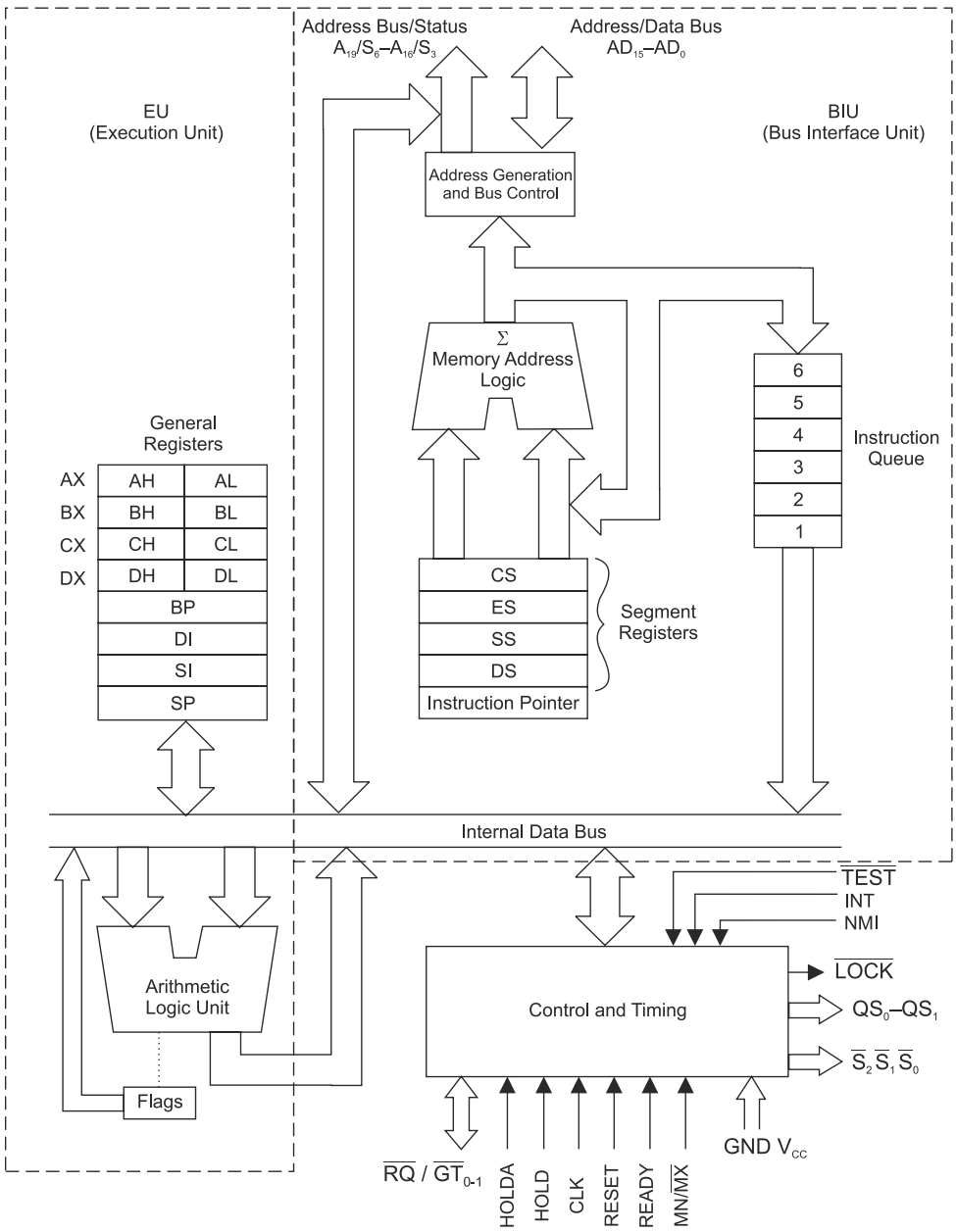


Fig. 9.1 Block diagram of 8086 microprocessor

such as Bus Interface Unit (BIU) and Execution Unit (EU). These two separate units are worked simultaneously for instruction execution based on two-stage instruction pipeline principles.

9.2.1 Bus Interface Unit (BIU)

The Bus Interface Unit (BIU) consists of bus interface logic, general-purpose registers, segment registers, stack pointer, base pointer and index registers, memory addressing logic and a 6-byte instruction queue. The BIU carries out all bus operations for the execution unit, and it is responsible for executing all external bus cycles.

The BIU performs data and addresses transfer between the processor and memory or I/O devices. This section computes and sends addresses, fetches instruction codes, stores fetched instruction codes in a first-in-first-out (FIFO) register which is called *queue*. The BIU is also used to read data from memory and I/O devices, and write data to memory and I/O devices. While the EU is busy in instruction execution, the BIU continues to fetch instructions from memory and stores it in the instruction queue.

This unit relocates addresses of operands while this unit gets un-relocated operand addresses from EU. The execution unit tells BIU from where to fetch instructions as well as to read data.When the EU executes an instruction; the BIU resets the queue, fetches the next instruction from the new memory location, and passes the instructions to the EU. In this way, the 8086 BIU fills the queue when the queue becomes empty spaces of two bytes. This process is known as *pipeline flush*.

9.2.2 Execution Unit (EU)

The Execution Unit (EU) consists of Arithmetic Logic Unit (ALU), general-purpose registers, flag registers (FLAGS), instruction decoder, pointers and index registers, and the control unit which are required to execute an instruction.

The EU gets the opcode of an instruction from the instruction queue. Then the EU decodes it and executes it. The BIU and EU operate independently. When the EU is decoding or executing an instruction, the BIU fetches instruction codes from the memory and stores them in the queue. This type of overlapped operation of the BIU and EU functional units of a microprocessor is called *pipelining*. This process becomes faster except for JUMP and CALL instructions, as the queue must be dumped and then reloaded from a new address. Hence, the function of the EU is to execute all instructions, provide address to the BIU for fetching opcode and operand and perform ALU operations after using various registers as well as the flag register.

9.2.3 Fetch and Execute

During fetch and execute instructions in the 8085 microprocessor, the non-pipeline concept follows so that instructions are fetched and executed sequentially as shown in Fig.9.2(a). In the 8086 processor, the BIU and EU performs the fetch and execute operations with overlap. The fetch and execute operations of 8086 are given below:

- The BIU output is the content of the Instruction Pointer register (IP), which is put on the address bus. Therefore, a byte or word can be read from a specified address into the BIU.
- The content of the instruction pointer register is incremented by 1 to get ready for the next instruction fetch.
- After receiving the opcode and operand of instruction, the instruction code must be passed to the queue which is a FIFO (first-in first-out) register.
- Initially, the queue is empty. As soon as the BIU puts the instruction on the queue, the EU draws the instruction from the queue and starts execution.

- While the EU is executing one instruction, the BIU will continue to fetch new instructions. Depending upon the execution time of the instruction, the BIU can fill the queue with instructions. When execution time is more, the queue will be filled completely before the EU is ready to get the next instruction for execution. Figure 9.2(b) shows the pipeline concept of fetch and execution in BIU and EU. In this architecture, BIU and EU are operating independently. The advantage of this architecture is that the EU executes instructions continuously without waiting of fetching the instruction in BIU.

But sometimes the EU can enter the wait mode. There are three different conditions when the EU operates in wait mode. The first condition is that an instruction wants to access a memory location which is not in the queue. Then BIU suspends fetching instructions and outputs the address of memory location. After waiting for memory access, the BIU can again start filling the queue and the EU also starts to execute instruction codes from the instruction queue.

While executing JUMP instruction, the control is to be transferred to a new address which is non-sequential. But it is known to us that instructions for queue will be executed sequentially. Due to the non-sequential new address of JUMP instruction, the existing instruction codes in queue will not be executed and the EU must wait while the instruction at jump address is fetched. During this operation, the existing bytes in the queue will be discarded.

The last condition for wait mode operation is possible, when the BIU suspends the instruction fetching operation. This is feasible when the EU operates slowly to execute an instruction. In case of AAM, ASCII adjusts for multiplication instruction requires about 83 clock pulses to execute completely. Generally, four clock cycles are generally required per instruction fetch. Consequently, the queue will be completely filled during the execution of AAM instruction. Then BIU must wait until the execution of slow instruction has been completed or the EU pulls one or two bytes from the queue.

Sometimes the instruction requires to read data from a memory location which is not in the queue. Then BIU should suspend instruction fetching and wait for output from the address of memory location. After waiting for reading data from the memory, the EU can again start executing instruction codes from the queue.

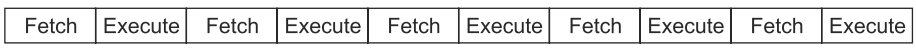


Fig. 9.2(a) General instruction fetch and execution for conventional processors (8085)

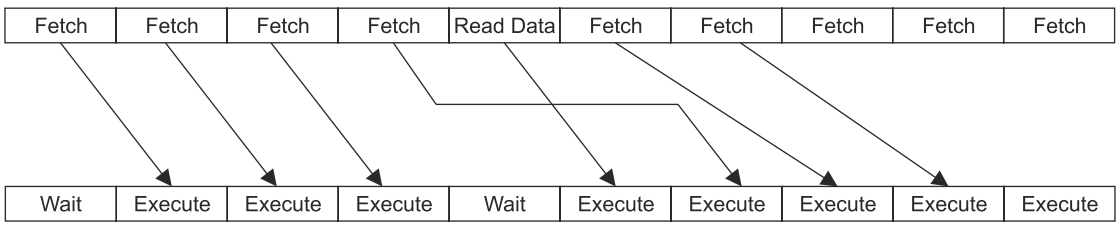


Fig. 9.2(b) Instruction fetch and execution of the 8086 processor

9.3 REGISTERS

The 8086 CPU has fourteen 16-bit registers as depicted in Fig. 9.3. All these registers are subdivided into different groups, namely, Data Register Group (four registers), Segment Register Group (four registers), Pointer and Index Register Group (four registers), Instruction Register (Program Counter) and Flag Register. In this section all registers are discussed.

9.3.1 Data Registers

The 8086 has four 16-bit general-purpose registers (AX, BX, CX and DX). These registers can be used in arithmetic, logical operations and temporary storage. Each of these 16-bit registers is further subdivided into two 8-bit registers (upper and lower bytes) as shown in Table 9.3.

Table 9.3 General purpose data registers

16-Bit Registers	8-Bit High-order Registers	8-Bit Low-order Registers
AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

The functions of each data register are discussed as follows:

AX Register The AX register serves as an accumulator. It performs input/output operations and processes data through AX or AH or AL. During execution of a 16-bit multiply and divide instruction, AX contains the one-word operand and the result is stored in the accumulator. In 32-bit multiply and divide instructions, AX can be used to hold the lower-order word operand. Instructions involving AX or AH or AL and immediately data usually require less program memory.

BX Register BX can be used as an index register for MOVE operation and a base register while computing the data memory address.

CX Register CX register can be used as count register for string operations and hold count value during large-number iterations. In LOOP instructions, CX holds the desired number of repetitions and is automatically decremented by one after each iteration. While CX becomes zero, the execution of instructions should be terminated. In the same way, the 8-bit CL register is used as a count register in bit-shifting and rotate instructions.

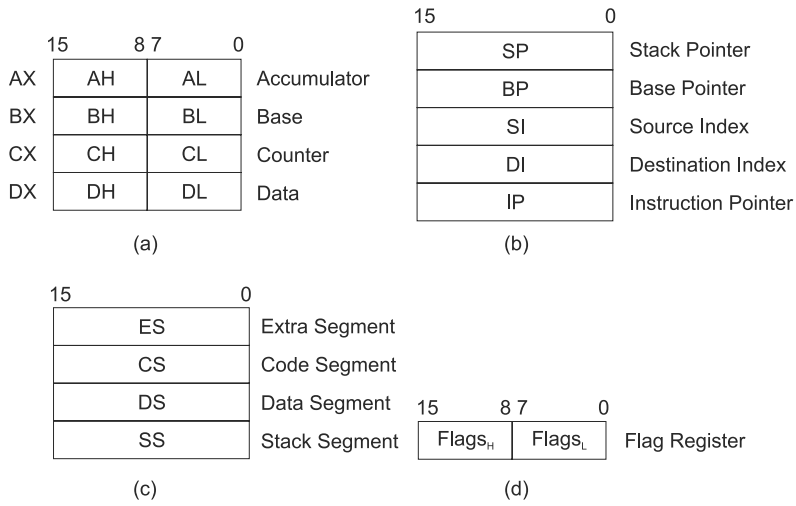


Fig. 9.3 Registers of 8086: (a) data registers, (b) pointer and index registers, (c) segment registers, and (d) flag registers

DX Register DX can be used as a port address for IN and OUT instructions. The DX may be used in I/O instructions, multiply and divide instructions. In 32-bit multiply and divide instructions, DX is used to hold the high-order word operand.

9.3.2 Segment Registers

The concept of memory segmentation has been introduced in the 8086 processor. In memory segmentation, the complete 1 MB memory can be divided into 16 parts which are called *segments*. Each segment thus contains 64 KB of memory. In 8086, there are four segment registers such as Code Segment (CS) Register, Data Segment (DS) Register, Stack Segment (SS) Register and Extra Segment (ES) Register. The 8086 microprocessor-based system memory is divided into four different segments, namely, Code Segment (CS), Data Segment (DS), Stack Segment (SS) and Extra Segment (ES). Each segment has a memory space of 64 KB as depicted in Fig. 9.4 and each segment can be addressed by 16-bit segment registers.

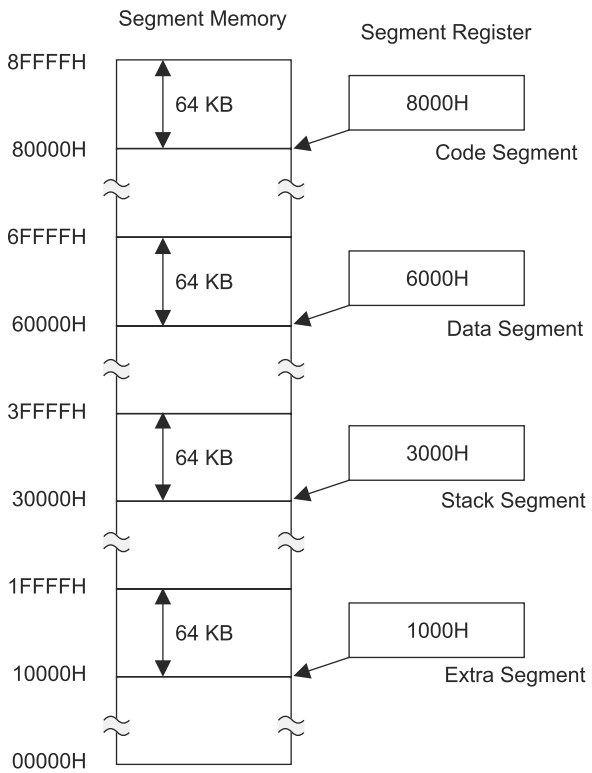


Fig. 9.4 Segment registers and segment memory

Code Segment (CS) The code segment register is used for addressing a memory location in the code segment of the memory in which the program is stored for execution.

Data Segment (DS) Data segment register points to the data segment of the memory, where data is stored.

Extra Segment (ES) The extra segment is a segment which can be used as another data segment of the memory. Therefore, extra segment contains data.

Stack Segment (SS) The stack segment register is used for addressing stack segment of memory in which stack data is stored. The CPU uses the stack for temporarily storing data, i.e., the content of all general-purpose registers which will be used later.

9.3.3 Pointer and Index Registers

The pointer and index registers of 8086 are as follows:

- Stack Pointer (SP)
- Base Pointer (BP)
- Source Index (SI)
- Destination Index (DI)
- Instruction Pointer (IP)

Stack Pointer (SP) The stack pointer is used to locate the stack top address. It contains offset address. In PUSH, POP, CALL and RET instructions, the stack address is determined after adding the contents of the stack segment register (SS), after 4-bit left-shift, to the contents of SP.

Base Pointer (BP) The Base Pointer (BP) register can provide indirect access to data in stack. The BP may also be used for general purpose data storage.

Source index (SI) and Destination Index (DI) These registers are used in memory or stack address computation for general data storage. The main purpose of these registers is to store offset or displacement. The memory address computation, depend upon addressing modes, the content of Data Segment (DS) and index registers.

Sometimes SI is used as source index and DI as destination index. If the content of SI is added with the content of DS to determine the physical address, it will be used as a source address of data, while the content of DI is added with the content of ES to find the destination address of the data. These registers can also be used as general purpose registers.

Instruction Pointer (IP) Generally, the instruction pointer register is used as a program counter. This is used for the calculation of memory addresses of instructions which will be executed. This register stores the offset for the instruction. The content of IP is automatically incremented while the execution of an instruction is going on. The address of the next instruction is computed after adding IP contents to the code segment register contents after 4 bit left-shift.

9.3.4 Flag Register

The 8086 has a 16-bit flag register. This register is also called *Program Status Word*. It has nine flags out of which six are status flags and three are control flags. The status flags are Carry flag (CF), Parity flag (PF), Auxiliary Carry flag (AF), Zero flag (ZF), Sign flag (SF) and Overflow flag (OF). These status flags are affected after the execution of arithmetic or logic instructions. The control flags are Trap flag (TF), Interrupt Flag (IF) and Direction Flag (DF). Figure 9.5 shows the 16-bit flag register of the 8086 processor.

Carry Flag (CF) The carry flag is set to 1, if after arithmetic operation a carry is generated or a borrow is generated in subtraction. When there is no carry out, the carry flag is reset or zero. This flag can also be used in some shift and rotate instructions.

Parity Flag (PF) If the result of 8 bits operation or, lower byte of the word operation contains an even number of 1s, parity flag is set.

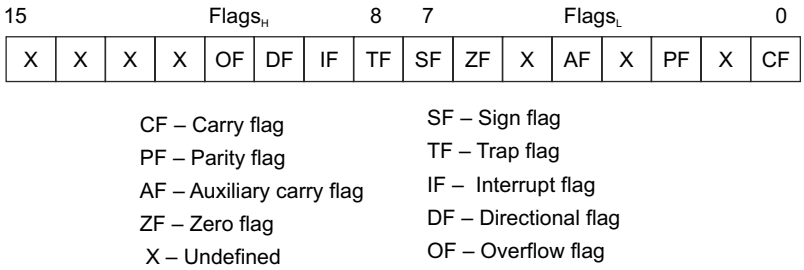


Fig. 9.5 Flag register of 8086

Auxiliary Carry Flag (AF) This flag is set to 1 if there is a carry out of the lower nibble to the higher nibble of an 8-bit operation. It is used for BCD operations.

Zero Flag (ZF) The zero flag is set to 1, if the result of any arithmetic or logical operation is zero. While the result is zero, it is reset.

Sign Flag (SF) The sign flag is set to 1, if the MSB of the result is 1 after the arithmetic or logic operations. This flag represents sign number. Logic 0 indicates positive number and logic 1 is used to represent negative number.

Overflow Flag (OF) This flag is set to 1 if the signed result cannot be expressed within the number of bits in the destination operand. This flag is used to detect magnitude overflow in signed arithmetic operations. During addition operation, the flag is set when there is a carry into the MSB and the flag is reset if there is no carry out of the MSB. For subtraction operation, the flag is set when the MSB desires a borrow, and the flag is reset if there is no borrow from MSB.

Direction Flag (DF) The direction flag is used in string operations. When it is set to 1, string bytes can be accessed from memory address in decrement order, i.e., high memory address to low memory address. If it is zero, string bytes can be accessed from memory address in increasing order, i.e., low memory address to high memory address. For example, in MOVS instruction if DF is set to 1, the contents of the index registers SI and DI are automatically decremented to access the string bytes. If DF = 0, index registers SI and DI are automatically incremented to access the string bytes.

Interrupt Enable Flag (IF) This flag can be used as an interrupt enable or disable flag. When this flag is set, the maskable interrupt is enabled and 8086 recognise the external interrupt requests, and the CPU transfer control to an interrupt vector specified location. When IF is 0, all maskable interrupts are disabled and there will be no effect on nonmaskable interrupts as well as internally generated interrupts. If 8086 is reset, IF is automatically cleared.

Trap Flag (TF) TF is a single-step flag. When TF is set to 1, a single step interrupt occurs after the execution of each instruction and the program can be executed in single-step mode. The TF will be cleared by the single step interrupt.

9.4 LOGICAL AND PHYSICAL ADDRESS

The 8086 sends a 20-bit address on the address bus to detect a memory location for memory read or write operation. Addresses within the segment can be varied from 0000H to FFFFH (64 KB). To detect a memory location, the segment register supplies the higher-order 16 bits of the 20-bit memory address. The lower-order

16 bits of the 20-bit memory address is stored in any of the pointers and index registers or BX register. Therefore, memory addresses of the 8086 are computed by summing the contents of the segment register which is shifted left by 4 bits and the content of offset address. The 20-bit address send by the 8086 processor is called the physical address as depicted in Fig. 9.5.

The physical address is calculated from the segment address and offset address. The segment register contains the higher-order 16 bits of the starting address of a memory segment. The CPU shifts the content of the segment register left by four bits or inserts four zeros for the lowest four bits of the 20-bit memory address. For example, if the content of the code segment register is 4000H, the starting address of the code segment will be 40000H. Hence the 64-KB memory segment may be anywhere within the complete 1MB memory based on the content of code segment register, and the starting address should be divisible by 16.

The offset address is used to determine the memory location distance from the starting address within the memory segment. An offset can be determined depending upon the addressing modes. The offset address will be different in different addressing modes. To locate a memory location within a memory segment, the 8086 processor generates a 20-bit physical address.

To determine the 20-bit physical address with the segment register and offset, the content of segment register is left shifted by 4 bits and then an offset is added to it. For example, if the content of CS is 4000H and an offset is 2000H, the computation of the 20-bit physical address is 42000H. Then 42000H represents the starting address of the segment in memory. Figure 9.6 shows the computation of physical address.

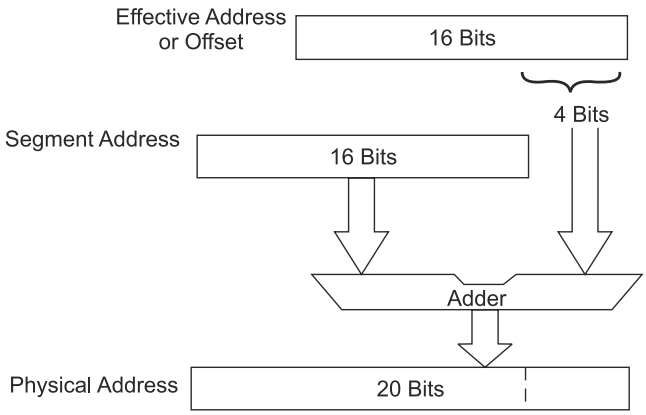


Fig. 9.6 Construction of physical address

Example 9.1 Determine the physical address when CS = 5300H and IP = 0200H. Write the starting and ending address of code segment.

Sol. The content of code segment is left shifted by 4 bits and the base address becomes 53000H. To determine the physical address, the content of IP will be added with the base address. Hence, physical address = 53000 + 0200 = 53200H

The starting of code segment memory = 53200H

As each segment memory consists of 64K memory locations, the end address will be computed after addition of 64K with the starting of code segment memory.

The ending address of code segment = 53200 + FFFF = 631FFH.

9.5 ADDRESS BUS, DATA BUS, CONTROL BUS

The 8086 processor is connected with memory and I/O devices through a set of parallel lines called buses. There are three different buses such as address bus, data bus and control bus which are explained below:

Address Bus The 8086 CPU uses the address bus to select the desired memory or I/O device by generating a unique address which corresponds to the memory location or the location of I/O device of the system. Address bus is unidirectional and this processor has 20-bit address lines.

Data Bus To transfer data between the CPU and memory and the CPU and I/O devices, data bus is used. The data which is in the data bus can be used as instruction for the CPU, or the CPU sends data to I/O device or the CPU receive data from I/O device. Therefore, a data bus is bidirectional.

Control Bus The control bus of 8086 carries control signals which are used to specify the memory and I/O devices. The control signals of 8086 CPU are $M/\overline{IO}$, $\overline{INTA}$, ALE, and $\overline{DEN}$, etc.

9.6 8086 MEMORY ADDRESSING

The 8086/8088 processor has 20-bit address lines and it can allow 2^{20} or 1048576 (1 MB) memory locations. Hence, the 8086 memory address space can be viewed as a sequence of 16 segments as depicted in Fig. 9.7(a). Capacity of each segment is 64 KB. Each memory location contains 8-bit data or one byte data and any two consecutive memory locations contain 16-bit data or a word. 524, 287 words are visualised in Fig. 9.7(b).

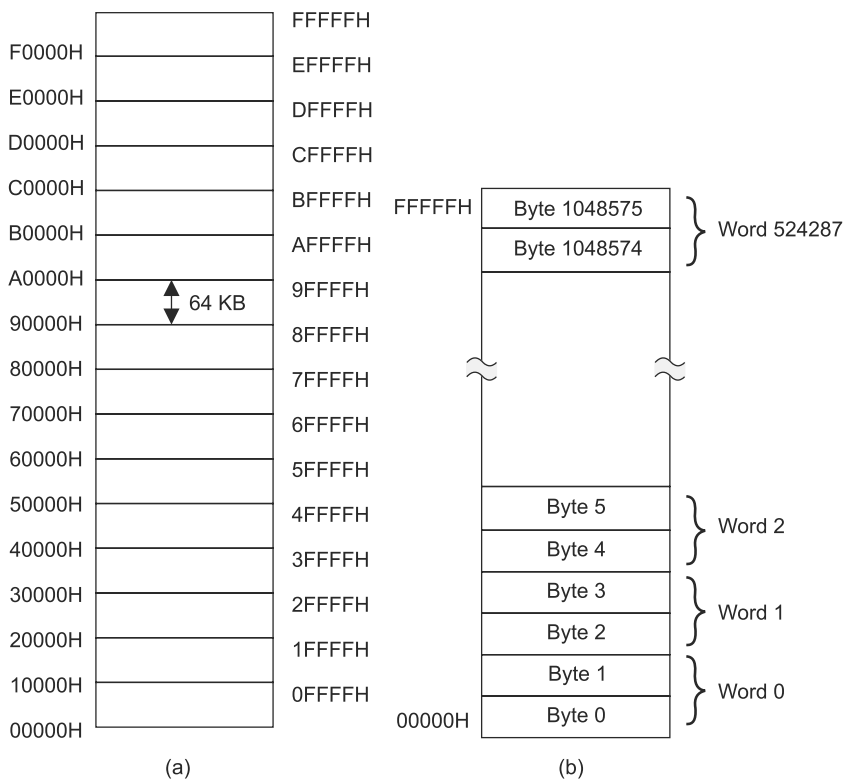


Fig. 9.7 (a) Memory map with 16 segments (b) Memory map with 524287 words

Physically, the memory can be organised as two banks such as even and odd bank and each bank consists of 512 KB memory size. The data lines D_7–D_0 are used for data transfer from even bank and D_{15}–D_8 are used for the odd bank. The even bank is selected by $A_0 = 0$ and $\overline{BHE} = 1$ and data bus D_7–D_0 is connected with this bank. When $A_0 = 1$ and $\overline{BHE}$ is low the odd bank is selected and data bus D_{15}–D_8 is connected. The address space is physically connected to a 16-bit data bus by dividing the address space into two 8-bit banks, namely, *odd-addressed bank* and *even-addressed bank* as depicted in Fig. 9.8.

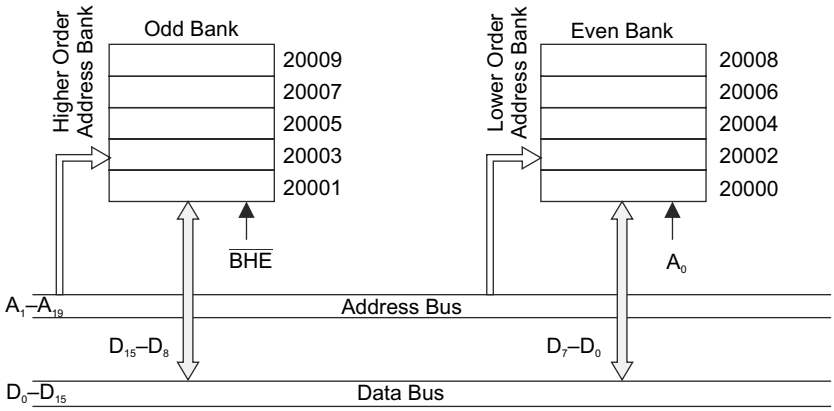


Fig. 9.8 Odd-bank and even-bank memory addressing

The 8086 reads 16 bits of data from an odd-addressed bank memory and an even addressed bank memory simultaneously. One bank is connected to the lower byte of the 16-bit data bus, D_7–D_0 and contains even address bytes, if $A_0 = 0$ and $\overline{BHE} = 1$ an even address bank is selected. The odd-addressed bank is connected to higher byte of data bus, D_{15}–D_8 and contains odd address bytes, when $A_0 = 1$ and Bus High Enable, $\overline{BHE}$ is low and odd address bank is selected. Any specific byte within the even-addressed or odd-addressed bank can be selected by address lines A_1–A_{19}. Table 9.4 shows the memory processing depending upon A_0 and $\overline{BHE}$. Data can be accessed from the memory in four different ways as given below:

- 8-bit data from even address bank
- 8-bit data from odd address bank
- 16-bit data starting from even address bank
- 16-bit data starting from odd address bank

Table 9.4 Selection of proper byte from even and odd memory bank for processing

$\overline{BHE}$	A_0	Processing
0	0	Both banks active 16-bit data transfer. 16-bit word transfer on AD_{15}–AD_0
0	1	Only high (odd) bank active, One byte transfer on AD_{15}–AD_8
1	0	Only low (even) bank active, One byte transfer on AD_7–AD_0
1	1	No bank active.

9.6.1 8-Bit Data from Even Address Bank

To access memory bytes from an even address, information is transferred over the lower half of the data bus D_7–D_0. If $A_0 = 0$, $\overline{BHE}$ is high to enable the even bank. For example, assume loading one byte data into BH

register from memory location within the even address bank. The data will be accessed from the even bank through D_7–D_0. Then this data will be transferred into the 8086 over lower 8-bit data lines, the 8086 redirects the data to the higher 8 bits of its internal 16-bit data path and hence data is loaded into the BH register.

Assume 20-bit address is 20002H. $A_0 = 0$, $\overline{BHE}$ =1, one byte data can be transferred from the memory. Only even bank is selected and only one byte will be transferred from 20002H to the data bus as depicted in Fig. 9.9.

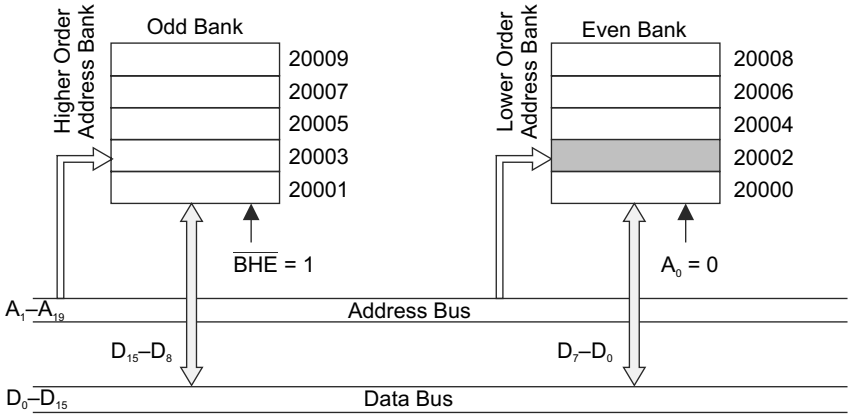


Fig. 9.9 Even bank memory addressing

9.6.2 8-Bit Data from Odd Address Bank

To read one byte from an odd address bank, information must be transferred over the higher order data bus, D_{15}–D_8. If $A_0 = 1$, the even memory bank is disabled and $\overline{BHE}$ is low to enable the odd bank as depicted in Fig. 9.10.

Assume the 20-bit address is 20003. As $A_0 = 1$ and $\overline{BHE} = 0$, only one byte has to be transferred from memory. As an odd bank is selected for data transfer, one byte will be transferred from odd bank memory. The data bus D_{15}–D_8 contents data from memory.

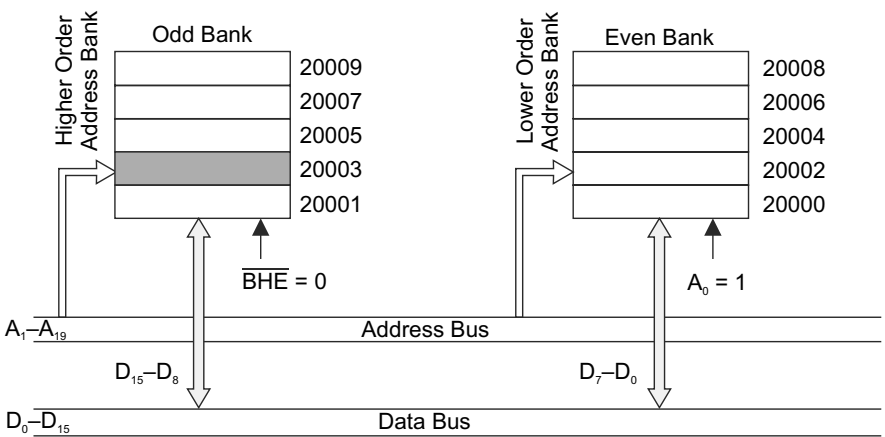


Fig. 9.10 Odd bank memory addressing

9.6.3 16-Bit Data Starting from Even Address Bank

Figure 9.11 shows that 16-bit data is accessed from an even address and an odd address respectively, within a single bus cycle. The address lines A_{19}–A_1 select the appropriate byte within each bank. While $A_0 = 0$ and $\overline{BHE}$ is low, the even and odd banks are enabled simultaneously. For example, the 20-bit address is 20002. Since $A_0 = 0$ and $\overline{BHE} = 0$, one word or two bytes has to be transferred from memory location 20002 and 20003 respectively. Data from the odd bank is transferred to D_{15}–D_8 and data from the even bank is transferred to D_7–D_0 data bus. Hence, data bus D_{15}–D_0 contents two bytes of data from memory. As $\overline{WR} = 0$, $M/\overline{IO}$ =1, 16-bit data can be copied into the data bus from memory bank.

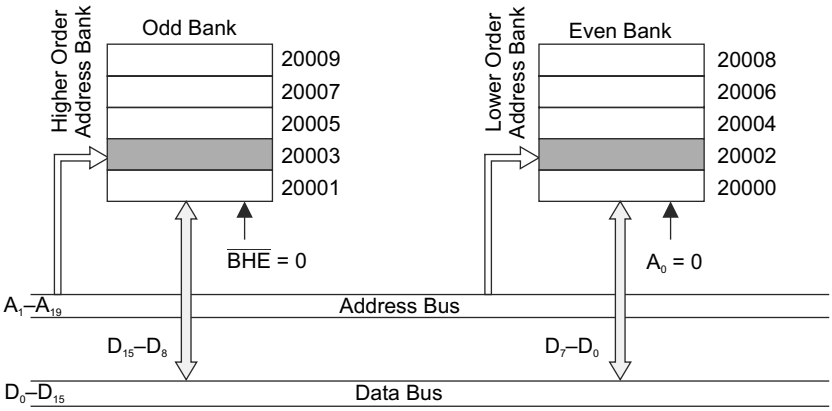


Fig. 9.11 Odd and even bank memory addressing

9.6.4 16-Bit Data Starting from Odd Address Bank

Generally, a 16-bit word located at an odd address is accessed using two bus cycles. Assume the 20-bit physical address is 20003H and the 8086 transfers a word in two bus cycles. During first cycle $A_0 = 1$ and $\overline{BHE}$ = 0; the odd bank becomes enabled for data transfer and even bank is disabled. $\overline{R}_D = 0$ and $M/\overline{IO} = 1$ for 8086, the odd memory places data on D_{15}–D_8 bus. During the first bus cycle the lower byte is accessed from memory location 20003H as depicted in Fig. 9.12(a). In the second cycle, $A_0 = 0$ and $\overline{BHE} = 1$, the even bank

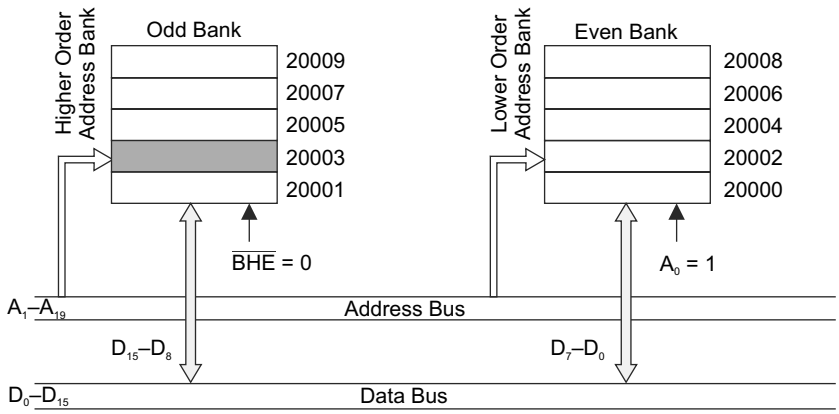


Fig. 9.12 (a) First cycle to access one byte from odd bank memory addressing

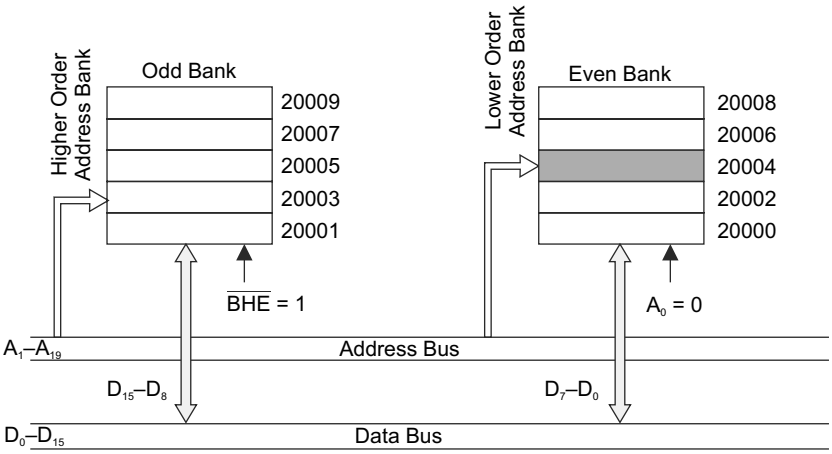


Fig. 9.12 (b) Second cycle to access one byte from even bank memory addressing

of memory becomes enabled and the odd bank is disabled. Then processor output $\overline{R}_D = 0$ and $M/\overline{IO} = 1$. The selected even bank memory location content is on D_7–D_0 bus. Then data is to be accessed. Therefore, during the second bus cycle, the upper byte is accessed from even address bank memory location 20004H.

9.7 PIN DESCRIPTION OF 8086

The pin diagram of the 8086 has been shown in Fig. 9.13. The 8086 can operate either in minimum mode or in maximum mode depending upon the status of the pin MN/$\overline{MX}$. When MN/$\overline{MX}$ = 5 V, the 8086 works in minimum mode meaning that 8086 operates in single processor environment. If MN/$\overline{MX}$ = GND, it works in maximum mode and the processor can be operated in multiprocessor environment. To differentiate the minimum and maximum mode operation, a set of the 8086 pins change their functions, but other pins have common functions in both the modes. The pin description of 8086 is as follows:

AD_{15}–AD_0 (Bi-directional) Address/Data Bus These lines constitute the time-multiplexed address/data bus. These lines are low-order address bus. They are act as address bus during the first clock cycle. When AD lines are used to transmit memory/IO address, the symbol A is used of AD. For example, A represents A_{15}–A_0. When data are transmitted through AD lines, the symbol D is used in place of AD. For example, D represents D_7–D_0, D_{15}–D_8 or D_{15}–D_0.

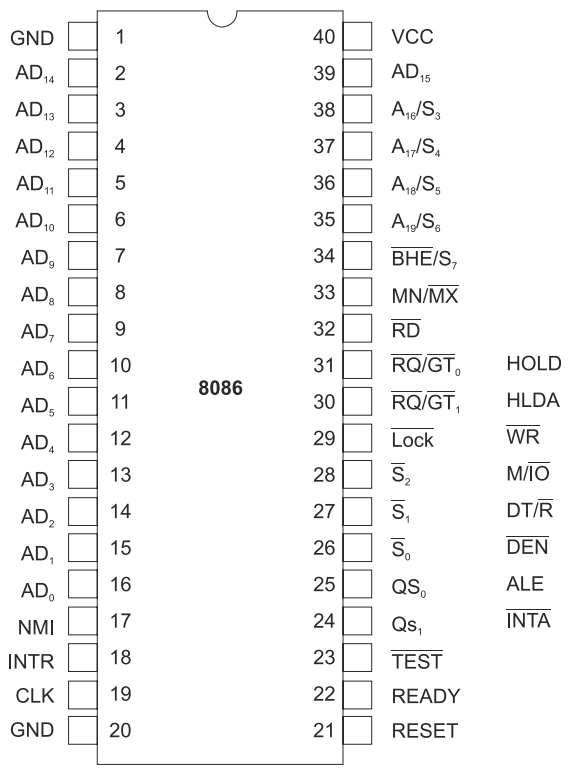


Fig. 9.13 Pin diagram of 8086

A_{19}–A_{16} (Output) These are high-order address lines and they are time multiplexed lines. During T_1, these lines can be used as higher-order 4 bits of memory address. But in I/O operation, these lines are low. During T_2, T_3, and T_4, they carry status signals.

A_{16}/S_3, A_{17}/S_4 (Output) A_{16} and A_{17} are time multiplexed with segment identifier signals S_3 and S_4. During T_1 clock cycle, A_{16} and A_{17} are used as address bits. In T_2 to T_4 clock cycle, these lines carry status signals. Table 9.5 shows memory segment identification.

Table 9.5 Memory segment identification

S_4	S_3	Function
0	0	Extra segment memory access
0	1	Stack segment memory access
1	0	Code segment memory access
1	1	Data segment memory access

A_{18} / S_5 (Output) A_{18} is time multiplexed with interrupt status S_5. During T_1 clock cycle, A_{18} is transmitted to address bus. During other clock cycles (T_2, T_3 and T_4), the status signal S_5 is transmitted through this line. S_5 is an interrupt enable status signal. At the beginning of each clock cycle, the status of the interrupt enable flag S_5 is updated.

A_{19} / S_6 (Output) A_{19} is multiplexed with status signal S_6. During T_1 clock cycle, A_{19} is transmitted to address bus. During T_2 to T_4, the status signal S_6 is available on this line. It is low during T_2 to T_4.

$\overline{BHE}/S_7$ (Output) Bus High Enable/Status During T_1, the bus high enable signal $\overline{BHE}$ can be used to enable data onto the most significant half of data bus, D_{15}–D_8. 8-bit devices connected to upper half of the data bus use $\overline{BHE}$ signal to condition ship select functions. $\overline{BHE}$ is low during T_1 for read, write and interrupt acknowledge cycles when a byte is to be transferred on the higher portion of the data bus. This pin is multiplexed with status signal S_7. The S_7 status signal is available during T_2 to T_4. The signal is active low, and floats to 3 state OFF in hold. It is low during T_1 for the first interrupt acknowledge cycle. Table 9.6 shows the function of $\overline{BHE}$ and A_0.

Table 9.6 Function of $\overline{BHE}$ and A_0

$\overline{BHE}$	A_0	Function
0	0	Whole word
0	1	Upper byte from/to odd address
1	0	Lower byte from/to even address
1	1	None

$\overline{RD}$ (Output) (Read) This control signal is used for read operation. It is an output signal. It is active when it is LOW. Read signal indicates that the processor is performing a memory or I/O read cycle, depending on the state of the S_2 pin. This signal is used to read devices which reside on the 8086 bus. $\overline{RD}$ is active low during T_2, T_3 and T_w of any read cycle and is guaranteed to remain high in T_2 until the 8086 local bus floated. This signal floats to tristate in hold acknowledge.

READY (Input) The addressed I/O or memory devices send acknowledgment through this pin and it indicates that the data transfer is completed. The READY signal from memory or I/O devices is synchronised with the 8284A clock generator to provide READY input to 8086. This signal is active HIGH. When READY is HIGH, it indicates that the peripheral is ready to transfer data.

INTR (Interrupt request) It is a level triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt vector look-up table located in system memory. It can be internally masked by software resetting the interrupt enable bit. INTR is internally synchronised. This signal is active high.

$\overline{\text{TEST}}$ (Input) This is used in conjunction with the WAIT instruction. If the $\overline{TEST}$ input is LOW, execution continues, otherwise the processor waits in an idle state. This input is synchronised internally during each clock cycle on the leading edge of CLK. When it is low, the microprocessor continues execution, otherwise it waits.

NMI (Input) Nonmaskable interrupt This is an edge-triggered input which causes a type 2 interrupt. A subroutine is vectored via an interrupt vector look-up table located in system memory. NMI is not maskable internally by software. A transition from LOW to HIGH initiates the interrupt at the end of the current instruction. This input is internally synchronised.

MN/$\overline{\text{MX}}$ (Input) The minimum/maximum signal indicates the operating mode of 8086. When it is high, the 8086 processor operates in minimum mode. If this pin is low, the processor operates in maximum mode.

RESET (Input) If the reset signal is active HIGH, processor immediately terminates its present activity and system is reset. The signal must be active high for at least four clock cycles. It restarts execution, as described in the instruction set, when RESET returns low. RESET is internally synchronised.

CLK (Input) The signal provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimised internal timing. Nonmaskable interrupt request.

VCC Power supply, +5 Vdc.

GND Ground.

9.7.1 Operating Modes of 8086

There are two different operating modes for Intel 8086, namely, the minimum mode and the maximum mode. When only one 8086 processor is to be used in a microcomputer system, the 8086 is used in the *minimum mode* of operation. In this mode, the CPU issues the control signals required by memory and I/O devices. In a multiprocessor system, the 8086 processor operates in the maximum mode. In *maximum mode* operation control signals are issued by Intel 8288 bus controller which is used with 8086 for this very purpose. The level of the pin *MN/$\overline{MX}$* decides the operating mode of 8086. When *MN/$\overline{MX}$* is high, the CPU operates in the minimum mode. When it is low, the CPU operates in the maximum mode. The pins 24 to 31 of 8086 issue two different sets of signals. One set of signals is issued when the processor operates in the minimum mode. The other set of signals is issued while the processor operates in the maximum mode. Thus the pins from 24 to 31 have alternate functions. In this section, the pin description of minimum mode and maximum mode operations are discussed.

9.7.2 Pin Description in Minimum Mode

In the minimum-mode operation, the pin *MN/$\overline{MX}$* is connected to 5V dc supply. The minimum mode operation is cheaper as all control signals for memory and I/O are generated by the microprocessor. The schematic pin diagram of 8086 for minimum mode operation is illustrated in Fig. 9.14. The functions of all pins except pins of 24 to 31 of 8086 are same in this mode. The pins description of 24 to 31 for the minimum mode are as follows:

$\overline{INTA}$ (Output) Interrupt Acknowledge This signal is used as a read strobe for interrupt acknowledge cycles. It is active low during T_2, T_3 and T_w of each interrupt acknowledge cycles. On receiving an interrupt signal, the processor issues an active LOW interrupt acknowledge signal.

ALE (Output) Address Latch Enable This signal is provided by the processor to latch the address into the 8282/8283 address latch. It is HIGH during T_1.

$\overline{DEN}$ (Output) Data Enable This signal can be provided as an output enable for the 8286/8287 in a minimum system which uses the transceiver. $\overline{DEN}$ is active low during each memory and I/O access and for INTA cycles. For a read or $\overline{INTA}$ cycle, it is active from the middle of T_2 until the middle of T_4, while for a write cycle it is active from the beginning of T_2 until the middle of T_4. $\overline{DEN}$ floats 3 state OFF in local bus hold acknowledge. When Intel 8286/8287 octal bus transceiver is used, this signal acts as an output enable signal. It is active LOW.

DT/$\overline{R}$ (Output) Data Transmit/Receive DT/$\overline{R}$ signal is required in minimum mode operation, when 8286/8287 data bus transceiver is used for data flow. This signal is used to control the direction of data flow through the transceiver. When it is HIGH, CPU send data to I/O device. When it is LOW, the CPU wants to access I/O device. Logically DT/$\overline{R}$ is equivalent to $\overline{S}_1$ in the maximum mode, and its timing is same as for $M/\overline{IO}$. This signal floats to 3 state OFF in local bus hold acknowledge.

$\overline{WR}$ (Output) Write This pin indicates that the processor is performing a memory write or I/O write cycle, depending on the state of the $M/\overline{IO}$ signal. $\overline{WR}$ is active for T_2, T_3 and T_w of any write cycle. It is active low and floats to 3 states OFF in local bus hold acknowledge. When it is LOW, the CPU performs memory or I/O write operation.

HLDA (Output) HOLD Acknowledge This signal is issued by the processor when it receives a HOLD signal. This is active HIGH signal. When HOLD request is removed, HLDA goes LOW.

HOLD (Input) Hold The HOLD signal indicates that another device (master) in the microcomputer

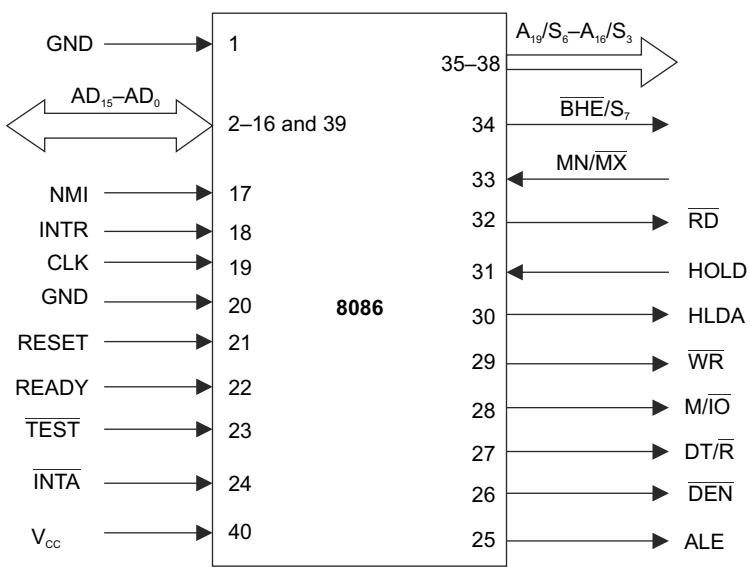


Fig. 9.14 Pin diagram of 8086 for minimum mode of operation

system is requesting for direct memory access or a local bus hold for using the address and data bus. Then the master sends a HOLD request to the processor through this pin. It is an active HIGH signal.

9.7.3 Pin Description in Maximum Mode

For the maximum mode of operation, the pin MN/$\overline{MX}$ is grounded. The maximum mode is designed to be used when a coprocessor exists in the system. The schematic pin diagram of 8086 for maximum mode operation is depicted in Fig. 9.15. In this mode, the functions of all pins except pins 24 to 31 of 8086 are same. The description of the pins from 24 to 31 is as follows.

QS_1, QS_0 (Output) Instruction Queue Status These two signals are decoded to provide instruction queue status as given below:

Table 9.7 Functions of queue status signals

QS_1	QS_0	Function
0	0	No operation
0	1	First byte of opcode from queue
1	0	Empty the queue
1	1	Subsequent byte from queue

$\overline{S}_2$, $\overline{S}_1$ and $\overline{S}_0$ (Output) Status Signals These signals are connected to the bus controller Intel 8288. The bus controller decodes these signals to generate eight separate memory and I/O access control signals as depicted in Table 9.8 which shows the logic for status signals.

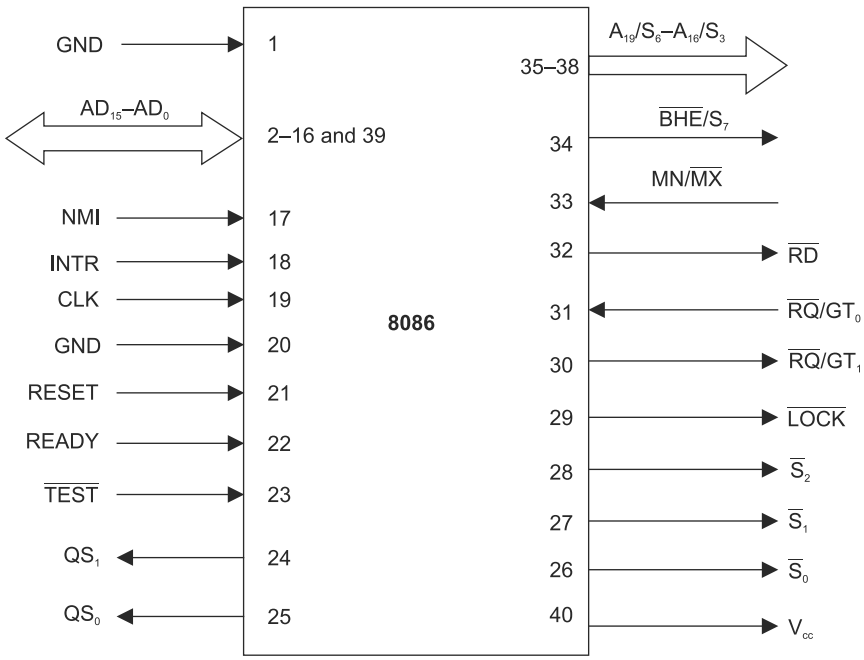


Fig. 9.15 Pin diagram of 8086 for maximum mode of operation

Table 9.8 Logic for status signals $\overline{S}_2$, $\overline{S}_1$ and $\overline{S}_0$

$\overline{S}_2$	$\overline{S}_1$	$\overline{S}_0$	Function
0	0	0	Interrupt acknowledge
0	0	1	Read data from I/O port
0	1	0	Write data into I/O port
0	1	1	Halt
1	0	0	Code access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive

$\overline{LOCK}$ (Output) This output pin indicates that other system bus masters are not to gain control of the system bus while $\overline{LOCK}$ is active low. The $\overline{LOCK}$ signal is activated by the LOCK prefix instruction and remains active until the completion of the next instruction. This signal is active low and floats to 3 states OFF in hold acknowledge.

$\overline{RQ}$ / $\overline{GT}_0$ and $\overline{RQ}$ / $\overline{GT}_1$ Request/Grant These pins are used by other processors in a multiprocessor environment. Local bus masters, of other processors force the processor to release the local bus at the end of the processor's current bus cycle. Each pin is bidirectional with $\overline{RQ}/\overline{GT}_0$ having higher priority than $\overline{RQ}/\overline{GT}_1$ and has internal pull-up resistors. Hence, they may be left unconnected.

9.8 MEMORY READ AND WRITE BUS CYCLE OF 8086

The bus cycle means the Bus Interface Unit (BIU) phenomenon. It is known to us that the EU and BIU work asynchronously. The EU takes instructions from the instruction queue and executes instructions in a number of clock periods continuously. These clock periods are not grouped to form machine cycles. Therefore, EU does not use machine cycles. The EU waits for BIU to handover the instruction whenever the program starts or the program executes any branch instruction or the BIU is executing data memory access for EU. The BIU fetches the instruction code from memory, reads data from memory or I/O devices, and writes data into memory or I/O devices. The clock periods are grouped whenever the BIU accesses memory and I/O devices for read and write operations. When any external memory or I/O devices are accessed, only four clock cycles are required to perform read or write operation. These four clock cycles are grouped which is called *bus cycle*. In this section, memory and I/O read and write bus cycles for both minimum and maximum modes are discussed.

9.8.1 Minimum Mode Operation and Timing Diagram

Figure 9.16 shows the minimum mode configuration of 8086. This figure shows a group of ICs which generate address bus, data bus and control bus signals. The ROM and RAM ICs and I/O ports are connected with CPU through these buses. The 8282 Latch ICs are used to latch the address from 8086 processor address/data bus. A_{15}–A_0, A_{19}–A_{16} and $\overline{BHE}$ are latched during T_1 state. The output enable ($\overline{OE}$) of 8288 I/O ports are grounded, therefore the bus will not be floated or high impedance state. The ALE signal from 8086 is used to strobe address lines in 8282 latches. As data bus is bidirectional, the bidirectional transceivers 8286 ICs are used. The working principle of 8086 in minimum mode can be explained with timing diagram. The opcode fetch cycle is similar to memory read cycle. In this section, memory read and write cycles are explained in detail. The types of data transfer depend on $M/\overline{IO}$, $\overline{RD}$ and $\overline{WR}$ signals as depicted in Table 9.9.

Table 9.9 Types of data transfer

$M/\overline{IO}$	$\overline{RD}$	$\overline{WR}$	Operation
0	0	1	I/O read
0	1	0	I/O write
1	0	1	Memory read
1	1	0	Memory write

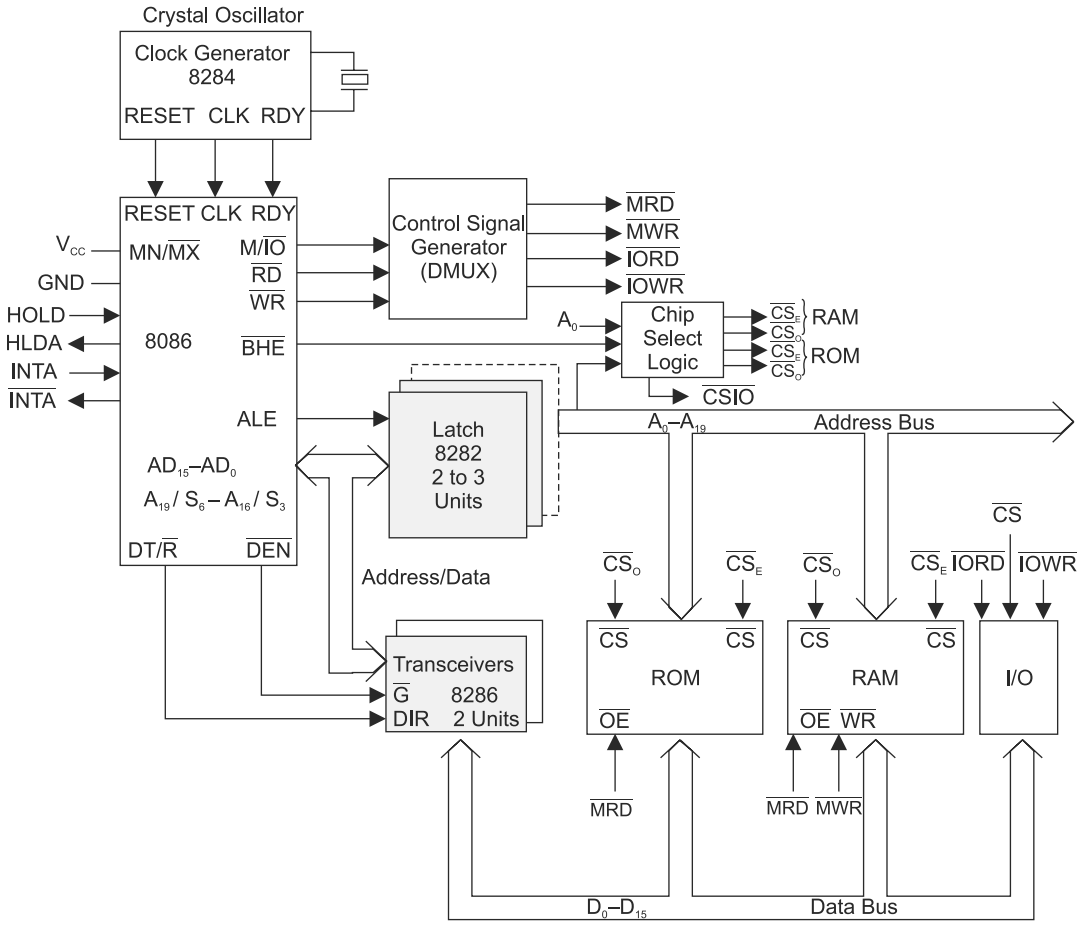


Fig. 9.16 Minimum mode configuration of 8086

Memory Read Bus Cycle for Minimum Mode During minimum mode operation of 8086 processor, MN/$\overline{MX}$ is +5 DC. The timing diagram for memory read bus cycle is shown in Fig. 9.17. The following actions take place during four different clock cycles:

T_1 Clock Cycle During T_1 clock cycle, the 8086 processor sends the 20-bit address on the address bus. The 16-bit least significant bits of address sends on AD_{15}– AD_0 and four most significant bits of address put on A_{19}–A_{16} lines.

- ALE is high only during T_1. Address is put in the address bus. The falling edge of ALE is used to latch the address from the address bus.
- $\overline{BHE}$ is high or low depending on 8- or 16-bit read from odd/even address.
- $M/\overline{IO}$ is high to indicate memory operation. It remains high during the entire bus cycle.
- DT/$\overline{R}$ is low and remains low throughout the complete bus cycle, to indicate the direction of data transfer as memory to the processor.

T_2 Clock Cycle During T_2 clock cycle, the AD_{15}–AD_0 go into high impedance state. This is indicated by dotted lines as depicted in Fig. 9.17 as bus drivers are disabled.

- $\overline{RD}$ goes low during T_2. This signal can be used by the selected memory IC to enable its output buffer. This is read-control signal.

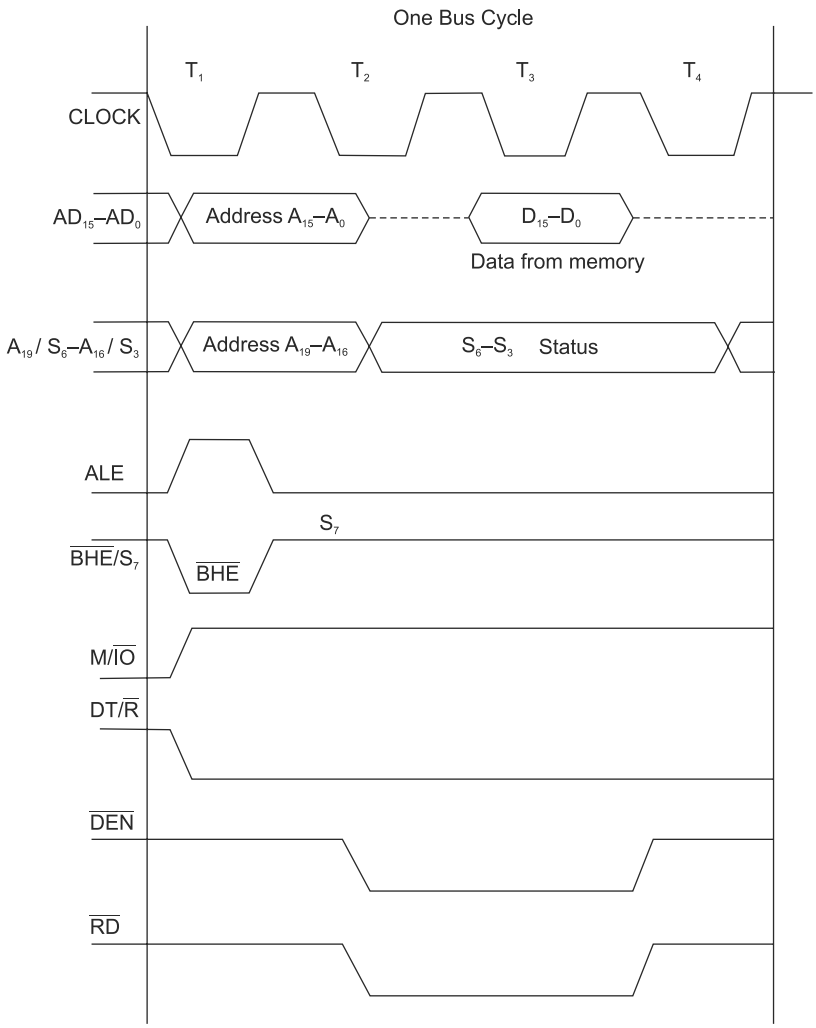


Fig. 9.17 Memory read bus cycle of 8086 for minimum mode

When READY signal is high during T_2, it indicates that the memory device is ready and the 8086 operates normally according to the bus cycle. The READY signal is used by slow memory devices. If the memory is not ready to transfer data, it does not make READY signal high otherwise the READY is low. If READY is low during T_2, the 8086 inserts a wait state between T_3 and T_4. The 8086 processor always takes necessary steps to read data from the memory only when READY becomes high.

- $\overline{DEN}$ goes high to enable the 8286/8287 transceiver
- $\overline{BHE}$ goes high
- During T_2 to T_4, Status signals S_3, S_4, S_5 and S_6 are put on the address lines A_{16}/S_3 to A_{19}/S_6. S_3 and S_4 are used to identify the memory segment. S_5 indicates interrupt enable status. The status of the interrupt enable flag bit, S_5 is updated at the beginning of each clock cycle.

T_3 Clock Cycle

$\overline{DEN}$ is low

- Data is put on lines AD_0–AD_{15}.

T_4 Clock Cycle

- $M/\overline{IO}$ goes low just after T_4 clock cycle.
- $\overline{RD}$ goes high.
- All bus signals are deactivated and prepared for the next bus cycle.

Memory Write Bus Cycle for Minimum Mode The timing diagram for memory write bus cycle is shown in Fig. 9.18. The following actions take place during four different clock cycles:

T_1 Clock Cycle The write cycle is similar to read cycle, but there are some differences. During T_1 clock cycle, the 8086 processor sends the 20-bit address on the address bus. The 16-bit least significant bits of address sends on AD_{15}–AD_0 and four most significant bits of address put on A_{16}–A_{19} lines.

- ALE is high only during T_1. Address is put in the address bus. The falling edge of ALE is used to latch the address from the address bus.
- $\overline{BHE}$ is high or low depending on 8- or 16-bit read from odd/even address.
- $M/\overline{IO}$ is high to indicate memory operation. It remains high during the entire bus cycle.
- DT/$\overline{R}$ is high and remains high throughout the complete bus cycle, to indicate the direction of data transfer from the processor to memory.
- $\overline{DEN}$ goes high to enable the 8286/8287 transceiver.

T_2 Clock Cycle

- $\overline{WR}$ is low as write control signal.
- $\overline{BHE}$ goes high.
- Status is put on the A_{19}–A_{16} lines. The activity starts during T_2 and continues till T_4.

T_3 Clock Cycle

- Data is put on the AD_{16}–AD_0 lines.

T_4 Clock Cycle

- $\overline{WR}$ becomes high.
- $M/\overline{IO}$ goes low.

- $\overline{DEN}$ is low.
- DT/$\overline{R}$ goes low.

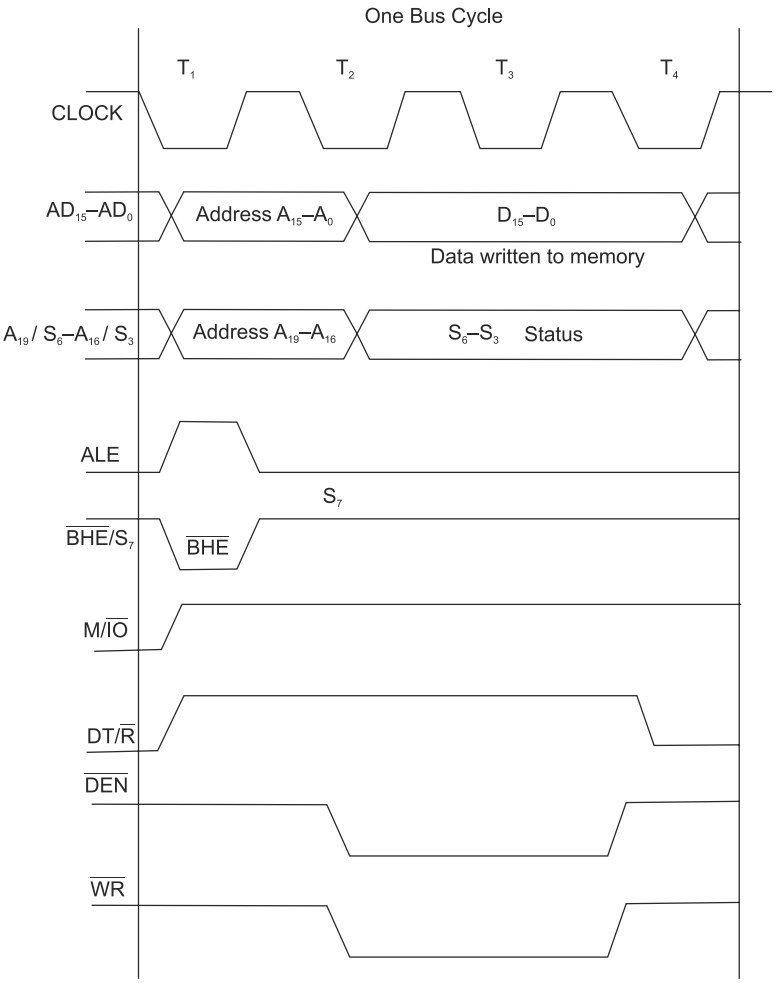


Fig. 9.18 Memory write bus cycle of 8086 for minimum mode

I/O READ and Write BUS CYCLE The I/O read bus cycle is similar to memory read cycle. The *M/$\overline{IO}$* signal is low for I/O read operation and all other signals are same as memory read operation as illustrated in Fig. 9.17 and can be used as I/O read bus cycle by changing the *M/$\overline{IO}$* signal only. The I/O write bus cycle is also similar to memory write cycle. The *M/$\overline{IO}$* signal is low for I/O write operation and all other signals are same for memory write operation as depicted in Fig. 9.18.

9.8.2 Maximum Mode Operation and Timing Diagram

Figure 9.19 shows the maximum-mode configuration of 8086 processor. In the maximum mode, *MN/$\overline{MX}$* pin of 8086 is strapped to grounded and pin 24 to 31 of 8086 are active in maximum mode. In this mode, microprocessor status signals $\overline{S}_2$, $\overline{S}_1$, and $\overline{S}_0$ are fed to an 8288 bus controller. The bus controller interprets status

signals $\overline{S}_2$, $\overline{S}_1$, and $\overline{S}_0$ generate bus timing and control signals. Here, the 8288 bus controller derive $\overline{RD}$, $\overline{WR}$, DEN, *DT/R* and ALE, $\overline{MRDC}$, $\overline{MWTC}$, $\overline{AMWC}$, $\overline{IORC}$, $\overline{IOWC}$ and $\overline{AIOWC}$ control output signals.

$\overline{IORC}$, $\overline{IOWC}$ are I/O read command and I/O write command signals respectively. These signals enable I/O read or write operations. $\overline{MRDC}$ and $\overline{MWTC}$ are used as memory read and memory write command signals respectively.

In this mode, more than one processors are interconnected within a system. $\overline{AEN}$, IOB and CEN pins are very useful for multiprocessor systems. $\overline{AEN}$, and IOB are grounded, but CEN pin is connected with +5 V. The $\overline{MCE/PDEN}$ pin output depends upon the status of the IOB. When IOB is grounded, this pin works as a master cascade enable to control cascaded 8259A. When IOB is not grounded, this pin acts as a peripheral data enable. $\overline{INTA}$ pin is used to generate two interrupt acknowledge signals and fed to the interrupt controller.

The working principle of 8086 in the maximum mode can be explained with timing diagram. The maximum mode timing diagram is subdivided into memory read cycle and memory write cycle. The address/data and address/status timings are similar to the minimum mode. In this section memory read and writes cycles are explained in detail.

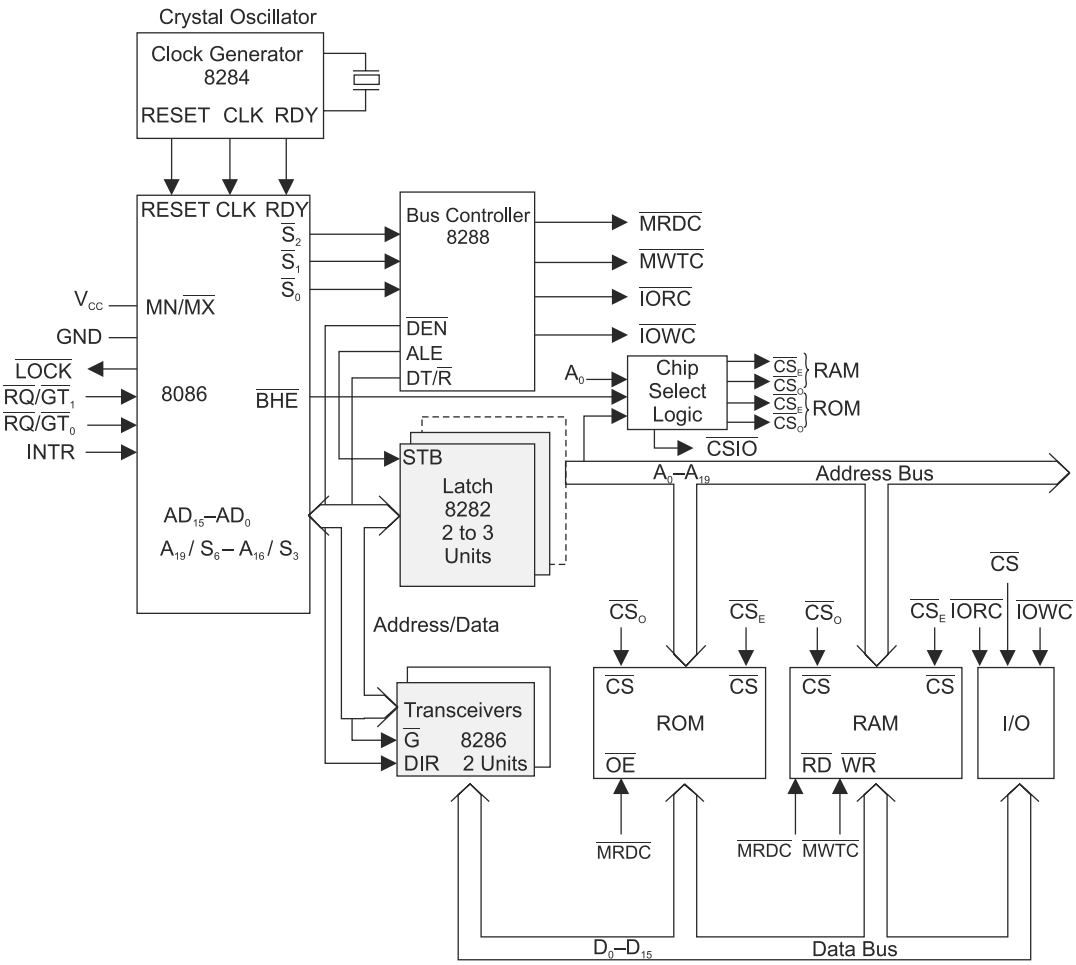


Fig. 9.19 Maximum mode configuration of 8086

Memory Read Bus Cycle for Maximum Mode During minimum-mode operation of 8086 processor, MN/$\overline{MX}$ is grounded. The timing diagram for memory read bus cycle in maximum mode operation of 8086 is shown in Fig. 9.20. The following actions take place during four different clock cycles:

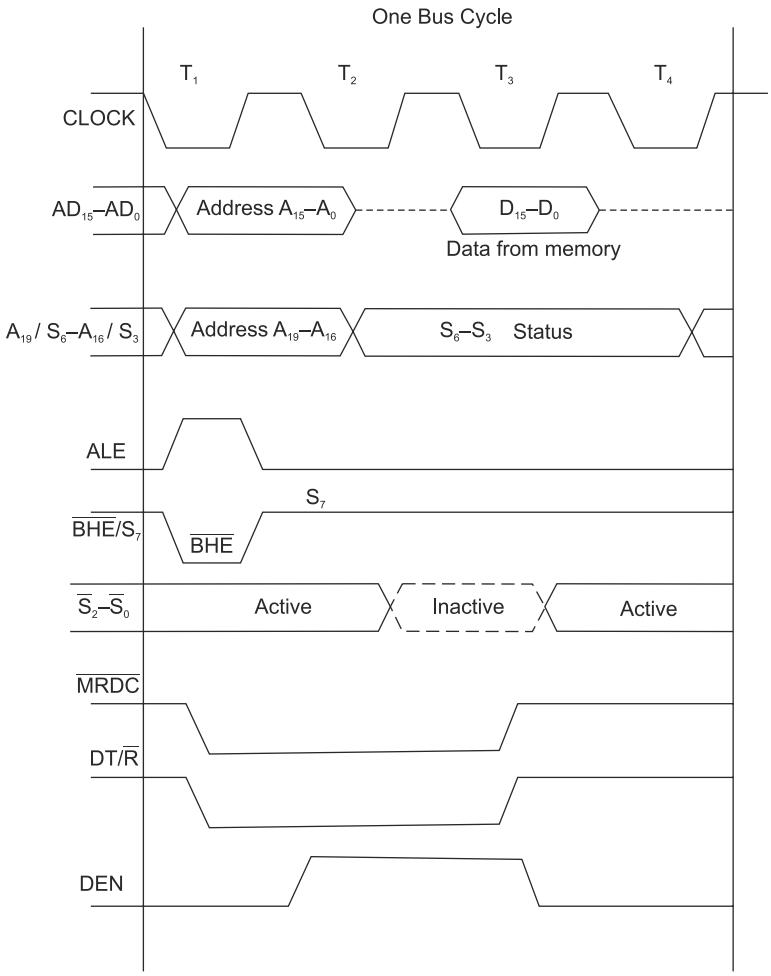


Fig. 9.20 Memory read bus cycle of 8086 for maximum mode

T_1 Clock Cycle

- $\overline{S}_2$, $\overline{S}_1$, and $\overline{S}_0$ are set by 8086 in the starting of clock cycle. These signals are decoded by the 8288 bus controller.
- ALE is high only during T_1. Address is put on the address bus. The falling edge of ALE is used to latch the address from the address bus.
- $\overline{BHE}$ is high or low depending on 8- or 16-bit read from odd/even address.
- DT/$\overline{R}$ goes low.

T_2 Clock Cycle

- $\overline{BHE}$ is high.
- $\overline{DEN}$ goes high to enable the 8286/8287 transceiver.
- $\overline{MRDC}$ is low for memory read control signal.

T_3 Clock Cycle

- Data is put on address/data lines from memory.
- The status signals $\overline{S}_2$, $\overline{S}_1$, and $\overline{S}_0$ become inactive.

T_4 Clock Cycle

- $\overline{MRDC}$ is high.
- $\overline{DEN}$ goes low to disable the 8286/8287 transceiver.
- DT/$\overline{R}$ goes high.
- The READY signal is sampled at the end of T_2. If it is low, wait states are inserted between T_3 and T_4.

Memory Write Bus Cycle for Maximum Mode The timing diagram for memory read bus cycle in maximum mode operation of 8086 is shown in Fig. 9.21. The following actions take place during four different clock cycles:

T_1 Clock Cycle

- $\overline{S}_2$, $\overline{S}_1$, and $\overline{S}_0$ are set by 8086 in the beginning of clock cycle. These signals are decoded by the 8288 bus controller.
- ALE is high only during T_1. Address is put on the address bus. The falling edge of ALE is used to latch the address from the address bus.
- $\overline{BHE}$ is high or low depending on 8- or 16-bit read from odd/even address.

T_2 Clock Cycle

- $\overline{BHE}$ is high.
- $\overline{DEN}$ goes **high** to enable the 8286/8287 transceiver.

T_3 Clock Cycle

- $\overline{MWTC}$ is low for memory write control signal.
- Data is put on address/data lines.
- The status signals $\overline{S}_2$, $\overline{S}_1$, and $\overline{S}_0$ become active.

T_4 Clock Cycle

- $\overline{MWTC}$ is high.
- $\overline{DEN}$ goes low to disable the 8286/8287 transceiver.

I/O READ and Write BUS CYCLE I/O write bus cycle is similar to memory write cycle. The memory write operation can be performed by write control signals $\overline{AMWTC}$ but $\overline{AIOWC}$ control signal is used for I/O write. Therefore, in I/O write bus cycle using $\overline{AIOWC}$, $\overline{AMWTC}$ will be replaced by $\overline{AIOWC}$ as depicted in Fig. 9.21. Similarly, $\overline{MRDC}$ will be replaced by $\overline{IORC}$ in memory read cycle timing diagram as I/O read bus cycle is similar to memory read cycle as given in Fig. 9.20.

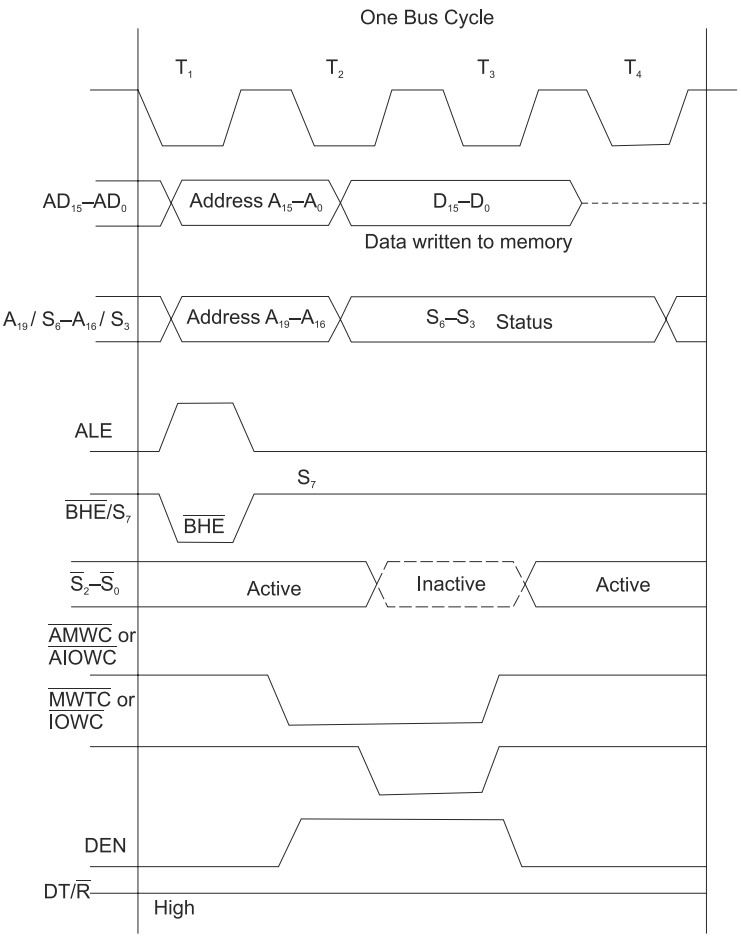


Fig. 9.21 Memory write bus cycle of 8086 for maximum mode

9.9 INTEL 8088 PROCESSOR

The 8086 processor has tremendous flexibility in programming as compared to 8085. Therefore, after the introduction of 8086, research work was done for a microprocessor chip which has the programming flexibility like 8086 and the external interface ICs of 8085, so that all existing circuits of 8085 can be compatible with the new processor. Then 8088 processor was developed. The Intel 8088 is an 8-bit microprocessor. Its internal architecture is same as that of 8086, i.e., architecture of a 16-bit microprocessor. But the 8088 has only 8 data lines and hence it can use 8-bit I/O devices which are cheaper compared to 16-bit I/O devices. Personal computers based on 8088 CPU are cheaper compared to personal computers based on 8086 and 80286 CPU. The clock frequency of 8088 is 5 MHz and that for 8088-2 is 8 MHz. The 8088 microprocessor is available in 40-pin IC and operates at +5 V dc supply. Its register set, instructions and addressing modes are same as those of 8086. Its CHMOS version 80C88A operates at 8 MHz clock. The instruction queue length in case of 8088 processor is of 4 bytes whereas that in 8086 is of 6 bytes. The 8088 CPU uses 8087 math coprocessor, 20-bit address bus and can directly address up to 1 MB of memory.

A computer built around 8088 CPU uses 8284 clock generator, 8282 latches, 8286 transceivers, 8288 bus controller, 8087 math coprocessor, 8237 DMA controller, 8259 interrupt controller, etc. The 8088 CPU was very popular and widely used in personal computer, PC/XT. Presently 8088-based computers are no longer manufactured. In this section, the architecture of 8088 processor, pin description, addressing and timing diagram are explained.

9.9.1 Architecture of 8088 Microprocessor

Figure 9.22 shows the architecture of the 8088 processor. The set of registers of 8088 is approximately same

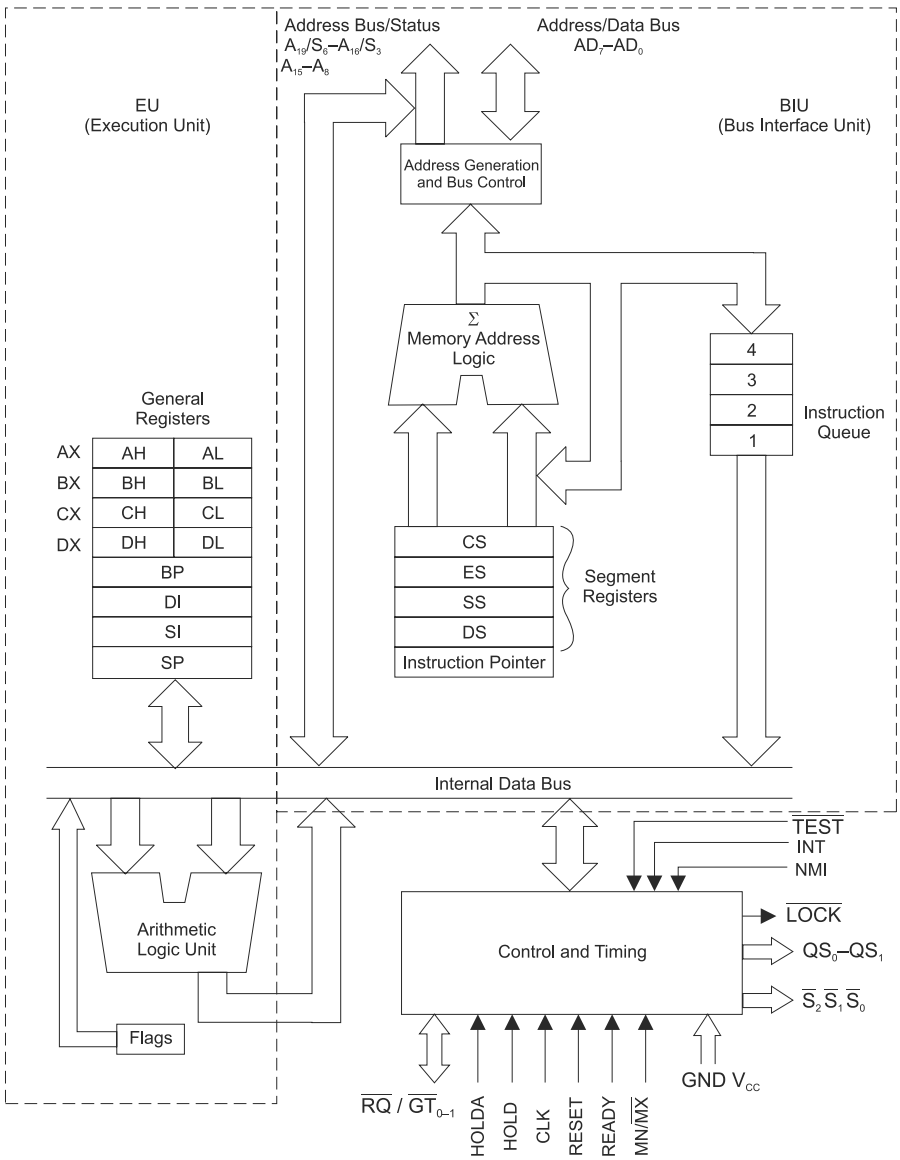


Fig. 9.22 Block diagram of 8088 microprocessor

as that of 8086. The architecture of 8088 is same as 8086 architecture, but there are two changes. The 8088 has a 4-byte instruction queue in place of 6-byte instruction queue in 8086 and data bus of 8088 is 8-bit. The other function blocks are the same as 8086 processor.

The 8088 processor has 1 Mbyte addressing capability and it has 20-bit address or 20 addressing lines. The concept of segmented memory and the computational method of physical address are used in 8088 processor without any change and it is same as 8086 processor. The memory organisation and addressing methods of 8088 are similar like 8086. There is no concept of even address bank and odd address bank of memory in 8088. Therefore, the complete memory is consistently addressed as a bank of 1Mbyte memory locations with the help of the segmented memory concept.

As the data bus is 8 bit, the 8088 can access only a byte at a time. Therefore, the speed of operation of 8088 will be reduced as compared to 8086, though internal data bus of 8088 is 16 bit and it can process the 16-bit data internally. Due to change in address and data bus structure, the timing diagrams of 8088 are different from 8086.

9.9.2 Pin Description of 8088 Processor

The pin diagram of 8088 is depicted in Fig. 9.23. The functions of 8088 pins except AD_7–AD_0, AD_{15}–AD_8,

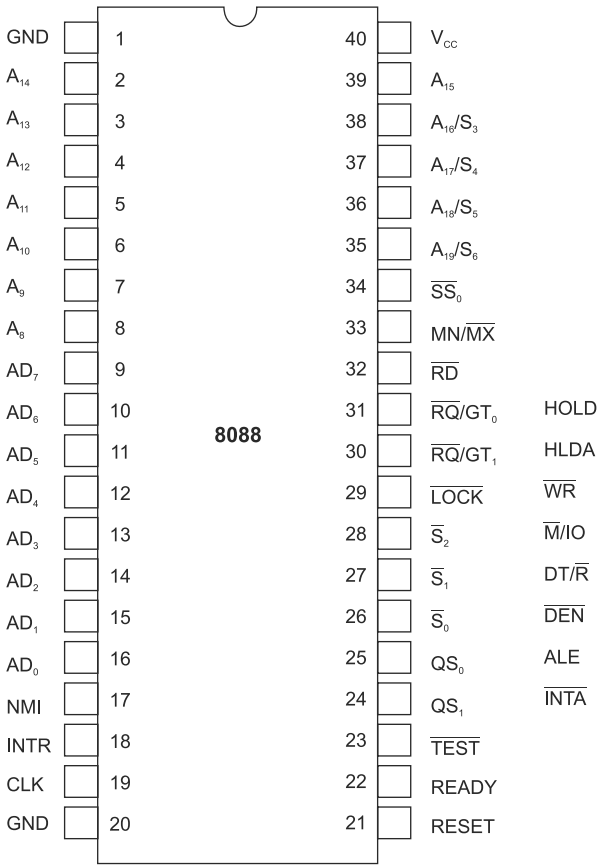


Fig. 9.23 Pin diagram of 8088

$\overline{SS}_0$ and $IO/\overline{M}$ pins are exactly same as the pins of 8086. Consequently, the pins functions of $\overline{SS}_0$, $IO/\overline{M}$, AD_7–AD_0, A_{15}–A_8 are explained in this section.

AD_7–AD_0 (Address/Data Bus) The AD_7–AD_0 lines are operated as time multiplexed address/data bus. When ALE signal is high, these lines can be used as the address of the lower order memory location address or I/O port address. When ALE is low, these lines are used as data bus D_7–D_0. During T_1 clock cycle, the AD_7–AD_0 bus is used as addresses bus. During T_2, T_3, and T_4, states these lines are used as data bus. These lines are in high impedance state in hold acknowledged and interrupt acknowledge cycles.

A_{15}–A_8 (Address Bus) The A_{15}–A_8 lines are used as lower order memory location address throughout the entire bus cycle. During hold acknowledge, these address lines are tristated or high impedance state.

$\overline{SS}_0$ This pin is newly introduced in 8088 processor instead of $\overline{BHE}$ pin in 8086. During minimum-mode operation, the pin $\overline{SS}_0$ is equivalent to the $\overline{S}_0$ in the maximum mode. $\overline{SS}_0$ is always high in maximum mode.

$IO/\overline{M}$ The $IO/\overline{M}$ pin is similar to the $M/\overline{IO}$ pin of 8086. The function of this pin is used to operate the 8088 processor as an 8085 processor, interfacing of memory and I/O devices.

In the minimum mode, the operations of the 8088 microprocessor depends on control signals $\overline{SS}_0$, $IO/\overline{M}$ and $DT/\overline{R}$ as given in Table 9.10. The $\overline{SS}_0$ pin is always high in the maximum-mode operation. The functions and timings of other pins of 8088 are same as 8086.

Table 9.10 Operation of 8088 processors based on control signals

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

9.9.3 Timing Diagram of 8088 Microprocessor

Each bus cycle of the 8088 processor consists of four T states, T_1, T_2, T_3 and T_4. During the first clock cycle T_1, ALE signal is high and A_{19}/S_6–A_{16}/S_3 are used as A_{19}–A_{16} address bus, and AD_7–AD_0 can be used as A_7–A_0 address bus. Hence the leading edge of ALE is used to latch the valid 20-bit address during T_1 states. After T_1 state A_{19}/S_6–A_{16}/S_3 are used as status signals S_6–S_3, the middle bus A_{15}–A_8 are always active as address bus but AD_7–AD_0 are tristated. During T_3 and T_4, data is read from memory and placed into data bus. Therefore, AD_7–AD_0 is used as data bus in T_3 and T_4 duration. Figure 9.24 shows the timing diagram for memory read operation of 8088 microprocessor. After T_4, the next bus cycle will be started.

Figure 9.25 shows the timing diagram of 8088 for memory write bus cycles. In the memory cycle, data will be available on the data bus during T_2, T_3 and T_4 and the status signals are valid for T_2, T_3 and T_4 duration. After T_4 address/data bus, AD_7–AD_0 are tristated.

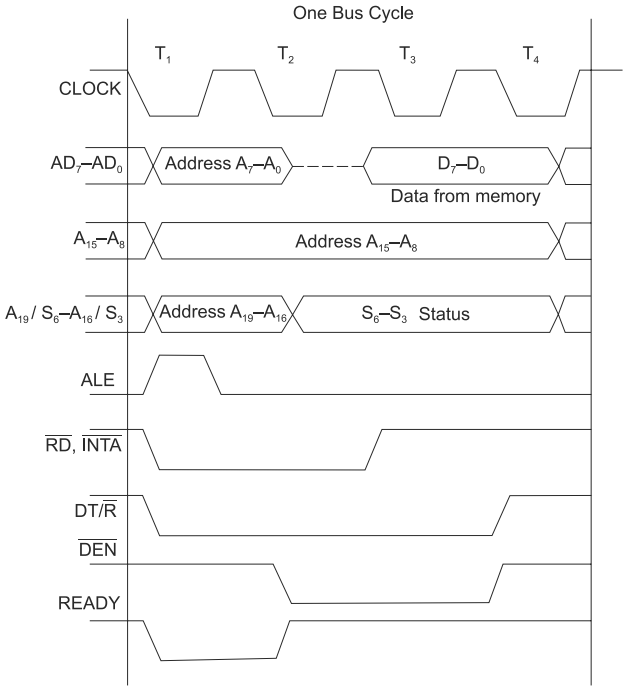


Fig. 9.24 Memory read bus cycle of 8088

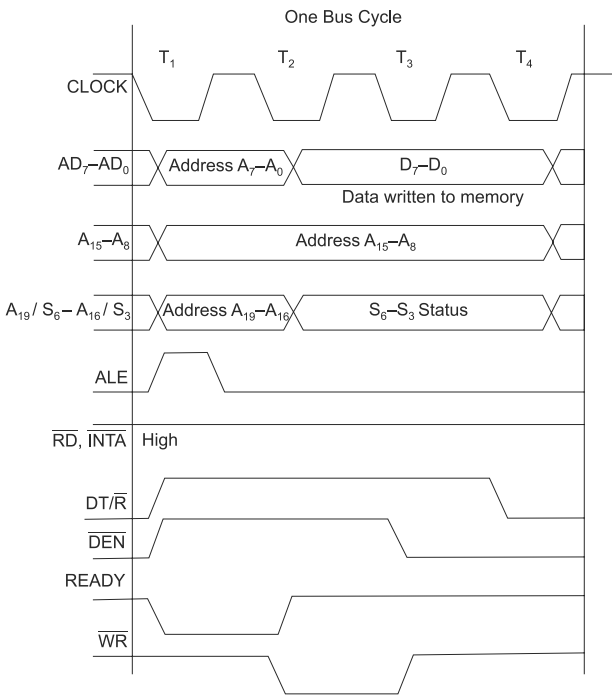


Fig. 9.25 Memory write bus cycle of 8088

9.10 DEMULTIPLEXING OF SYSTEM BUS IN 8086 AND 8088 MICROPROCESSOR

In the 8086/8088 processor, there are three buses: address, data and control buses. The address/data buses are operated in time multiplexed. The address bus is required to locate memory and I/O devices for data transfer through memory and I/O read or write cycles. The data bus is used to transfer data from microprocessor to memory/ I/O devices or vice versa. The control bus provides control signals to memory/I/O devices for data transfer operations.

9.10.1 Demultiplexing of System Bus in 8086

The 8086 microprocessor has time multiplexed 16-bit address/data bus AD_{15}–AD_0 and 4-bit address/status bus A_{19}/S_6–A_{16}/S_3. ALE signal is used to latch the address of 8086. Usually, latch ICs are available with eight separate latches. Therefore, three latch ICs should be used for demultiplexing twenty bit address lines. Figure 9.26 shows the circuit diagram for latching twenty bit address lines using three 74LS373 Latch ICs. In this arrangement, two ICs are fully utilised and one latch is partially used.

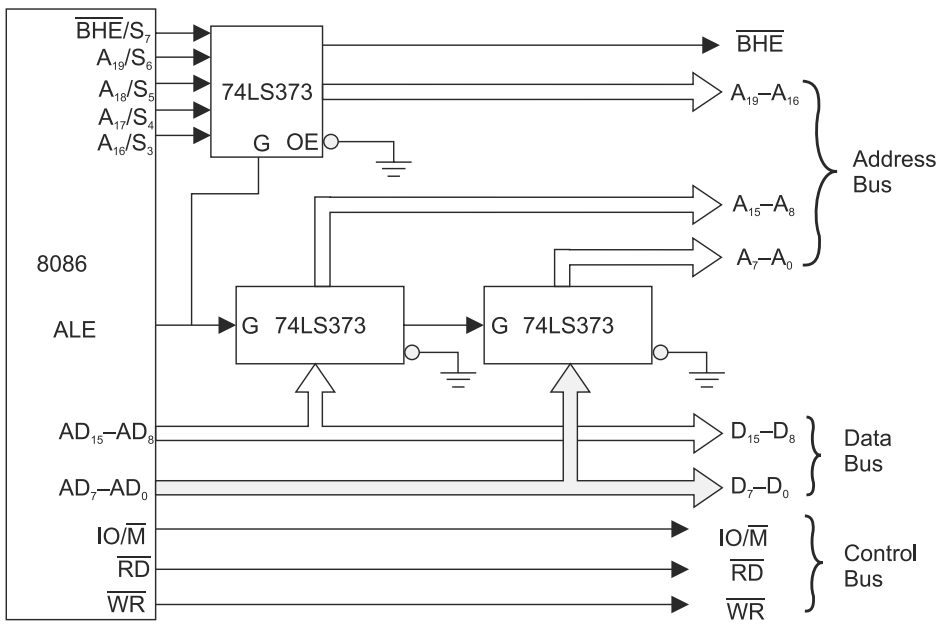


Fig. 9.26 Demultiplexing of 20-bit address bus in 8086 processor

The 8086 microprocessor has a 16-bit time multiplexed data bus which is available as address/data bus, AD_{15}–AD_0. Always the data bus is separated from address bus by using 74245 buffers as depicted in Fig. 9.27. The data bus is bi-directional and data can be transferred from microprocessor to memory and from memory to microprocessor for memory write and read operation respectively. The control signals $\overline{DEN}$ and *DT/*$\overline{R}$ represents the present of data on the data bus and directional flow of data. These signals are used to connect the chip enable CE and directional pins of 74245 buffers. While $\overline{DEN}$ is low, the data is available on the multiplexed bus.

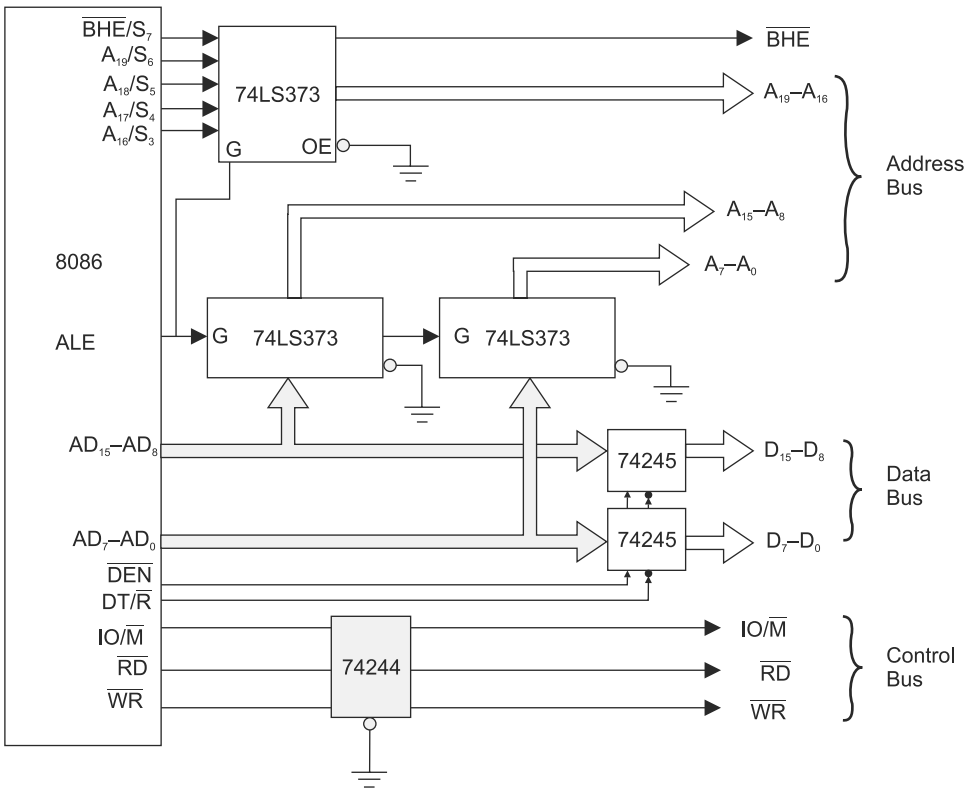


Fig. 9.27 Demultiplexing of fully buffered system bus in 8086 processor

9.10.2 Demultiplexing of System Bus in 8088

In 8086 A_{19}/S_6–A_{16}/S_3, AD_{15}–AD_0 and $\overline{BHE}/S_7$ are multiplexed but in 8088 only A_7–A_0 and A_{19}/S_6 – A_{16}/S_3 are time multiplexed. The demultiplexing of address bus of 8088 microprocessor is shown in Fig. 9.28. Here, 74LS373 latches are used to demultiplex address/data bus. When ALE = 1, the latches pass the inputs to the outputs. After one clock time T_1, ALE becomes to logic 0. A_{19}/S_6–A_{16}/S_3 are connected into top latch and A_7–A_0 are connected into the bottom latch. These address connections can be able to address 1MB memory space. The 8088 systems require only one data buffer due to the 8-bit data bus as depicted in Fig. 9.29.

9.11 SOME IMPORTANT ICs: 8284A, 8286/8287, 8282/8283, AND 8288

The 8086 processor requires a clock signal with very fast rise and fall times which is about 10ns and duty cycle of 33%. For proper operation, 8086 processor RESET signal must be synchronised with clock signal and persist for four T states. Actually, the 8284A clock generator IC meets the requirement of CLOCK and RESET signal.

Buses of 8086 microprocessors require buffering techniques for reliable data transmission. When any receiver receives data, it requires a dc load current from the transmitter. Due to this load, the high level output voltage V_{OH} will be reduced and low level output voltage V_{OL} will be increased. Hence noise immunity of the system will be reduced. In addition, each receiver has an ac load current due to its input capacitance.

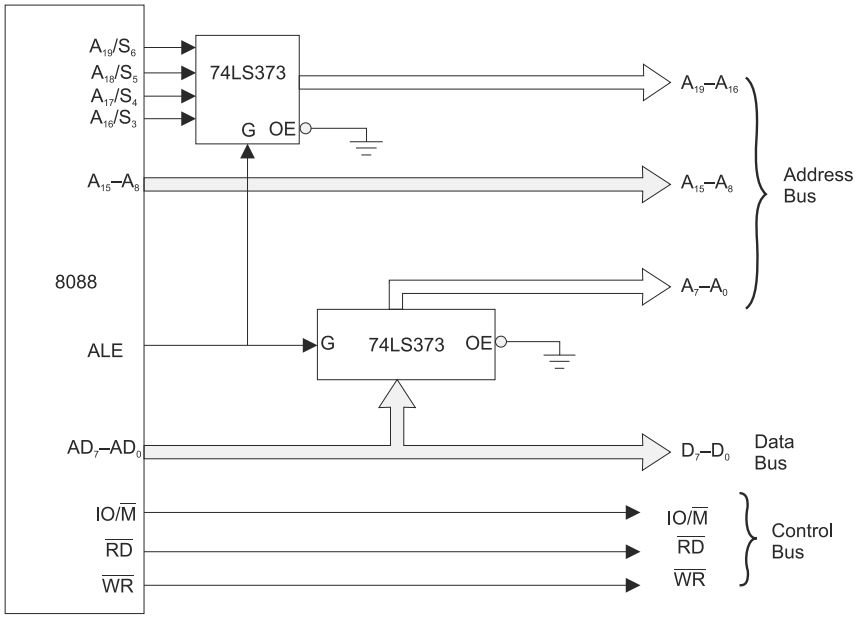


Fig. 9.28 Demultiplexing of address bus in 8088 processor

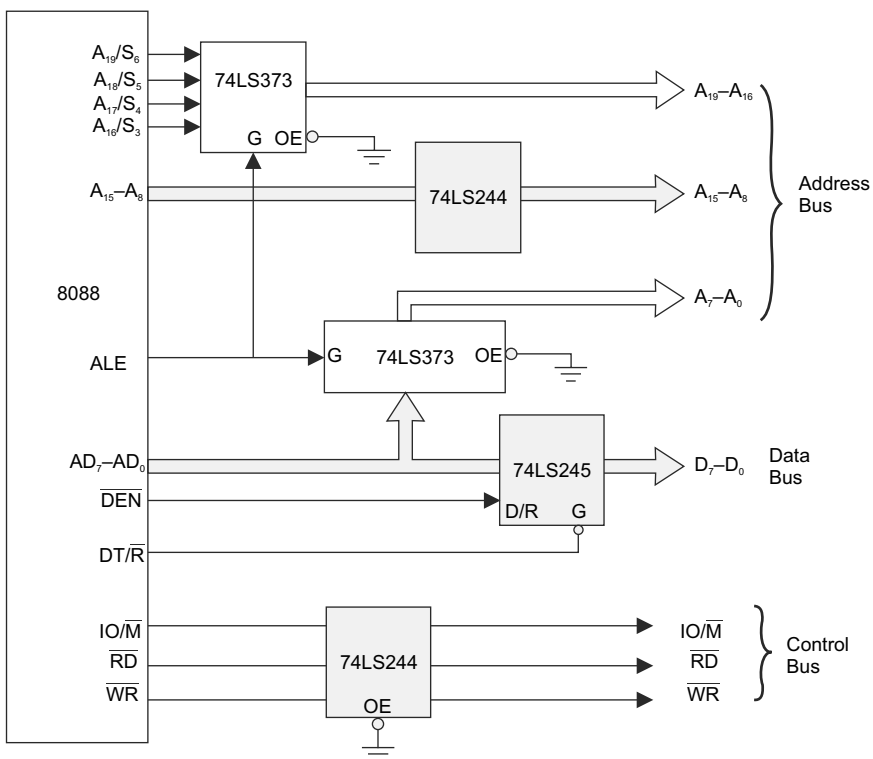


Fig. 9.29 Demultiplexing of fully buffered system bus in 8088 processor

Truly input capacitor must be charged and discharged whenever the transmitter output state changes from logic level 1 to 0 or logic level 0 to 1. As the propagation delay time of the transmitted signal increases, the time availability to the memory and I/O devices will be reduced. Therefore, to minimize the dc as well as ac loading effect, buffers are required for microprocessor buses. Usually, 8286/8287 buffer ICs are commonly used in microprocessor-based systems.

The 8282 and 8283 are 8-bit bipolar latches with three state output buffers. These ICs can be used as latches, buffers or multiplexers. The 8282 provides non-inverted outputs whereas the 8283 inverts the input data at its output. In the maximum-mode operation, 8086 requires system control signals such as $\overline{MRDC}$, $\overline{MWTC}$, $\overline{AMWC}$, $\overline{IORC}$, $\overline{IOWC}$ and $\overline{AIOWC}$. The 8288 bus controller chip generates the all control signals.

To design any microprocessor-based system, 8284A clock generator, 8286/8287 buffers, 8282/8283 I/O ports and 8288 bus controller are used. In minimum and maximum mode operation, first three ICs are extensively used but 8288 bus control is only used in maximum mode operation. In this section the detail operation of 8284A clock generator, 8286/8287 buffers, 8282/8283 I/O ports and 8288 bus controller are discussed in detail.

9.11.1 Clock Generator 8284A

During fetch and execute instructions, the 8086 and 8088 processors require clock pulse which has about 10 ns rise and fall times. The logic-0 level of clock is 0.5 V, to 0.6 V, logic–1 level is about 3.9 V to 5 V and clock duty cycle is about 33%. The 8086 processor has no clock generator inside the chip. So, an external clock generator IC must be connected with the processor. The 8284A is a clock generator IC and it is a supporting component to the 8086/ 8088 processors. The 8284A IC has the following features or additional basic functions such as clock generation, RESET Synchronisation, READY Synchronisation, and a TTL level peripheral clock signal. The operation of 8284A IC has been explained in this section.

Operation of 8284A The internal logic of the 8284A clock generator is depicted in Fig. 9.30. The upper half of the logic diagram represents the clock and reset synchronisation section of the 8284A clock generator. It is depicted in Fig. 9.30 that the crystal oscillator has 2 inputs: X_1 and X_2. When a crystal is connected to X_1 and X_2 terminals, the oscillator generates a square-wave signal and its frequency is same as the crystal frequency.

The output square-wave signal is fed to an AND gate and it is inverted by using an inverting buffer to generate the OSC output signal. The OSC signal can be used as an EFI input to other 8284A clock generators. When $F/\overline{C}$ is a logic-0, the output of the AND gate is fed to divide by 3 synchronous counter. If $F/\overline{C}$ is a logic-1, then EFI is steered through to the counter.

The output of the divide by 3 synchronous counters generates the ready signal for synchronisation. The CLK signal is buffered before output from the clock generator. Another divide by 2 synchronous counters generate the PCLK signal to the 8086/8088 microprocessor. When the output of the first $\div 3$ synchronous counters fed to the second $\div 2$ synchronous counters, the two cascaded counters generate $\div 6$ outputs at PCLK. Figure 9.31 shows the connection between an 8284A and the 8086/8088 processor. Usually, $F/\overline{C}$ and CSYNC are connected with ground to select the crystal oscillator. Then a 15-MHz crystal generates the normal 5-MHz CLK clock signal and a 2.5-MHz peripheral clock signal PCLK.

The reset section of the 8284A consists of an Schmitt trigger buffered and a D type flip-flop. The D flip-flop ensures that the timing requirements of the 8086/8088 RESET input are met. This circuit applies the RESET output signal of clock generator is fed to the microprocessor as shown in Fig. 9.32 and it is active on the negative edge of the clocks. Hence the reset section meets the timing requirements of the 8086/8088 microprocessor.

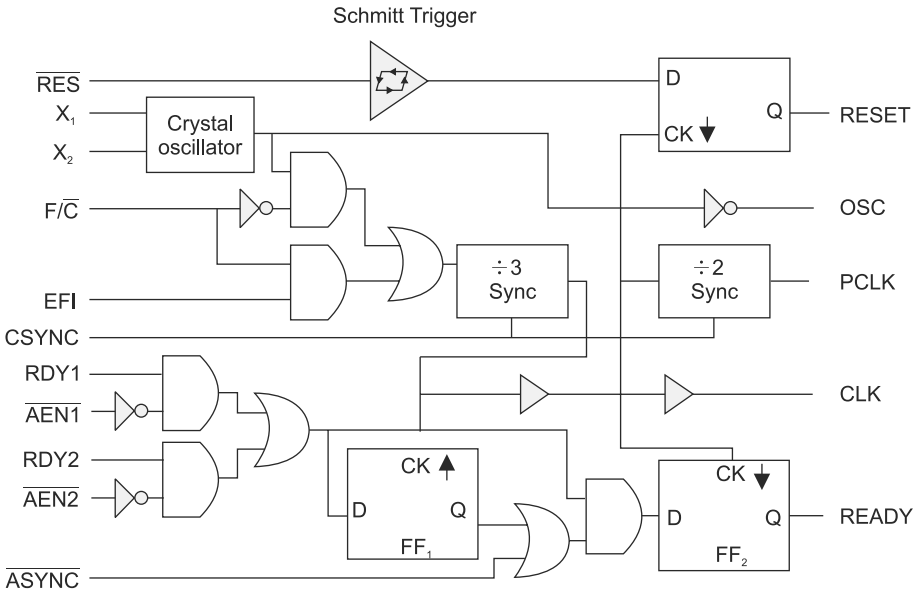


Fig. 9.30 Block diagram of 8284 clock generator

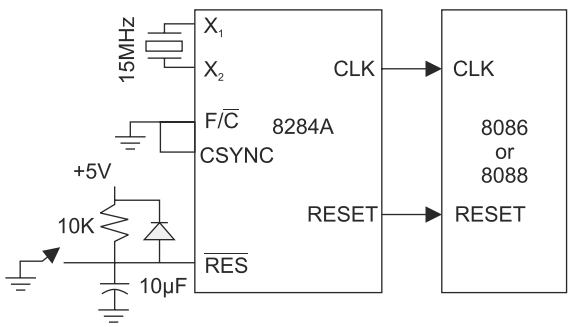


Fig. 9.31 CLK generator 8284A and 8086/88 microprocessor

Pin Functions of 8284A The 8284A is an 18-pin IC which is specifically designed for 8086/8088 microprocessor. The pin diagram of the chip is shown in Fig. 9.32. In this section, the functions of pins are explained.

X_1 and X_2 Crystal Inputs These pins are connected to an external crystal which is used as clock frequency source of the clock generator. The external crystal clock frequency will be about three times the required frequency. If the required frequency is 5–MHz, the crystal frequency will be 15–MHz.

CLK CLK is an output pin that provides the clock (CLK) signal which is used as input signal to the 8086/8088 microprocessor system. The CLK pin has an output signal with 33% duty cycle as required by the 8086/8088.

EFI (External Frequency Input) This is an alternate clock input when $F/\overline{C}$ pin is pulled high. The externally generated clock signal is supplied to EFI whenever the $F/\overline{C}$ pin is high.

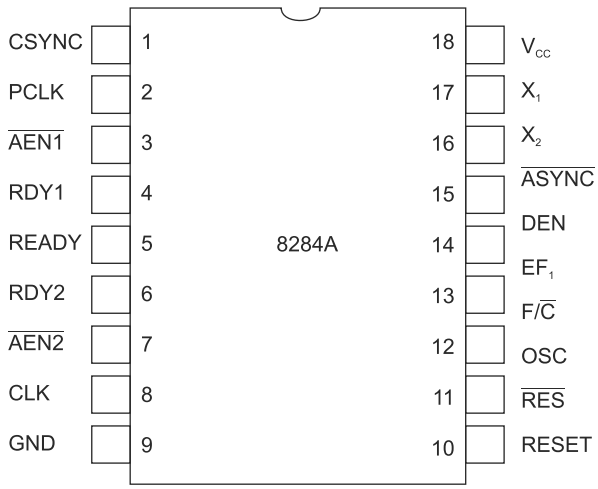


Fig. 9.32 Pin Diagram of the 8284A

PCLK (Peripheral Clock) This is a clock output signal that is one sixth of the crystal. PCLK is half of the clock frequency and has a 50% duty cycle. The PCLK output signal can be used as a clock signal to the peripheral equipments in 8086/8088 system.

OSC (Oscillator Output) This is an oscillator output signal which is running at crystal or EF_1 frequency. This signal can be used to provide clock signal at EF_1 to the other 8284 clock generators in some multiple processor system.

F/$\overline{C}$ (Frequency/Crystal) The voltage on this pin determines the clocking source for the 8284A. If this input pin is high, an external clock at EF_1 is selected. While it is low, the internal crystal oscillator provides the clock frequency signal.

CSYNC (Clock Synchronisation) This pin is used for synchronisation of clock signals in multiprocessor system where all processors receive clock at EF_1. If the internal crystal oscillator is used, this pin must be grounded. When CSYNC is high, the 8284A clock generator stops working.

$\overline{RES}$ (Reset Input) To reset 8086 processor, 8284A clock generator should send reset signal. Generally this pin is connected to a RC network for generating reset signal at power-on.

RESET (Reset Output) This signal is connected to the 8086/8088 RESETs input pin. The RESET signal must be synchronised with clock.

RDYl , RDY2 The slow memory or I/O devices can request for extension of bus cycles using RDYl or RDY2 pins. These two wait state ready inputs are provided to support multibus 8086/8088-based system.

READY The READY output pin connects to the 8086 READY input which enables the bus cycle extension through wait state clock period insertion between T_3 and T_4. The 8086 READY signal must be synchronised with the RDY1 and RDY2 inputs.

$\overline{ASYNC}$ (Ready Synchronisation Select) This input pin is used to select either one or two stages of synchronization for the RDY1 and RDY2 inputs. If it is low, one level is selected. When it is high, two levels of synchronization are selected.

$\overline{AEN}_1$, $\overline{AEN}_2$ Two ready inputs RDY1, RDY2 have been provided in the 8284A to support multibus system. 8086 CPU may be connected to two separate system buses, on which data transfer takes place. The memory or I/O devices of any system buses may like to insert wait states. Hence each system bus should have its own ready line. $\overline{AEN}_1$ and $\overline{AEN}_2$ are provided to arbitrate bus priorities whenever RDY1 and RDY2 are active. The 8284A responds to RDY1 when $\overline{AEN}_1$ is low. In the same way, clock generator responds to RDY2 if $\overline{AEN}_2$ is low.

V_{cc} (Power Supply Input) This pin is connected to +5 V $\pm$10%.

GND (Ground) This pin must be grounded.

9.11.2 Bidirectional Bus Transceiver 8286/8287

The Intel 8286 and 8287 are bidirectional system bus buffers-cum-drivers. Figure 9.33 shows the pin diagram of 8286 and 8287 transceiver.

A_7–A_0 Bidirectional Tristate

These lines are connected to microprocessor data/address bus.

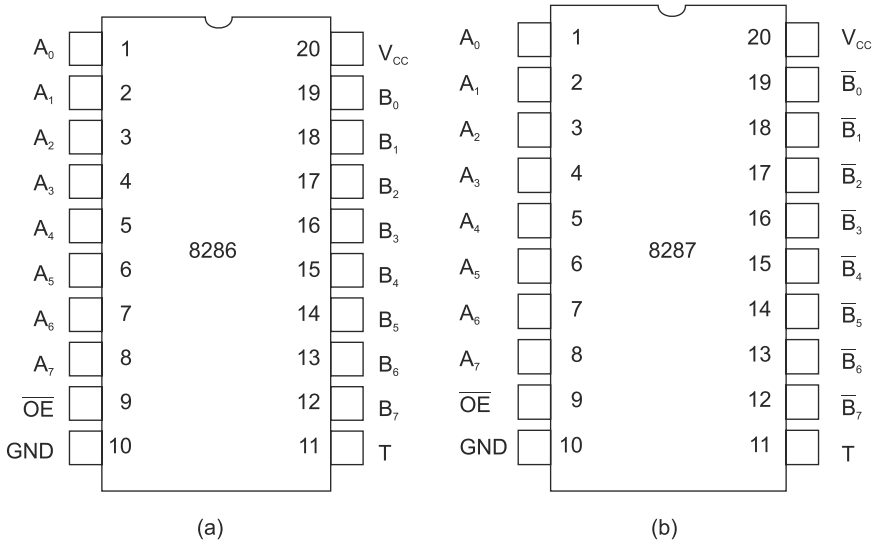


Fig. 9.33 Pin diagram of bidirectional bus transceiver (a) 8286 and (b) 8287

B_7–B_0 Bidirectional Tristate These lines are connected to system bus

$\overline{OE}$ Output enable

T Direction Select

V_{CC} Input Power Supply +5 V.

GND This pin is grounded

9.11.3 8-Bit Input-Output Port 8282/8283

The Intel 8282 and 8283 are unidirectional latch buffers. The difference between the 8282 and 8283 is that the IC 8282 does not change the data and the IC 8283 inverts the input data. Figure 9.34 shows the pin diagram of 8282 and 8283 input–output port. The pin functions are as follows.

DI_7–DI_0 Data input

DO_7–DO_0 Data output

$\overline{OE}$ Output enable

STB Input data strobe If STB is high, the data on output pins tracks the data on input pins. During transition of STB from high to low, the data is latched. The data remains unchanged when STB is low. The data is latched internally till $\overline{OE}$ is low. If $\overline{OE}$ is low, the data is put on output lines. The 8282 outputs the data without inverting and the 8283 inverts the data.

V_{cc} Input Power Supply +5 V

GND This pin is grounded

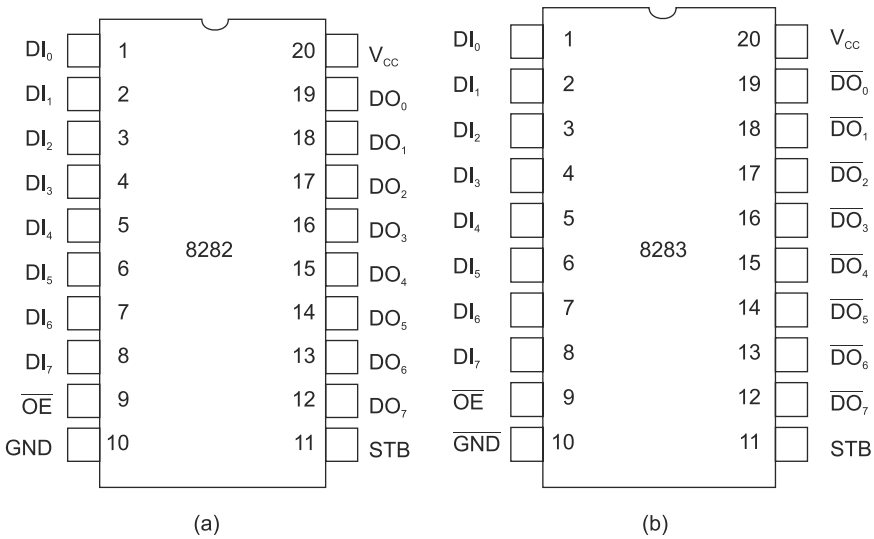


Fig. 9.34 Pin diagram of 8-bit I/O port (a) 8282 (b) 8283

9.11.4 8288 Bus Controller

Figure 9.35 shows the 8288 bus controller which is used in the maximum mode operation of 8086 CPU. This IC receives four inputs such as $\overline{S}_2$, $\overline{S}_1$, $\overline{S}_0$ status signals and CLK from 8086. There are two sets of output command signals.

The first set of output command signals are the MULTIBUS command signals. These are the conventional $\overline{MEMR}$, $\overline{MEMW}$, $\overline{IOR}$ and $\overline{IOW}$ control signals. These signals are renamed as $\overline{MRDC}$, $\overline{MWTC}$, $\overline{IORC}$ and $\overline{IOWC}$ respectively. These outputs are enabled one clock cycle earlier than the normal write commands. Some memory and I/O devices require this wider write pulse width.

The second set of output signals of the 8288 are the bus control signals *DT/$\overline{R}$*, DEN, ALE and *MCE/$\overline{PDEN}$*. *MCE/$\overline{PDEN}$* is an output signal which has two functions depending on the 8288's mode of operation such as I/O bus control or system bus control.

The three 8288 control inputs CEN, IOB and $\overline{AEN}$ determine the operating mode as given in Table 9.11. When CEN and IOB are high or logic level 1, the 8288 operates in the I/O bus mode and the *MCE/$\overline{PDEN}$* output acts as a peripheral data enable. The function of *MCE/$\overline{PDEN}$* is identical to DEN but it is active only

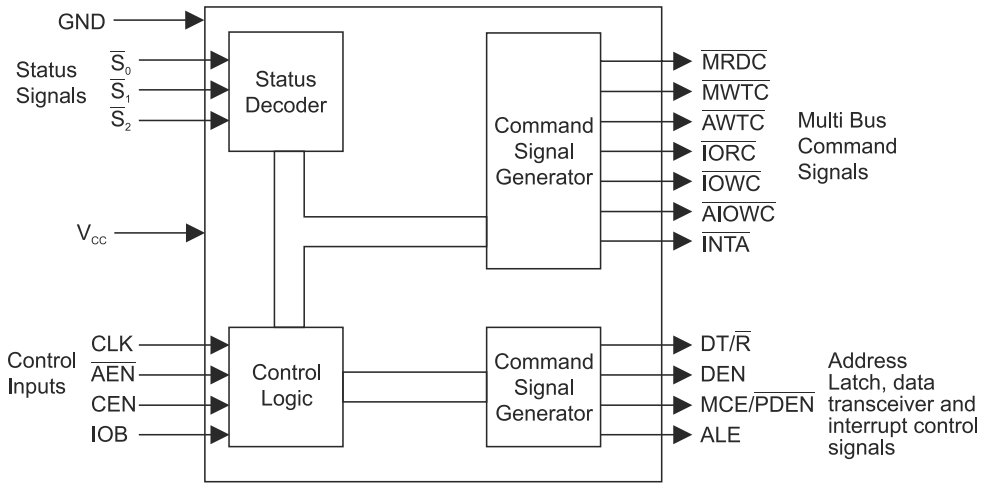


Fig. 9.35 Functional diagram of 8288 bus controller

during I/O instructions. This allows the 8288 to control two sets of buses such as the normal system buses and a special I/O bus dedicated to the processor.

During the system bus mode, the control signals are active only while the address enabled signal $\overline{AEN}$ and IOB inputs are low. In this mode of operation, several 8288 ICs and 8086 processors can be interfaced to the same set of bus lines. The bus mediator selects the active processor after enabling only one 8288 bus controller through $\overline{AEN}$ signal. The $MCE/\overline{PDEN}$ signal becomes MCE (master cascade enable). In this mode, the $MCE/\overline{PDEN}$ signal is used to read the address from a master priority interrupt controller, PIC.

Table 9.11 Operating modes of 8288 bus controller

CEN	IOB	$\overline{AEN}$	Operations
0	x	x	All command outputs and the DEN and PDEN outputs are disabled or open-circuited
1	0	0	System bus mode and all control signals are active. The bus is free for use and $MCE/\overline{PDEN} = MCE$
1	0	1	System bus mode but all control signals are disabled. The bus is busy, and is controlled by another bus master
1	1	x	I/O bus mode; all control lines are enabled and $MC/\overline{PDEN} = \overline{PDEN}$

The pin diagram of the 8288 bus controller is shown in Fig. 9.36. The functions of the pins are described in this section.

$\overline{S_2}$, $\overline{S_1}$, $\overline{S_0}$ (Status Input Signals) These are bus cycle status signals. These signals are decoded and control signals are generated.

CLK This is an input signal. It is connected to CLK output of the clock generator 8284.

$\overline{AEN}$, CEN, IOB These are bus priority and mode control signals. $\overline{AEN}$ (bus priority control/enable), CEN (Command enable) , and IOB (mode control) signals are used to generate various control signals.

$\overline{MRDC}$ (Memory Read Control Signals) This command signal is used to load the content of memory location on the data bus.

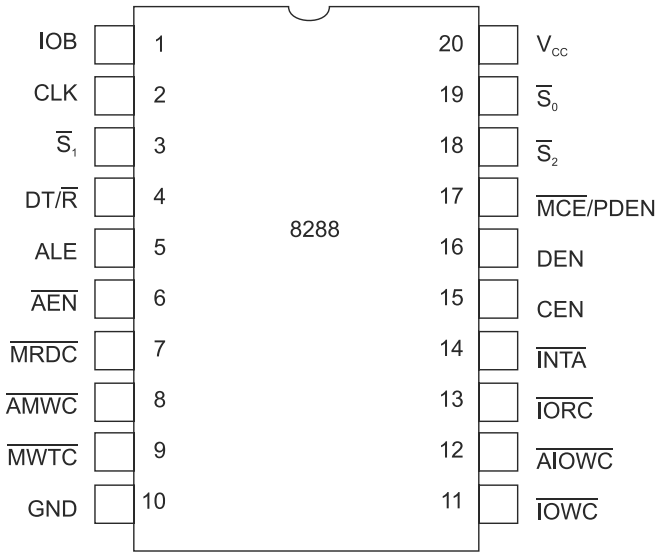


Fig. 9.36 Pin diagram of 8288 bus controller

$\overline{MWTC}$ (Memory Write Control Signals) This command signal is used to store the available data on the data bus to the specified memory location.

$\overline{IORC}$ (I/O Read Control Signals) The I/O device can be able to put the available data of the addressed port on the data bus.

$\overline{IOWC}$ (I/O Write Control Signals) The I/O device is able to accept the available data on the data bus and send to the addressed port.

$\overline{AMWC}$ (Advance Memory Write Control Signal) This signal is activated one clock period earlier than $\overline{MWTC}$.

$\overline{AIOWC}$ This signal is activated one clock period earlier than $\overline{IOWC}$.

MCE/$\overline{PDEN}$ This is cascade/peripheral data enable. This signal is used in Priority Interrupt Controller 8259A.

$\overline{INTA}$ (Interrupt Acknowledge) This is used as output signal during two interrupt acknowledge bus cycles and is used as memory read control signal.

ALE Address Latch Enable signal

DT/$\overline{R}$ Data direction control signal

DEN Data buffer control signal

V_{cc} Power Supply Input +5 V.

GND This pin is connected system ground.

9.12 INTERRUPTS OF 8086/8088 MICROPROCESSOR

An interrupt is an external signal which sends information to the CPU so that an external device gets service from CPU. This signal provides a mechanism for changing one program environment to another. Due to an interrupt, the microprocessor stops execution of its current instruction and calls a procedure to provide service to interrupt. At the end of the interrupt service procedure, an IRET instruction is executed to return back to the main program. The 8086/8088 processor has the following interrupts:

- Software interrupts
- Nonmaskable interrupts
- Internal interrupts
- External hardware interrupts
- Reset

9.12.1 Software Interrupts

When the source of interrupt is the execution of interrupt instructions, this interrupt is known as software interrupt. In 8086/8088, there are about 256 interrupts such as INT 00H, INT 01H, INT 02H to INT FFH. Whenever the INT interrupt instruction is executed, the microprocessor automatically saves the content of the flag register, Instruction Pointer (IP) and code segment register on the stack and jumps to a specified memory location. In 8086, the memory location is always four times the value of the interrupt number. When INT *n* is executed, the interrupt service routine is located at $n \times 4$ H memory address. For example, INT 02H goes to 00008H. Software interrupts are always generated by INT instructions and used for divide-by-zero error, single-step, NMI, break-point, and overflow interrupts.

For each interrupt, there is a program associated with a specified interrupt. This program is known as Interrupt Service Routine (ISR). It is also called *interrupt handler*. When the interrupts occurs, the processor runs the interrupt service routine according to the interrupt vector table as given in Table 9.12.

Table 9.12 Interrupt vector table

INT Number	Physical address assuming CS = 0000H
INT 00H	00000H
INT 01H	00004H
—	—
—	—
INT FEH	003F8H
INT FFH	003FCH

The interrupt vector table has 256 entries, each containing four bytes. Each interrupt vector consists of a 16-bit offset and the 16-bit segment address. The initial 32 interrupt vectors are spared for various microprocessor operations and the remaining 224 interrupt vectors are user defined. The lower vector number has the higher priority, when more than two interrupts occur simultaneously. Figure 9.37 shows the interrupt pointers. There are 256 address pointers. The starting addresses of their service routines are available in the program memory as depicted in Fig. 9.37.

Interrupt Type 0—INT 00H (Divide by Zero Error) The 8086 generates a type 0 interrupt, if the result of DIV or IDIV operation is too large to fit in the destination register. For this interrupt, 8086 pushes the content of flag register on the stack, resets IF and TF and also pushes the content of CS and IP onto the

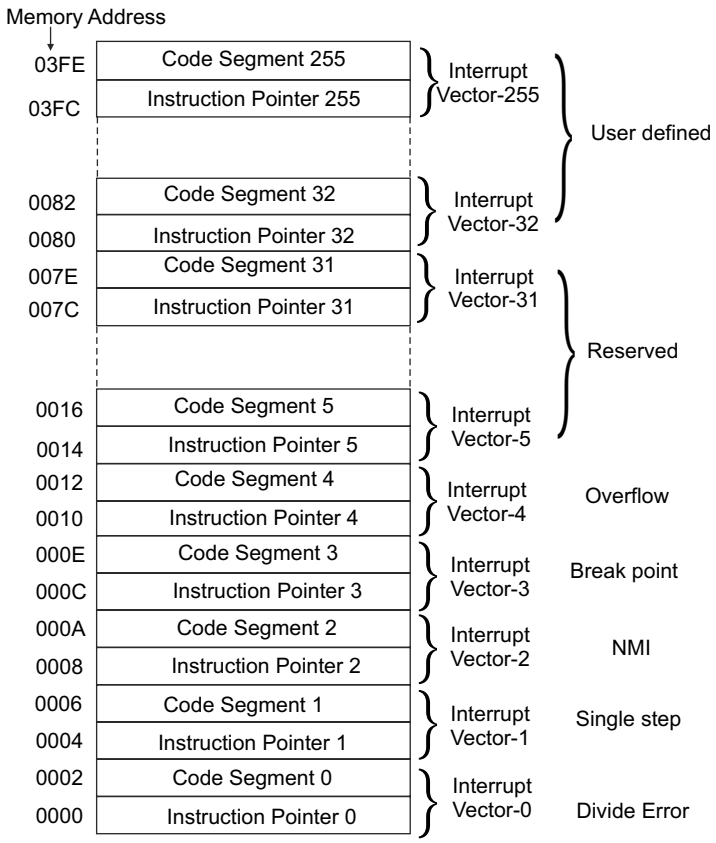


Fig. 9.37 Memory address of interrupt vectors

stack. Then 8086 gets the starting address of the interrupt service procedure from the interrupt pointer table. Therefore, load the new value of CS from addresses 00002H and 00003H; also load the new value of IP from addresses 00000H and 00001H.

Interrupt Type 1—INT 01H (Single Step) During execution of a sequence of instructions, there is frequently a need to examine the contents of the CPU's registers and system memory. This is done by executing one instruction at a time and then inspecting the registers and memory. If they are correct, the user can give the command to go on and execute the next instruction. This is called the *single stepping*. When the 8086 Trap Flag (TF) is set, the 8086 perform a type 1 interrupt after execution of each instruction.

When the CPU gets a type 1 interrupt, initially it pushes the flag register onto the stack, changing the trap bit and pops the flag register back from the stack. Then it loads the CS value from starting address 00006H and the IP value from starting address 00004H for the type 1 interrupt service routine. The 8086 has no instruction to set or reset the trap flag. A sequence of instructions is used to set the trap flag as given below:

```
PUSHF                ; Push flags onto stack
MOV BP, SP           ; Copy SP to BP
OR [BP+0], 001000H   ; Set TF bit
POP F                ; Pop flag register and TF is set
```

To reset the trap flag, the OR instruction will be replaced by AND [BP + 0], 0FEFFH. After reset the trap flag, when the 8086 processor sends a type 1 interrupt, single-step mode will be disabled.

Interrupt Type 2—INT 02H (Nonmaskable Interrupt) The 8086 performs a type 2 interrupt when the NMI pin receives a low to high transition signal. Then the CPU, 8086 pushes the content of flag register on the stack, resets IF and TF and also pushes the content of CS and IP onto the stack. After that, the CPU jumps to 00008H to fetch the CS: IP of the ISR associated with NMI. As the type 2 interrupt response cannot be disabled (masked) by any instruction, this interrupt is called a nonmaskable interrupt. Usually, the type 2 interrupt is used to switch off a circuit for protection.

Interrupt Type 3—INT 03H (Break Point) The type 3 interrupt is generated by the execution of INT 03H instruction. This interrupt is used to implement a break-point function in a system. When we insert a break point in the main program, the system executes all instructions up to the break point and then jumps to the break-point subroutine. When 8086 executes the INT 03H instruction, it pushes the content of the flag register onto the stack, resets TF and IF and pushes the CS and IP values onto the stack. Then 8086 gets the new IP value from the starting address 0000CH and the CS value from the starting address 0000EH.

Interrupt Type 4—INT 04H (Overflow Interrupt) If the result of addition of two signed numbers is too large to represent in the destination register, the overflow (OF) flag will be set. For example, if we add 0110 1100 (108_{10}) and 0101 0001(81_{10}), the result is 1011 1101. The above result will be correct only for unsigned binary numbers. For signed number addition, "1" in the MSB of the result represents that the result is negative and it is in 2's complement form. Hence the result, 1011 1101 represents -67_{10}, but the correct result is (189_{10}). If the overflow flag is set, the 8086 provides type 4 interrupt after executing the INTO instruction.

During execution of type 4 instruction, the 8086 pushes the content of flag register on the stack, resets TF and IF and pushes the values of CS and IP on the stack. Then 8086 gets a new IP value from the starting address 00010H and a new CS value from the starting address 00012H. After that instructions in the ISR perform the desired operation.

9.12.2 INTR Interrupts, Type 0 to 255

The INTR input pin of 8086 allows external signal to interrupt the execution of a program. INTR can be masked or disabled so that it cannot cause any interrupt. When the Interrupt Flag (IF) is cleared, INTR input pin becomes disabled. The IF can be cleared by Clear Interrupt Instruction (CLI).

When the IF is set, the INTR input will be enabled. The IF can be set by Set Interrupt Instruction (STI). After reset of 8086 microprocessor, the interrupt flag is automatically cleared. The INTR interrupt is sent to the 8086 from the 8259A interrupt controller as shown in Fig. 9.38.

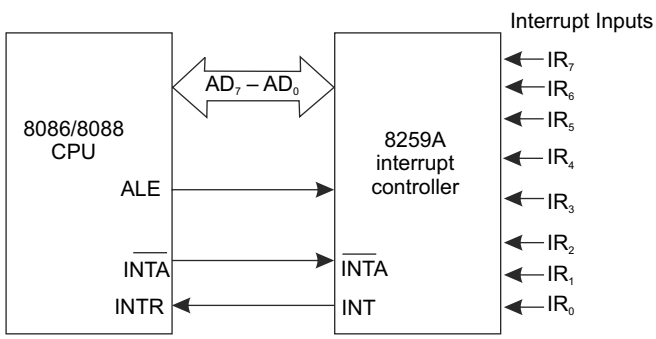


Fig. 9.38 8259A connected with 8086/8088 microprocessor

When the 8259A receives an interrupt signal on any one of the IR inputs, it provides an interrupt request signal to the INTR input of 8086. When the INTR input is enabled with an STI instruction, the 8086 processor sends an interrupt acknowledge signal. Figure 9.39 shows the interrupt acknowledge bus cycle of 8086.

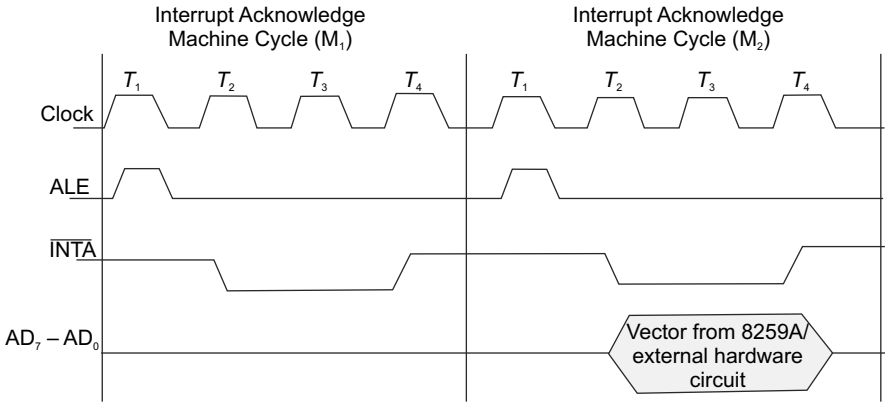


Fig. 9.39 8086/8088 interrupt acknowledge bus cycle

9.12.3 External Hardware Interrupt Interface

Figure 9.40 shows the minimum-mode hardware interrupt interface of 8086 microprocessor. The external hardware interrupt circuit can identify which of the pending interrupts has highest priority. Subsequently, hardware interrupt circuit passes its type number to the microprocessor. Then 8086 CPU samples INTR input during the last clock period of each interrupt execution cycle.

INTR is a level-triggered input. The logic level '1' must be maintained until it is sampled, but it must be removed before it is sampled next time. Otherwise the same interrupt service routine (ISR) will be repeated.

$\overline{INTA}$ becomes logic level '0' in the first interrupt bus cycle to acknowledge the interrupt as the 8086 CPU has decided to respond to the interrupt.

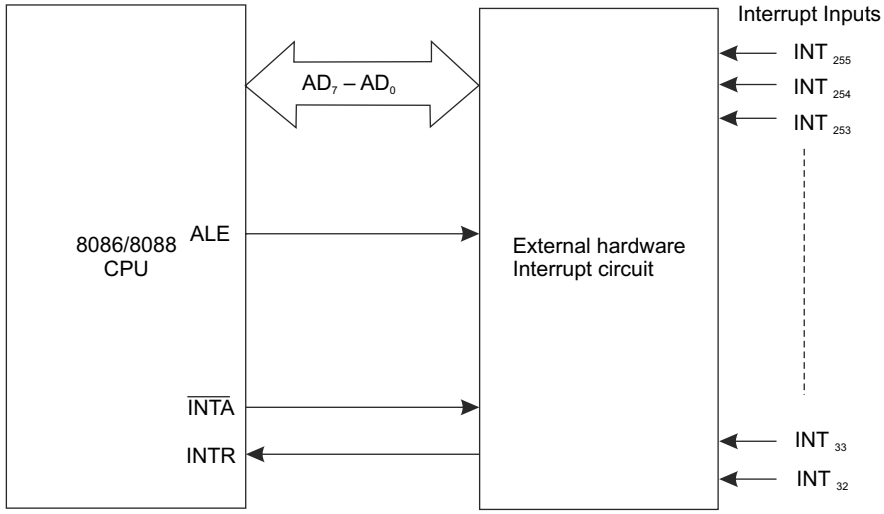


Fig. 9.40 External hardware interrupt interface with 80886/8088 CPU

It goes to '0' again in the second bus cycle to request for the interrupt vector type from external device. Then interrupt-type number is read by the microprocessor and the new value of CS and IP are also read from the memory. Figure 9.39 shows the interrupt acknowledge bus cycle of 8086.

9.12.4 Priority of 8086/8088 Interrupts

In the 8086/8088 microprocessor, all interrupts must be serviced as priority order. The highest-priority interrupt will be serviced first and then the next highest-priority interrupt will be serviced. Therefore, lower-priority interrupt service will be provided after a higher-priority one. The priority of interrupts will be controlled by ISR. The priority order of 8086/8088 interrupts are

- Reset
- Internal interrupts
- Software interrupts
- Nonmaskable interrupts
- Hardware interrupts

9.12.5 Interrupt Instructions of 8086/8088

The interrupt instructions of 8086/8088 microprocessors are CLI, STI, INT *n* INTO, HLT and WAIT. The functional operation of interrupt instructions are given in Table 9.13.

Table 9.13 Interrupt instructions

Mnemonics	Function	Operation
CLI	Clear interrupt instruction, IF affected	IF←0
STI	Set Interrupt flag, IF affected	IF←1
INT *n*	Type *n* software interrupts. This interrupt initiates a vectored call of an interrupt service subroutine	(SP – 2) ←Flags, TF,IF←0, (SP – 4)← CS, CS ←(2 + 4 × *n*), IP← (SP – 6), IP← 4 × *n*
IRET	Interrupt return, All flags affected	IP←(SP), CS←(SP + 2), Flags←(SP + 4), SP←(SP + 6)
INTO	Interrupt overflow, TF and IF affected	Same as INT 4
HLT	Halt	Wait for an external interrupt
WAIT	Wait	Wait for $\overline{TEST}$ input to become active

9.12.6 Interrupt Cycle of 8086/8088

The sequence of operations of any interrupt (interrupt cycle) is depicted in Fig. 9.41 and the step-by-step operation for interrupt is given below:

Step 1 The interrupt sequence starts when an external device requests service by sending an interrupt input.

Step 2 The external hardware circuit or interrupt controller evaluates the priority of the interrupt.

Step 3 The 8086 checks for the INTR at the last *T* state of the instruction.

Step 4 Check IF before sending interrupt acknowledge signal INTA.

Step 5 8086 initiates the INTA bus cycle. During T_1 of the first bus cycle, ALE is high and address/data bus AD_7–AD_0 is at high impedance (Z) state and stays high for the bus cycle. During the second interrupt acknowledge bus cycle, external circuit gates one of the interrupts.

Step 6 The contents of the flag register are pushed on the stack.

Step 7 The Interrupt Flag (IF) and Trap Flag (TF) are cleared. This disables the INTR pin and the trap or single-step feature.

Step 8 The contents of the Code Segment (CS) is pushed on the stack.

Step 9 The contents of the Instruction Pointer (IP) is pushed on the stack.

Step 10 The interrupt vector type number is multiplied by 4 and generates a memory address. The contents of this address are fetched and placed into IP. Subsequently, the contents of the memory address (interrupt vector type number × 4 + 2) are fetched and placed into CS. After that the next instruction executes at the interrupt service procedure addressed by the interrupt vector.

Step 11 To return from the interrupt service routine, the IRET instruction is executed.

Step 12 Flags return to their state prior to the interrupt. Operation restarts at the prior IP address after CS and IP are popped.

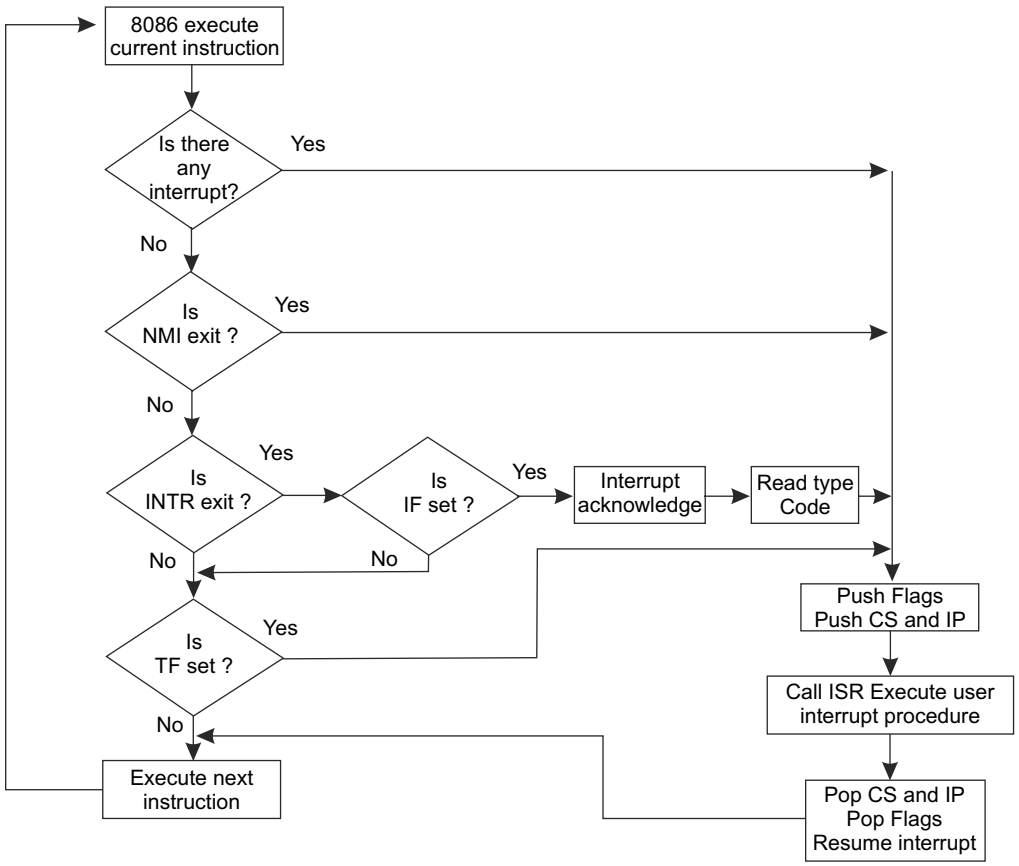


Fig. 9.41 Flow chart of interrupt operation

9.13 EPROM INTERFACING WITH 8086

The most commonly used EPROM ICs are 27C256, 2716, 2732A and 2764. IC 27C256A is an erasable programmable read only memory and it is represented by 32K × 8 EPROM. The 32K is referred as the number of memory locations in the EPROM. As 1K = 1024, 32 × 1024 or 32,768 memory locations are available in the device. The 8 represents the number of bits in each memory location.

In IC27C256, 27 is the standard number for EPROMs and 256 is the number of K bits stored on the EPROM. Actually, the entire series of EPROMs is represented by 27XXX. The IC27C256 EPROM has 32K bytes of memory with 8-bit wide data bus AD_7–AD_0 and 15 address lines A_{14} to A_0. Figure 9.42 shows the interfacing of EPROM IC27C256 with the 8086/8088 microprocessor.

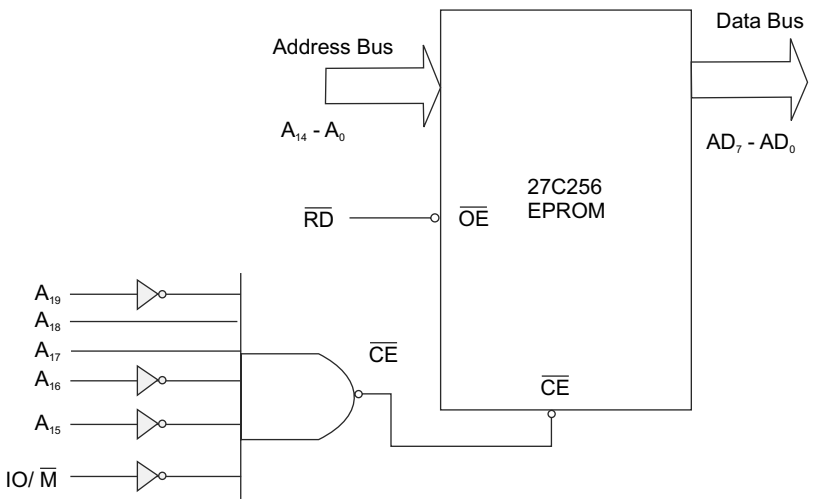


Fig. 9.42 EPROM 27C256 interfacing with 8086/8088 microprocessor

The data bus of the microprocessor AD_7–AD_0 is connected with the 8-bit data outputs of the EPROM. The address bus A_{14}–A_0 of the microprocessor is connected with the address lines A_{14}–A_0 of EPROM. The remaining address lines A_{19}–A_{15} for 8086 processor are used to select the memory devices. The memory read signal $\overline{RD}$ from the microprocessor is directly connected to the EPROM. The operation of EPROM is controlled by two pins such as chip enable $\overline{CE}$ and output enable $\overline{OE}$. The chip enable signal is generated from IO/$\overline{M}$ signal and address lines A_{19}–A_{15}. Therefore, a memory address decoder circuit is used to generate a $\overline{CE}$ signal. The output enable $\overline{OE}$ is used to enable the output buffers in the memory. This device has 8000H memory locations. When the starting address of EPROM is 60000H, the memory end address is 67FFFH. The memory map of EPROM 27C256 is given below:

	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
The starting address is 60000H =	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
The end address is 67FFFH =	0	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

To generate the chip enable signal, $A_{15} = 0$, $A_{16} = 0$, $A_{17} = 1$, $A_{18} = 1$ and $A_{19} = 0$ as depicted in Fig. 9.42.

The IC 2716 EPROM (2K × 8) has only 2 KB of memory and 11 address lines. A decoder can be used to decode the additional 9 address lines and generate a chip enable signal so that the EPROM can be placed in any 2KB section of the 1MB address space. If we assume the starting address of 2716 EPROM is FF800H,

the end address will be FFFFFFH. A NAND gate and an OR gate are used to generate chip enable signal using A_{19}–A_{11}, IO/$\overline{M}$ and $\overline{RD}$ from 8086/8088 microprocessor as depicted in Fig. 9.43. The memory map of IC 2716 is given below:

	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
The starting address is FF800H =	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
The end address is FFFFFH =	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

To generate the chip enable signal, $A_{11} = 1$, $A_{12} = 1$, $A_{13} = 1$, $A_{14} = 1$, $A_{15} = 1$, $A_{16} = 1$, $A_{17} = 1$, $A_{18} = 1$ and $A_{19} = 1$ as depicted in Fig. 9.43.

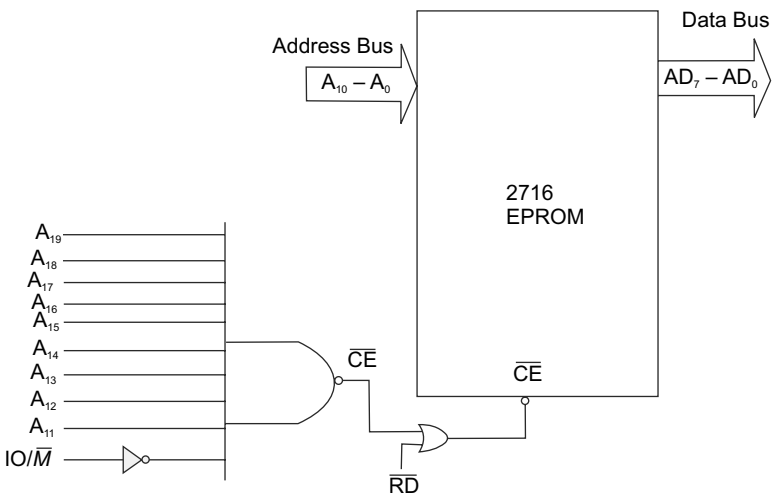


Fig. 9.43 EPROM 2716 interfacing with 8086/8088 microprocessor

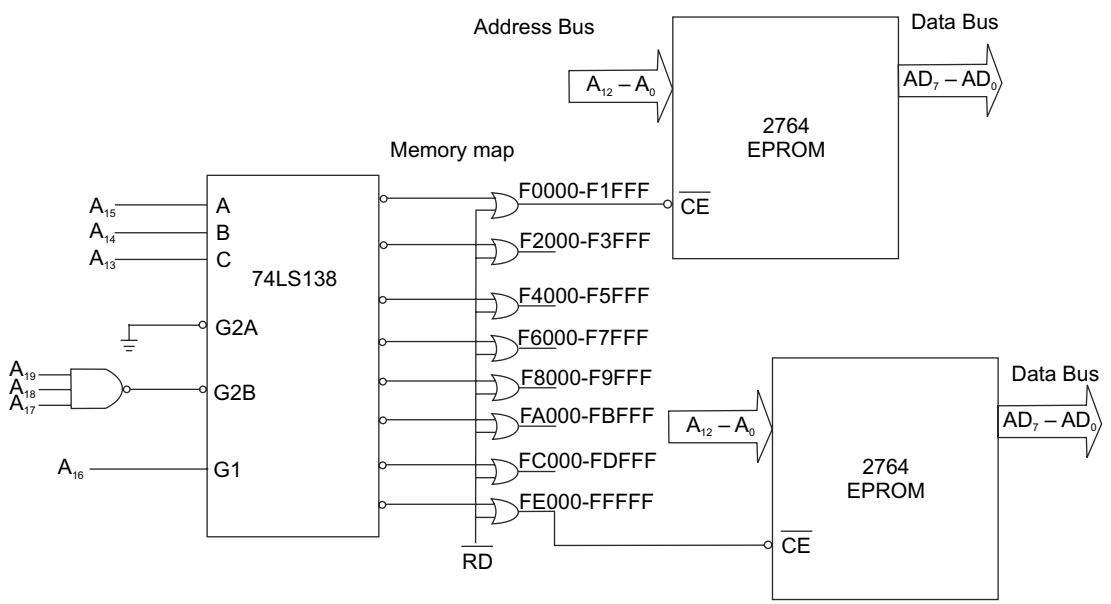


Fig. 9.44 EPROM 2764 interfacing with 8086/8088 microprocessor

In place of a NAND gate decoder, the 3-line to 8-line decoder IC74LS138 is used to select the memory devices and the 8086/8088 microprocessor can communicate with many EPROM ICs as shown in Fig. 8.22. There are three enable pins G2A, G2B and G1 which are active low, active low and active high respectively for proper operation of the decoder. Address lines A_{12} to A_0 are directly connected to IC 2764. A_{15} to A_{13} are used as decoder input and each output of the decoder can select a 2764 EPROM (8K × 8). A_{19}–A_{16} enable the decoder. The memory of each selected EPROM is depicted in Fig. 9.44.

Review Questions

9.1 What are the general-purpose registers of the 8086 and 8088 microprocessor?

9.2 Write difference between (a) 8085 and 8086 (b) 8086 and 8088.

9.3 What is the purpose of the queue? How many bytes can be stored in the queue of 8086 and 8088?

9.4 Draw the schematic block diagram of 8086 and explain the function of each block.

9.5 Define the logical address and physical address. What are the differences between the logical and physical memory of the 8086?

9.6 Determine the physical when CS = 6000H and offset address = 2300H.

9.7 What is pipelined architecture? How is it implemented in 8086?

9.8 Explain the concept of segmented memory? What are its advantages?

9.9 Draw the pin diagram of the 8086 microprocessor. Explain the function of the following pins of 8086:

(i) ALE (ii) NMI (iii) INTR (iv) HOLD (v) HLDA

(vi) BHE (vii) LOCK (viii) $M/\overline{IO}$ (ix) $\overline{DEN}$ (x) $DT/\overline{R}$

9.10 Draw the minimum-mode system configuration of 8086 with memory and I/O interface and give a list about the functions performed by each chip.

9.11 Draw bus cycle timing diagrams for memory read and write operation in the maximum mode and explain briefly.

9.12 What is demultiplexing of buses in 8086? Explain demultiplexing of address bus in 8086 and 8088.

9.13 How do you select the minimum and maximum modes of operation in the 8086/8088?

9.14 What are the functions of Index registers, pointer registers and instruction pointer?

9.15 What is the content of DS and IP to locate the physical address location 35678H? Assume the value of offset address.

9.16 What are the advantages of having segmentation? How does the 8086 microprocessor support segmentation?

9.17 What are the main functions performed by BIU and EV unit of 8086 microprocessor?

9.18 What do you mean by 16-bit microprocessor?

9.19 (a) What is the size of data bus and address bus of 8086μp. What is the size addressable memory present in 8086μp?

(b) Explain the operations of BIU and EU present in 8086 μp?

(c) What are the different types of segment registers present in 8086μp?

(d) What is the difference between MAX mode operation and MIN mode operation in 8086μp?

(e) What is the function of BHE pin in 8086 μp?

9.20 (a) Explain how 20-bit physical address is generated in 8086 microprocessor?
(b) What is the purpose of queue? How many words does the queue store in the 8086 microprocessor?
(c) How does 8086 support pipelining? Explain.

9.21 Draw the architecture of 8086. What are the main functions of BIU and EU unit of 8086 μp?

9.22 What are the different interrupts of 8086 microprocessor? Discuss software interrupts of 8086 in detail.

9.23 Draw the block diagram of external hardware interrupt interface with 8086/8088 microprocessor and explain briefly.

9.24 Discuss interrupt cycle of 8086 microprocessor with the help of flowchart of interrupt.

9.25 What are the interrupt pins of 8088/8086 microprocessors?

9.26 What is the priority order of 8086/8088 interrupts?

9.27 Draw the EPROM 27C256 interfacing with 8086/8088 microprocessor and state the memory map of EPROM IC with starting address 60000H

9.28 When the starting address of 2K EPRDM IC is FF800H, draw the EPROM Interfacing with 8086 microprocessor.

Multiple-Choice Questions

9.1 8086 has
(a) 16-bit data bus and 20-bit address bus (b) 8-bit data bus and 20-bit address bus
(c) 16-bit data bus and 16-bit address bus (d) 8-bit data bus and 16-bit address bus

9.2 16-bit register of 8086 consists of
(a) 16 flags (b) 8 flags (c) 9 flags (d) 7 flags

9.3 Instruction queue of 8086 consists of
(a) 6 data (b) 8 data (c) 4 data (d) 10 data

9.4 Instruction queue of 8088 consists of
(a) 6 data (b) 8 data (c) 4 data (d) 10 data

9.5 8086 has
(a) 6 memory segments (b) 8 memory segments
(c) 4 memory segments (d) 10 memory segments

9.6 Physical memory of 8086 is
(a) 1MB (b) 64 KB (c) 2 MB (d) 4 MB

9.7 Memory map of 8086 is
(a) 0000H to FFFFH (b) 00000H to FFFFFH
(c) 0000H to FFFFH (d) 0000H to FFFFH

9.8 Segment memory capacity of 8086 is
(a) 1 MB (b) 64 KB (c) 2 MB (d) 4 MB

9.9 Physical address of 8086 is
(a) 8-bits (b) 16-bits (c) 20-bits (d) 32-bits

9.10 Clock frequency of 8086 and 8088 is

(a) 5–10 MHz (b) 2–3 MHz (c) 1–3 MHz (d) 2–5 MHz

9.11 The 8086/88 can be operated in a single step when

(a) TF set (b) DF set (c) SF set (d) AF set

9.12 The physical address when DS = 2345H and IP = 1000H is

(a) 23450H (b) 24450H (c) 12345H (d) 2345H

9.13 What is the vector location of NMI?

(a) 00000H (b) 00008H (c) 00010H (d) 00014H

9.14 The total memory space available in 8086 is

(a) 16 KB (b) 64 KB (c) 1 MB (d) 256 KB

9.15 The number of mutiplexed buses in case of 8086 is

(a) 16 (b) 8 (c) 20 (d) 4

9.16 8086 exchanges data word with odd memory bank when

(a) BHE = 0 and $A_0 = 0$ (b) BHE = 0 and $A_0 = 1$

(c) BHE = 1 and $A_0 = 0$ (d) BHE = 1 and $A_0 = 1$

9.17 The segment and offset address of the instruction to be executed by 8086 microprocessor are pointed by

(a) CS and SI (b) DS and IP (c) CS and SP (d) CS and IP

9.18 What are the conditions that BIU can suspend fetching instruction?

(a) Current instruction requires access to emmroy or I/O port

(b) a transfer control (Jump or Call) instruction occurs

(c) Instruction queue is full

(d) All of these

Answers to Multiple-Choice Questions

9.1 (a)	9.2 (c)	9.3 (a)	9.4. (c)
9.5 (c)	9.6 (a)	9.7 (b)	9.8. (b)
9.9 (c)	9.10 (a)	9.11 (a)	9.12. (b)
9.13 (b)	9.14 (c)	9.15 (a)	9.16 (b)
9.17 (d)	9.18 (d)		

CHAPTER

10

Instruction Set of 8086 Microprocessor

10.1 INTRODUCTION

An instruction is a basic command given to a microprocessor to perform a specified operation with given data. Each instruction has two groups of bits. One group of bits is known as *operation code (opcode)* which defines what operation will be performed by the instruction. The other field is called *operand* which specifies data that will be used in arithmetic and logical operation. The addressing mode is used to locate the operand or data. There are different types of addressing modes depending upon the location of data in the 8086 processor.

The instruction format should have one or more number of fields to represent the instruction. The first field is called operation code or opcode field, and other fields are known as operand fields. The microprocessor executes the instruction based on the information of opcode and operand fields.

In this chapter, the general instruction format, different addressing modes of 8086/8088 processor along with examples are discussed. All types of instructions with examples are discussed elaborately. This chapter creates a background of assembly language programming of 8086/8088 processor.

10.2 ADDRESSING MODES

An instruction is divided into operation code (opcode) and operands. The opcode is a group of bits which indicate what operation can be performed by the processor. An operand is also known as data (datum) and it can identify the source and destination of data. The operand can specify a register, a memory location in any one of the memory segments or I/O ports. Figure 10.1 shows a general instruction format which consists of six bytes. Some instructions only have opcode and are called *single byte instructions*. Some instructions contain one, two, three or four byte operands. The detailed operation of instruction sets is explained in Section 10.3.

There are different ways to specify an operand. Each way of how an operand can be specified is called an *addressing mode*. The different addressing modes of 8086 microprocessors are as follows:

- Immediate addressing

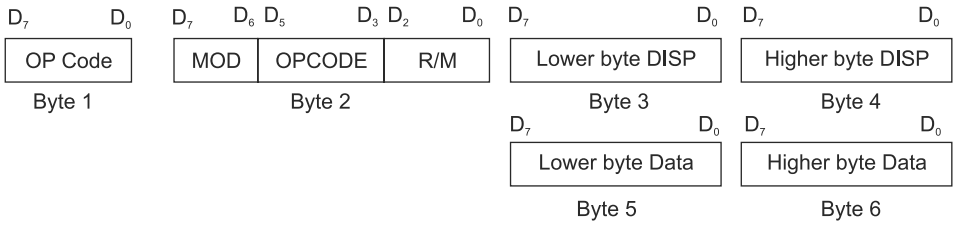


Fig. 10.1 General 8086 instruction format

- Register addressing
- Memory addressing
- Branch addressing

10.2.1 Immediate Addressing

In this mode of addressing, the 8-bits or 16-bits operand is a part of the instruction. For example, MOV AX, 4000H. In this instruction, the data 4000H can be loaded to the AX register immediately. Some other examples are given below:

MOV BX, 7000H; load 7000H in BX register

MOV CX, 4500H; store 4500H in CX register

10.2.2 Register Addressing

In 8086 microprocessor, some instructions are operated on the general-purpose registers. The data is in the register that is specified by the instruction. The format for register addressing is

MOV Destination, Source

In this instruction, the data from the source register can be copied into the destination register. The 8-bit registers (AL, AH, BL, BH, CL, CH, DL, DH) and 16-bit registers (AX, BX, CX, DX, SI, DI, SP, BP) may be used for this instruction. The only restriction is that both operands must be of the same length. For example,

MOV AL, BL ; Copies the value of BL into AL

MOV AX, BX ; Copies the contents of BX into AX

10.2.3 Memory Addressing

Memory addressing requires determination of a physical address. The physical address can be computed from the content of segment address and an effective address. The segment address identifies the starting location of the segment in the memory and the effective address represents the offset of the operand from the beginning of this segment of memory. The 20-bit effective address can be made up of base, index, and displacement. The basic formula for a 16-bit effective address and a 20-bit physical address is given below:

16-bit EA = Base + Index + Displacement

20-bit PA = Segment × 10 + Base + Index + Displacement

Memory addressing has the following combinations:

- Direct addressing
- Register indirect addressing
- Based addressing

- Indexed addressing
- Based Indexed addressing
- Based Indexed with displacement addressing

Direct Addressing In this mode of addressing, the instruction operand specifies the memory address where data is located. This addressing mode is similar to the immediate addressing mode, but the opcode follows an effective address instead of data. This is the most common addressing mode. The *displacement-only addressing mode* consists of an 8-bit or 16-bit constant that specifies the offset of the actual address of the target memory location. For example, MOV AX, [5000] copies 2 bytes of data starting from memory location DS × 10 + 5000H to the AX register. The lower byte is at the location DS × 10 + 5000H and the higher byte will be at the location DS × 10 + 5001H. Another example is MOV AL, DS:[5000H]. In this instruction, the content of the memory location DS × 10 + 5000H loads into the AL register.

In the instruction MOV DS:[2000H], AL means that the content of AL register is moved to memory location DS × 10 + [2000H]. The computation of memory location for the operand is illustrated in Fig. 10.2. In this figure, the effective address EA is 2000H and the physical address PA is PA = DS × 10 + EA = 4000 × 10 + 2000 = 42000. The physical address PA computation for other segment registers with same effective address is given below:

$$PA = CS \times 10 + EA,\ PA = SS \times 10 + EA, \text{ and } PA = ES \times 10 + EA$$

Generally, all displacement values or offsets are added with the data segment to determine the physical address. If anything other than data segment is required, we must use a segment override prefix before the address. For example, to access memory location 4000H in the stack segment (SS), the instruction will be MOV AX, SS: [2000H]. Similarly, to access the memory location in the extra segment (ES), the instruction will be written as MOV AX, ES: [2000H].

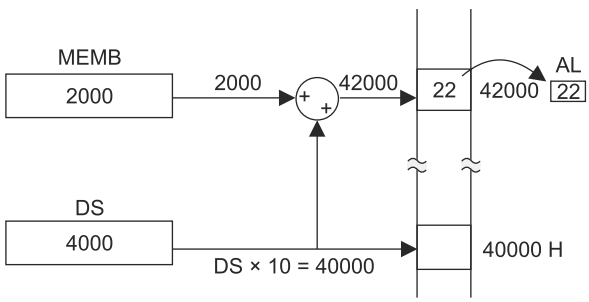


Fig. 10.2 Direct memory addressing

Register Indirect Addressing This instruction specifies a register containing an address, where data is located. The effective address of the data is in the base register BX or an index register that is specified by the instruction. This addressing mode works with index registers SI, DI, and base registers BX and BP registers.

The examples of this addressing mode in the 8086 microprocessor are as follows:

MOV AL, [BX]
MOV AH, [DI]
MOV AL, [SI]
MOV AH, [BP]

The BX, BP, SI, or DI registers use the DS segment by default. The base pointer uses a stack segment by

default. The segment override prefix symbols are used to access data in different segments. The examples of segment override instructions are as follows:

MOV AL, CS : [BX]

MOV AL, DS : [BP]

MOV AL, SS : [SI]

MOV AL, ES : [DI]

The effective address EA may either be in a base register (BX or BP) or in an index register (SI and DI). The physical address can be computed based on contents of the segment register, BX, BP, SI and DI registers as given below:

$$PA = CS \times 10 + BX,\ PA = DS \times 10 + BP,\ PA = SS \times 10 + DI, \text{ and } PA = ES \times 10 + SI$$

The general physical address expression for register indirect memory operand is depicted in Fig. 10.3. The content of BX is 1000H and CS is 2000. Then the physical address is CS × 10 + BX = 2000 × 10 + 1000 = 21000H. After execution the MOV AL, [BX] instruction, the contents of the memory location 21000H which is 44H will be stored in the AL register.

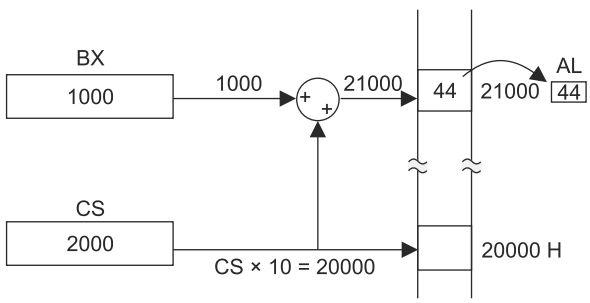


Fig. 10.3 Register indirect addressing

Based Addressing The 8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP), and the resulting value is a pointer to specify the location where the data resides. The effective address in the based addressing mode is obtained by adding the direct or indirect displacement to the contents of either the base register BX or the base pointer BP. The effective address and physical address computation are given below:

EA = BX + 8-bit displacement EA = BP + 8-bit displacement

EA = BX + 16-bit displacement EA = BP + 16-bit displacement

PA = Segment × 10 + BX + 8 bit displacement, PA = Segment × 10 + BP + 8 bit displacement

PA = Segment × 10 + BX + 16 bit displacement, PA = Segment × 10 + BP + 16 bit displacement

Segment will be any one of segments CS, DS, SS and ES. Figure 10.4 shows the physical address computation in base addressing mode. When 16-bit displacement DISP = 0025H, the contents of BX register is 0500H and the contents of DS register is 4000H, the physical address = DS × 10 + BX + DISP = 4000H × 10 + 0500 + 0025 = 40525H. After execution of MOV AL, DS: [BX + DISP] instruction, the contents of the memory location 40525 will be copied into AL register. The examples of base addressing mode instructions in 8086 microprocessor are

MOV AL, [BX+8-bit DISP]

MOV AH, [BX+8-bit DISP]

MOV AL, [BP + 8-bit DISP]
MOV AH, [BP + 8-bit DISP]

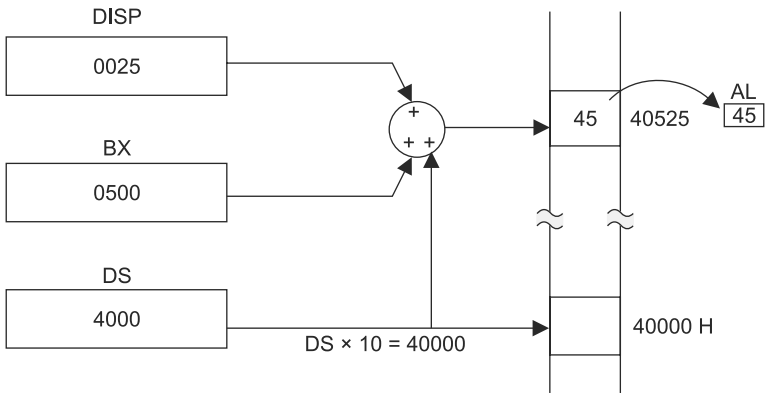


Fig. 10.4 Base addressing

Indexed Addressing These addressing modes can work similar to the based addressing mode. The 8-bit or 16-bit instruction operand is added to the contents of an index register (SI or DI), and the resulting value is a pointer to specify the location where data resides.

The displacement value is used as a pointer to the starting point of an array of data in memory and the contents of the specified register is used as an index. The EA and PA in the indexed addressing are as follows:

EA = SI + 8-bit displacement EA = DI + 8-bit displacement

EA = SI + 16-bit displacement EA = DI + 16-bit displacement

PA = Segment × 10 + SI + 8-bit displacement, PA = Segment × 10 + DI + 8-bit displacement

PA = Segment × 10 + SI + 16-bit displacement, PA = Segment × 10 + SI + 16-bit displacement

Segment will be any one of segment registers (CS, DS, SS and ES).

The index addressing modes generally involve BX, SI, and DI registers with the data segment. The [BP+ DISP] addressing mode uses the stack segment by default. In the register indirect addressing modes, the segment override prefixes can be used to specify different segments. The examples of these instructions are as follows:

MOV AL, SS: [BX+DISP] MOV AL, ES: [BP+DISP]

MOV AL, CS: [SI+DISP] MOV AL, SS: [DI+DISP]

Figure 14.5 shows the physical address computation in index addressing mode. When 16-bit displacement DISP = 0055H, the contents of SI is 0100H and the contents of DS register is 4000H, the physical address = DS × 10 + SI + DISP = 4000H × 10 + 0100 + 0055 = 40155H. If MOV AL, DS: [SI + 0025] is executed, the contents of memory location 40155H, FFH will be loaded into AL register.

Based Indexed Addressing The contents of a base register (BX or BP) is added to the contents of an index register (SI or DI), and the resulting value is a pointer to specify the location where the data resides. The effective address is the sum of a base register and an index register which are specified in the instruction. The based indexed addressing modes are simply combinations of the register indirect addressing modes. These addressing modes form the offset by adding together a base register (BX or BP) and an index register (SI or DI). The EA and the PA computation are given below:

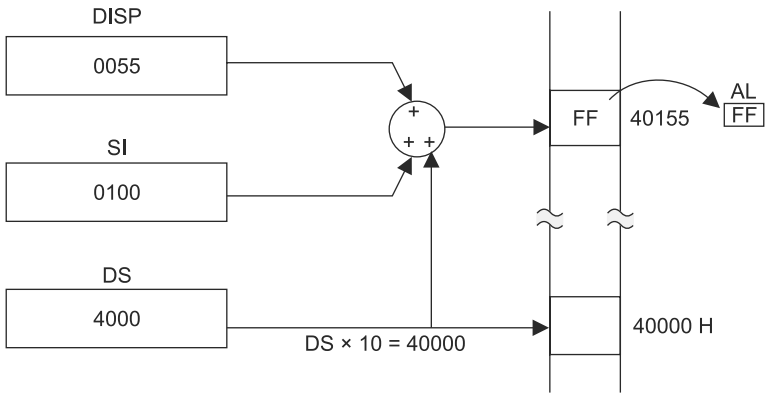


Fig. 10.5 Indexed addressing

EA = BX+SI EA = BX+DI

EA = BP+SI EA = BP+DI

PA= Segment × 10 + BX+SI, PA= Segment × 10 + BX+DI

PA= Segment × 10 + BP+SI, PA= Segment × 10 + BP+DI

Figure 10.6 shows the physical address computation in based indexed addressing mode. For example, if the content of BX register is 0200H and SI contains 0100H then the instruction MOV AL, [BX + SI] states that load the content of memory location DS × 10 + BX + SI into AH register. If DS = 4000H, the memory location address is 4000 × 10 + 0200 + 0100 = 40300H whose content 66H will be loaded into the AH register. The examples of this addressing mode instruction are as follows:

MOV AL, [BX + DI] MOV AL, [BX + SI]

MOV AL, [BP + SI] MOV AL, [BP + DI]

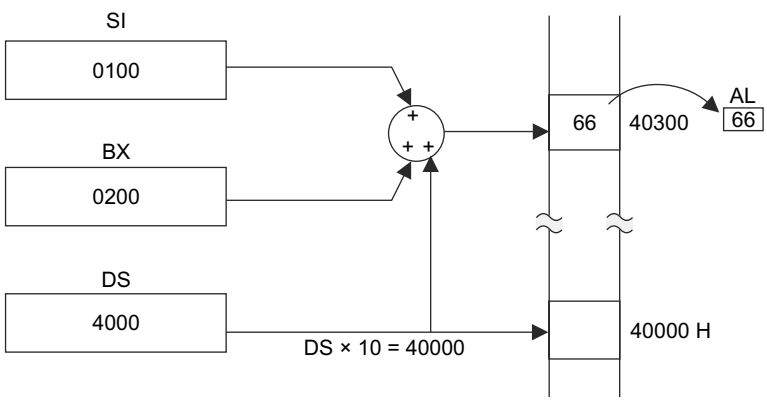


Fig. 10.6 Based indexed addressing

Based Indexed with Displacement Addressing The 8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP) and index register (SI or DI), and the resulting value is a pointer to specify the location where data resides. The effective address is the sum of an 8-bit or 16-bit displacement and based index address. The computation of EA and PA are given below:

EA = BX + SI + 8-bit or 16-bit instruction EA = BX + DI 8-bit or 16-bit instruction

EA = BP + SI + 8-bit or 16-bit instruction EA = BP + DI 8-bit or 16-bit instruction

PA = Segment × 10 + BX+SI + 8-bit or 16-bit instruction

PA = Segment × 10 + BX+DI + 8-bit or 16-bit instruction

PA = Segment × 10 + BP+SI + 8-bit or 16-bit instruction

PA = Segment × 10 + BP+DI + 8-bit or 16-bit instruction

Figure 10.7 shows the physical address computation in based indexed with displacement addressing mode. When 16-bit displacement DISP = 0020H, the contents of BX register is 4000H, the contents of SI is 0300 and the contents of DS register is 5000H, the physical address = DS × 10 + BX + SI + DISP = 5000H × 10 + 4000 + 0300 + 0020 = 54320H. When MOV AL, DS:[BX+SI+DISP] is executed, the content of memory location 54320H will be copied into AL register. The examples of this addressing mode instruction are as follows:

MOV AL, [BX+DI+DISP] MOV AL, [BX+SI+DISP]

MOV AL, [BP+SI+DISP] MOV AL, [BP+DI+DISP]

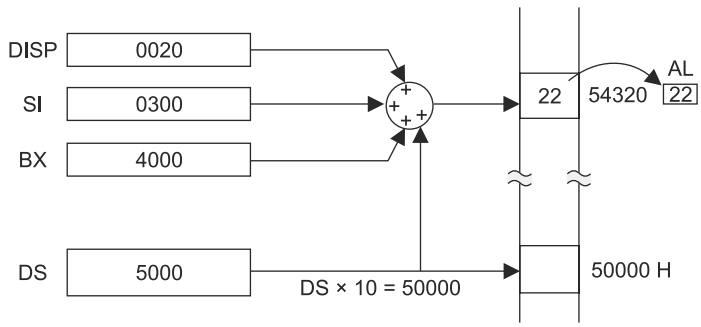


Fig. 10.7 Based indexed with displacement addressing

String Addressing Mode String is a sequence of bytes or words which are stored in the memory. Store characters in word processors and data tables are examples of strings. Some instructions are designed to handle a string of characters or data. These instructions have a special addressing mode where DS:SI is used as source of string and ES:DI is used to locate the destination address of a string. For example, MOV SB instruction is used to move a string of source data to the destination location.

When any MOV instruction is executed, data is always transferred from source to destination. The MOV instruction for different addressing modes is depicted in Table 10.1. The effective address computation depends on MOD and R/M bit patterns as shown in Table 10.2. Segment registers for different addressing modes may be different and its selection also depends on MOD and R/M as illustrated in Table 10.3.

Table 10.1 Addressing modes of 8086/8088 microprocessors

Addressing mode	Mnemonic	Symbolic Operation	Destination of Operand	Source of Operand	Functions
Immediate addressing mode	MOV AX, 2000H	AH←20H; AL←00	AX register	Data 2000	Source of data is within instruction

Contd.

Contd.

Register addressing mode	MOV AX, BX	AX←BX	AX register	BX register	Source and destination of data are registers of microprocessors
Direct addressing mode	MOV AH, [0400]	AH← [0400H]	AH register	0400H = Displacement	Memory address is available within the instruction
Register Indirect addressing mode	MOV AX, [SI]	AL← [SI]; AH←[SI + 1]	AX register	SI+DS×10= memory location	Memory address is supplied in any index or pointer registers
Indexed addressing mode	MOV AX, [SI + 06]	AL← [SI + 6]; AH←[SI + 7]	AX register	[SI+06]+ DS×10 = memory location	Memory address is the sum of the indexed register and a displacement within the instruction
Based addressing mode	MOV AX, [BP]	AL← [BP] ; AH←[BP + 1]	AX register	BP+DS×10 = memory location	Memory address is the content of BX or BP register within instruction
Based and Indexed addressing mode	MOV [BX + SI], AX	[BX + SI] ← AL; [BX + SI + 1] ← AH	BX+SI+DS× 10 = memory location	AX register	Memory address is the sum of an index register and a base register
Based and indexed with displacement addressing mode	MOV AX, [BX + SI + 10]	AL←[BX + SI + 10]: AH←[BX + SI + 11]	AX register	[BX+SI+10]+ DS×10 = memory location	Memory address is the sum of an index register, a base register and a displacement within instruction
Strings addressing mode	MOV SB	[ES : DI]← [DS : SI] If DF = 0, then SI←SI + 1; DI ←DI + 1. If DF = 1, then SI←SI – 1; DI←DI – 1.	DI + ES × 10 = memory location	SI + DS × 10 = memory location	The memory source address is register SI in the data segment. The memory destination address is register DI in the extra segment.

Table 10.2 Effective addressing computations corresponding to MOD and R/M fields

R/M	MOD 00	MOD 01	MOD 10	MOD 11	
				W=0	W=1
000	[BX]+[SI]	[BX]+[SI] + 8-bit DISP	[BX]+[SI]+ 16-bit DISP	AL	AX
001	[BX]+[DI]	[BX]+[DI] + 8-bit DISP	[BX]+[DI] + 16-bit DISP	CL	CX
010	[BP]+[SI]	[BP]+[SI] + 8-bit DISP	[BP]+[SI] + 16-bit DISP	DL	DX
011	[BP]+[DI]	[BP]+[DI] + 8-bit DISP	[BP]+[DI] + 16-bit DISP	BL	BX
100	[SI]	[SI] + 8-bit DISP	[SI] + 16-bit DISP	AH	SP
101	[DI]	[DI] + 8-bit DISP	[DI] + 16-bit DISP	CH	BP
110	[Direct Address]	[Direct Address] + 8-bit DISP	[Direct Address] + 16-bit DISP	DH	SI
111	[BX]+[SI]	[BX]+[SI] + 8-bit DISP	[BX]+[SI] + 16-bit DISP	BH	DI
	Memory Mode			Register Mode	

Table 10.3 Segment registers for different addressing modes corresponding to MOD and R/M fields

R/M	MOD 00	MOD 01	MOD 10	Segment Register
000	[BX]+[SI]	[BX]+[SI]+ 8-bit DISP	[BX]+[SI]+ 16-bit DISP	DS
001	[BX]+[DI]	[BX]+[DI] + 8-bit DISP	[BX]+[DI] + 16-bit DISP	DS
010	[BP]+[SI]	[BP]+[SI] + 8-bit DISP	[BP]+[SI] + 16-bit DISP	DS
011	[BP]+[DI]	[BP]+[DI] + 8-bit DISP	[BP]+[DI] + 16-bit DISP	DS
100	[SI]	[SI] + 8-bit DISP	[SI] + 16-bit DISP	DS
101	[DI]	[DI] + 8-bit DISP	[DI] + 16-bit DISP	DS
110	[Direct Address]	[Direct Address] + 8-bit DISP	[Direct Address] + 16-bit DISP	DS or SS
111	[BX]+[SI]	[BX]+[SI] + 8-bit DISP	[BX]+[SI] + 16-bit DISP	DS

8-bit DISP = 8-bit displacement 16-bit DISP = 16-bit displacement

10.2.4 Branch Addressing

The basic types of branch addressing are shown in Fig. 10.8. The intrasegment mode used to transfer the control to a destination that lies in the same segment where the control transfer instruction resides itself. In the intersegment mode, address is used to transfer the control to a destination that lies in the different segment.

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

During execution of program instruction, when the location to which the control to be transferred lies in a different segment other than the current one, the mode is called intersegment mode. If the destination location lies in the same segment the mode is called intrasegment mode.

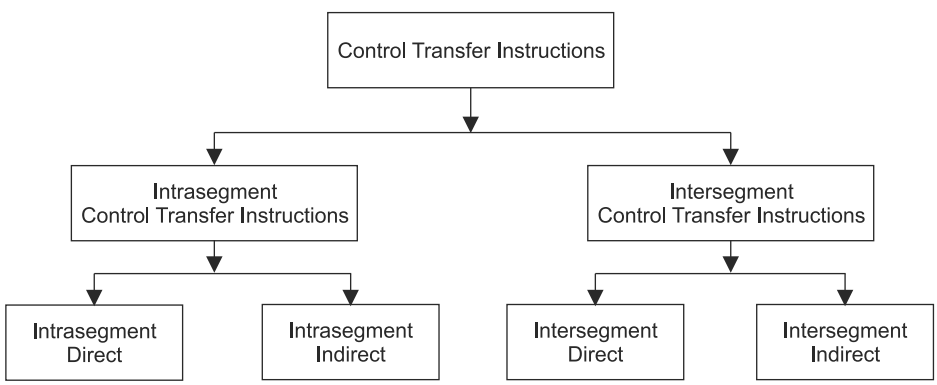


Fig. 10.8 Branch control transfer instructions

Intrasegment Direct In this branch addressing the effective branch address is sum of an 8-bit or 16-bit displacement and the current contents of IP. When the displacement is 8-bit long, it is referred to as a short jump. Intrasegment direct addressing is also referred as relative addressing because the displacement is computed 'relative' to the IP. It may be used with either conditional or unconditional branching, but a conditional branch instruction can have only 8-bit displacement.

Figure 10.9 shows intrasegment direct addressing. In the intrasegment direct mode, the destination location to which the control is transferred lies in the same segment where the control transfer instruction lies and appears directly in the instruction as an immediate displacement value. The displacement is relative to the contents of the IP. The expression for effective address in which the control is transferred is given below:

EA = Contents of IP + 8- bit or 16-bit displacement.

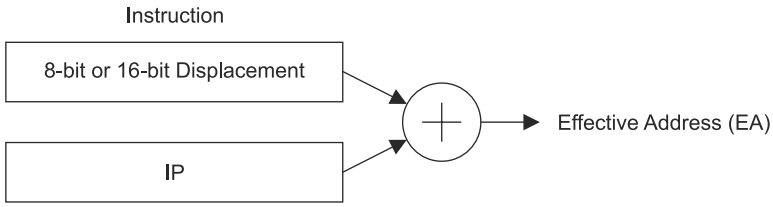


Fig. 10.9 Intrasegment direct addressing

Intrasegment Indirect In intrasegment indirect branch addressing the effective branch address is the contents of a register or memory location that is accessed using any of the addressing modes except the immediate mode. The contents of IP are replaced by the effective branch address. This addressing mode may be used only in unconditional branch instructions. Figure 10.10 shows intrasegment indirect addressing.

In this mode, the control to be transferred lies in the same segment where the control instructions lie and it is passed indirectly to the instruction. It uses unconditional branch instructions.

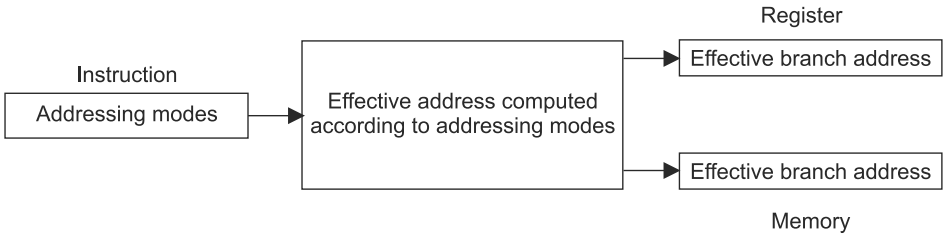


Fig. 10.10 Intrasegment indirect

Intersegment Direct This replaces the contents of IP with part of the instruction and the contents of CS with another part of the instruction. The purpose of this addressing mode is to provide a means of branching from one code segment to another. Figure 10.11 shows intersegment direct addressing.

During this mode of operation the location to which the control is to be transferred lies in a different segment, than in which the control transfer instruction lies, and is called intersegment. This addressing mode provides facility of branching from one segment to the other segment. The CS and IP specify the destination address directly in the instruction.

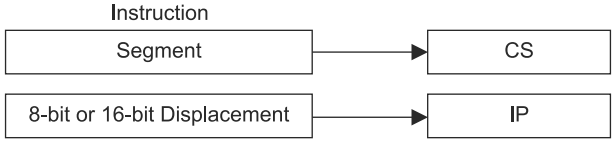


Fig. 10.11 Intersegment direct

Intersegment Indirect This replaces the contents of IP and CS with the contents of two consecutive words in memory that are referenced using any of the addressing modes except the immediate and register modes. Figure 10.12 shows intersegment indirect addressing.

During this mode of branch addressing the location to which the control is to be transferred lies in a different segment than the segment where the transfer control instruction lies and is passed to the instruction indirectly.

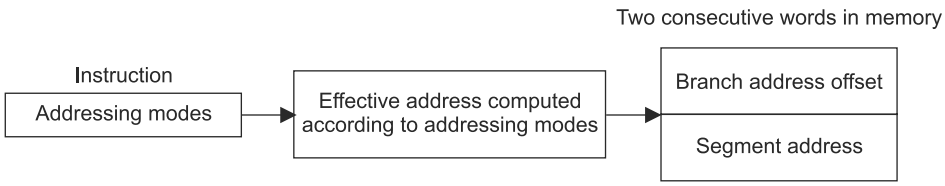


Fig. 10.12 Intersegment indirect

Example 10.1 Find the addressing modes of the following instructions:

(i) MOV CX, BX
(ii) MOV BX, 1234
(iii) MOV AX, [SI]
(iv) MOV [Offset Address], 2345
(v) MOV CX, [BX+SI]
(vi) MOV AX, [BX+SI+1234]

Sol.

(i) MOV CX, BX instruction is an example of register addressing mode.
(ii) MOV BX, 1234 instruction is an example of immediate addressing mode.
(iii) MOV AX, [SI] instruction is an example of indexed addressing mode.
(iv) MOV [offset address], 2345 instruction is an example of memory addressing mode.
(v) MOV CX, [BX+SI] instruction is an example of based indexed addressing mode.
(vi) MOV AX, [BX+SI+1234] instruction is an example of based indexed with displacement addressing mode.

10.3 8086 INSTRUCTION SET

The Intel 8086 Instruction Set is the core of the entire series of processors created by Intel. Some instructions have been added to this set to accommodate the extra features of later designs, but the set shown here contains the basic instructions understood by all of the processors. The 8086 instruction set consists of the following instructions:

- *Data Transfer Instructions* move, copy, load, exchange, input and output
- *Arithmetic Instructions* add, subtract, increment, decrement, convert byte/word and compare
- *Logical Instructions* AND, OR, exclusive OR, shift/rotate and test.
- *String Manipulation Instructions* load, store, move, compare and scan for byte/word

- *Control Transfer Instructions* conditional, unconditional, call subroutine and return from subroutine.
- *Input/Output Instructions*
- *Other Instructions* setting/clearing flag bits, stack operations, software interrupts, etc.

The instruction format consists of opcode and operand. Depending upon the opcode and number of operands present in the instruction, instructions are one byte to six bytes long. The general format of an instruction is illustrated in Fig.10.13.

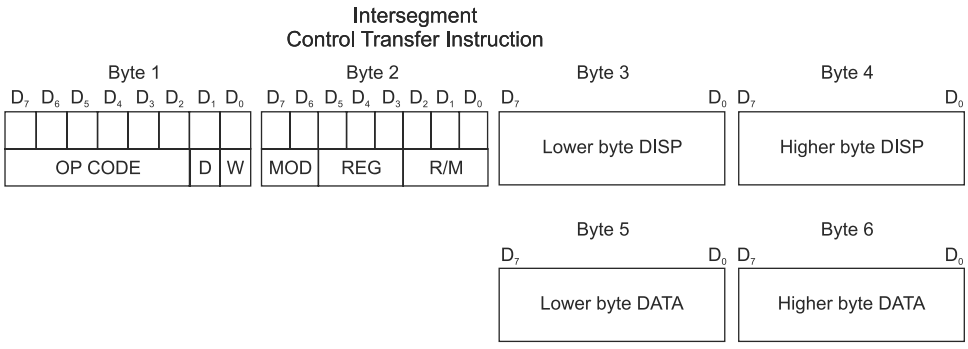


Fig. 10.13 Instruction format of 8086/8088 microprocessor

The first byte of any instruction is the opcode. The bits D_7 to D_2 specify the operation which will be carried out by the instruction. D_1 is the register direction bit (D). This bit defines whether the register operand in byte 2 is the source or destination operand. While D = 1, register operand is destination operand. If D = 0, the register operand is the source operand. D_0 represents data size (W), whether the data is of 8 bits or 16 bits. When W = 0, data is 8 bits and if W = 1, data will be 16 bits.

The second byte of the instruction specifies whether the operand is in the memory or the register. This byte consists of Mode (D_7 –D_6 bits), Register (D_5, D_4, D_3 bits) and R/M (D_2, D_1, D_0). The third and fourth bytes of the instruction specifiy lower 8-bit displacement and higher 8-bit displacement of memory respectively. Then the last two bytes (fifth and sixth) represent lower 8-bit data and higher 8-bit data.

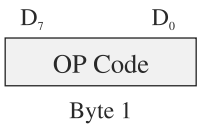
10.3.1 Classification of Instructions

Instructions are performed operations with 8-bit data and 16-bit data. 8-bit data can be obtained from a register or a memory location or input port. In the same way, 16-bit data may be available from any register pair or two consequent memory locations. Hence, binary codes of instructions are different. Due to different ways of specifying data for instructions, the machine or binary codes of all instructions are of different lengths. The Intel 8086/8088 instructions are classified into the following groups based on the number of bytes in instruction as given below:

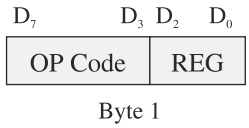
- One-byte instructions
- Two-byte instructions
- Three and four-byte instructions
- Five and six-byte instructions

One-byte Instructions This is a one-byte instruction which is used as opcode as well as data or operand. The least three bits of the opcode are used to specify the register operand.

One byte instruction – implied operand



One byte instruction – register mode



The example of one-byte instructions are XLAT, LAHF, SAHF, PUSH AX, POP DS, PUSHF and POPF. The opcode of these instructions are D_7 for XLAT, 9F for LAHF, 9E for SAHF, 50 for PUSH AX, 1F for POP DS, 9C for PUSHF, and 9D for POPF.

Two-byte Instructions Register-to-register and register to/from memory with no displacements instructions are two bytes long. In register-to-register instruction, the first byte of the code specifies the instruction operation (D_7–D_2) and width of the operand is specified by D_1–D_0 (W). The second byte represents the register operand and R/M field as given in Table 10.2.

The register to/from memory with no displacement instructions are same as register to register instructions except MOD field as depicted in Fig. 10.14. The MOD field can be used to represent different modes of addressing as given in Table 10.4.

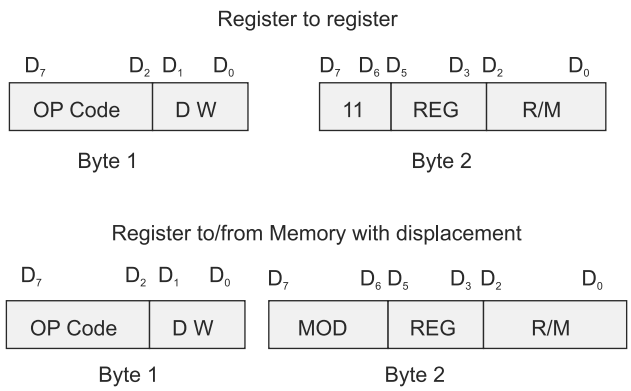


Fig. 10.14 Two-byte instructions

Table 10.4 Use of MOD field

MOD	Mode of addressing
00	Memory addressing without displacement
01	Memory addressing with 8-bit displacement
10	Memory addressing with 16-bit displacement
11	Register addressing with W = 0 for 8 bits and W = 1 for 16 bits

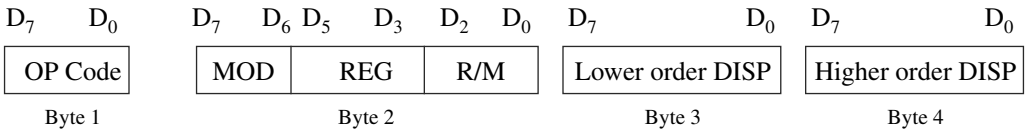
The example of two-byte instructions are MOV AX, BX; MOV AL, BL; IN AL, 01; and OUT 02, AL etc.

The opcode and operand of these instructions are as follows:

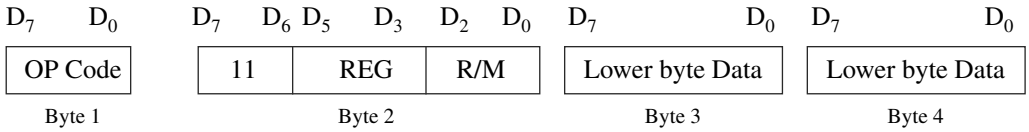
89 D8 for MOV AX, BX
88 D8 for MOV AL, BL
E4 01 for IN AL, 01
E6 02 for OUT 02, AL

Three-Byte and Four-Byte Instructions Register to/from memory with displacement and immediate operand to register instructions are four-byte instructions. The register to/from memory with displacement instruction consists of one or two additional bytes for displacement and the 2 bytes of the register to/from memory without displacement as given below:

Register to/from Memory with displacement



In immediate operand to register instruction, the first byte and the three bits D_5–D_3 of second byte are used for opcode. This instruction consists of one or two bytes of immediate data. The format of the instruction is given below:



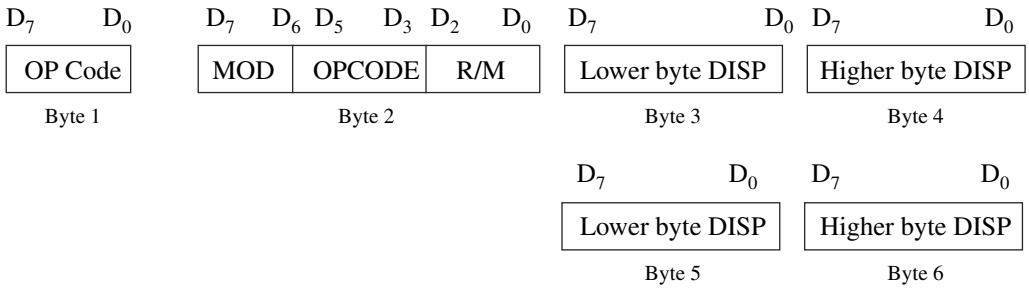
When the instruction consists of one byte of immediate data, it acts as three byte instructions. If the instruction has two bytes of immediate data, it works as four byte instructions. The example of three-byte instructions are MOV SI, 0300; MOV CX, 0005; MOV DI, 0100; MOV AL, [0300], and MOV [0400], AL, etc. The opcode and operands of these three-byte instructions are given below:

BE 00 03 for MOV SI,0300
B9 05 00 for MOV CX,0005
BF 00 01 for MOV DI,0100
A0 00 03 for MOV AL, [0300] and
A2 00 04 for MOV [0400], AL.

The four-byte instructions are MOV [BX+SI+1000], AX; MOV [BX+DI+0447], CL; MOV [BX+SI+0300], SP; MOV [BP+SI+0400], DL; and MOV [BP+DI+0100], BL. The opcode and operands of four-byte instructions are as follows:

89 80 00 10 for MOV [BX+SI+1000], AX
88 89 47 04 for MOV [BX+DI+0447], CL
89 A0 00 03 for MOV [BX+SI+0300], SP
88 92 00 04 for MOV [BP+SI+0400], DL and
88 9B 00 01 for MOV [BP+DI+0100], BL

Five-and Six-Byte Instructions Immediate operand to memory with 16-bit displacement instructions are six-byte instructions. The first two bytes represent OPCODE, MOD, OPCODE and R/M fields. The other four bytes consist of 2 bytes for displacement and 2 bytes for data as given below:



Immediate operand to Memory with 16-bit displacement. The example of five byte instructions are MOV [BX+ SI]+DISP, 22; MOV [BX+ DI] + DISP, 66; MOV [SI]+DISP, 44, etc. The opcode and operands of these five byte instructions are given below:

C6 80 $DISP_L$ $DISP_H$ 22 for MOV [BX+ SI]+DISP, 22
C6 81 $DISP_L$ $DISP_H$ 66 for MOV [BX+ DI]+DISP, 66
C6 84 $DISP_L$ $DISP_H$ 44 for MOV [SI]+DISP, 44

The six-byte instructions are MOV [BX+SI+1000], 2345; MOV [BX+DI+0447], 2000; MOV [SI+0300], 4466; The opcode and operands of six-byte instructions are as follows:

C7 80 00 10 45 23 for MOV [BX+SI+1000], 2345
C7 81 47 04 00 20 for MOV [BX+DI+0447], 2000
C7 84 00 03 66 44 for MOV [SI+0300], 4466

The 8086 instruction set can be divided into different categories based on their functions as follows:

Data Transfer Instructions These types of instructions are used to copy data from the source operand to the destination operand. All copy, store, move, load, input and output instructions fall in this category. Examples of instructions of this group are MOV, LDS, XCHG, PUSH and POP.

Arithmetic and Logical Instructions All the instructions performing arithmetic, logical, increment, decrement, compare and scan instructions are in this category. For example, ADD, SUB, MUL, DIV, INC, CMP and DAS, AND, OR, NOT, TEST, XOR.

Branch Instructions These instructions transfer control of execution to the specified address. All CALL, JUMP, interrupt and return instructions belong to this category.

Loop Instructions These instructions have REP prefix with CX used as count register, they can be used to implement unconditional and conditional loops. The LOOP, LOOPNZ and LOOPZ instructions are in this category. Usually, these instructions are used to implement different delay loop.

Processor Control Instructions These instructions control the machine status. CLC, CMC, CLI, STD, STI, NOP, HLT, WAIT and LOCK instructions are example of machine control instructions.

Flag Manipulation Instructions All these instructions which directly affect the flag register come under this group of instructions. CLD, STD, CLI, STI instructions belong to this category.

Shift and Rotate Instructions These instructions perform the bitwise shifting OR rotation in either direction with or without a count in CX register. The examples of instructions are RCL, RCR, ROL, ROR, SAL, SHL, SAR and SHR.

String Instructions These instructions perform various string manipulation operations like load, move, scan, compare, store, etc. The examples of string instructions are MOVS, LODS and STOS.

10.3.2 Data Transfer Instructions

The data transfer instructions are used to transfer data between registers, register and memory, register and immediate data, memory and immediate data. All data transfer instructions are explained in this section.

MOV Destination, Source (Copy data from source to destination)

Destination ← Source, Flag affected: None

This instruction performs data movement between registers, register and memory, register and immediate data, memory and immediate data, between two memory locations, between I/O port and register and between stack and memory or register. Both 8-bit and 16-bit data registers are used in data transfer instructions.

In case of immediate addressing mode, a segment register cannot be a destination register. Direct loading of the segment registers with immediate data is not permitted. To load the segment registers with immediate data, one will have to load any general-purpose register with data and then it will have to be moved to that particular segment register.

Destination	Source
Register	Immediate data
Memory	Immediate data
Register	Register
Register	Memory
Memory	Register
Segment	Memory
Memory	Segment
Register	Segment
Segment	Register

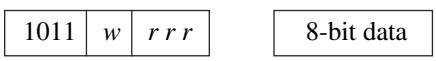
The MOV instruction cannot be able to

- set the value of the CS and IP registers.
- copy value of one segment register to another segment register (should copy to general register first). Copy immediate value to segment register (should copy to general register first).

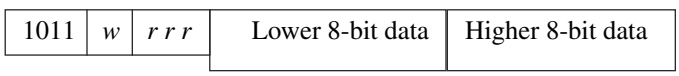
MOV Register, Immediate Data This instruction moves immediate 8-bit /16-bit data to the specified register. Its object code is either 2 or 3 bytes based on data.

The format of its object code is as follows:

w = 0 for 8 bit data



w = 1 for 16 bit data



$r\ r\ r$ = the address of register as illustrated in Table 10.2.

For example, MOV AL, 8-bit data

object code is 1011 $w\ r\ r\ r$ = 1011 0000 = B0 as w = 0 and $r\ r\ r$ = 000

If the instruction is MOV AL, FFH, the object code will be B0, FF H

MOV mem/reg, data When this instruction is executed, immediate 8-bit or 16-bit data moves to a specified register or a memory location(s) through this instruction. This is not used to transfer immediate data to a register. The format of its object code is given below:

w = 0 for 8 bits data

1100 011w	Mod 000 R/M	8-bit data

w = 1 for 16 bits data

1100 011w	Mod 000 R/M	Lower-8 bit data	Higher-8 bit data

Mod R/M and data are explained in Section 10.2.

In this instruction memory location can be addressed directly or by a register or a combination of register and displacement.

For example, MOV [0345], 23H

When this instruction is executed, 23H will be loaded into the memory location DS × 10 + 0345.The object code is as follows:

1100 011w	Mod 000 R/M	8-bit data

This instruction is direct addressing of a memory location. As per Table 10.2, for direct addressing mod = 00, R/M = 110 and w = 0 for 8-bit operation. Then object code is

1100 0110	00 000 110	23

= C6, 06, 23

Another example is MOV [0345], 2345H:

When this instruction is executed, 45H will be loaded into the memory location DS×10 + 0345 and 23H will be loaded into DS×10+0346. The object code format of this instruction as given below:

1100 011w	Mod 000 R/M	Lower-8 bit data	Higher-8 bit data

This instruction is direct addressing of a memory location. As per Table 10.2, for direct addressing Mod = 00, R/M = 110. W = 1 for 16-bit operation.

Then object code

1100 0110	00 000 110	45	23

= C7, 06, 45, 23

The example of MOV [reg], data instruction is MOV [BX], 45H. When this instruction is executed, data will be moved to the memory location specified by the content of BX register. The object code is

1100 011w	Mod 000 R/M	8-bit data

Here Mod = 00, R/M = 111 and w = 0 for 8-bit operation. Then object code is

1100 0110	00 000 111	45

= C6, 07, 45

MOV ACC, Memory When MOV ACC, memory instruction is executed, the 8-bit data moves from a memory location to AL or 16-bit data move from two consecutive memory locations to AX register. The object code of this instruction is

1010 000*w*	16-bit offset address

For $w = 0$, Object code = A0, Offset address

For $w = 1$, Object code = A1, Offset address

For example, MOV AL, [2340]. This instruction moves the content of offset address 2340H to AL and the object code is A0, 40, 23. If the content of memory location is 25H, after execution of MOV AL, [2340]; 25H will be stored in AL.

MOV Memory, ACC The content of accumulator will be stored into memory. This means that content of AL is stored in memory and contents of AX will be stored in two consecutive memory locations. The object code is

1010 001*w*	16-bit offset address

For $w = 1$, object code = A3, offset address

For $w = 0$, object code = A2, offset address

For example, MOV [4000], AL content of AL is stored in the memory location represented by offset address 4000H. Then object code is A2, 00, 40.

Another example is MOV [4000], AX. Content of AX is stored in the two consecutive memory locations represented by offset address 4000H.

MOV Memory/Register, Memory/Register This instruction is used to move 8-bit or 16-bit data from one register to another register, one memory to register and register to memory. This instruction cannot be used for data transfer from memory to memory. The object code is

1010 10d*w*	Mod reg R/M

The direction flag *d* is either 0 or 1. When $d = 0$, the specified register is the source of the operand. The Mod and R/M are used for the first operand (the content of memory/register-1) and reg represents the second operand (the content of memory/register-2). If $d = 1$, register specifies a register which works as the destination of the operand. The Mod and R/M are also used for the second operand (memory/register-2) and reg defines the first operand (mem/register-1).

For example, MOV BX, CX

This instruction is used for CX register to BX register data transfer. If $d = 0$, register specifies a register which is the source for the operand. When $d = 0$, Mod and R/M are used for the first operand, i.e., BX. As per Table 10.2 for BX, mod = 11, R/M = 011. The register defines the second operand, CX (source for the operand). Then reg is 001 and $w = 1$. In that case the object code is

1010 1001	Mod reg R/M

= 1000 1001, 11 001 011 = 89, CB

Mod for BX; reg for CX; R/M for BX

When $d = 1$, reg specifies a register which is used as the destination for the operand. If $d = 1$, Mod and R/M

are used for the second operand CX; and reg stands for the first operand BX as destination and $w = 1$. Then object code is 1000 1011, 11 011 001 = 8B, D9

Mod for CX	reg for BX	R/M for CX

Hence both codes 89, CB and 8B, D9 are valid for MOV BX, CX instruction.

Another example is MOV CX, [0500]. The object code of this instruction is

1000 10d*w*	Mod reg R/M

In this case, reg will specify register CX which acts as destination for the operand when $d = 1$. If $d = 1$, Mod and R/M are used direct addressing.

The mod = 00, R/M = 111, CX = 001 and $w = 1$

Then the object code is 1000 1011, 00 001 111= 8B, 0F

Example 10.2 Write instructions for the following operations:

(i) Move the content of DX register into SS register
(ii) Load 16 bit data from memory location offset address 0300 to AX
(iii) Load 8 bit data, FF in BL register
(iv) Source index address 0100 is stored in SI
(v) Destination index address 0400 is stored in DI

Sol.

(i) MOV SS,DX ; Move the content of DX register into SS register
(ii) MOV AX, [0300] ; Load 16 bit data from memory location offset address 0300 to AX
(iii) MOV BL, FF ; Load 8 bit data, FF in BL register
(iv) MOV SI, 0100 ; Source index address 0100 is stored in SI
(v) MOV DI, 0400; Destination index address 0400 is stored in DI

XCHG Destination, Source (Exchange data between source to destination)

Destination ↔ Source, Flag affected : None

Destination	Source
Accumulator	register
Memory	register
register	register

This instruction is used to exchange the contents of the specified source and destination operands, which may be registers or a memory location. But the exchange of contents of two memory locations is not allowed. Immediate data is not allowed in XCHG instructions. For example, XCHG [4000], AX exchange data between AX and a memory location represented by offset address 4000 H with the content of data segment. XCHG AX, BX instruction exchanges data between AX and BX. The data format for register to register and register to memory is

1000 011 *w*	Mod reg R/M

and the data format for register to accumulator is | 1001 0 reg |

The object code of XCHG AX,BX is 1000 011*w* Mod reg R/M = 1000 0111 11 000 011= 87 C3 as *w* = 1, Mod = 11, reg = 000 and R/M = 011. Similarly, the object code of XCHG AL, [BX] = 1000 0110, 00 000 111 = 86, 07 where *w* = 0, Mod = 00, reg = 000 and R/M = 111.

LAHF (Loads the lower flags byte into AH)

AH ← Flags, Flag not affected : O A C Z P

This instruction loads the lower byte of the flags word into the AH register. This command may be used to observe the status of all the condition code flags at a time. The LAHF instruction followed by a PUSH AX instruction has same effect of PUSH PSW instruction in 8085.

AH = flags register

AH bit:	7	6	5	4	3	2	1	0
	[SF]	[ZF]	[0]	[AF]	[0]	[PF]	[1]	[CF]

Here bits 1, 3, and 5 are reserved.

The object code of LAHF instruction is 9F

10001 1111

SAHF (Saves AH into lower flags byte)

Flags ← AH, Flag affected : None

This instruction saves the AH register bit pattern into the lower byte of the flags register.

The lower byte of 8086 flag register is same as flag register of 8085. The SAHF instruction replaces the equivalent flag byte of 8085 with a byte from AH register. POP PSW instruction of 8085 will be translated to POP AX SAHF on a 8086 processor. The SAHF instruction changes the flags in the lower byte of flag register. The object code of SAHF instruction is 9E.

10001 1110

IN port8 or DX (Input data from I/O device)

byte: AL ← port

word: AL ← [port]; AH ← [port+1] or AX ← (DX), Flag affected : None

This instruction is used to read data from an input port. The address of the input port can be specified within the instruction directly or indirectly. AL and AX registers can be used as destinations for 8-bit and 16-bit input operands respectively. DX is the only register which is allowed to carry the port address. To fetch a byte or word into AL or AX from an 8-bit port or the 16-bit address contained in DX, the 8-bit port supports the I/O technique of 8085 processors and inputs a byte or word from direct I/O ports 00H to FFH (0 to 255). When the port address consists of 16 bits, it must be addressed by DX. Input a byte or word from indirect I/O ports 0000H to FFFFH (0-65535); port address is in DX and flags are not affected. The object code of this instruction is

For fixed port	1110010*w*	port

For variable port	1110110*w*

The examples of IN instructions are

IN AL, 01; Load the content of 8-bit port address 01H to AL register. The object code of IN AL,01 is

1110010*w*, port address = E4, 01 as *w* = 0.

IN AX, DX; Read data from a 16-bit port address specified by DX register and stores it in AX register. The object code of IN AX, DX is 1110 110*w* = ED as *w* = 1.

OUT port8 or DX (Output data to I/O device)

byte: [port] ← AL

word: [port] ← AL [port+1]*AH or (DX← AX)

Flag affected : None

This instruction is used to write data (byte or word) on an output port. The address of the output port may be specified in the instruction directly or implicitly in DX. Therefore, the contents of AX or AL are transferred to a directly or indirectly addressed port after execution. This instruction can output a byte or word to direct I/O ports 00H to FFH (0 to 255).

It can send a byte or word to an 8-bit port address or the 16-bit port address contained in DX. The registers AL and AX are the allowed source operands for 8 bit and 16 bit operations respectively. If the port address is of 16 bits, it must be in DX. Output a byte or word to indirect I/O ports 0000H to FFFFH (0 to 65535); port address is in DX and flags are not affected. The object code of this instruction is

For fixed port	1110 011*w*	port
For variable port	1110 111*w*	

The examples of OUT instructions are OUT 02, AL and OUT DX, AX.

OUT 02, AL After execution of this instruction, sends the content of AL to a port address 02H. The object code of OUT 02, AL is 1110 011*w*, port address = E6, 02 as *w* = 0.

OUT DX, AX This instruction sends data available in AX to a port address which is specified by DX register. The object code of OUT DX, AX is 1110 111*w* = EF as *w* = 1.

Example 10.3 Write instructions for the following operations:

(i) Exchange the byte between memory location offset address 0300 and AL register
(ii) Load 8 bits of flags into AH register
(iii) Exchange the word between DX and AX registers
(iv) Copy a byte from the port address 03H to AL register
(v) Output the content of accumulator to port address 01H

Sol.

(i) XCHG AL, [0300] ; Exchange the byte between memory location offset address 0300 and AL register
(ii) LAHF ; Load 8 bits of flags in to AH register
(iii) XCHG DX, AX ; Exchange the word between DX and AX registers
(iv) IN AL, 03 ; Copy a byte from the port address 03H to AL register
(v) OUT 01, AX ; Output the content of accumulator to port address 01H

LEA reg16, addr (load effective address)

reg16 ← effective address (offset) of addr. Flag affected : None

Loads the effective address or offset of memory variable into reg16. This type of data transfer operation is important to load a segment or general-purpose register with an address directly from memory. The LEA

instruction is used to load a specified register with 16-bit offset address. The object code is

1000 1101	Mod reg R/M

For example, LEA SI, address states that the 16-bit effective address loads in the SI register.

LEA BX, ADR Effective address of ADR will be transferred to BX register. The object code of LEA BX, [0245] = 1000 1101 00 000 001 45 02 = 8D 1E 45 02.

LDS reg16, memory (load data segment)

reg16 ← [memory16]; DS ← [memory16 + 2] Flag affected: None

Loads the DS register and reg16 from memory with the segment and offset values. This instruction loads the specified register in the instruction with the content of memory location specified as source in the instruction. It also loads the contents of the memory locations following the specified memory locations into DS register. The object code is

1100 0101	Mod reg R/M

= C5, mod reg r/m

The example of LDS instruction is LDS AX, [BX]. Load the contents of memory locations specified by the content of BX register into AX. Here, mod for [BX] = 00, R/M for [BX] = 111 and reg for AX = 000. Then the object code = C5, 00000111 = C5, 07.

LES reg16, memory (Load extra segment)

reg16 ← [mem16]; ES ← [mem16+2] Flag affected : None

Loads the ES register and reg16 with the segment and offset values for the variable in memory. The object code is

1100 0100	Mod reg R/M

The example of LES instruction is LES CX, [4000]. The object code of LES CX, [4000] instruction is 1100 0100 00 001 110 00 40 = C4 0E 00 40.

XLAT (Translate byte in AL by table lookup)

AL ← DS : [BX+AL], Flag affected : None

This instruction is used to translate the byte in AL register by adding it to a base value in BX which has been set to locate the look-up table and the byte located is returned in AL. The physical address of memory location of look-up table is computed from DS: [BX+AL]. After execution XLAT, the data from the memory location of the look-up table is loaded into AL register. Using look-up table technique, this instruction is able to find out the codes in case of code conversion problems. The object code of XLAT instruction is 1101 0111 = D7.

Example 10.4 Write instructions for the following operations:

(i) Load the content of specified memory location represented by BX into AX register
(ii) Load the content of specified memory location represented by SI into AL register
(iii) Load the content of specified memory location represented by SI into AX register
(iv) Replace a byte in AL register with a byte from the look-up table

Sol.

(i) LDS AX,[BX] ; Load the content of specified memory location represented by BX into AX register
(ii) LODSB ; Load the content of specified memory location represented by SI into AL register
(iii) LODSW; Load the content of specified memory location represented by SI into AX register
(iv) XLAT; Replace a byte in AL register with a byte from the look-up table

10.3.3 PUSH and POP Instructions

These instructions are used to manipulate stack-related operations. All stack instructions are explained in this section.

PUSH Source Push source register onto stack

SP = SP-2; SS:[SP] ← source register, Flag affected : None

After execution of this instruction, the content of specified register is pushed onto the stack. The stack pointer (SP) is decremented by 2 after execution, and then stores the two-byte contents of the operand onto stack. Initially the higher byte is pushed and then the lower byte is pushed so that the higher byte is stored in the higher address and the lower byte is stored in the lower address. The actual operation of PUSH BX instruction is shown in Fig.10.15. The sequence of PUSH operation is as follows:

- The current stack top is already occupied in the stack segment memory. So that SP is decremented by one then store the content of BH into the address pointed by SP and stack segment SS.
- Again decrement SP by one and store BL into the memory location pointed by SP and stack segment SS.

In this way, SP is decremented by 2 and BH-BL contents are stored in the stack as shown in Fig.10.15.

Thereafter, contents of SP points to a new stack top. Assume the content of BX = 1234, SS = 4000 and SP = 01FF. Then content of BH, 12H is stored in 410FEH and content of BL, 34H is also stored in 410FDH.

The object code of PUSH instruction is

For register memory

1111 1111	Mod 110 R/M

For register

01010 reg

For Segment register

000 reg 110

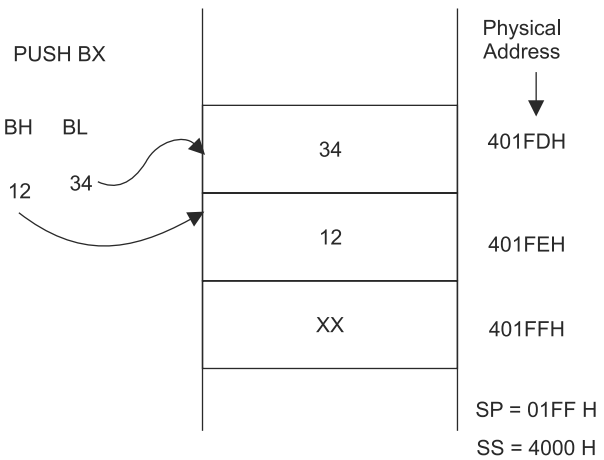


Fig. 10.15 The operation of PUSH BX instruction

The examples of these instructions are PUSH AX, PUSH BX, PUSH CX, PUSH DS, and PUSH [4000]. The object code for PUSH AX is 50H, for PUSH BX is 53H, for PUSH CX is 51H, for PUSH DS is 1E, for PUSH [4000] is FF 36 00 40.

PUSHF (Push flags word onto stack)

SP = SP-2; SS: [SP] ← flags, SP = SP-2; SS:[SP] ← Source, Flag affected : None

The push flag instruction pushes the content of flag register on the stack. First the upper byte $FLAG_U$ and then the lower byte $FLAG_L$ is pushed on it. The SP is decremented by 2, for each push operation. The general operation of this operation is similar to the PUSH operation. The object code of PUSHF is 100 111 00 = 9C.

POP destination (Pop word at top of stack to destination)

Destination ← SS:[SP]; SP = SP+2 Flag affected : None

When this instruction is executed, loads the specified register/memory location with the contents of the

memory location of which the address is formed using the current stack segment SS and stack pointer SP. The stack pointer is incremented by 2. The operation of POP instruction is exactly opposite of PUSH instruction. The actual operation of POP BX instruction is shown in Fig. 10.16. The sequence of POP operation is as follows:

- Copy the content of stack top memory location and stored in BL register and SP is incremented by one.
- Then content of memory location is pointed by SP are copied to BH register and SP is also incremented by 1.

Hence SP is incremented by 2 and points to next stack top as depicted in Fig.10.16. Assume 12H is stored in 410FEH and 34H is stored in 410FDH, the content of SS = 4000 and SP = 01FD. After execution of POP BX instruction, the content of memory location 410FDH is copied into BL and content of memory location 410FEH is also copied into BH.

The object code of POP is

For register/memory

1000 1111	Mod 000 R/M

For register

01011 reg

For Segment register

000 reg 111

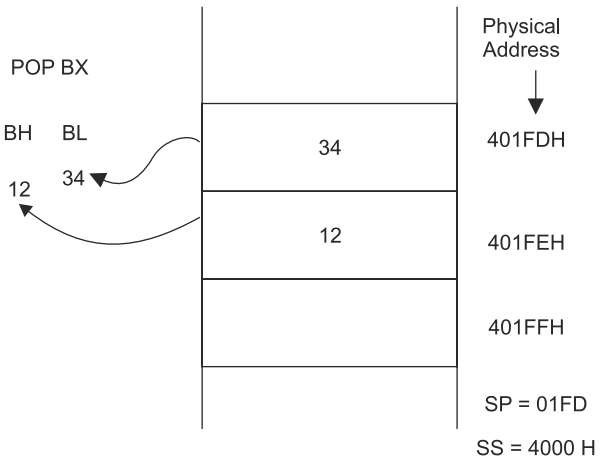


Fig. 10.16 The operation of POP BX instruction

The examples of POP instructions are POP AX, POP BX, POP CX, POP DS, and POP [4000]. The object code for POP AX is 58H, for POP BX is 5BH, for POP CX is 59H, for POP DS is 1F, for POP [4000] is 8F 06 00 40.

POPF (Pop word at top of stack to flags register)

flags ← SS:[SP]; SP = SP+2 Flag affected : All

The pop flags instruction loads the flags register completely from word contents of the memory location currently addressed by SP and SS. The SP is incremented by 2 for each pop operation.

The object code of POPF is 100 111 01 = 9D

Example 10.5 Write instructions for the following operations:

(i) Push the content of AX register onto the stack
(ii) Push the content of memory location offset address 0500 on to the stack
(iii) Store the content of stack top memory locations in AL and AH registers
(iv) Store the16-bit flags on to the stack
(v) Pop the top of the stack into the 16-bit flag word

Sol.

(i) PUSH AX ; Push the content of AX register on to the stack
(ii) PUSH [0500] ; Push the content of memory location offset address 0500 on to the stack

(iii) POP AX ; Store the content of stack top memory locations in AL and AH registers

(iv) PUSHF ; Store the16-bit flags on to the stack

(v) POPF ; Pop the top of the stack into the 16-bit flag word

10.3.4 Arithmetic Instructions

These instructions perform the arithmetic operations such as addition, subtraction, increment, decrement, negation, multiplication, division and comparing two values. The ASCII adjustment and decimal adjust instructions also belong to this type of instructions. The 8086/8088 instructions that handle these operations are ADD, ADC, SUB, SBB, INC, DEC, NEG , MUL, IMUL, DIV, IDIV, and other instructions such as AAA, AAD, AAM, AAS, DAA, and DAS. In this section, all arithmetic instructions are discussed below in significant details.

ADD Destination, Source (Add two operands, result remains in destination

Destination ← (Source + Destination), Flag affected : O S Z A P C

The ADD instruction adds the contents of source operand (Register or a memory location) specified in the instruction or an immediate data to the contents of destination (another register or memory location). After addition, the result is in the destination operand. But both the source and destination operands cannot be memory operands. It means that memory to memory addition is not possible. After addition the condition code flags, O, S, Z, A, P and C are affected, depending upon the result. The object code of ADD instruction is as follows:

Register/Memory with Register

0000 00*dw*	Mod reg R/M

Immediate to register/memory

1000 00*sw*	Mod 000 R/M	data	Data

Immediate to Accumulator

0000 010*w*	data	data

For example, ADD AX, 0100H instruction can add 16-bit immediate data (0100H) with the content of AX register and result is stored in AX register. The object code 0000 010*w*, 16-bit data. Here *w* = 1, the object code is 05, 00, 01.

The example of other ADD instructions are ADD AL, 22H; ADD AX, BX; ADD AL, [BX]; ADD [BX], CL and ADD [BX], CX. The object code for ADD AL, 22H is 04, 22, for ADD AX, BX is 01, D8, for ADD AL, [BX] is 02, 07, for ADD [BX], CL is 00, 0F and for ADD [BX], CX is 01, 0F.

ADC Destination, Source (Add two operands with carry from previous add)

Destination ← (Source + Destination + CF), Flag affected: O S Z A P C

The ADC instruction performs the same operation as ADD instruction, although the carry flag bit is added with the result. All the condition flags are affected after execution of this instruction. The object code of ADC instructions are as follows:

Register/Memory with Register

0001 00*dw*	Mod reg R/M

Immediate to register/memory

1000 00*sw*	Mod 010 R/M	data	data

Immediate to Accumulator

0001 010*w*	data	data

The examples ADC instructions are ADC AX, 1234H; ADC AX, CX; ADC AX, [SI]; ADC AX, [4000] ; ADC [SI], AX; and ADC [4000], BX. The object code for ADC AX, 1234H is 15 34 12; for ADC AX, CX is 11 C8; for ADC AX, [SI] is 13 04; for ADC AX, [4000] is 13 06 00 40; for ADC [SI], AX is 11 04; and for ADC [4000], BX is 11 1E 00 40.

SUB Destination, Source (Subtract source from destination, store result in destination)

Destination ← (Destination — Source), Flag affected : O S Z A P C

The *SUB destination, source* instruction subtracts the source operand from the destination operand and the result is stored in the destination operand. The source operand may be a register, memory location or immediate data. The destination operand may be a register or a memory location. But in an instruction, source and destination operands both will not be memory operands and destination operand must not be an immediate data. After execution of this instruction all the condition code flags, O, S, Z , A, P and C are affected.

The object code of SUB instruction is as follows:

Register/Memory with Register

0010 10*dw*	Mod reg R/M

Immediate to register/memory

1000 00*sw*	Mod 101 R/M	data	data

Immediate to Accumulator

0010 110*w*	data	data

For example SUB AX, 0100 Load 0100H to AX register immediately. The object code is

0010 110*w*	data	data

Here *w* = 1, object is 0010 1101, 16-bit data = 2D, 00, 01H

The other Examples of SUB instructions are SUB AL, 44H; SUB AX, BX; SUB AL, [BX]; SUB [BX], CL and SUB [BX], CX. The object code for SUB AL, 44H is 2C 44; for SUB AX, BX is 29 D8; for SUB AL, [BX] is 2A 07; for SUB [BX], CL is 28 0F and for SUB [BX], CX is 29 0F.

SBB Destination, Source (Subtract source and the carry flag bit from destination)

Destination ← ((Destination — Source) — CF), Flag affected : O S Z A P C

The SBB represents subtract with borrow. In this instruction, it subtracts the source operand and the borrow flag, which is the result of the previous operations, from the destination operand. The subtraction with borrow means that subtract 1 from the subtraction obtained by SUB. After subtraction, if carry is generated, a carry flag is set. The result is stored in the destination operand. All the flags O, S, Z, A, P and C are affected by this instruction. The object code is

Register/Memory with Register

0001 10*dw*	Mod reg R/M

Immediate to register/memory

1000 00*sw*	Mod 011 R/M	data	data

Immediate to Accumulator

0000 111*w*	data	data

For example, SBB AX, 0010. Subtract 0010H and the carry flag from AX register immediately. The object code is

0000 111*w*	data	data

Here $w = 1$, object is 0000 1111, 16-bit data = 0F, 00, 01H

The other Examples of SBB instructions are SBB AX, BX; SBB AL, [BX]; SBB [BX], CL and SBB [BX], CX, and SBB AX, [4000]. The object code for SBB AX, BX is 19 D8; for SBB AL, [BX] is 1A 07; for SBB [BX], CL is 18 0F; for SBB [BX], CX is 19 0F and for SBB AX, [4000] is 1B 06 00 40.

Example 10.6 Write instructions for the following operations:

(i) Add 2345H to the contents of AX register
(ii) Add 22H to the content of the specified memory location represented by the contents of BX register
(iii) Subtract the content of AX register from AX register
(iv) Subtract immediately 2345H from BX register with borrow
(v) Subtract immediately 1000 from memory with offset address 0100H

Sol.

(i) ADD AX, 2345 ; Add 2345H to the contents of AX register
(ii) ADD [BX], 22 ; Add 22H to the content of the specified memory location by BX register
(iii) SUB AX, BX ; Subtract the content of AX register from AX register
(iv) SBB AX, 2345; Subtract immediately 2345H from BX register with borrow
(v) SUB [0100], 1000; Subtract immediately 1000 from memory with offset address 0100H

INC Destination (Add 1 to destination)

Destination ← (Destination +1), Flag affected : O S Z A P

When this instruction is executed, the contents of the specified register or memory location increases by 1. After execution, the condition flags O, S, Z, A and P are affected but the carry flag is not affected by this instruction. In this instruction, immediate data cannot be operand. The object code of instruction is

Register/Memory

1111 111*w*	Mod 000 R/M

Register

01 000 reg

For example, INC AX. The object code is

01 000 reg

Here reg = 000 for AX register. Then object code is 0100 0000 = 40

Other examples of INC instructions are INC BX ; INC CX; INC DX and INC [BX]. The object code for INC BX is 43; for INC CX is 41; for INC DX is 42 and for INC [BX] is FF 07.

DEC Destination (Decrement destination by 1)

Destination ← (Destination—1), Flag affected: O S Z A P C

This instruction decrements the contents of the specified register or memory location by one or subtracts 1 from the contents of the specified register or memory location. After execution, all the condition flags O, S, Z, A, P and C are affected depending upon the result. But carry flag is not affected. In this instruction,

immediate data cannot be used as operand. The object code of instruction is

Register/Memory

1111 111*w*	Mod 001 R/M

Register

01 001 reg

For example, DEC AX. The object code is

01 001 reg

Here reg = 000 for AX register. Then object code is 0100 1000 = 48

Other examples of INC instructions are DEC BX ; DEC CX; DEC DX and DEC [BX]. The object code for DEC BX is 4B ; for DEC CX is 49 ; for DEC DX is 4A and for DEC [BX] is FF 0F.

NEG Destination (Changes the sign of an operand (negate))

Destination ← (0-Destination), Flag affected: O S Z A P C

This instruction performs a 2's complement of destination. To obtain a 2's complement, it subtracts the contents of destination from zero. Then result is stored in the destination operand which may be a register or a memory location. After execution this instruction all the condition flags O, S, Z, A, P and C are affected. While OF is set, it means that the operation has not completed successfully. The object code is

1111 011*w*	Mod 011 R/M

The example of these instructions are NEG AX ; NEG BX; NEG CX ; NEG DX; NEG AL; NEG BL; NEG CL and NEG DL. The object code for NEG AX is F7 D8 ; for NEG BX is F7 DB ; for NEG CX is F7 D9 ; for NEG DX is F7 DA ; for NEG AL is F6 D8 ; for NEG BL is F6 DB ; for NEG CL is F6 D9 and for NEG DL is F6 DA.

CMP Destination, Source (Compare by subtracting source from destination)

Destination — Source Flag affected : O S Z A P C

This instruction performs a nondestructive subtraction of source from destination but result is not stored. Actually, the source operand and destination operand are compared. The source operand will be a register or an immediate data or a memory location and the destination operand may be a register or a memory location. After comparison, the result will not store anywhere but the flags are affected depending upon the result of the subtraction. When both of the source and destination operands are equal, zero flag is set. While the source operand is greater than the destination operand, carry flag is set otherwise carry flag is reset. The object code is Register/Memory with Register.

0011 10*dw*	Mod reg R/M

Immediate to register/memory

1000 00*sw*	Mod 111 R/M	data	data

Immediate to Accumulator

0011 110*w*	data	data

For example, CMP AX, 0100 Compare 0100H with the content of AX register immediately. The object code is

0011 110*w*	data	data

Here w = 1, object is 0011 110w, 16-bit data = 0011 1101, 16-bit data = 3D, 00, 01H

The other examples of CMP instructions are CMP BX, 1234; CMP AL, 22; CMP BX, [SI]; CMP [0100], BX and CMP [BX], CX. The object code for CMP BX, 1234 is 81 FB 34 12; for CMP AL, 22 is 3C 22; for CMP BX, [SI] is 3B 1C; for CMP [0100], for BX is 39 1E 00 01 and for CMP [BX], CX is 39 0F.

Example 10.7 Write instructions for the following operations:

(i) Compare 16-bit immediately available data (4567H) from AX register

(ii) Increment the contents of CX register by one

(iii) Decrement the contents of memory location specified by BX register

(iv) 2's Complement of accumulator

(v) Compare 8 bit immediately available data (FFH) from contents of memory location specified by Source index address 0400

Sol.

(i) CMP AX,4567 ; Compare 16-bit immediately available data (4567H) from AX register

(ii) INC CX ; Increment the contents of CX register by one

(iii) DEC [BX] ; Decrement the contents of memory location specified by BX register

(iv) NEG AX ; 2's Complement of accumulator

(v) MOV SI,0400 ; load 0400 in SI

CMP [SI], FF ; Compare 8-bit data (FFH) with the contents of memory location specified by Source index.

10.3.5 Multiplication and Division Instructions

MUL Source (Multiply 8 or 16-bit source by 8-bit (AL) or 16-bit (AX) value (unsigned))

AX ← (AL * source 8)

DX:AX ←(AX * source 16), Flag affected : O C; the S, Z, A, and P flags are left in an indeterminate condition.

This instruction is an unsigned byte or word multiplication by the contents of AL or AX. An 8-bit source is multiplied by the contents of AL to generate a 16-bit result in AX. A 16-bit source is multiplied by the contents of AX to generate a 32-bit result. The most significant word of the result is stored in DX and the least significant word of the result is stored in AX. The unsigned byte or word will be one of the general purpose registers or memory locations. All the flags are modified depending upon the result. In this instruction immediate operand is not allowed.

The object code of MUL instruction is

1111 011w	Mod 100 R/M

Here, mod and R/M are for memory/register; i.e., for the second operand. The first operand is always in AL or AX.

The example is MUL BL. Assume the content of AL register is 22, and register of BL is 11H.

The object code = 1111 011w, mod 100 R/M = 1111 0110, 11 100011 = F6, E3 as Mod and R/M for BL are 11 and 011 respectively and w = 0.

The other examples of MUL instructions are MUL CL; MUL BX ; MUL CX ; MUL DX and MUL [BX+10]. The object code for MUL CL is F6 E1; for MUL BX is F7 E3; for MUL CX is F7 E1; for MUL DX is F7 E2 and for MUL [BX+10] is F7 67 10.

IMUL Source (Multiply an 8- or 16-bit source by 8-bit (AL) or 16-bit (AX) value (signed))

AX ← (AL * source 8)

DX:AX ← (AX * source 16), Flag affected : O C; the S, Z A and P flags are left in an indeterminate condition.

This instruction is a signed multiplication of two signed numbers. A signed byte in the source operand is multiplied by the contents of AL to generate a 16-bit result in AX. The source can be a general purpose register, memory operand, index register or base register, but it cannot be an immediate data. A 16-bit source operand is multiplied by the contents of AX to generate a 32-bit result. In case of 32-bit results, the higher order word or the higher 16 bits is stored in DX and the lower order word or the lower 16 bits is stored in AX. The AF, PF, SF and ZF flags are undefined after IMUL. If AH and DX contain parts of 16 and 32-bit result respectively, CF and OF both will be set. AL and AX are the implicit operands in case of 8 bits and 16 bits multiplications respectively. The unused higher bits of the result are filled by the sign bit and CF, AF are cleared.

The object code of IMUL is

1111 011*w*	Mod 101 R/M

Here, Mod and R/M are for the second operand which is either memory or register. The first operand is always in AL or AX. During 8-bit multiplication, 7 bits are used to represent a number and the eighth bit represents its sign. When the sign bit is 0, it represents a positive number. If the sign bit is 1, it represents a negative number. In case of 16-bit multiplication, 15 bits are used to represent a number and the sixteenth bit represents its sign.

The example of IMUL instruction is IMUL BL. Here w = 0, Mod and R/M for BL are 11, R/M = 001 respectively. Then object code = 1111 011w, mod 101 R/M = 1111 0110, 11 101 011 = F6, EB.

The other examples of IMUL instructions are IMUL CL; IMUL BH; IMUL BX ; IMUL CX ; IMUL DX and IMUL [BX+10]. The object code for IMUL CL is F6 E9; for IMUL BH is F6 EF; for IMUL BX is F7 EB; for IMUL CX is F7 E9; for IMUL DX is F7 EA and for IMUL [BX+10] is F7 2F 10.

DIV Source (Divide of 16-bit or 32-bit number by 8- or 16-bit number (unsigned))

AL ← (AX ÷ Source 8)

AH ← Remainder

AX ← (DX:AX ÷ Source 16)

DX ← Remainder

Flag affected: The O, S, Z, A, P, and C flags are left in an indeterminate condition.

This is an unsigned divide instruction. This instruction is used to divide a 16-bit unsigned number by an 8-bit unsigned number. When a 16-bit number in AX is divided by an 8-bit source operand, the quotient is stored in AL and the remainder is stored in AH. If the result is too big to fit in AL, a divide by zero (type 0) interrupt is generated.

This instruction is also used to divide a 16-bit unsigned number by a 16-bit or 8-bit operand. The dividend must be in AX for 16-bit operation and divisor may be specified using any one of the addressing modes except immediate. A 32-bit number in DX:AX is divided by a 16-bit source with the quotient remaining in AX and the remainder in DX. When the quotient of a 16-bit operation is greater than FFFFH, a divide-by-zero (type 0) interrupt is generated. This instruction does not affect any flag.

The object code DIV instruction is

1111 011*w*	Mod 110 R/M

The example of DIV instruction is DIV BL. This instruction consists of 8-bit divisor in BL and AX contains 16-bit dividend. The object code is 1111 011*w*, Mod 110 R/M =1111 0110, 11 110011 = F6, F3 as Mod = 11 and R/M = 011 for BL and *w* = 0.

The other examples of DIV instructions are DIV CL; DIV BX ; DIV CX ; DIV DX and DIV [BX+10]. The object code for DIV CL is F6 F1; for DIV BX is F7 F3; for DIV CX is F7 F1; for DIV DX is F7 F2 and for DIV [BX+10] is F7 37 10.

IDIV Source (Divide of signed 16-bit or 32-bit number by 8- or 16-bit number (signed))

AL ← (AX ÷ source 8)

AH ← Remainder

AX ← (DX:AX ÷ source 16)

DX ← Remainder

Flag affected: The O, S, Z, A, P, and C flags are left in an inderterminate condition.

This is a signed divide. This instruction performs the same operation as DIV instruction. A 16 bit value in AX is divided by an 8-bit source with the quotient remaining in AL and the remainder in AH. If the result is too big to fit in AL, a divide by zero (type 0) interrupt is generated.

A 32-bit number in DX:AX is divided by a 16-bit source with the quotient remaining in AX and the remainder in DX. Divide by 0 interrupt is generated. If the result (quotient) is too big to fit in AX, a divide by zero (type 0) interrupt is generated. All the flags are undefined after IDIV instruction.

The object code of IDIV is

1111 011*w*	Mod 111 R/M

The example of IDIV instruction is IDIV BL. The operation of this instruction is dividing AX by CL, both operands are signed numbers. The object code is 1111 011*w*, Mod 111 R/M = 1111 0110, 11 111011 = F6, FB as Mod = 11 and R/M = 011 for BL and *w* = 0.

The other examples of DIV instructions are IDIV BH; IDIV CL; IDIV BX ; IDIV CX ; IDIV DX and IDIV [BX+10]. The object code for IDIV BH is F6 FF; for IDIV CL is F6 F9; for IDIV BX is F7 FB; for IDIV CX is F7 F9; for IDIV DX is F7 FA and for IDIV [BX+10] is F7 3F 10.

Example 10.8 Write instructions for the following operations:

(i) Multiply the content of AL by the content of CL

(ii) Multiply the content of AX by the content of CX

(iii) Signed multiplication of AL and DL

(iv) Divide AX by the content of memory location represented by BX

(v) Signed division of AX and BL

Sol.

(i) MUL CL ; Multiply the content of AL by the content of CL

(ii) MUL CX ; Multiply the content of AX by the content of CX

(iii) IMUL DL ; Signed multiplication of AL and DL

(iv) DIV [BX] ; Divide AX by the content of memory location represented by BX

(v) IDIV BL ; Signed division of AX and BL

10.3.6 Arithmetic Adjust Instructions

DAA (Decimal adjustment after addition)

AL← (AL adjusted for BCD addition)

Flag affected: S Z A P C; the O flag is left in an indeterminate condition.

The DAA instruction is used to transfer the result of the addition of two packed BCD numbers to a valid BCD number. The result will be stored in AL register only. If after addition, the lower nibble is greater than 9, AF is set. Then 06 will be added to the lower nibble in AL. After addition of 06 in the lower nibble of AL, if the upper nibble of AL is greater than 9 or if the carry flag is set, 60H will be added to AL through DAA instruction. After execution of this instruction, AF, CF, PF, and ZF flags are affected. The OF is undefined.

The object code of DAA is 00100111= 27H

Example 10.9 Write instructions to add two numbers 54 and 26 and use DDA for adjustment of the result.

Sol.

```
MOV AL, 54 ; 54 in AL.
MOV BL, 26 ; 26 in BL.
ADD AL, BL ; add AL and BL, content of AL = 7A.
DAA ; adjust result in BCD. AL ← 80 = 7A + 06.
```

DAS (Decimal adjust for subtraction)

AL ← (AL adjusted for BCD subtraction), Flag affected S Z A P C; the O flag is left in an indeterminate condition.

The DAS instruction is used to convert the result of subtraction of two packed BCD numbers to a valid BCD number. The subtraction will be stored in AL register only. While the lower nibble of AL is greater than 9, 06 will be subtracted from the lower nibble of AL. If the result of subtraction sets the carry flag or if the upper nibble is greater than 9, 60H will be subtracted from AL. AF, CF, SF, PF and ZF flags are affected after execution of this instruction. The OF is undefined after DAS instruction.

The object code of DAS is 00101111= 2FH

AAA (ASCII adjust for addition)

AL ← (AL adjusted for ASCII addition)

Flag affected: A C; the O, S, Z, and P flags are left in an indeterminate state

This instruction follows an addition of 'unpacked' ASCII data. After execution of ADD instruction, this AAA instruction is executed for ASCII adjustment of result of addition of two numbers. The result will be stored in AL register. The AAA instruction converts the resulting contents of AL to unpacked decimal digits. When the AAA instruction is executed, the lower 4- bits of AL will be checked whether it is a valid BCD number in the range 0 to 9. If the lower 4- bits of AL is between 0 to 9 and AF is zero, AAA sets the higher order 4- bits of AL to 0. The content of AH must be cleared before addition. If the value in the lower 4-bits of AL is greater than 9 then the AL is incremented by 06, AH is incremented by 1, the AF and CF flags are set to 1, and the higher 4 bits of AL are cleared to 0. After the addition of 05H and 09H, the result 0E is stored in AL. As lower nibble of AL (E) is greater than 9, the AL is to be incremented by 06 and AH is incremented by 1. Hence the content of AL is 04H and the content of AH is 01H.

The object code of AAA is 00110111 = 37H.

AAS (ASCII adjust for subtraction)

AL ←(AL adjusted for ASCII subtraction)

Flag affected: A C, the O, S, Z, and P flags are left in an indeterminate state.

This instruction follows a subtraction of 'unpacked' ASCII data. The AAS instruction is used to convert the result in AL register after subtracting two unpacked ASCII operands. The result is stored in AL register which is an unpacked decimal number. When the lower 4 bits of AL register are greater than 9 or the AF flag is set to 1, the AL will be decremented by 6 and AH register is decremented by 1, the CF and AF are set to 1. If not, the CF and AF are set to 0, the result does not require any correction. Hence, the upper nibble of AL is 0 and lower nibble may be any number from 0 to 9.

The object code of AAS is 00111111 = 3FH.

AAM (ASCII adjust for multiplication)

AH : AL ←(AH : AL adjusted for ASCII multiplication), Flag affected S Z P; the O, A, and C flags are left in an indeterminate condition.

The AAM instruction is executed to converts the product available in AL into unpacked BCD format. The AMM (ASCII adjustment after multiplication) instruction follows a multiplication instruction that multiplies two unpacked BCD numbers, i.e, and higher nibbles of the multiplication number should be 0. Usually, the multiplication is performed using MUL instruction and the result of multiplication is available in AX. After execution of AAM instruction, the content of AH is replaced by tens of the decimal multiplication and the content of AL is replaced by ones of the decimal multiplication. The object code of AAM is 1101 0100 0000 1010 = D4 0A.

Example 10.10 Write instructions to multiply two unpacked BCD numbers 4 and 6 and use AAM for adjustment of the result.

Sol.

```
MOV AL, 04 ; 04 in AL.
MOV CL, 06 ; 06 in CL.
MUL CL ; multiply AL by CL, content of AL = 18.
AAM ; adjust result in BCD. AH ← 01 and AL ← 08
```

CBW (Convert from byte to word (16-bit ← 8-bit))

AH ← (filled with bit-7 of AL), AX ←(AL * source 8), Flag affected: None

This instruction converts a signed byte in AL to a signed word in AX. Actually, it copies the sign bit of a byte to be converted to all the bits in the higher byte of the result word. Flags are not affected after execution of CBW. The object code of CBW is 1001 1000 = 98H.

CWD (Convert from word to double word)

DX ← (filled with bit-15 of AX), AX ←(AL * source 8)

Flag affected: None

Converts a 16-bit word in AX to a 32-bit word in DX : AX by sign extension of bit 15 of AX through DX. Usually, this operation is to be done before signed division. Flags are not affected after execution of CWD. The object code of CWD is 1001 1001 = 99H.

10.3.7 Logical and Bit Manipulation Instructions

These type of instructions are used for

- Basic logical operations such as NOT, AND, OR, and XOR;
- Bit by bit shift operations such as SHL (shift logical left), SHR (shift logical right), SAL (shift arithmetic left), and SAR (shift arithmetic right); and

- Rotate operations such as ROR (rotate right without carry), ROL (rotate left without carry), RCR (rotate right through carry), and RCL (rotate left through carry).

After execution of the above instructions, all the condition code flags are affected depending upon the result. In this section the operations of logical and bit manipulation instructions are discussed in detail.

NOT Destination (1's complement of destination)

Destination ← (~Destination)

Flag affected: None

Converts 1's to 0's and 0's to 1's in destination.

The NOT instruction is used to generate complement of the contents of an operand register or a memory location, bit by bit.

The object code of NOT is

1111 011*w*	Mod 010 R/M

The example of NOT instruction is NOT AL. The object code of NOT AL is 1111 011*w* 1101 000 = F6 D0 as Mod = 11 and R/M = 000 for AL and *w* = 0. The other NOT instructions are NOT BL, NOT CL, NOT DL, NOT AX, NOT BX, NOT CX, and NOT [BX]. The object code for NOT BL is F6 D3; for NOT CL is F6 D1; for NOT DL is F6 D2; for NOT AX is F7 D0; for NOT BX is F7 D3; for NOT CX is F7 D1, and for NOT [BX] is F7 17.

AND Destination, Source (Logical AND)

Destination ← (Destination AND Source)

This instruction performs a bitwise logical AND of source and destination with the result remaining in the destination. The source operand may be immediate data or a register or a memory location and the destination operand may be a register or a memory location. For AND operation, at least one of the operands must be a register or a memory operand. For this instruction, both the operands will not be memory locations and immediate operands and a destination operand should not be an immediate operand.

The object codes of AND instruction are as follows:

Register/Memory with Register

0010 00*dw*	Mod reg R/M

Immediate to register/memory

1000 000*w*	Mod 100 R/M	data	data

Immediate to Accumulator

0010 010*w*	data	data

The example of AND instruction is AND AX, 045B and its object code is 0010 010*w* 5B 04 = 25 5B 04 as *w* = 1. The other examples are AND AX, BX; AND CX, DX; AND AX, [BX] and AND AX, [SI]. The object code for AND AX, BX is 21 D8; for AND CX, DX is 21 D1; for AND AX, [BX] is 23 07 and for AND AX, [SI] is 23 04.

OR Destination, Source (Logical OR)

Destination ←(Destination OR Source)

The OR instruction performs a bitwise logical OR of source and destination with result remaining in destination. The OR operation is same as described in case of AND operation. The limitations of OR instruction based on source and destination operands are also the same as in case of AND operation.

The object code of OR instruction is as follows:

Register/Memory with Register

0000 10*dw*	Mod reg R/M

Immediate to register/memory

1000 000*w*	Mod 001 R/M	data	data

Immediate to Accumulator

0000 110*w*	data	data

The example of OR instruction is OR AX, 2345 and its object code is 0000 110*w* 45 23 = 0D 45 23 as *w* =1. The other examples are OR AX, BX; OR CX, DX; OR AX, [BX] and OR AX, [SI]. The object code for OR AX, BX is 09 D8; for OR CX, DX is 09 D1; for OR AX, [BX] is 0B 07 and for OR AX, [SI] is 0B 04.

XOR Destination, Source (Exclusive logical OR)

Destination ←(Destination XOR Source)

The XOR instruction performs a bitwise logical exclusive OR of source and destination with result remaining in destination. This instruction carries out operations in a similar way to the AND and OR operation. The limitations of XOR instruction are also the same as in case of AND/OR operation.

The object code of XOR

Register/Memory with Register

0011 00*dw*	Mod reg R/M

Immediate to register/memory

1000 000*w*	Mod 110 R/M	data	data

Immediate to Accumulator

0011 010*w*	data	data

The example of XOR instruction is XOR AX, 1234 and its object code is 0011 010*w* 34 12 = 35 34 12 as *w* = 1. The other examples are XOR AX, BX; XOR CX, DX; XOR AX, [BX] and XOR AX, [SI]. The object code for XOR AX, BX is 31 D8; for XOR CX, DX is 31 D1; for XOR AX, [BX] is 33 07 and for XOR AX, [SI] is 33 04.

TEST Destination, Source (Nondestructive logical AND)

Flags ← (Destination AND Source)

The TEST instruction performs a nondestructive bitwise logical AND of source and destination, setting flags and leaving destination unchanged. The result of this ANDing operation will not be available, but the flags are affected. Generally OF, CF, SF, ZF and PF flags are affected. The source operands may be a register or a memory or immediate data and the destination operands may be a register or a memory.

The object code of TEST is given below:

Register/Memory and Register

1000 010*w*	Mod reg R/M

Immediate data and register/memory

1000 000*w*	Mod 110 R/M	data	data

Immediate data and Accumulator

1010 100*w*	data	data

The example of TEST instruction is TEST AX, 6789 and its object code = 1010 100*w* 89 67=A9 89 67 as *w* = 1. The other example of TEST instructions are TEST AX, BX; TEST CX, DX; TEST AX, [BX]; TEST AX, [DI]. The object code for TEST AX, BX is 85 C3 ; for TEST CX, DX is 85 CA ; for TEST AX, [BX] is 85 07; and for TEST AX, [DI] is 85 05.

Example 10.11 Write instructions for the following operations:

(i) 1's complement of the content of DX register
(ii) AND 1234H with the content of AX register
(iii) XOR operation between AL and DL registers
(iv) OR operation between BX and CX registers
(v) Perform TEST operation between AL and BL registers

Sol.

(i) NEG DX ; 1's complement of the content of DX register
(ii) AND AX, 1234 ; AND 1234H with the content of AX register
(iii) XOR AL,DL ; XOR operation between AL and DL registers
(iv) OR BX,CX ; OR operation between BX and CX registers
(v) TEST AL,BL ; Perform TEST operation between AL and BL registers

SHL/SAL (Shift logical/ Arithmetic left) $A_{n+1} \leftarrow A_n$, $A_{15} \leftarrow A_{14}$, $A_0 \leftarrow 0$, CF $\leftarrow A_{15}$ All flags are affected.

These instructions shift each bit in the destination operand (word or byte) to the left and insert zeros in the newly introduced least significant bits. The highest order bit shifts into the carry flag as shown in Fig. 10.17. The common format of SHL/SAL instruction is SAL Operand-1, Operand-2.

The operand-1 will be the content of register or the content of memory. The number of shifts is set by operand-2. The operand-2 will be an immediate data or content of CL register. The object code of SHL/SAL instruction is

1101 00*vw*	Mod 100 R/M

The example of SHL instructions are SHL AX, CL and SHL AX, 1. Since shifting an integer to the left one position is equivalent to the multiplication of specified operand by 2. Actually, the shift left instruction for multiplication by powers of two is as given below:

SHL AX, 1 ; Result is equivalent to AX*2
SHL AX, 2 ; Result is equivalent to AX*4
SHL AX, 3 ; Result is equivalent to AX*8
SHL AX, 8 ; Result is equivalent to AX*256

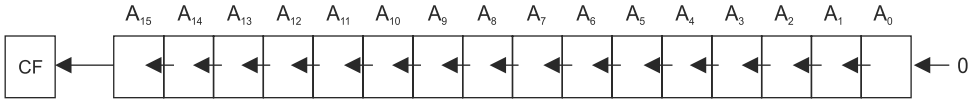


Fig. 10.17 Shift left operation

Assume the content of AX register is 1010 1010 1010 1010=AAAA. After execution of SHL AX, 1 the content of AX will be 5554 and CY flag set. The object code of SHL AX, 1 is D1 E0. After execution of SHL AX, 2 the content of AX will be AAA8. All flags are affected depending upon the result.

SHR (Shift logical right) $A_n \leftarrow A_{n+1}$, $CF \leftarrow A_0$, $A_{15} \leftarrow 0$, All flags are affected

This instruction performs bitwise right shifts on the destination operand word or byte and inserts zeros in the shifted positions. The general format of SHR instruction is SHR Operand-1, Operand-2. The operand-1 may be a register or a memory location. The number of shifts is set by the operand-2. The operand-2 will be an immediate data or content of CL register. The result is always stored in the destination operand.

Figure 10.18 shows the shift right operation.

The object code of SHR instruction is

1101 00*vw*	Mod 101 R/M

The example of SHR instructions are SHR AX, CL and SHR AX,1. If the SHR instruction shifts an integer to the right one position, it performs an unsigned division of destination operand by 2. Actually, each shift to the right is equivalent to dividing the value by 2 as given below:

SHR AX, 1 ; Result is equivalent to AX/ 2

SHR AX, 2 ; Result is equivalent to AX/ 4

SHR AX, 3 ; Result is equivalent to AX/ 8

SHR AX, 8 ; Result is equivalent to AX/256

When the content of AX register is 1010 1010 1010 1010=AAAA, after execution of SHR AX, 1 the content of AX will be 5555. The object code of SHR AX, 1 is D1 E8. After execution of MOV CL,02 and SHR AX, CL the content of AX will be 2AAA. All flags are affected depending upon the result. This shift operation shifts the operand through the carry flag.

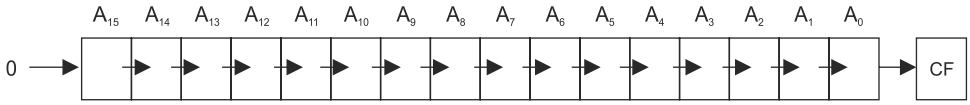


Fig. 10.18 Shift right operation

SAR (Shift arithmetic right)

$A_n \leftarrow A_{n+1}$, $CF \leftarrow A_0$, $A_{15} \leftarrow A_{15}$ All flags are affected.

The SAR instruction performs right shifts all the bits in the destination operand (word or byte) to the right one bit. This instruction is replicating the most significant bit of the operand in the newly inserted positions.

The common format of SAR instruction is SAR Operand-1, Operand-2. The Operand-1 may be a register or a memory location. The number of shifts is set by Operand-2. The Operand-2 will be an immediate data or content of CL register. The result is always stored in the destination operand. Figure 10.19 shows the arithmetic shift right operation.

The object code of SAR instruction is

1101 00*vw*	Mod 111 R/M

The example of SAR instructions are SAR AX, CL and SAR AX,1. As the SAR instruction shifts an integer to the right one position, it performs a signed division of destination operand by 2. Actually, each shift to the right divides the value by 2 as given below:

SAR AX, 1 ; Result is equivalent to signed division by 2

SAR AX, 2 ; Result is equivalent to signed division by 4

SAR AX, 3 ; Result is equivalent to signed division by 8

SAR AX, 8 ; Result is equivalent to signed division by 256

If the content of AX register is 1010 1010 1010 1010 = AAAA, after execution of SAR AX, 1 the content of AX will be D555. The object code of SAR AX,1 is D1 F8. After execution of SAR AX, 2 the content of AX will be EAAA. All flags are affected depending upon the result. This shift operation shifts the operand through the carry flag.

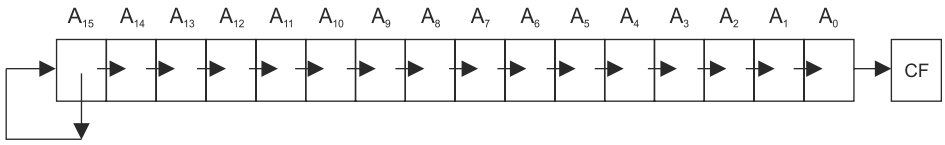


Fig. 10.19 Arithmetic shift right operation

ROR (Rotate right without carry) $A_n \leftarrow A_{n+1}$, $A_{15} \leftarrow A_0$, $CF \leftarrow A_0$. All flags are affected

The ROR instruction rotates the contents of the destination operand to the right bit-wise either by one or by the count specified in CL without carry. The least significant bit is stored into the carry flag and simultaneously it is transferred into the most significant bit position after each shift operation as shown in Fig.10.20.

The common format of ROR instruction is ROR Operand-1, Operand-2. The operand-1 may be a register except segment register or a memory location. The operand-2 will be an immediate data or content of CL register. The number of shifts is set by operand-2. The result is always stored in the destination operand.

The object code of ROR instruction is

1101 00*vw*	Mod 001 R/M

The example of ROR instructions are ROR AX, CL and ROR AX,1.

The PF, SF and ZF flags are left unchanged by this instruction.

Consider the content of AX register is 1010 1010 1111 1010 = AAFA. After execution of ROR AX, 1 the content of AX will be 557D. The object code of ROR AX,1 is D1 C8. After execution of MOV CL,02 and ROR AX, CL the content of AX will be AABE.

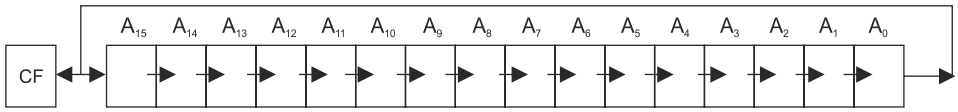


Fig. 10.20 Rotate right without carry

ROL (Rotate left without carry) $A_{n+1} \leftarrow A_n$, $A_0 \leftarrow A_{15}$, $CF \leftarrow A_{15}$. All flags are affected.

The ROL instruction rotates the content of the destination operand to the left by one or by the specified number of bits in CL without carry. The most significant bit is pushed into the carry flag as well as the least significant bit position after each bit shift operation. The other bits are shifted left subsequently as depicted in Fig.10.21. The PF, SF, and ZF flags are left unchanged by this operation. Its format is same as ROR. The object code of ROL instruction is

1101 00*vw*	Mod 000 R/M

The example of ROL instructions are ROL AX,1 and ROL AX, CL.

Assume the content of AX register is 1010 1010 1111 1010 = AAFA. After execution of ROL AX, 1 the

content of AX will be 55F5. The object code of ROL AX,1 is D1 C0. After execution of MOV CL,02 and ROL AX, CL the content of AX will be ABEA.

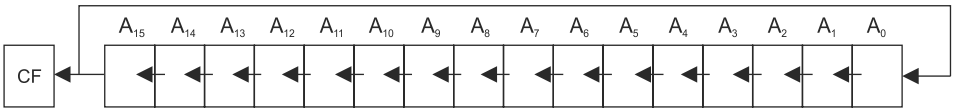


Fig. 10.21 Rotate left without carry

RCR (Rotate right through carry) $A_n \leftarrow A_{n+1}$, $A_{15} \leftarrow$ CF, CF $\leftarrow A_0$. All flags are affected

This instruction rotates the contents of the destination operand bits right by one or by the specified number of bits in CL through Carry Flag (CF). After each rotate operation, the carry flag is pushed into the MSB of the operand and the LSB is pushed into carry flag and the other bits are subsequently shifted right as given in Fig.10.22. The SF, PF, ZF are left unchanged. Its format is same as ROR. The object code of RCR instruction is

1101 00*vw*	Mod 011 R/M

The example of RCR instructions are RCR AX,1 and RCR AX, CL.

When the content of AX register is 1010 1010 1111 1010 = AAFA, after execution of RCR AX, 1 the content of AX will be 557D. The object code of RCR AX,1 is D1 D8. After execution of MOV CL,02 and RCR AX, CL the content of AX will be 2ABE.

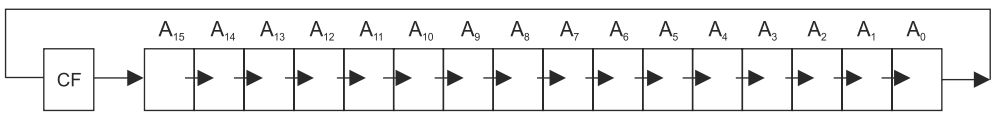


Fig. 10.22 Rotate right through carry

RCL (Rotate left through carry) $A_{n+1} \leftarrow A_n$, CF $\leftarrow A_{15}$, $A_0 \leftarrow$ CF. All flags are affected.

The RCL instruction rotates the contents of the destination operand left by one or by the specified number of bits in CL through carry flag (CF). After each rotate operation, the carry flag is pushed into LSB and the MSB of the operand is pushed into carry flag. The remaining bits are subsequently shifted left as shown in Fig.10.23. The SF, PF, ZF are left unchanged. The object code of RCL instruction is

1101 00*vw*	Mod 010 R/M

The example of RCL instructions are RCL AX,1 and RCL AX, CL.

When the content of AX register is 1010 1010 1111 1010 = AAFA, after execution of RCL AX, 1 the content of AX will be 55F4. The object code of RCL AX,1 is D1 D0. After execution of MOV CL,02 and RCL AX, CL the content of AX will be ABE9.

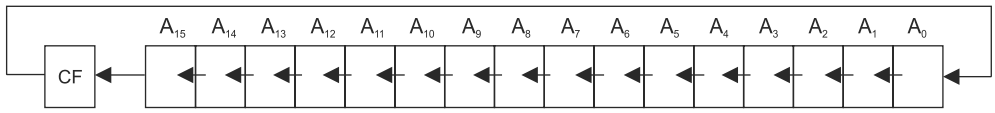


Fig. 10.23 Rotate left through carry

Example 10.12 Write instructions for the following operations:

(i) Shift content of BL left two times

(ii) Rotate content of BX left without carry one times
(iii) Shift logical right of the memory location represented by CS : SI one time
(iv) Rotate right without carry of AX registers three times

Sol.

(i) MOV CL, 02
SHL BL, CL ; Shift content of BL left two times
(ii) ROL BX, 1 ; Rotate content of BX left without carry one times
(iii) SHR [SI], 1 ; Shift logical right of the memory location represented by CS:SI one time
(iv) MOV CL, 03
ROR AX, CL ; Rotate right without carry of AX registers three times

10.3.8 Jump Instructions

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

JMP target (Unconditional jump to target).

The jump instruction unconditionally transfers the control of execution to the specified address using an 8-bit or 16-bit displacement or CS: IP. After execution of this instruction, no flags are affected. The Jump instructions have different formats to specify the jump address:

Sort: IP ← (IP+(target displacement sign-extended))

Near: IP ← (IP+(target displacement))

Indirect: IP ← (register or value in memory)

Far: CS ← targ_seg;

IP ← targ_offset AX ←(AL * source 8)

Flag affected: None

Short jumps are within ±128 bytes of jmp instruction —only IP is affected.

Near jumps are within same segment —only IP is affected. Near jump allows a jump within ±32 KB

Indirect jumps are within same segment —only IP is affected.

Far jumps are to a different segment —both CS and IP are affected.

The object code of unconditional JMP instruction is

Direct with segment

1110 1001	disp-low	disp-high

Direct within segment short

1110 1011	disp

Indirect within segment

1111 1111	Mod 100 R/M

Direct Intersegment

1110 1010	Offset-low	Offset-high

Seg-low	Seg-high

Indirect Intersegment

1111 1111	Mod 101 R/M

JCXZ Target(short) (Jump short if CX register is 0)

Flag affected: None

The object codes of JCXZ jump on CX zero instructions are

1110 0011	disp

Jcond (Jump on condition)

IP ← (IP+(8-bit displacement sign-extended to 16 bits)), Flag affected None

If conditional jump instructions are executed, program control can be transferred to the address specified by the instruction itself. If the condition are not satisfied, instructions are executed sequentially. Here, condition is the status of flag. After execution of these instructions, no flags are affected. The address will be specified in the instruction which will be varied from –80H (–128) bytes to 7FH(127) bytes. Therefore only short jumps can be implemented using conditional branch instructions. The conditions of Jump instruction are given in Table 10.5.

Table 10.5 Conditional JUMP instructions

Instruction	Condition	Operation
JO	O = 1	Jump on overflow set
JNO	O = 0	Jump on overflow clear
JB / JNAE	C = 1	Jump if below/Jump if not above or equal
JAE / JNB	C = 0	Jump if above or equal/Jump if not below
JE / JNZ	Z = 1	Jump if equal/Jump if not zero
JNE / JNZ	Z = 0	Jump if not equal/Jump if not zero
JBE / JNA	C = 1 or Z = 1	Jump if below or equal/Jump if not above
JA / JNBE	C = 0 and Z = 0	Jump if above/Jump if not below or equal
JS	S = 1	Jump on sign set
JNS	S = 0	Jump on sign clear
JP / JPE	P = 1	Jump on parity bit set (parity even)
JNP / JPO	P = 0	Jump on parity bit clear (parity odd)
JL / JNGE	S = 1 or O = 1	Jump if less/Jump if not greater than or equal to
JGE / JNL	S = O	Jump if greater than or equal to/Jump if not less
JLE / JNG	Z = 1 or S and O = 1	Jump if less than or equal to/Jump if not greater than
JG / JNLE	Z = 0 or S = O	Jump if greater than/Jump if not less than or equal to

The object codes of all conditional jump instructions are

JE/JZ Jump on equal/zero

0111 0100	disp

JL/JNGE Jump on less/not greater or equal

0111 1100	disp

JLE/JNZ Jump on less or equal/not greater

0111 1110	disp

JB/JNAE Jump on Below/not above or equal

0111 0010	disp

JBE/JNA Jump on below or equal/not above

0111 0110	disp

JP/JPE Jump on parity/parity even

0111 1010	disp

JO Jump on overflow

0111 0000	disp

JS Jump on sign

0111 1000	disp

JE/JZ Jump on equal/zero

0111 0100	disp

10.3.9 Loop Instructions

The LOOP instruction executes the part of the program from the level or address specified in the instruction up to the loop instruction, CX number of times. After each iteration, CX is decremented automatically. If the content of CX is not zero, the LOOP instruction transfers control to starting address of the LOOP for execution. If CX is zero, the execution of LOOP instruction is completed and then next instruction of the program will be executed. The LOOP instruction can be explained with an example as follows:

```
        LEA   SI, 0100     ;  Load SI with source address of data
        LEA   DI, 0200     ;  Load DI with destination address of data
        MOV  CX, 0009      ;  Number of bytes 9 is loaded in CX register
START   LODSB              ;  Data byte to AL and increment SI by 1.
        STOSB              ;  The content of AL is stored in destination address represented by DI increment
                              DI by 1.
        LOOP START         ;  repeat until CX = 0
```

The above example shows how a string of bytes can be shifted from one memory block specified by SI to other memory block specified by DI using LOOP instructions. The LODSB instruction is equivalent to

```
MOV AL, [SI]
INC SI
```

the STOSB instruction is equivalent to

```
MOV [DI], AL
INC DI
```

and the LOOP instruction is equivalent to

DEC CX
JNZ START

In this case LODSB and STOSB instructions are executed 9 times and a block of 9 byte data will be copied from source memory to destination memory sequentially.

LOOP Target (short) (Loop to short target)

CX ← (CX-1); Jump if CX != 0

The CX register is decremented by 1. If CX now is not equal to 0, loop back to target. Otherwise, continue. The object codes of LOOP is

1110 0010	DISP

LOOPE/Z Target (short) (Loop to short target if Z bit set)

CX ← (CX-1); jump if CX! = 0 and ZF = 1

The CX register is decremented by 1. If CX is not equal to 0 or if the Z bit is set, loop back to short target.

The object codes of all conditional jump instructions are

1110 0001	DISP

LOOPNE/NZ (Loop to short target if Z bit is clear)

CX ← (CX-1); jump if CX! = 0 and ZF = 0

The CX register is decremented by 1. If CX is not equal to 0 or if the Z bit is clear, loop back to short target.

The object codes of all conditional jump instructions are

1110 0000	DISP

10.3.10 CALL and RETURN Instructions

The CALL and RET (return) instructions are used to call a subroutine or a procedure that can be executed several times from a main program. The starting address of the subroutine or procedure can be specified directly or indirectly depending upon the addressing mode. There are two types of procedures namely intrasegment and intersegment. The subroutine within segment is known as intrasegment subroutine or NEAR CALL. The subroutine from one segment to another segment is known as intersegment subroutine or FAR CALL. These instructions are unconditional branch instructions. After execution of these instructions, the incremented IP and CS are stored onto the stack and loads the CS and IP registers with the segment and offset addresses of the procedure to be called. For NEAR CALL, only IP register is stored on stack. But for FAR CALL, both IP and CS are stored onto the stack. Hence the NEAR and FAR CALLs can be discriminated using opcode.

CALL target (Call a procedure) NEAR CALL: PUSH IP, JMP to target

SP ← IP, SP ← SP-2, IP ← IP + DISP

FAR CALL: PUSH CS, PUSH IP, JMP to target Flag affected None

SP ← CS, SP ← SP-2, SP ← IP, SP ← SP-2, IP← 16 bit DATA

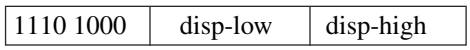
CS ← 16 bit DATA

The syntax for a near call (same segment) is CALL target.

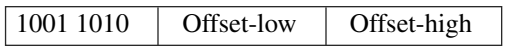
The syntax for a far call (different segment) is CALL FAR target.

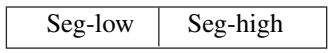
The object code of call instruction is

Direct within segment

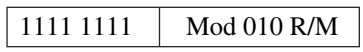


Direct Intersegment

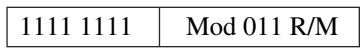




Indirect within segment



Indirect Intersegment



RET (Return from procedure)

RET n (return from procedure and add *n* to SP)

Near return: POP IP

IP ← SP, SP ← SP+2 The syntax for a near return is RET.

Far return: POP IP, POP CS Flag affected None

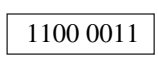
IP ← SP, SP ← SP+2, CS ← SP, SP ← SP + DISP The syntax for a far return is RET FAR.

During execution of CALL instruction, initially the IP and CS of the next instruction is pushed onto the stack, then the control is transferred to the procedure. At the end of execution of procedure, RET instruction must be executed. When RET instruction is executed, the previously stored content of IP and CS along with flags are retrieved into CS, IP and flag registers from the stack respectively. After that the execution of the main program again starts. Usually, the procedures are two types, namely, a near procedure and a far procedure. In case of a NEAR procedure, the current contents of SP points to IP but for a FAR procedure, the current contents of SP points to IP and CS at the time of return. Actually the RET instructions are four types such as

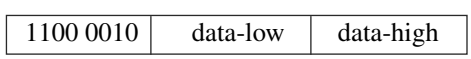
- Return within segment
- Return within segement adding 16 bit immediate displacement to the contents of SP.
- Return intersegment
- Return intersegemnt adding 16-bit immediate displacement to the contents of SP.

The object code of RET instruction is

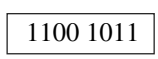
Within segment



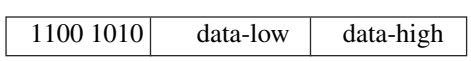
Within Segment Adding Immediate to SP



Intersegment



Intersegment Adding Immediate to SP



The RET *n* form adds *n* to the SP to compensate for stack growth when arguments are pushed onto the stack prior to a procedure call.

INT n (Interrupt (software)) PUSHF; IF ← 0; TF ← 0; PUSH CS; PUSH IP

IP ← 0000:[type * 4];

CS ← 0000:[(type * 4) + 2], Flag affected: I T

INT *n* is a software interrupt to be serviced. The flags and current CS : IP are pushed onto the stack. The CS:IP stored in the vector indicated by the interrupt number are then loaded and the next instruction is fetched from that interrupt service routine address. There are 256 interrupts corresponding to the types from 00H to FFH in the interrupt structure of 8086. When an INT *n* instruction is executed, the TYPE byte *N* is multiplied by 4 and the contents of IP and CS of the interrupt service routine will be taken from the hexadecimal multiplication ($N \times 4$) as offset address and 0000 as segment address.

INTO (Interrupt on overflow) If OF = 1, then perform INT through vector 4

Flag affected: None

Interrupts the system if the overflow bit is set following a mathematical instruction. This indicates a carry from a signed value.

This instruction is executed, when the overflow flag OF is set. The new contents of IP and CS are taken from the address 0000: 0010 as explained in INT type instruction. This is equivalent to a type 4 interrupt instruction.

The object code of INTO instruction is

1100 1110

IRET return from interrupt service routine

POP IP; POP CS; POPF AX ←(AL * src8), Flag affected All

When an interrupt service routine is to be called, before transferring control to it, the IP, CS and flag register are stored onto the stack to indicate the location from where the execution is to be continued after the ISR is executed. This instruction appears at the bottom of all interrupt service routines (ISR).

When IRET is executed, the values of IP, CS and flags are retrieved from the stack to continue the execution of the main program.

The object code of IRET instruction is

1100 1111

10.3.11 String Instructions

Usually, a series of data bytes is known as a *string of bytes* and a series of data words is known as a *string of words*. For moving a string of bytes or words, 8086/8088 processors have five instructions such as STOS (store string byte or word), LODS (load string byte or word), MOVS (move string byte or word), SCAS (scan string byte or word) and CMPS (compare string byte or word). For these instructions, source of string byte or word is DS:SI and destination of string byte or word is ES:SI. After execution of these instructions, the offset memory pointer SI and DI are incremented or decremented by one or two depending upon direction flag. In this section, STOS, LODS, MOVS, SCAS and CMPS are explained.

MOVSB (Move string byte) ES : [DI] ← DS : [SI]; DI = DI ± 1; SI = SI ± 1, Flag affected None.

Moves a string a byte at a time from source memory DS:SI to destination memory ES:DI. SI and DI are incremented or decremented by 1, depending on Direction Flag (DF). The object code of MOVSB is 1010 010*w* = A4 as *w* = 0.

MOVSW (Move string word) ES : [DI] ← DS : [SI]; DI = DI ± 2; SI = SI ± 2, Flag affected None.

Moves a string a word at a time from source memory DS:SI to destination memory ES:DI. SI and DI are incremented or decremented by 2, depending on Direction Flag (DF). The object code of MOVSW is 1010 010w =A5 as w = 1.

STOSB (Store string byte) ES : [DI] ← AL; DI = DI ± 1, Flag affected None.

Moves a string one byte at a time from AL to destination memory address ES:DI. DI is then incremented or decremented by 1, depending on Direction Flag (DF). The object code of STOSB is 1010 101w =AA as w = 0.

STOSW (Store string word) ES : [DI] ← AX; DI = DI ± 2, Flag affected None.

Moves a string one word at a time from AX to destination memory address ES:DI. DI is then incremented or decremented by 2, depending on Direction Flag (DF). The object code of STOSW is 1010 101w = AB as w = 1.

LODSB (Load string byte) AL ← DS : [SI]; SI = SI ± 1, Flag affected: None.

Moves a string one byte at a time from source memory address DS:SI to AL. Then SI is incremented or decremented by 1, depending on Direction Flag (DF). The object code of LODSB is 1010 110w = AC as w = 0.

LODSW (Load string word) AX ← DS : [SI]; SI = SI ± 2, Flag affected: None.

Moves a string one word at a time from source memory address DS:SI to AX. Then SI is incremented or decremented by 2, depending on Direction Flag (DF). The object code of LODSW is 1010 110w = AD as w = 1.

CMPSB (Compare string byte) Flags←(result of CMP DS : [SI], ES : [DI]); DI = DI ± 1; SI = SI ± 1, Flag affected as CMP.

The byte or 8-bit data at DS:SI is compared with the byte or 8-bit data at ES:DI and the flags are set accordingly. Both SI and DI are incremented or decremented by 1, depending on the Direction Flag (DF). This instruction is combined with an REP prefix, so that we can compare two strings and we can also find at what point two strings no longer are equal. The object code of CMPSB is 1010 011w = A6 as w = 0.

CMPSW (Compare string word) Flags ← (result of CMP DS : [SI], ES : [DI]) DI = DI ± 2; SI = SI ± 2, Flag affected as CMP.

The word or 16-bit data at DS:SI is compared with the word or 16-bit data at ES:DI and the flags are set accordingly. Both SI and DI are incremented or decremented by 2, depending on the Direction Flag (DF). This instruction is combined with an REP prefix, so that we can compare two strings and we can also locate at what point two strings no longer are equal. The object code of CMPSW is 1010 011w = A7 as w = 1.

SCASB (Scan string byte) Flags ← (result of CMP ES:[DI], AL); DI = DI ± 1, Flag affected same as CMP instruction.

The byte or 8-bit data at ES:DI is compared to the contents of AL and correspondingly flags are set. DI is incremented or decremented by 1 depending upon the Direction Flag (DF). The SCASB can be combined with a REP prefix, so that we can be able to scan a string looking for the first occurrence of a particular byte. The object code of SCASB is 1010 111w = AE as w = 0.

SCASW (Scan string word) Flags ← (result of CMP ES:[DI], AX); DI = DI ± 2, Flag affected same as CMP instruction.

The word or 16-bit data at ES:DI is compared to the contents of AX and correspondingly flags are set. DI is incremented or decremented by 2 depending upon the Direction Flag (DF). The SCASW can be combined with a REP prefix, so that we can be able to scan a string looking for the first occurrence of a particular word. The object code of SCASW is 1010 111w = AF as w = 1.

REPEAT Instructions The string instructions are used to operate on large blocks of data. To refer a string, two parameters are required such as (i) starting/end address of the string, and (ii) length of the string. Usually, starting/end address of the string is represented by DS:SI and the length of a string is stored as count in the CX register. After each iteration, the incrementing or decrementing of the pointer (SI or DI) depends upon the direction flag (DF) and the counter is decremented by one. To perform the string instructions repeatedly, REP (repeat) instructions are used. Hence the string instruction with the REP prefix is executed repeatedly until the CX register becomes zero. If CX becomes zero, the execution proceeds to the next instruction in sequence. The most commonly used REP instructions are REP (repeat), REPE (repeat while equal), REPZ (repeat while zero), REPNE (repeat while not equal), and REPNZ (repeat while not zero) which are explained below.

REP / REPE / REPZ (Repeat string instruction (prefix)) CX ← (CX-1); until CX=0, Flag affected Z

This is a prefix byte that forces a string operation to be repeated as long as CX is not equal to 0. CX is decremented once for each repetition. The object code of REPZ/REPE instruction is 1111 001Z = F3 as Z = 1.

REPNE / REPNZ (Repeat string instruction while not zero (prefix)) ZF← 0; CX ← (CX-1); String Operation repeats while (CX! = 0 and ZF! = 0), Flag affected Z.

This is a prefix byte that keeps a string operation repeating while CX is not zero and Z!=0. The object code of REPNZ/REPNE instruction is 1111 001Z = F2 as Z = 0.

Example 10.13 Write instruction to move a string of 9 bytes from source address DS:SI to destination address ES:DI. Assume DS = 4000H, ES = 6000H, SI = 0100H, DI = 0200H.

Sol.

```
MOV AX,4000  ; Load 4000H in AX register
MOV DS,AX    ; Load data segment address 4000H
MOV AX,6000  ; Load 6000H in AX register
MOV ES,AX    ; Load extra segment address 6000H
MOV CX,0009  ; Store  number of  data  in CX register
MOV SI,0100  ; SI register is loaded with 0100H
MOV DI,0200  ; DI register is loaded with 0200H
CLD          ; Clear direction flag DF
REP MOVSB    ; move a string of 9 bytes from  source address DS:SI to destination address ES:DI.
```

10.3.12 Processor Control Instructions

These instructions control the operation of processor and set or clear the status indicators. These instructions are classified into two types such as flag manipulation instructions and machine control instructions. The flag manipulation instructions directly change some flags of 8086 processor but the machine control instructions control the system bus functions.

The Carry (CF), Direction (DF) and Interrupt (IF) flags can be set or reset directly and the carry flag can be inverted by these instructions. The DF and IF are processor control bits. DF is used with the string instructions to change the content of pointer registers. When DF = 0, pointer register (DI) is incremented. When DF = 1, pointer register (DI) is decremented. The STD (set direction flag) and CLD (clear direction flag) instructions are used to set or clear this flag. The STI (Set Interrupt Flag) and CLI (clear interrupt flag) are used to enable or disable maskable interrupts on INTR line. When TF (trap flag) is set, a type 1 interrupt is generated after execution of each processor instructions. There are no specific instructions to set or reset the

TF. POPF and SAHF instructions, which are termed as data transfer instructions are used to modify flags. The machine control instructions are HLT, WAIT, NOP ESC and LOCK. Some instructions are specially used for coprocessors. There are three coprocessors instructions as WAIT, LOCK and ESC. In this section all processor control instructions are explained.

CLC (Clear the carry flag) CF ← 0, Flag affected C. The CLC instruction is used the carry flag low. The object code of CLC instruction is 1111 1000 = F8

CMC (Complement the carry flag) CF ← ~CF, Flag affected C.

The CMC instruction is used to complement the carry flag. The object code of CMC instruction is 1111 0101 = F5.

STC (Set the carry flag) CF ← 1, Flag affected C.

The CMC instruction is used to set the carry flag. The object code of STC instruction is 1111 1001 = F9

CLD (Clear direction flag) DF ← 0, Flag affected: D.

Clear direction flag to 0. When DF = 0, pointer register (SI or DI) is automatically incremented by 1.

The object code of CLD instruction is 1111 1101 = FC.

STD (Set direction flag) DF ← 1, Flag affected: D.

Sets direction flag to 1. When DF = 1, pointer register (SI or DI) is automatically decremented by 1.

The object code of STD instruction is 1111 1101 = FD.

CLI (Clear interrupt flag) IF ← 0, Flag affected: IF.

Clears the interrupt enable flag which disables interrupts.

The object code of CLI instruction is 1111 1010 = FA.

STI (Set interrupt flag) IF ← 1, Flag affected: IF.

Sets the interrupt enable flag which enables interrupts. The object code of STI instruction is 1111 1011 = FB.

HLT (Halt) Flag affected: None.

Halt instruction is used to ask the processor to stop execution. Actually hangs the processor in a series of self-inflicted NOP's until an interrupt occurs. The object code of HLT instruction is 1111 0100 = F4.

WAIT (Wait) Flag affected: None.

Causes the processor to wait for completion signal from coprocessor. The object code of WAIT instruction is 1001 1011= 9B.

LOCK (Lock bus) Flag affected: None.

This instruction is used to avoid any other processors. Actually, locks the bus attached to LOCK pin of device while a multi-cycle instruction completes. The object code of LOCK instruction is 1111 0000 = F0.

NOP (No operation) When NOP instruction is executed, this instruction does not allow the processor to perform any operation except for incrementing the IP by one. The object code of NOP instruction is 1001 0000 = 90.

TEST The TEST input is examined by a WAIT instruction.

When the WAIT instruction is executed, it holds the operation of processor with the current status till the logic level on the TEST pin is low. Therefore, the processor remains in idle state and the TEST pin goes low.

ESC (Escape) The ESC instruction is used as a prefix to the coprocessor instructions. The 8086 processor put the source operand on the data bus but no operation further takes place. The coprocessor continuously examined the data bus content and it is activated by ESC instruction and it reads two operands and thereafter starts execution. The detailed operation is illustrated in the chapter on Coprocessors.

10.4 8086 INSTRUCTION SET SUMMARY

The 8086 data transfer instructions set summary are shown in Table 10.6a and 10.6b respectively. Table 10.7 shows the 8086 PUSH and POP Instructions set summary. The 8086 Arithmetic instructions set summary is depicted in Table 10.8 and the 8086 multiplication and division instructions set summary are illustrated in Table 10.9. Table 10.10 shows Arithmetic Adjust instruction set summary and the 8086 logical instruction set summary is given in Table 10.11. The 8086 shift and rotate instructions set summary is depicted in Table 10.12 and the 8086 Jump instructions set summary are illustrated in Table 10.13, Table 10.14 shows the 8086 Loop Instruction Set Summary and the 8086 CALL and RETURN instruction set summary is given in Table 10.15. The 8086 string instructions set summary is depicted in Table 10.16 and Table 10.17 shows the 8086 string instruction set summary. The 8086 string instruction set summary is illustrated in Table 10.18 and the 8086 processor control instructions set summary is depicted in Table 10.19.

Table 10.6a 8085 data-transfer instructions set summary

Opcode	Operand	Mnemonics	Symbolic Operation	Comments
MOV	destination, source	MOV AX, BX	AX← BX	Register to register
		MOV AL, BL	AL← BL	Register to register
		MOV AX, MEMW	AL← [0100H]; AH← [0101H]	Memory to register
		MOV AL, MEMB	AL←[0100H]	Memory to register
		MOV MEMW, AX	[0100H] ←AL; [0101H] ←AH	Register to memory
		MOV MEMB, BL	[0100H] ←BL	Register to memory
		MOV MEMW, 2244H	[0100H] ← 44H; [0101H] ← 22H	Immediate data to memory
		MOV MEMB, 44H	[0100H] ← 44H	Immediate data to memory
		MOV AL, 22H	AL← 22H	Immediate data to register
		MOV AX, 2000H	AL← 00H; AH←20H	Immediate data to register
		MOV DS, AX	DS ← AX	General register to segment register
		MOV DX, ES	DX← ES	Segment register to general register
		MOV ES, MEMW	ES ← [0101H : 0100H]	Memory to segment register
		MOV MEMW, CS	[0101H: 0100H] ← CS	Segment register to memory
XCHG	destination, source	XCHG AX, BX	AX↔ BX	Exchange the contents of the word or byte source operand with the destination operand; none of the flags are affected.
		XCHG AL, BL	AL ↔ BL	
		XCHG [SI], BX	[SI] ↔ BL; [SI+1] ↔ BH	
LAHF		LAHF	AH ← $Flags_L$	Copy the low order flag byte into AH

SAHF		SAHF	$Flags_L \leftarrow AH$	Copy AH into the low order flag byte
IN	Accumulator, port	IN AL, 01H IN AX, 02H	AL← Port 01H AL ← Port 02H; AH← 03H	Input a byte or word from direct I/O ports 00H to FFH.
		IN AL, DX IN AX, DX	AL ← Port DX AL ← Port DX; AH← Port DX + 1	Input a byte or word from indirect I/O ports 0000H to FFFFH; the port address is in DX; None of the flags are affected
OUT	Port, accumulator	OUT 01H, AL OUT 02H, AX	Port 01H ← AL Port 02H ← AL; Port 03H ← AH	Output a byte or word to direct I/O ports 00H to FFH
		OUT DX, AL OUT DX, AX	Port DX ← AL Port DX ← AL; Port DX+1 ← AH	Output a byte or word to indirect I/O ports 0000H to FFFFH; the port address is in DX; The flags are not affected.

Table 10.6b 8086 data transfer instruction set summary

Opcode	Operand	Mnemonics	Symbolic Operation	Comments
LEA	destination, source	LEA DX, MEMB	BL ← 00; BH ← 01H	The effective address of the source operand is transferred to the destination operand; the flags are not affected
LDS	destination, source	LDS BX, DWORD PTR[SI]	BL ←[SI]; BH[SI+1]; DS← [SI+3:SI+2]	Transferred 32-bit pointer variable from
LES	destination, source	LES BX, DWORD PTR[SI]	BL ← [SI]; BH← [SI+1]; ES ← [SI+3:SI+2]	the source operand in memory to the destination register and register DS or ES; none of the flags are affected
XLAT		XLAT	AL ← [BX+AL]	Replace the byte in Al with the byte from the 256 byte lookup table beginning at [BX]; AL is used as an offset into this table; The flags are not affected

MEMB = 0100 is used to locate a byte in data segment, MEMW= 0100 is used to locate a word in data segment

Table 10.7 8086 PUSH and POP instructions set summary

Opcode	Operand	Mnemonics	Symbolic Operation	Comments
PUSH	Source	PUSH BX	SP ← SP – 2; [SP + 1] ← BH; [SP] ← BL	Decrement SP by 2 and transfer the word from the source ope-rand to

				the top of the
		PUSH DS	SP ← SP – 2; [SP + 1: SP] ← DS;	stack pointed by SP and SS
		PUSH [DI + 5]	SP ← SP – 2; [SP + 1] ← [DI + 6]; [SP] ← [DI + 5]	
POP	Destination	POP BX	BL← [SP]; BH ← [SP + 1]; SP ← SP + 2	Increment SP by 2 and transfer the word from the
		POP DS	DS ← [SP + 1: SP]; SP ← SP + 2	top of the stack pointed by SP and SS to the destina-
		POP [DI + 5]	[DI + 6] ← [SP + 1]; [DI + 5] ← [SP]; SP← SP + 2	tion operand
PUSHF	None	PUSHF	SP ← SP – 2; [SP + 1: SP] ← Flags;	Push the 16-bit flag word onto the top of stack
POPF	None	POPF	Flags ← [SP + 1: SP]; SP← SP + 2	Pop the top of the stack into the 16-bit flag word

Table 10.8 8086 arithmetic instructions set summary

Opcode	Operand	Mnemonics	Symbolic Operation	Comments

ADD	destination, source	ADD SI, AX
		ADD [BX], CL
		ADD DI, 4000H
		ADD MEMW, 4000H
			SI← SI+AX
			[BX] ←[BX]+CH
			DI← DI+4000H
			[0101H:0100H] ←
			[0101H:0100H]+ 4000H
				Substitute the destination byte or word with the sum of the source and destination operands; all flags are updated

ADC	destination, source	ADC SI, AX ADC [BX], CL ADC DI, 4000H ADC MEMW, 4000H	SI←SI +AX+ CF BX] ←[BX]+CL+ CF DI← DI+4000H+ CF [0101H:0100H]← [0101H:0100H]+4000H +CF	Replace the destination byte or word with the sum of the source and destination operands plus the carry; all flags are updated

SUB	destination, source	SUB SI, AX
		SUB [BX], CL
		SUB DI, 4000H
		SUB MEMW, 8000H
			SI← SI-AX
			[BX] ← [BX]-CL
			DI← DI-4000H
			[0101H:0100H] ←
			[0101H:0100H]-4000H
				Substitute the destination byte or word with the difference between of destination operands and source operand; all flags are updated

SBB	destination, source	SBB SI, AX SBB [BX], CL SBB DI, 4000H SBB MEMW, 8000H	SI← SI-AX-CF [BX] ←[BX]-CL-CF DI← DI-4000H-CF [0101H:0100H] ← [0101H:0100H]-4000H-CF	Replace the destination byte or word with the difference between of destination operands and source operand plus the carry; all flags are updated
INC	destination	INC CL INC WORD [DI] INC MEMBS	CL← CL +1 [DI+ 1:DI] ← [DI+1:DI]+1 [0100H] ← [0100H]+1	Increment by one or Add one the byte or word des tination operand; store the result in the destination operand; all flags except CF are updated.

DEC	destination	DEC CL DEC WORD [DI] DEC MEMB	CL← CL -1 [DI+ 1:DI] ← [DI+1:DI]-1 [0100H] ← [0100H]-1	Subtract one from byte or word destination operand; store the result in the destination operand; all flags except CF are updated.
NEG	destination	NEG AL NEG WORD [DI] NEG MEMB	AL← 0 - AL [DI+ 1:DI] ← 0 -[DI+1:DI] [0100H] ← 0- [0100H]	Find the 2's complement of the byte or word destination operand; all flags except CF are updated.

CMP	destination	CMP AL, BL
		CMP [DI],BX
		CMP MEMW, 4000H
		
		CMP DI,4000H
			AL-BL; update flags
			[DI+1:DI]-BX; update flags
			[0101H:0100H]-4000H; update flags
			DI-4000H; update flags
				Subtract the byte or word source operand from the similar destination operand; the operands remain unchanged; all flags are updated.

Table 10.9 8086 multiplication and division instructions set summary

Opcode	Operand	Mnemonics	Symbolic Operation	Comments
MUL	source	MUL BL	AX← AL × BL (Unsigned multiplication)	Unsigned multiplication of the source operand byte or word and the accumulator; results are stored in AX; Double word results are stored in DX: AX, if the result cannot be stored in a single word CF and OF are set; all other flag are undefined
		MUL BX	DX: AX ← AX × BX (Unsigned multiplication)	
		MUL [BX]	AX ← AL × [BX] (Unsigned multiplication)	
		MUL MEMW	DX:AX ← AX × [0101H:0100H] (Unsigned multiplication)	
IMUL	source	IMUL BL	AX← AL × BL (signed multiplication)	Its operation is same as MUL. The source operand is limited to –128 to +127 for byte multiplication and –32768 to +32767 for word multiplication. The CF and OF are set if the result cannot be represented in the low order register; then the sign bit is extended to the high order register and the other flags are undefined
		IMUL BX	DX: AX ← AX × BX (signed multiplication)	
		IMUL [BX]	AX ← AL × [BX] (signed multiplication)	
		IMUL MEMW	DX:AX ← AX × [0101H:0100H] (signed multiplication)	
DIV	source	DIV BL	AX← AL / BL (Unsigned division)	Unsigned division of the accumulator and the source operand byte or word; the result is stored in AL and the remainder is stored in AH; for word divisors the result is stored
		DIV BX	DX: AX ← AX / BX (Unsigned division)	
		DIV [BX]	AX ← AL / [BX] (Unsigned division)	
		DIVL MEMW	DX:AX ← AX / [0101H:0100H]	

			(Unsigned division)	in AX with remainder in DX; when the quotient exceeds the capacity of its destination register (AL or AX), a type 0 interrupt is generated; and all flags are not affected.
IDIV	source	DIV BL	AX← AL / BL (signed division)	Its operation is same as DIV; the source operand is limited to –128 to + 127 for byte division and –32768 to +32767 for word division
		DIV BX	DX: AX ← AX / BX (signed division)	
		DIV [BX]	AX ← AL / [BX] (signed division)	
		DIVL MEMW	DX:AX ← AX / [0101H:0100H] (signed division)	

Table 10.10 8086 arithmetic adjust instructions set summary

Opcode	Operand	Mnemonics	Symbolic Operation	Comments
DAA	none	DAA	If AL.0F >09 or AF = 1, then AL←AL + 6; AF← 1 If AL.F0 > 90 or CF= 1, then AL← AL + 60H; CF← 1	Adjust the content of AL to a pair of valid packed decimal digits though the addition of two valid packed or unpacked decimal operands; all flags except of are affected
DAS	none	DAS	If AL . 0F > 9 or AF = 1, then AL←AL - 6; AF← 1 If AL. F0 > 90 or CF= 1, then AL← AL - 60H; CF← 1	Adjust the content of AL to a pair of valid packed decimal digits after the subtraction of two valid packed or unpacked decimal operands; all flags except OF are affected
AAA	none	AAA	If AL. 0F > 9 or AF = 1, then AL←AL + 6; AF← AH + 1 AF← 1; CF← AF; AL← AL. 0F	Adjust the content of AL to a single unpacked decimal number following the addition of two valid unpacked decimal operands. The high order half-byte of AL is zeroed and AH is incremented by 1; all flags except AF and CF are not affected
AAS	none	AAS	If AL . 0F > 9 or AF = 1, then AL←AL - 6; AF← AH -1 AF← 1; CF← AF; AL← AL . 0F	Adjust the content of AL to a single unpacked decimal number following the subtraction of two valid unpacked decimal operands. The high order

				half-byte of AL is zeroed and AH is decremented by 1; all flags except AF and CF are not affected
AAM	None	AAM	AH ← AL/0AH AL← Remainder	After the multiplication of two valid unpacked decimal operands, AAM converts the result in AL to two valid unpacked decimal digits in AH and AL. PF, SF, and ZF are affected
AAD	None	AAD	AL ← (AH × 0AH) + AL AL← 0	Before dividing AX by a single-digit unpacked, decimal operand, AAD converts the two-digit unpacked decimal number in AX to a binary number in AL and 0 in AH. The quotient will be a valid unpacked decimal number in AL and remainder in AH. PF, SF, and ZF flags are affected
CBW	None	CBW	If AL > 80H , then AH ← 0 If AL ← 7F, then AH ← FFH	Before dividing AX by a byte operand, CBW extends the sign of a byte dividend in AL into AH, thus converting AL into a valid signed word in AX; flags are not affected
CWD	None	CWD	If AX < 8000H, then DX ← 0 If AX > 7FFFH, then DX ← FFFFH	It works as CBW but extends the sign of a word dividend in AX into double word in DX:AX; flags are not affected

Table 10.11 8086 logical instructions set summary

Opcode	Operand	Mnemonics	Symbolic Operation	Comments
NOT	Destination	NOT AX NOT [SI]	AX ← $\overline{AX}$ [SI] ← [$\overline{SI}$]	Complement all bits of the byte or word operand; flags are not affected
AND	Destination, source	AND AX, BX AND AL, [SI] AND AX,0200H	AX←AX. BX AL←AL. [SI] AX←AX. 0200H	Perform logical AND operation of the source and destination byte or word operands bit by bit; the result is stored in the destination operand; AF is undefined, all other flags

				are updated
OR	Destination, source	OR AX, BX OR AL,[SI] OR AX,0200H	AX←AX + BX AL←AL + [SI] AX←AX+0200H	Perform logical OR operation of the source and destination byte or word operands bit by bit; the result is stored in the destination operand. AF is undefined, all other flags are updated
XOR	Destination, source	XOR AX, BX XOR AL, [SI] XOR AX,0200H	AX←AX ⊕ BX AL←AL⊕ [SI] AX←AX⊕ 0200H	Perform logical exclusive-OR operation of the source and destination byte or word operands bit by bit; the result is stored in the destination operand. AF is undefined, all other flags are updated
TEST	Destination, source	TEST AX,BX TEST AL,[SI] TEST AX,0200H	AX. BX; update flags AL. [SI]; update flags AX. 0200H; update flags	Perform logical AND operation of the source and destination byte or word operands bit by bit: the operands remain unchanged; AF is undefined, all other flags are updated

Table 10.12 8086 shift and rotate instructions set summary

Opcode	Operand	Mnemonics	Symbolic Operation	Comments
SAL/SHL	Destination, count	SAL AX,1 SAL AX,CL	A_{n+1}←A_n, A_{15}←A_{14}, A_0←0 CF←A_{15}	Shift word or byte operand left or right once or CL times
SAR	Destination, count	SAR AX,1 SAR AX,CL	A_n←A_{n+1}, CF←A_0, A_{15} ← A_{15}	AF is undefined, all other flags are updated;
SHR	Destination, count	SHR AX,1 SHR AX,CL	A_n ← A_{n+1}, CF←A_0, A_{15}←0	For single-bit shift operation, OF is set if the sign of the operand changes
RCL	Destination, count	RCL AX,1 RCL AX,CL	A_{n+1}←A_n, CF←A_{15}, A_0←CF	Rotate word or byte operand left or right once or CL times; CF and OF are affected; For single-bit shift operation, OF is set if the sign of the operand changes.
RCR	Destination, count	RCR AX,1 RCR AX,CL	A_n←A_{n+1}, A_{15}←CF, CF←A_0	
ROL	Destination, count	ROL AX,1 ROL AX,CL	A_{n+1} ← A_n, A_0←A_{15}, CF←A_{15}	
ROR	Destination, count	ROR AX,1 ROR AX,CL	A_n ← A_{n+1}, A_{15}←A_0, CF←A_0	

Table 10.13 8086 Jump instructions set summary

Opcode	Operand	Mnemonics	Symbolic Operation	Comments
JMP	Near target	JMP MEM JMP [MEMW] JMP [BX] JMP AX	IP←MEM IP←[MEMW+1:MEMW] IP←[BX+1:BX] IP←AX	After execution of JMP instruction, transfer con trol to near target location within the segment; the addressing mode will be direct, memory indirect or register indirect
JMP	Short target	JMP SHORT MEM	IP←MEM	After execution of this instruction transfer control to short target location; the addressing mode will be direct only
JMP	Far target	JMP FAR MEMF JMP[MEMW] JMP DWORD [BX]	IP←0003H; CS←9000H IP←[0102H:1001H]; CS←[0104H:0103H] IP←[BX+1:BX]; CS←[BX+3:BX+2]	After execution of this instruction transfer control to far target location within the segment
Jcond	Short target	JNC MEM	If CF=0, then IP←MEMS	After execution of this instruction transfer control to the short target address if the condition is true. Conditional jumps are possible only for short targets
JCXZ	Short target	JCXZ MEM	If CX=0, then IP←MEMS	If CX=0, transfer control to the short target address

Table 10.14 8086 loop instructions set summary

Opcode	Operand	Mnemonics	Symbolic Operation	Comments
LOOP	Short target	LOOP MEM	CX ← CX – 1 If CX ≠ 0 , then IP ← MEM	Decrement CX register and transfer control to the short target address if CX ≠ 0
LOOPE/ LOOPZ	Short target	LOOPZ MEM	CX ← CX – 1 If (CX ≠ 0) . (ZF = 1) , then IP ← MEM	Decrement CX register and transfer control to the short target address if (CX ≠ 0). (ZF=1; this instruction affect the flag ZF = 1
LOOPNE/ LOOPNZ	Short target	LOOPNZ MEM	CX ← CX – 1 If (CX ≠ 0) . (ZF = 0), then IP ← MEM	Decrement CX register and transfer control to the short target address if (CX ≠ 0). (ZF = 0); this instruction affect the flag ZF = 0

Table 10.15 8086 CALL and RETURN instructions set summary

Opcode	Operand	Mnemonics	Symbolic Operation	Comments
Call	Near target	CALL MEM	SP ← SP – 2; [SP + 1: SP] ← IP; IP ← MEM	IP is pushed onto the top of the stack and control is transferred within the segment to the near target address
		CALL [MEMW]	SP ← SP – 2; [SP + 1: SP] ← IP; IP← [0101H; 0100H]	
		CALL [DI]	SP ← SP – 2; [SP + 1: SP] ← IP; IP← [DI+ 1: DI]	
		CALL DI	SP ← SP – 2; [SP + 1: SP] ← IP; IP← DI	
CALL	Far target	CALL FAR MEMF	SP ← SP – 2; [SP + 1: SP] ← CS; SP← SP- 2; [SP+ 1:SP] ← IP; IP ← 0100H	CS and IP are pushed onto the top of the stack and control is transferred to the new segment and far target address
		CALL [MEMW]	Same as above except: CS← [0103H : 0102H]; IP← [0101H;0100H]	
		CALL DWORD [DI]	Same as above except: CS← [DI +3:DI +2]; IP← [DI + 1: DI]	
RET	n(near)	RET	IP← [SP + 1: SP]; SP← SP+2	The word at the top of the stack is popped into IP transferring control to this new address; RET normally used to return control to the instruction following a near subroutine call; if included, the optional pop value (n) is added to SP
		RET 8	IP← [SP + 1: SP]; SP← SP+2+8	
RET	n(far)	RET	IP← [SP + 1: SP]; SP← SP+2; CS←[SP + 1: SP]; SP← SP+ 2;	As the above except that double word at the top of the stack is popped into IP and CS transferring control to this new far address
		RET 8	IP← [SP + 1: SP]; SP← SP+2; CS←[SP + 1: SP]; SP← SP+ 2 + 8 ;	

Table 10.16 8086 string instructions set summary

Opcode	Operand	Mnemonics	Symbolic Operation	Comments
STOSB	None	STOSB	ES:[DI] ←AL If DF = 0, DI ←DI+1. If DF = 1, DI ←DI-1.	Transfer a byte or word from register AL to the string element addressed by DI in the extra segment; When DF = 0, increment DI, otherwise decrement DI; Flags are not affected
STOSW	None	STOSW	ES:[DI] ←AL ES:[DI+1] ←AH. If DF = 0, DI ←DI+2. If DF = 1, DI ←DI-2.	Transfer a word from register AX to the string element addressed by DI in the extra segment; If DF = 0, increment DI, else decrement DI; Flags are not affected
STOS	Destination	STOS MEMB	ES:[MEMB] ←AL If DF=0, MEMB ← MEMB + 1. If DF=1, MEMB ← MEMB -1.	Transfer a byte from register AL to the string element addressed by DI in the extra segment; when DF = 0, increment MEMB, otherwise decrement MEMB. Flags are not affected
		STOS MEMW	ES:[MEMW] ←AL ES:[MEMW +1] ←AH. If DF=0, MEMW ← MEMW +2. If DF=1, MEMW ← MEMW -2.	Transfer a word from register AX to the string element addressed by DI in the extra segment; if DF=0, increment MEMW, else decrement MEMW. Flags are not affected
LODSB		LODSB	AL←DS:[SI]. If DF=0, SI←SI+1. If DF=1, SI← SI-1.	Transfer a byte from the string element addressed by DS:SI to register AL; If DF=0, increment SI, else decrement SI. Flags are not affected
LODSW		LODSW	AL←DS:[SI]. AH←DS:[SI+1]. If DF=0, SI←SI+2. If DF=1, SI←SI-2.	Transfer a word from the string element addressed by DS:SI to register AX; If DF=0, increment SI, else decrement SI; Flags are not affected
LODS	Source	LODS MEMB	AL←DS:[MEMB]. If DF=0, MEMB←MEMB+1. If DF=1, MEMB←MEMB-1.	Transfer a byte from the string element addressed by DS:MEMB to register AL; When DF=0, increment MEMB, else decrement MEMB; flags are not affected

		LODS MEMW	AL←DS:[MEMW]. AH←DS:[MEMW+1]. If DF=0, MEMWDS←MEMW+2. If DF=1, MEMW←MEMW-2.	Transfer a word from the string element addressed by DS: MEMW to register AX. when DF=0, increment MEMW, otherwise decrement MEMW; flags are not affected
MOVSB	None	MOVSB	ES:[DI] ←DS:[SI]. If DF=0, DI←DI+1, SI←SI+1. If DF=1, DI←DI-1, SI←SI-1.	Transfer a byte from the string element addressed by DS:SI to the string element addressed by ES: DI; if DF = 0, increment SI and DI, else decrement SI and DI. Flags are not affected
MOVSW	None	MOVSW	ES:[DI] ←DS:[SI] ES:[DI+1] ←DS:[SI+1] If DF=0, DI←DI+2 SI←SI+2 If DF=1, DI← DI-2 SI← SI-2.	Transfer a word from the string element addressed by DS:SI to the string element addressed by ES:DI; if DF = 0, increment SI and DI, else decrement SI and DI. Flags are not affected

Table 10.17 8086 string instructions set summary

Opcode	Operand	Mnemonics	Symbolic Operation	Comments
MOVS	Destination,Source	MOVS MEMBES ,MEMBDS	ES:[MEMBE] ←DS:[MEMBD]. If DF=0, MEMBE←MEMBE+1 MEMBD←MEMBD+1. If DF=1, MEMBE←MEMBE-1 MEMBD←MEMBD-1.	Transfer a byte from the string element addressed by DS:MEMBD to the string element addressed by ES:MEMBE; if DF=0, increment MEMBD and MEMBE, else decrement MEMBD and MEMBE. Flags are not affected
		MOVS MEMWE, MEMWD	ES:[MEMWE] ←DS:[MEMWD] ES:[MEMWE +1] ←DS: [MEMWD +1] If DF=0, MEMWE ← MEMWE +2 MEMWD ← MEMWD +2 If DF=1, MEMWE ← MEMWE -2 MEMWD ← MEMWD -2.	Transfer a word from the string element addressed by DS: MEMWD to the string element addressed by ES: MEMWE in the extra segment; if DF=0, increment MEMWD and MEMWE, else decrement MEMWD and MEMWE; flags are not affected
SCASB		SCASB	AL - ES:[DI]; If DF=0, DI←DI+1. If DF=1, DI←DI-1.	Subtract the byte of the string element addressed by ES:DI from AL. if DF=0, increment DI, else

				decrement DI; flags are updated
SCASW		SCASW	AX - ES:[DI+1:DI]; If DF = 0, DI←DI+2 If DF = 1, DI←DI-2.	Subtract the word of the string element addressed by ES:DI from AX; if DF=0, increment DI, else decrement DI; flags are updated
SCAS	Destination	SCAS MEMBE	AL - ES:[MEMBE]; If DF = 0, MEMBE ← MEMBE +1. If DF=1, MEMBE ← MEMBE -1.	Subtract the byte of the string element addressed by ES: MEMBE from AL; if DF=0, increment MEMBE, else decrement MEMBE. Flags are updated
		SCAS MEMWE	AX - ES:[MEMWE +1: MEMWE]; If DF=0, MEMWE I← MEMWE +2 If DF=1, MEMWE ← MEMWE -2.	Subtract the word of the string element addressed by ES: MEMWES from AX; if DF=0, increment MEMWE, else decrement MEMWE; flags are updated
CMPSB		CMPSB	DS:[SI] – ES:[DI]; If DF=0, DI←DI+1 SI←SI+1 If DF=1, DI←DI-1 SI←SI-1.	Subtract the byte of the destination string element addressed by ES:DI in the extra segment from byte of the source string element addressed by DS:SI; if DF = 0, increment DI and SI, else decrement SI and DI. Flags are updated
CMPSW		CMPSW	DS:[SI+1:SI] – ES:[DI+1 :DI] If DF=0, DI←DI+2 SI←SI+2. If DF=1, DI←DI-2 SI←SI-2	Subtract the word of the destination string ele ment addressed by ES:DI from word of the source string element addressed by DS:SI; if DF = 0, increment DI and SI, else decrement SI and DI; flags are updated

Table 10.18 8086 string instructions set summary

Opcode	Operand	Mnemonics	Symbolic Operation	Comments
CMPS	Dest, source	CMPS MEMBE, MEMBD	DS:[MEMBD] – ES: [MEMBDS]; If DF=0 MEMBE← MEMBE +1 MEMBD ← MEMBD +1 If DF=1,	Subtract the byte of the destination string element addressed by ES:MEMBE from byte of the source string element addressed by DS:MEMBD; if DF=0,

		MEMBE ← MEMBE -1 MEMBD ← MEMBD -1.	increment MEMBE and MEMBD, else decrement MEMBD and MEMBE; flags are updated
	CMPS MEMWES, MEMWDS	DS:[MEMWD+1: MEMWD] – ES: [MEMWE+1 : MEMWE]; If DF=0, MEMWE ← MEMWE +2 MEMWD ← MEMWDS +2. If DF=1, MEMWE ← MEMWE -2 MEMWD ← MEMWDS	Subtract the word of the destination string element addressed by ES:MEMBE from word of the source string element addressed by DS:MEMBD; if DF=0, increment MEMBES and MEMBDS, else decrement MEMBDS and MEMBES; flags are updated
REP	REP STOSB	CX←CX-1. Repeat until CX=0	The string instruction following the REP prefix is repeated until CX becomes to 0
	REP STOSW	CX←CX-1. Repeat until CX=0	
	REP MOVSB	CX←CX-1. Repeat until CX=0	
	REP MOVSW	CX←CX-1. Repeat until CX=0	
REPE/ REPZ	REPZ SCASB	CX←CX-1. Repeat if (ZF=1) and CX≠0	Repeat the string operation if (ZF = 1) and CX ≠ 0
	REPZ SCASW	CX←CX-1. Repeat if ZF=1 and CX≠0.	
	REPZ CMPSB	CX←CX-1. Repeat if ZF=1 and CX≠0.	
	REPZ CMPSW	CX←CX-1. Repeat if ZF=1 and CX≠0	
REPNE/ EPNZ	REPNE SCASB	CX←CX-1. Repeat if ZF=0 and CX≠0.	Repeat the string operation if (ZF=0) and CX≠0
	REPNE SCASW	CX←CX-1. . Repeat if ZF=0 and CX≠0	
	REPNE CMPSB	CX←CX-1. . Repeat if ZF=0 and CX≠0	
	REPNE CMPSW	CX←CX-1. Repeat if ZF=0 and CX≠0	

Table 10.19 8086 processor control instructions set summary

Opcode	Operand	Mnemonics	Symbolic Operation	Comments
STC	None	STC	CF ←1	Set carry flag
CLC	None	CLC	CF ←0	Clear carry flag
CMC	None	CMC	$CF \leftarrow \overline{CF}$	Complement carry flag
STD	None	STD	DF ←1	Set direction flag
CLD	None	CLD	DF ←0	Clear direction flag

STI	None	STI	IF ←1	Set interrupt flag
CLI	None	CLI	IF ←0	Clear interrupt flag
HLT	None	HLT	None	Halt
WAIT	None	WAIT	None	Wait state when $\overline{TEST}$ = 1
LOCK	Instruction	LOCK MOV AX,BX	None	$\overline{LOCK}$ = 0 used to prevent coprocessors from accessing the bus during execution of instruction
NOP	None	NOP	None	No operation
ESC	Number, source	ESC FF, MEMW	Data bus ← [MEMW]	Put the contents of the memory source operand on the data bus and execute NOP instruction

Review Questions

10.1 Define addressing modes of 8086 processors. What are the different addressing modes of 8086 microprocessors? Explain each addressing mode with examples.

10.2 Write the procedure to determine physical address for the following instructions as given below:

(i) MOV AX, [SI+03] (ii) MOV AL, CS:[BX+0400]

(iii) MOV AX, [3000] (iv) MOV AL, [BX+SI+22]

Assume CS = 4000H, IP = 2300, SI = 02300 and DS = 5000.

10.3 What is an instruction format? What are the types of instructions of 8086 microprocessors based on format?

10.4 Write the classification of 8086 instructions based on functions. Give a list of examples of different instructions.

10.5 Write the difference between the following instructions:

(i) MUL and IMUL (ii) DIV and IDIV (iii) JUMP and LOOP (iv) Shift and Rotate

10.6 Explain the execution of data transfer instructions with suitable examples.

10.7 Explain the difference between FAR CALL and NEAR CALL instructions.

10.8 Explain operation of the following instructions:

(i) ADD AX, [BX] (ii) INC SI (iii) MUL BX (iv) IMUL DX

(v) NEG AL (v) DEC DI (vi) XLAT (vi) PUSH and POP

10.9 Write the difference between following instructions:

(i) CBW and CWD (ii) MOV reg, immediate and LEA reg, address

(iii) DEC AX and SUB AX, 1 (iv) RCL and ROL (v) IRET and RET (far)

10.10 Explain the operation of the LOOP, LOOPE/LOOPZ, and LOOPNE/LOOPNZ instructions. What does the INT *n* instruction push onto the stack that the CALL FAR instruction does not? What is the JCXZ instruction typically used for?

10.11. Write instructions to perform the following operations:

(i) Copy content of BX to a memory location in the data segment with offset 0234H

(ii) Increment content of CX by 1

(iii) Multiply AX with 16 bit data 2467H

(iv) Rotate left the content of AL by two bits

10.12 Write results after execution of following instructions:

(a) MOV AL, 22; MOV BL, 44; ADD AL, BL;

(b) MOV AX, 1002; MOV BX, 44; MUL AX, BL;

(c) MOV CL, 34; MOV AL, FF; SUB AL, CL ;

(d) MOV AX, 8796; MOV CL, 2; ROR AX, CL;

10.13 Which registers are affected by the MUL, IMUL, DIV and IDIV instructions?

10.14 Which of the shift, rotate and logical instructions do not affect the zero flag?

10.15 Why does the SAR instructions always clear the overflow flag?

10.16 What does the NEG instruction do? What instruction is most similar to CMP? What instruction is most similar to TEST?

10.17 Give a list of processor control instructions and explain briefly.

10.18 What is the procedure? What are the different types of procedure in 8086? Discuss each type of procedure with examples.

Multiple-Choice Questions

10.1 What is the addressing mode of instruction MOV AX, [BX]?

(a) Register direct (b) Register indirect

(c) Immediate addressing (d) Indirect addressing

10.2 What is the addressing mode of instruction MOV AX, [BX+SI+06]?

(a) Index addressing (b) Base addressing

(c) Base index addressing (d) Base index displacement addressing

10.3 Which of the following instructions is immediate addressing?

(a) MOV AX, [2000] (b) MOV BX, 2000 (c) MOV AX, [SI] (d) MOV AX, BX

10.4 Which of the following instructions is based with 16-bit displacement addressing?

(a) MOV AX, [BX + 06] (b) MOV AX, [BP + 2000]

(c) MOV AX, [BP + 06] (d) MOV AX, [BP]

10.5 Which of the following instructions is a four-byte instruction?

(a) MOV AX, 2345 (b) MUL BX (c) DIV CL (d) ADD AX, [BP + 0200]

10.6 Which of the following instructions is a six-byte instruction?

(a) MOV [BX + DI + 0200], 2345 (b) MOV [SI], 5665

(c) DIV CL (d) ADD BX, [BP + 0200]

10.7 Which of the following instructions is a logical instruction?

(a) DIV AB (b) TEST (c) CALL (d) AAM

10.8 Which of the following instructions affects a carry flag?

(a) RCR (b) MUL AB (c) JZ (d) INC AX

10.9 Which of the following instructions is an arithmetic instruction?

(a) DIV AB (b) ROR (c) STI (d) WAIT

10.10 The example of a string instruction is
(a) MOV DX, [SI] (b) XLAT (c) MOVSB (d) AAD

10.11 2's complement instruction is
(a) NEG (b) NOT (c) CMP (d) CMC

10.12 Which of the following instructions is used to read string of BYTES and send it to other memory location?
(a) SCASB (b) MOVSB (c) LODSB (d) LODSB

10.13 LODSB instruction is used in which of the following register combinations?
(a) ES:SI (b) ES:DI (c) DS:SI (d) DS:DI

10.14 Coprocessor control instructions are
(a) WAIT, LOCK, ESC (b) HALT, STC, CLC
(c) ROR, RCR, ROL (d) DAA

10.15 When PUSH instruction is executed, initially
(a) Upper byte of data is stored on stack and SP = SP – 1
(b) Upper byte of data is stored on stack and SP = SP + 1
(c) Lower byte of data is stored on stack and SP = SP – 1
(d) Lower byte of data is stored on stack and SP = SP + 1

10.16 A procedure can be called using the instruction
(a) JMP (b) CALL (c) RET (d) INT n

10.17 To return a procedure, we use the instruction
(a) JMP (b) CALL (c) RET (d) INT n

10.18 Direction flag is used with which of the following instructions?
(a) Data transfer (b) Branch control instructions
(c) Spring instructions (d) Logical instructions

Answers to Multiple-Choice Questions

10.1 (b)	10.2 (d)	10.3 (b)	10.4 (b)
10.5 (d)	10.6 (a)	10.7 (b)	10.8 (a)
10.9 (a)	10.10 (c)	10.11 (a)	10.12 (b)
10.13 (c)	10.14 (a)	10.15 (a)	10.16 (b)
10.17 (c)	10.18 (c)		

CHAPTER

11

Assembly-Language Program of the 8086 Microprocessor

11.1 INTRODUCTION

Machine-language programming is coding of a program in terms of 0 and 1. During this programming, the memory control is directly in the hands of the programmer and the programmer is to manage the memory of the system more efficiently. But the programming, coding and memory management techniques in machine language programming are very tedious. As the programmer writes all functions in terms of 0 and 1, the possibility of human errors is more. To write and understand the programs, a programmer should have thorough knowledge about the architecture and instruction set of processor. The disadvantages of machine-language programming are given below:

- Writing a program is very complicated and time consuming
- Possibility of errors in programming are more and humans can only feed the program byte by byte into system
- Debugging the program is very difficult
- Only program designer is to understand the program; therefore, this program is not user friendly

Assembly-language programming is comparatively simpler than machine-language programming. In the assembly-language programs, the instruction mnemonics are used to write programs directly. These programs are more readable and understandable than that of machine-language programs. In the assembly language, the address values and the constants can be identified by labels. As the labels are clear, the program becomes more understandable. The tedious byte handling and manipulations are reduced as address and constants are available inside the program and it is not required to remember them. However, different logical segments and routines may be assigned with the labels rather than the different addresses. The memory control feature of machine-language programming is left unchanged by providing storage define facilities in assembly-language programming. The documentation facility is now available in assembly language.

An assembler is a program which converts any assembly-language program into the equivalent machine codes. During conversion, firstly the address of each label is initialised which also substitutes the values for each of the constants and variables. Thereafter, the assembler generates the equivalent machine code for all mnemonics and data. The assembler also generates information about syntax errors in the program but the

assembler cannot find out any logical errors in the program. The advantages of assembler are as follows:

- Writing programs in assembly language is comparatively easier than machine-level language programming.
- The chances of errors during editing a program is less as mnemonics are used to write program.
- It is very easy to enter the program in assembly language.
- The debugging is also easier than machine-code programming as mnemonics are purpose suggestive.
- This program is more user friendly as the constants and memory address locations can be labelled. MACROS makes the task of programming easier.
- After execution, the results of programs are stored in more user-friendly form.
- Flexibility of programming in assembly-language programming is more than in machine-language programming.

In any assembly language program, the programmer should mention constants, variables, logical names of the segments, types of the different routines and modules, and end of file. These types of helps are given to the assembler using a predefined alphabetical strings called *assembler directives*. Actually, assembler directives help the assembler to understand the assembly-language programs properly and help to generate the machine codes. Usually, the following directives are commonly used in the assembly-language programming.

DB: Define Byte
DW: Define Word
DQ: Define Quadword
DT: Define Ten Bytes
ASSUME: Assume Logical Segment Name
END: End of Program
ENDP: End of Procedure
ENDS: End of Segment
EQU: Equate
LABEL: Label
LENGTH: Byte Length of a Label
NAME: Logical Name of a Module
OFFSET: Offset of a Label
ORG : Origin
PROC: Procedure
SEG: Segment of a label

To edit an assembly-language program on an IBM PC in the DOS operating system, different text editors such as Norton's editor (NE.Com), Microsoft assembler (MASM.EXE), Linker (LIINK.EXE) and Debugger (DEBUG.EXE) are commonly used. In this section the basic operations of these editors are explained briefly.

11.1.1 Norton's Editor

To start Norton's editor, type NE after C and enter the directory.

C > NE ↵

After pushing the Enter key, Norton's editors opening page will be displayed as shown in Fig. 11.1. Then type the file name. For example, assume the file name is ABC and the screen display is shown in Fig. 11.2. When any key is pressed, the ABC.ASM file will be opened as depicted in Fig. 11.3. After that, enter text to edit the assembly-language program. A sample program ABC.ASM is edited to subtract two numbers as

shown in Fig. 11.4. If we want to open a file directly, the command is C>NE ABC.ASM. Then ABC.ASM file will be opened and displayed on the CRT screen. After editing the program or modifying the existing program, the F3-E command is used to save the program and exit from Norton's Editor by using the F3-Q command.

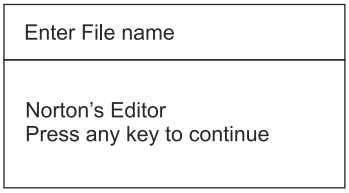


Fig. 11.1 Norton's Editor's opening page

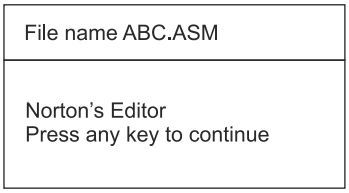


Fig. 11.2 Norton's Editor with file name ABC.ASM

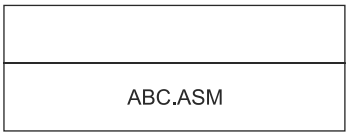


Fig. 11.3 Norton's Editor opens a file ABC.ASM

```
ASSUME    CS:CODE DS:DATA
DATA      SEGMENT
          DATA1 DW 3567 H
          DATA2 DW 1235 H
          RESULT DW 01H
DATA      ENDS
CODE      SEGMENT

LABEL     MOV AX,DATA
          MOV DS,AX
          MOV AX,DATA1
          MOV BX,DATA2
          SUB AX,BX
          MOVE AH,4CH
          INT 21H
CODE      ENDS
          END LEVEL
```

ABC.ASM

Fig. 11.4 Program for subtraction of two numbers in Norton's Editor

11.1.2 MASM Editor

The Microsoft assembler MASM is a most popular assembler and it is very easy to use. To enter an assembly-language program, the command is

C > MASM ABC ↵

or

C > MASM ABC.ASM ↵

When the above commands are executed, Fig. 11.5 will be displayed on the screen. If we enter the command C > MASM ↵, Fig. 11.6 will be displayed as the opening page of MASM.

In Fig. 11.6, the source filename should be typed in the source filename with or without extension.ASM. Then valid file name will be accepted if the Enter key is pressed. Thereafter, enter the .OBJ file name which creates the object file of the assembly language program. The listing file is identified by the source filename and an extension .LST. This file consists of LEVELS, OFFSET ADDRESS, MNEMONICS, DIRECTIVES and other necessary assembly-language related information. The cross reference filename is also entered in the same way as the listing file. This file is used for debugging the source program. The .CRF file contains information such as size of the file in bytes, number of labels, list of labels, and routines of the source program. After entering the cross-reference file name, the assembly-language process starts. Then syntax errors of the program are displayed using error code and the corresponding line number, if the program has any syntax errors. When the programmer removes all syntax errors, the assembly process will be completed

successfully. After successful assembly process, the .OBJ, .LST and .CRF files, are generated and these files can be used by the linker programmer to link the object modules and generate an executable (.EXE) file.

```
C>MASM ABC
Microsoft Macro assembler Version 5.10
Copyright Microsoft Corp. 1981,1989


Object filename [FILE.OBJ]:
List filename [NUL.LST]:
List filename [NUL.CRF]:
```

Fig. 11.5 MASM Opening Page

```
C>MASM
Microsoft Macro assembler Version 5.10
Copyright Microsoft Corp. 1981,1989


Object filename [.ASM]:
List filename [FILE.OBJ]:
List filename [NUL.LST]:
List filename [NUL.CRF]:
```

Fig. 11.6 MASM Opening Page

11.1.3 LINK

The LINK.EXE file links the different object modules of the source program and function library routines to generate the executable code of the source program. The input to the linker is the .OBJ file. The linker program is executed by the command

C>LINK ↵ or C>LINK ABC.OBJ ↵

and the display on the screen is shown in Fig. 11.7.

```
C>LINK
Microsoft Linker Version 3.64
Copyright Microsoft Corp. 1981,1989


Object Module [.OBJ]:
List File [.EXE]:
List filename [NUL.LST]:
Libraries [LIB]:
```

Fig. 11.7 Link command screen display

11.2 ASSEMBLY-LANGUAGE COMMANDS

DEBUG.COM is a program which allows the programmer to write, execute, debug and trouble shooting of assembly language programs. This command is used to examine the contents of registers and memory. It can also be used to execute a program, but one instruction at a time. To start DEBUG, it is required to type DEBUG after DOS prompt as given below:

C: \>DEBUG

After typing the above command, press the 'Enter' key. Then DEBUG command executed and a hyphen which is known as DEBUG prompt will be displayed as given below:

C: \>DEBUG

-

DEBUG has a number of valid commands which are used to write, execute and debug programs. Each

command is designed to perform an important task. There are commands to assemble and execute programs, display the contents of registers and memories, to run a program in single step mode, etc. Generally DEBUG command is represented by a one-letter symbol such as A for assemble, E for entry, D for display, etc.

11.2.1 A (Assemble Command)

The A command is used to enter mnemonics of an assembly-language program and convert it into machine codes. When A command is executed, the machine codes are generated and directly stored in the memory.

After getting DEBUG prompt the starting address is initialized as - A address

Assume the starting offset address as 1000H, then the assemble can be written as

- A 1000

When the above instruction is executed, the display on the screen will be

17DA : 1000

Here, 17DA is the content of Code Segment (CS) register and the offset address will be 1000H.

When offset is not written after A command, the DEBUG command will be executed assuming the offset address 1000H.

To edit some instructions, write instruction after address as given below:

Example 11.1 To load 16-bit data 2045H in AX register and another 16-bit data 6545 in BX register.

```
C: \>DEBUG
- A 1000
17DA : 1000  MOV AX,2045
17DA : 1003  MOV BX,6545
17DA : 1006  HLT
17DA : 1007
```

11.2.2 U (Un-assemble Command)

The U command is used to disassemble machine codes of specified memory locations and generates corresponding mnemonics and machine codes. This command displays machine codes and mnemonics on the screen. The default register is CS : IP and the general format is

-U address range

In Example 11.1, the program is written from 17DA: 1000 to 17DA : 1006. To display machine codes on the screen the U command can be written as follows:

-U 1000 1006

After execution the above command, the display on screen will be as given below:

```
17DA : 1000  B8 45  20  MOV AX,2045
17DA : 1003  BB 45 65  MOV BX,6545
17DA : 1006  F4          HLT
-
```

When the U command is applied without specifying any address range, DEBUG un-assembles the first 32 bytes starting from the address which is located by the content of IP register or it un-assembles 32 bytes since last U. The default register for U command is the CS:IP. For example, the display is shown on the screen after execution of U command without mentioning address.

Assume the following instructions are stored in the memory locations from 17DA: 1000 to 175A:1020

```
C: \>DEBUG
- A 1000
17DA : 1000  MOV AX, 2045   ; Load 2045H in AX register
17DA : 1003  MOV BX, 6545   ; Load 6545H in BX register
17DA : 1006  MOV CX, 1234   ; Load 1234H in CX register
17DA : 1009  MOV DX, 1234   ; Load 1234H in DX register
17DA : 100C  ADD  AX, BX    ; Add content of BX with AX
17DA : 100E  ADD  AX, CX    ; Add content of CX with AX
17DA : 1010  ADD  AX, DX    ; Add content of DX with AX
17DA : 1012  MOV  SI, 0100  ; Store 0100 in SI register
17DA : 1015  MOV  DI, 0100  ; Store 0100 in DI register
17DA : 1018   NEG AX        ; 2's complement of AX
17DA : 101A  NEG BX         ; 2's complement of BX
17DA : 101C  NEG CX         ; 2's complement of CX
17DA : 101E  NEG DX         ; 2's complement of DX
17DA : 1020  HLT
17DA : 1021
```

If we execute the –U 1000 command is executed, the display on screen will be

```
-U 1000
17DA : 1000   B8 45 20   MOV AX, 2045
17DA : 1003   BB 45 65   MOV BX, 6545
17DA : 1006   B9 34 12   MOV CX, 1234
17DA : 1009   BA 34 12   MOV DX, 1234
17DA : 100C   01 D8      ADD  AX, BX
17DA : 100E   01 C8      ADD  AX, CX
17DA : 1010   01 D0      ADD  AX, DX
17DA : 1012   BE 00 01   MOV  SI, 0100
17DA : 1015   BF 00 02   MOV  DI, 0100
17DA : 1018   F7 D8      NEG AX
17DA : 101A   F7 DB      NEG BX
17DA : 101C   F7 D9      NEG CX
17DA : 101E   F7 DA      NEG DX
```

11.2.3 R (Register Command)

The register command R can be used to display the contents of one or more registers. This instruction also displays the status of the flags. The general format of the R command is

-R register name

For example, the execution of –R AX command, the DEBUG displays the content of AX register as follows:

```
C: \>DEBUG
- A 1000
```

```
17DA : 1000  MOV AX, 2045
17DA : 1003  MOV BX, 6545
17DA : 1006  HLT
17DA : 1007
-GCS:1006
AX=20445  BX=6545  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17DA   ES=17DA SS=17DA CS=17DA IP=0100    NV UP EI PL NZ NA PO NC
17DA : 1006   F4  HLT
-R AX
 AX   2045
:
```

When the name of a register is given after R command, the name and the content of that register is displayed in the next line. A colon is then displayed as a prompt in the subsequent line. If a new value after the colon is typed and Enter key is pressed for execution, the content of register will be changed. After, that the debug prompt is displayed. Again, to see the content of register, write the R command with register name as given below.

```
-R AX
 AX   2045
:5000
AX 5000
:
```

Usually, the R command is given without a name of a register; the DEBUG displays the contents of all registers including status flags.

```
-R
AX=5000   BX=6545   CX=0000   DX=0000   SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17DA   ES=17DA  SS=17DA  CS=17DA   IP=0100     NV UP EI PL NZ NA PO NC
17DA : 1006   F4  HLT
```

Status Flags The status flags with their codes for RESET and SET are illustrated in Table 11.1. F is the flag register-name to visualise the status of flags. When -R F command is executed, the status of the flags are displayed in the beginning of the next line. At the end of the list of status of the flags, a hyphen (-) is displayed as given below. If we want to change status flags, after the hyphen type the desired flags in any order as per requirement, and then press Enter key.

```
-R F
```

After execution, the flag register will display the format as given below:

```
NV UP EI PL NZ NA PO NC -
```

If we want to change NC to CY, enter desired status of the desired flags after the hyphen as given below:

```
NV UP EI PL NZ NA PO NC - CY
```

To check the changed condition after given R F command, we execute -R F command and the display on the screen will be

```
-R F
```

OV UP EI NG ZR AC PO CY -

Table 11.1 Status flags for RESET and SET

Name of the flag	Reset	Set
Overflow	NV	OV
Direction	UP	DN
Interrupt	DI	EI
Sign	PL	NG
Zero	NZ	ZR
Auxiliary Carry	NA	AC
Parity	PO	PE
Carry	NC	CY

11.2.4 G (Go Command)

The Go command is used to execute any program. The general format of the G command is G = address. The equal sign (=) put before the specified address which indicates the starting address of program. The default register for the G command is the CS. Assume a program for addition of two 16-bit numbers which are loaded in AX and BX registers written from 17DA : 1000 as given below:

```
C: \>DEBUG
- A 1000
17DA : 1000  MOV AX, 2000
17DA : 1003  MOV BX, 3000
17DA : 1006  ADD AX, BX
17DA : 1008  HLT
17DA : 1009
-G=1000
Program terminated normally
```

To execute the above program and to visualise the results, we should write the command as follows:

```
-G 1008
```

In the above command, 1008 is the end address of the program.

Then the program will be starting from the address CS: 1000 and the contents of all registers and flags will be displayed as given below:

```
AX=5000   BX=3000   CX=0000   DX=0000   SP=0004   BP=20CD  SI=0000  DI=0000
DS=17DA   ES=17DA   SS=9FFF  CS=17DA   IP=1008   NV UP EI PL NZ NA PE NC
17DA:1008 F4          HLT
```

Sometimes, the GCS command is used to execute the program. The general format of GCS command is -GCS: end address of program. The example of GCS command for execution of the above program is given below:

```
C:\>DEBUG
-A1000
17DA:1000 MOV AX, 2000
17DA:1003 MOV BX, 3000
```

```
17DA:1006 ADD AX, BX
17DA:1008 HLT
17DA:1009
-GCS:1008
```

After execution, the result will be displayed as given below:

```
AX=5000  BX=3000  CX=0000  DX=0000  SP=0004  BP=20CD  SI=0000  DI=0000
DS=17DA  ES=17DA  SS=9FFF  CS=17DA  IP=1008  NV UP EI PL NZ NA PE NC
17DA:1008 F4          HLT
```

11.2.5 T (Trace Command)

The trace command T is used to run a program in single-step mode. The default register is the CS:IP and the general format of the trace command is

```
-T = address
```

To test the trace command, we should write a simple program as given below:

```
C:\>DEBUG
-A1000
17DA:1000  MOV AX, 2456
17DA:1003  MOV BX, 6000
17DA:1006  ADD AX, BX
17DA:1008  HLT
17DA:1009
```

To execute the above program in single-step mode, enter the command

```
-T = 1000
```

When we push the enter key after T=1000 command, the first instruction of the program will be executed and the result will be displayed on the screen as given below:

```
AX=2456  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17DA  ES=17DA  SS=17DA  CS=17DA  IP=1003  NV UP EI PL NZ NA PO NC
17DA:1003 BB0060       MOV     BX, 6000
```

The last line of the display is the next instruction which will be executed. To execute the next instruction, the T command is used in the following format as given below:

```
-T
AX=2456  BX=6000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17DA  ES=17DA  SS=17DA  CS=17DA  IP=1006  NV UP EI PL NZ NA PO NC
17DA:1006 01D8         ADD     AX, BX
-
```

When the T command is entered and the enter key is pressed, 17DA:1003 address instruction MOV BX, 6000 will be executed by default as the content of IP is 1003 and the results are displayed as given above.

To execute any instruction of the program use T = address command. For example, if we want to execute the third instruction at the memory location 17DA:1006 ADD AX, BX, we should write the following command:

```
-T=1006
AX=8456  BX=6000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17DA  ES=17DA  SS=17DA  CS=17DA  IP=1008   OV UP EI NG NZ NA PE NC
17DA:1008 F4            HLT
-
```

Before typing the T command, the contents of AX and BX register are 2456 and 6000 respectively. After execution T command, the third instruction will be executed and result will be stored in AX register as shown above.

If want to execute more than one instruction by a single command in single-step mode, the common format of the T command is

-T = address location value

Here, the address location is the starting address of first instruction from which execution will be started and value = *n*, the number of instruction to be executed by single command in single-step mode. Suppose we want to execute the first and second instructions of a program as illustrated by a single command in single-step mode, we write the command as given below:

```
C:\>DEBUG
-A 1000
17DA:1000 MOV AX,1234
17DA:1003 MOV BX, 3456
17DA:1006 ADD AX, BX
17DA:1008 HLT
17DA:1009
-T=1000 02
AX=1234  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17DA  ES=17DA  SS=17DA  CS=17DA  IP=1003   NV UP EI PL NZ NA PO NC
17DA:1003 BB0060        MOV     BX, 6000
AX=1234  BX=3456  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17DA  ES=17DA  SS=17DA  CS=17DA  IP=1006   NV UP EI PL NZ NA PO NC
17DA:1006 01D8          ADD     AX, BX
-
```

After applying the T=1000 02 command, the first instruction 17DA:1000 MOV AX, 1234 will be executed and the result will be displayed on screen. After that the second instruction 17DA:1003 MOV BX, 3456 will be executed and again the result will be displayed on screen as shown above.

11.2.6 D (Display Command)

The D command is used to display the contents of specified memory locations. The default register is DS. The general format of the D command is

-D or -D address

```
C:\>DEBUG
-A1000
17DA:1000 MOV AX, 2000
```

17DA:1003 MOV BX, 3000
17DA:1006 HLT
17DA:1007
-D

The display on screen will be from starting address 17DA:0100 by default as given below:

```
17DA:0100  00 00 00 00 00 00  00 00-00 00 00 00 00 00  00 00  ..........
17DA:0110  00 00 00 00 00 00  00 00-00 00 00 00 00 00  00 00  ..........
17DA:0120  00 00 00 00 00 00  00 00-00 00 00 00 00 00  00 00  ..........
17DA:0130  00 00 00 00 00 00  00 00-00 00 00 00 00 00  00 00  ..........
17DA:0140  00 00 00 00 00 00  00 00-00 00 00 00 00 00  00 00  ..........
17DA:0150  00 00 00 00 00 00  00 00-00 00 00 00 00 00  00 00  ..........
17DA:0160  00 00 00 00 00 00  00 00-00 00 00 00 00 00  00 00  ..........
17DA:0170  00 00 00 00 00 00  00 00-00 00 00 00 00 00  00 00  ..........
```

Each line can display about 16 bytes. There is a hyphen between the eighth and ninth bytes.

When the –D 1100 is executed, the following data will be displayed on screen

```
17DA:1100  B8 00 20 BB 00 30  F4 00-00 00 00 00 00 00  00 00  ..........
17DA:1110  00 00 00 00 00 00   00 00-00 00 00 00 00 00  00 00  ..........
17DA:1120  00 00 00 00 00 00   00 00-00 00 00 00 00 00  00 00  ..........
17DA:1130  00 00 00 00 00 00   00 00-00 00 00 00 00 00  00 00  ..........
17DA:1140  00 00 00 00 00 00   00 00-00 00 00 00 00 00  00 00  ..........
17DA:1150  00 00 00 00 00 00   00 00-00 00 00 00 00 00  00 00  ..........
17DA:1160  00 00 00 00 00 00   00 00-00 00 00 00 00 00  00 00  ..........
17DA:1170  00 00 00 00 00 00   00 00-00 00 00 00 00 00  00 00  ..........
```

The other format of D command is

-D address range

The D command incorporating address range is written as D 1100 1105, the display on screen will be

```
17DA:1100   B8 00 20 BB 00 30  F4
```

When we write the D command with starting address, the DEBUG command is executed and by default 80 (hex) bytes or 128 bytes (80 hex) starting from the given address will be displayed. The command D 0100 will display 128 bytes starting from the memory location DS:0100.

11.2.7 E (Enter Command)

The E command is used to enter machine codes or data. This instruction operates with the register DS by default. This command can be used in the following ways such as

- Sequentially enter data or machine codes
- Replace data or machine codes of certain memory locations

The common format to enter data or machine codes sequentially is

-E address list

When the data are to be entered in DS segment starting from the offset address 0500, the command will be written as

-E 0500

After writing the E command, data should be entered in the same line with one space between two adjacent data as given below:

-E 0500 01 02 03 04 05 06 07 08

Here data are 01 02 03 04 05 06 07 08 and the starting address is DS:0500. After execution the above command, data will be entered into memory location DS:0500 to DS:0507. The content of DS:0500 is 01 and the content of DS:0507 is 08.

When we want to replace data in DS:0400, we should write the command as E address. The example of replacing command is given below:

-E 0400

17DA:0400 00.12 00.34 00.56 00.78 00.99 00.AA 00.BB 00.CC

-E 0400

17DA:0400 12.01 34.02 56.03 78.04 99.05 AA.06 BB.07 CC.08

-E 0400

17DA:0400 01. 02. 03. 04. 05. 06. 07. 08.

When E 0400 command is executed, this will show the contents of the memory location DS:0400 in the next line as

17DA:0400 00.

where, 17DA is the initial setting for DS by default. 00 is the content of the memory location 17DA:0400. If we want to change the existing data, enter the new data 12 as shown below:

17DA:0400 00.12

If we want to change data of the next memory location, the space bar is to be pressed. It will show the content of the next memory location. To change the existing data 00, enter the new data 34. In this way, data replacement can be preceded. The maximum number of bytes that can be entered in a line is 8. After replacing the desired number of data, the Enter key is pressed. While data replacement is continued up to the end of the line, the next line with current memory address comes automatically.

11.2.8 F (Fill Command)

The Fill command is used to fill the specified range of memory locations with the values which are entered in a list. The default segment register is the DS. The general format is

-F address range list of data

The example of Fill command is

-F 0300 0304 12 34 56 78 90

Then a list of values from 12 to 90 will be filled in the memory locations DS:0300 0304. This can be verified using D command as given below:

C:\>DEBUG

-F 0300 0304 12 34 56 78 90

-D 0300 0304

17DA:0300 12 34 56 78 90

-

11.2.9 M (Move Command)

Generally, data movement operation is performed with the M command. The M command usually used to copy/move the block of data from one memory block into another memory block. By default the DS register is used to locate data. The general format of the M command is

-M range address

The example of move command is -M 0100• 0105 0200.

or -M 0100 L 06 0200

After execution of this command, the data starting from DS:0100 to DS:0105 are copied into the memory location address beginning from DS:0200 to DS:0205.

11.2.10 S (Search Command)

The search command S is used to search the specified memory locations for the specified list of bytes. By default data segment register, DS is used to locate data. The general format is

-S address range list

The list may contain one byte or more than one byte of data. If the list contains only one byte, all addresses of the byte in the specified range will be displayed. If the list contains more than one byte, then only the first addresses of the byte string are returned.

To search a byte (44H) in a specified memory range from DS: 0100 to DS: 0200, the command may be written as follows:

S 0100 0200 44.

11.3 ASSEMBLY-LANGUAGE PROGRAMS

11.3.1 Program for Addition of Two 8-Bit Numbers with a Sum of 8 Bits

Assume the first number 22H is stored in AL register and the second number 33H is stored in BL register. The result after addition of two numbers is to be stored in AL register.

Algorithm

1. Load first number in AL register.
2. Store the second data in BL register.
3. Add second data with AL register.

```
C:\>DEBUG
-A 1000
17DA:1000 MOV AL, 22   ; Load 22H in AL register
17DA:1002 MOV BL, 33   ; Load 33H in BL register
17DA:1004 ADD AL, BL  ; Add content of BL to AL
17DA:1006 HLT
17DA:1007
-G 1006
AX = 0055  BX = 0033  CX = 0000  DX = 0000  SP = 0004  BP = 20CD  SI = 0000  DI = 0000
```

```
DS = 17DA  ES = 17DA  SS = 9FFF  CS = 17DA  IP = 1006   NV UP EI PL NZ NA PE NC
17DA:1006 F4            HLT
-
```

The program is loaded in the memory location 17DA:1000 to 17DA:1006. After editing the program, when the above program is executed by G1006 command, the result will be displayed on the screen. The content of AL is 55H which is the addition of 22H and 33H.

11.3.2 Program for Addition of Two 16-Bit Numbers with a Sum of 16 Bits

The first 16-bit number is stored in AX register. The second 16-bit number is stored in BX register. After addition, the result will be stored in AX.

Algorithm

1. Store first 16-bit number in AX.
2. Store second 16-bit number in BX.
3. Addition of 1st and 2nd number.

```
C:\>DEBUG
-A 1000
17DA:1000 MOV AX, 4622   ; 16 bit data in AX.
17DA:1003 MOV BX, 3244   ; 16 bit data in BX.
17DA:1006 ADD AX, BX     ;  Contents of BX is added to AX.
17DA:1008 HLT
17DA:1009

-G 1008
AX = 7866  BX = 3244  CX = 0000  DX = 0000  SP = 0004  BP = 20CD  SI = 0000  DI = 0000
DS = 17DA  ES = 17DA  SS = 9FFF  CS = 17DA  IP = 1008   NV UP EI PL NZ NA PE NC
17DA:1008 F4            HLT
```

The program for addition of two 16-bit numbers is loaded in the memory location 17DA:1000 to 17DA:1008. After editing the program, when the above program is executed by G1008 command, the result will be displayed on the screen. The content of AX is 7866H which is the addition of 4622H and 3244H. As the sum is of 16 bits, no carry is generated.

11.3.3 Program for Addition of a String of Words with a Sum of 16 Bits

Assume the number of 16-bit data is stored in CX register and a string of words are stored in 17DA:0300 to 17DA:0305. After addition the result is stored in BX. Initially content of BX is 0000H

Algorithm

1. Initialize SI register with 0300H as source address of data.
2. Load number of bytes to be added in CX register.
3. Load a word in AX from the source specified by SI and SI is incremented by 2.
4. Addition of AX content and BX content.

5. Move content of AX to BX.
6. Continue step-3 to step-5 until CX=0.

```
C:\>DEBUG
-A 0100
17DA:0100  MOV SI, 0300    ; Source address in SI.
17DA:0103  MOV CX, 0005  ; Count value is loaded in CX.
17DA:0106  MOV AX, [SI]    ; Load AX with data which is located by SI
17DA:0108  ADD AX, BX     ; Contents of BX in AX.
17DA:010A  INC SI              ; Increment SI
17DA:010B  INC SI              ; Increment SI
17DA:010C  MOV BX, AX   ; Contents of BX in AX.
17DA:010E  DEC CX           ; Decrement CX
17DA:010F  JNZ 0106          ; Jump to 0106 if CX≠0
17DA:0111  HLT
17DA:0112
-ECS:0300
17DA:0300  00.01   00.01   00.02   00.02   00.03   00.03   00.04   00.04
17DA:0308  00.05   00.05
-G 0111
AX=0F0F  BX=0F0F  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=030A  DI=0000
DS=17DA  ES=17DA  SS=17DA  CS=17DA  IP=0111   NV UP EI PL ZR NA PE NC
17DA:0111 F4            HLT
-
```

The above program is entered in the memory location 17DA:0100 to 17DA:0111. Five 16-bit data 0101, 0202, 0303, 0404 and 0505 are entered by the command -ECS:0300 17DA:0300 00.01 00.01 00.02 00.02 00.03 00.03 00.04 00.04 17DA:0308 00.05 00.05. After addition 0101, 0202, 0303, 0404, and 0505 we get 0F0F. Therefore, when the program is executed by G0111 command, the result will be displayed on the screen as the content of BX and AX registers, 0F0F.

11.3.4 Program for Subtraction of Two 16-Bit Numbers

Consider 1st 16-bit number is in AX register and the 2nd number is in BX register. After subtraction, the result will be stored in AX.

Algorithm

1. Load 1st number in AX register.
2. Load 2nd number in BX register.
3. Subtract BX from AX.

```
C:\>DEBUG
-A1000
17DA:1000 MOV AX, FFFF   ; 16 bit data in AX
17DA:1003 MOV BX, 6666   ; 16 bit data in BX
17DA:1006 SUB AX,BX        ; Contents of BX is subtracted from AX
```

```
17DA:1008 HLT
17DA:1009
-G1008
AX=9999  BX=6666  CX=0000  DX=0000  SP=0004  BP=20CD  SI=0000  DI=0000
DS=17DA  ES=17DA  SS=9FFF  CS=17DA  IP=1008   NV UP EI NG NZ NA PE NC
17DA:1008 F4            HLT
-
```

The program for subtraction of two 16-bit numbers is entered in the memory location 17DA:1000 to 17DA:1008. When the above program is executed by G1008 command; the result will be available in AX. Therefore, the result is content of AX is 9999H which is the subtraction of FFFFH and 6666H.

11.3.5 Program for 1's Complement of a 16-Bit Number

A 16 bit number is stored in AX register. Result of 1's complement of AX will be stored in BX.

Algorithm

1. Load 16-bit data in AX register.
2. Complement the accumulator.
3. Store one's complement of AX in BX.

```
C:\>DEBUG
-A 1000
17DA:1000 MOV AX, 9999 ; 16 bit data in AX
17DA:1003 NOT AX           ; 1's complement of 16 bit data
17DA:1005 MOV BX, AX   ; Result is stored in BX
17DA:1007 HLT
17DA:1008

-G 1007
AX = 6666  BX = 6666  CX = 0000  DX = 0000  SP = 0004  BP = 20CD  SI = 0000  DI = 0000
DS = 17DA  ES = 17DA  SS = 9FFF  CS = 17DA  IP = 1007   NV UP EI NG NZ NA PO NC
17DA:1007 F4            HLT
-
```

The above program is used for 1's complement of a 16-bit number and it is entered in the memory location 17DA:1000 to 17DA:1007. The above program is executed by G1007 command and result will be available in AX. After that, the result is content of AX is stored in BX.

11.3.6 Program for 2's Complement of a 16-Bit Number

Assume that the 16-bit number is stored in AX register. Find the two's complement of the number and stored it in BX register.

Algorithm

1. Store the 16-bit number in AX register.

2. Determine 2's complement of AX.
3. Store two's complement of AX in BX.

```
C:\>DEBUG
-A 1000
17DA:1000 MOV AX, 2244 ; 16 bit data in AX
17DA:1003 NEG AX       ; 2's complement of 16-bit data
17DA:1005 MOV BX, AX   ; Result is stored in BX
17DA:1007 HLT
17DA:1008
-G 1007
AX = DDBC  BX = DDBC  CX = 0000  DX = 0000  SP = 0004  BP = 20CD  SI = 0000  DI = 0000
DS = 17DA  ES = 17DA  SS = 9FFF  CS = 17DA  IP = 1007   NV UP EI NG NZ AC PO CY
17DA:1007 F4           HLT
-
```

The program for 2's complement of a 16-bit number is stored in the memory location 17DA:1000 to 17DA:1007. When this program is executed by G1007 command, the result will be available in AX. Therefore, the content of AX is copied into BX.

11.3.7 Program for 2's Complement of a String of Words

Assume a string of words are stored in 17DA:0300 to 17DA:030B. The number of words is stored in CX register. Determine the 2's complement of the string words and stored at the destination address 17DA:0400 to 17DA:040B.

Algorithm

1. Initialize SI with 0300 as source address of data.
2. Load number of words in CX registers and initialise DI with 0400 as destination address.
3. Load a word in AX from the source specified by SI and SI is incremented by 2.
4. Determine 2's complement of AX and store at destination address represented by DI.
5. Continue step-3 and step-4 until CX=0.

```
C:\>DEBUG
-A 0100
17DA:0100  MOV SI,0300     ; load 0300H in source index register
17DA:0103  MOV CX,0005     ; load number of bytes in CX register
17DA:0106  MOV DI,0400     ; load 0400H in destination  index register
17DA:0109  LODSW           ; load AL with data addressed by SI and SI = SI + 1
17DA:010A  NEG AX          ; 2's complement of 16 bit data
17DA:010C  STOSW           ; store AL addressed by DI and DI = DI + 1. Result is stored in BX
17DA:010D  LOOPNZ 0109 ; Loop unless CX = 0
17DA:010F  HLT
17DA:0110
-ECS:0300
```

```
17DA:0300  00.01   00.01   00.02   00.02   00.03   00.03   00.04   00.04
17DA:0308  00.05   00.05   00.06   00.06
-G 010F
AX = FAFB  BX = 0000  CX = 0000  DX=0000  SP = FFEE  BP = 0000  SI = 030A  DI = 040A
DS = 17DA  ES = 17DA  SS = 17DA  CS = 17DA  IP = 010F   NV UP EI NG NZ AC PO CY
17DA:010F F4           HLT
-ECS:0400
17DA:0400  FF.    FE.    FE.    FD.    FD.    FC.    FC.    FB.
17DA:0408  FB.    FA.    00.
-
```

The above program is used to find 2's complement of a string of five 16-bit numbers which are stored in the memory location 17DA:0300 to 17DA:030B and the program is stored in the memory location 17DA:0100 to 17DA:010F. ECS:0300 command is used to store five 16-bit numbers such as 0101, 0202, 0303, 0404 and 0505. This program can be executed by G010F command and the result will be stored in destination memory location 17DA:0400 to 17DA:040B. To see the result, ECS:0400 command should be used and the result will be displayed as given above.

11.3.8 Program to Multiply Two 8-Bit Numbers

Assume the first number 11H is stored in AL register and the second number 22H is stored in BL register. Multiply contents of AL by BL and the result is to be stored in AX register.

Algorithm

1. Load first number in AL register.
2. Store the second data in BL register.
3. Multiply contents of AL by BL.

```
C:\>DEBUG
-A 1000
17DA:1000 MOV AL, 11   ; 8 bit multiplicand in AL
17DA:1002 MOV BL, 22   ; 8 bit multiplier in BL
17DA:1004 MUL BL       ; Multiply contents of AL by BL
17DA:1006 HLT
17DA:1007
-G 1006
AX=0242  BX=0022  CX=0000  DX=0000  SP=0004  BP=20CD  SI=0000  DI=0000
DS=17DA  ES=17DA  SS=9FFF  CS=17DA  IP=1006   OV UP EI NG NZ NA PO CY
17DA:1006 F4           HLT
-
```

The program for multiplication of two 8-bit numbers is edited in the memory location 17DA:1000 to 17DA:1006. This program is executed by G1006 command and result will be stored in AX register. Hence the result is content of AX=0242H.

11.3.9 Program to Multiply Two 16-Bit Numbers

Assume that the first number 1111H is stored in AX register and the second number 2222H is stored in BX register. Multiply contents of AX by BX and the result is to be stored in DX and AX registers.

Algorithm

1. Load first number in AX register.
4. Store the second data in BX register.
5. Multiply contents of AX by BX.

```
C:\>DEBUG
-A 1000
17DA:1000  MOV AX, 1111   ;16-bit multiplicand in AX
17DA:1003  MOV BX, 2222   ; 16-bit multiplicand in BX
17DA:1006  MUL BX         ; Multiply contents of AX by BX
17DA:1008  HLT
17DA:1009
-G 1008
AX = 8642  BX = 2222  CX = 0000  DX = 0246  SP = 0004  BP = 20CD  SI = 0000  DI = 0000
DS = 17DA  ES = 17DA  SS = 9FFF  CS = 17DA  IP = 1008   OV UP EI NG NZ NA PO CY
17DA:1008 F4          HLT
-
```

The above program is used to multiply two 16-bit numbers and it is stored in the memory location 17DA:1000 to 17DA:1006. This program can be executed by G1008 command and the result will be stored in DX and AX. The result after multiplication of 1111H and 2222H is more than 16 bits. Then lower 16 bits result is stored in AX and upper 16 bits (most significant bit) of result is stored in DX. Here result is the content of DX = 0246 and AX = 8642.

11.3.10 Program to Divide Two 8-Bit Numbers

Assume the first number 56H is stored in AL register and the second number 02H is stored in CL register. Divide contents of AL by CL and the result is to be stored in AX register.

Algorithm

1. Load first number in AL register.
2. Store the second data in CL register.
3. Divide contents of AL by CL and result is AX.

```
C:\>DEBUG
-A 1000
17DA:1000   MOV AL, 56   ; 8 bit dividend in AL
17DA:1002   MOV CL, 02   ; 8 bit divisor in CL
17DA:1004   DIV CL       ; Divide contents of AL by CL
17DA:1006   HLT
17DA:1007
-G 1006
AX=002B   BX=0000  CX=0002  DX=0000  SP=0004  BP=20CD  SI=0000  DI=0000
```

```
DS=17DA  ES=17DA  SS=9FFF  CS=17DA  IP=1006   NV UP EI NG NZ NA PO NC
17DA:1006 F4            HLT
-
```

The program for division of two 8-bit numbers is stored in the memory location 17DA:1000 to 17DA:1006. When this program is executed by G1006 command, the result will be stored in AX register. Quotient is stored in AL and remainder is stored in AH. Hence the result is quotient=content of AL= 2B and remainder =content of AH = 00H.

11.3.11 Program to Divide Two 16-Bit Numbers

Assume that the first number FFFFH is stored in AX register and the second number 2222H is stored in CX register. Divide AX by CX and the result is to be stored in DX and AX registers.

Algorithm

1. Load first number in AX register.
2. Store the second data in CX register.
3. Divide contents of AX by CX and store result in AX and DX registers.

```
C:\>DEBUG
-A 1000
17DA:1000   MOV AX, FFFF  ; 16 bit dividend in AX
17DA:1003   MOV CX, 2222  ; 16 bit divisor in CX
17DA:1006   DIV CX        ; Divide contents of AX by CX
17DA:1008   HLT
17DA:1009
-G 1008
AX = 0007  BX = 0000  CX = 2222  DX = 1111  SP = 0004  BP = 20CD  SI = 0000  DI = 0000
DS = 17DA  ES = 17DA  SS = 9FFF  CS = 17DA  IP = 1008   NV UP EI NG NZ NA PO NC
17DA:1008 F4            HLT
-
```

Quotient = 07 in AL and reminder in DX register

The above program is used to divide two 16-bit numbers and is stored in the memory location 17DA:1000 to 17DA:1008. If the program is executed by G1008 command, the result will be stored in DX and AX. Quotient is stored in AX and remainder is stored in DX. Hence the result is quotient = content of AX = 0007 and remainder =content of DX = 1111H.

11.3.12 Program for Rotating a 16-Bit Number Left through Carry by One Bit

The 16-bit number is stored in AX and rotated left one bit through carry. Store the result in AX.

Algorithm

1. Load 16-bit data, 1234 in AX register.
2. Rotate the content of AX left through carry by one bit

```
C:\>DEBUG
```

```
-A0100
17B3:0100   MOV AX, 1234     ; Load 1234 in AX
17B3:0103   RCL AX, 1        ; Rotate AX left by 1 bit
17B3:0105   HLT
-G 0105
AX = 2468  BX = 0000  CX = 0000  DX = 0000  SP = FFEE  BP = 0000  SI = 0000  DI = 0000
DS = 17B3  ES = 17B3  SS = 17B3  CS = 17B3  IP = 0105   NV UP EI PL NZ NA PO NC
17B3:0105 F4          HLT
```

The program for rotating a 16-bit number left through carry by one bit is stored in the memory location 17B3:0100 to 17B3:0103. After execution of this program by G0105 command and result will be stored in AX register. So the result is content of AX = 2468.

11.3.13 Program for Shift Left of a 16-Bit Number by One Bit

Assume that the 16-bit number is stored in AX and shift left carry. Store the result in memory location starting from 17DA:1010.

Algorithm

1. Load 16-bit data, 4567 in AX register.
2. Shift left AX by one bit and store the result in memory location.

```
C:\>DEBUG
-A1000
17DA:1000  MOV AX, 4567     ;16-bit data 4567H  in AX
17DA:1003  SHL AX, 1        ; Data is shifted left by one bit
17DA:1005  MOV [1010], AX  ; Save the content of AX in  17DA:1010
17DA:1008  HLT
-G1008
AX = 8ACE  BX = 0000  CX = 0000  DX = 0000  SP = 0004  BP = 20CD  SI = 0000  DI = 0000
DS = 17DA   ES = 17DA  SS = 9FFF  CS = 17DA  IP = 1008   OV UP EI NG NZ NA PO NC
17DA:1008 F4          HLT
-ECS:1010
17DA:1010  CE.     8A.
```

The program for shifting a 16-bit number left by one bit is entered in the memory location 17DA:1000 to 17DA:1008. When this program is executed by G1008 command and result will be available in AX register and the content of AX = 8ACE is stored in 17DA:1010 as it is displayed by ECS:1010 command.

11.3.14 Program to Find the Largest Number from a String of Words

The count value of number of words 05H is stored in CX register. A string of words is stored in the memory locations starting from 17B3:0300 to 17B3:0309. The largest word will be stored in memory location 17B3:0400.

Algorithm

1. Initialize SI as source offset address of word and load numbers of words in CX register.

2. Initialize AX register with 0000H.
3. Compare the word from memory with content of accumulator AX. If the word is greater than and equal to AX, jump to step-5.
4. Move word from memory to AX.
5. Increment SI by 2.
6. Decrement CX. If CX ≠ 0, jump to step-3.
7. Store the result in memory location 17B3:0400.

```
C:\>DEBUG
-A0100
17B3:0100 MOV SI, 0300      ; Initialize SI with 0300H as source offset address of word
17B3:0103 MOV CX, 0005      ; Count value of words in CX
17B3:0106 MOV AX, 0000      ; Initialize AX with 0000H
17B3:0109 CMP AX, [SI]      ; Compare word from memory with AX
17B3:010B JAE 010F          ; Jump to 010F if above and equal
17B3:010D MOV AX, [SI]      ; Load word from memory
17B3:010F INC SI            ; Increment SI
17B3:0110 INC SI            ; Increment SI
17B3:0111 LOOPNZ 0109       ; If CX ≠ 0, jump to 0109
17B3:0113 MOV [0400], AX    ; Store the content of AX at destination address
17B3:0116 HLT
-ECS:0300
17B3:0300  00.11   00.11   00.22   00.22   00.33   00.33   00.FF   00.FF
17B3:0308  00.99   00.99
-G0116
AX = FFFF  BX = 0000  CX = 0000  DX = 0000  SP = FFEE  BP = 0000  SI = 030A  DI = 0000
DS = 17B3  ES = 17B3  SS = 17B3  CS = 17B3  IP = 0116   NV UP EI PL NZ NA PE NC
17B3:0116 F4            HLT
-ECS:0400
17B3:0400  FF.     FF.
```

The five 16-bit data or word 1111, 2222, 3333, FFFF, and 9999 are stored in the memory location starting from 17B3:0300 to 17B3:0309. Therefore the largest number is FFFFH. After execution of above program, the largest number is stored in memory location 17B3:0400 and it can be displayed by ECS:0400 command as given above.

11.3.15 Program to Find the Smallest Number from a String of Words

The count value of words 05H is stored in CX register. A string of words is stored in the memory locations starting from17B3:0300 to 17B3:0309. The smallest word will be stored in memory location 17B3:0400.

Algorithm

1. Initialize SI as source offset address of word and load numbers of words in CX register.
2. Initialize AX register with FFFF .

3. Compare the content of memory with content of accumulator, if number in AX is a smaller jump to step-5.
4. Move number from memory to AX.
5. Increment SI by 2.
6. Decrement CX. If CX ≠ 0, jump to step-3.
7. Store result in memory location 17B3:0400.

```
C:\>DEBUG
-A0100
17B3:0100   MOV SI, 0300     ; Initialize SI with 0300H as source offset address of  word
17B3:0103   MOV CX, 0005   ; Count value of numbers in CX
17B3:0106  MOV AX, FFFF   ; Initialize AX with FFFFH
17B3:0109   CMP AX, [SI]     ; Compare word from memory with AX
17B3:010B  JB 010F             ; Jump  to 010F  if  number in AX is smaller
17B3:010D  MOV AX, [SI]     ; Load  word from memory
17B3:010F   INC SI               ; Increment SI
17B3:0110   INC SI               ; Increment SI
17B3:0111   LOOPNZ 0109   ; If CX ≠ 0, jump to 0108
17B3:0113   MOV [0400], AX; Store the content of AX at destination address
17B3:0116   HLT
-ECS:0300
17B3:0300  00.12   00.34   00.23   00.45   00.34   00.56   00.45   00.67
17B3:0308  00.56   00.78
-G0116
AX = 3412  BX = 0000  CX = 0000  DX = 0000  SP = FFEE  BP = 0000  SI = 030A  DI = 0000
DS = 17B3  ES = 17B3  SS = 17B3  CS = 17B3  IP = 0116   NV UP EI PL NZ NA PE CY
17B3:0116 F4            HLT
-ECS:0400
17B3:0400  12.     34.
-
```

The five 16-bit data or word 1234, 2345, 3456, 4567, and 5678 are stored in the memory location starting from 17B3:0300 to 17B3:0309. Therefore, the smallest number is 1234H. After execution of the above program, the smallest number is stored in memory location 17B3:0400 and it can be displayed by ECS:0400 command as given above.

11.3.16 Program to Transfer a Block of Data from One Section of Memory to the Other Section of Memory

Assume that a string of bytes is stored in the memory locations starting from 17B3:0300 to 17B3:030E. The count value of bytes 0FH is stored in CX register. Move the block of data starting from memory location 17B3:0300 to 17B3:030E to other memory location starting from 17B3:0400 to 17B3:040E. The count value of words 0FH is stored in CX register.

Algorithm

1. Initialize SI as source offset address of bytes and load numbers of bytes in CX register.
2. Initialize DI as destination offset address of bytes.
3. Move byte from source to destination.
4. Increment SI and DI by 1.
5. Decrement CX. If CX ≠ 0, jump to step-3.

```
C:\>DEBUG
-A0100
17B3:0100   MOV SI, 0300    ; Load Source address of data in SI
17B3:0103   MOV DI, 0400    ; Load Destination address of data in SI
17B3:0106   MOV CX, 000F    ; Count value for number of bytes in CX
17B3:0109   MOVSB           ; Move byte from source to destination
17B3:010A   LOOPNZ 0109     ; Decrement CX. If CX ≠ 0, jump 0109
17B3:010C   HLT
17B3:010D
-ECS:0300
17B3:0300  00.FF   00.EE   00.DD   00.CC   00.AA   00.BB   00.99   00.88
17B3:0308  00.77   00.66   00.55   00.44   00.33   00.22   00.11   00.00
17B3:0310  00.12   00.13
-G010C
AX = 0000  BX = 0000  CX = 0000  DX = 0000  SP = FFEE  BP = 0000  SI = 030F  DI = 040F
DS = 17B3  ES = 17B3  SS = 17B3  CS = 17B3  IP = 010C   NV UP EI PL NZ NA PO NC
17B3:010C F4            HLT
-ECS:0400
17B3:0400  FF.    EE.    DD.    CC.    AA.    BB.    99.    88.
17B3:0408  77.    66.    55.    44.    33.    22.    11.    00.
17B3:0410  00.
```

A block of fifteen 8-bit data FF, EE, DD, CC, AA, BB, 99, 88, 77, 66, 55, 44, 33, 22 and 11 are stored in the memory location starting from 17B3:0300 to 17B3:030E. After execution of the above program by the command G010C, the said data is stored in new memory location string from 17B3:0400 to 17B3:040E and it can be displayed by ECS:0400 command as shown above.

11.3.17 Program to Add Two ASCII Numbers

ASCII numbers are in values from 30 H to 39 H for representing numbers 0 to 9. ADD 39 and 38 which are ASCII numbers represents 9 and 8 respectively. AAA instruction is used for ASCII adjustment after addition. The program for addition of two ASCII numbers is stored in the memory location 17B3:0100 to 17B3:0109. After execution ADD AL, BL instruction, the result is 0071H which is stored in AX register. Thereafter execution of AAA instruction result is available in AX and it is 0107.

```
C:\>DEBUG
-A0100
```

```
17B3:0100   MOV AL, 39   ; Load first ASCII number in AL
17B3:0102   MOV BL, 38   ; Load second ASCII number in BL
17B3:0104   MOV AH, 00   ; Initialize AH with 00H
17B3:0106   ADD AL, BL   ; Add content of BL with AL
17B3:0108   AAA          ; ASCII adjustment of AX after addition
17B3:0109   HLT
-G0109
AX = 0107  BX = 0008  CX = 0000  DX = 0000  SP = FFEE  BP = 0000  SI = 0000  DI = 0000
DS = 17B3  ES = 17B3  SS = 17B3  CS = 17B3  IP = 0109   NV UP EI PL NZ AC PE CY
17B3:0109 F4      HLT
```

11.3.18 Program to Subtract Two ASCII Numbers

Subtract 39 and 34 which are ASCII numbers represent 9 and 4 respectively. AAS instruction can be used for ASCII adjustment after subtraction. The program for subtraction of two ASCII numbers is stored in the memory location 17DA:0100 to 17DA:010B. After the execution of SUB AL, BL instruction, the result is 0005H which is stored in AX register. Afterward execution of AAS and OR AL,30, the instruction result 0035 is available in AX.

```
C:\>DEBUG
-A0100
17DA:0100   MOV   AL,39  ; Load first ASCII number in AL
17DA:0102   MOV   BL,34  ; Load second ASCII number in BL
17DA:0104   MOV   AH,00  ; Initialize AH with 00H
17DA:0106   SUB    AL,BL ; Subtract content of BL from AL
17DA:0108   AAS          ; ASCII adjustment of AX after subtraction
17DA:0109   OR      AL,30  ; OR 30H with AL
17DA:010B  HLT
-G010B
AX = 0035  BX = 0034  CX = 0000  DX = 0000  SP = FFEE  BP = 0000  SI = 0000  DI = 0000
DS = 17DA  ES = 17DA  SS = 17DA  CS = 17DA  IP = 010B   NV UP EI PL NZ NA PE NC
17DA:010B F4          HLT
```

11.3.19 Program to Multiply Two ASCII Numbers

The multiplication of ASCII numbers should not be done unless the most significant number (MSN) is cleared. To multiply two ASCII numbers 38 and 32, initially after removing MSN, the numbers are represented as 08 and 02. AAM instruction can be used for ASCII adjustment after multiplication. The program for multiplication of two ASCII numbers is stored in the memory location 17B3:0100 to 17B3:0108. After the execution of MUL BL instruction, the result 0005H is stored in AX register. Subsequently execution of AAM instruction, final result 0106 is available in AX .

```
-A0100
17B3:0100  MOV AL,08 ; Load first ASCII number in AL
17B3:0102  MOV BL,02 ; Load second ASCII number in BL
```

```
17B3:0104  MUL BL      ; Multiply BL with AL
17B3:0106  AAM         ; ASCII adjustment after multiplication
17B3:0108  HLT
-G 0108
AX = 0106  BX = 0002  CX = 0000  DX = 0000  SP = FFEE  BP = 0000  SI = 0000  DI = 0000
DS = 17B3  ES = 17B3  SS = 17B3  CS = 17B3  IP = 0108   NV UP EI PL NZ NA PE NC
17B3:0108 F4           HLT
```

11.3.20 Program to Divide Two ASCII Numbers

The division of ASCII numbers should not be done unless the most significant number (MSN) is cleared. Therefore, AAD instruction requires a two-digit unpacked BCD number before execution. After entering the two digit unpacked BCD number, AAD is executed to adjust the content of AX. Then AX is divided by an unpacked BCD number to generate results. The program for division of two ASCII numbers (divide 75 by 8) is stored in the memory location 17B3:0100 to 17B3:0109. After execution of the above program, result is stored in AX register. The quotient 09 is in AL and reminder 03 is in AH register. The result is also an unpacked BCD.

```
C:\>DEBUG
-A0100
17B3:0100 MOV AX,0705  ; Load ASCII number (two digit unpacked BCD) in AX
17B3:0103 AAD          ; ASCII adjustment before division
17B3:0105 MOV BL,08    ; Load second ASCII number in BL
17B3:0107 DIV BL       ; Divide AL by BL
17B3:0109 HLT
-G0109
AX = 0309  BX = 0008  CX = 0000  DX = 0000  SP = FFEE  BP = 0000  SI = 0000  DI = 0000
DS = 17B3  ES = 17B3  SS = 17B3  CS = 17B3  IP = 0109   NV UP EI PL NZ NA PE NC
17B3:0109 F4           HLT
```

11.3.21 Program to Arrange a String of Words in Descending Order

A series of five words 1111H, 5555H, 3333H, 2222H, and 4444H are stored in memory locations from 17B3:0302 to 17B3:030B and number of words is stored in the memory location 17B3:0300. Arrange the above words in descending order.

Algorithm

1. Store 0005H, number of words to be arranged in DX register from memory and store number of comparisons in CX register.
2. Load the 1st word in accumulator from memory.
3. Increment SI register by 2 for addressing the next word.
4. Compare the next word from memory with accumulator. Store the smallest word in accumulator and largest word in memory.
5. Then next number (word) is compared with the accumulator and store the largest number in memory and smallest number in accumulator.

6. This process will continue till comparisons of all numbers have been completed. After completion of comparison of all numbers, the smallest number in accumulator is stored in memory. In this way, the first process will be completed.
7. At the starting of second process, DX register is decremented by one and store number of comparisons in CX register. Then repeat step-2 to step-6. After completion of this process, the smallest number in 17B3:030A and second smallest number in 17B3:0308.
8. DX register is decremented by one and the next process starts, if the content of DX register is not zero. Then repeat step-2 to step-6.

```
C:\>DEBUG
-A0100
17B3:0100   MOV SI, 0300      ; SI loaded with 0300 as source address
17B3:0103   MOV DX, [SI]      ; DX loaded with the content of  SI
17B3:0105   MOV CX, [SI]      ; CX loaded with the content of  SI
17B3:0107   DEC CX            ; Decrement CX
17B3:0108   INC SI            ; Increment SI
17B3:0109   INC SI            ; Increment SI
17B3:010A   MOV AX, [SI]      ; AX is loaded with word from memory represented by SI
17B3:010C   INC SI            ; Increment SI
17B3:010D   INC SI            ; Increment SI
17B3:010E   CMP AX,[SI]       ; Compare AX with the content of memory represented by SI
17B3:0110   JNB 0118          ; Jump to 0118
17B3:0112   XCHG AX, [SI]     ; Exchange AX and word stored at memory represented by SI
17B3:0114   DEC SI            ; Decrement SI
17B3:0115   DEC SI            ; Decrement SI
17B3:0116   MOV [SI], AX      ; Move AX  to memory represented by SI
17B3:0118   LOOP 010A         ; CX decrement by 1. If CX ≠ 0, jump to 010A
17B3:011A   DEC DX            ; Decrement DX
17B3:011B   MOV SI, 0300      ; SI loaded with 0300 as source address
17B3:011E   JNZ 0105          ; Jump not zero to 0105
17B3:0120   HLT
-ECS:0300
17B3:0300  00.05   00.00   00.11   00.11   00.55   00.55   00.33   00.33
17B3:0308  00.22   00.22   00.44   00.44
-G0120
AX = 3333  BX = 0000  CX = 0000  DX = 0000  SP = FFEE  BP = 0000  SI = 0300  DI = 0000
DS = 17B3  ES = 17B3  SS = 17B3  CS = 17B3  IP = 0120   NV UP EI PL ZR NA PE NC
17B3:0120 F4           HLT
-ECS:0300
17B3:0300  05.    00.    55.    55.    44.    44.    33.    33.
17B3:0308  22.    22.    11.    11.
```

The above program is used to arrange a string of words in descending order and this program is stored into memory 17B3:0100 to 17B3:0120. The 5 words (1111H, 5555H, 3333H, 2222H, and 4444H) are entered by the command ECS:300. Thereafter, G0120 is used to execute the program and words will be stored in descending order (5555H, 4444H 3333H, 2222H, and 1111H) as given above.

11.3.22 Program to Arrange a String of Words in Ascending Order

A series of five words 4444H, 1111H, 5555H, 2222H, and 3333H are stored in memory locations from 17B3:0302 to 17B3:031B and number of words is stored in memory location 17B3:0300. Arrange the above words in ascending order.

Algorithm

1. Store 0005H, number of words to be arranged in DX register from memory and store number of comparisons in CX register.
2. Load the 1st word in accumulator from memory.
3. Increment SI register for addressing the next word.
4. Compare next word from memory with accumulator. Store the largest number in accumulator and smallest number in memory.
5. Then next number is compared with the accumulator and store the smallest number in memory and largest number in accumulator.
6. This process will continue, till comparisons of all numbers have been completed. After completion of comparison of all numbers, the largest number in accumulator will be stored in memory. In this way, the first process will be completed.
7. At the starting of the second process, DX register is decremented by one and store number of comparisons in CX register. Then repeat step-2 to step-6. After completion of this process, largest number is stored in 17B3:0302A and second largest number, in 17B3:0308.
8. DX register is decremented and the next process starts, if the content of DX register is not zero. Then repeat step-2 to step-6.

```
C:\>DEBUG
-A0100
17B3:0100   MOV SI, 0300     ; SI loaded with 0300 as source address
17B3:0103   MOV DX, [SI]     ; DX loaded with the content of SI
17B3:0105   MOV CX, [SI]     ; CX loaded with the content of SI
17B3:0107   DEC CX           ; Decrement CX
17B3:0108   INC SI           ; Increment SI
17B3:0109   INC SI           ; Increment SI
17B3:010A   MOV AX, [SI]     ; AX is loaded with word from memory represented by SI
17B3:010C   INC SI           ; Increment SI
17B3:010D   INC SI           ; Increment SI
17B3:010E   CMP AX, [SI]     ; Compare AX with the content of memory represented by SI
17B3:0110   JB 0118          ; Jump to 0118
17B3:0112   XCHG AX, [SI]    ; Exchange AX and word stored at memory represented by SI
17B3:0114   DEC SI           ; Decrement SI
```

```
17B3:0115   DEC SI              ; Decrement SI
17B3:0116   MOV [SI], AX        ; Move AX  to memory represented by SI
17B3:0118   LOOP 010A           ; CX decrement by 1. If CX ≠ 0, jump to
17B3:011A   DEC DX              ; Decrement DX
17B3:011B   MOV SI, 0300        ; SI loaded with 0300 as source address
17B3:011E   JNZ 0105            ; Jump not zero to 0105
17B3:0120   HLT
-ECS:0300
17B3:0300  00.05   00.00   00.44   00.44   00.11   00.11   00.55   00.55
17B3:0308  00.22   00.22   00.33   00.33   00.66   00.66
-G0120
AX = 4444  BX = 0000  CX = 0000  DX = 0000  SP = FFEE  BP = 0000  SI = 0300  DI = 0000
DS = 17B3  ES = 17B3  SS = 17B3  CS = 17B3  IP = 0120   NV UP EI PL ZR NA PE CY
17B3:0120 F4            HLT
-ECS:0300
17B3:0300  05.    00.    11.    11.    22.    22.    33.    33.
17B3:0308  44.    44.    55.    55.    66.    66.
```

The above program is used to arrange a string of words in ascending order stored in memory locations 17B3:0100 to 17B3:0120. The 5 words (4444H, 1111H, 5555H, 2222H, and 3333H) are entered by the command ECS : 300. Thereafter, G0120 is used to execute the program and all words will be stored in ascending order (1111H, 2222H, 3333H, 4444H and 5555H) as given above.

11.3.23 Program to Find the Square of a Number using Look-up Table

Load the decimal number in the accumulator. Find the square of decimal number and store it in the memory location 17B3:0400. The square values of decimal numbers from 0 to 9 are stored in 17B3:0300 to 17B3:0309H and used for look-up table.

Algorithm

1. Store the decimal number in accumulator.
2. Load 03H is AH register.
3. If the decimal number is 02, the content of AH and AL registers are 03 and 02H respectively. Then the offset address of memory location will be 0302H denoted by AX register. Move AX content into SI register.
4. Move square of decimal number in AL from memory location 17B3:0302
5. Store the result, square value in 17B3:0400.

```
C:\>DEBUG
-A0100
17B3:0100 MOV AL, 02
17B3:0102 MOV AH, 03
17B3:0104 MOV SI, AX
17B3:0106 MOV AL, [SI]
```

```
17B3:0108 MOV DI, 0400
17B3:010B MOV [DI], AL
17B3:010D HLT
-ECS:0300
17B3:0300  00.00   00.01   00.04   00.09   00.16   00.25   00.36   00.49
-G010D
AX = 0304  BX = 0000  CX = 0000  DX = 0000  SP = FFEE  BP = 0000  SI = 0302  DI = 0400
DS = 17B3  ES = 17B3  SS = 17B3  CS = 17B3  IP = 010D   NV UP EI PL NZ NA PO NC
17B3:010D F4            HLT
-ECS:0400
17B3:0400  04.
```

The above program can be used to find the square of a decimal number and it is stored into memory 17B3:0100 to 17B3:010D. When G010D is used to execute the program, the square value of 02 is 04 will be stored in the memory location 17B3:0400.

11.3.24 Program to Find the Square Root of a Number using Look-up Table

Load the decimal number in accumulator. Find the square root of decimal number and store it in the memory location 17B3:0400. The square values of decimal numbers 0, 1, 4, 9, and 16 are stored in 17B3:0300 , 03001, 0304, 0309, and 0316 and used as look-up table.

Algorithm

1. Store the decimal number in the accumulator.
2. Load 03H is AH register.
3. If the decimal number is 09, the content of AH and AL registers are 03 and 09H respectively. Then the offset address of memory location will be 0309H denoted by AX register. Move AX content into SI register.
4. Move square root of decimal number in AL from memory location 17B3:0309.
5. Store the result, square value in 17B3:0400.

```
C:\>DEBUG
-A0100
17B3:0100 MOV AL, 09
17B3:0102 MOV AH, 03
17B3:0104 MOV SI, AX
17B3:0106 MOV AL, [SI]
17B3:0108 MOV DI, 0400
17B3:010B MOV [DI], AL
17B3:010D HLT
-ECS:0300
17B3:0300  00.     00.01
-ECS:0304
17B3:0304  00.02
```

```
-ECS:0309
17B3:0309  00.03
-ECS:0316
17B3:0316  00.04
-G010D
AX = 0303  BX = 0000  CX = 0000  DX = 0000  SP = FFEE  BP = 0000  SI = 0309  DI = 0400
DS = 17B3  ES = 17B3  SS = 17B3  CS = 17B3  IP = 010D   NV UP EI PL NZ NA PO NC
17B3:010D F4            HLT
-ECS:0400
17B3:0400  03.
```

The program to find out the square root of a decimal number is stored into memory 17B3:0100 to 17B3:010D. When G010D is used to execute the program and the square root value of 09 is 03, it will be stored in memory location 17B3:0400.

11.3.25 Program to Find the Addition of Two 3 × 3 Matrices

During addition of two matrices, the corresponding matrix elements are added and developed as a new matrix as shown below:

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

The matrix A is stored in the memory location 17B3:0200 to 17B3:0208. The matrix B is stored in the memory location 17B3:0300 to 17B3:0308. After addition of two matrices, the result is stored in the memory location 17B3:0400 to 17B3:0411.

Algorithm

1. Store source address of matrix A and B in SI and DI respectively.
2. Store destination address of A + B in BX register.
3. Load number of elements of a matrix in CX register.
4. Initialize AX register.
5. Load element of matrix A from memory to AL and add the corresponding element of matrix B with AL.
6. Store addition of two corresponding elements into destination address.
7. Increment SI, and DI by one. Increment BX by 2.
8. Decrement CX by one. If CX ≠ 0, execute step-4 to step-8

```
C:\>DEBUG
-A0100
17B3:0100   MOV      SI, 0200   ; SI loaded with 0200 as source address of matrix A
17B3:0103   MOV      DI, 0300   ; DI loaded with 0300 as source address of matrix B
17B3:0106   MOV      BX, 0400 ; BX loaded with 0300 as destination address of matrix A + B
17B3:0109   MOV      CX, 0009 ; Load number of elements in a matrix
17B3:010C   MOV      AX, 0000 ; Initialize AX with 0000H
```

```
17B3:010F  MOV     AL, [SI]   ; Load element of  matrix A
17B3:0111  ADD     AL, [DI]   ; Add corresponding element of matrix B with A
17B3:0113  JNB     0118       ; Jump no carry to 0118
17B3:0115  ADD     AH,01      ; Add 01 with AH if carry generated
17B3:0118  MOV     [BX], AX   ; store addition of corresponding elements in destination address
17B3:011A  INC     SI         ; Increment SI
17B3:011B  INC     DI         ; Increment DI
17B3:011C  ADD     BX, + 02   ; Increment BX by 2
17B3:011F  LOOPNZ  010C       ; CX decrement by1. If CX ≠ 0, jump to 010C
17B3:0121  HLT
-ECS:200
17B3:0200  00.11   00.22   00.33   00.44   00.55   00.66   00.77   00.88
17B3:0208  00.99
-ECS:300
17B3:0300  00.11   00.11   00.33   00.44   00.55   00.66   00.77   00.88
17B3:0308  00.99
-G0121
AX = 0132  BX = 0412  CX = 0000  DX = 0000  SP = FFEE  BP = 0000  SI = 0209  DI = 0309
DS = 17B3  ES = 17B3  SS = 17B3  CS = 17B3  IP = 0121   NV UP EI PL NZ NA PE NC
17B3:0121 F4          HLT
-ECS:0400
17B3:0400  22.    00.    33.    00.    66.    00.    88.    00.
17B3:0408  AA.    00.    CC.    00.    EE.    00.    10.    01.
17B3:0410  32.    01.
```

The above program can be used to add two 3 × 3 matrixes and it is stored into memory 17B3:0100 to 17B3:0121. ECS : 200 is used to enter the matrix elements of A such as 11, 22, 33, 44, 55, 66, 77, 88 and 99. Similarly the matrix elements of B 11, 22, 33, 44, 55, 66, 77, 88 and 99 are entered by ECS : 300 command. When G0121 is executed, the result will be displayed as 0022, 0033, 0066, 0088, 00AA, 00CC, 00EE, 0110 and 0132.

11.3.26 Program to Find the Transpose of a 3 × 3 Matrix

The transpose of a matrix $A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$ is $A^T = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{bmatrix}$

The matrix A is stored in the memory location 17B3:0200 to17B3:0209. After transpose, A^T matrix will be stored in the memory location 17B3:0300 to17B3:0309.

Algorithm

1. Store source address of matrix A in SI and store destination address of A^T in DI.
2. Load number of rows of a matrix in CL register.
3. Load element of matrix A from memory to AL.
4. Store element into destination address.

5. Increment SI by three and DI by one.
6. Load element of matrix A from memory to AL and store into destination address.
7. Increment SI by three and DI by one.
8. Load element of matrix A from memory to AL and store into destination address.
9. Subtract 05 from SI.
10. Decrement CL, If CL ≠ 0, continue step-3 to step-10.

```
C:\>DEBUG
-A0100
17B3:0100  MOV    SI, 0200   ; SI  loaded with 0200 as source address of matrix A
17B3:0103  MOV    DI, 0300   ; DI  loaded with 03200 as destination address of matrix A
17B3:0106  MOV    CL, 03     ; Number of rows in CL
17B3:0108  MOV    AL, [SI]   ; Load element of  matrix A
17B3:010A  MOV    [DI], AL   ; Store element of A in destination address
17B3:010C ADD     SI, +03    ; Add 03 with SI
17B3:010F  INC    DI         ; Increment DI
17B3:0110  MOV    AL, [SI]   ; Load element of  matrix A
17B3:0112  MOV    [DI],AL    ; Store element of A in destination address
17B3:0114  ADD    SI, +03    ; Add 03 with SI
17B3:0117  INC    DI         ; Increment DI
17B3:0118  MOV    AL, [SI]   ; Load element of  matrix A
17B3:011A  MOV    [DI], AL   ; Store element of A in destination address
17B3:011C  SUB    SI, +05    ; Subtract 05 from  SI
17B3:011F  INC    DI         ; Increment DI
17B3:0120  DEC    CL         ; Decrement  CL
17B3:0122  JNZ    0108       ; If CL ≠ 0, jump to 0108C
17B3:0124  HLT
-ECS:200
17B3:0200  11.    22.    33.    44.    55.    66.    77.    88.
17B3:0208  99.    00.
-G124
AX=0099  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0203  DI=0309
DS=17B3  ES=17B3  SS=17B3  CS=17B3  IP=0124   NV UP EI PL ZR NA PE NC
17B3:0124 F4           HLT
-ECS:0300
17B3:0300  11.    44.    77.    22.    55.    88.    33.    66.
17B3:0308  99.
-
```

This program can be used to find transpose of a 3 × 3 matrix. The ECS : 200 command is used to enter the matrix elements of A such as 11, 22, 33, 44, 55, 66, 77, 88 and 99. When we execute the program using

G0124 command, A^T of matrix A will be stored in the destination address. To display the result, ECS : 0300 will be used and A^T will be displayed on screen as 11, 44, 77, 22, 55, 88, 33, 66 and 99.

11.3.27 Program to Find the Multiplication of Two 3 × 3 Matrices

Assume two matrices are $A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$ then multiplication of two matrices is $A \times B =$

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

The matrix A is stored in the memory location 17B3:0200 to 17B3:0208. The matrix B is stored in the memory location 17B3:0300 to 17B3:0308. After addition of two matrices, the result is stored in the memory location 17B3:0400 to17B3:0411.

```
C:\>DEBUG
-A100
17DA:0100 MOV   BX, 0400   ; BX loaded with 0400 as  destination address of matrix A × B
17DA:0103 MOV   SI, 0200   ; SI  loaded with 0200 as source address of matrix A
17DA:0106 MOV   DH, 03     ; Load number of row/column in DH
17DA:0108 MOV   DI, 0300   ; DI  loaded with 0300 as source address of matrix B
17DA:010B MOV   CL, 03     ; Load number of row/column in CL
17DA:010D MOV   DL, 03     ; Load number of row/column in DL
17DA:010F  MOV  BP, 0000   ; Initialize BP with 0000H
17DA:0112  MOV  AX, 0000   ; Initialize AX with 0000H
17DA:0115  SAHF
17DA:0116  MOV  AL, [SI]   ; Load element of  matrix A into AL
17DA:0118  MOV  CH, [DI]   ; Move corresponding element of B into CH
17DA:011A  MUL  CH         ; Multiply corresponding element of B with AL
17DA:011C  ADD  BP,  AX    ; Add content of AX with BP
17DA:011E   INC  SI        ; Increment SI by one
17DA:011F  ADD  DI, + 03   ; Add 03 with DI
17DA:0122  DEC  DL         ; Decrement DL by one
17DA:0124  JNZ  0116       ; If DL ≠ 0, Jump to 0116
17DA:0126  SUB  DI, + 08   ; Subtract 08 from DI
17DA:0129  SUB  SI, + 03   ; Subtract 03 from SI
17DA:012C  MOV  [BX], BP   ; Store element of A × B into destination memory address
17DA:012E  ADD  BX, + 02   ; Add 02 with BX
17DA:0131  DEC  CL         ; Decrement CL by one
17DA:0133  JNZ  010D       ; If CL ≠ 0, Jump to 0106
```

```
17DA:0135  ADD     SI, + 03     ; Add 03 with SI
17DA:0138  DEC     DH           ; Decrement DH by one
17DA:013A  JNZ     0108         ; If DH ≠ 0, Jump to 0106
17DA:013C  HLT
-ECS:200
17DA:0200  00.1    00.1    00.1    00.1    00.1    00.1    00.1    00.1
17DA:0208  00.1    00.1    00.1
-ECS:300
17DA:0300  00.1    00.1    00.1    00.1    00.1    00.1    00.1    00.1
17DA:0308  00.1    00.1    00.1
-G013C
AX = 0001  BX = 0412  CX = 0100  DX = 0000  SP = FFEE  BP = 0003  SI = 0209  DI = 0303
DS = 17DA  ES = 17DA  SS = 17DA  CS = 17DA  IP = 013C   NV UP EI PL ZR NA PE NC
17DA:013C F4           HLT
-ECS:400
17DA:0400  03.    00.    03.    00.    03.    00.    03.    00.
17DA:0408  03.    00.    03.    00.    03.    00.    03.    00.
17DA:0410  03.    00.
```

The above program can be used to multiply two 3 × 3 matrices. To verify the result very easily, we assume both A and B matrices are unit matrix. The ECS : 200 is used to enter the matrix elements of A and the matrix elements of B are entered by ECS : 300. When G0121 is executed, the result will be stored in destination memory location and ECS : 400 command will be displayed the result on screen as shown above.

11.3.28 Program to Find the Gray Code Equivalent of a Binary Number

The 8 bit binary number is $B_7\ B_6\ B_5\ B_4\ B_3\ B_2\ B_1\ B_0$. The Gray code number is $G_7\ G_6\ G_5\ G_4\ G_3\ G_2\ G_1\ G_0$. The relationship between Gray code and Binary code is $G_7 = B_7$ and $G_i = B_i \oplus B_{i+1}$ where $i = 0$ to 6. The program for binary to Gray conversion is given below:

```
C:\>DEBUG
-A100
17DA:0100 MOV AL, 89   ; Load the binary number in AL
17DA:0102 MOV BL, AL   ; Move AL to BL
17DA:0104 CLC          ; Clear carry
17DA:0105 RCR AL,1     ; Rotate right through carry by one bit
17DA:0107 XOR BL, AL   ; XORing  the content of BL and DL
17DA:0109 MOV DL, BL   ; Store content of  BL in DL register
17DA:010B HLT
-G010B
AX = 0044   BX = 00CD  CX = 0000  DX = 00CD  SP = FFEE  BP = 0000  SI = 0000  DI = 0000
DS = 17DA  ES = 17DA  SS = 17DA  CS = 17DA  IP = 010B   NV UP EI NG NZ NA PO NC
17DA:010B F4           HLT
```

The above program can be used to convert binary number 89H into equivalent Gray code. When the program is executed by G010B command, the equivalent Gray code of binary number 89H will be stored in DL register, i.e. CD.

11.3.29 Program to Convert BCD Number to Equivalent Binary Number

The program for converting BCD number into binary equivalent number is given below. Assume the BCD number is 2345. When the program is executed by G011E command, the equivalent binary of BCD number 2345 will be stored in DX register i.e., 0929H.

```
C:\>DEBUG
-A100
17DA:0100  MOV    BX, 2345   ; The BCD number is stored in BX
17DA:0103  MOV    CX, 0000   ; Initialize CX as 0000H
17DA:0106  CMP    BX, +00    ; Compare BX with 0000H
17DA:0109  JZ     011C       ; Jump zero to 011C
17DA:010B MOV     AL, BL     ; Copy  the content of BL to AL
17DA:010D SUB     AL, 01     ; Subtract 01 from AL
17DA:010F  DAS               ; Decimal adjustment after subtraction
17DA:0110  MOV    BL, AL     ; Copy  the content of AL to BL
17DA:0112  MOV    AL, BH     ; Copy  the content of BH to AL
17DA:0114  SBB    AL, 00     ; Subtract 00 from AL with borrow
17DA:0116  DAS               ; Decimal adjustment after subtraction
17DA:0117  MOV    BH, AL     ; Copy  the content of AL to BH
17DA:0119  INC    CX         ; increment CX
17DA:011A JMP     0106       ; Jump to 0106
17DA:011C MOV     DX, CX     ; Store the Binary number in DX register
17DA:011E  HLT
-G11E
AX = 0000  BX = 0000  CX = 0929  DX = 0929  SP = FFEE  BP = 0000  SI = 0000  DI = 0000
DS = 17DA  ES = 17DA  SS = 17DA  CS = 17DA  IP = 011E   NV UP EI PL ZR NA PE NC
17DA:011E F4          HLT
```

11.3.30 Program to Find Factorial of BCD Number

The program to determine the factorial of one digit BCD number is developed based on the algorithm as given below:

Algorithm

1. If $N = 1$, then factorial = 1
2. If $N > 1$, the factorial = $N \times$ (factorial of $(N-1)$)

```
C:\>DEBUG
-A0100
17DA:0100  MOV CX, 0005 ; Store number in CX register
```

```
17DA:0103  MOV AX, CX   ; Copy the content of CX in AX register
17DA:0105  DEC CX       ; Decrement CX
17DA:0106  MUL CX       ; Multiply CX with AX
17DA:0108  DEC CX       ; Decrement CX
17DA:0109  JNZ 0106     ; Jump not zero to 0106
17DA:010B  MOV BX, AX   ; Store factorial value in BX register
17DA:010D  HLT
-G010D
AX = 0078  BX = 0078  CX = 0000  DX = 0000  SP = FFEE  BP = 0000  SI = 0000  DI = 0000
DS = 17DA  ES = 17DA  SS = 17DA  CS = 17DA  IP = 010D   NV UP EI PL ZR NA PE NC
17DA:010D F4          HLT
```

The above program can be used to determine factorial 5 or 5!. After editing the program, G010D is used to execute it. Then factorial 5 will be stored in BX register, i.e. 78H.

11.3.31 Program to Find the Number of Positive Numbers and Negative Numbers in a Series of Signed Numbers

The algorithm to find the number of positive and negative numbers in a series of signed numbers is illustrated below:

Algorithm

1. Load the number in AX register from memory location.
2. Rotate number left through carry. Carry flag represents the most significant bit of the number.
3. If the carry flag = 1, the number is negative.
   If the carry flag = 0, the number is positive.

```
C:\>DEBUG
-A100
17DA:0100  MOV   SI, 0200     ; SI is loaded 0200 as source address of data
17DA:0103  MOV   BX, 0000     ; Initialize BX register with 0000H
17DA:0106  MOV   DX, 0000     ; Initialize DX register with 0000H
17DA:0109  MOV   CL,05        ; Load number of data in CL register
17DA:010B MOV    AX,[SI]      ; Move data into AX register from memory
17DA:010D SHL    AX,1         ; Shift left through carry
17DA:010F  JB    0114         ; Jump carry to 0114
17DA:0111  INC   BX           ; Else increment BX register
17DA:0112  JMP   0115         ; Jump to 0115
17DA:0114  INC   DX           ; Increment DX register
17DA:0115  ADD   SI, + 02     ; Add 02 with SI to locate next data in memory
17DA:0118  DEC   CL           ; Decrement CL
17DA:011A  JNZ   010B         ; Jump not zero 010B
17DA:011C  HLT
-ECS:200
```

```
17DA:0200  00.23   00.67   00.01   00.24   00.90   00.90   00.44   00.98
17DA:0208  00.26   00.98
-G11C
AX = 304C  BX = 0002  CX = 0000  DX = 0003  SP = FFEE  BP = 0000  SI = 020A  DI = 0000
DS = 17DA  ES = 17DA  SS = 17DA  CS = 17DA  IP = 011C   NV UP EI PL ZR NA PE NC
17DA:011C F4           HLT
-
```

The above program is used to determine the number of positive as well as negative numbers in a series of signed numbers. ECS:200 command is used to enter five signed numbers 6723, 2401, 9090, 9844 and 9826. After execution of the program through G11C command, number of positive numbers will be stored in BX register, i.e. 2 and number of negative numbers will be stored in DX register, i.e. 3.

11.3.32 Program to Find the Number of Even Numbers and Odd Numbers in a Series of Signed Numbers

The algorithm to find the number of even and odd numbers in a series of signed numbers is illustrated below:

Algorithm

1. Load the number in AX register from memory location
2. Rotate number right through carry. Carry flag represents whether the number is even or odd. Actually, the carry flag is least significant bit of number.
3. If the carry flag = 1, the number is odd.
   If the carry flag = 0, the number is even.

```
C:\>DEBUG
-A100
17DA:0100  MOV SI, 0200     ; SI is loaded 0200 as source address of data
17DA:0103  MOV BX, 0000     ; Initialize BX register with 0000H
17DA:0106  MOV DX, 0000     ; Initialize DX register with 0000H
17DA:0109  MOV CL, 05       ; Load number of data in CL register
17DA:010B  MOV AX, [SI]     ; Move data into AX register from memory
17DA:010D  ROR AX, 1        ; Rotate right  through carry
17DA:010F  JC 0114          ; Jump carry to 0114
17DA:0111  INC BX           ; Else increment BX register
17DA:0112  JMP 0115         ; Jump to 0115
17DA:0114  INC DX           ; Increment DX register
17DA:0115  ADD SI, 2        ; Add 02 with SI to locate next data in memory
17DA:0118  DEC CL           ; Decrement CL
17DA:011A  JNZ 010B         : Jump not zero to 010B
17DA:011C  HLT
-ECS:200
17DA:0200  00.01   00.23   00.00   00.45   00.41   00.34   00.61   00.92
-ECS:208
```

```
17DA:0208  00.00   00.89
-G11C
AX = 4480  BX = 0002  CX = 0000  DX = 0003  SP = FFEE  BP = 0000  SI = 020A  DI = 0000
DS = 17DA  ES = 17DA  SS = 17DA  CS = 17DA  IP = 011C   NV UP EI PL ZR NA PE NC
17DA:011C F4           HLT
-
```

The above program can be used to determine the number of even as well as odd numbers in a series of signed numbers. ECS:200 command is used to enter five signed numbers 2301, 4500, 3441, 9261 and 8900. When the program is executed by G11C command, the number of even numbers will be stored in BX register, i.e. 2 and number of odd numbers will be stored in DX register, i.e. 3.

Review Questions

11.1 What is an assembler? What are the different assemblers used in 8086 programming?

11.2 Explain DEBUG with some of its important commands.

11.3 What is A command? What is U command? Explain the operation of the following commands:
(i) –U 0100 0120 (ii) –U 0100 L10 (iii) –D0100 L 10 (iv) –T = 0100 05
(v) –ECS:100 (vi) –M 0100 0105 0300 (vii) S 0200 0235 46

11.4 Write the commands for the following operations:
(i) To display the content of AX register
(ii) To display the content of memory locations CS: 0200 to CS: 0250
(iii) To enter data 22H, 44H, 66H 77H and FFH in the memory location starting from DS: 0300
(iv) To search a byte 45H from a string DS: 0400 to DS: 0500
(v) To display the status of flags

11.5 What are the advantages of assembly-language programming over machine-language programming?

11.6 Write an assembly-language program to find the largest number in a data array.

11.7 Write an assembly-language program to find the smallest number in a data array.

11.8 Write an assembly-language program to find the subtraction of two 3 × 3 matrix.

11.9 Write an assembly-language program to arrange numbers in descending order.

11.10 Write an assembly-language program to arrange numbers in ascending order.

11.11 Write an assembly-language program to block move from one memory location to another location.

11.12 Write an assembly-language program to find the sum of a series of 16-bit numbers.

11.13 Write an assembly-language program to find the factorial of a number.

11.14 Write an assembly-language program to find the subtraction of two 3 × 3 matrices.

11.15 Write an assembly-language program for addition, subtraction, multiplication and division of two numbers.

11.16 Write an assembly-language program for addition of the first 100 decimal numbers.

11.17 A block of 16-bit data is stored at the memory location starting from DS:0100. Move this block to the memory location starting from DS:0500.

11.18 Write an assembly-language program to convert a 16-bit binary number into equivalent GRAY code.

11.19 Write an assembly-language program to find out the number of occurrences of a byte 44H in a string of bytes which is stored in the memory location starting from CS:0300 to CS:0320.

11.20 Write assembly-language programs to find (i) n ! (ii) $\frac{n!}{(n-r)!\,r!}$ (iii) ${}^{n}C_{p}$, assume $N = 9$, $r = 2$ and p=3.

11.21 Write an assembly language program to convert a binary number to its equivalent BCD number.

11.22 Write an assembly language program to find transpose of a 3 × 3 matrix.

Multiple-Choice Questions

11.1 What is the output of DL after execution of the following instructions?

MOV DL, 36

AND DL, 0F

(a) 06H (b) 60H (c) 36H (d) 0FH

11.2 What is the content of AX and DX after execution of the following instructions?

MOV BL, 9

MOV AX, 0702

AAD

DIV BL

(a) AX = 0080H BX = 0009H (b) AX = 0008H BX = 0009H

(c) AX = 0008H BX = 0090H (d) AX = 0800H BX = 0900H

11.3 What is the result after addition of two ASCII numbers 33 and 37 by the following instructions?

MOV AX, 0037

ADD AX, 0033

AAA

OR AX, 3030

(a) 303 1 (b) 1303 (c) 3130 (d) 3310

11.4 After multiplication of two numbers, the result in AX will be

MOV AL, 05

MOV CL, 05

MUL CL

AAM

(a) AX = 0205 (b) AX = 0250 (c) AX = 0025 (d) AX = 2500

11.5 Which one of the following programs is the right program for complement of a number?

(a) MOV AX, 2345 (b) MOV AX, 2345 (c) MOV AX, 2345 (d) MOV AX, 2345

NEG AX CMP AX NOT AX CMC

11.6 Result of unsigned multiplication of two numbers is

MOV CL, 25

MOV AL, 35

MUL CL

(a) AX = 7A90H (b) AX = 907AH (c) AX = A907H (d) AX = 07A9H

11.7 Result for addition of two numbers is

MOV AX, A233
MOV BX, A455
ADD AX, BX

(a) AX = 4688H (b) AX = 4886H (c) AX = 8846H (d) AX = 6884H

11.8 How many T states are required to execute the following instructions?

MOV CX,2244	4 T states
DEC CX	2 T states
NOP	3 T states
JNZ Start	16 T states

(a) 256 T states (b) 184216 T states (c) 2560 T states (d) 2000 T states

11.9 Content of AX after execution of the following instructions is

MOV AH, 00H
MOV BL, 06
MOV AL, 08
SUB AL, BL
AAS

(a) 0002 (b) FF08 (c) F8F0 (d) 80FF

11.10 To execute a program, which command is used?

(a) R (b) G (c) E (d) F

11.11 Which command is used to see the flag register status?

(a) U (b) A (c) R (d) F

11.12 After execution of MOV AX, 9535 and RCL AX,1 the content of AX is

(a) 2A6A Carry = 0 (b) 2A6A Carry = 1 (c) A26A Carry = 1 (d) 2AA6 Carry = 1

Answers to Multiple-Choice Questions

11.1 (a)	11.2 (b)	11.3 (c)	11.4 (a)
11.5 (c)	11.6 (d)	11.7 (a)	11.8 (b)
11.9 (a)	11.10 (b)	11.11 (d)	11.12 (b)

CHAPTER

12

8255 Interfacing with 8085, 8086 and 8051 Microcontroller

12.1 INTRODUCTION

8255A is a programmable peripheral interface IC and is a multi-port input/output device. This is a general-purpose programmable I/O device, which may be used with many different microprocessors. There are 24 I/O pins, which may be individually programmed in 2 groups of 12 and used, in 3 major modes of operation.

The I/O ports can be programmed in a variety of ways as per requirement of the programmer. The features of this device are given below:

- 24 programmable I/O pins
- Fully TTL compatible
- High speed, no "Wait State" operation with 5 MHz 8085, 8 MHz 80C86 and 80C88
- Direct bit set/reset capability
- Enhanced control word read capability
- 2.5 mA drive capability on all I/O ports
- Low standby power static CMOS circuit design insures low operating power

Generally, this device is used to read data from an external device and write data into an external device. Some interface circuits are available for reading/writing data in an external device. These interface circuits are called *peripheral interface circuits*. These circuits are also known as *programmable I/O ports,* as I/O ports are programmed to perform specified functions.

12.2 ARCHITECTURE OF INTEL 8255A

The schematic and pin diagram of Intel 8255A are shown in Fig. 12.1 and Fig. 12.2 respectively. It is a 40-pin IC package and operates on a single +5 V dc supply. The 8255 has 24 I/O pins, which may be individually programmed in two groups of twelve input/output lines or three groups of eight lines. The two groups of I/O pins are called *Group A* and *Group B*. Each group contains a subgroup of eight bits known as *8-bit port* and a subgroup of four bits known as *4-bit port*. This IC has three eight-bit ports: **Port A** PA_7–PA_0, **Port B** PB_7–PB_0, and **Port C** PC_7–PC_0. The port C is divided into subgroups such as **Port C upper**, PC_7–PC_4

and **Port C lower**, PC_3–PC_0. Group A consists of Port A and Port C upper. Group B consists of Port B and Port C lower. The internal block diagram of 8255 is depicted in Fig. 12.3. The pins for various ports are as follows:

PA_0–PA_7	8 pins of port A
PB_0–PB_7	8 pins of port B
PC_0–PC_3	4 pins of port C_{lower}
PC_4–PC_7	4 pins of port C_{upper}

The functional descriptions of pins are as follows:

Symbol	Type	Description
PA_0–PA_7	I/O	**PORT A**: 8-bit input and output port. Depending upon the control words *bus hold high* and *bus hold low* which are present on this port.
PB_0–PB_7	I/O	**PORT B**: 8-bit input and output port. This port is used to hold high or low in the same way as Port A.
PC_0–PC_7	I/O	**PORT C**: 8-bit input and output port. This port may be used as output latch or input buffer.
D_0–D_7	I/O	**DATA BUS** The data bus lines are bi-directional three-state pins connected to the system data bus. This three-state bi-directional 8-bit buffer is used to interface the 82C55A to the system data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the microprocessor. Control words and status information are also transferred through the data bus buffer.
RESET	I	**RESET** A high on this input initializes the control register to 9Bh and all ports (A, B, C) are set to the input mode. "Bus hold" devices internal to the 82C55A will hold the I/O port inputs to a logic "1" state with a maximum hold current of 400μA.
$\overline{CS}$	I	**CHIP SELECT** Chip select is an active low input used to enable the 82C55A onto the Data Bus for CPU communications.
$\overline{RD}$ (Read)	I	**READ** Read is an active low input control signal used by the CPU to read status information or data via the data bus. When $\overline{RD}$ is LOW the 8255 send output data or status information to the microprocessor on the data bus or the microprocessor can read data from the input port of 8255.
$\overline{WR}$	I	**WRITE** Write is an active low input control signal used by the CPU to load control words and data into the 82C55A. When $\overline{WR}$ is LOW, the CPU writes data into the output port of 8255 or writes control word into the control word register of 8255.
A_0–A_1	I	**ADDRESS** These input signals, in conjunction with the $\overline{RD}$ and $\overline{WR}$ inputs, control the selection of one of the three ports or the control word register. A_0 and A_1 are normally connected to the least significant bits of the Address Bus A_0, A_1. These lines are used to select input ports and control word register.

12.3 GROUP A AND GROUP B CONTROLS

The functional configuration of each port can be programmed by the instruction. For this, the CPU stores a control word to the 82C55A. The control word contains information about mode of operation, bit set, bit reset, etc. The control word initializes the functional configuration of the 82C55A. Each of the control blocks, Group A and Group B, receive 'commands' from the control logic signals, $\overline{RD}$ and $\overline{WR}$ receives 'control words' from the internal data bus and issues the proper commands to its associated ports.

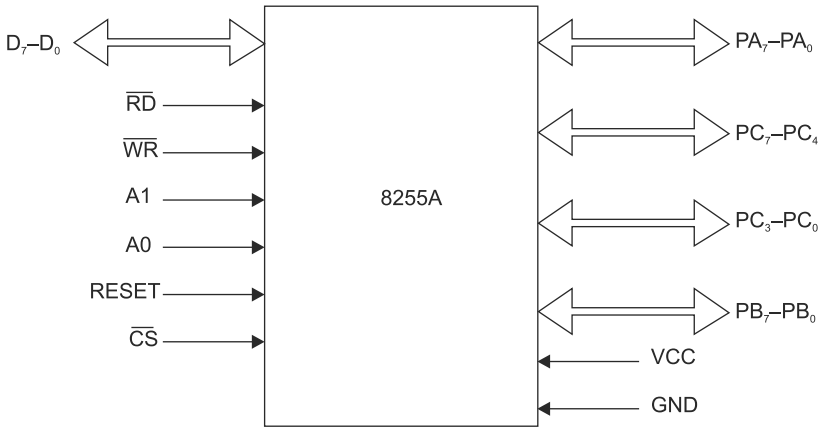


Fig. 12.1 Schematic pin diagram of 8255A

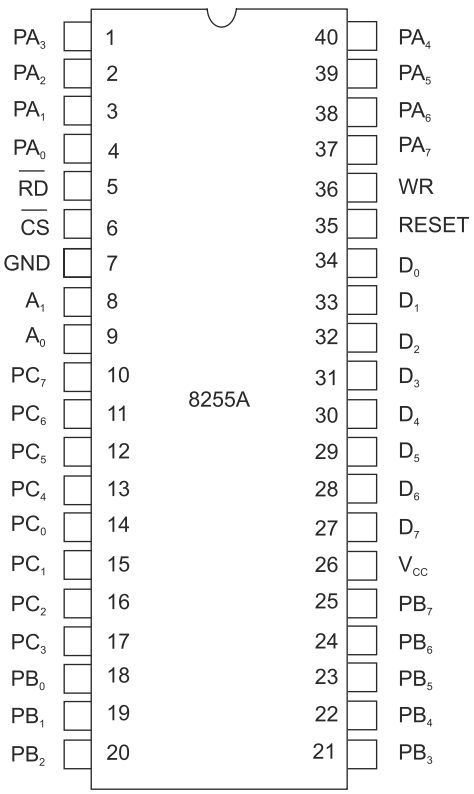


Fig. 12.2 Pin diagram of Intel 8255A

- *Control Group A* Port A and Port C upper ($PC_7 - PC_4$)
- *Control Group B* Port B and Port C lower ($PC_3 - PC_0$)

The control word register can be both read and write as shown in the 'Basic Operation' Table 12.1. The

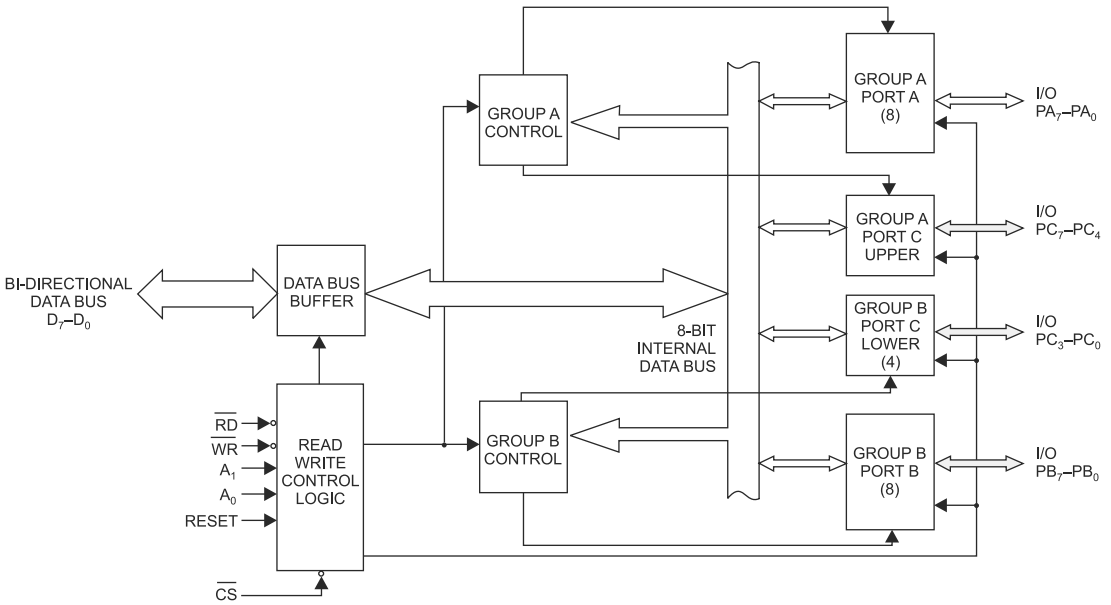


Fig. 12.3 Functional block diagram of 8255A

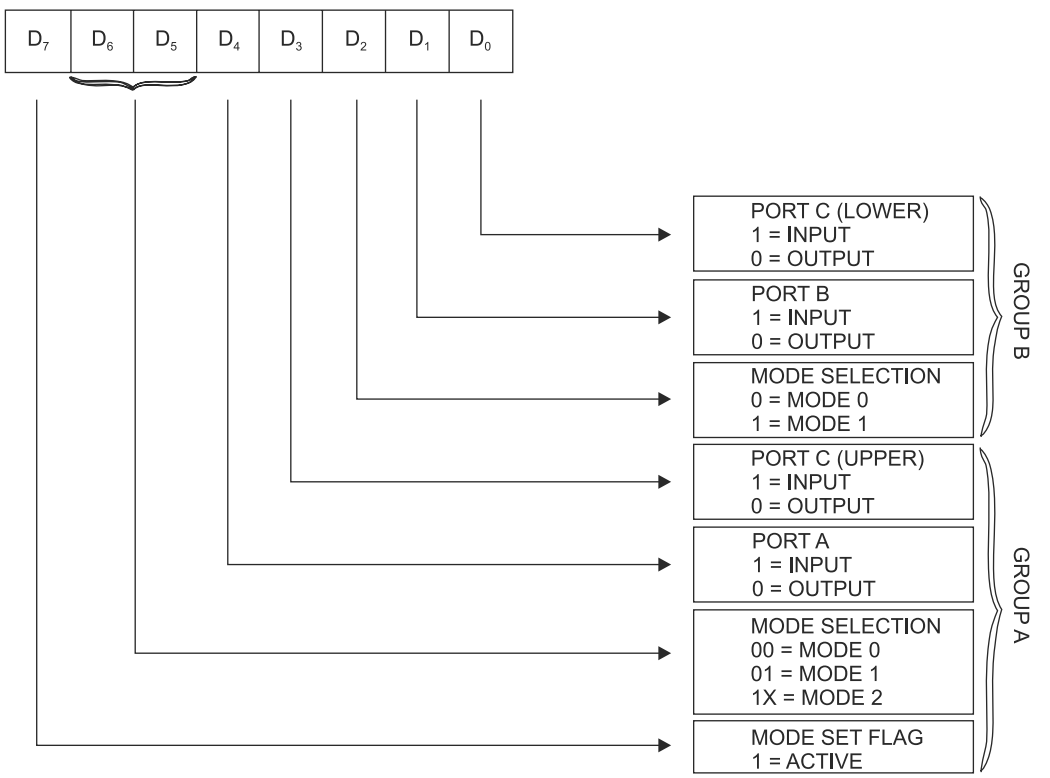


Fig. 12.4 Control word bits of 8255A

control word format for both Read and Write operations is depicted in Fig. 12.4. When the control word is read, bit D_7 will always be a logic '1', as this implies control word mode information.

Table 12.1(a) 82C55A basic input operation

A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	Input Operation (READ Cycle)
0	0	0	1	0	Port A to Data Bus
0	1	0	1	0	Port B to Data Bus
1	0	0	1	0	Port C to Data Bus
1	1	0	1	0	Control Word to Data Bus

Table 12.1(b) 82C55A basic output operation

A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	Output Operation (WRITE)
0	0	1	0	0	Data Bus to Port A
0	1	1	0	0	Data Bus to Port B
1	0	1	0	0	Data Bus to Port C
1	1	1	0	0	Data Bus to Control

Table 12.1(c) 82C55A disable operation

A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	Disable Function
X	X	X	X	1	Data Bus to Three-State
X	X	1	1	0	Data Bus to Three-State

12.4 OPERATING MODES

There are three basic modes of operation of 8255 as follows:

- **Mode 0** Basic Input/Output
- **Mode 1** Strobed Input/Output
- **Mode 2** Bi-directional Bus

The system software can select the mode of operation. When the reset input becomes 'high', all ports will be set to the input mode with all 24-port lines held at logic 'one' level by internal bus hold devices. When the reset is removed, the 82C55A can remain in the input mode with no additional initialization required. This eliminates the need to pull-up or pull-down resistors in all-CMOS designs. Then control word register will contain 9B H. During the execution of the system program, any of the other modes may be selected using a single output instruction. This allows a single 82C55A to service a variety of peripheral devices with a simple software maintenance routine. Any port programmed as an output port is initialized to all zeros when the control word is written.

The 8255 has two 8-bit ports (Port A and Port B) and two 4-bit ports (Port C_{upper} and Port C_{lower}). The modes for Port A and Port B can be separately defined, though Port C is divided into two portions as required by the Group A and Group B definitions.

12.4.1 Mode 0—Basic Input/Output

This functional configuration provides simple input and output operations for each of the four ports (Port A, Port B, Port C_{upper} and Port C_{lower}). Each of the four ports of 8255 can be programmed to be either an input or output port. No handshaking is required; data is simply written to or read from a specific port.

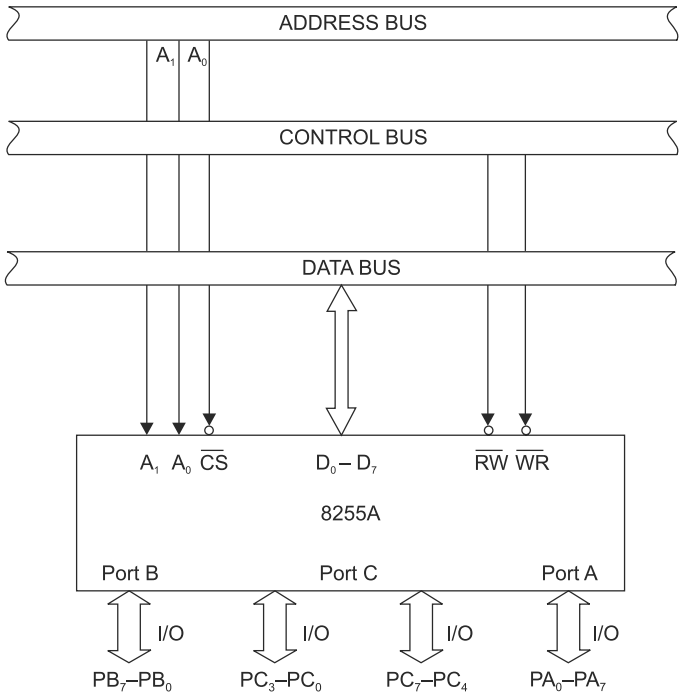


Fig. 12.5 Mode 0

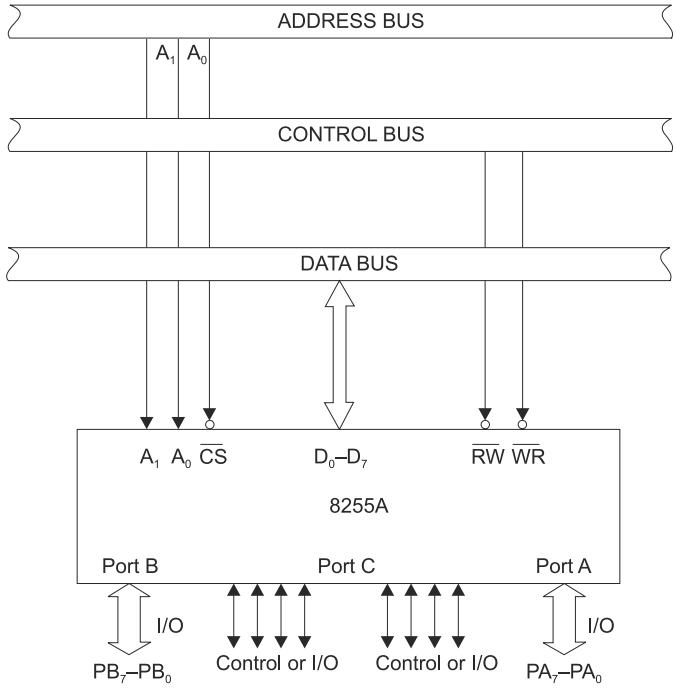


Fig. 12.6 Mode 1

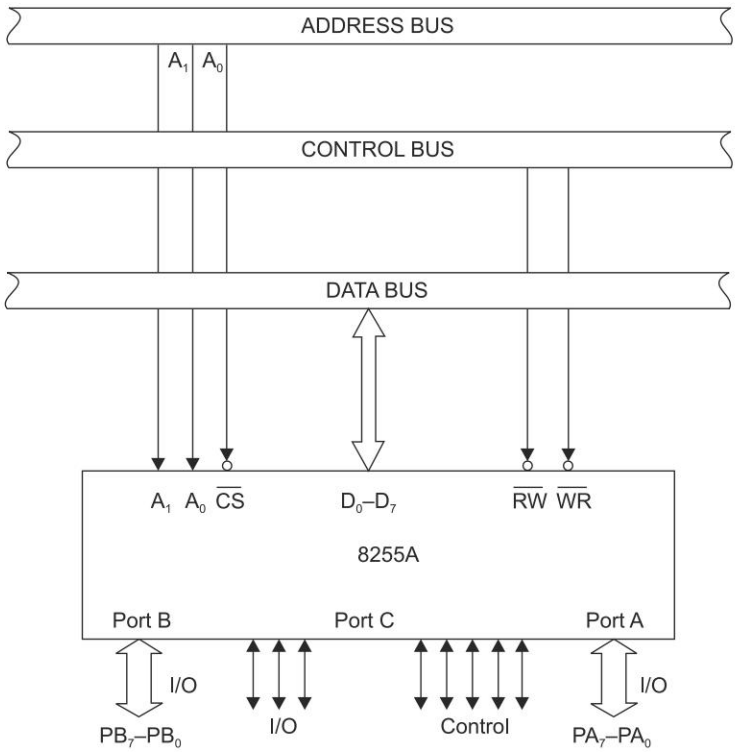


Fig. 12.7 Mode 2

Basic functional definitions of Mode 0 as follows:

- Two 8-bit ports and two 4-bit ports
- Any port can be input or output
- Outputs are latched
- Inputs are not latched
- 16 different Input/Output configurations possible

12.4.2 Mode 1—Strobed Input/Output

This is strobed input/output mode of operation. Only Port A and Port B both can be operating in this mode of operation. When Port A and Port B are programmed in Mode 1, six pins from Port C are used as control signals. These control signals are used for handshaking. PC_0, PC_1 and PC_2 pins of PC lower are used to control the Port B and PC_3, PC_4, and PC_5 pins of PC upper are used to control Port A. The other pins of Port C, i.e., PC_6 and PC_7 can be used as either input or output. While Port A is operated as an output port, pins PC_3, PC_6 and PC_7 are used for its control. The pins PC_4 and PC_5 can be used either as input or output. The combination of Mode 0 and Mode 1 operation is also possible. When Port A is programmed to operate in Mode 1, the Port B can also be operated in Mode 0. Figure 12.8 shows the timing diagram of mode 1 (input).

This functional configuration provides a means for transferring I/O data to or from a specified port in conjunction with strobes or handshaking signals. In mode 1, Port A and Port B use the lines on the Port C to generate or accept these handshaking signals. In Mode 1, the 8255 has two functional groups, namely, Group A and Group B. Each group contains one 8-bit port and a 4-bit control/data port. The 8-bit data port can be

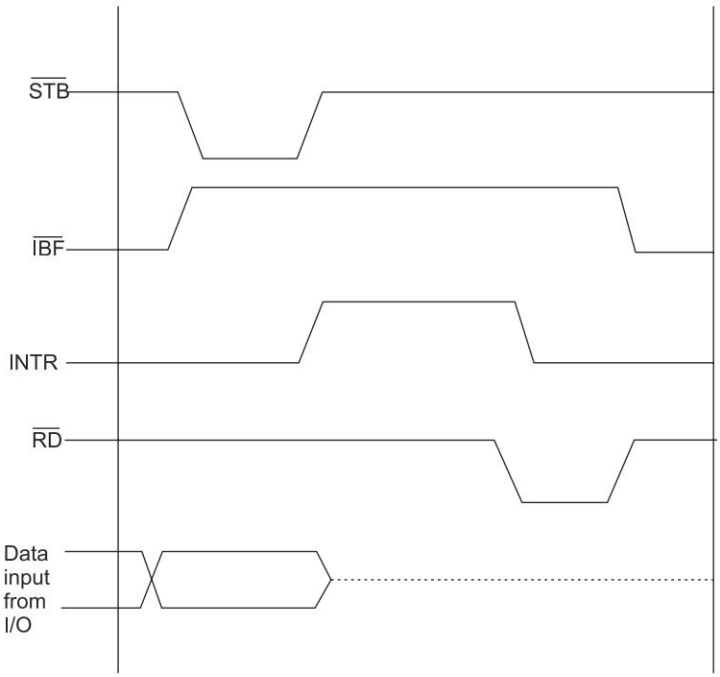


Fig. 12.8 Timing diagram of Mode 1

either input or output. Both inputs and outputs are latched. The 4-bit port can be used for control and status of the 8-bit port. Figure 12.9 shows the mode 1 operation of Port A and Port B as input port. Figure 12.10 shows the mode 1 operation of Port A and Port B as output port.

Input Control Signal

STB (Strobe Input) A low on this input loads data into the input latch.

IBF (Input Buffer Full F/F) A high on this output indicates that the data has been loaded into the input latch: in essence, and acknowledgment. IBF is set by $\overline{STB}$ input being low and is reset by the rising edge of the $\overline{RD}$ input.

INTR (Interrupt Request) A 'high' on this output can be used to interrupt the CPU when input device is requesting service. INTR is set by the condition: $\overline{STB}$ is a 'one', IBF is a 'one' and INTE is a 'one'. It is reset by the falling edge of $\overline{RD}$. This procedure allows an input device to request service from the CPU by simply strobing its data into the port.

INTE A Controlled by bit set/reset of PC_4.

INTE B Controlled by bit set/reset of PC_2.

Output Control Signal The $\overline{OBF}$ Output Buffer Full F/F output will go 'low' to indicate that the CPU has written data out to be specified port. This does not mean valid data is sent out of the port at this time since $\overline{OBF}$ can go true before data is available. Data is guaranteed valid at the rising edge of $\overline{OBF}$. The $\overline{OBF}$ F/F will be set by the rising edge of the $\overline{WR}$ input and reset by $\overline{ACK}$ input being low.

$\overline{ACK}$ *(Acknowledge Input)* A 'low'on this input informs the 82C55A that the data from Port A or Port B is ready to be accepted. In essence, a response from the peripheral device indicating that it is ready to accept data.

INTR (Interrupt Request) A 'high' on this output can be used to interrupt the CPU when an output device has been accepted data transmitted by the CPU. INTR is set when $\overline{ACK}$ is a 'one', OBF is a 'one' and INTE is a 'one'. It is reset by the falling edge of $\overline{WR}$.

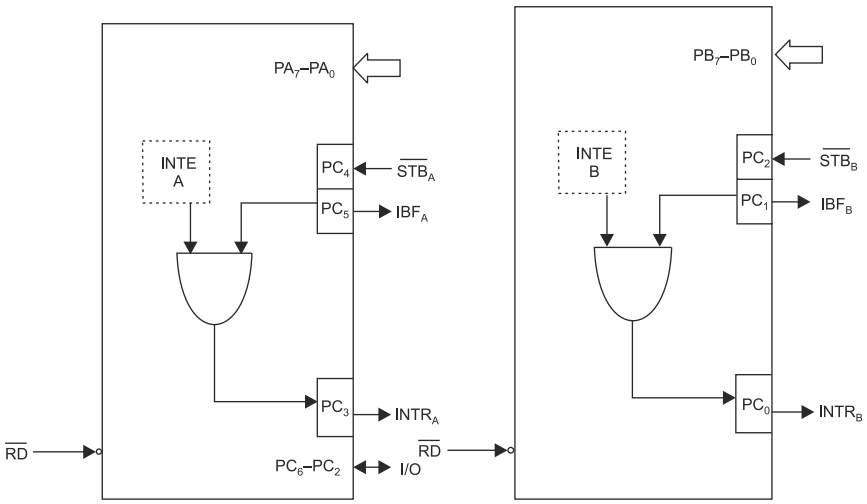


Fig. 12.9 Input mode of ports A and B in Mode 1

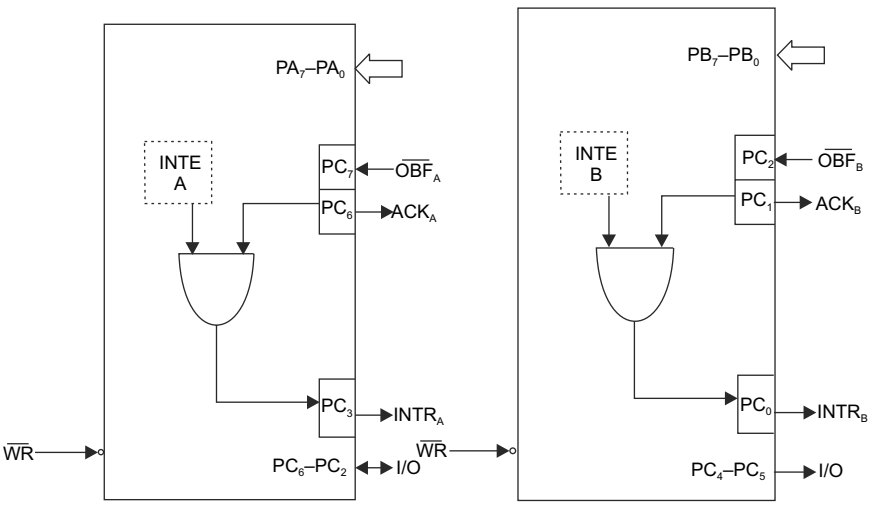


Fig. 12.10 Output of ports A and B in Mode 1

INTE A Controlled by Bit Set/Reset of PC_6.

INTE B Controlled by Bit Set/Reset of PC_2.

The strobe line is in a handshaking mode. The user needs to send $\overline{OBF}$ to the peripheral device, generates an $\overline{ACK}$ from the peripheral device and then latch data into the peripheral device on the rising edge of $\overline{OBF}$. The timing diagram of 8255 in the mode 1 operation of Port A and Port B as output port is illustrated in Fig. 12.11.

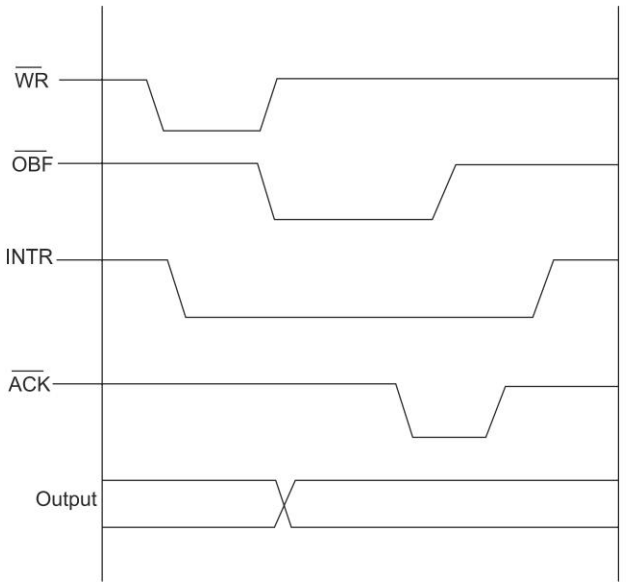


Fig. 12.11 Timing diagram of Mode 1 with Port A and Port B as output port

12.4.3 Mode 2—Bi-directional Bus

This mode is strobed bi-directional of operation of port with input and output capability. Mode 2 operation is only feasible for Port A. Therefore, Port A can be programmed to operate as a bi-directional port. If Port A is programmed in Mode 2, Port B can be used in either Mode 1 or Mode 0. In this mode of operation, PC_3 to PC_7 pins are used to control signals of Port A.

The basic functional definitions of Mode 2 are

- Used in Group A only
- One 8-bit, bi-directional bus port (Port A) and a 5-bit control port (Port C)
- Both inputs and outputs are latched
- The 5-bit control port (Port C) is used for control and status for the 8-bit, bi-directional bus port (Port A)

Bi-Directional Bus I/O Control Signal

INTR (Interrupt Request) A 'high' on this output can be used to interrupt the CPU for both input or output operations.

Output Operations

OBF (Output Buffer Full) The OBF output will go 'low' to indicate that the CPU has written data out to Port A.

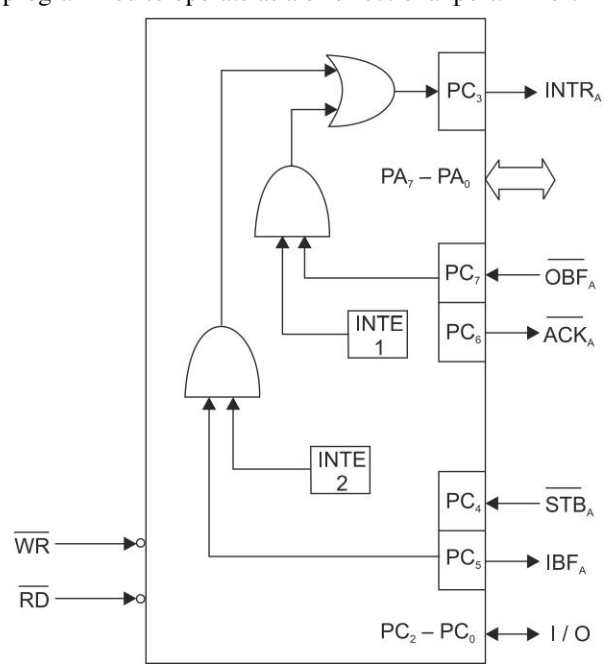


Fig. 12.12 Mode 2

ACK (Acknowledge) A 'low' on this input enables the three-state output buffer of Port A to send out the data. Otherwise, the output buffer will be in the high impedance state.

INTE 1 (INTE flip-flop associated with OBF) Controlled by bit set/reset of PC_4.

Input Operations

STB (Strobe Input) A 'low'on this input loads data into the input latch.

IBF (Input Buffer Full F/F) A 'high' on this output indicates that data has been loaded into the input latch.

INTE 2 (INTE flip-flop associated with IBF) Controlled by bit set/reset of PC_4.

The timing diagram of Mode 2 operation of 8255 is depicted in Fig. 12.13.

12.4.4 Single Bit Set/Reset (BSR) Mode

In this mode, any of the eight bits of Port C can be set or reset using a single output instruction. This feature reduces software requirements in control-based applications.

When Port C is being used as status/control for Port A or B, these bits can be set or reset by using the bit set/reset operation just as if they were output ports. Figure 12.14 shows the bit set/reset format.

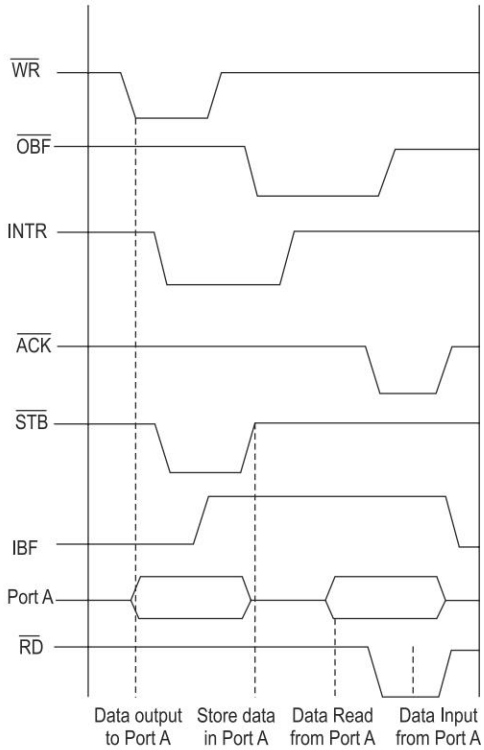


Fig. 12.13 Timing diagram of Mode 2

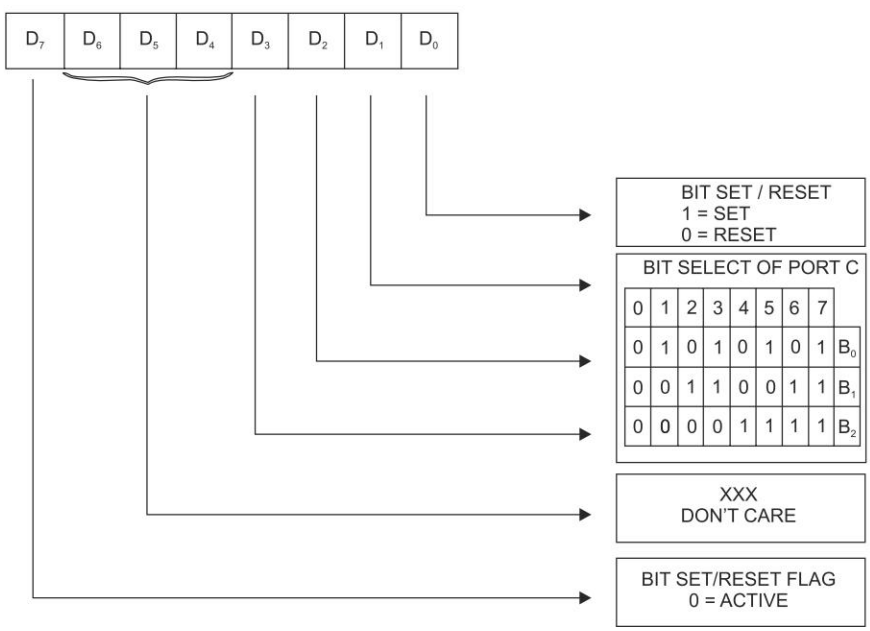


Fig. 12.14 Bit set/reset format

12.5 CONTROL WORD

The ports of 8255 can be operating any one mode by programming the internal register of 8255. This internal register of 8255 PPI is known as Control Word Register (CWR). To program the ports of 8255, a control word is formed. Figure 12.4 shows the bits of the control word in Section 12.3. Only write operation of the control word register is permissible and no read operation of the control word register is allowed. Writing the control word into the control word register, the IC will be configured to operate specified modes of operation. The functional description of the bits of the control word is as follows:

Bit D_0 The D_0 bit is used to set the Port C_{lower}. When this bit is set to 1, Port C_{lower} is an input port. If the bit is set to 0, Port C_{lower} is an output port.

Bit D_1 This bit is used for Port B. When this bit is set to 1, Port B is an input port. If the bit is set to 0, Port B is an output port.

Bit D_2 The bit D_2 is used for the selection of the mode operation of Port B. If this bit is set to 0, the Port B can be operating in Mode 0. For Mode 1 operation, D_2 is set 1.

Bit D_3 It is used for the Port C_{upper}. If the bit is set to 1, Port C_{upper} is an input port. When the bit is set to 0, Port C_{upper} is an output port.

Bit D_4 The bit D_4 sets the port A for input or output operation. When this bit is 1, the port A can be used as input port. When it is 0, the port A becomes the output port.

Bit D_5 and D_6 These bits are used to select the operating mode of Port A. The Port A can be operated in modes 0, 1 and 2. The mode of operation is selected by D_5 and D_6 as given below:

Mode of Port A	Bit D_6	Bit D_5
Mode 0	0	0
Mode 1	0	1
Mode 2	1	0 or 1

For Mode 2, Bit 5 is set to either 0 or 1; it is immaterial.

Bit D_7 This bit select the I/O mode or bit set/reset mode. When it is 1, Port A, B and C are defined as input/output ports. If it is set to 0, bit set/reset mode is selected.

Table 12.2 Control words of 8255 for Mode 0 operation

Control word bits								Control word	Port A	Port C upper	Port B	Port C lower
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0					
1	0	0	1	1	0	1	1	9B	input	input	input	input
1	0	0	1	1	0	1	0	9A	input	input	input	output
1	0	0	1	1	0	0	1	99	input	input	output	input
1	0	0	1	1	0	0	0	98	input	input	output	output
1	0	0	1	0	0	1	1	93	input	output	input	input
1	0	0	1	0	0	1	0	92	input	output	input	output
1	0	0	1	0	0	0	1	91	input	output	output	input
1	0	0	1	0	0	0	0	90	input	output	output	output

1	0	0	0	1	0	1	1	8B	output	input	input	input
1	0	0	0	1	0	1	0	8A	output	input	input	output
1	0	0	0	1	0	0	1	89	output	input	output	input
1	0	0	0	1	0	0	0	88	output	input	output	output
1	0	0	0	0	0	1	1	83	output	output	input	input
1	0	0	0	0	0	1	0	82	output	output	input	output
1	0	0	0	0	0	0	1	81	output	output	output	input
1	0	0	0	0	0	0	0	80	output	output	output	output

12.6 EXAMPLES TO DETERMINE THE CONTROL WORD

The determination of control word bit corresponding to a particular port is set to either 1 or 0 depending upon the definition of the port, whether it is to be made an input port or output port. If a particular port is to be made an input port, the bit corresponding to that port is set to 1. For making a port an output port, the corresponding bit for the port is set to 0.

The control words for various configurations of the ports of 8255 for Mode 0 operation are illustrated in Table 12.2. The following examples will illustrate how to make control words.

Example 12.1 Determine control words when the ports of Intel 8255 are defined as follows:

- Port A as an input port. Mode of Port A is Mode 0.
- Port B as an input port. Mode of Port B is Mode 0.
- Port C_{upper} and C_{lower} are input ports.

Sol. The control word bits for the above definition of ports are as shown in Fig. 12.15.

Bit D_0 is set to 1, as Port Clower is an input port.
Bit D_1 is set to 1, as Port B is an input port.
Bit D_2 is set to 0, as Port B has to operate in Mode 0.
Bit D_3 is set to 1, as Port Cupper is an input port.
Bit D_4 is set to 1, as Port A is an input port.
Bit D_5 and D_6 are set to 00 as Port A has to operate in Mode 0.
Bit D_7 is set to 1, as Ports A, B and C are used as simple input/output port.
Thus the control word for above operation is 9B H.

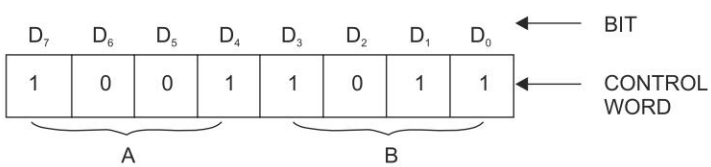


Fig. 12.15

Example 12.2 Find the control word for the following configuration of the ports of Intel 8255 for Mode 0 operation:

Port A – input, Port B – output, Port C_{lower} – output and Port C_{upper} – input

Sol. The control word bits for the above configuration of the ports are as shown in Fig. 12.16. The control word for the above definition of the ports of Intel 8255 is 98 H.

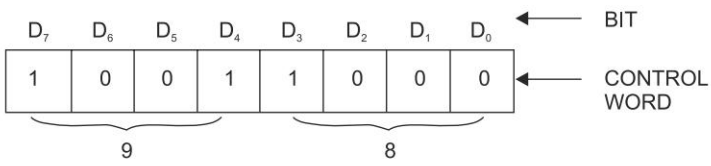


Fig. 12.16

Example 12.3 Make control word for the Intel 8255 with the following Mode 0 operations:

Port A – input, Port B – output, Port C_{upper} – output and Port C_{lower} – output

Sol. The control word bits for the above configuration of the ports are shown in Fig. 12.17.

The control word for the above definition of the ports of Intel 8255 is 90H.

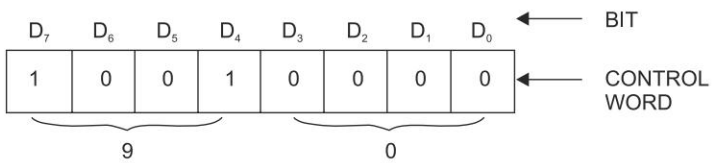


Fig. 12.17

Example 12.4 Determine control word the following configuration of the ports of Intel 8255 for Mode 1 operation:

- Port A is used as input and operation mode of Port A is Mode 1.
- Port B can be used as output and operates in Mode 1.
- PC_6 and PC_7 act as input.

Sol. Six pins of the Port C, PC_0–PC_5 are used to control Port A and Port B in Mode 1 operation. PC_0–PC_2 are used for the control of Port B. Port B can be programmed as an input or output port. When Port A is operated as an input port, PC_3–PC_5 are used to control this port. In this operating mode, PC_6 and PC_7 may be used as input or output. Fig. 12.18 shows the control word bits for the above configuration of the ports. The control word for the above definition of the ports of Intel 8255 is BD H.

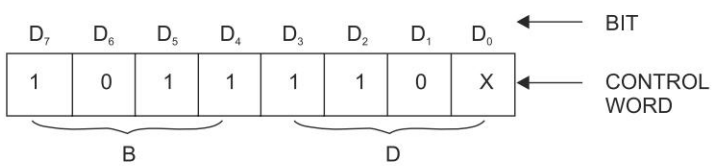


Fig. 12.18

Example 12.5 Find the control word for the following configuration of the ports of Intel 8255A for Mode 1 operation:

- Port A – output and Mode of Port A – Mode 1
- Port B – output and Mode of Port B –Mode 1

Remaining pins of the Port C, i.e. PC_4 and PC_5 are used as input

Sol. In this mode of operation, PC_0, PC_1 and PC_2 are used for the control of Port B. If Port A is used as an output port. PC_3, PC_6 and PC_7 are used for controlling Port A. PC_4 and PC_5 are available to be used either as input or output. The control word bits for the above configuration of the ports are shown in Fig. 12.19. The control word for the above operation of the ports of Intel 8255 is ADH.

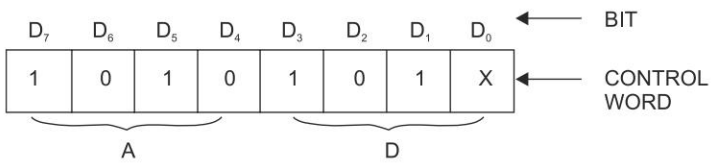


Fig. 12.19

12.7 APPLICATIONS OF 8255 PPI

The 8255 PPI IC is a very powerful tool for interfacing peripheral equipment to the microprocessor-based system. It represents the optimum use of available pins and is flexible enough to interface almost any I/O device without the need for additional external logic.

Each peripheral device in a microprocessor-based system usually has a 'service routine' associated with it. The routine manages the software interface between the device and the microprocessor. The functional definition of the 8255 can be programmed by the I/O service routine and becomes an extension of the system software. By examining the I/O device interface characteristics for both data transfer and timing, and matching this information to the examples and tables in the detailed operational description, a control word must be developed and loaded into control word register to initialize the 8255 IC to get specified operation. The typical applications of the 8255 are given below:

- Put on LED as specified by the designer
- Generate a square wave at Port A
- Interfacing A/D converter
- Keyboard operation
- Sequential switch ON lights
- Traffic light control
- Interfacing with dc motors and stepper motors, etc.

12.8 8255 INTERFACING WITH 8085 MICROPROCESSOR

Figure 12.20 shows the 8255 interfacing with 8085 microprocessor. A_0 and A_1 address lines of 8085 microprocessor are directly connected to the A_0 and A_1 pins of 8255 IC for selecting three ports such as Port A, Port B, port C and control word register address. $\overline{RD}$, $\overline{WR}$, RESET and data bus D_0–D_7 of 8255 is connected with the 8085 microprocessor. The chip select signal $\overline{CS}$ is generated from A_2 to A_7 address lines and $IO/\overline{M}$ as depicted in Fig. 12.20. The addresses of Port A, Port B, Port C and control word register are given in Table 12.3. The operating modes of 8255A are discussed in Section 12.4 in detail.

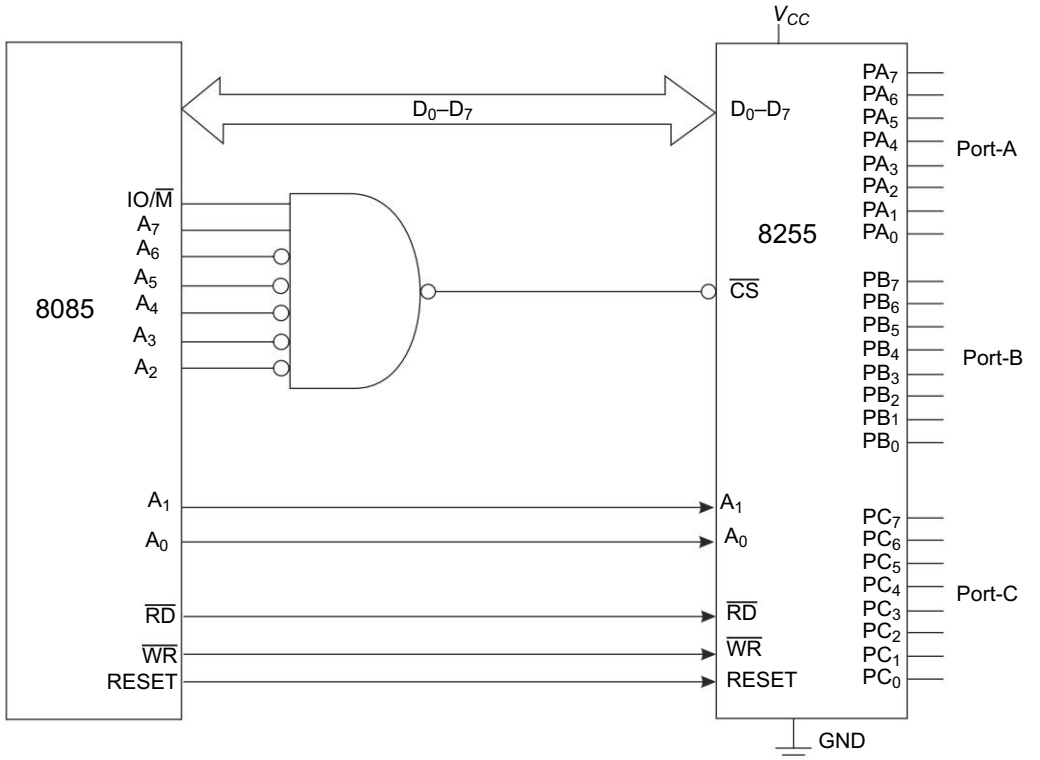


Fig. 12.20 8255A interface to 8085 microprocessor

Table 12.3 Address of ports and CWR

Ports and CWR	Status of I/O Address lines								Address in Hex
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
Port A	1	0	0	0	0	0	0	0	80H
Port B	1	0	0	0	0	0	0	1	81H
Port C	1	0	0	0	0	0	1	0	82H
CWR	1	0	0	0	0	0	1	1	83H

When all ports operate in Mode 0 and also operate as output port, the control word is 80H. If we want to send 20H in Port A, 30H in Port B and 40H in Port C, the following instructions will be executed.

MVI A, 80H	Load control word 80H for all ports operate in Mode 0 as output port
OUT 83H	Write the control word 80H in control word register
MVI A, 20H	Load 20H data in Accumulator
OUT 80H	Send 20H in port A and 20H will be output at Port A
MVI A, 30H	Load 30H data in Accumulator
OUT 81H	Send 30H in port B and 30H will be output at Port B
MVI A, 40H	Load 40H data in Accumulator
OUT 82H	Send 40H in port A and 40H will be output at Port C
HLT	Stop

12.9 8255 INTERFACING WITH 8086 MICROPROCESSOR

Figure 12.21 shows 8255 interfacing with 8086 microprocessor, In this 8086 microprocessor interfacing, 16 address lines are used to drive 8255 IC. A_1 and A_2 address lines of 8086 microprocessor are directly connected to the A_0 and A_1 pins of 8255 IC for selecting Port A, Port B, Port C and control word register address. The other lines A_0, A_3–A_7 and A_8–A_{15} are used for decoding and generate the chip select signal.

The 8055 is directly interfaced with the lower order data bus D_7–D_0. The A_0 and A_1 pins of 8255 are connected with A_1 and A_2 of 8086 microprocessor. In Fig.12.21, it is assumed that the 8086 operate in the minimum mode so that $\overline{IORD}$ and $\overline{IOWR}$ are connected with $\overline{RD}$ and $\overline{WR}$ pins of 8255 and the chip select pin of 8255 is connected with the decoder circuit output.

When 8255 interfaces with 8086 microprocessor as depicted in Fig. 12.21, the port address of Port A, Port B, Port C and the addresses of the control word register are 0740H, 0742H, 0744H and 0746H respectively. To generate the above address, the status of pins A_0–A_{15} is given in Table 12.4.

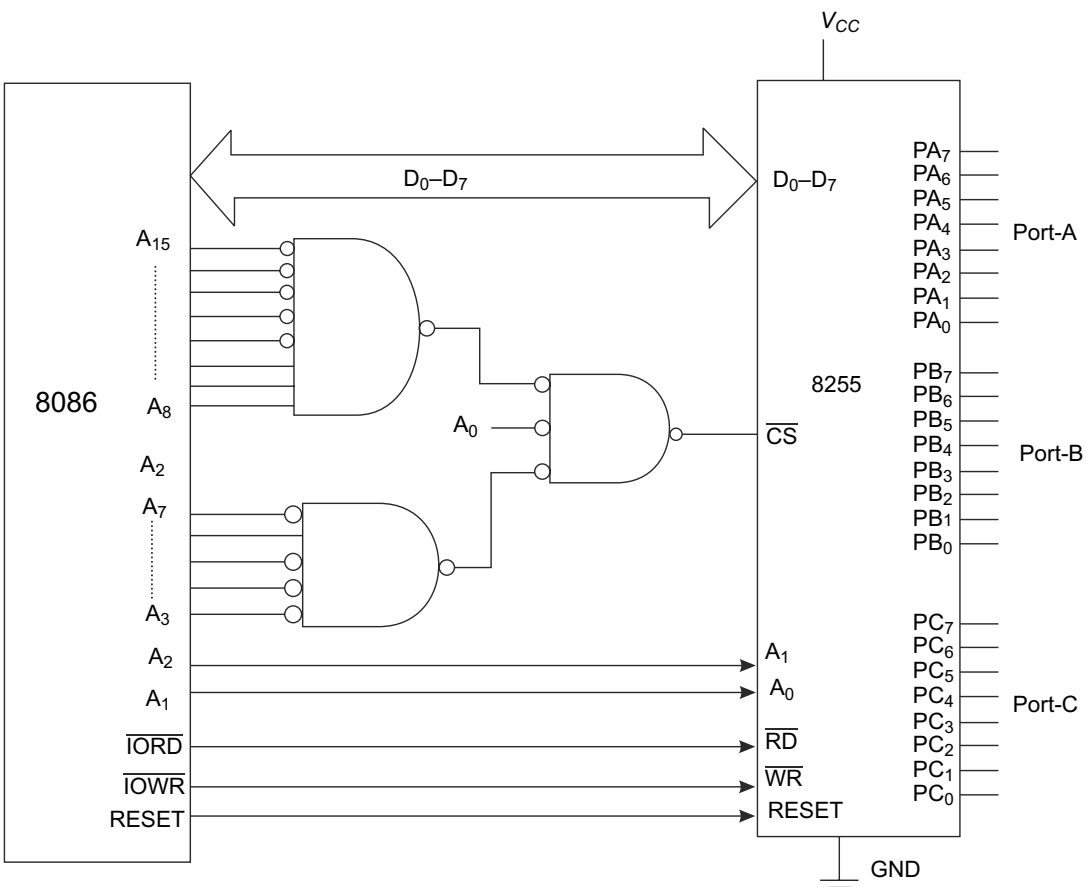


Fig. 12.21 8255A interface to 8086 microprocessor

Table 12.4 Address of ports and CWR

Ports and CWR	Status of I/O Address lines																Address in Hex
	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	
Port A	0	0	0	0	0	1	1	1	0	1	0	0	0	0	0	0	0740H
Port B	0	0	0	0	0	1	1	1	0	1	0	0	0	0	1	0	0742H
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

When Port A, Port B and Port C of 8255 operate as output ports and in Mode 0, the control word is 80H. If we want to send 77H to port C, 55H to Port B and 44H to Port A, the following instructions will be executed.

MOV DX, 0746H	Load control word register address 0746H in DX register
MOV AL, 80H	Load control word 80H in AL register
OUT DX, AL	Load control word 80H in control word register
SUB DX, 02	Get Port C address in DX
MOV AL, 77H	Load 77H in AL
OUT DX, AL	Out 77H in Port C
SUB DX, 02	Get Port B address in DX
MOV AL, 55H	Load 55H in AL
OUT DX, AL	Out 55H in Port B
SUB DX, 02	Get Port A address in DX
MOV AL, 44H	Load 44H in AL
OUT DX, AL	Out 44H in Port C
HLT	Stop

In 8086 microprocessor interfacing, when 8 address lines are used to drive 8255, A_1 and A_2 address lines of 8086 microprocessor are directly connected to the A_0 and A_1 pins of 8255 IC for selecting the address of three ports A, B, C and the control word register. The other lines A_0, and A_3–A_7 are used for decoding and generate chip select $\overline{CS}$ signal. Figure 12.22 shows the 8255 interfacing with 8086 microprocessor where the address of Port A is 80H. The D_7–D_0 of 8255 is directly interfaced with the lower order data bus D_7–D_0.

Neglecting the higher order address lines, the addresses of Port A, Port B, Port C and the address of control word register are 80H, 82H, 84H and 86H respectively. To generate the above addresses, the status of A_0–A_7 pins is given in Table 12.5

Table 12.5 Address of ports and CWR

Ports and CWR	Status of I/O Address lines								Address in Hex
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
Port A	1	0	0	0	0	0	0	0	80H
Port B	1	0	0	0	0	0	1	0	82H
Port C	1	0	0	0	0	1	0	0	84H
CWR	1	0	0	0	0	1	1	0	86H

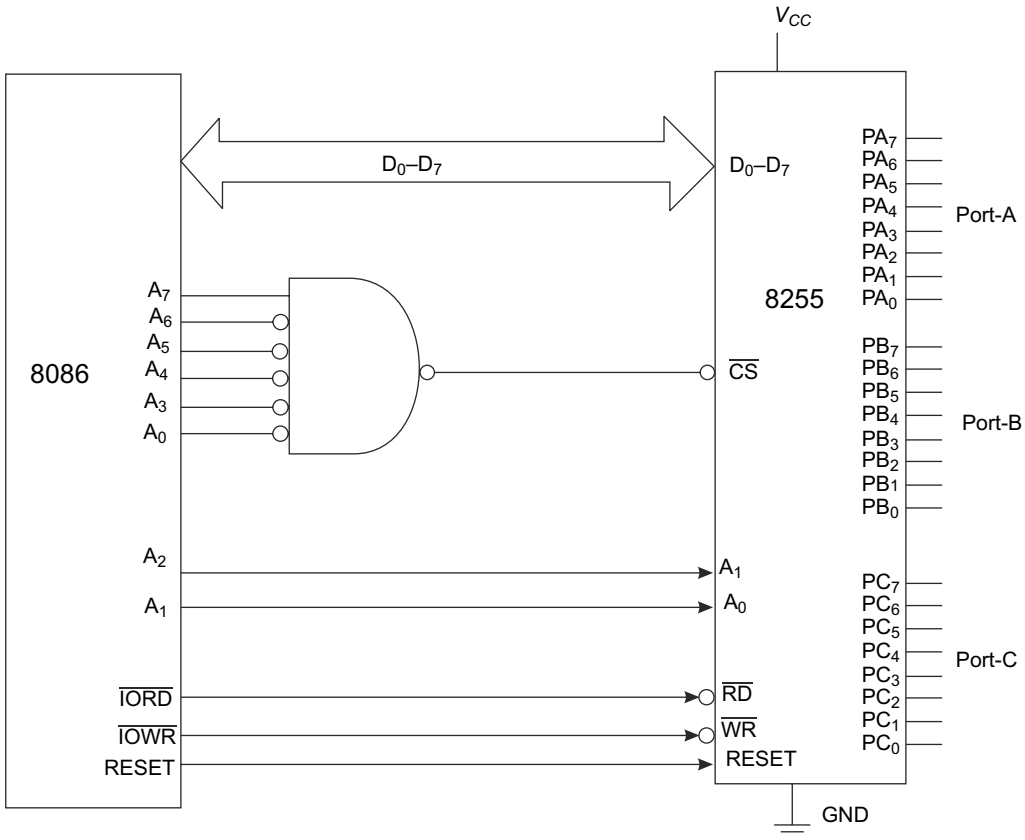


Fig. 12.22 8255A interface to 8086 microprocessor

12.10 8255 INTERFACING WITH 8051 MICROCONTROLLER

Figure 12.23 shows the interfacing of 8255 with 8051 microcontroller. A_0 and A_1 address lines of 8051 microcontroller are directly connected to the A_0 and A_1 pins of 8255 IC for selecting the address of Port A, Port B, Port C and the control word register. The chip select $\overline{CS}$ signal is generated from A_2 to A_7 and A_8 to A_{15} address lines. When the address of control word register is E803H, the addresses of Port A, Port B, Port C are E800H, E801H and E802H, respectively. To generate the above addresses, the status of pins A_0–A_{15} is given in Table 12.6.

Table 12.6 Address of ports and CWR

Ports and CWR	Status of I/O Address lines																Address in Hex
	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	
Port A	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	E800H
Port B	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1	E801H
Port C	1	1	1	0	1	0	0	0	0	0	0	0	0	0	1	0	E802H
CWR	1	1	1	0	1	0	0	0	0	0	0	0	0	0	1	1	E803H

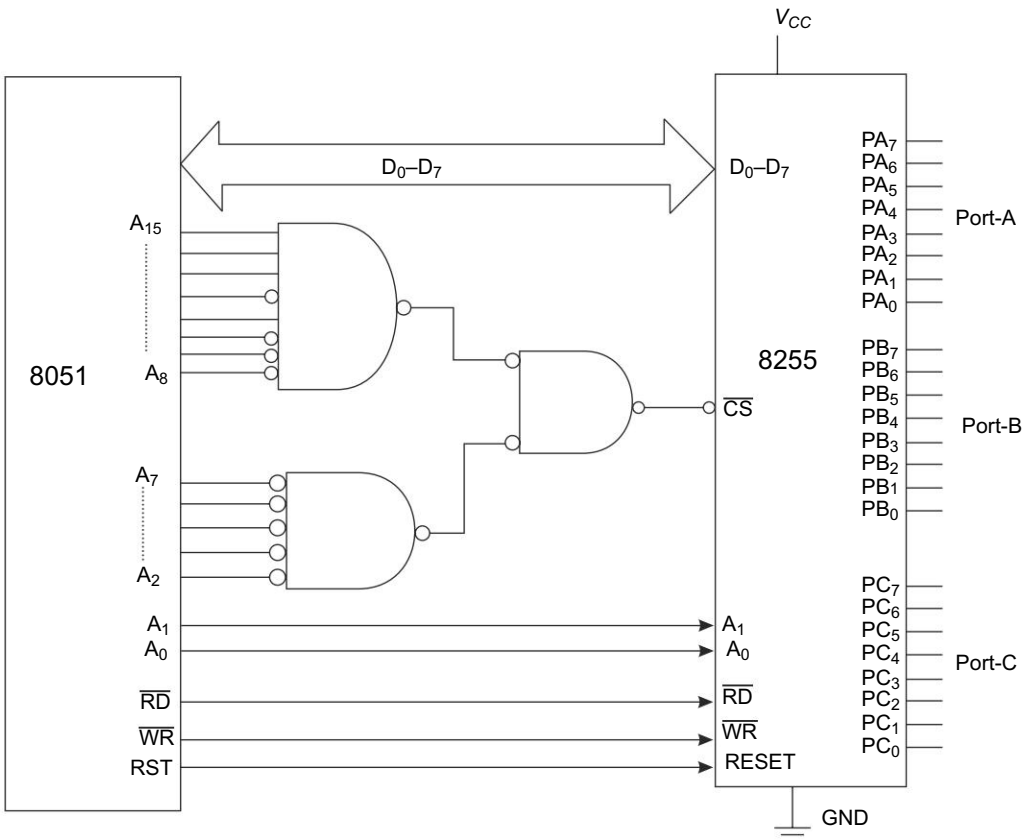


Fig. 12.23 8255A interface to 8051 microcontroller

When Port A, Port B and Port C of 8255 operate as output ports in Mode 0, the control word is 80H. If we want to send 22H to Port A, 55H to Port B and 88H to Port C, the following instructions will be executed.

MOV A, #80H	Load control word 80H in Accumulator
MOV DPTR, E803H	Load address of control word register E803H in DPTR
MOVX @DPTR, A	Load control word 80H in control word register
MOV A, #22H	Load 22H in accumulator
MOV DPTR, E800H	Load address of Port A E800H in DPTR
MOVX @DPTR, A	Out 22H in Port A
INC DPTR	Load address of Port B E801H in DPTR
MOV A, #55H	Load 55H in accumulator
MOVX @DPTR, A	Out 55H in Port B
INC DPTR	Load address of Port B E802H in DPTR
MOV A, #88H	Load 88H in accumulator
MOVX @DPTR, A	Out 88H in Port C
LJMP 00	

In 8051 microcontroller interfacing, when 8 address lines are used to drive 8255, A_0 and A_1 address lines of 8051 microcontroller are directly connected to the A_0 and A_1 pins of 8255 IC to select the address of Port A, Port B, Port C and control word register. The A_2–A_7 are used for decoding and generate chip select $\overline{CS}$ signal. The D_7–D_0 of 8255 is directly interfaced with data bus D_7–D_0. Figure 12.24 shows the 8255 interfacing with 8051 microprocessor where the address of Port A is 80H. Neglecting the higher order address lines, the address of address of Port A, Port B, Port C and the address of control word register are 80H, 81H, 82H and 83H respectively. To generate the above address, the status of A_0–A_7 pins is given in Table 12.7.

Table 2.7 Address of ports and CWR

Ports and CWR	Status of I/O Address lines								Address in Hex
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
Port A	1	0	0	0	0	0	0	0	80H
Port B	1	0	0	0	0	0	0	1	81H
Port C	1	0	0	0	0	0	1	0	82H
CWR	1	0	0	0	0	0	1	1	83H

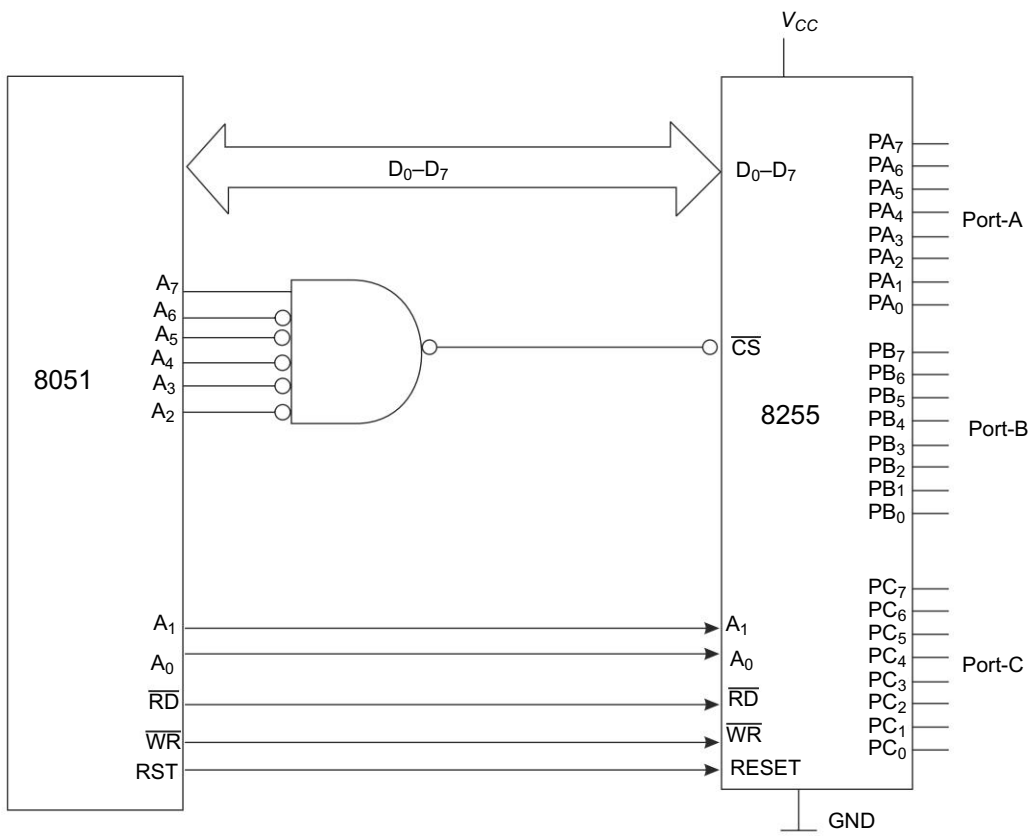


Fig. 12.24 8255A interface to 8051 microcontroller

Review Questions

12.1 Draw the functional block diagram of 8255 and explain the operation of each sub-block.

12.2 What are the different operating modes of 8255? Explain any one operating mode of 8255.

12.3 Explain different ports and control words of 8255.

12.4 Explain BSR mode of 8255. Discuss the control word format in the BSR mode.

12.5 Write a BSR control word to set bits PC_7 and PC_0 and to reset them after 1-second delay.

12.6 Write the control word format for I/O mode operation of 8255.

12.7 What happens when the RESET pin of 8255 is made high?

12.8 Explain the operation of 8255 PPI IC with its internal block diagram. Explain its mode 0, mode 1 and mode 2 operations.

12.9 Write control word in Mode 0 operation for the following cases:
(a) Port A = Input port, port B = output port, Port C = output port
(b) Port A = Input port, port B = output port, Port C_U = output port, Port C_L = input port

12.10 Draw Port A and the control signals when 8255 is operated in Mode 2. Explain Mode 2 operation with a timing diagram.

12.11 What are input and output control signals in Mode 2 and discuss them? Write applications of 8255.

12.12 Write a program to generate a square wave using 8255.

12.13 Explain Mode 1 operation and draw the timing diagram of Port A in Mode 1 operation.

12.14 Discuss the control signals when ports A and B act as output ports. Draw the timing diagram for strobed input.

12.15 Determine the control word for the following configuration of the ports of 8255:
- Port A—M output and Mode of Port A is Mode 1
- Port B—M output and Mode of Port B is Mode 1
- Remaining pins of Port C are used as input

12.16 (a) Draw the schematic diagram for 8255 interfacing with 8085 microprocessor and write the address of Port A, Port B, Port C and control word register.
(b) Write a program to send 2FH in Port A, 30H in Port B and 4FH in Port C for the above interfacing circuit.

12.17 (a) Design the 8255 interfacing with 8086 microprocessor when the address of Port A is 8740H
(b) Write an assembly language program to send 77H in Port A, 66H in Port B and 55H in Port C for the above interfacing circuit.

12.18 (a) Draw the 8255 interfacing with 8051 microcontroller and explain briefly.
(b) Write an assembly language program if we want to send 20H to Port A, 30H to Port B and 40H to Port C.

Multiple-Choice Questions

12.1 If A_0 and A_1 pins of 8255 are 00, which port will be selected ?
(a) Port A (b) Port B (c) Port C (d) None of these

12.2 To select Port B of 8255, A_0 and A_1 are

(a) 00 (b) 01 (c) 10 (d) None of these

12.3 When Port A is used as input, Port B and Port C are used as output. Then the control word of 8255 is

(a) 80H (b) 90H (c) 85H (d) 86H

12.4 When Port A, Port B and Port C are used as output ports, the control word of 8255 is

(a) 9BH (b) 90H (c) 89H (d) 80H

12.5 When Port A, Port B and Port C are used as input ports, the control word of 8255 is

(a) 80H (b) 90H (c) 9BH (d) 99H

12.6 In 8255 interfacing with 8086 microprocessor, if the address of Port A is 0740H, the address of control word register is

(a) 0742H (b) 0743H (c) 0744H (d) 0746H

12.7 In 8255 interfacing with 8086, A0 and A1 pins of 8255 are connected with ____ pins of 8086 respectively.

(a) A_1 and A_2 (b) A_0 and A_1 (c) A_0 and A_2 (d) A_0 and A_3

12.8 In 8255 interfacing with 8051 microcontroller, A0 and A1 pins of 8255 are connected with ____ pins of 8051 respectively.

(a) A_0 and A_1 (b) A_1 and A_0 (c) A_0 and A_2 (d) A_1 and A_2

Answers to Multiple-Choice Questions

12.1 (a)	12.2 (c)	12.3 (b)	12.4 (d)
12.5 (c)	12.6 (d)	12.7 (a)	12.8 (a)

CHAPTER

13

8253 Interfacing with 8085, 8086 and 8051 Microcontroller

13.1 INTRODUCTION

In the process control system or the automation industry, a number of operations are generally performed sequentially. Between two operations, a fixed time delay is always specified. In a microprocessor-based system, time delay can be generated using software. Sequences of operations are also performed based on software. Therefore, time delay, sequence and counting can be done under the control of a microprocessor. These most common problems can be solved using the 8253 in any microcomputer system.

The 8253 is a programmable interval timer/counter specifically designed for use in real-time application for timing and counting functions such as binary counting, generation of accurate time delay, generation of square waves, rate generation, hardware/software triggered strobe signal, one-shot signal of desired width, etc. The function of the 8253 timer is that of a general-purpose, multi-timing element which can be treated as an array of I/O ports in the system software.

The generation of accurate time delay using software control or writing instruction is possible. But instead of writing instructions for time delay loop, the 8253 timer may be used for this. The programmer configures the 8253 as per requirements. When the counters of the 8253 are initializing with the desired control word, the counter operates as per requirement. Then a command is given to the 8253 for counting the delay and it interrupts the CPU. At the instant it has completed its tasks, the output will be obtained from the output terminal. Multiple delays can easily be implemented by assignment of priority levels in a microprocessor.

Counter/timers such as a Programmable Rate Generator, Event Counter, Binary Rate Multiplier, Real Time Clock, Digital One-Shot, and Complex Motor Controller can also be used for non-delay in nature. The 8253 operates in the frequency range of dc to 2.6 MHz. The 8253 uses NMOS technology. The 8253 is compatible to the 8085 microprocessor. Generally, 8253 can operate in the following modes.

Mode 0 Interrupt on terminal count
Mode 1 Programmable one-shot
Mode 2 Rate generator
Mode 3 Square-wave generator
Mode 4 Software triggered mode
Mode 5 Hardware triggered mode

The pin diagram, block diagram of 8253, interfacing with 8085 microprocessor and operation of each mode have been explained in this section.

13.2 PIN DIAGRAM OF 8253

The 8253 timer is a 24-pin IC and operates at +5 V dc. It consists of three independent programmable16-bit counters: counter-0, counter-1, and counter-2. Each counter operates as a 16-bit down counter and each counter consists of clock input, gate input and output as depicted in Fig. 13.1. The schematic block diagram is given in Fig. 13.1. The gate input is used to enable the counting process. Therefore, the starting of counting may be controlled by external input pulse in gate terminal. After gate is triggered, the counter starts count-down. When the counter has completed counting, the output signal would be available at the output terminal.

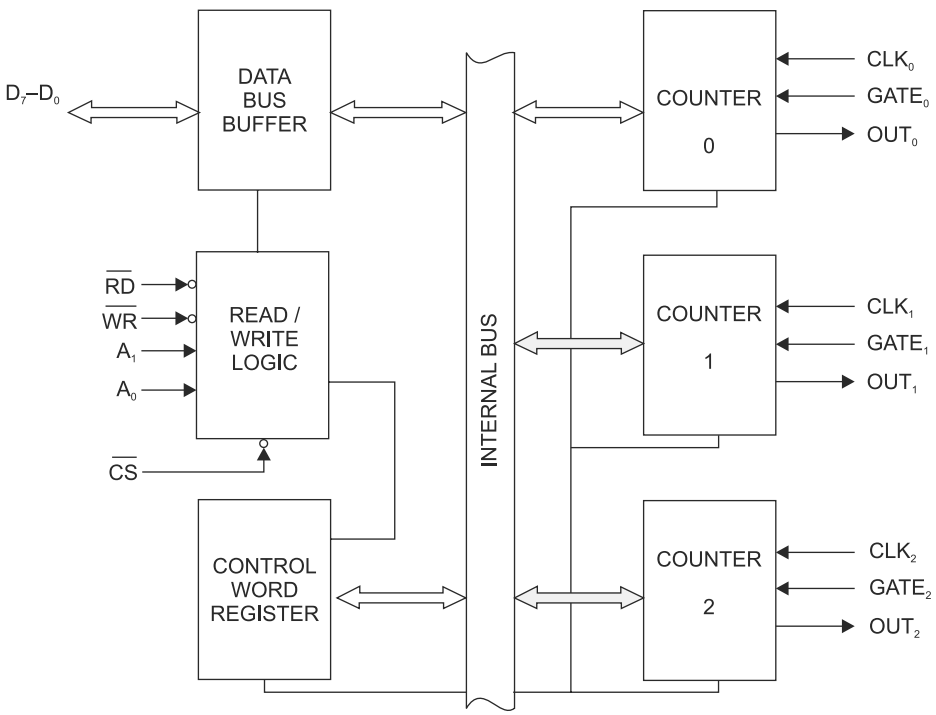


Fig. 13.1 Schematic block diagram of Intel 8253 timer/counter

The programmer can program 8253 using software in any one of the six operating modes: mode-0, mode-1, mode-2, mode-3, mode-4, and mode-5. The schematic pin diagram of 8253 is shown in Fig. 13.2 and Fig. 13.3. Shows the pin diagram of 8253. The functional descriptions of pins are as follows:

$\overline{RD}$ (Read) When this pin is low, the CPU is inputting data in the counter.

$\overline{WR}$ (Write) When this is low, the CPU is outputting data in the form of mode information or loading of counters.

A_0, A_1 These pins are normally connected to the address bus. The function of these pins is used to select one of the three counters to be operated and to address the control word registers for mode selection as follows.

A_1	A_0	Selection of Counters and Control word register
0	0	Counter-0
0	1	Counter-1
1	0	Counter-2
1	1	Control word register

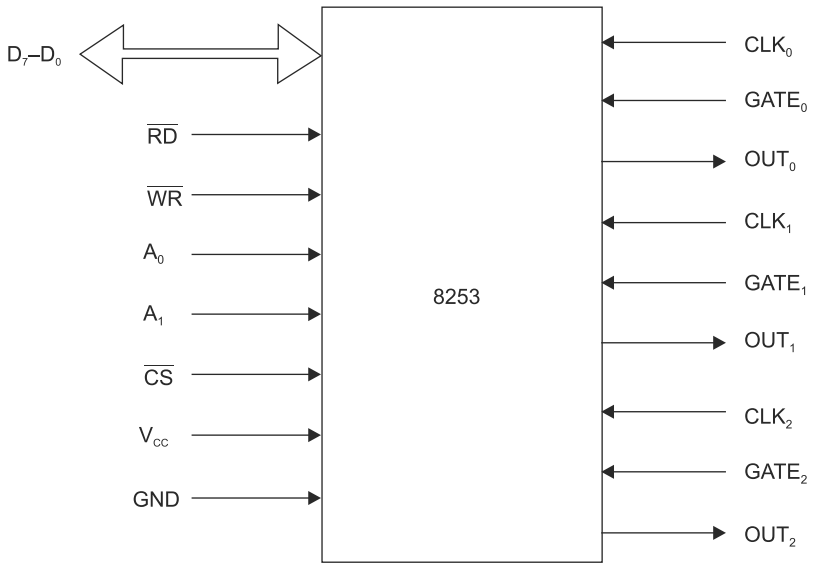


Fig. 13.2 Schematic pin diagram of 8253

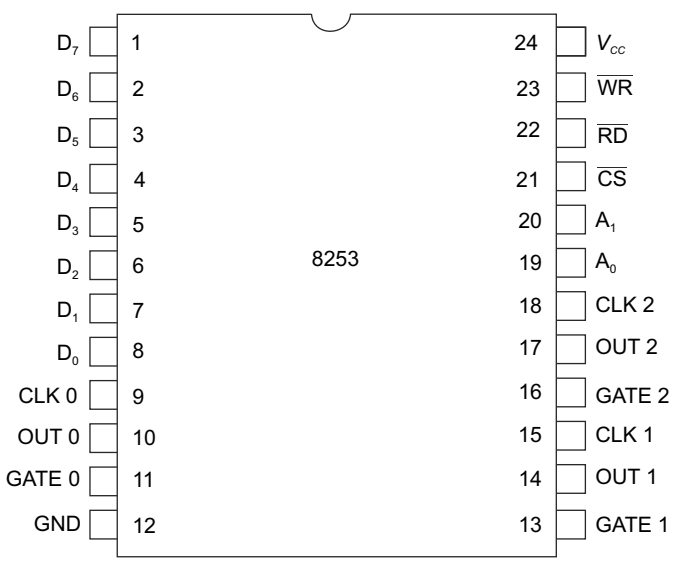


Fig. 13.3 Schematic pin diagram of 8253

$\overline{CS}$ Chip Select A 'low' on $\overline{CS}$ input enables the 8253. No reading or writing operation will be performed until the device is selected. The $\overline{CS}$ input signal is not used to control the actual operation of the counters.

Data Bus Buffer The 3-state, bi-directional, 8-bit buffers exist in 8253. These buffers are used to interface the 8253 to the systems data bus D_0–D_7 lines. Data can be transmitted or received by the buffer upon execution of input and output CPU instructions. The data bus buffer has three basic functions, namely, programming the modes of the 8253, loading the count registers and reading the count values.

D_0–D_7 Bi-directional Data Bus There are eight data lines through which the control word will be written in the control word register of 8253 counter/timer during programming. The counter will be written and read through a data bus.

Read/Write Logic The Read/Write Logic accepts inputs from the system bus and, in turn, generates control signals for operation of 8253. This is enabled by $\overline{CS}$. Therefore, no operation can take place to change the function unless the device has been selected by the system logic. Table 13.1 shows the various functions of 8253 based on the status of pins associated with read/write logic.

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_1	A_0	Function
0	1	0	0	0	Load Counter No. 0
0	1	0	0	1	Load Counter No. 1
0	1	0	1	0	Load Counter No. 2
0	1	0	1	1	Write Mode Word
0	0	1	0	0	Read Counter No. 0
0	0	1	0	1	Read Counter No. 1
0	0	1	1	0	Read Counter No. 2
0	0	1	1	1	No Operation 3-State
1	X	X	X	X	Disable 3 State
0	1	1	X	X	No Operation 3-State

CLK_0, CLK_1, CLK_2 CLK_0 , CLK_1 and CLK_2 are clocks for Counter 0, Counter 1 and Counter 2 respectively. The countdown of the counter takes place on each high to low transition of clock input.

$GATE_0$, $GATE_1$, $GATE_2$ $GATE_0$, $GATE_1$ and $GATE_2$ are gate terminals of Counter 0, Counter 1 and Counter 2 respectively. The function of the GATE in different modes is illustrated in Table 13.1.

OUT_0, OUT_1, OUT_2 OUT_0, OUT_1 and OUT_2 are output terminals of Counter 0, Counter 1 and Counter 2 respectively. The output of the 8253 timer depends upon the mode of operation.

Table 13.1 Different modes of operation corresponding to gate signal

Single status mode	Low or going low	Rising	High
0	Disables counting	–	Enables counting
1	–	Initiates counting Resets output after next clock	–
2	Disables counting Sets output immediately high	Initiates counting	Enables counting
3	Disables counting Sets output immediately high	Initiates counting	Enables counting
4	Disables counting	–	Enables counting
5	–	Initiates counting	–

13.3 BLOCK DIAGRAM

The functional block diagram is illustrated in Fig. 13.1. This device can be divided into functional blocks such as the counter section and the systems interface section.

Counter Section The 8253 consists of three programmable independent counters: Counter #0, Counter #1, and Counter #2. These three functional blocks of counters are identical in operation. Each counter consists of a single 16-bit DOWN counter. The counter can operate in either binary or BCD and its input, gate and output are configured by the selection of MODES stored in the control word register. The counters are fully independent and each can have a separate mode configuration and counting operation, binary or BCD. Each counter can be operated in any of six modes (Mode 0 to Mode 5).

The reading of the contents of each counter is available to the programmer with simple READ operations for event-counting applications. Special commands and logic are incorporated in the 8253 so that the contents of each counter can be read without having to inhibit the clock input.

Systems Interface The $\overline{CS}$ input signal enables the 8253 timer/counter IC. The $\overline{RD}$ and $\overline{WR}$ signals are used for read and write operations respectively. The 8253 can be interfaced with the microprocessor in the same manner as all other peripherals of the family. The 8253 timer/counter which consists of three counters and the control register, will be treated by the systems software as an array of peripheral I/O ports for all modes of programming.

Figure 13.4 shows the interfacing between the microprocessor and the 8253 timer. The data bus D_0–D_7

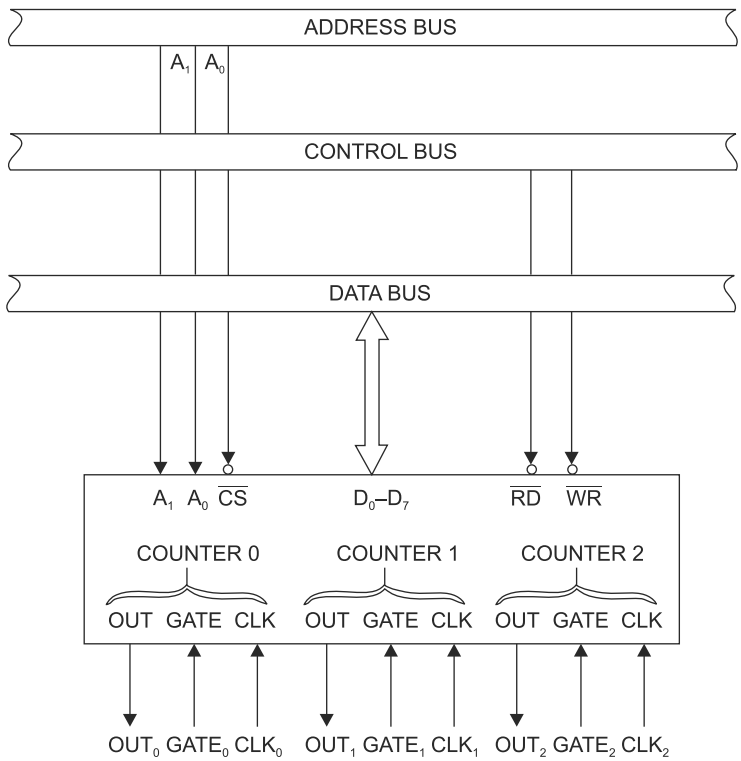


Fig. 13.4 8253 System interface

is connected with the data bus of the microprocessor. The select inputs A_0, A_1 of 8253 connect to the A_0, A_1 address bus signals of the CPU. The $\overline{CS}$ can be derived directly from the address bus using a linear select method or it can be connected to the output of a decoder.

13.4 CONTROL WORD REGISTER

The systems software programs all function of the 8253. This device is programmed to initialize the counter, select the specified counter mode, and read the count value. A control word must be sent out by the CPU to initialize each counter of the 8253 to operate in the desired Mode. Before initialization, the mode count, and output of all counters are undefined. The control words program the mode, loading sequence and selection of binary or BCD counting. Once programmed, the 8253 is ready to perform any operation which it is assigned to carry out.

The control word register is selected when the pins A_0, A_1 are 11. Then the control word register accepts information from the data bus buffer and stores it. The information stored in this register controls the operation of each counter. Each counter has three terminals—CLK, GATE and OUT. The output signal depends on the operating mode. The GATE signal controls the output signal.

All of the modes for each counter are programmed by simple instruction. Writing a control word into the control word register individually programs each counter of the 8253. The control word format is shown below:

Control Word Format

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC_1	SC_0	RL_1	RL_0	M_2	M_1	M_0	BCD

SC—Select Counter The SC_0 and SC_1 bits of the control word select a counter. The selection of counters is given below:

SC_1	SC_0	Select Counter
0	0	Select Counter- 0
0	1	Select Counter -1
1	0	Select Counter -2
1	1	Illegal

RL—Read/Load The RL_0 and RL_1 are used to load/read counts as follows:

RL_1	RL_0	Read/Load
0	0	Counter latching operation
0	1	Read/Load least significant byte only
1	0	Read/Load most significant byte only
1	1	Read/Load least significant byte first, then most significant byte

M—Mode Mode selecting bits M_0, M_1 and M_2 select any one of six modes as given below:

M_2	M_1	M_0	Mode
0	0	0	Mode 0
0	0	1	Mode 1
×	1	0	Mode 2

×	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

BCD

0 Binary counter, 16-bits

1 Binary Coded Decimal (BCD) counter (4 Decade)

Reading While Counting The 8253 timer has a command for latching the content of a counter to read the count value without stopping the counting. This device has a special internal logic to achieve this. The count value can be read after loading a control word in the control word register. The bit pattern for the control word for this operation is as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC_1	SC_0	0	0	×	×	×	×

SC_1 and SC_0 specify counter to be latched.

D_5 and D_4 00 makes counter latching operation

X indicates 'doesn't care'.

For example, the control word for reading the count value of the counter 1 is

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	1	0	0	×	×	×	×

Bits D_7 and D_6 are 0 and 1 respectively to represent the counter 1.

Bits D_5 and D_4 are 0 and 0 to represent latching operation.

Other bits are either 1 or 0.

Therefore, the control word for the above operation is 40H.

Whenever the microprocessor wants to latch the counter, the latching command must be issued every time and then it reads the content of the counter. In this mode of operation, the counter operation will not be affected by the latching command. After receiving the latching command, the 8253 latches the content of the counter and stores it in a storage register. Then the microprocessor reads the content of the register by issuing a read instruction. This read operation is generally specified by RL_1 and RL_0 bits of the original mode set command. For counter latching operation, RL_1 and RL_0 are '0' and '0' respectively. If $RL_1 = 0$ and $RL_0 = 1$, only read least significant byte LSB of the counter. When $RL_1 = 1$ and $RL_0 = 0$, only read MSB of the count. If $RL_1 = 1$ and $RL_0 = 1$, read LSB of the count first, and thereafter read MSB of the count.

13.5 OPERATIONAL MODES

The 8253 consists of three independent negative edge-triggered 16-bit down counters, namely, counter 0, counter 1 and counter 2. As the counters are fully independent, each counter of 8253 can be programmed in a different mode configuration and counting operation. The programmer must write the control word in the control word register and load the count value in a selected count register. Writing the mode control word, the counter may be selected in any sequence. Each counter's mode control word register has a separate address so that it can be loaded independently. Usually, the 8253 is available on a microprocessor kit. Sometimes 8253 timer/counter is also connected with the microprocessor kit externally. The clock frequency is about 1.5 MHz, which is available on the kit. If the clock frequency is about 3 MHz, an edge-triggered flip-flop can be

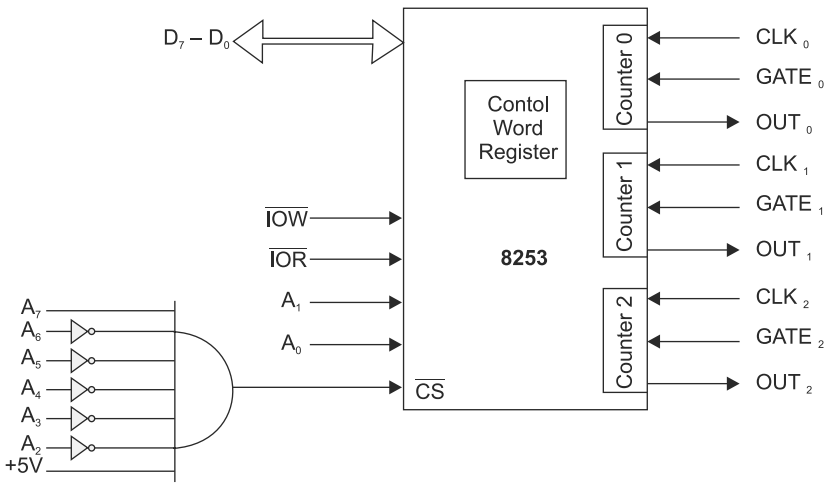


Fig. 13.5 Interfacing the 8253

used to divide this clock frequency by two to obtain a desired clock frequency for operating 8253 properly.

The port address for control word register and the counters of 8253 are as follows:

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
1	0	0	0	0	0	1	1	83H Address of control word register
1	0	0	0	0	0	0	0	80H Address of Counter 0
1	0	0	0	0	0	0	1	81H Address of Counter 1
1	0	0	0	0	0	1	0	82H Address of Counter 2

A counter can be used for various applications such as BCD/binary counter, programmable rate generator, square wave generator, hardware/software triggered strobe, programmable one-shot, generate time delay, etc. Descriptions of some applications are given below.

13.5.1 MODE 0—Interrupt on Terminal Count

Generally, Mode 0 is used to generate accurate time delay under software control. Firstly, any one counter of 8253 timer/counter is initialized and loaded with a suitable count to develop the desire time delay. After termination of counting, the counter interrupts the microprocessor. When counter interrupts the microprocessor, the specified operations will be performed by the microprocessor. When the control word is loaded into the control word register, 8253 timer/counter sets the mode. In Mode 0 operation, the output of the counter becomes initially low after the mode is set. After mode set operation, the selected counter must be loaded by the desired count value *N*.

In this mode of operation, GATE is kept high. Therefore, just after loading the count registers, the counter starts to decrement. When counting is going on, the counter output terminal OUT remains low. As soon as the terminal count reaches 0, the output becomes high. The output remains high until the counter is reloaded or a new count value is loaded into counter. When the count is reloaded or a new count is loaded, the counting restarts from new count value and again OUT becomes low. The timing diagram for Mode 0 operation is shown in Fig. 13.6.

While counting is going on, GATE becomes low suddenly and the counter stops counting operation. After some time if the GATE returns to 1, counting is resumed from the count value at which the counting discontinued. The count value can be changed at any time. A new count value can also be loaded while counting is going on. But the changes will be effective only after the next GATE trigger. This mode of operation can be

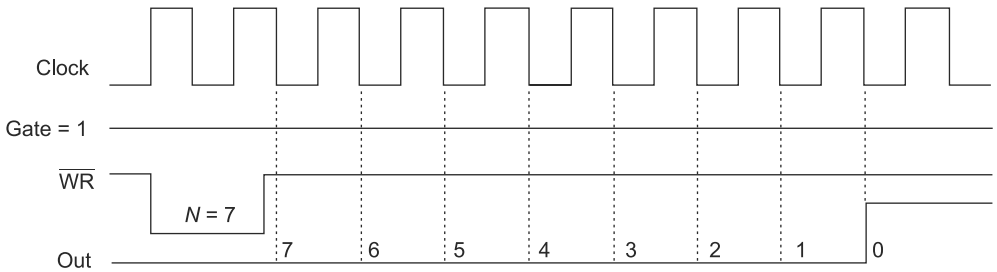


Fig. 13.6 Mode 0–interrupt on terminal count

used to generate accurate time delay. This can also be used to perform specified operation after some delay. The output of OUT terminal may be used to interrupt the microprocessor. The examples of Mode 0 operation are given below:

Example 13.1 Write a subroutine program to initialize counter 0 in Mode 0 with a count value of 8000H.

Sol. The control word for Mode 0 operation as given below:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	1	1	0	0	0	0 = 30H

D_7 and D_6 have been set to 00 to initialize the counter 0.
D_5 and D_4 have been set 11 to load the least significant byte of the count first, then the most significant byte.
D_3, D_2, D_1 are set to 000 for Mode 0 operation.
D_0 is set to 0 as counting is to be done in binary.
The address of the control register is 83H and the address of the counter 0 is 80H.
The program for loading the control word and 16-bit number in the counter 0 is given below:

Program

Memory address	Machine Codes	Mnemonics	Operands	Comments
9000	3E, 30	MVI	A, 30	Control word for Mode 0 to initialize the counter 0
9002	D3, 83	OUT	83	Write the control word into control word register
9004	3E, 00	MVI	A, 00 H	Least significant byte of the count.
9006	D3, 80	OUT	80	Load counter 0 by 00H, LSB of count
9008	3E, 80	MVI	A, 80	Most significant byte of the count.
900A	D3, 80	OUT	80	Load counter 0 by 80H, MSB of count

After execution of the program, the gate signal becomes ONE.

Example 13.2 Read the count value of the counter while the counting is going on. Assume the counter 0 in Mode 0 with a count value 8000H.

Sol. The 8253 timer/counter can be able to read the count value without stopping the counting. The count

value can be read after loading the control word in the control word register. The control word for this operation is as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	x	x	x	x = 00H

D_7 and D_6 have been set to 00 to represent the counter 0.
D_5 and D_4 have been set to 00 for latching operation.
D_3, D_2, D_1 and D_0 are not checked when D_5 and D_4 are 0 and 0.
The address of the control register is 83H and the address of the counter 0 is 80H
The program for reading count value of the counter 0 while counting is given below:

Program

Memory address	Machine Codes	Mnemonics	Operands	Comments
9000	3E, 30	MVI	A, 30	Control word for MODE 0 to initialize the counter 0
9002	D3, 83	OUT	83	Write the control word into control word register
9004	3E, 00	MVI	A, 00 H	Least significant byte of the count
9006	D3, 80	OUT	80	Load counter 0 by 00H, LSB of count
9008	3E, 80	MVI	A, 80	Most significant byte of the count
900A	D3, 80	OUT	80	Load counter 0 by 80H, MSB of count
900C	3E, 00	MVI	A, 00 H	Load the control word into control-word register
900E	D3, 83	OUT	83	
9010	DB, 80	IN	80	Read least significant byte of the count value
9012	5F	MOV	E, A	Store least significant byte in E register
9013	DB, 80	IN	80	Read most significant byte of the count value
9015	57	MOV	D, A	Store most significant byte in D register. The register pair BC contains the present value of the counter 0

Example 13.3 Counter 0 of 8253 timers is used in Mode 0 to perform addition of an array after certain delay. Assume count value for delay = 2000H. After completion of counting, the counter interrupts the microprocessor to jump a memory location for addition of two 8-bit number.

Sol. It is a very simple task to jump from one memory location to another memory location. But, here the task is to jump from one memory location to another memory location after the microprocessor is interrupted. For this, all interrupts must be enabled. Then the content of the accumulator will enable RST 7.5, 6.5 and 5.5 for SIM instruction as given below:

7	6	5	4	3	2	1	0
SOD	SOE	X	R7.5	MSE	M7.5	M6.5	M5.5
0	0	0	0	1	0	0	0 = 08H

The control word for Mode 0 operation of the counter 0 is as follows.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	1	1	0	0	0	0 = 30H

D_7 and D_6 are set to 0 and 0 respectively to initialize the counter 0.

D_5 and D_4 have been set 11 to load the least significant byte of the count first, after that the most significant byte must be loaded.

D_3, D_2, D_1 are set to 0 0 0 for Mode 0 operation.

D_0 is set to 0 as counting is to be done in binary. Connect OUT terminal of the counter 0 with RST 7.5 of the microprocessor.

Program

Memory address	Machine Codes	Mnemonics	Operands	Comments
8000	FB	EI		All interrupts are enable
8001	3E, 08	MVI	A, 08	Load bit pattern to accumulator to enable RST 7.5, 6.5 and 5.5
8003	30	SIM		RST 7.5, 6.5 and 5.5 are enable.
8004	3E, 30	MVI	A, 30	The control word for Mode 0 to initialize counter 0
8006	D3, 13	OUT	83	Write control word into control word register
8008	3E, 00	MVI	A, 00 H	Load least significant byte of the count in the counter 0.
800A	D3, 11	OUT	80	Load counter 0 by LSB of count value
800C	3E, 20	MVI	A, 20	MSB of count value
800E	D3, 11	OUT	80	Load the counter 0 by MSB of count value
8010	C3, 10, 80	JMP	8010	

As soon as the count value is loaded in the counter 0, the counter starts decrementing. When the counting has been completed, RST 7.5 interrupts microprocessor and the program jumps to the memory location 003C. After that the monitor transfers the program from 003C location to 9000H. Then the jump instruction which is stored at 9000H location can transfer the program from 9000H to the starting address of the subroutine, 8500H. The subroutine program is given below:

9000	C3, 00, 85	JMP	8500	Jump to subroutine at 8500H
SERVICE SUBROUTINE				
8500	21 00 86	LXI H	8600	Load 8600 in H-L register pair
8503	7E	MOV	A, M	Move content of 8600 into accumulator
8504	23	INX	H	Increment H-L pair
8505	86	ADD	M	Add the 2nd data with accumulator
8506	32 02 86	STA	8602	Store result in 8602 memory location
8509	FB	EI		Enable all interrupts
850A	C9	RET		
DATA				
8600	22			
8601	44			
RESULT				
8602	66			

After execution of subroutine, the program returns from the subroutine to the main program. Before return to the main program all interrupts must be enabled so that any additional interrupts can be acknowledged.

13.5.2 MODE 1—Programmable One-Shot

In this mode, the counter acts as a retriggerable and programmable one-shot. The rising edge trigger signal is applied to GATE terminal of counter. In this mode, initially OUT is high after the mode is set. The control word and the count value must be loaded into the counter. When the mode set operation is done and the counter is loaded by a count value, the counter starts a decrement count. At the first negative edge of the clock after the rising edge trigger signal of the GATE input, the output becomes low. Then the output (OUT) will be low for a number of count clock cycles. After completion of the count, the output becomes high as depicted in Fig. 13.7. The width of the output pulse depends upon the count value. Consequently, the width of the output pulse can be varied by changing the count value in the program. Therefore this mode of operation can be called as programmable one-shot.

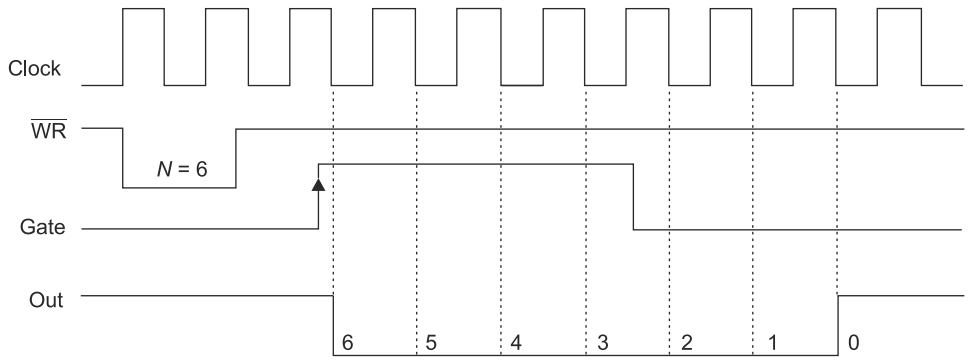


Fig. 13.7 Mode 1–Programmable one-shot

Example 13.4 Write a program to operate the counter 0 of 8253 timer/counter in MODE 1.

Sol. The control word for counter 0 in Mode 1 was determined as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	1	0	0	1	0 = 12H

D_7 and D_6 are set to 0 and 0 respectively to select the counter 0.

D_5 and D_4 are also set to 0 and 1 respectively for loading only least significant bits (LSB) of the count.

D_3, D_2 and D_1 are set to 001 for Mode 1.

D_0 is set to 0 for binary counting.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	3E, 12		MVI	A, 12H	Load control word to initialize counter 0 in MODE 1
8002	D3, 83		OUT	83	Write the control word in control word register
8004	3E, 10		MVI	A, 10	Get count
8006	D3, 80		OUT	80	Load counter 0 with the count

13.5.3 MODE 2—RATE Generator

In Mode 2, the counter behaves as a divide by N counter. Generally, it is used to generate a real-time clock interrupt. The control word and the count value are loaded into the control word register and counter respectively. After the mode is set, the output of the counter will be initially high. If the counter is loaded by a count of value N, the output remains high for (N–1) clock pulses. After (N–1) clock pulses, output will be low for one clock pulse and then output becomes high again as shown in Fig. 13.8. Thereafter the count value N is reloaded into counter and the output remains high for (N–1) clock pulses and will be low for one clock pulse. If a new count value is reloaded into the count register between output pulses, the present period is not affected. The subsequent period reflects the new value of the count.

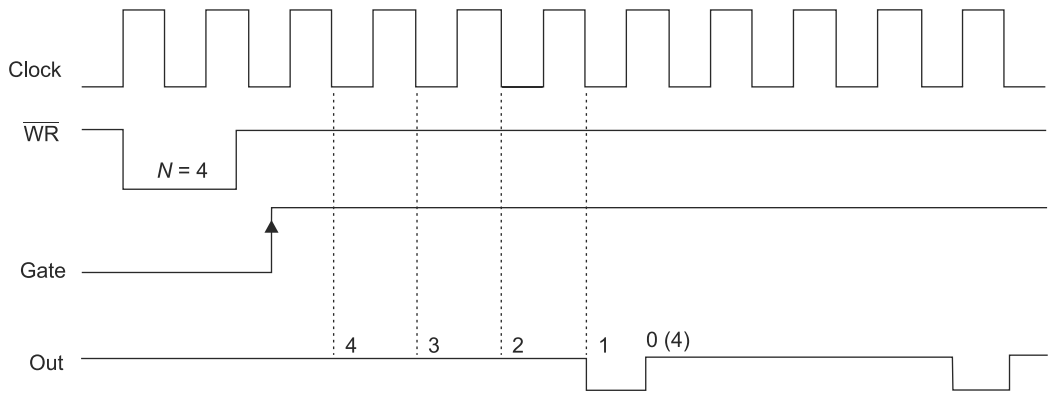


Fig. 13.8 Mode 2–rate generator

Example 13.5 The counter 1 of 8253 time operates in Mode 2, divide by -N binary counter. Assume N = 6 in decimal.

Sol. When the counter 1 of 8253 time operates in Mode 2 and divide by -N binary counter, the control word are as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	1	0	1	0	0	1	0 = 54 hex

D_7 and D_6 are set to 0 and 1 respectively to initialize the counter 1.

D_5 and D_4 have been set to 0 and 1 to load the only least significant byte of the count.

D_3, D_2, D_1 are used for mode select. Mode 0 operation D_3, D_2, D_1 are set to 0 1 0.

D_0 is set to 0 as counting is to be done in binary.

54H is the control word for MODE 2 operation.

In the counter 1, $GATE_1$, OUT_1 and CLK_1 terminals are available at microprocessor kit. Apply +5 to $GATE_1$ of counter 1 to make it high. The clock has been connected to terminal CLK_1. The clock frequency must be desired frequency approximately 1.5 MHz.

Program

Memory address	Machine Codes	Mnemonics	Operands	Comments
8000	3E, 54	MVI	A, 54H	Load control word for counter 1, MODE 2, binary counting

Contd.

Contd.

8002	D3, 83	OUT	83	83 is address for writing control word in control word used
8004	3E, 06	MVI	A, 06 H	Load count value N. A = 06H
8006	D3, 81	OUT	81	81 H is the address for counter-1
8008	76	HLT		Stop

13.5.4 MODE 3—Square Wave Generators

In this mode, the counter operates as a square-wave generator. To operate the counter 1 in mode 3, the control word must be loaded into control word register. After mode setting, the counter is loaded by a count of value *N*. When the GATE becomes high, the counter starts counting. The output remains high for half of the count value, *N*/2 clock pulses and it remains low for the rest of the count values or *N*/2 clock pulses. Therefore a continuous square wave of specified period can be generated at output terminal as depicted in Fig. 13.9.

For even values of *N*, the output is high for *N*/2 clock pulses and low for next *N*/2 clock pulses. For odd values of *N*, the output remains high for $(N + 1)/2$ clock pulses and low for remaining clock pulses. By changing the count value, time period of square wave can be controlled. After completion of count, the output state is changed and the counter is automatically reloaded with the full count and the above process will be repeated.

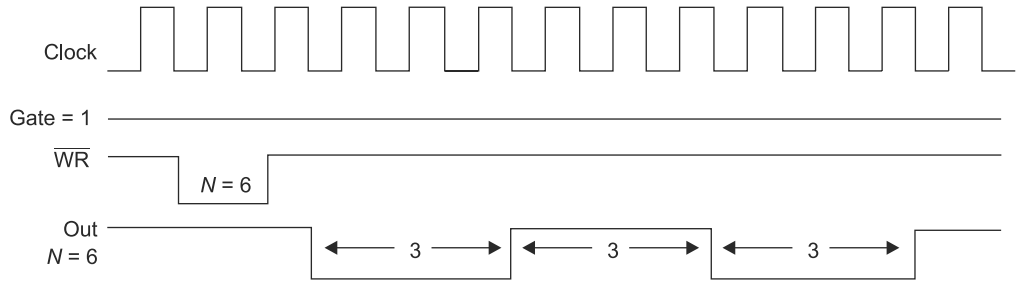


Fig. 13.9 (a) Mode 3–square wave generator with even count value

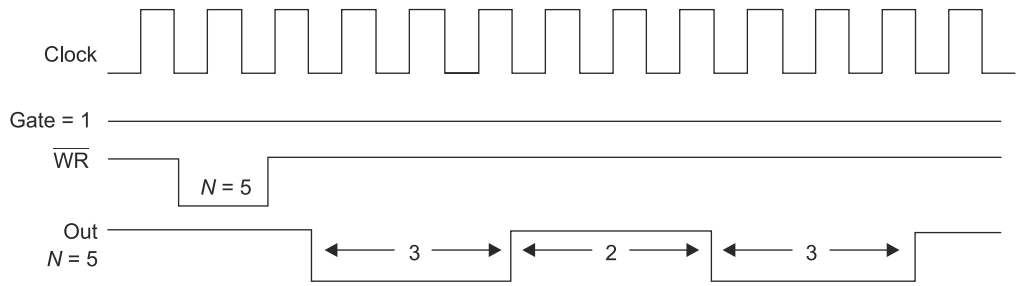


Fig. 13.9 (b) Mode 3–square wave generator with odd count value

Example 13.6 Write a program to use the counter 1 of 8253 in Mode 3 as a square-wave generator. Assume $N = 16$. The counter operates as a binary counter.

Sol. The counter 1 has been used as a square-wave generator and the control word for this operation is

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	1	0	1	0	1	1	0 = 56 H

D_7 and D_6 have been set to 0 and 1 to select the counter 1.

D_5 and D_4 are set to 0 and 1 for loading only LSB of the count.

D_3, D_2 and D_1 are set to 011 for Mode 3 operation.

D_0 is set to 0 for binary counting.

Program

Memory address	Machine Codes	Mnemonics	Operands	Comments
9000	3E, 56	MVI	A, 56H	Load the control word for MODE 3 in control word register to initialize counter 1
9002	D3, 83	OUT	83	Write in control word register
9004	3E, 10	MVI	A, 10 H	Load the count value for binary counting
9006	D3, 81	OUT	81	Load counter 1 with the count value
9008	76	HLT		Stop

If $N = 16$, the output will be high for 8 clock cycles and then it will be low for the next 8 clock cycles. When $N = 17$, the output remains high for 9 clock pulses and then low for remaining 8 clock pulses. A continuous square wave can be generated at the output as the counter is reloaded by the same value and the timer repeats the process repeatedly.

13.5.5 MODE 4—Software Triggered Strobe

In the software triggered strobe operation, the counter output will be high after the mode is set. The GATE is always high for this mode of operation. When the counter is loaded with a count value, the counter starts counting. As soon as the count value is loaded into the counter register, it triggers the generation of the strobe. Therefore, this mode of operation is known as software triggered strobe. When the counter content becomes 0, the output will be low for one clock period and thereafter; output will be remaining high as illustrated in Fig. 13.10.

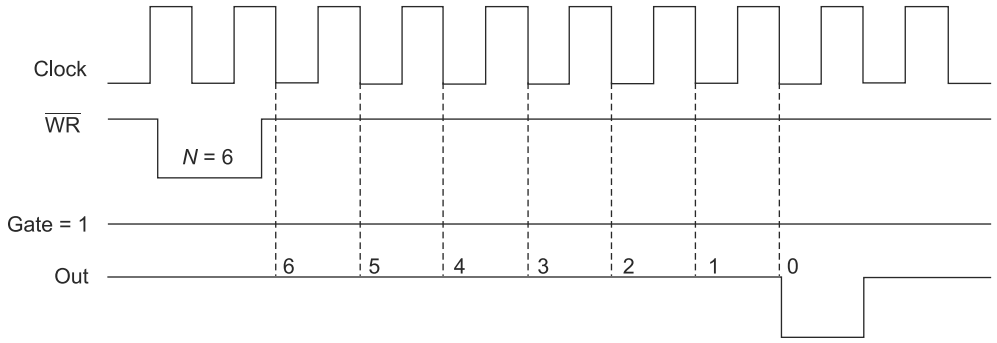


Fig. 13.10 Mode 4–Software triggered strobe

Example 13.7 Write a program use counter 2 of 8253 in MODE 4.

Sol. The control word for the counter 2 for Mode 4 operation is as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	1	1	1	0	0	0 = B8 hex

D_7 and D_6 are set to 10 to select the counter 2.

D_5 and D_4 have been set to 11 to load the LSB of the count value first, then load MSB of the count value.

D_3, D_2 and D_1 have been set to 100 for Mode 4 operation.
D_0 is set to 0 for binary counting.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	3E, B8		MVI	A, B8H	Load control word to initialize counter 2 for Mode 4 operation
8002	D3, 83		OUT	83	Write control word in control word register
8004	3E, 05	LOOP	MVI	A, 05	Load LSB of the count
8006	D3, 82		OUT	82	Load counter 2 with LSB of the count
8008	3E, 00		MVI	A, 00	Load MSB of the count
800A	D3, 82		OUT	82	Load counter 2 with MSB of the count
800C	C3, 04, 80		JMP	LOOP	

13.5.6 MODE 5—Hardware Triggered Strobe

In this mode of operation, initially the output is high. The control word and count value are loaded in the counter register. Here GATE input acts as a trigger signal. When low to high transition of the GATE input occurs, the counter starts to decrement the count value and the output becomes initially high. The output goes low for one clock period, when the counter completes the count value. As the low to high transition of the GATE input or the rising edge of GATE trigger the counter, this mode is called hardware triggered strobe as depicted in Fig. 13.11.

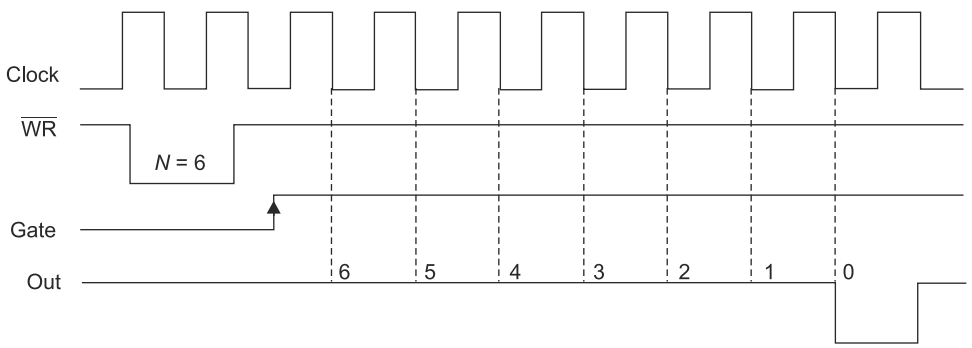


Fig. 13.11 Mode 5–hardware triggered strobe

This mode of the counter operation is also retriggerable. When the GATE input becomes low to high again, the counter is reloaded by the count value *N* and the counter starts to decrement the count value once again. The output will also be low for one clock period on terminal count.

Example 13.8 Write a program to use counter 2 of 8253 in Mode 5 operation.

Sol. The control word for the counter 2 in Mode 5 is as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	1	1	1	0	1	0 = BA H

D_7 and D_6 are set to 1 and 0 respectively to select counter 2. D_5 and D_4 are set to 11 for loading least significant bit first, then most significant bit. D_3, D_2 and D_1 are set to 101 for MODE 5 operation. D_0 is set to 0 for binary counting.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	3E, BA		MVI	A, BA	Load control word to initialize counter 2 in Mode 5 operations
8002	D3, 83		OUT	83	Write the control word into control word register
8004	3E, 06		MVI	A, 06	Load LSB of the count
8006	D3, 82		OUT	82	Load the counter 2 with LSB of the count value
8008	3E, 00		MVI	A, 00	Get MSB of the count
800A	D3, 82		OUT	82	Load MSB of counter into counter 2
800C	3E, 80	LOOP	MVI	A, 80 H	Initialize ports of 8255.2
800E	D3, 0B		OUT	0B	Load the counter 2 with MSB of the count value
8010	3E, 00		MVI	A, 00	Generate a pulse output at PC_0
8012	D3, 00A		OUT	0A	terminal, which is connected to GATE of 8253
8014	3E, 01		MVI	A, 01	
8016	D3, 0A		OUT	0A	
8018	C3, 0C, 80		JMP	LOOP	

13.6 8253 INTERFACING WITH 8085 MICROPROCESSOR

Figure 13.12 shows the interfacing of 8253 with the 8085 microprocessor. A_0 and A_1 address lines of 8085 microprocessor are directly connected to the A_0 and A_1 pins of 8253 IC to select the address of Counter 0, Counter 1 and Counter 2 and the control word register. The chip select $\overline{CS}$ signal is generated from A_2 to A_7 address lines and $IO/\overline{M}$ using a decoder. $\overline{RD}$, $\overline{WR}$, and data bus $D_0 - D_7$ of 8253 are directly connected with $\overline{IOR}$, $\overline{IOW}$ and $D_0 - D_7$ of the 8085 microprocessor. The port address of Counter 0, Counter 1 and Counter 2 and the control word register are given in Table 13.2. The operating modes of 8253 are discussed in Section 13.5 in detail.

Table 13.2 Address of counters and CWR

Ports and CWR	Status of I/O Address lines								Address in Hex
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
Counter 0	1	0	0	0	0	0	0	0	80H
Counter 1	1	0	0	0	0	0	0	1	81H
Counter 2	1	0	0	0	0	0	1	0	82H
CWR	1	0	0	0	0	0	1	1	83H

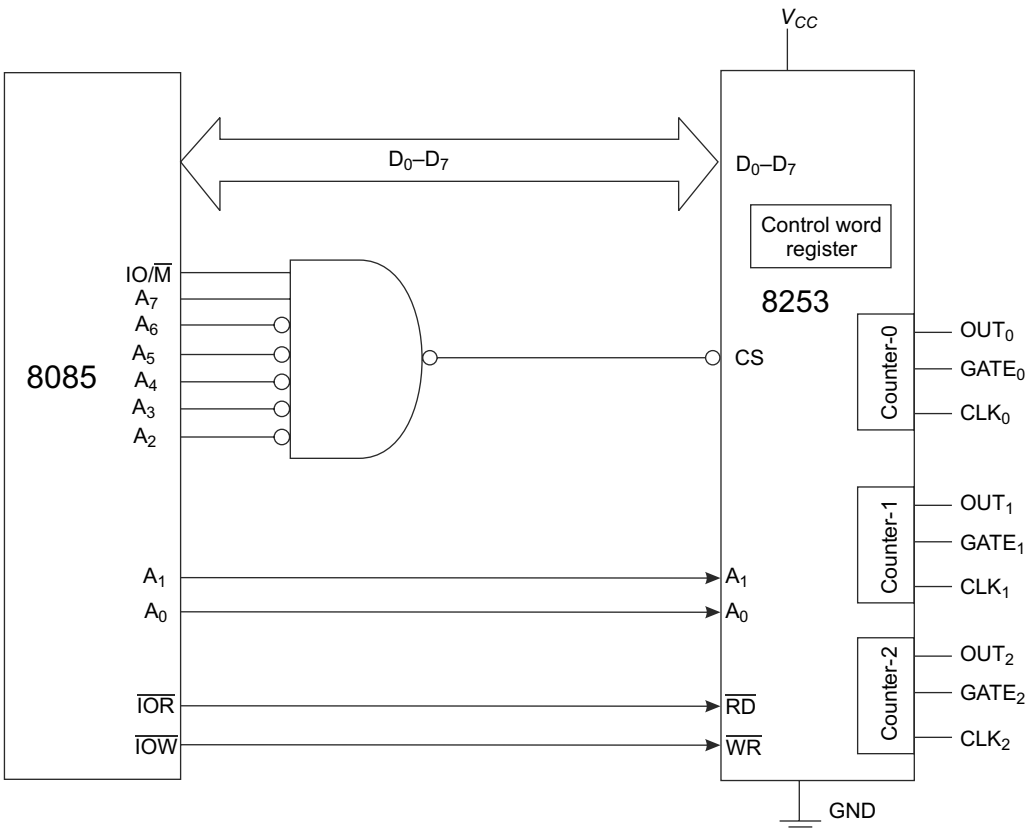


Fig. 13.12 8253 interfacing with 8085 microprocessor

13.7 8253 INTERFACING WITH 8086 MICROPROCESSOR

Figure 13.13 shows the 8253 interfacing with the 8086 microprocessor when the higher order address lines A_{15}–A_8 are neglected. The lower order data bus D_7–D_0 of 8086 is directly connected with D_7–D_0 of 8253. A_0 and A_1 pins of 8253 IC are connected with A_1 and A_2 address lines of 8086 microprocessor. $\overline{RD}$ and $\overline{WR}$ of 8253 is directly connected with $\overline{IORD}$ and $\overline{IOWR}$ of 8086. The clock signal of 8086 is divided by 4 and the output pulse divided by 4 counter is used as clock input of 8253. The chip select $\overline{CS}$ signals is generated from the address lines A_0 and A_3 to A_7 . Usually, a decoder is used to get $\overline{CS}$ signal. The port address of Counter 0, Counter 1 and Counter 2 and control word register are given in Table 13.3.

Table 13.3 Address of counters and CWR

Ports and CWR	Status of I/O Address lines								Address in Hex
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
Counter 0	0	1	0	0	0	0	0	0	40H
Counter 1	0	1	0	0	0	0	1	0	42H
Counter 2	0	1	0	0	0	1	0	0	44H
CWR	0	1	0	0	0	1	1	0	46H

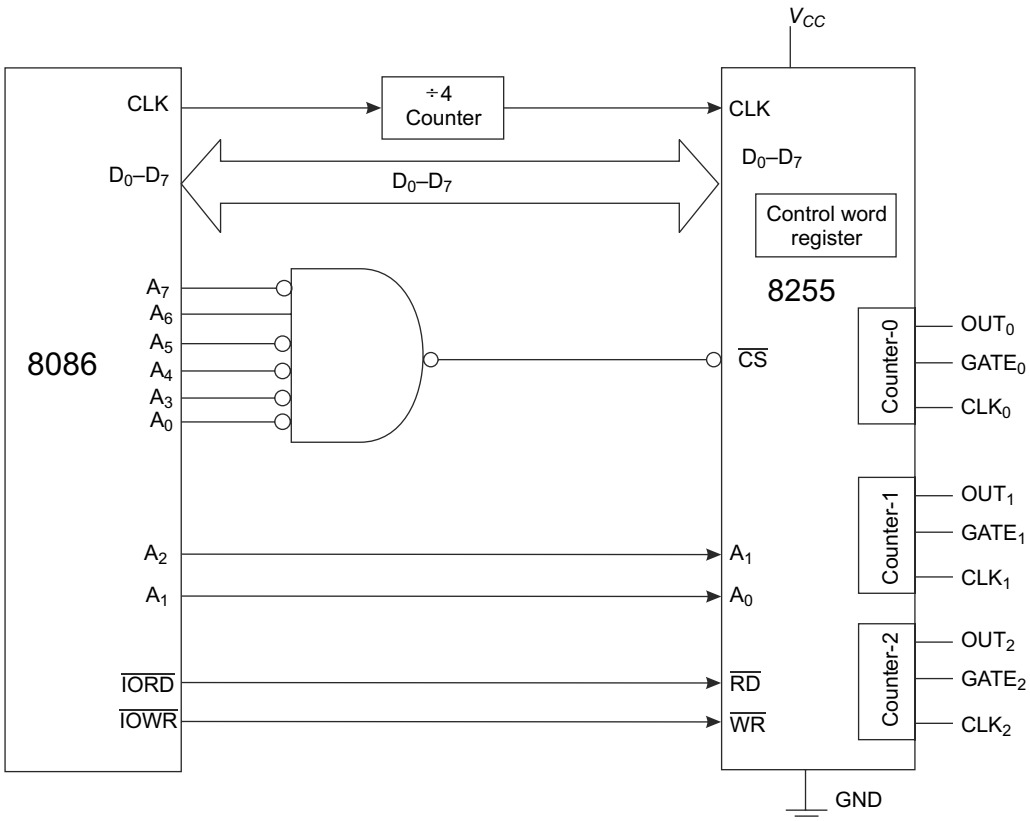


Fig. 13.13 8253 interfacing with 8086 microprocessor

To generate a square wave, 8253 must be operating in Mode 3. If Counter 0 is used for this purpose, it will be operated in BCD mode. Then the control word is 37H as given below:

SC_1	SC_0	RL_1	RL_0	M_2	M_1	M_0	BCD	Control Word
0	0	1	1	0	1	1	1	37H

If we assume clock frequency is 1.5 MHz, time period $T = \frac{1}{1.5 \times 10^6} = 0.66$ μs.

To generate a square wave of 1 ms time period, the number of T states are required $N = \frac{1 \times 10^{-3}}{0.66 \times 10^{-6}} = 1500$ states

The assembly-language program for the above operation is given below:

MOV AL, 37H	Load control word 37H in AL register
OUT 46H, AL	Load control word 37H in control word register and 8253 is initialized
MOV AL,00	Write 00 decimal in AL
OUT 40H, AL	Load LSB of count in counter 0
MOV AL, 15H	Write 15 decimal in AL

OUT 40H, AL	Load MSB of count in Counter 0
HLT	Stop

In the 8086 microprocessor interfacing with 8253, 16 address lines can also be used to drive 8253. In this case, A_1 and A_2 address lines of 8086 microprocessor are directly connected to the A_0 and A_1 pins of 8253 IC for selecting Counter 0, Counter 1, Counter 2 and control word register address. The other lines A_0, A_3–A_7 and A_8–A_{15} are used for generating $\overline{CS}$ signal. Figure 13.14 shows the 8255 interfacing with 8086 microprocessor where the address of control word register is 0736H. The addresses of Counter 0, Counter 1 and Counter 2 are 0730H, 0732H, and 0734H respectively. To generate the above addresses, the status of pins is given in Table 13.4.

Table 13.4 Port address of counters and CWR

Ports and CWR	Status of I/O Address lines																Address in Hex
	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	
Counter 0	0	0	0	0	0	1	1	1	0	0	1	1	0	0	0	0	0730H
Counter 1	0	0	0	0	0	1	1	1	0	0	1	1	0	0	1	0	0732H
Counter 2	0	0	0	0	0	1	1	1	0	0	1	1	0	1	0	0	0734H
CWR	0	0	0	0	0	1	1	1	0	0	1	1	0	1	1	0	0736H

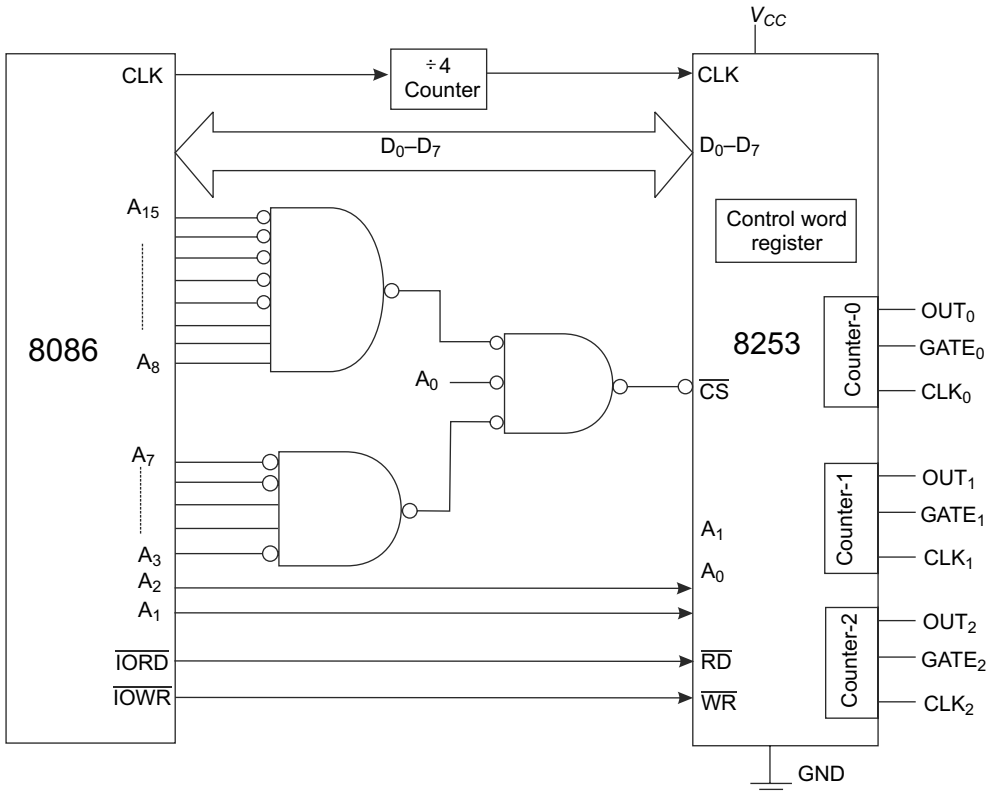


Fig. 13.14 8253 interfacing with 8086 microprocessor

13.8 8253 INTERFACING WITH 8051 MICROCONTROLLER

Figure 13.15 shows the interfacing of 8253 with 8051 microcontroller. A_0 and A_1 address lines of 8051 microcontroller are directly connected to the A_0 and A_1 pins of 8253 IC for selecting the addresses of Counter 0, Counter 1, Counter 2 and the control word register. The chip select $\overline{CS}$ signal is generated from A_2 to A_7 and A_8 to A_{15} address lines. When the address of the control word register is EA03H, the addresses of Counter 0, Counter 1, Counter 2 are EA00H, EA01H and EA02H respectively. To generate the above address, the status of pins A_0–A_{15} is given in Table 13.5.

Table 13.5 Address of ports and CWR

Ports and CWR	Status of I/O Address lines																Address in Hex
	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	
Counter 1	1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	EA00H
Counter 0	1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	1	EA01H
Counter 2	1	1	1	0	1	0	1	0	0	0	0	0	0	0	1	0	EA02H
CWR	1	1	1	0	1	0	1	0	0	0	0	0	0	0	1	1	EA03H

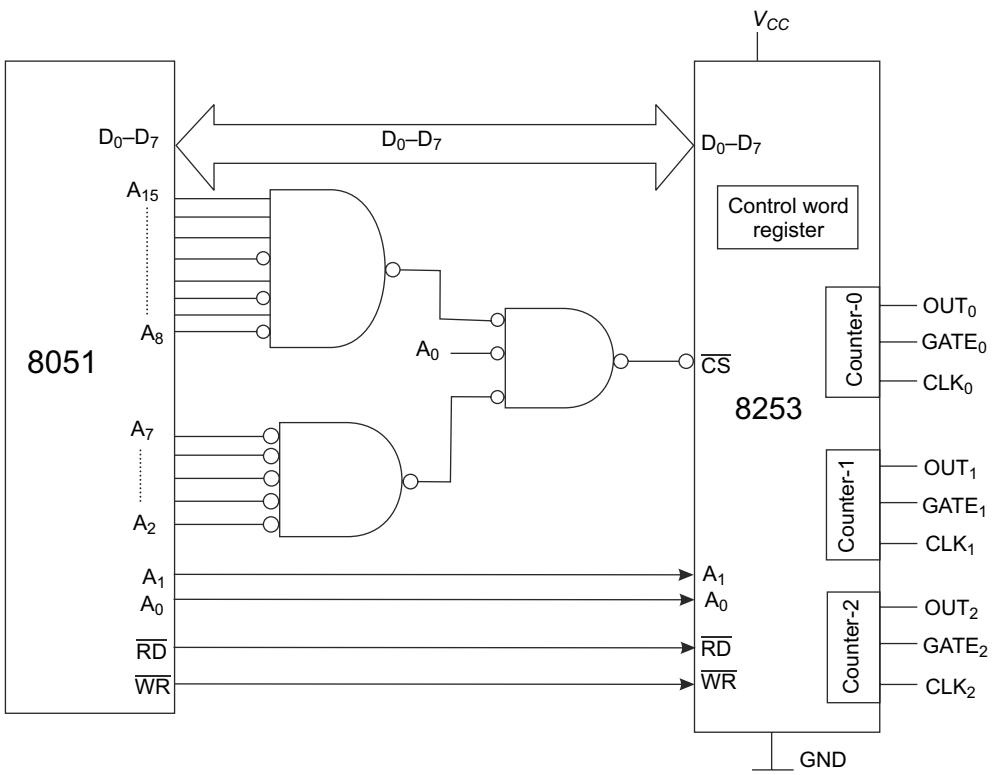


Fig. 13.15 8253 interfacing with 8051 microcontroller

Review Questions

13.1 Explain the operation of 8253 timer IC with its functional block diagram.

13.2 Mention the different modes of operation of 8253 IC.

13.3 Draw the functional block diagram of 8253. How many counters are there in 8253 and how many modes are there?

13.4 Explain Mode-0 operation with timing diagram.

13.5 Explain the importance of GATE signal. How is it used to control the operation of counters?

13.6 Explain Mode-1 and Mode-2 operations with timing diagrams. Write the difference between Mode-2 and Mode-3 of 8253.

13.7 Explain Mode-3 and Mode-4 of 8253 with timing diagrams.

13.8 Write the interfacing procedure to interface 8253 with the 8085 microprocessor.

13.9 Give a list of applications of the 8253 timer.

13.10 Write the control word format and explain all modes of operation.

13.11 Show in a tabular form, the conditions of different modes corresponding to the different status of GATE signals.

13.12 Discuss different methods of reading the value of the count in a counter while counting is in progress.

13.13 Write a program to read the count value of the counter while counting is going on. Assume the counter 0 in Mode 0 with count value 7000H.

13.14 Write a program to generate a square wave using 8253.

13.15 Write a program to use the counter 2 of 8253 in Mode 5 operation.

13.16 (a) Draw the 8253 interfacing with 8085 microprocessor and write the address of Counter - 0, Counter - 1, Counter - 2 and control word register.

(b) Write a program for square wave generator with 10 ms time period.

13.17 (a) Design the 8253 interfacing with 8086 microprocessor when the address of control word register is 46H

(b) Write an assembly language program to generate square wave of 1 ms time period.

13.18 (a) Draw the 8253 interfacing with 8051 microcontroller when the address of Counter-0 is EA00H and explain briefly.

(b) Write a program for rate generator for the above interfacing circuit.

Multiple-Choice Questions

13.1 Which pin is used to control the output of the counter 2 of 8253 in Mode 2?

(a) GATE 0 (b) GATE 1 (c) GATE 2 (d) GATE 3

13.2 What are the bits of the control word to select a counter?

(a) SC_0 SC_1 (b) RW_0 RW_1 (c) M_0 M_1 M_2 (d) BCD, RW_0 & RW_1

13.3 The control word register is selected by the read/write logic when

(a) A_0 A_1 = 11 (b) A_0 A_1 = 01 (c) A_0 A_1 = 10 (d) A_0 A_1 = 00

13.4 8253 has
(a) 6 modes of operation (b) 5 modes of operation
(c) 4 modes of operation (d) 3 modes of operation

13.5 8253 is capable to handle clock frequency at
(a) 1 MHz (b) 2 MHz (c) 3 MHz (d) 4 MHz

13.6 The control word for Mode 0 operation of Counter - 0 with a count value of 16-bit number is
(a) 30H (b) 40H (c) 50H (d) 60H

13.7 In 8253 interfacing with 8086 microprocessor, if the address of counter - 0 is 0740H, the address of control word register is
(a) 0743H (b) 0744H (c)0745H (d) 0746H

13.8 In 8253 interfacing with 8085, A_0 and A_1 pins of 8253 are connected with ____ pins of 8085 respectively.
(a) A_0 and A_1 (b) A_1 and A_0 (c) A_0 and A_2 (d) A_1 and A_2

13.9 During 8253 interfacing with 8086 microprocessor, A_0 and A_1 pins of 8253 are connected with ____ pins of 8086 respectively.
(a) A_1 and A_2 (b) A_2 and A_1 (c) A_0 and A_1 (d) A_1 and A_0

13.10 In 8253 interfacing with 8051 microcontroller, A_0 and A_1 pins of 8253 are connected with ____ pins of 8051 respectively.
(a) A_1 and A_0 (b) A_0 and A_1 (c) A_0 and A_2 (d) A_1 and A_2

Answers to Multiple-Choice Questions

13.1 (c)	13.2 (a)	13.3 (a)	13.4 (a)
13.5 (b)	13.6 (a)	13.7 (d)	13.8 (a)
13.9 (a)	13.10 (b)		

CHAPTER

14

8259 Interfacing with 8085, 8086 and 8051 Microcontroller

14.1 INTRODUCTION TO PROGRAMMABLE INTERRUPT CONTROLLER

A microprocessor-based system design requires many I/O devices such as keyboards, displays, sensors and other components. These devices should receive servicing in an efficient manner from the CPU. The most common method of servicing such devices is known as the *polled approach*. In this approach, the processor must test each device in sequence and find the device, which requires servicing. For this, a large portion of the

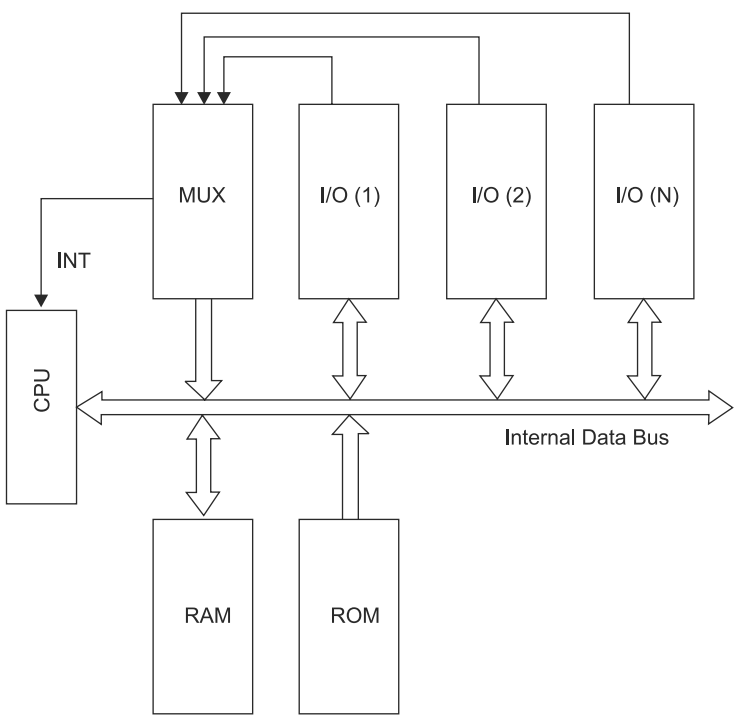


Fig. 14.1 Interrupt-driven CPU using MUX

main program is looped through this continuous polling cycle, and such a method has a serious detrimental effect on system throughput, thus limiting the tasks that could be assumed by the microprocessor and reducing the cost-effectiveness of using such devices. The other most desirable method is that the microprocessor can execute its main program and only stop to service peripheral devices when the CPU receives a signal from the device itself.

Then the processor should complete whatever instruction is currently being executed and fetch a new routine that will service the requesting device. However, after completion of service, the processor would resume exactly where it left off. This method is known as *interrupt*. Interrupts are used in a microcomputer system for different applications. When the number of I/O devices are less, the already available interrupts of microprocessors are sufficient and there is no requirement of Programmable Interrupt Controller as shown in Fig. 14.1.

The CPU can access many devices using interrupt signals. In multiple interrupt systems, the CPU must take care of the priorities for the interrupts and simultaneously occurred interrupts.

To overcome all difficulties a Programmable Interrupt Controller (PIC) has been designed and can be used to handle many interrupts at a time. This controller handles all simultaneous interrupt requests along with their priorities and the microprocessor will be relieved from this task. The Programmable Interrupt Controller (PIC) functions as an overall manager in an interrupt-driven system environment as depicted in Fig. 14.2. It accepts requests from the peripheral equipment, and then it determines priority value of all incoming requests and issues an interrupt to the CPU based on this determination. The 8259A Programmable Interrupt Controller can be interfaceable with 8085, 8086 and 8088 processors. The features of these devices are given below:

Features

- 8085, 8086, and 8088 compatible
- Eight-level priority controller

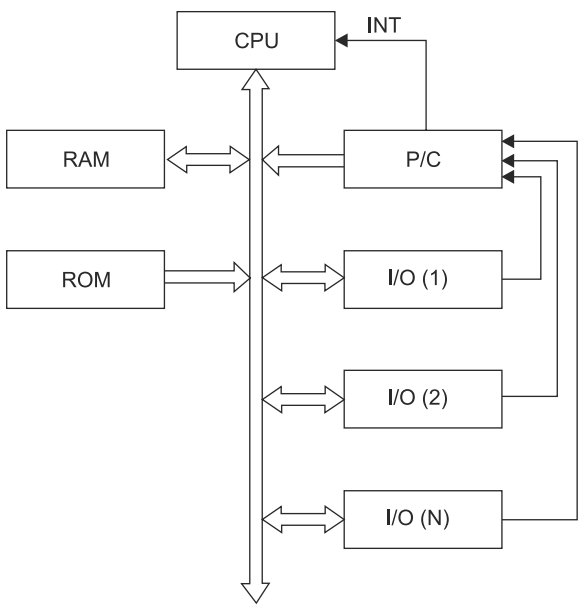


Fig. 14.2 PIC in an interrupt driven environment

- Can be expandable to 64 levels
- Programmable interrupt modes
- Individual request mask capability
- Single +5 V supply (no clocks)
- Available in 28-Pin DIP and 28-Lead
- Able to accept level-triggered or edge-triggered inputs

14.2 PIN DIAGRAM OF 8259A

The Intel 8259A Programmable Interrupt Controller handles up to eight-vectored priority interrupts for the CPU. This IC is cascadeable for up to 64-vectored priority interrupts without additional circuitry. This IC is available in a 28-pin DIP package and uses NMOS technology and requires a single 5-V supply. Circuitry is static, requiring no clock input. The schematic diagram of 8259 is depicted in Fig. 14.3. The pin diagram of 8259A is also shown in Fig. 14.4 and the pin functions are explained below:

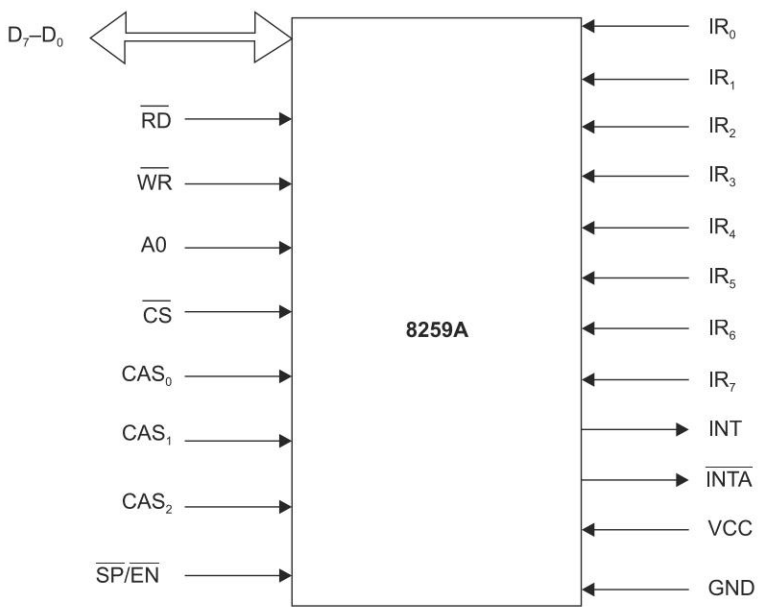


Fig. 14.3 Pin diagram of 8259

VCC 5-V supply

GND Ground

$\overline{CS}$ (Chip Select) When the chip select pin is active low, this pin enables RD and WR operation between the CPU and the 8259A. INTA functions are independent of CS.

$\overline{WR}$ (Write) A low on this pin, when CS is low, enables the 8259A for write operation. This pin also enables to accept command words from the CPU.

$\overline{RD}$ (Read) When RD is active low and CS is low, this pin enables the 8259A to release status onto the data bus for the CPU.

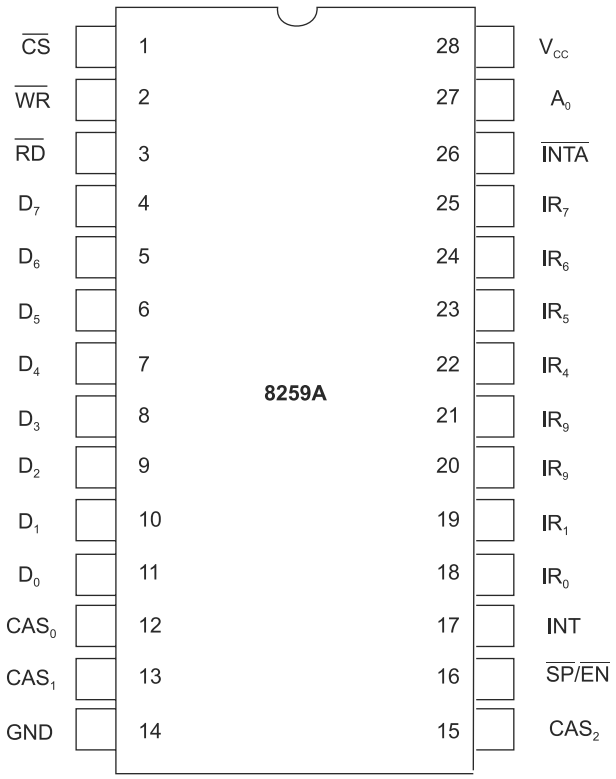


Fig. 14.4 Schematic diagram of 8259A

$D_7 - D_0$ (I/O Bi-directional Data Bus) These pins are used as bi-directional data bus. The control, status and interrupt-vector informations are transferred through this bus.

$CAS_0 - CAS_2$ (I/O Cascade Lines) A 8279A has only eight interrupts. When number of interrupts requirement is more, multiple interrupt controller must be connected in cascade. The CAS lines of a 8259A bus is used to control a multiple 8259A structure. These pins are outputs for a master 8259A and inputs for a slave 8259A.

$\overline{SP}$ / $\overline{EN}$ (I/O Slave Program/Enable Buffer) This is a dual function pin. When this IC is used in the buffered mode, it can be used as an output to control buffer transceivers (EN). If this IC is not in the buffered mode, it is used as an input to designate a master (SP = 1) or slave (SP = 0).

INT (Interrupt) This pin goes high whenever a valid interrupt request is asserted. This pin signal is used to interrupt the CPU. Therefore it is connected to the CPU's interrupt pin.

$IR_0 - IR_7$ (Interrupt Requests) These pins are used as asynchronous inputs. Each pin can be used to receive an interrupt request to the CPU by raising an IR input from low to high. The interrupt pin must be maintained high level until this is acknowledged (edge-triggered mode), or just by a high level on an IR input (level triggered mode).

$\overline{INTA}$ (Interrupt Acknowledge) This pin becomes high when a valid interrupt request is asserted. This pin is used to enable 8259A interrupt-vector data onto the data bus by a sequence of interrupt acknowledge pulses issued by the CPU.

A_0 (Address Line) This pin works in conjunction with the $\overline{CS}$, $\overline{WR}$, and $\overline{RD}$ pins. This is also used by the 8259A to read various command words the CPU writes and the status the CPU wishes to read. Generally, this is connected to the CPU A_0 address line.

14.3 FUNCTIONAL DESCRIPTION

The functional block diagram of 8259A Programmable Interrupt Controller is shown in Fig. 14.5. Each functional block has been explained below.

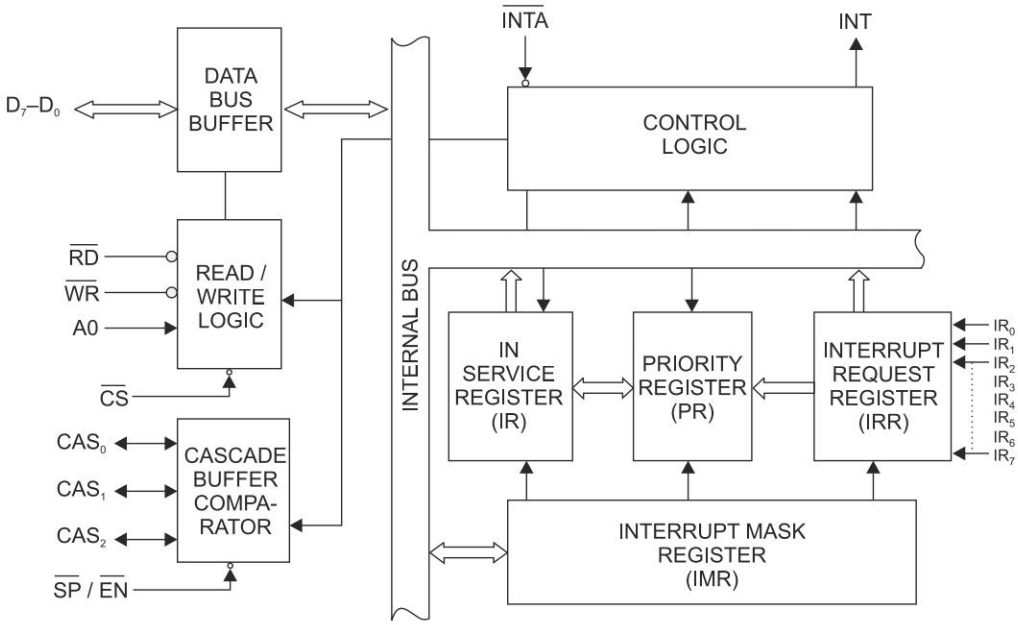


Fig. 14.5 Functional diagram of 8259A

Interrupt Request Register (IRR) The interrupts at the IR input lines are handled by two internal registers, the Interrupt Request Register (IRR) and the In-Service register (ISR). The IRR is used to store all the interrupt requests, which are requesting service and it provides service one by one on the priority basis.

In-Service Register (ISR) The In-Service Register (ISR) is used to store all the interrupt levels, which are being serviced, and also keeps a track of the request being served.

Priority Resolver This logic block determines the priorities of the interrupt requests in the IRR. The highest priority is selected and strobed into the corresponding interrupt request of the ISR during INTA pulse.

Interrupt Mask Register (IMR) This Interrupts Mask Register (IMR) stores the bits that mask the interrupt lines to be masked. The IMR operates on the IRR based priority resolver.

Interrupt Control Logic The interrupt control logic block manages the interrupt and the interrupt acknowledge signals. The Interrupt (INT) and Interrupt Acknowledge (INTA) signals are directly sent to the CPU interrupt input. The INTA signal from CPU that will cause the 8259A to release vectoring information onto the data bus.

Data Bus Buffer This 3-state, bi-directional 8-bit buffer is used to interface the 8259A to the microprocessor data bus. Control words and status information are transferred through the data bus buffer.

Read/Write Control Logic This block is used to accept output commands from the CPU. It contains the Initialization Command Word (ICW) registers and Operation Command Word (OCW) registers which store the various control formats for device operation. This function block also allows the status of the 8259A to be transferred onto the data bus.

Cascade Buffer/Comparator The cascade buffer/comparator block stores and compares the IDs of all 8259A's used in the microprocessor system. The associated three I/O pins (CAS_0, CAS_1 and CAS_2) are outputs when the 8259A is used as a master and are inputs when the 8259A is used as a slave. When the 8259A is used as a master, the 8259A sends the ID of the interrupting slave device onto the CAS_0 to CAS_2 lines. The slave thus selected, will send its preprogrammed subroutine address onto the data bus during the subsequent INTA pulses.

14.4 INTERRUPT SEQUENCE

The most powerful features of 8259A are programmability and the interrupt routine addressing capability. This device allows direct or indirect jumping to the specified interrupt service routine without any polling of the interrupting devices. The interrupt sequence for an interrupt in the 8085-microprocessor system has been explained below:

1. One or more of the INTERRUPT REQUEST lines (IR_7 to IR_0) are raised high, setting the corresponding IRR bits.
2. The 8259A evaluates these requests, and sends an INT to the CPU.
3. The CPU acknowledges the INT and responds with an INTA pulse.
4. After receiving an INTA from the CPU group, the highest priority ISR bit is set, and the corresponding IRR bit is reset. The 8259A will also release a CALL instruction code (11001101) onto the 8-bit data bus through its D_7 to D_0 pins.
5. This CALL instruction will initiate two more INTA pulses to be sent to the 8259A from the CPU group.
6. These two INTA pulses allow the 8259A to release its preprogrammed subroutine address onto the Data Bus. The lower 8-bit address is released at the first INTA pulse and the higher 8-bit address is released at the second INTA pulse.
7. This completes the 3-byte CALL instruction released by the 8259A. In the AEOI (Automatic End Of Interrupt) mode the ISR bit is reset at the end of the third INTA pulse. Otherwise, the ISR bit remains set until an appropriate EOI (End Of Interrupt) command is issued at the end of the interrupt sequence.

14.5 INTERFACING OF 8259A WITH 8085

Interfacing of 8259 with the 8085 microprocessor is illustrated in Fig. 14.6. When the 8259A PIC receives an interrupt, INT becomes active and an interrupt acknowledge cycle is started. If a higher priority interrupt occurs between the two INTA pulses, the INT line goes inactive immediately after the second INTA pulse. After an unspecified amount of time, the INT line is activated again to signify the higher priority interrupt waiting for service. This inactive time is not specified and can vary between parts. The designer should be

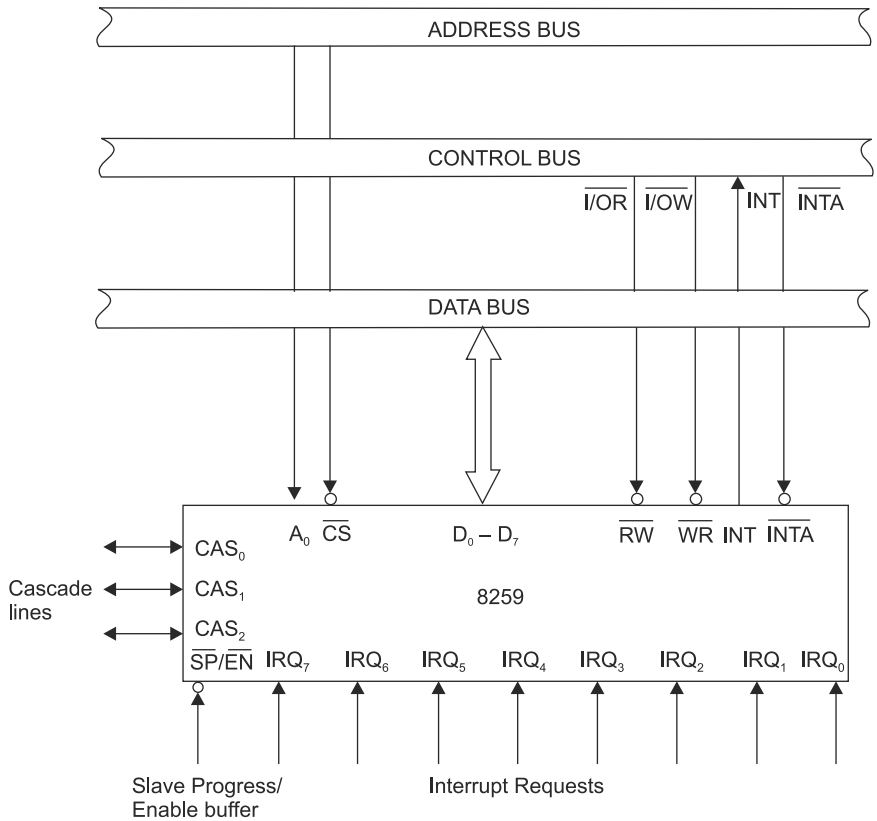


Fig. 14.6 Interfacing of 8259 with 8085 microprocessor

aware of this consideration when designing a system that uses the 8259A. It is recommended that proper asynchronous design techniques be followed.

The interfacing steps are explained below:

1. The address decoder output is connected with the $\overline{CS}$ input of the IC.
2. A_0 line is used to select one of the two internal addresses in the device. This is connected to A_0 of the address lines of the microprocessor.
3. As the device operates in I/O mapped I/O mode, the $\overline{RD}$ and $\overline{WR}$ signals are connected to $\overline{IOR}$ and $\overline{IOW}$ signals respectively.
4. The interrupt INT pin of 8259 is connected to the INTR input of the 8085 microprocessor.
5. INTA output of the processor is connected to the INTA input.
6. When $\overline{SP}/\overline{EN}$ pin is high, only one IC is used in the microprocessor-based system. If more than one IC are connected in cascade, this pin must be low.
7. CAS_0, CAS_1 and CAS_2 lines are generally opened.
8. There are eight IR input lines (IR_0–IR_7) are available. When the IR inputs are not used, they must be grounded properly to avoid noise pulse in interrupt lines.

14.6 PROGRAMMING OF 8259A

The 8259A accepts two types of command words generated by the microprocessor. These two types of command words are Initialisation Command Words (ICWs) and Operation Command Words (OCWs).

14.6.1 Initialisation Command Words (ICWs)

The 8259A programmable interrupt controller can be initialised by sending a sequence of Initialisation Control Words (ICWs) to the controller. There are four Initialisation Control Words (ICWs). The ICW_1 and ICW_2 always send to 8259 systems. When the system has any slave 8259A in the cascade node, ICW_3 must be used. In some special operations such as fully nested mode, ICW_4 can be used. All Initialisation command words are explained below:

Initialisation Command Word 1 (ICW_1) Whenever a write command is received with $A_0 = 0$ and $D_4 = 1$, it is interpreted by 8259 as Initialisation Command Word 1 (ICW_1). The ICW_1 starts the initialisation sequence during which the following automatically occur.

1. The edge sense circuit is reset, which means that following initialisation, an interrupt request (IR) input must make a low-to-high transition to generate an interrupt.
2. The Interrupt Mask Register (IMR) is cleared.
3. IR_7 input is assigned lowest priority 7.
4. The slave mode address is set to 7.
5. The Special Mask Mode is cleared and Status Read is set to IRR.
6. If IC_4 is 0, then all functions selected in ICW_4 are set to zero. Master/Slave in ICW_4 is only used in the buffered mode.

The format of ICW_1 is shown in Fig. 14.7.

Bit D_0 It indicates whether ICW_4 is needed or not. If it is 1, ICW_4 is needed and if 0, ICW_4 is not needed.

Bit D_1 When this bit is 0, then only one 8259 is in the system. If it is 1, the additional 8259 are there in the system.

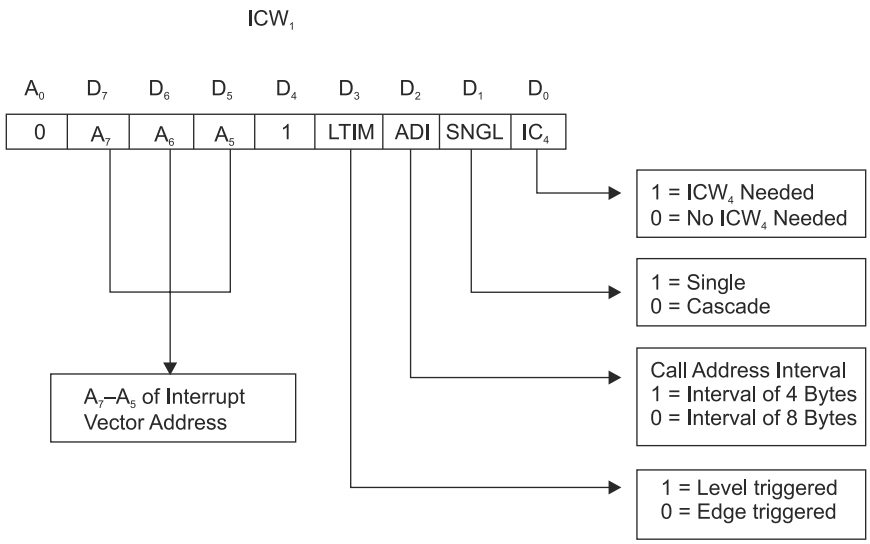


Fig. 14.7 Initialisation Command Words 1

Bit D_2 ADI stands for address interval. If this bit is '0', then call address interval is 8 and if it is '1' then call address interval becomes 4.

Bit D_3 Bit D_3 determines recognition of the interrupts either in level triggered or edge triggered mode. If this bit is 1 then the input interrupts will be recognised if they are in the level-triggered mode.

$D_5 - D_7$ These are A_5–A_7 bits as shown in Fig. 14.7. For an interval spacing of 4, A_0–A_4 bits are automatically inserted by 8259 while A_0–A_5 are inserted automatically for an interval of 8. A_5–A_7 bits are programmable as set by the bits D_5–D_7 of ICW_1.

Initialisation Command Word 2 (ICW_2) The Initialisation Command Word 2 (ICW_2) is shown in Fig. 14.8.

ICW_2

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	A_{15}/T_7	A_{14}/T_6	A_{13}/T_5	A_{12}/T_4	A_{11}/T_3	A_{10}	A_9	A_8

Fig. 14.8 Initialisation Command Words 2

Bits D_7–D_3 They specify address bits $A_{15} - A_{11}$ interrupt vector address when operating in MCS 80/85 mode.

Bits D_2–D_0 They specify address bits $A_{10} - A_8$ for the interrupt vector address when operating in MCS 80/85 mode. These bits can be set to 0 when working on an 8086 system. T_3–T_7 are interrupt vector addresses when the controller operates in 8086/8088 mode.

Initialisation Command Word 3 (ICW_3) The ICW_3 is used only when there is more than one 8259A in the system and cascading is used, in which case SNGL = 0. This will load the 8-bit slave register. The functions of this register are the following: In the master mode (either when SP = 1, or in buffered mode when M/S = 1 in ICW_4) a '1' is set for each slave in the system. The master then will release byte 1 of the call sequence (for MCS-80/85 system) and will enable the corresponding slave to release bytes 2 and 3 (for 8086 only byte 2) through the cascade lines.

In the slave mode (either when SP = 0, or if BUF = 1 and M/S = 0 in ICW_4) bits 2 ± 0 identify the slave. The slave compares its cascade input with these bits and, if they are equal, it releases bytes 2 and 3 of the call sequence for 8086 on the data bus.

Master Mode ICW_3

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

S_7–S_0 = 1 IR input has a slave and
= 0 IR –input does not have a slave

Fig. 14.9(a) Initialisation Command Words 3 (master mode)

Slave Mode ICW_3

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	ID_2	ID_1	ID_0

IR

0	1	2	3	4	5	6	7
0	1	0	1	0	1	0	1
0	0	1	1	0	0	1	1
0	0	0	0	1	1	1	1

Fig. 14.9(b) Initialisation Command Words 3 (slave mode)

Initialisation Command Word 4 (ICW_4)

The format of ICW_4 is shown in Fig. 14.10. ICW_4 is loaded only if D_0 bit of ICW_1 (IC_4) is set. The position D_0–D_4 are explained below:

Bit D_0 (MPM) MPM stands for Microprocessor Mode. MPM = 0 sets the 8259A for MCS-80, 85 system operation, but MPM =1 sets the 8259A for 8086 system operation.

Bit D_1 (AEOI) AEOI stands for the automatic end of interrupt mode. If AEOI = 1, then it is in auto EOI mode and if it is = 0, it is normal EOI mode.

Bit D_2 (M/S) When buffered mode is selected, M/S = 1 means the 8259A is programmed to be a master, and M/S = 0 means the 8259A is programmed to be a slave. If BUF = 0, M/S has no function.

Bit D_3 (BUF) The bit D_3 determines buffered/non-buffered mode of operation. If BUF = 1 the buffered mode is programmed. In buffered mode SP/EN becomes an enable output and the master/slave determination is by M/S.

Bit D_4 (SFNM) The SFNM stands for the special fully nested mode. If SFNM = 1, then this mode is programmed. In the cascaded mode of operation, when a slave receives a higher priority interrupt request than one, which is already in service (through the same slave), it would not be recognised by the master. This is happening as the master ISR bit is already in the set condition; thereby it ignores all requests of equal or lower priority. Therefore the higher priority interrupt won't be serviced until the master ISR bit is reset by an EOI command. This is most likely to happen after the completion of the lower priority routine.

If a truly fully nested structure is required within a slave 8259A, the especially fully nested mode should be used. The SFNM is programmed for the master mode only and done during master initialisation through ICW_4. In this mode, the master will ignore interrupt requests of lower priority, and will respond to requests of equal or higher priority.

The fully nested mode (FNM) is auto set after initialisation is over. In this mode all interrupt requests are arranged from highest priority (IR_0) to lowest priority (IR_7). This mode can also be changed through operation command words (OCWs). When 8259 acknowledges an interrupt request through INTR pin, the device can find out the highest priority and the corresponding bit in the Interrupt Service Register (ISR) is set. The SFNM is different from FNM and the difference between SFNM and FNM are given below.

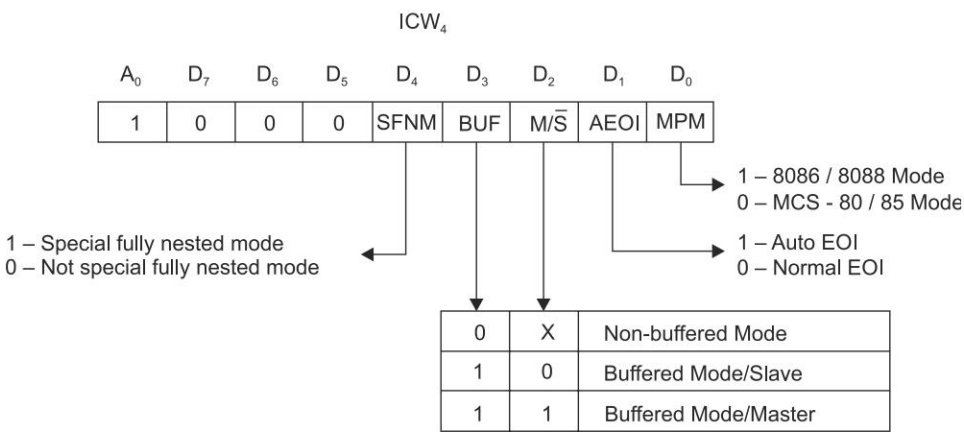


Fig. 14.10 Initialisation Command Words 4

1. In SFNM, the slave is able to place an interrupt request which has higher priority than the present interrupt being serviced. The master recognises the higher-level interrupt and can place this interrupt request to the CPU.
2. Before issuing an EOI command to the slave, the software must determine if any other slave interrupts are pending in SFNM. After that read its ISR (In Service Register). When the ISR contains all zeros, there is no interrupt from the slave in service and an EOI command can be sent to the master. If the ISR is not all zeros, an EOI command should not be sent to the master. When the master ISR bit is cleared with an EOI command while there are still slave interrupts in service, the lower priority interrupt may be recognised by the master.

14.6.2 Programming Sequence of 8259

8259 is programmed by issuing initialisation of command words and operation command words. Initialisation command words are issued in a sequence. The algorithm for initialising 8259 is as follows:

1. Write ICW_1
2. Write ICW_2
3. If 8259 does not in the cascade mode of operation, go to step 5
4. Write ICW_3
5. If ICW_4 is not required, go to step 7
6. Write ICW_4
7. Ready to accept interrupt sequence

The algorithm for initialisation 8259A programmable interrupt controller is depicted in Fig. 14.11.

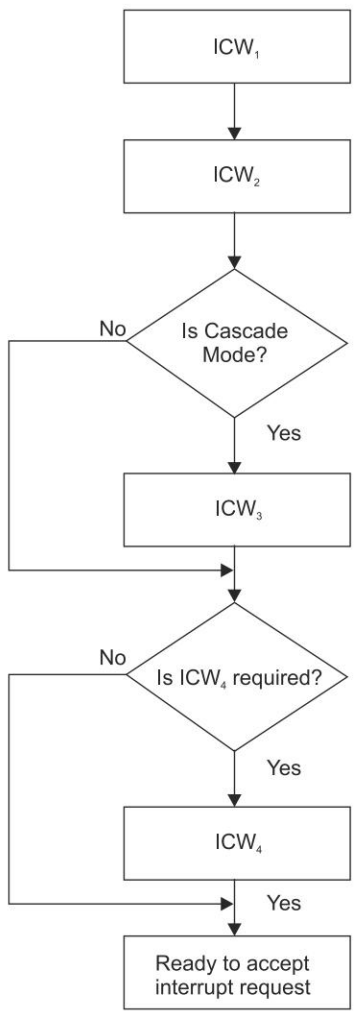


Fig. 14.11 Initialisation sequence of 8259

14.6.3 Operation Command Words (OCWs)

These are the command words which command the 8259A to operate in various interrupt modes. These operating modes are

1. Fully nested mode
2. Rotating priority mode
3. Special mask mode
4. Polled mode

There are three operation command words such as OCW_1, OCW_2 and OCW_3. These OCWs may be programmed to change the manner in which the interrupts are to be processed. The OCWs can be loaded into the 8259A anytime after initialisation.

Operation Command Word 1 (OCW_1) The format OCW_1 is shown in Fig. 14.12. OCW_1 sets and clears the mask bits by programming the Interrupt Mask Register (IMR). M_7–M_0 represents the eight mask bits. If M = 1, then corresponding Interrupt is masked (inhibited) and M = 0 indicates the interrupt is unmasked. A write command with $A_0 = 1$ is interpreted as OCW_1, and written after ICW_2.

OCW_1

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	M_7	M_6	M_5	M_4	M_3	M_2	M_1	M_0

1 = Mask Set, 0 = Mask Reset

Fig. 14.12 Operation Command Word 1

Operation Command Word 2 (OCW_2) The OCW_2 enables the user to program 8259 in different modes. The format OCW_2 is shown in Fig. 14.13. Bits 3 and 4 are always set to 0. L_2, L_1, and L_0 bits can be used to set the interrupt level acted upon which the controller must react. R, SL, EOI – these three bits control the rotate and end of interrupt modes and combinations of the two. A chart of these combinations is given below:

R	SL	EOI	Selection
0	0	0	Rotate in automatic EOI mode (CLEAR)
0	0	1	Non-specific EOI command
0	1	0	No Operation
0	1	1	Specific EOI command
1	0	0	Rotate in automatic EOI mode (SET)
1	0	1	Rotate in non-specific EOI
1	1	0	Set priority command
1	1	1	Rotate on specific EOI

The EOI command stands for End of Interrupt. The interrupt service bit may be reset by an EOI command. Generally, this command is issued by the CPU just before sending the interrupt service routine. There are two different EOI commands such as Non-specific EOI command and specific EOI command.

Non-specific EOI Command The CPU issues a non-specific EOI command. Actually, this is an OUT instruction by the CPU to 8259. The 8259 automatically determines the interrupt level, i.e. highest priority interrupt in

	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
	1	R	SL	EOI	0	0	L_2	L_1	L_0

IR Level to be acted upon

End of Interrupt	Non-specific EOI Command	0	0	1
	Specific EOI Command	0	1	1
Automatic Rotation	Rotate in NEOI Command	1	0	1
	Rotate in AEOI Mode (set)	1	0	0
	Rotate in AEOI Mode (Clear)	0	0	0
Specific Rotation	Rotate on specific EOI	1	1	1
	Set priority Ccommand	1	1	0
	No operation	0	1	0

L_2	L_1	L_0	IR No.
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Fig. 14.13 Operation Command Word 2

service and reset the correct bit in the ISR. The non-specific EOI command must be used when the most recent level acknowledged and serviced is always the highest priority level. On receiving a non-specific EOI command, 8259 simply resets the highest priority ISR bit, thus confirming to the 8259A that the highest priority routine of the routine in service is finished. The non-specific EOI command is also known as Fully Nested Mode (FNM). The advantage of the non-specific EOI command is that IR level specification is not necessary as in the specific EOI Command. But some special consideration must be taken when deciding to use the non-specific EOI.

Specific EOI Command (SEOI) An SEOI command sent from the microprocessor to 8259 lets it know when a service routine of a particular interrupt level is completed. An SEOI command resets a specific ISR bit and any one of the eight IR levels can be specified. An SEOI command is required when 8259 is unable to determine the IR level. This command is best suited for situations in which priorities of the interrupt levels are changed during an interrupt routine (Specific rotation).

Automatic EOI Mode (AEOI) In AEOI mode, no command has to be issued. Therefore, AEOI mode simplifies programming and lower code requirements within interrupt routines.

AEOI mode must be used continuously as the ISR bit of a routine presently in service is reset right after its acknowledgement. Thus it leaves no designation in the ISR that a service routine is being executed. If any interrupt request occurs during this time and interrupts are enabled, it will be serviced regardless of its priority, whether low or high. The problem of 'over nesting' may happen in this case. It occurs when an IR input keeps interrupting its own routine. This results in unnecessary stack pushes, which could fill up the stack in a worst-case condition.

Automatic Rotation—Equal Priority When several communication channels are connected to a microcomputer system, all the channels should be accorded equal priority in sharing information with the microcomputer.

In this method, once a peripheral is serviced, all other equal priority peripheral should be given a chance to be serviced before the original peripheral is serviced again. This is accomplished automatically assigning a peripheral the lowest priority after being serviced. In this way the device, presently being serviced, would have to wait until all other devices are serviced.

The automatic rotation is of two types:

1. Rotate on non-specific EOI command
2. Rotate on automatic EOI mode

Rotate on Non-specific EOI Command When the rotate on NSEOI command is issued, the highest ISR bit is reset. Just after it is reset by a non-specific EOI command, the corresponding IR level is assigned lowest priority. Figure 14.14 shows how the rotate on non-specific EOI command effects the interrupt priorities.

Let us assume that the IR_0 has the highest priority and IR_7 has the lowest priority before command as shown in Fig. 14.14(a). It is also assumed that IR_6 and IR_4 are already in service but neither is completed. As IR_4 has the highest priority, IR_4 routine be executed. When an NSEOI command is executed, Bit 4 in the ISR is reset. Then IR_4 becomes the lowest priority and IR_5 becomes the highest priority as depicted in Fig. 14.14(b).

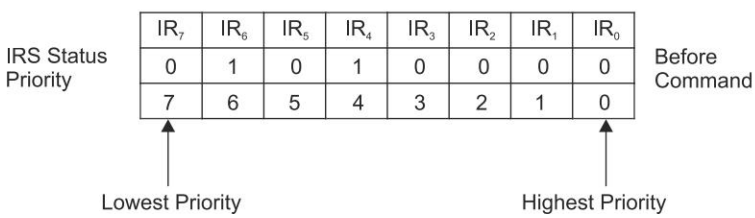


Fig. 14.14(a) Priority of interrupts before command

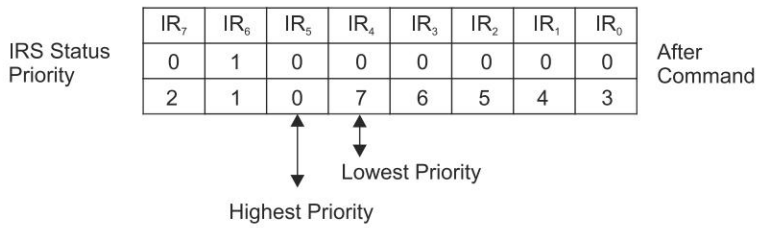


Fig. 14.14(b) Priority of interrupts after command

Rotate in Automatic EOI Mode The rotate in Automatic EOI Mode (AEOI) works like the rotate on Non-Specific EOI (NSEOI) command. The difference between NSEOI and AEOI is that priority routine is done automatically after the last $\overline{INTA}$ pulse of an interrupt request. To enter or exit from this mode, a rotate-in-automatic EOI set command and rotate-in-automatic EOI clear command are provided.

Set Priority Command The set priority command is used to assign an IR level the lowest priority. All other interrupt levels will be in the fully rested mode based on the newly assigned low priority. The relative priorities of the interrupt levels before the set priority command and after the set priority command are depicted in Fig. 14.15(a) and 14.15(b) respectively. As IR_3 has the highest priority, IR_3 routine be executed next. When the IR_3 routine is executing and set priority command is issued to the 8259A, priorities will be changed. Then IR_6 is the highest and IR_5 as the lowest priority as given in Fig. 14.15(b).

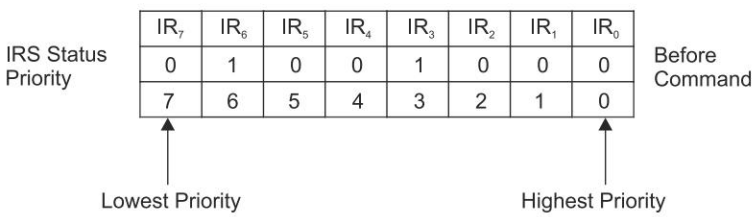


Fig. 14.15(a) Rotating interrupt priority before set priority command

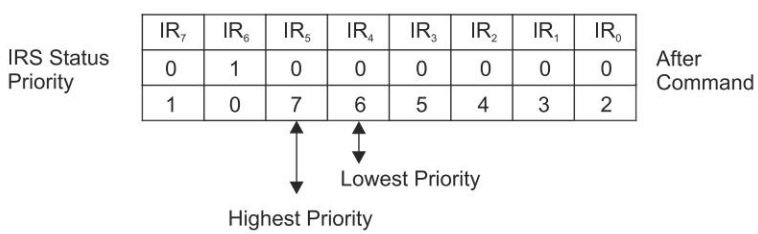


Fig. 14.15(b) Rotating interrupt priority after set priority command

Rotate on Specific EOI Command The rotate on specific EOI command is the combination of set priority command and specific EOI command. Just like the set priority command, a specified IR level will be assigned lowest priority. Similar to the specific EOI command, a specified level will be reset in the ISR. In this way the rotate on specific EOI command achieves two operations in only one command.

Operation Command Word 3 (OCW_3) The OCW_3 are used to perform the following operations:

1. To read the status of registers (IRR and ISR)
2. To set/reset the special mask and polled modes

The format of OCW_3 is shown in Fig. 14.16.

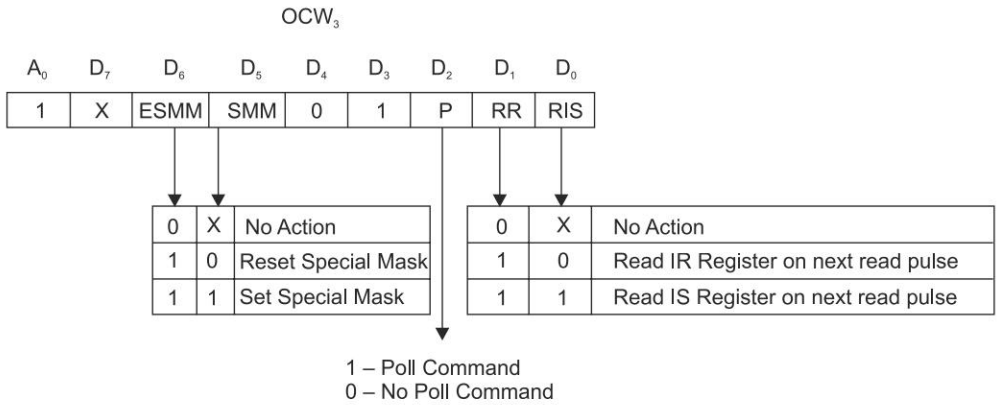


Fig. 14.16 Operation Command Word 3`

RIS This bit is used to select the In-Service Register (ISR) or the Interrupt Request Register (IRR) and then read register. When RIS = 0 and RR = 1, IRR is selected. If RIS = 1 and RR = 1, ISR is selected.

RR This bit is used to execute the read register command. When RR = 0, the read register command will not be issued and no action takes place. If RR=1, the read register command is issued and the state of RIS decide the register to be read.

P (Poll Mode) This bit is used for Poll command. When P = 1, the poll command is issued. If P = 0, the poll command is not issued. In this mode, the interrupting devices seeking services from 8085 are polled one after another and to detect which device has sought for interrupt request. In the polled mode (P = 1), 8259 is then read by masking its $\overline{RD}$ and $\overline{CS}$ pins '0'. The ISR bit is set corresponding to the highest level interrupt in the IRR. The poll word is shown in Fig. 14.17.

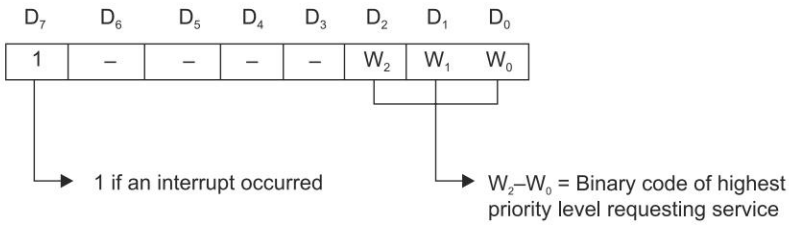


Fig. 14.17 Poll Word

Special Mask Mode (SMM) The Special Mask Mode (SMM) enables interrupt from all levels except the level currently in service. When SMM = 1, the special mask mode is selected. Once the special mask mode is set, it remains in effect until reset. If SMM = 0, the special mask mode is not selected. The special mask mode can be set when ESSM bit enables SMM (ESSM = 1) in OCW_3. The special mask mode can be cleared by loading OCW_3 with ESSM = 1 and SMM = 0.

ESMM This bit is used to enable/disable the effect of the special mask mode (SMM). When ESMM = 1, the special mask mode is enabled. If ESMM = 0, the special mask mode is disabled.

14.7 8259 INTERFACING WITH 8085 MICROPROCESSOR

Figure 14.18 shows 8259A interfacing with the 8085 microprocessor. The control lines from 8259A are D_7–D_0, $\overline{RD}$, $\overline{WR}$, A_0, *INT*, $\overline{INTA}$ and $\overline{CS}$ signals. The lines available from the 8085 microprocessor for 8259A interface are $\overline{RD}$, $\overline{WR}$, $IO/\overline{M}$, *INTR*, $\overline{INTA}$, D_7– D_0 and A_{15}– A_0 address lines. D_7– D_0, $\overline{RD}$, $\overline{WR}$, $\overline{INTA}$, are directly connected with the 8085 microprocessor. The other lines A_7– A_1 and $IO/\overline{M}$ are used to generate chip select $\overline{CS}$ signal. A_0 line of 8259 can be connected to AD_0 of 8085 microprocessor. When 80H and 81H is generated by the 8085 microprocessor, the 8259A chip will be selected. The generation of addresses 80H and 81H is given in Table 14.1.

Table 14.1 Address generation 80H and 81H for 8259A

Status of I/O Address lines								Address
A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	In Hex
1	0	0	0	0	0	0	0	80H
1	0	0	0	0	0	0	1	81H

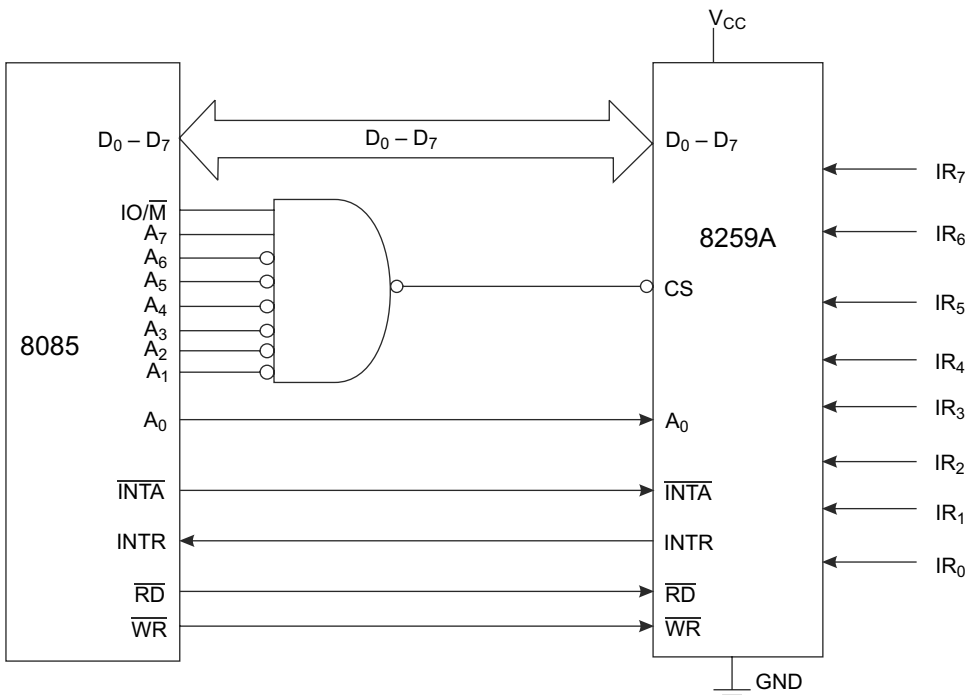


Fig. 14.18 8259A interfacing with 8085 microprocessor

14.8 8259 INTERFACING WITH 8086 MICROPROCESSOR

Figure 14.19 shows 8259A interfacing with 8086 microprocessor at the address 0740H and 0742H. The D_7–D_0 of 8259A is directly interfaced with lower byte of the 8086 data bus. A_1 line of 8086 microprocessor is connected with A_0 of 8259A. A_2 to A_5 and A_6 to A_{10} are used to generate chip select signal for 8259A. The

use of A_0 and $A_{11} - A_{15}$ is not compulsory to drive the chip select signal. When 0740H and 0742H is generated by 8086 microprocessor, the 8259A chip will be selected. The generation of address 0740H and 0742H is given in Table 14.2

Table 14.2 Address generation 0740H and 0742H for 8259A

Status of I/O Address lines																Address
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	In Hex
0	0	0	0	0	1	1	1	0	1	0	0	0	0	0	0	0740H
0	0	0	0	0	1	1	1	0	1	0	0	0	0	1	0	0742H

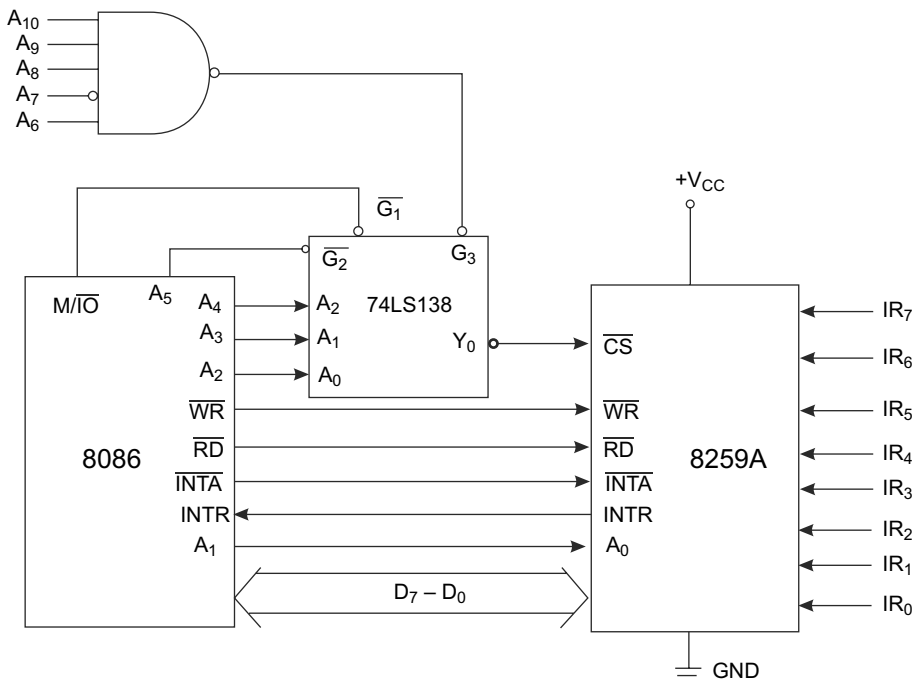


Fig. 14.19 8259A interfacing with 8086 microprocessor

14.9 8259 INTERFACING WITH 8051 MICROCONTROLLER

Figure 14.20 shows 8259A interfacing with 8051 microcontroller. The control lines from 8259A are D_7–D_0, $\overline{RD}$, $\overline{WR}$, A_0, *INT*, $\overline{INTA}$ and $\overline{CS}$ signals. The lines available from 8051 microcontroller for 8259A interface are $\overline{RD}$, $\overline{WR}$, $\overline{INT0}$, D_7–D_0 and A_{15}–A_0 address lines. D_7–D_0, $\overline{RD}$, $\overline{WR}$, and *INT* signals of 8259A are directly connected with 8051 microprocessor D_7–D_0, $\overline{RD}$, $\overline{WR}$, and $\overline{INT0}$ respectively. The $\overline{INT0}$ pin receives the external interrupt from 8259A and A_0 line of 8259 can be connected to AD_0 of 8051 microcontroller. The other lines A_7–A_0 are used to generate chip select $\overline{CS}$ signal as depicted in Fig.14.20. When 80H and 81H is generated by 8051 microcontroller, 8259A chip will be selected. The generation of addressES 80H and 81H is given in Table 14.3

Table 14.3 Address generation 80H and 81H for 8259A

Status of I/O Address lines								Address
A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	In Hex
1	0	0	0	0	0	0	0	80H
1	0	0	0	0	0	0	1	81H

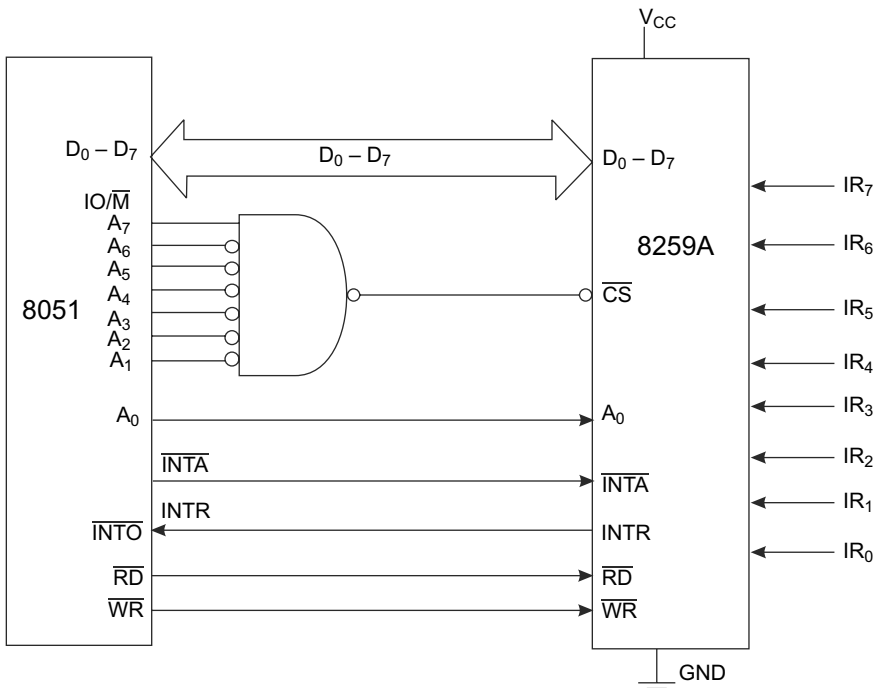


Fig. 14.20 8259A interfacing with 8051 microcontroller

Review Questions

14.1 What is a Programmable Interrupt Controller?

14.2 Draw the architecture of 8259A and explain briefly.

14.3 Write the functions of the following pins of 8259A:
(i) INT (ii) INTA (iii) IR_0–IR_7 (iv) CAS_0–CAS_2 (v) $\overline{SP}/\overline{EN}$.

14.4 How many interrupt levels can be handled by 8259?

14.5 What are ICWs and OCWs? Describe how the PIC 8259 responds to interrupts?

14.6 Describe the fully nested mode.

14.7 What are the types of rotating priority modes of interrupt? Explain any one.

14.8 Explain the interfacing of 8259A with the 8085 microprocessor.

14.9 Describe the priority scheme and EOI scheme of 8259.

14.10 Write down the format of ICW1 and ICW2 of 8259.

14.11 Write a short note on 8259 Programmable Interrupt Controller.

14.12 Draw the 8259 interfacing with 8085 microprocessor and explain the circuit operation briefly.

14.13 Design the 8259 interfacing with 8086 microprocessor.

14.14 Draw the 8259 interfacing with 8051 microcontroller and explain the circuit operation briefly.

14.15 What are the major components of 8259 interrupt controller? Explain their functions.

Multiple-Choice Questions

14.1 8259 is a

(a) programmable interrupt controller (b) DMA controller

(b) programmable keyboard display interface (d) programmable counter

14.2 8259 can handle

(a) 8-vectored priority interrupt controller (b) 18-vectored priority interrupt controller

(c) 16-vectored priority interrupt controller (d) 6-vectored priority interrupt controller

14.3 In cascade mode, 8259 can handle

(a) up to 64-vectored priority interrupts (b) up to 46-vectored priority interrupts

(c) up to 60-vectored priority interrupts (d) up to 40-vectored priority interrupts

14.4 The logic block which determines the priorities of the interrupt requests in the IRR is called

(a) Priority resolver (b) Interrupt mask register

(c) Interrupt Request Register (d) Cascade Buffer

14.5 The _____ pin signal is used to interrupt the CPU

(a) INT (b) $\overline{INTA}$ (c) $\overline{CS}$ (d) $\overline{RD}$

14.6 In 8259 interfacing with 8085 microprocessor, A_0 pin of 8259 is connected with ____ pin of 8085.

(a) A_0 (b) A_1 (c) A_2 (d) A_3

14.7 During 8259 interfacing with 8086 microprocessor, A_0 pin of 8259 is connected with ____ pin of 8086.

(a) A_0 (b) A_1 (c) A_2 (d) A_3

14.8 In 8259 interfacing with 8051 microcontroller, A_0 pin of 8259 is connected with ____ pin of 8051 microcontroller.

(a) A_0 (b) A_1 (c) A_2 (d) A_3

Answers to Multiple-Choice Questions

14.1 (a)	14.2 (a)	14.3 (a)	14.4 (a)
14.5 (a)	14.6 (a)	14.7 (b)	14.8 (a)

CHAPTER

15

8279 Interfacing with 8085, 8086 and 8051 Microcontroller

15.1 INTRODUCTION

In any microprocessor-based system, the keyboard is most commonly used as an input device and a seven-segment display is used as the output device. The programmer presses the keys on the keyboard as per desired to feed instruction or data to the CPU. Therefore, the key board is constantly scanned to detect a pressed key. The display section be also constantly supplied with data to hold it steady, while CPU operates as scan key and displays it. The CPU will be heavily loaded and system operation becomes slow, as less time will be available for data processing or ALU operations. If a specific IC performs these operations, then CPU can handle data processing or ALU operations very efficiently.

The 8279 is a general-purpose programmable keyboard and display I/O interface device designed for use in microprocessors. The keyboard portion can provide a scanned interface to a 64-contact key matrix. The keyboard section can also be interfaced to an array of sensors or a strobed interface keyboard. Key depressions can be 2-key lockout or *N*-key rollover. Keyboard entries are debounced and strobed in an 8-character FIFO. When more than 8 characters are entered, the overrun status is set. Key entries set the interrupt output line to the CPU.

The display portion provides a scanned display interface for LED, or any popular display devices. Both numeric and alphanumeric segment displays may be used as well as simple indicators. The 8279 has a 16 × 8 display RAM. This 16 × 8 display can be organised into dual 16 × 4. The CPU can load the RAM. Both right entry calculator and left entry typewriter display formats are possible. Both read and write of the display RAM can be done with auto-increment of the display RAM address. The features of 8279 are given below:

- Simultaneous keyboard and display operations
- Scanned keyboard mode
- Scanned sensor mode
- Strobed input entry mode
- 8-character keyboard FIFO
- 2-key lockout or *N*-key rollover with contact debounce
- Dual 8- or 16-numerical display
- Single 16-character display

- Right or left entry 16-byte display RAM
- Mode programmable from CPU
- Programmable scan timing
- Interrupt output on key entry
- Different Display modes
  - 8-8 bit Character Display-left entry
  - 16-16 bit Character Display-left entry
  - 8-8 bit Character Display-right entry.
  - 16-16 bit Character Display-right entry.

15.2 PIN DIAGRAM OF 8279

The 8279 is packaged in a 40-pin DIP. The pin configuration of 8279 is depicted in Fig.15.1. The schematic diagram of 8279 is shown in Fig. 15.2. The following is a functional description of each pin.

DB_0–DB_7 (Bi-Directional Data Bus) These are bidirectional data bus. All data and commands between the CPU and the programmable keyboard interface 8279 are transferred on these lines.

CLK (Clock) Generally, a system clock is used to generate internal timing.

RESET A high signal on this pin is used to reset the 8279. After being reset, the 8279 is placed in the following mode:

- 16 8-bit character display—left entry
- Encoded scan keyboard—2 key lockout
- The clock prescaler is set to 31

$\overline{CS}$ (Chip Select) A low on this pin enables the programmable keyboard interface, 8279 to receive or transmit data.

A_0 (Buffer Address) A high on this line indicates that the signals in or out are interpreted as a command or status. A low on this line indicates that they are data.

RD (Read) This output signal is activated from microprocessor to 8279 to receive data from external bus.

WR (Write) This signal enables the data buffers to send data to the external bus.

IRQ (Interrupt Request) In keyboard mode, the interrupt line is high when there is data in the FIFO/ Sensor RAM. The interrupt line becomes low with each FIFO/Sensor RAM read and returns high if there is some still information in the RAM. In sensor mode, the interrupt line goes high whenever a change in a sensor is detected.

SL_0–SL_3 (Scan Lines) Scan lines which are used to scan the key switch or sensor matrix and to select the display digits. These lines can be either encoded or decoded mode.

RL_0–RL_7 (Return Line) These are return line inputs. These are connected to the scan lines through the keys or sensor switches. They have active internal pull-ups to keep them high until switch closures are pulled down. The switches are connected between the scan lines and return lines. These lines also serve as an 8-bit input in the Strobed Input mode.

Shift (Shift) The shift input status is stored along with the key position on key closure in the scanned keyboard modes. Till a switch closure pulled low, it has an active internal pull up to keep it high.

CNTL/STB (Control/Strobed Input Mode) In keyboard modes, CNTL/STB line is used as a control

input and stored like status on a key closure. The line can be used as the strobe line that enters the data into the FIFO in the strobed input mode. It has an active internal pull-up to keep it high until the line is pulled down with a switch closure.

OUT A_0–OUT A_3 and OUT B_0–OUT B_3 (OUTPUTS) These are the output ports for the 16 × 4 display refresh registers. The data from these outputs is synchronised to the scan lines (SL_0–SL_3) for multiplexed digit displays. The two 4 bit ports may also be used as one 8-bit port. These two ports may be blanked independently.

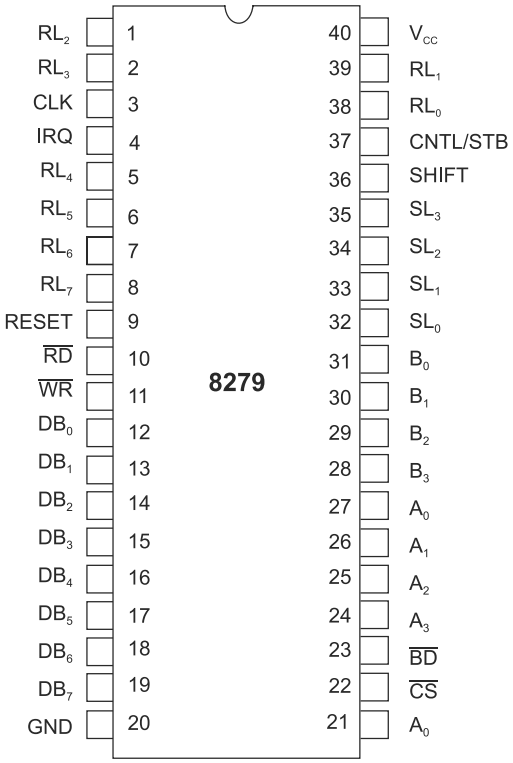


Fig. 15.1 Pin diagram of 8279

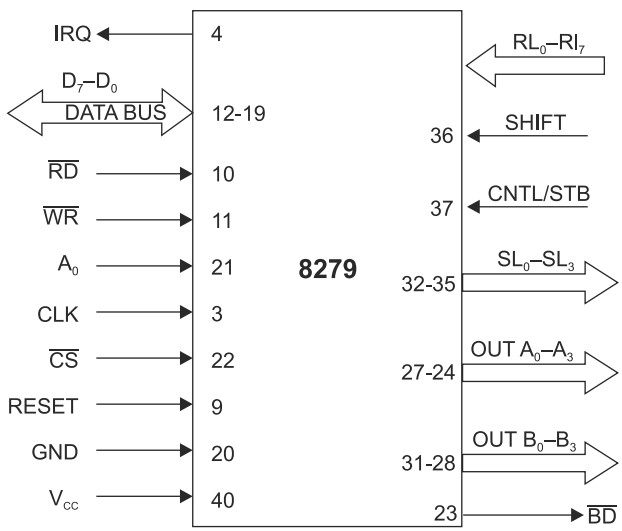


Fig. 15.2 The schematic diagram of 8279

$\overline{BD}$ (Blank Display) This output pin is used to blank the display during digit switching or by a blanking command.

15.3 FUNCTIONAL DESCRIPTION

As data input and display are an integral part of all microprocessor designs, the system designer needs an interface that can control these functions without placing a large load on the CPU. The 8279 provides this function for 8-bit microprocessors. The 8279 has two sections: *keyboard section* with a set of four scan lines and eight return lines and *display section* with a set of eight output lines for interfacing. The functional block diagram is shown in Fig. 15.3 and the description of the major elements of the 8279 Programmable Keyboard/Display interface device is given below.

I/O Control and Data Buffers The I/O control section uses the $\overline{CS}$, $\overline{WR}$, $\overline{RD}$ and A_0 lines to control data flow to and from the various internal registers and buffers. All data flow to and from the 8279 is enabled by $\overline{CS}$. The character of the information, given or desired by the CPU, is identified by A_0. A logic one means that the information is a command or status, but a logic zero means the information is data. $\overline{RD}$ and $\overline{WR}$ select the direction of data flow through the data buffers. The data buffers are bi-directional buffers that connect the internal bus to the external bus. When the chip is not selected ($\overline{CS}$ =1), the devices are in a high impedance state. The drivers input during $\overline{WR}$, $\overline{CS}$ and output during $\overline{RD}$, $\overline{CS}$.

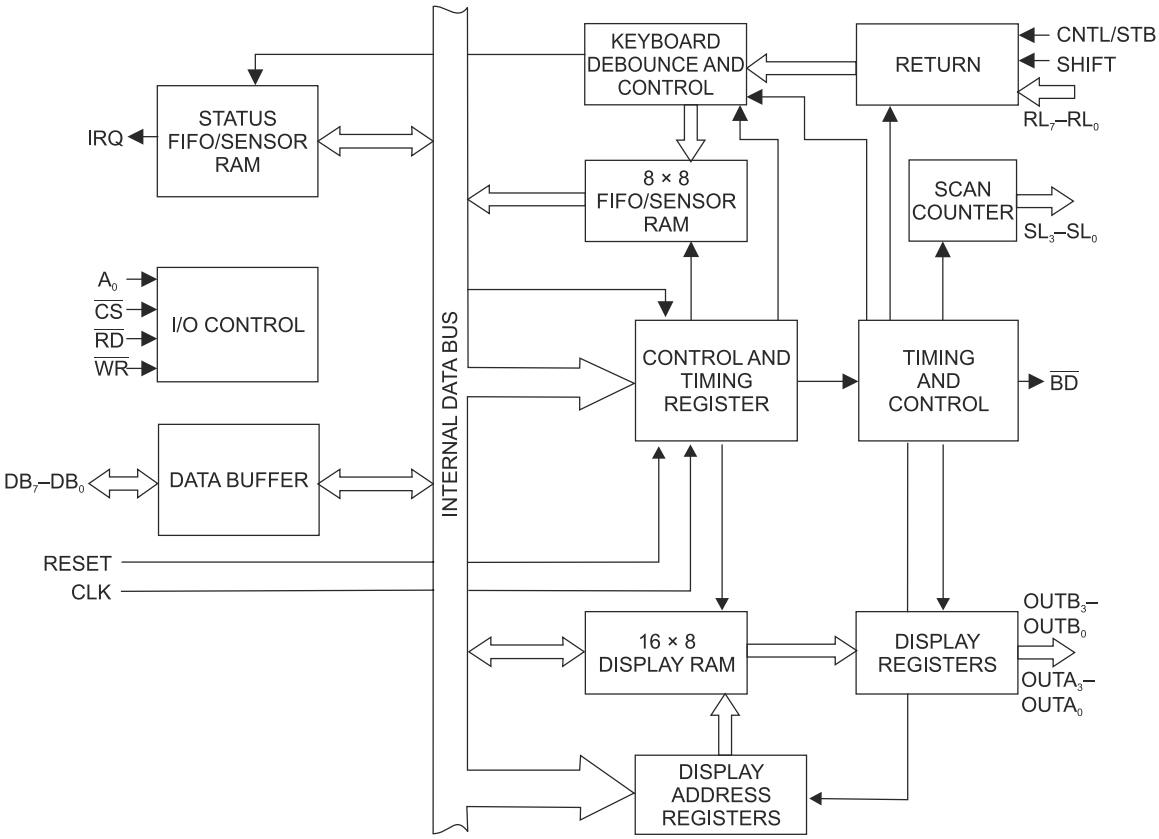


Fig. 15.3 Functional block diagram of 8279

Control and Timing Registers and Timing Control These registers store the keyboard and display modes and other operating conditions programmed by the CPU. When A_0 = 1, the modes are programmed by presenting the proper command on the data lines and then sending a $\overline{WR}$. The command is latched on the rising edge of $\overline{WR}$. The command is then decoded and the appropriate function is set. The timing control unit controls the basic timings of counter chain. The first counter is a $\div N$ pre-scaler that can be programmed to yield an internal frequency of 100 kHz which provides a 5.1 ms keyboard scan time and a 10.3 ms debounce time. The other counters divide down the internal operating frequency of 8279 to provide the proper key scan, row scan, keyboard scan, and display scan times.

Scan Counter The scan counter has two modes such as encoded mode and decoded mode. In the encoded mode, the counter provides a binary count that must be externally decoded to provide the scan lines for the

keyboard and display. In the decoded mode, the scan counter decodes the least significant 2 bits and provides a decoded 1 of 4 scan on SL_0–SL_3 while the keyboard is in decoded scan, it can display. This means that only the first 4 characters in the Display RAM are displayed.

In the encoded mode, the scan lines are active high outputs. But in the decoded mode, the scan lines are active low outputs.

Return Buffers and Keyboard Debounce and Control The 8 return lines are buffered and latched by the return buffers. In the keyboard mode, these lines are scanned for key closure in row-wise when the debounce circuit detects a closed switch, it waits about 10 ms to check if the switch remains closed. If the switch is closed, the address of the switch in the matrix, the status of SHIFT and CONTROL are transferred to the FIFO. In the scanned sensor matrix mode, the contents of the return lines are directly transferred to the corresponding row of the Sensor RAM (FIFO) during each key scan. In the strobed input mode, the contents of the return lines are transferred to the FIFO on the rising edge of the CNTL/STB.

FIFO/Sensor RAM and Status In the keyboard or strobed input mode, this block is a dual function 8 × 8 RAM and it operates in FIFO. Each new entry is written into successive RAM positions and then can be read in order of entry. FIFO status keeps track of the number of characters in the FIFO and whether it is full or empty. Too many reads or writes will be recognised as an error. The status can be read by an $\overline{RD}$ with $\overline{CS}$ low and A_0 high. The status logic also provides an IRQ signal when the FIFO is not empty. In scanned sensor matrix mode, the unit acts as a sensor RAM. Each row of the sensor RAM is loaded with the status of the corresponding row of sensor in the sensor matrix. In this mode, IRQ is high if any change in a sensor is detected.

Display Address Registers and Display RAM The display address registers hold the address of the word currently being written or read by the CPU and the two 4-bit nibbles can be displayed. The read/write addresses are programmed by CPU command. The address can be automatically updated after each read or write operation. Then CPU can directly read the display RAM after the address is set. The addresses for the A and B nibbles are automatically updated by the 8279 to match data entry by the CPU. The A and B nibbles can be entered independently or as one word, depending upon the mode set by the CPU. Data entry to the display can be set to either left or right entry.

15.4 OPERATING MODES OF 8279

The 8279 is designed to directly connect to the microprocessor bus. Then CPU can program all operating modes for the 8279. The 8279 operates in input (keyboard) modes and output (display) modes.

15.4.1 Input (Keyboard) Modes

8279 has three input modes such as scanned keyboard, scanned sensor matrix, and strobed input.

Scanned Keyboard In this mode, 8279 can be encoded (8 × 8 key keyboard) or decoded (4 × 8 key keyboard) scan lines. A key depression generates a 6-bit encoding of key position. Position and shift and control status are stored in the FIFO. Keys are automatically debounced with 2-key lockout or *N*-key rollover.

Scanned Sensor Matrix In this mode, a sensor array will be interfaced with 8279 with encoded (8×8 matrix switches) or decoded (4 × 8 matrix switches) scan lines. Key status are stored in RAM addressable by CPU.

Strobed Input Data on return lines during control line strobe is stored in the FIFO.

15.4.2 Output (Display) Modes

8279 provides two output modes such as display scan and display entry.

Display Scan In this mode, Programmable Key Board and Display Controller, 8279 provides 8 or 16 character multiplexed displays that can be organised as dual 4-bit or single 8-bit ($B_0 = D_0$, $A_3 = D_7$) display unit.

Display Entry Right entry or left entry display formats are executable for 8279 IC.

15.5 SOFTWARE OPERATION

8279 Commands The following commands program the 8279 operating modes. The commands are sent on the Data Bus with $\overline{CS} = 0$ and $A_0 = 1$ and are loaded to the 8279 on the rising edge of $\overline{WR}$. All commands of 8279 are discussed below.

Keyboard/Display Mode Set The command word format to select different modes of operation of 8279 is given below:

MSB							LSB
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	D	D	K	K	K

where DD is the Display Mode and KKK is the Keyboard Mode.

D D	Display mode
0 0	Eight 8-bit character display—Left entry
0 1	Sixteen 8-bit character display—Left entry
1 0	Eight 8-bit character display—Right entry
1 1	Sixteen 8-bit character display—Right entry

K K K	Keyboard modes
0 0 0	Encoded Scan Keyboard—2 Key Lockout
0 0 1	Decoded Scan Keyboard—2-Key Lockout
0 1 0	Encoded Scan Keyboard—*N*-Key Rollover
0 1 1	Decoded Scan Keyboard—*N*-Key Rollover
1 0 0	Encoded Scan Sensor Matrix
1 0 1	Decoded Scan Sensor Matrix
1 1 0	Strobed Input, Encoded Display Scan
1 1 1	Strobed Input, Decoded Display Scan

Program Clock The clock for operation of 8279 is programmable. All timing signals are generated by an internal prescaler, which divides the external clock by a programmable integer. Bits PPPPP determine the value of the integer from 2 to 31. When the system clock frequency is 2 MHz, it is divided by 20 (10100) to get the clock frequency 2/20 MHZ or 100 KHz.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	1	P	P	P	P	P

Read FIFO/Sensor RAM The command format of Read FIFO/Sensor RAM is given below:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	1	0	AI	x	A	A	A

X = Don't Care, AI—Auto-Increment flag, AAA—Address pointer to 8 bit FIFO RAM

Initially, the command word must be written and the CPU sets the 8279 for a read of the FIFO/Sensor RAM. In the scan keyboard mode, the Auto-Increment flag (AI) and the RAM address bits (AAA) are irrelevant. The 8279 will automatically drive the data bus for each subsequent read ($A_0 = 0$) in the same sequence in which the data first entered the FIFO. When AI flag is set, each subsequent read will be from the FIFO until another command is issued.

In the sensor matrix mode, the RAM address bits AAA select one of the 8 rows of the Sensor RAM. While the AI flag is set (AI = 1), each successive read will be from the subsequent row of the sensor RAM.

Read Display RAM To read the display RAM data, the command format is shown below:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	1	1	AI	A	A	A	A

AI—Auto-Increment flag, and AAAA— 4 bit Address for 16-byte display RAM.

Firstly, the CPU writes this command to the 8279 for a read of the Display RAM. The address bits AAAA are used to select one of the 16 rows of the Display RAM. When the AI flag is set (AI = 1), this row address will be incremented after each following read *or* write to the Display RAM. As the same counter is used for both reading and writing, this command sets the next read or write address. The Auto-Increment mode for both read and write operations.

Write Display RAM To write the display RAM data, the command format is given below:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	AI	A	A	A	A

AI—Auto-Increment flag, and AAAA— 4 bit Address for 16-byte display RAM to be written.

By writing this command, the CPU sets up the 8279 for a write to the Display RAM. After writing the command with $A_0 = 1$, all subsequent writes with $A_0 = 0$ will be to the Display RAM. The addressing and Auto-Increment functions are identical to those for the Read Display RAM.

Display Write Inhibit/Blanking This command format enables the programmer to display write inhibit/blanking operation.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	1	x	IW	IW	BL	BL
Output nibbles				A	B	A	B

The IW Bits can be used to mask nibble A and nibble B in case of separate 4-bit display ports. The output lines are divided into two nibbles (OUT A_0–OUT A_3) and (OUT B_0–OUT B_3). By setting the IW flag (IW = 1) for one of the ports, the port becomes marked so that entries to the Display RAM from the CPU do not affect that port. As a result, each nibble is input to a BCD decoder, and the CPU may write a digit to the Display RAM without affecting the other digit being displayed. In this case, the bit B_0 corresponds to bit D_0 on the CPU bus, and the bit A_3 corresponds to bit D_7.

If the user requirements is to blank the display, the BL flags are available for each nibble. Both BL bits will be cleared for blanking both nibbles.

Clear Display RAM This command format for clear display RAM operation is given below:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	1	CD_2	CD_1	CD_0	CF	CA

The CD_2, CD_1, and CD_0 bits are available in this command to clear all rows of the Display RAM to a selectable blanking code as given below:

CD_2	CD_1	CD_0	
1	0	x	All zeros x don't care
1	1	0	A_3–A_0 = 2(0010) and B_3–B_0 = 0(0000)
1	1	1	All ones

CD_2 must be 1 for enabling the clear display command. When CD_2 = 0, the clear display command is invoked by setting CA = 1 and CD_1, CD_0 bits must be same.

If the CF bit is 1, the FIFO status is cleared and the interrupt output line is reset. Also, the Sensor RAM pointer is set to row 0.

If the Clear All bit (CA) set to 1, this signal combines the effect of CD and CF bits; This CA uses the CD clearing code on the Display RAM and also clears FIFO status.

End Interrupt/Error Mode Set

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	1	E	x	x	x	x

For the sensor matrix modes, this command lowers the IRQ line and enables further writing into RAM. Therefore if any in sensor value is detected, the IRQ line becomes high which inhibits writing into the RAM.

For the *N*-key rollover mode, if the E bit is programmed to "1", the 8279 IC can operate in the special error mode.

Data Format In the scanned keyboard mode, the character entered into the FIFO corresponds to the position of the switch in the keyboard plus the status of the CNTL and SHIFT lines. The scanned keyboard data format is illustrated below:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CNTL	SHIFT	SCAN			RETURN		

CNTL is the MSB of the character and SHIFT is the next most significant bit. The next three bits D_5–D_3 are from the SCAN counter and indicate the position of the row the key was found in. The remaining three bits D_2–D_0 are from the column counter and indicate the position of the column on which the key is pressed.

In scanned sensor matrix mode, the data on the return lines (RL_7–RL_0) is entered directly in the row of the Sensor RAM that corresponds to the row in the matrix being scanned. Therefore, each switch position maps directly to a Sensor RAM position. The SHIFT and CNTL inputs are ignored in this mode and switches are not necessarily the only things that can be connected to the return lines. Any logic that can be triggered by the scan lines can enter data to the return line inputs. Eight multiplexed input ports could be tied to the return lines and scanned by the 8279.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
RL_7	RL_6	RL_5	RL_4	RL_3	RL_2	RL_1	RL_0

Display

Left Entry The left-entry mode is the simplest display format. This mode is just like typewriter mode. In this mode, each display position directly corresponds to a byte in the display RAM. The first entry goes to the leftmost display position of the display RAM. The second entry is to the right of the first one. In this way, the 16th entry goes to the 15th address position of the Display RAM. Entering characters from position zero causes the display to fill from the left. Therefore, the 17th entry goes to the Display RAM address 0 and 18th entry goes to Display RAM address 1 as depicted in Fig.15.4.

Right Entry The right-entry mode is most commonly used in electronic calculators. The first entry is placed in the rightmost display character of the Display RAM. The second entry goes to the rightmost character after the display is shifted left one character as shown in Fig. 15.5. In this way, the 17th entry goes to the Display RAM address 0

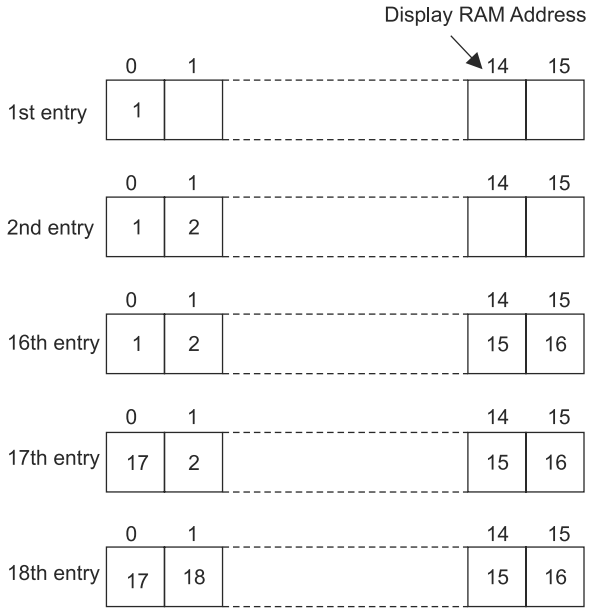


Fig. 15.4 Left-mode entry (auto increment)

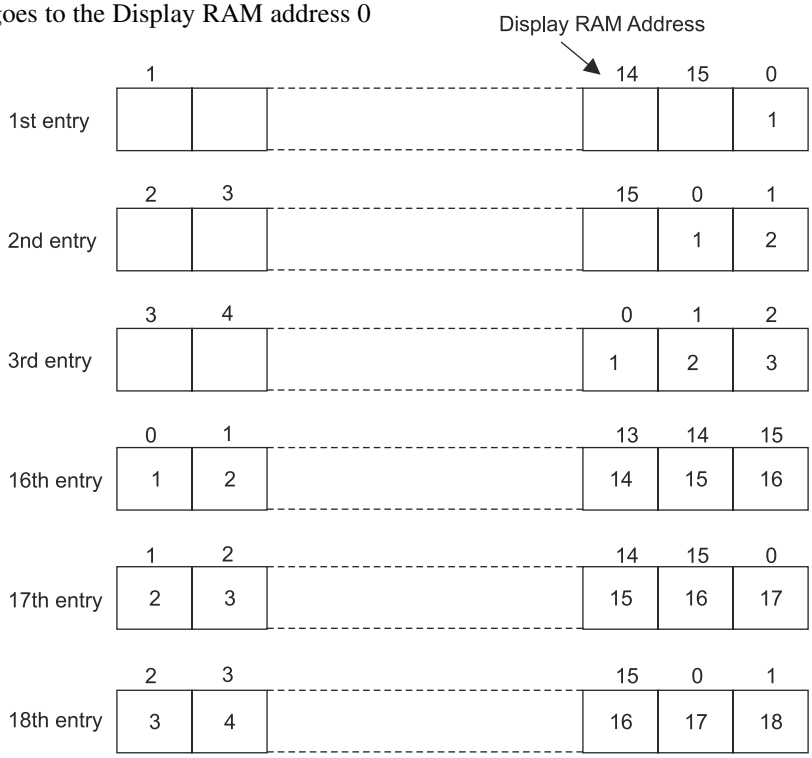


Fig. 15.5 Right-mode entry (auto increment)

and the 18th entry goes to Display RAM address 1 and the leftmost character is shifted off the end and is lost. In this mode, there is no correspondence between Display RAM address and the display position.

15.6 INTERFACING 8279 WITH 8085 MICROPROCESSOR

The 8279 can be used as a memory mapped I/O or as an I/O mapped I/O device. The interfacing of 8279 with microprocessor in I/O mapped I/O mode is depicted in Fig. 15.6. This circuit consists of 8 data lines, RESET, $\overline{RD}$, $\overline{WR}$, $\overline{CS}$, CLK and $C/\overline{D}$ which are explained below:

- The 8 data lines are connected to the data bus of the microprocessor.
- The RESET signal is connected to the RESET OUT of the microprocessor.
- $\overline{RD}$ and $\overline{WR}$ signals are connected to $\overline{IOR}$ and $\overline{IOW}$ control signals.
- The address decoder output is also connected to the $\overline{CS}$ pin of 8279 in order to access the IC as an I/O mapped device. When the $\overline{CS}$ signal is active low, the device can communicate with microprocessor. The address of data port is 30H and the address of command port is 31H.
- The command/data C/$\overline{D}$ signal is connected to A_0 address line of the microprocessor for addressing data register and command register sequentially.
- The clock signal, CLK is linked to the system clock.

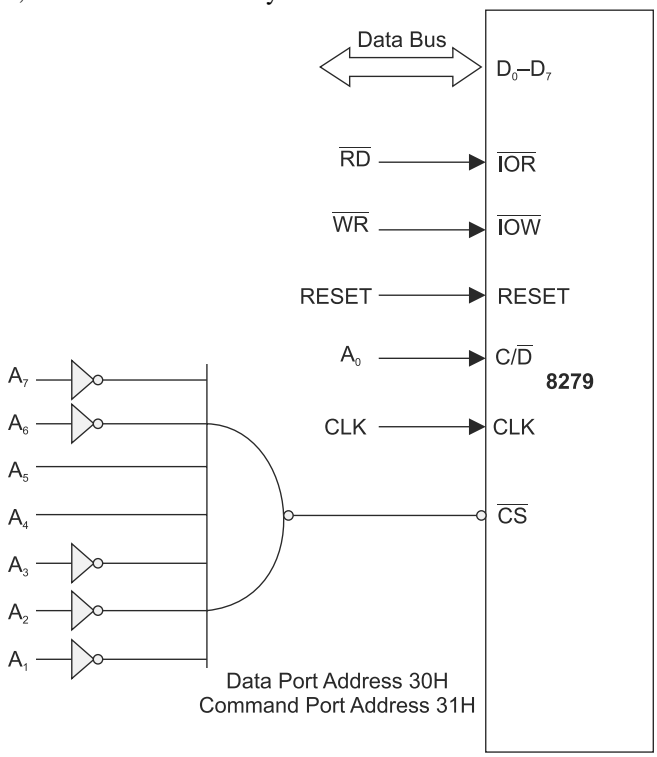


Fig. 15.6 Interfacing of 8279 with 8085 microprocessor

15.7 KEYBOARD INTERFACE OF 8279

The keyboard interface of 8279 is depicted in Fig. 15.7. To recognise the keyboard data, the device has a keyboard buffer RAM. The scan lines S_0, S_1 and S_3 are used as inputs of 3 lines to 8 lines decoder. The decoder

output lines drive eight rows of keys. The return lines RL_0–RL_7 are used as column lines for the keyboard. The 8 row and 8 column signals can enable the 8279 to interface a 64 key keyboard.

Sometimes, one key may have more than one function. The shift and control keys are generally used for this purpose. When any key and either shift or control or both are pressed together, three more function keys are available from a key. In this way, 64 keys are able to provide 256 functions. For this purpose, control and shift lines take care of the control and shift keys. Once a key is pressed, an 8-bit code must be loaded into the buffer RAM and the interrupt request send a signal to the microprocessor which can indicate that the buffer RAM is not empty.

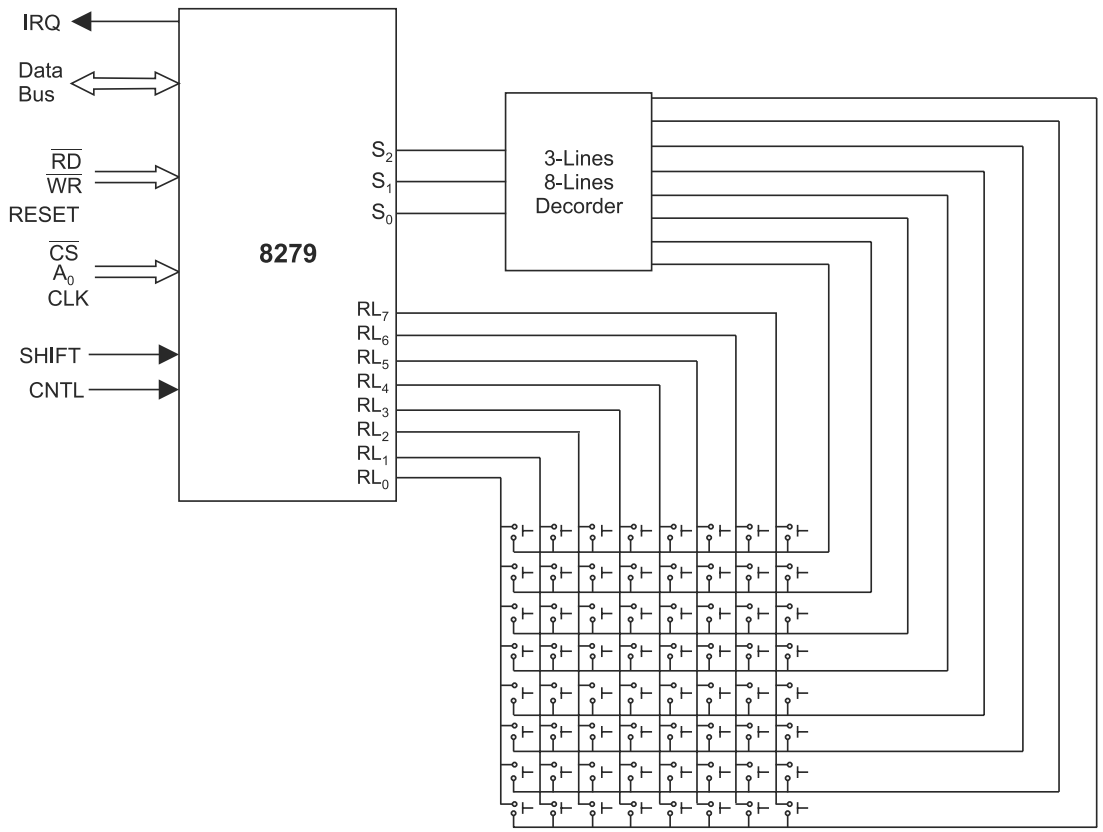


Fig. 15.7 Keyboard interface with 8279

15.8 SIXTEEN-DIGIT DISPLAY INTERFACE OF 8279

Figure 10.8 shows the sixteen-digit display using 8279. This consists of a sixteen 8-bit buffer RAM to hold the 8-bit data for 16 characters or digits. All display digits are multiplexed by four lines to sixteen lines decoder. Therefore, all digits are not turned ON at a time, but they are ON sequentially. The four scan lines output 0 to 15 in a cyclic manner, which can be decoded into 16 different lines by the decoder to select any one digit of the 16-character display. The lines A_0–A_3 and B_0–B_3 send the 8-bit information for display. The $\overline{BD}$ is used to blank all display digits. If the first digit is selected for display, the content of the first display buffer will be placed in A_0–A_3 and B_0–B_3 lines.

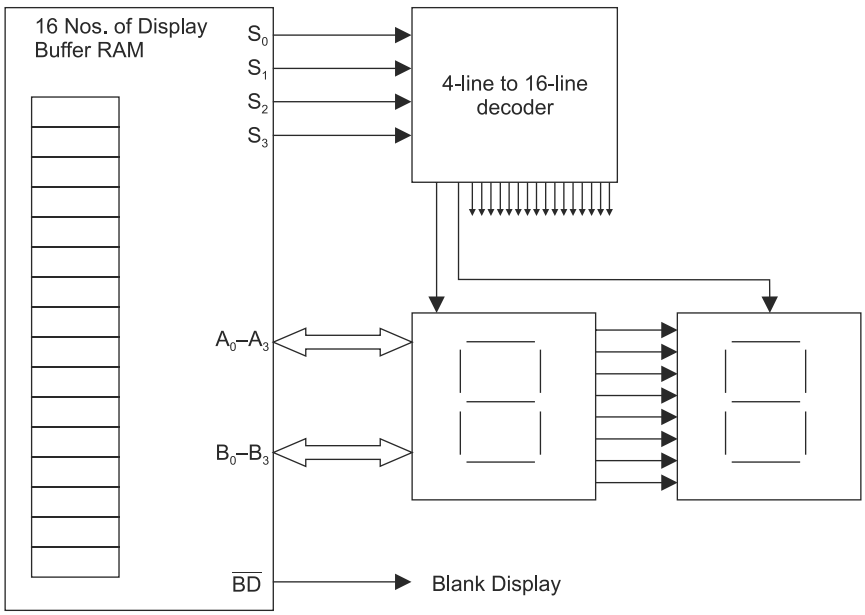


Fig. 15.8 Sixteen-digit display interface with 8279

15.9 8279 INTERFACING WITH 8086 MICROPROCESSOR

Figure 15.9 shows the 8279 interfacing with the 8086 microprocessor. Lower order data bus $D_7 - D_0$ is directly connected with $D_7 - D_0$ of 8279 IC. The RESET signal of 8259 is connected to the RESET pin of the 8086 microprocessor. $\overline{RD}$ and $\overline{WR}$ signals of 8086 are connected with $\overline{RD}$ and $\overline{WR}$ signals of 8279. Address line A_1 of 8086 is directly interfaced with A_0 input of 8279. The address decoder generates the chip select signal $\overline{CS}$ from address lines A_0, A_2 to A_7 and $M/\overline{IO}$. The address of the data register of 8279 is 0080H and the control/status write/read address is 0082H. The generation of data port address and control/status write/read address is given in Table 15.1

Table 15.1 Address generation 0740H and 0742H for 8259A

Status of I/O Address lines																Address
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	In Hex
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0080H
0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0082H

15.10 8279 INTERFACING WITH 8051 MICROCONTROLLER

Figure 10.17 shows the 8279 interfacing with the 8051 microcontroller. Data bus $D_7 - D_0$ of 8279 IC is directly connected with $D_7 - D_0$ of 8051. The RESET signal of 8259 is connected to the RST pin of 8051 microcontroller. $\overline{RD}$ and $\overline{WR}$ signals of the 8259 are connected with $\overline{RD}$ and $\overline{WR}$ signals of the 8051. Address line A_0 of 8051 microcontroller is directly interfaced with A_0 input of 8279. The address decoder generates the chip select signal $\overline{CS}$ from address lines A_1 to A_7, and A_8 to A_{15}. The address of the data register and the control/status write/read address of 8279 depends on address lines, A_0, A_1 to A_7, and A_8 to A_{15}.

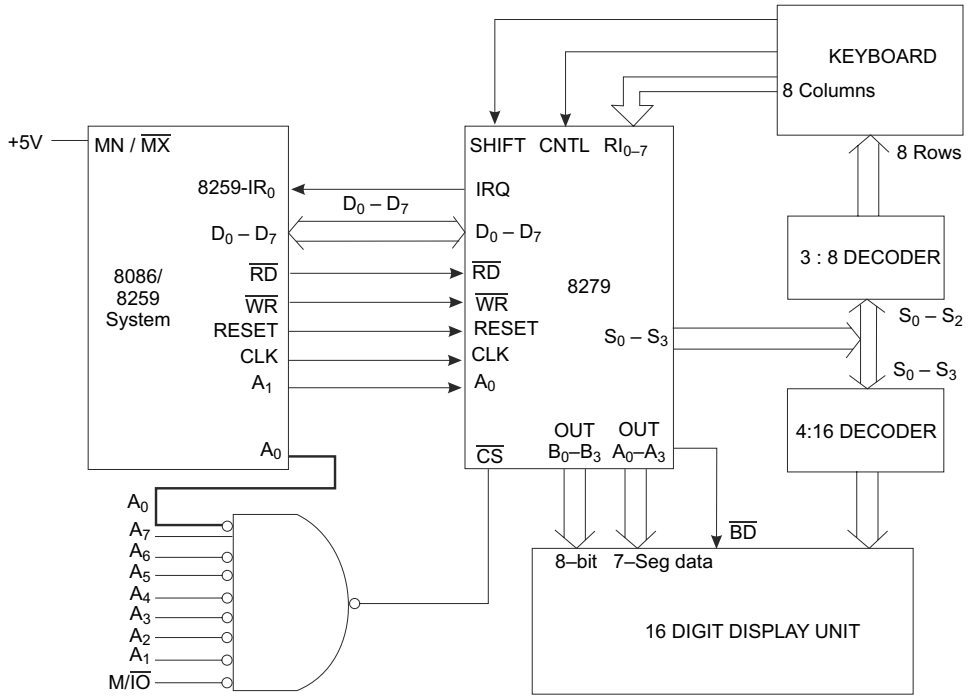


Fig. 15.9 8279 interfacing with 8086 microprocessor

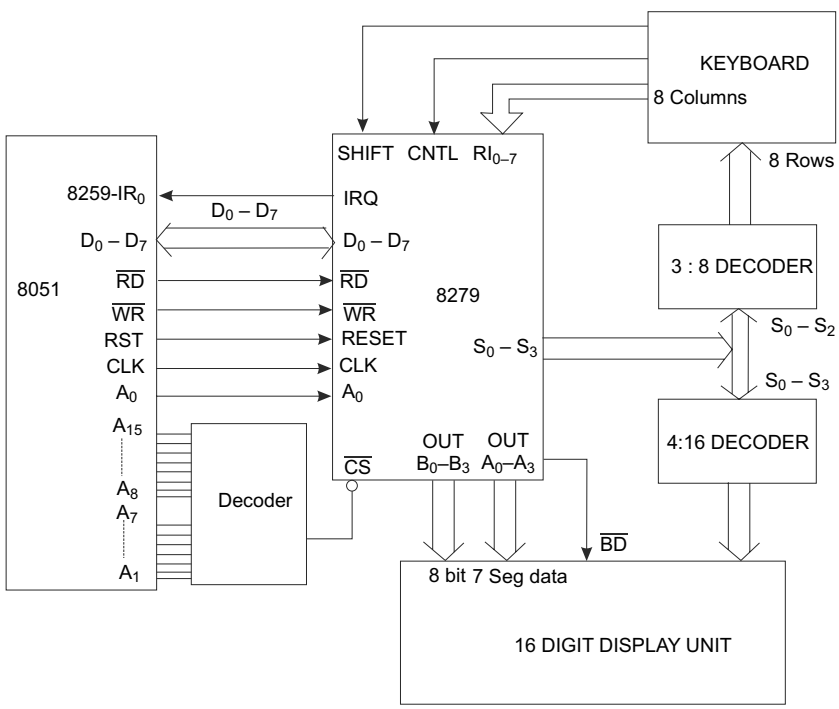


Fig. 15.10 8279 Interfacing with 8051 microcontroller

Review Questions

15.1. Draw the functional block diagram of 8279 and explain the operation of each block.

15.2. Write the different features of programmable keyboard and display controller.

15.3. Write the functions of the following signals

(i) SL_0–SL_3 (ii) RL_0–RL_3 (iii) CNTL/STB (iv) $OUTA_0$–$OUTA_3$
(v) $OUTA_0$–$OUTA_3$ (vi) IRQ (vii) SHIFT (viii) BD

15.4. What are the different modes of 8279 programmable keyboards and display controller? Explain each mode with an example.

15.5. What are the various input modes in which 8279 operates?

15.6. Discuss the left-entry mode of the display format.

15.7. Discuss the right-entry mode of the display format.

15.8. Explain the seven-segment display interfacing with 8279. How are sixteen-digit displays interfaced with 8279.

15.9. Draw a circuit diagram to interface 8279 with microprocessor and explain. Discuss the keyboard interface of 8279.

15.10. Write short notes on the following:

(i) Right entry (ii) Left entry (iii) *N*-key roller
(iv) Display RAM (v) 2-key roller (vi) FIFO
(vii) Command word format of 8279

15.11 List the major sections of the 8279 programmable keyboard/display interface.

15.12 In the context of 8279 programmable keyboard/display interface, explain the terms "2 key lock out" and "N key roll over"

15.13 What is 8279 better known as?

15.14 What is two key lockout and N-key rollover made of 8279? How an A/D converter can be interfaced with a 8085 microprocessor?

15.15 Write a short note on 8279 programmable keyboard/display interface

15.16 Draw the 8279 interfacing with 8085 microprocessor and explain the circuit operation briefly.

15.17 Design the 8279 interfacing with 8086 microprocessor and the generation of data port address and control/status write/read address.

15.18 Draw the 8279 interfacing with 8051 microcontroller and explain the circuit operation briefly.

Multiple-Choice Questions

15.1 8279 displays can operate in

(a) 8–8-bit character display-left entry only
(b) 16–16-bit character display-left entry only
(c) 8–8-bit character display-right entry only
(d) 8–8-bit character display-left and right entries and 16-16 bit character display-left and right entries

15.2 How many bytes of memory is available for the seven segment display?
(a) 16 × 8 display RAM (b) 16 × 4 display RAM
(c) 16 × 1 display RAM (d) None of these

15.3 How many bytes of memory is available for key pressed in 8279 ?
(a) 8 × 8 RAM (b) 8 × 4 RAM (c) 8 × 1 RAM (d) None of these

15.4 How many seven-segment displays can be connected with 8279?
(a) 16 (b) 12 (c) 10 (d) 8

15.5 How many character keyboards can be connected with 8279 in mode?
(a) 8 (b) 16 (c) 20 (d) 4

15.6 8279 is a
(a) DMA controller (b) programmable keyboard display interface
(c) counter (d) interrupt controller

15.7 In 8279 interfacing with 8085 microprocessor/8051 microcontroller, A_0 pin of 8279 is connected with ____ pin of 8085/8051.
(a) A_0 (b) A_1 (c) A_2 (d) A_3

15.8 During 8279 interfacing with 8086 microprocessor, A_0 pin of 8279 is connected with ____ pin of 8086.
(a) A_0 (b) A_1 (c) A_2 (d) A_3

Answers to Multiple-Choice Questions

15.1. (d)	15.2. (a)	15.3. (a)	15.4 (a)
15.5. (c)	15.6. (b)	15.7 (a)	15.8 (b)

CHAPTER

16

8251 Interfacing with 8085, 8086 and 8051 Microcontroller

16.1 INTRODUCTION

The serial data transfer is a method of data transfer in which one bit will be transferred at a time. This transmission is always used when the distance is greater than five metres. This method of transmission requires very few data lines compared to parallel transmission. Serial data transmission can be classified as simplex, half duplex and full duplex data transfer. In *simplex serial data transfer system* data is transferred only in one direction. In the *half duplex system*, data can be transferred in either direction but in one direction at a time only. In the *full duplex system* data can be transmitted in both direction simultaneously.

The serial data transfer systems can also be classified based on timing signals such as synchronous and asynchronous data transfer. The difference between synchronous and asynchronous data transfers is given in Table 16.1.

Table 16.1 Differences between synchronous and asynchronous data transfer

Asynchronous Data Transfer	Synchronous Data Transfer
In asynchronous data transfer, a word or character is preceded by a start bit and is followed by a stop bit. The start bit is a logical 0. The stop bit(s) is (are) a logical 1.	In synchronous data transfer, the transmission begins with a block header, which is a sequence of bits.
Data can be sent one character at a time.	This can be used for transferring large amounts of data without frequent starts or stops.
When no data is sent over the line, it is maintained at an idle value, logic '1'.	Since the data sent is synchronous, the end of data is indicated by the sync character(s). After that the line can be either low or high.
Parity bit can be included along with each word or character. Each character data can be 5, 6, 7 or 8 bits.	Parity bit can be included along with each word or character. Each character data can be 5, 6, 7 or 8 bits.
The start and stop bits are sent with each character. Generally, the stop bits may be either one or more bits. The stop bits must be sent at the end of the character. It is used to ensure that the start bit of	In synchronous data transfer, the transmitter sends synchronous characters, which is a pattern of bits to indicate end of transmission.

the next character will cause a start bit transition on the line.	
Asynchronous mode data transfer is used for low speed data transfer. Data can move in simplex, half duplex and full duplex methods.	Synchronous mode data transfer is used for high-speed data transfer. Data can move in simplex, half duplex and full duplex methods.
In this data transfer, the transmitter is not synchronised with the receiver by the same clock. The clock is an integral multiple of the baud rate (number of bits per second). Generally, this multiplication factor is 1, 16, or 64.	In synchronous mode data transfer, the receiver and transmitter is perfectly synchronised on the same clock pulse.
Synchronisation between the receiver and transmitter is required only for the duration of a single character at a time.	Synchronism between the transmitter and receiver is maintained over a block of characters.
Asynchronous data transfer can be implemented by hardware and software.	Synchronous data transfer can be implemented by hardware.

8251 is a powerful programmable communication interface ICs through which the serial data transfer can be effectively carried out. The Programmable Communication Interface 8251 is a programmable I/O device designed for serial communication. This IC can be used either in synchronous mode or asynchronous mode. Therefore, it is called Universal Synchronous Asynchronous Receiver and Transmitter (USART).

The IC chip is fabricated using *N*-channel silicon gate technology. The 8251 can be used to transmit and receive serial data. It accepts data in parallel format from the microprocessor and converts them into serial data for transmission. This IC also receives serial data and converts them into parallel and sends the data in parallel. It is available in a 28-pin dual in line package and has the following features:

- Synchronous and asynchronous operation
- Programmable data word length, parity and stop bits
- Parity, overrun and framing error checking instructions and counting loop interactions
- Programmed for three different baud rates
- Supports up to 1.750 Mbps transmission rates
- Divide-by 1, –16, –64 mode
- False start bit deletion
- The number of stop increase of asynchronous data transfer can be 1 bit, 1½ or 2 bits
- Full duplex double buffered transmitter and receiver
- Automatic break detection
- Internal and external synchronous character detection
- Peripheral modem control functions

This device is mainly used as the asynchronous serial interface between the processor and the external equipment. This IC can also be used to generate the baud rate clock using external clock and convert outgoing parallel data into serial data. This IC can also be used to convert incoming serial data into parallel data and it can control the modem.

16.2 FUNCTIONAL BLOCK DIAGRAM

The functional block diagram of 8251 IC is shown in Fig. 16.1. This IC consists of four major sections, namely, transmitter, receiver, modem control and microprocessor interface section. These four sections are communicated with each other on an internal data bus for serial data transfer.

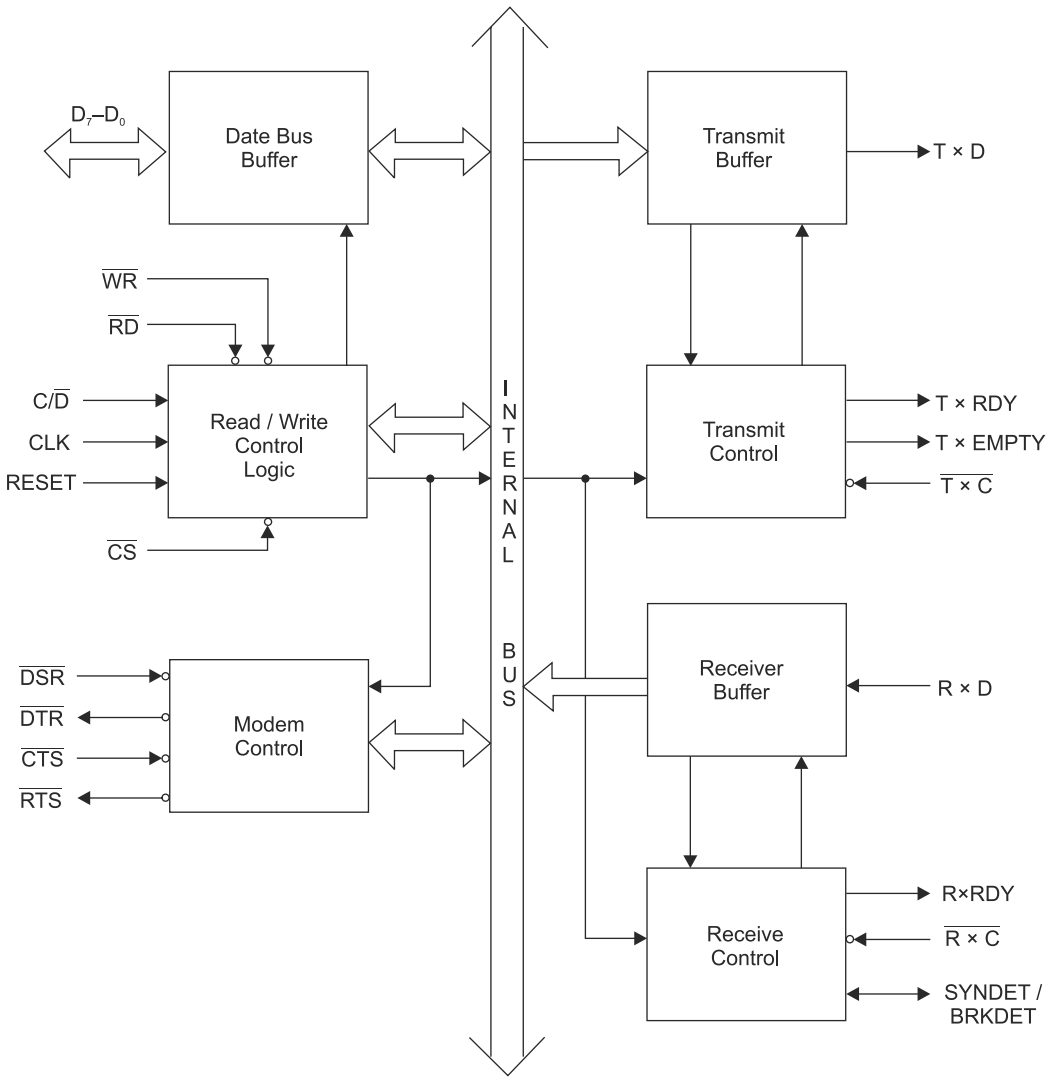


Fig. 16.1 Functional block diagram of 8251 IC

16.2.1 The Transmitter

Figure 16.2 shows the transmitter section. This section consists of transmitter buffer register, output register and transmits control logic. This section has one input and three outputs. The transmitter buffer register accepts parallel data from the data bus of microprocessor. Then data can be shifted out of the output register on the T × D pin after addition of framing bits. The serial data bits are preceded by START bit and succeeded by STOP bit, which are known as framing bits. For this operation the transmitter must be enabled and $\overline{CTS}$ signal must be active low. $\overline{T \times C}$ signal is the transmitter clock signal which controls the bit rate on the T × D line. The clock frequency may be 1, 16 or 64 times the baud.

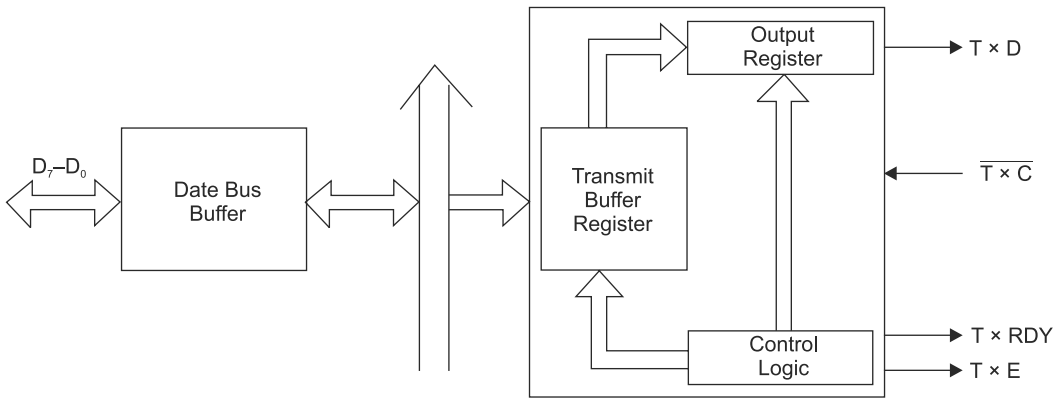


Fig. 16.2 Block diagram of transmitter

In the asynchronous mode, the transmitter adds a START bit, depending on how the unit is programmed; it also adds an optional even or odd parity bit, and either 1, 1½ or 2 STOP bits.

When the transmitter buffer register is empty, then it outputs a signal T × RDY. It indicates to the CPU that the 8251 is ready to accept a data character.

When there is no data in the transmitter output register, then it raises T × E signal to indicate that the transmission is stopped. It is reset when the data is transferred from the buffer register.

16.2.2 The Receiver

The block diagram of the receiver section is depicted in Fig. 16.3. This section consists of a receiver buffer register, receiver control logic and input register. The receiver section of 8251 USART accepts serial data on the R × D pin. Then converts it to parallel data according to the required format. When the 8251 is in the asynchronous mode and it is ready to accept a character, it looks for a low level on the R × D line.

When it gets a low level, it assumes that it is a START bit and enables an internal counter. At a count equivalent to one-half of a bit time, the R × D line is sampled again. If the line is still low, a valid START bit has probably been received and the 8251 proceeds to assemble the character. If the R × D line is high when

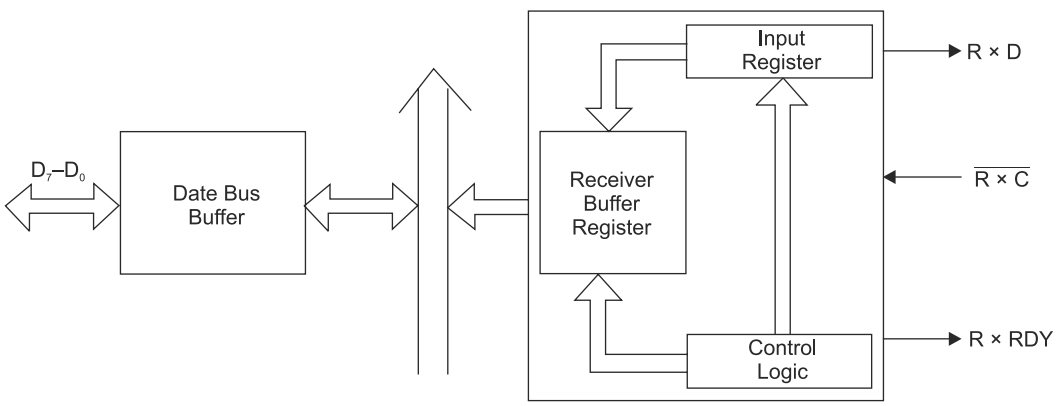


Fig. 16.3 Block diagram of receiver

it is sampled, then either a noise pulse has occurred or the receiver has become enabled in the middle of the transmission of a character. In either case, the receiver aborts its operation and prepares itself to accept a new character. After the successful reception of a START bit, the 8251 clocks in the data, parity and STOP bits in the input register, the data is separated and converted into parallel data and then it transfers the data to the Receiver buffer register. The R × RDY signal is asserted to indicate that a character is available. The R × C falling edge clock signal is used to disassemble the bits from the serial data.

When this register is full, the R × RDY line becomes high. This line is then used either to interrupt the microprocessor or to indicate its own status. The microprocessor then accepts the data from the register.

$\overline{R \times C}$ line stands for receiver clock. This signal controls the rate at which bits are received by the input register. The clock can be set to 1, 16, or 64 times the baud in the asynchronous mode.

16.2.3 Modem Control

The modem connection of the IC 8251 is shown in Fig. 16.4. The modem control section provides the generation of $\overline{RTS}$ (request to send) and the reception of $\overline{CTS}$ (clear to send). This section also provides a general-purpose output $\overline{DTR}$ (Data Terminal Ready) and a general-purpose input $\overline{DSR}$ (Data Set Ready). $\overline{DTR}$ is generally assigned to modem, indicating that the terminal is ready to communicate and $\overline{DSR}$ is a signal from the modem indicating that it is ready for communication.

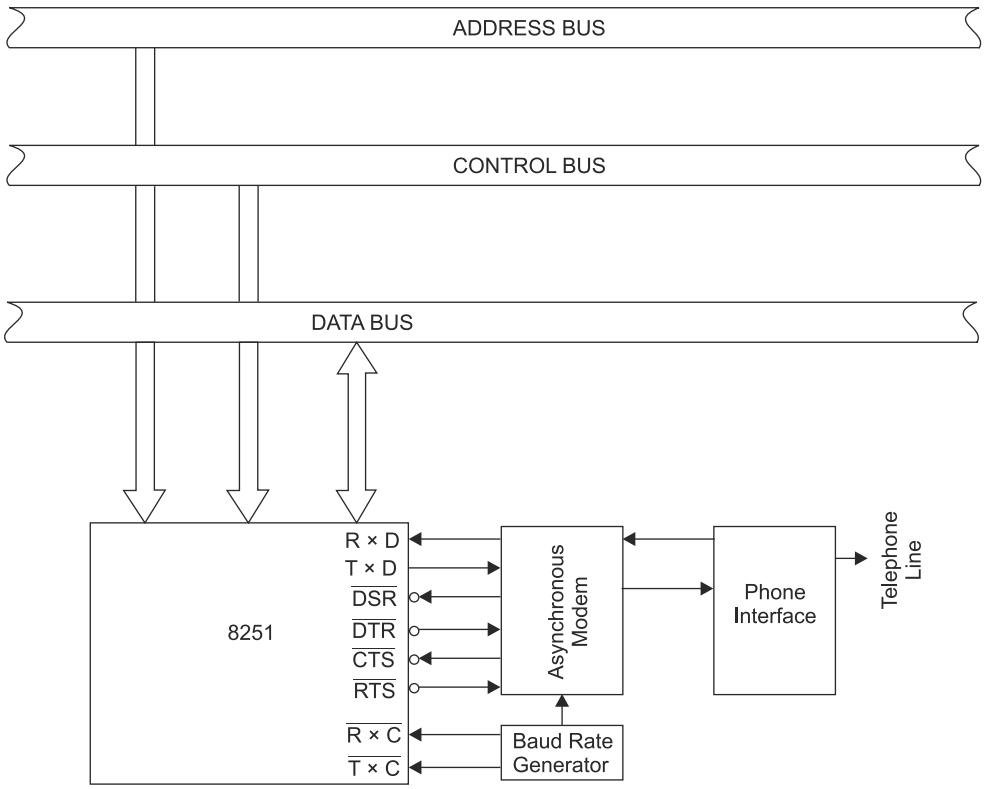


Fig. 16.4 Block diagram of modem control

16.3 PIN DIAGRAM OF 8251

Figure 16.5 shows a schematic diagram of Intel 8251 and the pin configuration of 8251A is depicted in Fig. 16.6. The description of pins is as follows:

D_0–D_7 (Data Bus) The 8-bit data bus is used to read or write status, command word or data from or to the 8251 A.

Read/Write Control Logic Signals The Read/Write Control Logic consists of three buffer registers such as data buffer register, control register and status register. It has six input signals $\overline{CS}$, $C/\overline{D}$, $\overline{RD}$, $\overline{WR}$, RESET and CLK.

$\overline{CS}$ (Chip select) An active-low on this input select inputs 8251 A for communication. When $\overline{CS}$ is high, no reading or writing operation can be performed. The data bus is tristated and $\overline{RD}$ and $\overline{WR}$ have no effect on the device.

$C/\overline{D}$ (Control Word/Data) This pin is used to inform the 8251A that the word on the data bus is either data or control word/status information. When $C/\overline{D}$ pin is high, either the control register or status register will be selected. If this pin is low, data bus buffer is selected. The $\overline{RD}$ and $\overline{WR}$ signals are used to distinguish the control register and the status register respectively.

Pin Name	Function
D_0–D_7	Data Bus
$\overline{CS}$	Chip select
$C/\overline{D}$	Control Word/Data
$\overline{RD}$	Read
$\overline{WR}$	Write
RESET	RESET
CLK	CLOCK
T × D	Transmit data
$\overline{T \times C}$	Transmitter clock
T × RDY	Transmitter ready
T × E	Transmitter empty
R × D	Receiving data
$\overline{R \times C}$	Receiver clock
R × RDY	Receiver ready
$\overline{DSR}$	Data set ready
$\overline{DTR}$	Data terminal ready
$\overline{CTS}$	Clear to send
$\overline{RTS}$	Request to send

$\overline{RD}$ (Read) An active-low on this input informs 8251A that the microprocessor is reading either data or status information from internal registers of 8251.

$\overline{WR}$ (Write) The active-low input on $\overline{WR}$ is used to inform it that the microprocessor is writing data or control word to 8251.

RESET A high on this input forces the 8251A into an 'idle' state. This device will be remaining idle until a

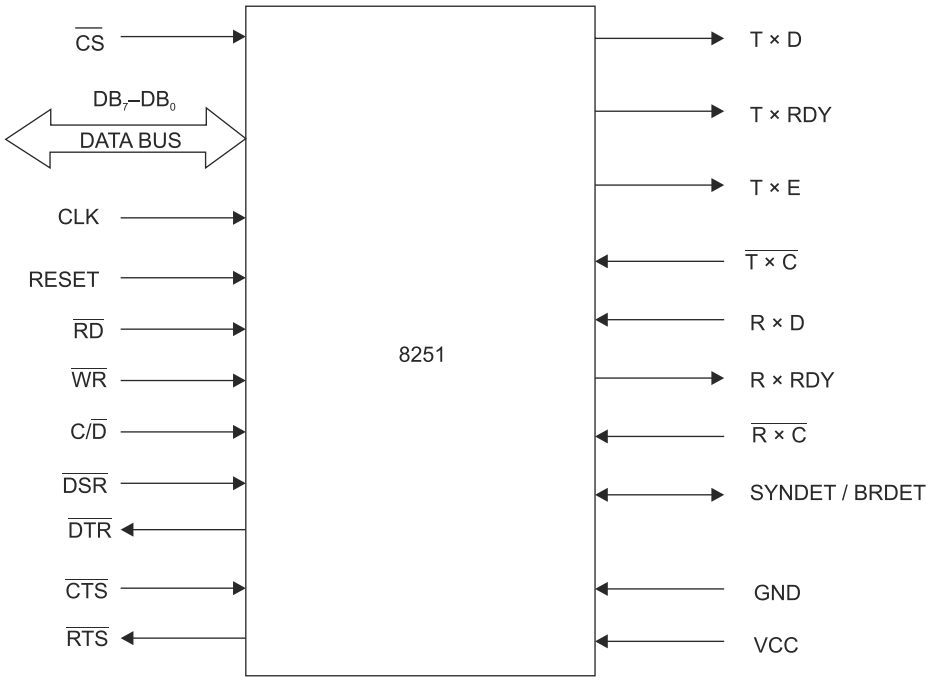


Fig. 16.5 Schematic diagram of Intel 8251

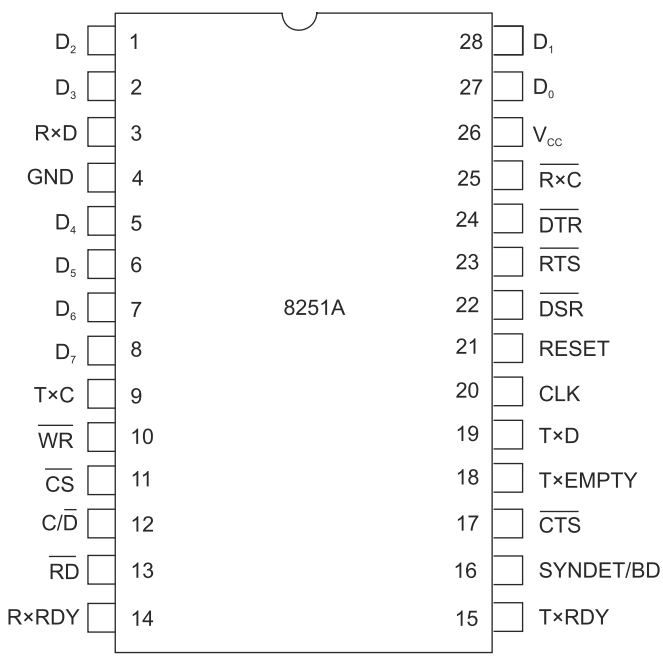


Fig. 16.6 Pin diagram of Intel 8251

new set of control words is written into it. The minimum required reset pulse width is 6 clock states for each reset operation.

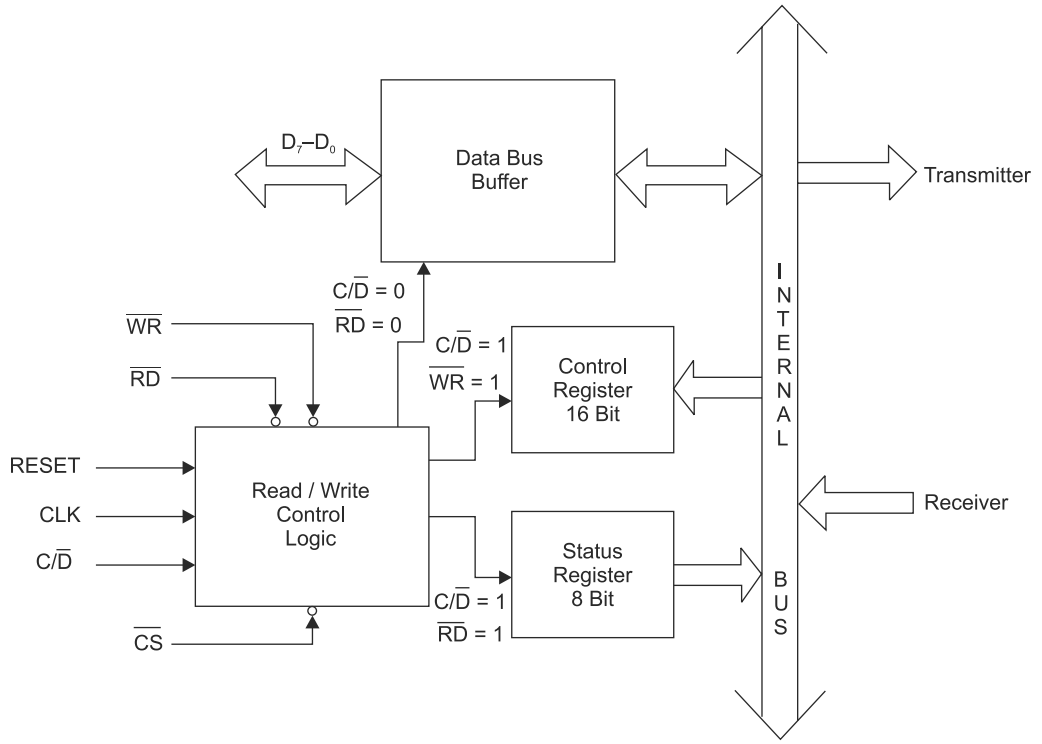


Fig. 16.7 Control logic and registers of IC

CLK (CLOCK) The CLK input is used to generate internal device timings and is normally connected to the output of the clock generator. The input frequency of CLK should be greater than 30 times the receiver or transmitter data bit transfer rate.

Table 16.2 shows the status of the control signals $\overline{CS}$, $C/\overline{D}$, $\overline{RD}$, $\overline{WR}$ and CLK for accessing the different registers.

Table 16.2 Status of some control signals

$\overline{CS}$	$C/\overline{D}$	$\overline{RD}$	$\overline{WR}$	State
0	1	1	0	Microprocessor writes instructions in the control register
0	1	0	1	Microprocessor reads status from the status register
0	0	1	0	Microprocessor outputs data to the data buffer
0	0	0	1	Microprocessor accepts data from data buffer
1	x	x	X	USART is not selected for communication

16.3.1 Transmitter

Transmitter control pins are T × D, $\overline{T \times C}$, T × RDY, and T×E which are explained below:

T × D (Transmit Data Output) The serial data output from the output register is transmitted on the T × D pin. The transmitted data bits consist of data along with other informations such as start bit, stop bits and parity bit.

$\overline{T \times C}$ (Transmitter Clock Input) The transmitter clock input $\overline{T \times C}$ controls the rate at which the data is to be transmitted. The baud rate is equal to the T × C frequency in synchronous transmission mode. In asynchronous transmission mode, the baud rate is 1, 1/16 or 1/64 times the T × C. The serial data is transmitted out by the successive negative edge of the T × C.

T × RDY (Transmitter Ready) This is the output signal, which indicates to the CPU that the transmitter buffer is empty. The internal circuit of the transmitter is ready to receive a byte of data from the CPU for transmission. The T × RDY signal is set only when $\overline{CTS}$ and T × E are active. The T × RDY is reset when the CPU writes a byte into the buffer register by the rising edge of the $\overline{WR}$ signal. The T × RDY status bit will indicate the empty or full status of the transmitter data input register.

T×E (Transmitter Empty) When the T × E output is high, the 8251 has no characters to transmit. This automatically goes low when a character is received from the CPU for further transmission. If this pin is high in synchronous mode, it indicates that a character has not been loaded and the SYNC character or characters are being transmitted or about to be transmitted. The T × E signal can be used to indicate the end of a transmission mode.

16.3.2 Receiver

Transmitter control pins are R × D, $\overline{R \times C}$, and R×RDY which are explained below:

R×D-Receive Data Input This input pin of 8251A receives serial data from the outside environment and delivers to the input register via R × D line which subsequently puts it into parallel form and places in the receiver buffer register.

$\overline{R \times C}$ (Receiver Clock Input) The $\overline{R \times C}$ receiver clock input pin controls the rate at which the bits are received by the input register. In the synchronous mode, the baud rate is equal to the R × C frequency. In the asynchronous mode, the baud rate can be set to either 1 or 1/16 or l/64th of the R × C frequency. The received data is read into the 8251 on rising edge of R×C.

R × RDY (Receiver Ready) This is a output pin which indicates that 8251A contains a character to be read by the CPU. This signal can be used to interrupt the CPU as well as polled by the CPU. In synchronous mode, the receiver must be enabled to set the R × RDY signal and a character must finish assembly and then be transferred to the data output register. When the data does not read properly from the receiver data output register before assembly of the next data byte, the overrun condition error flag is set and the previous byte is over written by the next byte of the incoming data and hence it is lost.

16.3.3 Modem Control Pins

8251 has four modem control pins $\overline{DSR}$, $\overline{DTR}$, $\overline{CTS}$ and $\overline{RTS}$. The $\overline{DSR}$ and $\overline{RTS}$ are inputs but $\overline{DTR}$ and $\overline{CTS}$ are output pins. All these pins are active low. The description of modem control pins are given below:

$\overline{DSR}$ (Data Set Ready) The $\overline{DSR}$ input can be used as a general-purpose one-bit inverting input port. The CPU using a status read operation can test its status. This input is normally used to check the modem condition such as data set is ready.

$\overline{DTR}$ (Data Terminal Ready) This is a general-purpose one-bit inverting output port. This can be used by 8251 to signal the modem about the information that the device is ready to accept data. This port can be programmed using the command word.

$\overline{CTS}$ (Clear to Send) This is a one-bit inverting input port. When $\overline{CTS}$ input line is low, the 8251A will be enabled to send the serial data, provided D_0, the enable bit in the command instruction word should be enabled if D_0 becomes low in the command instruction word while data transmission takes place. If $\overline{CTS}$ is switched off, the transmitter will compel sending the stored data.

$\overline{RTS}$ (Request to Send Data) This is a general-purpose one-bit inverting output port. This can be used by 8251 to indicate the modem that the receiver is ready to receive a data byte from the modem. Bit D_5 of command instruction format controls the status of the pin.

SYNDET/BD (Synchronous Detect/Break Detect) This pin is used for detection of synchronous characters in synchronous mode and break characters in asynchronous mode. This pin can be programmed using appropriate control word. In the input mode or the external synchronous detect mode, a rising edge on this pin will cause 8251 to start collecting data characters on the rising edge of the next $\overline{R \times C}$.

This pin can be used as a break detect in the asynchronous mode. When R × D pin remains low through two consecutive stop bit sequences, the stop bit sequence contains a stop bit, a start bit, data bits and parity bits. This is reset when master chip reset or the R × D becomes high.

16.4 8251 INTERFACE WITH 8085 MICROPROCESSOR

The interfacing connection of 8251 with the microprocessor is shown in Fig. 16.8. This circuit consists of eight data lines, which are connected to the data bus of the CPU. The 8251 IC can be used either in

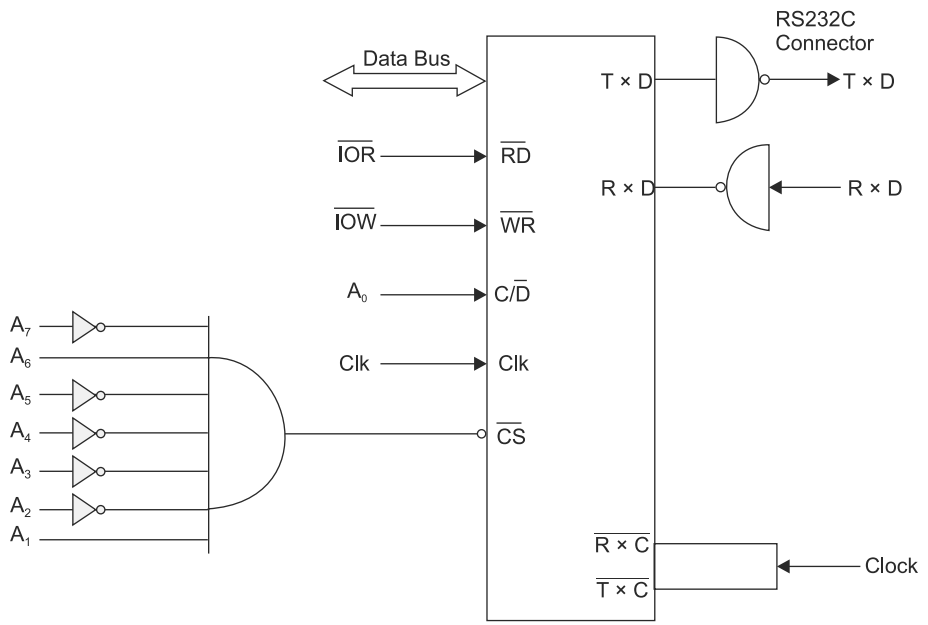


Fig. 16.8 Interfacing of 8251 with 8085 microprocessor

I/O mapped I/O or memory mapped I/O mode. In I/O mapped I/O interface mode, the $\overline{RD}$ and $\overline{WR}$ lines are connected to $\overline{IOR}$ and $\overline{IOW}$ control lines. The CLK pin is connected with to the CLK OUT of the microprocessor to provide synchronization between the microprocessor and 8251. The RESET terminal is connected to the RESET OUT of the microprocessor. When RESET pin is in high level, the IC 8251 is forced into the idle mode. The address decoder output is connected to the $\overline{CS}$ terminal of 8251. The $C/\overline{D}$ terminal is used to select internal registers such as control register and data register. Generally, this is connected to the A_0 address bus.

16.5 PROGRAMMING AND OPERATING MODES OF 8251

The 8251 can be operated in different modes based on mode control words. A set of control words can be written into the internal registers of 8251A to make it operate in the desired mode. The control words of 8251A are two functional types, namely,

1. Mode Instruction Control Word
2. Command Instruction Control Word

There are two 8-bit control registers in 8251 to load mode word and to load command word. The control logic and registers of the 8251 IC is depicted in Fig. 16.7. The mode instruction word informs about the initial parameters such as mode, baud rate, stop bits and parity bit. The command instruction word explains about enabling the transmitter and receiver section. The mode instruction word and command instruction control word are explained below.

16.5.1 Mode Instruction Control Word

Mode instruction control word defines the general operational characteristics of 8251A. After reset by using internal reset command or external reset command, the mode instruction control word must be loaded into 8251 to configure the device as per requirements. These control words are different for synchronous and asynchronous mode operation. Once mode instruction control word has been written into 8251, SYNC characters (synchronous mode only) or command instructions (synchronous or asynchronous mode) may be programmed. The mode of operation from synchronous to asynchronous or from asynchronous to synchronous can be changed by reset the 8251. The typical data is given in Fig. 16.9. The mode instruction format for asynchronous mode is shown in Fig. 16.10.

$C/\overline{D}$ = 1	Mode instruction	
$C/\overline{D}$ = 1	Sync character 1	Sync mode only
$C/\overline{D}$ = 1	Sync character 2	
$C/\overline{D}$ = 1	Command instruction	
$C/\overline{D}$ = 0	Data	
$C/\overline{D}$ = 1	Command instruction	
$C/\overline{D}$ = 0	Data	
$C/\overline{D}$ = 1	Command instruction	

Fig. 16.9 Typical data block

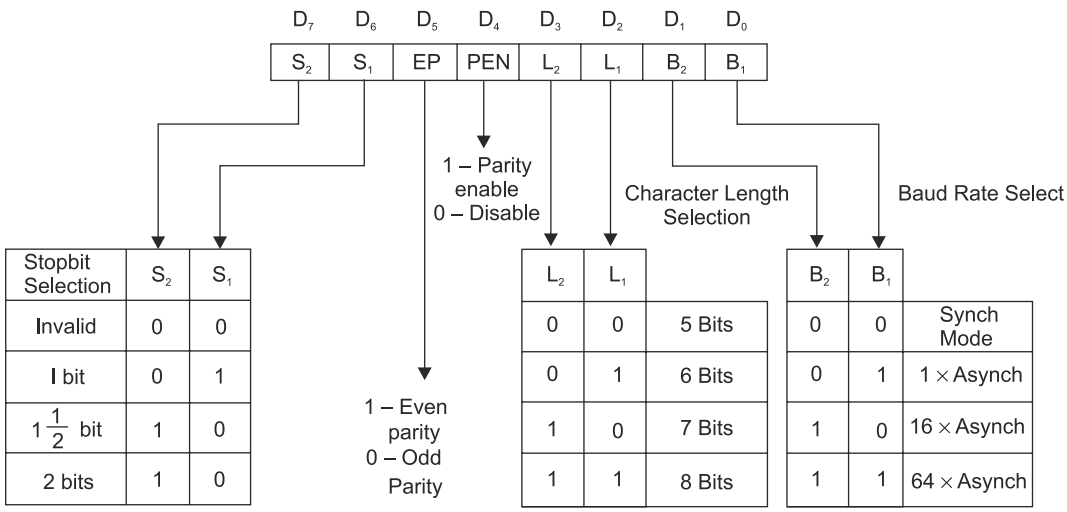


Fig. 16.10 Instruction format of asynchronous mode

Example 16.1 Find the mode instruction for the following operations:

- 8251 can be operated in asynchronous mode for data transmit.
- The baud rate is 16 × Asynch.
- The length of character is 8 bits and number of stop bits is 2.

Assume Odd parity and the address of the control register is 41H and the address of data register is 40H. The mode instruction word for the above operations is DEH as shown in Fig. 16.11.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	0	1	1	1	1	0

Fig. 16.11

To load the instruction word into the control word register, the following statements will be written

```
MVI A, DEH
OUT 41H
```

Asynchronous Mode (Transmission) The general transmission format for asynchronous communication is shown in Fig. 16.12. The transmission format consists of start bit, data character, parity bit and stop bits. 8251 starts to send data on the T × D pin after adding a start bit which is a 1 to 0 transmission. Then data bits are transmitted using the T × D pin on the falling edge of transmitter clock ($\overline{T \times C}$) followed by stop bits. When no data is transmitted by the CPU to 8251, the T × D output pin remains 'high. If a 'break' has been detected, T × D line will go low.

Asynchronous Mode (Receive) The general receive format for asynchronous communication is shown in Fig. 16.12. Data reception starts with a falling edge of R × D input which indicates the arrival of start bit. The high to low transition on the R × D line triggers the 'False start Bit Detection Circuit' and the output of this circuit samples the R × D line half-a-bit time later to confirm about the start bit. If R × D is low, it indicates a valid start bit which starts counting. Then the bit counter locates data bits, parity bits and stop bits. If any error is occurred during the receiving of data with regard to parity, framing or overrun, the

corresponding flags in the status word will be set. The receiver requires only a one-stop bit to mark end of the data bit string though the number of stop bits affects the transmitter.

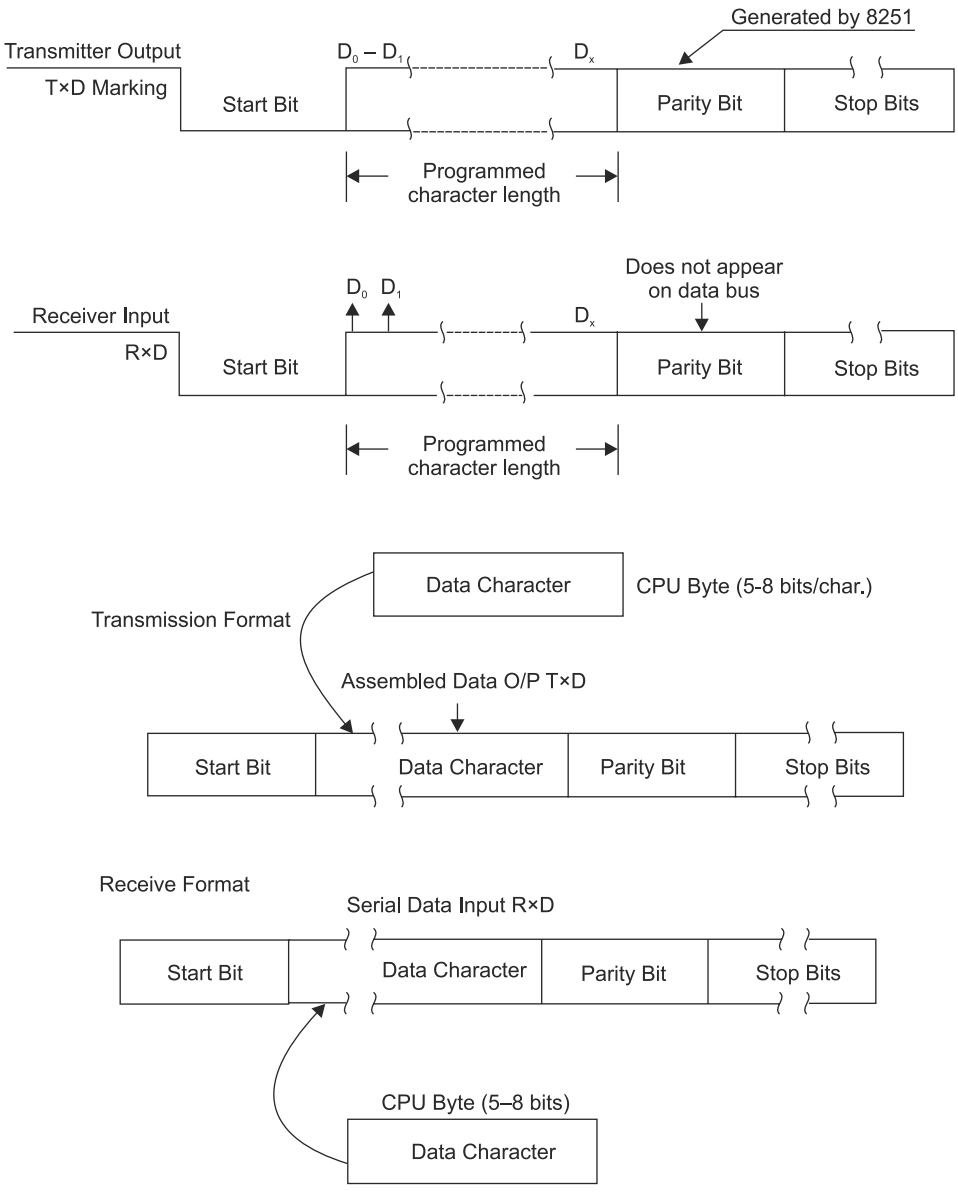


Fig. 16.12 Asynchronous mode transmission and receive formats

Synchronous Mode Instruction Format The synchronous mode instruction format is shown in Fig. 16.13. For synchronous transition/receiving of data, both D_0, D_1 will be low. Bit D_2 and D_3 indicate the character length. Bit D_4 stands for parity enable (PEN) and D_5 also stands for even parity (EP). Bit D_6 stands for external synchronous detect (ESD). $D_6 = 1$ for input and $D_6 = 0$ for output. Bit $D_7 = 1$ indicates a single synchronous character (SCS). But $D_7 = 0$ represents double synchronous characters.

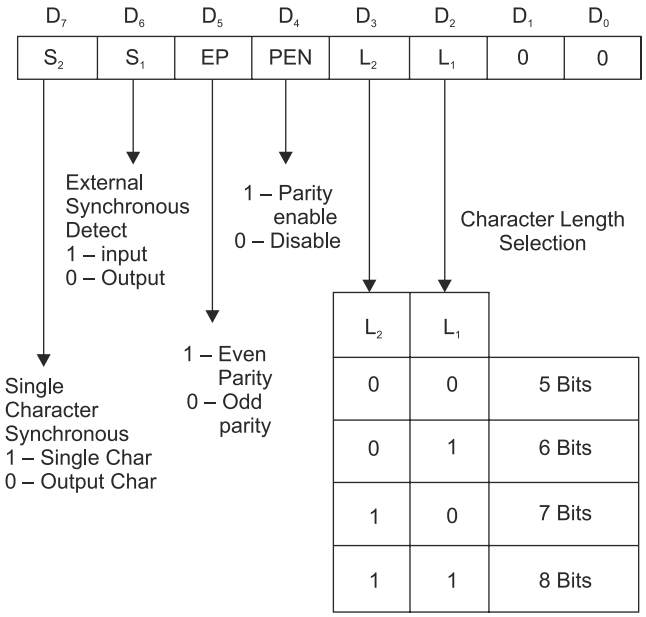


Fig. 16.13 Synchronous mode instruction format

Synchronous Mode (Transmission) The general transmission/receive format for synchronous communication is shown in Fig. 16.14. In transmission format one or two synchronous characters are sent followed by data characters. When $\overline{CTS}$ line becomes low, the first character is serially transmitted out. All the characters are shifted out of the serial output register on the falling edge of the transmitter clock ($\overline{T \times C}$) at the same rate as $\overline{T \times C}$. After completion of transmission, the CPU replenishes the transmitter buffer. When the CPU fails to provide a character before the transmitter buffer becomes empty, 8251 should send SYNC characters. In that case, the T × E pin becomes high to indicate that the transmitter buffer is empty.

Synchronous Mode (Receiver) In the synchronous receive mode, the character synchronisation can be achieved internally or externally. In the internal SYNC mode, the receiver samples the data available as the R × D pin on the rising edge of $\overline{R \times C}$. When 8251 is programmed in this mode, 'ENTER HUNT' command should be included in the first command instruction word. The data on R × D pin is sampled on rising edge of the R × C. The receiver buffer content is compared with the first SYNC character at every edge till a match occurs. When 8251A is initially programmed for two sync characters, the process can be extended to two SYNC characters. When both the characters match, the hunting stops. After HUNTING is over, the system goes for character boundary synchronisation. The SYNMDET pin is set and is rest automatically by a status read operation. The SYNDET pin gets set in the middle of the parity bit, if the parity is enabled; otherwise in the middle of the last data bit. In the external SYNC mode, synchronisation can be achieved by applying a high level on the SYNDET input pin, which forces 8251A out of HUNT mode. The parity and overrun error can be checked in the same way as in asynchronous mode.

16.5.2 Command Instruction Word

The command instruction controls the actual operations of the selected format like enable transmit/receive, error reset and modem controls. Once the mode instruction has been written into 8251A and the SYNC characters are loaded (only in synchronous mode), the device is ready for data communication. The command

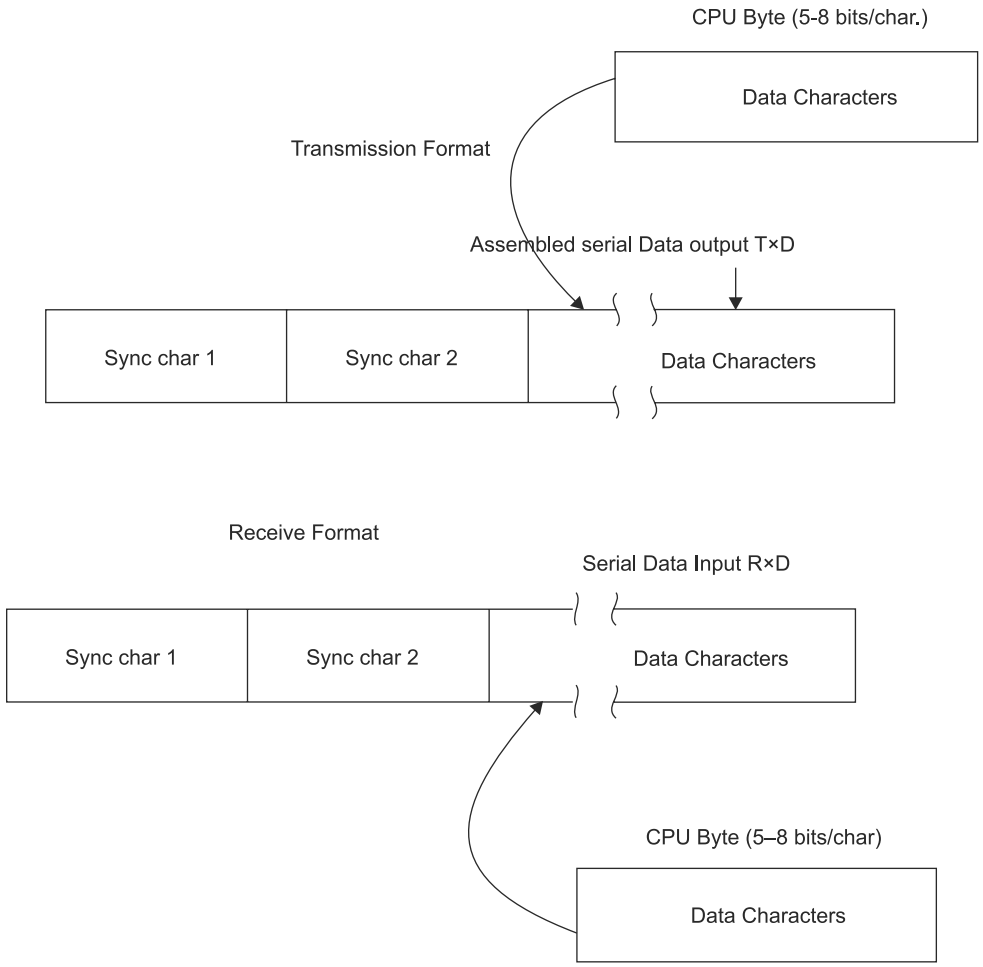


Fig. 16.14 Synchronous mode instruction format

instructions can be accepted only after mode instruction in case of asynchronous mode. All further control words written with $C/\overline{D}$ will load a command instruction. A reset operation returns the 8251 back to mode instruction format from the command instruction format. The format of command instruction is depicted in Fig. 16.15.

When the 8251 has been programmed by the mode instruction word, the device is ready for data communication. The command instruction controls the actual operation of the selected format. The command instruction format is explained discussed below:

The D_6 bit is used as the internal reset. The command word with $D_6 = 1$ returns 8251 in the mode instruction format. When D_0 (T × EN) is high, the transmitter becomes enable and data transmission is possible. If D_2 (R × EN) is high, it enables the receiver for reception. The D_1 bit controls the data terminal ready. The D_3 bit forces the transmitter to send continuous break characters. A high on D_4 resets the error flags—PE, OE and FE (parity, overrun and framing errors respectively). The D_5 bit controls the request to send pin of the device. The D_7 bit is used in synchronous mode. This pin enables the receiver to look for the synchronising data.

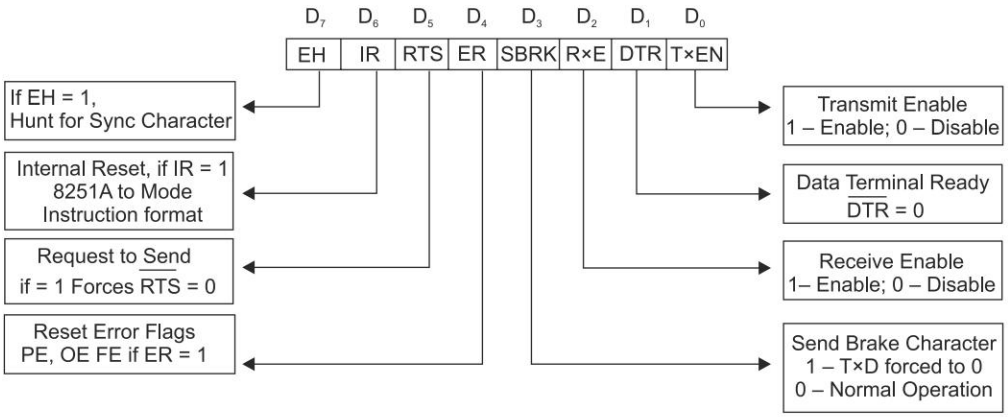


Fig. 16.15 Command instruction word

Status Word Register Format The status word can be read with $C/\overline{D} = 1$. The CPU requires various information to operate properly. All required information are provided by the status word. The status word is continuously updated by 8251, except during the CPU reads the status word. The status word format is shown in Fig. 16.16.

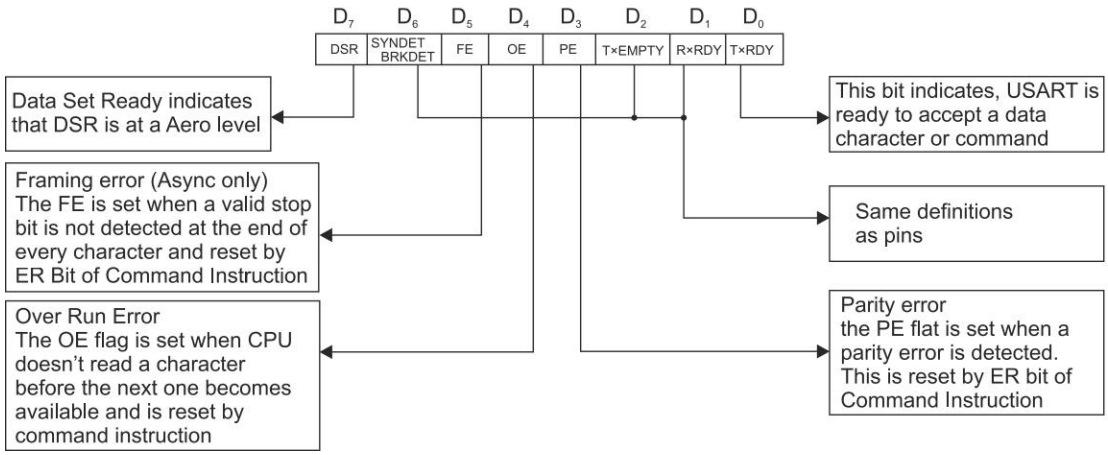


Fig. 16.16 Status word format

D_0 stands for the status of the pin T × RDY.

D_1 represents the status of the pin R × RDY.

D_2 correspond to the status of the pin T × E.

D_3 represents *parity error*. It is set when there is a parity error.

D_4 stands for *overrun error*. It is set when the CPU does not read a character before the next one becomes available.

D_5 represents *framing error*. It is set when a valid stop bit is not detected.

D_6 is used for synchronous mode (SYNDET)

D_7 reflects the logic level of the DSR (modem control) pin.

16.6 8251 INTERFACING WITH 8086 MICROPROCESSOR

Figure 16.17 shows the interfacing of 8251 with the 8086 microprocessor. Lower order data bus $D_7 - D_0$ of 8086 is directly connected with $D_7 - D_0$ of 8251 IC. $\overline{RD}$, $\overline{WR}$, RESET and clock (CLK) signals of 8251 are connected with $\overline{IOR}$, $\overline{IOW}$, RESET and clock (CLK) signals of 8086 respectively. The $C/\overline{D}$ terminal of 8251 is used to select internal registers such as data register and control register. Usually, $C/\overline{D}$ is connected to the address line A_1 of 8086. The chip select $\overline{CS}$ signal is generated from address lines A_0 and A_2 to A_7. Actually, the address decoder output is connected to the $\overline{CS}$ terminal of 8251. According to Fig. 16.17, the address of data register of 8251 is FEH and the control resister is FFH. The generation of data port address and control register address is given in Table 16.3.

Table 16.3 Address generation FEH and FFH for 8251A

Status of I/O Address lines								Address
A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	in Hex
1	1	1	1	1	1	1	0	FEH
1	1	1	1	1	1	1	1	FFH

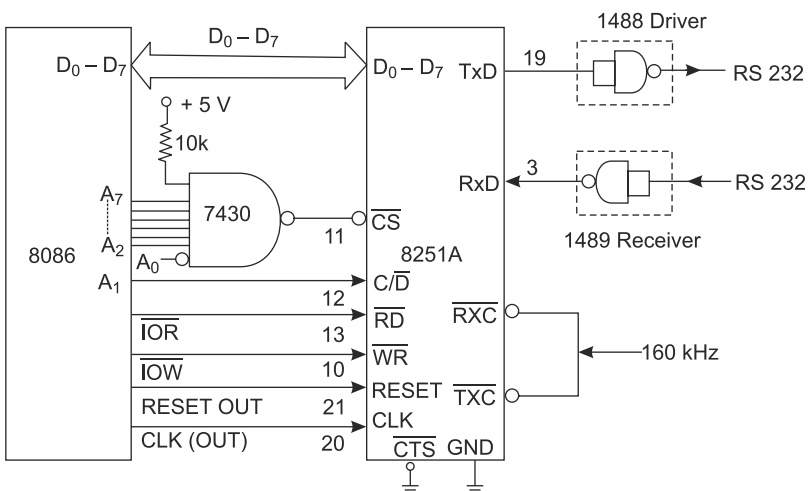


Fig. 16.17 8251 Interfacing with 8086 microprocessor

16.7 8251 INTERFACING WITH 8051 MICROCONTROLLER

Figure 16.18 shows the interfacing of 8251 with the 8051 microcontroller. Data bus $D_7 - D_0$ of 8251 IC is directly connected with $D_7 - D_0$ of 8051. $\overline{RD}$, $\overline{WR}$, and RESET signals of 8251 are connected with $\overline{RD}$, $\overline{WR}$ and RST signals of 8051 respectively. The $C / \overline{D}$ terminal of 8251 is used to select internal registers such as data register and control register. Usually, $C / \overline{D}$ is connected to the address line A_0 of 8051. The chip select $\overline{CS}$ signal is generated from address lines A_0, A_2 to A_7 and A_8 to A_{15}. The address decoder output is connected to the $\overline{CS}$ terminal of 8251.

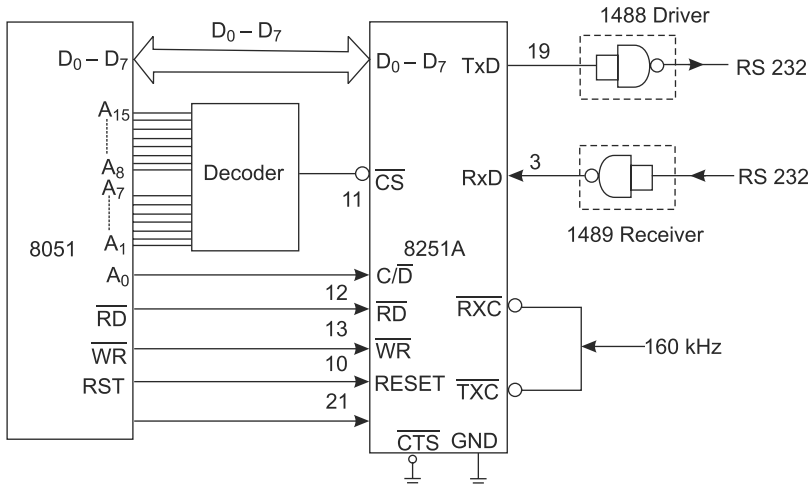


Fig. 16.18 8251 Interfacing with 8086 microprocessor

Review Questions

16.1. What is serial data transfer? Write the difference between synchronous and asynchronous data transfer.

16.2. Define simplex, half duplex and full duplex data transfer.

16.3. Draw the block diagram of 8251 chip and explain its working principles.

16.4. Write the functions of the following pins of 8259A:

(i) T × D (ii) T × E (iii) R × D (iv) T × RDY

(v) $\overline{DSR}$ (vi) $\overline{DTR}$ (vii) $C/\overline{D}$ (viii) $\overline{RTS}$

16.5. Draw the functional block diagram of 8251 and explain the operation of each block.

16.6. Describe the Read/Write control logic and registers of 8251.

16.7. Explain the operation of the transmitter section of 8251. How does the CPU know the transmitter buffer is empty?

16.8. Explain the operation of the receiving section of 8251. Why modems used are in case of digital transmission of data?

16.9. Explain the function of SYNDET/BD pin of 8251.

16.10. What are the modem control pins associated with 8251? Describe the functioning of these pins.

16.11. Discuss the mode instruction format for asynchronous transmission/reception.

16.12. Explain synchronous mode instruction format and command instruction format.

16.13. Draw the general transmission/receive format for synchronous communication.

16.14. Draw the status word format and explain the same.

16.15. Discuss the mode instruction format for synchronous transmission/reception case.

16.16. Show the command instruction format and explain briefly.

16.17. Explain how data can be transferred using 8251 USART at different baud rates. Write the features of 8251.

16.18 Draw the 8251 interfacing with 8085 microprocessor and explain the circuit operation briefly.

16.19 Design the 8251 interfacing with 8086 microprocessor and the generation of data port address and control/status write/read address.

16.20 Draw the 8251 interfacing with 8051 microcontroller and explain the circuit operation briefly.

Multiple-Choice Questions

16.1. At what speed (baud rate) is data transferred in 8251?
(a) 300 (b) 256 (c) 150 (d) 9600

16.2. 8251 is a
(a) USART IC (b) counter
(c) interrupt controller (d) programmable peripheral interface

16.3. The number of stop bits in case of asynchronous data transfer is
(a) 2 bits (b) 3 bits (c) 4 bits (d) 5 bits

16.4. 8251 operates in _____ mode.
(a) synchronous (b) asynchronous
(c) synchronous and asynchronous (d) none of these

16.5. When serial data can be transferred in either direction but in one direction at a time, the data transfer is known as
(a) simplex (b) half duplex (c) full duplex (d) none of these

16.6. The UART performs
(a) a serial to parallel conversion (b) A parallel-to serial conversion
(c) control and monitoring functions (d) All

16.7. The USART consists of
(a) data bus buffer (b) control and logic function
(b) transmit and receive buffer (c) all of these

16.8. The modems are used in serial data communication
(a) as telephone circuits may be connected in some places of the circuit
(b) as the switched telephone network may be connected in the circuit to reach anyone in the system
(c) as modulator-demodulator to translate digital into audio frequency signals for transmission through telephone lines
(d) all of the above

16.9 In 8251 interfacing with 8085 microprocessor/8051 microcontroller, $C/\overline{D}$ pin of 8251 is connected with ____ pin of 8085/8051.
(a) A_0 (b) A_1 (c) A_2 (d) A_3

16.10 During 8251 interfacing with 8086 microprocessor, $C/\overline{D}$ pin of 8251 is connected with ____ pin of 8086.
(a) A_0 (b) A_1 (c) A_2 (d) A_3

Answers to Multiple-Choice Questions

16.1 (d)	16.2 (a)	16.3 (a)	16.4 (c)
16.5 (b)	16.6 (b)	16.7 (c)	16.8 (d)
16.9 (a)	16.10 (b)		

CHAPTER

17

Direct Memory Access (DMA) Controller 8257

17.1 INTRODUCTION

During a bulk of data transfer between memory and any peripheral devices, if I/O data transfer technique is used, this process takes more time as each byte of the data is transferred through the CPU. If we want to transfer data at a faster rate, the CPU must be isolated and data can be transferred between memory and peripheral devices directly. This I/O technique is known as Direct Memory Access (DMA) operation.

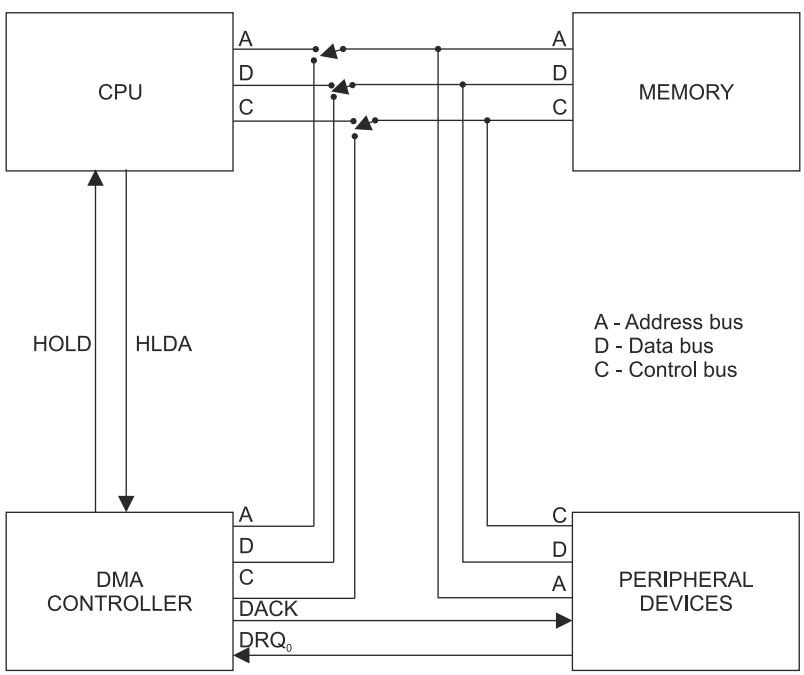


Fig. 17.1 DMA operation

Figure 17.1 shows the DMA operation, which consists of CPU, memory devices, I/O peripheral devices, DMA controller and switches. Usually, the switch positions will be such that the memory and peripheral devices are connected to the CPU. Therefore, the data bus, address bus and the control bus of the memory and I/O peripheral devices are connected to the CPU. While the DMA operation is to be performed, the CPU is completely isolated and the address bus and control bus are taken over by the DMA controller circuitry. The DMA operation can be carried out by the following sequence as given below:

- Initially the device, which requires data transfer between the device and the memory, should send the DMA request (DRQ) to the DMA controller.
- The DMA controller sends a Hold Request (HRQ) line to the CPU and waits for the CPU to assert the HLDA.
- Then microprocessor tri-states all the address bus, data bus and control bus. The CPU relinquishes the control of the bus and acknowledges the Hold input signal through Hold Acknowledge (HLDA) output signal. The CPU remains in the HOLD state; the DMA controller becomes the master of bus. Actually, DMA controller circuit manage the switching of address, data and control buses between CPU, memory, and I/O devices.
- The HLDA signal is fed to the DMA controller. When the DMA controller receive the HLDA signal, the DMA controller takes care of direct data transfer operation between memory and I/O devices. The DMA controller sends DACK signal to the peripheral device, which requested for DMA operation.
- Then DMA operation can be performed by sending a proper address to the memory and required control signals to transfer a bank of data.

At the starting of the DMA operation, the DMA controller should know the starting address of the memory location, number of bytes to be transferred and type of data transfer from memory to I/O or from I/O to memory.

17.2 PIN DIAGRAM

The 8257 IC is a programmable DMA controller. This is available in 40-pin dual in line package. The schematic diagram of 8257 is shown in Fig. 17.2. The pin diagram of 8257 is depicted in Fig. 17.3 and the pin functions are described below.

DRQ_0–DRQ_3 These are the four separate DMA request lines. Any I/O device sends a DMA request signal on one of the DRQ_0–DRQ_3 lines. When DRQ is high, a DMA request signal is received by the DMA controller. Among four DMA request lines—DRQ_0 has the highest priority and DRQ_3 has the lowest priority in the fixed priority mode.

$DACK_0$–$DACK_3$ These are the DMA acknowledge output lines which sends an acknowledged signal through any one of these lines to the I/O peripheral devices when this signal is active-low, the line acknowledges the I/O devices.

A_0–A_3 These are the least significant address lines. A_0–A_3 are bi-directional lines. In master mode, these four least significant memory address lines generated by 8257.

A_4–A_7 This is the four most significant address lines of lower byte address generated by 8257 in the master mode DMA operation.

D_0–D_7 These are bi-directional data lines. These lines are used to interface CPU with internal data bus of 8257 DMA. During programming of the DMA controller, the CPU sends data through data lines for DMA

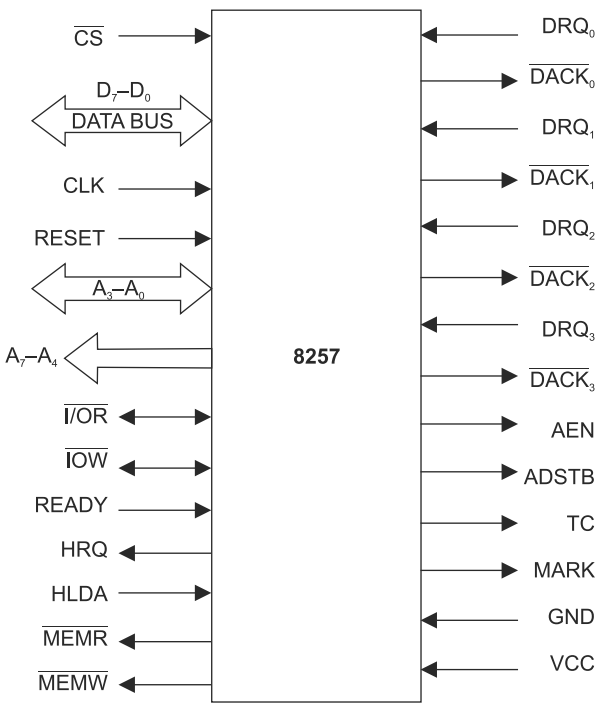
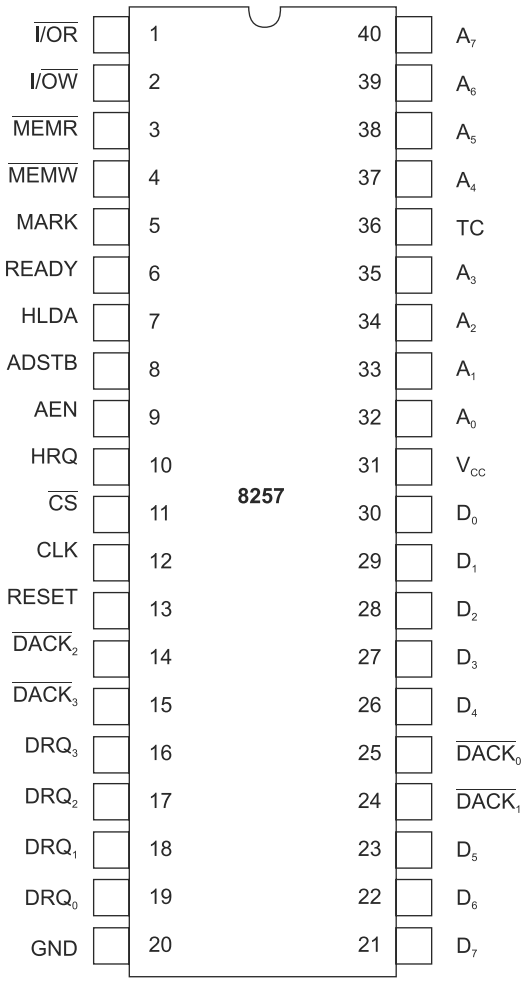


Fig. 17.2 Schematic pin diagram of 8257



17.3 Pin diagram of 8257

address register; byte count register and mode set register. When the 8257 operate in master mode, these lines used to send higher byte of the memory address. Then these 8 MSBs of address are latched using ADSTB signal. During the first clock cycle of DMA operation, the address is transferred over D_0–D_7. After that data bus is available for data transfer during the rest DMA cycle.

$\overline{IOR}$ This is a bi-directional tristate input line. In the slave mode, the IOR signal is used to read the internal registers of 8257 by the CPU. This line operates as output in the master mode. In master mode, IOR is used to read data from I/O peripheral device during DMA write cycle.

$\overline{IOW}$ This is a bi-directional tristate input line. In the slave mode, IOW signal is used to load the content of data lines to the upper or lower byte of a 16-bit DMA address register or terminal count register. In the master mode, data is transferred from memory to I/O devices during DMA memory read cycle.

$\overline{MEMR}$ This is a memory read output signal. When this is active low, data will be read from memory during the DMA read cycle.

$\overline{MEMW}$ This is a memory write output signal. When this is active low, data will be written to the memory during a DMA write cycle.

CLK A clock input is applied to 8257 for internal operation of 8257, which will be synchronized with the clock.

RESET This is an asynchronous input signal. When this is active high, all DMA channels must be disabled, all mode registers will be cleared and all the control lines will be in tristates.

$\overline{CS}$ This is an active-low chip select signal, which enables the 8257 for read and write operations in slave mode. In the master mode, this is disabled to prevent the chip from getting selected (by CPU) during the DMA operation.

AEN (Address Enable) This is the address enable signal. When it is high, it indicates that the DMA operation will be performed. This AEN output can be used to disable the data bus and the control bus driven by the processor. This output can be used to disable the selection of an I/O device in the system.

ADSTB (Address Strobe) This output from 8257 strobes the most significant byte of the memory address generated by the DMA controller into the latches.

TC (Terminal Count) This output indicates that the terminal count register content is zero. If the TC STOP bit in the mode set register is set, the selected channel will be automatically disabled at the end of the DMA cycle. This pin will be activated when the 14-bit content of the terminal count register of the selected channel is equal to zero. The lower order 14 bits of the terminal count register are to be programmed for the desired number of DMA cycles.

MARK When the MARK output is one level, it indicates that the current DMA cycle is the 128th cycle since the previous MARK output. The mark may be activated after each 128 cycles for the particular peripheral devices.

READY This is an asynchronous input signal. This is used to extend memory read and write cycles of 8257 by inserting wait states. This is suitable for interfacing slower I/O peripheral devices.

HRQ (Hold Request) The DMA controller sends the hold request as it is connected to the hold signal input of the microprocessor. This output requests the microprocessor to access the system bus. In the master mode, this is connected with the HOLD pin of the CPU. In the slave mode, this pin of a slave is connected with a DRQ input line of the master 8257 and the master is connected with HOLD input of the CPU.

HLDA (Hold Acknowledge) This pin is connected to the HLDA output of the CPU. This input is high indicates that the microprocessor tristates all the address bus, data bus and control bus.

V + 5 V supply

GND Ground.

17.3 ARCHITECTURE OF 8257

8257 is a programmable four independent channel DMA controller. Therefore, four peripherals can send request data transfer simultaneously. The block diagram of internal architecture of 8257 is depicted in Fig. 17.4. This consists of four DMA channels, control logic for data transfer, read/write logic and data bus buffer.

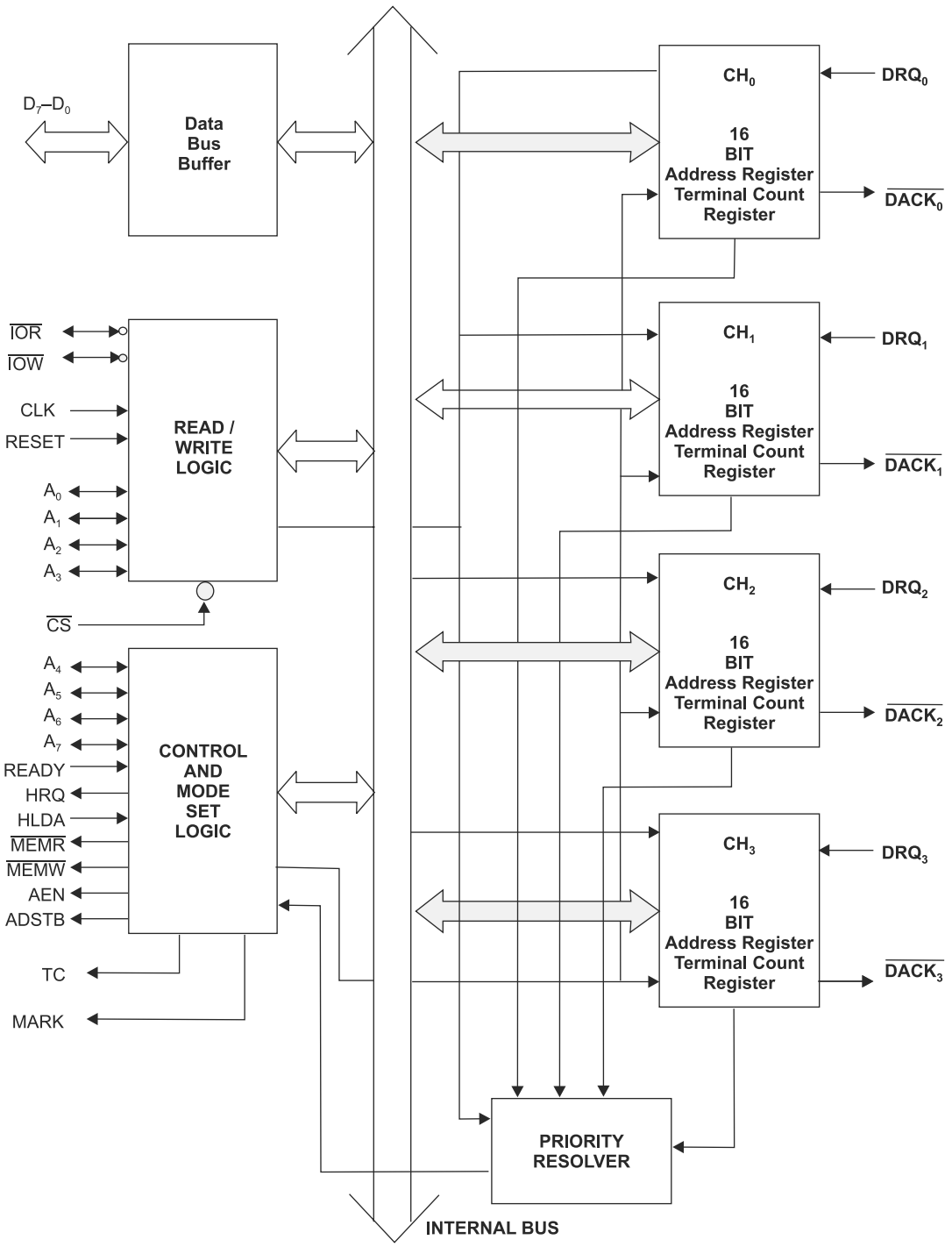


Fig. 17.4 Architecture of 8257

17.3.1 Register Organisation of 8257

The 8257 performs the DMA operation using four independent DMA channels. Each DMA channel of 8257 has two 16-bit registers, namely, DMA address register and terminal count register. There are also two common registers for all the channels such as mode set register and status register. Therefore, there are ten registers of 8257. The CPU can select any one of the ten registers using address lines A_0–A_3. Table 17.1 shows the selection of one of these registers based on A_0–A_3. All registers are explained below.

DMA Address Registers 8257 has four separate DMA channels CH-0 to CH-3. Each channel has a DMA request and DMA acknowledge signal. Each DMA channel has one separate DMA address register. The DMA address register is used to store the starting address of the memory location from where data will be accessed by the DMA channel. Therefore, the starting address of the memory block will be loaded in the DMA address register of the DMA channel. Generally, the 8257 DMA controller will access the block of memory with the starting address stored in the DMA Address Register and transfer to I/O devices through the DMA channel.

Table 17.1 8257 register selection

Register	Byte	Address inputs				Bi-directional data bus							
		A_3	A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CH-0 DMA	LSB	0	0	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
ADDRESS	MSB	0	0	0	0	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
CH-0 Terminal	LSB	0	0	0	1	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
Count	MSB	0	0	0	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
CH-1 DMA	LSB	0	0	1	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
ADDRESS	MSB	0	0	1	0	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
CH-1 Terminal	LSB	0	0	1	1	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
Count	MSB	0	0	1	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
CH-2 DMA	LSB	0	1	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
ADDRESS	MSB	0	1	0	0	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
CH-2 Terminal	LSB	0	1	0	1	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
Count	MSB	0	1	0	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
CH-3 DMA	LSB	0	1	1	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
ADDRESS	MSB	0	1	1	0	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
CH-3 Terminal	LSB	0	1	1	1	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
Count	MSB	0	1	1	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
Mode Set (Program only)	–	1	0	0	0	AL	TCS	CH	RP	EN_3	EN_2	EN_1	EN_0
Status (Read only)	–	1	0	0	0	0	0	0	UP	TC_3	TC_2	TC_1	TC_0

A_{10}–A_{15}—DMA Starting Address, C_0–C_{13}—Terminal Count Value,
Rd & Wr—DMA verify (00), Write (01) or Read (10) cycle selection.
AL—Auto Load, TCS—TC STOP, EW—Extended Write, RP—Rotating Priority,
EN_3–EN_0—Channel Mask Enable, UP—Update Flag, TC_3–TC_0—Terminal Count Status Bits.

Terminal Count Register Each DMA channel of 8257 has one Terminal Count register (TC). The count register is a 16-bit register and this is used to store the number of bytes, which will be transferred through a

DMA channel. Therefore, before starting the actual DMA operation the register must be loaded the number of bytes. The first 14-bits (D_0–D_{13}) of the terminal count register are used for this purpose. The maximum data transfer using 8257 during one DMA operation can be 16K bytes. One of data transfer is known as a DMA cycle. Hence to transfer a block of data, large numbers of DMA cycles are required.

The most significant bits D_{14} and D_{15} of the count register indicate the type of the DMA function such as DMA write cycle, DMA read cycle or DMA verify cycle. The DMA operation selection and the corresponding bit configuration of the bits D_{14} and D_{15} of the terminal count register is shown in Table 17.2. In DMA write operation, this device is able to transfer data from peripheral devices to the memory. During the DMA read operation, the data is transferred from memory to peripheral devices. In DMA verify operation, the 8257 does not involve with the data transfer.

Table 17.2 Selection of DMA Operation Using A_{I5}/RD and A_{I4} /WR

Bit 15	Bit 14	DMA Operation
0	0	Verify DMA Cycle
0	1	Write DMA Cycle (I/O read and Memory write)
1	0	Read DMA Cycle (Memory read and I/O write)
1	1	(Illegal)

Mode Set Register The mode set register of 8257 can be programmed as per requirement of the programmer. The control word in the mode set register is used to enable or disable DMA channels individually and also determines various modes of operation. To enable a DMA channel, the DMA address register and the terminal count register must be loaded with proper information. The mode set register format is shown in Fig. 17.5.

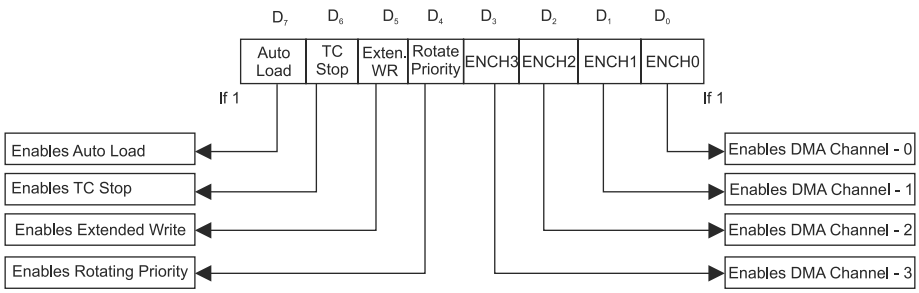


Fig. 17.5 Bit definitions of the mode set register

The bits D_0–D_3 are used to enable or disable any one of the four DMA channels. When D_0 is ' 1', channel 0 is enabled.

If Bit D_4 of mode set register is set, rotating priority is enabled, or else the normal priority, i.e., fixed priority. In the rotating priority mode, the priority of the channels has a circular sequence. The rotating priority is depicted in Fig. 17.6. After each DMA cycle, the priority of channels are changed. The channel, which has just been serviced, will have the lowest priority. When the rotating priority bit is reset, each DMA channel has a fixed priority. In the fixed priority mode, channel 0 has the highest priority and channel 3 has the lowest priority. The priority of DMA operation is shown in Table 17.3.

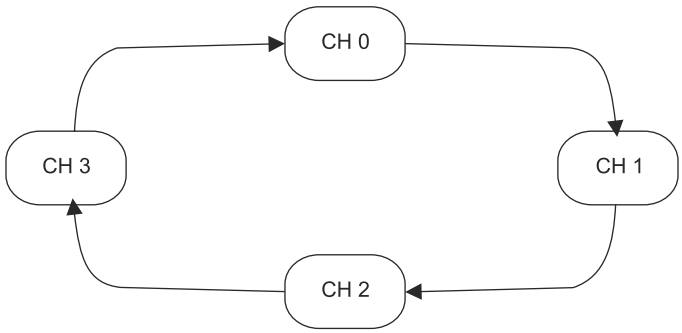


Fig. 17.6 Rotating priority of DMA channels

Table 17.3 Priority operations of DMA channels

Priority	Channel Just Serviced			
	CH 0	**CH 1**	**CH 2**	**CH 3**
Highest priority	CH 1	CH 2	CH 3	CH 0
	CH 2	CH 3	CH 0	CH 1
	CH 3	CH 0	CH 1	CH 2
Lowest priority	CH 0	CH 1	CH 2	CH 3

Bit D_5 can enable the extended write operation. If the bit is set, the duration of MEMW and I/OW signals are extended by activating them earlier in the DMA cycle. This is very useful to interface the peripheral devices with different access times. When the peripheral devices are not accessed within the stipulated time, it is requested to give one or more wait states in the DMA cycle.

Bit D_6 enables terminal count (TC) STOP. When the TC STOP bit is set, a channel is automatically disabled after the terminal count output goes high and prevent any further DMA operation on the same channel. If the DMA operation is to be continued or else if another operation is to begin, the DMA channel must be enabled by a fresh mode set operation in the mode set register.

Bit D_7 of mode set register enables the auto load mode. This bit is set when some DMA operation is repeatedly desired—like sending data to CRT monitor. This is known as repetitive or chained DMA operation. Channels 2 and 3 are used for repetitive DMA operation. Generally, Channel 2 registers are initialised as usual for the first data block, while Channel 3 registers are used to store the block re-initialisation parameters, i.e. the DMA starting address and the terminal count. After the first data block is transferred using DMA Channel 2, the parameters stored in Channel 3 registers are transferred to Channel 2, if the update flag is set.

Status Register The status word register of 8257 is shown in Fig. 17.7. The status word can be read to know the status of the terminal counts of the four channels CH 0–CH 3. Any of the lower order 4-bits of the status word register (D_0–D_3) is set, when the terminal count output corresponding to that channel becomes high. These bits remain set till the status register is read or the 8257 is reset.

The update flag is not affected when the status read operation is performed. This flag can be cleared either by resetting 8257 or by resetting the auto load bit in the mode set register. When the update flag is set, the contents of Channel 3 registers are reloaded to the corresponding registers of Channel 2.

Data Bus Buffer The 8-bit, bi-directional data bus buffer is interfaced with the internal bus of 8257 and also with the external system bus. The data bus buffer is tristate type.

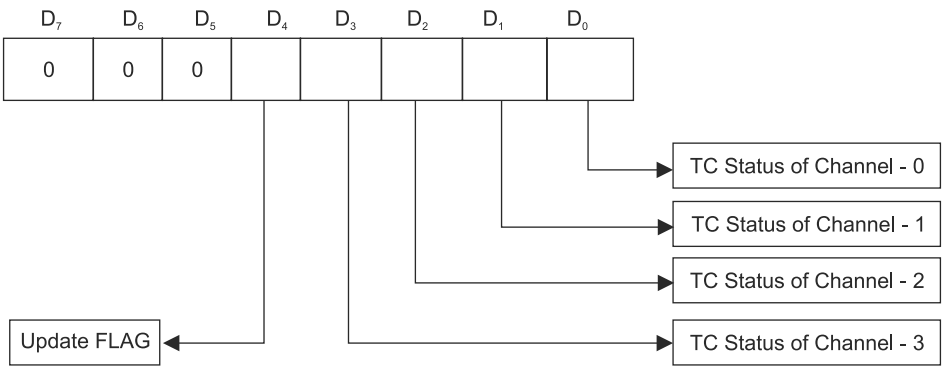


Fig. 17.7 Status register

Read/Write Logic The 8257 can operate in either slave mode or master mode. In the slave mode, the read/write logic accepts the I/O Read ($\overline{IOR}$) or I/O Write ($\overline{IOW}$) control signals. It decodes the A_0–A_3 lines and either writes the contents of the data bus to the addressed internal register or reads the contents of the selected register depending on $\overline{IOW}$ or $\overline{IOR}$. During the master mode operation, the read/write logic generates the $\overline{IOR}$ and $\overline{IOW}$ signals to control the data flow to or from the selected peripheral devices.

Control Unit The control logic unit controls the sequences of DMA operations and generates the following control signals: AEN, ADSTB, MEMR, MEMW, TC and MARK. This unit generates the address lines A_4–A_7 in master mode.

Priority Resolver It resolves the priority of DMA channels of 8257 depending on fixed priority or rotating priority.

17.4 DMA OPERATIONS

The 8257 is able to perform three types of operations such as verify DMA operation, write operation and read operation. This device operates in four different modes, namely, single byte data transfer mode, bust mode, control over drive mode and not ready mode.

17.4.1 Single-Byte Data Transfer Mode

The complete DMA operation of 8257 in single-byte data transfer mode is described below with the help of a flow chart as shown in Fig. 17.8.

S_1 State S_1 is the idle state of DMA operation. In this state, the DMA controller will sample the DMA request inputs (DRQ_n) to check whether any peripheral device wants to data transfer between the device and memory. When any DRQ request is received by the 8257, it sends HRQ (HOLD request) signal to the microprocessor and enter S_0 state.

S_0 State In this state, the DMA controller waits for acknowledgement signal from the CPU at the HLDA input and resolves the priorities of the DMA requests. After detection of a valid HLDA, it exists S_0 and enters S_1 state.

S_1 State When the DMA controller receives HLDA signal, it indicates that the bus is available for the transfer. In the S_1 state, the DMA controller write the MSB of the DMA address register on its D_0–D_7 pins

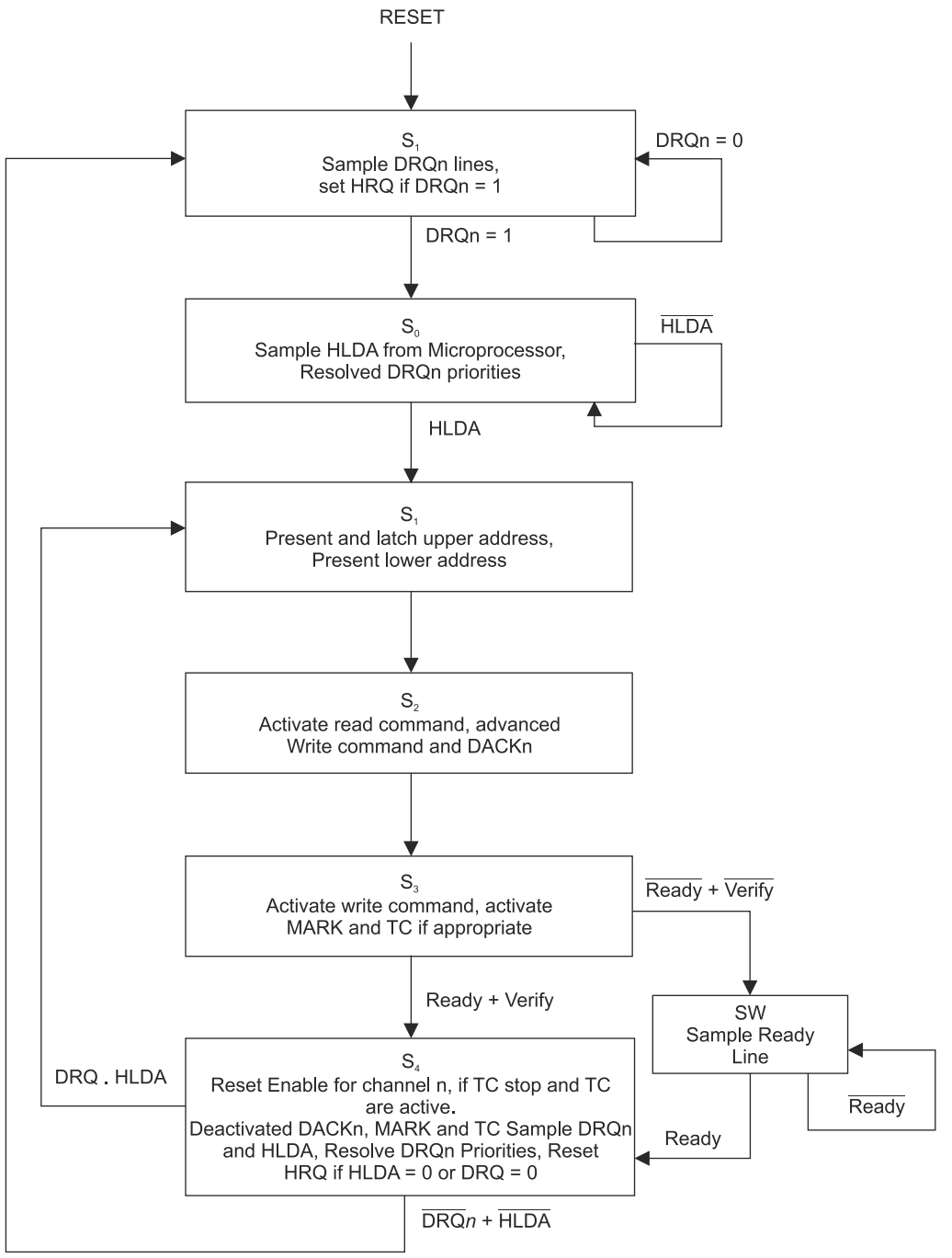


Fig. 17.8 DMA operation state diagram

and some external device is used to latch the MSB by making use of the ADSTB signal. The LSB of the DMA address register is put out on the A_0–A_7 pins. After that it enters S_2 state.

S_2 State In the S_2 state, the DACK line of the used channel is pulled down by the DMA controller to indicate the peripheral device, which already sends DMA request for the DMA transfer. The read command is activated in this state. I/OR is used for a DMA write operation, and MEMR is used for a DMA read operation. When the extended write option is set in this state, it activates the write command. MEMW is used for a DMA write operation, and I/OW is also used for a DMA read operation. After completion of S_2 state, it enters the S_3 state.

S_3 State In the S_3 state, the write command is activated. Then DMA controller sets the TC and MARK outputs if the appropriate conditions are satisfied, and it enters the S_4 state. During S_3 state, it samples the ready input. If the device in which data will be written is not ready, the device makes the ready input to the DMA controller low. Then DMA controller enters into the wait state if the ready input is low. It continues to execute wait cycles until ready input becomes high. When ready input is high, it enters the S_4 state.

S_4 State In this state, if the TC stop and the TC are active (high), the channel just serviced is disabled. The DACK, MARK and TC are deactivated. The DMA controller again samples the DMA inputs, and determines their priorities. If there is no DMA request, it resets the HRQ (HLDA = 0 or DRQ = 0) and enters the S_1 state.

17.4. 2 Burst Mode and Consecutive Transfers

When more than one channel request services at a time, the DMA controller operates in burst mode for data transfer. No overhead is required in switching from one channel to another. In the S_4 state, DRQ lines are sampled and the highest priority request must be indicated for next DMA operation. After completion of highest priority DMA channel operations, the next higher priority DMA request must be serviced. The HRQ line is maintained active till all the DRQ lines become low.

17. 4. 3 Control Override Mode

An external device can interrupt the continuous or burst DMA transfer mode by lowering the HLDA line. After each DMA transfer, the 8257 samples the HLDA line to insure that it is still active. If it is not active, the 8257 completes the current transfer; releases the HRQ line and then returns to the idle state. When DRQ lines are still active, the 8257 may raise the HRQ line in the third cycle and continue normally.

17.4.4 Not Ready Mode

8257 uses four clock cycles to complete a transfer. Figure 17.9 shows the timing diagram of DMA operation. The READY input pin is used to interface 8257 with low speed devices. The READY pin status is sampled in S_3 of the state diagram. If READY= 0, the 8257 enters a wait state. The status of READY pin is sampled in every state till it becomes high. Once the READY = 1, the 8257 proceeds to state S_4 from S_3 state to complete the transfer.

17.5 INTERFACING OF 8257 WITH 8085 MICROPROCESSOR

The 8257 can be interfaced as a memory mapped device or an I/O mapped device. This device can be operated in slave mode as well as master mode. In this section, the slave mode and master mode operations are explained briefly with circuit diagram.

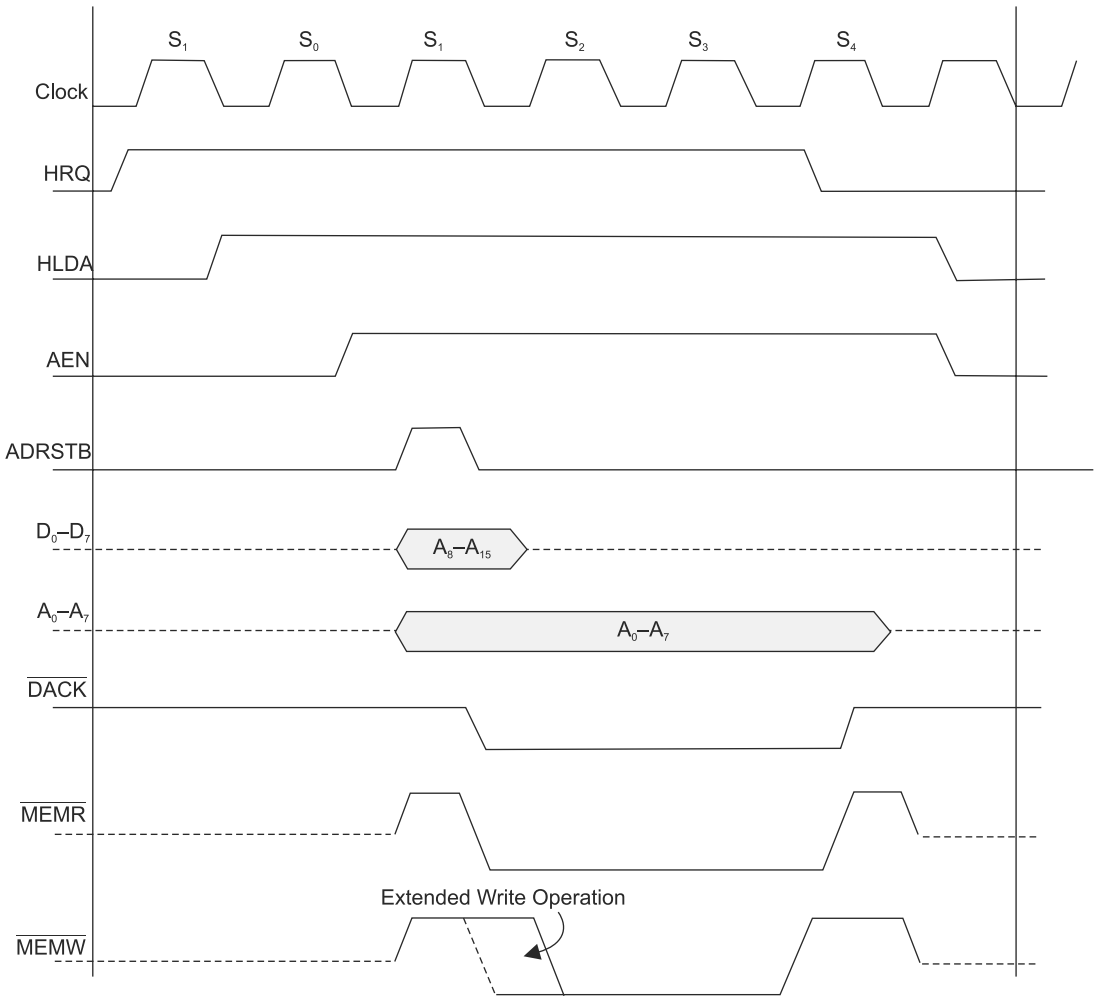


Fig. 17.9 Timing diagram of DMA operation

17.5.1 Slave Mode Operation

The interfacing of 8257 with the 8085 processor in slave mode operation is shown in Fig. 17.10. In this case, the 8257 IC is connected in I/O mapped I/O mode and the $\overline{IOR}$ and $\overline{IOW}$ pins of the IC are connected to the $\overline{IOR}$ and $\overline{IOW}$ control signals. The data lines D_0–D_7 are connected to the data bus of the microprocessor.

The 8257 can also be connected to the system bus as a memory device instead of as an I/O device. This device operates in memory mapped I/O mode by connecting the system memory control lines to the 8257 I/O control lines and the system I/O control lines to 8257 memory control lines. In this case, the $\overline{MEMR}$ and $\overline{MEMW}$ control lines of the system should be connected to the $\overline{IOR}$ and $\overline{IOW}$ input lines of 8257 as shown in Fig. 17.11. The programming of bit 15 (D_{15}) and bit 14 (D_{14}) in the terminal count register is used for different purposes as shown in Table 17.2.

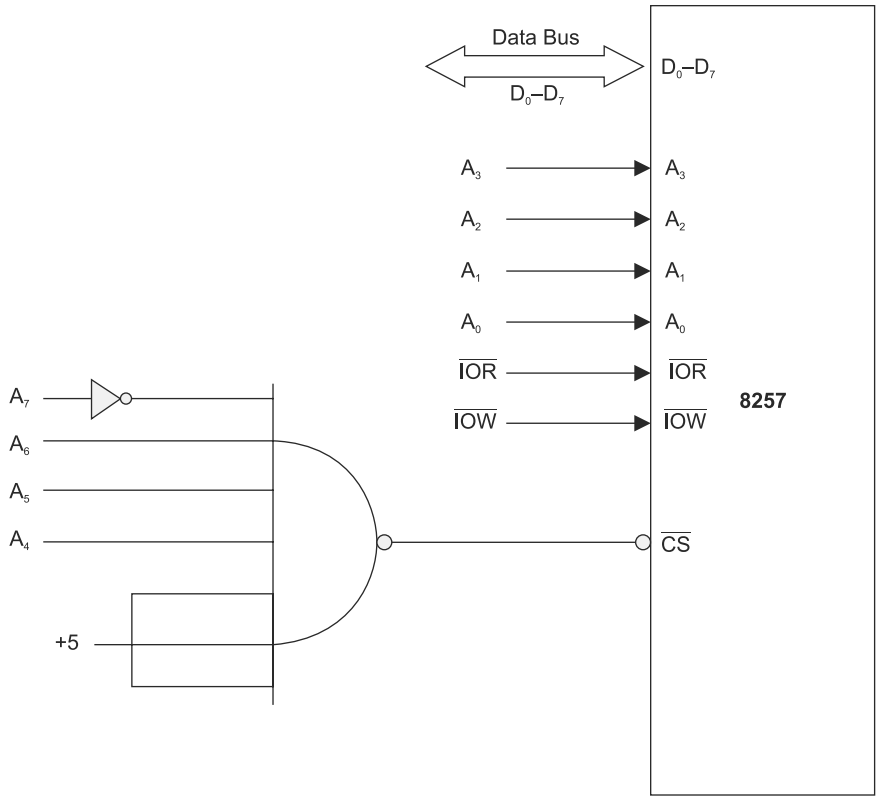


Fig. 17.10 Interfacing of 8257 with 8085 microprocessor in slave mode

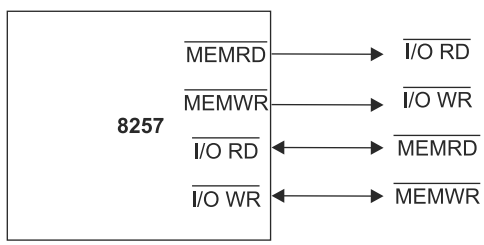


Fig. 17.11 System interface for memory mapped I/O

17.5.2 Master Mode Operation

The 8257 operate in master mode when more than one DMA request lines become active simultaneously. In this mode, CPU is isolated; the DMA controller is activated for data transfer. The DMA controller should send the address of memory location and control signals $\overline{MEMR}$, $\overline{MEMW}$, $\overline{IOR}$ and $\overline{IOW}$. The Interfacing of 8257 with 8085 microprocessor in master mode operation is depicted in Fig.17.12.

In master mode, the data lines D_0–D_7 acts as the higher order address line A_0–A_{15}. The 8257 enable the signal AEN (address enable). The AEN is used to disable the demultiplexed address bus of A_0–A_7 of the 8085

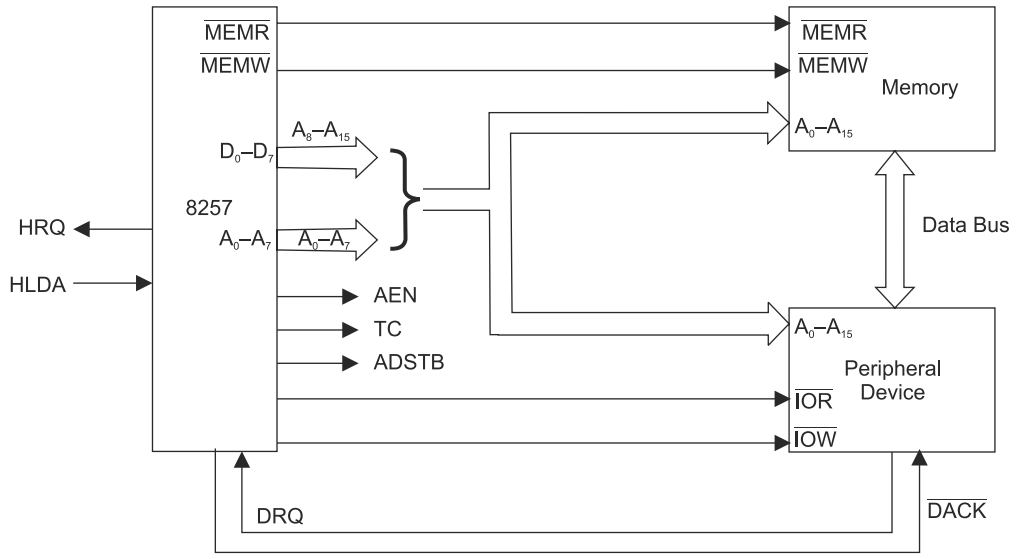


Fig. 17.12 Interfacing of 8257 with 8085 microprocessor in master mode

processor. The 8257 load the low-order address byte of the DMA address register in A_0–A_7 lines. If the AEN signal is high, the ADSTB (Address Strobe) signal strobes the high order byte of the DMA address register using the data lines D_0–D_7.

Depending upon the DMA read or DMA write operation, the control $\overline{MEMR}$, $\overline{MEMW}$, $\overline{IOR}$ and $\overline{IOW}$ are activated properly by the 8257. After completion of one byte data transfer, the content of count register is decremented by one and the address of the DMA address register is incremented by one. Then 8257 send data necessary control signals to transfer next byte. For each byte of data transfer, the $\overline{DACK}$ signal is active low. When all the bytes are transferred, the terminal count (TC) signal becomes high.

Example 17.1 Write a program for the data transfer from memory to a disk. Assume the starting address of memory location is 8000H and sixteen data will be transferred. The address of the DMA address register is 70H and the Terminal Count (TC) register is 71H. The address of mode set register is 78H. Data transfer is done though Channel - 0.

Sol. Figure 17.13 shows the application of DMA for the data transfer from memory to a disk. The program for the data transfer from memory to a disk is given below:

PROGRAM

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
9000	3E, 41		MVI	A,41H	Bit D_0 = 1 to enable channel 0. Bit D_6 = 1 to enable terminal count stop bit. The control word of mode set register is 41 H
9002	D3, 78		OUT	78H	Load 41H into mode set register

9004	3E, 10	MVI	A,10H	Number of data byte (10H)
9006	D3, 71	OUT	71H	will be loaded into least significant byte of terminal count register.
9008	3E, 80	MVI	A,80H	Bit D_7 = 1 to indicate the read
900A	D3, 71	OUT	71H	operation. 80H will be loaded into most significant byte of terminal count register.
900C	3E, 00	MVI	A,00H	16 bit starting address of mem-
900E	D3, 70	OUT	70H	ory location (8000H) will be
9010	3E, 80	MVI	A,80H	written into DMA address reg-
9012	D3, 70	OUT	70H	ister of CH-0

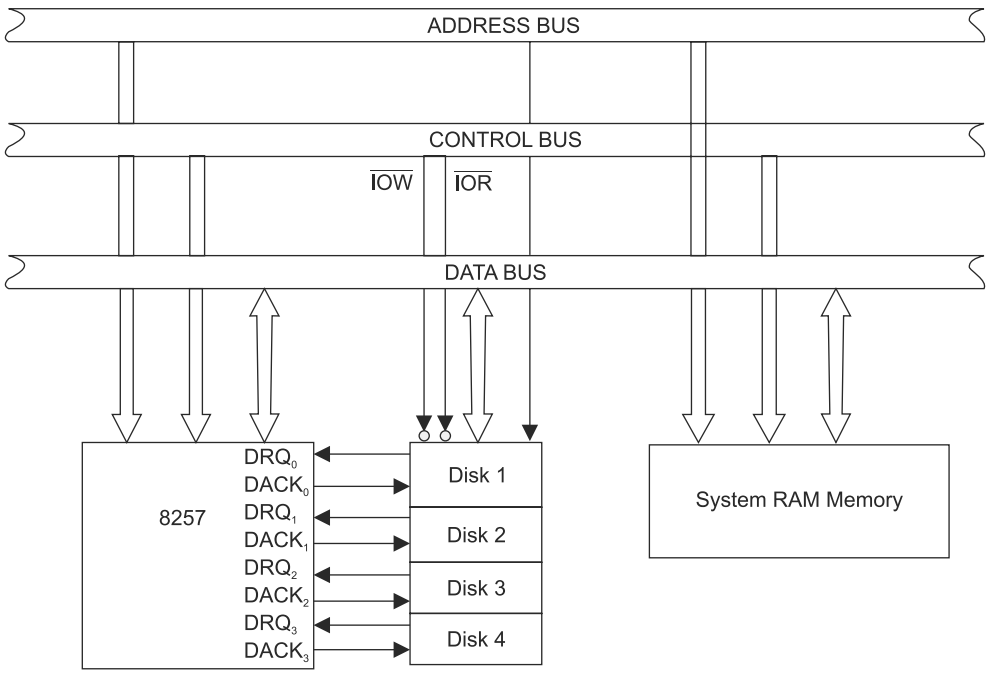


Fig. 17.13 Disk controller using 8257

Example 17.2 Write a program to transfer 45H byte data from a peripheral device to memory. Assume the starting address of memory location is 8000H. The address of the DMA address register is 72H and the terminal count (TC) register is 73H. The address of mode set register is 78H. Data is to be input through Channel 1.

Sol.

PROGRAM

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
9000	3E, 41		MVI	A, 42H	Bit D_1 = 1 to enable Channel 0. Bit D_6 = 1 to enable terminal count stop bit. The control word of mode set register is 42 H.
9002	D3, 78		OUT	78H	Load 42H into mode set register.
9004	3E, 10		MVI	A, 45H	Number of data byte (45H)
9006	D3, 71		OUT	73H	will be loaded into least significant byte of terminal count register.
9008	3E, 80		MVI	A, 40H	Bit D_{15} = 0, D_{14} = 1 and D_{13}–
900A	D3, 71		OUT	73H	D_8 = 0 for write DMA cycle. 40H will be loaded into most significant byte of terminal count register.
900C	3E, 00		MVI	A, 00H	16 bit starting address of memory location (8000H) will be written into DMA address register of CH-1.
900E	D3, 70		OUT	72H	
9010	3E, 80		MVI	A, 80H	
9012	D3, 70		OUT	72H	

Review Questions

17.1. Explain the DMA operation with a suitable diagram. Why are DMA controlled data transfers faster?
17.2. What are the advantages of DMA controlled data transfer over interrupt-driven data transfer?
17.3. Draw the functional block diagram of 8257 DMA and explain operating principle.
17.4. What are the building blocks of 8257?
17.5. Describe the features of 8257. How many I/O devices can access 8257?
17.6. Draw the architecture of 8257 and explain briefly.
17.7. What is the maximum value of KB of data that 8257 can transfer?
17.8. What are the registers available in 8257? How is the 8257 initialised?
17.9. Describe the flow chart of DMA mode of data transfer. What do you mean by DMA cycle?
17.10. Explain how the address registers and terminal count registers for each of CH_0–CH_3 are selected as also the mode set register and status word register.
17.11. Describe the status word register of 8257. Draw a timing diagram for DMA operation.
17.12. Explain the function of the following pins of 8257:
(i) HRQ (ii) HLDA (iii)TC (iv) READY (v) $\overline{DACK}$ (vi) $\overline{DRQ}$ (vii) AEN
(viii) ADSTB (ix) MARK
17.13 What is the purpose of DMA controller ? With respect to 8237/8257 explain the DMA operation.

17.14 What are the priorities of DMA request? Enumerate them

17.15 What are the different transfer modes of 8257/8237? Explain them in brief.

17.16 What is the advantage of DMA controlled data transfer over program controlled data transfer?

17.17 What do you mean by DMA operation? Write down the steps of DMA operation.

17.18 What is fixed priority mode and what is rotating priority mode?

Multiple-Choice Questions

17.1 8257 is a

(a) DMA controller (b) programmable keyboard display interface
(c) counter (d) interrupt controller

17.2 The maximum number of data that will be transferred through 8257 is

(a) 64K (b) 46K (c) 16K (d) 14K

17.3 DMA has

(a) one channel (b) two channels (c) three channels (d) four channels

17.4 The signals used for DMA operation are:

(a) HRQ (b) HLDA (c) HRQ and HLDA (d) none of these

17.5 If a DMA request is sent to the microprocessor with a high signal to the HOLD pin, the microprocessor acknowledge the request

(a) after completing the present cycle (b) immediately after receiving the signal
(c) after completing the program (d) none of these

17.6 For 8257 controller _____ is the highest priority channel by default

(a) CH-3 (b) CH-0 (c) CH-1 (d) any channel

Answers to Multiple-Choice Questions

17.1 (a) 17.2 (c) 17.3 (d) 17.4 (c)
17.5 (a) 17.6 (b)

CHAPTER

18

ADC, DAC, Keyboard, Multiplex Display and LCD Interfacing with 8085, 8086 and 8051

18.1 INTRODUCTION

The Analog to Digital Conversion (ADC) is the reverse operation of Digital to Analog Conversion (DAC). Figure 18.1 shows the block diagram of ADC, which consists of filter, sample and hold, quantizer and digital processor. The filter circuit is used to avoid the aliasing of high frequency signals and passes the baseband frequency signal of ADC. Sometimes this filter is also called antialiasing filter. After the filter, a sample and hold circuit is used to maintain constant the analog input voltage of ADC during the period when the analog signal is converted into digital. This time period is also called conversion time of ADC. The quantizer circuit is used after sample and hold to segment the reference voltage into different ranges. If N number of digital bits represent analog voltages, there are 2^N possible subranges. The quantizer determines the specified subranges corresponding to an analog input voltage. The digital processor can encode the corresponding digital output. There are different types of ADCs. The classifications of ADC architectures based on speed are slow speed ADCs, medium speed ADCs and fast speed ADCs. Single slope and dual slope serial ADCs are slow-speed type and their resolution is very high and accuracy is very good. Medium speeds ADCs are successive approximation ADCs and Parallel or flash ADCs is high speed ADCs. Resolution is moderate for medium speed ADCs and resolution is low for flash ADCs. Accuracy of medium speed ADCs is good but flash ADCs have limited accuracy.

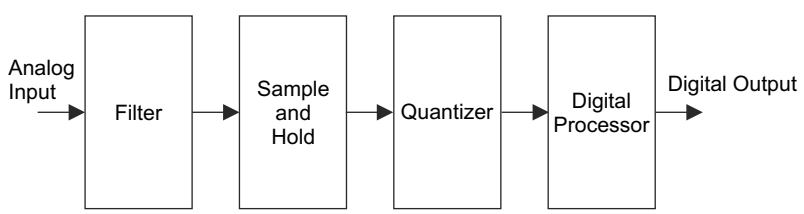


Fig. 18.1 Block diagram of ADC

18.2 COUNTING TYPE A/D CONVERTER

Counting type ADCs are of two types; single-slope serial ADC and dual-slope serial ADC. The operation of a counting A/D converter is explained in this section.

The principle of operation of a single-slope serial ADC is to generate a ramp voltage using DAC, which is compared with the analog input voltage. At the start of the ramp, the counter is started to count from initial value. When the ramp reaches the analog input voltage, the counter is stopped. The digital value in the counter is directly related to the input voltage. This converter takes a longer time to convert a large voltage than a small one and some control signals are required for the start of conversions and end of conversions. The maximum conversion time is $2^{N-}T$, when 2^N clock pulses are required to convert, where N is the number of bits, T is the clock period. The disadvantage of this ADC is that it is unipolar due to a single-slope ramp generator.

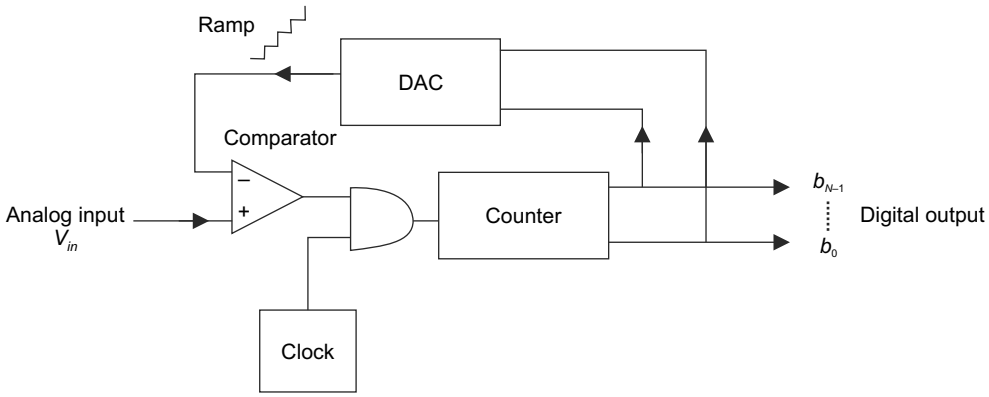


Fig. 18.2 Block diagram of single-slope serial ADC

Figure 18.2 shows the block diagram of single slope analog to digital converter. This converter consists of ramp generator, binary counter, comparator, and AND gate. Here, the counter is used to generate digital output. Initially, analog input is sampled and holds and then applied to positive terminal of the comparator. The counter is in reset condition and clock is applied in AND gate and counter. When the first clock pulse

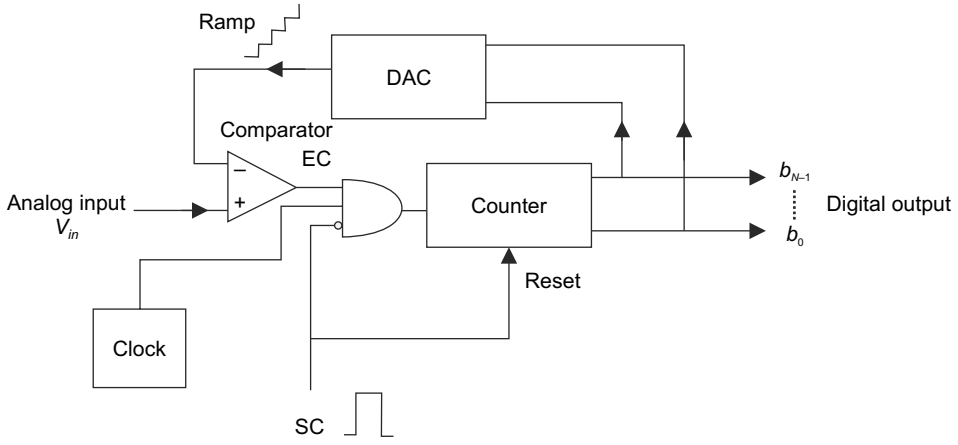


Fig. 18.3 Single-slope serial ADC with SC and EC signals

is applied, the ramp generator starts to integrate a reference voltage *V*. When *Vin* is greater than the output of the ramp generator, the comparator output is high and clock pulse is applied to a counter to count clock pulses. If the output of ramp generator is equal to *Vin*, the comparator output is low. The output of counter is the desired digital output of analog voltage. Single-slope serial ADC with Start of Conversion (SC) and End of Conversion (EC) is illustrated in Fig. 18.3. The conversion sequence of a single slope ADC is given below:

(i) Start of conversion signal resets the counter to zero, and enables the gate to allow clock pulses to be counted in the counter.
(ii) The counter outputs are fed into a DAC to generate a ramp output.
(iii) Then the ramp output is compared with the sampled input signal. The gate output is high till the ramp voltage equals the input signal.
(iv) When the ramp output voltage is equal to the input signal, the gate output becomes low and counting stops.
(v) The gate-disabled signal can be used to indicate the end of conversion.

Figure 18.4 shows the timing diagram of single slope ADC.

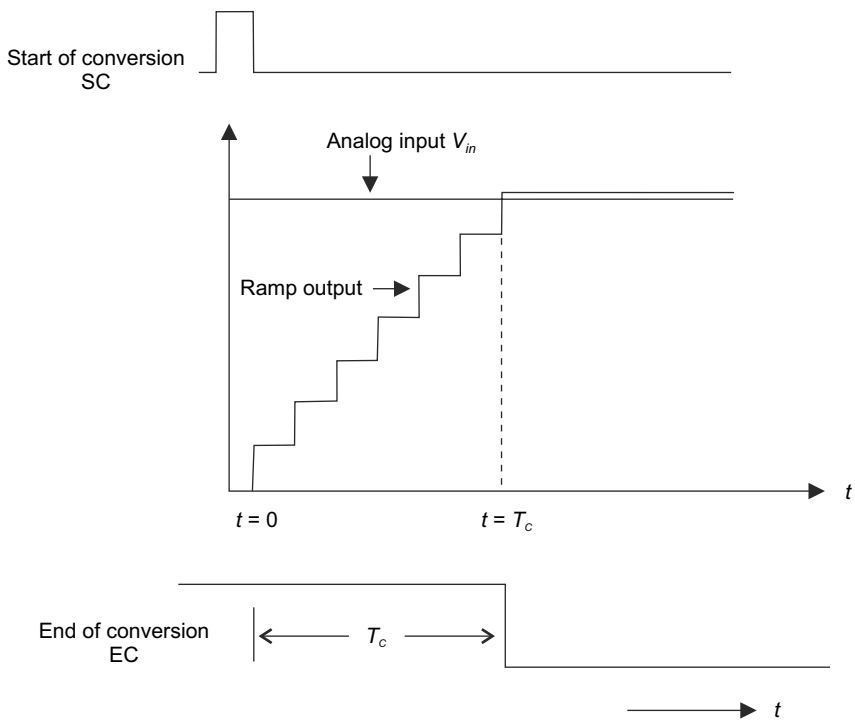


Fig. 18.4 Waveforms of single-slope serial ADC with SC and EC signals

18.3 SUCCESSIVE APPROXIMATION ADC

The major drawback of a single-slope and dual-slope ramp and counter types of ADC is that the length of conversion time is very high. The maximum conversion time is 2^N clock cycles, where *N* is the number of bits. To reduce the conversion time, successive approximation type ADC is very much useful. This converter is similar to a counter-type ADC, but this converter uses a pattern generator rather than a clock to obtain digital equivalent value. The pattern generator simply sets one bit at a time starting with the MSB. Therefore,

the approximation starts by placing logic 1 on the Most Significant Bit (MSB). Then output of the DAC is compared with the sampled input signal. If the output of the DAC is too high, the MSB is reset to logic 0, but if it is too low, it is left at logic 1 and the next bit is set. This process is repeated until all bits are at the correct logic levels in sequence. Consequently, an *N*-bit ADC will only need *N* attempts before all the bits are corrected. Therefore, conversion time is independent of the size of the analog voltage but it depends upon number of bits.

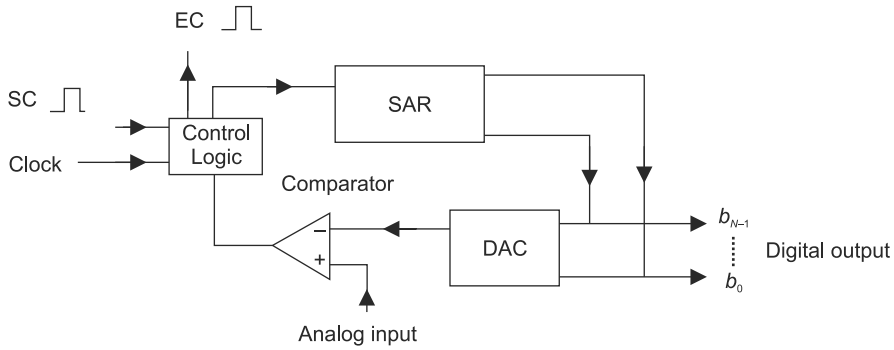


Fig. 18.5 Successive approximation type ADC

Figure 18.5 shows the successive approximation type ADC converter. This converter consists of a comparator, a DAC, digital control logic and Successive Approximation Register (SAR). The function of the digital control logic is to determine the value of each bit in a sequential manner based on the output of the comparator. The conversion processes start with sampling and holding the analog voltage when the start of conversion signal is given. The digital control logic sets the MSB and resets all other bits. This digital data is fed to the DAC, which generates an analog voltage V_{ref} /2 and applied to the comparator to compare with the input voltage V_{in}. When the comparator output is high, the digital control logic makes the MSB 1. If the comparator output is low, the digital control logic makes the MSB 0. After completion of this step, the next MSB is 1 and other bits are 0. Again, the sampled input is compared to the output of the DAC with this digital data. When the comparator is high, the second bit is proven to be 1. If the comparator is low, the second bit is 0. In this way the process will continue until all bits of digital data have not checked by successive approximation. The successive approximation process for converging to the analog output voltage of DAC is shown in Fig. 18.6 (a) and (b). The number of cycles for conversion for *N* bit ADC is *N*. The bipolar analog to digital conversion can be achieved by using a sign bit either $+V$ or $-V$.

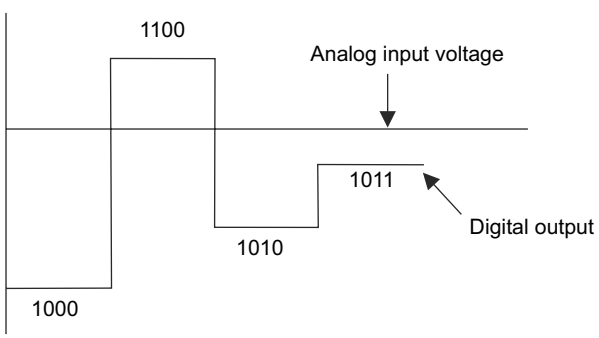


Fig. 18.6(a) Successive approximation ADC for a analog input voltage

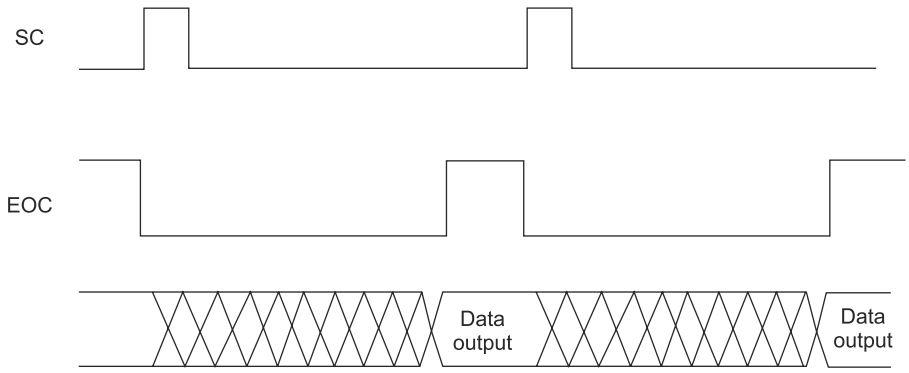


Fig. 18.6(b) Timing diagram of ADC

18.4 PARALLEL OR FLASH CONVERTER

Figure 18.7 shows a flash ADC. In case of a three-bit flash ADC, the reference voltage V is divided into eight different voltages $\frac{V}{14}, \frac{3}{14}V, \frac{5}{14}V, \frac{7}{14}V, \frac{9}{14}V, \frac{11}{14}V, \frac{13}{14}V$, and V. Each voltage is applied to the noninverting terminal of a comparator. The outputs of comparators are fed to the encoder and the encoder output is the digital data of analog input. When V_i is 0.7 V, the output of comparators C_7 and C_6 are 1 and the other comparator C_5, C_4, C_3, C_2, C_1 are low. In this case, the digital output of the encoder is 101, which is equivalent to an analog input voltage. Therefore, a flash-type ADC converter converts analog voltage into a digital output in one clock pulse but in two phases. In the first phase, the analog input voltage is sampled and applied to the comparator inputs. In the second phase, the digital encoder determines the correct digital output and stores it in a register. Flash ADC can be used as a bipolar converter when weighted resistances are connected between $+V$ and $-V$. Table 18.1 shows the analog input, comparator output, and digital output of a flash-type ADC.

Advantage of a flash converter is high speed but many comparators are required. For a three-bit flash converter, 7 comparators are required and an 8-bit flash converter requires 255 comparators on a chip. Therefore, power dissipation is very large.

Table 18.1 Analog input, comparator output and digital output of flash converter

Analog input voltage	**Comparator outputs**				**Digital output**					
V_i	C_7	C_6	C_5	C_4	C_3	C_2	C_1	b_2	b_1	b_0
$0 \le V_i \le V/14$	0	0	0	0	0	0	0	0	0	0
$V/14 \le V_i < 3\ V/14$	0	0	0	0	0	0	1	0	0	1
$3\ V/14 \le V_i < 5\ V/14$	0	0	0	0	0	1	1	0	1	0
$5\ V/14 \le V_i < 7\ V/14$	0	0	0	0	1	1	1	0	1	1
$7\ V/14 \le V_i < 9\ V/14$	0	0	0	1	1	1	1	1	0	0
$9\ V/14 \le V_i < 11\ V/14$	0	0	1	1	1	1	1	1	0	1
$11\ V/14 \le V_i < 13\ V/14$	0	1	1	1	1	1	1	1	1	0
$13\ V/14 \le V_i \le V$	1	1	1	1	1	1	1	1	1	1

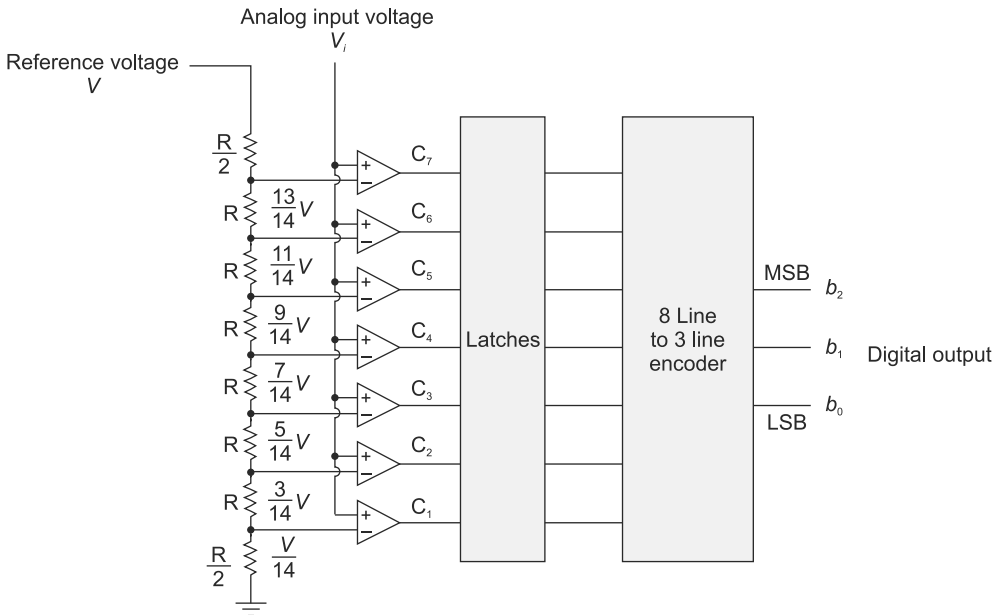


Fig. 18.7 Flash ADC

18.5 SPECIFICATION OF ADC

Generally, manufacturers use the following specifications of an analog to digital converter:

- Analog input voltage range
- Input impedance
- Accuracy
- Quantization error
- Resolution
- Conversion time
- Format of digital output
- Temperature stability

Analog Input Voltage Range It is the maximum allowable input–voltage range in which ADC will operate properly. Actually, it is the difference between the smallest and largest analog input voltages to use the full range of digital outputs. Typical values are 0 to 10 V, 0 to 12 V, $\pm$5 V, $\pm$10 V, and $\pm$12 V.

Input Impedance The input impedance of ADC varies from 1 Kohm to 1 Mohm, depending on the type of ADC. Input capacitance of ADC is approximately some picofarads.

Quantization Error The full-scale range of analog input voltage is quantized for conversion to a finite number of steps. The error is a process of quantization called quantization error. Generally, the quantization error is specified as ½ LSB.

Accuracy The accuracy of an ADC depends on quantization error, digital system noise, gain error, offset error, and deviation from linearity, etc. Accuracy is determined from the sum of all types of errors. Typical values of accuracy are $\pm$0.001%, $\pm$0.01%, $\pm$0.02%, and $\pm$0.04% of full-scale value.

Resolution The resolution is defined by the ratio of reference voltage to number of output states. Actually, it is the smallest change in analog voltage for LSB.

Resolution = Reference voltage / (2^N-1) where N = number of bits of the ADC.

Conversion Time The conversion time of a medium-speed ADC is about 50 μs and for a high-speed ADC, the conversion time is about a few ns. Therefore conversion time varies from 50 μs to a few ns for slow/medium speed to a high-speed ADC.

Format of Digital Output Generally, an ADC always uses any standard code namely unipolar binary, bipolar binary, offset binary, ones complement and twos complement, etc.

Temperature Stability Accuracy of an A/D converter depends on temperature variation. Typical temperature coefficients of error are 30 ppm/ºC.

18.6 ADC ICs

The simplified configuration of an ADC IC is shown in Fig. 18.8. The IC performs analog to digital conversion by using Start of Conversion (SC), End of Conversion (EOC) and output enable signals. Commonly available ADC ICs are single channel 8-bit A/D converter ADC0800, eight channels 8-bit A/D converter ADC0808/0809, twelve channels 8-bit A/D converter ADC0816/0817, 12-bit A/D converter ADC80. The ADC0800 is an 8-bit monolithic A/D converter using P channel ion-implanted MOS technology. It consists of a high-input impedance comparator, 256 series resistors and analog switches, control logic and output latches as shown in Fig. 18.9. Conversion is performed using a successive approximation technique where the unknown analog voltage is compared to the voltage of R network using analog switches. When the appropriate R network voltage matches the unknown voltage, conversion is complete and the digital outputs will be an 8-bit complementary binary word corresponding to the unknown voltage. Figure 18.10 shows the timing diagram of this converter. The features of the ADC0800 are low cost, input ranges ± 5 V to ± 10 V, no missing codes, ratiometric conversion, TRI-STATE outputs, contains output latches, TTL compatible, supply voltages 5 V DC and 12 V DC, resolution 8 bits, linearity ± 1 LSB, conversion speed 40 clock periods, clock range 50 to 800 kHz. Table 18.2 shows the maximum values of ADC's performance characteristics.

Table 18.2 Performance characteristics of ADC0800

Parameters	Maximum value
Non-Linearity	± 2 LSB
Differential Non-Linearity	$\pm$½ LSB
Zero Error	± 2 LSB
Zero Error Temperature Coefficient	0.01%/ºC
Full-Scale Error	± 2 LSB
Full-Scale Error Temperature Coefficient	0.01%/ºC
Input Leakage current	1 μA
Clock Frequency	800KHz
Clock Pulse Duty Cycle	60%
TRI-STATE Enable/Disable Time	1 μs
Start Conversion Pulse	3½ clock pulse
Power Supply Current	20 mA

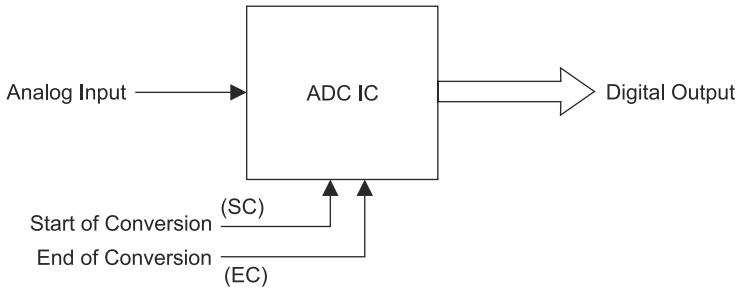


Fig. 18.8 Schematic diagram of ADC

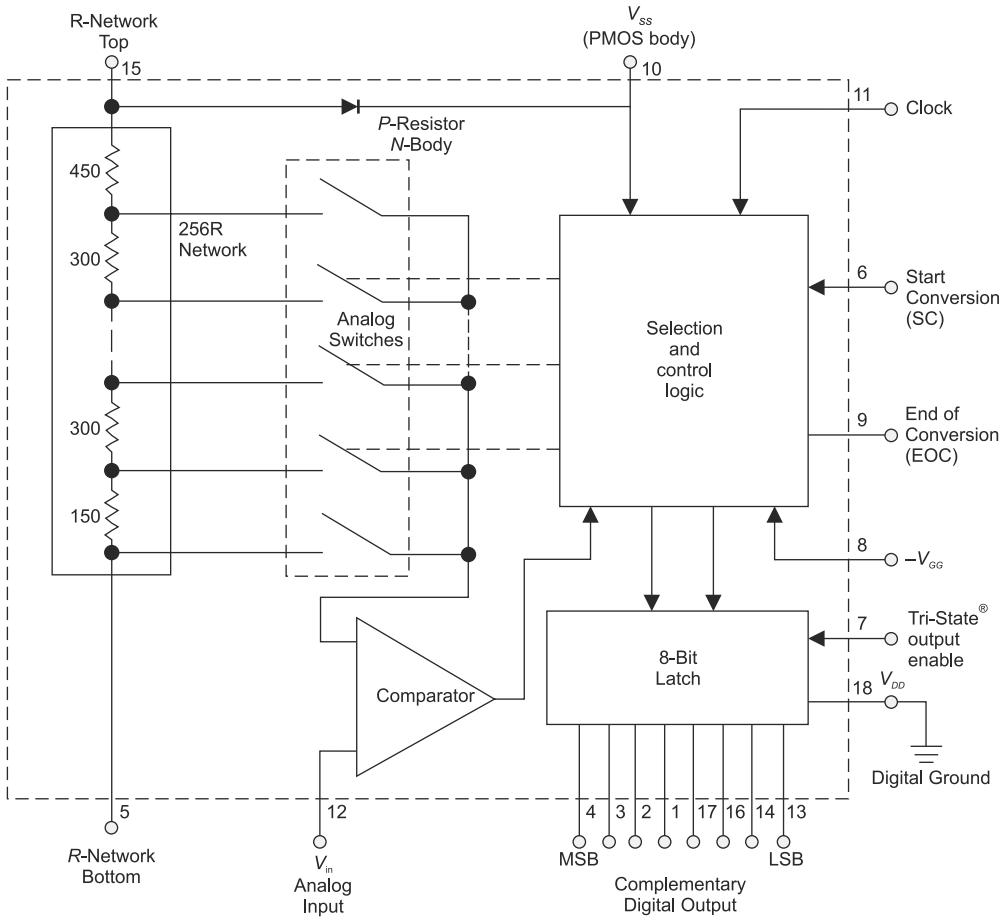


Fig. 18.9 Logic diagram of ADC0800

The ADC80 is a 12-bit successive approximation type A/D converter. It is available in 32 pin DIP. The important performance characteristics of ADC80 is given in Table 18.3.

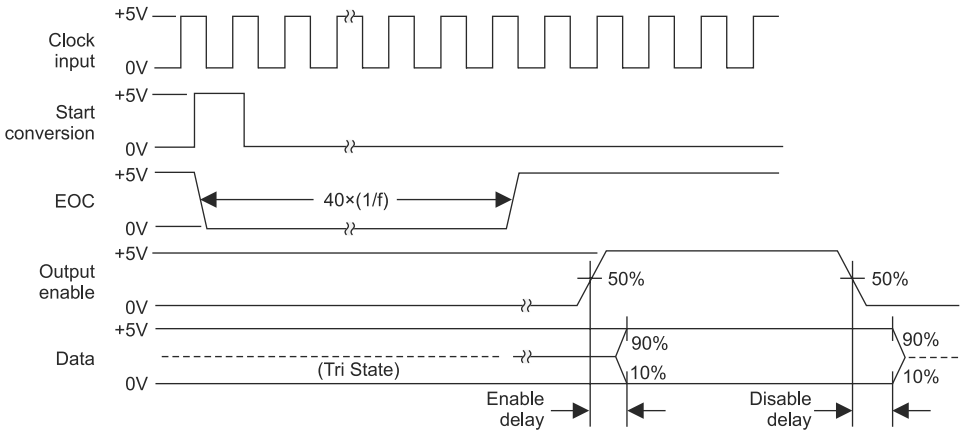


Fig. 18.10 Timing diagram of ADC0800

Table 18.3 Performance characteristics of ADC80

Parameters	Maximum value
Linearity error	±0.012%
Differential Non-Linearity	±½ LSB
Full-Scale Error Temperature Coefficient	30 ppm/°C
Conversion time	25 μs
Analog input voltage	±2.5 V, ±5 V, ±10 V, 0 to 5 V, 0 to10 V
Digital output format	Unipolar and bipolar
Power loss	800 mW

18.7 INTERFACING OF ADC0800 WITH 8085 USING 8255

Figure 18.11 shows the interfacing connections of ADC0800 with 8085 microprocessor using 8255. Here, 8255 is used in between ADC 0800 and 8085 microprocessor. Port A and Port C upper of 8255 are used as inputs and Port B and Port C lower of 8255 are used as outputs. The control word of 8255 when Port A and Port C upper are used as inputs and Port B and Port C lower are used as outputs is 98H. The address of port A is 00H, the address of port B is 01H and the address of port C is 02H. The control word address is 03H.

The Start of Conversion (SC) of ADC is connected with PC_3 of Port C lower, the end of conversion (EOC) is connected with the PC_7 of Port C upper. The output of ADC IC is also connected with Port A of 8255.

Initially, the start of conversion signal will be high to start the conversion process. For this, 08H is sent to Port C by the microprocessor. The port address of Port C is 02H. The instruction OUT 02 sends the content of accumulator to Port C lower and SC pin becomes high. This pin signal will be high only for clock pulse duration as it is used to start the conversion process only. Therefore, 00H is loaded into the accumulator by instruction MVI A,00H. The OUT 02 instruction makes the pin PC_3 low. When the analog to digital conversion has been started, some time is taken by the conversion process. At the end of conversion, the ADC sends the End of Conversion signal (EOC). So the microprocessor should check the EOC signal time to time. If EOC is high, ADC conversion has been completed. To check EOC signal, IN 00H and RAL instructions are

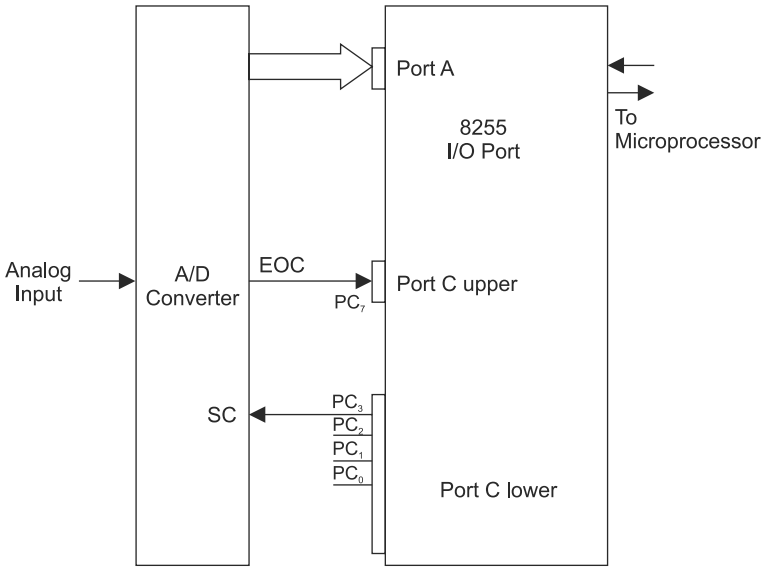


Fig. 18.11 Interfacing of ADC 0800 with 8085 using 8255

used. If carry is generated, the EOC becomes high and the conversion has been completed. When no carry is generated, it means that conversion has not completed. So it jumps to the level LOOP to recheck the status of PC_7. After completion of conversion, the microprocessor reads the output of ADC through the instruction IN 00H. As the ADC output is available in complement form, the CMA instruction is used to convert into the final result. Then the result, content of accumulator can be stored into a specified memory location. Result of ADC conversion is given in Table 18.4. The program for ADC interfacing is given below:

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	3E, 98		MVI	A, 98H	Load control word (98H) of 8255 in accumulator.
8002	D3,03		OUT	03H	Write control word in control word register and initialize ports.
8004	3E, 08		MVI	A, 08H	Send start of conversion signal through PC_3.
8006	D3, 02		OUT	02H	PC_3 is high.
8008	3E, 00		MVI	A, 00H	As PC_3 will be high for one or two clock pulse, make it 0.
800A	D3, 02		OUT	02H	PC_3 becomes low.
800C	DB, 02	LOOP	IN	02	Read end of conversion signal.
800E	17		RAL		Rotate accumulator to check either conversion is over or not.
800F	D2, 0C, 80		JNC	LOOP	If conversion is not completed, jump to LOOP.
8012	DB, 00		IN	00	Read digital output of A/D converter.

Contd.

Contd.

8014	2F	CMA		Complement of ADC output.
8015	21, 00, 81	LXI	H,8100H	
8018	77	MOV	M, A	Store accumulator content in 8000H location.
8019	76	HLT		Stop.

Table 18.4 Result of ADC conversion

Analog Input	Digital Output
5 V	FF
4 V	ED
3 V	C7
2 V	B9
1 V	93
0 V	80

It is clear from the above result that for 5 V analog input digital output is FFH and for 0V input, digital output is 80H. To modify the output result, 80 is subtracted from result. Then the modified result is given in Table 18.5. The modified program is given below.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	3E, 98		MVI	A,98H	Load control word of 8255 in accumulator.
8002	D3, 03		OUT	03H	Write control word in control word register and initialise ports.
8004	3E, 08		MVI	A, 08H	Send start of conversion signal through PC_3.
8006	D3, 02		OUT	02H	PC_3 is high.
8008	3E, 00		MVI	A, 00H	As PC_3 will be high for one or two clock pulse, make it 0.
800A	D3, 02		OUT	02H	PC_3 becomes low.
800C	DB, 02	LOOP	IN	02	Read end of conversion signal.
800E	17		RAL		Rotate accumulator to check whether conversion is over or not.
800F	D2, 0C, 80		JNC	LOOP	If conversion is not completed, jump to LOOP.
8012	DB, 00		IN	00	Read digital output of A/D converter.
8014	2F		CMA		Complement of ADC output .
8015	D6, 80		SUI	80H	Subtract 80H.
8017	21, 00, 81		LXI	H,8100H	
801A	77		MOV	M,A	Store accumulator content in 8100H location.
801B	76		HLT		Stop.

Table 18.5 Modified result

Analog Input	Digital Output
5 V	7F
4 V	6D
3 V	47
2 V	39
1 V	13
0 V	00

18.8 INTERFACING OF ADC 0800 AND MULTIPLEXER WITH 8085 USING 8255

An analog multiplexer is required for analog to digital conversion of larger number of analog inputs. The microprocessor sends the channel select signals to the multiplexer to get the desired analog input voltage from the selected channel. Figure 18.12 shows the schematic circuit diagram, which consists of an analog multiplexer, A/D converter, and 8255. The analog multiplexer has eight channels. To select any one channel, send channel select signals through the Port C lower. The SC signal is connected with the pin PC_3 and EOC signal is connected with PC_7. Analog input is applied to Channel 1 of the multiplexer. When microprocessor sends the 00H into Port C, Channel 1 will be selected and the input voltage of Channel 1 is fed to ADC converter IC. After that the microprocessor sends the SC signal to ADC to start the conversion process. This SC signal will be high only for one clock pulse duration. After that the microprocessor checks the end of conversion signal as to whether the conversion process is completed or not. When EOC is high, the conversion has completed and the microprocessor reads the ADC output and stores it in a memory location. If the analog

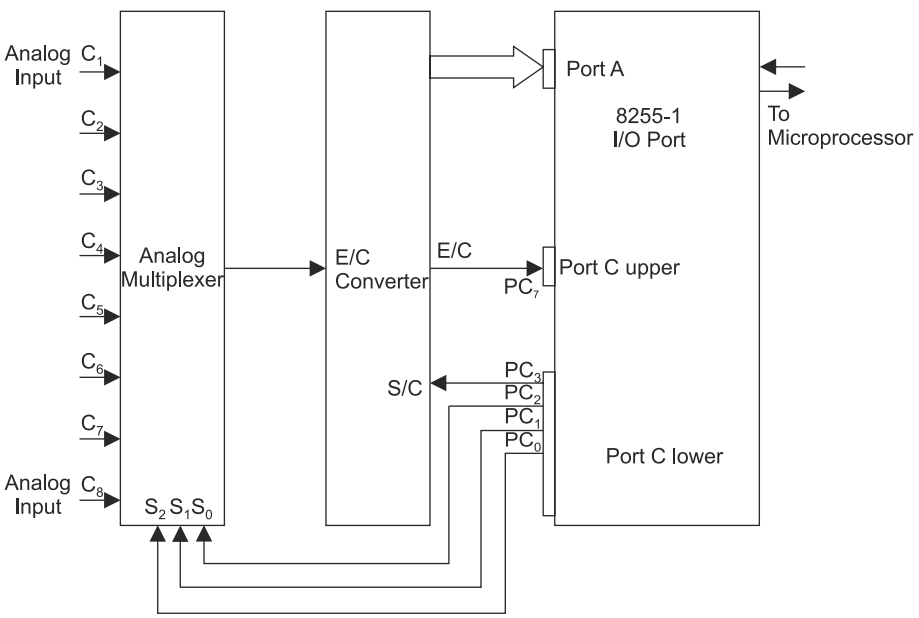


Fig. 18.12 Interfacing of ADC 0800 and multiplexer with 8085 using 8255

voltage is applied to any one of the channels, the applied voltage will be converted into its digital equivalent value in the same way.

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	3E, 98		MVI	A, 98H	Load control word of 8255 in accumulator.
8002	D3, 03		OUT	03H	Write control word in control word register and initialize ports.
8004	3E, 00		MVI	A, 00H	Load 00H to select the multiplexer channel.
8006	D3, 02		OUT	02H	Channel 1 is selected.
8008	3E, 08		MVI	A,08H	Send start of conversion signal through PC_3.
800A	D3, 02		OUT	02H	PC_3 is high.
800C	3E, 00		MVI	A, 00H	As PC_3 will be high for 1 or two clock pulse, make it 0.
800E	D3, 02		OUT	02H	PC_3 becomes low.
8010	DB, 02	LOOP	IN	02	Read end of conversion signal
8012	17		RAL		Rotate accumulator to check either conversion is over or not.
8013	D2, 10, 80		JNC	LOOP	If conversion is not completed, jump to LOOP.
8016	DB, 00		IN	00	Read digital output of A/D converter.
8018	2F		CMA		Complement of ADC output
8019	D6, 80		SUI	80H	Subtract 80H.
801B	21, 00, 81		LXI	H, 8100H	
801E	77		MOV	M, A	Store accumulator content in 8000H location.
801F	76		HLT		Stop.

18.9 INTERFACING OF 12-BIT ADC 0800 WITH 8085 USING 8255

Figure 18.13 shows the interfacing of 12-bit ADC with 8085 microprocessor through 8255. Port A, Port B and Port C upper of 8255 are used as inputs and Port C lower of 8255 is used as output. The start of conversion (SC) of ADC is connected with PC_3 of Port C lower and the End Of Conversion (EOC) is connected with the PC_7 of Port C upper. The output of ADC IC is also connected with Port A and Port B of 8255. PA_0–PA_7 are considered as LSBs of digital output of ADC and PB_0–PB_3 are used as MSBs of digital output of ADC. The PB_4–PB_7 are opened and considered as logic 1. The program for 12-bit ADC interfacing is given below.

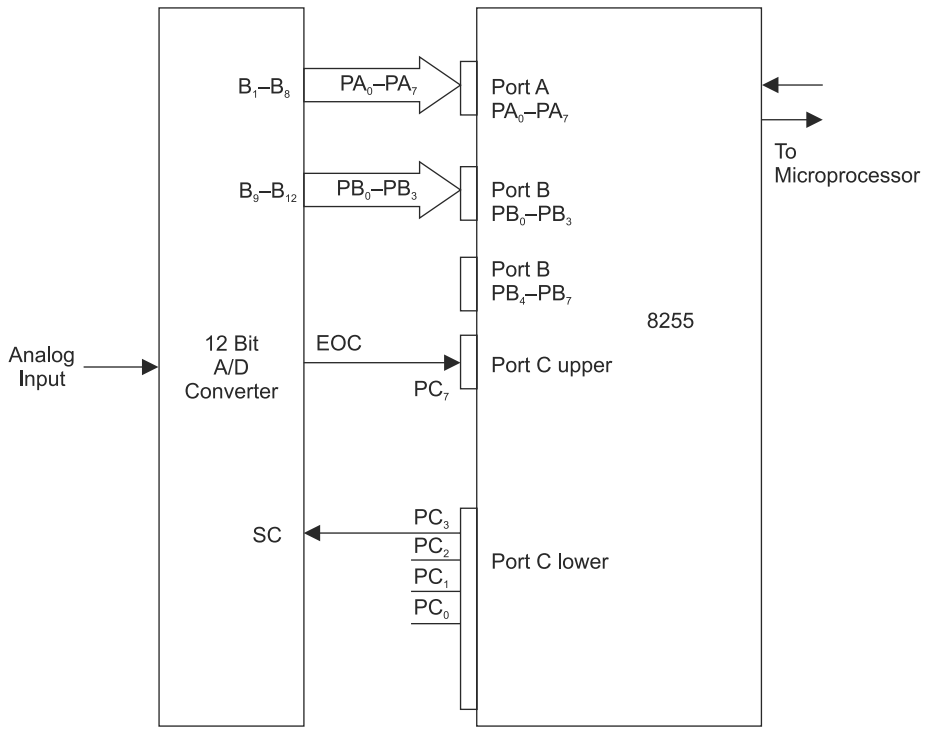


Fig. 18.13 12-bit ADC interfacing with 8085 microprocessor through 8255

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	3E, 9A		MVI	A, 9AH	Load control word (9AH) of 8255 in accumulator.
8002	D3, 03		OUT	03H	Write control word in control word register and initialize ports.
8004	3E, 08		MVI	A, 08H	Send start of conversion signal through PC_3.
8006	D3, 02		OUT	02H	PC_3 is high.
8008	3E, 00		MVI	A, 00H	As PC_3 will be high for 1 or two clock pulse, make it 0.
800A	D3, 02		OUT	02H	PC_3 becomes low.
800C	DB, 02	LOOP	IN	02	Read end of conversion signal.
800E	17		RAL		Rotate accumulator to check either conversion is over or not.
800F	D2, 0C, 80		JNC	LOOP	If conversion is not completed, jump to LOOP.
8012	DB, 00		IN	00	Read digital output of A/D converter from port A.
8014	2F		CMA		Complement of ADC output (LSBs).

Contd.

Contd.

8015	21, 00, 81	LXI	H,8100H	
8018	77	MOV	M,A	Store accumulator content (LSB of output) in 8100H location.
8019	23	INX	H	
801A	DB, 01	IN	01	Read digital output of A/D converter from port B.
801C	2F	CMA		Complement of ADC output (MSBs).
801D	77	MOV	M,A	Store accumulator content (MSB) in 8101H location.
801E	76	HLT		Stop.

18.10 ADC 0808 INTERFACING WITH 8086 USING 8255

Figure 18.14 shows ADC 0808 interfacing with 8086 using 8255 PPI. The analog input is applied to I/P_2 terminal of ADC 0808. Therefore, address pins A, B, C should be 0, 1, 0 respectively to select I/P_2 channel input. The OE and ALE pins are kept at +5V to select the ADC and enable the outputs. The Start Of Conversion (SOC) will be sent through PC_0 of Port C and the End Of Conversion (EOC) will be received from PC_7 of Port C. Port A acts as an 8-bit input data port to receive the digital data output from the ADC. In this case, Port A and Port C upper are used as input ports and Port B and Port C lower are used as output ports. Consequently, the control word of 8255 is 98H, as given below:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Control word
1	0	0	1	1	0	0	0	98H

The assembly-language program for analog-to-digital conversion is given below:

	MOV AL,98H	Load control 98H in AL register
	OUT CWR, AL	Load 98H in control word register and 8255 is initialized
	MOV AL,02H	Load 02H in AL register
	OUT Port B, AL	Get 02H in Port B and select the input channel I/P_2
	MOV AL, 00H	Load 00H in AL register
	OUT Port C, AL	Get 00H in Port C and PC_0 becomes 0
	MOV AL, 01H	Load 01H in AL register
	OUT Port C, AL	Get 01H in Port C and PC_0 becomes 1 and SOC = 1, then conversion will be started.
	MOV AL,00H	Load 00H in AL register
	OUT Port C,AL	Get 00H in Port C and PC_0 becomes 0
WAIT	IN AL, Port C	Read Port C and status of Port C is stored in AL
	RCL	Rotate the content of AL through carry to check EOC, i.e. PC_7 =1
	JNC WAIT	If $PC_7 \neq 7$, jump to WAIT
	IN AL, Port A	If PC_7 = 1 and conversion is completed, then read port A and store data, i.e. equivalent to analog input in AL
	HLT	Stop

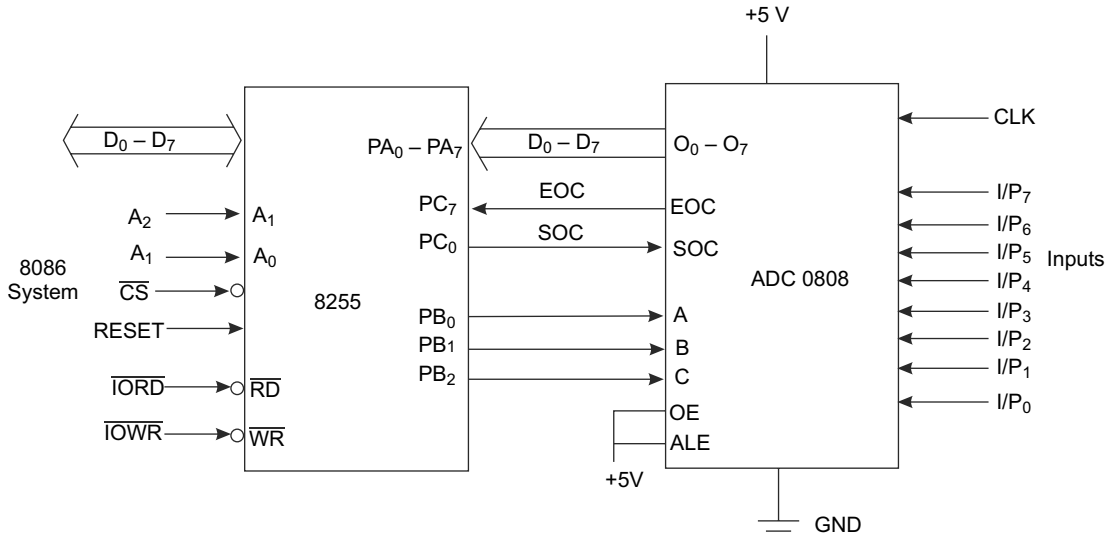


Fig. 18.14 ADC 0808 interfacing with 8086 using 8255

18.11 ADC 0808 INTERFACING WITH 8051 MICROCONTROLLER USING 8255

Figure 18.15 shows ADC 0808 interfacing with 8051 microcontroller using 8255 PPI. The analog input is applied to I/P_2 terminal of ADC 0808. Therefore, address pins A, B, C should be 0, 1, 0 respectively to select I/P_2 channel input. The OE and ALE pins are kept at +5 V to select the ADC and enable the outputs. The start of conversion (SOC) will be sent through PC_0 of Port C and the end of conversion (EOC) will be received from PC_7 of Port C. Port A acts as an 8-bit input data port to receive the digital data output from the ADC. In this case, Port A and Port C upper are used as input ports and Port B and Port C lower are used as output ports. Consequently, the control word of 8255 is 98H as given below:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Control word
1	0	0	1	1	0	0	0	98H

The address of Port A, Port B, Port C and the address of the control word register are E800H, E801H, E802H and E803H respectively. The assembly-language program for analog-to-digital conversion is given below:

```
MOV 0A0,#0E8    Load address of control word register E803H
MOV R0, #03
MOV A, #98H     Load control word 98H in Accumulator
MOVX @R0, A     Load control word 98H in control word register E803H and 8255 is initialized
MOV A, #02H     Load 02H in Accumulator
MOV R0, #01     Load 01H in R0
MOVX @R0, A     Get 02H in Port B and select the input channel I/P2
MOV A, #00H     Load 00H in Accumulator
MOV R0, #02     Load 02H in R0
```

```
        MOVX @R0, A      Get 00H in Port C and PC0 becomes 0
        MOV A, #01H      Load 01H in AL register
        MOVX @R0, A      Get 01H in Port C and PC0 becomes 1 and SOC=1, then conversion will be started
        MOV A, #00H      Load 00H in Accumulator
        MOVX @R0, A      Get 00H in Port C and PC0 becomes 0
WAIT    MOVX A,@R0       Read Port C and status of Port C is stored in AL
        RLC A            Rotate the content of A through carry to check EOC, i.e. PC7 = 1
        JNC WAIT         If PC7 ≠ 1, Jump to WAIT
        MOV R0, #00      If PC7 = 1 and conversion is completed, then read Port A and store data, i.e. equivalent
                         to analog input in A
        MOVX A,@R0
        LJMP 0000
```

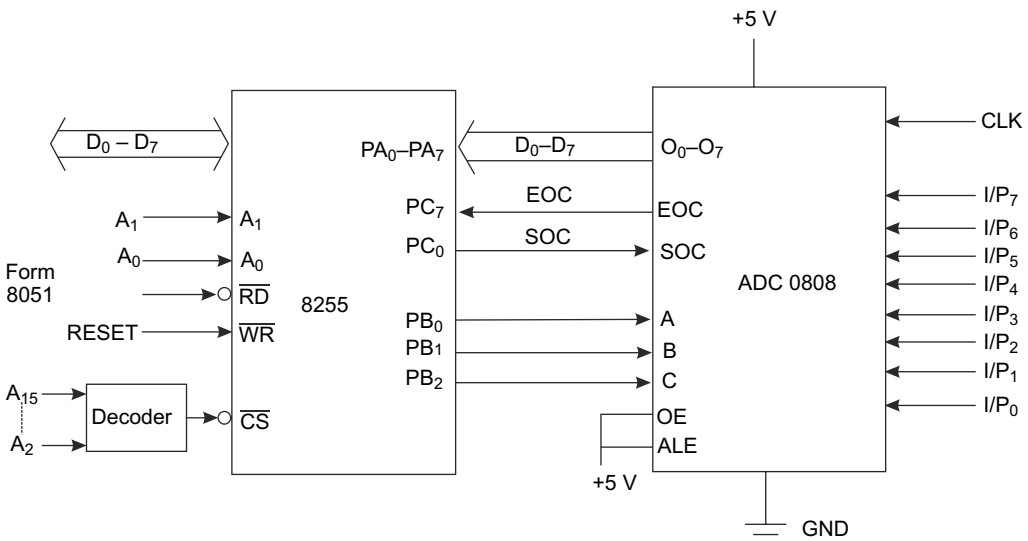


Fig. 18.15 ADC 0808 interfacing with 8051 microcontroller using 8255

18.12 DIGITAL TO ANALOG CONVERTERS (DAC)

The Digital to Analog Converter (DAC) has ability to convert digital signals to analog signals. The digital-to-analog conversion is a process in which digital words are applied to the input of the DAC and an analog output signal is generated to represent the respective digital input. In this conversion process, an *N*-bit digital data can be mapped into a single analog output voltage. Therefore the analog output of the DAC is a voltage that is some fraction of a reference voltage.

So, $$V_{Out} = K \times V_{Ref}$$

where, V_{Out} is the analog voltage output, V_{Ref} is the reference voltage and K is the fraction.

Figure 18.16 shows the block diagram of a DAC converter. When a DAC has N-bits digital inputs (b_0, b_1, b_2, b_3… b_{N-1}) and a reference voltage, V_{Ref}. The voltage output, V_{Out} can be expressed as

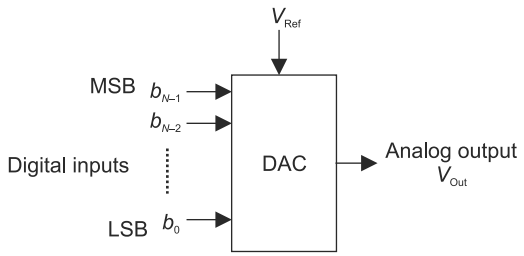


Fig. 18.16 Block diagram of a digital to analog converter

$$V_{Out} = K \times V_{Ref} \times \text{digital inputs}$$

where, K is the scaling factor

$$\text{Digital input} = 2^{N-1}b_{N-1} + 2^{N-2}b_{N-2} + 2^{N-3}b_{N-3} + \ldots + 2^2b_2 + 2^1b_1 + 2^0b_0$$

N = number of bits, b_{N-1}= most significant bit, b_0 least significant bit

18.13 BINARY WEIGHTED OR R/2^N R DAC

Weighted binary DAC and R–2R ladder are the two types of DAC. Each DAC converter input is a multi-bit digital signal, and generates an analog output signal equivalent to digital. Each bit of the signal has a different binary weight. The bit is multiplied by its weighting factor to give its contribution to the whole. The contribution from each bit is then summed, to give the analog equivalent.

The binary-weighted-input DAC circuit is a variation on the inverting summer op-amp circuit. The classic inverting summer circuit is an operational amplifier using negative feedback for controlled gain, with several voltage inputs and one voltage output. The output voltage is the inverted sum of all input voltages when all equal resistances are used in the circuit. If any of the input resistors were different, the input voltages would have different degrees of effect on the output, and the output voltage would not be a true sum. Assume the input resistor values are multiple powers of two: R, $2R$, $4R$ and $8R$, instead of R_1, R_2, R_3 and R_4, respectively.

Figure 18.17 shows the circuit diagram of weighted 4-bit binary DAC. The analog output voltage of a 4-bit weighted DAC can be expressed as follows.

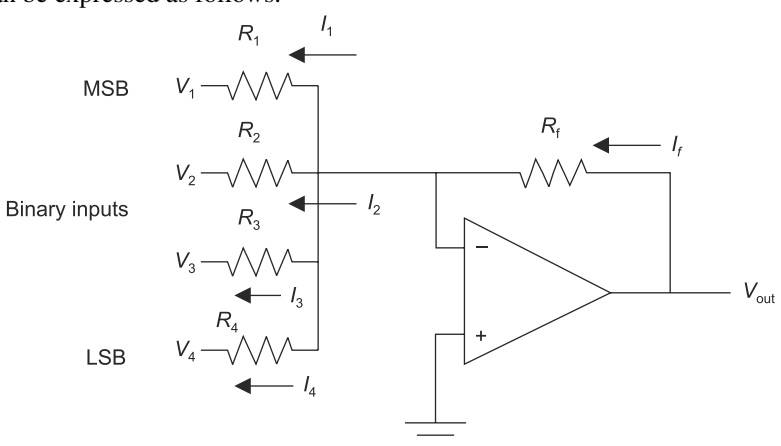


Fig. 18.17 Weighted 4 bit binary DAC

Four input currents I_1, I_2, I_3, I_4 and feedback current I_f are determined from the following expressions:

$$I_1 = V_1/R_1,\ I_2 = V_2/R_2,\ I_3 = V_3/R_3,\ I_4 = V_4/R_4 \text{ and } I_f = V_{out}/R_f$$

The sum of the input currents is equal to feedback current I_f

$$I_f = I_1 + I_2 + I_3 + I_4$$

After substituting all current values in the above expressions,

$$-\frac{V_{out}}{R_f} = \frac{V_1}{R_1} + \frac{V_2}{R_2} + \frac{V_3}{R_3} + \frac{V_4}{R_4}$$

or,
$$-V_{out} = \frac{R_f}{R_1}V_1 + \frac{R_f}{R_2}V_2 + \frac{R_f}{R_3}V_3 + \frac{R_f}{R_4}V_4$$

when $\frac{R_f}{R_1} = 8$, $\frac{R_f}{R_2} = 4$, $\frac{R_f}{R_3} = 2$, $\frac{R_f}{R_4} = 1$, $V_1 = b_3$, $V_2 = b_2$, $V_3 = b_1$ and $V_4 = b_0$ the output voltage can be expressed as

$$-V_{out} = 8b_3 + 4b_2 + 2b_1 + b_0$$

or,
$$-V_{out} = 2^3b_3 + 2^2b_2 + 2^1b_1 + 2^0b_0$$

The weighting of bits b_0, b_1, b_2 and b_3 is 1, 2, 4 and 8 respectively. For this weighting R_4 must be the largest resistor value, but the others resistances are half the value of the previous.

18.14 R–2R LADDER CIRCUIT

R–2R ladder circuit can eliminate the larger component spread in a binary weighted DAC. The R–2R ladder circuit for converting digital to analog converter uses only resistances of R and 2R as shown in Fig. 18.18. Mathematically, analyzing this ladder network is little bit difficult than weighted resistance DAC. In a weighted resistance DAC, each bit effect on output is very easily calculated. But in a R–2R ladder network each binary inputs effect on output can be determined using Thevenin's theorem for each binary input.

The effect of b_0, b_1, b_2 are determined as follows:

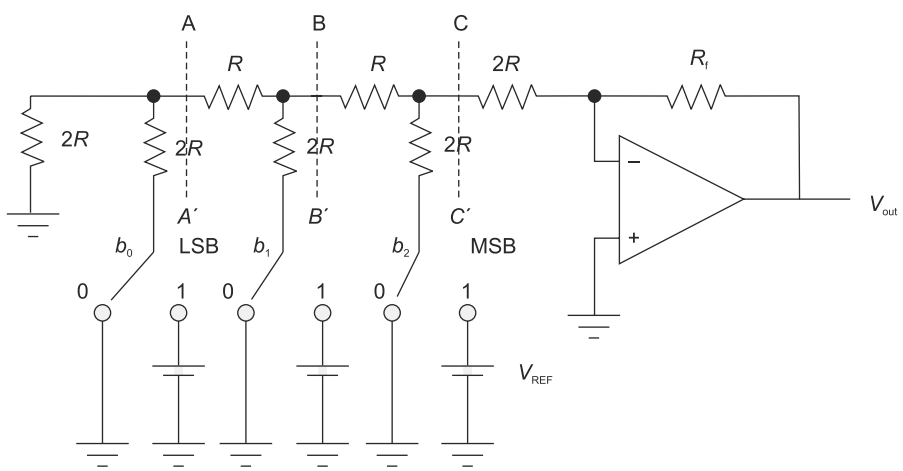


Fig. 18.18 R–2R ladder circuit of 3 bit DAC

When $b_0 = 1$, $b_1 = 0$, $b_2 = 0$, the Thevenin's equivalent resistance and voltage can be determined as given below:

Looking from section A – A', $2R \| 2R$ the equivalent resistance = R and equivalent voltage is $\frac{V_{REF}}{2}$.

Looking from section B – B', R is series with R and sum of these resistance parallel with $2R$ the equivalent resistance = R and equivalent voltage is $\frac{V_{REF}}{2^2}$.

Looking from section C – C′, R is series with R and sum of these resistances parallel with $2R$. Then equivalent resistance is equal to R and equivalent voltage is $\frac{V_{REF}}{2^3}$.

Similarly the equivalent circuit for b_1 is $R_{equ} = 3R$ and $V_{in} = \frac{V_{REF}}{2^2}$ and the equivalent circuit for b_2 is $R_{equ} = 3R$ and $V_{in} = \frac{V_{REF}}{2}$. The equivalent circuit is shown in Fig. 18.19.

R–$2R$ ladder circuit works on the fact that the current reduced by a factor of 2 for each digital input from LSB to MSB. The I_0, I_1, and I_2 currents are as follows:

$$I_0 = \frac{V_{REF}}{2^3 \times 3R} = \quad I_1 = \frac{V_{REF}}{2^2 \times 3R} = \quad I_2 = \frac{V_{REF}}{2 \times 3R}$$

When all bits are 1, the currents flow into the operational amplifier and produces an output voltage

$$\begin{aligned} V_{out} &= -(I_2 + I_1 + I_0) R_f \\ &= -\left(\frac{V_{REF}}{2 \times 3R} b_2 + \frac{V_{REF}}{2^2 \times 3R}, \; b_1 + \frac{V_{REF}}{2^3 \times 3R} b_0\right) R_f \\ &= -\frac{V_{REF}}{2} \frac{R_f}{3R} (4b_2 + 2b_1 + 1b_0) \\ &= -K (4b_2 + 2b_1 + 1b_0) \end{aligned}$$

where, $K = +\frac{V_{REF}}{2} \frac{R_f}{3R}$

In this method of DAC, the larger component spread problem is eliminated and current flowing through the resistance cannot be changed due to switching and behaves as a constant voltage. This DAC is as fast as the binary weighted resistance DAC.

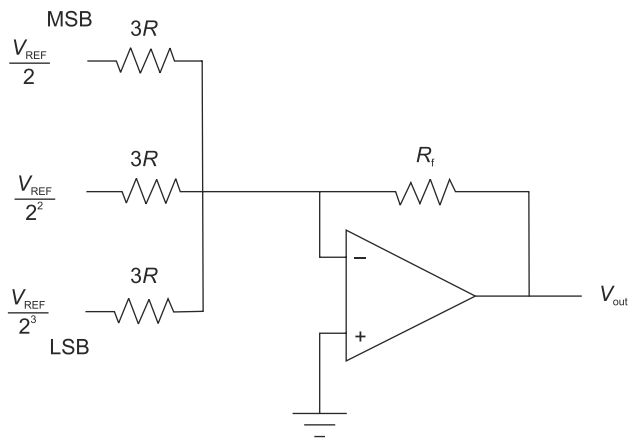


Fig. 18.19 Equivalent circuit of R–2R ladder DAC

18.15 D/A CONVERTER SPECIFICATION

The performance of a D/A converter is measured based on the following parameters: resolution, accuracy, linearity, settling time, temperature sensitivity. Generally the manufacturers specify these parameters in data sheets.

Resolution The resolution of a D/A converter refers to the smallest change in the analog output voltage. It is equivalent to the value of the Least Significant Bit (LSB). For an n bit D/A converter, maximum number of steps is 2^N–1. When the reference voltage is V, the least significant bit value is given below:

$$\text{Resolution} = \frac{\text{Reference voltage}}{\text{Number of steps}} = \frac{V}{2^N - 1}$$

For an 8-bit D/A converter with a full-scale output of 10 V, the resolution is

$$\frac{10}{2^8 - 1} = \frac{10}{255} = 39.2 \text{ mV}$$

Accuracy The output voltage of a D/A converter is different from ideal case. Therefore there is always some error. The accuracy is measured from the difference actual output voltage and voltage for ideal case. When the accuracy of D/A converter is ± 0.25 per cent, the error of converter is $0.25 \times 12/100 = 30$ mV for full-scale voltage $V = 12$ V.

Offset/Zero Scale Error An input code of zero may be expected to give 0 V output. A small offset may be present and the transfer characteristic does not pass through the origin.

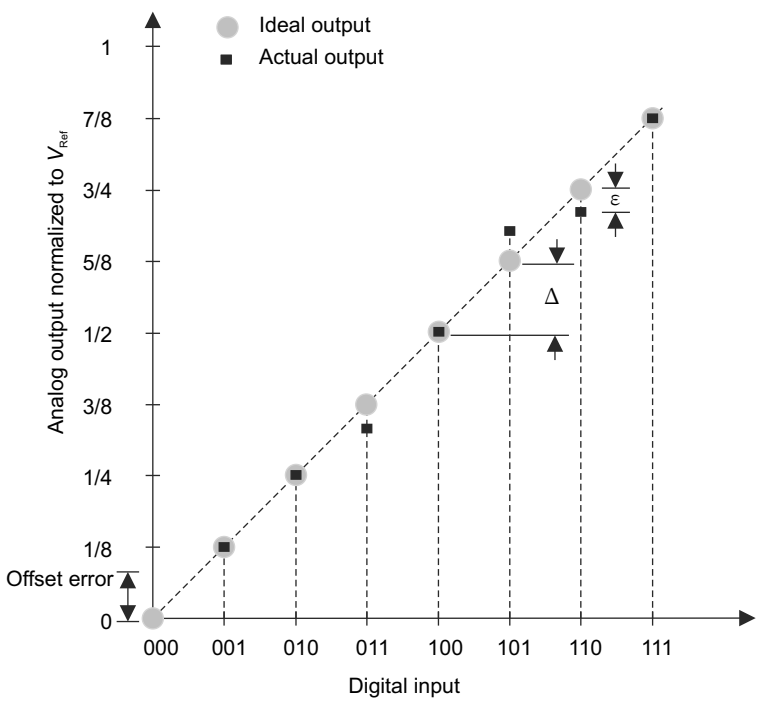


Fig. 18.20 Input/Output characteristics of a D/A converter

Linearity Figure 18.20 shows the input–output characteristics of a D/A converter. Zero offset and gain can develop the characteristic, which passes through the origin and full-scale points. But it is not sure that intermediate points will always lie on a straight line. A very small error in the weighting factor for a fraction LSB will cause nonlinearity. Linearity can be expressed by deviation from the ideal line as a percentage, or a fraction of LSB. It is generally specified as LSB or $\pm \frac{1}{2}$ LSB or $|\varepsilon| < \frac{1}{2} \Delta$.

Settling Time This is usually expressed as the time taken to settle within half LSB. Generally settling time will be about 500 ns.

Temperature Sensitivity The D/A converters are temperature sensitive. When the digital inputs are fixed, the analog output may be varied with temperature due to the temperature sensitivities of the reference voltages, the operational amplifier and converter circuit resistances, etc. Generally, temperature sensitivity of DACs is about $\pm$50 ppm/ºC in general-purpose converters.

18.16 INTERFACING OF DAC ICs WITH 8085 USING 8255

Most commonly DAC ICs are 8-bit DAC0800, 12-bit DAC80, 16-bit PCM54 and PCM55, etc. The DAC0800 ICs are monolithic 8-bit high-speed current output digital to analog converters with typical settling times of 100 ns. When it is used as a multiplying DAC, monotonic performance over a 40 to 1 reference current range is possible. These ICs have high compliance complementary current outputs to allow differential output voltages of 20 Vp-p with simple resistance load as depicted in Fig. 18.21. The features of DAC0800 ICs are as follows.

- Fast settling output current 100 ns
- Full-scale error $\pm$1LSB

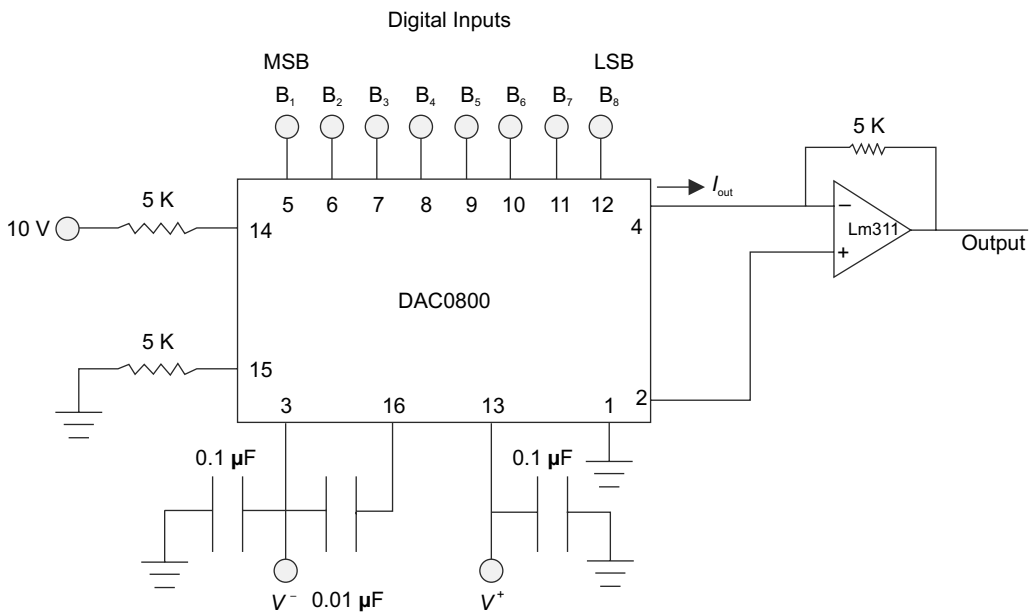


Fig. 18.21 8-bit DAC0800

- Nonlinearity over temperature ±0.1%
- Full-scale current drift ±100 ppm/C
- High output compliance –10 V to +18 V
- Complementary current outputs
- Interface ability with TTL, CMOS, PMOS, etc.
- 2 quadrant wide range multiplying capability
- Power supply range ±4.5 V to ±18 V
- Low power consumption 33 mW at ±5 V
- Low cost

Figure 18.22 shows the circuit diagram for interfacing between DAC0800 and the microprocessor using 8255. The output of Port A is directly interconnected with DAC. All ports of 8255 are operating in output mode and mode of operation is mode 0. In this configuration, the control word is 80H. The programming of DAC is given below:

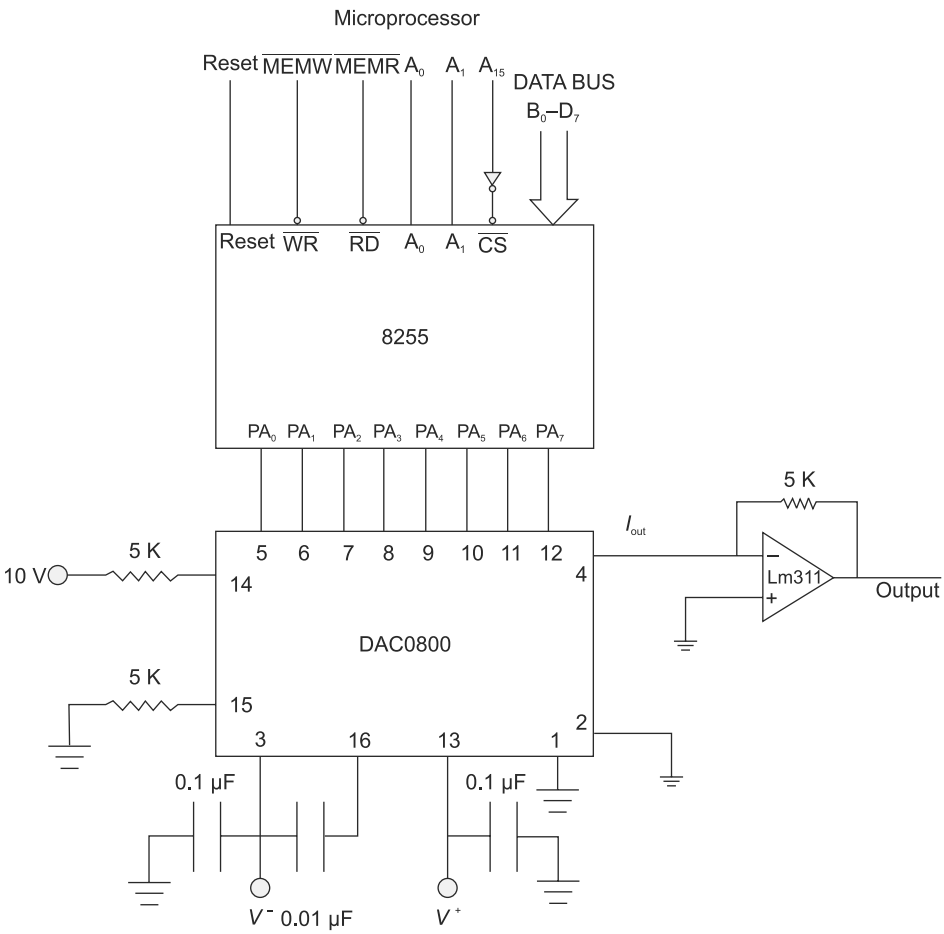


Fig. 18.22 Interfacing of DAC 0800 with 8085 using 8255

Program

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
9000	3E, 80		MVI	A, 80H	Load control word of 8255 in accumulator.
9002	D3, 03		OUT	03H	Write control word in control word register.
9004	3E, FF		MVI	A,FFH	Get FF for digital input to DAC.
9006	D3, 00		OUT	00H	Send to port A for input into DAC.
9008	76		HLT		Stop.

When the digital input is FFH, the analog output voltage is 10 V. If the digital input is 00H the output will be 0V. Table 18.6 shows the analog output voltage with respect to digital input.

Table 18.6 Analog output voltage w.r.t. digital input

Digital input of DAC	Analog output voltage
FFH	10 V
80H	5 V
00H	0 V

The analog output voltage is $I_{out} \times R_L$

$$I_{out} = \frac{(\text{Digital..Input})_{10}}{(256)_{10}} I_{ref}$$

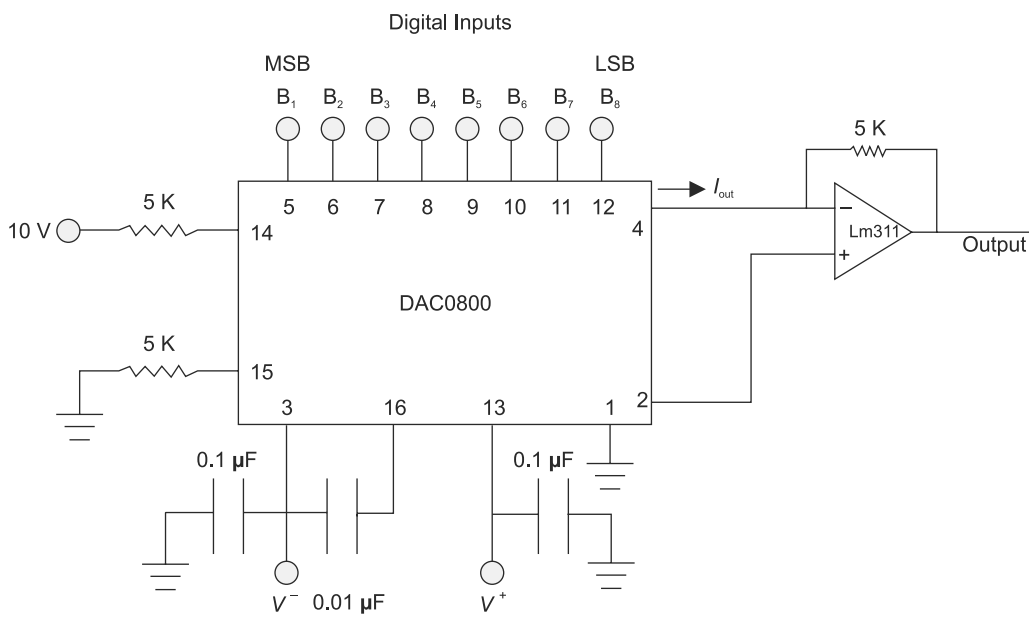


Fig. 18.23 Interfacing of DAC0800

The reference current $I_{ref} = \frac{V_{ref}}{R_{ref}}$

When V_{ref} is 10 V applied through 5K ohms, reference current $I_{ref} = \frac{V_{ref}}{R_{ref}} = \frac{10\,V}{5\,K} = 2$ mA.

The bipolar operation of DAC is shown in Fig. 18.23. Pin 2 of DAC is connected with the non-inverting terminal of operational amplifier. Pin 4 is connected with the inverting terminal of operational amplifier. In this case, when FF input is applied to DAC, output is equal to 10 V. If input is 00H, output will be –10 V. The input and output relationship of bipolar DAC is given in Table 18.7.

Table 18.7 Input/Output relationship of bipolar DAC

Digital input of DAC	Analog output voltage
FFH	10 V
80H	0 V
00H	–10 V

18.17 DAC 0808 INTERFACING WITH 8086 USING 8255

Figure 18.24 DAC0808 Interfacing with 8086 using 8255 PPI. The output of Port A is directly interconnected with DAC. Assume all ports of 8255 are operating in output mode and mode of operation is Mode 0. Consequently, the control word of 8255 is 80H as given below:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Control word
1	0	0	0	0	0	0	0	80H

The reference voltage V_{ref} should be connected with +5 V to generate an output voltage of +5 V amplitude when input is FFH. The assembly language program for digital to analog conversion is given below:

MOV AL,80H	Load control 98H in AL register
OUT CWR, AL	Load 98H in control word register and 8255 is initialized
MOV AL,FFH	Load FFH in AL register
OUT Port A, AL	Get FFH in Port A and FFH will be converted into analog output voltage +5 V
HLT	Stop

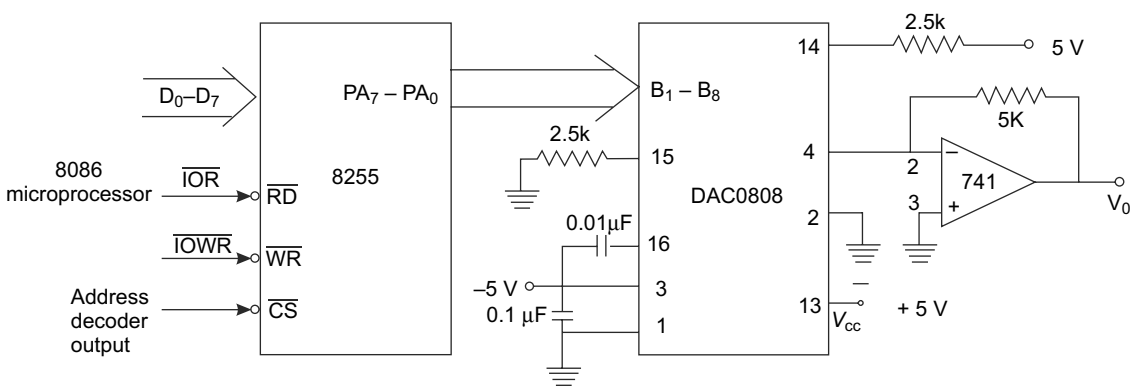


Fig. 18.24 DAC 0808 Interfacing with 8086 using 8255

18.18 DAC 0808 INTERFACING WITH 8051 MICROCONTROLLER USING 8255

Figure 18.25 shows DAC0808 interfacing with 8051 microcontroller using 8255 PPI. The output of Port A is directly interconnected with DAC. Assume all ports of 8255 are operating in output mode and mode of operation is Mode 0. Consequently, the control word of 8255 is 80 H as given below:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Control word
1	0	0	0	0	0	0	0	80H

The reference voltage V_{ref} should be connected with +5 V to generate an output voltage of +5 V amplitude when input is FFH. The address of Port A, Port B, Port C and the address of control word register are E800H, E801H, E802H and E803H respectively. The assembly-language program for digital-to-analog conversion is given below:

```
MOV 0A0,#0E8      Load address of control word register E803H
MOV R0, #03
MOV A, #80H       Load control word 80H in Accumulator
MOVX @R0, A       Load control word 80H in control word register E803H and 8255 is initialized
MOV R0, #00
MOV A, #FFH
MOVX @R0, A       Get FFH in Port A and FFH will be converted into analog output voltage +5 V
LJMP 00
```

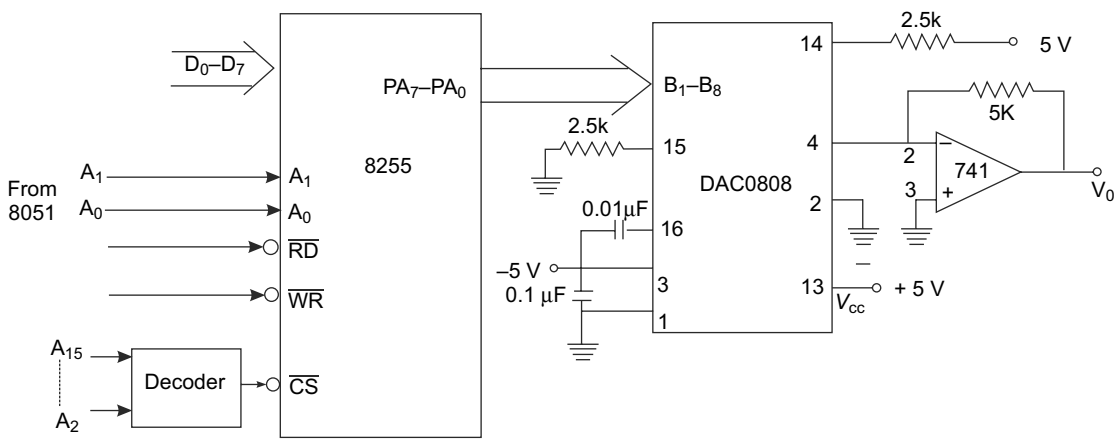


Fig. 18.25 DAC 0808 Interfacing with 8051 microcontroller using 8255

18.19 KEYBOARD INTERFACING WITH 8085 MICROPROCESSOR

The 4×4 keyboard interfacing with 8085 microprocessor using 8255 is depicted in Fig. 18.26. Port A is used as input port for sensing a row of keys and Port B is also used as input port to sense the column number of any closed key. Initially Port B is continuously read to sense any closed key. If any key is pressed, then

read Port B to find the column number and read Port A to get the row number. When row number and column number are known, the numeric value of the key pressed by the user is determined by using formula:

$$\text{Key pressed} = \text{row} \times 4 = \text{column}$$

It is clear from Fig. 18.26 that the address of the control word register is 83H whereas the address of Port A and Port B are 80H and 81H respectively. When Port A and Port C are used as output ports and Port B operates as input port, the control word is 9BH. The assembly-language program for keyboard interfacing is given below:

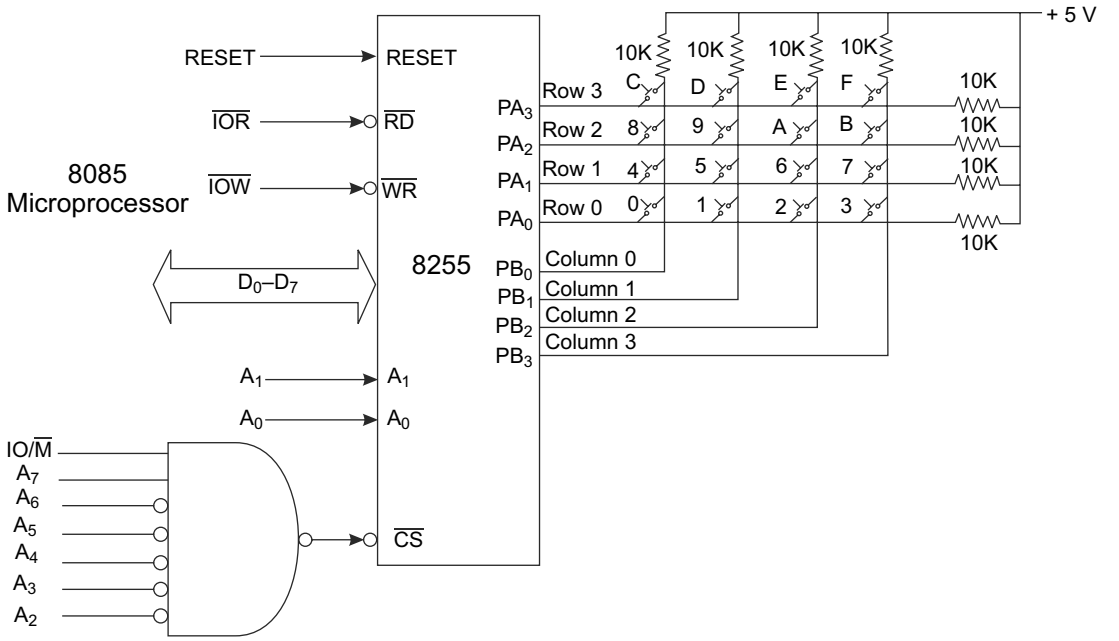


Fig. 18.26 14×4 keyboard interfacing with 8085 microprocessor using 8255

	MVI A, 9BH	Load control word 9BH in Accumulator
	OUT 83H	Load 9BH in control word register
	XRA A	Clear all flags or reset CY flag
	IN 81H	Read Port B, i.e. all columns
	ANI 0FH	Mask data lies D_4–D_7
LOOP I	CPI 0FH	Check if any key is closed
	JZ LOOP I	If any key is not closed, jump to LOOP I
	MVI C, 00H	C=00H, initialize C register
	IN 81H	Read Port B, i.e. all columns
LOOP II	RRC	Content of A will rotate right without carry by one bit
	JNC LOOP III	If no carry, Jump to LOOP III with column number in C
	INR C	Increment C
	JMP LOOP II	Jump to Loop II for rotate right A by one bit and check carry bit
LOOP III	MVI B, 00H	B=00H, initialize B
	IN 80H	Read Port A, i.e. all rows

LOOP IV	RRC	Content of A will rotate right without carry by one bit
	JNC LOOP V	If no carry, Jump to LOOP III with row number in B
	INR BL	Increment B
	JMP LOOP IV	Jump to Loop IV for rotate right A by one bit and check carry bit
LOOP V	MOV A, B	Move row number in A
	RLC	Rotate Accumulator left or A is multiplied by 2
	RLC	Rotate Accumulator left or A is multiplied by 2
	ADD C	Add C with A to get the key number
	HLT	

18.20 KEYBOARD INTERFACING WITH 8086 MICROPROCESSOR

Figure 18.27 shows the 4 × 4 keyboard interfacing with 8086 microprocessor using 8255. Port A is used as input port for sensing a row of keys and Port B is also used as input port to sense the column number of any closed key. Initially Port B is continuously read to sense any closed key. If any key is pressed, then read Port B to find the column number and read Port A to get the row number. When row number and column number are known, the numeric value of the key pressed by user is determined by using formula:

Key pressed = row × 4 = column

It is clear from Fig.18.27 that the address of control word register is 9006 whereas the address of Port A and Port B are 9000H and 9002H respectively. When Port A and Port C are used as output ports and Port B operates as input port, the control word is 9BH. The assembly-language program for keyboard interfacing is given below:

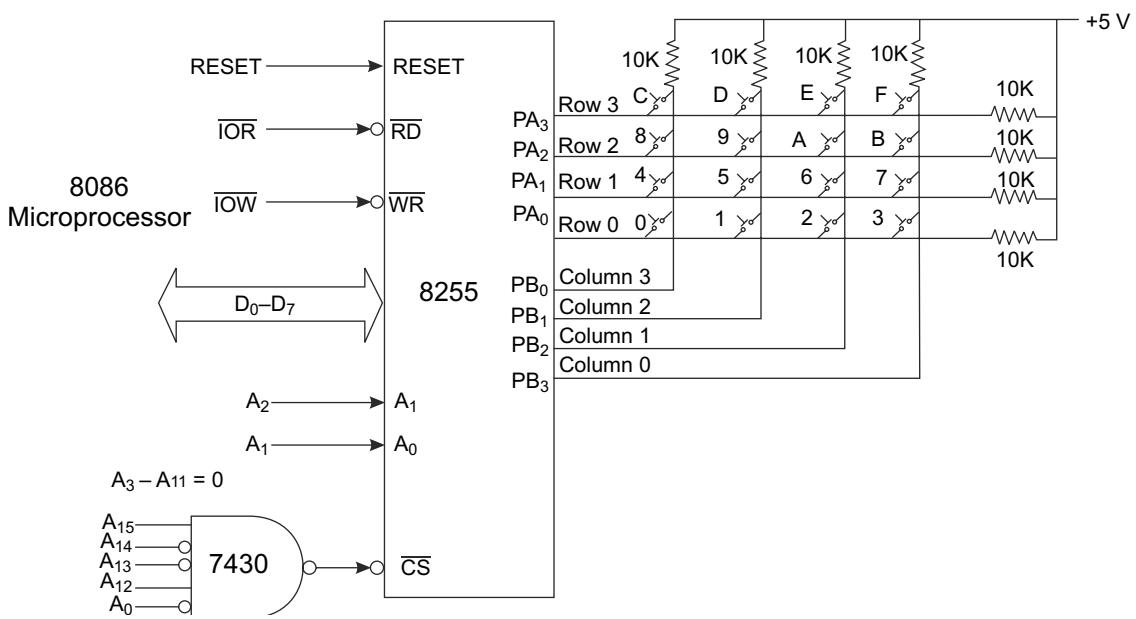


Fig. 18.27 4 × 4 keyboard interfacing with 8086 microprocessor using 8255

	MOV AL, 9BH	Load control word 9BH in AL
	MOV DX, 9006H	Load address of control word register 9006H in DX
	OUT DX, AL	Load 9BH in control word register
	MOV BL, 00H	Initialize BL=00H
	XOR AX, AX	Clear all flags
	MOV DX, 9002H	Load address of Port A 9000H in DX
LOOP I	IN AL, DX	Read Port B, i.e. all columns
	AND AL, 0FH	Mask data lies D_4–D_7
	CMP AL, 0FH	Check if any key is closed
	JZ LOOP I	If any key is not closed, jump to LOOP I
	MOV CL, 00H	CL=00H, initialize CL
	IN AL,DX	Read Port B, i.e. all columns
LOOP II	ROR AL, 01	Rotate right without carry of AL by one bit
	JNC LOOP III	If no carry, Jump to LOOP III with column number in CL
	INC CL	Increment CL
	JMP LOOP II	Jump to Loop II for rotate right AL by one bit and check carry bit
LOOP III	MOV BL,00H	BL=00H, initialize BL
	SUB DX,02	Get address of Port A 9000H in DX
	IN AL,DX	Read Port A, i.e. all rows
LOOP IV	ROR AL, 01	Rotate right without carry of AL by one bit
	JNC LOOP V	If no carry, Jump to LOOP III with row number in BL
	INC BL	Increment BL
	JMP LOOP IV	Jump to Loop IV for rotate right AL by one bit and check carry bit
LOOP V	MOV AL, BL	Move row number in AL
	MOV BL,04H	Load 04H in BL
	MUL BL	Multiply AL with 04H and result in AL
	ADD CL	Add CL with AL to get the key number
	HLT	

18.21 KEYBOARD INTERFACING WITH 8051 MICROCONTROLLER

Figure 18.28 shows the keyboard interfacing with 8051 microcontroller. It is clear from Fig.18.28 that the keyboard is wired as 4 × 4 row-column matrix. The low-order nibble of port 0 is connected to the rows and the high order nibble of port 0 is connected to columns. All rows and columns are connected with the 10K pull-up resistors. As the I/O ports of the 8051 microcontroller can be used as bidirectional port to perform both read and write operations. Therefore, the status of port 0 can be read to scan the keyboard. Three different subroutines such as ROW_READ, COLUMN_READ and CONVERT are used to scan the key which is actually pressed.

To find out the row of depressed key, assume all of the columns are LOW and all of the rows are HIGH. This is possible by executing the instruction MOV P_0, #0FH. The HIGH on the rows is actually a floating state and the rows to be read. The other instructions of ROW_READ subroutine are executed to read each row and to determine the row number that is LOW and the other three rows will be high.

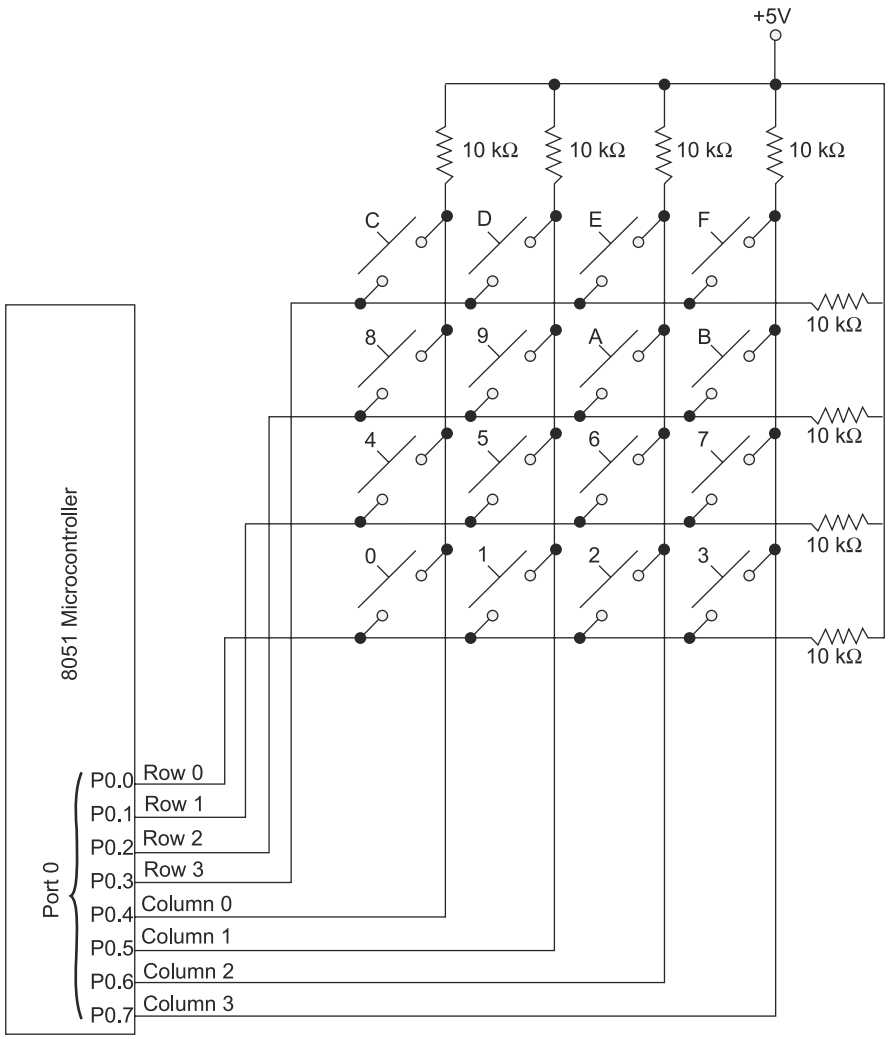


Fig. 18.28 Keyboard interfacing with 8051 microcontroller

After the row read operation, column must be read to determine the column number by executing COLUMN_READ subroutine. When the instruction MOV P_0, #F_0 is executed, all of the rows are LOW and all of the columns are HIGH that float the columns. When any key is still depressed, column of the key will be LOW. After execution of COLUMN_READ subroutine, the column number will be stored in R_1 register.

Lastly CONVERT subroutine converts the row-column combination to the numeric value of the key pressed. Rows 0, 1, 2 and 3 have weighting factors of 0, 4, 8 and 12 respectively. Similarly columns 0, 1, 2 and 3 have weighted factors 0, 1, 2 and 3 respectively. The CONVERT subroutine determines the numeric value of the key pressed by using formula: Key pressed = row × 4 + column. The program for keyboard interfacing with 8051 microcontroller is given below:

Labels	Mnemonics	Operands	Comments
	CALL	ROW_READ	Find out row of key pressed.
	CALL	COLUMN_READ	Find out column of key pressed.
	CALL	CONVERT	Convert row/column to key value.
	LJMP	0000	
ROW_READ	MOV	P_0, #0F	Output 0s to all columns.
	MOV	R_0, #00	ROW = 0.
	JNB	$P_{0.0}$, RET_1	If row 0 is LOW, return to RET_1.
	MOV	R_0, #01	ROW = 1.
	JNB	$P_{0.1}$, RET_1	If row 1 is LOW, return to RET_1.
	MOV	R_0, #02	ROW = 2.
	JNB	$P_{0.2}$, RET_1	If row 2 is LOW, return to RET_1.
	MOV	R_0, #03	ROW = 3.
	JNB	$P_{0.3}$, RET_1	If row 3 is LOW, return to RET_1.
	JMP	ROW_READ	Jump to ROW_READ.
RET_1	RET		Return.

Labels	Mnemonics	Operands	Comments
COLUMN_READ	MOV	P_0, #0F	Output 0s to all rows.
	MOV	R_1, #00	COLUMN=0.
	JNB	$P_{0.4}$, RET_2	If column 0 is LOW, return to RET_2.
	MOV	R_1, #01	COLUMN = 1.
	JNB	$P_{0.4}$, RET_2	If column 1 is LOW, return to RET_2.
	MOV	R_1, #02	COLUMN = 2.
	JNB	$P_{0.5}$, RET_2	If column 2 is LOW, return to RET_2.
	MOV	R_1, #03	COLUMN = 3.
	JNB	$P_{0.6}$, RET_2	If column 3 is LOW, return to RET_2.
	JMP	COLUMN_READ	Jump to COLUMN _READ.
RET_2	RET		Return.

Labels	Mnemonics	Operands	Comments
CONVERT	MOV	B, #04	Move multiplication factor = 04 to B register.
	MOV	A, R_0	Move row number to A.
	MUL	AB	A = row × 4.
	ADD	A, R_1	A = row × 4 + column which is the key value.
	RET		Return.

18.22 LCD INTERFACING WITH 8051 MICROCONTROLLER

Nowadays LCD is widely used in different display applications and it has replaced LEDs for the following reasons:

(i) The price of LEDs is pretty high compared to LCDs and the price of LCDs is declining day by day.

(ii) LCDs can be used to display numbers, characters and graphics.
(iii) LCDs have self refreshing ability.
(iv) There is ease of programming for characters and graphics.

LCD Pin Description The LCD discussed in this section has 16 pins. The function of each pin is given in Table 18.8.

Table 18.8 Function of each LCD pin

Pin	Symbol	I/O	Description
1	V_{ss}	-	Ground
2	V_{cc}	-	+5 V Power supply
3	V_{EE}	-	Power supply to control Contrast
4	*RS*	I	*RS* = 0 to select command register *RS* = 1 to select data register
5	$R/\overline{W}$	I	$R/\overline{W}$ = 0 for write, $R/\overline{W}$ = 1 for read
6	*E*	I/O	Enable
7	DB0	I/O	8-bit data bus
8	DB2	I/O	8-bit data bus
9	DB3	I/O	8-bit data bus
10	DB4	I/O	8-bit data bus
11	DB5	I/O	8-bit data bus
12	DB6	I/O	8-bit data bus
13	DB7	I/O	8-bit data bus
14	LED–	I	Ground for LED backlight
15	LED+	I	+5 V for LED backlight

LCD Command Codes The LCD command codes are given in Table 18.9.

Table 18.9 LCD command codes

Hex Code	Command to LCD instruction register
1	Clear display screen
2	Return home
4	Shift cursor to left or decrement cursor
5	Shift display right
6	Shift cursor to right or increment cursor
7	Shift display left
8	Display off, cursor off
A	Display off, cursor on
C	Display on, cursor off
E	Display on, cursor blinking
F	Display on, cursor blinking
10	Shift cursor position to left
14	Shift cursor position to right
18	Shift entire display to the left
1C	Shift entire display to the right
80	Force cursor to the beginning to 1st line
C0	Force cursor to the beginning to 2nd line
38	5 × 7 matrix and 2 lines

In Figure 18.29, the LCD module is connected to the 8051 microcontroller through its I/O ports. It can also be connected directly to the data bus with the addition of address decoding logic.

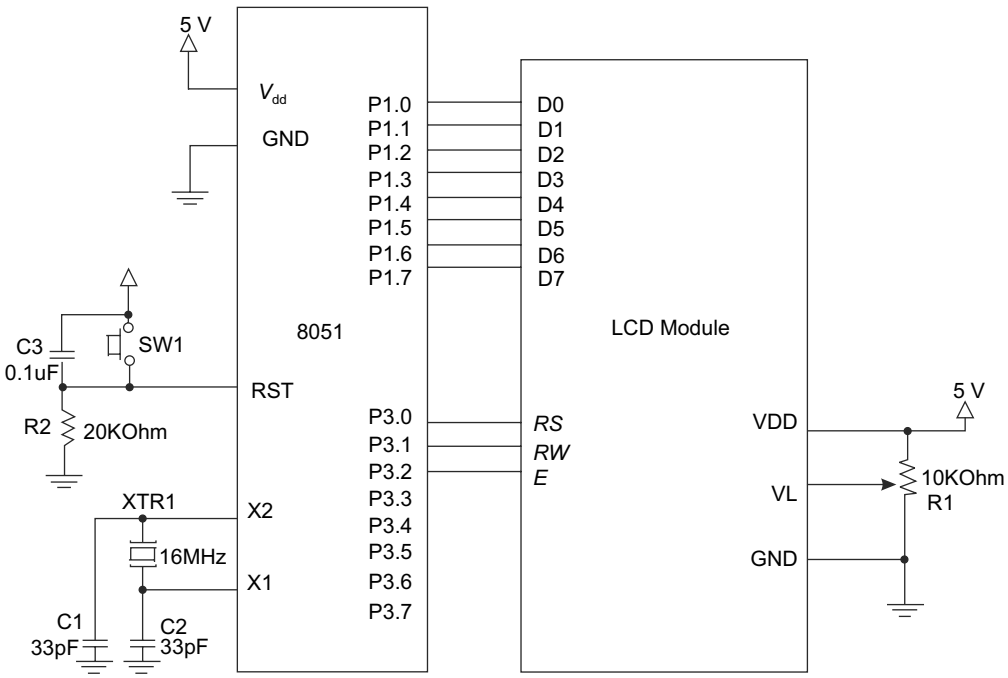


Fig. 18.29 LCD interfacing with 8051

In this module, the process of displaying characters and numerical numbers is divided into three steps. During the first step, the module must be initialized and this can set up the built-in LCD controller chip. In the second step, some user-designed characters must be uploaded to the CGRAM. Then it is possible to display up to 8 custom characters in addition to the 192 characters permanently stored in the module. Lastly, a message consisting of a mix of standard ASCII characters and custom designed symbols is displayed. An assembly-language program to display "LCD" on the display module is given below:

Mnemonics, Operands	*Comments*
MOV A,#38H	Function set- LCD 2 lines, 5 × 7 matrix
ACALL LCD_ command	Call Command subroutine
ACALL LCD_ delay	Call delay subroutine
MOV A,#0EH	Display on and cursor on
ACALL LCD_ command	
ACALL LCD_ delay	
MOV A,#01H	Clear LCD display
ACALL LCD_ command	
ACALL LCD_ delay	
MOV A,#06H	Shift cursor right
ACALL LCD_ command	

```
                ACALL LCD_ delay
                MOV A,#84H          Cursor at line 1 and position 4
                ACALL LCD_ command
                ACALL LCD_ delay
                MOV A,#'L'          Display L
                ACALL LCD_ Data
                ACALL LCD_ delay
                MOV A,#'C'          Display C
                ACALL LCD_ Data
                ACALL LCD_ delay
                MOV A,#'D'          Display D
                ACALL LCD_ Data
                LJMP 0000H
LCD_ command    MOV P1,A            Send the content of A to Port 1
                CLR P3.0            RS = 0 for command
                CLR P3.1            R/W = 0 for write
                SETB P3.2           E = 1 for high pulse
                CLR P3.2            E = 0 for high to low pulse
                RET
LCD_Data        MOV P1, A           Send the content of A to Port 1
                SETB P3.0           RS = 1 for data
                CLR P3.1            R/W = 0 for write
                SETB P3.2           E = 1 for high pulse
                CLR P3.2            E = 0 for high to low pulse

LCD_Delay       MOV R3,#FFH         Load FFH in R3
LOOP1           MOV R4,#FFH         Load FFH in R4
LOOP2           DJNZ R4, LOOP2      LOOP2 continue, if R4 ≠ 0
                DJNZ R3, LOOP1      LOOP1 continue, if R3 ≠ 0
                RET
```

Example 18.1 (a) Give the hardware and software to interface, one seven-segment display with 8085 μp whose address is FC23H

(b) Which addressing mode is used in the above scheme? What change is required if address of the display is FC H?

(a) Figure 18.30 shows one seven segment display unit interfaced to the 8085 microprocessor using a latch. The display code in hexadecimal for 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 are C0, F9, A4, B0, 99, 82, F8, 80 and 98 respectively. The port address FC23H is used to interface the display. The address of seven segment display unit is selected by $A_{15} = A_{14} = A_{13} = A_{12} = A_{11} = A_{10} = 1$, $A_9 = A_8 = A_7 = A_6 = 0$, $A_5 = 1$, $A_4 = A_3 = A_2 = 0$, $A_1 = A_0 = 0$, $IO/\overline{M} = 0$ and $\overline{WR} = 0$. In this interface circuit, $A_0A_1A_2A_3$ are coded using OR gate-1, $A_4A_5A_6A_7$ are coded using OR-gate-2, $A_8A_9A_{10}A_{11}$ are coded using OR-gate-3, $A_{12}A_{13}A_{14}A_{15}$ are coded using OR-gate-4. After that the output of OR-gate1, OR-gate-2, OR-gate-3, and OR-gate-4 are ORed with $IO/\overline{M}$ and $\overline{WR}$, and a low output signal is generated. Therefore, the CLK to 74LS374 will be applied when the decoding logic

input is FC23H. For any other address, the logic gates are not enabled and the seven segment display is not active to latch the data.

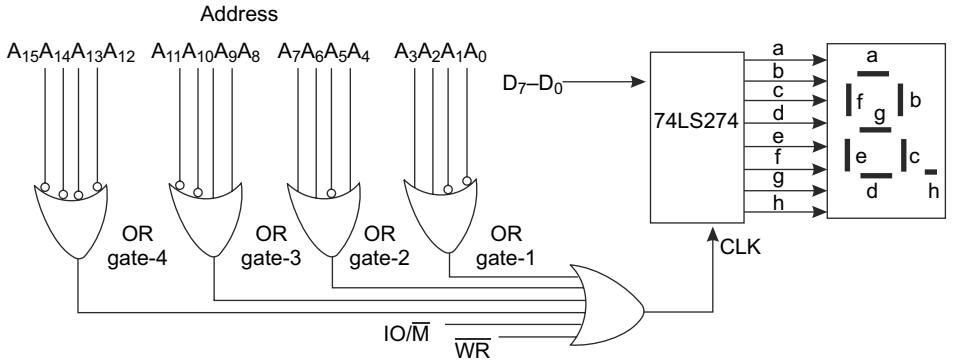


Fig. 18.30 Seven-segment display whose address is FC23H

If we want to display 0 in the seven segment display unit, the following instructions will be executed:

MVI A, C0H	Load C0H into accumulator. The display code in hexadecimal for 0 is C0H.
LXI H, FC23H	Load FC23H in HL Register pair. FC23H is the address of seven segment display unit.
MOV M, A	Content of accumulator send to address FC23H and 0 will be display on the seven segment display unit.

(b) The above scheme is a memory mapped I/O scheme and the indirect addressing mode is used.

If the address of the display is FC H, an I/O mapped I/O scheme will be used and the interface of the seven segment display through 74LS374 is depicted in Fig.18.31.

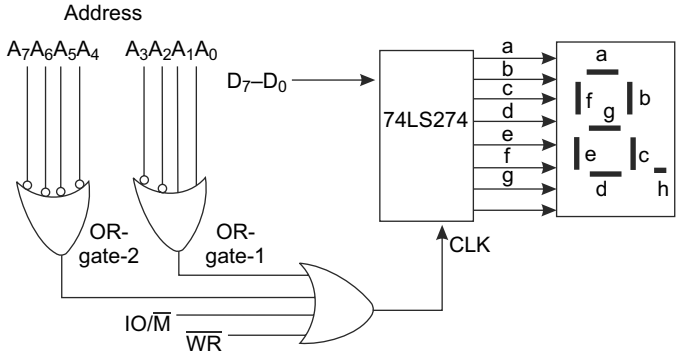


Fig. 18.31 Seven-segment display whose address is FCH

Since the port address FCH is used to interface the seven segment display unit, the address of the seven-segment display unit is selected by $A_7 = A_6 = A_5 = A_4 = 1$, $A_3 = A_2 = 1$, $A_1 = A_0 = 0$, $IO/\overline{M} = 0$ and $\overline{WR} = 0$. In the above interface circuit, $A_0A_1A_2A_3$ are coded using OR-gate-1, and $A_4A_5A_6A_7$ are coded using OR-gate-2. When the output of OR-gate-1 and OR-gate-2 are ORed with $IO/\overline{M}$ and $\overline{WR}$, a low output signal is generated. Therefore, the CLK will be applied to 74LS374 when the decoding logic input is FCH.

If we want to display 0 in the seven-segment display unit, the following instructions will be executed:

MVI A, C0H	Load C0H into accumulator. The display code in hexadecimal for 0 is C0H.
OUT FCH	Content of accumulator send to port address FCH and 0 will be display on the seven segment display unit.

18.23 SEVEN-SEGMENT DISPLAY

Seven-segment display is widely used in calculators, digital watches, and measuring instruments, etc. Generally, Light Emitting Diode (LED), Liquid Crystal Display (LCD) segments provide the display output of numerical numbers and characters. To display any number and character, seven-segment display is most commonly used. Figure 10.32(a) shows the segment identification, and display of decimal numbers from 0 to 9 is given in Fig. 10.32(b). The light emitting diodes emit light when the anode is positive with respect to the cathode. There are two possible connections, namely, common anode and common cathode. In *common-anode connection,* seven anodes connected to a common voltage and cathode will be controlled individually to get the proper display. But in *common cathode connection*, anodes can be controlled individually for display when all cathodes are connected to a common ground of supply voltage as depicted in Fig. 10.33.

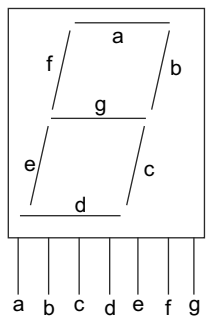


Fig. 18.32 (a) Segment identification (b) Numerical displays

Figure. 10.34 shows the block diagram of a 7-segment display. The decimal number 0 to 9 can be displayed

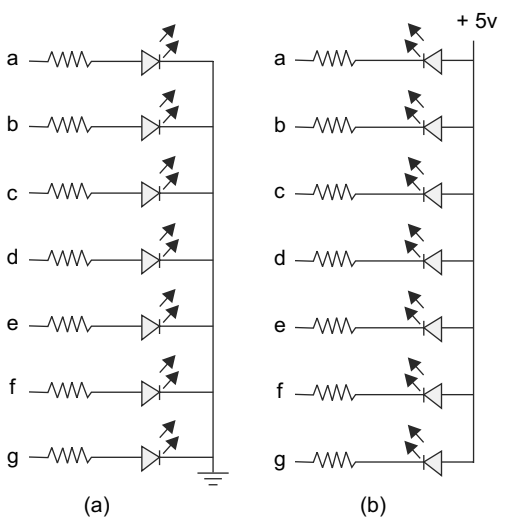


Fig. 18.33 (a) Common-cathode connection, and (b) Common-anode connection of seven segment display

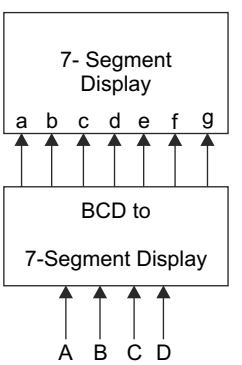


Fig. 18.34 Block diagram of seven-segment display

by the binary coded decimal input. For example, the segments a, b, c, d, e, and f will be bright for decimal number 0. Table 10.1 shows the different segments, which will be bright for decimal numbers 0 to 9. IC 7447 can be used as a decoder circuit for converting binary coded decimal inputs into seven-segment display. Figure. 10.4 shows the pin configuration of IC 7447 and the pin description is given below:

A_0–A_3 BCD inputs, $\overline{RB1}$ Ripple blanking, $\overline{LT}$ Lamp test input

$\overline{BI}$ / $\overline{BRO}$ Blanking input/Ripple blanking output, $\overline{a} - \overline{g}$ Segment outputs

The IC 7447 decodes the input data given in the truth table 18.10. IC 7447 is BCD to 7-segment decoder with open-collector outputs. The 74LS47 has four input lines of BCD (8421) data, and it generates their complements internally. Then the decoder decodes the data and decoder outputs can be used to drive indicator segments directly. Each segment output sinks about 24 mA in the ON/LOW state and can withstand 15 V in the OFF/HIGH state. Some auxiliary inputs, namely, ripple blanking, lamp test and cascadable zero-suppression functions are also provided in IC 7447. Zero suppression logic is very useful in multi-seven-segment decoder.

Table 18.10 Truth table for seven-segment display

Decimal Number	Inputs				Outputs						
	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1

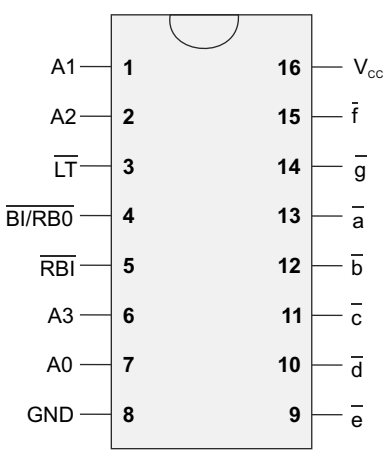


Fig. 18.35 Pin configuration of IC 7447

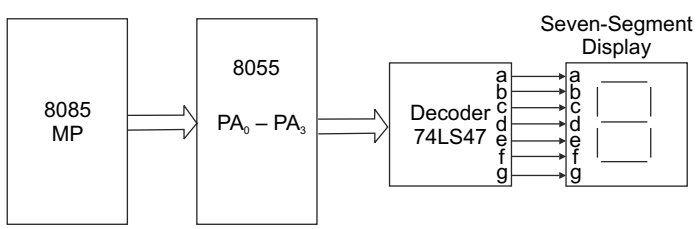


Fig. 18.36 Block diagram of single-digit display

18.23.1 Single-Digit Display

The seven-segment displays are not directly connected with the I/O ports of 8255. Actually they are connected through either buffers or drivers or decoders. Figure 18.36 shows the interfacing of a decoder and a seven-segment display with microprocessor. Assume all I/O ports of 8255 operate as output port. Then control word of 8255 will be 80H. The pins PA_0–PA_3 of Port A are connected to the decoder 74LS47. Therefore, binary inputs corresponding to the decimal number 0 to 9 are applied to 74LS47 and the decimal number 0 to 9 will be displayed in the seven-segment display. The program for displaying the decimal number is given below:

PROGRAM 10.1 for Single Digit Display

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	3E, 80		MVI	A, 80H	Load control word of 8255 in accumulator
8002	D3, 03		OUT	03H	Write control word in control word register and initialize ports
8004	3E, 09		MVI	A,09H	Load 09 in accumulator
8006	D3, 00		OUT	00	Send 09 to Port A for display
8008	76		HLT		Stop

Initially control word of 8255, 80H is loaded in accumulator and writes the control word in control word register to initialise all ports as output ports. After that, 09H is loaded in the accumulator and microprocessor outputs 09H in Port A. The binary logic for 9 is output on the pins PA_0–PA_3, which are connected with seven-segment display for display. The pins PA_4–PA_7 are the MSD of the decimal number 09. Hence the logic for 0 is output on the pins PA_4–PA_7. These pins are not connected anywhere and consequently '0' is not displayed.

18.23.2 Two-Digit Display

In two-digit display, two decoder drivers and two seven-segment displays are used as shown in Fig. 18.37. The LSB will be displayed in one and MSD will be displayed in another seven-segment display. Thus, two display units will display two-digit decimal numbers. To display 99H, the program is illustrated. After execution of this program, PA_0–PA_3 will be 9H and PA_4–PA_7 will be 9H. As Port A outputs are connected to two seven-segment display units though decoder IC 74LS47, 99H will be displayed in the seven-segment display.

PROGRAM 10.2 for Two-Digit Display

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	3E, 80		MVI	A, 80H	Load control word of 8255 in accumulator
8002	D3, 03		OUT	03H	Write control word in control word register and initialize ports
8004	3E, 99		MVI	A, 99H	Load 99 in accumulator
8006	D3, 00		OUT	00	Send 99 to Port A for display
8008	76		HLT		Stop

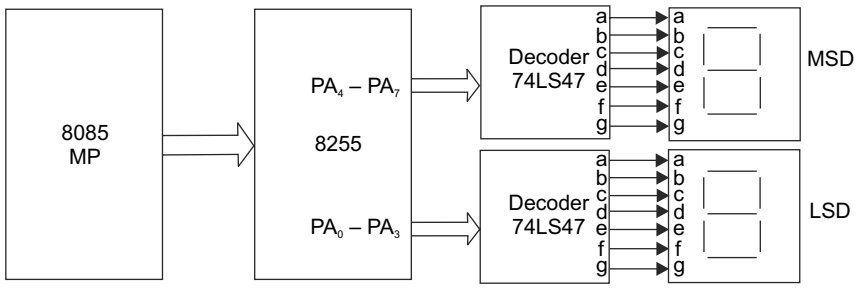


Fig. 18.37 Block diagram of two-digit display

18.23.3 Four-Digit Display

Figure 18.38 shows the four-digit display, which consists of four decoder drivers and four seven-segment display units. Port A and Port B are used for driving the decoder. The program for four-digit display is as follows:

PROGRAM 10.3 for Four-Digit Display

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	3E, 80		MVI	A, 80H	Load control word of 8255 in accumulator
8002	D3, 03		OUT	03H	Write control word in control word register and initialize ports
8004	3E, 12		MVI	A, 12H	Load 12 in accumulator
8006	D3, 00		OUT	00	Send 12 to Port A for display
8008	3E, 34		MVI	A, 34H	Load 34 in accumulator
800A	D3, 01		OUT	01	Send 34 to Port B for display
800C	76		HLT		Stop

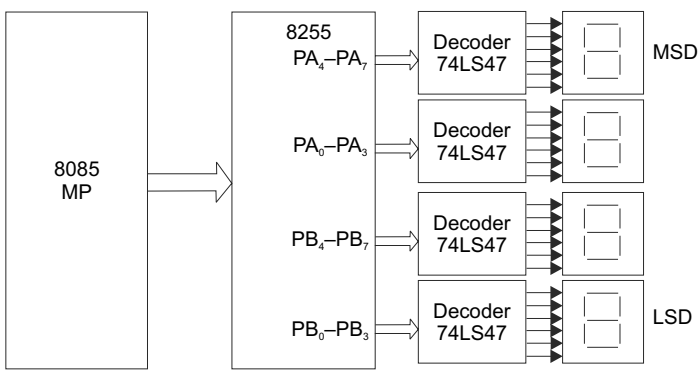


Fig. 18.38 Block diagram of four-digit display

Here, Port A is used to display most significant digits and Port B displays the least significant digits. After execution of the above program, 1234 will be displayed in the seven-segment display section.

To display more than four digits, more I/O ports and large number interfacing devices are required. To reduce the number of I/O ports and interfacing components, multiplexing technique is used to display large number of digits or letters or characters. Figure 18.39 shows the block diagram of a multi-digit display system. In this system, only one digit will display at a time. PA_0–PA_3 of Port A is connected to the decoder, and the decoder outputs are connected to seven-segment display. PB_0–PB_3 of Port B are fed to the multiplexer for selecting any one of the several seven-segment display units. Each seven-segment display unit will be turned ON and OFF in a sequence and this process will be repeated continuously. In this scheme, sixteen digits can be displayed simultaneously as the multiplexer has four inputs. Firstly, one of the sixteen seven-segment display units will be selected through the multiplexer and afterward the desired segments of the seven segments of the LED are to be turned ON to display any number. The same process may be repeated for the second, third and other seven-segment LEDs. The process can be repeated in cyclic order with a minimum time delay.

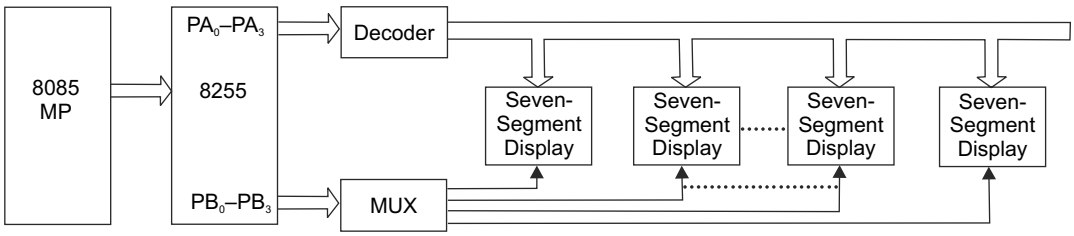


Fig. 18.39 Block diagram of multi-digit display using multiplexer

Review Questions

18.1 Define ADC. What are the types of ADC? Write some applications of ADCs.

18.2 Explain counting-type ADC with a suitable diagram. What are the limitations of this converter? How can you improve the performance of ADC.

18.3 Explain successive approximation type ADC. Compare dual-slope ADC and successive approximation ADC.

18.4 Define resolution. What is the resolution of 12-bit successive approximation ADC?

18.5 What is DAC? Write some applications of DACs.

18.6 Draw *N*-bit binary weight DAC and explain its operation. What are the disadvantages of binary weight DAC? What is the difference between binary weight DAC and R–2R ladder DAC.

18.7 Justify the following statements:

(i) *N*-bit successive approximation ADC requires only *N* clock pulses for complete conversion

(ii) Successive approximation ADC is faster than counting type ADC

(iii) Quantization error is $\pm$½ LSB

(iv) *N*-bit flash comparator requires 2^N–1 comparators.

18.8 Interface an A/D converter to 8085 and write a program to convert the analog input to digital.

18.9 Interface a D/A converter to 8085 and write a program to convert the digital input to analog output.

18.10 Draw the interfacing circuit of a seven-segement display with its decoder to the 8085 microprocessor.

18.11 Explain principles of multidigit display. Write assembly-language program in 8085 for

(a) Single-digit display (b) two-digit display

(c) four-digit display

18.12 Draw the ADC 0808 interfacing with 8051 microcontroller using 8255. Write assembly-language program in 8051 for ADC interfacing.

18.13 Draw the keyboard interfacing with the 8085 microprocessor. Write the assembly-language program in 8085 for reading a pressed key.

18.14 Explain LCD interfacing with 8051 microprocessor. Write a assembly language program in 8051 to display 'THANK YOU' on the LCD display screen.

18.15 Draw the ADC 0808 Interfacing with 8086 using 8255 PPI and explain the circuit operation briefly. Write an assembly-language program in 8086 for analog-to-digital conversion.

18.16 Design the ADC 0808 interfacing with 8051 microcontroller using 8255 PPI and explain the circuit operation briefly. Write an assembly-language program in 8051 for analog-to-digital conversion.

18.17 Draw the schematic block diagram for DAC 0808 interfacing with 8086 using 8255 PPI and discuss the circuit operation briefly. Write an assembly-language program in 8086 for digital-to-analog conversion.

18.18 Design the DAC 0808 interfacing with 8051 microcontroller using 8255 PPI and explain the circuit operation briefly. Write an assembly-language program in 8051 for digital-to-analog conversion.

18.19 Explain the keyboard interfacing with 8085 microprocessor. Write an assembly-language program for the keyboard interfacing With 8085.

18.20 Draw the keyboard interfacing with 8086 microprocessor and explain briefly. Write an assembly-language program for the keyboard interfacing with 8086.

18.21 Design the keyboard interfacing with 8051 microcontroller and write an assembly-language program for the keyboard interfacing with 8051.

18.22 Design the LCD interfacing with the 8051 microcontroller and write an assembly-language program for the LCD display.

18.23 Give the hardware and software to interface, one seven-segment display with 8085 *μp* whose address is FC23H.

18.24 Which addressing mode is used in the scheme of Q. 18.23? What change is required if the address of the display is FCH?

Multiple-Choice Questions

18.1 The analog voltage corresponding to the LSB of 12-bit A/D converter is

(a) $V/(2^{12}-1)$ (b) $V/(2^{12}+1)$ (c) $V/2^{12}$ (d) none of these

18.2 The resolution of an 8-bit D/A converter with a full-scale output of 10 V is

(a) $\frac{10}{2^8+1}$ (b) $\frac{10}{2^8-1}$ (c) $\frac{10}{2^8}$ (d) none of these

18.3 A digital instrument is used to measure analog voltage and display it in 7–segment display devices. The instrument has
(a) an ADC at the input and a DAC at the output
(b) an ADC at the input
(c) a DAC at the input
(d) an ADC at the output

18.4 Resolution of an N DAC is
(a) full scale value/2^N (b) full scale value/(2^N-1)
(c) full scale value/(2^N-1) (d) none of these

18.5 The resolution of a D/A converter is 0.4 per cent of full scale range. It is
(a) an 8-bit converter (b) a 10-bit converter
(c) a 12-bit converter (d) a 16-bit converter

18.6 A D/A converter's full scale output voltage is 10 V and its accuracy is +0.4%. The maximum error of DAC will be
(a) 20 mV (b) 30 mV (c) 40 mV (d) none of these

18.7 The speed of conversion is maximum in
(a) successive approximation ADC (b) flash ADC
(c) single-slope serial ADC (d) dual-slope ADC

18.8 In an N-bit flash converter, the number of comparators needed is
(a) 2^N-1 (b) 2^N (c) 2^N+1 (d) none of these

18.9 The N-bit successive approximation ADC requires
(a) 2^N-1 clock pulses (b) 2^N clock pulses
(c) N clock pulses (d) none of these

18.10 A 12-bit A/D converter has the input voltage signal from 0 V to +10 V. The voltage equivalent to 1 LSB will be
(a) 0 (b) 1.2 mV (c) 2.4 mV (d) 0.833 V

18.11 An 8-bit A/D converter has a resolution of
(a) $\frac{1}{2^4}$ (b) $\frac{1}{2^8}$ (c) $\frac{1}{2^2}$ (d) $\frac{1}{2^{16}}$

Answers to Multiple-Choice Questions

18.1 (a)	18.2 (b)	18.3 (d)	18.4 (c)
18.5 (a)	18.6 (c)	18.7 (b)	18.8 (a)
18.9 (c)	18.10 (c)	18.11 (b)	

CHAPTER

19

Introduction to PIC Microcontroller (16F877)

19.1 INTRODUCTION

The Peripheral Interface Controller (PIC) family of microcontrollers was developed by Microchip in 1985. PIC microcontrollers are based on Harvard architecture and Reduced Instruction Set (RISC). During 1997, 8-bit microcontrollers were introduced by Atmel, based on reduced instruction set. The 8-bit PIC microcontrollers such as PIC16CXX, PIC 16F87X, PIC17CXX were also developed by microchip and manufactured using CMOS Technology. These microcontrollers are extensively used in industry due to their very good performance, low cost and small size. The most commonly used PIC microcontrollers are PIC16C54, PIC16C55, PIC16C56, PIC16C57, PIC16C71, PIC17C42A, PIC17C43, PIC17C44 and PIC17C752 and their features are illustrated in Table 19.1. The comparative study of PIC 16F873, PIC 16F874, PIC 16F876 and PIC 16F 877 are given in Table 19.2.

Table 19.1 Comparative studies of salient features of PIC16C54, PIC16C55, PIC16C56, PIC16C57, PIC16C71, PIC17C42A, PIC17C43, PIC17C44 and PIC17C752 microcontrollers

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

Cont.

				Count			
PIC17C42A	2K bytes	232 bytes	58 single-word instructions	40	33	4	-
PIC17C43	4K bytes	454 bytes	58 single-word instructions	40	33	4	-
PIC17C44	8K bytes	454 bytes	58 single-word instructions	40	33	4	-
PIC17C752	8K×16 bytes	678 bytes	58 single-word instructions	40	33	4	12-channels 10-bit ADC

Table 19.2 The Comparative study of PIC 16F873, PIC 16F874, PIC 16F876 and PIC 16F877

Name	**PIC 16F873**	**PIC 16F874**	**PIC 16F876**	**PIC 16F877**
Program Memory	4k	4k	4k	4k
Data Memory (bytes)	192	192	368	368
EEPROM Data Memory (bytes)	128	128	256	256
I/O ports	3 (A, B, C)	5 (A, B, C, D, E)	3 (A, B, C)	5 (A, B, C, D, E)
Timers	3	3	3	3
Interrupts	13	14	13	14
10 bit Analog to Digital Module	5 input channels	8 input channels	5 input channels	8 input channels
Instruction Set	35 instructions	35 instructions	35 instructions	35 instructions
Capture/Compare/PWM Modules	2	2	2	2

19.2 FEATURES OF PIC16F877 MICROCONTROLLER

The PIC16F877 microcontroller is the member of PIC 16F87X family of 8-bit microcontrollers. This IC is manufactured with CMOS technology and available in 40-pin DIP. The features of PIC16F877 microcontrollers are given below:

- High-performance RISC CPU
- Only 35 single-word instructions are available for these microcontrollers
- All single-cycle instructions, except for program branches which are two cycles
- *Operating speed*: 20 MHz dc clock input and 200 ns dc instruction cycle
- 8K × 14 words of FLASH program memory, 368 × 8 bytes of data memory (RAM),
- 256 × 8 bytes of EEPROM data memory
- 14 Interrupt sources capability
- 8-bits Parallel Slave Port (PSP)
- Direct, indirect and relative addressing modes
- Power-On Reset (POR)
- Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own on-chip *RC* oscillator for reliable operation
- 3 timers: Timer 0, Timer 1 and Timer 2
  - *Timer 0*: 8-bit timer/counter with 8-bit prescaler

  - *Timer 1*: 16-bit timer/counter with prescaler, can be incremented during sleep via external crystal/clock
  - Timer 2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler
- Programmable code-protection
- Power-saving SLEEP mode
- Selectable oscillator options
- Low-power, high-speed CMOS FLASH/EEPROM technology
- Fully static design
- In-circuit serial programming capability
- In-circuit debugging
- Processor read/write access to program memory
- The wide operating voltage range from 2.0 V to 5.5 V
- High sink/source current, about 25 mA
- Operate in commercial as well as industrial temperature ranges
- Low-power consumption
- Two Capture, Compare, PWM modules
  - Capture is 16-bit, maximum resolution is 12.5 ns
  - Compare is 16-bit, maximum resolution is 200 ns
  - PWM maximum resolution is 10-bit
- 10-bit multi-channel analog-to-digital converter
- Synchronous Serial Port (SSP) with Master Mode and Master/Slave
- Universal Synchronous Asynchronous Receiver Transmitter (USART) with 9-bit address detection
- Parallel Slave Port (PSP) 8-bits wide, with external $\overline{RD}$, $\overline{WR}$ and $\overline{CS}$ CS control signals

19.3 PIN DIAGRAM AND ARCHITECTURE OF PIC16F877 MICROCONTROLLER

The schematic pin diagram of the 40-pin DIP PIC 16F877 is depicted in Fig.19.1 and the detailed pin diagram of DIP PIC 16F877 is shown in Fig.19.2. There are five ports such as Port A, Port B, Port C, Port D and Port E. Port A has 6 I/O pins which are used as A/D converter inputs. Port E is also used as A/D converter inputs and this port has 3 I/O pins. Port B has 8 I/O pins and it can be used as external interrupt sources. Port C has 8 I/O pins and used as serial ports and timer I/O. Port D has 8 I/O pins which are used as parallel slave port. Table 19.3 shows the applications of different ports. The function of each pin of PIC 16F877 is given in Table 19.4. The basic architecture of PICF877 is depicted in Fig. 19.3.

Table 19.3 Applications of different ports

Port Name	No. of I/O Pins	Use of Ports
Port A	6	A/D converter inputs
Port B	8	External interrupt sources
Port C	8	Serial ports, Timer I/O
Port D	8	Parallel slave port
Port E	3	A/D converter inputs

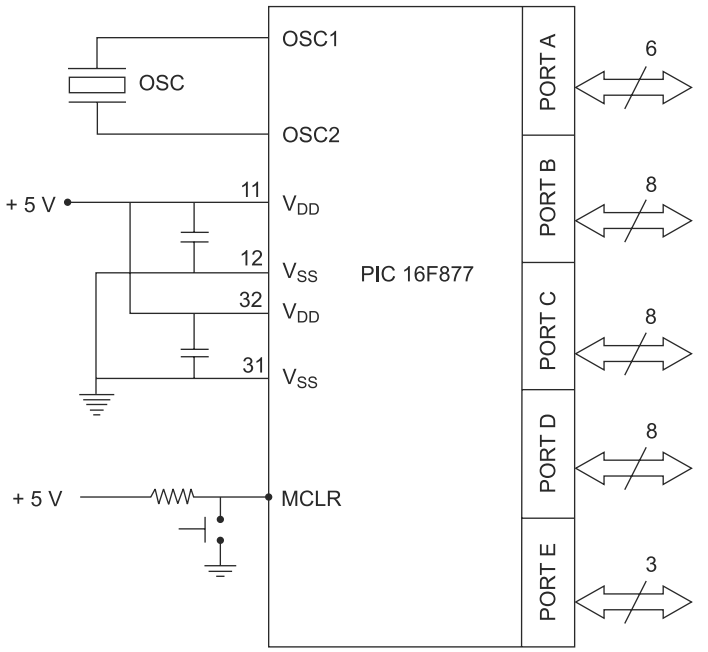


Fig. 19.1 40 pin DIP PIC 16F877

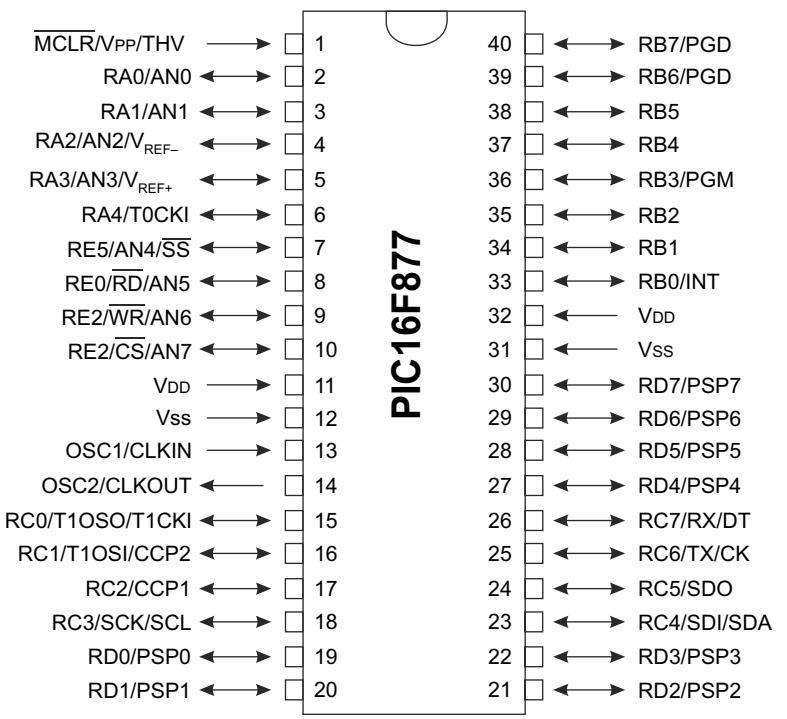


Fig. 19.2 Pin diagram of PIC16F877

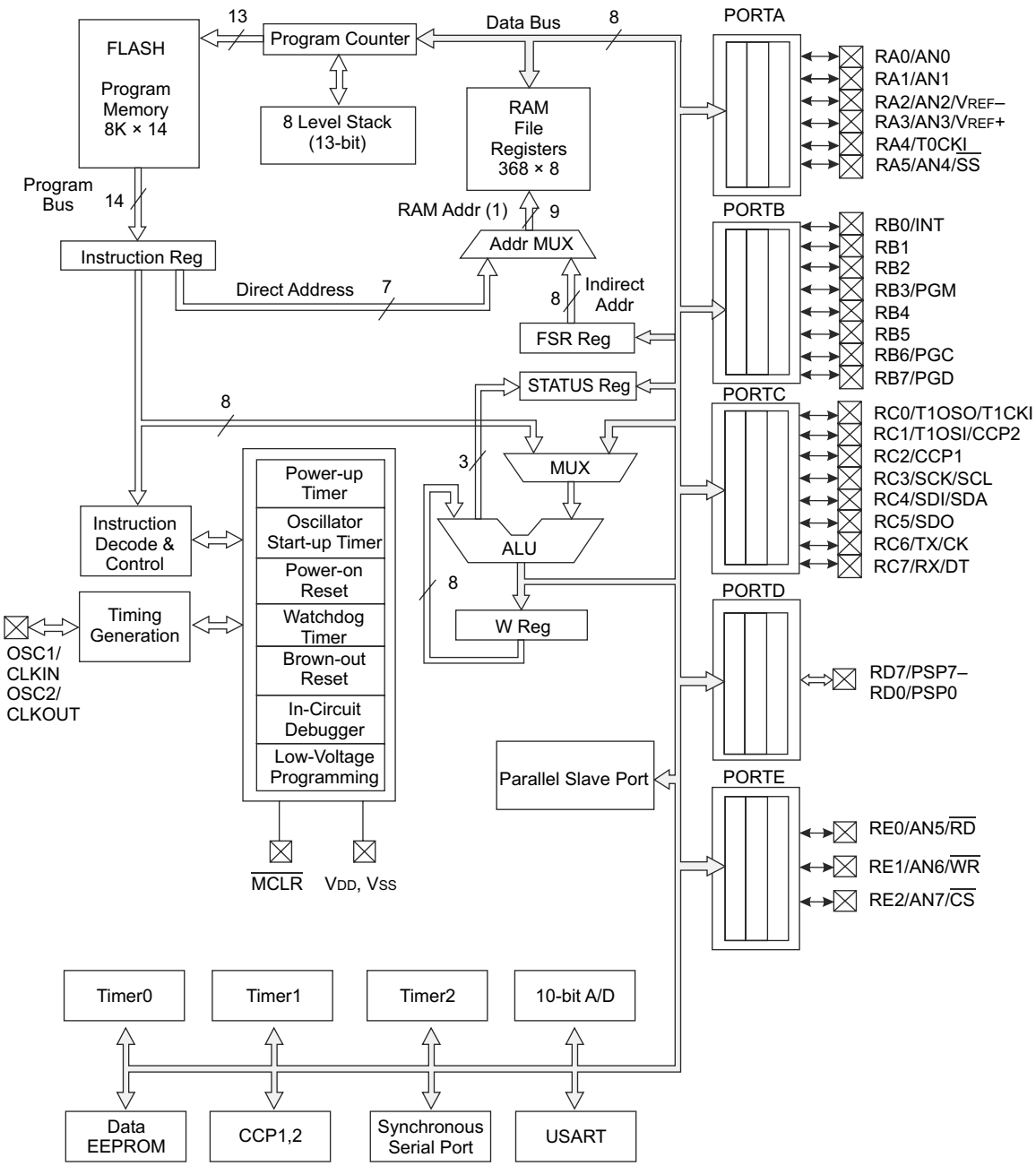


Fig. 19.3 Architecture of PICF877

Table 19.4 Pin function of PIC 16F877

Pin Name	Pin No.	I/O/P Type	Function
$\overline{MCLR}$/VPP	1	I/P	Master clear (reset) input or programming voltage input. This pin is an active low reset to the device.
Port A is a bidirectional I/O port : Pin 2 to Pin 7			
RA0/AN0	2	I/O	RA0 can be analog input-0.
RA1/AN1	3	I/O	RA1 can be analog input-1.
RA2/AN2/VREF–	4	I/O	RA2 can be analog input-2 or negative analog reference voltage.
RA3/AN3/VREF+	5	I/O	RA3 can be analog input-3 or positive analog reference voltage.
RA4/T0CKI	6	I/O	RA4 can be the clock input to the Timer 0 (timer/Counter). Output is open drain type.
RA5/$\overline{SS}$/AN4	7	I/O	RA5 can be analog input-4 or the slave select for the synchronous serial port.
Port E is a bidirectional I/O port : Pin 8 to Pin 10			
RE0/ $\overline{RD}$/AN5	8	I/O	RE0 can be read control for the parallel slave port, or analog input-5.
RE1/$\overline{WR}$/AN6	9	I/O	RE1 can be write control for the parallel slave port, or analog input-6.
RE2/$\overline{CS}$/AN7	10	I/O	RE2 can be select control for the parallel slave port, or analog input-7.
VDD	11, 32	P	Positive supply for logic and I/O pins.
VSS	12, 31	P	Ground reference for logic and I/O pins.
OSC1/CLKIN	13	I	Oscillator crystal input or external clock source input.
OSC2/CLKOUT	14	O	Oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode.
Port C is a bidirectional I/O port : Pin 15 to Pin 18 and Pin 23 to Pin 26			
RC0/T1OSO/T1CKI	15	I/O	RC0 can be the Timer 1 oscillator output or a Timer 1 clock input.
RC1/T1OSI/CCP2	16	I/O	RC1 can be the Timer 1 oscillator input or Capture 2 input/ Compare 2 output/PWM-2 output.
RC2/CCP1	17	I/O	RC2 can be the Capture 1 input/Compare 1 output/PWM-1 output.
RC3/SCK/SCL	18	I/O	RC3 can be the synchronous serial clock input or output for both Master and master–slave modes.
RC4/SDI/SDA	23	I/O	RC4 can be the SPI data in (SPI mode) or data I/O (I^2C mode).
RC5/SDO	24	I/O	RC5 can be the SPI data out (SPI mode).
RC6/TX/CK	25	I/O	RC6 can also be the USART asynchronous transmit or synchronous clock.
RC7/RX/DT	26	I/O	RC7 can be the USART asynchronous receive or synchronous data.
Port D : Pin 19, 20, 21, 22, 27, 28, 29, 30			
RD0/PSP0	19	I/O	Port D is a bi-directional I/O port or parallel slave port when interfacing to a microprocessor bus.

Count

Count			
RD1/PSP1	20	I/O	
RD2/PSP2	21	I/O	
RD3/PSP3	22	I/O	
RD4/PSP4	27	I/O	
RD5/PSP5	28	I/O	
RD6/PSP6	29	I/O	
RD7/PSP7	30	I/O	
Port B : Pin 33 to Pin 40			
Port B is a bidirectional I/O port. Port B can be software programmable.			
RB0/INT	33	I/O	RB0 can be the external interrupt pin.
RB1	34	I/O	
RB2	35	I/O	
RB3/PGM	36	I/O	RB3 can be the low-voltage programming input.
RB4	37	I/O	Interrupt on change pin.
RB5	38	I/O	Interrupt on change pin.
RB6/PGC	39	I/O	Interrupt on change pin / in-circuit debugger pin. Serial programming clock.
RB7/PGD	40	I/O	Interrupt on change pin / in-circuit debugger pin, serial programming data.

19.4 MEMORY ORGANIZATION OF PIC 16F877

The PIC 16F877 microcontroller has separate program memory organization and data memory organization which are given below:

Program Memory Organization PIC 16F877 has 8K × 14 program memory space from 0000H to 1FFFH. This device has a 13-bit program counter. The reset vector is at 0000H and the interrupt vector at 0004H. Figure 19.4 shows the program memory map and stack of PIC 16F877.

Data Memory Organization The data memory is partitioned into four banks which contain general purpose registers and special-function registers. The bits RP1 and RP0 are used as the bank select bits. Table 19.5 shows the bank selection based on RP1 and RP0.

Table 19.5 Bank selection based on RP1 and RP0

RP1	RP0	Bank
0	0	0
0	1	1
1	0	2
1	1	3

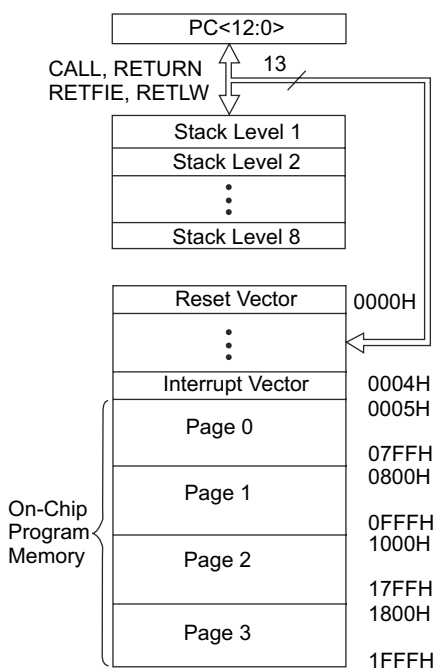


Fig. 19.4 Program memory organization

Each bank can be extended up to 7FH or 128 bytes. The lower addresses of each bank are reserved for the special-function registers. Above the special-function registers, general-purpose registers are placed and these registers are implemented as static RAM. Figure 19.5 shows the register file map of PIC 16F877.

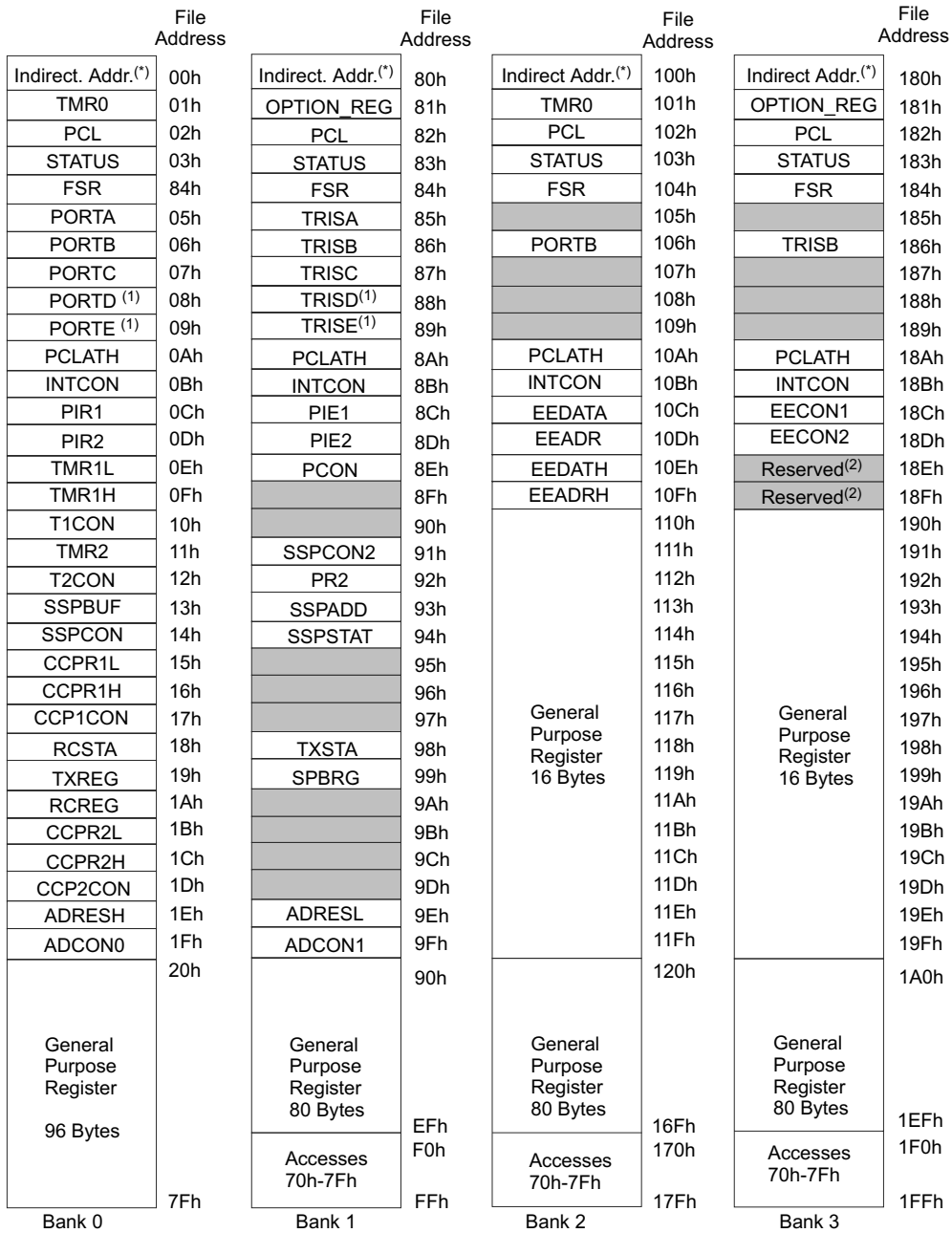


Fig. 19.5 Register file map of PIC 16F877

19.5 CPU REGISTERS

The CPU registers of PIC 16F877 are

(i) W register, (ii) Status Register, (iii) FSR (indirect data memory address pointer), (iv) INDF, and (v) PC (Program counter).

The above registers are used in execution of the instruction set.

19.5.1 W Register

The W register is known as working register. This register is used by instructions as the source of an operand. After execution of this instruction, the result can be stored in the W register. Therefore, it acts as an accumulator of other microcontrollers.

19.5.2 Status Register

The status register contains the arithmetic status of the arithmetic logic unit (ALU), the reset status and the bank select bits for data memory. Figure 19.6 shows the status register.

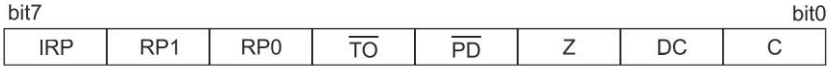


Fig. 19.6 Status register

Bit 0: C : Carry/borrow bit After execution of ADDWF, ADDLW, SUBLW, SUBWF instructions,
C = 1 if a carry-out from the most significant bit of the result occurred and
C = 0 if no carry-out from the most significant bit of the result occurred

Bit 1: DC: Digit carry/borrow bit After execution of ADDWF, ADDLW, SUBLW, SUBWF instructions,
DC = 1 when a carry-out from the 4^{th} lower order bit of the result occurred and
DC = 0 if no carry-out from the 4^{th} lower order bit of the result

Bit 2: Z: Zero bit If the result of an arithmetic or logic operation is zero, Z = 1
If result of an arithmetic or logic operation is not zero, Z = 0

Bit 3: $\overline{PD}$: Power-down bit $\overline{PD}$ will be 1 just after power-up or by the CLRWDT instruction, and
$\overline{PD}$ will be 0 by execution of the SLEEP instruction.

Bit 4: $\overline{TO}$: Time-out bit $\overline{TO}$ will be 1 after power-up, and execution of CLRWDT instruction or SLEEP instruction, and
$\overline{TO}$ will be 0 when a WDT time-out occurred.

bit 6–5: RP1:RP0: Register Bank Select bits. These bits are used for direct addressing.
When RP1:RP0 = 00, Bank 0 is selected (00H–7FH)
When RP1:RP0 = 01, Bank 1 is selected (80H–FFH)
When RP1:RP0 = 10, Bank 2 is selected (100H–17FH)
When RP1:RP0 = 11, Bank 3 is selected (180H–1FFH)
It is clear from the above that each bank has 128 bytes.

bit 7: IRP: Register Bank Select bit. This bit is used for indirect addressing.

When IRP will be 1, Bank 2, 3 will be selected (100H–1FFH)

When IRP will be 0, Bank 0, 1 will be selected (00H–FFH)

19.5.3 Program Counter (PC)

The program counter (PC) is a 13-bit wide register. The low byte comes from the PCL register, and the PCL register is a readable and writeable register. The upper bits (PC<12:8>) are not readable, but are indirectly writeable through the PCLATH register. Just after any reset operation, the upper bits of the PC will be cleared. Figure 19.7 shows the two different conditions for the loading of the PC. In Fig. 19.7(a), the PC is loaded during a CALL or GOTO instruction (PCLATH<4:3> ® PCH); and in Fig.19.7(b), the PC is loaded on a write to PCL (PCLATH<4:0> ® PCH).

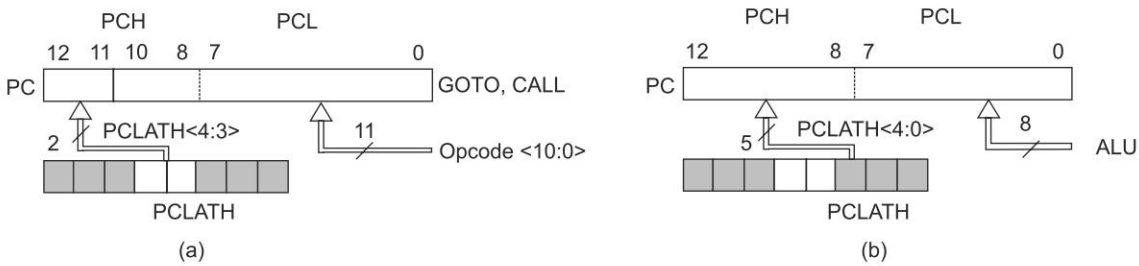


Fig. 19.7 Loading of PC

19.6 ADDRESSING MODES

Usually, only two addressing modes, namely direct addressing mode and indirect addressing mode, are used in the PIC16F877 microcontroller.

Direct Addressing Mode In the direct addressing mode, 7 bits (0 to 6) of the instruction can identify the register file address and the 8th bit of the register comes from the register bank select bit RP0.

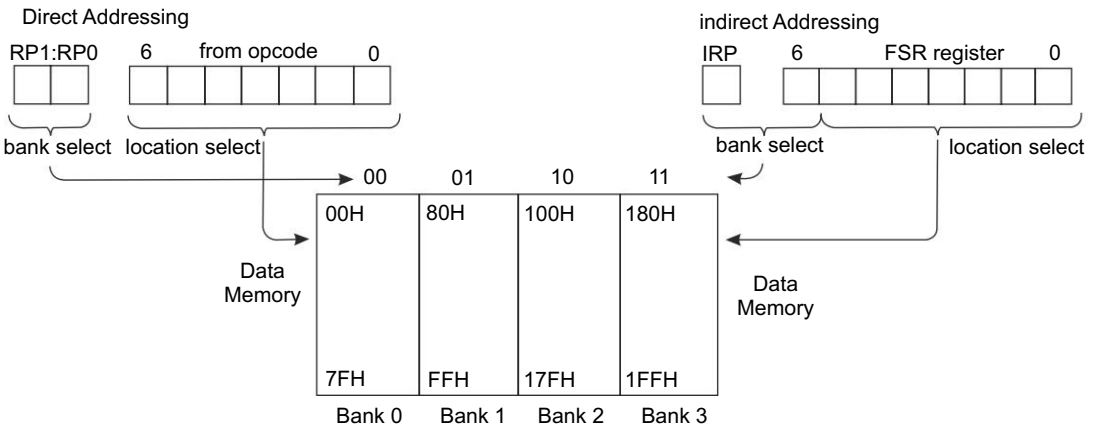


Fig. 19.8 Addressing modes of PIC16F877 microcontroller

Indirect Addressing Mode In the indirect addressing mode, the full 8 bits (0 to 7) register file address is written into FSR register. The register bank will be selected through the 7^{th} bit of FSR register and the content of IRP. Figure 19.8 shows the direct addressing and indirect addressing of PIC16F877 microcontroller.

19.7 INSTRUCTION SET OF PIC 16F877

Each instruction of 16F877 is a 14-bit word which is divided into an opcode of one or more operands. An opcode specifies the instruction type and operands specify the operation of the instruction. The instruction set of PIC16F877 is classified into three basic categories such as

(i) Byte-oriented instructions
(ii) Bit-oriented instructions, and
(iii) Literal and control instructions

Byte-oriented instructions In byte-oriented instructions, 'f' represents a file register designator and 'd' represents a destination designator. The file register designator specifies which file register is to be used by the instruction. The destination designator specifies where the result of the operation is to be stored. When 'd' is zero, the result is stored in the W register. If 'd' is one, the result is placed in the file register specified in the instruction. The byte-oriented file register operations are depicted in Fig.19.9

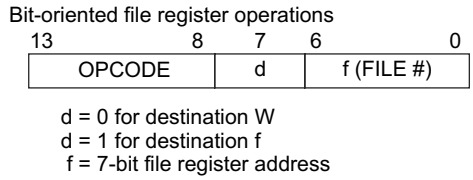


Fig. 19.9 Byte-oriented file register operations

Bit-oriented instructions In bit-oriented instructions, 'b' represents a bit field designator which selects the number of the bit affected by the operation, but 'f' represents the number of the file in which the bit is located. The bit-oriented file register operations are depicted in Fig.19.10.

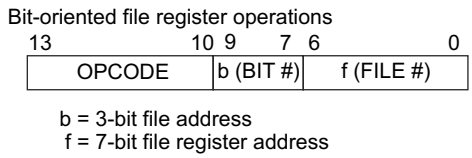


Fig. 19.10 Bit-oriented file register operations

Literal and control instructions In literal and control operations, 'k' represents an eight- or eleven- bit constant or literal value. The literal and control operations are depicted in Fig. 19.11.

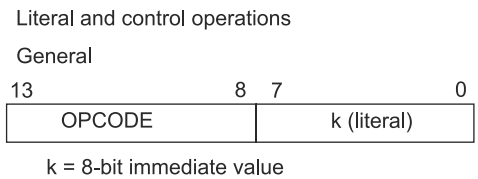


Fig. 19.11 Literal and control operations

All instructions are executed within one single instruction cycle, except when a conditional test is true or the program counter is changed as a result of an instruction. In case of conditional instructions, the execution takes two instruction cycles where the second cycle is executed as an NOP. Actually, one instruction cycle consists of four oscillator periods. When the PIC microcontroller oscillator frequency is 4 MHz, the normal instruction execution time is 1μs. When a conditional test is true or the program counter is changed as a result

of an instruction, the instruction execution time is 2 μs. Table 19.6 shows the opcode field descriptions and a set of instructions for PIC16F877 is given in Table 19.7.

Table 19.6 The opcode field descriptions

Field	Function
f	Register file address (0 × 00 to 00 × 7F)
w	Working register/accumulator
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
x	Don't care location (= 0 or 1). The assembler will generate code with x = 0
d	Destination select; d = 0: store result in W, d = 1: store result in file register f.
PC	Program counter
TO	Time-out bit
PD	Power-down bit

Table 19.7(a) Instructions: Byte-oriented file register operations

Mnemonics, Operands	Description	Cycles
ADDWF f,d	Add W and f	1
ANDWF f,d	AND W with f	1
CLRF f	Clear f	1
CLRW –	Clear W	1
COMF f, d	Complement f	1
DECF f, d	Decrement f	1
DECFSZ f, d	Decrement f, Skip if 0	1(2)
INCF f, d	Increment f	1
INCFSZ f, d	Increment f, Skip if 0	1(2)
IORWF f, d	Inclusive OR W with f	1
MOVF f, d	Move f	1
MOVWF f	Move W to f	1
NOP –	No operation	1
RLF f, d	Rotate left f through carry	1
RRF f, d	Rotate right f through carry	1
SUBWF f, d	Subtract W from f	1
SWAPF f, d	Swap nibbles in f	1
XORWF f, d	Exclusive OR W with f	1

Table 19.7(b) Instructions : Bit-oriented file register operations

Mnemonics, Operands	Description	Cycles
BCF f, b	Bit Clear f	1
BSF f, b	Bit set f	1
BTFSC f, b	Bit test f, skip if clear	1(2)
BTFSS f, b	Bit test f, skip if set	1(2)

Table 19.7(c) Instructions : Literal and control operations

Mnemonics, Operands	Description	Cycles
ADDLW k	Add literal and W	1
ANDLW k	AND literal with W	1
CALL k	Call subroutine	2
CLRWDT -	Clear Watchdog Timer	1
GOTO k	Go to address	2
IORLW k	Inclusive OR literal with W	1
MOVLW k	Move literal to W	1
RETFIE -	Return from interrupt	2
RETLW k	Return with literal in W	2
RETURN -	Return from subroutine	2
SLEEP -	Go into standby mode	1
SUBLW k	Subtract W from literal	1
XORLW k	Exclusive OR literal with W	1

19.8 APPLICATIONS OF PIC 16F877

The PIC microcontrollers are commonly used in the following applications:

Household Appliances: Washing machine, Remote control, light control, video games, TV tuner, intercom and sewing machine

Office Equipment: Telephones, fax machines, printers and security systems

Instruments: Digital thermometers, level meters, and multi-meters

Peripheral Devices: Keyboard controllers, modems and plotters

Motor Control: Speed control of ac and dc motors, position control using servomotors and stepper motors.

Industry: Process control systems and automobile applications.

Review Questions

19.1 What is a PIC? Give a list of the PIC microcontrollers.
19.2 Write the different features of the PIC 16F877 microcontroller.
19.3 How many instructions are used in the PIC 16F877 microcontroller?
19.4 Draw the schematic pin diagram of the PIC 16F877 microcontroller and explain pin function.
19.5 Discuss the memory organization of the PIC 16F877 microcontroller.
19.6 Give a list of applications of PIC 16F877.

Multiple-Choice Questions

19.1 The PIC 16F877 microcontroller has _____ instructions.
(a) 25 (b) 30 (c) 35 (d) 40

19.2 The PIC 16F877 microcontroller operates in _____ addressing modes.
(a) 2 (b) 3 (c) 4 (d) 5

19.3 The PIC 16F877 microcontroller has _____program memory.
(a) 1K (b) 2K (c) 4K (d) 8K

19.4 The PIC 16F877 microcontroller has _____data memory.
(a) 128 bytes (b) 192 bytes (c) 256 bytes (d) 368 bytes

Answers to Multiple-Choice Questions

19.1 (c) 19.2 (a) 19.3 (d) 19.4 (d)

APPENDIX A

8085 Instruction Set

MNEMONIC		HEX CODE	MNEMONIC		HEX CODE	MNEMONIC		HEX CODE
MOV	A.A	7F	MOV	H,B	60	POP	B	C1
MOV	A.B	78	MOV	H,C	61	POP	D	D1
MOV	A.C	79	MOV	H,D	62	POP	H	E1
MOV	A,D	7A	MOV	H,E	63	POP	PSW	F1
MOV	A,E	7B	MOV	H,H	64	PUSH	B	C5
MOV	A,H	7C	MOV	H,L	65	PUSH	D	D5
MOV	A,L	7D	MOV	H,M	66	PUSH	H	E5
MOV	A,M	7E	MOV	L,A	6F	PUSH	PSW	F5
MOV	B,A	47	MOV	L,B	68	PCHL		E9
MOV	B,B	40	MOV	L,C	69	RAL		17
MOV	B,C	41	MOV	L,D	6A	RAR		1F
MOV	B,D	42	MOV	L,E	6B	RC		D8
MOV	B,E	43	MOV	L,H	6C	RET		C9
MOV	B,H	44	MOV	L,L	6D	RIM		20
MOV	B,L	45	MOV	L,M	6E	RLC		07
MOV	B,M	46	MOV	M,A	77	RM		F8
MOV	C,A	4F	MOV	M,B	70	RNC		D0
MOV	C,B	48	MOV	M,C	71	RNZ		C0
MOV	C,C	49	MOV	M,D	72	RP		F0
MOV	C,D	4A	MOV	M,E	73	RPE		E8
MOV	C,E	4B	MOV	M,H	74	RPO		E0
MOV	C,H	4C	MOV	M,L	75	RRC		0F
MOV	C,L	4D	MVI	A,8-bit	3E	RST	0	C7
MOV	C,M	4E	MVI	B,8-bit	06	RST	1	CF
MOV	D,A	57	MVI	C,8-bit	0E	RST	2	D7
MOV	D,B	50	MVI	D,8-bit	16	RST	3	DF
MOV	D,C	51	MVI	E,8-bit	1E	RST	4	E7
MOV	D,D	52	MVI	H,8-bit	26	RST	5	EF

MNEMONIC		HEX CODE	MNEMONIC		HEX CODE	MNEMONIC		HEX CODE
MOV	D,E	53	MVI	L,8-bit	2E	RST	6	F7
MOV	D,H	54	MVI	M,8-bit	36	RST	7	FF
MOV	D,L	55	NOP		00	RZ		C8
MOV	D,M	56	ORA	A	B7	SBB	A	9F
MOV	E,A	5F	ORA	B	B0	SBB	B	98
MOV	E,B	58	ORA	C	Bl	SBB	C	99
MOV	E,C	59	ORA	D	B2	SBB	D	9A
MOV	E,D	5A	ORA	E	B3	SBB	E	9B
MOV	E,E	5B	ORA	H	B4	SBB	H	9C
MOV	E,H	5C	ORA	L	B5	SBB	L	9D
MOV	E,L	5D	ORA	M	B6	SBB	M	9E
MOV	E,M	5E	ORI	8-bit	F6	SBI	8-Bit	DE
MOV	H,A	67	OUT	8-bit	D3	SHLD	16-Bit	22

8085 INSTRUCTION SET

MNEMONICHEX		CODE	MNEMONIC		HEX CODE	MNEMONICHEX		CODE
SIM		30	ACI	8-Bit	CE	CP	16-Bit	F4
SPHL		F9	ADC	A	8F	CPE	16-Bit	EC
STA	16-Bit	32	ADC	B	88	CPI	8-Bit	FE
STAX	B	02	ADC	C	89	CPO	16-Bit	E4
STAX	D	12	ADC	D	8A	CZ	16-Bit	CC
STC		37	ADC	E	8B	DAA		27
SUB	A	97	ADC	H	8C	DAD	B	09
SUB	B	90	ADC	L	8D	DAD	D	19
SUB	C	91	ADC	M	8E	DAD	H	29
SUB	D	92	ADD	A	87	DAD	SP	39
SUB	E	93	ADD	B	80	DCR	A	3D
SUB	H	94	ADD	C	81	DCR	B	05
SUB	L	95	ADD	D	82	DCR	C	0D
SUB	M	96	ADD	E	83	DCR	D	15
SUI	8-Bit	D6	ADD	H	84	DCR	E	1D
XCHG		EB	ADD	L	85	DCR	H	25
XRA	A	AF	ADD	M	86	DCR	L	2D
XRA	B	A8	ADI	8-Bit	C6	DCR	M	35
XRA	C	A9	ANA	A	A7	DCX	B	0B

XRA	D	AA	ANA	B	A0	DCX	D	1B
XRA	E	AB	ANA	C	A1	DCX	H	2B
XRA	H	AC	ANA	D	A2	DCX	SP	3B
XRA	L	AD	ANA	E	A3	DI		F3
XRA	M	AE	ANA	H	A4	EI		FB
xri	8-Bit	EE	ANA	L	A5	HLT		76
XTHL		E3	ANA	M	A6	IN	8-Bit	DB
			ANI	8-Bit	E6	INR	A	3C
			CALL	16-Bit	CD	INR	B	04
JNC	16-Bit	D2	CC	16-Bit	DC	INR	C	0C
JNZ	16-Bit.	C2	CM	16-Bit	FC	INR	D	14
JP	16-Bit	F2	CMA		2F	INR	E	1C
JPE	16-Bit	EA	CMC		3F	INR	H	24
JPO	16-Bit	E2	CMP	A	BF	INR	L	2C
JZ	16-Bit	CA	CMP	B	B8	INR	M	34
LDA	16-Bit	3A	CMP	C	B9	INX	B	03
LDAX	B	0A	CMP	D	BA	I NX	D	13
LDAX	D	1A	CMP	E	BB	INX	H	23
LHLD	16-Bit	2A	CMP	H	BC	INX	SP	33
LXI	B, 16-Bit	01	CMP	L	BD	JC	16-Bit	DA
LXI	D, 16-Bii	11	CMP	M	BE	JM	16-Bit	FA
LXI	H, 16-Bit	21	CNC	16-Bit	D4	JMP	16-Bit	C3
LXI	SP. 16-Bit	31	CNZ	16-Bit	C4			

Model Question Paper 1

Full Marks: 70 **Duration: 3 Hours**

GROUP–A
(Multiple-Choice Questions)

1. Choose the appropriate answer for the following questions from the given options. (10×1=10)

(i) Data bus of a microprocessor is
(a) unidirectional (b) bi-directional
(c) unidirectional as well as bi-directional (d) none-of these

(ii) MOV A, C is executed by
(a) 1 machine cycle (b) 2 machine cycle (c) 3 machine cycle (d) 4 machine cycle

(iii) When CALL instruction is executed, the stack pointer register is
(a) decremented by two (b) incremented by two
(c) decremented by one (d) incremented by one

(iv) The PC contains 8452H and SP contains 88D6H. What will be the content of PC and SP following a CALL to subroutine at location 82AFH
(a) 82AF, 88D4 (b) 82AF, 8450 (c) 8450, 88D4 (d) 82AF, 8452

(v) Which is the highest priority interrupt?
(a) TRAP (b) RST 6.5 (c) RST 5.5 (d) RST 7.5

(vi) 8259 is a
(a) programmable interrupt controller (b) DMA controller
(c) programmable keyboard display interface (d) programmable counter

(vii) If A_0 and A_1 pins of 8255 are 00, which port will be selected?
(a) Port A (b) Port B (c) Port C (d) None of these

(viii) 8253 has
(a) 6 modes of operation (b) 5 modes of operation
(c) 4 modes of operation (d) 3 modes of operation

(ix) 8279 displays can operate in
(a) 8–8 bit character display-left entry only
(b) 16–6 bit character display-left entry only
(c) 8–8 bit character display-right entry only
(d) 8–8 bit character display-left and right entry and 16–16 bit character display-left and right entry

(x) The signals are used for DMA operation
(a) HRQ (b) HLD (c) HRQ and HLDA (d) none of these

(xi) The resolution of a 8-bit D/A converter with a full-scale output of 10 V is

(a) $\frac{10}{2^8 - 1}$ (b) $\frac{10}{2^8 + 1}$ (c) $\frac{10}{2^8}$ (d) None of these

(xii) 8086 has

(a) 16-bit data bus and 20-bit address bus (b) 8-bit data bus and 20 bit address bus
(c) 16-bit data bus and 16-bit address bus (d) 8-bit data bus and 16 bit address bus

(xiii) What is the addressing mode of instruction MOV AX, [BX+SI+06] ?

(a) index addressing (b) base addressing
(c) base index addressing (d) base index displacement addressing

(xiv) What is the output of DL after execution of the following instructions?

MOV DL, 36
AND DL, 0F

(a) DL = 06H (b) 60H (c) 36H (d) 0FH

(xv) The 80C51 microcontroller family has

(a) 32 pins for I/O (b) 24 pins for I/O (c) 16 pins for I/O (d) 8 pins for I/O

GROUP–B
(Short-Answer Type Questions)

Answer any three questions: (3 × 15 = 45)

2. Draw the schematic diagram to generate Read/Write control signals for memory and I/O devices in 8085-microprocessor.
3. Mention the different modes of operation of 8253 IC. Explain rate generator mode of 8253.
4. Write the functions of the following pins of 8259A.
   (i) T×D (ii) R×D (iii) T×RDY (iv) $C/\overline{D}$ (v) $\overline{DSR}$
5. Define interrupt. Discuss the different interrupts available in 8085 microprocessor.
6. Explain the functions of the following instructions:
   (i) LDA 7360H (ii) ADC D (iii) STA 8000H

GROUP–C
(Long-Answer Type Questions)

Answer any three questions: (3×15 = 45)

7. (a) A block of 32 bytes data is stored at the memory location starting from 8000H. Write a program to move this block to the memory location starting from 9000H.
   (b) Draw the timing diagram of I/O read cycle and explain briefly.
   (c) Draw the RIM instruction format and discuss with example. "A RIM instruction should be performed immediately after TRAP occurs" Why?
8. (a) What are the different operating modes of 8255? Explain any one operating mode of 8255.
   (b) Write control word in mode 0 operation for the following cases:
   (i) Port A = input port, Port B = output port, Port C = output port
   (ii) Port A = input port, Port B = output port, Port C_U = output port, Port CL = input port
   (c) Explain the importance of GATE signal in 8253. How is it used to control the operation of counters?
9. (a) Explain the DMA operation with a suitable diagram. Why DMA controlled data transfers faster? What are the building blocks of 8257?

(b) Draw an interface circuit of an A/D converter to 8085 and write a program to convert the analog input to digital.
(c) Show the timing diagram for execution of LDA 3050H instruction.

10. (a) What is instructions format? What are the types of instructions of 8086 microprocessors?
(b) Write instructions to perform the following operations.
(i) Copy the content of BX to a memory location in the data segment with offset 0234H.
(ii) Increment content of CX by 1.
(iii) Multiply AX with 16-bit data 2467H.
(iv) Rotate left the content of AL by two bits.
(c) Write an assembly language program for 2ms time delay. Assume the system clock time period is equal to 0.33 μsec.

11. Write short notes on the following:
(a) 8259 Interrupt Controller.
(b) Minimum/Maximum mode operation of 8086μP.
(c) Segment memory.
(d) Architecture of 8051 microcontroller.

Model Question Paper 2

Full Marks: 70 **Duration: 3 Hours**

GROUP–A
(Multiple-Choice Questions)

1. Choose the appropriate answer for the following questions from the given options. (10×1=10)

(i) The program counter(PC) in a microprocessor
- (a) keeps the address of the next instruction to be fetched.
- (b) counts the number of instructions being executed on the microprocessor.
- (c) counts the number of programs being executed on the microprocessor.
- (d) counts the number of interrupts handled by the microprocessor.

(ii) CALL 8000H is an instruction of
- (a) direct addressing mode (b) indirect addressing mode
- (c) register addressing mode (d) immediate addressing mode

(iii) OUT 02H is executed by
- (a) one machine cycle (b) two machine cycle
- (c) three machine cycle (d) four machine cycle

(iv) When the RET instruction is executed at the end of a subroutine
- (a) the memory address of the RET instruction is transferred to the program counter.
- (b) two data bytes stored in the top locations of the stack are transferred to stack pointer.
- (c) the data where the stack is initialised is transferred to the stack pointer.
- (d) two data bytes stored in the top two locations of the stack are transferred to the program counter.

(v) The number of address lines are required to access 2M byte of data from microprocessor
- (a)16-bit address lines (b) 8-bit address lines
- (c) 20-bit address lines (d) 12-bit address lines

(vi) RIM instruction is used to
- (a) enable RST 7.5, 6.5 and 5.5
- (b) disable RST 7.5, 6.5 and 5.5
- (c) enable or disable RST 7.5, 6.5 and 5.5
- (d) none of these

(vii) When port A is used as input, Port B and Port C are used as output, the control word of 8255 is
- (a) 80H (b) 90H (c) 85H (d) 86H

(viii) Which pin is used to control the output of counters 2 of 8253 in mode 2?
- (a) GATE 0 (b) GATE 1 (c) GATE 2 (d) GATE 3

(ix) The UART performs
- (a) a serial to parallel conversion (b) a parallel-to-serial conversion
- (c) control and monitoring functions (d) all

x) 8279 is known as
(a) DMA controller (b) programmable keyboard display interface
(c) counter (d) interrupt controller

xi) The resolution of a D/A converter is 0.4 percent of full scale range. It is a/an
(a) 8-bit converter (b) 10-bit converter
(c) 12-bit converter (d) 16-bit converter

xii) Physical memory of 8086 is (a) 1MB (b) 64 KB (c) 2 MB (d) 4 MB

xiii) What are the control signals of 8085 microprocessor are used to interface I/O devices.
(a) $IO/\overline{M}$, $\overline{RD}$, $\overline{WR}$ (b) $IO/\overline{M}$ (c) $\overline{RD}$ (d) $\overline{WR}$

xiv) Which of the following instructions is indirect addressing ?
(a) MOV A, R_0 (b) MOV A, 40H
(c) MOV R_7, #55 (d) MOV A,@R_0

xv) Which of the following signals indicates an 8-bit data transfer from odd address bank.
(a) $A_0 = 0$ & BHE = 0 (b) $A_0 = 1$ & BHE = 1
(c) $A_0 = 0$ & BHE = 1 (d) $A_0 = 1$ & BHE = 0

GROUP–B
(Short-Answer Type Questions)

Answer any three questions: (3 × 5 = 15)

2. Draw and explain the time multiplexing of $AD_0 – AD_7$.
3. What are the control signals are used for memory and I/O read and writes operations?
4. Distinguish between:
   (a) Vectored and non-vectored interrupt.
   (b) Maskable and non-maskable interrupt.
5. Write a program to use counter 2 of 8253 in Mode 5 operation.
6. What is serial data transfer? Write the difference between synchronous and asynchronous data transfer.

GROUP–C
(Long-Answer Type Questions)

Answer any three questions: (3 × 15 = 45)

7. (a) What is the difference between microprocessor and microcontroller? Describe how data can flow between microprocessor, memory and I/O devices.
   (b) Data byte 28H is stored in register B and data byte 97H is stored in the accumulator. Show the contents of registers B, C, and the accumulator after the execution of the following instructions:
   (a) MOV C, A (b) MOV A, B (c) ADD B
   (c) Calculate the square of the contents of memory location 9100H using look-up table and place the result in memory location 9501.
8. (a) Explain memory interfacing with 8085 microprocessor. Design a memory interfacing circuit to interface the following memory ICs
   i. 2K × 8-bit EPROM 2716. Assume starting address is 8000H.
   ii. 2K × 4-bit RAM 6116. Consider starting address is 9000H.

Write the memory map.

(b) Explain memory mapped I/O and I/O mapped I/O. Write the comparison between memory mapped I/O and I/O mapped I/O. What are the instructions available in memory mapped I/O and I/O mapped I/O scheme?

(c) Write an assembly language program to add 10 bytes of data in the DS segment starting from 2000H:3000H and store the result in 3000H:2000H location.

9. (a) What are the software interrupts of 8085 microprocessor? Mention interrupts instructions with their Hex code and vector address. How is the vector address for a software interrupt in determined?

(b) Write the control word format for I/O mode operation of 8255.

(c) Write an program to generate square wave using 8255.

10. (a) What will be the contents of the accumulator after the following instruction sequence is executed?

```
LXI H, C490H
XCHG
MVI A, 40H
ADD E
HLT
```

(b) If the system clock is 2MHz, find the time needed to execute the given instructions:

```
MVI A, 5AH
MVI B, A7H
ADD B
INR A
XRA A
HLT/RST 1
```

(c) If the carry flag is reset, specify the contents of the accumulator & CY flag after it executes the RAL instruction. Assume the content of the accumulator is C4H

11. Write short notes on the following:

(a) Interrupt operation/modes of 8259 PIC.

(b) Rate generator mode of 8253.

(c) Function of 8251 USART.

(d) DMA data transfer scheme.

Model Question Paper 3

Full Marks: 70 **Duration: 3 Hours**

GROUP–A
(Multiple-Choice Questions)

1. Choose the appropriate answer for the following questions from the given options. (10×1=10)

(i) If a microprocessor is capable of addressing 64K bytes of memory, its address bus width is
(a) 16 bits (b) 20 bits (c) 8 bits (d) none of these

(ii) A three-byte instruction should have
(a) Opcode and an operand (b) Opcode only
(c) Opcode and two operand (d) Operand only

(iii) SUB A instruction in 8085 microprocessor
(a) resets the carry and sign flag (b) sets the zero and parity flag
(c) sets the zero and carry flag (d) resets the zero and sign flag

(iv) Opcode is
(a) the part of the construction which tells the computer what operation to perform
(b) an auxiliary register that stores the data to be added or subtracted from the accumulator
(c) the register that receives the information from memory
(d) is the data which will be used in data manipulation of instruction

(v) To design a 4KB RAM with 1024 byte RAM ICs, how many ICs are required
(a) 4 (b) 8 (c) 2 (d) None of these

(vi) 8257 is a
(a) DMA controller (b) Programmable keyboard display interface
(c) Counter (d) Interrupt controller

(vii) The *N*-bit successive approximation ADC requires
(a) 2^N–1 clock pulses (b) 2^N clock pulses
(c) *N* clock pulses (d) None of these

(viii) The physical address when DS = 2345H and IP = 1000H is
(a) 23450H (b) 24450H (c) 12345H (d) 2345H

(ix) 2's complement instruction is
(a) NEG (b) NOT (c) CMP (d) CMC

(x) 8251 is a
(a) USART IC (b) counter
(c) interrupt controller (d) Programmable peripheral interface

(xi) Which of the following instruction is index addressing?
(a) MOVC A,@A + DPTR (b) MOVX @DPTR, A
(c) MOVX A,@DPTR (d) MOVX A,@R_0

(xii) What will be the output after execution of the following instructions?

MOV A, #55
ANL A, #67

(a) 54 (b) 45 (c) 55 (d) 67

(xiii) The 8051 microcontroller has

(a) 4K bytes of on-chip ROM (a) 8K bytes of on-chip ROM
(c) 16K bytes of on-chip ROM (d) 32K bytes of on-chip ROM

(xiv) The Bit set reset mode in 8255 is used with one of the following:

(a) Port A (b) Port B (c) Port C (d) None of these

(xv) In 8259, which is the lowest priority interrupt

(a) IR_0 (a) IR_3 (c) IR_4 (d) IR_7

GROUP–B
(Short-Answer Type Questions)

Answer any three questions. (3 × 15 = 45)

2. Mention the purpose of SID and SOD lines. Explain the functions of SIM and RIM instructions in 8088 μP.
3. Define stack. Explain function of PUSH and POP instructions.
4. Compare the memory mapped I/O with peripheral mapped I/O.
5. Write the functions of the following signals
   (i) SL_0–SL_3 (ii) RL_0–RL_3 (iii) CNTL/STB
   (iv) $OUTA_0$–$OUTA_3$ (v) $OUTA_0$–$OUTA_3$ (vi) IRQ
6. What are the advantages of DMA controlled data transfer over interrupt driven data transfer?

GROUP–C
(Long-Answer Type Questions)

Answer any three questions. (3 × 15 = 45)

7. (a) What are the flags in 8085? Discuss the flag register with some example.
   (b) Explain the generation of $\overline{MEMR}$, $\overline{MEMW}$, $\overline{IOR}$ and $\overline{IOW}$ control signals from $IO/\overline{M}$, $\overline{RD}$, $\overline{WR}$ signals.
   (c) Explain time delay loop using register and register pair. Write some applications of time delay loop.
   Calculate the time required to execute the following two instructions if the system clock frequency is 1MHz.

   LOOP: MOVA, B 5 T-states
   JMP LOOP 10 T-states

8. (a) What are the advantages and disadvantages of I/O mapped I/O over memory mapped I/O? Explain why I/O mapped I/O data transfer technique is limited to 256 input and 256 output peripherals.

(b) What do you mean by priority interrupts? Explain the operation of different interrupts available in 8085, with the help of circuit diagram.

(c) Draw an interface circuit of a D/A converter with 8085μP and write a program to convert the digital input to analog output.

9. (a) Write a BSR control word to set bits PC_7 and PC_0 and to reset them after 1-second delay.

(b) Determine the control word for the following configuration of the ports of 8255.
Port A—output and mode of port A is mode 1
Port B—output and mode of port B is mode 1
Remaining pins of Port C are used as input

(c) Write the interfacing procedure to interface 8253 with 8085 microprocessor.

10. (a) Define physical address and logical address.

(b) Write the procedure to determine physical address for the following instructions as given below:

(i) MOV AX, [SI + 03] (ii) MOV AL, CS:[BX + 0400]
(iii) MOV AX, [3000] (iv) MOV AL, [BX + SI + 22]
Assume CS = 4000H, IP = 2300, SI = 02300 and DS = 5000.

(c) From memory location 00490H successively 0AH, 9CH, B2H and 78H are stored respectively. What does AX contain after execution of each following instructions. Assume that SI contains 0490H and BP contains 0002H.

(i) MOV AX,SI (ii) MOV AX, (SI+1)
(iii) MOV AX, (SI+BP)

11. (a) Draw the timing diagram for execution of the instruction MVI A, 80H

(b) Specify the contents of register A and B and the status of flags S, Z and CY as the following instructions are executed.

```
XRA A
MVI B, 4AH
SUI 4F
ANA B
HLT
```

(c) Write an assembly language program to add two sixteen-bit data. Store the result and carry in two different register pairs.

Solution of 2009 WBUT Paper

GROUP–A
(Multiple-Choice Questions)

1. Choose the correct alternatives for any *ten* of the following: (10 × 1 = 10)

(i) Whenever the PUSH instruction is executed the stack pointer is
(a) decremented by 1 (b) decremented by 2
(c) incremented by 1 (d) incremented by 2

(ii) A single instruction to clear the lower four bits of the accumulator in 8085 microprocessor is
(a)XRI OFH (b) ANI FOH (c) ANI OFH (d) XRI FOH

(iii) Machine cycles in "CALL" instruction are
(a) 6 (b) 5 (c) 4 (d) 3

(iv) Address lines required for 32 k-byte memory chip are
(a) 13 (b) 14 (c) 15 (d) 16

(v) For 8255 PPI the bi-directional mode of operation is supported in
(a) Mode 1 (b) Mode 0 (c) Mode 2 (d) Either (a) and (c)

(vi) The CWR address of 8255 connected to 8085 is FBh, what will be the address for Port A?
(a) F8h (b) FAh (c) FC h (d) F9 h

(vii) The segment and offset address of the instruction to be executed by 8086 microprocessor are pointed by
(a) CS and SI (b) DS and IP (c) CS and SP (d) CS and IP

(viii) Mode 2 of 8253 is
(a) square wave generator (b) rate generator
(c) software trigger stroke (d) hardware trigger strobe

(ix) PSW is a ..... register.
(a) 8 bits (b) 16 bits (c) 20 bits (d) 32 bits

(x) If READY pin is grounded, it will introduce ..... states into the bus cycle of 8086 microprocessor
(a) wait (b) idle (c) wait and remain idle d) all of these

(xi) The call location for TRAP interrupt is
(a) 0000h (b) 0020h (c) 0024h (d) 0034h

(xii) In order to enable TRAP interrupt, which of the following instructions is are needed?
(a) EI only (b) SIM only (c) EI and SIM (d) none of these

(xiii) In 8085 microprocessor, the addressable memory is
(a) 64 KB (b) 1 MB (c) 4 KB (d) 16 KB

(xiv) What is the length of SP (stack pointer) of 8085 μP?
(a) 6 bits (b) 8 bits (c) 12 bits (d) 16 bits

(xv) What will be the content of the accumulator and status of CY flag after RLC operation, if the content of the accumulator is BC H and CY is 0?
(a) 79 H, 1 (b) 78 H, 1 (c) 5E H, 0 (d) 5D H, 0

Solution

(i) (b) decremented by 2	(ii) (b) ANI F0H	(iii) (b) 5	(iv) (c) 15
(v) (c) mode 2	(vi) (a) F8H	(vii) (d) CS and IP	(viii) (b) rate generator
(ix) (a) 8 bit	(x) (a) wait	(xi) (c) 0024H	(xii) (a) EI only
(xiii) (a) 64KB	(xiv) (d) 16 bit	(xv) (a) 79H, 1	

GROUP–B
(Short-Answer Questions)

Answer any *three* of the following. (3 × 5 = 15)

2. (a) Differentiate between peripheral mapped I/O and memory mapped I/O.
Refer Section 5.8 and Table 5.6.

(b) What are the functions of ALE, HOLD and READY?
Refer Section 2.3.

3. The following sequences of instructions are executed by 8085 µP:

C000	LX1 SP, D050	D050	05
C003	POPH	D051	40
C004	XRA A	D052	52
C005	MOV A, H	D053	03
C006	ADD L	D054	XX
C007	MOV H, A		
C008	PUSH H		
C009	PUSH PSW		
COOA	HLT		

What are the contents of Stack Pointer (SP), Program Counter (PC), Accumulator and HL pair?

The content of SP = D050H. After execution of POP H, the content of SP = D050 + 2 = D052H. When PUSH H is executed, the content of SP will be D052-2 = D050H. After execution of PUSH PSW, the content of SP = D050-2 = D04E.

Program Counter (PC) content is C00B after complete execution of above program.

The content of HL register after execution of LX1 SP, D050 and POP H is 4005. When XRA A is executed, Accumulator will be cleared or 00H. After execution of MOV A,H and ADD L, the content of A is 45H. Therefore the content of HL register pair is 4505H.

4. Discuss the functions of following instruction of 8085:
RAR, LHLD C020H, DAD, CALL D050H, DCX B
Refer Section 3.5.

5. What are the advantages of having segmentation? How does 8086 µP support segmentation?
The concept of memory segmentation has been introduced in the 8086 processor. Segmentation of memory is possible through the different segment registers such as Code Segment (CS), Data Segment (DS) Extra Segment (ES) and Stack Segment (SS). The segmentation can be used by programmer in relocating the program very easily. In a multiprogramming environment, access rights can easily be implemented with segments. It is also possible to share segments by different process. 8086 processor has separate data and code segments. So that, logical address of program can be loaded into specified segment register and run the program from anywhere in memory. Sometimes, one program can work on several different sets of data. This is possible by reloading logical address in DS register and the same program can be run with the new data.

The *advantages of segment memory* are given below:

(i) Allow the memory capacity to be 1MB even though the addresses associated with the individual instructions are 16 bits wide.

(ii) Allow the use of separate memory areas for the program code and data and stack portion of the program

(iii) Permit a program and/or its data to be placed into different areas of memory whenever the program is executed.

(iv) Multitasking becomes easy

(v) The advantage of having separate code and data segments is that one program can work on different sets of data. This is possible by reloading data segment register DS to the point to the new data.

(vi) Advantage of segment memory is that the reference logical addressed can be loaded into instruction pointer IP and run the program any where of segment memory as the logical address varies from 0000H to FFFFH.

(vii) Programs are re-locatable so that programs can be run at any location in memory.

In 8086, the code segment register is used for addressing a memory location in the code segment of the memory in which the program is stored for execution. Data segment register points to the data segment of the memory, where data is stored. The extra segment is a segment

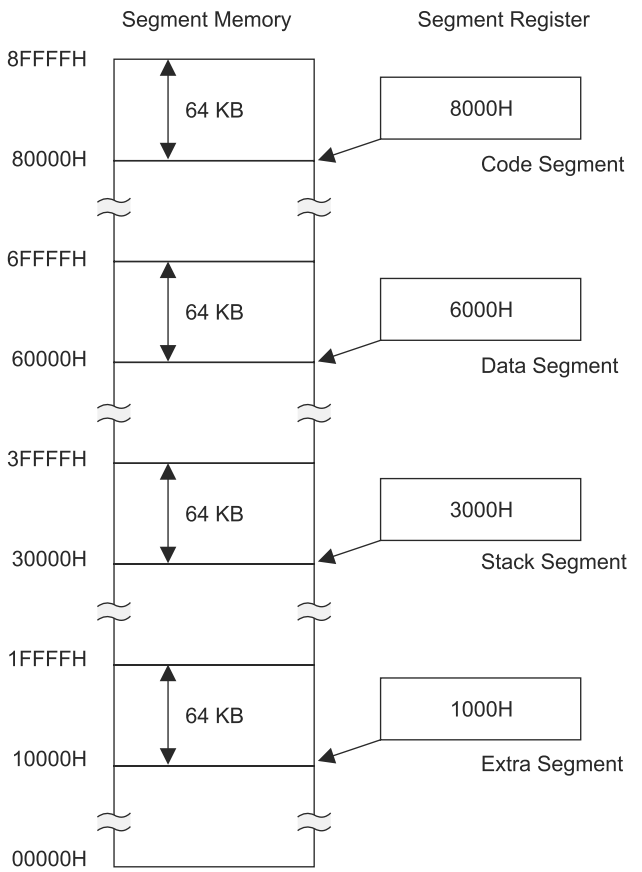


Fig. 1

which can be used as another data segment of the memory. Therefore, extra segment contains data. The stack segment register is used for addressing stack segment of memory in which stack data is stored. The CPU uses the stack for temporarily store data, i.e. the content of all general purpose registers which will be used later. In memory segmentation, the complete 1MB memory can be divided into 16 parts which are called segments. Each segment thus contains 64 KB of memory. Figure 1

6. What is subroutine? What is the difference between CALL and JMP instructions?
Refer Section 4.6 and Section 3.5.4.

GROUP–C
(Long-Answer Questions)

Answer any *three* of the following (3 × 15 = 45)

7. (a) How many ports are there in 8255 and what are they?
There are three eight-bit ports in 8255 such as Port A, Port B, and Port C.

(b) Discuss the different bits of the control word of 8255.
Refer Section 12.5.

(c) Write down the MODE-0 control word for the following:

(i) Port A = Input

(ii) Port B not used

(iii) Port C upper = Input, Port C lower = output.

The Control word Bits of 8255 is given below

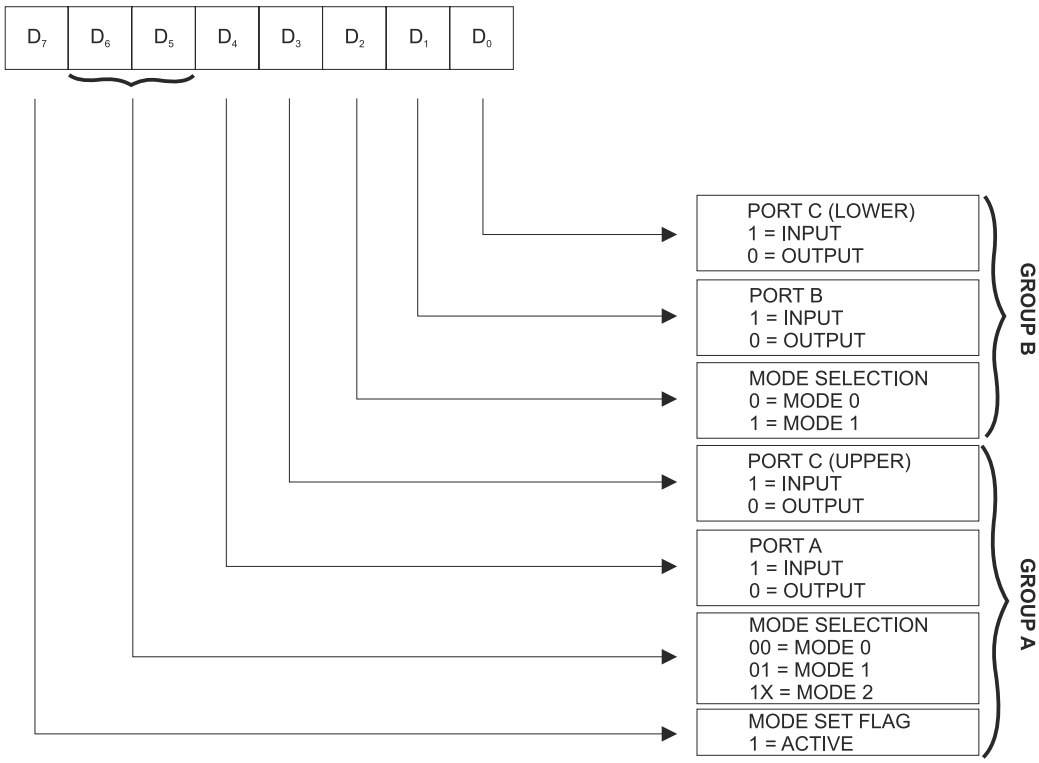


Fig. 1

When 8255 operate in mode 0, Port A as Input, Port C upper as Input, Port C lower as output and Port B not used, the control word will be = 10011010 = 9AH as $D_0 = 1$, $D_1 = 1$ (assuming Port B as input), $D_2 = 0$, $D_3 = 1$, $D_4 = 1$, D_5 $D_6 = 00$ and $D_7 = 1$.

(d) Discuss BSR operation of 8255.

Refer Section 12.4.4 and Fig. 12.14.

8. (a) Explain how 20-bit physical address is generated in 8086 microprocessor.

Refer Section 9.4.

(b) What is the purpose of queue? How many words does the queue store in the 8086 microprocessor?

Refer Section 9.2.1.

(c)How does 8086 support pipelining? Explain.

Refer Section 9.2.3.

(d) What are the advantages of having memory segmentation?

Refer Solution 2009 Question 5.

9. (a) Describe the priority scheme and EOI scheme of 8259.

Refer Sections 14.3 and 14.4.

(b) Write down the format of ICW1 and ICW2 of 8259.

Refer Section 14.6.1.

(c) With respect to 8237, explain the DMA operation.

8237 is an advanced Programmable DMA controller. It provides a better performance compared to 8257. This is capable of transferring a byte or a bulk of data between system memory and peripherals in either direction. Memory to memory data transfer is also possible in this peripheral. The 8237 can support four independent DMA channels which can be expanded to any number after cascading more number of 8237.

8237 operates in two cycles such as passive cycle and active cycle. Each cycle contains a fixed number of states. The 8237 can assume six states, when it is in active cycle. During idle cycle, it is in idle state (SI).

The 8237 is initially in a state SI means an idle state where the 8237 does not have any valid pending DMA request. During this time, although the 8237 may be idle, the CPU may program it in this state. Once there is a DMA request, the 8237 enters state S_0, which is the first state of the DMA operation. When the 8237 requests the CPU for a DMA operation and the CPU has not acknowledged the request, the 8237 waits in S_0 state. The acknowledge signal from the CPU indicates that the data transfer may now begin. The S_1, S_2, S_3 and S_4 are the working states of DMA operation, in which the actual data transfer is carried out. If more time is required to complete a transfer than that is allowed, wait states may be inserted between S_2 and S_3 or S_3 and S_4 using the READY pin of 8237. So it is clear that a memory read or a memory write DMA operation actually requires four states S_1 to S_4.

10. (a) Write a program to find out the largest number, starting from D000 H of 10 numbers and store result in D050 H.

The count value of numbers 0AH is stored in C register directly and the numbers are stored in the memory locations from D000 to D009. The largest number will be stored in D050 location. Assume the program memory starts from 9100H. The algorithm to find out the largest number from an array is given below:

Step 1 Load count value of numbers 0AH in C register immediately
Step 2 Load the first number in accumulator from memory location D000H
Step 3 Move first number in accumulator
Step 4 Decrement the count value by one
Step 5 Move to next memory location for next data
Step 6 Compare the content of memory with content of accumulator
Step 7 If carry is generated, copy content of memory in accumulator
Step 8 Decrement the count value by one.
Step 9 If count value does not equal zero, repeat step 5 to step 8
Step 10 Store result in D050H location

Memory address	Labels	Mnemonics	Operands	Comments
9100		MVI	C,0A	Load count value in C register
9102		LXI	H,D000	Load address of first data in HL register pair
9105		MOV	A,M	Copy 1st data in accumulator
9106		DCR	C	Decrement C register
9107	LOOP	INX	H	Increment HL register for address of next data
9108		CMP	M	Compare next data with the content of accumulator
9109		JNC	LEVEL	If carry is not generated, jump to LEVEL
910C		MOV	A,M	Copy large number in accumulator from memory
910D	LEVEL	DCR	C	Decrement C register
910E		JNZ	LOOP	Jump not zero to LOOP
9111		STA	D050	Store largest number in D050H location
9114		HLT		

Example

DATA		RESULT	
Memory location	**Data**	**Memory location**	**Data**
D000	02H		
D001	23H	D050	FF
D002	FFH		
D003	47H		
D004	92H		
D005	10H		
D006	56H		
D007	27H		
D008	98H		
D009	40H		

(b) Write a program to find out square of a data using look-up table.
Refer Section 4.11.20, Page 4.39.

11. (a) What are the vectored and non-vectored interrupts?

Refer Sections 6.3 and 6.4.

(b) Explain the instruction RIM and SIM. Write the program for enable the RST-7.5, RST-6.5 and disable RST-5.5.

Refer Section 6.5 for RIM and SIM instruction

Initially determine the contents of the accumulator for enable the RST-7.5, RST-6.5 and disable RST-5.5

Disable 5.5	bit $D_0 = 1$
Enable 6.5	bit $D_1 = 0$
Enable 7.5	bit $D_2 = 0$
Allow setting the masks	bit $D_3 = 1$
Don't reset the flip flop	bit $D_4 = 0$
Bit 5 is not used	bit $D_5 = 0$
Don't use serial data	bit $D_6 = 0$
Serial data is ignored	bit $D_7 = 0$

SOD	SDE	xxx	R7.5	MSE	M7.5	M6.5	M5.5
0	0	0	0	1	0	0	1

Therefore, the content of accumulator is 09H. The program for above operation is given below:

Memory address	Labels	Mnemonics	Operands	Comments
8000		EI		Enable all interrupts
8001		MVI	A, 09H	Mask to enable RST 7.5, and 6.5, disable 5.5
8003		SIM		Apply the settings RST masks

(c)Discuss how 8253 is used to generate square wave.

Refer Section 13.5.4.

(d) What are the major components of 8259A interrupt controller? Explain their functions.

Refer Section 14.3.

(e)Write the BSR control word for setting PC_4 in 8255A.

In Single Bit Set/Reset (BSR) mode any of the eight bits of Port C can be set or reset using a single Output instruction. This feature reduces software requirements in control-based applications.

When Port C is being used as status/control for Port A or B, these bits can be set or reset by using the Bit Set/Reset operation just as if they were output ports. Figure 4 shows the Bit set/reset format.

The BSR control word to set bit PC_4 is 00001001 = 09H as $D_0 = 1$, $D_1 = 0$, $D_2 = 0$, $D_3 = 1$, $D_4 = 0$, $D_5 = 0$, $D_6 = 0$, and $D_7 = 0$.

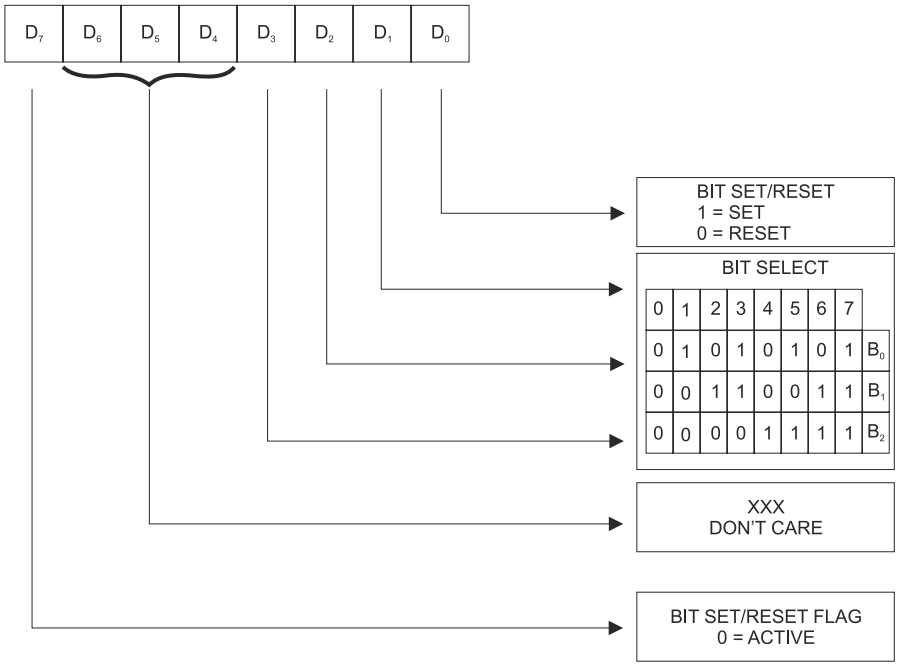


Fig. 2 Bit set/reset format

Solution of 2010 WBUT Paper (EI-405)

GROUP–A
(Multiple-Choice Questions)

1. Choose the correct alternatives for any *ten* of the following: (10 × 1 = 10)

(i) RST 7.5 interrupt is
(a) vectored and maskable (b) vectored and nonmaskable
(c) direct and maskable (d) direct and nonmaskable.

(ii) The address lines required for 16K byte memory chip are
(a) 13 (b) 14 (c) 15 (d) 16

(iii) The instruction register holds
(a) flag conditions (b) op code
(c) instruction address (d) hex codes

(iv) Through which of the following pins does the 8085A microprocessor communicate with serial address
(a) 2 and 3 (b) 4 and 5 (c) 6 and 7 (d) 8 and 9

(v) In the 8085 microprocessor, the JMP 8050 instruction affects the
(a) accumulator (b) stack pointer (c) HL pair register (d) program counter

(vi) A single instruction to clear the lower four bits of the accumulator in the 8085 microprocessor is
(a) XRI 0FH (b) ANI F0H (c) ANI 0FH (d) XRI F0H

(vii) The accumulator is a
(a) register (b) memory address (c) flip-flop (d) address bus

(viii) Nibble is group of …… bits.
(a) 2 (b) 4 (c) 8 (d) 1

(ix) When the LHLD is executed, number of *T* states required are
(a) 10 (b) 14 (c) 13 (d) 15

(x) When the subroutine is called, the address of the instruction next to CALL is saved in
(a) stack-pointer register (b) program counter
(c) stack (d) PSW

(xi) For 8255 PPI, the bidirectional mode of operation is supported in
(a) Mode-1 (b) Mode-2 (c) Mode-0 (d) either (a) and (b)

(xii) If the content of the accumulator is F8 , after execution of the RCC instruction, the content of the accumulator will be
(a) 9E (b) 8C (c) 8B (d) 7C

Solution

(i) (a) vectored and maskable (ii) (b) 14 (iii) (b) Op code
(iv) (b) 4 and 5 (v) (d) Program counter (vi) (b) ANI 0FH
(vii) (a) register (viii) (b) 4 (ix) (d) 15
(x) (c) stack (xi) (b) Mode-2 (xii) (d) 7C

GROUP–B
(Short-Answer Questions)

Answer any *three* of the following. (3 × 5 = 15)

2. Write the function of the following instructions

(i) JNC

Refer Page 3.21.

(ii) CALL

Refer Page 3.22.

(iii) DAD

Refer Page 3.13.

(iv) RAR

Refer Page 3.18.

(v) MVI

Refer Page 3.10.

3. What is a microprocessor? Explain with a block diagram.

Refer Section 1.2.

4. Explain the difference between I/O mapped I/O and memory mapped I/O.

Refer Section 5.8 and Table 5.6.

5. What is the advantage of multiplexed address and data bus? Show how it can be demultiplexed in 8085.

The advantage of multiplexed address and data bus is that the address and data lines are shared. As a result, less number of pins are required. But it is required to separate the address and data lines. Therefore, an extra IC is required to demultiplex address line A_0–A_7 and data line D_0–D_7. When the multiplex address/data bus is compared with parallel address/data bus, the parallel address/data bus requires less time to read/write data and the speed of operation increases.

The 8085 microprocessor has higher-order address bus A_8–A_{15} and lower order address data bus $AD_0 - AD_7$. The lower-order address data bus is multiplexed as address bus and data bus. During the first clock pulse of a machine cycle, the program counter releases the lower order address in $AD_0 - AD_7$ and higher-order address $A_8 - A_{15}$. Then ALE signal is high; the $AD_0 - AD_7$ can be used as lower order address bus and not a data bus. The external latch circuit makes the difference between the address and data bus as shown in Fig. 1.

6. How many flag bits are there in the 8085 microprocessor? Explain each of them.

Refer Page 2.10.

GROUP–C
(Long-Answer Questions)

Answer any *three* of the following (3 × 15 = 45)

7. (a) What is the difference between a microprocessor and microcontroller?

Refer Page 7.2.

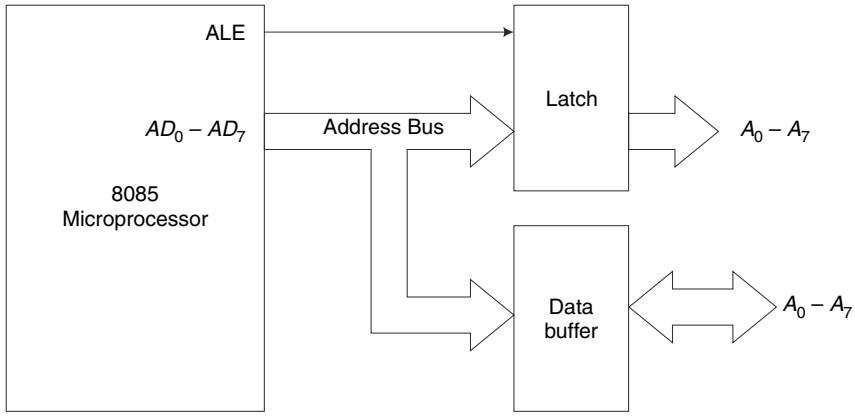


Fig. 1 Multiplexing of lower order address and data bus

(b) Discuss the memory organization of 8051 microcontroller. What is the function of program status word (PSW) in 8052?

Refer Section 7.3 and Page 7.6.

(c) Explain interrupts in 8051 controller.

There are five interrupt sources for the 8051 microcontroller. The priority-wise five different interrupts of 8051 microcontroller are given below:

- External Interrupt 0
- Timer 0
- External Interrupt 1
- Timer 1
- Serial Port

These interrupts can recognize 5 different events that can interrupt regular program execution.

- Each interrupt can be enabled separately.
- Each interrupt type has a separate vector address.
- Each interrupt type can be programmed to one of two priority levels.
- External interrupts can be programmed for edge or level sensitivity.
- Each interrupt can be enabled or disabled by setting bits of the IE register. Likewise, the whole interrupt system can be disabled by clearing the EA bit of the same register as shown in Fig. 2.

If the IT0 and IT1 bits of the TCON register are set, an interrupt will be generated on high to low transition, i.e., on the falling pulse edge (only in that moment). If these bits are cleared, an interrupt will be continuously executed as far as the pins are held low.

IE is Interrupt Enable Register which is shown in Fig.2. The function of EA, ES, ET1, EX1, ET0 and EX0 are given below:

- **EA** — global interrupt enable/disable:
  0 — disables all interrupt requests
  1 — enables all individual interrupt requests
- **ES** — enables or disables serial interrupt:
  0 — UART system cannot generate an interrupt
  1 — UART system enables an interrupt

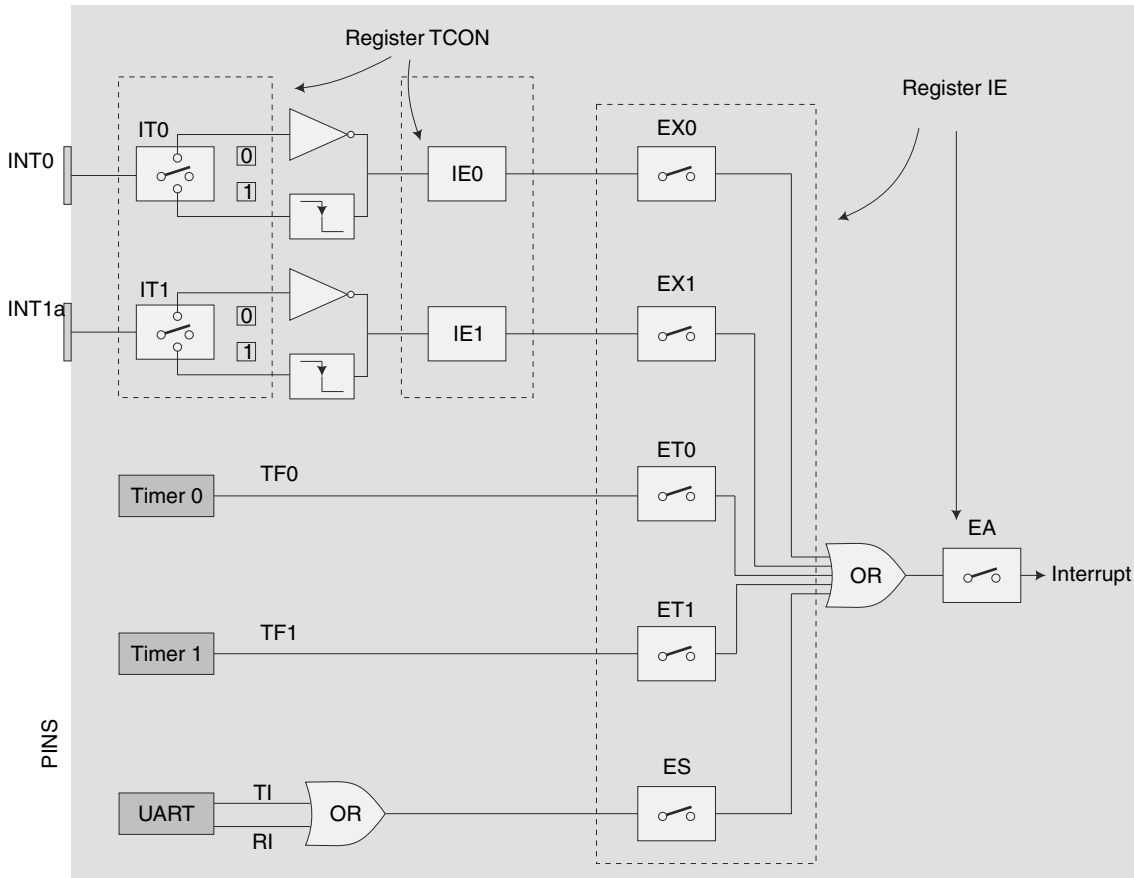


Fig. 2

- **ET1** — bit enables or disables Timer 1 interrupt:
  0 — Timer 1 cannot generate an interrupt
  1 — Timer 1 enables an interrupt
- **EX1** — bit enables or disables external 1 interrupt:
  0 — change of the pin INT0 logic state cannot generate an interrupt
  1 — enables an external interrupt on the pin INT0 state change
- **ET0** — bit enables or disables timer 0 interrupt:
  0 — Timer 0 cannot generate an interrupt
  1 — enables timer 0 interrupt
- **EX0** — bit enables or disables external 0 interrupt:
  0 — change of the INT1 pin logic state cannot generate an interrupt
  1 — enables an external interrupt on the pin INT1 state change and interrupt vector addresses are given in Table 1.

Fig. 3

Table 1 Interrupt Vector Addresses

Source	Address
IE0	03H
TF0	0BH
IE1	13H
TF1	1BH
RI & TI	23H

8. (a) Describe the different addressing modes of the 8085 microprocessor.

Refer Section 3.2.

(b) Specify the contents of registers *A* and *B* and the status flags *S*, *Z* and *CR* when the following instructions are executed:

XRA A
MVI B,4AH
MVI4FH
ANA B
HLT

XRA A	The contents of the accumulator is exclusive ORed with the content of accumulator. After execution, the content of accumulator will be 00H. The CY and Ac flags are reset.
MVI B,4AH	Move 4AH into B register immediately. Flags are not affected.
SBI 4FH	The 8-bit data 4FH and the borrow flag is subtracted from the content of the accumulator. All flags are affected to reflect the result of the subtraction.
ANA B	The content of the accumulator is logically ANDed with the contents of the register. All flags are modified to reflect the result of the AND operation. CY is reset and Z flag is set.
HLT	When the HLT instruction is executed, the microprocessor finishes execution of the current instruction and halts any further execution. No status flags are affected.

(c) Write an assembly-language program to generate a square wave of 400 μs delay and use 020 H for the output.

Levels	*Instructions*	*Comments*
	MOV A, 80H	80H is the control word when all ports of 8255 operates as output port. Load control word 80H in the accumulator.
	OUT 23H	Write control word in control word register.
START	MVI A,FFH	Load FFH in the accumulator.
	OUT 20H	Send FFH to Port *A* and all pins of Port *A* are high
	CALL DELAY	Call a delay loop for 400 μs delay.
	MVI A,00H	Load 00H in accumulator.
	OUT 20H	Send 00H to Port *A* and all pins of Port *A* are low.
	CALL DELAY	Call a delay loop for 400 μs delay.
	JMP START	Jump to START location for generating square wave continuously.

Levels	*Instructions*	*T states*	*Comments*
DELAY	MVI B,A6H	7	Load A6H in register *B* immediately.
LOOP	DCR B	5	Decrement the content of *B* by 1.
	JNZ LOOP	10	Jump not zero to LOOP.

Total number of *T* states for execution of LOOP is
= (*T* states for DCR B +T states for JNZ) × Number of times LOOP is executed
= (7 + 5) × *N*
If the operating frequency of microprocessor is 5 MHz, the total delay time is

$$12\,N \times \frac{1}{f} = 12\,N \times \frac{1}{5 \times 10^6} = 400 \times 10^{-6}$$

$$\text{Or } N = \frac{400 \times 5}{12} = 166.67 = \text{A6H}$$

Then the content of register *B* will be A6H for 400 μs delay.

9. (a) What is tri-state? Why is it important?.

Any logic gate has two output states: logic '0' and logic '1', but the tri-state logic gates have three output states as given below:

- Low level state or logic '0' state
- High level state or logic '1' state
- High impedance state (Z)

The graphic symbol of tri-state logic gates are shown in Figure 4. The tri-state gate consists of an extra input terminal called enable or control input (C). When the control input is at logic '0', the gate performs its normal operation. If the control input is at logic '1', the output of gate becomes tri-state or high impedance state irrespective of inputs. In high impedance state, the gate is physically disconnected from its output terminal when C is at logic '1' and the output appears as an open circuit.

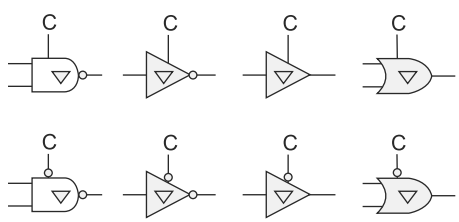


Fig. 4 Symbols of tri-state logic gates

Usually, tri-state logic gates are used for proper functioning of microprocessor. In a microprocessor based system, peripheral devices are connected in parallel between address bus and data bus. As the tri-state logic devices are used in system, peripheral devices do not load the system. Therefore, the tri-state logic devices are very necessary to avoid loading of the system bus. Hence the microprocessor can communicate with one peripheral device at a time after enabling the tri-state control signal.

(b) Can an input port and output port has the same address? Justify your answer.

Refer WBUT 2010 Question 4(iii) [EI-502].

(c) What is the function of a subroutine? How is a subroutine handled in a microprocessor?

Refer Section 4.6.

(d) What is the advantage of DMA controlled data transfer over program-controlled data transfer?

Refer Section 17.1.

10. (a) Explain the different modes of operation of 8255.

Refer Section 12.4 and Page 12.5.

(b) Explain the control word format of the 8255 in I/O and BSR mode.

Refer Page 12.4, Fig.12.4 and Sections 12.5 and 12.4.4.

(c) Write the control word to set Port *A* as input mode 1 and load this control word into the control word register. Briefly describe the process of data transfer from input device to processor with handshaking signal. Draw its timing diagram.

To compute the control word, we assume that

- Port *A* is used as input and operation mode of Port *A* is Mode 1
- Port *B* can be used as output and operates in Mode 1
- PC_6 and PC_7 act as input

Six pins of the Port *C*, $PC_0 - PC_5$ are used to control Port *A* and Port *B* in Mode 1 operation. $PC_0 - PC_2$ are used for the control of the Port *B*. Port *B* can be programmed as an input or output port. When Port *A* is operated as an input port, $PC_3 - PC_5$ are used to control this port. In this operating mode PC_6 and PC_7 may be used as input or output. Figure 5 shows the control-word bits for the above configuration of the ports. The control word for the above definition of the ports of Intel 8255 is BD H. To load the control word in control word register, the following instructions are required

Instructions	*Comments*
MOV A, BDH	BDH is the control word when all ports of 8255 operates as output port. Load control word BDH in the accumulator.
OUT 23H	Write control word in control word register

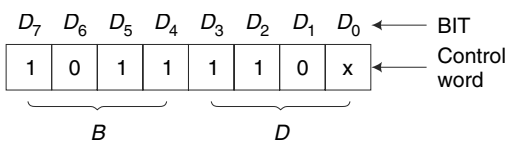


Fig. 5

Refer Section 7.4.2 and Fig.7.8.

11. (a) Explain how 8254 can be used on a square-wave generator with a 1 ms period. The input frequency to 8254 is 2MHz.

We assume that 80H is address of counter-0, 81H is address of counter-1, 82H is address of counter-2 and 83H is address of control word register.

To generate a square wave, 8254 should be used in mode 3 and counter-0 will be operating in BCD mode. The count value can be calculated for 1ms delay period.

Assume 8254 operates at 2 MHz.

The time period is 0.5 μs.

The number of T states is required for 1 ms delay is

$$N = \frac{1 \times 10^{-3}}{0.5 \times 10^{-6}} = 2000 \text{ states.}$$

The control word of 8254 is 37H as shown below:

SC_1	SC_0	RL_1	RL_0	M_2	M_1	M_0	BCD
0	0	1	1	0	1	1	1

The following instruction will be executed to generate a square wave.

Levels	*Instructions*	*Comments*
START	MOV A, 37H	Intialize 8254 counter with control word 37H
	OUT 83	Load control word into control word register. Counter-0 operates in mode 3
	MVI A, 00H	Write 00H in Accumulator
	OUT 80	LSB of count value is 00H which is loaded in the counter
	MVI A, 20H	Write 20H in Accumulator
	OUT 80, AL	MSB of count value is 20H which is loaded in the counter
	HLT	Halt

(a) Write a note on DMA controller.

Refer Section 17.1.

Solution of 2010 WBUT Paper (EI-502)

GROUP–A
(Multiple-Choice Questions)

1. Choose the correct alternatives for any *ten* of the following: (10 × 1 = 10)

(i) What is the vector location of NMI?
(a) 00000H (b) 00008H (c) 00010H (d) 00014H

(ii) The interrupt pin available in the 8085A microprocessor chip is
(a) ALE (b) HLDA (c) INTER (d) SOD

(iii) For 8257 controller _____ is the highest priority channel by default.
(a) CH-3 (b) CH-0 (c) CH-1 (d) any channel

(iv) The clock frequency is 60 Hz. The clock interrupt handler on a computer needs 2 ms per clock tick. What percentage of the CPU is devoted to the clock?
(a) 1.2 (b) 7.5 (c) 12 (d) 18.5

(v) What is the BSR control word to set PC4?
(a) 09 (b) 07 (c) 04 (d) 05

(vi) The total memory space available in 8086 is
(a) 16 KB (b) 64 KB (c) 1 MB (d) 256 KB

(vii) An 8-bit A/D converter has a resolution of
(a) $\frac{1}{2^4}$ (b) $\frac{1}{2^8}$ (c) $\frac{1}{2^2}$ (d) $\frac{1}{2^{16}}$

(viii) Select the invalid instruction:
(a) MOV M,A (b) ADI 67 (c) LDAX B (d) STAX H

(ix) The maximum operating frequency of 8054 is
(a) 2 MHz (b) 3 MHz (c) 6 MHz (d) 8 MHz

(x) The number of multiplexed buses in case of 8086 is
(a) 16 (b) 8 (c) 20 (d) 4

(xi) 8086 exchanges data word with odd memory bank when
(a) $\overline{BHE} = 0$ and $A_0 = 0$ (b) $\overline{BHE} = 0$ and $A_0 = 1$
(c) $\overline{BHE} = 1$ and $A_0 = 0$ (d) $\overline{BHE} = 1$ and $A_0 = 1$

(xii) If ready pin is grounded, it will introduce _____ states into the bus cycle of 8086/8088 μP.
(a) wait (b) ideal
(c) wait and remain ideal (d) all of these

Solution

(i) (b) 00008H (ii) (c) INTER (iii) (b) CH-0 (iv) (c) 12
(v) (a) 09 (vi) (c) 1 MB (vii) (b) $1/2^8$ (viii) STAX H
(ix) (d) 8 MHz (x) (a) 16 (xi) (b) $\overline{BHE} = 0$ and $A_0 = 1$
(xii) (a) wait

GROUP–B
(Short-Answer Questions)

Answer any *three* of the following. (3 × 5 = 15)

2. Write down the steps of execution of an instruction sequentially by 8085 microprocessor.
Refer Section 2.2.2.

3. Write an ALP in 8085 to find the total number of zeros and ones in a given string. Store the results in the memory location 9055H onwards.

Levels	*Instructions*	*Comments*
	LXI H,8000H	Address of number of bytes in H-L register pair
	MOV C,M	Transfer number of bytes from memory location to register *C*
	XRA A	Clear accumulator register
	MVI B,00H	Initialize register *B* with 00H as number of zeros
	MVI D,00H	Initialize register *D* with 00H as number of ones
START	MOV E,08H	Load 08H for number of rotate the accumulator right through carry
	INXH	Increment HL register pair
	MOV A,M	Load the first number in the accumulator
LOOP2	RAR	Rotate the accumulator right through carry
	JC LOOP1	Jump with carry to LOO 1
	INC B	Increment *B* when carry bit is 0
LOOP1	INC D	Increment *D* when carry bit is 0
	DEC E	Decrement *E* register
	JNZ LOOP2	Jump not zero toe LOOP 2
	DEC C	Decrement register *C*
	JNZ START	Jump not zero to START
	MOV A, B	Move content of *B* into register *A*
	STAX 9055H	Store number of zeros in memory location 9055H
	MOV A, D	Move content of *D* into register *A*
	STAX 9055H	Store number of zeros in memory location 9056H
	HLT	

4. (i) Explain why the number of output ports in the peripheral mapped I/O is restricted to 256 ports.

The input/output instructions (IN and OUT) are used as a 1-byte address. The operation of IN and OUT instructions are given below.

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

A←[Port]

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

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

[Port]←A

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

With the 1-byte address of IN and OUT instructions, it is possible to address 00H to FFH or 256 ports. Therefore, the number of output ports in the peripheral mapped I/O is restricted to 256 ports.

(ii) Specify the 8085 signals that are used to enable an input and output port.

Input output read and write signals ($\overline{IOR}$ and $\overline{IOW}$) are used to enable input and output port.

Refer Section 5.8.1

(iii) If an output port can have the same 8-bit address, how does the 8085 differentiate between ports?

The microprocessor can communicate with I/O devices through 8255. It can read data from any I/O and it can also write data to any I/O devices. The I/O devices are identified by port addresses. Usually, the I/O read and write operations are performed by using software instructions such as IN and OUT. The I/O read and write operations are controlled by control signals — IOR and IOW respectively. The port addresses are varied from 00H to FFH. Therefore, 256 I/O devices may be connected with microprocessor in I/O mapped I/O devices. As IN and OUT instructions are used for read and write operation respectively, the input port and output port address may be same but at a time only one operation can be performed either input or output.

5. Draw the timing diagram of MOV A,M.

The instruction MOV A, M is a one-byte instruction, two machine cycles such as instruction fetch cycle and one memory read cycle, are required to execute this instruction. In fetch cycle, the microprocessor fetches the opcode of an instruction from the memory. During memory read cycle, the microprocessors read data from memory and load into accumulator. Figure 1 shows the timing diagram for an opcode fetch cycle of an instruction MOV A, M. Assume that the opcode of instruction MOV A,M is 7EH which is stored in 8000H.

Memory location	*Opcode*	*Mnemonics*
8000H	7EH	MOV A,M

Machine Cycle - 1

To execute the instruction fetch cycle, four consecutive clock cycles T_1, T_2, T_3 and T_4 are required. The sequences of operations are given below:

First Clock Cycle

- In the first clock cycle, T_1, the microprocessor, places the content of program counter, address of the memory location 8000H, where the opcode is available on the 16-bit address bus. The 8 MSBs of the memory address (80H) are placed on the high-order address bus,A_{15}-A_8 and 8 LSBs of the memory address (00H) are placed on the low-order address bus,AD_7-AD_0. Since the AD bus is needed to transfer data during subsequent clock cycles, it is used in time-multiplexed mode.
- The microprocessor sends an address latch enable (ALE) signal to go high and latch the 8 LSBs of the memory address. Therefore, low-order address bus is demultiplexed and the complete 16-bit memory address is available in the subsequent clock cycles to get the opcode from the specified memory address, 8000H.

- The microprocessor sends the status signals IO/$\overline{M}$=0, S_0 =1 and S_1=1 to indicate opcode fetch operation.

Second Clock Cycle

- During T_2, low-order bus $AD_7 - AD_0$ is ready to carry data from memory location. The microprocessor sends the control signal $\overline{RD} = 0$ to enable memory and the program counter is incremented by 1 to 8001H. Now the opcode from the specified memory location 8000H is placed on the data bus.

Third Clock Cycle

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

Fourth Clock Cycle

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

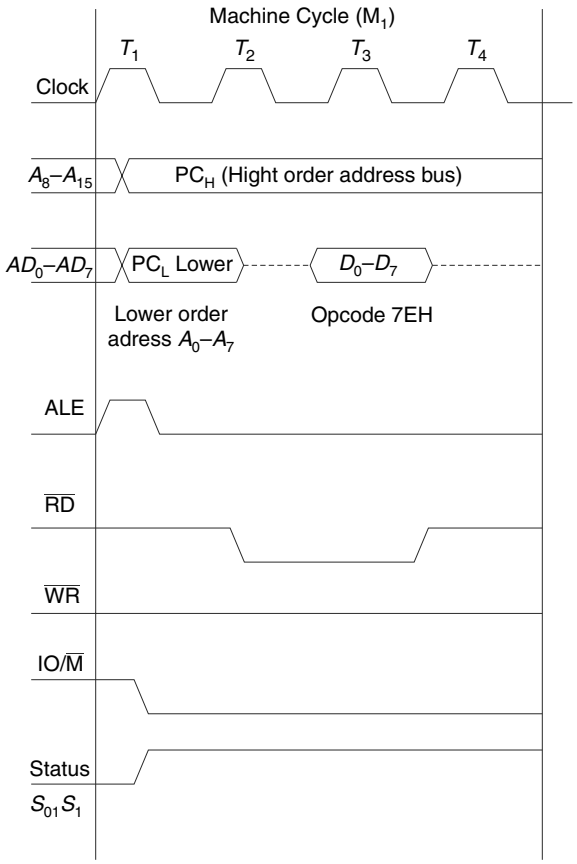


Fig. 1 Timing diagram of Opcode Fetch Operation (MOV A,M)

Machine Cycle – 2

During T_1 state of machine cycle-2, the contents of HL registers are loaded to the address bus instead of program counter. Therefore, the contents of program counter will not be modified during T_1 state. ALE signal is active.

During T_2 state, the $\overline{RD}$ signal is low and the data from memory located by the address bus is loaded into the data bus.

During T_3 state, the $\overline{RD}$ signal is high and data from the data bus is transferred into the accumulator register.

Refer memory read cycle from Chapter 3.

6. State the difference between architectures of 8086 and 8088 µP.

Refer Table 9.2, Page 9.2.

GROUP–C
(Long-Answer Questions)

Answer any *three* of the following (3 × 15 = 45)

7. (a) What do you mean by DMA operation? Write down the steps of DMA operation. What is fixed priority mode and what is rotating priority mode?

Refer Section 17.1

Refer Section 17.3.1, Fig. 17.6.

(b) If the clock frequency is 5 MHz, how much time is required to execute the instruction MVI B,08H(7T state)? Draw the timing diagram.

The clock frequency is 5 MHz.

The time period of clock is $\frac{1}{5 \times 10^6} = 0.2\ \mu s$

As 7T states are required to execute the instruction MVI B,08H, the total execution time of the instruction MVI B,08H is $7 \times 0.2\ \mu s = 1.4\ \mu s$.

The timing diagram for a two-byte instruction MVI B, 08H data is illustrated in Fig. 2.

The MVI B, 08H is a two-byte instruction. In the coded from it is written as 06, 08 where 06 is opcode for MVI B instruction and 08 is data. This instruction is stored in two consecutive memory location 8000H and 8001H.

Memory location	*Opcode*	*Mnemonics*
8000H	06 H	MVI B, 08H
8001H	08 H	

This instruction requires two machine cycles, M_1 and M_2. The first machine cycle M_1 is known as the fetch cycle to fetch operation code 3E from the memory. The second machine cycle M_2 is used to read the operand (FFH) from the memory. Actually this is a memory read cycle. Fig. shows the machine cycle M_2 and its operation is explained below:

First Clock Cycle of M_2

- In the first clock cycle (T_1) the microprocessor places the content of program counter, 8001H, which is the address of operand on the 16-bit address bus. The 8 MSBs of the memory address, 80H are placed on the high-order address bus, $A_{15} - A_8$ and 8 LSBs of the memory address, 00H are placed on the low-order address bus, $AD_7 - AD_0$.

- The microprocessor sends an address latch enable (ALE) signal to go high and latch the 8 LSBs of the memory address. Then low-order address bus is demultiplexed and the complete 16-bit memory address is available in the subsequent clock cycles to get the operand from memory location, 8001H.
- The status signals IO/$\overline{M}$=0, $S_0 = 0$ and $S_1 = 1$ to identify the memory read operation.

Second Clock Cycle of M_2

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

Third Clock Cycle of M_2

- During T_3, the microprocessor reads the operand $\overline{RD}$. becomes high during T_3 and the memory is disabled.
- Microprocessor also places the operand in register *B*.

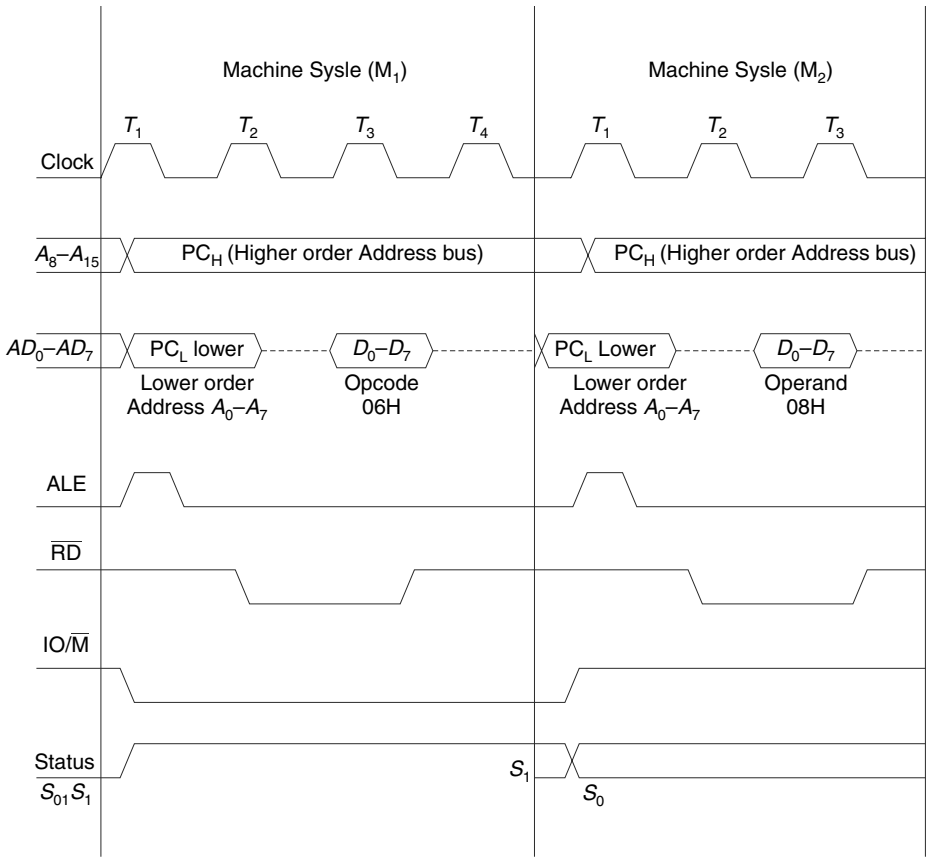


Fig. 2 Timing diagram of MVI R, data (MVI B, 08H)

8. (a) What is two key lockout and *N*-key rollover mode of 8279? How an A/D converter can be interfaced with a 8085 microprocessor?

Refer Solution of 2008 WBUT Question No. 11(e) and Refer Section 18.7.

(b) Write an assembly-language program for BCD to binary conversion.

Refer Page 11.36.

(c) What the functions of RESET, HOLD, INTERRUPT and READY pins?

Refer Section 2.3, Page 2.16 and 2.17.

9. (a) Describe the different addressing modes of the 8086 microprocessor.

Refer Section 10.2.

(b) Draw the architecture of 8086. What are the main functions of BIU and EU unit of 8086 µp ?

Refer Fig. 9.1 and Sections 9.2.1 and 9.2.2.

(c) How is pipelining achieved in the 8086 microprocessor?

Refer Section 9.2.3.

10. (a) What do you mean by an interrupt-driven system? Arrange the interrupt according to their priority. Define maskable and nonmaskable interrupts. What is interrupt call location?

Refer Sections 6.1, 6.2, 6.3 and 6.4.

(b) Why is a decoder circuit needed? Using 74LS138, explain the interfacing of memory and IO devices.

Refer Section 5.8.

The microprocessor communicates with various memory ICs. The interfacing between microprocessor, memory and I/O devices through address bus, data bus and control bus is depicted in Fig. 3. The address decoder is used to select proper memory and I/O devices is given in Fig. 4.

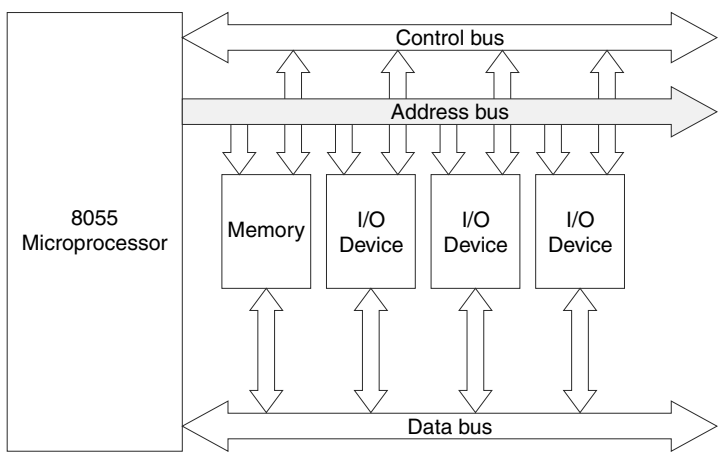


Fig. 3 Schematic block diagram for memory and I/O interfacing with microprocessor

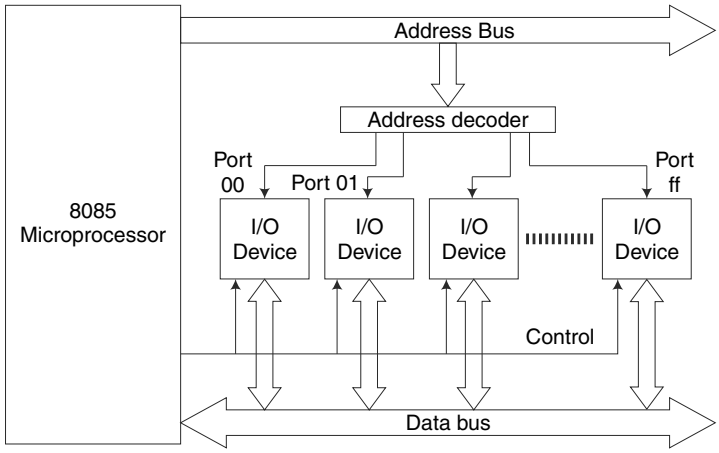


Fig. 4 I/O mapped I/O devices

When address decoder is enabled and chip select signals are applied to decoder, RAM or EPROM or I/O devices are selected. Data will be stored or read from memory devices or I/O devices. The total 64K addresses are to be assigned to memories and I/O devices. There are two types of address mapping: memory mapped I/O and I/O mapped I/O.

In some microprocessors, memory and I/O operation can be differentiated by control signals. The control signal IO/$\overline{M}$ is available to distinguish between memory and I/O operations. When the control signal IO/$\overline{M}$ is high, an I/O operation can be performed. If the control signal IO/$\overline{M}$ is low, memory operations will be performed. In this case, the same address can be assigned to I/O devices as well as memory location. Generally, two separate address spaces exist and each address space can be entirely assigned to either memory or I/O devices. This technique is known as I/O mapped I/O.

In an I/O-mapped-I/O scheme, I/O device cannot be considered as memory location. The I/O-mapped-I/O scheme requires special instruction like IN/OUT to access I/O devices and special signals IO/$\overline{M}$. In this scheme, 8085 can access 256 I/O ports. In 8085 microprocessor, this scheme requires 8-bit address lines. It requires less hardware to decode 8-bit address. Arithmetical or logical operations cannot be directly performed with the input data.

Figure 8.16 shows the connection between microprocessor and I/O devices. The I/O devices are identified by port addresses. The I/O read and write operations are performed by using software instructions such as IN and OUT. The I/O read and write operations are controlled by control signals—IOR and IOW respectively. In this scheme port address are varied from 00H to FFH. Therefore, 256 I/O devices may be connected with microprocessor in I/O mapped I/O devices.

Figure 5 shows the address decoding technique of 8085 microprocessor. $A_0 - A_{10}$ are used for addressing the EPROM IC. $A_{11} - A_{15}$ address lines are applied to a NAND gate to generate the chip select signal $\overline{CS}$. The memory map of EPROM is given below:

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	= 0000H
⋮																
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	= 03FFH

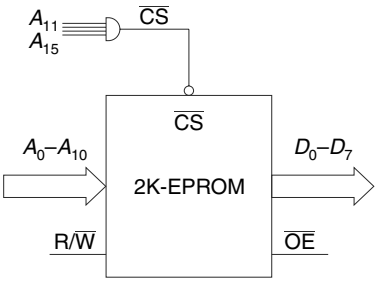


Fig. 5 Address decoding of 2K EPROM

Figure 5 shows the complete memory and address decoder circuit. In this case, a 3 line to 8-line decoder can be used to select any one output. Based on inputs at A_{11}, A_{12}, A_{13} any one output of $O_0 - O_7$ will be active low and other output lines remain high. The output lines are connected to the chip select of memory ICs. It is depicted in Fig. 6 that O_0 is connected with the chip select of 2K EPROM and O_7 is connected with the 2 K RAM. The output lines and corresponding memory address capability is given in Table 1.

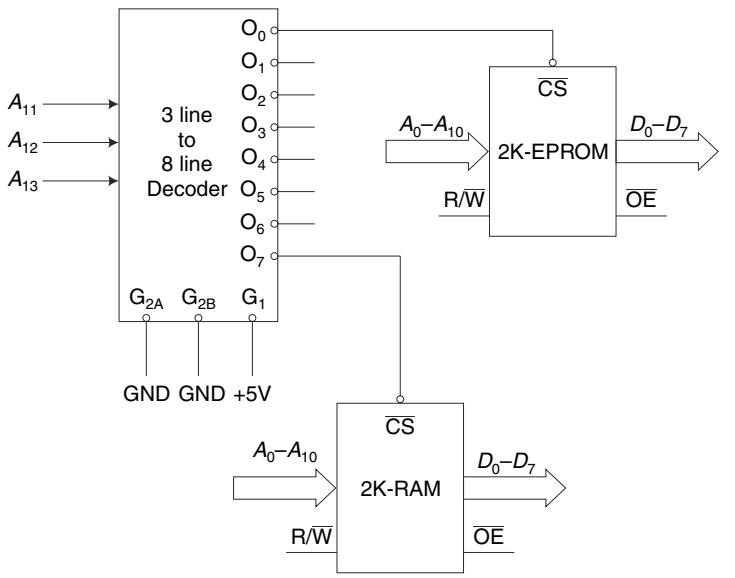


Fig. 6 Address decoding EPROM using decode

Table 1 Memory address selected by the decoder

Output lines	*Memory address*
O_0	0000H–07FFH
O_1	0800H–0FFFH
O_2	1000H–17FFH
O_3	1800H–1FFFH
O_4	2000H–27FFH
O_5	2800H–2FFFH
O_6	3000H–37FFH
O_7	3800H–3FFFH

11. (a) Write short notes on any *three* of the following: 3× 5

(a)Addressing modes of 8051 microcontroller

Refer Section 8.2.

(b) 8259 interrupt controller

Refer Sections 14.2 and 14.3.

(c) BSR mode of 8255

Refer Section 12.4.4.

(d) MIN/MAX mode operation of 8086 microprocessor

Refer Solution of 2008 WBUT Question No. 10(d).

Refer Sections 9.7.1, 9.7.2 and 9.7.3.

(e) Hardware interrupt of 8085 CPU

Refer Sections 6.2, 6.3 and 6.4.

Solution of 2011 WBUT Paper

GROUP–A
(Multiple-Choice Questions)

1. Choose the correct alternatives for any *ten* of the following: (10 × 1 = 10)

(i) In 8085 the addressable memory is
(a) 64 kB (b) 1 MB (c) 4 kB (d) 16 kB

(ii) The addressing mode of the instruction LDA address is
(a) combined (b) implied (c) register (d) direct

(iii) The instruction XCHG exchanges the contents of
(a) ACC and HL pair (b) BC pair and HL pair
(c) DE pair and HL pair (d) HL pair and memory location

(iv) Machine cycles for IN instruction are
(a) 6 (b) 5 (c) 4 (d) 3

(v) The content of the accumulator is 08H. Then the XRI 80H instruction was executed. The content of the accumulator is
(a) 80H (b) FFH (c) 88H (d) 08H

(vi) RST 7.5 interrupt is
(a) vectored and maskable (b) non-vectored and maskable
(c) non-vectored and non-maskable (d) vectored and non-maskable

(vii) When a subroutine is called, the address of the instruction next to CALL is saved in
(a) stack pointer (b) program counter
(c) stock (d) combination of flag and AX register

(viii) An 8K × 8 ROM holding the monitor program in a microprocessor trainer kit has the end address
(a) 8000 H (b) 4000 H (c) 1FFF H (d) 3FFF H

(ix) How many address lines are there in 8086 microprocessors?
(a) 16 (b) 8 (c) 20 (d) 12

(x) The total I/O space available in 8085 if peripheral mapped I/O is used is
(a) 64 (b) 128 (c) 256 (d) 512

(xi) 8251 is a
(a) USART IC (b) counter (c) interrupt controller (d) none of these

(xii) A single instruction to clear the lower four bits of the accumulator in 8085 microprocessor is
(a) XRI 0F H (b) ANI F0 H (c) ANI 0F H (d) XRI F0H

Solution

(i) (a) 64 kB (ii) (d) direct (iii) (c) DE pair and HL pair
(iv) (d) 3 (v) (c) 88H (vi) (a) vectored and maskable
(vii) (a) stack pointer (viii) (c) 1FFF H (ix) (c) 20
(x) (c) 256 (xi) (a) USART IC (xii) (b) ANI F0 H

GROUP–B
(Short-Answer Type Questions)

Answer any *three* of the following. (3 × 5 = 15)

2. State the uses of any three special purpose registers available in 8085 microprocessor.
Refer Section 2.2.5.

3. Draw the timing waveform of op-code fetch machine cycle of 8085 microprocessor.
Refer Section 3.7.1.

4. Write a subroutine for 1-second delay using 8085 assembly-level instructions.
Using only one register in a delay loop, a limited time delay is generated. If very high time delay is required, a register pair will be used in place of a register. For example, a 16-bit operand is loaded in the DE register pair. Then DE register pair is decremented by one using DCX D instruction. The DCX instruction does not set the zero flag. Therefore, additional instructions are used for testing zero flag and JNZ instruction is executed only when the zero flag is set. To generate about 1 s delay, the typical instructions of the time delay loop using a register pair (LOOP-I) and a register (LOOP-II) are given below:

PROGRAM

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments	T state
8100	06, 0D		MOV	B, 0D		7
8102	11, 00, 80	LOOP II	LXI	D, 4000	Initialise the DE register pair	10
8105	1B	LOOP-I	DCX	D	Decrement DE register pair	5
8106	7B		MOV	A,E	Copy content of E register in accumulator	5
8107	B2		ORA	D	OR D with accumulator	4
8108	C2, 05, 81		JNZ	LOOP-I	Jump not zero to LOOP	10
810C	05		DCR	B		5
810D	C2, 02, 81		JNZ	LOOP II		10

The number of T states for execution of LOOP-I is

= (*T* states for DCX D + *T* states for MOV A, E + *T* states for ORA D + *T* states for JNZ) × Number of times LOOP is executed

= (5 + 5 + 4 + 10) × 16384 T states = 393216 T states

If the microprocessor clock frequency is 5 MHz, time delay in LOOP-I is equal to $T_L = T \times$ number of *T* states for execution of LOOP = $\frac{1}{5} \times 10^{-6} \times 393216$s = 78.6432 ms

Initially, assume the content of register B will be N. Then time delay for LOOP –II is equal to

Time delay for LOOP-I × N + T × N × (*T* states for LXI D,4000 + *T* states for DCR B + *T* states for JNZ LOOP II) = 78.6432 ms × N + $\frac{1}{5} \times 10^{-6} \times$ N × (10 + 5 + 10) s

Total time delay is

$$T_D = T \times T \text{ states for MOV B, } N + 157.2864 \text{ ms} \times N + \frac{1}{5} \times 10^{-6} \times N \times (10 + 5 + 10) \text{ s}$$
$$= \frac{1}{5} \times 10^{-6} + \times 7\text{s} + 78.6432 \text{ ms} \times \text{N} + \frac{1}{5} \times 10^{-6} \times \text{N} \times (10 + 5 + 10) \text{ s} = 1000 \text{ ms}$$
$$= 0.0014 \text{ ms} + 78.6432 \text{ ms} \times \text{N} + 0.005 \text{ N} = 1000 \text{ ms}$$

Therefore, N = 12.71 =0DH (approx)

5. (a) What are the functions of ALE, HOLD and READY signals?

Refer Section 2.3.

(b) Define machine cycle and instruction cycle.

Refer Pages 3.30 and 3.29.

6. (a) Give the bit configuration of 8085 flag register

Refer page 2.10.

(b) Write down the mode-0 control word of 8255A for the following:
PORT A = input, PORT B not used, PORT C(upper) = input, PORT C(lower) = output

The control word bits for the above definition of ports are as shown in Fig.1.
Bit. No. D_0 is set to 0, as the Port C_{lower} is an output port.
As Port B is not used, assume Port B as output and operates in mode 0.
Therefore, Bit No. D_1 is set to 0, as the Port B is an output port.
Bit No. D_2 is set to 0, as the Port B has to operate in Mode 0.
Bit No. D_3 is set to 1, as the Port Cupper is an input port.
Bit No. D_4 is set to 1, as the Port A is an input port.
Bit No. D_5 and D_6 are set to 00 as the port A has to operate in Mode 0.
Bit No. D_7 is set to 1, as the ports A, B and C are used as simple input/output ports.
Thus the control word for above operation is 98 H.

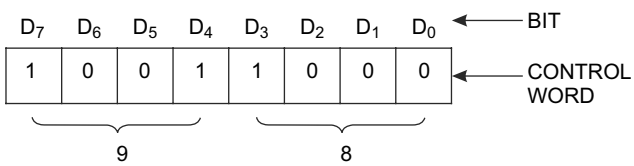


Fig.1

GROUP – C
(Long-Answer Type Questions)

Answer any *three* of the following: (3 × 15 = 45)

7. (a) What are different interrupts in 8085? Give their locations. Distinguish between maskable and unmaskable interrupts.

Refer Sections 6.3 and 6.2.

(b) After the execution of RIM instruction, the accumulator contains 49H. Explain the accumulator contents.

The RIM instruction loads the accumulator with 8 bits, which consists of the status of the interrupt mask, the interrupt, enable, the pending interrupts and one bit of serial input data. Figure 2 shows the accumulator content for RIM instruction

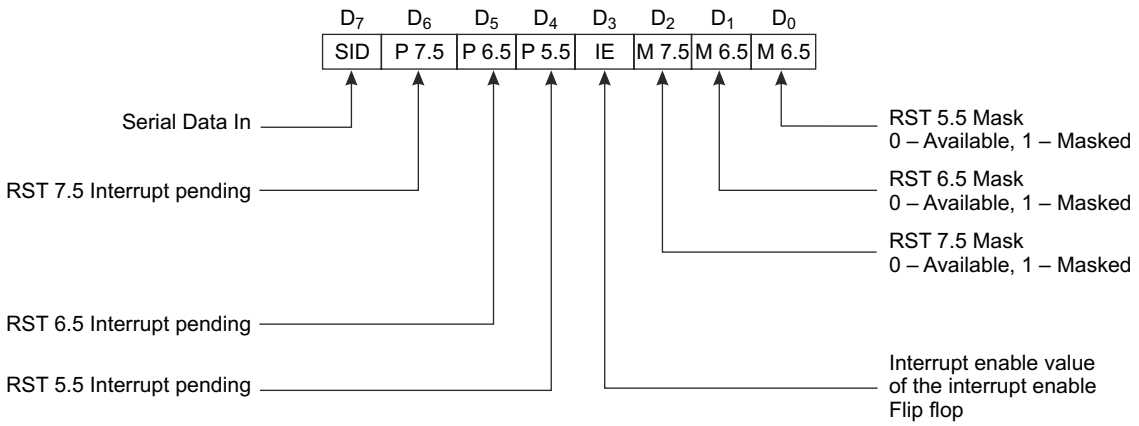


Fig. 2 Accumulator content for RIM instruction

INTERRUPT MASK BITS D_0, D_1, and D_2: Bits D_0 D_1 and D_2 represent the current setting of the mask for each of RST 7.5, RST 6.5 and RST 5.5. A high level shows that interrupt is masked and low level means that interrupt is not masked. Bits D_0 D_1 and D_2 return the contents of the three mask flip-flops. These bits can be used by a program to read the mask settings in order to modify only the right mask.

INTERRUPT ENABLE BIT D_3: Bit D_3 is the Interrupt Enable flag. This bit shows whether the maskable interrupt process is enabled or not. When it is high, interrupt is enabled. If it is low, interrupt is disabled. It returns the contents of the Interrupt Enable flip-flop. It can be used by a program to determine whether or not interrupts are enabled.

INTERRUPTS PENDING BITS D_4, D_5, D_6: Bits D_4, D_5 and D_6 represent the pending interrupts. Bits D_4 and D_5 return the current values of the RST 5.5 and RST 6.5 pins. Bit D_6 returns the current value of the RST 7.5 memory flip-flop. A high level on bits D_4, D_5 and D_6 states that interrupt are pending. A low level on bits D_4, D_5 and D_6 states that interrupts are not pending.

SERIAL INPUT DATA BIT D_7: Bit D_7 is used for serial data input. The RIM instruction reads the value of the SID pin on the microprocessor and returns it in this bit.

After execution of RIM instruction, the accumulator content is 49H. Therefore, $D_7 = 0$, $D_6 = 1$, $D_5 = 0$, $D_4 = 0$, $D_3 = 1$, $D_2 = 0$, $D_1 = 0$ and $D_0 = 1$. Since $D_0 = 1$, RST 5.5 is masked. As $D_1 = 0$ and $D_2 = 0$, RST 6.5 and RST 7.5 are available. As $D_3 = 1$, interrupt is enabled. Since $D_5 = 0$ and $D_4 = 0$, RST 5.5 and RST 6.5 interrupts are not pending. As $D_6 = 1$, RST 7.5 interrupts is pending. SID = 0, as $D_7 = 0$.

(c) Which interrupts are marked after the execution of the following instructions?
MVI A,1DH; SIM

Sometimes it is required to enable some selected interrupts and disable some other interrupts. The selected interrupts are enabling through the Set Interrupt mask. The accumulator (A) is loaded with the specified mask bits. The SIM instruction reads the accumulator content and enables and disables the interrupts.

The individual masks for RST 5.5, RST 6.5 and RST 7.5 are manipulated using the SIM instruction. This instruction takes the bit pattern in the accumulator. The SIM instruction reads the accumulator content and enables or disables the specific interrupts. Figure 3 shows the accumulator content for SIM instruction.

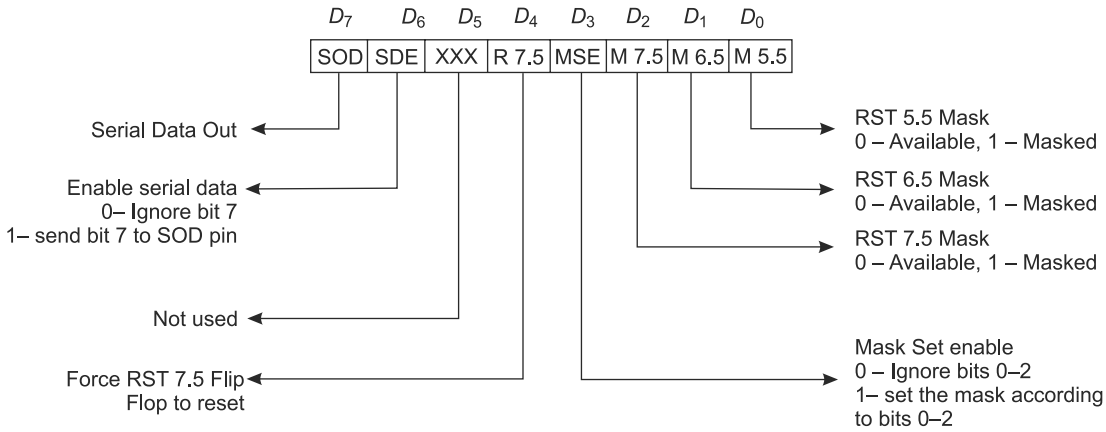


Fig. 3 Accumulator content for SIM

RST MASKS BITS D_0, D_1, and D_2 : Bit D_0 is the mask for RST 5.5, bit D_1 is the mask for RST 6.5 and bit D_2 is the mask for RST 7.5. If the mask bit is 0, the interrupt is available. If the mask bit is 1, the interrupt is masked. If bits D_0 or D_1 are set to 1, a signal applied to their respective pins causes no action. When D_0 or D_1 are set to 0, their respective bits will be visible through the RIM instruction, and the call to the interrupt vector will occur. In the case of bit D_2, the RIM instruction can indicate that RST 7.5 interrupt is pending, and an automatic call will not occur.

MASK SET ENABLE BIT D_3: Bit D_3 is Mask Set Enable (MSE) and this is an enable for setting the mask. If it is set to 0, the mask is ignored and the old settings remain. If it is set to 1, the new settings are applied. The SIM instruction is used for multiple purposes and not only for setting interrupt masks. It is also used to control functionality such as Serial Data Transmission. Therefore, bit D_3 is necessary to tell the microprocessor whether or not the interrupt masks should be modified.

RST 7.5 RESET BIT D_4: Bit D_4 is RST 7.5. The RST 7.5 interrupt is the only 8085 interrupt that has memory. If a signal on RST7.5 arrives while it is masked, a flip-flop will remember the signal. When RST7.5 is unmasked, the microprocessor will be interrupted even if the device has removed the interrupt signal. This flip-flop will be automatically reset when the microprocessor responds to an RST 7.5 interrupt. Bit 4 of the accumulator in the SIM instruction allows explicitly resetting the RST 7.5 memory even if the microprocessor did not respond to it.

UNDEFINED BIT D_5: Bit D_5 is not used by the SIM instruction

SOD ENABLE BIT D_6: Bit D_6 is used for serial output data enable.

SERIAL OUTPUT DATA BIT D_7: Bit D_7 is used for serial output data. The SIM instruction is used for serial data transmission. When the SIM instruction is executed, the content of bit D_7 of accumulator will be output on the SOD line.

After the execution of the following instructions

MVI A,1DH;
SIM

the content of accumulator is 1D H as shown below.

SOD	SDE	xxx	R7.5	MSE	M7.5	M6.5	M5.5
0	0	0	1	1	1	0	1

Since the content of accumulator is 1D H, it determines the following operation:

Disable RST 5.5	as bit $D_0 = 1$
Enable RST 6.5	as bit $D_1 = 0$
Disable RST 7.5	as bit $D_2 = 1$
Allow setting the masks	as bit $D_3 = 1$
Reset the flip-flop	as bit $D_4 = 1$
Bit 5 is not used	as bit $D_5 = 0$
Don't use serial data	bit $D_6 = 0$
Serial data is ignored	bit $D_7 = 0$

8. (a) Discuss the advantages and disadvantages of memory mapped I/O and I/O mapped I/O scheme. Which scheme is supported by the 8085 microprocessor and how?

Refer Section 5.8.

(b) Give the hardware and software to interface, one seven-segment display with 8085 µp whose address is FC23H.

Seven-segment display is widely used in calculators, digital watches, and measuring instruments, etc. Generally, Light Emitting Diode (LED) and Liquid Crystal Display (LCD) segments provide the display output of numerical numbers and characters. To display any number and character, the seven-segment display is most commonly used. Figure 4(a) shows the segment identification and display of decimal numbers 0 to 9 is given in Fig. 4(b). The

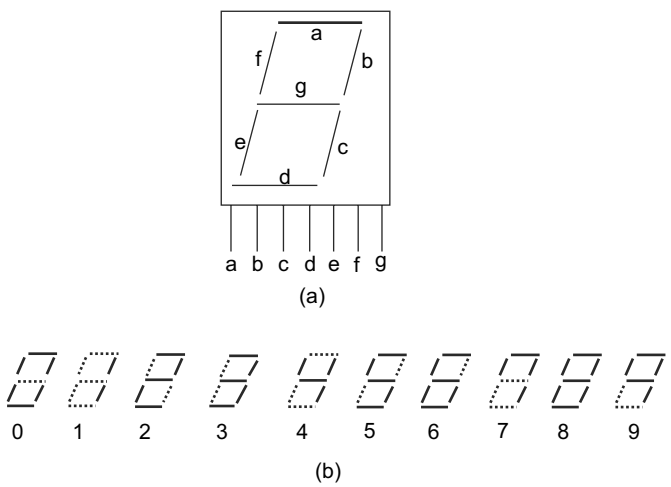


Fig. 4 (a) Segment identification, and (b) numerical displays

light emitting diodes emit light when the anode is positive with respect to the cathode. There are two possible connections, namely common anode and common cathode. In common anode connection, seven anodes connected to a common voltage and the cathode will be controlled individually to get the proper display. But in common cathode connection, anodes can be controlled individually for display when all cathodes are connected to a common ground of supply voltage as depicted in Fig. 5.

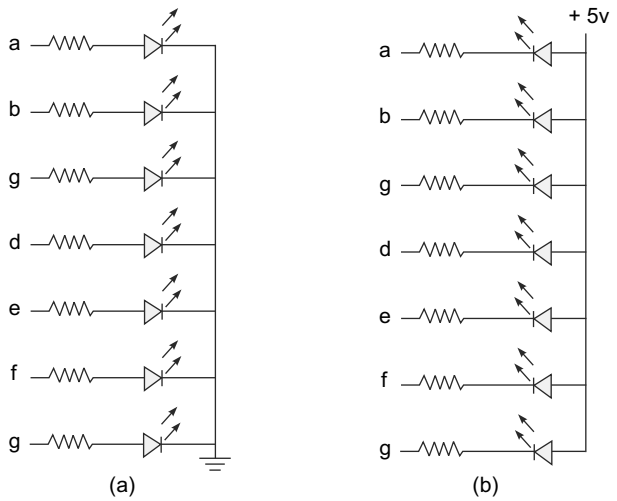


Fig. 5 (a) Common cathode connection (b) Common anode connection of seven-segment display

Figure 6 shows one seven-segment display unit interfaced to the 8085 microprocessor using a latch. The display code in hexadecimal for 0, 1, 2, 3, 4, 5, 6, 7 , 8, 9 are C0, F9, A4, B0, 99, 82, F8, 80 and 98 respectively. The port address FC23H is used to interface the display. The address of seven segment display unit is selected by $A_{15} = A_{14} = A_{13} = A_{12} = A_{11} = A_{10} = 1$, $A_9 = A_8 = A_7$

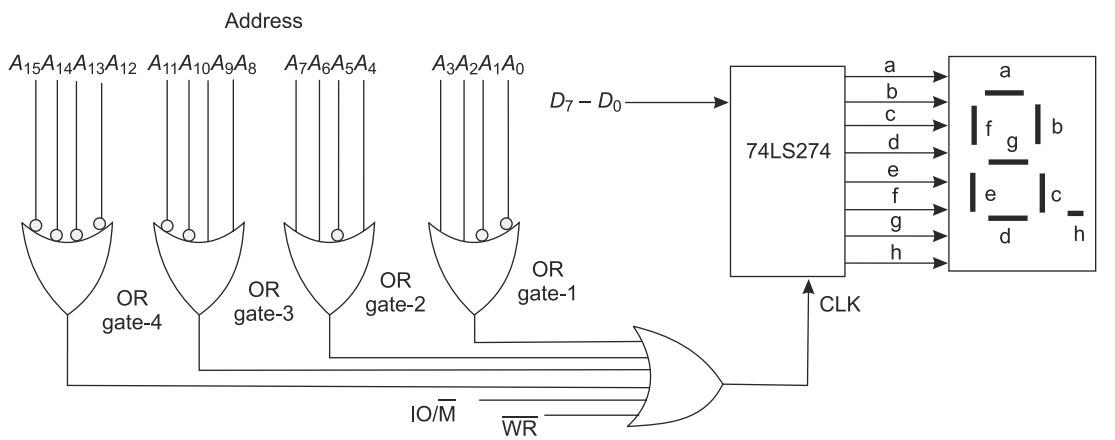


Fig. 6 Seven segment display whose address is FC23H

= $A_6 = 0$, $A_5 = 1$, $A_4 = A_3 = A_2 = 0$, $A_1 = A_0 = 1$, IO/$\overline{M}$ = 0 and $\overline{WR}$ = 0. In this interface circuit, $A_0 A_1 A_2 A_3$ are coded using OR gate-1, $A_4 A_5 A_6 A_7$ are coded using OR gate-2, $A_8 A_9 A_{10} A_{11}$ are coded using OR gate-3, $A_{12} A_{13} A_{14} A_{15}$ are coded using OR gate-4. After that the output of OR-gate1, OR-gate-2, OR-gate-3, and OR-gate-4 are ORed with IO/$\overline{M}$ and $\overline{WR}$, and a low output signal is generated. Therefore, the CLK to 74LS374 will be applied when the decoding logic input is FC23H. For any other address, the logic gates are not enabled and the seven-segment display is not active to latch the data.

If we want to display 0 in seven-segment display unit, the following instructions will be executed.

MVI A, C0H Load C0H into accumulator. The display code in hexadecimal for 0 is C0H.

LXI H, FC23H Load FC23H in HL Register pair. FC23H is the address of seven-segment display unit.

MOV M, A Content of accumulator send to address FC23H and 0 will be display on the seven segment display unit.

(c) Which addressing mode is used in the above scheme? What change is required if address of the display is FC H?

The above scheme is the memory mapped I/O scheme and the indirect addressing mode is used. If address of the display is FC H, I/O mapped I/O scheme will be used and the interface of seven segment-display through 74LS374 is depicted in Fig.7.

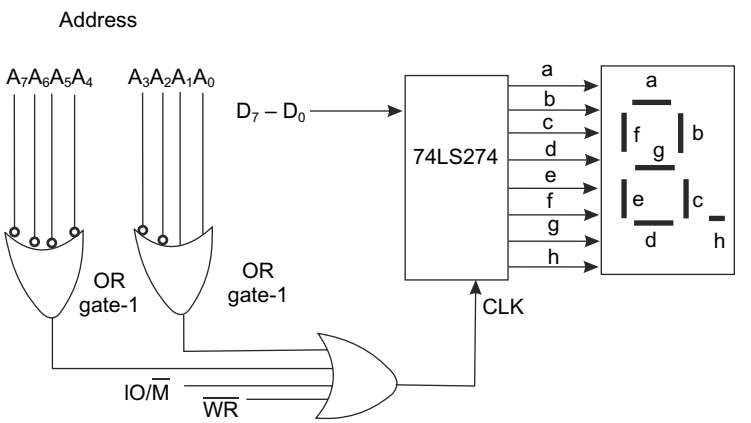


Fig. 7 Seven-segment display whose address is FCH

Since the port address FCH is used to interface the seven-segment display unit, the address of seven-segment display unit is selected by $A_7 = A_6 = A_5 = A_4 = 1$, $A_3 = A_2 = 1$, $A_1 = A_0 = 0$, IO/$\overline{M}$ = 0and $\overline{WR}$ = 0. In the above interface circuit, $A_0 A_1 A_2 A_3$ are coded using OR gate-1, and $A_4 A_5 A_6 A_7$ are coded using OR gate-2. When the output of OR-gate-1 and OR-gate-2 are ORed with IO/$\overline{M}$ and $\overline{WR}$, a low output signal is generated. Therefore, the CLK will be applied to 74LS374 when the decoding logic input is FCH.

If we want to display 0 in seven segment display unit, the following instructions will be executed.

MVI A, C0H Load C0H into accumulator. The display code in hexadecimal for 0 is C0H.

OUT FCH Content of accumulator send to port address FCH and 0 will be display on the seven segment display unit

9. (a) Describe the different addressing modes of 8086 microprocessor.
Refer Section 10.2.

(b) What are the main functions performed by BIU and EU unit of the 8086 microprocessor?
Refer Sections 9.2.1 and 9.2.2.

(c) How is pipeline achieved in 8086 microprocessor?
Refer Section 9.2.3.

10. Discuss the hardware and software of any microprocessor-based industrial applications.
Nowadays microprocessors are used to implement the traffic control system. Figure 8 shows the simple model of microprocessor-based traffic control system. The various control signals such as red, green, orange, forward arrow, right arrow and left arrow are used in this scheme. The forward, right and left arrows are used to indicate forward, right and left movement respectively. The red (R) signal is used to stop the traffic in the required lane and the yellow (Y) signal is used as standby, which indicates that the traffic must wait for the next signal. The green (G) light for a particular lane remains ON for DELAY-1 seconds followed by the stand by signal for DELAY-2 seconds. However, at a time 3 out of the four roads, the left signal or the left arrow remains ON even though that lane may have a red signal. The traffic light control is implemented using a 8085 microprocessor kit having 8255 on board and the interfacing circuit is illustrated in Fig. 9. Each signal is controlled by separate pin of I/O ports. The total number of logic signals required for this arrangement is twenty-four. The programmable peripheral interface device 8255 is used to interface these 24 logic signals with the lamps. The logic '0' and '1' represent the state of the lamp. Logic '1' represents ON and '0' represents OFF. All ports of 8255 are used as output ports. The control word to make all ports as output ports for Mode 0 operation is 80H. The traffic light control program can be written by the flowing steps:

Step-1 Initialize all ports of the 8255 as output ports.

Step-2 Determine the required status of port A, port B and port C of 8255 for north to south traffic movement. Load data into Accumulator and send to port A, port B and port C for north to South traffic movement.

Step-3 Call delay subroutine -1.

Step-4 Before starting east to west traffic movement, North to south traffic movement will be ready to stop and east to west traffic must be ready for movement. Therefore, determine the required status of port A, port B and port C for this operation. Then load data into accumulator and send to port A, port B and port C for north to south traffic movement will be ready to stop and east to west traffic must be ready for movement.

Step-5 Call delay subroutine-2

Step-6 For east to west traffic movement, determine the required status of port A, port B and port C of 8255. Load data into accumulator and send to port A, port B and port C for east to west traffic movement.

Step-7 Call delay subroutine-1.

Step-8 Prior to starting south to north traffic movement, east to west traffic will be ready to stop and South to North traffic must be ready for movement. For this operation determine the status of port A, port B and port C of 8255. Load required data into accumulator and send to port A, port B and port C for east to west traffic will be ready to stop and south to north traffic must be

ready for movement

Step-9 Call delay subroutine-2.

Step-10 Determine the status of port A, port B and port C for south to north traffic movement. Load required data into accumulator and send to port A, port B and port C for south to north movement.

Step-11 Call delay subroutine-1.

Step-12 Before starting west to east traffic movement, south to north traffic movement will be ready to stop and west to east traffic must be ready for movement. Find out the status of port A, port B and port C for this operation. Load required data into accumulator and send to port A, port B and port C for south to north traffic movement will be ready to stop and west to east traffic must be ready for movement

Step-13 Call delay subroutine-2.

Step-14 For west to east traffic movement, determine the status of port A, port B and port C of 8255. Load necessary data into accumulator and send to port A, port B and port C for west to east traffic movement.

Step-15 Call delay subroutine-1.

Step-16 Subsequently, west to east traffic movement will be ready to stop and north to south traffic must be ready for movement. Determine the status of port A, port B and port C for this operation. Load needed data into Accumulator and send to port A, port B and port C west to east traffic movement will be ready to stop and north to south traffic must be ready for movement

Step-17 Call delay subroutine-2.

Step-18 Jump to step-2.

Figure 8 shows the bit assignment of ports. Putting 0s and 1s in required position the data byte for each port can be derived. For example, during north to south traffic movement, the status of port A, port B and port C are as follows:

PA_7	PA_6	PA_5	PA_4	PA_3	PA_2	PA_1	PA_0
0	0	1	0	0	0	0	1
PB_7	PB_6	PB_5	PB_4	PB_3	PB_2	PB_1	PB_0
0	0	0	0	0	1	0	0
PC_7	PC_6	PC_5	PC_4	PC_3	PC_2	PC_1	PC_0
1	1	1	1	1	0	0	1

When north to south traffic movement will be ready to stop and east to west traffic must be ready for movement, the status of port A, port B and port C are as follows:

PA_7	PA_6	PA_5	PA_4	PA_3	PA_2	PA_1	PA_0
0	0	0	1	0	0	1	0
PB_7	PB_6	PB_5	PB_4	PB_3	PB_2	PB_1	PB_0
0	0	0	0	0	1	0	0
PC_7	PC_6	PC_5	PC_4	PC_3	PC_2	PC_1	PC_0
0	0	0	0	1	0	0	1

The calculated necessary data bytes of port A, port B and port C for all types of traffic movement are illustrated in a table as given below.

Table 1 Traffic Movement and Status of Ports

Traffic movement	Status of Port A	Status of Port B	Status of Port C
North to south traffic movement	21H	04H	F9H
North to south traffic movement be ready to stop and east to west traffic be ready for start	12H	04H	09H
East to west traffic movement	0CH	27H	89H
East to west traffic movement be ready to stop and south to north traffic be ready for start	94H	20H	08H
South to north traffic movement	64H	3CH	18H
South to north traffic movement be ready to stop and west to east traffic be ready for start	A4H	00H	14H
West to east traffic movement	24H	D0H	93H
West to east traffic movement be ready to stop and north to south traffic be ready for start	22H	00H	85H

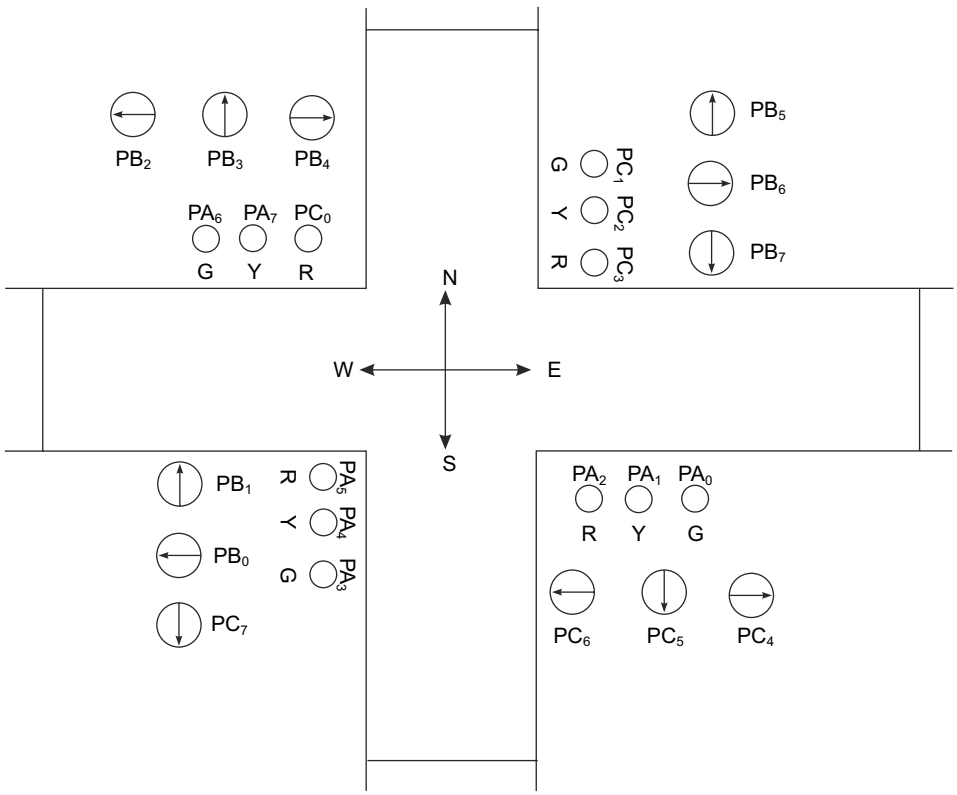


Fig. 8 Traffic light control

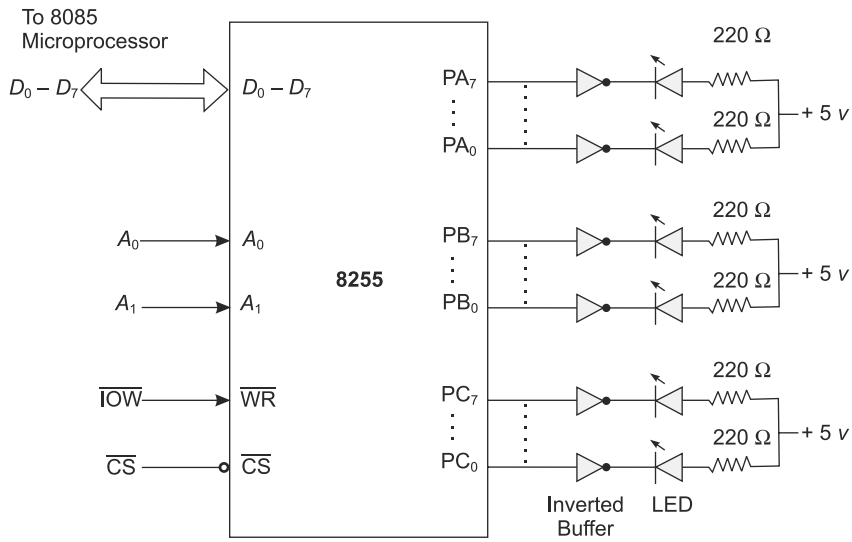


Fig. 9 The interfacing circuit for traffic light control

The program for traffic light control as follows:

PROGRAM for Traffic Light Control

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	3E, 80		MVI	A,80H	Load control word of 8255 in accumulator
8002	D3,0B		OUT	03H	Write control word in control word register and initialize ports
8004	3E, 21	START	MVI	A, 21H	Send 21H in Port A, F9H in Port C
8006	D3, 00		OUT	00H	and 04H in Port B for north to south
8008	3E, F9		MVI	A, F9H	traffic movement
800A	D3, 02		OUT	02H	
800C	3E, 04		MVI	A, 04H	
800E	D3, 01		OUT	01H	
8010	CD, 00, 81		CALL	DELAY_1	Delay-1 memory location is 8100
8013	3E, 12		MVI	A, 12H	Send 12H in Port A, 09H in Port C
8015	D3, 00		OUT	00H	and 04H in Port B for north to south
8017	3E, 09		MVI	A, 09H	traffic movement will be ready to
8019	D3, 02		OUT	02H	stop and east to west traffic
801B	3E, 04		MVI	A, 04H	movement is ready to start
801C	D3, 01		OUT	01H	
801E	CD, 00, 82		CALL	DELAY_2	Delay-2 memory location is 8200
8021	3E, 0C		MVI	A, 0CH	Send 0CH in Port A, 89H in Port C

8023	D3, 00	OUT	00H	and 27H in Port B for east to west
8025	3E, 89	MVI	A, 89H	traffic movement
8027	D3, 02	OUT	02H	
8029	3E, 27	MVI	A, 27H	
802B	D3, 01	OUT	01H	
802D	CD, 00, 81	CALL	DELAY_1	
8030	3E, 94	MVI	A, 94H	Send 94H in Port A, 08H in Port C
8032	D3, 00	OUT	00H	and 20H in Port B for east to west
8034	3E, 08	MVI	A, 08H	traffic movement will be ready to
8036	D3, 02	OUT	02H	stop and south to north traffic
8038	3E, 20	MVI	A, 20H	movement is ready to start
803A	D3, 01	OUT	01H	
803C	CD, 00, 82	CALL	DELAY_2	
803F	3E, 64	MVI	A, 64H	Send 64H in Port A, 18H in Port C
8041	D3, 00	OUT	00H	and 3CH in Port B for South to
8043	3E, 18	MVI	A, 18H	North traffic movement
8045	D3, 02	OUT	02H	
8047	3E, 3C	MVI	A, 3CH	
8049	D3, 01	OUT	01H	
804B	CD, 00, 81	CALL	DELAY_1	
804E	3E, A4	MVI	A, A4H	Send A4H in Port A, 14H in Port
8050	D3, 00	OUT	00H	C and 00H in Port B for south to
8052	3E, 14	MVI	A, 14H	north traffic movement is ready to
8054	D3, 02	OUT	02H	stop and west to east traffic
8056	3E, 00	MVI	A, 00H	movement will be ready to start
8058	D3, 01	OUT	01H	
805A	CD, 00, 82	CALL	DELAY_2	
805D	3E, 24	MVI	A, 24H	Send 24H in Port A, 93H in Port C
805F	D3, 00	OUT	00H	and D0H in Port B for west to east
8061	3E, 93	MVI	A, 93H	traffic movement
8063	D3, 02	OUT	02H	
8065	3E, D0	MVI	A, D0H	
8067	D3, 01	OUT	01H	
8069	CD, 00, 81	CALL	DELAY_1	
806C	3E, 22	MVI	A, 22H	Send 22H in Port A, 85H in Port
806E	D3, 00	OUT	00H	C and 00H in Port B for west to east
8070	3E, 85	MVI	A, 85H	traffic movement is ready to stop
8072	D3, 02	OUT	02H	and north to south traffic movement
8074	3E, 00	MVI	A, 00H	will be ready to to start
8076	D3, 01	OUT	01H	
8078	CD, 00, 82	CALL	DELAY_2	
807B	C3	JMP	START	

DELAY SUBROUTINE – 1 (DELAY_1)

Subprogram for Delay Subroutine - 1

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8100	11, 00, 80		LXI	D, 8000	Load suitable delay value in DE register pair
8103	1B	Level_1	DCX	D	Decrement the D-E register pair by 1
8104	7A		MOV	A, D	Move the content of D into A
8105	B3		ORA	E	OR operation between A and D
8106	C2, 03, 81		JNZ	Level_1	If DE is not equal to zero, jump to Level-1
8109	C9		RET		

DELAY SUBROUTINE – 2 (DELAY_2)

Subprogram for Delay Subroutine - 2

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8200	16, FF		MVI	D, FF	Load suitable delay value FF in D register
8202	15	Level-2	DCR	D	Decrement the D register by 1
8203	C2, 02, 82		JNZ	Level-2	If D is not equal to zero, jump to Level-2
8206	C9		RET		

11. Write notes on any three of the following:

(a) Synchronous mode of data transfer

Refer Sections 16.1, 16.2, Fig. 16.13, Fig. 16.14.

(b) Interrupt Service Subroutine

Refer Section 6.1.

(c) BSR mode of 8255 PPI

Refer Section 12.4.4.

(d) Designing I/O ports

8255 is a programmable peripheral interface IC and is a multi-port input/output device. This is a general-purpose programmable I/O device, which may be used with many different microprocessors. The 8255 has 24 I/O pins, which may be individually programmed in two groups of twelve input/output lines or three groups of eight lines. The two groups of I/O pins are called as Group A and Group B. Each group contains of a subgroup of eight bits known as 8 bit port and a subgroup of four bits known as 4 bit port. This IC has three eight bit ports: Port A: $PA_7 – PA_0$, Port B: $PB_7 – PB_0$, Port C: $PC_7 – PC_0$. The port C is divided into subgroup such as Port C upper, $PC_7 – PC_4$ and Port C lower, $PC_3 – PC_0$. The Group A consists of Port A and the port C upper. The Group B consists of Port B

and the port C lower. The I/O ports can be programmed in a variety of ways as per requirement of the programmer. The interface of 8255 PPI with 8085 microprocessor is shown in Fig.10. It is clear from Fig.10 that

The number of address lines required is 2, i.e. A_1 and A_0.

The chip select signal ($\overline{CS}$) is generated by address lines A_7, A_6, A_5, A_4, A_3, A_2 and IO / $\overline{M}$.

The decoding logic of Port A, Port B, Port C and Control word is given in Table 1.

Table 2

Address	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
Port A Address 40H	0	1	0	0	0	0	0	0
Port B Address 41H	0	1	0	0	0	0	0	1
Port C Address 42H	0	1	0	0	0	0	1	0
Control Word Address 43H	0	1	0	0	0	0	1	1

As per Table 2, the port address of Port A, Port B, Port C and Control word are 40H, 41H, 42H and 43H respectively.

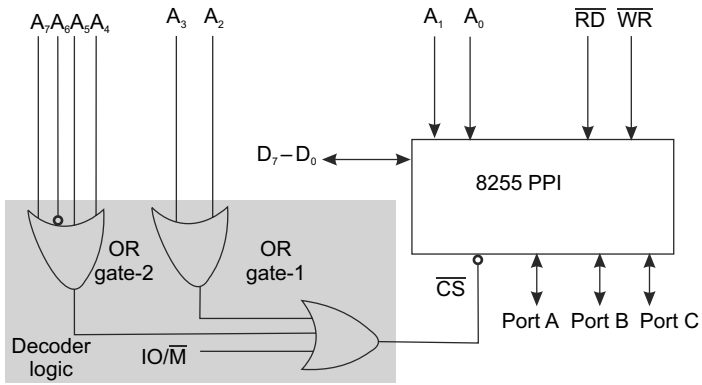


Fig. 10 8255 Interface with 8085 microprocessor

(e) Serial mode of operation using 8085 microprocessor

For serial mode of operation, the 8085 microprocessor has two serial input and output pins such as SID and SOD. SID represents serial input data and SOD represents serial output data. The SID and SOD pins are used to read/write one bit data to and from peripheral devices. There are two software instructions such as RIM and SIM which are associated with SID and SOD lines.

The Serial input data (SID) line exist inside the 8085 microprocessor as Pin number 5. One bit data can be externally read and stored using the SID line. The data which is read is stored in the A_7 bit of the accumulator

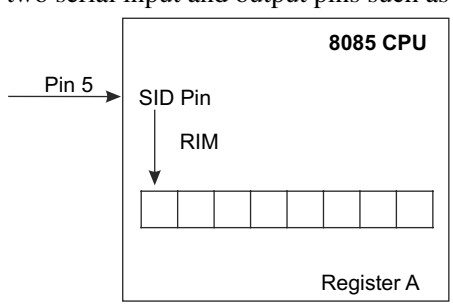


Fig. 11

whenever RIM instruction is executed. Figure11 shows the serial input data through SID pin and RIM instruction.

When the SID line is connected with +5 V and RIM instruction is executed, the Accumulator's MSB bit will be loaded with a logic 1 and the content of accumulator after the execution of RIM instruction is 80 H as depicted in Fig.12.

Fig. 12 Accumulator content after the execution of RIM instruction with SID = 1

If the SID is connected with 0 V (Ground) and RIM is executed, the Accumulator's MSB bit will be loaded with a logic 0. Then the content of Accumulator after the execution of RIM instruction is 00 H as shown in Fig.13.

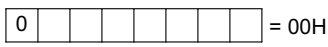


Fig. 13 Accumulator content after the execution of RIM instruction with SID = 0

The Serial Output Data (SOD) line exists inside the 8085 microprocessor as Pin number 4. One bit data can be externally written in this port. To write data into this port, SIM instruction must be executed. The data which is to be written in this port must be stored in the A_7 bit of the accumulator. The A_6 bit of the accumulator is called SOE (serial output enable) and this bit should be 1 to enable serial data output. Figure 14 shows the serial output data through SOD pin and SIM instruction.

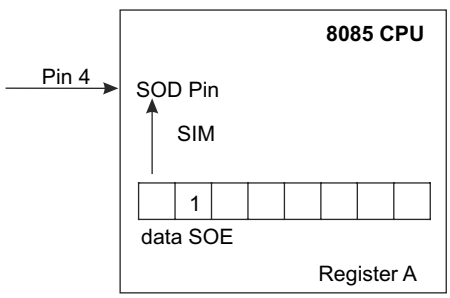


Fig. 14

To write 1 in the SOD line, load the accumulator with C0H and execute the SIM instruction as shown in Fig.15.

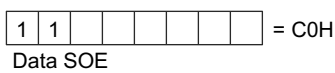


Fig. 15 Accumulator content after the execution of SIM instruction

To write 0 in the SOD line, load the accumulator with 40H and execute the SIM instruction as shown in Fig.16.

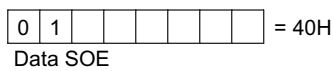


Fig. 16 Accumulator content after the execution of SIM instruction

Solution of 2011 WBUT Paper
EI(EC)502

GROUP–A
(Multiple-Choice Questions)

1. Choose the correct alternatives for any *ten* of the following: (10 × 1 = 10)

(i) The instruction XCHG exchanges the contents of
(a) ACC and HL pair (b) BC pair and HL pair
(c) DE pair and HL pair (d) HL pair and memory location

(ii) Machine cycles for IN instruction are
(a) 6 (b) 5 (c) 4 (d) 3

(iii) RST 7.5 interrupt is
(a) vectored and maskable (b) non-vectored and maskable
(c) non-vectored and non-maskable (d) vectored and non-maskable

(iv) When a subroutine is called the address of the instruction next to CALL is saved in
(a) stack pointer (b) program counter
(c) stock (d) combination of flag and AX register

(v) Which the BSR control word to set PC4?
(a) 09H (b) 07H (c) 04H (d) 05H

(vi) An 8K × 8 ROM holding the monitor program in a microprocessor trainer kit has the end address
(a) 8000 H (b) 4000 H (c) 1FFF H (d) 3FFF H

(vii) What will be the content of the accumulator and the status of CY flag after RLC operation, if the content of the accumulator is BCH and CY is 0?
(a) 79H, 1 (b) 78H, 1 (c) 5E H, 0 (d) 5D H, 0

(viii) How many address lines are there in 8086 microprocessors?
(a) 16 (b) 8 (c) 20 (d) 12

(ix) The total I/O space available in 8085 if used peripheral mapped I/O
(a) 64 (b) 128 (c) 256 (d) 512

(x) 8251 is a
(a) USART IC (b) counter (c) Interrupt controller (d) none of these

(xi) If the crystal with 8085 is 2MHz, the time required to execute an instruction of 20T states is
(a) 20 μs (b) 10 μs (c) 40 μs (d) 5 μs

(xii) A single instruction to clear the lower four bits of the accumulator in 8085 microprocessor is
(a) XRI 0F H (b) ANI F0 H (c) ANI 0F H (d) XRI F0H

Solution

(i) (c) DE pair and HL pair (ii) (d) 3 (iii) (a) vectored and maskable
(iv) (a) stack pointer (v) (a) 09H (vi) (c) 1FFF H
(vii) (a) 79H, 1 (viii) (c) 20 (ix) (c) 256
(x) (a) USART IC (xi) (b) 10 μs (xii) (b) ANI F0 H

GROUP–B
(Short-Answer Type Questions)

Answer any *three* of the following. (3 × 5 = 15)

2. Describe the addressing modes of 8085.
Refer Section 3.2.

3. (a) What are functions of ALE, HOLD and READY signals?
Refer Section 2.3.

(b) Differentiate between I/O mapped I/O and memory mapped I/O
Refer Table 5.6 (Page 5.15).

4. Calculate the total time delay for the following loop in the 8085 microprocessor, assuming the clock period is 0.5 microsecond

```
        LXI B, 238H     ;  10T
LOOP :  DCX B           ;  6T
        MOV A,C         ;  4T
        ORA B           ;  4T
        JNZ LOOP        ;  10/7 T
```

In the above instructions, LXI B, 238H is executed once and the other instructions (DCX B, MOV A,C, ORA B and JNZ) are executed for N times where N = 238H = 568_D.
The number of *T* states for execution of LOOP is

= N × *T* states for DCX B + N × T states for MOV A,C + N × *T* states for ORA B+ (N – 1) × *T* states for JNZ + 7

= 568 × 6 + 568 × 4 + 568 × 4 + (568 – 1)× 10 + 7 T states = 13629 *T* states

The total number of *T* states required for the above instructions are 10 + 13629 *T* states = 13639 *T* states
If the microprocessor clock period is 0.5 micro-second, total time delay in LOOP is equal to

= T × number of *T* states for execution of LOOP = 0.5 × 13639 μs = 6.8195ms (approx).

5. (a) Give the bit configuration of 8085 flag register.
Refer Fig. 2.10 (Page 2.10)

(b) Write down the mode-0 control word of 8255A for the following:
PORT A = input, PORT B not used, PORT C (upper) = input,
PORT C (lower) = output

Refer Solution of Question 6(b) of 2011 WBUT Solution (EI-502)

6. Draw the timing diagram of Memory Read machine cycle of 8085 microprocessor.
Refer Section 3.7.2.

GROUP – C
(Long-Answer Type Questions)

Answer any *three* of the following. (3 × 15 = 45)

7. (a) What are vectored and non-vectored interrupts?
Refer Sections 6.3 and 6.4.

Explain the instructions RIM and SIM .
Refer Sections 6.5.1 and 6.5.2.

Write an instruction to enable the RST 7.5, RST 6.5 and disable RST 5.5
Refer Solution of 2009 WBUT Question 11(b).

(b) Discuss how 8253 is used to generate square waves.
Refer Section 13.5.4.

(c) What is the difference between CALL and JMP instructions of 8085 microprocessor?
The branch group instructions are generally used to change the sequence of the program execution. There are two types of branch instructions, namely conditional and unconditional. The conditional branch instructions transfer the program to the specified address when condition is satisfied only. The unconditional branch instructions transfer the program to the specified address unconditionally. CALL and JMP instructions are examples of branch group instructions. The difference between CALL and JMP instructions are given below:

CALL	JMP
The specified format of Unconditional CALL instruction is **CALL 16-bit address** (Unconditional subroutine CALL) ([SP] – 1) ← PCH , ([ISP]-2)←PCL, [SP] ← [SP]-2, PC ← 16 bit address Machine cycles: 5, States: 9/18. The program sequence is transferred to the memory location specified by the 16-bit address given in the instruction. Before the transfer, the contents of the program counter (the address of the next instruction after CALL) are pushed onto the stack. Example: CALL 8700H	The specified format of Unconditional JUMP instruction is **JMP 16-bit address** (Jump Unconditionally) PC←Label (16-bit address) Machine cycles: 3, States 10 The program sequence is transferred to the memory location specified by the 16-bit address given in the operand. For example, in JMP 8000H, the program jumps to the instruction specified by the address location 8000H unconditionally.

(Contd.)

Table *(Contd.)*

In the conditional CALL instruction, the program sequence is transferred to the memory location specified by the 16-bit address given in the operand of the instruction based on the specified flag of the PSW. Before the transfer, the address of the next instruction after the call, or the contents of the program counter is pushed to the stack. The examples of conditional jump instruction are CC (Call on Carry), CNC (Call on no Carry), CP (Call on Positive), CZ (Call on Zero), CNZ (Call on no Zero), CPE (Call on Parity Even) and CPO (Call on Parity Odd)	In the conditional jump instruction; the program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW. The examples of conditional jump instruction are JC(Jump on Carry), JNC (Jump on no Carry), JP (Jump on Positive), JZ (Jump on Zero), JNZ (Jump on No Zero), JPE (Jump on Parity Even) and JPO (Jump on Parity Odd)

8. (a) Describe the different addressing modes of 8086 microprocessor.
Refer Section 10.2.

(b) What are the main functions performed by BIU and EU unit of 8086 microprocessor?
Refer Sections 9.2.1 and 9.2.2.

(c) How is pipeline achieved in 8086 microprocessor?
Refer Section 9.2.3.

9. (a) Discuss the memory organization of 8051 microcontroller.
Refer Section 7.3.

(b) What are the different interrupts available in 8051 microcontroller?
Refer Solution of Question 7(c) of 2010 WBUT Solution EI 405.

(c) Discuss the different addressing modes of 8051 microcontroller.
Refer Section 8.2.

10. Discuss the hardware and software of any microprocessor based industrial applications.
Electrical appliances or any electrical and electronics instruments always require protection against over and under voltage. The conventional relays are already used for the under and overvoltage, the maximum and minimum level of voltage are not changeable. Though a microprocessor-based system is high cost but the advantage of this system is that the same system may provide protection against maximum and minimum allowable current and voltage with a scope to adjust maximum and minimum limits.

The schematic block diagram of the system over voltage protection is shown in Fig.1. It is depicted in Fig.1 that a single-phase AC supply is connected to a load (electrical appliance) through an electromagnetic relay. This electrical appliance must be protected from overvoltage as well as undervoltage. For this, a potential transformer (PT) has been used to collect voltage signal. The output of PT is fed to the input of peak detector circuit to detect the peak value of the voltage. The output of peak detector circuit is applied to the A/D Converter for analog-to-digital conversion. For protection against over and undervoltage, an opto-coupler circuit, MCT2E is used to connect to the pin PB_0 of the I/O port. 5 V DC supply has been connected to the opto-coupler circuit and output has been connected to the energizing coil of an electromagnetic relay through a diode IN 4007.

In microprocessor-based protection, initially the upper and lower limiting values of voltage are stored in memory. Initialize the port A and port C upper as input ports and port B and port C lower as output ports. The microprocessor receives the output of A/D converter and compares the same with the upper and lower limiting values of voltage (V_{UL} and V_{LL}). Within the safe limit, the microprocessor sends 0 signal through PB_0 and relay is OFF and supply current flow through load. If voltage value is either less than V_{LL} or greater than V_{UL}, microprocessor send '1' signal through PB_0 and relay coil is energized. As the relay becomes ON, supply voltage is disconnected from system and the system will be protected. The flowchart of program is given in Fig. 2 and the assembly-language programming for voltage protection is given below.

PROGRAM for Voltage Protection

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
8000	21, 00, 81		LXI	H, 8100H	Initialise memory location 8100H
8003	3E, 50		MVI	A, 50H	Store digital equivalent of V_{LL} in
8005	77		MOV	M, A	8100H location
8006	23		INX	H	Increment HL register pair
8007	3E,70		MVI	A, 70H	Store digital equivalent of V_{UL} in
8009	77		MOV	M, A	8100H location
800A	3E, 98		MVI	A, 98H	Load control word of 8255-1 in accumulator
800C	D3, 03		OUT	03H	Write control word in control word register and initialize ports
800E	3E, 08	START	MVI	A, 08H	Send start of conversion signal through PC_3
8010	D3, 02		OUT	02H	PC_3 is high
8012	3E, 00		MVI	A, 00H	As PC_3 will be high for 1or two clock pulse, make it 0
8013	D3, 02		OUT	02H	PC_3 becomes low
8015	DB, 02	LOOP	IN	02	Read end of conversion signal
8017	17		RAL		Rotate accumulator to check either conversion is over or not
8018	D2,15, 80		JNC	LOOP	If conversion is not completed, jump to LOOP
801B	DB, 00		IN	00	Read digital output of A/D converter corresponding to voltage
801D	2F		CMA		Complement of ADC output
801E	D6, 80		SUI	80H	Subtract 80H

8020	4F	MOV	C, A	Store digital equivalent of voltage in C register
8021	21, 00, 81	LXI	H, 8100H	Load 8100H in HL-register pair
8024	7E	MOV	A, M	Move digital equivalent of V_{LL} voltage into accumulator from 8100H location
8025	B9	CMP	C	Compare C with V_{LL}
8026	DA, 32, 80	JC	TRIP	If carry flag is set, jump to trip
8029	79	MOV	A, C	Move content of C register to accumulator
802A	23	INX	H	Increment HL register pair
802B	BE	CMP	M	Compare the content of Memory V_{UL} with accumulator
802C	DA, 32, 80	JC	TRIP	If carry flag is set, jump to trip
802F	C3, 0E, 80	JMP	START	Jump to start
8032	3E, 01 TRIP	MVI	A, 01H	
8034	D3, 01	OUT	01H	Send $PB_0 = 1$ and trip the circuit
8035	76	HLT		Stop

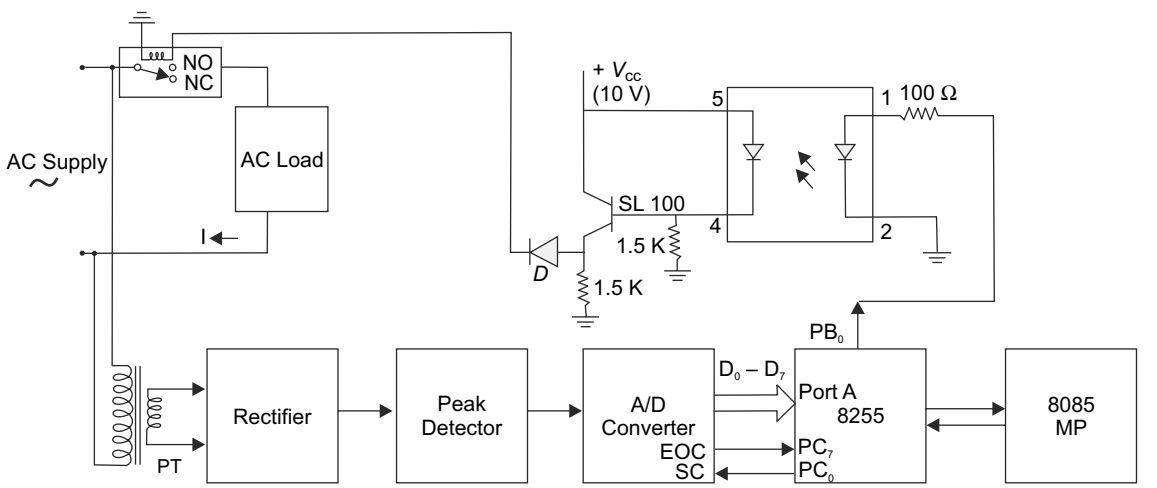


Fig. 1 Schematic block diagram of overvoltage protection

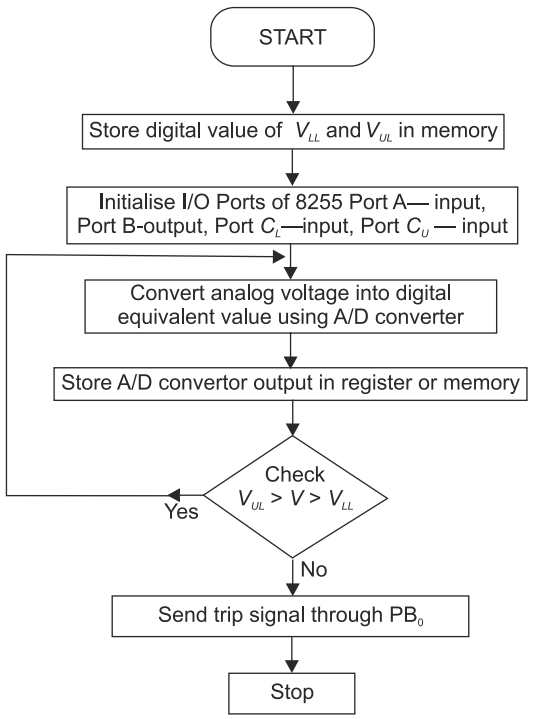


Fig. 2 Flow chart for voltage protection

11.Write notes on any three of the following:

(a) Synchronous mode of data transfer
Refer Sections 16.1, 16.2, Fig. 16.13, 16.14.

(b) Serial mode of operation using 8085 microprocessor
Refer Solution of 2011 WBUT (EI502) Question No. 11 (e).

(c) Interfacing memory with a microprocessor
Refer Section 5.8, Fig. 5.17 and Fig. 5.18.

(d) Designing I/O ports
Refer Solution of 2011 WBUT (EI502) Question No. 11 (d).

(e) Interrupt Service Subroutine
Refer Section 6.1.

Solution of 2012 WBUT Paper (EI402)

GROUP–A

(Multiple-Choice Questions)

1. Choose the correct alternatives for any *ten* of the following: (10 × 1 = 10)

(i) Whenever the PUSH instruction is executed in case of 8085 CPU, the stack pointer is
(a) decremented by 1 (b) decremented by 2
(c) incremented by 1 (d) incremented by 2

(ii) A single instruction to clear the lower four bits of the accumulator in the 8085 microprocessor is
(a) XRI 0FH (b) ANI F0H (c) ANI 0FH (d) XRI F0H

(iii) Machine cycle in "CALL" instruction are
(a) 6 (b) 5 (c) 4 (d) 3

(iv) Address lines required for 32k-byte memory chip is
(a) 13 (b) 14 (c) 15 (d) 16

(v) The addressing mode used in the instruction of LDAX B
(a) register (b) immediate (c) register-indirect (d) direct

(vi) Tri-state buffers are often used to make sure the unselected devices have their data outputs placed in the
(a) Logic 1 state (b) high-impedance state
(c) Logic 0 state (d) input stage

(vii) To perform Handshake I/O using 8255 PPI, which mode should be chosen?
(a) Mode 0 (b) Mode 1
(c) Mode 2 (d) Any of (a) and (b)

(viii) For Opcode fetch machine cycles the status signals of 8085 microprocessor is
(a) $IO/\overline{M} = 0, S_1 = 0, S_0 = 1$ (b) $IO/\overline{M} = 1, S_1 = 1, S_0 = 0$
(c) $IO/\overline{M} = 0, S_1 = 1, S_0 = 1$ (d) $IO/\overline{M} = 1, S_1 = 0, S_0 = 1$

(ix) Which one of the following is the non-vectored interrupt of 8085A microprocessor?
(a) RST 7.5 (b) EI (c) INTR (d) TRAP

(x) The content of "HL" register pair is 204AH. What will be the content of "HL" register pair after executing the instruction DAD H?
(a) 204A (b) 4096 (c) 4094 (d) 2096

(xi) Whenever the POP instruction is executed, the stack pointer is
(a) decremented by 1 (b) decremented by 2
(c) incremented by 1 (d) incremented by 2

(xii) When the instruction LHLD is executed, number of T-states required are
(a) 10 (b) 14 (c) 13 (d) 15

(xiii) SP register holds the
(a) base address of stack
(b) address of stack top

(c) address of the instruction to be fetched
(d) none of these

(xiv) Mode-2 of 8254 is
(a) square wave generator (b) rate generator
(c) software trigger strobe (d) hardware trigger strobe

Solution

(i) (b) decremented by 2 (ii) (b) ANI F0H (iii) (b) 5
(iv) (c) 15 (v) (c) register-indirect (vi) (b) High–impedance state
(vii) (b) Mode 1 (viii) (c) $IO/\overline{M} = 0, S_1 = 1, S_0 = 1$ (ix) (c) INTR
(x) (c) 4094 (xi) (d) incremented by 2 (xii) (d) 15
(xiii) (b) address of stack top (xiv) (b) rate generator

GROUP–B
(Short–Answer Type Questions)

Answer any *three* of the following. (3 × 5 = 15)

2. (a) What purpose does "READY" signal serve in Intel-8085 microprocessor?
Refer Section 2.3, Page 2.16.

(b) Describe the bit assignment of the Flag register in the 8085 microprocessor.
Refer Section 2.2.5, Page 2.10, Fig.2.10.

3. (a) Define instruction cycle, machine cycle and T-state.
For instruction cycle, Refer Section 3.6, Page 3.29
For machine cycle, Refer Section 3.6.3, Page 3.30
For T-state, Refer Section 3.6, Page 3.29

(b) What will be the content of a DE register pair at the end of the program?

```
LXI SP, 2000H
LXI  H, 1000H
DAD  SP
XCHG
HLT
```

After execution of the instruction LXI SP, 2000H, 2000H should be loaded in stack pointer immediately. When the instruction LXIH, 1000H is executed, 1000H should be loaded in Hand L register pair immediately. After that if DAD SP instruction is executed, the content of the stack pointer will be added with the content of the H and L register pair. Hence the content of H and L register pair will be 3000H. When the instruction XCHG is executed, the content of D and E register pair and H and L register pair will be exchanged. Therefore, the content of the DE register pair at the end of the program will be 3000H.

4. (a) Differentiate memory mapped I/O and I/O Mapped I/O schemes
Refer Table 5.6, Page 5.15.

(b) What do you mean by non-maskable (NMI) and vectored interrupt?
Refer Sections 6.2, 6.3 and 6.4.

5. (a) What are the functions of program counter, stack pointer, and ALE signal?
For program counter and stack pointer, Refer Section 2.2.5, Page 2.10.
For ALE signal, Refer Section 2.3, Page 2.14.

(b) Write the control word format for I/O mode in 8255.
Refer Section 12.3 (Fig. 12.4), Page 12.4 and Section 12.5.

6. Draw the Timing diagram of MOV A,M instruction.
Refer Solution of 2010 WBUT (EI502) Question No. 5.

GROUP–C
(Long–Answer Type Questions)

Answer any *three* of the following. (3 × 15 = 45)

7. (a) The following block of data is stored in the memory locations from XX55H to XX5AH. Transfer the data to the locations XX80H to XX85H in the reverse order (e.g. the data byte 22H should be stored at XX85H and 37H at XX80H), Data(H) 22,A5, B2, 99, 7F, 37.

To transfer a block of data from one section of memory to the other section of memory, the following algorithm will be followed:

Algorithm

1. Store the address of number of data in HL register pair.
2. Load number of data in C register from memory.
3. Store the starting address of destination in DE register pair.
4. Increment HL register pair to get data from source.
5. Copy data from source to accumulator.
6. Exchange HL and DE register pair, store the content of accumulator in destination address.
7. Exchange HL and DE register pair.
8. Increment HL and decrement DE register.
9. Decrement C register.
10. If the C register is not zero, repeat steps 5 to 9.

Labels	Mnemonics	Operands	Comments
	MVI	C,06H	Load number of data 06 in C register immediately
	LXI	H,XX55	Store the address of number of data, XX55H in HL register pair
	LXI	D,XX85	Store the destination address in DE register pair
LOOP	MOV	A,M	Move data from source to accumulator
	XCHG		Exchange the content of HL and DE
	MOV	M,A	Store the content of accumulator, data in destination address
	INX	H	Increment source address
	DCX	D	Increment destination address
	DCR	C	Decrement C register
	JNZ	NEXT	If C is not zero, Jump to LOOP
	HLT		

Example

Input		Result	
ADDRESS	DATA	ADDRESS	DATA
XX55 H	22 H	XX85	37 H
XX56 H	A5 H	XX84	7F H
XX57 H	B2 H	XX83	99 H
XX58 H	99 H	XX82	B2 H
XX59 H	7F H	XX81	A5 H
XX5A	37 H	XX80	22 H

(b) Write an assembly-language program for packing and unpacking of any number.

A byte contains two nibbles. When two BCD digits are stored in each nibble, the number is called a packed BCD. Using packed BCD numbers, the memory can be utilized effectively. In an unpacked BCD, the BCD is stored in the lower nibble, but the upper nibble contains all 0's. For example, 45 is a packed BCD number whereas 04 and 05 are unpacked BCD numbers. For the conversion from packed BCD to unpacked BCD, the assembly-language program is given below:

Mnemonics	Operands	Comments
MVI	A, 45H	Load the packed BCD number 45 in accumulator immediately
MOV	B, A	Copy the content of accumulator into B register
ANI	0FH	Perform logical AND immediate 8-bit data i.e. 0FH with accumulator. The unpacked BCD number 05 is stored in Accumulator
MOV	E, A	Store the unpacked BCD number 05 in E register
MOV	A, B	Copy the content of B register into accumulator
ANI	F0H	Perform logical AND immediate 8-bit data i.e. F0H with accumulator. The content of accumulator is 40H.
RRC		Rotate accumulator right
RRC		Rotate accumulator right
RRC		Rotate accumulator right
RRC		Rotate accumulator right and the unpacked BCD number 04 is stored in accumulator
MOV	D, A	Store the unpacked BCD number 04 in D register
HLT		

(c) What are interrupts?

Refer Section 6.14.

(d) How many hardware interrupts are there?

There are five hardware interrupts, namely TRAP, RST 7.5, RST 6.5, RST 5.5 and INTR

(e) What are maskable and non-maskable interrupts?
RST 7.5, RST 6.5, RST 5.5 and INTR are maskable interrupt but TRAP is non-maskable interrupt non-maskable interrupt.

8. (a) Draw the timing diagram of the instruction STA 4000H and explain it.
Refer Example 3.2 in page 3.40 and page 3.41.

(b) A set of five 8-bit data is stored in five consecutive locations from XX00 to XX04. Write a program to arrange them in ascending order. (Choose XX as per your kit.)
Assume XX00 is equivalent to 9000. It is also assumed that a series of five 8-bit numbers such as F2H, 05H, 88H, 23H, 65H are stored in memory locations from 9000H to 9004H. Arrange the above numbers in ascending order and to be stored in 9000H to 9004H locations.

Algorithm

1. Store 05H, number of data to be arranged in C register and store number of comparisons in D register.
2. Initialise the memory location 9000H of first data.
3. Load the first data in accumulator from memory.
4. Increment HL register pair for addressing next data.
5. Load the next data in B register from memory.
6. Compare next data with accumulator. Store the largest number in accumulator and smallest number in memory.
7. Then next number is compared with accumulator and store the smallest number in memory and largest number in accumulator.
8. This process will continue, till comparison of all numbers have been completed. After completion of comparison of all numbers, the largest number in accumulator and store it in memory. In this way first process will be completed.
9. At the starting of second process, C register is decremented and store number of comparisons in D register. Then repeat Step 2 to Step 8. After completion of this process, largest number in 9004H and second largest number in 9003H.
10. C register is decremented and the next process starts, if the content of C register is not zero.

Labels	Mnemonics	Operands	Comments
	MVI	C,05H	Load count value of number of data in register C register
START	MVI	D,05	Load count for number of comparisons in D register
	LXI	H,9000H	Load memory location of first number
	MOV	A,M	First number in accumulator
LOOP	INX	H	Increment H-L register pair for addressing next number
	MOV	B,M	Copy next number in B register from memory
	CMP	B	Compare next number with accumulator
	JNC	LEVEL_1	If the content of accumulator > next number, Jump to LEVEL_1

	DCX	H	Decrement H-L register pair to locate the addressing for storing smallest number
	MOV	M,A	Store smallest of the two numbers in memory
	MOV	A,B	Move largest of the two numbers in accumulator from B register
	JMP	LEVEL_2	Jump to LEVEL_2
LEVEL_1	DCX	H	Store smallest of the two numbers in memory
	MOV	M,B	
LEVEL_2	INX	H	Increment H-L register pair by one
	DCR	D	Decrement D register to count for number of comparisons
	JNZ	LOOP	Jump zero to
	MOV	M,A	Place largest number in memory
	DCR	C	Decrement count value
	JNZ	START	Jump not zero to START
	HLT		Halt

Example

DATA			**RESULT**			
Memory location	**Data**	**Memory location**	**After first process**	**After second process**	**After third process**	**After fourth process**
9000	F2H	9000	05H	05H	05H	05H
9001	05H	9001	88H	23H	23H	23H
9002	88H	9002	23H	65H	65H	65H
9003	23H	9003	65H	88H	88H	88H
9004	65H	9004	F2H	F2H	F2H	F2H

9. (a) Write the accumulator bit pattern for RIM and SIM instructions.

Refer Section 6.5.1(Fig.6.8) and 6.5.2 (Fig.6.9).

(b) Write a program to set PC_4 and reset PC_7 lines using BSR mode in 8255.

Bit number 7 of control word for the 8255 can be used to differentiate between the I/O mode and the BSR mode. When it is set to 1, 8225 can operate as I/O mode. If it is set to 0, individual pins of Port C are set or reset as 8255 operate in BSR mode.

The BSR control word to set bit PC_4 is 0000 1001 = 09H as $D_0 = 1$, $D_1 = 0$, $D_2 = 0$, $D_3 = 1$, $D_4 = 0$, $D_5 = 0$, $D_6 = 0$ and $D_7 = 0$

The BSR control word to reset bit PC_7 is 0000 1110 = 0DH as $D_0 = 0$, $D_1 = 1$, $D_2 = 1$, $D_3 = 1$, $D_4 = 0$, $D_5 = 0$, $D_6 = 0$ and $D_7 = 0$.

The assembly-language program to set PC_4 and reset PC_7 lines using BSR mode in 8255 is given below:

Mnemonics	Operands	Comments
MVI	A,09H	Load 09H, i.e. the BSR control word to set bit PC_4 in accumulator.
OUT	03H	Write the control word 09H in control word register and PC_4 will be set
MVI	A,0DH	Load 0DH, i.e. the BSR control word to reset bit PC_7 in accumulator.
OUT	03H	Write the control word 0DH in control word register and PC_7 will be reset

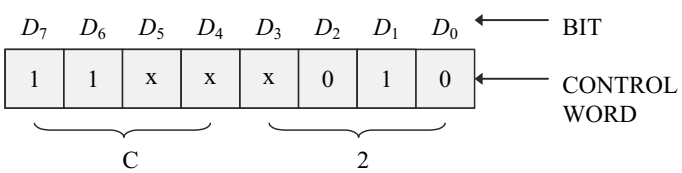
(c) Describe how 8255 is used as input port (both port A port B) in Mode 1 with handshake signals.

Refer Section 12.4.2, Fig.12.8 and Fig.12.9.

(d) Design the I/O control word bit pattern to set port A in Mode 2 and port B in mode 0 as input port.

In Mode 2, the bi-directional operation of port, i.e. both input and output capability is possible. Mode 2 operation is only feasible for Port A. Hence, Port A can be programmed to operate as a bi-directional port. When Port A is programmed in Mode 2, Port B can be used in either Mode 1 or Mode 0. In this mode of operation, PC_3 to PC_7 pins are used to control signals of Port A. As Port C (upper) pins are treated as control lines, D_3 will be in don't care state. As Port A is operated in Mode 0, PC_0, PC_1 , PC_2 pins are free to be used as either input or output. Hence D_0 of control word is assumed as 0.

When Port A of 8255 operates in Mode 2 and Port B operates in mode 0 as input Port, the control word bits are as shown in Fig.1.



Assume $X = 0$

Fig.1

Bit D_0 is set to 0, as Port C_{lower} is an output port.
Bit D_1 is set to 1, as Port operates as an input port.
Bit D_2 is set to 0, as Port B has to operate in Mode 0.
Bit D_3 is set to X, as Port C_{upper} pins are treated as control lines.
Bit D_4 is set to X, as Port A is an input port or output port.
Bit D_4 is set to 0, as Port A is an output port.
Bit D_6 and D_5 are set to 0X as the Port A has to operate in Mode 2.
Bit D_7 is set to 1, as ports A, B and C are used as simple input/output port.
Thus the control word for above operation is C2H when Port A of 8255 operates in Mode 2 and Port B operates in Mode 0 as input Port.

10. (a) Explain organization of a digital computer.

Refer Sections 1.2, 1.2.1, 1.2.2, 1.2.3, and 1.2.4.

(b) Differentiate between Harvard architecture and Von-Neumann architecture.

Von Neumann Architecture

Figure 2 shows the Von Neumann architecture of processors and this architecture is most commonly used in processors. In this architecture, one memory chip is used to store both instructions and data. The processor interacts with the memory through address and data buses to fetch instructions as well as data.

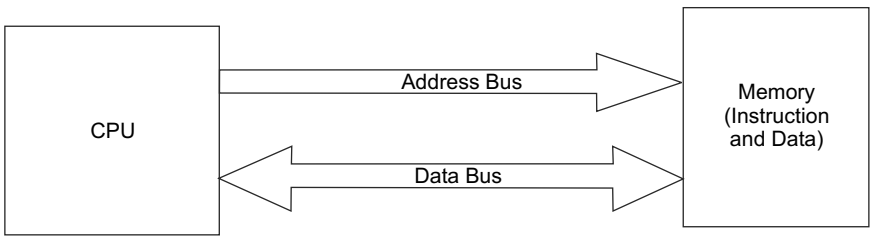


Fig. 2 Von Neumann architecture of processors

Harvard Architecture

Figure 3 shows the Harvard architecture of a processor. In this processor architecture, two separate memory blocks, namely, program memory and data memory are used. The program memory is used to store only instructions and data memory is used to store data. The program memory address bus is used to locate the program memory and through program memory data bus, the processor can write/read instructions to/from memory. Similarly, the data memory address bus is used to locate data memory and the data memory data bus can be used to access the data memory. Consequently, this architecture is efficient than Von Neumann architecture as the instructions and data will be accessed very fast.

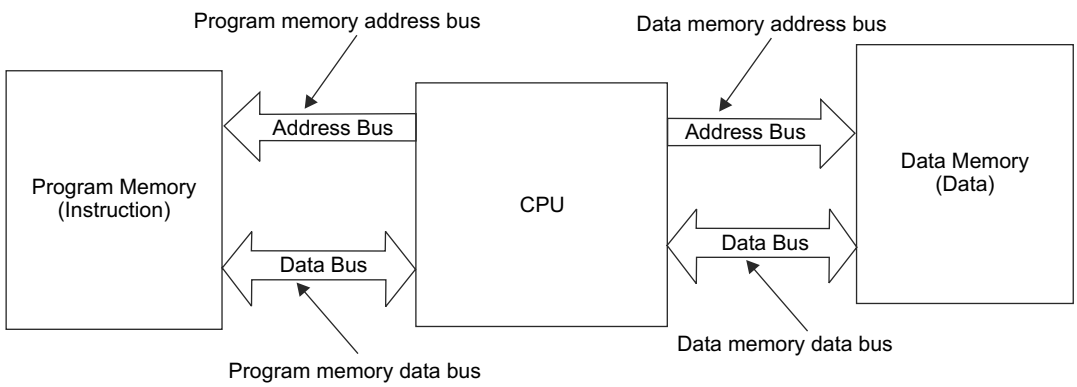


Fig. 3 Harvard architecture of a processor

(c) Describe the pipelining concept.

Refer Section 9.2.3.

(d) Differentiate between array processor and multiprocessor.

The difference between an array processor and multi-processor is given in Table 1.

Table 1 Difference between an array processor and multi-processor

Array processor	**Multi-processor**
An array processor can handle multiple data elements simultaneously in a parallel fashion.	A multiprocessor can handle multiple processes simultaneously which may include more than one data element in each process.
An array processor can be used for array operations in an optimized way, has its own set of instructions, large memory block moves, logical operations on many array elements, etc. and may itself be a multi-processor or massively parallel. An array processor has an interface where a host loads memory locations with the array to be processed (or perhaps a data file). The array processor uses its specialized structure to do what was asked of it on the array, then tell the host it is done and the result may be found in memory locations or perhaps a data file. Many of the jobs of supercomputers are done using array operations.	Multiple processors per system have long been used in systems that need a lot of processing power, like high traffic servers and when lots of computational power is needed. However, these systems have been expensive and are not needed by normal home or office users.
An array processor can also be smaller; a graphics processor handling the video display is an array processor. Typical operations such as *move the image to the right, move all the pixels to the right,* are done by an array processor.	A general-purpose computer may be a multi-processor or multi-core processor. In recent years, one processor has 2, 3, 4 or even 8 cores. One core can only do one task at a time. Multiple cores can run multiple processes at once in the real world.

11. Write short notes on any three of the following:

a. Synchronous mode of data transfer
Refer Sections 16.1, 16.2, Fig.16.13, Fig.16.14.

b. Interrupt service subroutine
Refer Section 6.1.

c. Handshaking mode of 8255
Refer Section 12.4.2.

d. Designing I/O ports
Refer Solution of 2011 WBUT (EI502) Question 11(d).

c. Pipeline hazards
Refer Section 9.2.3.

Pipelining can be used efficiently to increase the performance of a processor by overlapping execution of instructions. However, the efficiency of the pipelining depends upon how the problem encountered during the implementation of pipelining is handled. These problems are known as pipeline hazards. There are three types of pipeline hazards, namely

- Structural Hazards (Resource Bound)
- Control Hazards (Pipelining Bubbles)
- Data Hazards (Data dependencies)

Structural Hazards During the pipelining of processors, the overlapped execution of instructions requires pipelining of functional units and duplication of resources to allow all possible combinations of instructions in the pipeline. When some combination of instructions cannot be accommodated because of a resource conflict, the machine is said to have a structural hazard. The structural hazards occur when two activities require the same resource simultaneously. The common instances of structural hazards developed are

(i) If some functional unit is not fully pipelined then a sequence of instructions using that un-pipelined unit cannot proceed at the rate of one per clock cycle

(ii) If some resource has not been duplicated enough to allow all combinations of instructions in the pipeline to execute

For example, a processor has shared a single-memory pipeline for data and instructions. When an instruction contains a data-memory reference (load-MEM), it will conflict with the instruction reference for a later instruction (instruction 3-IF) as shown in Fig.1.

	Clock cycle number							
Instr	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Load	IF	ID	EX	MEM*	WB			
Instr 1		IF	ID	EX	MEM	WB		
Instr 2			IF	ID	EX	MEM	WB	
Instr 3				IF*	ID	EX	MEM	WB

Fig. 1 Clock cycle number of Load, Instr-1, Instr-2 and Instr-3

To resolve the pipeline hazards, the pipeline will stall for one clock cycle when a data-memory access occurs. The effect of the stall is actually to occupy the resources for that instruction slot. Figure 2 shows how the stalls are actually implemented. Instruction-1 is assumed not be a data memory, otherwise instruction-3 cannot start execution because of structural hazard.

	Clock cycle number								
Instr	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Load	IF	ID	EX	MEM	WB				
Instr 1		IF	ID	EX	MEM	WB			
Instr 2			IF	ID	EX	MEM	WB		
Instr 3				stall	IF	ID	EX	MEM	WB

Fig. 2 Clock cycle number of Load, Instr-1, Instr-2 and Instr-3 to resolve pipeline hazards

Control Hazard This type of hazard is developed due to uncertainty of execution path, branch taken or not taken. The control hazard arises when an attempt is made to make a decision before condition is evaluated. As a result, when processor branches to a new location in the program, invalidating everything, the processor can load in the pipeline.

Data Hazards The data hazard occurs when the pipeline changes the order of read/write accesses to operands so that the order differs from the order seen by sequentially executing instructions on the un-pipelined processor. Data hazards are also called *data dependency*. The data dependency is the condition in which the outcome of the current operation depends on the outcome of a previous instruction that has not yet been executed to completion because of the effect of the pipeline. Actually, this type of hazard arises because of the need to preserve the order of the execution of instructions.

Solution of 2012 WBUT Paper (EC502)

GROUP–A
(Multiple-Choice Questions)

1. Choose the correct alternatives for any *ten* of the following:

(i) The instruction MOV A,B belongs to
(a) immediate addressing (b) directing addressing
(c) implied addressing (d) register addressing

(ii) In 8085, TRAP is
(a) always maskable (b) can't interrupt a service sub-routine
(c) use for temporary power failure (c) lowest priority interrupt

(iii) How many hardware interrupt requests can a single interrupt controller IC 8259A process?
(a) 8 (b) 15 (c) 16 (d) 64

(iv) In DMA operation, data transfer takes place between
(a) memory and CPU (b) CPU and I/O
(c) I/O and Memory (d) different CPUs

(v) The programmable interval timer is
(a) 8253 (b) 8251 (c) 8250 (d) 8275

(vi) How many flag registers are in 8051?
(a) 9 (b) 8 (c) 6 (d) 5

(vii) 8259 is a
(a) programmable DMA controller (b) programmable interval timer
(c) programmable interrupt controller (d) none of these

(vii) The interrupt masks in 8085 can set or reset by the instruction
(a) EI (b) DI (c) RIM (d) SIM

(ix) The vector address corresponding to software interrupt command RST 7 in 8085 a microprocessor is
(a) 0017 H (b) 0027 H (c) 0038 H (d) 0700H

(x) A microprocessor is said to be a 8-bit, 16-bit, etc. depending on its
(a) data bus (b) address bus (c) ALU (d) control bus

(xi) When a subroutine is called, the address of the instruction next to "CALL" is saved in
(a) stack pointer register (b) program counter
(c) stack (d) PSW

(xii) The number of register pairs of 8085 microprocessor are
(a) 3 (b) 4 (c) 2 (d) 5

Solution

(i) (d) register addressing (ii) (c) use for temporary power failure (iii) (a) 8
(iv) (c) I/O and Memory (v) (a) 8253 (vi) (d) 5
(vii) (c) programmable interrupt controller (viii) (d) SIM
(ix) (c) 0038H (x) (a) data bus (xi) (c) stack
(xii) (a) 3

GROUP–B
(Short-Answer Questions)

Answer any *three* of the following. (3 × 5 = 15)

2. Draw the timing diagram of OUT instruction.
Refer Section 3.7.5, Fig.3.14.

3. What do you mean by conditional and unconditional jump? Give examples.
Refer Section 3.5.4.

4. What is the function of DAD instruction in 8085 processor?
DAD Register pair (Add register pair to H and L registers)
H-L ← H-L + Register pair.
Machine cycles: 3, States: 10. Flags: CS. Register addressing one byte instruction.
The 16-bit contents of the specified register pair can be added to the contents of the H-L register pair and the sum is stored in the H and L registers. The contents of the source register pair cannot be modified. When the result is greater than 16 bits, the CY flag will be set and no other flags are affected. *Example:* DAD B. The instruction DAD B states that the contents of the B-C register pair will be added with the content of the H-L register pair.

Write the output if input is F0H.

```
LXI  H, 2050H
MOV A, M
CMA
ADI 01H
STA 2060H
```

After execution of the LXI H, 2050H instruction, 2050H will be loaded in the HL register pair. When the execution of MOV A,M instruction is completed, the content of the 2050H memory location will be copied into accumulator. Since the input is F0H, the content of accumulator will be F0H.

When the CMA instruction is executed, the content of accumulator will be complemented. Subsequently, the content of the accumulator is 0FH. 01H will be added with the accumulator when the instruction ADI 01H is executed. Hence, the content of accumulator is 0FH + 01H=10H. After completion of execution of instruction STA 2060H, the content of accumulator, i.e. 10H will be stored in the 2060H memory location.

5. What is the difference between SIM and RIM instructions?
Refer Sections 6.5.1 and 6.5.2.

6. Explain the memory segmentation scheme with reference to 8086 microprocessor.
Refer Solution of 2009 WBUT Question 5.

GROUP–C
(Long-Answer Questions)

Answer any *three* of the following. (3 × 15 = 45)

7. (a) What are the different addressing modes of the 8085 microprocessor? Explain with at least two examples for each.
There are five addressing modes of the 8085 microprocessor such as
- Direct addressing
- Register addressing
- Register indirect addressing
- Immediate addressing
- Implicit addressing

Refer Sections 3.2.1 and 3.2.2.

(b) Explain the function of RIM instruction.
Refer Section 6.5.2.

(c) Write a program to enable RST 6.5 and disable RST 7. 5, RST 5.5.
Initially determine the contents of the accumulator to enable RST 6.5 and disable RST 7. 5, RST 5.5

Disable RST 5.5	bit $D_0 = 1$
Enable RST 6.5	bit $D_1 = 0$
Disable RST 7.5	bit $D_2 = 1$
Allow setting the masks	bit $D_3 = 1$
Don't reset the flip-flop	bit $D_4 = 0$
Bit 5 is not used	bit $D_5 = 0$
Don't use serial data	bit $D_6 = 0$
Serial data is ignored	bit $D_7 = 0$

SOD	SDE	xxx	R7.5	MSE	M7.5	M6.5	M5.5
0	0	0	0	1	1	0	1

Content of accumulator is 0DH. The program for the above operation is given below:

PROGRAM					
Memory address	**Machine Codes**	**Labels**	**Mnemonics**	**Operands**	**Comments**
8000	EB		EI		Enable all interrupts
8001	3E, 0D		MVI	A, 0DH	mask to enable RST 5.5, and disable RST 7.5, RST 5.5
8003	30		SIM		Apply the settings RST masks

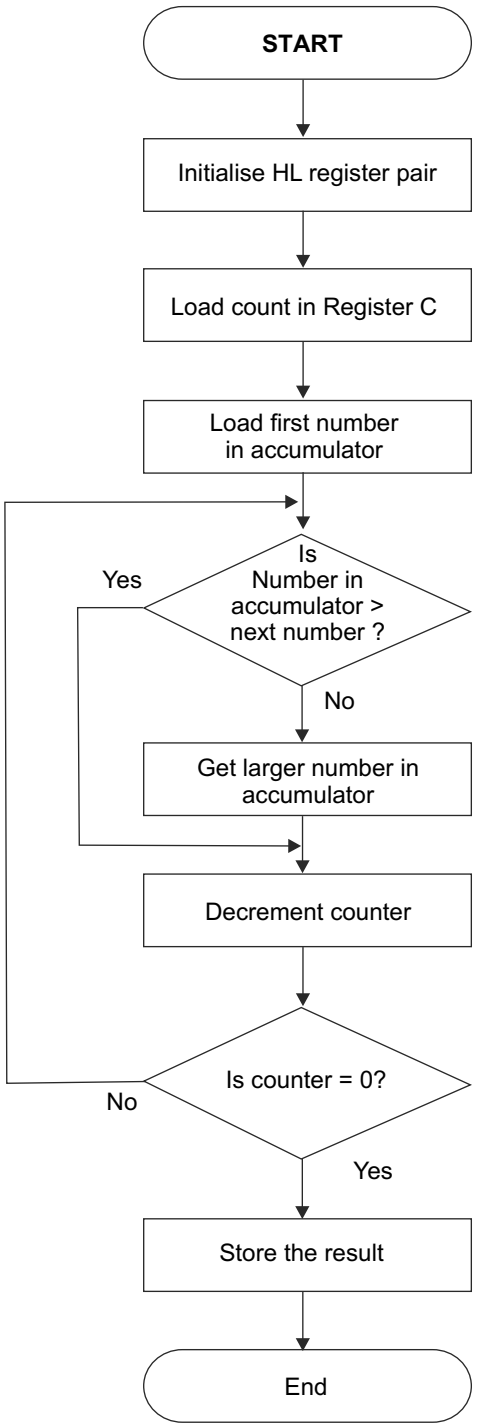


Fig. 1 Flow chart to find out the largest number from an array

8. a) With respect to 8237, explain the DMA operation.
Refer Solution 2009 WBUT Question 9(c).

b) What are the priorities of DMA request? Enumerate them.
Refer Section 17.3.1, Fig.17.6.

c) What are the major components of 8259A interrupt controller? Explain their functions.
The major components of 8259A interrupt controller are
- Interrupt Request Register (IRR)
- In-Service Register (ISR)
- Priority Resolver
- Interrupt Mask Register (IMR)
- Interrupt Control Logic
- Data Bus Buffer
- Read/Write Control Logic
- Cascade Buffer/Comparator

Refer Section 14.3.

9. (a) Draw and explain the timing diagram of the instruction IN 00H.
Refer Section 3.7.3.

(b) Write an ALP to find out the largest number from a given array of 10 numbers.
The count value of numbers 0AH is stored in the C register directly and the numbers are stored in the memory locations from 9001 to 900A. The largest number will be stored in the 900B location. Assume the program memory starts from 9100H. The flow chart to find out the largest number from an array is depicted in Fig.1.

Algorithm
1. Load count value of numbers 0AH in C register immediately.
2. Load the first number in accumulator from memory location 9001H.
3. Move the first number in accumulator.
4. Decrement the count value by one.
5. Move to next memory location for next data.
6. Compare the content of memory with content of accumulator.
7. If carry is generated, copy content of memory in accumulator.
8. Decrement the count value by one.
9. If count value does not equal to zero, repeat Step 5 to Step 8.
10. Store result in 900BH location.

PROGRAM

Memory address	Machine codes	Labels	Mnemonics	Operands	Comments
9100	0E, 0A		MVI	C,0A	Load count value in C register
9102	21, 01, 90		LXI	H,9001	Load address of first data in HL register pair
9105	7E		MOV	A,M	Copy first data in accumulator
9106	0D		DCR	C	Decrement C register

9107	23	LOOP	INX	H	Increment HL register for address of next data
9108	BE		CMP	M	Compare next data with the content of accumulator
9109	D2,0D,91		JNC	LEVEL	If carry is not generated, jump to LEVEL
910C	7E		MOV	A,M	Copy large number in accumulator from memory
910D	0D	LEVEL	DCR	C	Decrement C register
910E	C2, 07, 91		JNZ	LOOP	Jump not zero to LOOP
9111	32, 0B, 90		STA	900B	Store largest number in 900BH location
9114	76		HLT		

Example

DATA		RESULT	
Memory location	**Data**	**Memory location**	**Data**
9001	23H	900B	FFH
9002	FFH		
9003	47H		
9004	92H		
9005	10H		
9006	25H		
9007	67H		
9008	35H		
9009	98H		
900A	24H		

c) Differentiate between peripheral mapped I/O and memory mapped I/O.
Refer Section 5.8 and Table 5.6.

10. a) What do you mean by Mode 0, Mode 1 and Mode 2?
Refer Section 12.4.

b) Write down the control word for the following in Mode 0:

Port A = Input, Port B = Not used, Port C_U = Input, Port C_L = output

When the ports of 8255 operate in Mode 0 and Port A = Input, Port B = Not used, Port C_U = Input, and Port C_L = output, the control word bits are as shown in Fig.2.

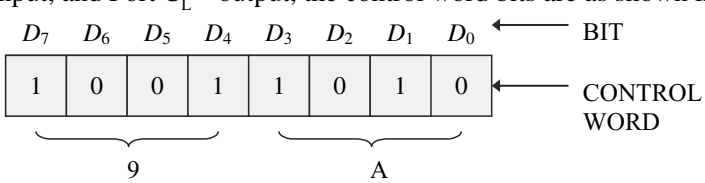


Fig. 2

Bit D_0 is set to 0, as Port C_{lower} is an output port.
Bit D_1 is set to 1, as Port B is not used. Assume Port B operates as an input port.
Bit D_2 is set to 0, as Port B has to operate in Mode 0.
Bit D_3 is set to 1, as Port C_{upper} is an input port.
Bit D_4 is set to 1, as Port A is an input port.
Bit D_5 and D_6 are set to 00 as Port A has to operate in Mode 0.
Bit D_7 is set to 1, as Ports A, B and C are used as simple input/output port.

Thus, the control word for above operation is 9A H.

(c) Write a BSR control word subroutine to set bits PC_7 and PC_3 and reset them after 10 ms. Assume that a delay subroutine is available and Hex address of Port A = 80H.

Refer Solution 2008 WBUT Question 9(c).

(d) Explain how bidirectional communication can be done between two computer, using 8255A.

The bidirectional communication between two 8086 microprocessors using two 8255 PPI is shown in Fig. 3 PC_3 of 8255(1) is connected with INTR of 8086(1) and PC_3 of 8255(2)is also connected with INTR of 8086(2). Port A of 8255(1) and Port A of 8255(2) ar connected together. The other PIN connections are depicted in Figure 3.

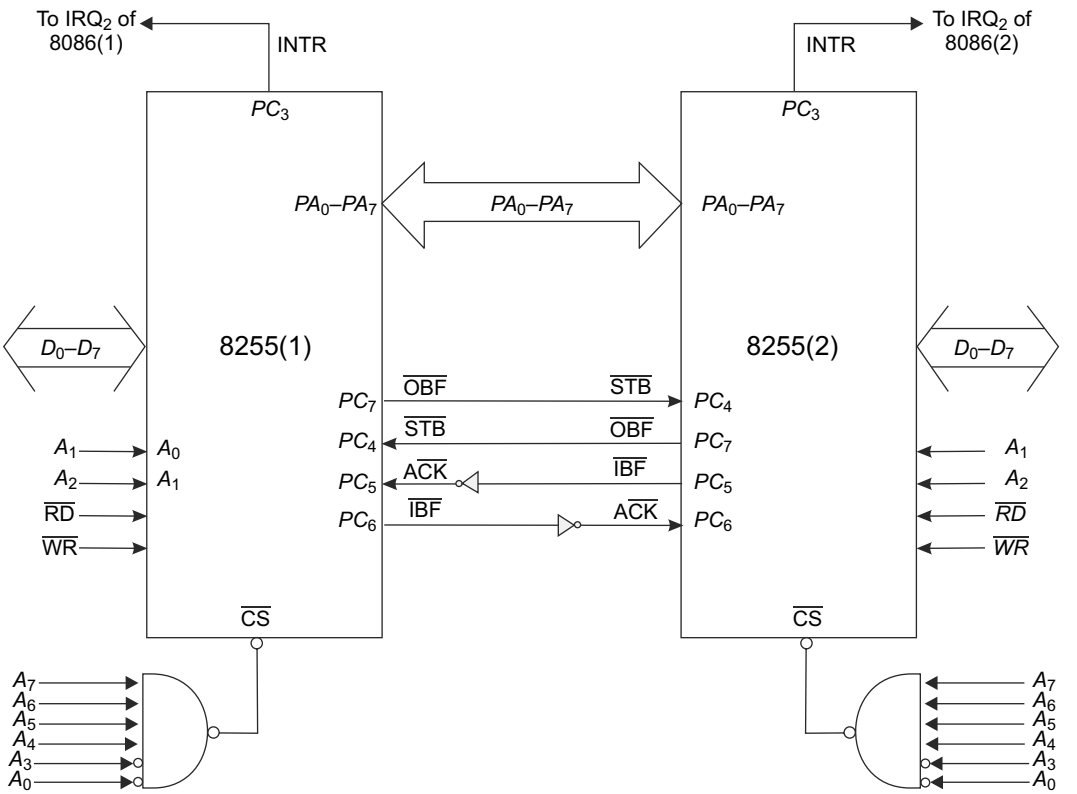


Fig. 3. Interconnection between two 8255 and two 8086 microprocessor

When the first processor is used as transmitter or recieve, other processor can be used as receiver or transmitter. For bidirecitonal data flow between two microprocessors, 8086(1) sends data on the data bus to Port A of 8255(1). After that 8086(2) can read data from Port A of 8255(2) through data bus. in the same way, data can be transmitted from 8086(2) to 8086(1). For this bi-directional data flow, programming must be written in assembly language. The complete program is divided into two parts such as transmitter part and receiver part. The transmitter program part sends data to receiver while the reciever program accepts the data from the transmitting 8086 microprocessor.

11. (a) What do you mean by pipelined architecture? How is it implemented in 8086?
Refer Section 9.2.2.

(b) Explain how a 20-bit physical address is generated in the 8086 microprocessor.
Refer Section 9.4.

(c) Explain the operations of BIU and EU present in the 8086 microprocessor.
Refer Sections 9.2.1 and 9.2.2.

12. Write short notes on any three of the following:

(a) Addressing modes of 8051 microcontroller
Refer Section 8.2.

(b) MAX mode and MIN mode
Refer Sections 9.7.1, 9.7.2 and 9.7.3.

(c) Memory organization of 8051 microcontroller
Refer Section 7.3.

(d) PIC microcontroller
Refer Sections 19.1 and 19.2.

(e) Stack memory
Refer Section 4.5.

Solution of 2013 WBUT Paper (EC502)

GROUP–A
(Multiple-Choice Questions)

1. Choose the correct alternatives for any *ten* of the following: **(10×1=10)**

(i) The address lines required for 16K byte memory chip are
(a) 13 (b) 14 (c) 15 (d) 16

(ii) The interrupt line having highest priority is
(a) RST 7.5 (b) READY (c) TRAP (d) INTR

(iii) How many interrupts are controlled by 8259A?
(a) 8 (b) 6 (c) 9 (d) 5

(iv) PSW in the 8085 microprocessor is a
(a) 8-bit register (b) 16-bit register
(c) 4-bit register (d) 32-bit register

(v) The Intel 8086 processor is
(a) 16-bit (b) 32-bit (c) 64-bit (d) none of these

(vi) The 8085 microprocessor operates at a frequency of
(a) 6 MHz (b) 3.2 MHz (c) 5 MHz (d) 3 MHz

(vii) READY is used for
(a) input (b) output (c) both (a) and (b) (d) none of these

(vii) The memory map of a 4 kB memory chip begins at the location 3000 H. The last location of the memory address and number of pages in the chip are
(a) 3FFFH, 16 (b) 4000H, 16 (c) 3F00H, 8 (d) 300FH, 4

(ix) The number of segment registers in 8086 microprocessor are
(a) 8 (b) 4 (c) 16 (d) 32

(x) On-chip ROM size of the 8051 microcontroller is
(a) 1 kB (b) 16 kB (c) 4 kB (d) 8 kB

(xi) In the 8255 programmable peripheral interface, bidirectional mode operation is supported in
(a) Mode 1 (b) Mode 0 (c) Mode 0 and Mode 1 (d) Mode 2

(xii) In the 8051 microcontroller, external ROM is selected using
(a) $\overline{EA}$ (b) $\overline{PSEN}$ (c) RESET (d) ALE

Solution

(i) (b) 14 (ii) (c) TRAP (iii) (a) 8 (iv) (a) 8-bit register
(v) (a) 16-bit (vi) (d) 3 MHz (vii) (a) input (viii) (b) 4000 H, 16
(ix) (b) 4 (x) (c) 4kB (xi) (d) Mode-2 (xii) (a) $\overline{EA}$

GROUP–B
(Short-Answer Questions)

Answer any *three* of the following. (3 × 5 = 15)

2. Interface two 2K × 8 RAMs with the 8085 microprocessor by using the IC 74138 decoder such that the starting addresses assigned to them are 8000H and 9000H respectively.

Figure 1 shows the address decoding technique of the 8085 microprocessor. A_0–A_{10} are used for addressing the 2K × 8 RAM IC_1. $A_{11} – A_{15}$ address lines are used to generate the chip-select signal $\overline{CS}$. The memory map of 2K× 8 RAM IC_1 which has the starting address 8000H is given below:

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	= 8000H
⋮																⋮
1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	= 87FFH

Fig. 1 Memory map of 2K× 8 RAM IC_1 with starting address 8000H

Similarly, A_0-A_{10} are used for addressing the 2K× 8 RAM IC_2 and $A_{11} – A_{15}$ address lines are used to generate the chip-select signal $\overline{CS}$. The memory map of 2K × 8 RAM IC_2 which has the starting address 9000H is given below in Fig. 2:

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	
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	= 9000H
⋮																⋮
1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	= 97FFH

Fig. 2 Memory map of 2K× 8 RAM IC_2 with starting address 9000H

Figure 3 shows the address decoder circuit with $A_{14} = 0$ and $A_{15} = 1$, and A_{11}, A_{12}, A_{13} are used as decoder inputs. In this case, a 3-line to 8-line decoder can be used to select any one output. Based on inputs at A_{11}, A_{12}, A_{13}, any one output of $O_0 – O_7$ will be active low and other output lines remain high. The output lines are connected to the chip select of memory ICs. It is depicted in Fig. 3 that O_0 is connected with the chip select of 2K × 8 RAM IC_1 and O_2 is connected with the 2K × 8 RAM IC_2. The output lines and corresponding memory address capability are given in Table 1.

Table 1 Memory address selected by the decoder

Output lines	Memory address
O_0	8000H-87FFH
O_1	8800H-8FFFH
O_2	9000H-97FFH
O_3	9800H-9FFFH
O_4	A000H-A7FFH
O_5	A800H-AFFFH
O_6	B000H-B7FFH
O_7	B800H-BFFFH

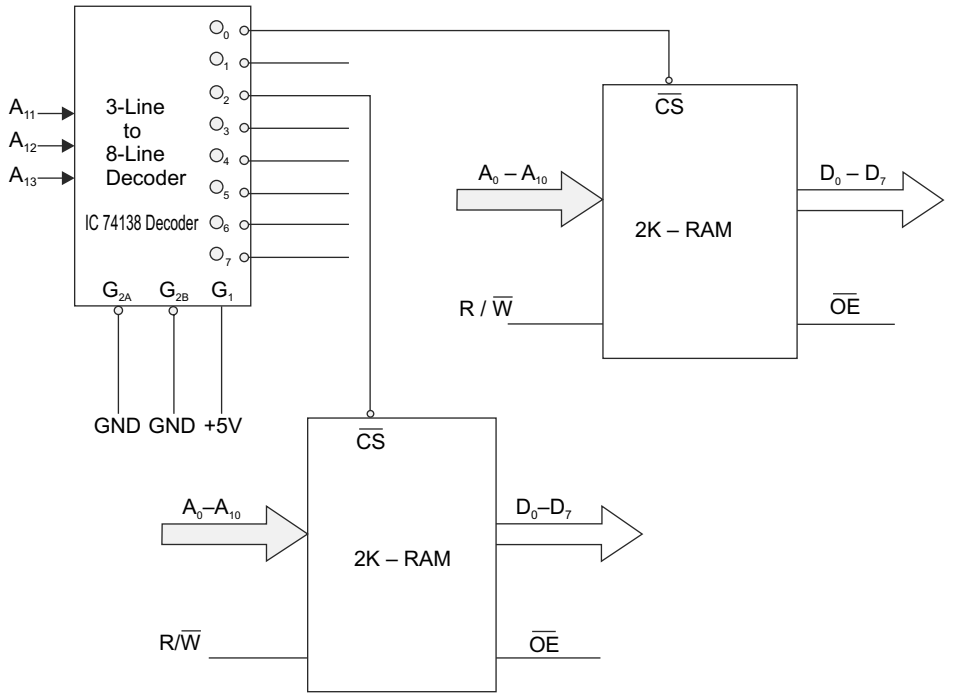


Fig. 3 Address decoding of two 2K× 8 RAM IC using IC 74138 Decoder when $A_{14} = 0$ and $A_{15} = 1$

b) What are maskable interrupts? Give an example.

Refer Sections 6.3 and 6.4.

Maskable interrupts of the 8085 microprocessor are INTR, RST 5.5, RST 6.5 and RST 7.5.

3. (a) What do the following instructions do?

(i) XRA A

XRA A (EXCLUSIVE-OR the content of accumulator with accumulator content)

$A \leftarrow A \oplus$ A States 4; Flags: all. Register addressing: One-byte instructions

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

(ii) LHLD 8000H

LHLD 8000H (Load H and L registers direct)

L ← [8000], H ← [8000+1]. States: 16; Flags: none; Direct addressing: three-byte instructions.

The instruction copies the contents of the memory location located by the 16-bit address 8000H into the register L and also copies the content of the next memory location 8001H into the register H. The contents of source memory locations are not changed.

(iii) RRC

RRC (Rotate accumulator right)

$A_7 \leftarrow A_o$, CS ← A_0, $A_n \leftarrow A_{n+1}$

States: 4; Flags: CS implicit; Addressing: One-byte instructions

Each binary bit of the accumulator is shifted right by one position. Bit D_0 is placed in the position of D_7 as well as in the Carry flag. Therefore, CY is modified accordingly as depicted in Fig.1. S, Z, P, AC are not affected.

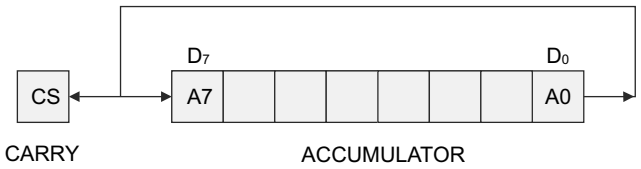


Fig. 4 Diagram for RRC

(b) Discuss the 'fetch' and 'execute' operations of the 8086 microprocessor
Refer Section 9.2.3.

4. **Write an assembly-language program to add two 16-bit numbers using the 8051 controller.**
Assume that the first number is 2498H and the second number is FE4CH. Here, after addition of two 16-bit numbers, the sum is more than 16-bit. It is also assumed that after addition, the result will be stored in R_2, R_1 and R_0 registers.

PROGRAM

Memory address	Machine codes	Labels	Mnemonics	Operands	Comments
8000	C3		CLRC		Clear carry C=0
8001	7A 00		MOV	R2, #00H	R_2 register is initialized.
8003	74 98		MOV	A, #98	Least significant byte of first number in accumulator.
8005	24 4C		ADD	A, #4CH	Add least significant byte of second number with accumulator.
8007	F8		MOV	R_0, A	Store result of lower byte in R_0 register.
8008	E5 24		MOV	A, 24H	Most significant byte of first number in accumulator.
800A	35 FE		ADDC	A, FEH	Add most significant byte of second number with accumulator.
800C	50 01		JNC	Level_1	Jump no carry to Level_1.
800E	0A		INC R_2		Increment R_2 register.
800F	F9	Level_1	MOV	R_1, A	Move the content of accumulator into R_1 register.
8010	02 00 00		LJMP	0000	

Result: After addition, the contents of R_2, R_1 and R_0 registers are as follows: R_2 = 01H, R_1 = 22H and R_0 = E4H

	2	4	9	8	H	first number
	+F	E	4	C	H	second number
Sum: 0 1	2	2	E	4	H	

5. Write an assembly-language program to load a block of data from the memory location 80XX H to the memory location 80XY H. Clearly mention the assumptions.

Assume that a block of data is available from the memory location 80XX H to the memory location 80XY H. Transfer the block so that it can be stored from 90XX H to 90XY H. The number of bytes in the block is stored in 80X0H. Assume that the program starts from 8000H, X = 1 and Y = 9. Then 80XX H = 8011H, 80XY H = 8019H, 90XX H = 9011H, 90XY H = 9019 H,

Algorithm

1. Store the address of number of data in HL register pair.
2. Load number of data in C register from memory.
3. Store the starting address of the destination in DE register pair.
4. Increment HL register pair to get data from source.
5. Copy data from source to accumulator.
6. Exchange HL and DE register pairs, store the content of the accumulator in destination address.
7. Exchange HL and DE register pairs.
8. Increment HL and DE registers.
9. Decrement C register.
10. If C register is not zero, repeat steps 5 to 9.

PROGRAM

Memory address	Machine codes	Labels	Mnemonics	Operands	Comments
8500	21, 10, 80		LXI	H, 8010	Store the address of number of data, 8010H in HL register pair.
8503	46		MOV	B,M	Load number of data in B register from memory.
	7E		MOV	A,M	Copy first data in accumulator
8504	21, 11, 90		LXI	D,9011	Store the destination address in DE register pair.
8507	23		INX	H	Increment H-L register pair.
8508	7E	LOOP	MOV	A,M	Move data from source to accumulator.
8509	EB		XCHG		Exchange the content of HL and DE.
850A	77		MOV	M,A	Store the content of accumulator, data in destination address.
850B	23		INX	H	Increment source address.
850C	13		INX	D	Increment destination address.
850D	05		DCR	B	Decrement B register.
850E	C2, 08, 85		JNZ	LOOP	If B is not zero, jump to LOOP.
8511	76		HLT		

Example

Input		Result	
ADDRESS	**DATA**	**ADDRESS**	**DATA**
8011	48 H	9011	48 H
8012	1A H	9012	1A H
8013	F2 H	9013	F2 H
8014	06 H	9014	06 H
8015	33H	9015	33H
8016	01H	9016	01H
8017	72H	9017	72H
8018	44H	9018	44H
8019	67H	9019	67H

GROUP–C
(Long-Answer Questions)

Answer any *three* of the following. (3 × 15 = 45)

6. (a) Briefly discuss the different transfer modes of 8237 DMA controller.

Refer Sections 17.4, 17.4.1, 17.4.2, 17.4.3,and 17.4.4

8237 is an advanced programmable DMA controller. It provides a better performance compared to 8257. This is capable of transferring a byte or a bulk of data between system memory and peripherals in either direction. Memory to memory data transfer is also possible in this peripheral. The 8237 can support four independent DMA channels which can be expanded to any number by cascading more units of 8237.

The 8237 operates in two cycles: passive cycle and active cycle. Each cycle contains a fixed number of states. The 8237 can assume six states when it is in active cycle. During idle cycle, it is in idle state (SI).

The 8237 is initially in a state SI, meaning an idle state where the 8237 does not have any valid pending DMA request. During this time, although the 8237 may be idle, the CPU may program it in this state. Once there is a DMA request, the 8237 enters the state S_0, which is the first state of the DMA operation. When the 8237 requests the CPU for a DMA operation and the CPU has not acknowledged the request, the 8237 waits in S_0 state. The acknowledge signal from the CPU indicates that the data transfer may now begin. The S_1, S_2, S_3 and S_4 are the working states of DMA operation, in which the actual data transfer is carried out. If more time is required to complete a transfer than what is allowed, wait states may be inserted between S_2 and S_3 or S_3 and S_4 using the READY pin of 8237. So it is clear that a memory read or a memory write DMA operation actually requires four states S_1 to S_4.

(b) Draw a timing diagram for opcode 'fetch' machine cycles of the 8085 microprocessor.

In the fetch cycle, the microprocessor fetches the opcode of an instruction from the memory. Fig.1 shows the timing diagram for an opcode fetch cycle of an instruction MOV A,B. Assume

that the opcode of instruction MOV A,B is stored in 8000H and the content of register B is 4FH. To execute this instruction, four consecutive clock cycles T_1, T_2, T_3 and T_4 are required. The sequence of operations is given below:

First Clock Cycle

- In the first clock cycle, T_1, the microprocessor places the content of program counter, address of the memory location 8000H, where the opcode is available on the 16-bit address bus. The 8 MSBs of the memory address (80H) are placed on the high-order address bus, A_{15}–A_8 and 8 LSBs of the memory address (00H) are placed on the low-order address bus, AD_7–AD_0. Since the AD bus is needed to transfer data during subsequent clock cycles, it is used in a time-multiplexed mode.
- The microprocessor sends an address latch enable (ALE) signal to go high and latch the 8 LSBs of the memory address. Therefore, low-order address bus is demultiplexed and the complete 16-bit memory address is available in the subsequent clock cycles to get the opcode from the specified memory address, 8000H.

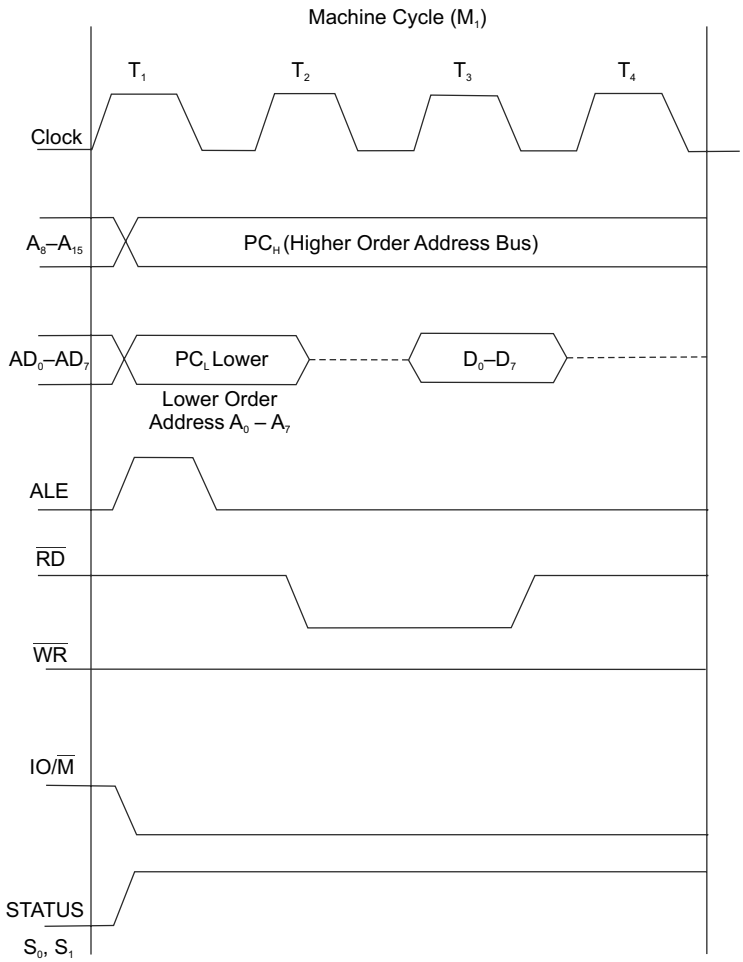


Fig. 5 Timing diagram of opcode fetch operation

- The microprocessor sends the status signals IO/$\overline{M}$=0, S_0 =1 and S_1=1 to indicate opcode fetch operation.

Second Clock Cycle

- During T_2, the low-order bus AD_7-AD_0 is ready to carry data from the memory location. The microprocessor sends the control signal $\overline{RD}$ = 0 to enable memory and the program counter is incremented by 1 to 8001H. Now the opcode from the specified memory location 8000H gets placed on the data bus.

Third Clock Cycle

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

Fourth Clock Cycle

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

(c) How much time is required to execute the following instruction?

MVI B, 07 (07 T state)

When the operating frequency of the microprocessor is 5 MHz, the time period of one T state is equal to $T = \frac{1}{f} = \frac{1}{5 \times 10^6} = 0.2\,\mu s$.

As 07 *T* states are required to execute the instruction MVI B, 07; the total time required is equal to 7 T= 7 × 0.2 μs = 1.4 μs.

(d) What are the different modes of operation of 8255 PPI?

Refer Sections 12.4, 12.4.1, 12.4.2, 12.4.3 and 12.4.4.

7. a) How does the 8086 microprocessor support memory segmentation?

Refer Sections 9.3.2 and 9.6 and Solution of 2009 WBUT Paper Q.5, Page S.9.

b) How is pipelining implemented in 8086?

Refer Sections 9.2.1, 9.2.2 and 9.2.3.

c) What is the relationship between logical address and physical address in 8086?

Refer Section 9.4.

d) Discuss the flag register of 8086.

Refer Section 9.3.4.

8. a) Write an assembly-language program using 8085 assembly language to arrange a string of 10-byte length in ascending order.

A series of ten numbers: F2H, 05H, 88H, 23H, 65H, 24H, 66H, 77H, 88H and 44H are stored in memory locations from 9001H to 900AH. Arrange the above numbers in ascending order to be stored in 9001H to 900AH locations. Assume the program memory starts from 9100H.

Algorithm

1. Store 0AH, number of data to be arranged in C register from memory and store number of comparisons in D register.
2. Initialise the memory location 9001H of first data.
3. Load the first data in the accumulator from the memory.

4. Increment HL register pair for addressing the next data.
5. Load the next data in B register from the memory.
6. Compare the next data with the accumulator. Store the largest number in the accumulator and the smallest number in the memory.
7. Then the next number is compared with the accumulator and store the smallest number in the memory and the largest number in the accumulator.
8. This process will continue, till comparison of all numbers has been completed. After completion of comparison of all numbers, the largest number will be in the accumulator and smallest in the memory. In this way, the first process will be completed.
9. At the starting of the second process, C register is decremented and store the number of comparisons in D register. Then repeat step-2 to step-8. After completion of this process, the largest number is in 900AH and the second largest number is in 9009H.
10. C register is decremented and the next process starts, if the content of C register is not zero.

PROGRAM

Memory address	Machine codes	Labels	Mnemonics	Operands	Comments
9100	0E, 05		MVI	C,0AH	Load count value of number of data in C register.
9102	16, 05	START	MVI	D,0A	Load count for number of comparisons in D register.
9104	21, 01, 90		LXI	H,9001H	Load memory location of first number.
9107	7E		MOV	A,M	First number in accumulator.
9108	23	LOOP	INX	H	Increment HL register pair for addressing next number.
9109	46		MOV	B,M	Copy next number in B register from memory.
910A	B8		CMP	B	Compare next number with accumulator.
910B	D2, 27, 91		JNC	LEVEL_1	If the content of accumulator > next number, jump to LEVEL_1
910E	2B		DCX	H	Increment HL register pair to locate the addressing for storing smallest number.
910F	77		MOV	M,A	Store smallest of the two numbers in memory.
9120	78		MOV	A,B	Move largest of the two numbers in accumulator from B register.
9121	C3, 2B, 91		JMP	LEVEL_2	Jump to LEVEL_2
9124	2B		DCX	H	
9125	77		MOV	M,A	Place greater of the two numbers in the accumulator.

9126	C3, 2B, 91		JMP	LEVEL_2	Jump to LEVEL_2
9129	2B	LEVEL_1	DCX	H	Store smallest of the two numbers in memory.
912A	70		MOV	M,B	
912B	23	LEVEL_2	INX	H	
912C	15		DCR	D	Decrement D register to count for number of comparisons.
912D	C2, 08, 91		JNZ	LOOP	Jump to zero.
9130	77		MOV	M,A	Place largest number in memory.
9131	0D		DCR	C	Decrement count value.
9132	C2, 02, 91		JNZ	START	Jump to zero to START.
9135	76		HLT		Halt.

Example

DATA		**RESULT**				
Memory location	**Data**	**Memory location**	**After 1st process**	**After 2nd process**	**After 3rd process**	**After 10th process**
9001	F2H	9001	05H	05H	05H	05H
9002	05H	9002	88H	23H	23H	23H
9003	88H	9003	23H	65H	65H	24H
9004	23H	9004	65H	24H	24H	44H
9005	65H	9005	24H	66H	66H	65H
9006	24H	9006	66H	77H	77H	66H
9007	66H	9007	77H	82H	44H	77H
9008	77H	9008	82H	44H	82H	82H
9009	82H	9009	44H	88H	88H	88H
900A	44H	900A	F2H	F2H	F2H	F2H

(b) Explain bidirectional data transfer using 8255 PPI.

Refer Section 12.4.3.

9. (a) What will be the contents of the accumulator and flag after the following instructions from a program that are executed sequentially?

```
MVI A, 01
MVI B, 02
ADD B
XRA A
HLT
```

After execution of the instruction MVI A, 01; 01H will be loaded in the accumulator and flag will not be affected. After that, instruction MVI B, 02 is executed. Just after execution of the instruction MVI B, 02; 02H will be loaded in B register and no flag is affected. Subsequently, instruction ADD B is executed. After completion of execution of instruction ADD B, the content

of the accumulator is 03H and flags are affected. Result is nonzero and flag Z=0. There are two number of 1s and flag P is set to 1 (P=1). AC=0, CS=0, MSB of the Sum is zero and flag S is set to 0 (S=0). When the instruction XRA A is executed, the content of Accumulator is $A \leftarrow A \oplus A$ and all flags are affected. Therefore, the content of A is $A \leftarrow 03 \oplus 03 = 00$. As result is zero, flag Z=1. There is no 1s and flag P is set to 1 (P=1). AC=0, CS=0, MSB of the Sum is zero and flag S is set to 0 (S=0). When the instruction HLT is executed, the microprocessor finishes executing the current instruction and halts any further execution and no flag is affected.

(b) Draw the block diagram of 8254 timer and briefly discuss its operation and organization.
Refer Section 13.2.

(c) Describe the priority scheme and EOI scheme of 8259A.
Refer Section 14.6.3.

10. Write short notes on any three of the following: 3×5

(a) Function of 8251 USART
Refer Section 16.1.

(b) Serial mode operation using 8085 microprocessor
Refer Solution of 2011 WBUT Paper Q.11 (e), Page S.48.

(c) Subroutine organization (including calls) in 8086 microprocessor
Refer Section 4.6, 4.6.1, 4.6.1 and 10.3.10.

(d) BIU and EU of 8086 microprocessor
Refer Section 9.2.1 and 9.2.2.

(e) DMA
Refer Section 17.1.