

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/264005162>

# An Introduction to Microprocessor 8085

Book · January 2010

---

CITATIONS

0

READS

199,839

1 author:



D.K. Kaushik

Shobhit University, Gangoh (Saharanpur) India

44 PUBLICATIONS 38 CITATIONS

SEE PROFILE

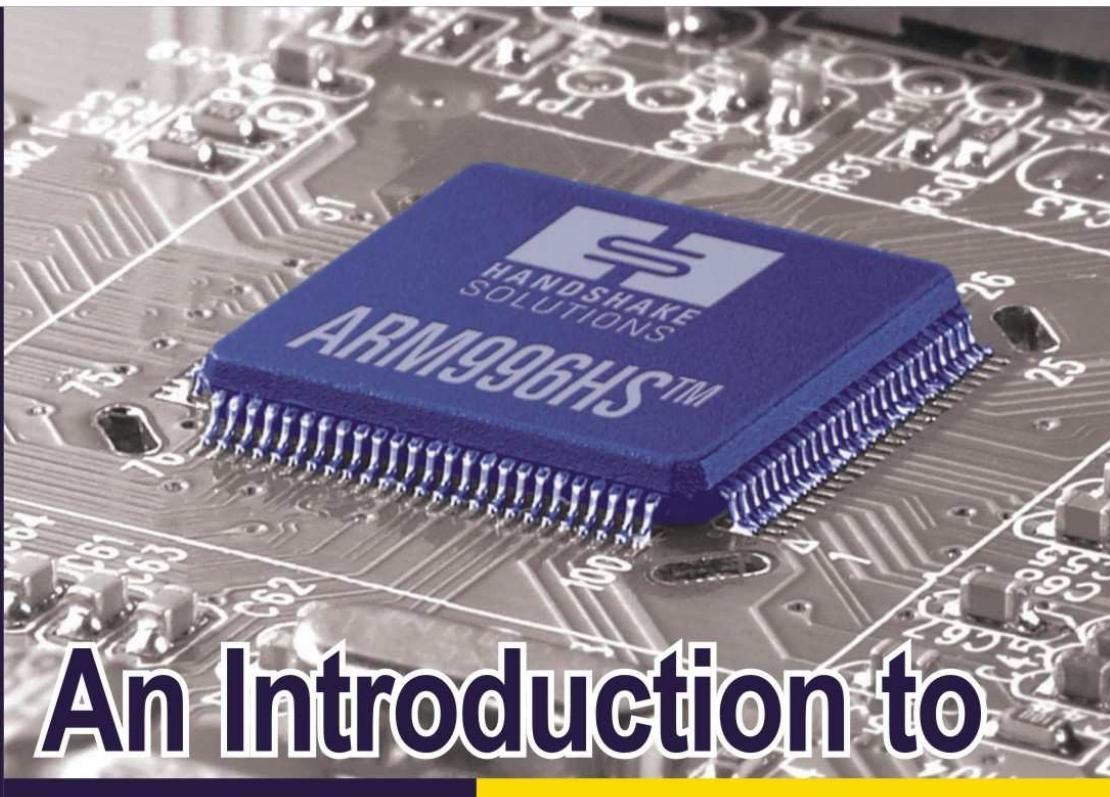
Some of the authors of this publication are also working on these related projects:



Modeling [View project](#)

An Introduction To Microprocessor 8085

Dr. D.K. Kaushik



# An Introduction to

## Microprocessor

**8085**

**Dr. D.K. Kaushik**

**DHANPAT RAI PUBLISHING COMPANY**



Dr. D. K. Kaushik is presently working as the Principal, Manohar Memorial Post-Graduate College, Fatehabad (Haryana). Earlier, he was the Head, Department of Electronics, Dayanand Post-Graduate College, Hisar. He obtained his master's degree in Physics from Meerut University and Ph.D. in 1981 from Kurukshetra University, Kurukshetra. He has more than 28 years teaching experience in the subject of Electronics and published more than 24 research papers in journals of high repute. He has two patents for indigenous design of Electronic Thin Film Thickness Monitors, to his credit. He had been the Member board of studies in Electronics of Kurukshetra University, Kurukshetra many times. His areas of interest are Semiconductor Electronics, Microprocessor, Computer Architecture and Thin Film Technology.

#### SOME OTHER USEFUL BOOKS BY THE AUTHOR

- Analog Electronics (Circuits and Design) .....Dr.D.K.Kaushik
- Digital Electronics .....Dr.D.K.Kaushik
- Analog Electronic Circuits .....Dr.D.K.Kaushik



**DHANPAT RAI PUBLISHING COMPANY (P) LTD.**  
**4779/23, Ansari Road, Darya Ganj, New Delhi - 110002**  
Phones : 011-23257511, 23257526 Fax : 011-23257525 email : mail@dhanpatrai.in  
Website : [www.dhanpatrai.in](http://www.dhanpatrai.in)

# **AN INTRODUCTION TO MICROPROCESSOR 8085**

By

Dr. D. K. Kaushik  
Principal, Manohar Memorial (P.G.) College,  
Fatehabad (Haryana) India

Dhanpat Rai Publishing co., New Delhi

# **AN INTRODUCTION TO MICROPROCESSOR 8085**

## **Chapter 1 Evolution and History of Microprocessors**

- 1.1 Introduction
- 1.2 **Evolution/History of Microprocessors**
- 1.3 **Basic Microprocessor System**
  - 1.3.1 Address Bus
  - 1.3.2 Data Bus
  - 1.3.3 Control Bus
- 1.4 Brief Description of 8-Bit Microprocessor 8080A
- 1.5 Computer Programming Languages
  - 1.5.1 Low-Level Programming Language
  - 1.5.2 High-Level Programming Language

## **Chapter 2 SAP – I**

- 2.1 Architecture of SAP – I
- 2.2 Instruction Set of SAP-I Computer
- 2.3 Programming of SAP-I Computer
- 2.4 Working of SAP-I Computer
  - 2.4.1 Fetch Cycle
  - 2.4.2 Execution Cycle
- 2.5 Hardware Design of SAP-I Computer
  - 2.5.1 Design of Program Counter
  - 2.5.2 Memory Unit
  - 2.5.3 Instruction Register
  - 2.5.4 Controller-Sequencer
  - 2.5.5 Accumulator
  - 2.5.6 Adder-Subtractor
  - 2.5.7 B-Register
  - 2.5.8 Output Register

## **Chapter 3 SAP – II**

- 3.1 Architecture of SAP-II Computer
- 3.2 Instruction Set of SAP-II Computers
- 3.3 Machine Cycle and Instruction Cycle
- 3.4 Addressing Modes
- 3.5 Instruction Types
- 3.6 Flags
- 3.7 Assembly Language Programming
- 3.8 Delay Calculations

## **Chapter 4 SAP – III**

- 4.1 Programming Model of SAP-III Computer
- 4.2 Instruction Set of SAP-III Computer
  - 4.2.1 Data Transfer Group
  - 4.2.2 Arithmetic Group of Instructions
  - 4.2.3 Logic Transfer Group
  - 4.2.4 Branch Group

- 4.2.5 Stack and Input / Output Instructions
- 4.3 Time Delay Introduced by a Register Pair

## Chapter 5 The 8085 Microprocessor

- 5.1 **Architecture** of 8085 Microprocessor
  - 5.1.1 Register Section
  - 5.1.2 Address Buffer and Address-Data Buffer
  - 5.1.3 Arithmetic and Logical Unit (ALU)
  - 5.1.4 **Timing and Control Unit**
  - 5.1.5 **Interrupt Control**
- 5.2 **Pin Description** of 8085
- 5.3 **Instruction Set** of 8085 Microprocessor
- 5.4 Timing Diagram for 8085 Instructions
  - 5.4.1 Timing Diagram of *MOV reg, M*
  - 5.4.2 Timing Diagram of *MOV M, reg*
  - 5.4.3 Timing Diagram of *MVI reg, data*
  - 5.4.4 Timing Diagram of *MOV reg2, reg1*
  - 5.4.5 Timing Diagram of *MVI M, data*
  - 5.4.6 Timing Diagram of *XCHG*
  - 5.4.7 Timing Diagram of *LXI rp, dbyte*
  - 5.4.8 Timing Diagram of *IN byte*

## Chapter 6 Programming of 8085

- 6.1 Simple Programs
- 6.2 Programs on code conversion
  - 6.2.1 BCD to Binary Conversion
  - 6.2.2 Binary to BCD (Unpacked) Conversion
  - 6.2.3 Binary to ASCII Conversion
  - 6.2.4 ASCII to Binary Conversion
- 6.3 Programs on addition and subtraction
- 6.4 Programs to find largest or smallest number
- 6.5 Programs to arrange a given series in ascending or descending order
- 6.6 Program on Multiplication
- 6.7 Program on 8-Bit Division
- 6.8 Miscellaneous Programs

## Chapter 7 **Interrupt Instructions** of 8085

- 7.1 Methods of I/O Operations
  - 7.1.1 Memory Mapped I/O
  - 7.1.2 I/O Mapped I/O or Isolated I/O
- 7.2 Data Transfer Schemes
  - 7.2.1 Programmed I/O Data Transfer
  - 7.2.2 Interrupt Driven I/O Data Transfer
  - 7.2.3 Direct Memory Access (DMA) Data Transfer
- 7.3 The 8085 Interrupts
- 7.4 Software Interrupts
- 7.5 Hardware Interrupts
- 7.6 Interrupt Control Circuit
- 7.7 Interrupt Instructions

## 7.8 Serial Input and Output Data Transfer

### Chapter 8 Programmable Peripheral Interface (PPI) 8255A

- 8.1 Details of PPI IC 8255A
- 8.2 Operational Modes of 8255A
- 8.3 Control Word Format for 8255A
- 8.4 Programming in Mode 0
- 8.5 Programming in Mode 1 (strobed Input/Output)
  - 8.5.1 Input Control Signals in Mode 1
  - 8.5.2 Output Control Signals in Mode 1
- 8.6 Programming in Mode 2 (Strobed Bidirectional Bus I/O)
- 8.7 Bit Set/Reset (BSR) Mode

### Chapter 9 Programmable Interval Timer/Counter: 8253

- 9.1 INTEL Programmable Interval Timer 8253
- 9.2 Block Diagram of 8253
- 9.3 Logics for Counters
- 9.4 Control Word Format of 8253
- 9.5 Interfacing and programming of 8253
- 9.6 Programming of 8253 in Mode 0: Interrupt on Terminal Count
- 9.7 Programming of 8253 in Mode 1: Programmable One Shot
- 9.8 Programming of 8253 in Mode 2: Rate Generator
- 9.9 Programming of 8253 in mode 3: Square Wave Generator
- 9.10 Programming of 8253 in mode 4: Software Triggered Strobe

### Chapter 10 Programmable Keyboard and Display Interface: 8279

- 10.1 INTEL Programmable Keyboard/Display Interface 8279
- 10.2 Block Diagram of 8279
- 10.3 Functional Description of 8279
- 10.4 Key Board San
- 10.5 Scanned Keyboard
  - 10.5.1 Two-key Lockout
  - 10.5.2 N-key Rollover
- 10.6 Scanned Sensor Matrix
- 10.7 Strobed Input
- 10.8 Display Interface
- 10.9 Display Modes
  - 10.9.1 Left Entry Mode (Type Writer Mode)
  - 10.9.2 Right Entry Mode (Calculator Mode)
- 10.10 Programming of 8279
  - 10.10.1 Keybaord/Display Mode Set
  - 10.10.2 Program Clock
  - 10.10.3 Read FIFO/Sensor RAM
  - 10.10.4 Read Display RAM
  - 10.10.5 Write Display RAM
  - 10.10.6 Display Write Inhibit/Blanking
  - 10.10.7 Clear
  - 10.10.8 End Interrupt/Error Mode Set
- 10.11 Status Register (IN Operation)

10.12 Interfacing of 8279 with 8085

## Chapter 11 Programmable Interrupt Controller: 8259

11.1 Programmable Interrupt Controller 8259

11.2 Block Diagram of 8259

11.3 Interfacing of 8259A with 8085A

11.4 Vectoring Data Formats for 8259

11.5 Initialization of 8259

11.6 Initialization Command Words (ICWs)

11.7 Operation Command Words (OCWs)

11.8 Interrupt Modes of 8259A

    11.8.1 Fully Nested Mode (FNM)

    11.8.2 Rotating Priority Mode

    11.8.3 Special Mask Mode

    11.8.4 Polled Mode

11.9 Status Read Operation of 8259

## Chapter 12 Direct Memory Access Controller: 8257

12.1 Block Diagram of 8257

    12.1.1 DMA Channels

    12.1.2 Data Bus Buffer

    12.1.3 Read/ Write Logic

    12.1.4 Control Logic

    12.1.5 Mode Set Register

    12.1.6 Status Word Register

12.2 Programming of 8257

12.3 DMA Interfacing Circuit

## Chapter 13 Interfacing Data converters: A/D and D/A Converters

13.1 Digital to Analog Converter

    13.1.1 Resistive Divider D/A converter

    13.1.2 Binary Ladder D/A Converter

13.2 Performance Criteria for D/A Converter

13.3 D/A Converter IC 0808

13.4 Interfacing of D/A Converter

13.5 Microprocessor Compatible D/A Converter

13.6 Analog to Digital Converter

13.7 Simultaneous A/D Converter

13.8 Successive Approximation A/D Converter

13.9 Counter or Digital Ramp Type A/D Converter

13.10 Single Slope A/D Converter

13.11 Dual Slope A/D Converter

13.12 ADC 0808/0809

13.13 A/D Converter Using D/A Converter and Software

## Chapter 14 Serial Communication and Programmable Communication Interface

- 14.1 Serial Data Communication
- 14.2 Modem
- 14.3 Serial Communication Standard
- 14.4 Asynchronous Software Approach
- 14.5 Programmable Communication Interface
- 14.6 Block Diagram of 8251A
  - 14.6.1 Read/Write Control Logic
  - 14.6.2 Transmitter Section
  - 14.6.3 Receiver Section
  - 14.6.4 Modem Control
- 14.7 Interfacing of 8251A
- 14.8 Programming of 8251A
  - 14.8.1 Initialization of 8251A in Asynchronous Mode
  - 14.8.2 Initialization of 8251A in Synchronous Mode

## Chapter 15 Applications of Microprocessor

- 15.1 Real Time Clock with On/Off Timer
- 15.2 Microprocessor Based LED Dial Clock
- 15.3 Design of Microprocessor Based Running light
- 15.4 Microprocessor Based Automatic School bell system
- 15.5 Microprocessor Based Traffic Light
  - 15.5.1 Another Design of Microprocessor Based Traffic Light
- 15.6 Microprocessor Based Stepper Motor Control
- 15.7 Microprocessor Based Washing Machine Controller
- 15.8 Microprocessor Based Water Level Controller
- 15.9 Microprocessor Based Temperature Controller

## Appendices

# 1

## Evolution and History of Microprocessors

---

To understand the working and programming of microprocessors, it is very necessary to know the details of evolution and history of microprocessors which will be discussed in this chapter. In the succeeding chapters of this book, the detailed discussion on the basics of microprocessors will be made. For the beginners it is very difficult to understand the operation of a digital computer as it contains large details, so efforts have been made to envisage the operation of a digital computer in step by step manner.

### 1.1 INTRODUCTION

Microprocessor is a digital device on a chip which can fetch instructions from a memory, decode and execute them i.e. performs certain arithmetic and logical operations, accept data from input device, and send results to output devices. Therefore, a microprocessor interfaced with memory and Input/ Output devices forms a Microcomputer. Basically, there are five building blocks of a digital computer namely:

**Input Unit** Through this unit data and instructions are fed to the memory of the computer. The basic purpose of this unit is to read the data into the machine. The program from the memory is read into the machine along with the input data which are required to solve or compute the problem by the machine. The typical devices which are used for this purpose are keyboards, paper tape reader and toggle switches etc.

**Memory Unit** The memory unit of a digital computer consists of devices which are capable of storing information. The memory of a computer is used for storing two distinct type of information such as data to be processed by the computer and program through which the result of the desired problem is obtained. Computer program and data are stored in the Memory Unit. This usually consists of chips of both ROMs (Read Only Memories) and RAMs (Random Access Memories) either bipolar or MOS.

**Arithmetic and Logical Unit (ALU)**

This unit is used for performing arithmetic operations such as Addition, Subtraction, Multiplications, division and other logical operations on the data.

The control unit guides ALU which of the operations are to be performed. The sequence of the instructions is controlled by the control unit.

**Control Unit**

The control unit performs the most important function in a computer. It controls all other units and also controls the flow of data from one unit to another for performing computations. It also sequences the operations. It instructs all the units to perform the task in a particular sequence with the help of clock pulses.

**Output Unit**

After processing of the data in the Arithmetic and Logical Unit, the results are displayed to the output world through this unit. The CRTs (Cathode Ray Tubes), LEDs (Light Emitting Diodes) and Printer etc. form the output unit.

In a computer system ALU and Control Unit are combined in one unit called Central Processing Unit (CPU). The block diagram of a computer is shown in figure 1.1.

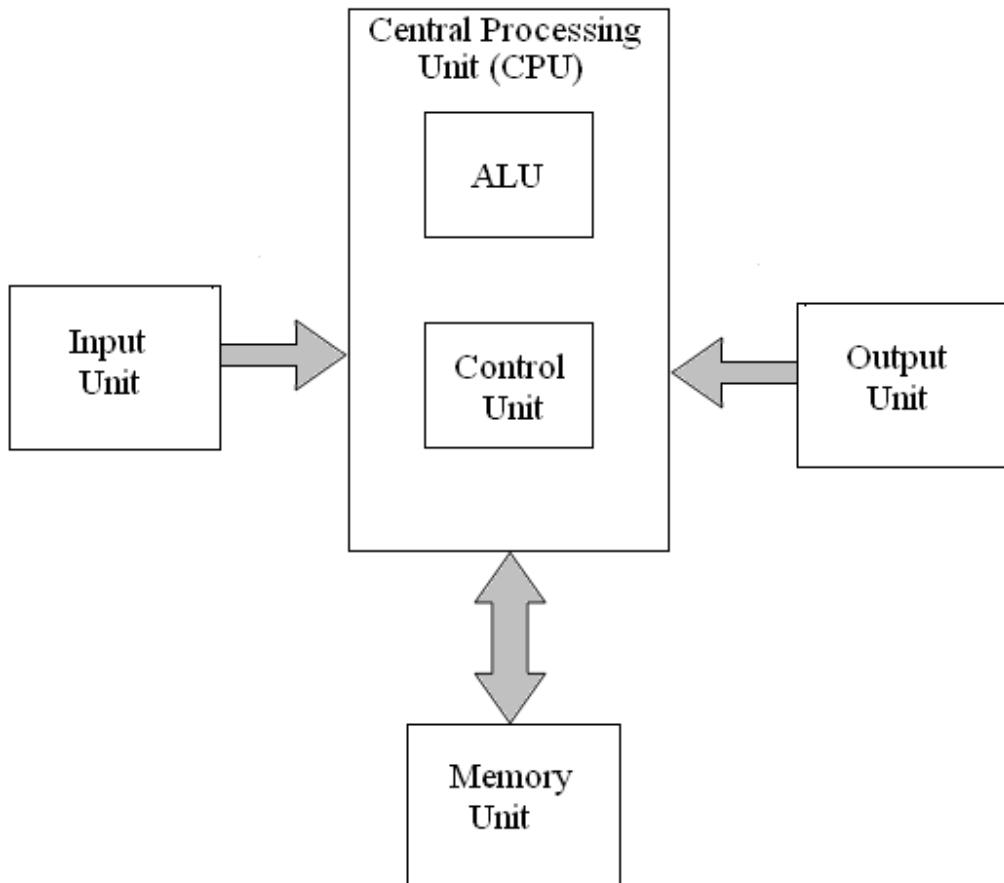


Fig. 1.1

The Central Processing Unit is analogous to the human brain as all the decisions as per the instructions are made by CPU. All other parts are also controlled by this unit. A microprocessor is an integrated circuit designed for use as Central Processing Unit of a computer. The CPU is the primary and central player in communicating with devices such as memory, input and output. However, the timing of communication process is controlled by the group of circuits called control unit.

The term ‘Microprocessor’ came into existence, in 1971, when the Intel Corporation of America, developed the first microprocessor (INTEL-4004) which is a 4-bit microprocessor ( $\mu p$ ). A microprocessor is a programmable digital electronic component that incorporates the functions of a Central Processing Unit (CPU) on single semi-conducting Integrated Circuits (ICs). As such, a system with microprocessor as an integral part is termed as a microprocessor based system. When a computer is microprocessor based, it is called a microcomputer ( $\mu c$ ).

A microprocessor is specified by its ‘Word Size’, e.g. 4-bit, 8-bit, 16-bit etc. By the term ‘word size’ means the number of bits of data that is processed by the microprocessor as a unit. For example, an 8-bit microprocessor performs various operations on 8-bit data. It also specifies the width of the data bus. As discussed above, a microcomputer consists of input/ output devices, and memory, in addition to microprocessor which acts its CPU. In fact CPU is commonly referred to microprocessor ( $\mu p$ ). Microprocessors made possible the advent of the microcomputer in the mid-1970s. Before this period, electronic CPUs were typically made from bulky discrete switching devices. Later on small-scale integrated circuits were used to design the CPUs. By integrating the processor onto one or very few large-scale integrated circuit package (containing the equivalent of thousands or millions of discrete transistors), the cost of processor was greatly reduced.

The evolution of microprocessors has been known to follow Moore’s law when it comes to steadily increasing performance over the years. This law suggests that the complexity of an integrated circuit, with respect to minimum component cost, doubles in every 18 months. This dictum has generally proven true since the early 1970’s. This lead to the dominance of microprocessors over every other form of computer; every system from the largest mainframes to the smallest hand held computers now uses a microprocessor as its core.

The microprocessor based systems play significant role in the every functioning of industrialized societies. The microprocessor can be viewed as a programmable logic device that can be used to control processes or to turn on/off devices. So the microprocessor can be viewed as a data processing unit or a computing unit of a computer. The microprocessor is a programmable integrated device that has computing and decision making capability similar to that of the central processing unit (CPU) of a computer. Nowadays, the microprocessor is being used in a wide range of products called microprocessor based products or systems. The microprocessor communicates and operates in the binary numbers 0 and 1, called bits. Each microprocessor has a fixed set

of instructions in the form of binary pattern called a machine language. However, it is difficult for human beings to communicate in the language of 0s and 1s. Therefore, the binary instructions are given abbreviated names, called mnemonics, which form the assembly language for a given microprocessor.

## 1.2 EVOLUTION/HISTORY OF MICROPROCESSORS

An English Mathematician Charles Babbage was the first man to propose the basic principle of modern computers from 1792-1871. He gave the concept of a programmable machine having computer similar to modern digital computers. He is, therefore, known Father of Modern Computers. In 1930s successful general purpose mechanical computer was developed. Before this, mechanical calculators were built to perform simple mathematical operations such as addition, subtraction, multiplication and division. Improvement continued and in 1944 Prof. H. Aiken developed a first practical electro-mechanical digital computer in collaboration with IBM. This computer was known as HAWARD MARK-I. It was in large size (51' long and 8'high) and weighing about 2 tons. The punch cards were used to input the data in the computer.

During the development of the HAWARD MARK-I Computer, Konard Joos of Germany was busy in developing another computer based on 0's and 1's rather than decimal numbers. So he developed a computer making use of relays (on-off for 1's and 0's) during 1936-44. Joos also developed a language for the computer. The giant machines during 1940-50 were thus designed using relays and vacuum tubes.

In 1945, John J. Mauchy and J. Presper Eckert of University of Pennsylvania developed first electronic computer ENIAC (Electronic Numerical Integrator and Calculator). It was too huge weighing 30 tons and occupied an area 30'X50' and made use of 18000 vacuum tubes, more than 30000 resistors, 10000 capacitors and 6000 switches. It took 200  $\mu$ S for addition, and 3mS for 10-digit multiplication. It had separate memory for program and data. It used 20 electronic accumulators for memory. Each accumulator stored signed 10-digit decimal number. A number of computers using vacuum tubes were developed during 1940-55. The main drawback with the ENIAC was the life of the vacuum tube components, which required the frequent maintenance.

The invention of semiconductor transistors in 1948 at Bell Laboratories leads further development of computers. The use of semiconductor transistors could not only reduced the size of computers but also increased its capability to a great extent. This leads reduction in cost. Further, the invention of Integrated circuits in 1958 by Jack Kilby of Texas Instrument made a revolution in electronic circuitry. The use of ICs made the size of computers very small and became more versatile in functions. Finally, the advent of IC technology leads to the development of first microprocessor (INTEL 4004) in 1971 at Intel Corporation by an engineer Marcian E. Hoff. It was a 4-bit microprocessor – a programmable controller on a chip. This was called the first generation microprocessor. It was fabricated using P-channel MOSFET technology and had an instruction set of 45 different instructions. It addressed 4096 four-bit wide memory locations. The P-channel MOSFET technology gave low cost but low speed not compatible with TTL (Transistor-

Transistor Logic) technology. It has to use at least 30 ICs to form a system. As INTEL 4004 had very small number of instructions, it could be used in limited applications such as early video games and small microprocessor-based controllers. Seeing microprocessor as a viable product, Intel Corporation released the 8008 microprocessor – an extended 8 bit version of the 4004 microprocessor in 1972. Soon a variety of microprocessors was released by different manufacturers. A few first generation microprocessors are listed in table 1.1.

**Table 1.1**

| <b>4-bit Microprocessors</b> | <b>8-bit Microprocessors</b> |
|------------------------------|------------------------------|
| INTEL 4004                   | INTEL 8008                   |
| INTEL 4040                   | NATIONAL IMP-8               |
| FAIRCHILD PPS-25             | ROCKWELL PPS-8               |
| ROCKWELL PPS-4               | AMI 7200                     |
| NATIONAL IMP-4               | MOSTEK 5065                  |

Second generation microprocessors appeared in 1973 and used NMOS-technology which offered faster speed, higher density and still better reliability. In the year 1974, an 8-bit microprocessor INTEL 8080 was developed using NMOS technology. It requires only two additional devices to design a functional CPU. It is much faster than 8008 and has more instructions than 8008 that facilitates the programming. The 8080 was compatible with TTL, whereas the 8008 was not directly compatible. The 8080 microprocessor could also address four times more memory (64K bytes) than the 8008 microprocessor (16K bytes). INTEL Corporation in 1977 developed another 8-bit microprocessor 8085 which was proved to be a better version than 8080. The execution time of two 8-bit numbers is  $2.0 \mu S$  for 8085 whereas it is  $1.3 \mu S$  for 8080. The main advantages of the 8085 were its internal clock generator, internal system controller, and higher clock frequency. Some of the important second generation microprocessors are given in table 1.2.

**Table 1.2**

| <b>8-bit Microprocessors</b> | <b>12-bit Microprocessors</b> |
|------------------------------|-------------------------------|
| INTEL 8080                   | INTERSIL 6100                 |
| INTEL 8085                   | TOSHIBA TLCS-12               |
| FAIRCHILD F8                 |                               |
| MOTOROLA M6800               |                               |
| ZILOG Z-80                   |                               |
| SIGNETICS 2650               |                               |

The advantages of second generation microprocessor are given below:

- (i) Larger chip size (170 x 200 mils),
- (ii) 40 pins,
- (iii) More number of on-chip decoded timing signals,
- (iv) Ability to address larger memory space,
- (v) Ability to address more I/O Ports,
- (vi) More powerful instruction set,
- (vii) Faster operation,
- (viii) Better Interrupt handling capabilities.

Third generation microprocessors were introduced in 1978. These were 16-bit microprocessors, designed using HMOS (High Density MOS) technology. These microprocessors offered better speed and higher packing density than NMOS. Some important third generation microprocessors are given in table 1.3.

**Table 1.3**

| <b>16-bit Microprocessors</b>                          |   |  |
|--|---|--|
| INTEL 8086<br>INTEL 8088<br>INTEL 80186<br>INTEL 80286 | MOTOROLA-68000<br>MOTOROLA-68010<br>NATIONAL NS-16016<br>TEXAS INSTRUMENT-TMS-99000 | INTERSIL 6100<br>TOSHIBA TLCS-12<br>ZILOG Z-8000 |

In 1978, 16-bit INTEL 8086 microprocessor of 64 pins was introduced and in 1979 other 16-bit microprocessor 8088 was developed. In addition to the other performances, these  $\mu$ Ps contain multiply/divide/arithmetic hardware. The memory addressing capabilities has been increased to very large i.e., 1MB to 16MB through a variety of flexible and powerful addressing mode. The other characteristics of third generation are given below:

- (i) These microprocessors were 40/48/64 pins,
- (ii) High speed and very strong processing capability,
- (iii) Easier to program,
- (iv) Allow for dynamically re-locatable programs,
- (v) Size of internal registers were 8/16/32 bits,
- (vi) These  $\mu$ Ps had the multiply/divide/arithmetic hardware,
- (vii) Physical memory space was from 1 to 16 Mega-bytes (MB),
- (viii) Flexible 10 port addresses,
- (ix) More powerful interrupt and hardware capabilities,
- (x) Segmented address and virtual memory features.

Fourth generation microprocessors of 32 bits were introduced in the form of 80386 in 1985 and 80486 in 1989. The instruction set of the 80386 microprocessor was upward compatible with the earlier 8086, 8088 and 80286 microprocessors. However, the 80486 is an improved version of the 80386 microprocessor. The 80386 executes many instructions in 2 clock cycles while the 80486 executes in one clock cycle. These microprocessors are of low power version of HMOS technology. Some important fourth generation microprocessors are given in table 1.4.

**Table 1.4**

| <b>32-bit Microprocessors</b>                                       |  |
|---|--|
| INTEL 80386<br>INTEL 80486<br>NATIONAL NS16022<br>MOTOROLA MC 88100 | MOTOROLA M-68020<br>MOTOROLA M-68030<br>BELLMAC-32 |

Fifth generation microprocessor was introduced by INTEL Corporation in 1993 in the form of PENTIUM with 64 data bus. The Pentium was similar to the 80386 and 80486 microprocessor. The two introductory versions of the Pentium operated with a clock frequency of 60 MHz and 66 MHz and a speed of 110 MIPS (Million Instructions Per Second). With better and more advanced technologies, the speed of  $\mu$ Ps has increased tremendously. The old 8085 of 1977 executed 0.5 million instruction/sec. (0.5 MIPS), while the 80486 executes 54 million instruction per sec.

The Pentium Pro Processor is the Sixth generation microprocessor introduced in 1995 having better architecture but more in size. The Pentium Pro Microprocessor contains 21 million transistors, 3 integer units as well as a floating unit to increase the performance of most software. The basic clock frequency is 150 MHz and 166 MHz.

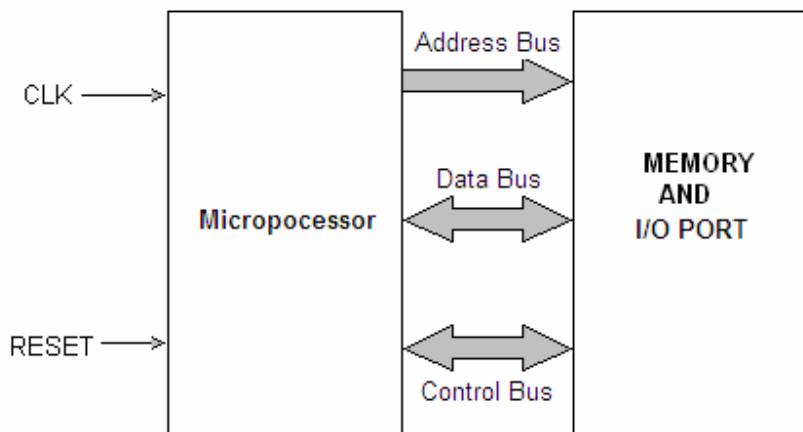
### **1.3 BASIC MICROPROCESSOR SYSTEM**

The Microprocessor alone does not serve any useful purpose unless it is supported by memory and I/O ports. The combination of memory and I/O ports with microprocessor is known as microprocessor based system. As discussed above the microprocessor which is the central processing unit executes the program stored in the memory and transfer data to and from the outside world through I/O ports. The microprocessor is interconnected with memory and I/O ports by the data bus, the Address bus and the control bus.

A bus is basically a communication link between the processing unit and the peripheral devices as shown in figure 1.2.

#### **1.3.1 Address Bus**

The address bus is unidirectional and is to be used by the CPU to send out address of the memory location to be accessed. It is also used by the CPU to select a particular input or output port. It may consist of 8, 12, 16, 20 or even more number of parallel lines. Number of bits in the address bus determines the minimum number of bytes of data in the memory that can be accessed. A 16-bit address bus for instance can access  $2^{16}$  bytes of data. It is labeled as  $A_0 \dots A_{n-1}$ , where  $n$  is the width of bits of the address bus.



**Fig. 1.2**

### 1.3.2 Data Bus

Data bus is bidirectional, that is, data flow occurs both to and from CPU and peripherals. There is an internal data bus which may not be of the same width as the external data bus by that connects the I/O and memory. A microprocessor is characterized by the width of its data bus. All those microprocessors having internal and external data buses of different widths are characterized either by their internal or external data buses. The size of the internal data bus determines the largest number that can be processed by a microprocessor, for instance, having a 16-bit internal data bus is 65536 (64K). The bus is labeled as:  $D_0 \dots D_{n-1}$ , where  $n$  is the data bus width in bits

### 1.3.3 Control Bus

Control bus contains a number of individual lines carrying synchronizing signals. The control bus sends out control signal to memory, I/O ports and other peripheral devices to ensure proper operation. It carries control signals like MEMORY READ, MEMORY WRITE, READ INPUT PORT, WRITE OUTPUT PORT, HOLD, INTERRUPT etc. For instance, if it is desired to read the contents of a particular memory location, the CPU first sends out address of that very location on the address bus and a 'Memory Read' control signal on the control bus. The memory responds by outputting data stored in the addressed memory location on the data bus.

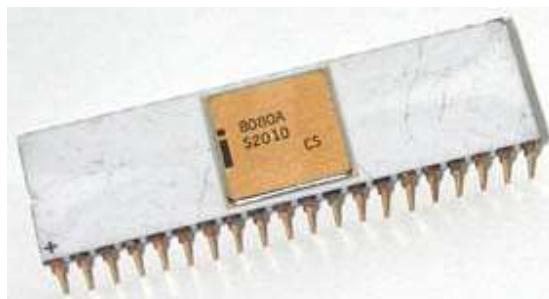
This book will confine the detailed study of 8085 microprocessor because it is most commonly used microprocessor. However, evolution of microprocessors has been discussed in this chapter, in order to have the knowledge microprocessors introduced so far. Before discussing the details of the 8085, brief discussion of 8-bit microprocessor 8080 is given here.

#### 1.4 BRIEF DESCRIPTION OF 8-BIT MICROPROCESSOR 8080A

Intel 8080 microprocessor is a successor to the Intel 8008 CPU. The Intel 8080/8080A was not object-code compatible with the 8008, but it was source-code compatible with it. The 8080 CPU had the same interrupt processing logic as the 8008, which made porting of old applications easier. Maximum memory size on the Intel 8080 was increased from 16 KB to 64 KB. The number of I/O ports was increased to 256. In addition to all 8008 instructions and addressing modes the 8080 processor included many new instructions and direct addressing mode. The 8080 also included new Stack Pointer (SP) register. The SP was used to specify position of external stack in CPU memory, and the stack could grow as large as the size of memory. Thus, the CPU was no longer limited to 7-level internal stack, like the 8008 did.

The Intel 8080, an 8-bit microprocessor was very popular. Fig. 1.2 shows the shape of the microprocessor and Fig. 1.3 shows the pin out configuration of the 8080A microprocessor. Its salient features include:

- a) A two-phase clock input Q1 and Q2
- b) 16-bit address bus
- c) 8-bit data bus
- d) Power supply input of +5V, -5V and +12V required.
- e) The 8080A places the status of the operation on the data bus during the earlier part of the cycle and places data on the bus during later part of cycle.



**Fig. 1.2**

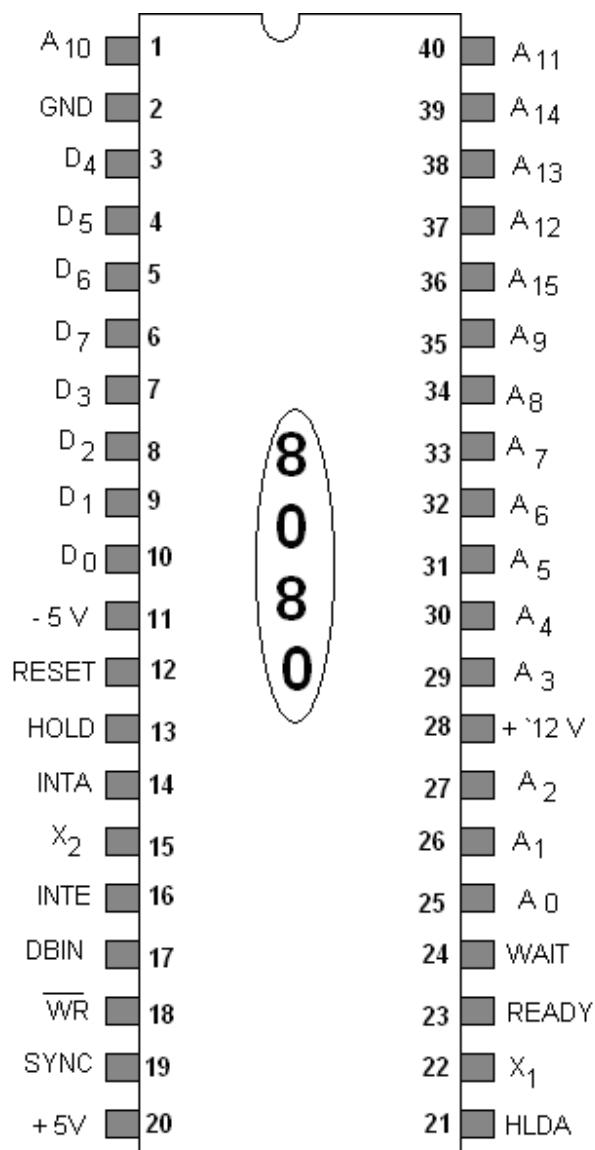


Fig. 1.3

**Program memory** Program can be located anywhere in memory. Jump, branch and call instructions use 16-bit addresses, i.e. they can be used to jump/branch anywhere within 64 KB. All jump/branch instructions use absolute addressing.

**Data memory** The processor always uses 16-bit addresses so that data can be placed anywhere.

**Stack memory** It is limited only by the size of memory. Stack grows downward.

**Interrupts** The processor supports maskable interrupts. When an interrupt occurs the processor fetches from the bus one instruction, usually one of these instructions:

- One of the 8 RST instructions (RST0 - RST7). The processor saves current program counter into stack and branches to memory location  $N * 8$  (where N is a 3-bit number from 0 to 7 supplied with the RST instruction).
- CALL instruction (3 byte instruction). The processor calls the subroutine, address of which is specified in the second and third bytes of the instruction.

The interrupt can be enabled or disabled using EI (Enable Interrupts) and DI (Disable Interrupts) instructions.

### I/O ports

256 Input ports

256 Output ports

### Registers

**Accumulator** or A register is an 8-bit register used for arithmetic, logic, I/O and load/store operations.

**Flag** is an 8-bit register contains the following five, 1-bit flags:

- Sign flag - set if the most significant bit of the result is set.
- Zero - set if the result is zero.
- Auxiliary carry flag - set if there was a carry out from bit 3 to bit 4 of the result.
- Parity flag - set if the parity (the number of set bits in the result) is even.
- Carry flag - set if there was a carry during addition, or borrow during subtraction/comparison.

### General registers:

- 8-bit B and 8-bit C registers can be used as one 16-bit BC register pair. When used as a pair the C register contains low-order byte. Some instructions may use BC register as a data pointer.
- 8-bit D and 8-bit E registers can be used as one 16-bit DE register pair. When used as a pair the E register contains low-order byte. Some instructions may use DE register as a data pointer.

- 8-bit H and 8-bit L registers can be used as one 16-bit HL register pair. When used as a pair the L register contains low-order byte. HL register usually contains a data pointer used to reference memory addresses.

**Stack pointer** It is a 16 bit register. This register is always incremented/decremented by 2.

**Program counter** It is also a 16-bit register.

### **Instruction Set**

8080 instruction set consists of the following instructions:

- Data moving instructions.
- Arithmetic - add, subtract, increment and decrement.
- Logic - AND, OR, XOR and rotate.
- Control transfer – conditional, unconditional, call subroutine, return from subroutine and restarts.
- Input/Output instructions.
- Other – setting/clearing flag bits, enabling/disabling interrupts, stack operations, etc.

## **1.5 COMPUTER PROGRAMMING LANGUAGES**

In order for computers to accept commands from human and perform tasks vital to productivity, a means of communication must exist. Programming languages provide this necessary link between man and machine. Because they are quite simple compared to human language, rarely containing more than few hundred distinct words, programming languages must contain very specific instructions. There are more than 2,000 different programming languages in existence, although most programs are written in one of several popular languages, like BASIC, COBOL, C++, or Java. Programming languages have different strengths and weaknesses. Depending on the kind of program being written, the computer will run on the experience of the programmer, and the way in which the program will be used, the suitability of one programming language over another will vary. One can categorize computer language as low Level and high level programming languages which are being discussed in the subsequent subsections.

### **1.5.1 Low-Level Programming Language**

A low-level programming language is a language that provides little or no abstraction from a computer's instruction set architecture. The word "low" refers to the small or nonexistent amount of abstraction between the language and machine language; because of this, low-level languages are sometimes described as being "close to the

hardware." A low-level language does not need a compiler or interpreter to run; the processor for which the language was written is able to run the code without using either of these.

By comparison, a high-level programming language isolates the execution semantics of computer architecture from the specification of the program, making the process of developing a program simpler and more understandable.

Low-level programming languages are sometimes divided into two categories: first generation, and second generation programming languages.

### **First Generation Programming Language**

The First-Generation Programming Language (1GL) is machine code or machine language. Machine language is a language which is directly understood by a computer. It is also called binary language as it is based on 0s or 1s. Any instruction in machine language is represented in terms of 0's and 1's, even the memory addresses are given in binary mode. Programs in machine language are very difficult to read and understand as binary codes of each command can not easily be remembered. So it is very difficult and complicated to write the computer program in machine language. The experienced programmer can only work in machine language that too after having the good knowledge of machine hardware. Programs written in machine language cannot easily be understood by other programmers. Currently, programmers almost never write programs directly in machine code, because as discussed above it not only require attention to numerous details which a high-level language would handle automatically, but it also requires memorizing or looking up numerical codes for every instruction that is used.

### **Second Generation Programming Language**

The Second-Generation Programming Language (2GL) is assembly language. Assembly language was first developed in the 1950s and it is different for different microprocessors. It was the first step to improve the computer programming. For writing the programs in assembly language it is necessary that the programmers should have the knowledge of machine hardware. The assembly language eliminated much of the error-prone and time-consuming first-generation programming needed with the earliest computers, freeing the programmer from tedious or boring jobs such as remembering numeric codes and calculating memory addresses. The assembly language was once widely used for all sorts of programming. However, by the 1980s (1990s on small computers), the use of assembly languages had largely been supplanted by high-level languages, in the search for improved programming productivity.

Assembly languages are basically a family of low-level languages for programming computers, microprocessors, microcontrollers etc. They implement a symbolic representation of the numeric machine codes and other constants needed to program a particular CPU architecture. This representation is usually defined by the hardware manufacturer, and is based on abbreviations (called mnemonics) that help the

programmer remember individual instructions, registers, etc. An assembly language is thus specific to certain physical or virtual computer architecture. Instructions (statements) in assembly language are generally very simple, unlike those in high-level languages. Generally, an opcode is a symbolic name for a single executable machine language instruction, and there is at least one opcode mnemonic defined for each machine language instruction. Each instruction typically consists of an *operation* or *opcode* plus zero or more *operands*. Most instructions refer to a single value, or a pair of values. Operands can either be immediate (typically one byte values, coded in the instruction itself) or the addresses of data located elsewhere in storage.

A typical assembly language statement of 8080A or 8085 microprocessor written by the programmer is given below, which is divided in to four fields namely, Label, Mnemonics or Operation code (Opcode), Operand and comments.

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>                |
|--------------|------------------|----------------|--------------------------------|
| START:       | LXI H,           | 2500 H         | ; Initialize H-L register pair |

A label for an instruction is optional, but it is very essential for specifying jump locations. Similarly, comments are also optional but it is required for good documentation. The four fields of assembly language statements shown above are separated by the following delimiters:

| <b>Delimiters</b> | <b>Placement</b>                                      |
|-------------------|---|
| Colon (:)         | A colon is placed after the Label. Label is optional. |
| Space ( )         | Space is left between an opcode and operand.          |
| Comma (,)         | A comma is placed between two operands.               |
| Semicolon (;)     | Semicolon is placed between the comments.             |

The program written in assembly language is converted to machine language manually. For writing the program in machine language, the starting address, where the program is to be stored should be known. Now the op code of the instruction is to be written in first location (starting address) and in the consecutive memory locations data /address of the operand is written. While storing the address in the memory locations, lower byte of the address is stored first then the upper byte.

A utility program called an assembler is used to translate assembly language statements into the target computer's machine code. The assembler performs a more or less isomorphic translation (a one-to-one mapping) from mnemonic statements into machine instructions and data. The reverse process that is conversion of machine language to the assembly language is done by deassembler.

For software development for a microprocessor/ microcomputer (written in assembly language in large number of instructions), it is absolutely essential to use an assembler. In fact assembler translates mnemonics into binary code with speed and accuracy; thus eliminating human error in looking for the opcodes. Other advantages of using the assembler for the software development are as follows:

- It assigns appropriate values to the symbols used in a program. This facilitates specifying jump locations.
- The assembler checks syntax errors, such as wrong labels and expressions, and provides error messages. However it cannot check logic errors in a program.
- It is easy to insert or delete instructions in a program; the assembler can reassemble the entire program quickly with new memory locations and modified addresses from jump locations. This avoids rewriting the program manually.

### 1.5.2 High-Level Programming Language

The machine language and assembly languages discussed above are the first and second generation programming languages which fall in the category of low level languages. These languages require deep knowledge of computer hardware. The high level computer languages developed around 1960s are machine independent i.e. computer hardware is not necessary to know for the programmers. In high level languages one has to know only the instructions in English word and logic of the problem irrespective of the types of computer being used. For the computer programming prepared in high level language, only the use of English alphabets and mathematical systems like +, -, /, \* etc. are made. In fact high level language is more close to user and is easy to read and understand. The high level languages are called procedural language and are designed to solve general and specific problems.

The term "high-level language" does not imply that the language is superior to low-level programming languages - in fact high level refers to the higher level of abstraction from machine language. They have no opcodes that can directly compile the language into machine code, unlike low-level assembly language.

In high level languages the words in English are converted into binary language of different microprocessors with the help of a program called **Interpreter** or **Compiler**. The compiler or interpreter accepts English like statements as the input called Source code. The source codes are translated into machine language compatible with the microprocessor being used in the machine. The translation in the machine language from the source code is called the object code. Figure 1.4 shows the block diagram for translation of high level language program into machine code. Each microprocessor needs its own compiler or an interpreter for each high level language.



**Fig. 1.4**

The difference between a compiler and an interpreter is that the compiler reads the entire program first and then generates the object code; whereas the interpreter reads one instruction at a time and produces its object code which is executed at the same time before reading the next instruction. The high level programming language developed so far may be categorized into third, fourth and fifth generation programming languages whose brief discussion is given below.

### Third Generation Programming Language

A third-generation programming language (3GL) is a refinement of a second-generation programming language. Whereas a second generation language is more aimed to fix logical structure to the language, a third generation language aims to refine the usability of the language in such a way to make it more user friendly. A third generation language improves over a second generation language by having more refinement on the usability of the language itself from the perspective of the user.

Languages like ALGOL, COBOL, FORTRAN IV etc. are examples of this generation and were considered as high level languages. Most of these languages had compilers and the advantage of this was speed. Independence was another factor as these languages were machine independent and could run on different machines. FORTRAN (FORmula TRANslating) and COBOL (CComputer Business Oriented Language) were the first high-level programming languages to make an impact on the field of computer science. Along with assembly language, these two high-level languages have influenced the development of many modern programming languages, including Java, C++, and BASIC.

FORTRAN is well suited for math, science, and engineering programs because of its ability to perform numeric computations. The language was developed in New York by IBM's John Backus. FORTRAN was known as user's friendly, because it was easy to learn in a short period of time and required no previous computer knowledge. It eliminated the need for engineers, scientists, and other users to rely on assembly programmers in order to communicate with computers. Although FORTRAN is often referred to as a language of the past, computer science students were still taught the language in the early 2000s for historical reasons, and because FORTRAN code still exists in some applications.

COBOL was another high level and third generation programming language well suited for creating business applications. COBOL's strength is in processing data, and in its simplicity.

Other languages like BASIC, C, C++, C#, Pascal, and Java are also third-generation languages.

### **Fourth-Generation Programming Language**

A fourth-generation programming languages (4GL) (1970s-1990) are the programming language designed with a specific purpose in mind, such as the development of commercial business software. In the evolution of computing, the fourth generation language followed the third generation language in an upward trend toward higher abstraction and statement power. The fourth generation language was followed by efforts to define and use a fifth generation language (5GL). Basically the fourth generation languages are languages that consist of statements similar to statements in a human language. Fourth generation languages are commonly used in database programming and scripts. The commonly used fourth generation languages are FoxPro , SQL , MATLAB etc.

### **Fifth-Generation Programming Language**

A fifth-generation programming language (5GL) is a programming language based around solving problems using constraints given to the program, rather than using an algorithm written by a programmer. Most constraint-based and logic programming languages and some declarative languages are fifth-generation languages.

While fourth-generation programming languages are designed to build specific programs, fifth-generation languages are designed to make the computer solve a given problem without the programmer. This way, the programmer only needs to worry about what problems need to be solved and what conditions need to be met, without worrying about how to implement a routine or algorithm to solve them. Fifth-generation languages are used mainly in artificial intelligence research. Prolog, OPS5, Visual Basic, and Mercury etc. are examples of fifth-generation languages.

### **Problems**

- 1.1 Draw the block diagram of a general computer and discuss in detail the five blocks of a digital computer.
- 1.2 Discuss History and Evolution of Microprocessor.
- 1.3 What is microprocessor? What is the difference between a microprocessor and a microcomputer?
- 1.4 What is the difference between the 4-bit microprocessor and 8-bit microprocessor?
- 1.5 Name the main 8-bit microprocessors. Give the brief description of 8-bit microprocessor 8080A.

- 1.6 Discuss basic microprocessor system with the help of block diagram.
  - 1.7 What are low level computer programming languages? Discuss them.
  - 1.8 Discuss first generation computer programming languages.
  - 1.9 Discuss second generation computer programming languages.
  - 1.10 Write short note on high level computer programming languages.
  - 1.11 What is the difference between assembly language and machine language?
  - 1.12 Mention the brief description of Assembly Language. What are the advantages of assembler?
-

# 2

## SAP – I

---

For the beginners it is very difficult to understand the operation of a digital computer as it contains large details. To understand the step by step operation of a digital computer, the concept of Simple as Possible (SAP) computer has been introduced. Simple as possible computer, a conceptual computer will be discussed in three stages namely SAP – I, SAP – II and SAP – III computers. All the necessary details for the operation of digital computers will be discussed in three stages. After the study of these three stages of SAP computers, we will be in a position to understand clearly the fundamentals of microprocessor 8085 including the architecture, programming and interfacing devices. In this chapter the organization, programming and circuits of SAP – I computer will be discussed; the succeeding chapters will have the details of other stages of SAP computers.

### 2.1 ARCHITECTURE OF SAP - I

SAP – I, a conceptual computer is an 8-bit computer, as it can process the data of 8-bits. Further it is a simple computer and is considered for the basic understanding of the operation of digital computers, so it is assumed that it can store only 16 words (each word being 8 bit long). The length of the memory address register will be 4 bits since  $2^4 = 16$  (16 is the total capacity of the memory unit). The address of 16 memory locations of memory address register (MAR) will be 0000 to 1111 i.e.

| Memory locations<br>(in Hexadecimal) | Address | Memory locations<br>(in Hexadecimal) | Address |
|--------------------------------------|---------|--------------------------------------|---------|
| 0 H                                  | 0000    | 8 H                                  | 1000    |
| 1 H                                  | 0001    | 9 H                                  | 1001    |
| 2 H                                  | 0010    | A H                                  | 1010    |
| 3 H                                  | 0011    | B H                                  | 1011    |
| 4 H                                  | 0100    | C H                                  | 1100    |
| 5 H                                  | 0101    | D H                                  | 1101    |
| 6 H                                  | 0110    | E H                                  | 1110    |
| 7 H                                  | 0111    | F H                                  | 1111    |

The basic architecture of this computer is shown in figure 2.1. It contains an 8-bit W-Bus (Wire Bus), which is used for data transfer to various 8-bit registers. A Bus is a group of conducting wires. In this figure, all register outputs connected to W-Bus are three-state which allows ordinary transfer of data. Remaining other register outputs continuously drive the boxes they are connected to.

A brief discussion of each block is given below:

## Program Counter

First block of the SAP -I computers is the Program Counter (PC). It is basically the part of the control unit, its function is to send to the memory the address of the next instruction to be fetched and executed. Program counter is also called as the pointer as it is like someone pointing a finger at a list of instructions, saying do this first and do this next and so on.

In the beginning, program and data is stored in the memory through the input unit. A 4-bit binary address (0000 to 1111) is sufficient to address a word in the memory. The first instruction of the program is stored in the memory location 0000, second instruction at 0001 location, and the third instruction at 0010 location and so on. When the computer starts executing the program, the program counter is reset. The program counter is then incremented by one to get the next address of the memory location. After the first instruction is fetched and executed, the PC sends address 0001 to memory. Again the PC is incremented by one. After execution of second instruction, PC sends the next address to memory and so on.

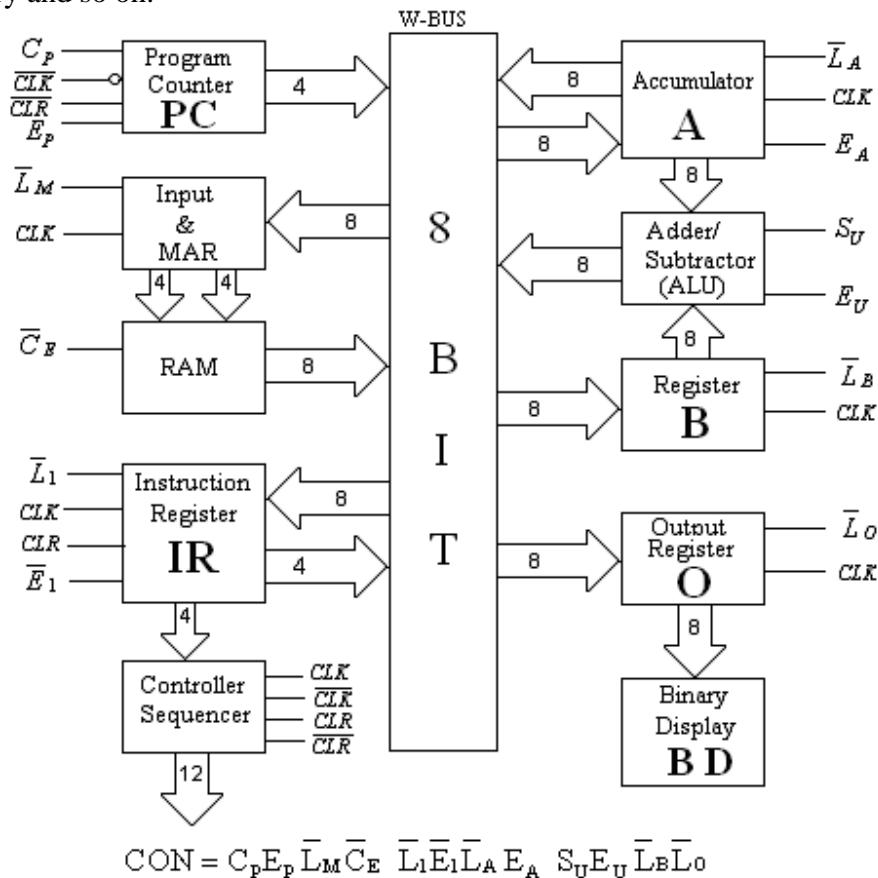


Fig. 2.1

## Input and MAR

The second block of SAP -I computer is Input and Memory Address Register (MAR). It includes 4 bit address register and 8 bit data register. These registers are basically the parts of input unit. The input and MAR sends 4 bit address and 8 bit data to memory unit, this unit helps in storing the instructions and data to the memory, before the computer run starts. This unit includes a matrix of switches (micro-switches) for address

and data. In this SAP-I computer, the switch matrix allows to send 4 address bits and 8 data bits to memory. The relevant switch is opened or closed to program the memory. This can be done by providing switches on the front panel of the computer.

The memory address register is also the part of SAP-I memory. When SAP-I computer starts executing the program, the address in the PC is latched into the MAR. The MAR will then send this address to RAM for the read operation.

### **Memory Unit**

It is 16x8 static TTL RAM (Random Access Memory) i.e. it is capable of storing 16 words (total capacity is only of 16 words) and each word is of 8 bit long. It stores the program and data before the execution of program; and during the execution it sends the stored instruction and data to the W-bus for further transfer to other registers as soon as address is received by it from MAR.

### **Instruction Register**

The instruction register of SAP-I computer is the part of control unit. As already discussed during the execution of program, the content of addressed memory location is placed on the W-Bus. At the same time, during the next positive edge of the clock pulse this instruction (content of memory location) is loaded into the Instruction Register. The instruction register now splits the content of the instruction (8-bit) into two nibbles (one nibble is of 4-bit long). Instruction register sends the upper nibble directly to Controller/Sequencer, the lower nibble is, however, placed by the Instruction register to W-Bus for the read operation whenever needed.

### **Controller Sequencer**

The block controller sequencer is basically control unit. This is a key unit for the automatic operation of this SAP-I computer. It generates a control word of 12 bits as given below:

$$CON = C_P E_P \bar{L}_M \bar{C}_E \bar{L}_I \bar{E}_I \bar{L}_A E_A S_U E_U \bar{L}_B \bar{L}_O$$

The symbolic notations of the control signal (CON) used in the computer are given in table 2.1.

**Table 2.1**

| Notation    | Interpretation                                 |
|-------------|--|
| $C_P$       | PC is incremented, when $C_P$ is high.         |
| $E_P$       | Enable PC, when $E_P$ is high.                 |
| $\bar{L}_M$ | Loads MAR from W-Bus, when $\bar{L}_M$ is low. |
| $\bar{C}_E$ | Loads RAM, when $\bar{C}_E$ is low.            |
| $\bar{L}_I$ | Loads instruction register, when it is low.    |
| $\bar{E}_I$ | Enable instruction register, when it is low.   |

|             |   |
|-------------|---|
| $\bar{L}_A$ | Loads the accumulator, when it is low.                                    |
| $E_A$       | Enable the accumulator, when it is high.                                  |
| $S_U$       | Enables subtraction, when it is high and enables addition when it is low. |
| $E_U$       | Enable Adder/Subtractor to send the answer to W-Bus, when it is high.     |
| $\bar{L}_B$ | Loads the register B, when it is low.                                     |
| $\bar{L}_O$ | Loads the output register, when it is low.                                |

### Accumulator

The accumulator or register A is an 8-bit register and it is also called the buffer register. When  $\bar{L}_A$  is low, the data from the W-Bus is loaded to the accumulator at the positive edge of the clock pulse. This data also directly goes to the Adder/Subtractor. The answer of Adder/Subtractor may also be loaded to the accumulator via W-Bus. The answer or the data stored in Accumulator will go to W-Bus when  $E_A$  is high.

### Adder-Subtractor

The adder-Subtractor is the part of Arithmetic and Logical Unit (ALU) of the computer. This block can add the content of B-register to accumulator content when  $S_U$  is low. Similarly, this block can subtract the content of B-register from the accumulator content when  $S_U$  is high. The answer of addition or subtraction may be loaded to W-Bus when  $E_U$  is high. The subtraction in the block adder/subtrator is done using 2's complement method.

### Register B

The register B is the part of Arithmetic and Logical Unit (ALU) of the computer. It is another buffer register of 8 bit long. The content to be added to the accumulator or subtracted from the accumulator, is loaded to B-register from W-Bus when  $\bar{L}_B$  is low.

### Output Register O

The output register **O** is the part of the output unit of the SAP-I computer. At the end of the execution of program the answer available at the accumulator may be transferred to output register **O**, at the positive edge of the clock pulse and when  $\bar{L}_O$  is low, this register is also called the output port.

### Binary Display (BD)

The last block of SAP-I computer is Binary Display (BD), which is the part of output unit. This display unit contains 8 LEDs (Light Emitting Diodes) connected to 8 bit output port through 8 flip-flops. When the data or answer is available at the output port, the same is transferred to LEDs indicating the answer is in binary form.

## 2.2 INSTRUCTION SET OF SAP-I COMPUTER

The instruction set is a set of instructions that can be executed by the computer. The computer programming is done using these instructions. Any problem to be solved on the computer is to be written in the form of a program. The SAP-I computer has got only five instructions as it is a very simple computer. These instructions are given in table 2.2. The instructions are written in abbreviated form or short form. These short forms of the instructions are known as Mnemonics (Memory aids). In fact the instructions in short form can easily be remembered by the programmers or users. Further, every instruction is stored in computer in coded form known as Op Code (Operation Code). The Op codes for the instructions of SAP-I computer given in table 2.2 are shown in binary form. From the above discussion it is clear that mnemonics is the short form of instruction for the user's remembrance and op code is the coded form of the instruction in machine language.

**Table 2.2**

| Mnemonic | Op Code<br>(Binary) | Operation   |
|----------|---------------------|---|
| LDA      | 0000                | It loads the accumulator the content from addressed location.                 |
| ADD      | 0001                | It adds the content of the memory location to the accumulator content.        |
| SUB      | 0010                | It subtracts the content of the memory location from the accumulator content. |
| OUT      | 1110                | It transfers the accumulator content to the output port.                      |
| HLT      | 1111                | It stops the computer for further execution.                                  |

#### **LDA Instruction**

The LDA instruction loads the content from the specified memory location to the accumulator. There is always an operand with this instruction. The operand with the LDA instruction is the address of the memory location whose content is to be transferred to the accumulator.

For example

LDA 9 H

will load the accumulator, the content from the memory location 9 H. The alphabet H denotes hexadecimal number, as the op code for LDA is 0000 and binary equivalent of 9H is 1001. The machine language for this instruction is

0000 1001.

It must be remembered that LDA 9 H is known as the assembly language of the instruction and 0000 1001 (09H) is known as machine language of the instruction.

If the memory location 9H has the content 0001 0010 (12 H), then after execution of the instruction in SAP-I computer, it will load the content 0001 0010 to the accumulator, i.e.

$$A \leftarrow 00010010 \quad \text{or} \quad A \leftarrow 12H$$

#### **ADD Instruction**

ADD instruction adds the content of memory location to the accumulator content. The address of the memory location is the operand of this instruction.

For example

ADD E H

adds the content of memory location E H (1110) to the accumulator content.

If before the execution of this instruction accumulator A is having data say 00010101, and 0000 0001 is already stored in memory location E H, then after execution of the instruction ADD E H the accumulator is loaded with the data 0001 0110,

$$\text{as } 0000\ 10101 + 0000\ 0001 = 0001\ 0110.$$

$$\text{i.e. } A \leftarrow 00010110 \quad \text{or} \quad A \leftarrow 16H$$

### SUB Instruction

SUB instruction subtracts the content of the memory location from the accumulator content. The operand for this instruction is the address of the memory location.

For example

SUB C H

subtracts the content of memory location C H (1100) from the accumulator content.

If before the execution of this instruction accumulator A is having the content say 0000 0011 and the content in memory location C H is 0000 0010, then after the execution of this instruction, the accumulator A will have the answer as:

$$0000\ 0011 - 0000\ 0010 = 0000\ 0001$$

$$\text{So } A \leftarrow 00000001 \quad \text{or} \quad A \leftarrow 01H$$

### OUT Instruction

The SAP-I instructions LDA, ADD and SUB discussed above are the memory reference instructions since the address of the memory location is the operand for these instructions. OUT instruction is not the memory reference instruction, as this instruction is itself complete and operand is not needed. The OUT instruction is used to transfer the accumulator content (answer after processing) to the output port.

### HLT Instruction

HLT instruction is also not memory reference instruction, as it is complete itself and no operand is used with this instruction. HLT instruction stops further processing of the computer. This instruction must be used as the last instruction of every program otherwise meaningless answer will be obtained.

The complete assembly and machine code (Binary and Hexadecimal) of all the operations are shown in table 2.3.

Table 2.3

| <b>Assembly Code</b> | <b>Machine Code</b> |            | <b>Operations</b>  |
|----------------------|---------------------|------------|--|
|                      | <b>Binary</b>       | <b>Hex</b> |  |
| LDA 5H               | 0000 0101           | 05 H       | Loads the Acc., the content of memory location 5 H.                |
| ADD CH               | 0001 1100           | 1C H       | Adds the content of memory location CH to Acc. content.            |
| SUB BH               | 0010 1011           | 2B H       | Subtracts the content of memory location BH from the Acc. content. |
| OUT                  | 1110 xxxx           | Ex H       | Sends Acc. content to the output port.                             |
| HLT                  | 1111 xxxx           | Fx H       | Stops the processing.  |

## 2.3 PROGRAMMING OF SAP-I COMPUTER

Programming means set of instructions written by the user or programmer for performing a particular task. These instructions are then fed to the computer to get the

desired result. The program for the particular task is written by the programmer in assembly language (in mnemonic form) and then fed to the computer in the consecutive memory locations in the form of op codes (machine language) along with data. The program written in mnemonics form or assembly language is also called Source Program and the program written in machine language is known as Object Program. It will now be illustrated in more detail by taking an example.

Suppose we wish to get the addition of three numbers (decimal) 2, 3 and 5. Let these numbers are stored in three different memory locations 6, 7 and 8 respectively. Following steps will be taken for its execution:

|          |   |
|----------|---|
| Step-I   | First number from the memory location 6 should be transferred to the accumulator (LDA 6 H).   |
| Step-II  | Second number from memory location 7 should be added to the accumulator and sum of these two numbers should remain in the accumulator (ADD 7 H).  |
| Step-III | Third number from memory location 8 should now be added to the accumulator content and final sum should also remain in the accumulator (ADD 8 H). |
| Step-IV  | Final answer (accumulator content) should be sent to the output port for the display in binary form (OUT).  |
| Step-V   | Stop the processing of the computer (HLT).  |

The program is now fed in the memory locations starting at 0 H and data in the memory locations starting at 6 H, as:

| <b>Memory location<br/>(in Hex)</b> | <b>Mnemonic</b> |
|-------------------------------------|-----------------|
|-------------------------------------|-----------------|

|     |         |
|-----|---------|
| 0 H | LDA 6 H |
| 1 H | ADD 7 H |
| 2 H | ADD 8 H |
| 3 H | OUT     |
| 4 H | HLT     |
| 5 H | xx H    |
| 6 H | 02 H    |
| 7 H | 03 H    |
| 8 H | 05 H    |

This program in machine language is given as:

| <b>Memory location</b> | <b>Op code</b> |
|------------------------|----------------|
|------------------------|----------------|

|      |           |
|------|-----------|
| 0000 | 0000 0110 |
| 0001 | 0001 0111 |
| 0010 | 0001 1000 |
| 0011 | 1110 xxxx |
| 0100 | 1111 xxxx |
| 0101 | xxxx xxxx |
| 0110 | 0000 0010 |

|      |           |
|------|-----------|
| 0111 | 0000 0011 |
| 1000 | 0000 0101 |

Note: x – sign represents any binary digit 0 or 1.

**Example 2.1** Write an assembly language program that performs the following operation in SAP-I computer.

$$7 + 6 - 4 + 5 - 3$$

Use memory locations 7H to BH for the data. Further convert this program in machine language.

**Solution.** SAP-I assembly language program of the given problem is as given below:

| Memory location<br>(in Hex) | Mnemonic |
|-----------------------------|----------|
|-----------------------------|----------|

|     |         |
|-----|---------|
| 0 H | LDA 7 H |
| 1 H | ADD 8 H |
| 2 H | SUB 9 H |
| 3 H | ADD A H |
| 4 H | SUB B H |
| 5 H | OUT     |
| 6 H | HLT     |
| 7 H | 07 H    |
| 8 H | 06 H    |
| 9 H | 04 H    |
| A H | 05 H    |
| B H | 03 H    |

This program in machine language is given by:

| Memory location | Op code |
|-----------------|---------|
|-----------------|---------|

|      |           |
|------|-----------|
| 0000 | 0000 0111 |
| 0001 | 0001 1000 |
| 0010 | 0010 1001 |
| 0011 | 0001 1010 |
| 0100 | 0010 1011 |
| 0101 | 1110 xxxx |
| 0110 | 1111 xxxx |
| 0111 | 0000 0111 |
| 1000 | 0000 0110 |
| 1001 | 0000 0100 |
| 1010 | 0000 0101 |
| 1011 | 0000 0011 |

**Example 2.2** Write an assembly language program to calculate the following expression on SAP-I computer:

$$Y + 2Z - 3W$$

The data Y, Z and W (as 6, 7 and 2) are stored in memory locations 8H to AH.

**Solution.**

SAP-I assembly language program of the given problem is as given below:

| <b>Memory location<br/>(in Hex)</b> | <b>Mnemonic</b> |
|-------------------------------------|-----------------|
| 0 H                                 | LDA 8 H         |
| 1 H                                 | ADD 9 H         |
| 2 H                                 | ADD 9 H         |
| 3 H                                 | SUB A H         |
| 4 H                                 | SUB A H         |
| 5 H                                 | SUB A H         |
| 6 H                                 | OUT             |
| 7 H                                 | HLT             |
| 8 H                                 | 06 H            |
| 9 H                                 | 05 H            |
| A H                                 | 02 H            |

**Example 2.3** Write assembly program for SAP-I computer to solve the following problem:

$$17 + 22 + 20 - 33.$$

The numbers are given in decimal. Also give the machine language program.

#### **Solution.**

First the decimal numbers are converted to Hexadecimal numbers. The Hexadecimal equivalents of these numbers are given as:

$$17 = 11 \text{ H}$$

$$22 = 16 \text{ H}$$

$$20 = 14 \text{ H}$$

$$33 = 21 \text{ H}$$

Let us assume that these numbers are stored in memory locations 7H to AH.

| <b>Memory location<br/>(in Hex)</b> | <b>Mnemonic</b> |
|-------------------------------------|-----------------|
| 0 H                                 | LDA 7 H         |
| 1 H                                 | ADD 8 H         |
| 2 H                                 | ADD 9 H         |
| 3 H                                 | SUB A H         |
| 4 H                                 | OUT             |
| 5 H                                 | HLT             |
| 6 H                                 | x x H           |
| 7 H                                 | 11 H            |
| 8 H                                 | 16 H            |
| 9 H                                 | 14 H            |
| A H                                 | 21 H            |

The machine language program is given as follows:

| <b>Memory location</b> | <b>Op code</b> |
|------------------------|----------------|
| 0000                   | 0000 0111      |
| 0001                   | 0001 1000      |
| 0010                   | 0001 1001      |
| 0011                   | 0010 1010      |
| 0100                   | 1110 xxxx      |

|      |           |
|------|-----------|
| 0101 | 1111 xxxx |
| 0110 | xxxx xxxx |
| 0111 | 0001 0001 |
| 1000 | 0001 0110 |
| 1001 | 0001 0100 |
| 1010 | 0010 0001 |

## **2.4 WORKING OF SAP-I COMPUTER**

For the working of SAP-I computer, instructions are fetched from RAM and then executed one by one in a sequence till a Halt instruction is executed. The SAP-I computer executes program starting from 0 H memory location. For this controller / sequencer unit generates a CLK signal for all the register and CLR signal for the Program counter. As soon as the computer starts executing the program, program counter receives a CLR signal which resets the program counter. The program counter will send the address of 0 H memory location.

In SAP-I computer the loading of a register takes place only when setup and hold times of the clock pulse are satisfied. For this 50% duty cycle clock pulse is used and positive edge occurs half way through each state. Waiting half a cycle before loading the register satisfies setup time; waiting half a cycle after loading satisfies the hold time. Secondly, the reason for waiting half a cycle before loading a register is that when enable input of the sending register goes active, the content of the register suddenly dumped on to register. Stray capacitance and lead inductance may, however, prevent the voltage level immediately. In other words the transients on the W-Bus will be eliminated if the clock has the wait time to die out the transients and valid data is transferred. The control word generated by the controller sequencer unit does the automatic operation of the computer. The execution of a program is carried out by fetching and execution operations of the instructions.

The instruction cycle for the execution of an instruction consists of the following two cycles:

- |      |                 |                                  |
|------|-----------------|----------------------------------|
| (i)  | Fetch cycle     | fetches a word from memory       |
| (ii) | Execution cycle | executes the fetched instruction |

We will now discuss these cycles in detail.

### 2.4.1 Fetch Cycle

The operation of fetch cycle is performed in three states namely:

- (a) Address State
  - (b) Increment State
  - (c) Memory State

During the address state, the content of PC is transferred to MAR via W-Bus:

MAR  $\leftarrow$  PC

During the increment state, the PC gets incremented. The incremented content will be sent to MAR when next fetch cycle occurs.

$\text{PC} \leftarrow \text{PC} + 1$

During the memory state, memory read operation is performed.

$\mathbf{W-Bus} \leftarrow \mathbf{M}_{\text{BS}}$

The control word generated by the controller sequencer for these three states is given below:

|                | $C_p$ | $E_p$ | $\bar{L}_M$ | $\bar{C}_E$ | $\bar{L}_1$ | $\bar{E}_1$ | $\bar{L}_A$ | $E_A$ | $S_U$ | $E_U$ | $\bar{L}_B$ | $\bar{L}_O$ |
|----------------|-------|-------|-------------|-------------|-------------|-------------|-------------|-------|-------|-------|-------------|-------------|
| T <sub>1</sub> | 0     | 1     | 0           | 1           | 1           | 1           | 1           | 0     | 0     | 0     | 1           | 1           |
| T <sub>2</sub> | 1     | 0     | 1           | 1           | 1           | 1           | 1           | 0     | 0     | 0     | 1           | 1           |
| T <sub>3</sub> | 0     | 0     | 1           | 0           | 0           | 1           | 1           | 0     | 0     | 0     | 1           | 1           |

### Address State

Only  $E_p$  and  $\bar{L}_M$  are active. Figure 2.2 shows shaded boxes as active.

### Increment State

Only  $C_p$  is active. Figure 2.3 shows shaded boxes as active.

### Memory State

Only  $C_E$  and  $\bar{L}_1$  are active. Figure 2.4 shows shaded boxes as active.

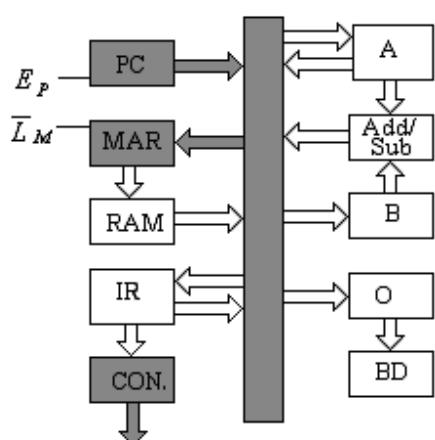


Fig. 2.2

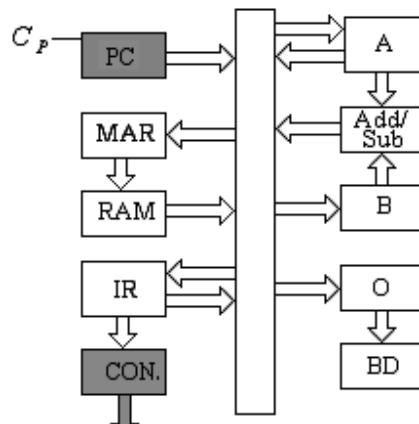


Fig. 2.3

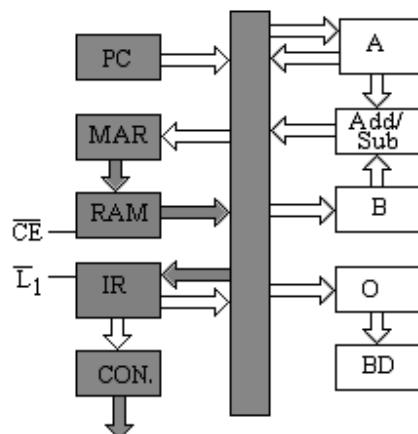
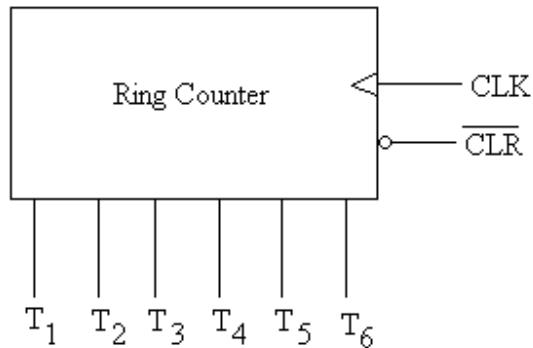
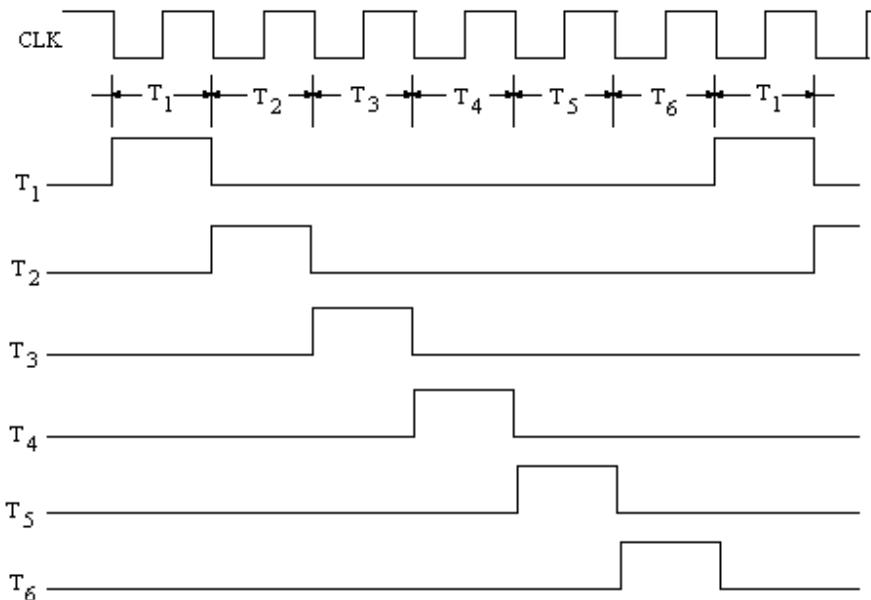


Fig. 2.4



**Fig. 2.5(a)**



**Fig. 2.5(b)**

The Controller-sequencer keeps a track of three different states of Fetch cycle and generates control signals accordingly. A ring counter is used for this purpose.

Consider a 6-bit synchronous ring counter as shown in figure 2.5(a). The output T of the ring counter is given by:

$$T = T_6 \ T_5 \ T_4 \ T_3 \ T_2 \ T_1$$

At the start of the computer run, 6-bit output of the ring counter at the successive clock pulse is shown in figure 2.5(b).

The details of the same are given below:

|                               | T <sub>6</sub> | T <sub>5</sub> | T <sub>4</sub> | T <sub>3</sub> | T <sub>2</sub> | T <sub>1</sub> |
|-------------------------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 1 <sup>st</sup> clock pulse = | 0              | 0              | 0              | 0              | 0              | 1              |
| 2 <sup>nd</sup> clock pulse = | 0              | 0              | 0              | 0              | 1              | 0              |
| 3 <sup>rd</sup> clock pulse = | 0              | 0              | 0              | 1              | 0              | 0              |
| 4 <sup>th</sup> clock pulse = | 0              | 0              | 1              | 0              | 0              | 0              |

|                               |             |
|-------------------------------|-------------|
| 5 <sup>th</sup> clock pulse = | 0 1 0 0 0 0 |
| 6 <sup>th</sup> clock pulse = | 1 0 0 0 0 0 |
| 7 <sup>th</sup> clock pulse = | 0 0 0 0 0 1 |
| and so on.                    |             |

The different outputs are also called T-states or timing states. During first three clock pulses the operation of fetch cycle occurs which are explained below:

During T<sub>1</sub> state of the timing signal, Controller-sequencer generates a signal so that E<sub>P</sub> is high and  $\bar{L}_M$  is low which sends the content of PC (address of the instruction stored in RAM) to W-Bus and at the positive edge of the clock pulse (midway of the clock pulse) the content of the W-Bus are latched into MAR due to low  $\bar{L}_M$  – **This state is called Address State of the fetch cycle.**

During T<sub>2</sub> state of the timing signal, controller sequencer generates a signal giving high to C<sub>P</sub>. At the positive edge of the clock pulse (midway of the clock pulse), PC advances the count by 1 – **This state is called Increment State of the fetch cycle.**

During T<sub>3</sub> stat, control signal generated by Controller-sequencer makes  $\bar{C}_E$  and  $\bar{L}_1$  low. The addressed RAM word is transferred to W-Bus and at positive edge of CLK (midway of clock) the data is transferred to W-Bus and then loaded to Instruction register – **This state is called Memory State of the fetch cycle.**

The next three states of clock pulse are called the execution cycle of SAP-I instructions.

#### 2.4.2 Execution Cycle

After the instruction is fetched and transferred to the Instruction Register (IR) during the fetch cycle, the IR splits the instruction word (8-bit) into two nibbles. The upper nibble goes directly to the controller-sequencer, where it is decoded and accordingly the control word is generated to act as per the direction of the instruction. The decoded output will be different for different instructions (LDA, ADD, SUB, OUT, and HLT). So during the execution cycle (T<sub>4</sub> to T<sub>6</sub> of the clock pulse), the operation of control signal will be different for different instructions. Now we will discuss these operations for each cycle.

##### Execution Cycle of LDA Instruction:

For the discussion of the operation LDA instruction, let us assume the instruction is      LDA 6 H = 0 0 0 0 0 1 1 0

In this instruction 0 0 0 0 is the op code of LDA and 6 H (0 1 1 0) is the address of the location where the data is stored.

During T<sub>3</sub> state of CLK, 0000 0110 is loaded to Instruction Register (IR). The upper nibble 0000 directly goes to controller-sequencer, where it is decoded. During T<sub>4</sub>, T<sub>5</sub> and T<sub>6</sub> states for the execution of LDA instruction, the controller-sequencer will generate the following control words in sequence:

|                | C <sub>P</sub> | E <sub>P</sub> | $\bar{L}_M$ | $\bar{C}_E$ | $\bar{L}_1$ | $\bar{E}_1$ | $\bar{L}_A$ | E <sub>A</sub> | S <sub>U</sub> | E <sub>U</sub> | $\bar{L}_B$ | $\bar{L}_O$ |  |
|----------------|----------------|----------------|-------------|-------------|-------------|-------------|-------------|----------------|----------------|----------------|-------------|-------------|--|
| T <sub>4</sub> | 0              | 0              | 0           | 1           | 1           | 0           | 1           | 0              | 0              | 0              | 1           | 1           | MAR $\leftarrow$ IR <sub>LNIBBLE</sub> |
| T <sub>5</sub> | 0              | 0              | 1           | 0           | 1           | 1           | 0           | 0              | 0              | 0              | 1           | 1           | A $\leftarrow$ M(MAR)                  |
| T <sub>6</sub> | 0              | 0              | 1           | 1           | 1           | 1           | 1           | 0              | 0              | 0              | 1           | 1           | NO OPERATION                           |

During T<sub>4</sub> state of CLK, lower nibble already available at the W-Bus is loaded into MAR since  $\bar{E}_1$  and  $\bar{L}_M$  are active.

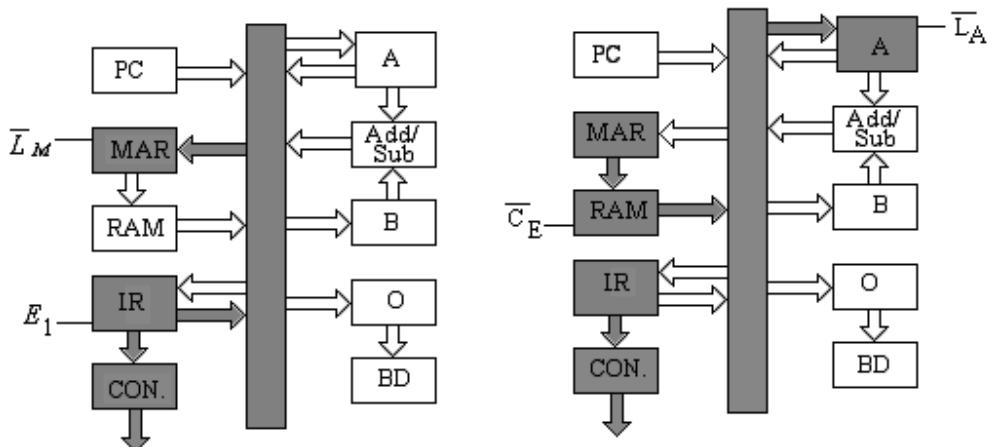
During T<sub>5</sub> state of CLK,  $\bar{C}_E$  and  $\bar{L}_A$  are active which loads the addressed content from RAM to Accumulator. This was the function of LDA instruction.

During T<sub>6</sub> state of CLK, no operation is performed as nothing was left for this instruction to do.

$T_4$  to  $T_6$  states of LDA instruction are shown in figure 2.6 (a to c) with active parts as shaded boxes and figure 2.7 shows timing diagram for Fetch and execution cycle of LDA instruction.

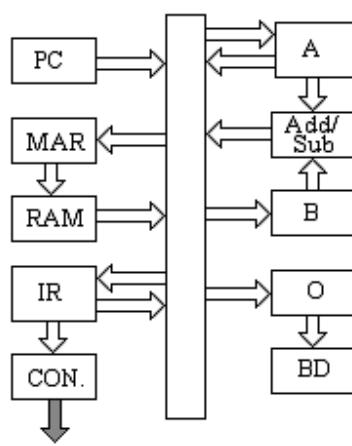
The control word generated for each T-state is called as Microinstruction. One of the instructions in instruction set is known as Macroinstruction. The microinstruction for T<sub>4</sub>, T<sub>5</sub> and T<sub>6</sub> states of LDA instruction is given below:

| <b>States</b>  | <b>Microinstruction</b> | <b>(12 Bit Control Word)</b> |
|----------------|-------------------------|------------------------------|
| T <sub>4</sub> | 1A3 H                   | 0001 1010 0011               |
| T <sub>5</sub> | 2C3 H                   | 0010 1100 0011               |
| T <sub>6</sub> | 3E3 H                   | 0011 1110 0011               |



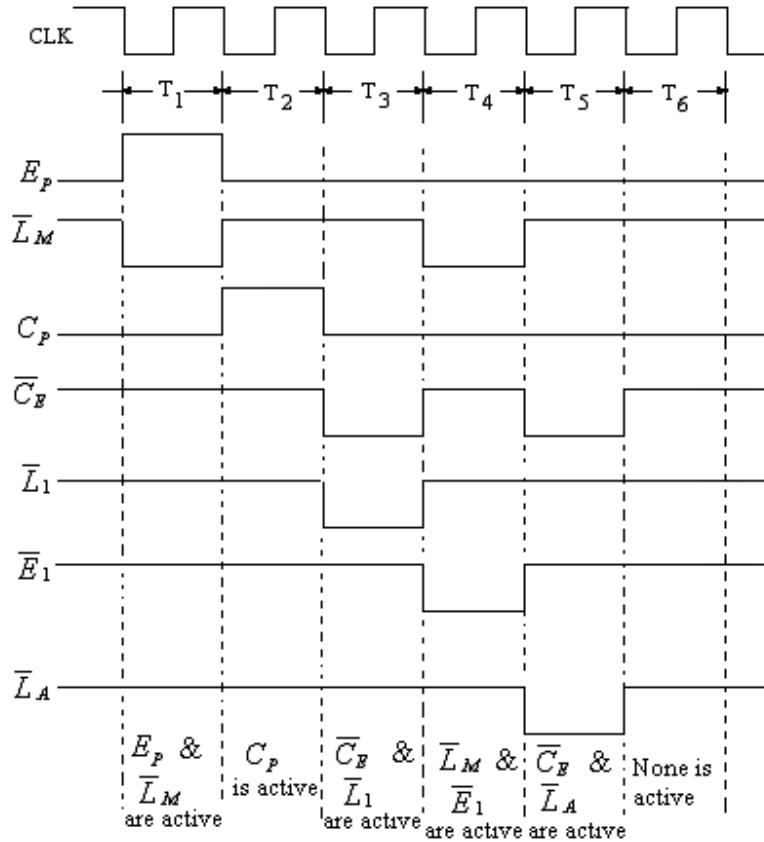
(a)

(b)



(c)

Fig. 2.6



**Fig. 2.7**

#### Execution Cycle of ADD Instruction:

For the execution of ADD instruction following control signals in sequence during T<sub>4</sub>, T<sub>5</sub> and T<sub>6</sub> states will be observed:

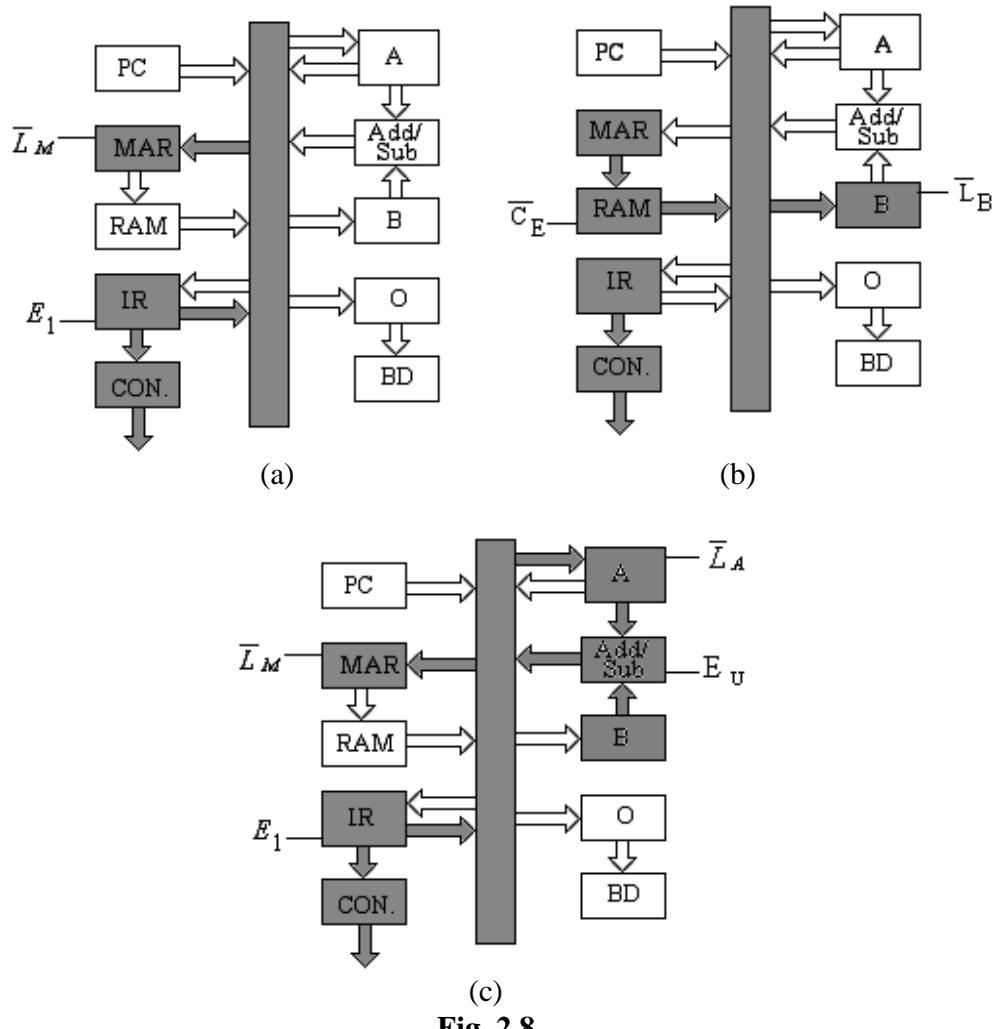
|                | C <sub>P</sub> | E <sub>P</sub> | LM | C <sub>E</sub> | L | E <sub>1</sub> | L <sub>A</sub> | E <sub>A</sub> | S <sub>U</sub> | E <sub>U</sub> | L <sub>B</sub> | L <sub>O</sub> |  |  |
|----------------|----------------|----------------|----|----------------|---|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--|--|
| T <sub>4</sub> | 0              | 0              | 0  | 1              | 1 | 0              | 1              | 0              | 0              | 0              | 1              | 1              |  | MAR $\leftarrow$ IR <sub>LNIBBLE</sub> |
| T <sub>5</sub> | 0              | 0              | 1  | 0              | 1 | 1              | 1              | 0              | 0              | 0              | 0              | 1              |  | B $\leftarrow$ M(MAR)                  |
| T <sub>6</sub> | 0              | 0              | 1  | 1              | 1 | 1              | 0              | 0              | 0              | 1              | 1              | 1              |  | A $\leftarrow$ SUM                     |

During T<sub>4</sub> state, the address part of ADD instruction moves to MAR as  $\bar{L}_M$  and  $\bar{E}_1$  are active. The data from MAR is fetched and loaded into register B during T<sub>5</sub> state as  $\bar{C}_E$  and  $\bar{L}_B$  are active. The adder-subtractor adds the content of Accumulator and register B as S<sub>U</sub> is inactive. During T<sub>6</sub> state the answer (Sum) is loaded to accumulator, as E<sub>U</sub> and  $\bar{L}_A$  are active.

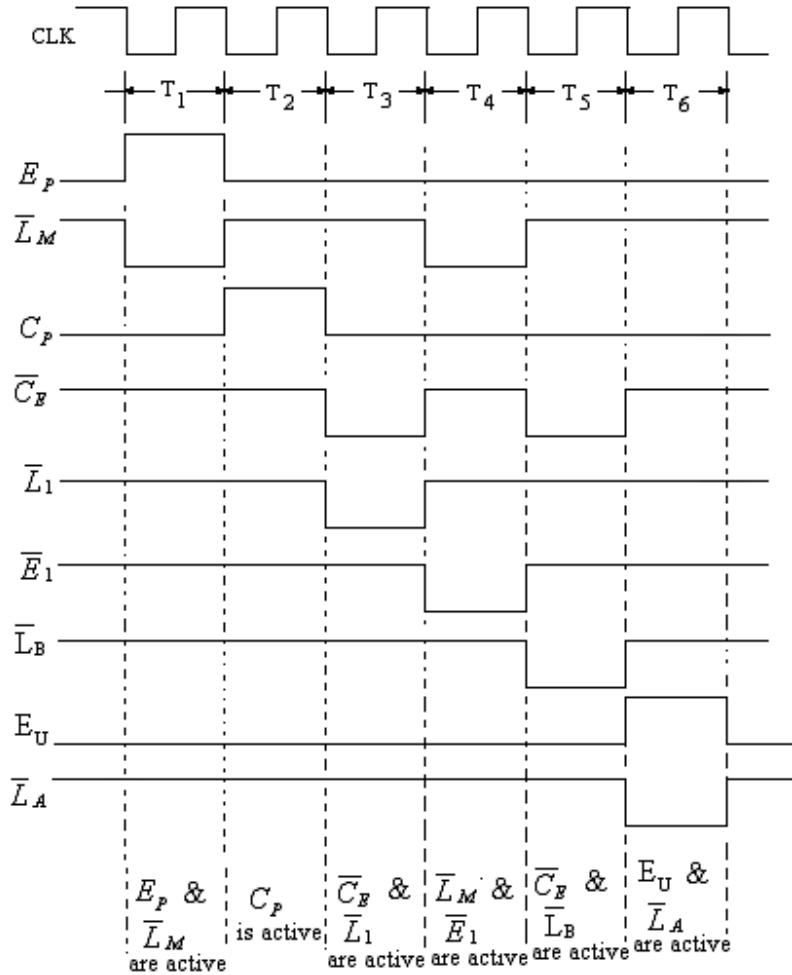
T states of ADD instruction are shown in figures 2.8 (a to c) with active parts as shaded boxes. The timing diagram of Fetch and ADD instruction is shown in figure 2.9.

The microinstruction for T<sub>4</sub>, T<sub>5</sub> and T<sub>6</sub> states of ADD instruction is given below:

| <b>States</b>  | <b>Microinstruction</b> | <b>12 Bit Control Word</b> |
|----------------|-------------------------|----------------------------|
| T <sub>4</sub> | 1A3 H                   | 0001 1010 0011             |
| T <sub>5</sub> | 2E1 H                   | 0010 1110 0001             |
| T <sub>6</sub> | 3C7 H                   | 0011 1100 0111             |



**Fig. 2.8**



**Fig. 2.9**

#### Execution Cycle of SUB Instruction:

The third instruction of SAP-I instruction set is SUB-instruction. For the execution of SUB instruction, following control signals in sequence will be observed during T<sub>4</sub>, T<sub>5</sub> and T<sub>6</sub> states.

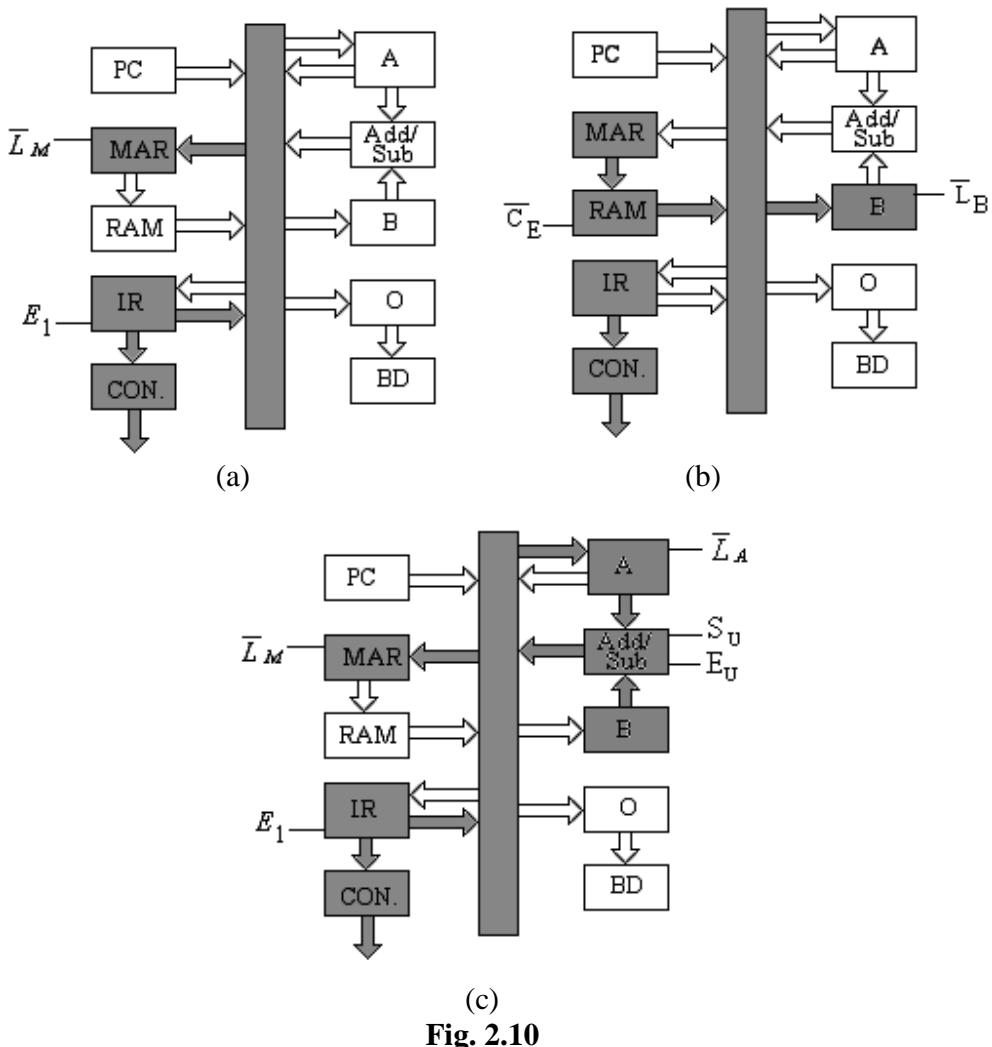
|                | C <sub>P</sub> | E <sub>P</sub> | L <sub>M</sub> | C <sub>E</sub> | ̄L <sub>1</sub> | ̄E <sub>1</sub> | ̄L <sub>A</sub> | E <sub>A</sub> | S <sub>U</sub> | E <sub>U</sub> | ̄L <sub>B</sub> | ̄L <sub>O</sub> |  |  |
|----------------|----------------|----------------|----------------|----------------|-----------------|-----------------|-----------------|----------------|----------------|----------------|-----------------|-----------------|--|--|
| T <sub>4</sub> | 0              | 0              | 0              | 1              | 1               | 0               | 1               | 0              | 0              | 0              | 1               | 1               |  | MAR $\leftarrow$ IR <sub>LNIBBLE</sub> |
| T <sub>5</sub> | 0              | 0              | 1              | 0              | 1               | 1               | 1               | 0              | 0              | 0              | 0               | 1               |  | B $\leftarrow$ M(MAR)                  |
| T <sub>6</sub> | 0              | 0              | 1              | 1              | 1               | 1               | 0               | 0              | 1              | 1              | 1               | 1               |  | A $\leftarrow$ DIFF.                   |

From the above sequence it is clear that SUB instruction follow the same sequence as ADD instruction except that during T<sub>5</sub> state S<sub>U</sub> is high.

T states (T<sub>4</sub> to T<sub>6</sub>) of SUB instruction are shown in figures 2.10 (a to c) with active parts as shaded boxes. The timing diagram of Fetch and SUB instruction is shown in figure 2.11.

The microinstruction for T<sub>4</sub>, T<sub>5</sub> and T<sub>6</sub> states of SUB instruction is given below:

| States         | Microinstruction | 12 Bit Control Word |
|----------------|------------------|---------------------|
| T <sub>4</sub> | 1A3 H            | 0001 1010 0011      |
| T <sub>5</sub> | 2E1 H            | 0010 1110 0001      |
| T <sub>6</sub> | 3CF H            | 0011 1100 1111      |



**Fig. 2.10**

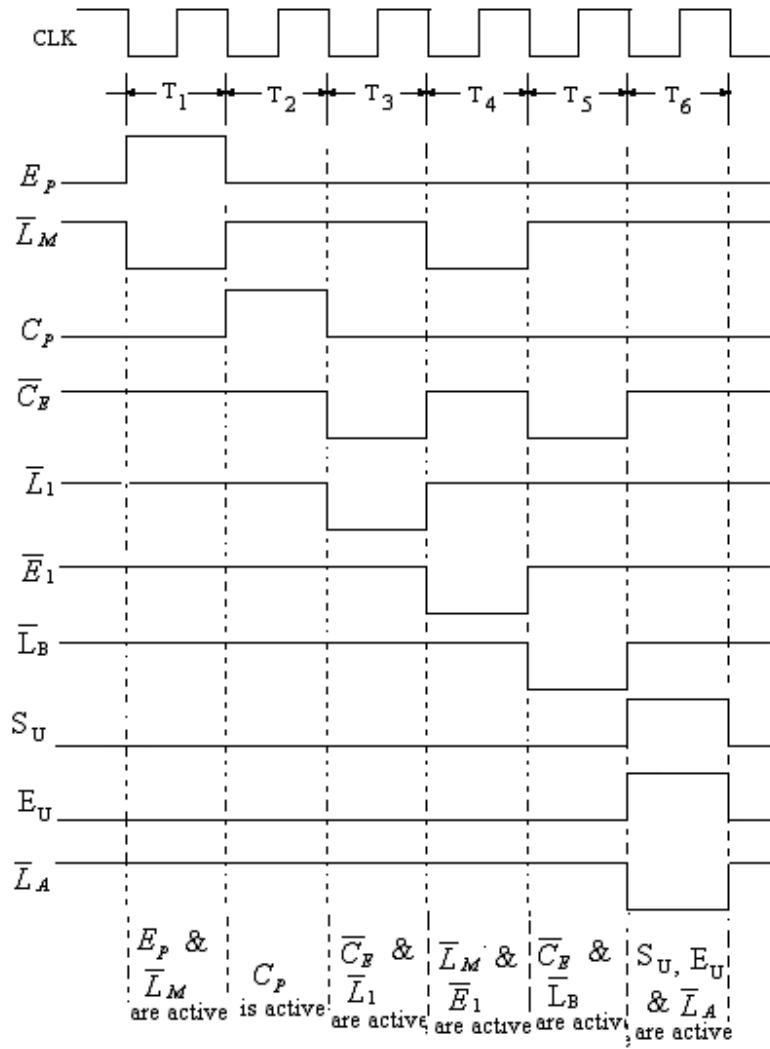


Fig. 2.11

#### Execution Cycle of OUT Instruction:

For the execution of this instruction, following control signals in sequence will be observed during T<sub>4</sub>, T<sub>5</sub> and T<sub>6</sub> states of execution cycle.

|                | $C_P E_P \bar{L}_M \bar{C}_E$ |   |   |   | $\bar{L}_1 \bar{E}_1 \bar{L}_A E_A$ |   |   |   | $S_U E_U \bar{L}_B \bar{L}_O$ |   |   |   |
|----------------|-------------------------------|---|---|---|-------------------------------------|---|---|---|-------------------------------|---|---|---|
| T <sub>4</sub> | 0                             | 0 | 1 | 1 | 1                                   | 1 | 1 | 1 | 0                             | 0 | 1 | 0 |
| T <sub>5</sub> | 0                             | 0 | 1 | 1 | 1                                   | 1 | 1 | 0 | 0                             | 0 | 1 | 1 |
| T <sub>6</sub> | 0                             | 0 | 1 | 1 | 1                                   | 1 | 1 | 0 | 0                             | 0 | 1 | 1 |

During T<sub>4</sub> state,  $E_A$  and  $\bar{L}_O$  are active due to which at the positive edge of clock pulse accumulator content is loaded to the output register. High  $E_A$  send the accumulator data to W-Bus and low  $\bar{L}_O$  loads the data from W-Bus to output register. The work of out instruction is complete. So no-operation is performed during T<sub>5</sub> and T<sub>6</sub> state i.e. all the signals of the control word are inactive.

T states of OUT instruction are shown in figures 2.12 (a to c) with active parts as shaded boxes. The timing diagram of Fetch and OUT instruction is shown in figure 2.13.

The microinstruction for T<sub>4</sub>, T<sub>5</sub> and T<sub>6</sub> states of OUT instruction is given below:

| <b>States</b>  | <b>Microinstruction</b> | <b>(12 Bit Control)</b> |
|----------------|-------------------------|-------------------------|
| T <sub>4</sub> | 3F2 H                   | 0011 1111 0010          |
| T <sub>5</sub> | 3E3 H                   | 0011 1110 0011          |
| T <sub>6</sub> | 3E3 H                   | 0011 1110 0011          |

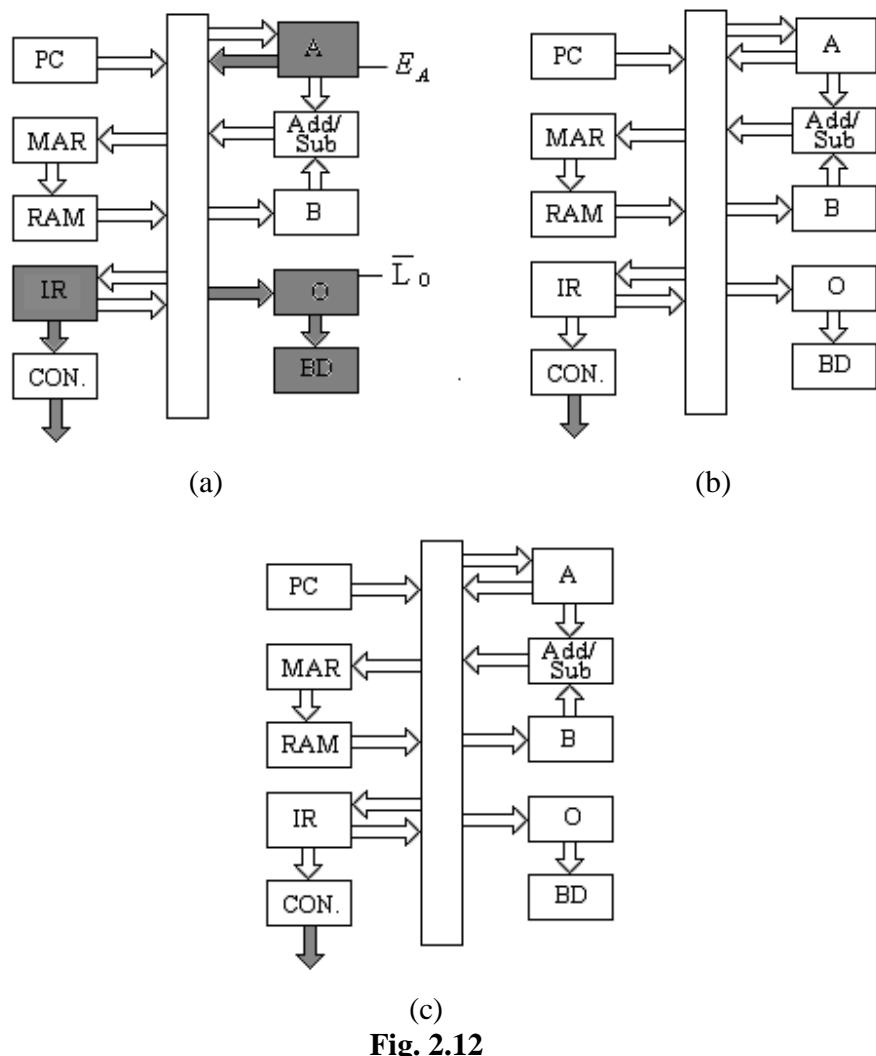
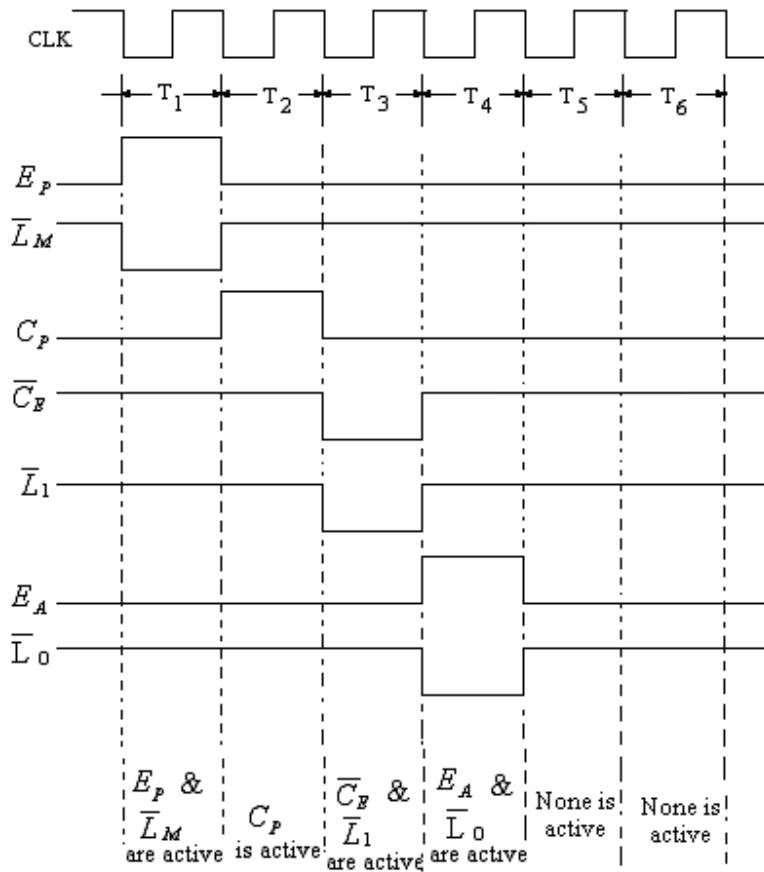


Fig. 2.12



**Fig. 2.13**

#### Execution Cycle of HLT Instruction:

HLT instruction does not require any T-cycle for its execution. Thus during T<sub>4</sub>, T<sub>5</sub> and T<sub>6</sub> state all the signals of the control word are inactive i.e. no-operation is performed during these states.

|                | C <sub>P</sub> | E <sub>P</sub> | L <sub>M</sub> | C <sub>E</sub> | L <sub>1</sub> | E <sub>1</sub> | L <sub>A</sub> | E <sub>A</sub> | S <sub>U</sub> | E <sub>U</sub> | L <sub>B</sub> | L <sub>O</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| T <sub>4</sub> | 0              | 0              | 1              | 1              | 1              | 1              | 1              | 0              | 0              | 0              | 1              | 1              |
| T <sub>5</sub> | 0              | 0              | 1              | 1              | 1              | 1              | 1              | 0              | 0              | 0              | 1              | 1              |
| T <sub>6</sub> | 0              | 0              | 1              | 1              | 1              | 1              | 1              | 0              | 0              | 0              | 1              | 1              |

The microinstruction for T<sub>4</sub>, T<sub>5</sub> and T<sub>6</sub> states of HLT instruction is given below:

| States         | Microinstruction | (12 Bit Control Word) |
|----------------|------------------|-----------------------|
| T <sub>4</sub> | 3E3 H            | 0011 1110 0011        |
| T <sub>5</sub> | 3E3 H            | 0011 1110 0011        |
| T <sub>6</sub> | 3E3 H            | 0011 1110 0011        |

**Example 2.4** What are Microinstructions for:

- (i) LDA instruction
- (ii) ADD instruction

**Solution.** (i) During T<sub>4</sub> state of LDA instruction  $\bar{L}_M$  and  $\bar{E}_1$  are active.

During T<sub>5</sub> state of LDA instruction  $\bar{C}_E$  and  $\bar{L}_A$  are active  
and During T<sub>6</sub> state of LDA instruction None is active.

So Microinstructions for LDA are

For  $T_4$  1A3 H  
 $T_5$  2C3 H  
 $T_6$  3E3 H

(ii) Similarly, Microinstruction for ADD instruction are

For  $T_4$  1A3 H  
 $T_5$  2C3 H  
 $T_6$  3E7 H

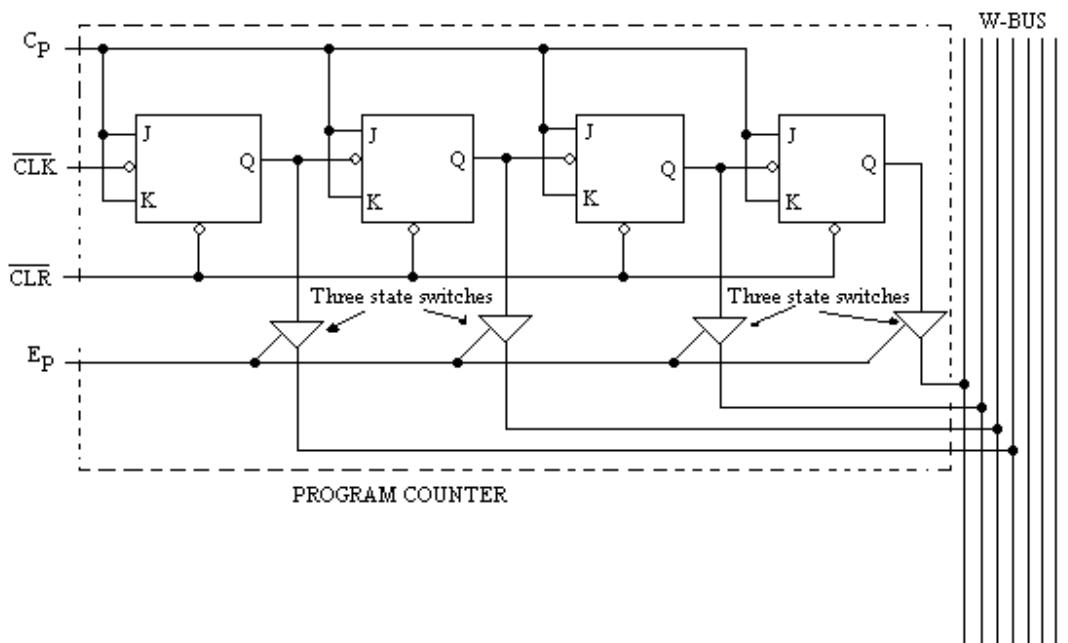
## 2.5 HARDWARE DESIGN OF SAP-I COMPUTER

In this section we shall discuss the hardware details of the following blocks of SAP-I computer:

- Program Counter
- Memory Unit
- Instruction Register
- Controller-Sequencer
- Accumulator
- Adder-Sutractor
- B-Register
- Output Register and Binary Display

### 2.5.1 Design of Program Counter

The program counter (PC) is designed using four J-K flip-flops. The circuit diagram of program counter is shown in figure 2.14. From this diagram it can well be understood that during the start of computer run a low  $\overline{CLR}$  signal resets the program counter to 0 0 0 0. During  $T_1$  state of fetch cycle, a high  $E_P$  sends the address to the W-Bus. During  $T_2$  state, high  $C_P$  increments the Program counter, at the midway through  $T_2$ , the program counter is, however, inactive during  $T_3$  through  $T_6$  states.



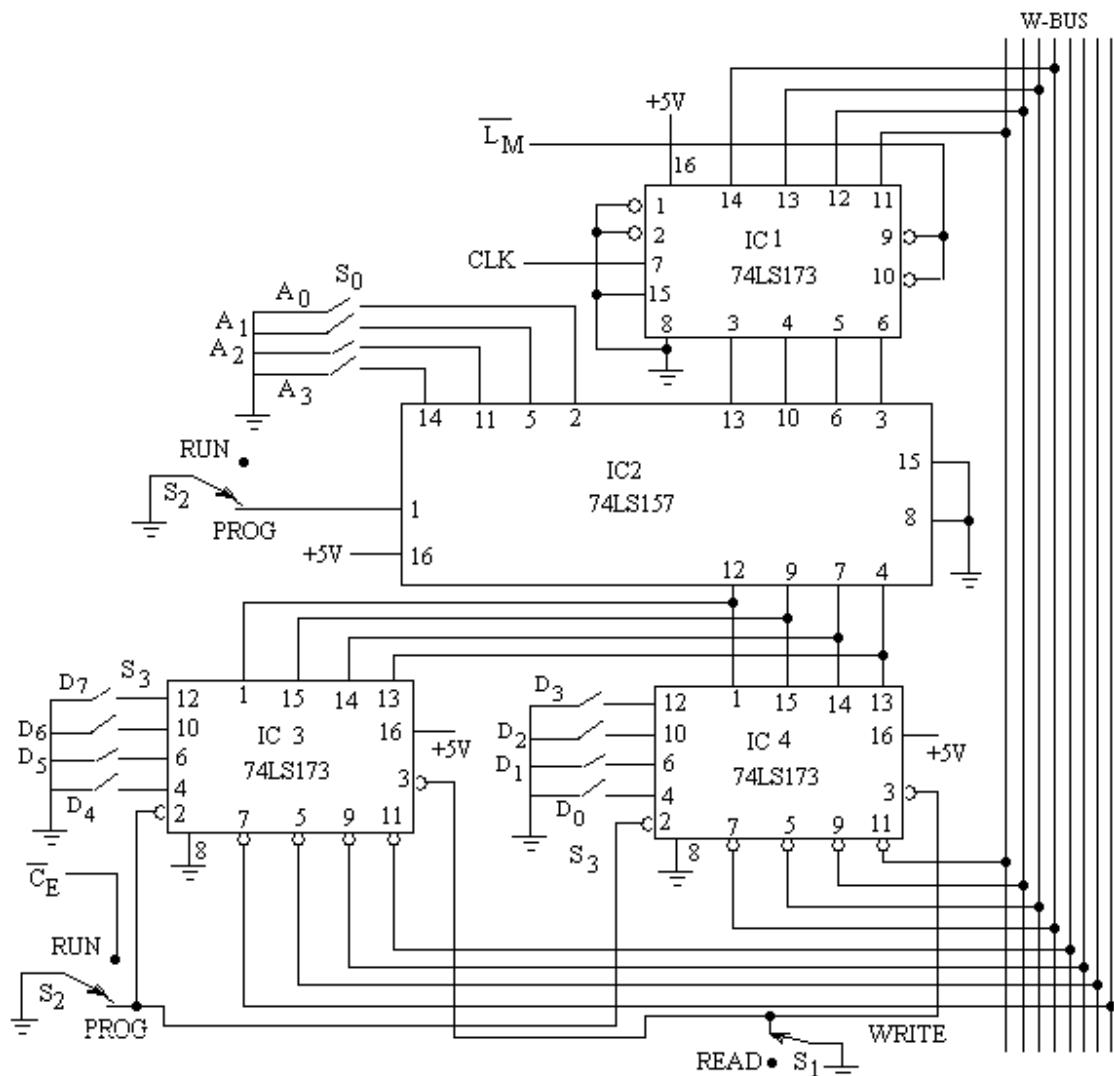
**Fig. 2.14**

### 2.5.2 Memory Unit

Figure 2.15 shows the circuit diagram of memory unit, which consists of Input & MAR and RAM. In this figure IC1 and IC2 form Input & MAR; IC3 and IC4 form read/write memory. IC1 is a 4-bit buffer register. It is a three state output register which is configured here in a two state output, as it is not to be connected to W-Bus. IC2 is basically 2 to 1 nibble multiplexer.

When the switch  $S_1$  is connected to Run position, the address bits are directly available at the output of IC2. If on the contrary  $S_1$  is at the Run position, the address bit from the W-Bus will be available at the output of this IC2.

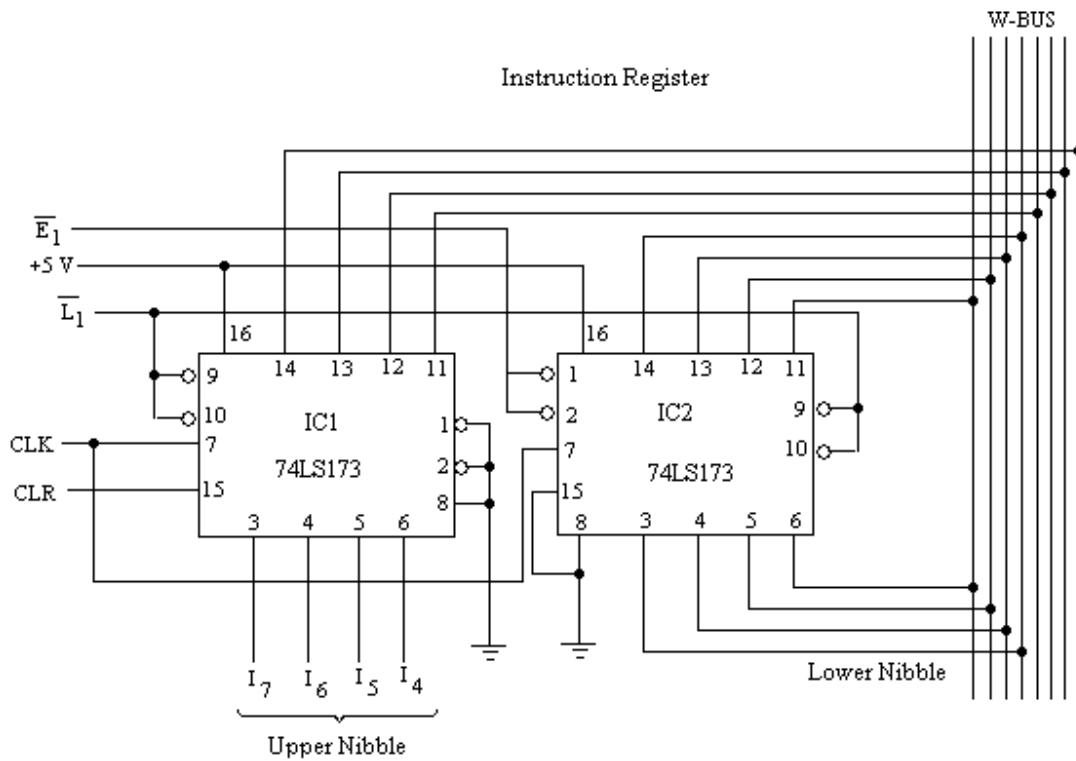
IC3 and IC4 are two 16X4 static RAM, which form the RAM of SAP-I. These two IC's are configured to form, 16X8 read/write memory. When switch S<sub>1</sub> is at PROG position, S<sub>2</sub> is at write position, the data from D<sub>0</sub> to D<sub>7</sub> will be written into static RAM. If S<sub>1</sub> is at Run position, the data already stored in the memory will be available at the W-Bus.



**Fig. 2.15**

### 2.5.3 Instruction Register

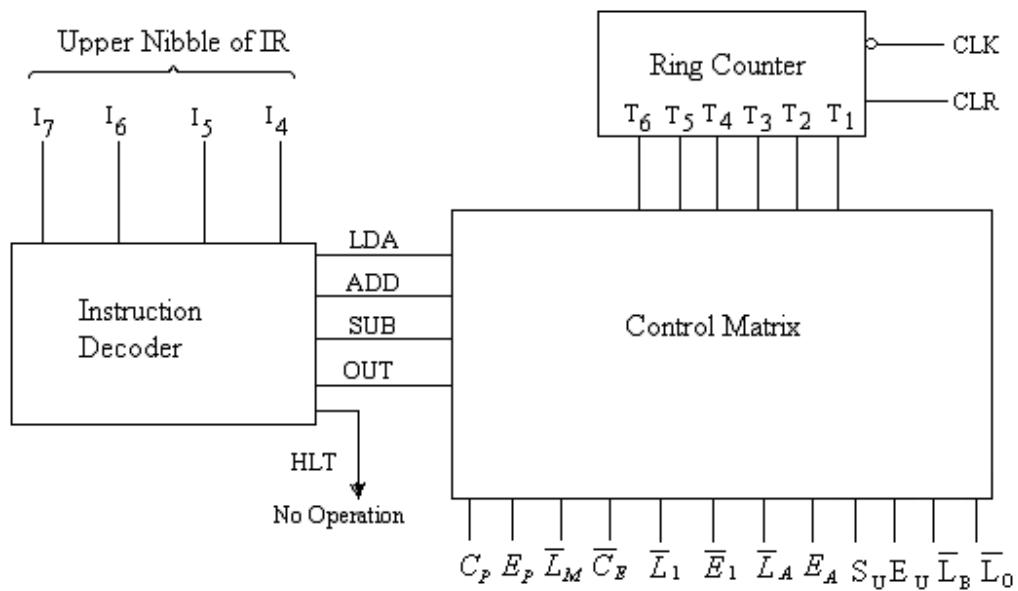
IC1 and IC2 forms the Instruction Register. These are two four-bit buffer registers. These buffer registers are three state registers. IC1 is configured as two state output. The outputs of this IC are upper nibble  $I_7 I_6 I_5 I_4$  which directly goes to controller-sequencer where these are decoded. The outputs of IC2 are the lower nibble (operand of the fetched instruction), which are directly connected to W-Bus. The logic circuit diagram of the Instruction register is shown in figure 2.16.



**Fig. 2.16**

#### 2.5.4 Controller-Sequencer

The schematic block diagram of controller-sequencer is shown in figure 2.17. It consists of a ring counter, instruction decoder and control matrix for the generation of 12-bit control word. The instruction decoder is a simple decoder, which provides the outputs LDA, ADD, Sub, OUT and HLT. In fact it is a 4-to-5 decoder.



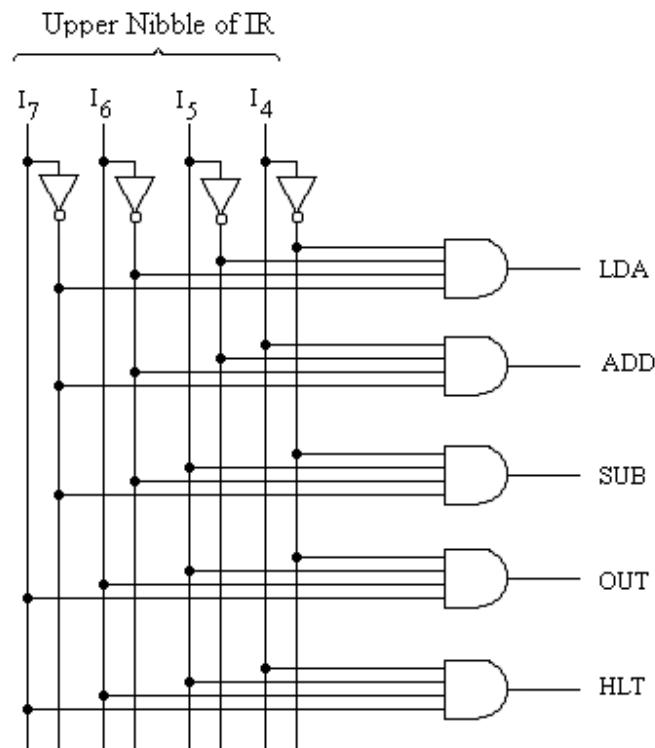
**Fig. 2.17**

The required logic is shown in table 2.4.

Table 2.4

| I <sub>7</sub> | I <sub>6</sub> | I <sub>5</sub> | I <sub>4</sub> | OUTPUT |
|----------------|----------------|----------------|----------------|--------|
| 0              | 0              | 0              | 0              | LDA    |
| 0              | 0              | 0              | 1              | ADD    |
| 0              | 0              | 1              | 0              | SUB    |
| 1              | 1              | 1              | 0              | OUT    |
| 1              | 1              | 1              | 1              | HLT    |

The logic diagram for the Instruction decoder of SAP-I computer is shown in figure 2.18, which is as per the table 2.4.



**Fig. 2.18**

The synchronous counter has already been discussed. This ring counter gives 6-bit output (T<sub>1</sub> to T<sub>6</sub>) at the successive clock pulse.

For the generation of 12-bit control signal, the twelve outputs may be obtained from the logic expressions given below:

$$C_P = T_2$$

$$E_P = T_1$$

$$\bar{L}_M = T_1 + T_4 \cdot \text{LDA} + T_4 \cdot \text{ADD} + T_4 \cdot \text{SUB}$$

$$\bar{C}_E = T_3 + T_5 \cdot \text{LDA} + T_5 \cdot \text{ADD} + T_5 \cdot \text{SUB}$$

$$\bar{L}_1 = T_3$$

$$\bar{E}_1 = T_4 \cdot \text{LDA} + T_4 \cdot \text{ADD} + T_4 \cdot \text{SUB}$$

$$\bar{L}_A = T_5 \cdot \text{LDA} + T_6 \cdot \text{ADD} + T_6 \cdot \text{SUB}$$

$$E_A = T_4 \cdot \text{OUT}$$

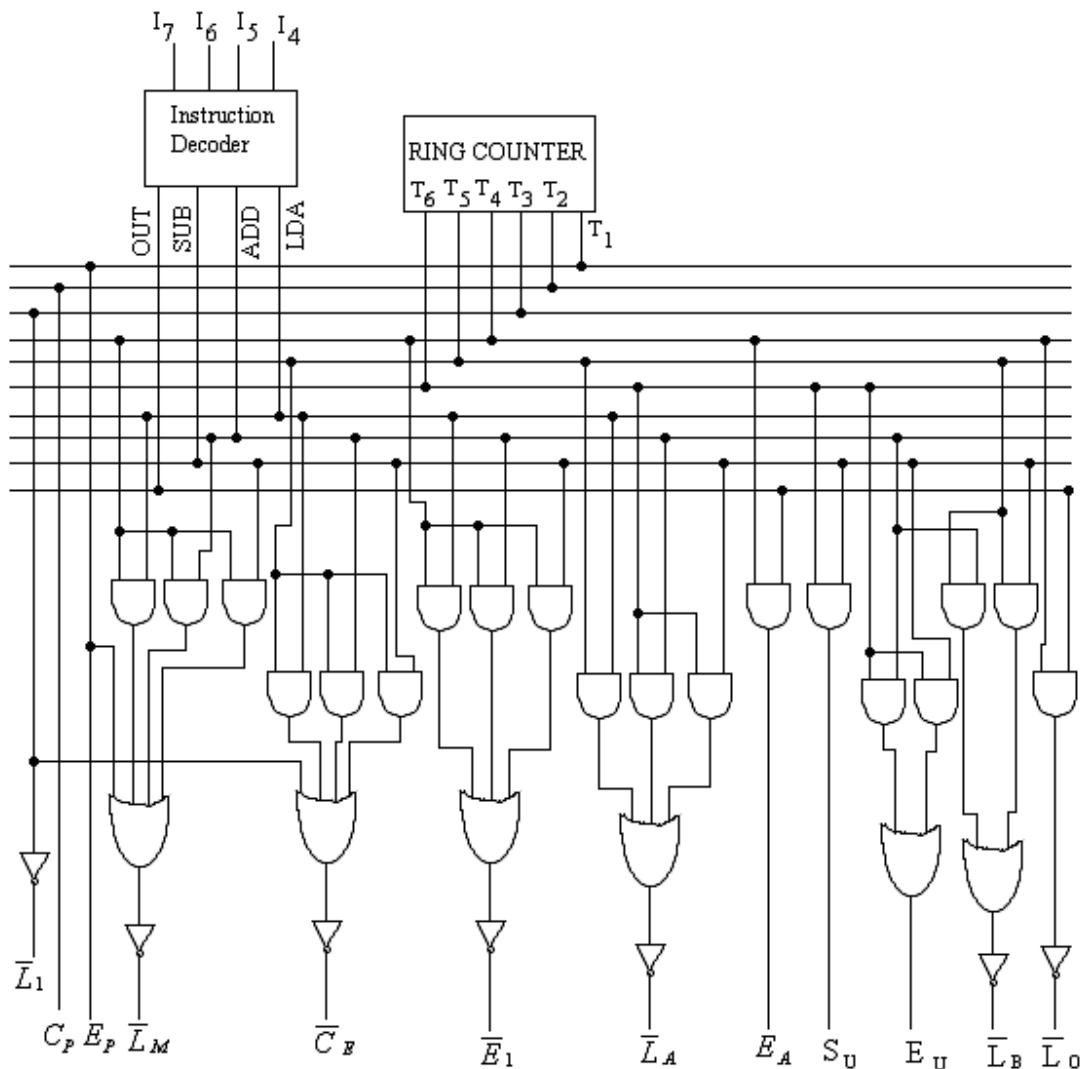
$$S_U = T_6 \cdot \text{SUB}$$

$$E_U = T_6 \cdot \text{ADD} + T_6 \cdot \text{SUB}$$

$$\bar{L}_B = T_5 \cdot \text{ADD} + T_5 \cdot \text{SUB}$$

$$\bar{L}_O = T_4 \cdot \text{OUT}$$

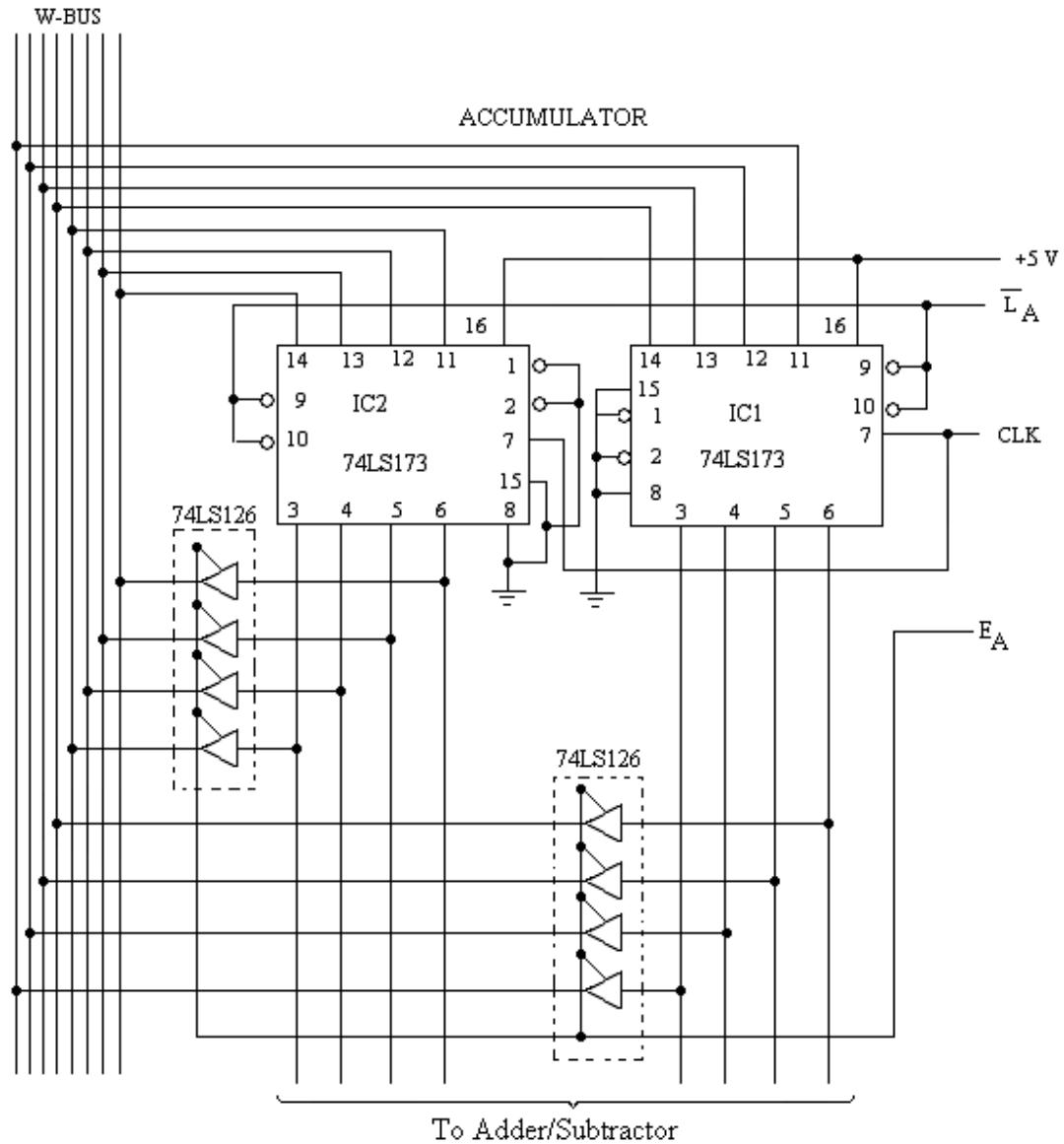
The logic circuit diagram of the above expressions is shown in figure 2.19.



**Fig. 2.19**

### 2.5.5 Accumulator

Figure 2.20 shows the logic diagram of 8-bit buffer register known as accumulator which is designed using two ICs' IC1 and IC2 (both 74LS173). These ICs are configured as two-state output. The outputs of both the two ICs directly go to Adder-Subtractor. Three state switches send the content of accumulator to W-Bus when the signal  $E_A$  is high.

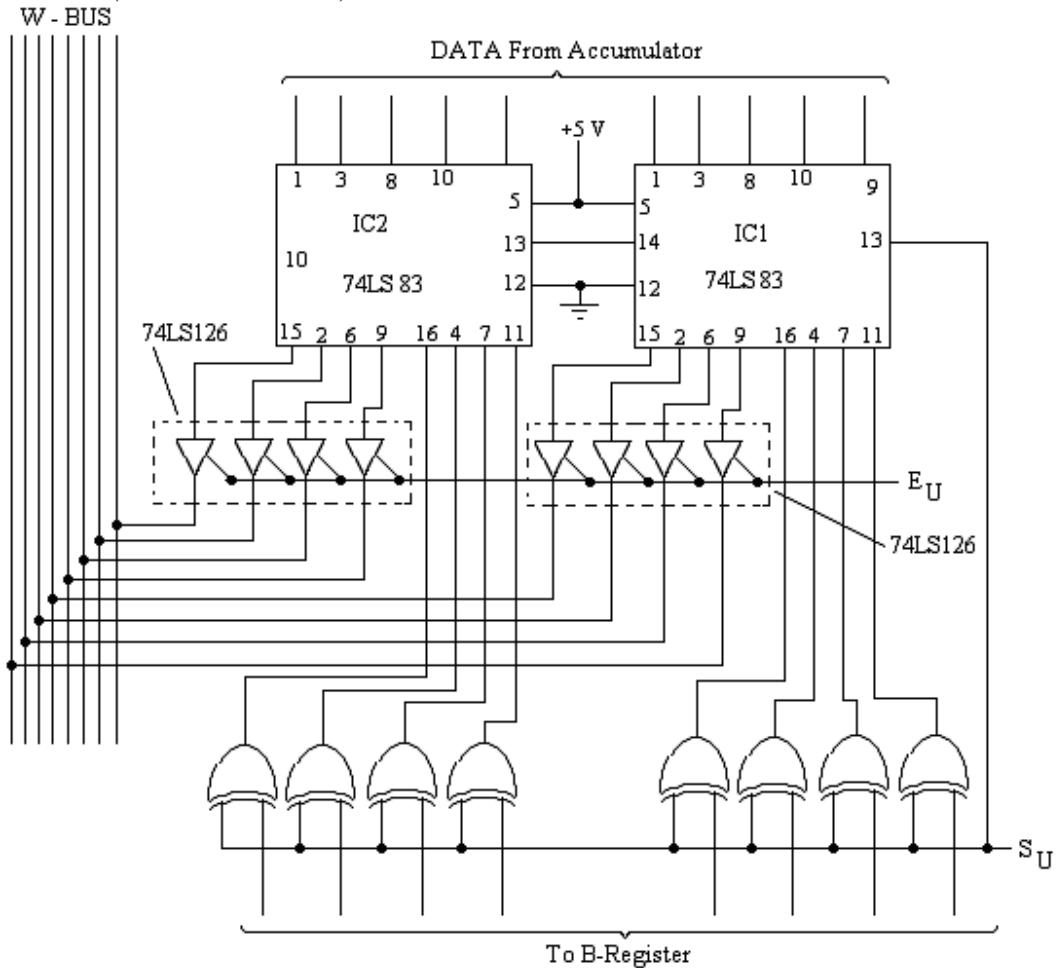


**Fig. 2.20**

### 2.5.6 Adder-Subtractor

Figure 2.21 shows the logic circuit used for Adder-subtractor of SAP-I computer. It basically consists of two 2's complement adder/subtractor ICs (7483). The data from the accumulator directly goes to the Adder-subtractor as shown in figure 2. . The outputs of B-register are also connected to these two ICs through eight exclusive-OR gates. The one terminal each of the eight exclusive-OR gates is being used as control Input ( $S_U$ ). When  $S_U$  is low the content of B-register is directly loaded to Adder/Subtractor i.e. it will act as simple adder. If on the other hand  $S_U$  is high, 1's complement of B-register is being added to adder/subtractor ICs. Logic 1 is also added to LSB to form its as 2's complement (it therefore works as 2's complement subtractor). These two

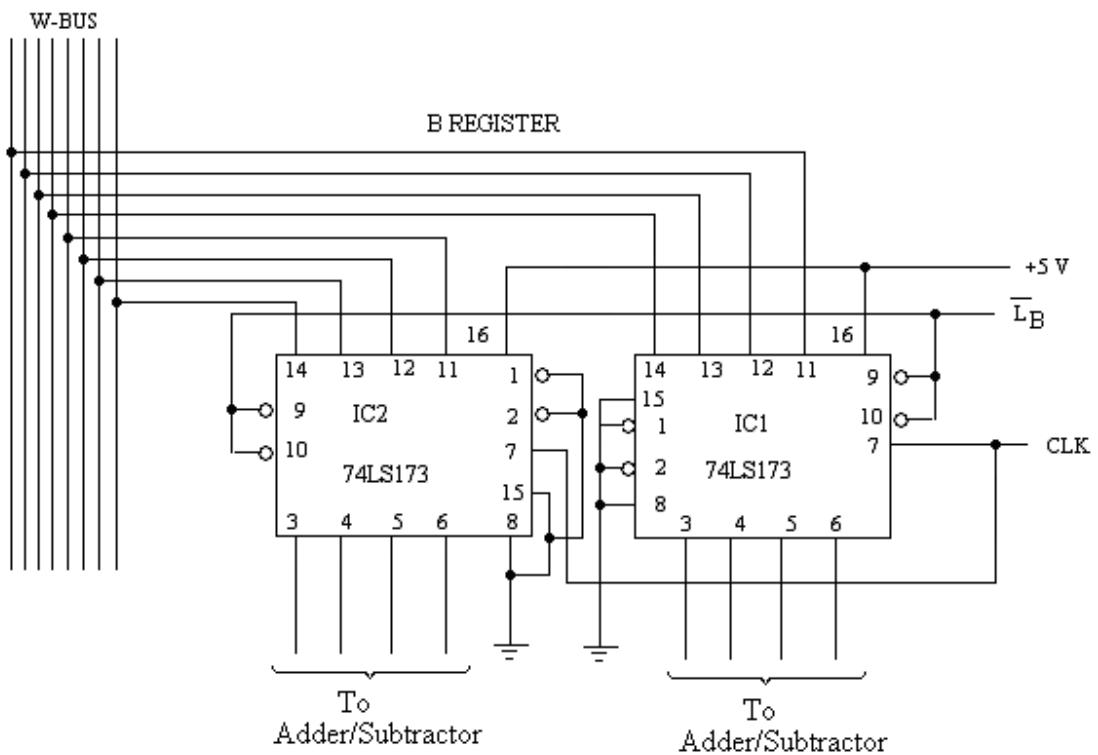
adder/subtractor ICs give the 8-bit addition or subtraction. The 8-bit three state switches load the answer (sum or difference) to W-bus.



**Fig. 2.21**

### 2.5.7 B-Register

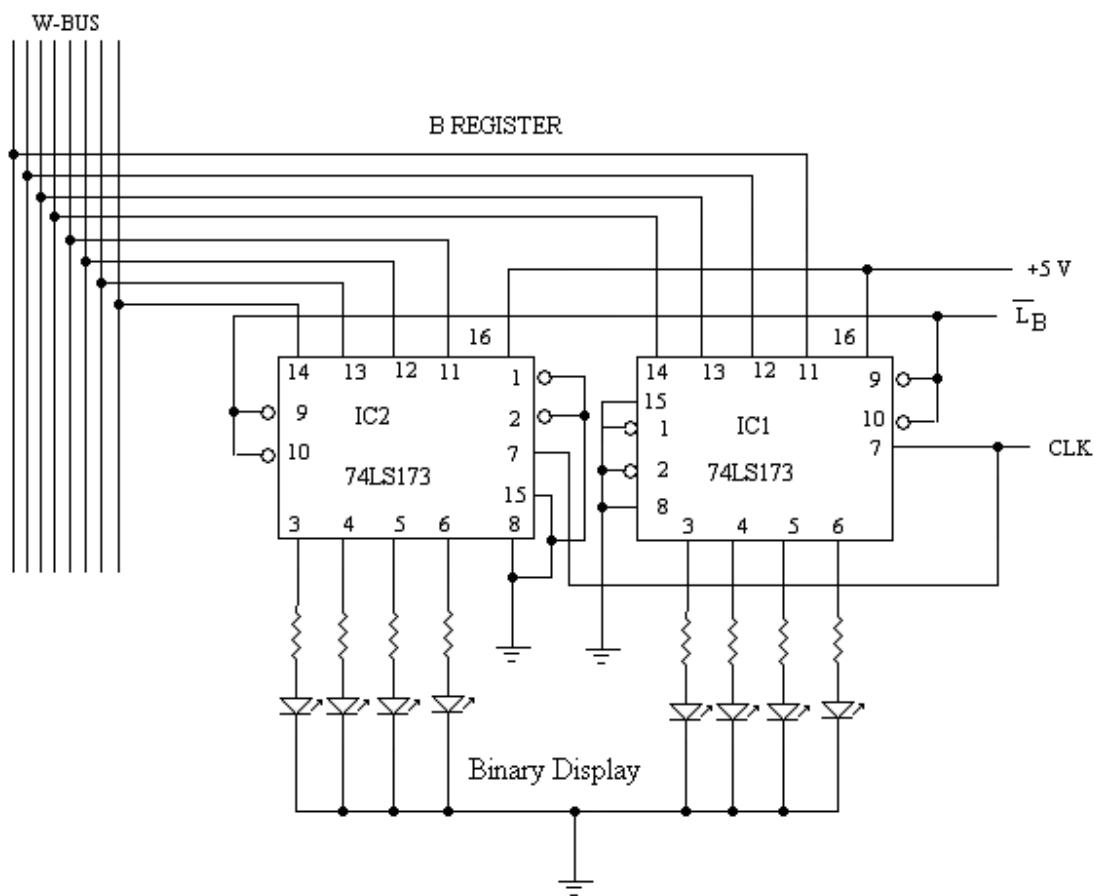
Similar to Accumulator, B-register is also designed using the Buffer register ICs 74LS173. The logic circuit diagram of this register is shown in figure 2.22. These two ICs are configured as two state register. When  $\bar{L}_B$  is low, the data from the W-Bus is loaded to B-register and then transferred to Adder/Subtractor.



**Fig. 2.22**

### 2.5.8 Output Register

Output register is also formed by two state Buffer register using 74LS173 ICs. The output register with Binary display unit of SAP-I computer is shown in figure 2.23. When  $\bar{L}_o$  is low, the data from the W-Bus will be loaded to output register. The output terminals of output register are connected to 8 LEDs, which will glow as per the data available at the output register.



**Fig.2.23**

### Problems

- 2.1 Draw the block diagram of SAP-I computer. Discuss the function of each block.
- 2.2 What is the function of Controller/Sequencer in SAP-I computer? How a control word is generate in SAP-I computer?
- 2.3 Draw the logic circuit diagram of Program counter and discuss its function.
- 2.4 Explain with the help of timing diagram the fetch and execution cycle of LDA-instruction.
- 2.5 Explain with the help of timing diagram the fetch and execution cycle of ADD-instruction.
- 2.6 Explain with the help of timing diagram the fetch and execution cycle of SUB-instruction.
- 2.7 Explain with the help of timing diagram the fetch and execution cycle of OUT-instruction.
- 2.8 Explain with the help of timing diagram the fetch and execution cycle of HLT-instruction. Further explain what will happen if HLT instruction is not given at the end of the program?
- 2.9 Differentiate between:
  - (i) Micro-instruction and Macro-instruction

- (ii) Assembly language and machine language
  - (iii) Source program and object program
- 2.10 Explain the instruction set of SAP-I computer. What is the size of MAR of the SAP-computer?
- 2.11 Why the positive clock edge occurs half way through each T-state in SAP-I computer?
- 2.12 What is the role of instruction register in SAP-I computer? Draw the logic diagram for Instruction register of SAP-I computer?
- 2.13 Write an assembly language program that perform the following operation in SAP-I computer.
- $$8 - 3 + 5 + 2 - 4$$
- Use memory locations 7H to BH for the data. Write also its program in machine language.
- 2.14 Write an assembly language program for SAP-I computer that will display the result of  $9 + 3 - 2$ .
- 2.15 Write an assembly language program for SAP-I computer that will display the result of  $7 + 3 - 2 + 4 - 1 + 6$ .  
Use AH to FH memory locations for starting the data.
- 2.16 Write an assembly language program for calculating the following expression  $A + 3B - 2C$  on SAP-I computer. The data A, B and C are stored in memory locations 9 H to B H. Write also its machine language.
- 2.17 What are microinstructions for SUB and OUT instructions of SAP-I computer. Express the answer in binary also.
- 2.18 The timing diagram for Fetch and Execution cycle of ADD instruction is shown in figure 2.24. What are the microinstructions for Fetch and Execution cycle of ADD instruction?

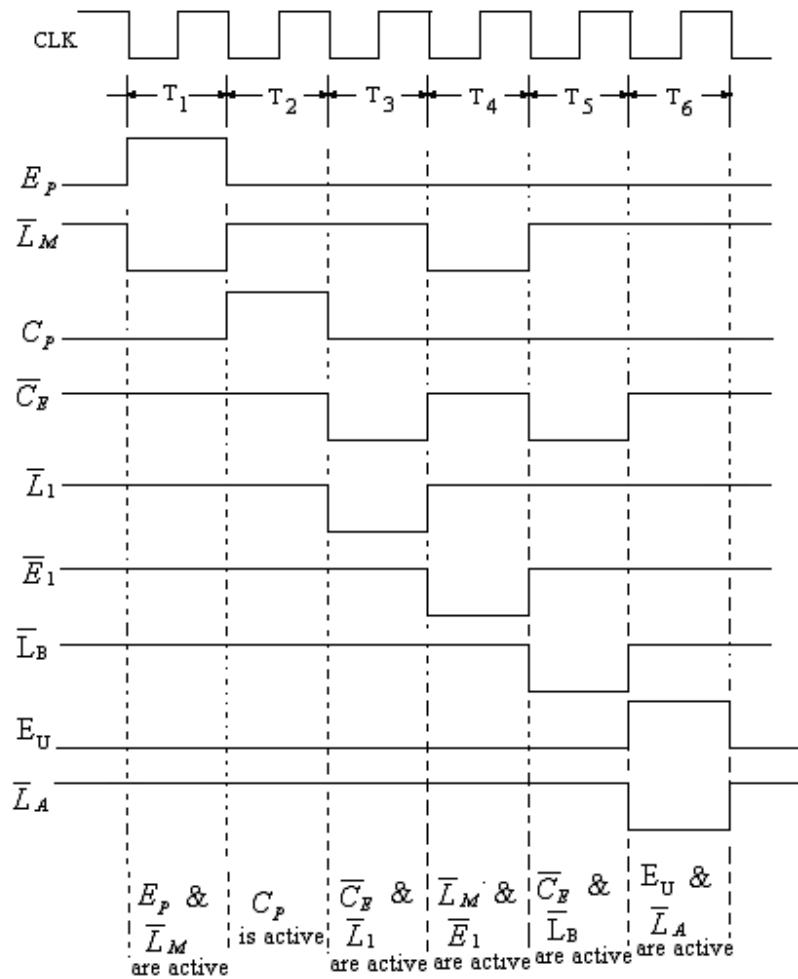


Fig. 2.24

# 3

## SAP – II

---

In the preceding chapter of this book, the architecture, Instruction set and Programming of a conceptual computer named SAP-I computer was discussed. In order to understand the basic knowledge of a computer, it was the first step in the evolution towards the modern computers. The further step in this direction is SAP-II computer. The details of SAP-II computers will now be discussed.

### 3.1 ARCHITECTURE OF SAP-II COMPUTER

The architecture of another conceptual computer named as SAP-II computer is shown in figure 3.1. The SAP-II computer has more registers and other blocks, for giving more flexibility in programming. The architectural details of the Simple as Possible computer SAP-II are basically same as that of SAP-I, but SAP-II has more additional features which are given below:

- (1) It is an eight bit computer, as 8-bit data can be processed in this computer. The SAP-I computer was also 8-bit computer.
- (2) It has bidirectional BUS shown by double headed arrows, which indicate that the data can move either way after having the proper control signal. In SAP-I computer separate BUS is used for each way.
- (3) The memory size of SAP-II computers is 64K (65536) words and each word is of 8-bit long. The data may be loaded or retrieved from the memory, through a buffer register known as Memory Data Register (MDR) or Memory Buffer Register (MBR).

The address for the location is 16 bits as  $64K = 65536 = 2^{16}$ .

The address will start from:

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

to

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

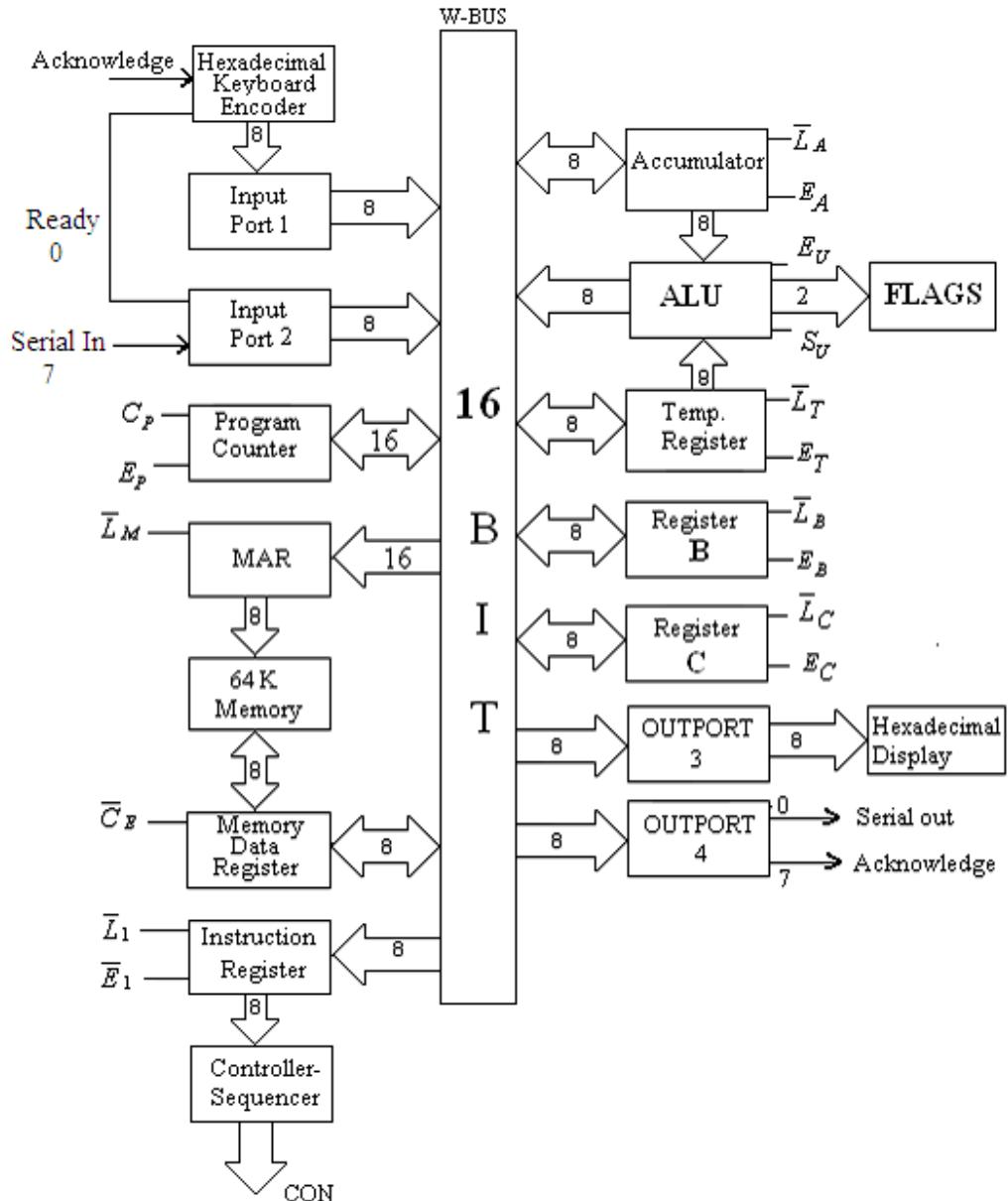
The hexadecimal equivalent of the address is

0 0 0 0 H to F F F F H.

However, memory size of SAP-I computers was only 16. As  $2^4 = 16$ , so 4 address lines were used.

- (4) The width of the W-BUS is 16, since it has to carry a 16 bit address. The width of W-bus for SAP-I computer was only of 8-bits.
- (5) The program counter of SAP-II computer is 16-bit wide as it has to send the address of 16 bits to W-BUS.
- (6) Memory Address Register (MAR) of SAP-II computer is of 16 bits as it receives the address of 16-bits from the W-BUS.

- (7) The controller sequencer of SAP-II computer generates a control word of bigger size than that of SAP-I computer; since it has to control more registers and blocks of the computer. The control word of SAP-I computers was of 12 bits.



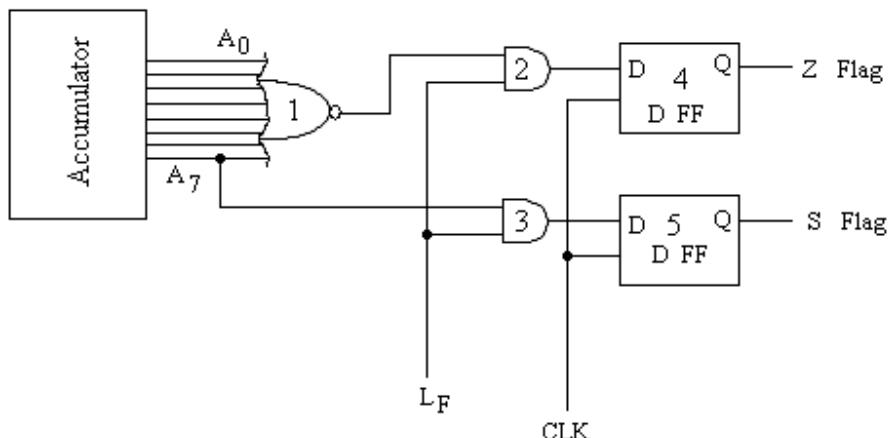
**Fig. 3.1**

- (8) There are three registers in SAP-II computers namely Accumulator (Register A), Register B and Register C. All these registers are 8-bit registers. The registers B and C are general purpose registers. These registers give more flexibility in moving the data from one register to other register during the computer run. In SAP-I computer B-register was available for holding the data being added to or subtracted from the

accumulator. In SAP-II computers, there is a temporary register (T-register) for this purpose.

- (9) SAP-II computer has Arithmetic and Logical unit (ALU) which can perform the arithmetic and logical operations on 8 bit data. Two flags (Sign flag **S** and Zero flag **Z**) are available with ALU. The accumulator content goes negative or zero during execution of some instructions. This affects the sign and zero flags. Figure 3.2 shows the circuit used for setting of flags of SAP-II computers.

It is clear from this figure that when accumulator content is zero all the bits ( $A_0$  to  $A_7$ ) are zero and the output of NOR gate is one. This NOR gate derives the AND gate (numbered 2), if the gating signal  $L_F$  is high and the flag will be updated. Similarly, if the accumulator content is negative, most significant bit ( $A_7$ ) of the accumulator will be one which drives the lower AND gate (numbered 3) and sign flag will be updated when  $L_F$  is high. The sign flag will be set if accumulator content is negative and reset if positive. The zero flag will be set if accumulator content is zero and reset if it is non zero.



**Fig. 3.2**

- (10) There are two input ports (Input Port-1 and Input Port-2) in SAP-II computers as shown in figure 3.1. A hexadecimal keyboard encoder is connected to Port 1. The instruction and data fed to the computer through this input port 1. The encoder sends a READY signal to bit 0 of Input Port-2. The data may also be accepted in serial form through the Input Port-2.
- (11) SAP-II computer has two Output Ports namely output port-1 and output port-2. Through the output port-1, the processed data (answer from accumulator) can be visualized on the hexadecimal display. The accumulator content can also be sent to output port serially.
- (12) Because of the availability of more registers and large size of memory, SAP-II computer has an instruction set with 42 instructions. SAP-I computer has only 5 instruction in its instruction set.

### 3.2 INSTRUCTION SET OF SAP-II COMPUTERS

As already discussed SAP-II computer has 42 instructions in its instruction set which are listed in table 3.1. These instructions are divided into 5 different categories as:

Memory Reference Instructions

Register Instructions

Jump and Call instructions

Logical Instructions

Miscellaneous Instructions

**Table 3.1**

| Category                      | Instruction  | Op. Code | Operation  |
|-------------------------------|--------------|----------|--|
| Memory Reference instructions | LDA address  | 3A       | $[A] \leftarrow [M_{address}]$                           |
|                               | STA address  | 32       | $[M_{address}] \leftarrow [A]$                           |
|                               | MVI A, data  | 3E       | $[A] \leftarrow Data$                                    |
|                               | MVI B, data  | 06       | $[B] \leftarrow Data$                                    |
|                               | MVI C, data  | 0E       | $[C] \leftarrow Data$                                    |
| Register Instructions         | MOV A, B     | 78       | $[A] \leftarrow [B]$                                     |
|                               | MOV A, C     | 79       | $[A] \leftarrow [C]$                                     |
|                               | MOV B, A     | 47       | $[B] \leftarrow [A]$                                     |
|                               | MOV B, C     | 41       | $[B] \leftarrow [C]$                                     |
|                               | MOV C, A     | 4F       | $[C] \leftarrow [A]$                                     |
|                               | MOV C, B     | 48       | $[C] \leftarrow [B]$                                     |
|                               | ADD B        | 80       | $[A] \leftarrow [A]+[B]$                                 |
|                               | ADD C        | 81       | $[A] \leftarrow [A]+[C]$                                 |
|                               | SUB B        | 90       | $[A] \leftarrow [A]-[B]$                                 |
|                               | SUB C        | 91       | $[A] \leftarrow [A]-[C]$                                 |
|                               | INR A        | 3C       | $[A] \leftarrow [A]+1$                                   |
|                               | INR B        | 04       | $[B] \leftarrow [B]+1$                                   |
|                               | INR C        | 0C       | $[C] \leftarrow [C]+1$                                   |
|                               | DCR A        | 3D       | $[A] \leftarrow [A]-1$                                   |
|                               | DCR B        | 05       | $[B] \leftarrow [B]-1$                                   |
|                               | DCR C        | 0D       | $[C] \leftarrow [C]-1$                                   |
| Jump and Call instructions    | JMP address  | C3       | $[PC] \leftarrow Address$                                |
|                               | JM address   | FA       | $[PC] \leftarrow Address$ ; if acc. content is negative. |
|                               | JZ address   | CA       | $[PC] \leftarrow Address$ ; if acc. content is zero.     |
|                               | JNZ address  | C2       | $[PC] \leftarrow Address$ ; if acc. content is not zero. |
|                               | CALL address | CD       | $Stack \leftarrow [PC]$ and                              |

|                           |           |    |                                       |
|---------------------------|-----------|----|---------------------------------------|
|                           |           |    | $[PC] \leftarrow Address$             |
|                           | RET       | C9 | $[PC] \leftarrow Stack$               |
| Logical instructions      | CMA       | 2F | $A \leftarrow \bar{A}$                |
|                           | ANA B     | A0 | $A \leftarrow A \cdot AND \cdot B$    |
|                           | ANA C     | A1 | $A \leftarrow A \cdot AND \cdot C$    |
|                           | ORA B     | B0 | $A \leftarrow A \cdot OR \cdot B$     |
|                           | ORA C     | B1 | $A \leftarrow A \cdot OR \cdot C$     |
|                           | XRA B     | A8 | $A \leftarrow A \oplus B$             |
|                           | XRA C     | A9 | $A \leftarrow A \oplus C$             |
|                           | ANI, data | E6 | $A \leftarrow A \cdot AND \cdot data$ |
|                           | ORI, data | F6 | $A \leftarrow A \cdot OR \cdot data$  |
|                           | XRI, data | FE | $A \leftarrow A \oplus data$          |
| Miscellaneous instruction | NOP       | 00 | No operation                          |
|                           | IN Port   | DB | $[A] \leftarrow [Portaddress]$        |
|                           | OUT Port  | D3 | $[Outport] \leftarrow [A]$            |
|                           | RAL       | 17 | Rotate acc. left                      |
|                           | RAR       | 1F | Rotate acc. right                     |
|                           | HLT       | 76 | Stops Processing                      |

The operation to be performed by each instruction has been explained in table 3.1. After going through these instructions, it will be possible to write complicated programs in solving the different problems.

The SAP-II computer has memory reference instruction like ‘STA address’ which allows to store the content of accumulator to the memory location whose address is given with the instruction. For example if accumulator has the content  $A = 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0$  before execution of an instruction. Let the instruction is

STA 2050 H

After execution of this instruction the memory location 2050 H will have the data

1 0 1 0 1 0 1 0.

$$\text{i.e. } M_{2050H} = 10101010$$

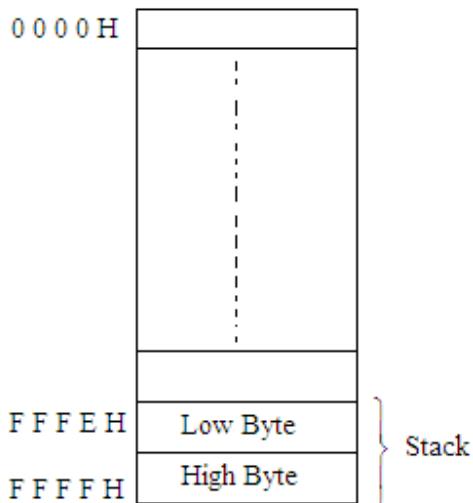
In register instructions, **MOV C, B** copies the content of B-register into C-register ( $C \leftarrow B$ ).

In Call instructions, the instruction ‘CALL address’ copies the present address of program counter to Stack and the given address with the instruction is copied into program counter. So whenever the next instruction is fetched, the program counter will send this new address. This instruction helps in jumping from the normal routine of the program to subroutine program. The instruction RET stands for return. This statement is used at the end of subroutine program and its sends back to the original program as the content of the stack is copied back to the program counter. In SAP-II computer there is, however, no register for Stack. The last two memory locations FFFE H and FFFF H are used for this purpose as shown in figure 3.3 i.e. these two locations are exclusively used for saving the return address of the subroutine program.

The SAP-II computer has the logical instructions such as ‘ANA reg’, ‘ORA reg’, ‘XRA reg’, ‘ANI data’, ‘ORI data’ and ‘CMA’. The ‘ANA B’ instruction

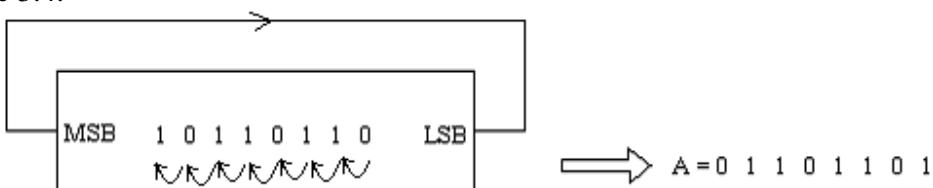
AND the contents of A register with the contents of B register bit by bit and the answer is loaded into accumulator.

In miscellaneous instructions, ‘IN port’ loads the data word from an input port to the accumulator.



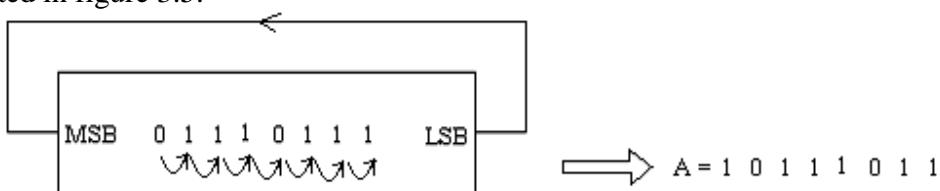
**Fig. 3.3**

The RAL instruction rotates the accumulator left. This instruction will shift all bits of accumulator content to the left and move the MSB into LSB position as illustrated in figure 3.4.



**Fig. 3.4**

Similarly, the RAR instruction rotates the accumulator right. This instruction will shift all bits of accumulator content to the right and move the LSB into MSB position as illustrated in figure 3.5.



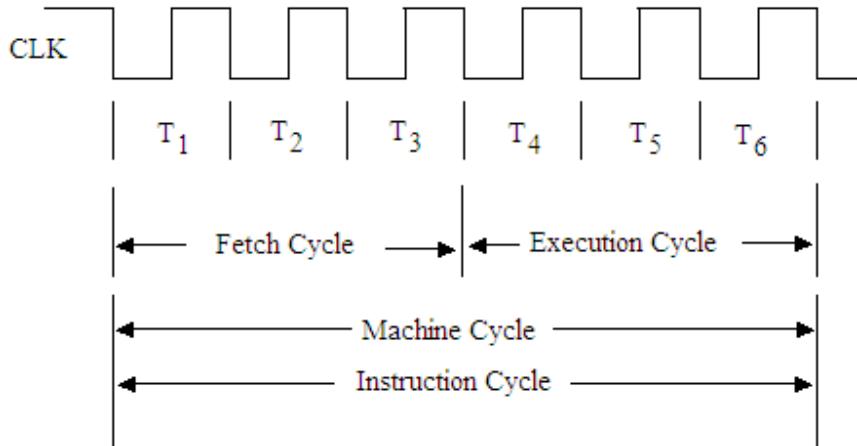
**Fig. 3.5**

### 3.3 MACHINE CYCLE AND INSTRUCTION CYCLE

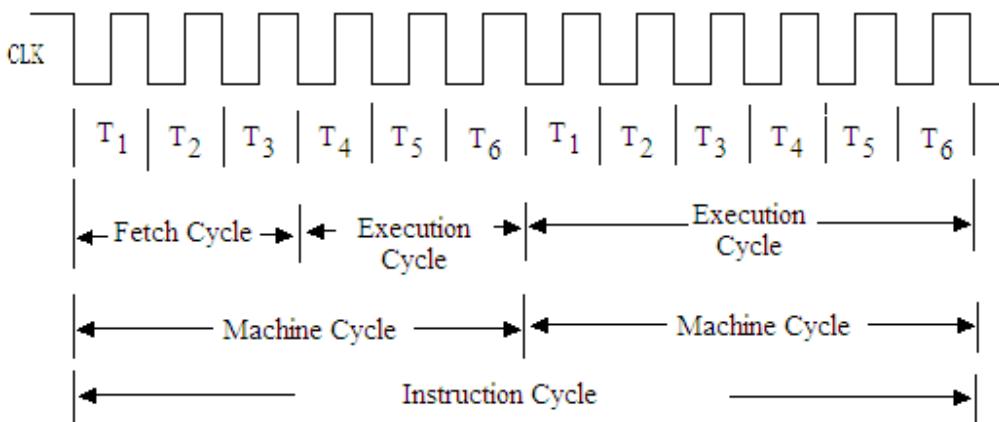
It has been discussed in the last chapter that the SAP-I computer takes 6-T states to fetch and execute an instruction. These six states form a machine cycle. The number of T-states needed to fetch and execute an instruction is called Instruction cycle. So in SAP-I computer instruction cycle is machine cycle as shown in figure 3.6.

The SAP-II computer has, however, more than one machine cycle to fetch and execute an instruction. As shown in figure 3.7, first three T-states are used to fetch an

instruction and other 9 T-states are used for the execution of an instruction. So in SAP-II or other computers the instruction cycle may have two or more machine cycles. Table 3.2 shows the number of T-states used for the instruction cycle of each instruction.



**Fig. 3.6**



**Fig. 3.7**

The instruction ‘STA address’ takes 13 T-states to fetch and execute this instruction. If the system clock frequency is 3 MHz, then ‘STA address’ will take

$$13 \times \frac{1}{3} \mu\text{sec} = 4.33 \mu\text{sec}$$

time to fetch and execute this instruction i.e. it is the time of instruction cycle of the instruction ‘STA address’.

**Table 3.2**

| Instruction | Addressing | No of T-states | Flags affected | No of bytes of instr. |
|-------------|------------|----------------|----------------|-----------------------|
| LDA address | Direct     | 13             | None           | 3                     |
| STA address | Direct     | 13             | None           | 3                     |
| MVI A, data | Immediate  | 7              | None           | 2                     |
| MVI B, data | Immediate  | 7              | None           | 2                     |
| MVI C, data | Immediate  | 7              | None           | 2                     |

|              |           |      |      |   |
|--------------|-----------|------|------|---|
| MOV A, B     | Register  | 4    | None | 1 |
| MOV A, C     | Register  | 4    | None | 1 |
| MOV B, A     | Register  | 4    | None | 1 |
| MOV B, C     | Register  | 4    | None | 1 |
| MOV C, A     | Register  | 4    | None | 1 |
| MOV C, B     | Register  | 4    | None | 1 |
| ADD B        | Register  | 4    | S, Z | 1 |
| ADD C        | Register  | 4    | S, Z | 1 |
| SUB B        | Register  | 4    | S, Z | 1 |
| SUB C        | Register  | 4    | S, Z | 1 |
| INR A        | Register  | 4    | S, Z | 1 |
| INR B        | Register  | 4    | S, Z | 1 |
| INR C        | Register  | 4    | S, Z | 1 |
| DCR A        | Register  | 4    | S, Z | 1 |
| DCR B        | Register  | 4    | S, Z | 1 |
| DCR C        | Register  | 4    | S, Z | 1 |
| JMP address  | Immediate | 10   | None | 3 |
| JM address   | Immediate | 10/7 | None | 3 |
| JZ address   | Immediate | 10/7 | None | 3 |
| JNZ address  | Immediate | 10/7 | None | 3 |
| CALL address | Immediate | 18   | None | 3 |
| RET          | Implied   | 10   | None | 1 |
| CMA          | Implied   | 4    | None | 1 |
| ANA B        | Register  | 4    | S, Z | 1 |
| ANA C        | Register  | 4    | S, Z | 1 |
| ORA B        | Register  | 4    | S, Z | 1 |
| ORA C        | Register  | 4    | S, Z | 1 |
| XRA B        | Register  | 4    | S, Z | 1 |
| XRA C        | Register  | 4    | S, Z | 1 |
| ANI, data    | Immediate | 7    | S, Z | 2 |
| ORI, data    | Immediate | 7    | S, Z | 2 |
| XRI, data    | Immediate | 7    | S, Z | 2 |
| NOP          | -         | 4    | --   | 1 |
| IN Port      | Direct    | 10   | None | 2 |
| OUT Port     | Direct    | 10   | None | 2 |
| RAL          | Implied   | 4    | None | 1 |
| RAR          | Implied   | 4    | None | 1 |
| HLT          | -         | 5    | --   | 1 |

### 3.4 ADDRESSING MODES

There are various techniques to specify the data for instruction. These techniques are called addressing modes. SAP-II computer has the following addressing modes:

1. Direct Addressing
2. Register Addressing
3. Immediate Addressing
4. Implied Addressing

### **Direct Addressing**

In this mode of addressing, the address of the operand is given in the instruction itself.

For example      LDA 2100 H  
                    OUT 03 H                etc.

### **Register Addressing**

In this mode of addressing, the operands are in registers.

For example      MOV A , B  
                    ADD C                etc.

### **Immediate Addressing**

In immediate addressing mode, the operand is specified in the instruction itself.

For example      MVI B, 08 H  
                    ANI 07 H                etc.

### **Implied Addressing**

There are certain instructions which operate on the contents of the accumulator. Such instructions do not require the address of the operand, since the operand is implied in the instruction itself. So this type of addressing mode is called as Implied Addressing.

For example      CMA  
                    RAL  
                    RAR                etc.

The addressing modes of all the instructions of SAP-II computers are given in table 3.2.

## **3.5 INSTRUCTION TYPES**

The instruction set of SAP-II computer may be divided into three types. As already seen, there are different ways for specifying the data for instructions, so the machine codes of all instructions are not of same length. Following are the types of instructions:

- (i)      One Byte Instruction
- (ii)     Two Byte Instruction
- (iii)   Three Byte Instruction

### **One Byte Instruction**

This type of instruction has only op code part of one byte and no operand is given. The instruction length is only of one byte. It can be stored only in one memory location.

For example      MOV A, C  
                    ADD C  
                    CMA  
                    RAL  
                    RAR                etc.

If ‘MOV A, C’ instruction is to be stored in some location say 2000 H, then its op code of one byte is to be fed in this memory location.

i.e.      2000 H              79 H  
where 79 H is the op code of the instruction ‘MOV A, C’.

### **Two Byte Instruction**

In a two byte instruction, first byte of the instruction is its op code and second byte is the given data.

Such instruction is stored in two consecutive memory locations.

For example

|             |
|-------------|
| MVI A, 06 H |
| OUT 03 H    |
| ANI 76 H    |
| etc.        |

In order to store the instruction say ‘MVI A, 06 H’ in the memory locations of the computer, we have to use two consecutive memory locations. In one memory location the op code of MVI A is to be stored and in the second location the data 06H is to be stored. This type of instruction to be stored in two locations say in 2101 H and 2102 H is given below:

|       |      |                    |
|-------|------|--------------------|
| 2101H | 3E H | (op code of MVI A) |
| 2102H | 06 H | (given data)       |

### Three Byte Instruction

In a three byte instruction, first byte is used for its op code and second and third bytes are used for 16 bit address. Such an instruction is stored in three consecutive memory locations.

For example

|            |
|------------|
| LDA 2100 H |
| STA 3000 H |
| JMP 2500 H |
| etc.       |

In order to store the instruction say ‘LDA 2100 H’ three consecutive memory locations are to be used. In the first memory location op code of the instruction is stored, in second location lower byte of the address is to be stored and in the third byte upper byte of the address is to be stored. This instruction loaded in three consecutive memory location 2000H, 2001H and 2002H is given below:

|       |      |                                |
|-------|------|--------------------------------|
| 2000H | 3A H | (op code of LDA)               |
| 2001H | 00 H | (Lower byte of address 2100 H) |
| 2002H | 21 H | (Upper byte of address 2100 H) |

The type of instructions of all the 42 instructions of SAP-II computer is also given in table 3.2.

### 3.6 FLAGS

As already discussed, there are two flags associated with the accumulator of SAP-II computer, sign flag (S) and Zero flag (Z). These flags may be set or reset during certain instructions. If the accumulator content is negative after execution of certain instruction the sign flag is set otherwise it is reset. Similarly, if the accumulator content is zero after the operation of certain instruction the zero flag is set otherwise reset. These instructions which affect the flags are listed in table 3.3.

**Table 3.3**

| Instruction | Flags affected |
|-------------|----------------|
| ADD         | S, Z           |
| SUB         | S, Z           |
| INR         | S, Z           |
| DCR         | S, Z           |
| ANA         | S, Z           |
| ORA         | S, Z           |
| ANI         | S, Z           |
| ORI         | S, Z           |
| XRI         | S, Z           |

For example in ADD instruction say ADD B, the contents of B register gets added with the accumulator content and after addition the result is stored in accumulator. If the result is zero then zero flag is set otherwise reset. Similarly, if the result is negative the sign flag is set otherwise reset.

In INR or DCR instructions both sign and zero flags are affected. For example if there is an instruction INR B, the content of B register is incremented by sending the contents to accumulator and adds 1 to it. The result is then sent back to B register. If the accumulator goes negative while INR instruction is executed, the sign flag is set; and if the accumulator content is zero then the zero flag is set.

### 3.7 ASSEMBLY LANGUAGE PROGRAMMING

The programming of the problem is generally written in assembly language. The assembly language is written in mnemonics. The mnemonics are the initials or short form of the English word of the operation to be performed by the instruction. Assembly language statements are written in standard format as given below:

| <b>Label</b>    | <b>Mnemonic</b>   | <b>Operand</b> | <b>Comment</b> |
|-----------------|---|----------------|----------------|
| <b>Label</b>    | A label is a symbol or group of symbols used to represent an address of the location which is not specifically known at the time of program is written. The label can be one to six characters, the first character of which must be a letter. Following are the acceptable labels.<br>NEXT, BACK, DELAY, A2 etc. |                |                |
| <b>Mnemonic</b> | Short form of the operation to be performed.  |                |                |
| <b>Operand</b>  | Operand is the data on which the operation is performed. It can be a data, memory address, register or port address.  |                |                |
| <b>Comment</b>  | The comment statement is started with the semicolon. It gives the idea of the program to the user. The comments are not the part of the machine language program.   |                |                |

The program written in assembly language can be converted to machine language by hand. For writing the program in machine language, the starting address, where the program is to be stored should be known. Now the op code of the instruction is to be written in first location (starting address) and in the consecutive memory locations data /address of the operand is written. While storing the address in the memory locations, lower byte of the address is stored first then the upper byte as discussed above.

**Example 3.1** Write assembly language program using the instructions of SAP-II computer of the following statement. Also write the program in machine language.

Load the contents of memory locations 2100 H and 2101 H in B-register and C-register respectively. The content of memory locations 2100 H and 2101H are 16 H and 19 H respectively.

**Solution.**

| <b>Label</b> | <b>Mnemonic</b> | <b>Operand</b> | <b>Comment</b> |
|--------------|-----------------|----------------|----------------|
|--------------|-----------------|----------------|----------------|

|        |       |  |
|--------|-------|--|
| LDA    | 2100H | ; Loads the content of 2100H into accumulator.                         |
| MOV B, | A     | ; moves the content of accumulator to B-register ( $B \leftarrow A$ ). |
| LDA    | 2101H | ; Loads the content of 2101H into accumulator.                         |
| MOV C, | A     | ; moves the content of accumulator to C-register ( $C \leftarrow A$ ). |
| HLT    |       | ; Stop processing.   |

The assembly language program may be converted to machine language by writing the op codes of the instructions as given below. Let the starting address of the program is 2000H.

| Memory Address | Content              |
|----------------|----------------------|
| 2000 H         | 3A H      LDA 2100 H |
| 2001 H         | 00 H                 |
| 2002 H         | 21 H                 |
| 2003 H         | 47 H      MOV B, A   |
| 2004 H         | 3A H      LDA 2101 H |
| 2005 H         | 01 H                 |
| 2006 H         | 21 H                 |
| 2007 H         | 4F H      MOV C, A   |
| 2008 H         | 76 H      HLT        |
| 2101 H         | 16 H      Data       |
| 2102 H         | 19 H      Data       |

**Example 3.2** Write an assembly language program to find the 2's complement of a hexadecimal number. The hexadecimal number 6A H is stored in memory location 2100H and the answer is to be stored in 2101 H. Use SAP-II instructions to program it.

**Solution.**

| Label | Mnemonic | Operand | Comment  |
|-------|----------|---------|--|
|       | LDA      | 2100 H  | ; Loads the content of 2100 H into accumulator.                    |
|       | CMA      |         | ; Complements the accumulator content (1's complement).            |
|       | INR A    |         | ; 1 is added to the accumulator content to get the 2's complement. |
|       | STA      | 2101 H  | ; Loads the accumulator contents into memory location 2101 H.      |
|       | HLT      |         | ; Stop processing.   |

**Example 3.3** Write an assembly language program using SAP-II instructions to add two numbers (decimal) 38 and 64, then subtract decimal number 3 from the sum. The final answer is to be stored in memory location 2100 H.

**Solution.** First of all decimal numbers 38 and 64 should be converted to hexadecimal numbers, as it works in hexadecimal.

Decimal number 38 = 26 H

Decimal number 64 = 40 H

| <b>Label</b> | <b>Mnemonic</b> | <b>Operand</b> | <b>Comment</b>  |
|--------------|-----------------|----------------|---|
|              | MVI A,          | 26 H           | ; Loads the first number to accumulator.  |
|              | MVI B,          | 40 H           | ; Loads the second number to B-register.  |
|              | MVI C,          | 03 H           | ; Loads the third number to C-register.   |
|              | ADD B           |                | ; Adds the contents of B-register with the contents of accumulator and the answer is stored in A. |
|              | SUB C           |                | ; Content of C gets subtracted from accumulator and difference is stored in A.                    |
|              | STA             | 2100 H         | ; Answer is stored in 2100 H location.  |
|              | HLT             |                | ; Stop processing.  |

**Example 3.4** Write a program in assembly language for SAP-II computer to mask off the least significant 4 bits of a given hexadecimal number. The answer should be stored in memory location 2200 H. Let the given number is B3 H.

**Solution.** The binary equivalent of B3 H is 1011 0011. The masking of the 4 least significant bits 0011 means to make 0011 to 0000. However, the four most significant bits should not be changed.

This can be done if the given number is ANDed with F0 H (1111 0000). In doing so when 4 most significant bits are ANDed with 1111 no change will be there, but 4 least significant bits will be 0000 as required. The assembly language program for this will be as follows:

| <b>Label</b> | <b>Mnemonic</b> | <b>Operand</b> | <b>Comment</b>   |
|--------------|-----------------|----------------|--|
|              | MVI A,          | B3 H           | ; Loads the number B3H to accumulator.                               |
|              | ANI             | F0 H           | ; ANDs the accumulator with F0H and answer is loaded to accumulator. |
|              | STA             | 2200 H         | ; Answer is stored in 2200 H location.                               |
|              | HLT             |                | ; Stop processing.   |

**Example 3.5** Write a program in assembly language for SAP-II computer to load a number 79 H in B-register and mask off all bits except A<sub>2</sub> bit. The result is to be transferred to C-register.

**Solution.** To mask off the third LSB (A<sub>2</sub> bit) the ANDing of the given content will be with FBH (11111011). The program will be as given below:

| <b>Label</b> | <b>Mnemonic</b> | <b>Operand</b> | <b>Comment</b>                                    |
|--------------|-----------------|----------------|---|
|              | MVI B,          | 79 H           | ; Loads the number 79 H to accumulator.           |
|              | MOV A,          | B              | ; Moves the content of B-register to accumulator. |

|   |        |      |   |
|---|--------|------|---|
|   | ANI    | FB H | ; ANDs the accumulator with FB H and answer is loaded to accumulator. |
| . | MOV C, | A    | ; Answer is loaded to C-register.                                     |
|   | HLT    |      | ; Stop processing.  |

**Example 3.6** Write a program in assembly language for SAP-II computer to interchange (swap) the contents of two memory locations 2100 H and 2101 H.

**Solution.** The program is given below which is self explanatory.

| Label | Mnemonic | Operand | Comment  |
|-------|----------|---------|--|
|       | LDA      | 2100H   | ; Loads the content of 2100H into accumulator. |
|       | MOV B,   | A       | ; Moves the content of Acc to B-register.      |
|       | LDA      | 2101 H  | ; Loads the content of 2101H into accumulator. |
|       | STA      | 2100 H  | ; Loads the acc. content to 2100 H location.   |
|       | MOV A,   | B       | ; Moves the content of B-register to Acc.      |
|       | STA      | 2101 H  | ; Loads the acc. content to 2101 H location.   |
|       | HLT      |         | ; Stop processing.                             |

**Example 3.7** Write a program in assembly language for SAP-II computer to multiply two decimal numbers 23 and 9 and store the answer in some memory location. Also write this program in machine language.

**Solution.** Hexadecimal equivalent of decimal number 23 is 17 H.

The multiplication of these two numbers may be obtained by adding 23 (17 H) by 9 times in the accumulator which should be 00 H at the beginning. So the program of this problem may be as given below:

| Label | Mnemonic | Operand | Comment  |
|-------|----------|---------|--|
|       | MVI A,   | 00 H    | ; Loads the acc. 00 H. [A] ← 00H .             |
|       | MVI B,   | 17 H    | ; Loads B-register with 17H [B] ← 17H .        |
|       | MVI C,   | 09 H    | ; Loads C-register with 09H [C] ← 09H .        |
| AGAIN | ADD B    |         | ; Adds the content of B-register to acc.       |
|       | DCR C    |         | ; Decrements C-register.                       |
|       | JZ       | END     | ; Checks for zero; if zero jump to END.        |
|       | JMP      | AGAIN   | ; Repeats the addition.                        |
| END   | STA      | 2100    | ; Stores the answer to memory location 2100 H. |
|       | HLT      |         | ; Stop processing.                             |

The program can be written in machine language starting at address 2000 H as given below:

| <b>Memory Address</b> |      | <b>Content</b> |
|-----------------------|------|----------------|
| 2000 H                | 3E H | MVI A, 00 H    |
| 2001 H                | 00 H |                |
| 2002 H                | 06 H | MVI B, 17 H    |
| 2003 H                | 17 H |                |
| 2004 H                | 0E H | MVI C, 09 H    |
| 2005 H                | 09 H |                |
| 2006 H                | 80 H | ADD B          |
| 2007 H                | 0D H | DCR C          |
| 2008 H                | CA H | JZ 200E H      |
| 2009 H                | 0E H |                |
| 200A H                | 20 H |                |
| 200B H                | C3 H | JMP 2006 H     |
| 200C H                | 06 H |                |
| 200D H                | 20 H |                |
| 200E H                | 32 H | STA 2100 H     |
| 200F H                | 00 H |                |
| 2010 H                | 21 H |                |
| 2011 H                | 76 H |                |

Alternative method of writing this program using JNZ is as given below:

| <b>Label</b> | <b>Mnemonic</b> | <b>Operand</b> | <b>Comment</b>   |
|--------------|-----------------|----------------|--|
|              | MVI A,          | 00 H           | ; Loads the acc. 00 H. $[A] \leftarrow 00H$ .                    |
|              | MVI B,          | 17 H           | ; Loads B-register with 17 H<br>$[B] \leftarrow 17H$ .           |
|              | MVI C,          | 09 H           | ; Loads C-register with 09 H<br>$[C] \leftarrow 09H$ .           |
| AGAIN        | ADD B           |                | ; Adds the content of B-register to acc.                         |
|              | DCR C           |                | ; Decrements C-register.   |
|              | JNZ             | AGAIN          | ; Repeats the addition if the content of C-register is not zero. |
|              | STA             | 2100 H         | ; Stores the answer to memory location 2100 H.                   |
|              | HLT             |                | ; Stop processing.   |

This program can also be written by using subroutine program as follows:

#### Main Program

| <b>Label</b> | <b>Mnemonic</b> | <b>Operand</b> | <b>Comment</b>   |
|--------------|-----------------|----------------|--|
|              | MVI A,          | 00 H           | ; Loads the acc. 00 H. $[A] \leftarrow 00H$ .          |
|              | MVI B,          | 17 H           | ; Loads B-register with 17 H<br>$[B] \leftarrow 17H$ . |
|              | MVI C,          | 09 H           | ; Loads C-register with 09H<br>$[C] \leftarrow 09H$ .  |
|              | CALL            | MUL            | ; Calls subroutine program for multiplication.         |

|  |     |        |  |
|--|-----|--------|--|
|  | STA | 2100 H | ; Stores the answer to memory location 2100 H. |
|  | HLT |        | ; Stop processing.                             |

### Subroutine Program

| Label | Mnemonic | Operand | Comment  |
|-------|----------|---------|--|
| MUL   | ADD B    |         | ; Adds the content of B-register to acc.                         |
|       | DCR C    |         | ;Decrement C-register.   |
|       | JNZ      | MUL     | ; Repeats the addition if the content of C-register is not zero. |
|       | RET      |         | ; Returns to main program.                                       |

### 3.8 DELAY CALCULATIONS

Through programming time delay can be introduced, which is very useful in various applications such as digital clocks, traffic controls, digital process control and other data transfer controls. Time delay can be introduced by loading the registers with some desired number and then decremented through the loops to zero value. The delay introduced in the system will depend on the clock period of the system and the number of times the instructions are executed inside the loop.

To generate very small delay only one register can be used. Consider a subroutine program given below in which C-register is loaded with 10 H (decimal number 16). It is decremented in a loop to make it zero. Figure 3.8 shows the flow chart for the same.

| Label | Mnemonic | Operand | No. of T-states |
|-------|----------|---------|-----------------|
|       | MVI C,   | 10 H    | 7               |
| LOOP  | DCR C    |         | 4               |
|       | JNZ      | LOOP    | 10/7            |
|       | RET      |         | 10              |

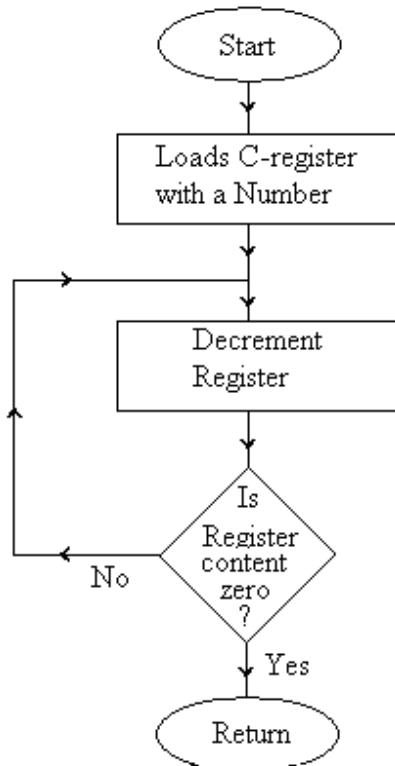
In this program the instruction MVI C, 10 H is executed only once and it takes 7 T-states to execute.

The instruction DCR C is executed 16 times and thus takes  $16 \times 4 = 64$  T-states, since DCR instruction takes 4 T-states for its execution.

For the execution of JNZ instruction, it will go to loop 15 times, as the content of C-register will not be zero. So  $15 \times 10 = 150$  T-states will be used in its calculation (till the content of C-register is not zero). When the contents of C-register becomes zero, it will jump to loop and it will take 7 T-states. For the execution of RET statement it will take 10 T-states.

Thus total number of T-states taken for the execution of this program will be given by:

| Mnemonic     | T-states                          |
|--------------|-----------------------------------|
| MVI C, 10 H  | $7 \times 1 = 7$                  |
| DCR C        | $4 \times 16 = 64$                |
| JNZ          | $10 \times 15 + 1 \times 7 = 157$ |
| RET          | $1 \times 10 = 10$                |
| <b>Total</b> | <b>238 T-states</b>               |



**Fig. 3.8**

Total 238 T-states are used for the execution of this program. If the system frequency is 2 MHz, the time taken by one T-state is  $\frac{1}{2 \times 10^6} \text{ sec} = 0.5 \mu\text{sec}$ .

The total time delay introduced in the execution of this program is:

$$\begin{aligned}
 \text{Time delay} &= 238 \times 0.5 \mu\text{sec} \\
 &= 119 \mu\text{sec} \\
 &= 0.119 \text{ msec}
 \end{aligned}$$

Maximum delay with this register C may be obtained by loading the number FF H (decimal number 255) in C-register. The maximum delay thus is:

|              |                                     |
|--------------|-------------------------------------|
| MVI C, FF H  | $7 \times 1 = 7$                    |
| DCR C        | $4 \times 255 = 1020$               |
| JNZ          | $10 \times 254 + 1 \times 7 = 2547$ |
| RET          | $1 \times 10 = 10$                  |
| <b>Total</b> | <b>3584 T-states</b>                |

$$\begin{aligned}
 \text{Time delay} &= 3584 \times 0.5 \mu\text{sec} \\
 &= 1.792 \text{ msec}
 \end{aligned}$$

Total delay introduced by the computer using only one register may be given in the following general form as:

$$T_{\text{Delay}} = [1x7 + 4N + (N - 1)x10 + 1x7 + 10] \times \text{time of one T-state.}$$

where N is the decimal number given with the register.

$$\begin{aligned}
 &= [14N + 14] \times \text{time of one T-state.} \\
 &= 14[N + 1] \times \text{time of one T-state.}
 \end{aligned}$$

As discussed earlier the total time delay introduced by a register is 1.79 millisec if system clock frequency is 2 MHz. To introduce more time delay two registers may be used as shown in the flow chart shown in figure 3.9, in which register B is used for the outer loop and register C is used for the inner loop.

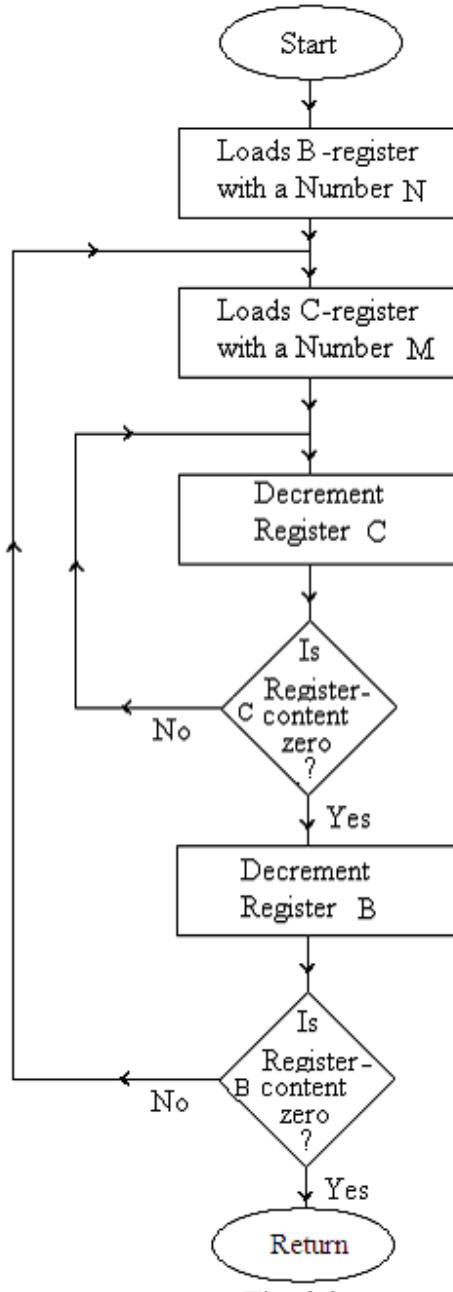


Fig. 3.9

| <b>Label</b> | <b>Mnemonic</b> | <b>Operand</b> | <b>No. of T-states</b> |
|--------------|-----------------|----------------|------------------------|
|              | MVI B,          | M H            | 7                      |
| LOOP1        | MVI C,          | N H            | 7                      |
| LOOP         | DCR C           |                | 4                      |

|       |       |      |
|-------|-------|------|
| JNZ   | LOOP  | 10/7 |
| DCR B |       | 4    |
| JNZ   | LOOP1 | 10/7 |
| RET   |       | 10   |

Total T-states used in this program will be given by:

$$\begin{aligned}
 &= 1x7 + M[1x7 + Nx4 + (N-1)X10 + 1x7] + (Mx4) + (M-1)x10 + 1x7 + 1x10 \\
 &\quad \text{MVI} \quad \text{DCR} \quad \text{JNZ} \quad \text{JNZ} \quad \text{DCR} \quad \text{JNZ} \quad \text{JNZ} \quad \text{RET} \\
 &= 1x7 + M[7 + 4N + 10N - 10 + 7] + 4M + 10M - 10 + 7 + 10 \\
 &= 14 + M[14N + 4] + 14M \\
 &= 14MN + 18M + 14
 \end{aligned}$$

This is the general form for calculating the number of T-states of the program given above, in which M is the counts in register-B (in decimal number) in outer loop and N is the counts in register-C (in decimal number) in inner loop.

Total time delay is therefore given by:

$$T_{\text{Delay}} = (14MN + 18M + 14) \times \text{Time of one T-state}$$

**Example 3.8** Write a delay subroutine program for SAP-II computer to introduce a time delay of 1 millisec using only one register. Let the system frequency is 2 MHz.

**Solution.** System frequency = 2 MHz

Time delay of T-state = 0.5  $\mu$ sec.

We shall now find the value of N to be stored to the register.

$$\begin{aligned}
 T_{\text{Delay}} &= 14[N+1] \times 0.5 \mu\text{sec} \\
 1000 \mu\text{sec} &= 14[N+1] \times 0.5 \mu\text{sec} \\
 N &= \frac{1000}{7} - 1 \\
 &= \frac{993}{7} \cong 142 \text{ decimal number} \\
 142 &= 8E H
 \end{aligned}$$

So the program for the delay of 1 millisec is as given below:

| Label | Mnemonic | Operand |
|-------|----------|---------|
|       | MVI C,   | 8E H    |
| LOOP  | DCR C    |         |
|       | JNZ      | LOOP    |
|       | RET      |         |

**Example 3.9** Write a delay subroutine program for SAP-II computer to introduce a time delay of 10 millisec using two registers. Decimal number 10 may be stored in register B (for outer loop). Let the system frequency is 2 MHz.

**Solution.** System frequency = 2 MHz

In this case  $M = 10_{10} = 0A H$

N is to be calculated as:

$$\begin{aligned}
 T_{\text{Delay}} &= (14MN + 18M + 14) \times \text{Time of one T-state} \\
 \text{Time of T-state} &= 0.5 \mu\text{sec} \\
 10ms &= (14 \times 10 \times N + 18 \times 10 + 14) \times 0.5 \mu\text{s}
 \end{aligned}$$

$$10000\mu s = (140N + 194) \times 0.5\mu s$$

$$20000 = (140N + 194)$$

$$140N = (20000 - 194)$$

$$N = \frac{19806}{140} = 142 = 8E H$$

The Subroutine program for the required time delay is therefore given as:

| <b>Label</b> | <b>Mnemonic</b> | <b>Operand</b> |
|--------------|-----------------|----------------|
|              | MVI B,          | 0A H           |
| LOOP1        | MVI C,          | 8E H           |
| LOOP         | DCR C           |                |
|              | JNZ             | LOOP           |
|              | DCR B           |                |
|              | JNZ             | LOOP1          |
|              | RET             |                |

**Example 3.10** (a) What will be the time delay introduced in the computer, if the following subroutine program is executed. Assume the system frequency is 2 MHz.

|       |        |       |
|-------|--------|-------|
|       | MVI B, | 64 H  |
| LOOP1 | MVI C, | 8E H  |
| LOOP  | DCR C  |       |
|       | JNZ    | LOOP  |
|       | DCR B  |       |
|       | JNZ    | LOOP1 |
|       | RET    |       |

(b) How much maximum delay can be introduced by this subroutine program?

(c) Modify this program to introduce a time delay of 1 sec.

(d) Modify this program to introduce a time delay of 10 sec.

**Solution.** In this subroutine program  $M = 64 H = 100_{10}$

$$N = 8E H = 142_{10}$$

Time of one T-state = 0.5  $\mu$ sec

$$\begin{aligned} T_{Delay} &= (14MN + 18M + 14) \times \text{Time of one T-state} \\ &= (14MN + 18M + 14) \times 0.5\mu\text{sec} \\ &= (14 \times 100 \times 142 + 18 \times 100 + 14) \times 0.5\mu\text{sec} \\ &= (198800 + 1800 + 14) \times 0.5\mu\text{sec} \\ &= 200614 \times 0.5\mu\text{sec} \\ &= 100307 \mu\text{sec} \\ &= 100.3 \text{ msec} \\ &\approx 0.1 \text{ sec} \end{aligned}$$

Approx. 0.1 sec delay is introduced by this subroutine program.

(b) The maximum time delay that can be introduced by this subroutine program is calculated if we consider the maximum value of M and N as FF H.

$$\text{i.e. } M = FF H = 255_{10}$$

$$N = FF H = 255_{10}$$

$$\begin{aligned} T_{Delay} &= (14MN + 18M + 14) \times \text{Time of one T-state} \\ &= (14MN + 18M + 14) \times 0.5\mu\text{sec} \end{aligned}$$

$$\begin{aligned}
 &= (14 \times 255 \times 255 + 18 \times 255 + 14) \times 0.5 \mu\text{sec} \\
 &= 914954 \times 0.5 \mu\text{sec} \\
 &= 457477 \mu\text{sec} \\
 &\approx 0.46 \text{ sec}
 \end{aligned}$$

- (c) To introduce a time delay of 1 sec, the given program can be run 10 times using one more register say A-register. The modified program is given below:

| <b>Label</b> | <b>Mnemonic</b> | <b>Operand</b> |
|--------------|-----------------|----------------|
|              | MVI A,          | 0A H           |
| LOPP3        | MVI B,          | 64 H           |
| LOOP2        | MVI C,          | 8E H           |
| LOOP1        | DCR C           |                |
|              | JNZ             | LOOP1          |
|              | DCR B           |                |
|              | JNZ             | LOOP2          |
|              | DCR A           |                |
|              | JNZ             | LOOP3          |
|              | RET             |                |

- (d) To introduce a time delay of 10 sec, in the above program 64 H ( $100_{10}$ ) can be taken in place of 0A H ( $10_{10}$ ). The subroutine program will be as given below:

| <b>Label</b> | <b>Mnemonic</b> | <b>Operand</b> |
|--------------|-----------------|----------------|
|              | MVI A,          | 64 H           |
| LOPP3        | MVI B,          | 64 H           |
| LOOP2        | MVI C,          | 8E H           |
| LOOP1        | DCR C           |                |
|              | JNZ             | LOOP1          |
|              | DCR B           |                |
|              | JNZ             | LOOP2          |
|              | DCR A           |                |
|              | JNZ             | LOOP3          |
|              | RET             |                |

**Example 3.11** Write a program in assembly language of SAP-II computer to count continuously in hexadecimal from FF H to 00 H. Use subroutine program also to set up a one millisecond delay between each count and display the number at one of the output port. Assume the system frequency is 2 MHz.

**Solution.** For its programming load a count 00 H in one register say B-register. Decrement the content of B-register, so that in B-register the counts are FF H. Display this count on one of the output port say 04 H. Then introduce a delay of 1 millisec in a subroutine program. After the delay, decrement the counts in B-register and proceed continuously as discussed above.

The program is given below:

#### Main Program

| <b>Label</b> | <b>Mnemonic</b> | <b>Operand</b> | <b>Comment</b>  |
|--------------|-----------------|----------------|---|
|              | MVI B,          | 00 H           | ; Loads the count 00 H in B-register.<br>[B] $\leftarrow$ 00H . |

|        |            |   |
|--------|------------|---|
| START  | DCR B      | ; Decrements the counts of B-register.  |
|        | CALL DELAY | ; Calls the delay subroutine program.   |
| MOV A, | B          | ; Moves the content of B to A.          |
| OUT    | PORT 04 H  | ; Displays the data at the output port. |
| JMP    | START      |   |

### Subroutine Program (Delay program of 1msec)

| <b>Label</b> | <b>Mnemonic</b> | <b>Operand</b> |
|--------------|-----------------|----------------|
| DELAY        | MVI C           | 8E H           |
| LOOP         | DCR C           |                |
|              | JNZ             | LOOP           |
|              | RET             |                |

Subroutine program is the same as discussed in example 3.7.

**Example 3.12** Suppose SAP-II computer can input a data (one byte) from port 2. Write an assembly language program to find if the bit 1 ( $A_1$ ) is zero or one. If the bit 1 is one, it should load the accumulator with ASCII Y otherwise ASCII N. The accumulator data should be available at output port 4. The hexadecimal number for ASCII Y is 59 H and for ASCII N is 4E H.

**Solution.**

| <b>Label</b> | <b>Mnemonic</b> | <b>Operand</b> | <b>Comment</b>                             |
|--------------|-----------------|----------------|--|
|              | IN              | 02 H           | ; Input a byte (data) from input port 02H. |
|              | ANI             | 02 H           | ; Isolate bit $A_1$ .                      |
|              | JZ              | NO             | ; Jump if $A_1$ is zero.                   |
|              | MVI A,          | 59 H           | ; Loads Y to Acc.                          |
|              | JMP             | END            | ; Jump to END.                             |
| NO           | MVI A,          | 4E H           | ; Loads N to Acc.                          |
| END          | OUT             | 04 H           | ; Output is available at port 04 H.        |
|              | HLT             |                | ; Stop processing.                         |

**Example 3.13** Write a program in assembly language of SAP-II computer to subtract a number stored in memory location 2100 H from the number in memory location 2101 H using addition method. The result should be stored in memory location 2102 H. If the result is negative, memory location should be loaded with 00 H.

**Solution.** In this problem subtraction is to be carried out using addition method. So we have to add 2's complement of the number in memory location 2101 H.

| <b>Label</b> | <b>Mnemonic</b> | <b>Operand</b> | <b>Comment</b>   |
|--------------|-----------------|----------------|--|
|              | LDA             | 2100 H         | ; Loads the Acc. the content stored in 2100 H.             |
|              | CMA             |                | ; Takes 1's complement of the number in Acc.               |
|              | INR A           |                | ; Acc. content is added with to get 2's complement in Acc. |

|     |        |        |  |
|-----|--------|--------|--|
|     | MOV B, | A      | ; Moves the Acc. content to B-register.                        |
|     | LDA    | 2101 H | ; Loads the Acc. the content stored in 2101 H (second number). |
|     | ADD B  |        | ; Adds 2's complement of the number with Acc.                  |
|     | JM     | END    | ; Jump if the sum is negative.                                 |
|     | STA    | 2102 H | ; Stores the answer if positive.                               |
|     | HLT    |        | ; Stop processing.   |
| END | MVI A, | 00 H   | ; Moves 00 H to Acc.   |
|     | STA    | 2102 H | ; If answer is negative store 00 H to 2102 H.                  |
|     | HLT    |        | ; Stop processing.   |

**Example 3.14** Write a program in assembly language using SAP-II instructions that inputs a byte from port 2 and determine if decimal number is even or odd. If the input byte is even load FF H otherwise 00 H to memory location 2500 H.

**Solution.**

| <b>Label</b> | <b>Mnemonic</b> | <b>Operand</b> | <b>Comment</b>                             |
|--------------|-----------------|----------------|--|
|              | IN              | 02 H           | ; Input a byte (data) from input port 02H. |
|              | ANI             | 01 H           | ; Isolate bit A <sub>0</sub> .             |
|              | JNZ             | ODD            | ; Jump if Odd.                             |
|              | MVI A,          | FF H           | ; Loads FF H to Acc.                       |
|              | JMP             | END            | ; jump to END.                             |
| ODD          | MVI A,          | 00 H           | ; Loads 00 H to Acc.                       |
| END          | STA             | 2500 H         | ; Stores the answer in 2500 H.             |
|              | HLT             |                | ; Stop processing.                         |

## PROBLEMS

1. Draw the block diagram of the architecture of SAP-II computers. Discuss the working of each block.
2. What is the difference between the architectures of SAP-I and SAP-II computers?
3. How the sign and zero flags work in arithmetic and logic unit of SAP-II computers?
4. Name and discuss the five different categories in which the instruction set of SAP-II computers are divided.
5. Name and discuss the different addressing modes to specify the instruction of SAP-II computers.
6. Discuss Implied addressing to specify the data of instructions of SAP-II computers.
7. Describe with examples one byte, two byte and three byte instructions of SAP-II computers.
8. What is the difference between assembly language program and machine language program? What do you understand by mnemonics?

9. How delay is introduced through software in SAP-II computers using only one register? How much maximum delay can be introduced with one register if the frequency of the clock is (i) 1 MHz (ii) 3 MHz?
  10. How much delay can be introduced using two registers in SAP-II computers, if the system frequency is 2 MHz?
  11. Write the subroutine program to introduce a delay of 1 sec using all the three registers of SAP-II computers. Further assume that the system clock frequency is 2 MHz.
  12. Write a program in assembly language using SAP-II instructions to multiply the two decimal numbers 13 and 10. The answer should be stored in memory location 2100 H.
  13. Write a program in assembly language using SAP-II instructions to complement a number lying at 2100 H memory location. Store the complement at 2101 H.
  14. Write a program in assembly language using SAP-II instructions to perform the following arithmetic operation:  

$$X + Y - Z - W$$

where X, Y, Z and W are hexadecimal numbers stored in memory locations 2101 H to 2104 H. The final answer should be stored in 2100 H. Assume that there is no carry or borrow.
  15. Write a program in assembly language using SAP-II instructions to get 2's complement of a number stored in memory location 2501 H. Store the answer at 2502 H .
  16. Write a program in assembly language using SAP-II instructions to add decimal numbers 70 and 36. Answer is to be stored in memory location 2100 H.
  17. Write a program in assembly language using SAP-II instructions to multiply two decimal numbers 33 and 6 and store the answer in memory location 2100 H. Use subroutine for multiplication.
  18. Write a subroutine program (in assembly language of SAP-II) to introduce a time delay of 20 msec. Let the system frequency is 2 MHz.
  19. Write a subroutine program (in assembly language of SAP-II) to introduce a time delay of 1 minute. Let the system frequency is 1 MHz.
  20. Write an assembly language program using SAP-II instructions to mask off A<sub>1</sub> and A<sub>2</sub> bits of a given number. Let the given number is 6E H.
-

# 4

## SAP – III

---

In this chapter, programming model and instruction set of SAP-III computer will be discussed. After the study of this computer we will be in a position to understand the details of the popular 8-bit microprocessor 8085 with ease.

### 4.1 PROGRAMMING MODEL OF SAP-III COMPUTER

One step ahead of the evolution of modern computers is the 8-bit microcomputer named as SAP-III computer. The CPU of this computer is upward compatible with 8085 microprocessor. The programming model or software model of SAP-III computer is given in figure 4.1. It consists of some more registers than SAP-II computers. In addition to Accumulator (A), B and C registers it contains four more 8-bit registers named as D, E, H and L registers. Because of these registers the flexibility in programming is much more. With special instructions the registers B, C, D, E, H and L may be used as extended register pairs B-C register pair, D-E register pair and H-L register pair, so that 16-bit data may be operated with these registers. It has 16-bit Program counter and 16-bit Stack pointer. The stack pointer is used for stack purposes which will be discussed later in this chapter.

| Accumulator (8)           | FLAG (4)       |
|---------------------------|----------------|
| B-register (8)            | C-register (8) |
| D-register (8)            | E-register (8) |
| H-register (8)            | L-register (8) |
| Stack Pointer (SP) (16)   |                |
| Program Counter (PC) (16) |                |

FLAG

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| S | Z | X | X | X | P | X | C | Y |
|---|---|---|---|---|---|---|---|---|

X - Undefined

**Fig. 4.1**

SAP-III computer has a four-bit flag register associated with Accumulator. The four flags are Sign flag (S), Zero flag (Z), Carry flag (CY) and Parity flag (P).

**Sign Flag (S)**

The sign flag is set ( $S = 1$ ), if accumulator content is negative and it is reset ( $S = 0$ ) if accumulator content is positive. The logic circuit is the same as discussed for SAP-II computers.

### Zero Flag (Z)

The zero flag is set ( $Z = 1$ ), if accumulator content is zero and it is reset ( $Z = 0$ ) if accumulator content is not zero. The logic circuit is the same as discussed for SAP-II computers.

### Carry Flag (CY)

The carry flag (CY) is used to detect overflow in some arithmetic and logic operations. In SAP-III computer, the accumulator of CPU is only 8-bit wide. The unsigned binary numbers from 0 to 255 or signed numbers (2's complement) from  $-128$  to  $+127$  can be the content of the accumulator. The arithmetic operations addition and subtraction are performed using 2's complement adder / subtractor circuit. The logic circuit diagram of such a 2's complement adder / subtractor is shown in figure 4.2.

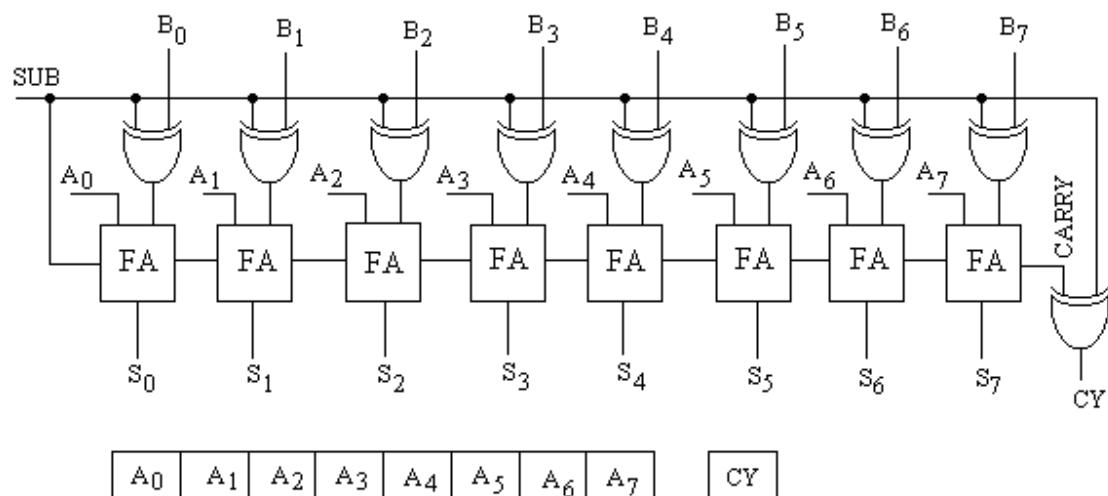


Fig. 4.2

In this circuit one input of each of the exclusive – OR gates are connected to common terminal named as SUB; this common terminal is also connected to carry bit of first full adder. During the addition, SUB terminal is kept low and for subtraction it is kept high. The working of this circuit may be described as given below.

The accumulator content directly goes to full adders. The content, which is to be added to or subtracted from the accumulator content is applied to full adders through the exclusive OR gates. SUB terminal being low during the addition, so the exclusive-OR gates send directly the addend to the full adders. The full adders add the two contents. If there is any overflow, CARRY becomes high which finally sets CY flag. If on the other hand there is no overflow, CARRY is low and CY flag is reset.

So  $CY = 1$ , if there is carry and  $CY = 0$ , if there is no carry.

Similarly, during subtraction  $SUB = 1$  and the exclusive OR gates converts the subtrahend to its equivalent 1's complement. Since SUB terminal ( $SUB = 1$ ) is connected to first full adder so 1's complement of the subtrahend gets converted to its equivalent 2's complement. For subtraction, 2's complement of the subtrahend is added with the

accumulator content. If the CARRY signal is high ( $CARRY = 1$ ), there is an end around carry (EAC), the final exclusive OR gate gives  $CY = 0$  indicating there is no borrow. On the other hand, if there is no CARRY ( $CARRY = 0$  or no EAC), carry flag CY will be set ( $CY = 1$ ) indicating that there is borrow.

So  $CY = 0$  (Carry flag is restet) there is no carry or borrow.

$CY = 1$  (Carry flag is set) there is carry or borrow.

The carry flag acts as carry bit for addition and it works as borrow bit for subtraction.

### Parity Flag (P)

In addition to sign, carry and zero flags there is also a parity flag. The parity flag is set if there are even number of 1's in the accumulator and it is reset if there are odd number of 1's.

So  $P = 1$  for even parity

and  $P = 0$  for odd parity.

The logic circuit diagram of the parity bit generator for showing the correct parity is shown in figure 4.3.

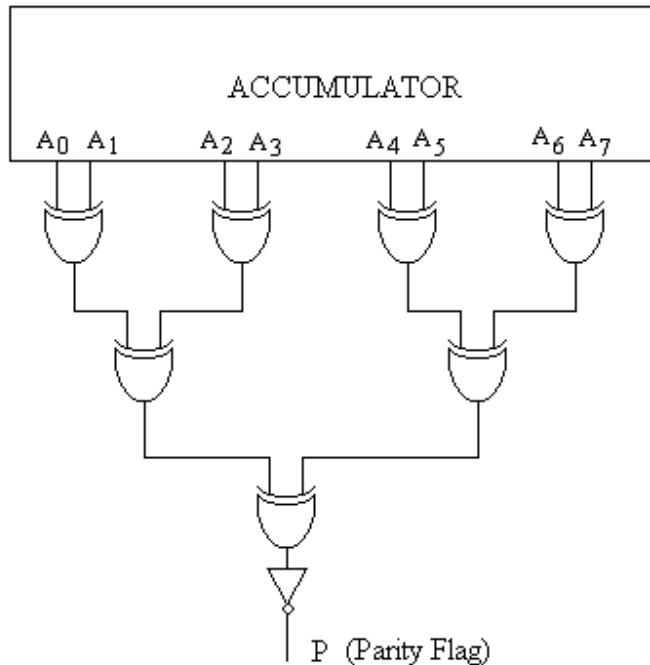


Fig. 4.3

## 4.2 INSTRUCTION SET OF SAP-III COMPUTER

All the instructions which have been discussed in SAP-II computers are also used in SAP-III computers. In addition to these instructions there are many more instructions which will be discussed here.

The instruction set of SAP-III computers has been classified into following groups.

1. Data Transfer Group
2. Arithmetic Group
3. Branch Group
4. Stack, Input/Output and Machine Control Group

#### 4.2.1 Data Transfer Group

The function of data transfer group of instructions is to transfer the data from register to register, register to memory and also immediate transfer of data (given) to memory location. This group of instruction is also there in SAP-II computers but in SAP-III computers more instruction are there due to more registers. Data transfer group of instructions are given in table 4.1.

These data transfer instructions can further be subdivided on the basis of modes of addressing i.e. direct, immediate and register addressing.

**Table 4.1**

**Data Transfer Group**

| Instruction | Op. Code | Addressing modes | No of T-state s | No of bytes of instr. | Flags affected | Operation                      |
|-------------|----------|------------------|-----------------|-----------------------|----------------|--------------------------------|
| LDA address | 3A       | Direct           | 13              | 3                     | None           | $[A] \leftarrow [M_{address}]$ |
| STA address | 32       | Direct           | 13              | 3                     | None           | $[M_{address}] \leftarrow [A]$ |
| MOV A, A    | 7F       | Register         | 4               | 1                     | None           | $[A] \leftarrow [A]$           |
| MOV A, B    | 78       | Register         | 4               | 1                     | None           | $[A] \leftarrow [B]$           |
| MOV A, C    | 79       | Register         | 4               | 1                     | None           | $[A] \leftarrow [C]$           |
| MOV A, D    | 7A       | Register         | 4               | 1                     | None           | $[A] \leftarrow [D]$           |
| MOV A, E    | 7B       | Register         | 4               | 1                     | None           | $[A] \leftarrow [E]$           |
| MOV A, H    | 7C       | Register         | 4               | 1                     | None           | $[A] \leftarrow [H]$           |
| MOV A, L    | 7D       | Register         | 4               | 1                     | None           | $[A] \leftarrow [L]$           |
| MOV B, A    | 47       | Register         | 4               | 1                     | None           | $[B] \leftarrow [A]$           |
| MOV B, B    | 40       | Register         | 4               | 1                     | None           | $[B] \leftarrow [B]$           |
| MOV B, C    | 41       | Register         | 4               | 1                     | None           | $[B] \leftarrow [C]$           |
| MOV B, D    | 42       | Register         | 4               | 1                     | None           | $[B] \leftarrow [D]$           |
| MOV B, E    | 43       | Register         | 4               | 1                     | None           | $[B] \leftarrow [E]$           |
| MOV B, H    | 44       | Register         | 4               | 1                     | None           | $[B] \leftarrow [H]$           |
| MOV B, L    | 45       | Register         | 4               | 1                     | None           | $[B] \leftarrow [L]$           |
| MOV C, A    | 4F       | Register         | 4               | 1                     | None           | $[C] \leftarrow [A]$           |
| MOV C, B    | 48       | Register         | 4               | 1                     | None           | $[C] \leftarrow [B]$           |
| MOV C, C    | 49       | Register         | 4               | 1                     | None           | $[C] \leftarrow [C]$           |
| MOV C, D    | 4A       | Register         | 4               | 1                     | None           | $[C] \leftarrow [D]$           |
| MOV C, E    | 4B       | Register         | 4               | 1                     | None           | $[C] \leftarrow [E]$           |
| MOV C, H    | 4C       | Register         | 4               | 1                     | None           | $[C] \leftarrow [H]$           |
| MOV C, L    | 4D       | Register         | 4               | 1                     | None           | $[C] \leftarrow [L]$           |
| MOV D, A    | 57       | Register         | 4               | 1                     | None           | $[D] \leftarrow [A]$           |
| MOV D, B    | 50       | Register         | 4               | 1                     | None           | $[D] \leftarrow [B]$           |

|          |    |                      |   |   |      |                            |
|----------|----|----------------------|---|---|------|----------------------------|
| MOV D, C | 51 | Register             | 4 | 1 | None | $[D] \leftarrow [C]$       |
| MOV D, D | 52 | Register             | 4 | 1 | None | $[D] \leftarrow [D]$       |
| MOV D, E | 53 | Register             | 4 | 1 | None | $[D] \leftarrow [E]$       |
| MOV D, H | 54 | Register             | 4 | 1 | None | $[D] \leftarrow [H]$       |
| MOV D, L | 55 | Register             | 4 | 1 | None | $[D] \leftarrow [L]$       |
| MOV E, A | 5F | Register             | 4 | 1 | None | $[E] \leftarrow [A]$       |
| MOV E, B | 58 | Register             | 4 | 1 | None | $[E] \leftarrow [B]$       |
| MOV E, C | 59 | Register             | 4 | 1 | None | $[E] \leftarrow [C]$       |
| MOV E, D | 5A | Register             | 4 | 1 | None | $[E] \leftarrow [D]$       |
| MOV E, E | 5B | Register             | 4 | 1 | None | $[E] \leftarrow [E]$       |
| MOV E, H | 5C | Register             | 4 | 1 | None | $[E] \leftarrow [H]$       |
| MOV E, L | 5D | Register             | 4 | 1 | None | $[E] \leftarrow [L]$       |
| MOV H, A | 67 | Register             | 4 | 1 | None | $[H] \leftarrow [A]$       |
| MOV H, B | 60 | Register             | 4 | 1 | None | $[H] \leftarrow [B]$       |
| MOV H, C | 61 | Register             | 4 | 1 | None | $[H] \leftarrow [C]$       |
| MOV H, D | 62 | Register             | 4 | 1 | None | $[H] \leftarrow [D]$       |
| MOV H, E | 63 | Register             | 4 | 1 | None | $[H] \leftarrow [E]$       |
| MOV H, H | 64 | Register             | 4 | 1 | None | $[H] \leftarrow [H]$       |
| MOV H, L | 65 | Register             | 4 | 1 | None | $[H] \leftarrow [L]$       |
| MOV L, A | 6F | Register             | 4 | 1 | None | $[L] \leftarrow [A]$       |
| MOV L, B | 68 | Register             | 4 | 1 | None | $[L] \leftarrow [B]$       |
| MOV L, C | 69 | Register             | 4 | 1 | None | $[L] \leftarrow [C]$       |
| MOV L, D | 6A | Register             | 4 | 1 | None | $[L] \leftarrow [D]$       |
| MOV L, E | 6B | Register             | 4 | 1 | None | $[L] \leftarrow [E]$       |
| MOV L, H | 6C | Register             | 4 | 1 | None | $[L] \leftarrow [H]$       |
| MOV L, L | 6D | Register             | 4 | 1 | None | $[L] \leftarrow [L]$       |
| MOV A, M | 7E | Register             | 7 | 1 | None | $[A] \leftarrow [M_{H-L}]$ |
| MOV B, M | 46 | Register             | 7 | 1 | None | $[A] \leftarrow [M_{H-L}]$ |
| MOV C, M | 4E | Register             | 7 | 1 | None | $[A] \leftarrow [M_{H-L}]$ |
| MOV D, M | 56 | Register             | 7 | 1 | None | $[A] \leftarrow [M_{H-L}]$ |
| MOV E, M | 5E | Register<br>indirect | 7 | 1 | None | $[A] \leftarrow [M_{H-L}]$ |
| MOV H, M | 66 | Register<br>indirect | 7 | 1 | None | $[A] \leftarrow [M_{H-L}]$ |
| MOV L, M | 6E | Register             | 7 | 1 | None | $[A] \leftarrow [M_{H-L}]$ |
| MOV M, A | 77 | Register<br>indirect | 7 | 1 | None | $[M_{H-L}] \leftarrow [A]$ |
| MOV M, B | 70 | Register             | 7 | 1 | None | $[M_{H-L}] \leftarrow [B]$ |

|             |    | Indirect          |    |   |      |                                    |
|-------------|----|-------------------|----|---|------|------------------------------------|
| MOV M, C    | 71 | Register indirect | 7  | 1 | None | $[M_{H-L}] \leftarrow [C]$         |
| MOV M, D    | 72 | Register indirect | 7  | 1 | None | $[M_{H-L}] \leftarrow [D]$         |
| MOV M, E    | 73 | Register indirect | 7  | 1 | None | $[M_{H-L}] \leftarrow [E]$         |
| MOV M, H    | 74 | Register indirect | 7  | 1 | None | $[M_{H-L}] \leftarrow [H]$         |
| MOV M, L    | 75 | Register indirect | 7  | 1 | None | $[M_{H-L}] \leftarrow [L]$         |
| MVI A, data | 3E | Immediate         | 7  | 2 | None | $[A] \leftarrow \text{data}$       |
| MVI B, data | 06 | Immediate         | 7  | 2 | None | $[B] \leftarrow \text{data}$       |
| MVI C, data | 0E | Immediate         | 7  | 2 | None | $[C] \leftarrow \text{data}$       |
| MVI D, data | 16 | Immediate         | 7  | 2 | None | $[D] \leftarrow \text{data}$       |
| MVI E, data | 1E | Immediate         | 7  | 2 | None | $[E] \leftarrow \text{data}$       |
| MVI H, data | 26 | Immediate         | 7  | 2 | None | $[H] \leftarrow \text{data}$       |
| MVI L, data | 2E | Immediate         | 7  | 2 | None | $[L] \leftarrow \text{data}$       |
| MVI M, data | 36 | Immediate         | 10 | 2 | None | $[M_{H-L}] \leftarrow \text{data}$ |

### (a) Direct Data Transfer Instructions

There is basically two direct data transfer instructions for SAP-III computers which are the same used in SAP-II computers. These are:

#### (i) LDA address

This is mnemonic for **Loads the accumulator direct**. It transfers the content stored in the addressed memory location (given by address) to accumulator.

$$[A] \leftarrow [M_{\text{address}}]$$

No flag is affected in this instruction. It is three byte instruction.

For example if 2AH data is stored in memory location 2500H before the execution of LDA 2500H instruction, then after the execution of this instruction, the data 2AH will be transferred to accumulator.

i.e.  $[A] \leftarrow 2A$

#### (ii) STA address

This is mnemonic for **Stores the accumulator direct**. It transfers the content stored in the accumulator to addressed memory location (given by address).

$$[M_{\text{address}}] \leftarrow [A]$$

No flag is affected in this instruction. It is also three byte instruction.

For example if 16H data is stored in the accumulator before the execution of STA 2100H instruction, then after the execution of this instruction, the data 16H will be transferred to the addressed memory location.

i.e.  $[M_{2100}] \leftarrow 16$

### (b) Register Data Transfer Instructions

These instructions transfer 8-bit data stored in one register to other register. The registers in SAP-III computer are A, B, C, D, E, H and L so the data stored in one of these registers may be transferred to other register. The format for this data transfer instruction is given as:

***MOV reg1, reg2      (Move Register)***

where *reg1* = A, B, C, D, E, H or L

*reg2* = A, B, C, D, E, H or L

This instruction copies (transfers) the content stored in *reg2* to *reg1*.

$[reg1] \leftarrow [reg2]$

Following are the possible combinations of MOV instruction.

|                 |                 |
|-----------------|-----------------|
| <i>MOV A, A</i> | <i>MOV B, A</i> |
|-----------------|-----------------|

|                 |                 |
|-----------------|-----------------|
| <i>MOV A, B</i> | <i>MOV B, B</i> |
|-----------------|-----------------|

|                 |                 |
|-----------------|-----------------|
| <i>MOV A, C</i> | <i>MOV B, C</i> |
|-----------------|-----------------|

|                 |                 |
|-----------------|-----------------|
| <i>MOV A, D</i> | <i>MOV B, D</i> |
|-----------------|-----------------|

|                 |                 |
|-----------------|-----------------|
| <i>MOV A, E</i> | <i>MOV B, E</i> |
|-----------------|-----------------|

|                 |                 |
|-----------------|-----------------|
| <i>MOV A, H</i> | <i>MOV B, H</i> |
|-----------------|-----------------|

|                 |                 |
|-----------------|-----------------|
| <i>MOV A, L</i> | <i>MOV B, L</i> |
|-----------------|-----------------|

|                 |                 |
|-----------------|-----------------|
| <i>MOV C, A</i> | <i>MOV D, A</i> |
|-----------------|-----------------|

|                 |                 |
|-----------------|-----------------|
| <i>MOV C, B</i> | <i>MOV D, B</i> |
|-----------------|-----------------|

|                 |                 |
|-----------------|-----------------|
| <i>MOV C, C</i> | <i>MOV D, C</i> |
|-----------------|-----------------|

|                 |                 |
|-----------------|-----------------|
| <i>MOV C, D</i> | <i>MOV D, D</i> |
|-----------------|-----------------|

|                 |                 |
|-----------------|-----------------|
| <i>MOV C, E</i> | <i>MOV D, E</i> |
|-----------------|-----------------|

|                 |                 |
|-----------------|-----------------|
| <i>MOV C, H</i> | <i>MOV D, H</i> |
|-----------------|-----------------|

|                 |                 |
|-----------------|-----------------|
| <i>MOV C, L</i> | <i>MOV D, L</i> |
|-----------------|-----------------|

|                 |                 |
|-----------------|-----------------|
| <i>MOV E, A</i> | <i>MOV H, A</i> |
|-----------------|-----------------|

|                 |                 |
|-----------------|-----------------|
| <i>MOV E, B</i> | <i>MOV H, B</i> |
|-----------------|-----------------|

|                 |                 |
|-----------------|-----------------|
| <i>MOV E, C</i> | <i>MOV H, C</i> |
|-----------------|-----------------|

|                 |                 |
|-----------------|-----------------|
| <i>MOV E, D</i> | <i>MOV H, D</i> |
|-----------------|-----------------|

|                 |                 |
|-----------------|-----------------|
| <i>MOV E, E</i> | <i>MOV H, E</i> |
|-----------------|-----------------|

|                 |                 |
|-----------------|-----------------|
| <i>MOV E, H</i> | <i>MOV H, H</i> |
|-----------------|-----------------|

|                 |                 |
|-----------------|-----------------|
| <i>MOV E, L</i> | <i>MOV H, L</i> |
|-----------------|-----------------|

|                 |  |
|-----------------|--|
| <i>MOV L, A</i> |  |
|-----------------|--|

|                 |  |
|-----------------|--|
| <i>MOV L, B</i> |  |
|-----------------|--|

|                 |  |
|-----------------|--|
| <i>MOV L, C</i> |  |
|-----------------|--|

|                 |  |
|-----------------|--|
| <i>MOV L, D</i> |  |
|-----------------|--|

|                 |  |
|-----------------|--|
| <i>MOV L, E</i> |  |
|-----------------|--|

|                 |  |
|-----------------|--|
| <i>MOV L, H</i> |  |
|-----------------|--|

|                 |  |
|-----------------|--|
| <i>MOV L, L</i> |  |
|-----------------|--|

These instructions are of one byte instruction and no flag is affected in these instructions.

For example if  $L = 23\text{ H}$  before the execution of instruction  $MOV C, L$  then after the execution of this instruction  $23\text{ H}$  data in register  $L$  will be transferred to register  $C$ .

i.e.  $[L] \leftarrow 23H$

### (c) Register Indirect Data Transfer Instructions

In SAP-III computers, there are some register indirect data transfer instructions in which H-L register pair acts like ‘data pointer’. The data pointer is represented by  $M$  which denotes the memory location whose address is given in the H-L register pair. It is basically represented by  $M_{HL}$ .

For example  $H\ L = 2500\text{ H}$  then  $M \Rightarrow M_{HL}$  represents the memory location whose address is  $2500\text{ H}$ . It indicates the memory location  $M_{2500}$ .

The instructions related to register immediate data transfer are given as:

#### (i) $MOV reg, M$ (Moves register from memory)

where  $reg = A, B, C, D, E, H$  or  $L$

This instruction is indirect read instruction. It moves / copies the data stored in memory location whose address is given in H-L register pair, to the given register.

i.e.

$$[reg] \leftarrow [M_{HL}]$$

The possible combinations of this instruction are:

$MOV A, M$

$MOV B, M$

$MOV C, M$

$MOV D, M$

$MOV E, M$

$MOV H, M$

$MOV L, M$

For example consider  $12H$  data is stored in the memory location whose address is given in H-L register pair (i.e.  $H-L = 2500H$ ). After the execution of the instruction  $MOV B, M$  the data  $12H$  will be stored in  $B$  register.

$$[B] \leftarrow 12$$

These are one byte instruction and no flag is affected in these instructions.

#### (ii) $MOV M, reg$ (Moves memory from register)

where  $reg = A, B, C, D, E, H$  or  $L$

This instruction moves / copies the data in the given register to the memory location addressed by H-L register pair.

i.e.

$$[M_{HL}] \leftarrow [reg]$$

It performs reverse work of  $MOV reg, M$  instruction. The possible combinations of this instruction are:

$MOV M, A$

$MOV M, B$

$MOV M, C$

$MOV M, D$

$MOV M, E$

$MOV M, H$



|          |    |                   |    |   |     |                                       |
|----------|----|-------------------|----|---|-----|---------------------------------------|
| ADD H    | 84 | Register          | 4  | 1 | All | $[A] \leftarrow [A] + [H]$            |
| ADD L    | 85 | Register          | 4  | 1 | All | $[A] \leftarrow [A] + [L]$            |
| ADD M    | 86 | Register indirect | 7  | 1 | All | $[A] \leftarrow [A] + [M_{H-L}]$      |
| ADI data | C6 | Immediate         | 7  | 2 | All | $[A] \leftarrow [A] + data$           |
| ADC A    | 8F | Register          | 4  | 1 | All | $[A] \leftarrow [A] + [A] + CY$       |
| ADC B    | 88 | Register          | 4  | 1 | All | $[A] \leftarrow [A] + [B] + CY$       |
| ADC C    | 89 | Register          | 4  | 1 | All | $[A] \leftarrow [A] + [C] + CY$       |
| ADC D    | 8A | Register          | 4  | 1 | All | $[A] \leftarrow [A] + [D] + CY$       |
| ADC E    | 8B | Register          | 4  | 1 | All | $[A] \leftarrow [A] + [E] + CY$       |
| ADC H    | 8C | Register          | 4  | 1 | All | $[A] \leftarrow [A] + [H] + CY$       |
| ADC L    | 8D | Register          | 4  | 1 | All | $[A] \leftarrow [A] + [L] + CY$       |
| ADC M    | 8E | Register indirect | 7  | 1 | All | $[A] \leftarrow [A] + [M_{H-L}] + CY$ |
| ACI data | CE | Immediate         | 7  | 2 | All | $[A] \leftarrow [A] + data + CY$      |
| DAD B    | 09 | Register          | 10 | 1 | CY  | $BC \leftarrow BC + BC$               |
| DAD D    | 19 | Register          | 10 | 1 | CY  | $DE \leftarrow DE + DE$               |
| DAD H    | 29 | Register          | 10 | 1 | CY  | $HL \leftarrow HL + HL$               |
| SUB A    | 97 | Register          | 4  | 1 | All | $[A] \leftarrow [A] - [A]$            |
| SUB B    | 90 | Register          | 4  | 1 | All | $[A] \leftarrow [A] - [B]$            |
| SUB C    | 91 | Register          | 4  | 1 | All | $[A] \leftarrow [A] - [C]$            |
| SUB D    | 92 | Register          | 4  | 1 | All | $[A] \leftarrow [A] - [D]$            |
| SUB E    | 93 | Register          | 4  | 1 | All | $[A] \leftarrow [A] - [E]$            |
| SUB H    | 94 | Register          | 4  | 1 | All | $[A] \leftarrow [A] - [H]$            |
| SUB L    | 95 | Register          | 4  | 1 | All | $[A] \leftarrow [A] - [L]$            |
| SUB M    | 96 | Register indirect | 7  | 1 | All | $[A] \leftarrow [A] - [M_{H-L}]$      |
| SUI data | D6 | Immediate         | 7  | 2 | All | $[A] \leftarrow [A] - data$           |
| SBB A    | 9F | Register          | 4  | 1 | All | $[A] \leftarrow [A] - [A] - CY$       |
| SBB B    | 98 | Register          | 4  | 1 | All | $[A] \leftarrow [A] - [B] - CY$       |
| SBB C    | 99 | Register          | 4  | 1 | All | $[A] \leftarrow [A] - [C] - CY$       |
| SBB D    | 9A | Register          | 4  | 1 | All | $[A] \leftarrow [A] - [D] - CY$       |
| SBB E    | 9B | Register          | 4  | 1 | All | $[A] \leftarrow [A] - [E] - CY$       |
| SBB H    | 9C | Register          | 4  | 1 | All | $[A] \leftarrow [A] - [H] - CY$       |
| SBB L    | 9D | Register          | 4  | 1 | All | $[A] \leftarrow [A] - [L] - CY$       |
| SBB M    | 9E | Register indirect | 7  | 1 | All | $[A] \leftarrow [A] - [M_{H-L}] - CY$ |
| SBI data | DE | Immediate         | 7  | 2 | All | $[A] \leftarrow [A] - data - CY$      |

|       |    |                      |    |   |                   |                                    |
|-------|----|----------------------|----|---|-------------------|------------------------------------|
| INR A | 3C | Register             | 4  | 1 | All but<br>not CY | $[A] \leftarrow [A]+1$             |
| INR B | 04 | Register             | 4  | 1 | All but<br>not CY | $[B] \leftarrow [B]+1$             |
| INR C | 0C | Register             | 4  | 1 | All but<br>not CY | $[C] \leftarrow [C]+1$             |
| INR D | 14 | Register             | 4  | 1 | All but<br>not CY | $[D] \leftarrow [D]+1$             |
| INR E | 1C | Register             | 4  | 1 | All but<br>not CY | $[E] \leftarrow [E]+1$             |
| INR H | 24 | Register             | 4  | 1 | All but<br>not CY | $[H] \leftarrow [H]+1$             |
| INR L | 2C | Register             | 4  | 1 | All but<br>not CY | $[L] \leftarrow [L]+1$             |
| INR M | 34 | Register<br>indirect | 10 | 1 | All but<br>not CY | $[M_{H-L}] \leftarrow [M_{H-L}]+1$ |
| INX B | 03 | Register             | 6  | 1 | None              | $BC \leftarrow BC+1$               |
| INX D | 13 | Register             | 6  | 1 | None              | $DE \leftarrow DE+1$               |
| INX H | 23 | Register             | 6  | 1 | None              | $HL \leftarrow HL+1$               |
| DCR A | 3D | Register             | 4  | 1 | All but<br>not CY | $[A] \leftarrow [A]-1$             |
| DCR B | 05 | Register             | 4  | 1 | All but<br>not CY | $[B] \leftarrow [B]-1$             |
| DCR C | 0D | Register             | 4  | 1 | All but<br>not CY | $[C] \leftarrow [C]-1$             |
| DCR D | 15 | Register             | 4  | 1 | All but<br>not CY | $[D] \leftarrow [D]-1$             |
| DCR E | 1D | Register             | 4  | 1 | All but<br>not CY | $[E] \leftarrow [E]-1$             |
| DCR H | 25 | Register             | 4  | 1 | All but<br>not CY | $[H] \leftarrow [H]-1$             |
| DCR L | 2D | Register             | 4  | 1 | All but<br>not CY | $[L] \leftarrow [L]-1$             |
| DCR M | 35 | Register<br>indirect | 7  | 1 | All but<br>not CY | $[M_{H-L}] \leftarrow [M_{H-L}]-1$ |
| DCX B | 0B | Register             | 6  | 1 | None              | $BC \leftarrow BC-1$               |
| DCX D | 1B | Register             | 6  | 1 | None              | $DE \leftarrow DE-1$               |
| DCX H | 2B | Register             | 6  | 1 | None              | $HL \leftarrow HL-1$               |
| RLC   | 07 | ---                  | 4  | 1 | CY                | Rotate left through<br>carry       |
| RAL   | 17 | ---                  | 4  | 1 | CY                | Rotate all left                    |
| RRC   | 0F | ---                  | 4  | 1 | CY                | Rotate right through<br>carry      |
| RAR   | 1F | ---                  | 4  | 1 | CY                | Rotate all right                   |

**(a) Add instructions**

Following are the add instructions:

(i) *ADD reg* (Add register)

where  $reg = A, B, C, D, E, H$  or  $L$ .

It adds the content stored in given register with the accumulator. The result of this addition is stored in accumulator.

$[A] \leftarrow [A] + [reg]$

All the flags are affected with this ‘*ADD reg*’ instruction.

The possible combinations of this instruction are as given below:

*ADD A  
ADD B  
ADD C  
ADD D  
ADD E  
ADD H  
ADD L*

Suppose before the execution of the instruction *ADD E*

$$A = 10101111$$

$$E = 10110101$$

$$CY = 0, \quad S = 1, \quad Z = 0 \quad \text{and} \quad P = 1$$

then after the execution of the instruction *ADD E* we get the following result:

$$\begin{array}{l} A = \\ E = \end{array} \quad \begin{array}{cccccccc} 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \end{array}$$

$$A = \begin{matrix} & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ CY & & & & & & & & & \end{matrix}$$

The flags will be affected as:

$$CY \equiv 1, \quad S \equiv 0, \quad Z \equiv 0 \quad \text{and} \quad P \equiv 0$$

$CY \equiv 1$  Since there is a final carry.

$S \equiv 0$  Since Acc content is positive as MSB is zero

$Z \equiv 0$  Since Acc content is non zero

$P \equiv 0$  Since Acc content has odd parity

### (ii) ADD M (Add Memory)

This instruction is one byte instruction and adds the content of memory location whose address is given in H-L register pair with the accumulator and the answer is stored in accumulator.

$$[A] \leftarrow [A] + [M_{\text{err}}]$$

In this instruction too all flags are affected. This instruction is similar to *ADD reg.*

For example let  $A = 40\text{ H}$   $H = 21\text{ H}$   $L = 00\text{ H}$

and  $M_{\text{max}} \equiv 3AH$

Then after execution of the instruction  $ADD M$  will produce the result:

$$A = 7AH$$

All flag will, however, be affected as per the instruction.

(iii) *ADJ data* (Adds immediately the data)

It immediately adds the given data with the accumulator and the answer will be stored in Accumulator.

$$[A] \leftarrow [A] + data$$

This is a two-byte instruction. All flags will be affected with this instruction similar to *ADD reg* instruction.

#### (iv) **ADC instructions**

The format for ADC instruction is

$$ADC \ reg \quad (\text{Add with carry})$$

where *reg* = A, B, C, D, E, H or L.

It adds the content stored in given register and content of CY flag with the content of accumulator. The result of this addition is stored in accumulator.

$$[A] \leftarrow [A] + [reg] + [CY]$$

All the flags are affected with this 'ADC reg' instruction.

The possible combinations of this instruction are as given below:

$$ADC \ A$$

$$ADC \ B$$

$$ADC \ C$$

$$ADC \ D$$

$$ADC \ E$$

$$ADC \ H$$

$$ADC \ L$$

Suppose before the execution of the instruction *ADC D*

$$A = 10101001$$

$$D = 10111101$$

$$CY = 1, S = 1, Z = 0 \text{ and } P = 1$$

then after the execution of the instruction *ADD D* we get the following result:

$$\begin{array}{r} CY = & & & & & & 1 \\ A = & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ D = & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ \hline A = & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ & CY & & & & & & & & \end{array}$$

All flags will be affected.

#### (v) **ADC M** (Add Memory with Carry)

This instruction is one byte instruction and adds the content of memory location whose address is given in H-L register pair to the accumulator with carry and the answer is stored in accumulator.

$$[A] \leftarrow [A] + [M_{HL}] + CY$$

In this instruction too all flags are affected. This instruction is similar to *ADD reg*.

For example let  $A = 50 \text{ H}$   $CY = 1$   $H = 25 \text{ H}$   $L = 00 \text{ H}$

and  $M_{2500} = 3BH$

Then after execution of the instruction *ADC M* will produce the result:

$$A = 8C \text{ H}$$

All flag will, however, be affected as per the result.





$$\begin{array}{l} H = \begin{array}{ccccccccc} 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \end{array} \\ CY = \begin{array}{c} \\ \\ \\ \\ \\ \\ \end{array} \end{array}$$


---


$$\begin{array}{l} A = \begin{array}{ccccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \end{array} \\ CY \end{array}$$

All flags will be affected as per the accumulator contents.

Similar to *SBB reg*, this instruction subtracts the content already stored in memory location addressed by H-L register pair with borrow bit (carry flag) from the accumulator contents and the answer is stored in accumulator.

$$[A] \leftarrow [A] - [M_{HL}] - [CY]$$

All flags will be affected as per the accumulator contents.

*SBI data* instruction subtracts the given data with borrow bit (carry flag) from the accumulator contents and it stores the answer in accumulator.

$[A] \leftarrow [A] - data - [CY]$

All flags will be affected as per the accumulator contents.

### (c) Increment Instructions

The increment instructions used in SAP-III computer are given below:

where  $reg = A, B, C, D, E, H$  or  $L$ .

It increments the contents of given register by one and answer is stored in the given register.

$[reg] \leftarrow [reg] + 1$

The possible combinations of this instruction are:

INR A

INR B

INR C

INR D

INR E

*INR H*

INR L

These instructions do not affect the CY flag but affect all other flags.

For example if  $Z = 0$ ,  $S = 1$ , and  $CY = 0$  and contents of register is

C = 1 1 1 1 1 1 1 1 , then after the execution of the instruction INR C we have:

$C = 00000000$   
 $Z = 1, S = 0$ , and  $CY = 0$

It increments the contents of memory location addressed by H-L register pair by 1 and stores the answer in the addressed memory location.

$$[M_{HL}] \leftarrow [M_{HL}] + 1$$

Similar to *INR reg*, all flags except carry flag will be affected after the execution of this instruction.

(iii) *INX rp* (Increment register pair)



The possible combinations of this instruction are:

*DCX B*

*DCX D*

*DCX H*

No flag will be affected with the execution of this instruction like INX rp.

For example if      D = 23 H and E = 00 H

After the execution of the instruction DCX D, we have the contents in D-E pair as:      D = 22 H and E = FF H

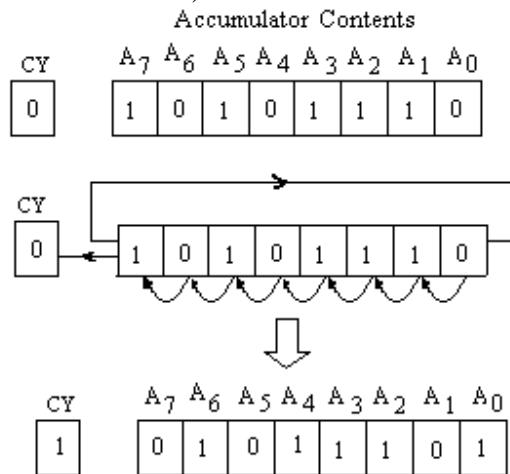
#### (e) Rotate Instructions

In rotate instructions, the accumulator contents are shifted either left or right. In some instructions shifting may be through CY flag or without CY flag. Following are the rotate instructions.

##### (i) RLC

##### (Rotate Accumulator Left)

In this instruction, the bits of the accumulator contents are shifted or rotated left. The LSB of the accumulator is changed as MSB (before the execution). The CY flag is modified as MSB (before the execution).



**Fig. 4.4**

For example  $A = AE\text{ H}$  and  $CY = 0$ , before the execution of the instruction RLC. After the execution of the instruction MSB is saved in CY flag and also in the LSB of the accumulator. The other bits are shifted left as shown in figure 4.4.

$$\text{i.e. } [A_{n+1}] \leftarrow [A_n]$$

$$[A_0] \leftarrow [A_7] \quad \text{also} \quad [CY] \leftarrow [A_7]$$

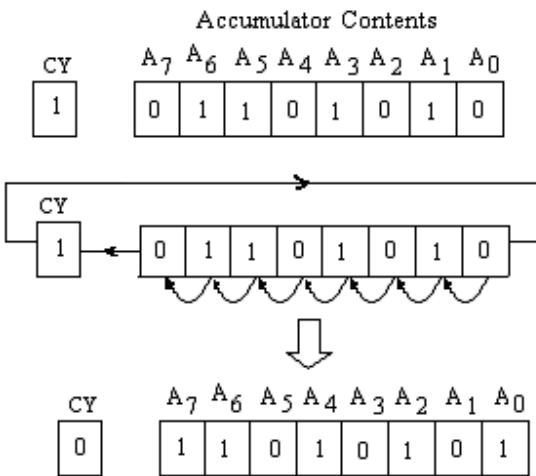
Only carry flag CY will be affected in this instruction and all other flags will be unaffected.

##### (ii) RAL

##### (Rotate Accumulator Left Through Carry)

In this instruction, the bits of the accumulator contents will be shifted / rotated left through carry. The content of carry flag CY will be stored in LSB of the accumulator and MSB of the accumulator will be stored in CY flag. All other bits of the accumulator will be shifted to the left.

For example if  $A = 6A\text{ H}$  and  $CY = 1$  before the execution of RAL instruction, then after the execution of this instruction the accumulator contents will be shifted as shown in figure 4.5.



**Fig. 4.5**

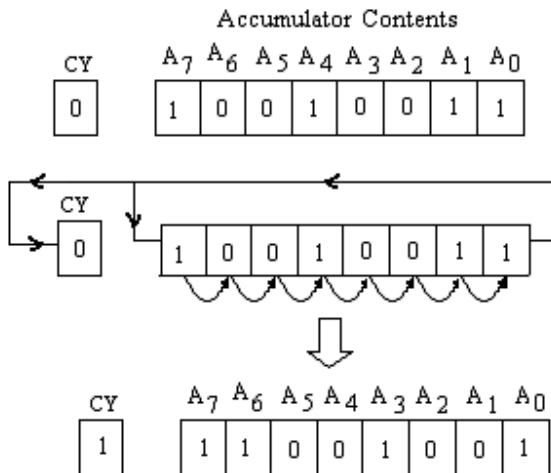
In this instruction too only carry flag will be affected.

**(iii) RRC**

**(Rotate Accumulator Right)**

In this instruction, all the bits of accumulator are shifted or rotated right. The MSB of the accumulator is changed as LSB (before the execution). The CY flag is modified as MSB (before the execution).

For example  $A = 93 \text{ H}$  and  $CY = 0$ , before the execution of the instruction RRC. After the execution of this instruction LSB is saved in CY flag and also in the MSB of the accumulator. The other bits are shifted left as shown in figure 4.6.



**Fig. 4.6**

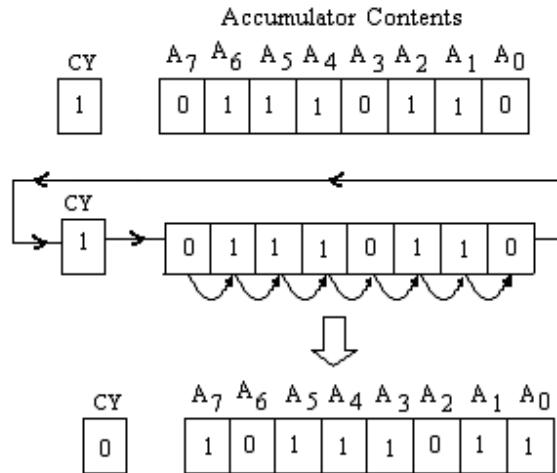
In this instruction only carry flag CY will be affected and all other flags will be unaffected.

**(iv)RAR**

**(Rotate Accumulator Right Through Carry)**

In this rotate instruction, all the bits of the accumulator contents will be shifted / rotated right through carry. The content of carry flag CY will be stored in MSB of the accumulator and LSB of the accumulator will be stored in CY flag; and all other bits of the accumulator will be shifted to the right.

For example if  $A = 76\text{ H}$  and  $CY = 1$  before the execution of RAR instruction, then after the execution of this instruction the accumulator contents will be shifted as shown in figure 4.7.



**Fig. 4.7**

In this instruction too only carry flag will be affected.

#### 4.2.3 Logic Transfer Group

The logic group of instructions operates on registers, memory and conditional flags. This group of instructions contains ANDing, ORing, XORing, complementing and comparing of data (table 4.3).

**Table 4.3**

**Logic Transfer Group**

| Instruction | Op. Code | Addressing modes  | No of T-states | No of bytes of instr. | Flags affected | Operation                          |
|-------------|----------|-------------------|----------------|-----------------------|----------------|------------------------------------|
| ANA A       | A7       | Register          | 4              | 1                     | All (CY=0)     | $[A] \leftarrow [A].AND.[A]$       |
| ANA B       | A0       | Register          | 4              | 1                     | All (CY=0)     | $[A] \leftarrow [A].AND.[B]$       |
| ANA C       | A1       | Register          | 4              | 1                     | All (CY=0)     | $[A] \leftarrow [A].AND.[C]$       |
| ANA D       | A2       | Register          | 4              | 1                     | All (CY=0)     | $[A] \leftarrow [A].AND.[D]$       |
| ANA E       | A3       | Register          | 4              | 1                     | All (CY=0)     | $[A] \leftarrow [A].AND.[E]$       |
| ANA H       | A4       | Register          | 4              | 1                     | All (CY=0)     | $[A] \leftarrow [A].AND.[H]$       |
| ANA L       | A5       | Register          | 4              | 1                     | All (CY=0)     | $[A] \leftarrow [A].AND.[L]$       |
| ANA M       | A6       | Register Indirect | 7              | 1                     | All (CY=0)     | $[A] \leftarrow [A].AND.[M_{H-L}]$ |
| ANI data    | E6       | Immediate         | 7              | 2                     | All (CY=0)     | $[A] \leftarrow [A].AND.data$      |

|          |    |                      |   |   |               |                                       |
|----------|----|----------------------|---|---|---------------|---------------------------------------|
| ORA A    | B7 | Register             | 4 | 1 | All<br>(CY=0) | $[A] \leftarrow [A].OR.[A]$           |
| ORA B    | B0 | Register             | 4 | 1 | All<br>(CY=0) | $[A] \leftarrow [A].OR.[B]$           |
| ORA C    | B1 | Register             | 4 | 1 | All<br>(CY=0) | $[A] \leftarrow [A].OR.[C]$           |
| ORA D    | B2 | Register             | 4 | 1 | All<br>(CY=0) | $[A] \leftarrow [A].OR.[D]$           |
| ORA E    | B3 | Register             | 4 | 1 | All<br>(CY=0) | $[A] \leftarrow [A].OR.[E]$           |
| ORA H    | B4 | Register             | 4 | 1 | All<br>(CY=0) | $[A] \leftarrow [A].OR.[H]$           |
| ORA L    | B5 | Register             | 4 | 1 | All           | $[A] \leftarrow [A].OR.[L]$           |
| ORA M    | B6 | Register<br>Indirect | 7 | 1 | All<br>(CY=0) | $[A] \leftarrow [A].OR.[M_{H-L}]$     |
| ORI data | F6 | Immediate            | 7 | 2 | All<br>(CY=0) | $[A] \leftarrow [A].OR.data$          |
| XRA A    | AF | Register             | 4 | 1 | All<br>(CY=0) | $[A] \leftarrow [A] \oplus [A]$       |
| XRA B    | A8 | Register             | 4 | 1 | All<br>(CY=0) | $[A] \leftarrow [A] \oplus [B]$       |
| XRA C    | A9 | Register             | 4 | 1 | All<br>(CY=0) | $[A] \leftarrow [A] \oplus [C]$       |
| XRA D    | AA | Register             | 4 | 1 | All<br>(CY=0) | $[A] \leftarrow [A] \oplus [D]$       |
| XRA E    | AB | Register             | 4 | 1 | All<br>(CY=0) | $[A] \leftarrow [A] \oplus [E]$       |
| XRA H    | AC | Register             | 4 | 1 | All<br>(CY=0) | $[A] \leftarrow [A] \oplus [H]$       |
| XRA L    | AD | Register             | 4 | 1 | All<br>(CY=0) | $[A] \leftarrow [A] \oplus [L]$       |
| XRA M    | AE | Register<br>Indirect | 7 | 1 | All<br>(CY=0) | $[A] \leftarrow [A] \oplus [M_{H-L}]$ |
| XRI data | EE | Immediate            | 7 | 2 | All<br>(CY=0) | $[A] \leftarrow [A] \oplus data$      |
| CMP A    | BF | Register             | 4 | 1 | All           |                                       |
| CMP B    | B8 | Register             | 4 | 1 | All           |                                       |
| CMP C    | B9 | Register             | 4 | 1 | All           |                                       |
| CMP D    | BA | Register             | 4 | 1 | All           |                                       |
| CMP E    | BB | Register             | 4 | 1 | All           |                                       |
| CMP H    | BC | Register             | 4 | 1 | All           |                                       |
| CMP L    | BD | Register             | 4 | 1 | All           |                                       |
| CMP M    | BE | Register<br>Indirect | 7 | 1 | All           |                                       |
| CPI data | FE | Immediate            | 7 | 2 | All           |                                       |

|     |    |     |   |   |      |                               |
|-----|----|-----|---|---|------|-------------------------------|
| CMA | 2F | --- | 4 | 1 | None | $[A] \leftarrow [\bar{A}]$    |
| CMC | 3F | --- | 4 | 1 | CY   | $CY \leftarrow \overline{CY}$ |
| STC | 37 | --- | 4 | 1 | CY   | $CY \leftarrow 1$             |

The details of these instructions are given below:

### (a) Logical Instructions



In this instruction each bit of the given register contents are ANDed with each bit of the accumulator contents (bit by bit). The result is saved in the accumulator. It does not affect the contents of the given register.

$[A] \leftarrow [A].AND.[reg]$

The possible combinations of this instruction are:

*AND A  
ANA B  
ANA C  
ANA D  
ANA E  
ANA H  
ANA L*

The *ANA reg* instruction clears (resets) the CY flag and all other flags are modified according to the data conditions of the result. This instruction is one byte instruction.

Let  $A = 73H$ ,  $C = C3H$ ,  $CY = 1$   
 before the execution of the instruction  $ANA\ C$ . Then after the instruction is executed we get:

$$\begin{array}{ccccccccc}
 A = & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\
 C = & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
 \hline
 A \equiv & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1
 \end{array} \quad CY = 1 \quad CY \equiv 0$$

- (ii) *ANA M* (AND Memory)

In this instruction each bit of the data stored in the memory location addressed by H-L register pair are ANDed with each bit of the accumulator contents (bit by bit). The result is stored in the accumulator like *ANA reg*.

$$[A] \leftarrow [A].AND.[M_{H-I}]$$

Similar to ANA reg instruction data of the memory location addressed by H-L register pair will not be affected; the carry flag will, however, be reset after the execution of this instruction and other flags will be affected as per the data in the accumulator.

$$\text{Let } A = 19 \text{ H} \quad H = 25 \text{ H} \quad L = 00 \text{ H} \\ M_{2500} = 37 \text{ H} \quad \text{and} \quad CY = 1$$

before the execution of the instruction  $ANA\ M$ . Then after the instruction is executed we get:

$$A = \begin{array}{ccccccccc} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{array} \quad CY = 1$$

$$M_{2500} = \begin{array}{ccccccccc} 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{array}$$

$$\overline{A = \begin{array}{ccccccccc} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}} \quad CY = 0$$

**(iii) ANI data (AND Immediate)**

In this instruction each bit of the given data is immediately ANDed with each bit of the accumulator contents (bit by bit). The result is stored in the accumulator. The difference between *ANA reg* and *ANI data* instruction is that in *ANA reg* the data given in the register whereas in the *ANI data* instruction, the data is given with the instruction itself.

$$[A] \leftarrow [A] AND.data$$

The carry flag will be reset after the execution of this instruction and other flags will be affected as per the result.

For example, if  $A = AB\ H$  and  $CY = 1$  before the execution of  $ANI\ 06\ H$ , then after the execution of this instruction we have:

$$A = \begin{array}{ccccccccc} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{array} \quad CY = 1$$

$$data = \begin{array}{ccccccccc} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}$$

$$\overline{A = \begin{array}{ccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}} \quad CY = 0$$

**(iv) ORA reg (OR register)**

where  $reg = A, B, C, D, E, H$  or  $L$ .

In this instruction each bit of the given register contents are ORed with each bit of the accumulator contents (bit by bit). The result is saved in the accumulator. It does not affect the contents of the given register.

$$[A] \leftarrow [A] OR.[reg]$$

The possible combinations of this instruction are:

- ORA A*
- ORA B*
- ORA C*
- ORA D*
- ORA E*
- ORA H*
- ORA L*

The *ORA reg* instruction clears (resets) the CY flag and all other flags are modified according to the data conditions of the result. This instruction is one byte instruction.

Let  $A = 73\ H$      $C = C3\ H$      $CY = 1$   
before the execution of the instruction *ORA B*. Then after the instruction is executed we get:

$$A = \begin{array}{ccccccccc} 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \end{array} \quad CY = 1$$

$$B = \begin{array}{ccccccccc} 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{array}$$

$$\overline{A = \begin{array}{ccccccccc} 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \end{array}} \quad CY = 0$$

**(v) ORA M****(OR Memory)**

In this instruction each bit of the data stored in the memory location addressed by H-L register pair are ORed with each bit of the accumulator contents (bit by bit). The result is stored in the accumulator.

$$[A] \leftarrow [A] OR [M_{H-L}]$$

Similar to *ORA reg* instruction data of the memory location addressed by H-L register pair will not be affected; the carry flag will however be reset after the execution of this instruction and other flags will be affected as per the data in the accumulator.

Let       $A = 19\text{ H}$        $H = 21\text{ H}$        $L = 00\text{ H}$

$M_{2100} = 37\text{ H}$       and       $CY = 1$

before the execution of the instruction *ORA M*. Then after the instruction is executed we get:

$$A = \begin{array}{ccccccccc} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{array} \quad CY = 1$$

$$M_{2100} = \begin{array}{ccccccccc} 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{array}$$

$$\overline{A = \begin{array}{ccccccccc} 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}} \quad CY = 0$$

**(vi) ORI data****(OR Immediate)**

In this instruction each bit of the given data is immediately ORed with each bit of the accumulator contents (bit by bit). The result is stored in the accumulator.

$$[A] \leftarrow [A] OR.data$$

The carry flag will be reset after the execution of this instruction and other flags will be affected as per the result.

For example, if  $A = AB\text{ H}$  and  $CY = 1$  before the execution of ANI 16 H, then after the execution of this instruction we have:

$$A = \begin{array}{ccccccccc} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{array} \quad CY = 1$$

$$data = \begin{array}{ccccccccc} 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

$$\overline{A = \begin{array}{ccccccccc} 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}} \quad CY = 0$$

**(vii) XRA reg****(Exclusive OR register)**

where  $reg = A, B, C, D, E, H$  or  $L$ .

In this instruction each bit of the given register contents are XORed with each bit of the accumulator contents (bit by bit). The result is saved in the accumulator. It does not affect the contents of the given register.

$$[A] \leftarrow [A] XOR.[reg]$$

The possible combinations of this instruction are:

*XRA A*

*XRA B*

*XRA C*

*XRA D*

*XRA E*

*XRA H*

*XRA L*

The *XRA reg* instruction clears (resets) the CY flag and all other flags are modified according to the data conditions of the result. This instruction is one byte instruction.

Let  $A = 73\text{H}$      $D = C3\text{H}$      $CY = 1$   
 before the execution of the instruction  $XRA D$ . Then after the instruction is executed we get:

$$\begin{array}{ccccccccc}
 A = & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\
 & \hline
 D = & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
 & \hline
 A = & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0
 \end{array} \quad CY = 1 \quad CY = 0$$

(viii) *XRA M* (Exclusive QR Memory)

$$[A] \leftarrow [A].XOR.[M_{H-L}]$$

The carry flag will be reset after the execution of this instruction and other flags will be affected as per the data in the accumulator.

$$\text{Let } A = 19 \text{ H} \quad H = 22 \text{ H} \quad L = 00 \text{ H} \\ M_{2200} = 37 \text{ H} \quad \text{and} \quad CY = 1$$

before the execution of the instruction *ORA M*. Then after the instruction is executed we get:

$$\begin{array}{ccccccccc}
 A = & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
 \\ 
 200 = & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\
 \\ 
 \hline
 A \equiv & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
 & & & & & & & & \\
 CY \equiv 0 & & & & & & & &
 \end{array}$$

In this instruction each bit of the given data is immediately XORed with each bit of the accumulator contents (bit by bit). The result is stored in the accumulator.

$[A] \leftarrow [A].XOR.data$

The carry flag will be reset after the execution of this instruction and other flags will be affected as per the result.

For example, if  $A = AB$  H and  $CY = 1$  before the execution of ANI 12 H, then after the execution of this instruction we have:

$$\begin{array}{ccccccccc}
 A = & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
 \hline
 data = & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
 \hline
 A = & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & CY = 0
 \end{array}$$

(x) CMA (Complement Accumulator)

This is one byte implied addressing instruction as no operand is required with the instruction. The execution of this instruction inverts each bit of the accumulator contents and the result is saved in the accumulator. Basically it produces 1's complement of the accumulator contents. No flag is affected with this instruction.

$$[A] \leftarrow [\bar{A}]$$

For example, if  $A = 0B\text{ H}$  before the execution of *CMA* instruction, then after the execution of this instruction we have:

$$\begin{array}{r}
 A = \quad 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \\
 = \quad 1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0 \\
 \hline
 A = \quad 1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0
 \end{array} \quad (F4)$$

**(b) Compare Instructions**

(i) *CMP reg* (Compare Register)

The contents of the given register are compared with the accumulator contents. In fact the contents of the register are subtracted from the contents of accumulator and the accumulator contents remain unchanged. However, as a result of the subtraction the flags are modified as per the result.

The zero flag Z is set if  $[A]=[reg]$  otherwise reset. Similarly, carry flag CY is set if  $[A]<[reg]$  otherwise reset.

The possible combinations of *CMP reg* are given below:

*CMP A*  
*CMP B*  
*CMP C*  
*CMP D*  
*CMP E*  
*CMP H*  
*CMP L*

To illustrate the above instruction let us consider the following assembly language program.

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> |
|--------------|------------------|----------------|
|              | MVI B,           | 00 H           |
|              | MVI A,           | 09 H           |
| LOOP         | INR B            |                |
|              | CMP B            |                |
|              | JNZ              | LOOP           |
|              | ----             |                |
|              | ----             |                |
|              | ----             |                |
|              | ----             |                |

In this program before the loop A = 09 H and B = 00 H. The value of B will increase by each go in the loop and each time it is compared with accumulator contents. Till the contents of B register are not equal to the contents of Accumulator, the computer will execute the instructions inside the loop. When the value of B register becomes equal to 09 H, the zero flag will be set and it will allow coming out of the loop to execute the next instructions.

(ii) **CMP M** (Compare Memory)

It is similar to *CMP reg* instruction with the difference that the contents of the addressed memory location will be compared by the accumulator contents. The flags will be modified as per the result.

**(iii) CPI data**

**(Compare immediate with data)**

It is similar to CMP instruction with the difference that the data is directly given with the instruction. In this instruction the given data is compared with the accumulator contents. The flags will be modified as per the result.

**(c) Miscellaneous Logical Instructions**

**(i) CMC**

**(Complement the carry)**

This instruction complements the carry flag.

$$CY \leftarrow \overline{CY}$$

If CY = 1 before the execution of CMC instruction, the carry flag will be reset (CY = 0) after the execution of this instruction. Similarly, If CY = 0 before the execution of CMC instruction, the carry flag will be set (CY = 1) after the execution of this instruction.

In this only carry flag will be affected and all other flags will not be affected.

**(ii) STC**

**(Set the carry)**

It sets the carry flag.

$$CY \leftarrow 1$$

The carry flag will be set (CY = 1) irrespective of the carry flag is set or reset before the execution of this instruction STC.

Only carry flag gets affected with this instruction.

#### 4.2.4 Branch Group

This group of instructions changes the normal sequence of the program. The branch instructions include JUMP, CALL and RETURN instructions, which are further sub-divided as unconditional and conditional JUMP, CALL and RETURN instructions.

**Table 4.4**

**Branch Group**

| Instruction | Op. Code | Addressing modes | No of T-states | No of bytes of instr. | Flags affected | Operation                         |
|-------------|----------|------------------|----------------|-----------------------|----------------|-----------------------------------|
| JMP addr    | C3       | Immediate        | 10             | 3                     | None           | $[PC] \leftarrow [addr]$          |
| JNZ addr    | C2       | Immediate        | 10/7           | 3                     | None           | $[PC] \leftarrow [addr]$ if Z=0.  |
| JZ addr     | CA       | Immediate        | 10/7           | 3                     | None           | $[PC] \leftarrow [addr]$ if Z=1.  |
| JNC addr    | D2       | Immediate        | 10/7           | 3                     | None           | $[PC] \leftarrow [addr]$ if CY=0. |
| JC addr     | DA       | Immediate        | 10/7           | 3                     | None           | $[PC] \leftarrow [addr]$ if CY=1. |
| JM addr     | FA       | Immediate        | 10/7           | 3                     | None           | $[PC] \leftarrow [addr]$ if S=1.  |
| JP addr     | F2       | Immediate        | 10/7           | 3                     | None           | $[PC] \leftarrow [addr]$ if S=0.  |
| JPO addr    | E2       | Immediate        | 10/7           | 3                     | None           | $[PC] \leftarrow [addr]$ if P=0.  |
| JPE addr    | EA       | Immediate        | 10/7           | 3                     | None           | $[PC] \leftarrow [addr]$ if P=1.  |
| CALL addr   | CD       | Immediate        | 18             | 3                     | None           | Calls subroutine program.         |

|          |    |                   |      |   |      |                                   |
|----------|----|-------------------|------|---|------|-----------------------------------|
| CNZ addr | C4 | Immediate         | 18/9 | 3 | None | Calls subroutine program if Z=1.  |
| CZ addr  | CC | Immediate         | 18/9 | 3 | None | Calls subroutine program if Z=0.  |
| CNC addr | D4 | Immediate         | 18/9 | 3 | None | Calls subroutine program if CY=1. |
| CC addr  | DC | Immediate         | 18/9 | 3 | None | Calls subroutine program if CY=0. |
| CM addr  | FC | Immediate         | 18/9 | 3 | None | Calls subroutine program if S=1.  |
| CP addr  | F4 | Immediate         | 18/9 | 3 | None | Calls subroutine program if S=0.  |
| CPO addr | FE | Immediate         | 18/9 | 3 | None | Calls subroutine program if P=0.  |
| CPE addr | EC | Immediate         | 18/9 | 3 | None | Calls subroutine program if P=1.  |
| RNZ      | C0 | Register indirect | 12/6 | 1 | None | Returns to main program if Z=1.   |
| RZ       | C8 | Register indirect | 12/6 | 1 | None | Returns to main program if Z=0.   |
| RNC      | D0 | Register indirect | 12/6 | 1 | None | Returns to main program if CY=0.  |
| RC       | D8 | Register indirect | 12/6 | 1 | None | Returns to main program if CY=1.  |
| RM       | F8 | Register indirect | 12/6 | 1 | None | Returns to main program if S=1.   |
| RP       | F0 | Register indirect | 12/6 | 1 | None | Returns to main program if S=0.   |
| RPO      | E0 | Register indirect | 12/6 | 1 | None | Returns to main program if P=0.   |
| RPE      | E8 | Register indirect | 12/6 | 1 | None | Returns to main program if P=1.   |

### **(a) Unconditional Jump Instructions**

(i) *JMP address (label)*

(Jump to Address)

This is an unconditional jump instruction. With the execution of this instruction, the program jumps to the address (or label) specified with the instruction. This is a three byte instruction and no flag is affected. In fact during the execution of *JMP address* (label) instruction, the address of the label is copied in the program counter; and whenever the program fetches the next instruction the program counter will send the address of this given label.

$[PC] \leftarrow [LABEL]$

### (a) Conditional Jump Instructions

In conditional jump instructions the program jumps to the instructions specified by the address (or label) if the given condition is fulfilled. However, if the given condition is not satisfied, the program will not jump to the specified address (or label).

rather it will proceed to the normal sequence. In all these conditional jump instructions, if program jumps to the specified address after the given condition is satisfied, it will take 10 T-states for its execution otherwise it takes 7 T-states.

Following are the conditional Jump instructions. Some of them have also been used in SAP-II computers.

When this instruction is executed, the program jumps to the instruction specified by the address (or label), if the result is not zero; otherwise it will proceed to the next instruction. Here the result of the preceding instruction is considered.

In this case the address (or label) is copied in the program counter if zero flag is reset ( $Z = 0$ ).

$[PC] \leftarrow LABEL$  if  $Z = 0$ .

This is illustrated if we consider the following instructions:

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> |
|--------------|------------------|----------------|
|              | ----             |                |
|              | ----             |                |
|              | ----             |                |
|              | ----             |                |
| NEXT         | DCR              | C              |
|              | JNZ              | NEXT           |
|              | MOV A, M         |                |
|              | STA              | 2500 H         |
|              | HLT              |                |

For the execution of JNZ instruction, the result of C-register will be considered. If  $[C] \neq 0$  ( $Z = 0$ ), it will jump to the instruction (*STA 2500 H*) specified by the label NEXT, otherwise it will jump to the next instruction (*MOVA, M*).

(ii) *JZ address (label)*      (Jump if the result is zero)

The condition of this instruction is reverse to that of *JNZ address*. In this case the program will jump to the instruction specified by the address (or label), if the result of the preceding instruction is zero ( $Z = 1$ ) otherwise it will proceed to the next instruction in the normal sequence.

The address of the label will be copied in the program counter if Z flag is set (or result is zero).

$[PC] \leftarrow LABEL$  if  $Z = 1$ .

During the execution of this instruction, it will check up the Carry flag modified by the preceding instruction. If there is no carry ( $CY = 0$  or CY flag is reset), the program will jump to the instruction specified by the address (or label) otherwise it will proceed to the next instruction of the normal sequence.

In this case the address of the label will be copied in the program counter if CY = 0 or CY is reset.

$[PC] \leftarrow LABEL$ , if CY = 0

(iv) JC address (label) (Jump if carry)

The program will jump to the instruction specified by the address (label) if the CY flag is set ( $CY = 1$ ) which is modified by the preceding instruction. However, if the carry

is reset ( $CY = 0$ ), it will proceed to the next instruction of the normal sequence. The condition of this instruction is opposite to that of the *JNC address*.

In this case the address of the label will be copied in the program counter if  $CY = 1$  or  $CY$  is set.

$$[PC] \leftarrow LABEL, \text{ if } CY = 1$$

(v) ***JM address (label)*** **(Jump if Minus)**

When this instruction is executed, the program will jump to the instruction specified by the address (label) if the result of the preceding instruction is minus or sign flag is set ( $S = 1$ ) otherwise it will proceed to the next instruction of the normal sequence.

$$[PC] \leftarrow LABEL, \text{ if } S = 1$$

(vi) ***JP address (label)*** **(Jump if Positive)**

If the result of the preceding instruction is positive or sign flag is reset ( $S = 0$ ), the program will jump to the instruction specified by the address (label). However, if the condition is not satisfied it will proceed to the next instruction of the normal sequence.

$$[PC] \leftarrow LABEL, \text{ if } S = 0$$

(vii) ***JPO address (label)*** **(Jump if Parity is Odd)**

If the parity is odd or parity flag is reset ( $P = 0$ ) as a result of the preceding instruction, the program will jump to the instruction specified by the address (label) otherwise next instruction of the normal sequence will be executed.

$$[PC] \leftarrow LABEL, \text{ if } P = 0$$

(viii) ***JPE address (label)*** **(Jump if Parity is Even)**

If the parity is even or parity flag is set ( $P = 1$ ) as a result of the preceding instruction, the program will jump to the instruction specified by the address (label) otherwise next instruction of the normal sequence will be executed.

$$[PC] \leftarrow LABEL, \text{ if } P = 1$$

### (c) CALL Instructions

The call instructions allow calling the subroutine program. The address of the subroutine program is specified with the CALL instruction. During the execution of CALL instruction, the current contents of program counter are saved on the stack and the address of subroutine (specified with the CALL instruction) is copied in the program counter. Like the jump instructions, the CALL instructions are also of two types.

1. Unconditional Call Instructions
2. Conditional Call Instructions

#### 1. Unconditional Call Instructions

***CALL address*** **(Calls the addressed subroutine program)**

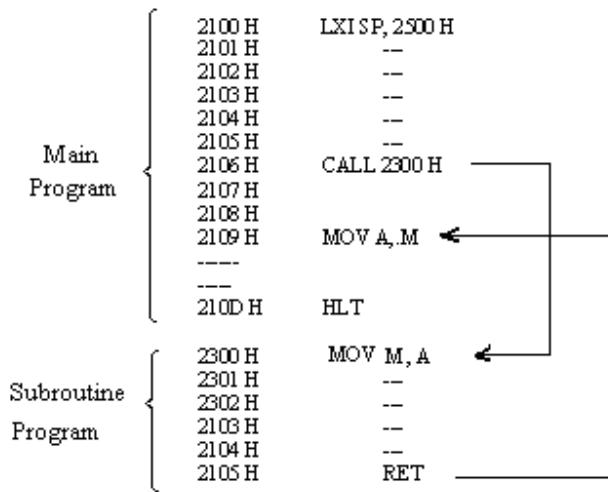
This is the format for unconditional call instruction which is of three bytes. The current contents of program counter are saved on the Stack and the address specified with CALL instruction is copied in the program counter.

i.e.  $[SP - 1] \leftarrow [PCH]$

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

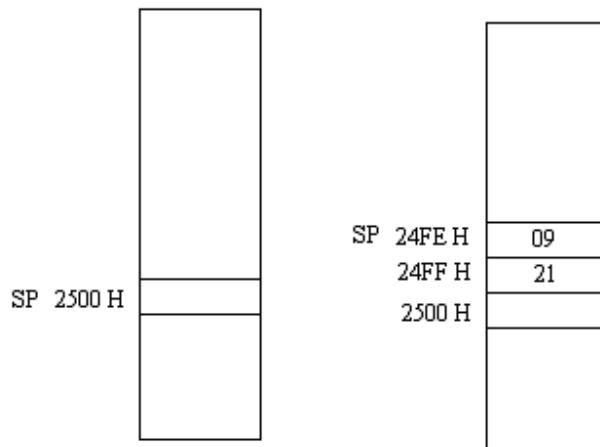
and  $[PC] \leftarrow address$

To illustrate the CALL address instruction, consider the following assembly language program:



In this program *LXI SP, 2500 H* initializes the stack pointer (stack pointer is represented by 2500 H.) i.e. the area or memory locations less than 2500 H will be used for stack purposes. When the instruction *CALL 2300 H* is executed, the address of the program counter will be 2109 H. The address of the program counter will be saved on the stack. For this stack pointer will be decremented by 1 and the high byte of the program counter (21 H) will be saved on to it; the stack pointer will further be decremented by 1 and lower byte of the program counter (09 H) will be stored on to it. The decremented of the stack pointer will be current position of the stack. This is shown in figure 4.8.

The address given with the *CALL* instruction i.e. 2300 H will be saved or copied in the program counter.



**Fig. 4.8**

The subroutine program ends at the instruction *RET* i.e. *RET* instruction will be the last instruction of the subroutine program. The *RET* instruction takes the computer back to the main program. The return saved (or pushed) on to the stack will be popped back to the program counter. Further the data stored (or saved) at the top of the stack will be copied as the lower byte of the program counter i.e.  $[PCL] \leftarrow [M_{SP}]$ . The program

counter will now be incremented by one and data stored in new stack will be saved to higher byte of the program counter i.e.  $[PCH] \leftarrow [M_{SP+1}]$ .

In the above program, after the execution of RET instruction **PC = 2109 H**.

## 2. Conditional Call Instructions

Similar to conditional jump instruction SAP-III computer has the following conditional instructions.



This instruction will call the subroutine program specified by the address provided the zero flag is reset.

i.e. if  $Z = 0$  or the result is not zero then

and  $[PC] \leftarrow address$

- (ii) CZ address (Call if zero)

It will call the subroutine (specified by the address) if zero flags is set.

i.e. if  $Z = 1$  or the result is 1 zero then

and  $[PC] \leftarrow address$



This instruction will check the carry flag. If the carry flag is reset (or CY=0) as per the executed preceding instruction, then the computer will call the addressed subroutine program.

i.e. if CY = 0 or there is no carry, then

and  $[PC] \leftarrow address$

- (iv) CC address (Call if carry)

The addressed subroutine will be called by the computer, there is a carry (or carry flag is set ; CY = 1) as per the result of the preceding instruction.

i.e. if CY = 1 or there is a carry, then

and  $[PC] \leftarrow address$

- (v) CM address (Call if minus)

During the execution of this instruction the sign flag will be checked. If the result of the preceding instruction is negative or sign flag is set ( $S = 1$ ), then only the computer will call the subroutine program specified by the address.

So if  $S \equiv 1$ , then

, then  
 $[SP-1] \leftarrow [PCH]$   
 $[SP-2] \leftarrow [PCL]$   
 and       $[PC] \leftarrow address$

- (vi) CP address (Call if positive)

In this CALL instruction too the sign flag will be checked. If the result of the preceding instruction is positive or sign flag is reset ( $S = 0$ ), then the computer will call the subroutine program specified by the address.

So if  $S = 0$ , then

and  $[PC] \leftarrow address$

**(vii) CPO address**

(Call if parity is odd)

If the parity of the result of the preceding instruction is odd (or parity flag is reset;  $P = 0$ ), then the computer will call the addressed subroutine program.

So if  $P = 0$ , then

and  $[PC] \leftarrow address$

**(ix) CPE address**

**(Call if parity is even)**

The computer will call the addressed subroutine program, if the parity of the result of the preceding instruction is even (or parity flag is set;  $P = 1$ ), then.

So if  $P = 0$ , then

$$\begin{array}{l} [SP-1] \leftarrow [PCH] \\ [SP-2] \leftarrow [PCL] \\ \text{and} \quad [PC] \leftarrow \text{address} \end{array}$$

## Return Instructions

As already discussed, the last instruction of the subroutine program is RET (Return instruction). This return instruction takes the computer back to the main program. The return instructions may also be conditional. The following are the conditional return instructions:

|            |                       |
|------------|-----------------------|
| <b>RNZ</b> | Return if no zero     |
| <b>RZ</b>  | Return if zero        |
| <b>RNC</b> | Return if no carry    |
| <b>RC</b>  | Return if carry       |
| <b>RM</b>  | Return if minus       |
| <b>RP</b>  | Return if positive    |
| <b>RPO</b> | Return parity is odd  |
| <b>RPE</b> | Return parity is even |

All the above return instructions are one byte instructions. If the specified condition is satisfied, it will return to the main program; otherwise the execution of the next instruction in the subroutine will be carried out. Further the return instruction will pop the return address to the program counter.

For example, we consider the instruction **RNC**.

It will check the carry flag. Only if the carry flag is reset ( $CY = 0$  or no carry), the computer return to main program.

i.e. if  $CY = 0$ , then

$$\begin{aligned} [PCL] &\leftarrow [SP] \\ [PCH] &\leftarrow [SP+1] \end{aligned}$$

#### **4.2.5 Stack and Input / Output Instructions**

Stack is a portion of read/write memory, which is primarily used for saving return address and data. As discussed earlier, in SAP-II computers, last two memory locations of the read / write memory FFFF H and FFFE H are exclusively used for return address of a subroutine call. In SAP-III computers, the programmer can specify any area of the memory for the stack purposes. The portion of memory located chosen by the programmer for the stack purposes can not be used for saving the return address of subroutine call. For this purpose a 16 bit register called stack pointer is provided.

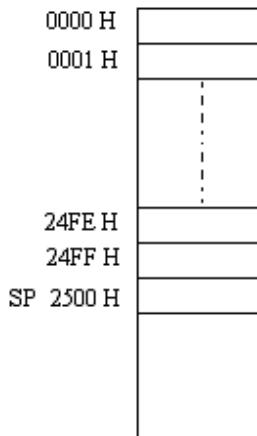
**Table 4.5**

## Stack I/O Group

| Instruction  | Op. Code | Addressing modes  | No of T-States | No of bytes of instr. | Flags affected | Operation |
|--------------|----------|-------------------|----------------|-----------------------|----------------|-----------|
| LXI SP, addr | 31       | Immediate         | 10             | 3                     | None           |           |
| PUSH B       | C5       | Register indirect | 12             | 1                     | None           |           |
| PUSH D       | D5       | Register indirect | 12             | 1                     | None           |           |
| PUSH H       | E5       | Register indirect | 12             | 1                     | None           |           |
| PUSH PSW     | F5       | Register indirect | 12             | 1                     | None           |           |
| POP B        | C1       | Register indirect | 10             | 1                     | None           |           |
| POP D        | D1       | Register indirect | 10             | 1                     | None           |           |
| POP H        | E1       | Register indirect | 10             | 1                     | None           |           |
| POP PSW      | F1       | Register indirect | 10             | 1                     | None           |           |
| IN Port      | DB       | Immediate         | 10             | 2                     | None           |           |
| OUT Port     | D3       | Immediate         | 10             | 2                     | None           |           |

The stack is initialized by the instruction given below:

For example *LXI SP, 2500 H* will indicate the stack pointer as 2500 H as



**Fig. 4.9**

shown in figure 4.9. So the memory locations lower than 2500 H may be used for stack purposes i.e. the data or return addresses may be pushed to stack in the sequence discussed earlier in CALL instructions.

Following are the Stack instructions:

#### (i) Push Instructions

In the stack not only the return address of the main program during the execution of CALL address instruction is pushed but also the contents of registers may also be pushed to stack. For this following is the format of PUSH instructions:

**PUSH rp**

where rp stands for register pair. It may be  
B for B-C register pair,  
D for D-E register pair,  
H for H-L register pair,  
PSW is known as program status word; it represents the accumulator and the flag contents i.e. **A F**

So we have the following PUSH instructions.

PUSH B ; it pushes or saves the contents of B and C registers on the stack.

PUSH D ; it pushes or saves the contents of D and E registers on the stack.

PUSH H ; it pushes or saves the contents of H and L registers on the stack.

PUSH PSW ; it pushes or saves the contents of accumulator (A) and F (flag) registers on the stack.

Following steps are carried out during the execution of **PUSH rp** instructions.

1. The stack pointer is decremented by one to get a new value of stack pointer as SP-1.

2. The contents of higher register of the given register pair (say B register in B-C register pair).

$$\text{i.e. } [M_{SP-1}] \leftarrow [rp_H]$$

3. The stack pointer is further decremented by one to get new value of stack pointer as SP-2.
4. The contents of lower register ( C register in B-C register pair) is saved in the memory location specified by SP-2.

$$\text{i.e. } [M_{SP-2}] \leftarrow [rp_L]$$

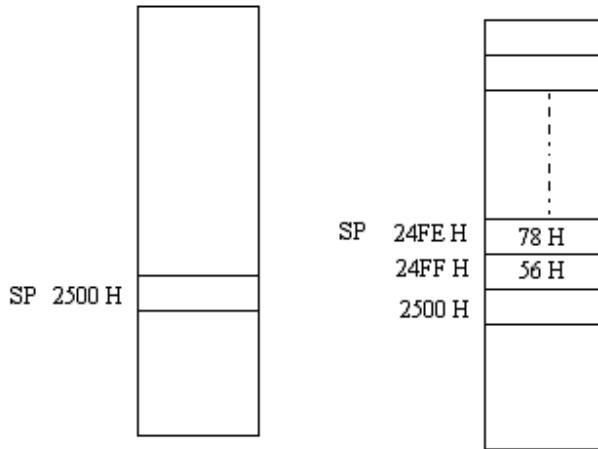
5. SP-2 will now be the stack pointer for other PUSH instructions.

For example we have

$$SP = 2500 \text{ H}$$

$$HL = 5678 \text{ H}$$

After the execution of PUSH H instruction 56 H will be loaded to memory location 24FF H and 78 H will be loaded to 24FE H memory location as shown in figure 4.10.



**Fig. 4.10**

### (ii) Pop Instructions

To retrieve the contents of registers from the stack, following POP instructions are used.

POP B ; where B stands for B-C register pair.

POP D ; where D stands for D-E register pair.

POP H ; where H stands for H-L register pair.

POP PSW ; where PSW stands for Program status word i.e. accumulator (A) and F (flag) register.

During the execution of POP instructions following steps are carried out.

- The byte stored in memory location addressed by stack pointer is saved in the lower register of the given register pair (say C in B-C register pair). The contents of higher register of the given register pair (say B register in B-C register pair).

$$\text{i.e. } [rp_L] \leftarrow [M_{SP}]$$

- The stack pointer is incremented by one to get new value of stack pointer as  $SP+1$ .
- The contents stored in the memory location addressed by stack pointer (new value as  $SP+1$ ) is saved in higher register of the given register pair ( B register in B-C register pair).

$$\text{i.e. } [rp_H] \leftarrow [M_{SP+1}]$$

- The stack pointer is further incremented by one to get the new value of stack pointer as  $SP+2$  i.e.  $SP+2$  will now work as stack pointer.

Suppose the data saved in stack is shown in figure 4.11. The stack pointer is shown as 24FC H.

Let    A = 01 H              F = 00 H  
          D = 12 H              E = 6E H

before the execution of

POP PSW

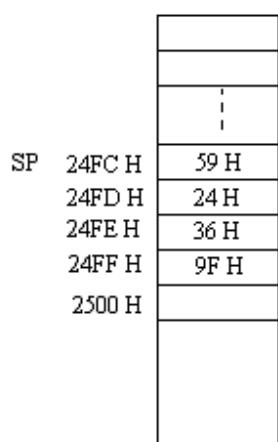
and

POP D                         instructions.

After the execution of these instructions we have the following data in the registers.

F = 59 H  
  A = 24 H  
  E = 36 H  
  D = 9F H

Now 2500 H will be the new value of stack pointer.



**Fig. 4.11**

It should be remembered that when a subroutine is called, it should save the contents of any register being used in the main program using PUSH instruction in the stack; and when it returns back to the main program from the subroutine program the

contents of the register should be retrieved from the stack using POP instruction as discussed above.

The sequence of PUSH and POP instructions should be as:

```
PUSH PSW  
PUSH D  
PUSH B  
PUSH H  
-----  
-----  
POP H  
POP B  
POP D  
POP PSW
```

It should be noted that POP retrieve the data from the stack just in reverse order of the data was PUSHed i.e. as Last In First Out (LIFO) method is used in the stack.

PUSH and POP instructions may also be used in the main program. PUSH instruction is used before the CALL instruction and POP (in the reverse order) after the CALL instruction in the main program as given below:

Main Program:

```
-----  
-----  
PUSH PSW  
PUSH H  
CALL LABEL  
POP H  
POP PSW  
-----  
-----
```

In this program the contents of Accumulator, Flag, H and L registers are saved in the Stack before the execution of CALL instruction. After the execution of subroutine program the contents of the Accumulator, Flag, H and L registers are retrieved and next instructions of the main program will be executed with restoration of earlier values of the registers.

Further, the PUSH and POP instructions may be used in the subroutine program as given below. This is generally done when the CALL instruction is used many times.

Main Program:

```
-----  
-----  
CALL LABEL  
-----  
-----
```

Subroutine Program:

|       |          |
|-------|----------|
| LABEL | PUSH PSW |
|       | PUSH D   |
|       | PUSH B   |
|       | -----    |
|       | -----    |
|       | POP B    |
|       | POP D    |
|       | POP PSW  |
|       | RET      |

## **Input / Output Instructions**

The input / output instructions deal with the input / output operations. These instructions are same used in SAP-II computer.

(i) **IN Port** (Inputs the data from the port)

In this instruction the data available at the specified port address is moved to the accumulator.

$[A] \leftarrow$  *data from port*

It is two byte instruction and no flag is affected after the execution of this instruction. This instruction is basically used to read the data from the input devices such as data read from key board, switches etc. during the computer run.

(ii) **OUT Port** (Outputs the data to the port)

This two byte output instruction is used to send the contents of accumulator to the specified port.

*Output*  $\leftarrow [A]$

#### 4.3 TIME DELAY INTRODUCED BY A REGISTER PAIR

The time delay introduced by different registers has been discussed in the last chapter using SAP-II instructions. The time delay can considerably be increased by using a register pair. A 16-bit number may be taken in two registers (register pair).

Consider the following subroutine assembly language program:

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>   |
|--------------|------------------|----------------|---|
|              | LXI D,           | F424 H         | ;Loads DE register pair with a 16-bit number.                                   |
| LOOP         | DCX              | D              | ;Decrement DE register pair by one.   |
|              | MOV A,           | E              | ;Moves the contents of E register to accumulator.                               |
|              | ORA              | D              | ;ORing of the contents of D and E registers are performed to set the zero flag. |
|              | JNZ              | LOOP           | ;If result ≠ 0 jump to loop   |
|              | RET              |                | ;Go back to main program.   |

Total T-states used for the above program are given as:

| Mnemonics | T-states |
|-----------|----------|
| LXI       | 10       |
| DCX       | 5        |
| MOV       | 5        |
| ORA       | 4        |
| JNZ LOOP  | 10/7     |
| RET       | 10       |

24 T-states are used for the inner loop and  $10+7+10 = 27$  T-states are used for outer loop.

In this program the execution of loop is for 62500 times (as  $F424\ H = 62500\_{10}$ ). The condition for the check of zero flag can not be applied just after DCX instruction, since no flag gets affected with this instruction. So to check the zero flag ORA instruction affect the zero flag. The zero flag will be set if the contents of both D and E registers are zero.

The time delay introduced by the inner loop is:

$$T_{LOOP} = 62500 \times 24 \times \text{Time of one T-state.}$$

If the system clock frequency is 3 MHz, then

$$\begin{aligned} T_{LOOP} &= 62500 \times 24 \times \frac{1}{3} \mu\text{sec} \\ &= 500000 \mu\text{sec} \\ &= 0.5 \text{ sec.} \end{aligned}$$

Delay introduced for outside loop is:

$$\begin{aligned} T_{out} &= 27 \times 1 \times \frac{1}{3} \mu\text{sec} \\ &= 9 \mu\text{sec} \end{aligned}$$

So the total time delay introduced by the above subroutine program is given by:

$$\begin{aligned} T_{Delay} &= 0.5 \text{ sec} + 9 \mu\text{sec} \\ &\approx 0.5 \text{ sec} \end{aligned}$$

**Example 4.1** What maximum delay can be introduced by the subroutine program using a register pair? Let the system clock frequency is 3 MHz.

**Solution.** The subroutine program for the delay using a register pair is given as:

| Label | Mnemonics | Operand |
|-------|-----------|---------|
|       | LXI D,    | FFFF H  |
| LOOP  | DCX       | D       |
|       | MOV       | A, E    |
|       | ORA       | D       |
|       | JNZ       | LOOP    |
|       | RET       |         |

The maximum delay can be introduced if the maximum number (FFFF H) is taken in DE register pair.

$$FFFF\ H = 65535\_{10}$$

$$\begin{aligned} \text{So } T_{LOOP} &= 65535 \times 24 \times \frac{1}{3} \mu\text{sec} + 27 \times 1 \times \frac{1}{3} \mu\text{sec} \\ &= 524280 + 9 \mu\text{sec} \end{aligned}$$

$$\begin{aligned}
 &= 524289 \mu\text{sec} \\
 &= 0.52 \text{ sec}
 \end{aligned}$$

**Example 4.2** Write a program in assembly language to introduce a time delay of 1 sec using a register pair. Let the system clock frequency is 3 MHz.

**Solution.** For one sec delay, the program discussed in section in 4.3 can be executed twice as given below:

**Main Program:**

| Label | Mnemonics | Operand |
|-------|-----------|---------|
| LOOP1 | MVI C,    | 02 H    |
|       | CALL      | DELAY   |
|       | DCR       | C       |
|       | JNZ       | LOOP1   |
|       | HLT       |         |

**Subroutine Program:**

| Label | Mnemonics | Operand |
|-------|-----------|---------|
| DELAY | LXI D,    | F424 H  |
| LOOP  | DCX       | D       |
|       | MOV       | A, E    |
|       | ORA       | D       |
|       | JNZ       | LOOP    |
|       | RET       |         |

In this program the subroutine program is run two times as subroutine program in one go introduces a time delay of 0.5 sec.

**Example 4.3** It is desired to clear the accumulator contents of a SAP-III computer. Explain the possible instructions for this purpose.

**Solution.** To clear the accumulator contents, the possible instructions are:

- (i) MVI A, 00 H      Two byte instruction and no flag gets affected.
- (ii) XRA A      One byte instruction, CY flag will be reset and all other flags will be modified as per the result.
- (iii) ANI 00 H      Two byte instruction, CY flag will be reset and all other flags will be modified as per the result.
- (iv) SUB A      One byte instruction and all flags will be affected as per the result.

**Example 4.4** Write a program in assembly language using instructions of SAP-III computers to complement the 256 bytes stored in memory locations 2500 H through 25FF H. Store the complemented data in memory locations starting at 2600 H.

**Solution.**

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
|       | LXI H,    | 2500 H  | ; Loads the address of the first number in H-L register pair. |
|       | MVI C,    | FF H    | ; 256 bytes (FF H) is stored in C-register as index register. |
| LOOP  | MOV A,    | M       | ; Moves the contents of memory location addressed             |

|                  |           |  |
|------------------|-----------|--|
|                  |           | by H-L register pair to accumulator.   |
| CMA              |           | ;complements the accumulator contents.   |
| MVI H,<br>MOV M, | 26 H<br>A | ;Stores 26 H in H-register<br>;Complemented data is stored in the destination memory location. |
| MVI H,<br>INX H  | 25 H      | ;Stores 25 H in H-register<br>;Increments the H-L register pair.                               |
| DCR C            |           | ;Decrement the contents of C register.   |
| JNZ              | LOOP      | ;Go back to loop for the operation of next data.   |

HLT

**Example 4.5** Write a program in assembly language using instructions of SAP-III computers to find the smaller of two numbers stored in memory locations 2501 H and 2502 H. Store the result in 2503 H memory location.

**Solution.**

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
|       | LXI H,    | 2501 H  | ;Loads the address of the first number in H-L register pair. |
|       | MOV A,    | M       | ;Saves the first number in accumulator.                      |
|       | INX H     |         | ;Increments the H-L register pair.                           |
|       | CMP M     |         | ;compares the two numbers.                                   |
|       | JC        | NEXT    | ;if carry smaller number is in accumulator go to NEXT.       |
|       | MOV A,    | M       | ;If no carry then move the smaller number to accumulator.    |
| NEXT  | STA       | 2503 H  | ;Stores the smaller number in 2503 memory location .         |

HLT

**Example 4.6** Write a program in assembly language using instructions of SAP-III computers to add two hexadecimal numbers stored in memory locations 2501 H and 2502 H. The answer should be stored in 2503 H memory location. The carry if any should be stored in 2504 H memory location.

**Solution.**

| Label | Mnemonics | Operand | Comments |
|-------|-----------|---------|----------|
|-------|-----------|---------|----------|

|      |        |        |   |
|------|--------|--------|---|
|      | LXI H, | 2501 H | ;Loads the address of the first number in H-L register pair.      |
|      | MVI C, | 00 H   | ;Clears C-register for carry (most significant bit).              |
|      | MOV A, | M      | ;Saves the first number in accumulator.                           |
|      | INX H  |        | ;Increments the H-L register pair.                                |
|      | ADD M  |        | ;Adds the two numbers.  |
|      | JNC    | NEXT   | ;if no carry go to NEXT.  |
|      | INR C  |        | ;Else increment the content of C-register.                        |
| NEXT | STA    | 2503 H | ;Stores the answer .  |
|      | MOV A, | C      | ;Moves the contents of C register (carry contents) to accumulator |
|      | STA    | 2504 H | ;Saves the contents of carry.                                     |
|      | HLT    |        |   |

**Example 4.7** Write a program in assembly language for SAP-III computers to find the sum of a series  $1+2+3+\dots+10$  (or sum of first 10 natural numbers).

| Solution. Label | Mnemonics | Operand | Comments   |
|-----------------|-----------|---------|--|
|                 | MVI B,    | 01 H    | ;Loads first number of series to B-register.                       |
| LOOP            | MVI C,    | 00 H    | ;Clears C-register.  |
|                 | MOV A,    | C       | ;Saves the contents of C-register to accumulator.                  |
|                 | ADD B     |         | ;Adds the contents of B-register with the contents of accumulator. |
|                 | MOV C,    | A       | ;Saves the partial sum to C-register.                              |
|                 | INR B     |         | ;increments the contents of B-register.                            |
|                 | CPI 0B H  |         | ;Compares with natural number 11.                                  |
|                 | JNZ       | LOOP    | ;If not 11 go back to LOOP.  |
|                 | MOV A,    | C       | ;Saves the answer to accumulator.                                  |
|                 | STA       | 2500 H  | ;Saves the answer to memory location.                              |
|                 | HLT       |         |  |

**Example 4.8** Reset all the flags, in a subroutine program, without performing arithmetic or logical instructions. However the contents of other registers should not be changed.

**Solution.**

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>  |
|--------------|------------------|----------------|--|
|              | PUSH H           |                | ;Saves the contents of H-L register pair to stack.           |
|              | PUSH PSW         |                | ;Saves the contents of A-F to stack.                         |
|              | LXI H, 0000 H    |                | ;Loads 0000 H to H-L register pair.                          |
|              | PUSH H           |                | ; Saves the contents of H-L register pair to stack.          |
|              | POP PSW          |                | ;Saves the contents of stack in A-F registers.               |
|              | POP H            |                | ; Retrieve the contents of H-L register pair from the stack. |
|              | MOV A, H         |                | ;Moves the contents of H-register to accumulator.            |
|              | POP H            |                | ;Pops the contents from stack to H-L register pair.          |
|              | RET              |                |  |

**Example 4.9** How will you exchange the contents of BC and HL register pairs?

**Solution.** Following program may exchange the contents of BC and HL register pairs:

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>  |
|--------------|------------------|----------------|--|
|              | LXI SP, XXX H    |                | ;Initializes the stack pointer.  |
|              | PUSH B           |                | ;Saves the contents of B-C register pair to stack.                             |
|              | PUSH H           |                | ;Saves the contents of H-L register pair to stack.                             |
|              | POP B            |                | ;Pops the contents of stack (H-L register pair contents) to B-C register pair. |
|              | POP H            |                | ;Pops the contents of stack (B-C register pair contents) to H-L register pair. |
|              | HLT              |                |  |

**Example 4.10** Write a subroutine program using SAP-III instructions, which will subtract the contents of D-E register pair from H-L register pair and the result is to be stored in H-L register pair.

**Solution.**

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>   |
|--------------|------------------|----------------|---|
|              | MOV A, L         |                | ;Transfer the contents of L register to accumulator.              |
|              | SUB E            |                | ;Contents of E register are subtracted from accumulator contents. |

|        |   |   |
|--------|---|---|
| MOV L, | A | ;Answer (list significant digit of the answer) is loaded to L register. |
| MOV A, | H | ;Moves the contents of H register to accumulator.                       |
| SBB D  |   | ;Subtracts the content of D register from acc. with carry.              |
| MOV H, | A | ;Most significant digit of the answer is saved in H register.           |
| HLT    |   |   |

**Example 4.11** What will be the value of accumulator and flags (CY, S, P and Z), after the execution of the following program.

| Label | Mnemonics | Operand |
|-------|-----------|---------|
|       | MVI D,    | 7F H    |
|       | MVI C,    | 3E H    |
|       | MOV A,    | C       |
|       | RLC       |         |
|       | RLC       |         |
|       | ANA D     |         |
|       | HLT       |         |

**Solution.** After the execution of the above program we have:

|                  |                     |        |
|------------------|---------------------|--------|
| D =              | 0 1 1 1 1 1 1 1     |        |
| C =              | 0 0 1 1 1 1 1 0     |        |
| A =              | 0 0 1 1 1 1 1 0     |        |
| After first RLC  | A = 0 1 1 1 1 1 0 0 | CY = 0 |
| After second RLC | A = 1 1 1 1 1 0 0 0 | CY = 0 |
| ANDing of D      | = 0 1 1 1 1 1 1 1   |        |

---

The value of accumulator is  
The values of Flag register are:

|      |                 |                      |
|------|-----------------|----------------------|
| A =  | 0 1 1 1 1 0 0 0 | = 78 H               |
| CY = | 0               | (reset after ANDing) |
| S =  | 0               |                      |
| P =  | 1               |                      |
| Z =  | 0               |                      |

**Example 4.12** Write a program using SAP-III instructions to check the even parity or the odd parity of the number stored in memory location 2010 H. Send 00 H or EE H at the output port 02 H if the parity is odd or even respectively.

**Solution.**

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
|       | LXI H,    | 2010 H  | ;Initializes the H-L register pair with the address of the location. |
|       | MOV A, M  |         | ;Moves the number to accumulator.                                    |

|     |             |  |  |
|-----|-------------|--|--|
|     | ORA A       |  | ;ORing of A with A will load the same number to accumulator. The parity flag will be affected with this operation. |
|     | JPO ODD     |  | ;Jump to ODD if parity is odd.   |
|     | MVI A, EE H |  | ;Load EE H to accumulator for even parity.   |
|     | OUT 02 H    |  | ;EE is sent to output port 02H.  |
|     | JMP END     |  | ;Jump to end.  |
| ODD | MVI A, 00 H |  | ;Load 00 H to accumulator for odd parity.  |
|     | OUT 02 H    |  | ;00 is sent to output port 02H.  |
| END | HLT         |  |  |

**Example 4.13 (a)** What will be the value of B-register after the following program is executed.

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b> |
|--------------|------------------|----------------|-----------------|
|              | MVI A,           | 07 H           |                 |
|              | STC              |                |                 |
|              | CMC              |                |                 |
|              | RAL              |                |                 |
|              | MOV B,           | A              |                 |
|              | STC              |                |                 |
|              | CMC              |                |                 |
|              | RAL              |                |                 |
|              | STC              |                |                 |
|              | CMC              |                |                 |
|              | RAL              |                |                 |
|              | ADD B            |                |                 |
|              | MOVB,            | A              |                 |
|              | HLT              |                |                 |

**(b)** Suggest some alternative program that can perform the same operation.

**Solution.** (a) In the beginning accumulator is loaded with a hexadecimal number 07 H.

The instruction *STC* and *CMC* resets the carry flag and *RAL* instruction moves the accumulator contents left through carry giving us the contents 0E H ( $=14_{10}$ ) i.e. *STC*, *CMC* and *RAL* instructions multiplies the accumulator contents by a factor of two. The accumulator contents are now saved in register B.

Three instructions *STC*, *CMC* and *RAL* are further used twice so that accumulator contents are multiplied by a factor of 4. This way after the third *RAL* in the program we have  $A = 8 \times 07 H = 38 H = 56_{10}$ .

After *ADD B* we get:

$$\begin{aligned} A &= 38 H + 0E H \\ &= 46 H \\ &= 70_{10} \end{aligned}$$

So this program multiplies the accumulator contents by a factor of 10 (decimal number) which is stored in B-register.

$$\begin{array}{ll} \text{So} & B = 46 \text{ H} \\ & = 70_{10} \end{array}$$

(b) The program given in (a) multiplies the accumulator contents by a factor of 10 (decimal number) which is stored in B-register. The following program also perform the same operation.

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>   |
|--------------|------------------|----------------|---|
|              | MVI A,           | 07 H           | ;Loads 07 H to accumulator.   |
|              | ANI              | FF H           | ;It clears the carry flag.  |
|              | RAL              |                | ;Shifts left by one bit or multiplies accumulator contents by a factor of two.                          |
|              | MOV B, A         |                | ;Moves the answer to B-register.  |
|              | ANI              | FF H           | ;It clears the carry flag.  |
|              | RAL              |                | ;Shifts left by one bit or multiplies accumulator contents by a factor of two.<br>So $A = 4 \times A$ . |
|              | ANI              | FF H           | ;It again clears the carry flag.  |
|              | RAL              |                | ;Shifts left by one bit or multiplies accumulator contents by a factor of two.<br>So $A = 8 \times A$ . |
|              | ADD B            |                | $;[A] \leftarrow [A] + [B]$ So it gives $A = 10 \times A$   |
|              | MOV B, A         |                | ;Result is stored in B register.  |
|              | HLT              |                |   |

This program does the same operation as given in (a), however, less number of instructions is used in this program.

**Example 4.14** A byte of data is stored in memory location 2500 H. Display at the output port 02 H the number of 1's present in the data byte stored in memory location 2500 H.

**Solution.**

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>   |
|--------------|------------------|----------------|---|
|              | LDA              | 2500 H         | ;Loads the data byte in accumulator.  |
|              | MVI C,           | 08 H           | ;08 H is loaded to C-register to use it as a counter.                             |
| LOOP         | MVI B,           | 00 H           | ;00 H is loaded to B-register.  |
|              | RLC              |                | ;MSB is shifted to carry flag.  |
|              | JNC              | NEXT           | ;Checks the carry flag, if not zero go to next (i.e. no 1 is present at the MSB). |
|              | INR B            |                | ;Increments the B-register by one.  |

|      |        |      |   |
|------|--------|------|---|
| NEXT | DCR C  |      | ;Decrements C-register.                       |
|      | JNZ    | LOOP | ;Jumps to LOOP if Z = 0.                      |
|      | MOV A, | B    | ;No. of 1's present is loaded to accumulator. |
|      | OUT    | 02 H | ;Sends the data to the output port.           |
|      |        | HLT  |   |

### PROBLEMS

1. Give the programming model (Software model) of SAP-III computers.
2. How many flags are used in SAP-III computers? How are these flags affected with the result?
3. Explain with the help of block diagram how the carry flag gets affected with the result.
4. In how many groups the instruction set of SAP-III computers are classified. Explain with examples the instructions of arithmetic group.
5. Discuss the rotate instructions used in SAP-III computers. How the flags are affected with these instructions.
6. What is the difference between the stack and stack pointer? Discuss PUSH and POP instructions.
7. Explain what operations are performed when the following instructions are executed. Give at least one example for each operation.
  - (i) LXI H, address
  - (ii) ADD M
  - (iii) CMP reg
8. What is difference between *CMP reg* and *SUB reg* instructions? Explain with suitable examples.
9. If carry flag is zero, then show that RAL instruction produces a multiplication of accumulator contents by a factor of 2.
10. If carry flag is zero, then show that RAR instruction produces a division of accumulator contents by a factor of 2.
11. Show that the instruction DAD H shifts left the data in H-L register pair by one bit.
12. Discuss various addressing modes of SAP-III computers instructions and give at least two examples of each addressing mode.
13. What do you understand by subroutine? Explain the use of stack and stack pointer in calling a subroutine. Explain any other use of stack.
14. Write a program for SAP-III computers introduce a time delay of 0.5 sec if the system clock frequency is 6 MHz.
15. Explain the following instruction of SAP-III computer. Also discuss which flags get affected with the execution of these instructions.  
ADD M, CMP B, POP PSW and MOV A, M.
16. An *ORA reg* or *ORI data* instruction can be used to make a selected bit of a specified register as 1. Justify this statement. Also write the mnemonic of an instruction that will set bit 6 of the accumulator without changing any of the other bits in the register.  
**(Ans.: ORI 40 H)**

17. What will be contents of accumulator and flags (CY, S, P and Z), after the execution of ADD D instruction; if A = C3 H and D = 3D H.

(Ans: A = 00 H, CY = 1, S = 0, P = 1 and Z = 1)

18. What will be contents of accumulator and flags (CY, S, P and Z), after the execution of SUB D instruction; if A = C3 H and D = 3D H.

(Ans: A = 85 H, CY = 0, S = 1, P = 0 and Z = 0)

19. What will be contents of memory location 2500 H and flags (CY, S, P and Z), after the execution of the following program:

```
MVI C, C8 H  
MVI A, 11 H  
ADD C  
STA 2500 H  
HLT
```

(Ans: M<sub>2500</sub> = D9 H, CY = 0, S = 1, P = 0 and Z = 0)

20. What will be the value of stack pointer, after the following program written in SAP-III instructions is executed.

```
LXI SP, 27FF H  
PUSH D  
CALL SUBROUT  
POP D  
ADD D  
PUSH D  
HLT
```

What will be the value of stack pointer after this program?

(Ans: SP = 27FD H)

21. What will be the contents in B-register, after the execution of the following assembly language program?

| Label | Mnemonics | Operand |
|-------|-----------|---------|
|       | MVI A,    | 08 H    |
|       | STC       |         |
|       | CMC       |         |
|       | RAL       |         |
|       | MOV B,    | A       |
|       | HLT       |         |

(Ans.: B = 10 H = 16<sub>10</sub>)

---

# 5

## The 8085 Microprocessor

---

After the study of conceptual computers SAP-I, SAP-II and SAP-III, we are now in the position to understand the fundamentals of 8-bit microprocessor 8085. As discussed in the preceding chapters, the basic components of all the computers are Keyboard, Display devices, Memory devices and Central Processing unit. The central processing unit (CPU) is the heart of the computer and also called microprocessor.

The technological revolution brought in recent years, the invention of micro-programmable computer on microprocessor chip. First four bit microprocessor chip INTEL-4004 was developed by Intel Corporation of America in 1971. Intel introduced in 1972 an 8-bit microprocessor 8008 and in 1973 another 8-bit microprocessor 8080. The microprocessor 8080 was the most popular microprocessor of the early 70s. In the year 1974, Intel developed a 40 pin microprocessor chip 8085, which was the enhanced version of 8080. Intel 8080 microprocessor was not in fact a complete CPU on a chip because the clock and controller were on separate chip. Further, it utilizes two separate power supplies. The 8085 microprocessor has the advantages over 8080 that it has on-chip clock and control circuit. It needs only one power supply of +5 volts.

### 5.1 ARCHITECTURE OF 8085 MICROPROCESSOR

Intel 8085, an 8-bit NMOS microprocessor is available in the form of 40 Pin dual in line IC package. It is fabricated on a single LSI chip. It operates on +5 V d.c. supply. The clock speed used in this microprocessor is about 3 MHZ. General Purpose 8-bit microprocessor is capable of addressing up to 64 K bytes (i.e.  $2^{16} = 65536$  bytes) of memory. The functional block diagram is shown in figure 5.1.

The main functional components of 8085A microprocessor are as given below:

- (i) Register Section
- (ii) Arithmetic and Logic Unit
- (iii) Timing and Control Section
- (iv) Interrupt Control
- (v) Serial Input / Output Control

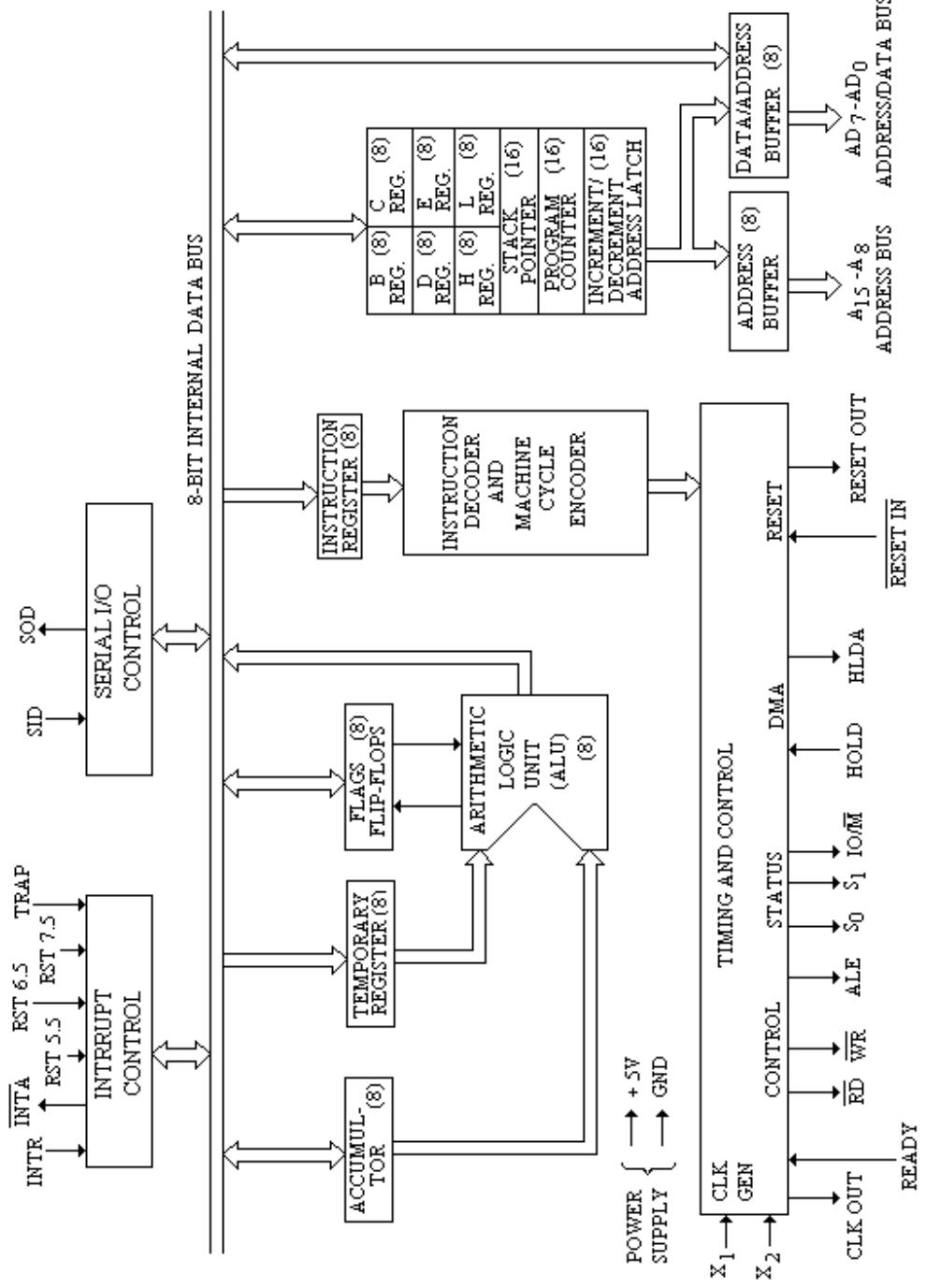


Fig. 5.1

### **5.1.1 Register Section**

The 8085 microprocessor contains eight addressable 8-bit registers namely:

|   |                                 |
|---|---------------------------------|
| A | (Accumulator) register          |
| F | Flag register (Flag flip-flops) |
| B | register                        |
| C | register                        |
| D | register                        |
| E | register                        |
| H | register                        |
| L | register                        |

Out of these registers B, C, D, E, H and L registers are 8-bit general purpose registers. These registers can either be used as single register or a combination of two registers as 16 bit register pair. As discussed in SAP-III computers, the valid register pairs are B-C, D-E or H-L register pairs. The higher order byte of 16 bit data is stored in first register (B in B-C register pair), and low order byte in the second register (C in B-C register pair).

The H-L register pair can also be used for register indirect addressing; since this register pair can also function as data pointer.

The large number of general purpose registers gives more flexibility and ease in the programming. However, the general purpose registers are limited as registers occupy more space on the silicon chip.

Beside these general purpose registers, the 8085 has remaining two 8-bit registers Accumulator (A) and Flag (F) as special purpose registers and two 16 bit registers namely Program counter (PC) and stack pointer (SP).

#### **Accumulator (A)**

As discussed in preceding chapters, accumulator is a 8 bit buffer register extensively used in arithmetic, logic, load and store operations as well as in input / output instructions. All the arithmetic and logical operations are performed on the accumulator contents; i.e. one of the operand is always taken into the accumulator.

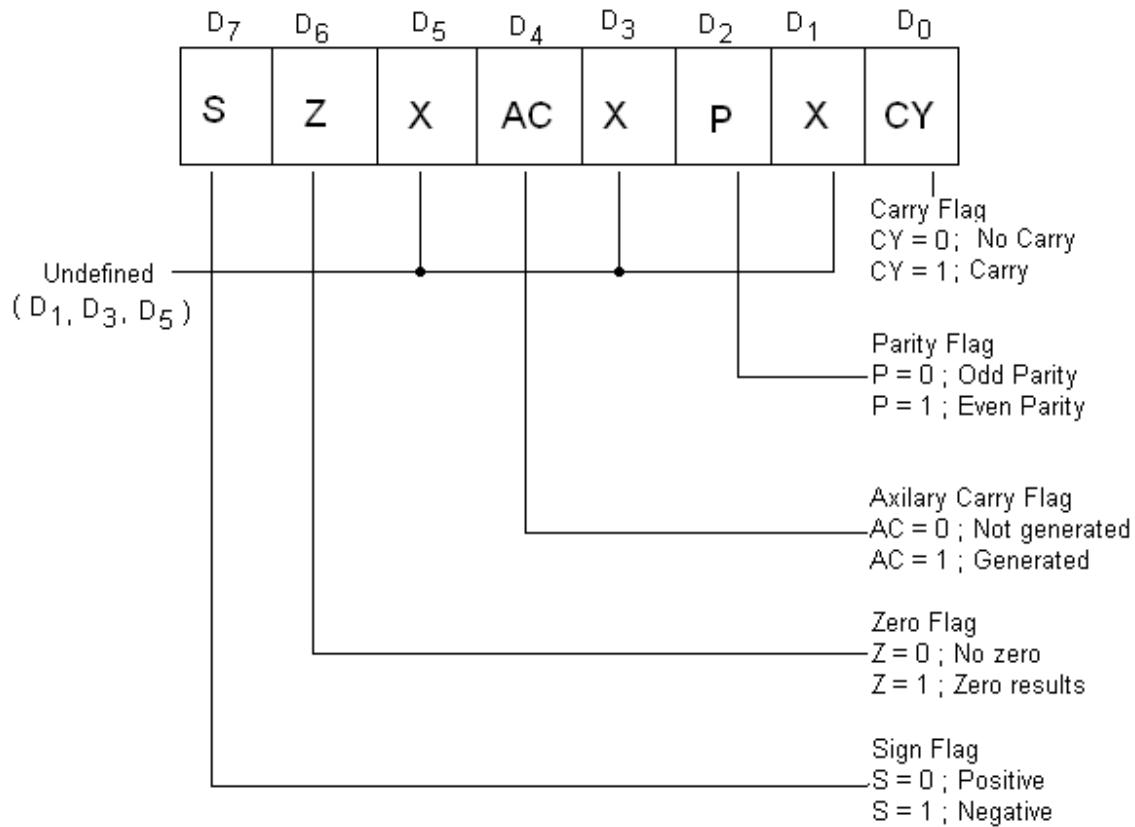
#### **Flag (F) Register**

It is an 8-bit register associated with the execution of instructions in the microprocessor. Out of the 8 bits of flag register, 5 bits contains significant information in terms of status flags. The five flags are:

- (i) Sign flag (S)
- (ii) Zero flag (Z)
- (iii) Carry flag (CY)
- (iv) Parity flag (P)
- (v) Auxiliary Carry flag (AC)

All the flags except the Auxiliary carry (AC) flag have been discussed in SAP-III computers.

The bit positions reserved for these flags in the flag register (F) is shown in figure 5.2.



**Fig. 5.2**

**(i) Sign flag (S)**

The sign flag is set ( $S = 1$ ), if the result of the operation of the instruction is negative (MSB of the result is 1); otherwise it is reset ( $S = 0$ ) for the positive result (MSB is zero).

**(ii) Zero flag (Z)**

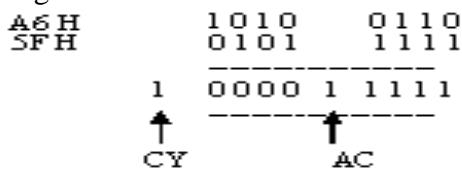
The zero flag is set ( $Z = 1$ ) if the result of the operation of the instruction is zero otherwise this flag is reset ( $Z = 0$ ).  
i.e.  $Z = 1$  if the result is zero,  
and  $Z = 0$  if the result is not zero.

**(iii) Carry flag (CY)**

The carry flag is set to 1, if there exist a carry (or borrow) to the highest order bit (non-existent 9<sup>th</sup> position) as a result of the execution of addition or subtraction instructions. If there is no carry (or borrow) to the higher order bit, the carry flag is reset.  
i.e.  $CY = 1$  if there is a carry to the highest order bit (or overflow), and  $CY = 0$  if there is no carry to the highest order bit (or no overflow).

**(iv) Parity flag (P)** After an arithmetic and logic operation, if the result has even number of 1s, then parity bit is set. If on the other hand the result has odd number of 1s, the parity flag is reset.  
i.e.  $P = 1$ , if the result has even number of 1s,  
and  $P = 0$ , if the result has odd number of 1s.

**(v) Auxiliary carry (AC)** This is a new flag in 8085 microprocessor. This flag (AC) is set to 1, if there is an overflow at bit 3 of the accumulator. AC flag is used in BCD arithmetic. This is illustrated as given below:



As shown in figure 5.2, the five bits in flag register are defined. The three bits are undefined. The Accumulator and 8 bits (including three undefined bits) of flag register form a Program Status Word (PSW). The accumulator and flag registers are treated as a 16 bit unit for stack operation.

#### Program Counter – 16 bit register

The program counter is a 16 bit register. It is used to send 16 bit address to fetch the instruction from the memory. It acts as a pointer which indicates the address of the next instruction to be fetched and executed. The program counter is updated after an instruction has been fetched by the processor. If an instruction is one byte instruction, then the program counter will be updated by one (i.e.  $PC = PC + 1$ ). Similarly, for two and three byte instructions, the program counter will be updated by two (i.e.  $PC = PC + 2$ ) or three (i.e.  $PC = PC + 3$ ) locations respectively.

#### Stack Pointer – 16 bit register

The stack is an area of RAM (random access memory or read / write memory) in which temporary information is stored. It is stored on First-In-Last-Out (FILO) basis. An address in the RAM area is assigned to the stack pointer where the first information is stored as the first stack entry. This is done by initializing stack pointer by an instruction. Higher stack entries are made at the progressively decreasing addresses.

#### 5.1.2 Address Buffer and Address-Data Buffer

It has already been discussed that 8085 requires 8-bit data bus and 16-bit address bus, as the memory address is of 16 bits. More number of IC pins are required if separate address and data bus are introduced. To restrict the number of pins of 8085 to only 40, lower address lines  $A_0-A_7$  and data lines  $D_0-D_7$  are used in multiplexed mode. The multiplexed lines are designed as Address/Data Bus ( $AD_0-AD_7$ ). So whenever 16-bit address is transmitted by the microprocessor 8 MSBs of the address lines are sent on the Address Bus ( $A_{15}-A_8$ ) and 8 LSBs of the lines are sent on the Address/Data Bus ( $AD_7-AD_0$ ). The 8 LSBs of the address are then latched either into memory or external latch so that the complete address remains available for further operation. The 8-bit Address/Data Bus will now be free for the data transmission.

#### 5.1.3 Arithmetic and Logical Unit (ALU)

The arithmetic and logical unit (ALU) basically consists of accumulator (A), flag register (F) and a temporary register (which is inaccessible by the programmer or user). This unit carries out the arithmetic and logic calculations of the data stored in general purpose registers or in memory locations. The arithmetic operations are ADD, SUB, compare, increments, decrements and complements etc.; while logical operations are AND, OR, XOR and Rotate. The result of these operations could be placed in the accumulator or elsewhere through the internal bus. Arithmetic operations are usually carried out in 2's complement adder / subtractor discussed in the preceding chapter. For these operations, ALU receives the control signals from the timing and control unit.

#### 5.1.4 Timing and Control Unit

This unit consists of the following sections:

1. Instruction Register and Decoder
2. Control signals

##### 1. Instruction Register and Decoder

As discussed in the preceding chapter, the CPU fetches an instruction from the memory for its execution. This instruction can be of 1-3 byte long. The first byte contains the op code of instruction which basically specifies the nature of operation to be performed indicating the length of the instruction. The first byte (op code of the instruction) transferred to 8-bit instruction register through the internal bus of the CPU, becomes available at the instruction decoder. The decoder decodes the op code and directs the control unit to produce the necessary control signals.

##### 2. Control Signals

Following are the control signals of 8085 microprocessor needed for the operation of CPU.

###### (i) X<sub>1</sub>, X<sub>2</sub> and CLK Out

Two pins X<sub>1</sub> and X<sub>2</sub> are provided to be externally connected to a quartz crystal. The clock signal of fixed frequency is generated through the internal circuitry of the processor. The frequency at which the microprocessor 8085 works is half of the crystal frequency. The quartz crystal of 6.144 MHz is used in this processor. This gives the clock frequency of 3.072 MHz (half of the crystal frequency) of 50% duty cycle. The clock period is of about 320 nsec. The output of the clock frequency is also available at CLK out terminal.

###### (ii) Address Latch Enable (ALE)

The 16 bit address bus is basically divided into two sets. The most significant bits A<sub>7</sub>-A<sub>15</sub> of the address bus are used separately and the least significant bits of the address AD<sub>0</sub>-AD<sub>7</sub> are time multiplexed with the bits of bidirectional data bus (D<sub>0</sub>-D<sub>7</sub>). The AD<sub>0</sub>-AD<sub>7</sub> bus serves the dual purpose as they can be used as low-order address bus as well as bidirectional data bus at different times. This is used as address bus, during the first clock cycle of the machine cycle involving memory; and during the remaining clock cycle of the machine cycle, it acts as the data bus. This is accomplished by address latch enable (ALE) signal provided in the processor. During the first clock cycle of the machine cycle ALE is high which enables the lower 8-bit of the address to be latched either into the memory or external latch.

###### (iii) $\overline{R_D}$ (Read ) Signal

This is an active low signal to be connected to memory read input (output enable signal to memories) or to input / output read signal to enable input / output buffer.

#### (iv) $\overline{W_R}$ (Write) Signal

Similar to read signal ( $\overline{R_D}$ ), write signal ( $\overline{W_R}$ ) is also active low. This signal is used to write to the memory or input / output devices.

#### (v) $IO/\overline{M}$ (Input Output / Memory)

This signal  $IO/\overline{M}$  distinguishes that the address and data is meant for either I/O devices or memory. Whenever this signal is high (1), microprocessor will communicate to the I/O devices and whenever it is low (0), microprocessor will communicate to the memory.

#### (vi) Status Signals ( $S_0, S_1$ )

The status signals ( $S_0, S_1$ ) along with  $IO/\overline{M}$  signal indicate the type of machine cycle in progress. The type of machine cycle are op code fetch cycle, memory read cycle, memory write cycle, I/O read cycle or I/O write cycle.

Various types of status codes are given in table 5.2.

**Table 5.2**

| Machine Cycle       | $IO/\overline{M}$ | $S_1$ | $S_0$ |
|---------------------|-------------------|-------|-------|
| Op code fetch Cycle | 0                 | 1     | 1     |
| Memory Read Cycle   | 0                 | 1     | 0     |
| Memory Write Cycle  | 0                 | 0     | 1     |
| I/O Read Cycle      | 1                 | 1     | 0     |
| I/O Write Cycle     | 1                 | 0     | 1     |
| INTR Acknowledge    | 1                 | 1     | 1     |
| Halt                | Hi-Z              | 0     | 0     |

#### (vi) Hold and HLDA

HOLD and HLDA (Hold Acknowledge) signals are used for DMA (Direct Memory Access) operation. In a microprocessor, the data transfer between I/O devices and memory will take place through the microprocessor. The involvement of the processor slows down the data transfer between I/O devices and memory. The transfer of data directly from I/O devices to memory without involvement of microprocessor is called DMA. The DMA will save the time as CPU relinquishes the control of Buses. In this way DMA transfers the large amount of data in a relatively short time. The HOLD and HLDA signals are used in the operation. Whenever HOLD signal is high, CPU temporarily relinquishes its operation by floating the address, data and control buses; and DMA operation is started. A high HLDA (Hold Acknowledge) signal is also sent to DMA controller, indicating that CPU has received the hold request. Whenever the data transfer is complete, then the control to CPU is returned back by sending a HOLD signal. Further the HLDA signal goes low.

#### (viii) READY signal (Input)

Some peripheral devices connected to 8085 microprocessor operate at much slower speed than the processor. To synchronize the speed of CPU and peripheral devices or to slow down the speed of 8085, the READY signal is used. If the READY signal is high the peripheral device is ready and the processor can complete the data transfer. If

this signal is low the microprocessor waits (by generating a number of NOP T-states) till it goes high.

#### (ix) RESET IN and RESET OUT

The signal RESET IN may be low from the operator Reset button or from the processor. When the signal is low, the CPU will reset the program counter, instruction register and other circuits. It also sends a high RESET OUT. The RESET OUT signal

goes to peripheral devices to reset or initialized. When RESET IN signal goes high and RESET OUT goes low, the data processing may begin.

#### 5.1.5 Interrupt Control

Sometimes it is necessary to interrupt the execution of the main program. For this an interrupt request is obtained from the I/O devices. After receiving the interrupt request (INTR), processor temporarily stops what it was doing and attends to the I/O device.

INTA is an interrupt acknowledge signal which is sent by the microprocessor after INTR signal is received. After the work of the I/O device is complete it returns to what it was doing earlier.

Basically 8085A has five hardware interrupts namely:

**INTR**  
**RST 5.5**  
**RST 6.5**  
**RST 7.5**  
and      **TRAP**

If two or more of these interrupts are active at the same time, the 8085 takes them in order of priority level. The priority levels of these interrupts are given in table 5.3.

**Table 5.3**

| Interrupts     | Priority |
|----------------|----------|
| <b>TRAP</b>    | <b>1</b> |
| <b>RST 7.5</b> | <b>2</b> |
| <b>RST 6.5</b> | <b>3</b> |
| <b>RST 5.5</b> | <b>4</b> |
| <b>INTR</b>    | <b>5</b> |

The details of these instructions will be discussed in chapter 7.

#### 5.1.6 Serial I/O Control

Serial input / output control circuit incorporated in this microprocessor is used for the data transmission. For this purpose two pins SID and SOD are provided in the serial input/output control unit. The SID (Serial Input Data) terminal receives the serial data stream from an input device, the control unit converts serial data stream to parallel data before it is used by the computer. After the conversion 8-bit parallel data is stored in the accumulator. Similarly, SOD (Serial Out Data) terminal outputs the 8-bit parallel available with the accumulator into serial form to the peripheral device connected with the computer.

#### 5.2 PIN DESCRIPTION OF 8085

The pin details and logical schematics of the 40 pin dual line package (DIP) IC 8085 are shown in figures 5.3(a) and (b) respectively. Fig. 5.3 (c) shows the shape of the microprocessor. The descriptions of various pins of the microprocessor are given below:

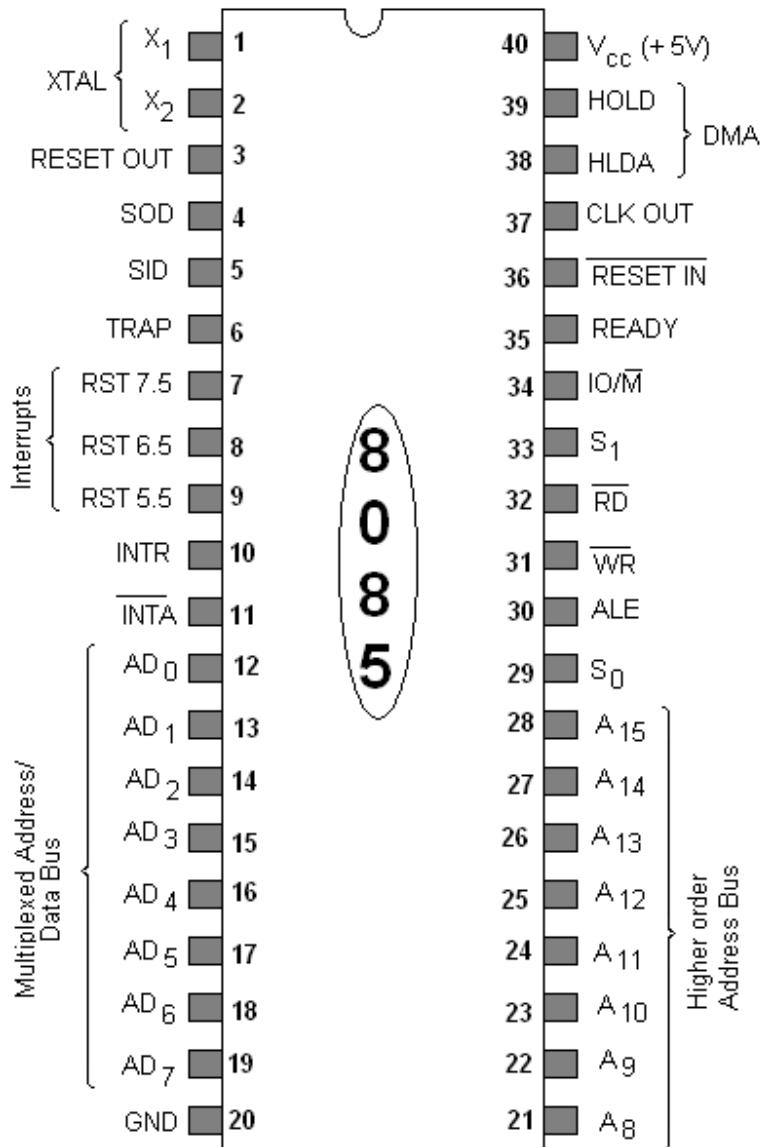


Fig. 5.3 (a)

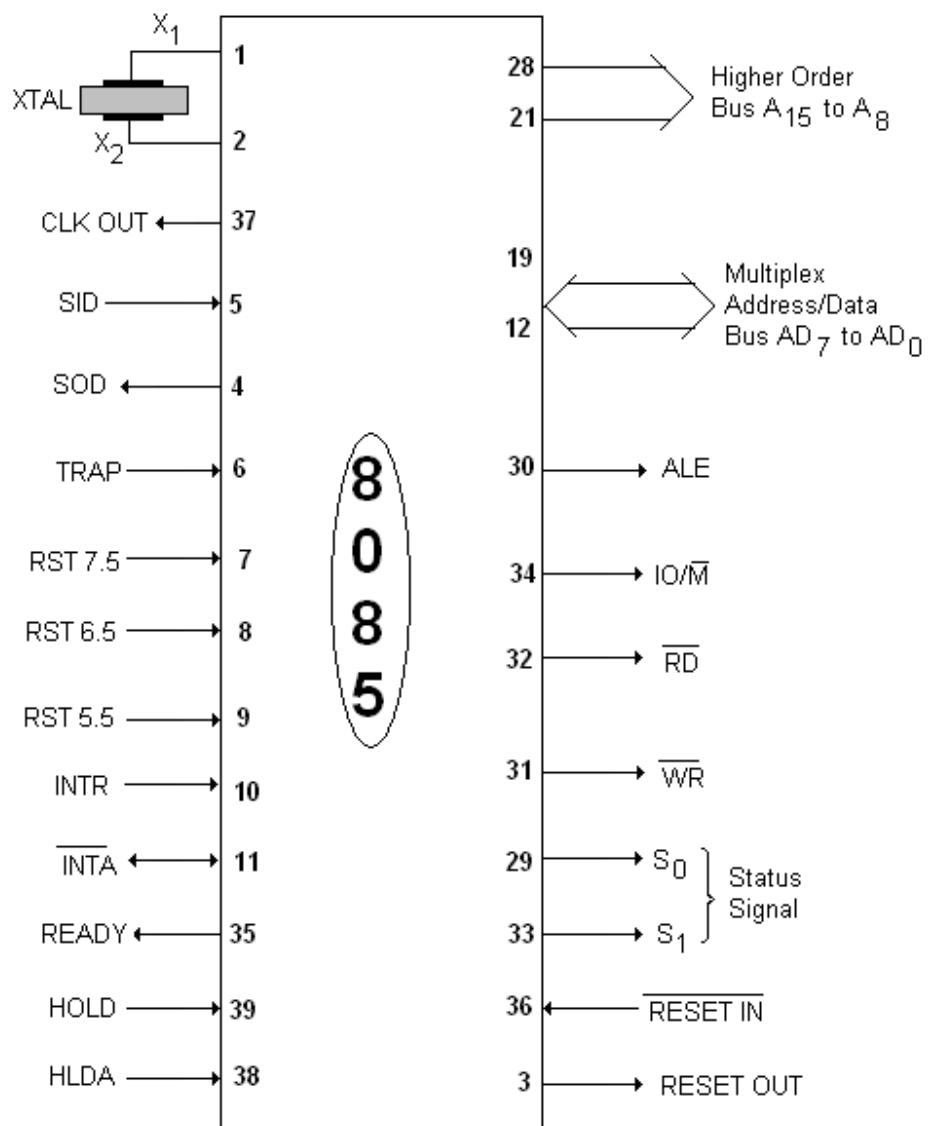


Fig. 5.3 (b)

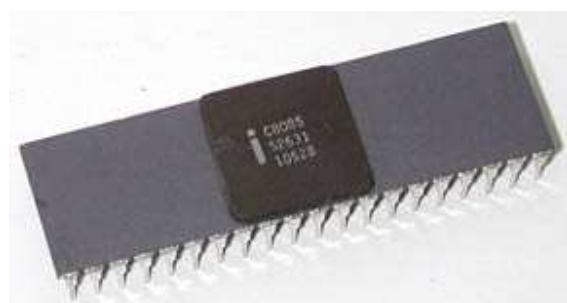


Fig. 5.3 (c)

### PIN NOS. 1 and 2:

These  $X_1$  and  $X_2$  pins are to be connected to an external quartz crystal, L-C or R-C network which drives the internal clock generator. The clock signal of appropriate frequency is determined when a quartz crystal is connected to the on-chip oscillator as shown in figure 5.4(a). The oscillator output from the Schmitt trigger drives a flip-flop which divides the frequency by a factor of two. The circuit produces two clock signals  $\Phi 1$  (CLK) and  $\Phi 2$  ( $\overline{\text{CLK}}$ ) to derive the internal circuit of the microprocessor. A 6.25 MHz crystal is used to provide 3.125 MHz internal clock frequency.

Generally, quartz crystal is used for the On-chip oscillator for the accurate and stable clock frequency, though a parallel resonant L-C circuit may be used for the frequency determining network as shown in figure 5.4(b). The network produces a signal whose frequency tolerance is about  $\pm 10\%$ . The component values may be chosen from the following formula:

$$f = \frac{1}{2\pi\sqrt{L(C + C_{in})}}$$

The input capacitance  $C_{in}$  is approximately 15 Pf. To minimize the variations in frequency, it is recommended to choose  $C$  as 30 Pf.

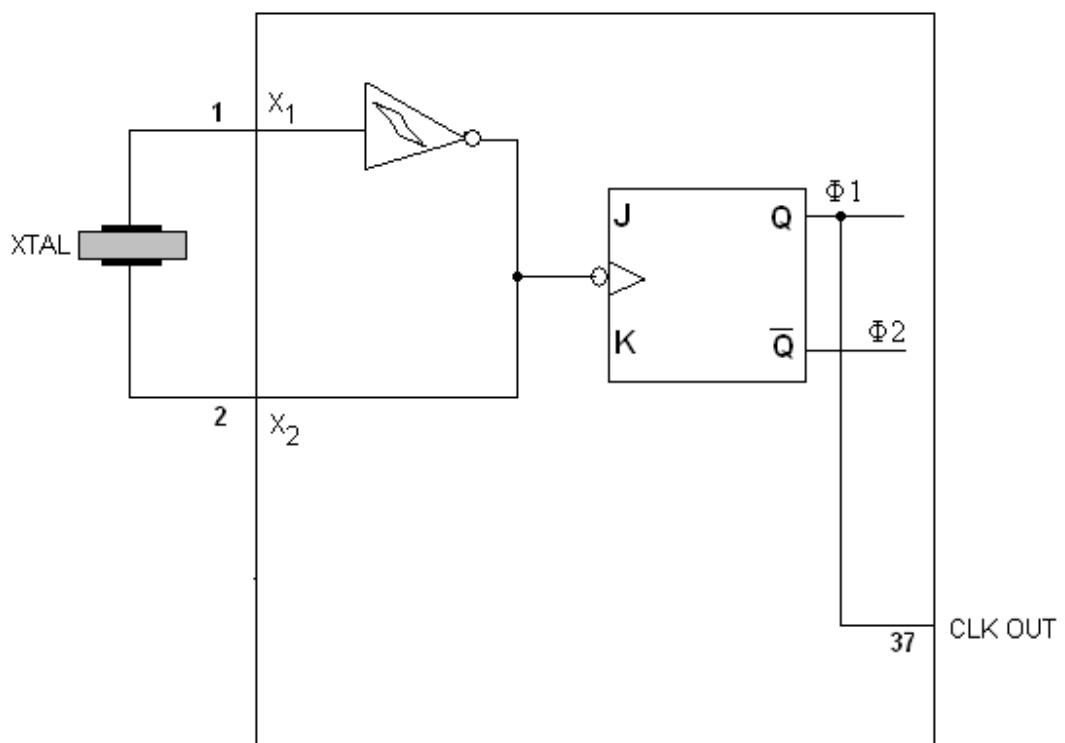
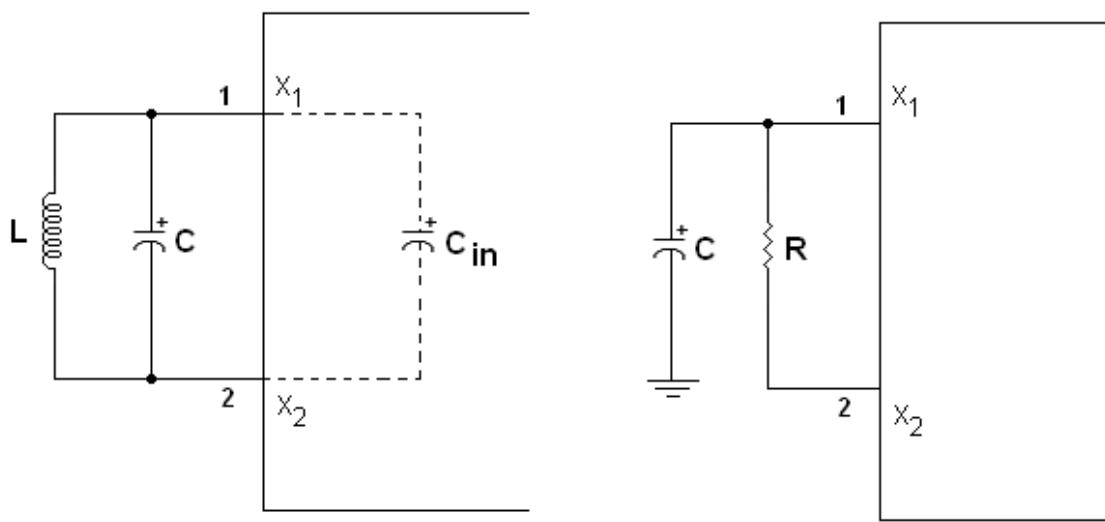


Fig. 5.4(a)



**Fig. 5.4 (b)**

**Fig. 5.4 (c)**

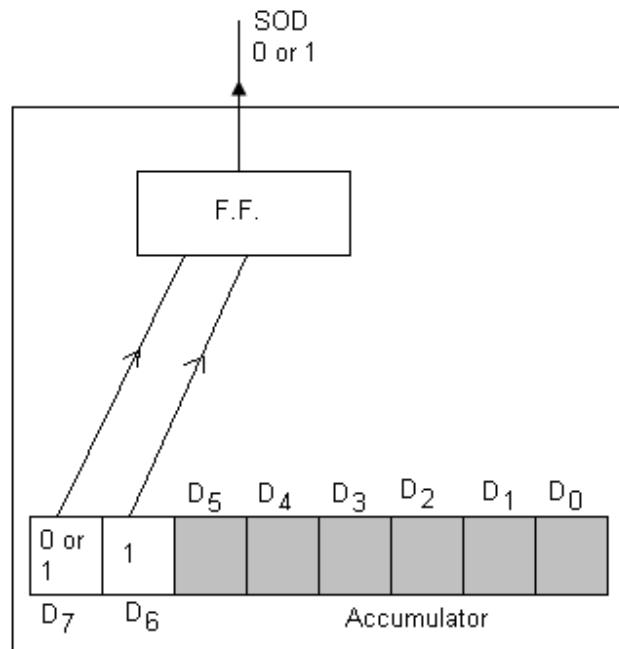
An R-C network may also be used as the frequency determining network for the on-chip oscillator of the microprocessor as shown in figure 5.4(c). The driving frequency generated by this circuit is approximately 3MHz. It is not recommended to use the frequencies higher or lower than this.

#### PIN NO. 3

This is **RESET OUT** signal, which indicates that CPU is being reset. When it is high, system is reset. The signal is synchronized to the processor clock and lasts for an integral number for clock periods. When the **RESET OUT** signal goes low, the processing begins.

#### PIN NOS. 4 and 5

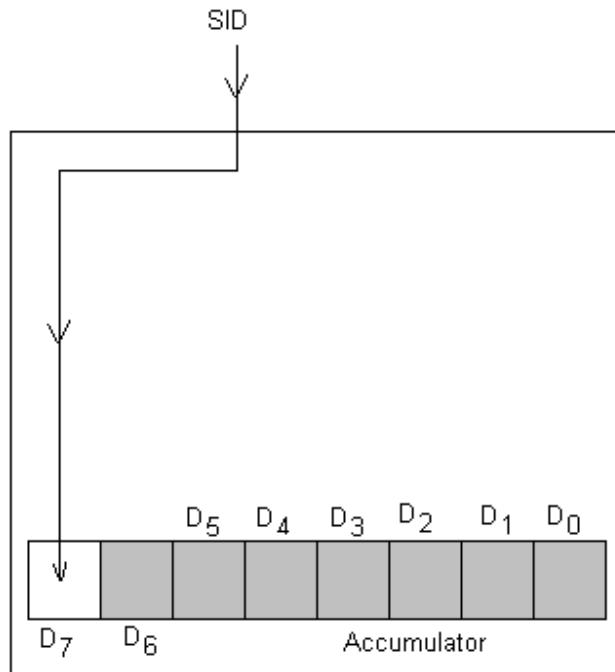
Pin Nos. 4 and 5 indicate SOD (Serial Out Data) and SID (Serial In Data) terminals respectively. These pins are associated with Serial Input/Output control unit for 8085 microprocessor. As already discussed these pins are used for the serial data transmission. The SOD output pin can deliver a serial data stream to a peripheral device. On executing SIM (Set Interrupt Mask) instruction, if bit D<sub>6</sub> is set to 1, the content of D<sub>7</sub> bit (set or reset) of the accumulator is latched on the SOD pin as shown in figure 5.5(a).



**Fig. 5.5 (a)**

The data on the SID line (PIN 5) loads into accumulator at bit D<sub>7</sub> whenever a RIM instruction is executed as shown in figure 5.5(b).

The details of SIM and RIM instructions will be discussed in chapter 7.



**Fig. 5.5 (b)**

**PIN NOS. 6 to 11**

The interrupt control unit of the microprocessor contains these pins. The Pins 6 to 11 are restart interrupts named as:

|                     |              |
|---------------------|--------------|
| TRAP (Pin No.6)     | I Priority   |
| RST 7.5 (Pin No. 7) | II Priority  |
| RST 6.5 (Pin No. 8) | III Priority |
| RST 5.5 (Pin No. 9) | IV Priority  |
| INTR (Pin No. 10)   | V Priority   |

The TRAP has the highest priority and INTR has the lowest priority. The priority level is of importance if two or more interrupts become active at the same time. The TRAP is non-maskable interrupt. It is both edge and level sensitive.

The interrupts (TRAP, RST 7.5, RST 6.5 and RST 5.5) are also called vector interrupts, as each interrupt has fixed memory location (vector location) for the transfer of control from the normal execution of the routine. The vector locations of these interrupts are given in table 5.4. As soon as any of these pins 6 to 10 are active (high), the internal circuit of 8085 stops the normal execution of program and the program control is transferred to the corresponding memory location (vector location).

**Table 5.4**

| Interrupts     | Memory locations |
|----------------|------------------|
| <b>TRAP</b>    | <b>0024 H</b>    |
| <b>RST 7.5</b> | <b>003C H</b>    |
| <b>RST 6.5</b> | <b>0034 H</b>    |
| <b>RST 5.5</b> | <b>002C H</b>    |

INTR (Pin No. 10) is a general purpose interrupt and has the lowest priority. As soon as Pin No. 10 is high, the microprocessor stops the execution of normal program and after completing the instruction at hand, it goes to CALL instruction. The INTR is enabled or disabled by the instructions ET (Enable Interrupts) or DI (Disable Interrupts) respectively.

The Pin No. 11 is an Interrupt Acknowledge (INTA) signal. A low (logic 0) to this pin indicate that the microprocessor has acknowledged the request from the peripheral device. It is also used to activate the interrupt controller.

#### **PIN NOS. 12 to 19**

Pin Nos. 12 to 19 (AD<sub>0</sub>-AD<sub>7</sub>) form bi-directional multiplexed Address/Data Bus. The least significant 8 bits of the memory address (or I/O Address) appear on the bus during the first T-states of a machine cycle. It then becomes the data bus during the next T-states.

#### **PIN NO. 20**

Pin No. 20 is the ground terminal.

#### **PIN NOS. 21 to 28**

The Pin Nos. 21 to 28 (A<sub>8</sub>-A<sub>15</sub>) form unidirectional most significant 8 bits of memory address or 8 bits of the I/O address. It remains in the high impedance state during HOLD, HALT and RESET modes.

#### **PIN NOS. 29 to 33**

The Pin Nos. 29 to 33 labeled as  $S_0$  and  $S_1$  respectively are known as status signals. These status signals along with  $IO/\bar{M}$  signal indicate the various operations as indicated in table 5.5.

**Table 5.5**

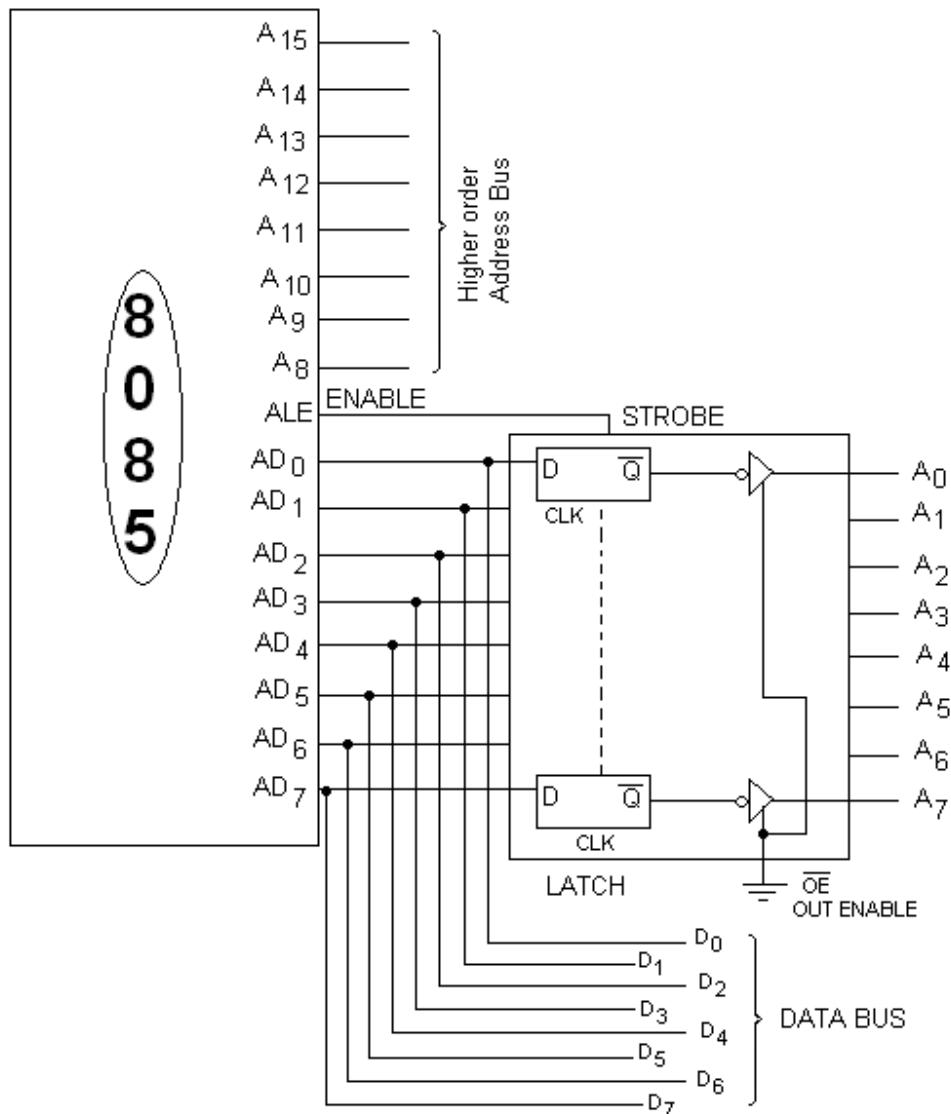
| Machine cycle         | $IO/\bar{M}$ | Status<br>$S_1$ | $S_0$    | Control signals                    |
|-----------------------|--------------|-----------------|----------|------------------------------------|
| <b>Op code Fetch</b>  | <b>0</b>     | <b>1</b>        | <b>1</b> | $\overline{RD} = 0$                |
| <b>Memory Read</b>    | <b>0</b>     | <b>1</b>        | <b>0</b> | $\overline{RD} = 0$                |
| <b>Memory Write</b>   | <b>0</b>     | <b>0</b>        | <b>1</b> | $\overline{WR} = 0$                |
| <b>I/O Read</b>       | <b>1</b>     | <b>1</b>        | <b>0</b> | $\overline{RD} = 0$                |
| <b>I/O Write</b>      | <b>1</b>     | <b>0</b>        | <b>1</b> | $\overline{WR} = 0$                |
| <b>Interrupt Ack.</b> | <b>1</b>     | <b>1</b>        | <b>1</b> | $\overline{INTA} = 0$              |
| <b>HALT</b>           | <b>HI-Z</b>  | <b>0</b>        | <b>0</b> |                                    |
| <b>HOLD</b>           | <b>HI-Z</b>  | <b>X</b>        | <b>X</b> | $\overline{RD}, \overline{WR} = Z$ |
| <b>RESET</b>          | <b>HI-Z</b>  | <b>X</b>        | <b>X</b> | $\overline{INTA} = 1$              |

**HI-Z = High Impedance State**

**X = Unspecified**

### **PIN NO. 30**

The Pin No. 30 is known as ALE (Address Latch Enable) terminal. When this signal is high the information carried on the multiplexed address/data bus ( $AD_0-AD_7$ ) is the lower 8 bits of the address. It also enables the low order address ( $AD_0-AD_7$ ) from the multiplexed address/data bus to latch either into the memory or the external latch. The ALE signal separates the low order address and data from the multiplexed Address/data Bus. This is illustrated in figure 5.6.



**Fig. 5.6**

#### PIN NOS. 31, 32 and 34

The Pin Nos. 31 and 32 are the two control signals  $\overline{WR}$  (Write bar) and  $\overline{RD}$  (Read bar) respectively. The pin 34 carries  $IO/\overline{M}$  signal which is one of the status signals. The other status signals are S0 and S1 discussed earlier. A low  $\overline{WR}$  signal generated by the microprocessor sends (writes) data into I/O devices or memory. Similarly, a low  $\overline{RD}$  signal generated by the microprocessor reads (receives) the data from the I/O devices or memory locations. The  $IO/\overline{M}$  signal indicates whether the address on the address bus is meant for I/O devices. However, a low to this signal indicates that the address on the address bus is meant for memory location. The  $\overline{RD}$ ,  $\overline{WR}$  and  $IO/\overline{M}$  signals function together.

#### PIN NO. 35

The Pin No. 35 is known as READ signal which forces the microprocessor to wait till the data become available from the memory or input/output devices. This signal is needed to synchronize the speed of the microprocessor with I/O devices or memory as the memory or I/O devices are not as fast as the microprocessor. When the READ signal is low, the microprocessor waits till the READY signal is 1. As soon as READY signal is 1, the microprocessor knows that the data are available from the memory or I/O devices.

#### **PIN NO. 36**

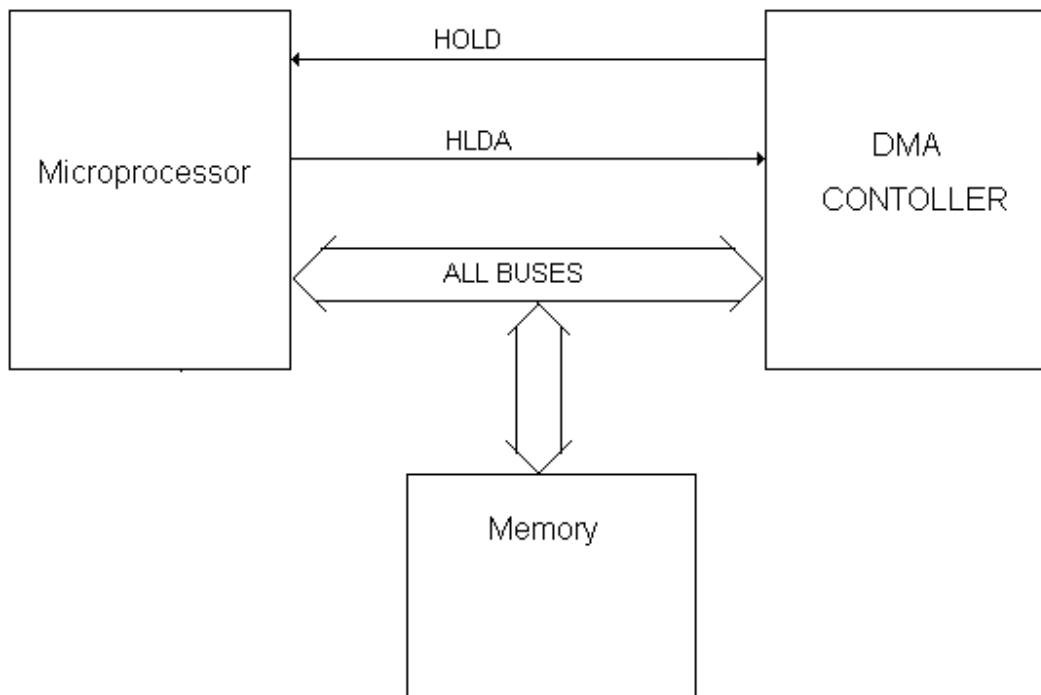
This pin is RESET IN signal. This input carrying signal may be operated by the operator using the RESET button provided externally or it may be operated directly from the other source. When this signal is low (momentarily), the CPU will reset the program counter, instruction register, all interrupts (except TRAP) are disabled, SOD signal becomes low and Data, address and control buses are floated. When this signal goes high, the data processing begins.

#### **PIN NO. 37**

This pin carries CLK OUT signal. It is derived from the on-chip oscillator, which goes to peripherals to synchronize their timings.

#### **PIN NOS. 38-39**

The Pin Nos. 38 and 39 are the HOLD and HLDA (Hold Acknowledge) signals respectively. These signals are used in DMA (Direct Memory Access) operations. As shown in figure 5.7, when any I/O device indicates that the data are ready for DMA transfer, a high HOLD signal is sent by the DMA controller to the 8085 microprocessor. It is in fact a request signal from the DMA controller to the microprocessor. The microprocessor then sends a high signal to DMA controller indicating that the microprocessor has received the request from the I/O devices and will relinquish the address, data and control bus after completing the current instruction. The DMA controller thus carries out the data transfer. A low HOLD signal will return the control to the microprocessor.



**Fig. 5.7**

#### PIN NO. 40

The pin 40 is +VCC, which is to be externally connected to +5 volt d.c. supply.

#### 5.3 INSTRUCTION SET OF 8085 MICROPROCESSOR

The 8085 includes all the instructions of SAP-III. In addition there are few more instructions which will be discussed below. These new instructions were not considered in SAP-3 because of its architecture.

##### **LHLD address**

**(Loads the H-L pair direct)**

This instruction loads the H-L pair direct with two bytes already stored in two consecutive memory locations starting at the specified memory address. The contents stored in the memory location whose address is given with the instruction will be loaded to the L-register; and the contents stored in the next memory location (address + 1) will be loaded to the H-register.

$$\text{i.e. } [L] \leftarrow [M_{\text{address}}]$$

$$\text{and } [H] \leftarrow [M_{\text{address+1}}]$$

For example, let 2A H is stored in the memory location 2100 H and 2B H is stored in the memory location 2101 H, then after the execution of the instruction LHLD 2100 H, the L-register will have 2A H and H-register will have 2B H.

None of the flags is affected with this instruction.

##### **SHLD address**

**(Stores the H-L pair direct)**

This instruction does the reverse operation of LHLD. The instruction SHLD address stores the contents of L-register to memory location whose address is given with the instruction; and the contents of H-register are stored in the next consecutive memory location (address + 1).

i.e.  $[M_{\text{address}}] \leftarrow [L]$   
 and  $[M_{\text{address+1}}] \leftarrow [H]$

No flag is affected with this instruction too.

For example, if  $[L] = 3A$  H and  $[H] = 3B$  H, then after the execution of the instruction SHLD 2200 H will result.

$[M_{2200H}] \leftarrow 3A$   
 and  $[M_{2201H}] \leftarrow 3B$

#### **LDAX rp** (Loads the Accumulator Indirect)

This instruction loads the accumulator, the contents already stored in the memory location addressed by the register pair (rp). Here rp represents B-C or D-E register pair. The H-L register pair is not included in this instruction.

i.e.  $[A] \leftarrow [M_{\text{rp}}]$

The possible combinations of the instruction are:

LDAX B  
 LDAX D

No flag is affected with the execution of this instruction.

For example, if  $[D] = 25$  H,  $[E] = 00$  H

and  $M_{2500H} = 34$  H,

then after the execution of the instruction LDAX D, the accumulator will have:

$[A] \leftarrow [M_{2500H}]$   
 i.e.  $A = 34$  H

It is worth while to mention that the instruction LDAX H does not exist, because the contents stored in the memory location addressed by H-L register pair may be loaded to accumulator by the instruction MOVA, M.

#### **STAX rp** (Stores the Accumulator Indirect)

The STAX rp instruction does the reverse operation of LDAX rp. This instruction stores the accumulator contents in the memory location addressed by the register pair (rp). Here too rp represents B-C or D-E register pair. The H-L register pair is not included in this instruction.

i.e.  $[M_{\text{rp}}] \leftarrow [A]$

The possible combination of this instruction are:

STAX B  
 STAX D

No flag is affected with the execution of this instruction.

For example, if  $B = 21$  H,  $C = 00$  H

and  $A = 3A$  H,

then after the execution of the instruction STAX B, 3A H will be stored in the memory location 2100 H.

i.e.  $[M_{2100H}] = 3AH$

The combination STAX H is not included in this instruction as MOV A, M performs the same operation.

**XCHG****(Exchange the contents of H-L register with D-E register)**

This is one byte instruction and no operand is needed with it. It exchanges the contents of H and L register with D and E registers respectively.

i.e.

$$[H] \leftrightarrow [D] \text{ and } [L] \leftrightarrow [E]$$

For example:

If       $H = 25\text{ H}$ ,     $L = 32\text{ H}$   
 and      $D = 12\text{ H}$ ,     $E = 1B\text{ H}$

then after the execution of XCHG instruction, we have:

$H = 12\text{ H}$ ,     $L = 1B\text{ H}$   
 and      $D = 25\text{ H}$ ,     $E = 32\text{ H}$

The instruction XCHG is generally used to keep track of more than one memory location at a time without using LDAX and STAX instructions.

Let us write a program to add two numbers stored in memory locations 2100 H and 2201 H without using LDAX and STAX instructions. The answer is to be loaded in the memory location 2100 H.

```

LXI H, 2201 H ; Loads H = 22 H and L = 01 H
LXI D, 2100 H ; Loads D = 21 H and E = 00 H
MOV A, M      ; [A] ← [M2201H]
XCHG          ; H = 21 H, L = 00 H and D = 22 H, E = 01 H
ADD M          ; [A] ← [A] + [M2100H]
MOV M, A      ; [M2100H] ← [A]
HLT

```

In this program no LDAX and STAX instructions are used.

**DAA****(Decimal Adjust the Accumulator)**

The DAA is one byte instruction and no operand is needed with this instruction. It adjusts the accumulator to packed BCD (Binary Coded Decimal) after addition of two BCDs. In other words, after addition of two hexadecimal numbers if this instruction is used then the result in decimal form is obtained. For this Auxiliary Carry Flag (AC) and Carry Flag (CY) take care of this instruction.

It functions in two steps:

1. If the lower nibble (lower 4-bits) of the accumulator is greater than 9 or Auxiliary carry flag is set, then it adds 06 H to the accumulator.
2. Subsequently, if the higher nibble (higher 4-bits) of the accumulator is now greater than 9 or the carry flag (CY) is set, it adds 60 H to the accumulator.

All the flags are affected with this instruction.

**Example 5.1** What will be the value of accumulator, CY and AC flags after the execution of the following program:

```

MVI A, 38 H
ADI 87 H
DAA
HLT

```

**Solution.**     $A = 38\text{ H} \quad 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0$   
                 Adds 87 H     $1\ 0\ 0\ 0\ 0\ 1\ 1\ 1$

---

|   |                                       |
|---|---------------------------------------|
| $\begin{array}{r} 101111 \\ 00000110 \\ \hline \end{array}$   | Lower nibble is more than 9           |
| $\begin{array}{r} 11000101 \\ 01100000 \\ \hline \end{array}$ | AC = 1<br>Upper nibble is more than 9 |
| $\begin{array}{r} 100100101 \\ \hline \end{array}$            | Result = 125 (Decimal)                |

**PC** **HL** A = 0 0 1 0 0 1 0 1 CY = 1 AC = 1 (Copies H-L to PC)

This is one byte instruction and no operand is needed with this instruction. It copies the contents of H-register to high-order byte of the program counter (PC) and the contents of L-register to low order byte of the program counter.

$[PC] \leftarrow [HL]$       i.e.       $[PCH] \leftarrow [H]$       and       $[PCL] \leftarrow [L]$

For example if  $PC = 2106\text{ H}$  and  $HL = 2500\text{ H}$

then after the execution of the instruction *PCHL* will result:

PC = 2500 H

No flag is affected with the instruction.

This instruction is basically an unconditional jump instruction as is evident from the above example.

**SPHL** (Copies HL to Stack Pointer SP)

**SPHL** (Copies HL to Stack Pointer SP)  
This is also one byte instruction as no operand is used. The *SPHL* instruction copies the contents of H-register to high order byte stack pointer (SP) and the contents of L-register to low order byte of stack pointer (SP).

$$[\text{SP}] \leftarrow [\text{HL}] \quad \text{i.e.} \quad [\text{SPH}] \leftarrow [\text{H}] \quad \text{and} \quad [\text{SPL}] \leftarrow [\text{L}]$$

None of the flags is affected with this instruction.

For example if  $H = 23\text{ H}$  and  $L = 45\text{ H}$   
and  $SP \equiv 2501\text{ H}$

then after the execution of the instruction *SPHL* will result:

SP = 2501 H

This is another way of initializing the stack pointer.

**XTHL** (Exchanges the top of the stack with H-L register pair)

The **XTHL** is one byte instruction and does not require any operand. The top byte of the stack is exchanged with L-register and next byte of the stack is exchanged with H-register.

i.e.  $[L] \leftrightarrow [M_{-+}]$

and  $[H] \leftrightarrow [M^-]$

For example if  $H = 21 H$  and  $M_1 = 1 A H$  then  $M_2 = 2 C H$

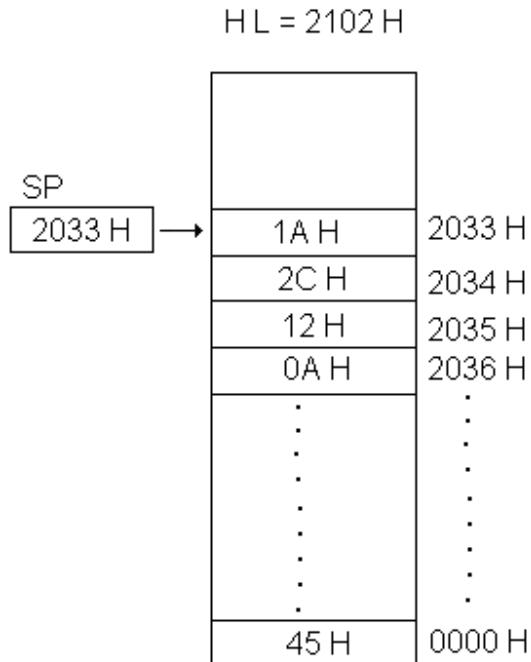
as shown in figure 5-8(a), then after the execution of the instruction *XTHL*, will result:

$$H = 3CH \quad J = 1\Delta H$$

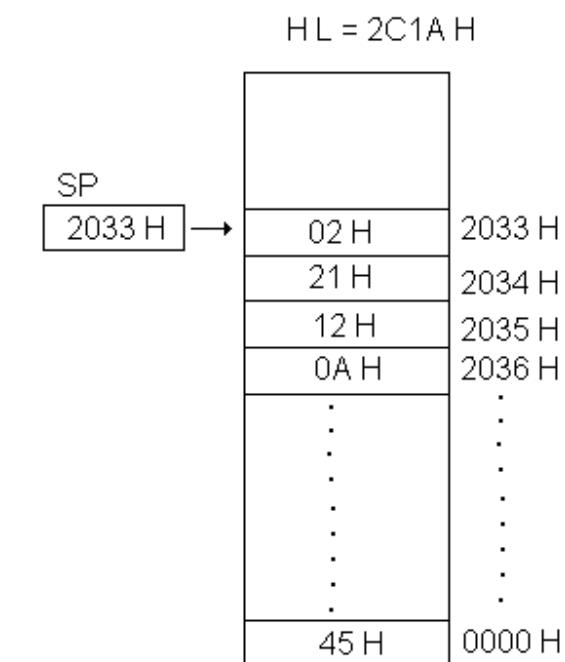
and  $M_{SP} = 02 \text{ H}$

as shown in figure 5-8(h).

No flag is affected with the instruction



**Fig. 5.8 (a)**



**Fig. 5.8 (b)**

## 5.4 TIMING DIAGRAM FOR 8085 INSTRUCTIONS

The working of 8085 microprocessor can best be understood by considering the timing diagrams of its few instructions. The graphical representation depicting the necessary steps carried out in a machine cycle is known as Timing Diagram. As is well known that the total time required for the execution of an instruction is the time required to fetch and execute an instruction.

i.e. **Instruction Cycle = Instruction fetch + Instruction execution**

The instruction cycle may be of one, two or three bytes. During the fetch cycle, an instruction of the program (op code) is extracted from the memory locations and copied in the instruction register (IR) of C.P.U.

The op code of the instruction copied in the instruction register (IR) is decoded during the execution cycle to perform the specific activities.

It may further be mentioned that an instruction cycle may take one to five machine cycles. Generally, first machine cycle known as op code fetch cycle, has either four or six T-states and the remaining machine cycles called execution cycles, have two or three T-states. A T-state represents one clock cycle.

The NOP is the shortest instruction which takes only one machine cycle with four T-states. However, the CALL is the longest instruction which takes five machine cycles with 18 T-states.

Here the timing diagrams of a few instructions will be discussed.

### 5.4.1 Timing Diagram of *MOV reg, M*

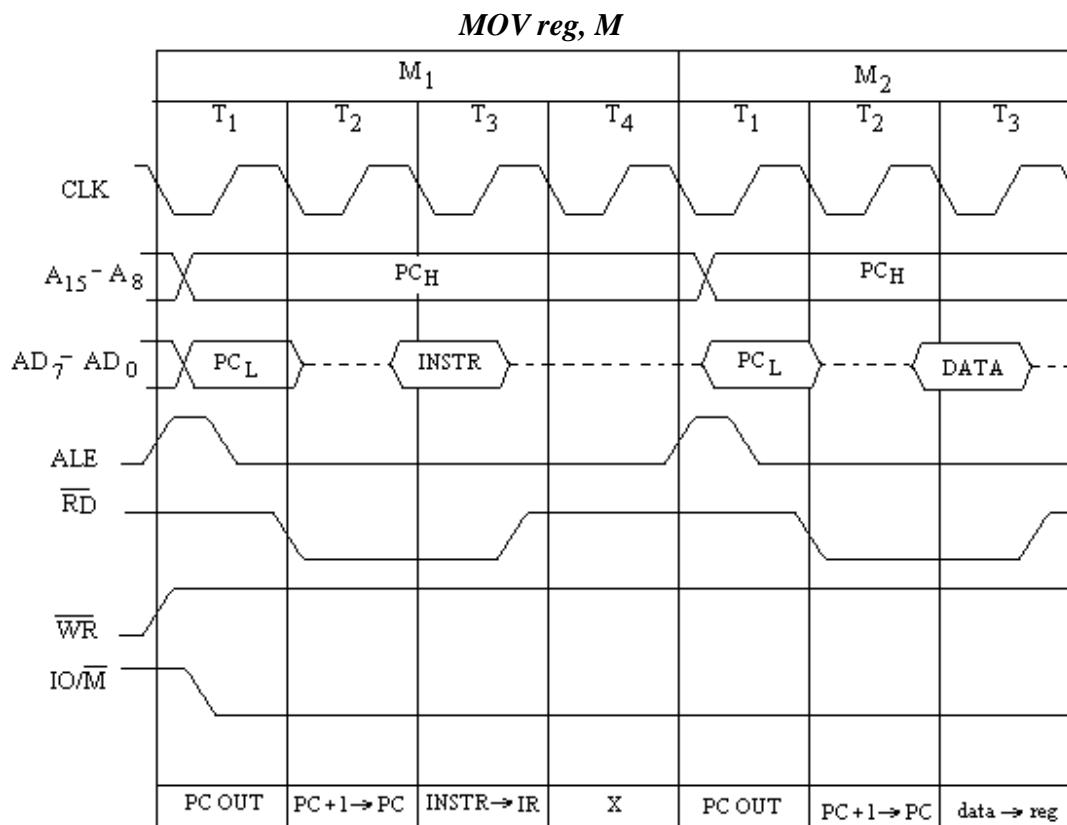
The timing diagram of the instruction

### ***MOV reg, M***

is shown in figure 5.9.

This is an indirect read instruction and takes two memory cycles ( $M_1$  and  $M_2$ ). The first machine cycle ( $M_1$ ) is known as instruction fetch cycle, during which the op code of the instruction is fetched from the memory. This machine cycle takes four T-states. The second machine cycle  $M_2$  is known as the execution cycle during which the data from the memory location addressed by H-L register pair is transferred to the given register. The second machine takes three states.

During the first T-state ( $T_1$ ) of  $M_1$  cycle, microprocessor sends the address of the memory location where the op code of the instruction ***MOV reg, M*** is stored, to the address lines. The high order byte of the PC (PCH) is placed on  $A_8-A_{15}$  lines and it stays on till  $T_4$ . The low order byte of PC (PCL) is placed on the address data lines ( $AD_0-AD_7$ ) which stays on only during  $T_1$ -state. For this purpose ALE (Address Latch Enable) signal gives a positive pulse midway through first T-state ( $T_1$ ), which latches the low order byte of the address into the memory chips. The  $IO/M$  signal goes low at the beginning of  $T_1$  state; this enables the peripheral chips for a memory operation rather than input/output operation. It is customary to represent the address lines by double sided waveforms as the address bits may be high or low (ref. fig. 5.9).



**Fig. 5.9**

During the second T-state ( $T_2$ ) of this op code fetch cycle, program counter (PC) is incremented ( $PC = PC + 1$ ). The address disappears from the address data bus ( $AD_7-AD_0$ ) at the beginning of  $T_2$  state. This is shown by dashed line indicating the data on the

bus is invalid or meaningless. At the beginning of  $T_2$  state  $\overline{RD}$  signal goes low and remains low till the middle of the  $T_3$  state.

During the third T-state ( $T_3$ ) of  $M_1$  machine cycle, the op code of the instruction is read out from the memory which is sent to the instruction register (IR) i.e.  $INSTR \rightarrow IR$ .

The fourth T-state ( $T_4$ ) of  $M_1$  machine cycle is denoted by  $X$  which indicates that this T-state is needed for the instruction decoding and other internal operations before the execution cycle.

Now the second machine cycle  $M_2$  known as execution cycle starts. During the first T-state of this execution cycle the contents of H-L register pair are placed on the address bus and address data bus. Basically, it performs the same operation as the  $T_1$ -state of  $M_1$  cycle. During  $T_2$  and  $T_3$  states of  $M_2$  machine cycle the data is read from the memory and copied in the specified register.

This completes the instruction ***MOV reg, M***. It may be noted that this instruction needs two machine cycles with 7 T-states.

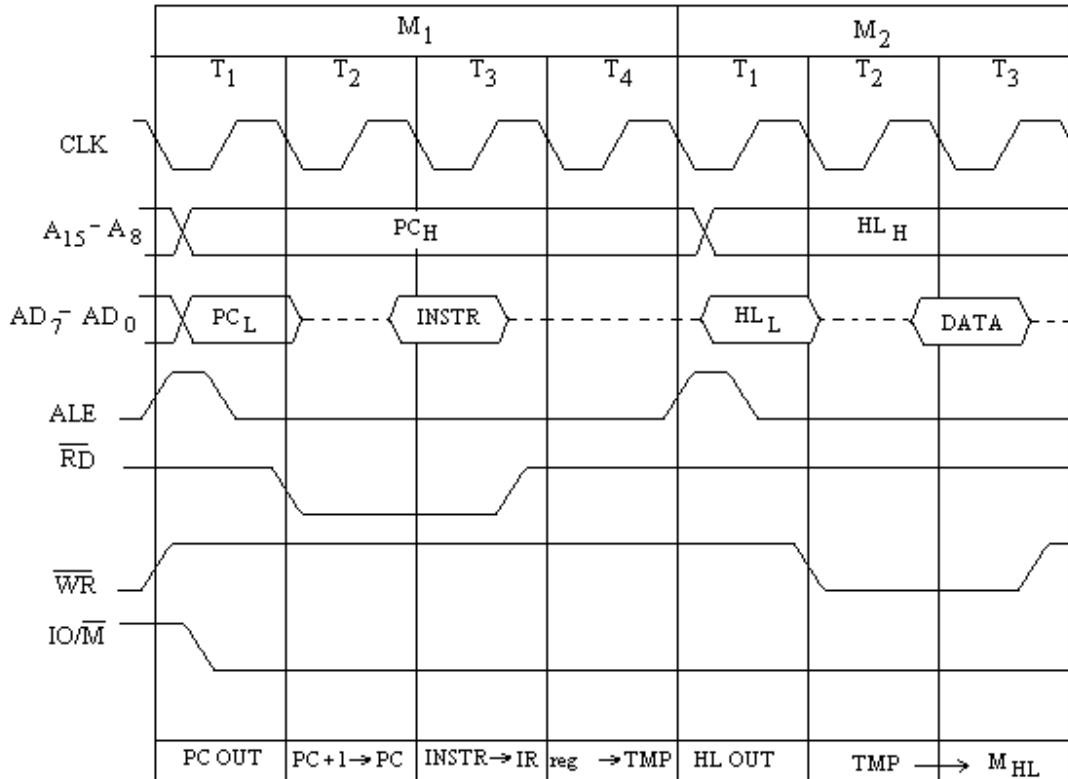
#### 5.4.2 Timing Diagram of ***MOV M, reg***

This is an indirect write instruction. The timing diagram of ***MOV M, reg*** instruction is shown in figure 5.10. The first three states of memory fetch cycle  $M_1$  are the same as for ***MOV reg, M***. During fourth T-state ( $T_4$ ) of  $M_1$ , the contents of specified register are copied in the temporary register.

During the  $T_1$ -state of second machine cycle  $M_2$ , the contents of H-L pair are placed on the address and address data bus. As usual during this state ALE sends a high pulse. During  $T_2$  and  $T_3$  states of machine cycle  $M_2$ , contents of temporary register are transferred (copied) to the specified address. Since it is memory write instruction, so at the beginning of  $T_2$ -state  $\overline{WR}$  signal activates (becomes low) and it remains low till half way through its  $T_3$ -state.

This instruction also takes two machine cycles with 7 T-states.

### ***MOV M, reg***

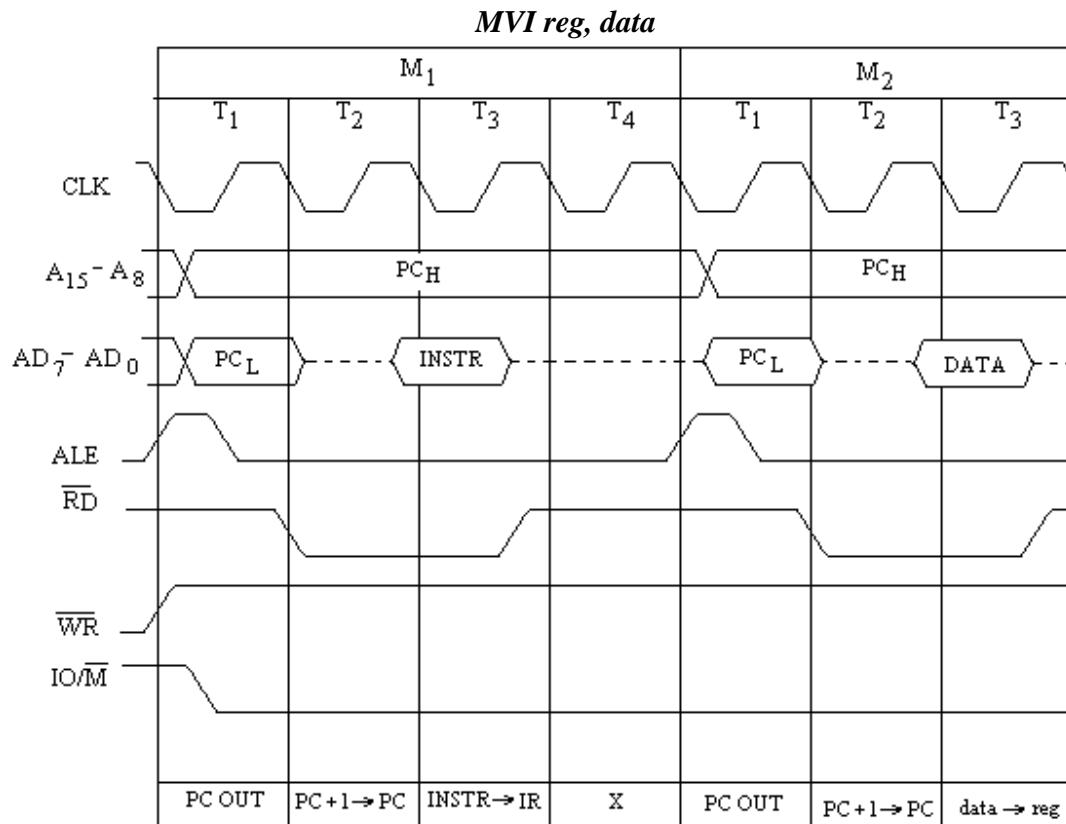


**Fig. 5.10**

#### **5.4.3 Timing Diagram of *MVI reg, data***

Figure 5.11 illustrates the timing diagram of the instruction ***MVI reg, data***. This is an immediate move instruction. It is two byte instruction; one byte is used for the op code of the instruction and the other byte is used for the data. During T<sub>1</sub>, T<sub>2</sub> states of op code fetch cycle M<sub>1</sub> the op code of the instruction (INSTR) is loaded in the instruction register (IR). The operation for these T-states is similar to other instructions discussed earlier. The T<sub>4</sub>-state of this cycle M<sub>1</sub> is for decoding.

In the second machine cycle M<sub>2</sub>, the contents of the program counter (PC) is placed on the address and address data buses during its T<sub>1</sub> state. This is the address of the second byte. During T<sub>2</sub> of M<sub>2</sub> cycle, the PC is incremented. At the same time memory is accessed and immediate read data is loaded to the specified register during T<sub>3</sub>-state of M<sub>2</sub> machine cycle. This instruction takes two machine cycles with 7 T-states.



**Fig. 5.11**

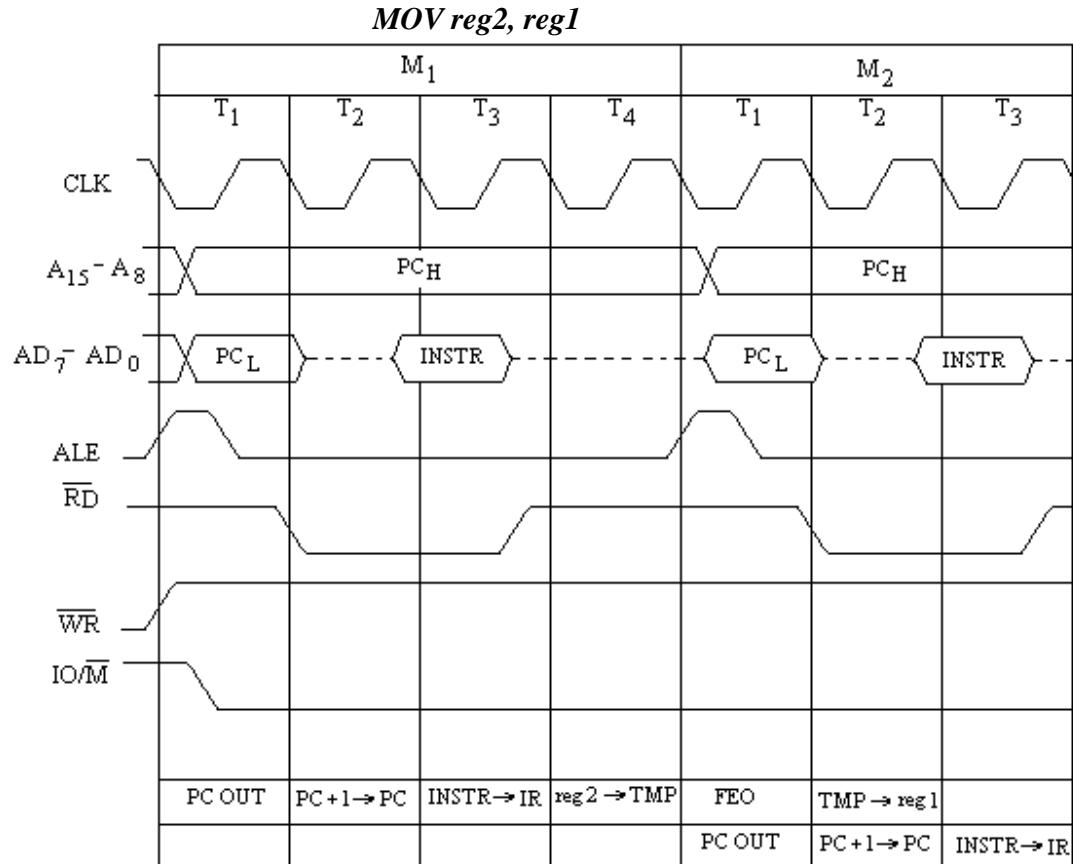
#### 5.4.4 Timing Diagram of **MOV reg2, reg1**

The timing diagram of **MOV reg2, reg1** is shown in figure 5.12. This instruction copies the contents of reg2 to reg1. So T<sub>1</sub>, T<sub>2</sub> and T<sub>3</sub> states of the op code fetch cycle M<sub>1</sub> as usual indicate PC OUT, increment state ( $PC + 1 \rightarrow PC$ ), copying of the op code on the instruction to the instruction register ( $INSTR \rightarrow IR$ ) operations respectively. Accordingly, ALE sends a high pulse at the beginning of T<sub>1</sub>-state,  $\overline{RD}$  activates (low) at the beginning of T<sub>2</sub> state till the middle of T<sub>3</sub> state till the middle of T<sub>3</sub> state of M<sub>1</sub> cycle. The  $IO/\overline{M}$  signal becomes low at the beginning of T<sub>1</sub> till the end of T<sub>3</sub> of M<sub>2</sub> machine cycle. During T<sub>4</sub> state the contents of reg2 is copied in Temporary register ( $reg2 \rightarrow TMP$ ).

This instruction does not need address bus and address data bus during the second machine cycle M<sub>2</sub>. The only thing to happen during M<sub>2</sub> cycle is the transfer of the contents of Temporary register to reg1, for which address and address data buses are not required. So time process starts i.e. during the second machine cycle M<sub>2</sub> of the instruction **MOV reg2, reg1**, fetching operation of the next instruction will take place. This is called Fetch Execute Overlap (FEO). Because of FEO, the new address of PC is placed on the address and address data buses during T<sub>1</sub> state of M<sub>2</sub> machine cycle. Thus at the trailing

edge of ALE signal, this address is latched into the memory chips. The contents of temporary register are copied into reg1 at the T2 state of this machine cycle. This completes the execution **MOV reg2, reg1** instruction. However at the same T-state, program counter is incremented. The op code of the next instruction is copied in IR during T<sub>3</sub>-state of M<sub>2</sub>.

It may be mentioned here that as shown in figure 5.12, this instruction takes two machine cycles with 7 T-states; but because of FEO only one machine cycle with 4 T-states are counted to fetch and execute this instruction.



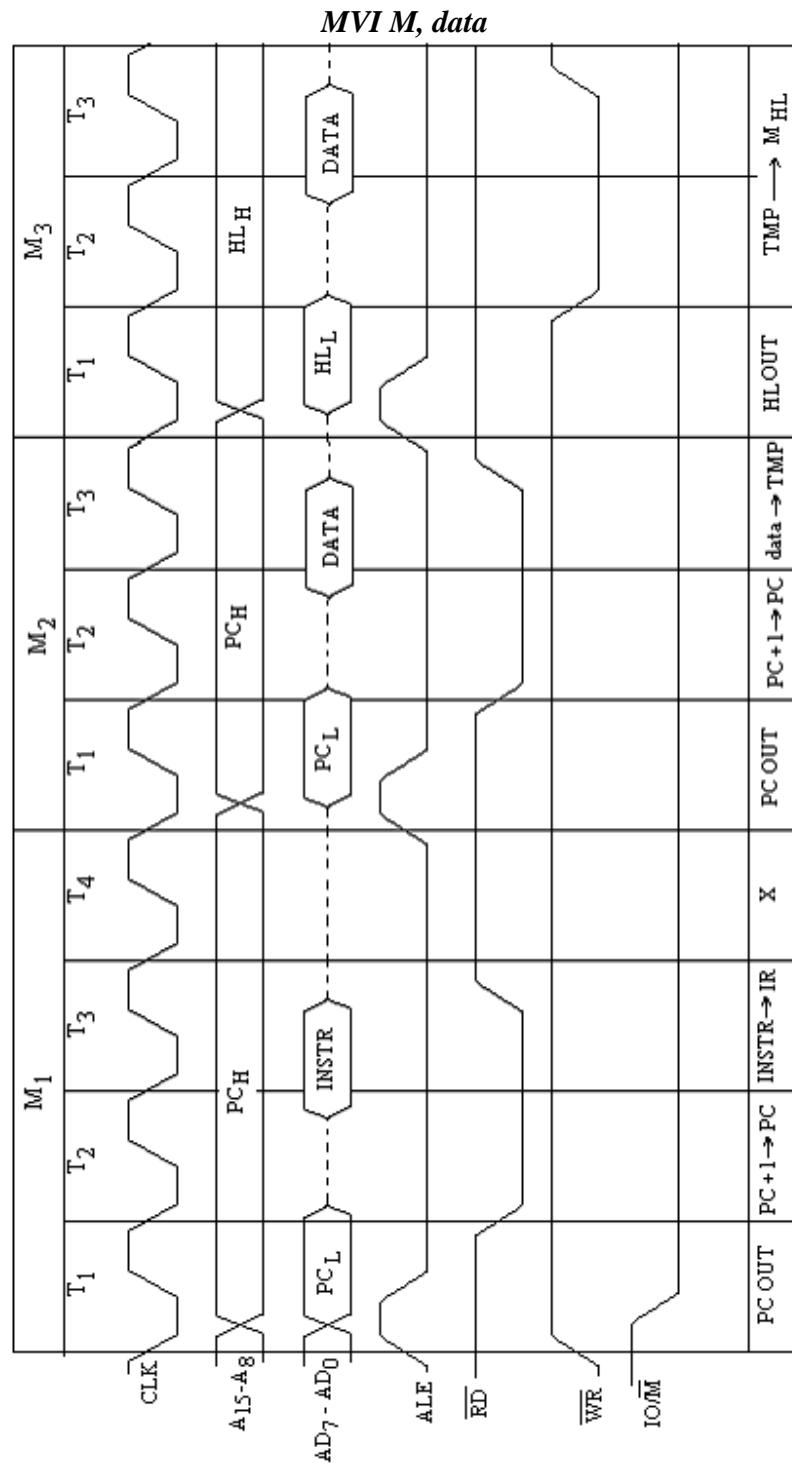
**Fig. 5.12**

#### 5.4.5 Timing Diagram of **MVI M, data**

The timing diagram of **MVI M, data** instruction is shown in figure 5.13. This is a two byte instruction which takes 3 machine cycles with 10 T-states. The machine cycle M<sub>1</sub> is the op code fetch cycle, whose operation has already been discussed. In the T<sub>1</sub> state of M<sub>2</sub> cycle, the incremented address of PC is latched into memory chips. The T<sub>2</sub>-state is the increment state and during the T<sub>3</sub> state of M<sub>2</sub> data is transferred to temporary register. This machine cycle is memory read cycle.

The third machine cycle is basically memory write cycle during which the contents of temporary register are copied in the memory location addressed by H-L register pair. In the T<sub>1</sub> state of the machine cycle M<sub>3</sub>, the contents of H-1 register pair is placed on the address bus and address data bus. It is then latched in the memory chips. In T<sub>2</sub> state the  $\overline{WR}$  signal is low till the middle of T<sub>3</sub> state of this cycle. So the contents of

temporary register are copied in  $M_{H-L}$  location during  $T_2$  and  $T_3$  state of  $M_3$  machine cycle.



**Fig. 5.13**

#### 5.4.6 Timing Diagram of XCHG

Let us discuss the timing diagram of one byte instruction XCHG. Its timing diagram is shown in figure 5.14. It exchanges the contents of H-L register pair with contents of D-E register pair.

The operation of  $M_1$  machine cycle of this instruction is similar to other instructions discussed above. In this instruction the contents of H-L and D-E register pairs are to be exchanged, so address bus and address data bus are not needed. For this the operation FEO (Fetch Execute Overlap) occurs and as usual the next instruction is fetched from the memory for the  $M_2$  machine cycle. However, during  $T_2$  state of this machine cycle the contents of H-E and D-E pairs are exchanged ( $H-L \leftrightarrow D-E$ ).

Because of FEO, XCHG instruction is considered to take one machine cycle with 4 T-states.

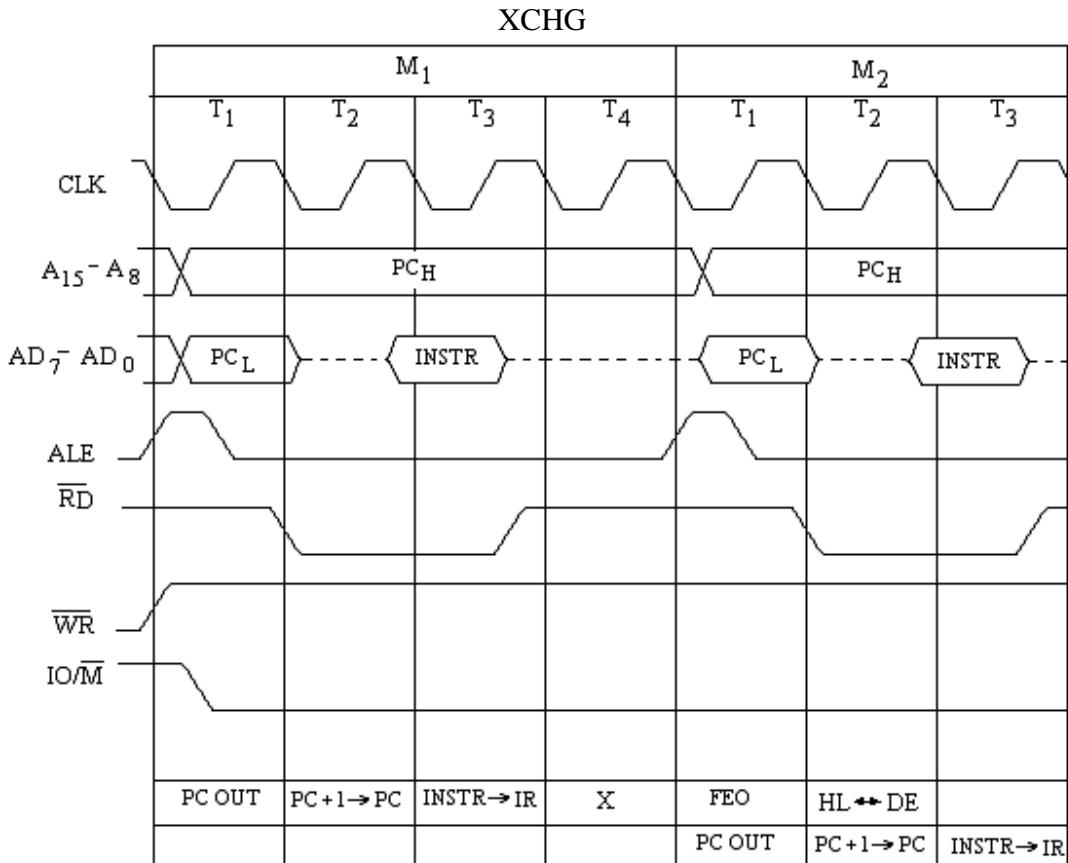
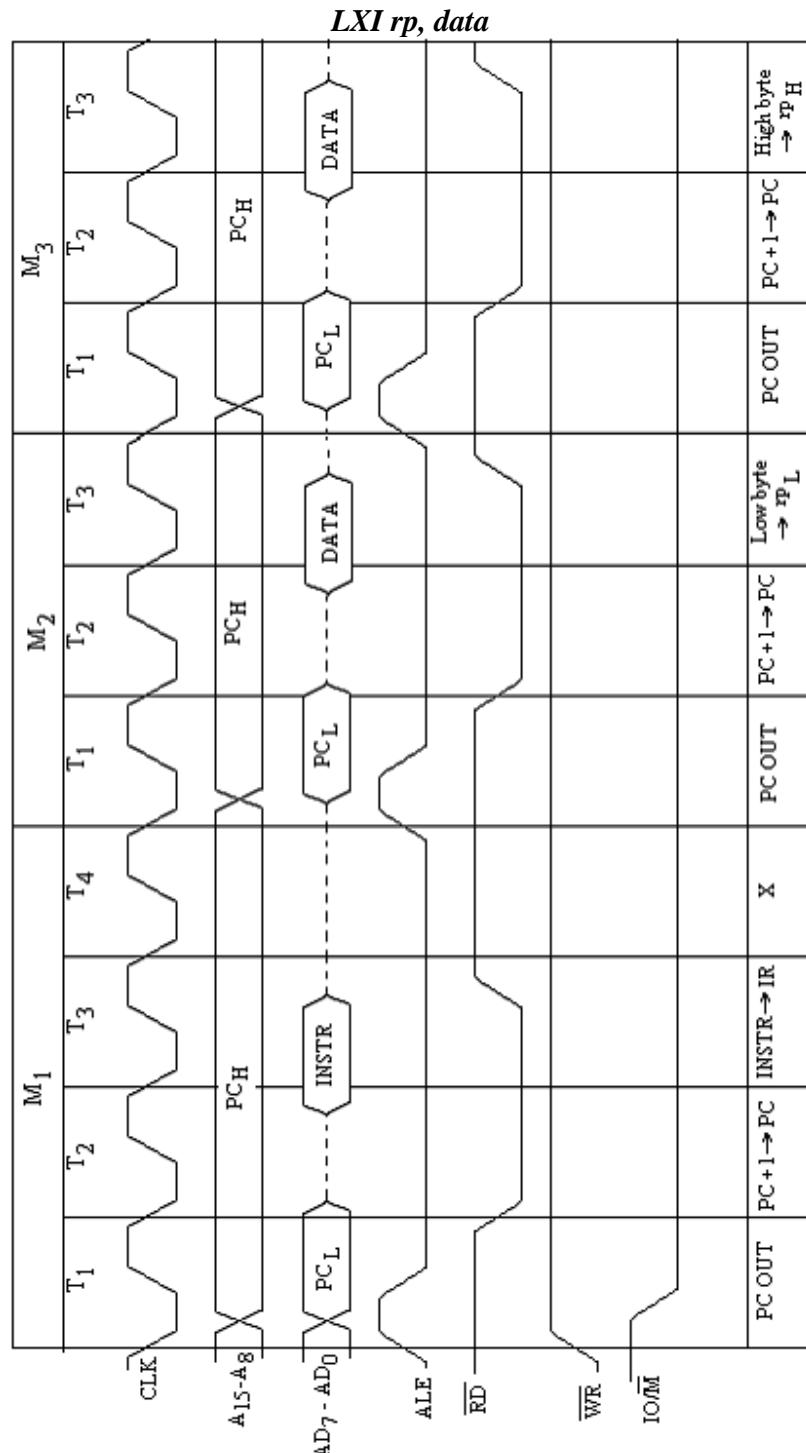


Fig. 5.14

#### 5.4.7 Timing Diagram of LXI rp, dbyte

The timing diagram of 3-byte instruction **LXI rp, dbyte** is shown in figure 5.15. It loads the low byte of dbyte (double byte data) to the lower register of the given register pair *rp* and high byte to the high register of *rp*. This instruction takes three memory cycles with 10 T-states. First machine cycle is the op code fetch cycle which fetches the

op code of the instruction using the same procedure as discussed above for the other examples.



**Fig. 5.15**

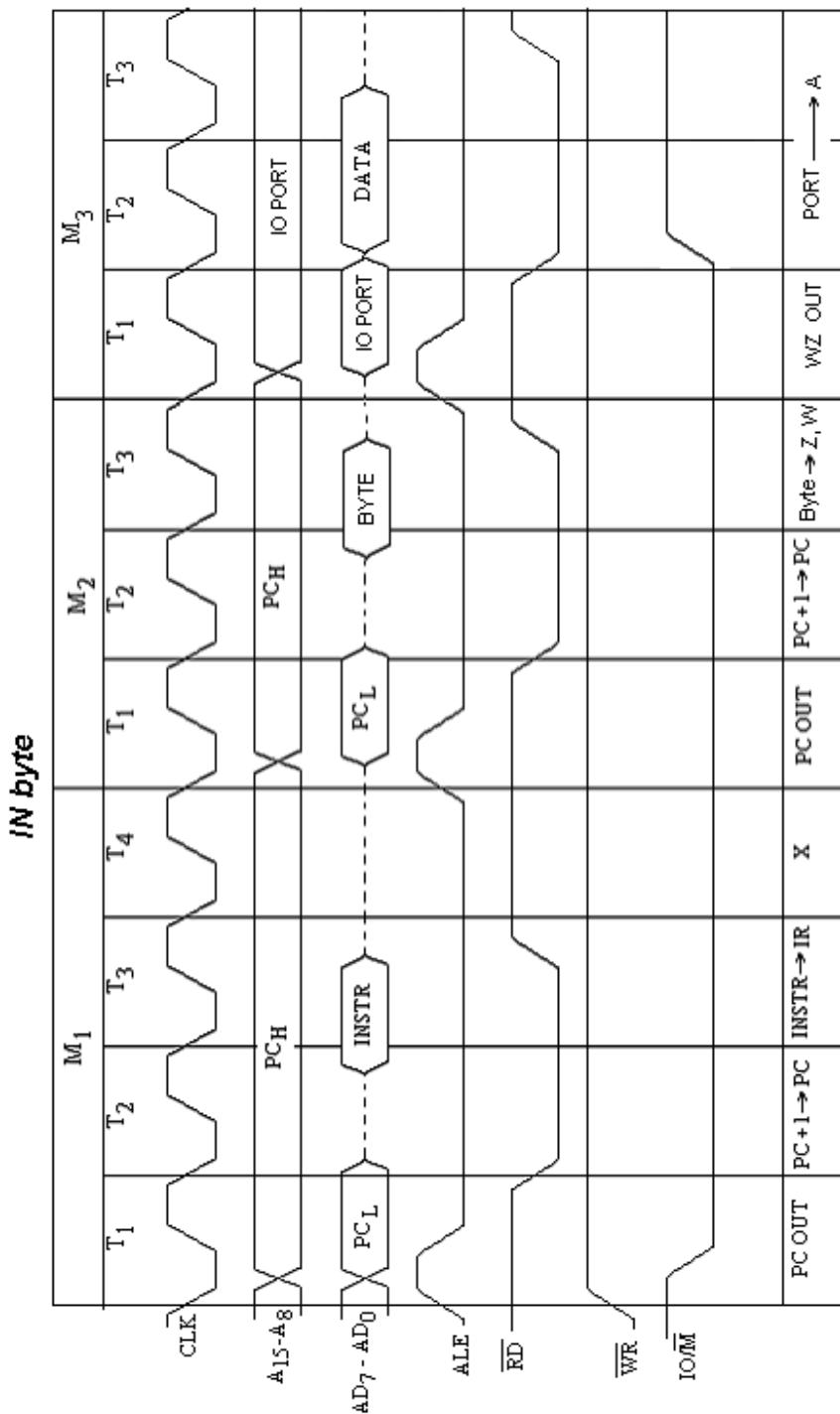
In the  $T_1$ -state of the second machine cycle, address bus and address data bus are used to fetch the second byte (low byte of dbyte), for which ALE is enable at the beginning to the middle of this T-state. During  $T_2$  state of  $M_2$ , PC is incremented and at  $T_3$ -state low byte of the dbyte is stored in lower register of  $rp$  (Low byte  $\rightarrow rp_L$ ). This is a read cycle so  $\overline{RD}$  is low at the beginning of this T-state to the middle of  $T_3$  state. Same operation as for  $M_2$  is, therefore, performed during  $M_3$  machine cycle so that high byte of dbyte is stored in high register of register pair  $rp$  (High byte  $\rightarrow rp_H$ ).

#### 5.4.8 Timing Diagram of *IN* byte

The timing diagram of *IN byte* shown in figure 5.16 will now be discussed. This is an I/O read cycle and the microprocessor reads the data available at an input port or input device. The address of the port is of one byte given with the instruction. The data read out from the output port will be placed in the accumulator (A). This instruction is a two byte long and takes three memory cycles with 10 T-states. First machine cycle is an op code fetch cycle, the second is memory read cycle and the third is output read cycle.

We are familiar with the op code fetch cycle  $M_1$ . So the discussion will be made for the second and third machine cycles. In the  $T_1$ -state of second machine cycle  $M_2$ , the address of PC (incremented address) will be latched into the memory chips. At the beginning of  $T_1$  state ALE sends a high pulse. During the  $T_2$ -state PC is incremented ( $PC + 1 \rightarrow PC$ ). Near the end of  $T_2$ -state of  $M_2$ , a byte appears on address data bus. During the  $T_3$ -state of  $M_2$  machine cycle, this byte is copied in the W and Z registers (byte  $\rightarrow Z, W$ ), i.e. W and Z registers have the same byte. This byte is port address which is of 8 bits.

During  $T_1$ -state of the third machine cycle  $M_3$ ,  $IO/M$  signal goes high which enables the peripheral chips for an I/O operation rather than memory operation. During this T-state, the contents of W-register are placed on the address bus and the contents of Z-register on address data bus (WZ OUT). In case of I/O device or I/O port the address is only of 8 bit long and therefore, the address of I/O device is duplicated on both address bus and address data bus. During  $T_2$ -state of  $M_3$  cycle,  $\overline{RD}$  signal goes low which indicates that it is I/O read operation. The data read out from the input port (input device) will be copied in the accumulator during  $T_2$  and  $T_3$  states.



**Fig. 5.16**

## PROBLEMS

1. Draw and discuss the architecture of 8085 microprocessor.
2. Mention various flags provided in 8085 microprocessor and discuss their roles.
3. Discuss the functions of program counter, Stack pointer and status flags in the architecture of 8085 microprocessor.
4. Discuss the role of address buffer and address data buffer in the architecture of 8085 microprocessor.
5. Discuss the functions of the following signals of 8085 microprocessor:  
ALE,  $\overline{WR}$ ,  $\overline{RD}$ ,  $S_0$  and  $S_1$ .
6. Discuss *LHLD address*, *SHLD address*, *LDAX rp* and *STAX rp* instructions of 8085.
7. Discuss the following instructions of 8085  
**XCHG, XTHL, SPHL and PCHL**  
Which flags are affected by these instructions?
8. Discuss DAA instructions of 8085. Explain how flags are affected with this instruction.
9. Draw and discuss the timing diagram of *MOV reg2, reg1*.
10. Draw and discuss the timing diagram of *MOV reg, M*.
11. Draw and discuss the timing diagram of *MOV M, reg*.
12. Draw and discuss the timing diagram of *MVI reg, data*.
13. Draw and discuss the timing diagram of *MVI M, data*.
14. Draw and discuss the timing diagram of *LXI rp, dbyte*.
15. Draw and discuss the timing diagram of *XCHG*.
16. Draw and discuss the timing diagram of *IN byte*.
17. Discuss *LDAX rp* instruction of 8085. What instruction should be used to move the contents stored in memory location whose address is stored in H-L register pair.
18. What instruction should be used to store the accumulator contents to the memory location addressed by D-E register pair? Discuss that instruction in detail.
19. Discuss PCHL instruction of 8085. Explain how this instruction is said to be used as unconditional jump instruction.
20. What will be contents of accumulator and flag register, after the execution of following program:

MVI A, 47 H  
MVI B, 37 H  
ADD B  
DAA  
HLT

---

# 6

## Programming of 8085

---

In this chapter assembly language programming of different programs will be discussed. It will help readers to go into the details of 8085 programming operations and their applications. For this, one is supposed to be well acquainted with 8085 instruction set discussed in the preceding chapters. Though some programming examples have also been discussed in these chapters, yet this chapter will exclusively deal with some more programming examples. The assembly language program written by the programmer in mnemonic form is also called as source program. The instructions, operand and data may be converted to binary (machine language) either by hand assembler or machine assembler. In case of hand assembler the hexadecimal data (op code / operand and data) are fed to the microprocessor kit through the hexadecimal key board. But in machine assembler, instructions (in mnemonic form) with data or operand are directly fed to the microprocessor kit through Alpha-numeric key board (computer key board). Here we are concerned with the assembly language program (source program).

In computers high level language like C, C++, Pascal, BASIC etc are used, which are easier than the assembly language. The computer is used for the conversion of high level language to assembly language. The advantage of using assembly language is that it can directly go into the details of the microprocessor's register and manipulate as the requirement.

### 6.1 SIMPLE PROGAMS

The simple programs based on the instruction set of 8085 microprocessor are given in the following examples.

**Example 6.1.** Write an assembly language program of 8085 to find 1's complement of the data stored in memory location 2500 H and the result is to be stored in memory location 2501 H.

**Solution.**

**Program:**

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
|       | LDA       | 2500 H  | ; Load the data from 2500 H memory location to accumulator.            |
|       | CMA       |         | ; Complement the contents of accumulator (Converts in 1's complement). |
|       | STA       | 2501 H  | ; Store the result in memory location 2501 H.                          |
|       | HLT       |         | ; Stop processing.   |

**Example 6.2.** Write an assembly language program of 8085 to find 2's complement of the data stored in memory location 2100 H and the result is to be stored in memory location 2101 H.

**Solution.**

**Program:**

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
|       | LDA       | 2100 H  | ; Load the data from 2100 H memory location to accumulator.            |
|       | CMA       |         | ; Complement the contents of accumulator (Converts in 1's complement). |
|       | ADI       | 01 H    | ; 01 is added to the accumulator contents to get the 2's complement.   |
|       | STA       | 2101 H  | ; Store the result in memory location 2101 H.                          |
|       | HLT       |         | ; Stop processing.   |

**Example 6.3.** Write an assembly language program of 8085 to find 1's complement of  $n$  (decimal number) bytes of data stored in memory location starting at 2501 H. The number  $n$  (decimal number) is stored in memory location 2500 H. Store the result in memory locations starting at 2601 H.

**Solution.**

**Program:**

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
|       | LXI D,    | 2601 H  | ; Get first address of the destination in D-E register pair.            |
|       | LXI H,    | 2500 H  | ; Get H-L pair with 2500 H.   |
|       | MOV C,    | M       | ; Data in 2500 H is loaded to C-register which will be used as counter. |
|       | INXH      |         | ; Increment H-L register pair.  |
| LOOP  | MOV A,    | M       | ; Accumulator is loaded with the data.                                  |
|       | CMA       |         | ; Complement the contents of accumulator (Converts in 1's complement)   |
|       | STAX D    |         | ; Store the result in memory location addressed by D-E register pair.   |
|       | INX D     |         | ; Increment D-E register pair.  |
|       | INX H     |         | ; Increment H-L register pair.  |
|       | DCR C     |         | ; Decrement the C-register data.  |
|       | JNZ       | LOOP    | ; If data in C-reg. is not zero jump to LOOP for next conversion.       |
|       | HLT       |         | ; Stop processing.  |

**Example 6.4.** Write an assembly language program of 8085 to find 9's complement of  $n$  bytes (in BCD) of data stored in memory location starting from 2501 H. The number  $n$

(decimal number) is stored in memory location 2500 H. Store the result in memory locations starting from 2601 H.

### Solution.

#### Program:

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
|       | LXI D,    | 2601 H  | ; Get first address of the destination in D-E register pair.            |
|       | LXI H,    | 2500 H  | ; Get H-L pair with 2500 H.   |
|       | MOV C,    | M       | ; Data in 2500 H is loaded to C-register which will be used as counter. |
|       | INX H     |         | ; Increment H-L register pair.  |
| LOOP  | MVI A,    | 99 H    | ; Get 99 H in accumulator.  |
|       | SUB M     |         | ; Subtract each byte by 99 H to get 9's complement.                     |
|       | STAX D    |         | ; Store the result in memory location addressed by D-E register pair.   |
|       | INX D     |         | ; Increment D-E register pair.  |
|       | INX H     |         | ; Increment H-L register pair.  |
|       | DCR C     |         | ; Decrement the C-register data.  |
|       | JNZ       | LOOP    | ; If data in C-reg. is not zero jump to loop for next conversion.       |
|       | HLT       |         | ; Stop processing.  |

It is clear from this example that for 9's complement each BCD byte is subtracted from 99.

**Example 6.5.** Write an assembly language program of 8085 to find 2's complement of a 16 bit number stored in memory locations 2101 H and 2102 H. The least significant byte is in 2101 H. The result is to be stored in memory locations 2103 H and 2104 H.

### Solution.

#### Program:

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
|       | LXI H,    | 2101 H  | ; Get H-L pair with 2101 H.  |
|       | MVI B,    | 00 H    | ; Store 00 to register B which will be used to store carry.            |
|       | MOV A,    | M       | ; Accumulator is loaded with the data.                                 |
|       | CMA       |         | ; Complement the contents of accumulator (Converts in 1's complement). |
|       | ADI       | 01 H    | ; Add 01 H to the accumulator content to get 2's complement.           |
|       | STA       | 2103 H  | ; Store the result in memory location 2103 H.                          |
|       | JNC       | NXT     | ; If there is no carry jump to NXT.                                    |
|       | INR B     |         | ; Increment 1 to B-register as carry.                                  |

|     |        |        |  |
|-----|--------|--------|--|
| NXT | INX H  |        | ; Increment H-L register pair for the second byte. |
|     | MOV A, | M      | ; Move the second byte to accumulator.             |
|     | CMA    |        | ; Complement the contents of accumulator.          |
|     | ADD B  |        | ; Add carry stored in register B.                  |
|     | STA    | 2504 H | ; Store in 2504 H.                                 |
|     | HLT    |        | ; Stop processing.                                 |

From this example it is clear that 1 is added to the 1's complement of LS byte and to the 1's complement of MS byte 1 is added if there is a carry from the previous byte.

**Example 6.6.** Write an assembly language program of 8085 to find 10's complement of a 16 bit number (BCD) stored in memory locations 2101 H and 2102 H. The least significant byte is in 2101 H. The result is to be stored in memory locations 2103 H and 2104 H.

**Solution.**

**Program:**

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
| NXT   | LXI H,    | 2101 H  | ; Get H-L pair with 2101 H.                                   |
|       | MVI B,    | 00 H    | ; Store 00 to register B which will be used to store carry.   |
|       | MVI A,    | 99 H    | ; Store 99 in accumulator                                     |
|       | SUB M     |         | ; Subtract each byte by 99 H to get 9's complement.           |
|       | ADI       | 01 H    | ; Add 01 H to the accumulator content to get 10's complement. |
|       | STA       | 2103 H  | ; Store the result in memory location 2103 H.                 |
|       | JNC       | NXT     | ; If there is no carry jump to NXT.                           |
|       | INR B     |         | ; Increment 1 to B-register as carry.                         |
|       | INX H     |         | ; Increment H-L register pair for the second byte.            |
|       | MVI A,    | 99 H    | ; Store 99 in accumulator                                     |
|       | SUB M     |         | ; Subtract each byte by 99 H to get 9's complement.           |
|       | ADD B     |         | ; Add carry stored in register B.                             |
|       | STA       | 2104 H  | ; Store in 2104 H.  |
|       | HLT       |         | ; Stop processing.  |

From this example it is clear that 1 is added to the 9's complement of LS byte and to the 9's complement of MS byte 1 is added if there is a carry from the previous byte.

**Example 6.7.** Write an assembly language program of 8085 to find 2's complement of N bytes ( $N \geq 2$ ). The number of bytes N in hexadecimal is stored in 2100 H. The bytes are stored in memory locations starting at 2101 H. The least significant byte is in 2101 H. The result is to be stored in memory locations starting at 2201 H.

### Solution.

#### Program:

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>  |
|--------------|------------------|----------------|--|
|              | LXI H,           | 2100 H         | ; Get H-L pair with 2100 H.  |
|              | LXI D,           | 2201 H         | ; Get D-E pair with 2201 H.  |
|              | MOV C,           | M              | ; Get the number N in c-register to be used as counter.                |
|              | INX H            |                | ; Increment H-L pair.  |
|              | MOV A,           | M              | ; Accumulator is loaded with first data.                               |
|              | CMA              |                | ; Complement the contents of accumulator (Converts in 1's complement). |
|              | ADI              | 01 H           | ; Add 01 H to the accumulator content to get 2's complement.           |
|              | STAX D           |                | ; Store the result in memory location addressed by D-E pair.           |
| NXT          | JNC              | NXT            | ; If there is no carry jump to NXT.                                    |
|              | MVI B,           | 01 H           | ; Store 01 to B-register as carry.                                     |
|              | JMP              | NXT1           | ; Jump to NXT1.  |
| NXT1         | MVI B,           | 00 H           | ; Store 00 to B-register as carry.                                     |
|              | DCR C            |                | ; Decrement C-reg.   |
| LOOP         | INX H            |                | ; Increment H-L pair.  |
|              | INX D            |                | ; Increment D-E pair.  |
|              | MOV A,           | M              | ; Move the next byte to accumulator.                                   |
|              | CMA              |                | ; Complement the contents of accumulator.                              |
|              | ADD B            |                | ; Add carry stored in register B.                                      |
|              | STAX D           |                | ; Store in the memory location addressed by D-E pair.                  |
| NXT2         | JNC              | NXT2           | ; If there is no carry jump to NXT2.                                   |
|              | MVI B,           | 01 H           | ; Store 01 to B-register as carry.                                     |
|              | JMP              | NXT3           | ; Jump to NXT3.  |
| NXT3         | MVI B,           | 00 H           | ; Store 00 to b-register as carry.                                     |
|              | DCR C            |                | ; Decrement C-reg.   |
|              | JNZ              | LOOP           | ; If not zero jump to LOOP.  |
|              | HLT              |                | ; Stop processing.   |

**Example 6.8.** Write an assembly language program of 8085 to find 10's complement of N bytes ( $N \geq 2$ ). The number of bytes N in hexadecimal is stored in 2100 H. The bytes are stored in memory locations starting at 2101 H. The least significant byte is in 2101 H. The result is to be stored in memory locations starting at 2201 H.

### Solution.

#### Program:

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>             |
|--------------|------------------|----------------|-----------------------------|
|              | LXI H,           | 2100 H         | ; Get H-L pair with 2100 H. |
|              | LXI D,           | 2201 H         | ; Get D-E pair with 2201 H. |

|      |        |      |  |
|------|--------|------|--|
|      | MOV C, | M    | ; Get the number N in c-register to be used as counter.                          |
|      | INX H  |      | ; Increment H-L pair.  |
|      | MVI A, | 99 H | ; Accumulator is loaded with 99 H.   |
|      | SUB M  |      | ; Get the 9's Complement of the number stored in location addressed by H-L pair. |
|      | ADI    | 01 H | ; Add 01 H to the accumulator content to get 2's complement.                     |
|      | STAX D |      | ; Store the result in memory location addressed by D-E pair.                     |
|      | JNC    | NXT  | ; If there is no carry jump to NXT.  |
|      | MVI B, | 01 H | ; Store 01 to B-register as carry.   |
|      | JMP    | NXT1 | ; Jump to NXT1.  |
| NXT  | MVI B, | 00 H | ; Store 00 to b-register as carry.   |
| NXT1 | DCR C  |      | ; Decrement C-reg.   |
| LOOP | INX H  |      | ; Increment H-L pair.  |
|      | INX D  |      | ; Increment D-E pair.  |
|      | MVI A, | 99 H | ; Get 99 in accumulator.   |
|      | SUB M  |      | ; Get 9's Complement of the contents of accumulator.                             |
|      | ADD B  |      | ; Add carry stored in register B to get 10's complement.                         |
|      | STAX D |      | ; Store in the memory location addressed by D-E pair.                            |
|      | JNC    | NXT2 | ; If there is no carry jump to NXT2.   |
|      | MVI B, | 01 H | ; Store 01 to B-register as carry.   |
|      | JMP    | NXT3 | ; Jump to NXT3.  |
| NXT2 | MVI B, | 00 H | ; Store 00 to b-register as carry.   |
| NXT3 | DCR C  |      | ; Decrement C-reg.   |
|      | JNZ    | LOOP | ; If not zero jump to LOOP.  |
|      | HLT    |      | ; Stop processing.   |

**Example 6.9.** Write an assembly language program of 8085 to combine two hex nibbles stored in 2500 H and 2501 H memory locations, to form a byte. The least significant nibble is stored in 2500 H and most significant nibble is stored in 2501 H. The byte thus combined should be stored in 2502 H. (Let 08 H is stored in 2500 H and 09 H is stored in 2501 H, after the program is executed 2502 H should be loaded with the combined byte 89 H).

**Solution.**

**Program:**

| Label | Mnemonics | Operand | Comments                                   |
|-------|-----------|---------|--|
|       | LXI H,    | 2500 H  | ; Get H-L pair with 2500 H.                |
|       | MOV A,    | M       | ; Data in 2500 H is loaded to Accumulator. |
|       | RLC       |         | ; Rotate left                              |

|          |  |   |
|----------|--|---|
| RLC      |  | ; Rotate left   |
| RLC      |  | ; Rotate left   |
| RLC      |  | ; Rotate left (Rotated left four times so that it is moved to MS nibble).                             |
| INX H    |  | ; Increment H-L register pair.  |
| ORA M    |  | ; Oring of two nibbles combines them to form a byte in accumulator.                                   |
| INX H    |  | ; Increment H-L register pair.  |
| MOV M, A |  | ; Move the accumulator content (required byte) to the memory location addressed by H-L register pair. |
| HLT      |  | ; Stop processing.  |

**Example 6.10.** Write an assembly language program of 8085 to separate a hexadecimal number into two nibbles. The hexadecimal number is stored in 2501 H memory location. The least significant nibble of the byte is to be stored in 2502 memory location and the most significant nibble is to be stored in 2503 H memory location. (Suppose 3A H is a byte stored in memory location 2501 H and the lower nibble 0A should be stored in 2502 H and 03 should be stored in 2503 H)

**Solution.**

**Program:**

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
|       | LXI H,    | 2501 H  | ; Get H-L pair with 2501 H.  |
|       | MOV A,    | M       | ; Data in 2501 H is loaded to Accumulator.                                       |
|       | MOV B,    | A       | ; Accumulator content are also loaded to B register.                             |
|       | ANI       | 0F H    | ; Mask off the first four digits (higher nibble).                                |
|       | INX H     |         | ; Increment H-L register pair.   |
|       | MOV M,    | A       | ; Lower nibble is loaded to the memory location addressed by H-L register pair.  |
|       | MOV A,    | B       | ; Get the byte again in the accumulator.   |
|       | ANI       | F0 H    | ; Mask off the lower nibble.   |
|       | INX H     |         | ; Increment H-L register pair.   |
|       | MOV M,    | A       | ; Higher nibble is loaded to the memory location addressed by H-L register pair. |
|       | HLT       |         | ; Stop processing.   |

**Example 6.11.** Write an assembly language program of 8085 to check a hexadecimal number stored in 2500 H memory location for odd or even parity. If the parity is even store data EE H to memory location 2501H otherwise store 00 H in 2501 H.

**Solution.**

**Program:**

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
|       | LXI H,    | 2500 H  | ; Get H-L pair with 2500 H.  |
|       | MOV A,    | M       | ; Data in 2500 H is loaded to Accumulator.                         |
|       | ORA A     |         | ; Set the flag.  |
|       | JPE       | EVEN    | ; Check for even parity, if parity is even jump to EVEN.           |
|       | INX H     |         | ; Increment H-L register pair.                                     |
|       | MVI M,    | 00 H    | ; Load 00 H to the memory location addressed by H-L register pair. |
|       | JMP       | DONE    | ; Jump to DONE.  |
| EVEN  | INX H     |         | ; Increment H-L register pair.                                     |
|       | MVI M,    | EE H    | ; Load EE H to the memory location addressed by H-L register pair. |
| DONE  | HLT       |         | ; Stop processing.   |

**Example 6.12.** n (decimal number) data bytes are stored in the memory locations starting at 2501 H. Write an assembly language program of 8085 to check if 12 H is stored in any of the given locations. If any of the locations has 12 H then store 12 H in that location else load 00 H in that memory location.

**Solution.**

**Program:**

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
|       | LXI H,    | 2500 H  | ; Get H-L pair with 2500 H.   |
|       | MOV C,    | M       | ; Data in 2500 H is loaded to C reg. which is used as the counter.      |
| LOOP  | INX H     |         | ; Increment H-L register pair.  |
|       | MOV A,    | M       | ; Load the content of M <sub>HL</sub> to accumulator.                   |
|       | CPI       | 12 H    | ; Compare if the accumulator data are 12 H.                             |
|       | JZ        | NXT     | ; If it is 12 H then jump to NXT.                                       |
|       | MVI M,    | 00 H    | ; Else load 00 H to the memory location addressed by H-L register pair. |
|       | JMP       | DONE    | ; Jump to DONE.   |
| NXT   | MVI M,    | 12 H    | ; Load 12 H to the memory location addressed by H-L register pair.      |

|      |       |  |
|------|-------|--|
| DONE | DCR C | ; Decrement the content of C-reg.<br>for next byte.            |
| JNZ  | LOOP  | ; If the contents of C-reg. are not<br>zero then jump to LOOP. |
| HLT  |       | ; Stop processing.   |

**Example 6.13.** 16 data bytes are stored in memory locations 2001 H to 2010 H. Write an assembly language program of 8085 to transfer this block of data bytes to memory locations 2501 to 2510 H.

**Solution.**

**Program:**

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
|       | LXI H,    | 2001 H  | ; Get H-L pair with 2001 H.   |
|       | LXI D,    | 2501 H  | ; Get the address of the memory<br>location (destination) in D-E<br>register pair               |
|       | MVI C,    | 10 H    | ; Get 10 H ( $16_{10}$ ) Data in C register<br>which is used as the counter.                    |
| LOOP  | MOV A,    | M       | ; Load the content of M <sub>HL</sub> to<br>accumulator.  |
|       | STAX D    |         | ; Load the content of accumulator to<br>the memory locations addressed by<br>D-E register pair. |
|       | INX H     |         | ; Increment H-L register pair.  |
|       | INX D     |         | ; Increment D-E register pair.  |
|       | DCR C     |         | ; Decrement the content of C-reg.<br>for next byte.   |
|       | JNZ       | LOOP    | ; If the contents of C-reg. are not<br>zero then jump to LOOP.                                  |
|       | HLT       |         | ; Stop processing.  |

**Example 6.14.** 16 data bytes are stored in memory locations 2001 H to 2010 H. Write an assembly language program of 8085 to transfer this block of data bytes to memory locations 2501 to 2510 H in the reverse order (i.e. the data of 2001 is to transferred to 2510 H and data of 2002 H to 250F H and so on).

**Solution.**

**Program:**

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
|       | LXI H,    | 2001 H  | ; Get H-L pair with 2001 H.   |
|       | LXI D,    | 2510 H  | ; Get the address of the memory<br>location (destination) in D-E<br>register pair |
|       | MVI C,    | 10 H    | ; Get 10 H ( $16_{10}$ ) Data in C register<br>which is used as the counter.      |

|      |        |      |   |
|------|--------|------|---|
| LOOP | MOV A, | M    | ; Load the content of M <sub>HL</sub> to accumulator.                                     |
|      | STAX D |      | ; Load the content of accumulator to the memory locations addressed by D-E register pair. |
|      | INX H  |      | ; Increment H-L register pair.  |
|      | DCX D  |      | ; Decrement D-E register pair.  |
|      | DCR C  |      | ; Decrement the content of C-reg. for next byte.  |
|      | JNZ    | LOOP | ; If the contents of C-reg. are not zero then jump to LOOP.                               |
|      | HLT    |      | ; Stop processing.  |

**Example 6.15.** Write an assembly language program of 8085 to clear the memory locations starting at 2001 (i.e. each location should have 00 H). The length of data bytes is given in 2000 H memory location.

**Solution.**

**Program:**

| Label | Mnemonics   | Operand          | Comments  |
|-------|---|------------------|---|
| LOOP  | LXI H,<br>MOV C,<br>XRA A<br>INX H<br>MOV M,<br>DCR C | 2000 H<br>M<br>A | ; Get H-L pair with 2000 H.<br>; Use C register as counter.<br>; Clear the accumulator.<br>; Increment H-L register pair.<br>; Load the accumulator content to the memory location.<br>; Decrement the content of C-reg. for next byte. |
|       | JNZ   | LOOP             | ; If the contents of C-reg. are not zero then jump to LOOP.   |
|       | HLT   |                  | ; Stop processing.  |

**Example 6.16.** Write an assembly language program of 8085 to add five bytes of data stored in memory locations starting at 2000 H. If the sum generates a carry, stop addition and store 01 H to the memory location 2501 H; else continue addition and store the sum at 2501 memory location..

**Solution.**

**Program:**

| Label | Mnemonics                          | Operand        | Comments  |
|-------|------------------------------------|----------------|---|
| LOOP  | LXI H,<br>MVI C,<br>XRA A<br>ADD M | 2000 H<br>05 H | ; Get H-L pair with 2000 H.<br>; Store the number of data bytes in C register as counter.<br>; Clear the accumulator and CY flag.<br>; Add the byte in accumulator. |
|       |                                    |                |   |

|     |        |        |   |
|-----|--------|--------|---|
|     | JC     | NXT    | ; If there is a carry jump to NXT.                          |
|     | INX H  |        | ; Increment the H-L register pair.                          |
|     | DCR C  |        | ; Decrement the content of C-reg. for next byte.            |
|     | JNZ    | LOOP   | ; If the contents of C-reg. are not zero then jump to LOOP. |
|     | STA    | 2501 H | ; Store the sum in memory location 2501 H.                  |
|     | HLT    |        | ; Stop processing.  |
| NXT | MVI A, | 01 H   | ; If carry occurs load 01 to accumulator.                   |
|     | STA    | 2501 H | ; Store 01 H to 2501 H.                                     |
|     | HLT    |        | ; Stop processing.  |

**Example 6.17.** Write an assembly language program of 8085 to add five bytes of data stored in memory locations starting at 2000 H. The sum may be more than one byte. Store the result at two consecutive memory locations 2500 H and 2501 H.

**Solution.**

**Program:**

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
|       | LXI H,    | 2000 H  | ; Get H-L pair with 2000 H.  |
|       | MVI C,    | 05 H    | ; Store the number of data bytes in C register as counter.         |
|       | XRA A     |         | ; Clear the accumulator and CY flag.                               |
|       | MOV B,    | A       | ; Store the accumulator contents to B-register also for the carry. |
| LOOP  | ADD M     |         | ; Add the byte in accumulator.                                     |
|       | JNC       | NXT     | ; If there is a no carry jump to NXT.                              |
|       | INR B     |         | ; Increment B as carry is generated.                               |
| NXT   | INX H     |         | ; Increment the H-L register pair.                                 |
|       | DCR C     |         | ; Decrement the content of C-reg. for next byte.                   |
|       | JNZ       | LOOP    | ; If the contents of C-reg. are not zero then jump to LOOP.        |
|       | LXI H,    | 2500 H  | ; Store H-L pair with 2500 H (destination address).                |
|       | MOV M,    | A       | ; Store the sum to 2500 H.   |
|       | INX H     |         |  |
|       | MOV M,    | B       | ; Content of B are stored in 2501 H.                               |
|       | HLT       |         | ; Stop processing.   |

**Example 6.18.** Write a program in assembly language of 8085 to test 6<sup>th</sup> bit ( $D_6$  bit) of a byte stored at 2500 H memory location. If the bit  $D_6$  is zero then store 00 H at 2501 H else store the same number at 2501 H.

**Solution.**

**Program:**

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>                              |
|--------------|------------------|----------------|--|
|              | LXI H,           | 2500 H         | ; Get H-L pair with 2500 H.                  |
|              | MOV A,           | M              | ; Store the byte in accumulator              |
|              | ANI              | 40 H           | ; Reset all the bits except D <sub>6</sub> . |
|              | JZ               | END            | ; If this bit is zero jump to END.           |
|              | MOV A,           | M              | ; Store the byte in accumulator again.       |
|              | INX H            |                | ; Increment H-L pair.                        |
|              | MOV M,           | A              | ; The byte is stored in 2501 H.              |
|              | HLT              |                | ; Stop processing.                           |
| END          | XRA A            |                | ; Clear accumulator.                         |
|              | INX H            |                | ; Increment H-L pair.                        |
|              | MOV M,           | A              | ; 00 H is stored in 2501 H.                  |
|              | HLT              |                | ; Stop processing.                           |

**Example 6.19.** Write an ALP (Assembly Language Program) of 8085 to find logical OR and exclusive OR of two numbers stored at 2501 H and 2502 H memory locations. The results should be stored at 2503 H (logical OR operation) and 2504 H (XOR operation) memory locations.

**Solution.**

**Program:**

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>                         |
|--------------|------------------|----------------|---|
|              | LXI H,           | 2501 H         | ; Get H-L pair with 2501 H.             |
|              | MOV B,           | M              | ; Store the byte in B-register.         |
|              | MOV A,           | B              | ; Store this byte in accumulator also.  |
|              | INX H            |                | ; Increment H-L pair.                   |
|              | MOV C,           | M              | ; Store second byte to C register.      |
|              | ORA C            |                | ; Logical OR of two bytes.              |
|              | INX H            |                | ; Increment H-L pair.                   |
|              | MOV M,           | A              | ; Store the result (OR) in 2503 H.      |
|              | MOV A,           | B              | ; Load the first number in accumulator. |
|              | XRA C            |                | ; Ex-OR the two number.                 |
|              | INX H            |                | ; Increment H-L pair.                   |
|              | MOV M,           | A              | ; Store the result (XOR)                |
|              | HLT              |                | ; Stop processing.                      |

**Example 6.20.** Write an ALP (Assembly Language Program) for 8085 to shift an 8-bit number left by one bit. The number is stored in 2101 H memory location. The result is to be stored in 2102 H memory location.

**Solution.**

**Program:**

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>                  |
|--------------|------------------|----------------|----------------------------------|
|              | LDA              | 2101 H         | ; Get the number in accumulator. |
|              | ADD A            |                | ; Shift it left by one bit.      |
|              | STA              | 2102 H         | ; Store the result in 2502 H.    |
|              | HLT              |                | ; Stop processing.               |

**Example 6.21.** Write an ALP 8085 to shift a 16-bit number left by one bit. The number is stored in 2101 H and 2102 H memory locations. The result is to be stored in 2103 H and 2104 H memory locations.

**Solution.**

**Program:**

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
|       | LHLD      | 2101 H  | ; Get the 16 bit number in H-L register pair.             |
|       | DAD H     |         | ; Shift left by one bit.                                  |
|       | SHLD      | 2103 H  | ; Store the result in 2103 H and 2104 H memory locations. |
|       | HLT       |         | ; Stop processing.  |

## 6.2 PROGAMS ON CODE CONVERSION

The programs on code conversion will now be discussed.

### 6.2.1 BCD to Binary Conversion

Suppose we have 0 to 99 decimal numbers (BCD numbers) and we wish to convert any of these numbers to its equivalent binary number, we proceed in the following way:

For example 49 ( $4 \times 10 + 9$ ) is the given decimal number, its binary equivalent is ( $00011\ 0001_2 = 31\ H$ ).

- Step I      Unpack the number in MSD and LSD form.  
0000 1001    LSD in four least significant nibble in a register.  
0000 0100    MSD in four least significant nibble in another register.
- Step II     Multiply MSD by decimal number 10.  
or MSD is added 10 times with 0000.  
So we get ( $4 \times 10 = 0010\ 1000$ ).
- Step III    ADD LSD to the result of step II.  
i.e.     $00101000 + 00001001 = 00110001$   
Hence we get the result.

**Example 6.22.** A BCD number (0 to 99) is stored in a memory location 2500 H, write an ALP of 8085 to convert the BCD number into its equivalent binary number. Store the result in 2600 H memory location.

**Solution.**

**Main Program:**

| Label | Mnemonics | Operand | Comments                              |
|-------|-----------|---------|---------------------------------------|
|       | LXI SP,   | XXXX H  | ; Initialize the stack pointer.       |
|       | LXI H,    | 2500 H  | ; Get H-L pair with 2500 H.           |
|       | LXI D,    | 2600 H  | ; Get D-E pair with 2600 H.           |
|       | MOV A,    | M       | ; Load the byte to accumulator.       |
|       | CALL      | CONV    | ; Call conversion subroutine program. |
|       | STAX D    |         | ; Store the result in 2600 H.         |
|       | HLT       |         |                                       |

**Subroutine Program:**

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>                                    |
|--------------|------------------|----------------|--|
| CONV         | PUSH D           |                | ; Push the contents of D-E register pair to stack. |
|              | PUSH H           |                | ; Push the contents of H-L register pair to stack. |
|              | ANI              | 0F H           | ; Separate LSD                                     |
|              | MOV C,           | A              | ; Store it to C-register.                          |
|              | MOV A,           | M              | ; Load the byte to accumulator again.              |
|              | ANI              | F0 H           | ; Separate MSD.                                    |
|              | RRC              |                | ; Rotate right four times                          |
|              | RRC              |                | ; to shift MSD to four                             |
|              | RRC              |                | ; least significant nibble                         |
|              | RRC              |                | ; of accumulator.                                  |
|              | MOV D,           | A              | ; Store MSD to D register.                         |
|              | XRA A            |                | ; Clear accumulator.                               |
|              | MVI E,           | 0A H           | ; Get 0A (decimal number 10) to E-register.        |
| SUM          | ADD D            |                | ; Add MSD ten times to 00 H.                       |
|              | DCR E            |                | ; Decrement C                                      |
|              | JNZ              | SUM            | ; If addition not complete then move to SUM.       |
|              | ADD C            |                | ; Add LSD to 10 X MSD                              |
|              | POP H            |                | ; Pop the contents of H-L pair.                    |
|              | POP D            |                | ; Pop the contents of D-E pair.                    |
|              | RET              |                | ; Go back to main program.                         |

### 6.2.2 Binary to BCD (Unpacked) Conversion

For example binary number is 1111 1111 (FF H). Its decimal equivalent is 255<sub>10</sub> having the unpacked BCD numbers as

|    |             |       |
|----|-------------|-------|
| 05 | (0000 0101) | BCD 1 |
| 05 | (0000 0101) | BCD 2 |
| 02 | (0000 0010) | BCD 3 |

The procedure for converting the 8 bit binary number (0000 0000 to 1111 1111) into unpacked BCD number is given in the following steps.

- Step I      If the number is less than 100 go to step II. If the number is more than 100, divide the number by 100 or subtract 100 repeatedly till the remainder is less than 100. The quotient is MS BCD (BCD 3).
- Step II     If the number (or remainder) is less than 10, go to step III. If number is >10 < 100, then subtract 10 repeatedly till the remainder is less than 10. The quotient is BCD 2.
- Step III    The remainder from step II is BCD 1.

**Example 6.23.** A binary number (between 00 H to FF H) is stored in a memory location 2500 H, write an ALP of 8085 to convert this number into BCD number. Store each BCD as unpacked BCD digit in the memory locations at 2601 H to 2603 H.

**Solution.**

**Main Program:**

| Label | Mnemonics | Operand | Comments                              |
|-------|-----------|---------|---------------------------------------|
|       | LXI SP,   | XXXX H  | ; Initialize the stack pointer.       |
|       | LXI H,    | 2500 H  | ; Get H-L pair with 2500 H.           |
|       | MOV A,    | M       | ; Load the byte to accumulator.       |
|       | CALL      | CONV    | ; Call conversion subroutine program. |
|       | HLT       |         |                                       |

**Subroutine Program 1:**

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
| CONV  | LXI H,    | 2601 H  | ; Get H-L pair with 2601 H.                                |
|       | MVI B,    | 64 H    | ; Load 100 in B-reg.                                       |
|       | CALL      | CONV1   | ; Call another subroutine program for the division by 100. |
|       | MVI B,    | 0A H    | ; Load 10 in B-reg.  |
|       | CALL      | CONV1   | ; Call the subroutine again for the division by 10.        |
|       | MOV M,    | A       | ; Load the accumulator contents to the memory location.    |
|       | RET       |         | ; Go back to main program.                                 |

**Subroutine Program 2:**

| Label | Mnemonics | Operand | Comments                                      |
|-------|-----------|---------|---|
| CONV1 | MVI M,    | FF H    | ; Get FF H in the M <sub>H-L</sub> .          |
| NXT   | INR M     |         | ; Increment [M <sub>H-L</sub> ]               |
|       | SUB B     |         | ; Subtract the content of B from accumulator. |
|       | JNC       | NXT     | ; Jump to NXT if no carry.                    |
|       | ADD B     |         | ; Add the contents of B to accumulator.       |
|       | INX H     |         | ; Increment H-L register pair.                |
|       | RET       |         | ; Go back.                                    |

**Example 6.24.** Write ALP of 8085 for the following statement:

A set of three packed BCD numbers (six digits or three bytes) are stored in memory locations starting at 2500 H. The seven segment codes of the digits 0 to 9 for common cathode LED are stored in memory locations starting at 2050 H; and the answer is to be stored in memory locations starting at 2070 H.

**Solution.**

**Main Program:**

| Label | Mnemonics | Operand | Comments                        |
|-------|-----------|---------|---------------------------------|
|       | LXI SP,   | XXXX H  | ; Initialize the stack pointer. |
|       | LXI H,    | 2500 H  | ; Get H-L pair with 2500 H.     |

|        |      |  |
|--------|------|--|
| MVI D, | 03 H | ; Number of bytes to be converted is placed in D-register. |
| CALL   | CONV | ; Call conversion subroutine program.                      |

HLT

#### **Subroutine Program 1:**

| <b>Label</b> | <b>Mnemonics</b>                        | <b>Operand</b>       | <b>Comments</b>  |
|--------------|---|----------------------|--|
| CONV         | LXI B,<br>MOV A,                        | 2070 H<br>M          | ; Get B-C pair with 2070 H.<br>; Move the content of $[M_{H-L}]$ to accumulator.   |
| NXT          | ANI<br>RRC<br>RRC<br>RRC<br>RRC<br>CALL | F0 H                 | ; Separate MSD.<br>; Rotate right four times<br>; to shift MSD to four least significant nibble of accumulator.  |
|              | SEGMENT                                 |                      | ; Call subroutine program for the conversion of unpacked BCD to seven segment.   |
|              | INX B<br>MOV A,<br>ANI<br>CALL          | M<br>0F H<br>SEGMENT | ; Increment B-C register pair.<br>; Get the byte again.<br>; Separate LSD.<br>; Call subroutine program for the conversion of unpacked BCD (LSD) to seven segment. |
|              | INX B<br>INX H<br>DCR D                 |                      | ; Increment B-C register pair.<br>; Increment H-L register pair.<br>; Decrement the contents of D register.  |
|              | JNZ                                     | NXT                  | ; If the content in D register is not then jump to NXT for next byte.  |
|              | RET                                     |                      | ; Return to main program.  |

#### **Subroutine Program 2:**

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>   |
|--------------|------------------|----------------|---|
| SEGMENT      | PUSH H           |                | ; Push the contents of H-L register pair to stack.  |
|              | LXI H,<br>ADD L  | 2050 H         | ; Get H-L pair with 2050 H.<br>; Add the content of L register to accumulator to get right location for the digit from the look-up table. |
|              | MOV L,<br>MOV A, | A<br>M         | ; Store it to L-register.<br>; Load the corresponding data from the look-up table to accumulator.   |
|              | STAX B           |                | ; Store the result in the memory location addressed by B-C register pair.   |

```

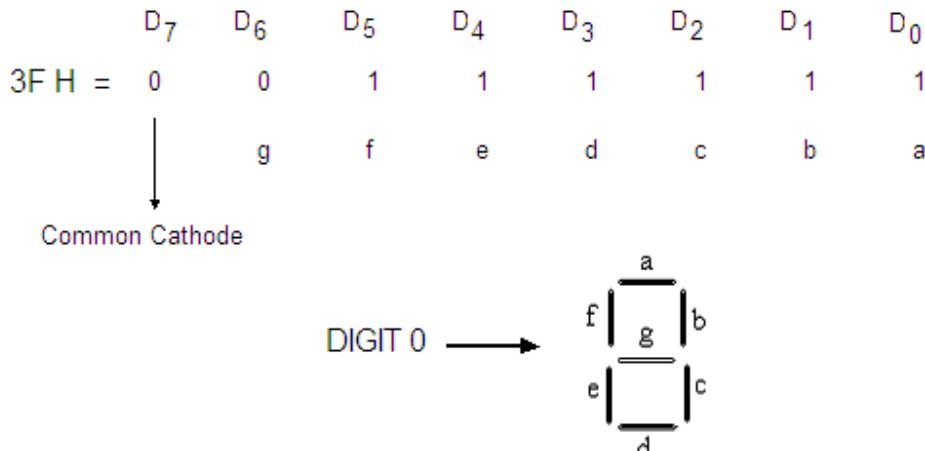
POP H ; Get the contents of H-L register
        ; pair from the stack.
RET   ; Go back

```

DATA (in the form of look-up table) is stored in the following memory locations:

|        |      |           |
|--------|------|-----------|
| 2050 H | 3F H | (Digit 0) |
| 2051 H | 06 H | (Digit 1) |
| 2052 H | 5B H | (Digit 2) |
| 2053 H | 4F H | (Digit 3) |
| 2054 H | 66 H | (Digit 4) |
| 2055 H | 6D H | (Digit 5) |
| 2056 H | 7D H | (Digit 6) |
| 2057 H | 07 H | (Digit 7) |
| 2058 H | 7F H | (Digit 8) |
| 2059 H | 6F H | (Digit 9) |

The 3F H data is given for the digit 0. It is clear from the figure 6.1.



**Fig. 6.1**

### 6.2.3 Binary to ASCII Conversion

ASCII (American Standard Code for Information Interchange) is a most commonly used alphanumeric code. It is a seven bit code. The hexadecimal numbers 30 to 39 (0011 0000 to 0011 1001) represent 0 to 9 ASCII decimal numbers. Similarly, 41 H to 5A H (0100 0001 to 0101 1010) represent capital letters A to Z in ASCII.

For example, the ASCII representation of a binary number 8E is 38 H and 45 H (as 8 is represented by 0011 1000 or 38 H; and E is represented by 0100 0101 or 45 H).

The following steps are carried out for the conversion of Binary to ASCII

- Step I      First separate the LSD and MSD of the given binary number (or byte). Each digit is then converted to ASCII.
- Step II     If the digit is less than 10 (0A H) then add 30 H (0011 0000) to the binary number else add 37 H (0011 0111).

For binary number 8E H, 30 is added to 08 and we get 38 H (the ASCII Code for 8) and 37 is added to 0E H to get 45 H (the ASCII Code for E).

**Example 6.25.** An eight bit binary number is stored in 2500 H. Write an ALP for 8085 to convert the binary number to its equivalent ASCII code. The ASCII code for most

*significant binary digit should be stored to 2601 H location; and the code for least significant binary digit should be stored to 2602 H location.*

### Solution.

#### Main Program:

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
|       | LXI SP,   | XXXX H  | ; Initialize the stack pointer.   |
|       | LXI H,    | 2500 H  | ; Get H-L pair with 2500 H.   |
|       | LXI D,    | 2601 H  | ; Get D-E pair with 2601 H.   |
|       | MOV A,    | M       | ; Move the content of $[M_{H-L}]$ to accumulator.   |
|       | MOV B,    | A       | ; Move the accumulator content to B-register.   |
|       | RRC       |         | ; Rotate right four times   |
|       | RRC       |         | ; to shift MSD to four  |
|       | RRC       |         | ; least significant nibble  |
|       | RRC       |         | ; of accumulator.   |
|       | CALL      | ASCII   | ; Call the subroutine to convert MSD to ASCII.  |
|       | STAX D    |         | ; Store the ASCII code of MS digit to the memory location addressed by D-E register pair. |
|       | INX D     |         | ; Increment D-E register pair.  |
|       | MOV A,    | B       | ; Move the binary number again to accumulator.  |
|       | CALL      | ASCII   | ; Call the subroutine to convert LSD to ASCII.  |
|       | STAX D    |         | ; Store the ASCII code of LS digit to the memory location addressed by D-E register pair. |
|       | HLT       |         | ; Stop processing.  |

#### Subroutine Program:

| Label | Mnemonics | Operand | Comments                           |
|-------|-----------|---------|------------------------------------|
| ASCII | ANI       | 0F H    | ; Isolate MS digit.                |
|       | CPI       | 0A      | ; Compare with 10 (0A H).          |
|       | JC        | NXT     | ; If it less than 10, jump to NXT. |
|       | ADI       | 07 H    | ; Else add 07 H.                   |
| NXT   | ADI       | 30 H    | ; Add 30 H.                        |
|       | RET       |         | ; Go back to main program.         |

#### 6.2.4 ASCII to Binary Conversion

The following steps are carried out to convert the ASCII code to binary (Hex):

- Step I      Subtract 30 H (0011 0000) from the given binary (Hex) number.
- Step II     If the difference after subtraction in step I is less than 10, then it is the required binary (Hex) number.

**Step III** If the difference after subtraction in step II is more than 10 then subtract 07 also from it. This will then give the required binary (Hex) number.

**Example 6.26.** An ASCII number is stored in 2500 H memory location. Write an ALP for 8085 to convert this number into its equivalent binary (Hex) number and store it to 2501 memory location.

**Solution.**

**Main Program:**

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>  |
|--------------|------------------|----------------|--|
|              | LXI SP,          | XXXX H         | ; Initialize the stack pointer.                        |
|              | LXI H,           | 2500 H         | ; Get H-L pair with 2500 H.                            |
|              | MOV A,           | M              | ; Move the content of $[M_{H-L}]$ to accumulator.      |
|              | CALL             | CONV           | ; Call the subroutine to convert ASCII to binary.      |
|              | INX H            |                | ; Increment H-L register pair.                         |
|              | MOV M,           | A              | ; Store the binary number to required memory location. |
|              | HLT              |                | ; Stop processing.                                     |

**Subroutine Program:**

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>                               |
|--------------|------------------|----------------|---|
| CONV         | SUI              | 30 H           | ; Subtract 30 H from accumulator.             |
|              | CPI              | 0A             | ; Compare with 10 (0A H).                     |
|              | RC               |                | ; If it less than 10 go back to main program. |
|              | SUI              | 07 H           | ; Else subtract 07 H.                         |
|              | RET              |                | ; Go back to main program.                    |

### 6.3 PROGAMS ON ADDITION AND SUBTRACTION

Now we shall take up the problems on BCD or decimal addition and subtraction of two bytes or multibyte. These problems are self explanatory and can be understood by considering proper logic.

**Example 6.27.** Write an ALP for 8085 to add two 16-bit numbers. The augend's LS byte is stored in 2101 H and MS byte in 2102 H. The addend's LS and MS byte are stored in 2103 H and 2104 H respectively. The result is to be stored in 2105 H to 2107 H.

**Solution.**

**Main Program:**

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>  |
|--------------|------------------|----------------|--|
|              | LHLD             | 2101 H         | ; Load the H-L pair direct with the contents of 2101 H and 2102 H Locations. |
|              | XCHG             |                | ; Exchange the contents of H-L and D-E pair.                                 |
|              | LHLD             | 2103 H         | ; Load the H-L pair direct with the contents of 2103 H and 2104 H Locations. |

|     |        |        |  |
|-----|--------|--------|--|
|     | MVI C, | 00 H   | ; Store 00 H to C register for carry.                                      |
|     | DAD D  |        | ; Add the contents of H-L and D-E register pair and result is in H-L pair. |
|     | JNC    | NXT    | ; If there is no carry after the addition jump to NXT.                     |
|     | INR C  |        | ; Increment the carry.   |
| NXT | SHLD   | 2105 H | ; Store the contents in the required locations.                            |
|     | MOV A, | C      | ; Move carry to accumulator.   |
|     | STA    | 2107 H | ; Store carry in 2107 H.   |
|     | HLT    |        | ; Stop processing.   |

**Example 6.28.** Write an ALP for 8085 to add two N byte numbers. The augend's bytes are stored in memory locations starting at 2101 H and the addend's bytes are stored in memory locations starting at 2201 H. The number N is stored in memory location 2100 H. The result is to be stored in the memory locations starting at 2201 H.

**Solution.**

**Main Program:**

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
|       | LXI H,    | 2100 H  | ; Get H-L pair with 2100 H.                               |
|       | MOV C,    | M       | ; Move the number N to C-register for counter.            |
|       | INX H     |         | ; Increment the H-L pair.                                 |
|       | LXI D,    | 2201 H  | ; Get D-E pair with 2201 H.                               |
|       | XRA A     |         | ; Clear the accumulator and carry flag.                   |
| LOOP  | LDAX D    |         | ; Get the addend byte to accumulator.                     |
|       | ADC M     |         | ; Add with carry the two bytes.                           |
|       | MOV M,    | A       | ; Store the result in the location addressed by H-L pair. |
|       | INX D     |         | ; Increment the D-E pair.                                 |
|       | INX H     |         | ; Increment the H-L pair.                                 |
|       | DCR C     |         | ; Decrement C.  |
|       | JNZ       | LOOP    | ; If C is not zero, then jump to LOOP.                    |
|       | HLT       |         | ; Stop processing.  |

**Example 6.29.** Write an ALP for 8085 to add two N byte numbers. The augend's bytes are stored in memory locations starting at 2101 H and the addend's bytes are stored in memory locations starting at 2201 H. The number N is stored in memory location 2100 H. The result should in decimal form and is to be stored in the memory locations starting at 2201 H.

**Solution.**

**Main Program:**

| Label | Mnemonics | Operand | Comments |
|-------|-----------|---------|----------|
|-------|-----------|---------|----------|

|      |                          |             |   |
|------|--------------------------|-------------|---|
|      | LXI H,<br>MOV C,         | 2100 H<br>M | ; Get H-L pair with 2100 H.<br>; Move the number N to C-register for counter.                       |
|      | INX H<br>LXI D,<br>XRA A | 2201 H      | ; Increment the H-L pair.<br>; Get D-E pair with 2201 H.<br>; Clear the accumulator and carry flag. |
| LOOP | LDAX D                   |             | ; Get the addend byte to accumulator.   |
|      | ADC M<br>DAA             |             | ; Add with carry the two bytes.<br>; Decimal adjust the accumulator for the result in decimal form. |
|      | MOV M, A                 |             | ; Store the result in the location addressed by H-L pair.   |
|      | INX D<br>INX H<br>DCR C  |             | ; Increment the D-E pair.<br>; Increment the H-L pair.<br>; Decrement C.                            |
|      | JNZ LOOP                 |             | ; If C is not zero, then jump to LOOP.  |
|      | HLT                      |             | ; Stop processing.  |

It may be noted from this example that DAA (decimal adjust the accumulator) instruction is used after ADC M for the conversion of the result in decimal form.

**Example 6.30.** Write an ALP for 8085 for multibyte ( $N$  bytes) subtraction. The number of bytes ( $N$ ) is stored in 2100 H location. The minuend bytes are stored in the memory locations starting at 2101 H and the subtrahend bytes are stored in the memory locations starting at 2201 H. The answer should be stored in the memory locations starting at 2101 H.

### Solution.

#### Main Program:

| Label | Mnemonics                | Operand     | Comments  |
|-------|--------------------------|-------------|---|
|       | LXI H,<br>MOV C,         | 2100 H<br>M | ; Get H-L pair with 2100 H.<br>; Move the number N to C-register for counter.                       |
|       | INX H<br>LXI D,<br>XRA A | 2201 H      | ; Increment the H-L pair.<br>; Get D-E pair with 2201 H.<br>; Clear the accumulator and carry flag. |
| LOOP  | LDAX D                   |             | ; Get the addend byte to accumulator.   |
|       | SBB M                    |             | ; Subtract with borrow the contents of $[M_{H-L}]$ from the accumulator contents.                   |
|       | MOV M, A                 |             | ; Store the result in the location addressed by H-L pair.   |
|       | INX D<br>INX H           |             | ; Increment the D-E pair.<br>; Increment the H-L pair.  |

|       |      |  |
|-------|------|--|
| DCR C |      | ; Decrement C.                         |
| JNZ   | LOOP | ; If C is not zero, then jump to LOOP. |
| HLT   |      | ; Stop processing.                     |

### Decimal Subtraction

It may be noted from the above example that SBB M (subtract with borrow) instruction is used for the subtraction. The result will not be obtained in the decimal form as DAA instruction can not be used after Subtract instructions. For decimal subtractions, the subtrahend number should be converted to its equivalent 10's complement which will then be added to minuend.

**Example 6.31.** Write an ALP for 8085 to subtract 78 from 96. The answer in decimal form should be stored in 2100 H memory location.

### Solution.

#### Main Program:

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
|       | MVI C,    | 96 H    | ; Get 96 in C register.   |
|       | MVI B,    | 78 H    | ; Get 78 in B register.   |
|       | MVI A,    | 99 H    | ; Get 99 in accumulator.  |
|       | SUB B     |         | ; Subtract the content of B-register from 99 to get the 9's complement of 78. |
|       | INR A     |         | ; Increment accumulator to get 10's complement.                               |
|       | ADD C     |         | ; Add the content of C to the accumulator.                                    |
|       | DAA       |         | ; Decimal adjust the accumulator, to get the answer in decimal for.           |
|       | STA       | 2100 H  | ; Store the answer in decimal form at 2100 H location.                        |
|       | HLT       |         | ; Stop processing.  |

**Example 6.32.** Write an ALP for 8085 for multibyte (N bytes) decimal subtraction. The number of bytes (N) is stored in 2100 H location. The minuend bytes are stored in the memory locations starting at 2101 H and the subtrahend bytes are stored in the memory locations starting at 2201 H. The answer obtained in decimal form should be stored in the memory locations starting at 2101 H.

### Solution.

#### Main Program:

| Label | Mnemonics | Operand | Comments                                       |
|-------|-----------|---------|--|
|       | LXI SP,   | XXXX H  | ; Initialize the stack pointer.                |
|       | LXI H,    | 2100 H  | ; Get H-L pair with 2100 H.                    |
|       | MOV C,    | M       | ; Move the number N to C-register for counter. |
|       | INX H     |         | ; Increment the H-L pair.                      |
|       | LXI D,    | 2201 H  | ; Get D-E pair with 2201 H.                    |
|       | LDAX D    |         | ; Get the subtrahend byte in accumulator.      |

|      |          |      |   |
|------|----------|------|---|
|      | MOV B,   | A    | ; Move the accumulator content to B-register.                             |
|      | MVI A,   | 99 H | ; Store 99 to accumulator.  |
|      | SUB B    |      | ; Get 9's complement of the subtrahend.                                   |
|      | INR A    |      | ; Get 10's complement of the subtrahend.                                  |
|      | ADD M    |      | ; Add 10's complement of the subtrahend to minuend.                       |
|      | DAA      |      | ; Give the answer in decimal form.  |
|      | PUSH PSW |      | ; Push the carry to stack.  |
|      | MOV M,   | A    | ; Store the result in the location addressed by H-L pair.                 |
| LOOP | INX D    |      | ; Increment the D-E pair.   |
|      | INX H    |      | ; Increment the H-L pair.   |
|      | LDAX D   |      | ; Get the second number to accumulator.                                   |
|      | MOV B,   | A    | ; Store it to B-register.   |
|      | MVI A,   | 99 H | ; Store 99 to accumulator.  |
|      | SUB B    |      | ; Get 9's complement.   |
|      | MOV B,   | A    | ; Store it to B-register.   |
|      | POP PSW  |      | ; Get the carry of the previous addition from the stack.                  |
|      | MOV A,   | B    | ; Get the B-register content to accumulator.                              |
|      | ADC M    |      | ; Add with carry.   |
|      | DAA      |      | ; Give the answer in decimal form.  |
|      | PUH PSW  |      | ; Store the carry to stack for further addition.                          |
|      | MOV M,   | A    | ; Store the result in the memory location addressed by H-L register pair. |
|      | DCR C    |      | ; Decrement the content of C-register.                                    |
|      | JNZ      | LOOP | ; If C is not zero, then jump to LOOP.                                    |
|      | HLT      |      | ; Stop processing.  |

#### 6.4 PROGAMS TO FIND LARGEST OR SMALLEST NUMBER

**Example 6.33** Write an ALP for 8085 to find the larger of two numbers stored in memory locations 2101 H and 2102 H. Store the result in memory location 2103 H.

**Solution.**

**Main Program:**

| Label | Mnemonics | Operand | Comments                                |
|-------|-----------|---------|---|
|       | LXI H,    | 2101 H  | ; Get H-L pair with 2101 H.             |
|       | MOV A,    | M       | ; Move the first number in accumulator. |

|     |        |     |  |
|-----|--------|-----|--|
|     | INX H  |     | ; Increment the H-L pair.                                |
|     | CMP M  |     | ; Compare the second number with first number.           |
|     | JNC    | NXT | ; If the accumulator content is larger then jump to NXT. |
|     | MOV A, | M   | ; Else move $[M_{H-L}]$ to accumulator.                  |
| NXT | INX H  |     | ; Increment the H-L pair.                                |
|     | MOV M, | A   | ; Store the result in the required memory location.      |
|     | HLT    |     | ; Stop processing.                                       |

Note: To get the smaller number from the given two numbers, use the instruction JC in place of JNC.

**Example 6.34.** Write an ALP for 8085 to find the largest number among three numbers stored in memory locations staring at 2101 H. Store the result in memory location 2104H.

**Solution.**

**Main Program:**

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
|       | LXI H,    | 2101 H  | ; Get H-L pair with 2101 H.                                      |
|       | MOV A,    | M       | ; Move the first number in accumulator.                          |
|       | INX H     |         | ; Increment the H-L pair.  |
|       | CMP M     |         | ; Compare the second number with first number.                   |
|       | JNC       | NXT     | ; If the accumulator content is larger then jump to NXT.         |
|       | MOV A,    | M       | ; Else move $[M_{H-L}]$ to accumulator.                          |
| NXT   | INX H     |         | ; Increment the H-L pair.  |
|       | CMP M     |         | ; Compare the third number with the larger of first two numbers. |
|       | JNC       | NXT1    | ; If the accumulator content is larger then jump to NXT1.        |
|       | MOV A,    | M       | ; Else move $[M_{H-L}]$ to accumulator.                          |
| NXT1  | INX H     |         | ; Increment the H-L pair.  |
|       | MOV M,    | A       | ; Store the result in the required memory location.              |
|       | HLT       |         | ; Stop processing.   |

**Example 6.35.** Write an ALP for 8085 to find the largest number among a series of N numbers stored in memory locations staring at 2101 H. The number N is stored in memory location 2100 H. Store the result in memory location 2201 H.

**Solution.**

**Main Program:**

| Label | Mnemonics | Operand | Comments |
|-------|-----------|---------|----------|
|-------|-----------|---------|----------|

|      |        |        |   |
|------|--------|--------|---|
|      | LXI H, | 2100 H | ; Get H-L pair with 2100 H.   |
|      | MOV C, | M      | ; Store the number N in C-register which will be used as the counter. |
|      | INX H  |        | ; Increment the H-L pair.   |
|      | MOV A, | M      | ; Move the first number in accumulator.                               |
|      | DCR C  |        | ; Decrement count.  |
| LOOP | INX H  |        | ; Increment the H-L pair.   |
|      | CMP M  |        | ; Compare the second number with first number.                        |
|      | JNC    | NXT    | ; If the accumulator content is larger then jump to NXT.              |
|      | MOV A, | M      | ; Else move $[M_{H-L}]$ to accumulator.                               |
| NXT  | DCR C  |        | ; Decrement count.  |
|      | JNZ    | LOOP   | ; Jump to LOOP if not zero.   |
|      | STA    | 2201 H | ; Store the result in 2201 H.   |
|      | HLT    |        | ; Stop processing.  |

## 6.5 PROGAMS TO ARRANGE A GIVEN SERIES IN ASCENDING OR DESCENDING ORDER

**Example 6.36.** Write an ALP for 8085 to arrange a series of N-numbers in descending order. The number N is stored in memory location 2100 H. The series is stored in memory location starting at 2101 H. The result is to be stored in memory locations starting at 2201 H.

**Solution.**

**Main Program:**

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
|       | LXI SP,   | XXXX H  | ; Initialize stack pointer.                              |
|       | LXI D,    | 2201 H  | ; Get D-E pair with 2201 H.                              |
|       | LXI H,    | 2100 H  | ; Get H-L pair with 2100 H.                              |
|       | MOV B,    | M       | ; Store the count N in B-register.                       |
| START | LXI H,    | 2100 H  | ; Get H-L pair with 2100 H.                              |
|       | MOV C,    | M       | ; Store the number N in C-register also.                 |
|       | INX H     |         | ; Increment the H-L pair.                                |
|       | MOV A,    | M       | ; Move the first number in accumulator.                  |
|       | DCR C     |         | ; Decrement count.                                       |
| LOOP  | INX H     |         | ; Increment the H-L pair.                                |
|       | CMP M     |         | ; Compare the second number with first number.           |
|       | JNC       | NXT     | ; If the accumulator content is larger then jump to NXT. |
|       | MOV A,    | M       | ; Else move $[M_{H-L}]$ to accumulator.                  |
| NXT   | DCR C     |         | ; Decrement count.                                       |
|       | JNZ       | LOOP    | ; Jump to LOOP if not zero.                              |

|        |       |  |   |
|--------|-------|--|---|
| STAX D |       |  | ; Store the largest number in memory location addressed by D-E register pair.                     |
| CALL   | SUBR  |  | ; Call the subroutine SUBR to put 00 H to the memory location where the largest number was found. |
| INX D  |       |  | ; Increment D-E register pair.  |
| DCR B  |       |  | ; Decrement B.  |
| JNZ    | START |  | ; Jump to START if not zero.  |
| HLT    |       |  | ; Stop processing.  |

**Subroutine Program:**

| Label | Mnemonics        | Operand     | Comments  |
|-------|------------------|-------------|---|
| SUBR  | LXI H,<br>MOV C, | 2100 H<br>M | ; Get H-L pair with 2100 H.<br>; Store the number N in C-register which will be used as the counter.    |
| AGAIN | INX H<br>CMP M   |             | ; Increment the H-L pair.<br>; Compare the number with the largest number obtained in the main program. |
|       | JZ               | NXT         | ; If it is largest jump to NXT.   |
|       | DCR C            |             | ; Decrement count.  |
|       | JNZ              | AGAIN       | ; If counts not complete jump to AGAIN.   |
|       | MVI A,<br>MOV M, | 00 H<br>A   | ; Store 00 H to the accumulator.<br>; Put 00 H to the location at which the largest number was found.   |
|       | RET              |             | ; Go back to main program.  |

**Example 6.37.** Write an ALP for 8085 to arrange a series of  $N$ -numbers in ascending order. The number  $N$  is stored in memory location 2100 H. The series is stored in memory location starting at 2101 H. The result is to be stored in memory locations starting at 2201 H.

**Solution.**

**Main Program:**

| Label | Mnemonics | Operand | Comments                                   |
|-------|-----------|---------|--|
|       | LXI SP,   | XXXX H  | ; Initialize stack pointer.                |
|       | LXI D,    | 2201 H  | ; Get D-E pair with 2201 H.                |
|       | LXI H,    | 2100 H  | ; Get H-L pair with 2100 H.                |
|       | MOV B,    | M       | ; Store the count $N$ in B-register.       |
| START | LXI H,    | 2100 H  | ; Get H-L pair with 2100 H.                |
|       | MOV C,    | M       | ; Store the number $N$ in C-register also. |
|       | INX H     |         | ; Increment the H-L pair.                  |
|       | MOV A,    | M       | ; Move the first number in accumulator.    |
|       | DCR C     |         | ; Decrement count.                         |
| LOOP  | INX H     |         | ; Increment the H-L pair.                  |

|     |        |       |  |
|-----|--------|-------|--|
|     | CMP M  |       | ; Compare the second number with first number.   |
|     | JC     | NXT   | ; If the accumulator content is smaller then jump to NXT.  |
|     | MOV A, | M     | ; Else move $[M_{H-L}]$ to accumulator.  |
| NXT | DCR C  |       | ; Decrement count.   |
|     | JNZ    | LOOP  | ; Jump to LOOP if not zero.  |
|     | STAX D |       | ; Store the largest number in memory location addressed by D-E register pair.                      |
|     | CALL   | SUBR  | ; Call the subroutine SUBR to put FF H to the memory location where the smallest number was found. |
|     | INX D  |       | ; Increment D-E register pair.   |
|     | DCR B  |       | ; Decrement B.   |
|     | JNZ    | START | ; Jump to START if not zero.   |
|     | HLT    |       | ; Stop processing.   |

#### Subroutine Program:

| Label | Mnemonics        | Operand     | Comments  |
|-------|------------------|-------------|---|
| SUBR  | LXI H,<br>MOV C, | 2100 H<br>M | ; Get H-L pair with 2100 H.<br>; Store the number N is C-register which will be used as the counter.    |
| AGAIN | INX H<br>CMP M   |             | ; Increment the H-L pair.<br>; Compare the number with the smalest number obtained in the main program. |
|       | JZ               | NXT         | ; If it is smallest jump to NXT.  |
|       | DCR C            |             | ; Decrement count.  |
|       | JNZ              | AGAIN       | ; If counts not complete jump to AGAIN.   |
|       | MVI A,<br>MOV M, | FF H<br>A   | ; Store FF H to the accumulator.<br>; Put FF H to the location at which the smallest number was found.  |
|       | RET              |             | ; Go back to main program.  |

#### 6.6 PROGRAMS ON MULTIPLICATION

There are two methods for multiplication of two numbers.

1. Repetitive Addition Method
2. Shift and Add Method

##### Repetitive Addition Method

In this method multiplicand is added by the multiplier times.

For example, we wish to multiply 8 and 4 numbers. So add 08 to 00 for four times.

i.e.

$$\begin{array}{r}
 00 \\
 + 08 \quad 1 \text{ time} \\
 \hline
 08 \\
 + 08 \quad 2 \text{ times} \\
 \hline
 16 \\
 + 08 \quad 3 \text{ times} \\
 \hline
 24 \\
 + 08 \quad 4 \text{ times} \\
 \hline
 32 \quad \text{Answer}
 \end{array}$$

### Shift and Add Method

For example, we wish to multiply two decimal numbers 32 and 12 as:

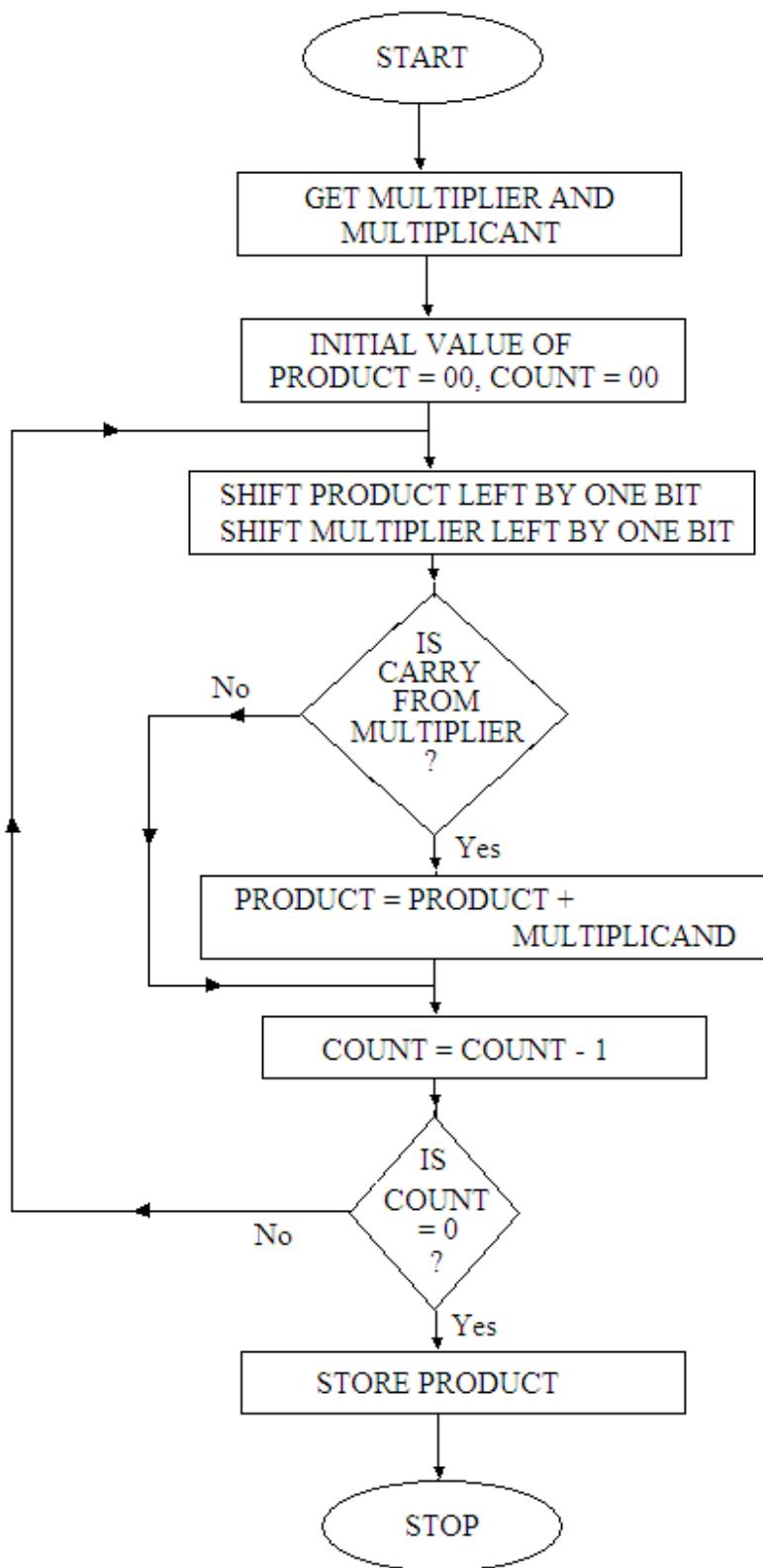
$$\begin{array}{r}
 32 \quad \text{Multiplicand} \\
 \times 12 \quad \text{Multiplier} \\
 \hline
 64 \\
 32 \quad \text{Shift by one digit} \\
 \hline
 384 \quad \text{Answer}
 \end{array}$$

In this example, first of all multiplicand 32 is multiplied by 2 and the result 64 is copied in the temporary register (or written on the scratch pad). Again 32 is multiplied by 1 and result 32 is shifted left by one digit to make it 320. Now 64 and 320 are added and we get the answer 384.

Similar method used in binary multiplication. In binary multiplication, when a multiplicand is multiplied by 1, we get the product equal to multiplicand. When a multiplicand is multiplied by 0, we get the product zero. For example 08 and 04 are multiplied as:

$$\begin{array}{r}
 00001000 \\
 \times 00000100 \\
 \hline
 00000000 \\
 00000000 \\
 00001000 \\
 00000000 \\
 00000000 \\
 00000000 \\
 00000000 \\
 00000000 \\
 \hline
 000000000100000
 \end{array}$$

The answer is 0020 H ( $32_{10}$ ). It may be noted that the answer is not in BCD. This method may further be illustrated using the following flow chart.



**Fig. 6.1**

**Example 6.38.** Write an ALP for 8085 to multiply any number (stored in memory location 2101 H) by 2 and store the result in 2102 H memory location.

**Solution.**

**Program:**

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
|       | LXI H,    | 2101 H  | ; Get the H-L pair with 2101 H.  |
|       | MOV A,    | M       | ; Move the number in accumulator.  |
|       | STC       |         | ; Set the carry flag.  |
|       | CMC       |         | ; Complement the carry ( it clear the carry).                                      |
|       | RAL       |         | ; Rotate accumulator with carry i.e. it multiplies the accumulator content by two. |
|       | INX H     |         | ; Increment the H-L pair.  |
|       | MOV M,    | A       | ; Store the result in 2102 H memory location.                                      |
|       | HLT       |         | ; Stop processing.   |

**Example 6.39.** Write an ALP for 8085 to multiply two numbers using repetitive addition method. The two numbers are in memory locations 2101 H and 2102 H. Store the result in 2103 H and 2104 H.

**Solution.**

**Main Program:**

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
|       | LXI H,    | 2101 H  | ; Get H-L pair with 2101 H.   |
|       | MOV B,    | M       | ; Get first number in B-register.   |
|       | INX H     |         | ; Increment H-L pair.   |
|       | MOV A,    | M       | ; Get the second number in accumulator.                                     |
|       | CMP B     |         | ; Compare the two numbers.  |
|       | JNC       | NXT     | ; If A < B then use A as counter.   |
|       | MVI D,    | 00 H    | ; Move 00 to D register.  |
|       | MOV E,    | B       | ; Move first number to E register.  |
|       | JMP       | NXT1    | ; Jump to NXT1.   |
| NXT   | MVI D,    | 00 H    | ; If A>B, so B should be used as counter.                                   |
|       | MOV E,    | A       | ; Store accumulator content in E-register.                                  |
|       | MOV A,    | B       | ; Store B-register content in accumulator.                                  |
| NXT1  | LXI H,    | 0000 H  | ; Put 00 to H and L registers.  |
|       | ORA A     |         | ; ORing of A with A to find if A = 00 H.                                    |
|       | JZ        | END     | ; If zero jump to End.  |
|       | DAD D     |         | ; Add the contents of H-L pair with D-E pair and the result is in H-L pair. |

|     |       |        |                             |
|-----|-------|--------|-----------------------------|
|     | DCR A |        | ; Decrement count.          |
|     | JNZ   | LOOP   | ; If not zero jump to LOOP. |
| END | SHLD  | 2103 H | ; Store the result.         |
|     | HLT   |        | ; Stop processing.          |

**Example 6.40.** Write an ALP for 8085 to multiply two numbers using shift and add method. The two numbers are in memory locations 2101 H and 2102 H. Store the result in 2103 H and 2104 H.

**Solution.**

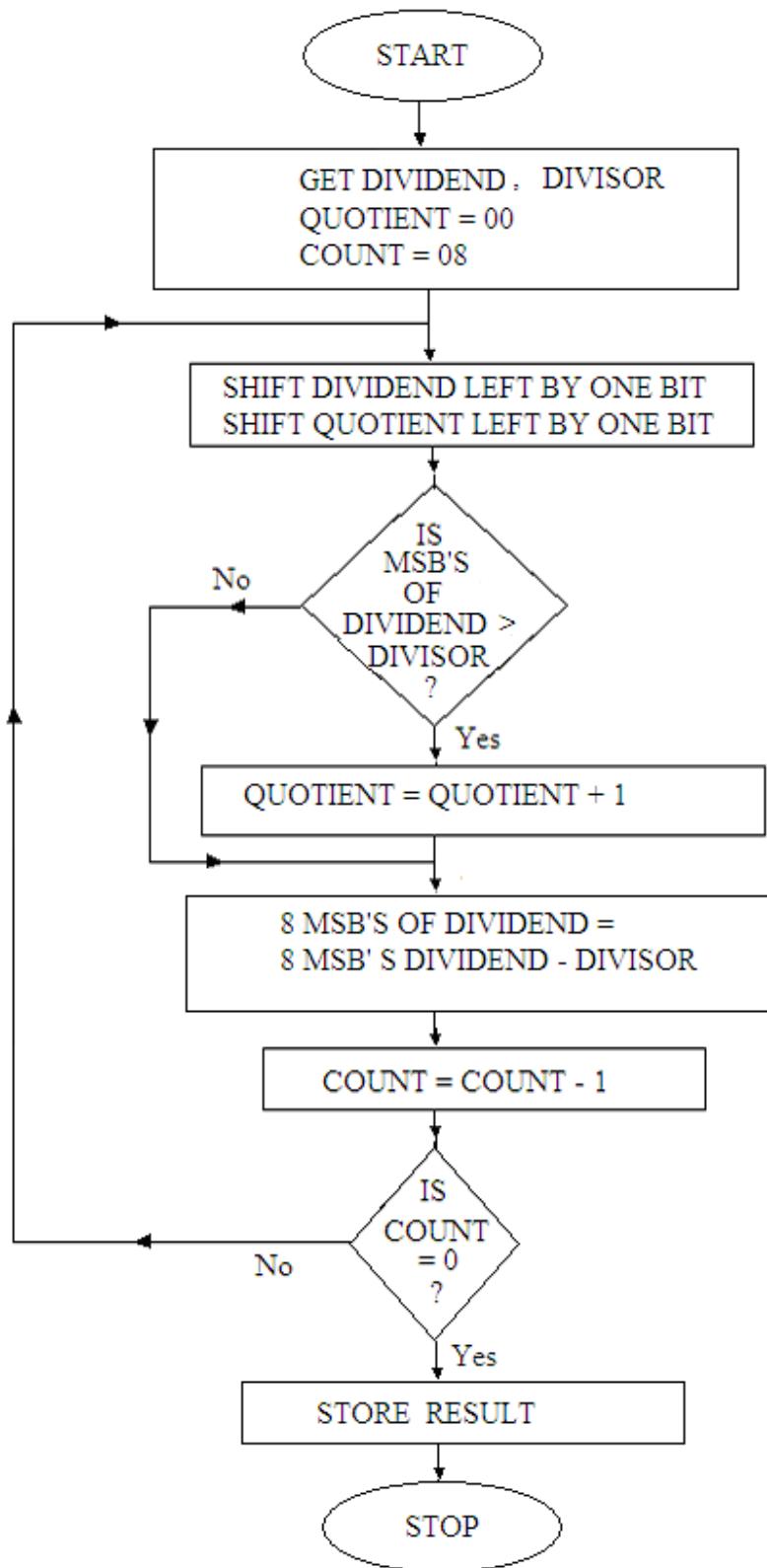
**Main Program:**

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
| MULT  | LXI H,    | 2101 H  | ; Get H-L pair with 2101 H.                                     |
|       | MOV E,    | M       | ; Get first number in E-register.                               |
|       | MVI D,    | 00 H    | ; Extend to 16 bits.  |
|       | INX H     |         | ; Increment H-L pair.   |
|       | MOV A,    | M       | ; Get the multiplier in accumulator.                            |
|       | LXI H,    | 0000 H  | ; Put 00 to H and L registers.                                  |
|       | MVI B,    | 08 H    | ; Get count = 08.   |
|       | DAD H     |         | ; Multiplicand = 2 x Multiplicand..                             |
|       | RAL       |         | ; Rotate accumulator left to find if most significant bit is 1. |
|       | JNC       | NXT     | ; If no carry jump to NXT.                                      |
| NXT   | DAD D     |         | ; Get Product = product + multiplicand.                         |
|       | DCR B     |         | ; Decrement B.  |
|       | JNZ       | MULT    | ; If not zero jump to MULT.                                     |
|       | SHLD      | 2103 H  | ; Store the result in 2103 H and 2104 H.                        |
|       | HLT       |         | ; Stop processing.  |

## 6.7 PROGRAM ON 8-BIT DIVISION

To divide 16 bit dividend with 8 bit divisor, the divisor is subtracted from the 8 most significant bits of the dividend. If there is no borrow, the bit of the quotient is set to 1, otherwise 0. The dividend is then shifted left by one bit before each trial of subtraction. The dividend and quotient share a register pair. Due to shift of dividend one bit of the register falls vacant in each subtraction. The quotient is stored in vacant bit positions. It is illustrated by the flow chart shown in figure 6.2.

If the dividend of the division is an 8-bit number, then it is extended to a 16-bit number by placing zeros in the MSBs positions.



**Fig. 6.2**

**Example 6.41.** Write an ALP for 8085 to divide a 16-bit number by an 8-bit number. The dividend bytes are stored in memory locations 2101 H and 2102 H. (LS byte in 2101 H and MS byte in 2102 H). The divisor is stored in memory location 2103 H. The quotient is to be stored in the memory location 2104 H and the remainder is to be stored in the memory location 2105 H.

**Solution.**

**Main Program:**

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
| LOOP  | LHLD      | 2101 H  | ; Get 16-bit dividend in H-L pair.                     |
|       | LDA       | 2103 H  | ; Get divisor in accumulator.                          |
|       | MVI C,    | 08 H    | ; Get count 08 in C-register.                          |
|       | MVI D,    | 00 H    | ; Quotient = 00.                                       |
|       | MOV B,    | A       | ; Keep the divisor in B-register also.                 |
|       | DAD H     |         | ; Shift dividend left by one bit.                      |
|       | MOV A,    | D       | ; Keep the quotient in accumulator.                    |
|       | ADD A     |         | ; Shift quotient left by one bit.                      |
|       | MOV D,    | A       | ; Store the quotient in D-register.                    |
|       | SUB B     |         | ; Perform trial subtraction.                           |
| NXT   | JC        | NXT     | ; If carry then dividend = previous dividend.          |
|       | MOV H,    | A       | ; Put most significant bits of dividend in register H. |
|       | INR D     |         | ; Increment quotient by 1.                             |
|       | DCR C     |         | ; Decrement C.   |
|       | JNZ       | LOOP    | ; If not zero jump to LOOP.                            |
|       | MOV L,    | D       | ; Store the quotient in L-register.                    |
|       | SHLD      | 2104 H  | ; Store the result in 2104 H and 2105 H.               |
|       | HLT       |         | ; Stop processing.                                     |

## 6.8 MISCELLANEOUS PROGRAMS

**Example 6.42.** Write an ALP for 8085 to find the square of a given number stored in memory location 2101 H and the result is to be stored in memory location 2102 H. Use Look-up table starting at 2200 H.

**Solution.**

**Main Program:**

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
|       | LXI D,    | 2200 H  | ; Get D-E pair with 2200 H (Starting address of the look-up table). |
|       | LDA       | 2101 H  | ; Get the number in accumulator.                                    |
|       | MOV L,    | A       | ; Load the number to L-register.                                    |
|       | MVI H,    | 00 H    | ; Get 00 H to H-register.   |
|       | DAD D     |         | ; Get effective address in H-L register pair.                       |
|       | MOV A,    | M       | ; Get the result in accumulator.                                    |
|       | SHLD      | 2102 H  |   |
|       | HLT       |         |   |
|       | END       |         |   |

|     |        |   |
|-----|--------|---|
| STA | 2102 H | ; Store the result in the required memory location. |
| HLT |        | ; Stop processing.                                  |

**Look-up table:**

**Location      Data**

|        |      |
|--------|------|
| 2200 H | 00 H |
| 2201 H | 01 H |
| 2202 H | 04 H |
| 2203 H | 09 H |
| 2204 H | 10 H |
| 2205 H | 19 H |
| 2206 H | 24 H |
| 2207 H | 31 H |
| 2208 H | 40 H |
| 2209 H | 51 H |
| 220A H | 64 H |
| 220B H | 79 H |
| 220C H | 90 H |
| 220D H | A9 H |
| 220E H | C4 H |
| 220F H | E1 H |

**Example 6.43.** Write an ALP for 8085 to find the square of a given number stored in memory location 2101 H and the result is to be stored in memory location 2102 H. Use the addition of successive odd integers.

**Solution.**

**Main Program:**

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
|       | LDA       | 2101 H  | ; Get the number in accumulator.                                     |
|       | MVI L,    | 00H     | ; Load 00 H to L-register.   |
|       | ORA L     |         | ; Find if number is zero.  |
|       | JZ        | END     | ; If number is zero then no need to find the square and jump to END. |
|       | MOV C,    | A       | ; Move the number to C-register.                                     |
|       | MOV A,    | L       | ; Move the content of L-reg to accumulator so that square is zero.   |
|       | MVI B,    | 01 H    | ; Move 01 to B-register (first odd number).                          |
| LOOP  | ADD B     |         | ; Add next odd number.   |
|       | INR B     |         | ; Find next odd number to add  |
|       | INR B     |         | ; 02.  |
|       | DCR C     |         | ; Decrement C-register contents.                                     |
|       | JNZ       | LOOP    | ; Continue if not zero.  |
| END   | STA       | 2102 H  | ; Store the result in the memory location 2102 H.                    |

|   |                  |                |   |
|---|------------------|----------------|---|
|   | HLT              |                | ; Stop processing.  |
| <b>Example 6.44.</b> Write an ALP for 8085 to find the square root of a given number (perfect positive square) stored in memory location 2101 H and the result is to be stored in memory location 2102 H. Use the subtraction of successive odd integers. |                  |                |   |
| <b>Solution.</b>  |                  |                |   |
| <b>Main Program:</b>  |                  |                |   |
| <b>Label</b>  | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>   |
|   | LDA              | 2101 H         | ; Get the number in accumulator.  |
|   | ORA A            |                | ; Find if number is zero.   |
|   | JZ               | END            | ; If number is zero then no need to find the square root and jump to END. |
|   | MVI B,           | 01 H           | ; Move 01 to B-register (first odd number).                               |
| LOOP  | MVI C,           | 01 H           | ; Initial value of square root as one.                                    |
|   | SUB B            |                | ; Subtract next odd number.   |
|   | JZ               | END            | ; If number is zero then jump to END.                                     |
|   | INR B            |                | ; Find next odd number to add   |
|   | INR B            |                | ; 02.   |
|   | INR C            |                | ; Decrement square root value by 1.                                       |
|   | JNC              | LOOP           | ; Continue if no carry.   |
| END   | MOV A,           | C              | ; Store the result in accumulator.  |
|   | STA              | 2102 H         | ; Store the result in the memory location 2102 H.                         |
|   | HLT              |                | ; Stop processing.  |

**Example 6.45.** Write an ALP for 8085 to find the Fibonacci series. Sixteen terms of this series are to be store in the memory locations starting at 2101 H. Let 00 H and 01 H data are stored in memory locations 2101 H and 2102 H respectively before the execution of the program.

**Solution.** The terms of Fibonacci series are given as:

$$1, 1, 2, 3, 5, 8, 13, 21, \dots$$

This sequence is defined by assuming first two terms have the same value 1, and each term afterwards is the sum of the previous two terms, that is,

$$1 + 1 = 2, \quad 1 + 2 = 3, \quad 2 + 3 = 5, \quad 3 + 5 = 8, \text{ and so on.}$$

The required terms of the

**Main Program:**

|              |                  |                |   |
|--------------|------------------|----------------|---|
| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>   |
|              | MVI C,           | 10 H           | ; Decimal number 16 (10 H) is stored in C-register which will be used as counter. |
|              | LXI H,           | 2100 H         | ; Get H-L pair with 2100 H.   |
|              | MOV A,           | M              | ; Move first data to accumulator.   |
|              | INX H            |                | ; Point to next data.   |
| LOOP         | ADD M            |                | ; Get next term of the series.  |

|          |  |  |
|----------|--|--|
| INX H    |  | ; Point to the next memory location for storing the next term. |
| MOV M, A |  | ; Store the next term.   |
| DCX H    |  | ; Get previous term of the Fibonacci series.                   |
| MOV A, M |  | ; Get this term in accumulator.                                |
| INX H    |  | ; Point to the next memory location.                           |
| DCR C    |  | ; Decrement count.   |
| JNZ LOOP |  | ; If not zero jump to LOOP.                                    |
| HLT      |  | ; Stop processing.   |

Data stored before the execution of the program.

|        |      |
|--------|------|
| 2100 H | 00 H |
| 2101 H | 01 H |

### PROBLEMS

1. Write an assembly language program of 8085 to fill the RAM area from 2500 H to 25FF H with a byte 33 H.
2. Sixteen bytes of data are stored in memory locations 2001 H to 2010 H. Write an assembly language program of 8085 to transfer this block of data to 2006 H to 2015 H.  
(Hint: In this case data will be transferred in the reverse order otherwise some of the data will coincide.)
3. Write a program (WAP) in assembly language of 8085 to compare two hexadecimal numbers, stored in 2001 H and 2002 H memory locations, for equality. If the numbers are equal, the number itself will be stored in 2003 H memory location, else 00 H will be stored in the memory location 2003 H.
4. WAP in assembly language of 8085 to test 3<sup>rd</sup> bit (D<sub>3</sub> bit) of a byte stored at 2000 H memory location. If the bit D<sub>3</sub> is zero store 00 at 2101 H else store the same number at 2101 H.
5. Write an assembly language program (ALP) of 8085 to find the logical AND and Logical OR of 26 H and 39 H. Store the result in 2500 H and 2501 H.
6. Write an ALP of 8085 to load EE H to the memory locations starting at 2501 H. The length of data bytes is given in 2500 H memory location.
7. Write an ALP for 8085 to add two 8-bit numbers stored in memory locations 2101 H and 2102 H. The result should be stored in 2103 H and the carry if any should be stored in 2104 H.
8. Write an assembly language program for 8085 to add two 8 bit numbers stored in memory locations 2101 H and 2102 H. The answer is required in decimal form and it should be stored in 2103 H and the carry if any should be stored in 2104 H.
9. Write an ALP for 8085 to add 833 and 776 decimal numbers. The result should be stored in 2101 H to 2103 H memory locations. The memory location 2103 H should store the carry bit if any.  
(Hint: First convert the decimal numbers 833 and 776 to hexadecimal numbers.)
10. Write an ALP for 8085 to subtract an 8 bit number (stored in 2101 H memory location) from another 8 bit number (stored in 2102 H memory location). The

answer should be stored in 2103 H and the borrow bit if any should be stored in 2104 H.

11. Repeat the problem 10 to get the result in decimal form.
12. Write an ALP for 8085 to find smaller of two numbers stored in 2301 H and 2302 H. Store the smaller number in memory location 2303 H.
13. Write an ALP for 8085 to find the smallest of the three numbers stored in memory locations starting at 2301 H. Store the smallest number in 2304 H.
14. Write an ALP for 8085 to find the smallest number among a series of 16 numbers stored in the memory locations starting at 2301 H. The number 16 (10 H) is stored in 2300 H. The smallest number should be stored in 2401 H.
15. Write an ALP to find 13 terms of Fibonacci series. The terms of the series are to be stored in the memory locations starting at 2501 H.
16. Write an ALP for 8085 to shift an 8-bit number left by two bits. The number is stored in memory location 2501 H and the result is to be stored in 2502 H memory location.  
(Hint: Keep the number in accumulator and use ADD A instruction twice.)
17. Write an ALP for 8085 to shift a 16-bit number left by two bits. The number is stored in memory locations 2501 H and 2502 H. The result is to be stored in 2503 H and 2504 H memory locations.

# 7

## Interrupt Instructions of 8085

---

In the preceding chapters the architecture, instruction set and programming details of 8085 microprocessor were discussed. The microprocessor forms the parts of microcomputers. Many complicated and sophisticated operations can be performed by using the microprocessor based systems. These systems have the facilities to transfer the data from the microprocessor to the outside world or vice-versa. The transfer of data takes place through the data bus with the help of address and control buses. Thus special circuitry is needed to convert data from a wide range of input devices into suitable form for microprocessor buses. The circuitries used between the microprocessor and input/output devices are known as interfaces. This chapter deals how the external devices (peripherals) get connected to the microprocessor. In addition, different ways of data transfer from the microprocessor to the peripheral devices and vice-versa, and the restart instructions will also be discussed.

### 7.1 METHODS OF I/O OPERATIONS

There are two ways for the transfer of data from microprocessor to I/O devices and vice-versa.

1. Memory Mapped I/O
2. I/O Mapped I/O or Isolated I/O

#### 7.1.1 Memory Mapped I/O

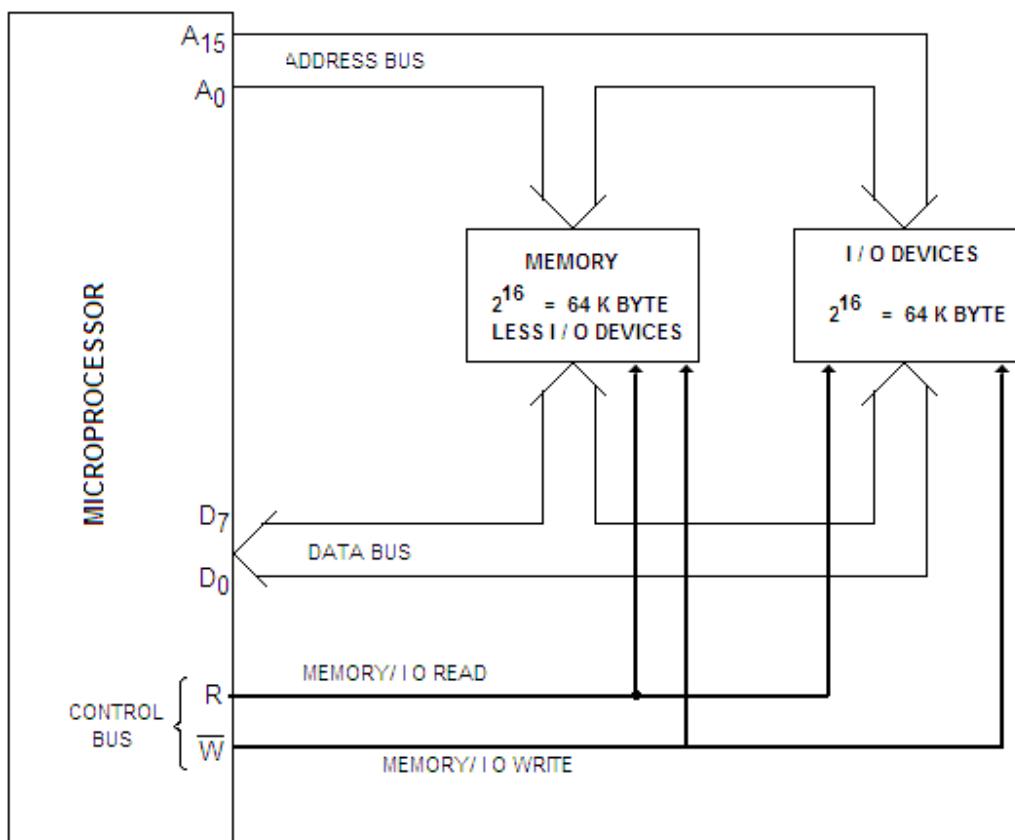
Memory Mapped I/O system is used in smaller system. In this system there is only one address space, some addresses are assigned to memories and some addresses to I/O devices i.e. the addresses to I/O devices are different from the addresses which have been assigned to memories. In other words the I/O devices and memory share the memory map, with some address reserved for the I/O devices and rest for the memory. This scheme is shown in figure 7.1. The microprocessor use R/W signals to determine the direction of data flow. The main advantage of this scheme is that it does not require additional decoding circuitry and the set of instructions may be used to fetch data either from the I/O devices or from the memory locations.

Following instructions may be used for the data transfer in this scheme:

MOV M, A

It moves the contents of accumulator to  $M_{H-L}$ . If H-L pair contains the address of memory location, then the data will be transferred to memory location; if on the other hand H-L pair contains the address of the I/O devices, then the accumulator data will be transferred to I/O devices.

|             |  |
|-------------|--|
| MOV A, M    | It moves the contents of $M_{H-L}$ to accumulator. If H-L pair contains the address of memory location, then the data will be transferred from memory location; similarly if H-L pair contains the address of the I/O devices, then the data from I/O devices will be transferred to accumulator.  |
| STA address | It stores the contents of accumulator to addressed location. If the address represents the address of memory location then the accumulator contents will be stored to memory location; if on the other hand address represents the address of the I/O devices, then the accumulator data will be transferred to I/O devices.                     |
| LDA address | It stores the addressed contents to the accumulator. If the address represents the address of memory location then the contents stored in memory location will be transferred to accumulator; if on the other hand address represents the address of the I/O devices, then the data from the I/O devices will be transferred to the accumulator. |



**Fig. 7.1**

Many other instructions like LDAX, STAX and other arithmetic and logic instructions etc may also be used in this memory mapped I/O systems.

#### 7.1.2 I/O Mapped I/O or Isolated I/O

Larger systems use this technique for the data communication to I/O devices. Figure 7.2 shows this system used in 8085 microprocessor. In this technique the

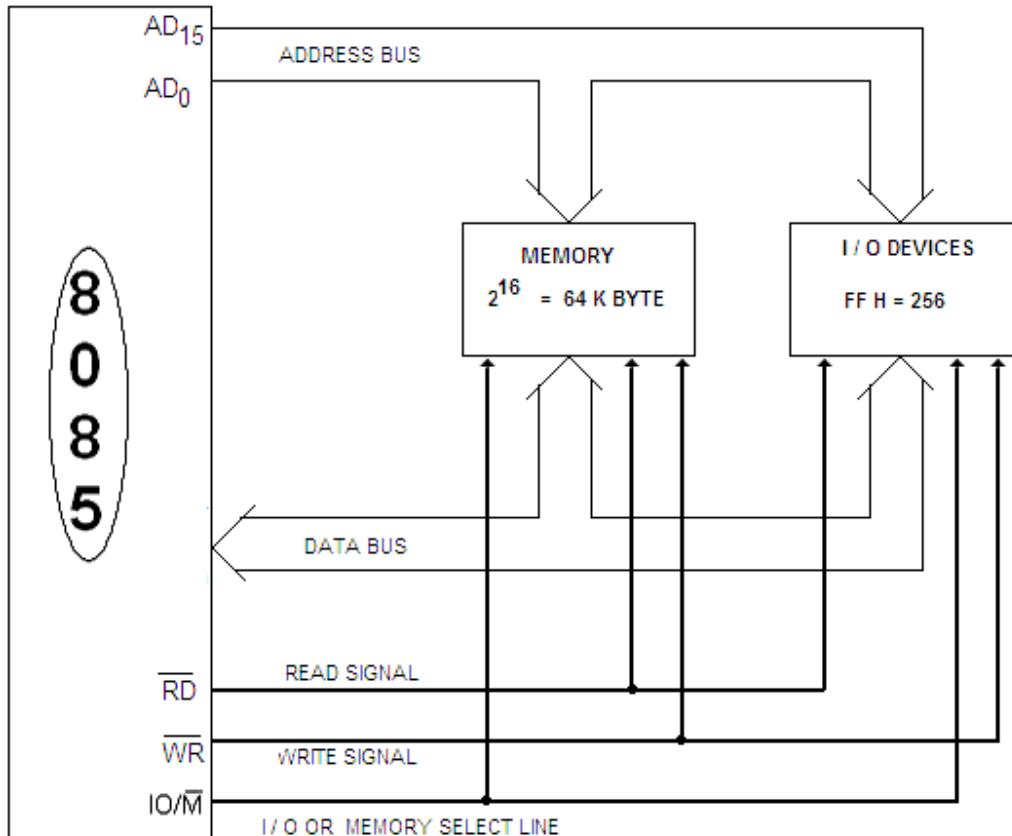


Fig 7.2

microprocessor treats the memory and I/O devices separately, using different control lines for each of them. In 8085A the  $IO/M$  signal is used for this purpose. If this signal is low (0), it represents the memory. However, if this signal is high (1), it represents the I/O operations.  $RD$  and  $WR$  signals in association with  $IO/M$  signal help in performing I/O read/write operations or memory read/write operations. Table 7.1 shows the operations of these signals.

Table 7.1

| $RD$ | $WR$ | $IO/M$ | Operations          | Address      |
|------|------|--------|---------------------|--------------|
| 0    | 1    | 0      | $MEMR$ Memory Read  | $A_{15}-A_0$ |
| 1    | 0    | 0      | $MEMW$ Memory Write | $A_{15}-A_0$ |
| 0    | 1    | 1      | $IOR$ I/O Read      | $A_7-A_0$    |
| 1    | 0    | 1      | $IOW$ I/O Write     | $A_7-A_0$    |

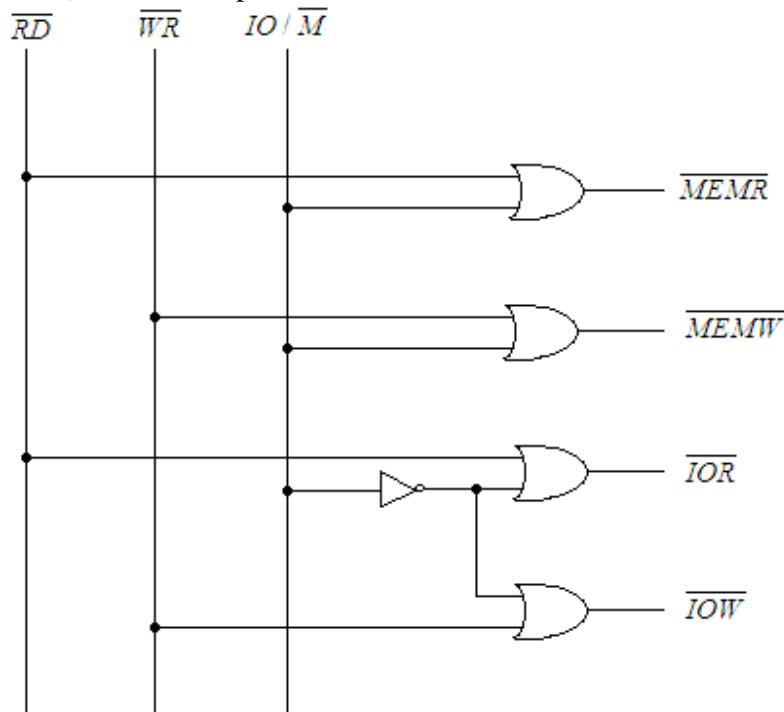
Figure 7.3 shows the generation of  $MEMR$  (Memory Read),  $MEMW$  (Memory Write),  $IOR$  (I/O Read) and  $IOW$  (I/O Write) signals.

In this scheme, special instructions are used to perform the I/O operations. The special instructions are:

**IN XX H** (XX H is the Port address of 1 byte , which is in between 00 H to FF H)

**OUT XX H** (XX H is the Port address of 1 byte , which is in between 00 H to FF H)

The address XX H of the I/O port is duplicated on both the high and low order address bus and the signal  $IO/M$  ensures that the memory location corresponding to that address (XXXX H) does not respond.



**Fig. 7.3**

Comparison of Memory Mapped I/O and I/O Mapped I/O is given in table 7.2.

| Memory Mapped I/O  | I/O Mapped I/O  |
|--|---|
| 1. Large Number of instructions like MOV A, M, MOV M, A, LDA, STA etc are used for the data transfer.  | Only two instructions IN Port, OUT Port are used for the data transfer from microprocessor to I/O devices and vice versa. |
| 2. Data transfer is possible between any register of C.P.U. and I/O devices.   | Data transfer is possible only between Accumulator and I/O devices.   |
| 3. Arithmetic and logic operations can be performed with the data of I/O Ports using with the instructions like ORA M, XRA M, ANA M, ADD M, SUB M etc. | Arithmetic and logic operations can not be performed in this scheme.  |

|  |  |
|--|--|
| <p>4. Memory mapping of 64 K byte is shared between system memory and I/O ports.</p> <p>5. 16-bit address lines are needed.</p> <p>6. It takes different T-states depending on the instructions.</p> | <p>256 (FF H) I/O devices may be interfaced.</p> <p>8-bit address lines are needed.</p> <p>It takes 10 T-states for its execution.</p> |
|--|--|

## 7.2 DATA TRANSFER SCHEMES

In microprocessor based systems several input / output devices are connected and data transfer may take place between microprocessor and memory, microprocessor and I/O devices and memory & I/O devices. Not much of the problems arise for the data communication between microprocessor and memory as same technology is used in the manufacturing of memory and microprocessor. The speed of the memory is almost compatible with the speed of microprocessor. But for the data transfer between the microprocessor and I/O devices, the problems arise due to mismatch of the speed of the I/O devices and the speed of microprocessor or memory. To overcome the problem of speed mismatch between the microprocessor and I/O devices following data transfer schemes (fig. 7.4) may be considered. The data transfer schemes were categorized depending upon the capabilities of I/O devices for accepting serial or parallel data.

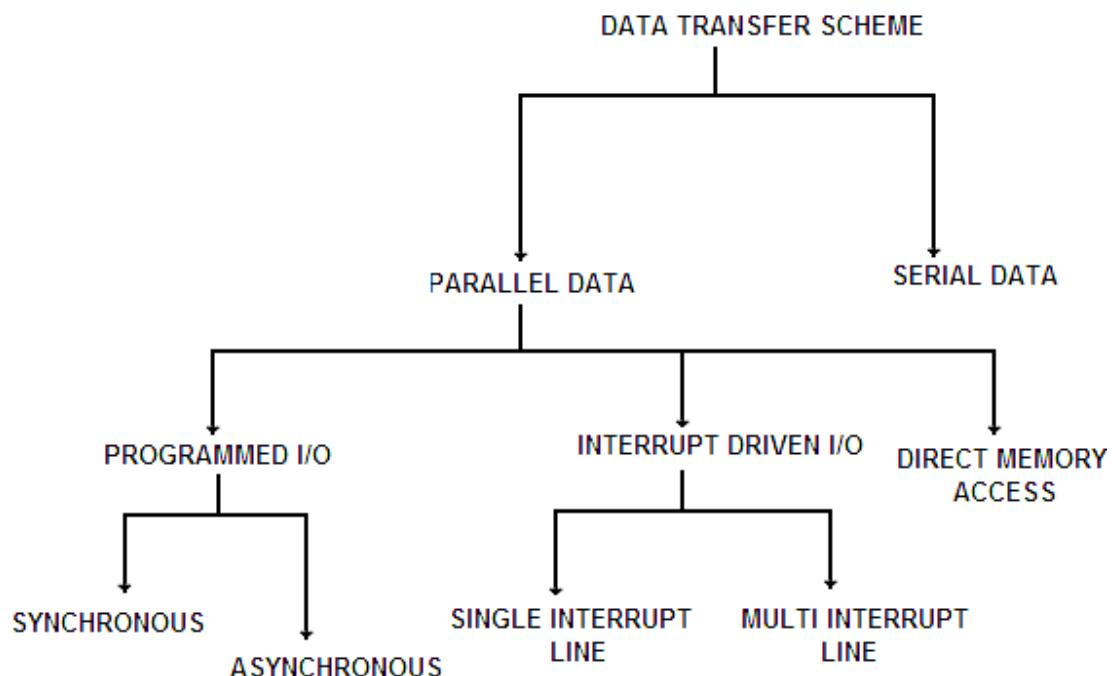


Fig. 7.4

The 8085 microprocessor is a parallel device i.e. it transfers eight bits of data simultaneously over eight data lines (parallel I/O mode). However in many situations, the parallel I/O mode is either impractical or impossible. For example, parallel data communication over a long distance becomes very expensive. Similarly, parallel data communication is not possible with devices such as CRT terminal or Cassette tape etc. For these devices, therefore, serial I/O mode is used which transfer a single bit on a single line at a time. For serial data transmission, 8-bit parallel word is converted to a stream of eight serial bit using parallel-to-serial conversion. Similarly, in serial reception of data, the microprocessor receives a stream of 8-bit one by one which are then converted to 8-bit parallel word using serial-to-parallel conversion. The parallel data transfer will now be discussed in the following sub-sections.

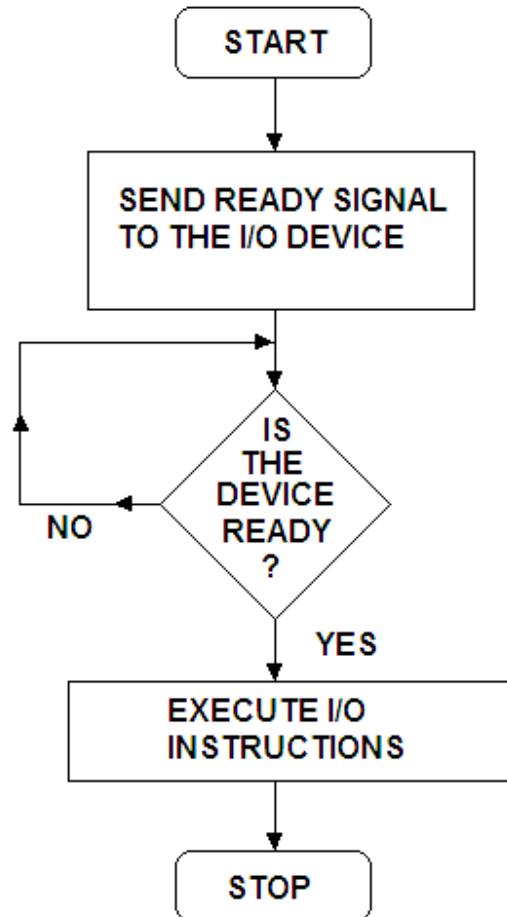
### **7.2.1 Programmed I/O Data Transfer**

This method of data transfer is generally used in the simple microprocessor systems where speed is unimportant. This method uses instructions to get the data into or out of the microprocessor. The data transfer can be synchronous or asynchronous depending upon the type and the speed of the I/O devices.

Synchronous type of data transfer can be used when the speed of the I/O devices matches with the speed of the microprocessor. The common clock pulse synchronizes the microprocessor and the I/O devices. In such data transfer scheme because of the matching of the speed, the microprocessor does not have to wait for the availability of the data; the microprocessor immediately sends data for the transfer as soon as the microprocessor issues a signal.

The asynchronous data transfer method is used when the speed of the I/O devices is slower than the speed of the microprocessor. Because of the mismatch of the speed, the

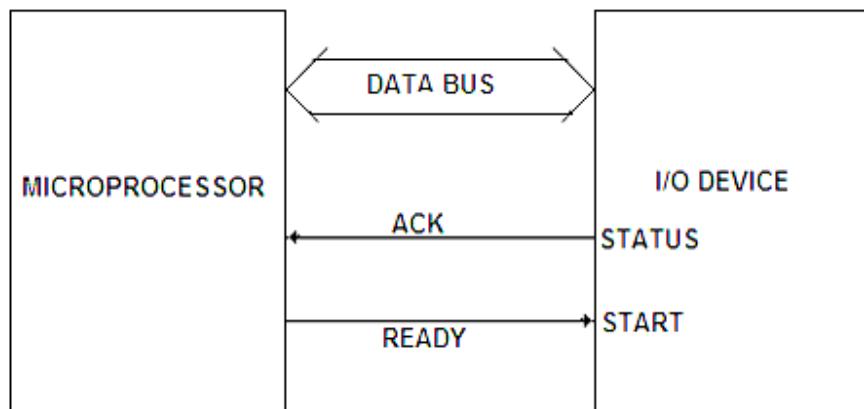
internal timing of the I/O device is independent from the microprocessor and thus two



**Fig. 7.5**

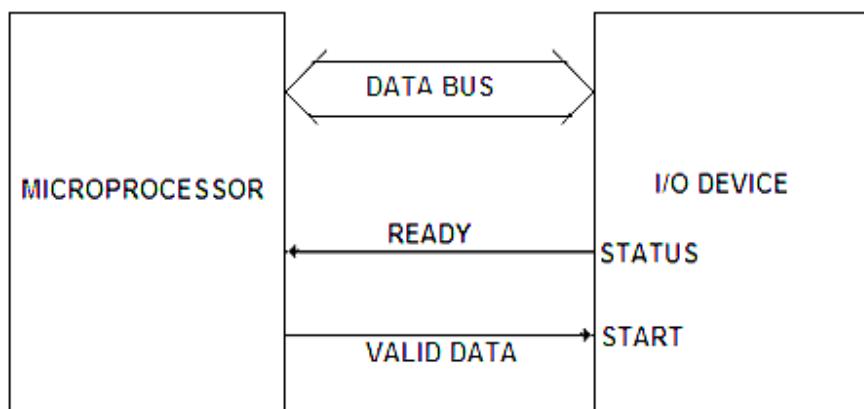
units are said to be asynchronous to each other. The asynchronous data transfer is normally implemented using ‘handshaking’ mode. In the handshaking mode some signals are exchanged between the I/O device and microprocessor before the data transfer takes place. The microprocessor has to check the status to the input/output device, if the device is ready for the data transfer. The microprocessor initiates the I/O device to get ready; the status of the I/O device is continuously checked by the microprocessor till the I/O device becomes ready, the microprocessor sends instructions to transfer the data. Flow chart for this mode of data transfer is shown in figure 7.5.

Fig. 7.6 illustrates the asynchronous handshaking process to transfer the data from the microprocessor to I/O device. In this figure, the microprocessor sends a ready signal to I/O device. When the device is ready to accept the data, the I/O device sends an ‘ACK’ (Acknowledge) signal to microprocessor indicating that the I/O device has acknowledged the ‘Ready’ signal and is ready for the transfer of data.



**Fig. 7.6**

Figure 7.7 shows the asynchronous handshaking process to transfer the data from the I/O device to microprocessor. In this case I/O device issues the ready signal to microprocessor indicating that I/O device is ready to send the data to microprocessor. In response to this signal, valid data signal is sent by the microprocessor to I/O device and then the valid data is put on the data bus for the transfer.

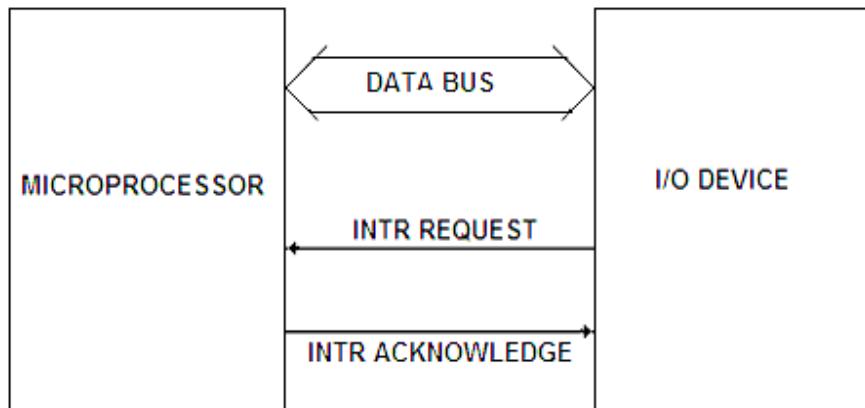


**Fig. 7.7**

### 7.2.2 Interrupt Driven I/O Data Transfer

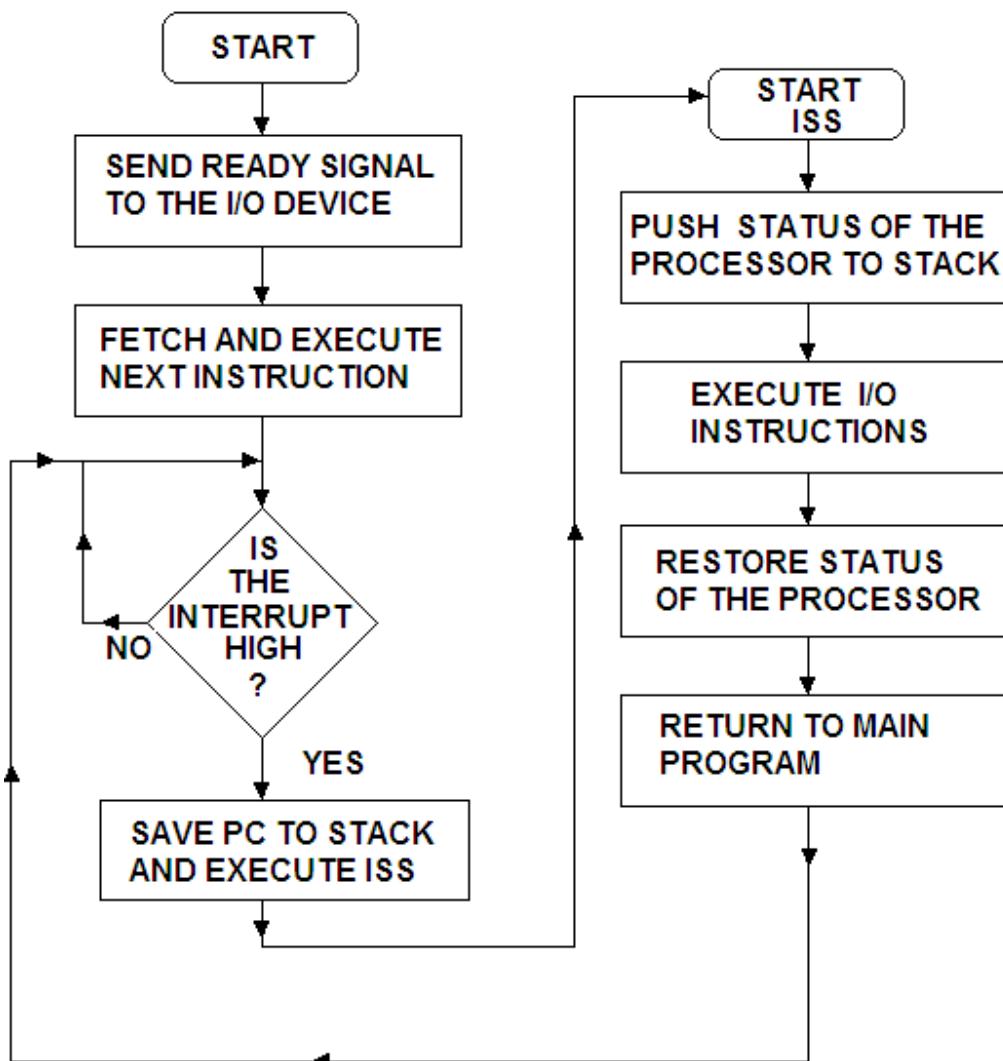
In the programmed I/O data transfer method discussed above, the microprocessor is busy all the time in checking for the availability of data from the slower I/O devices and also in checking if I/O device is ready for the data transfer. In other words in this data transfer scheme, some of the microprocessor time is wasted in waiting while an I/O device is getting ready. The interrupt driven I/O data transfer method is efficient as no microprocessor time is wasted in waiting for an I/O device to be ready. The I/O device informs the microprocessor for the data transfer whenever the I/O device is ready. This is achieved by interrupting the microprocessor. The interrupt is hardware facilities provided on the microprocessor. In the beginning the microprocessor initiates data transfer by requesting the I/O device 'to get ready' and then continue executing its original program rather wasting its time by checking the status of I/O device. Whenever the device is ready to accept or supply data, it informs the processor through a control signal known as

interrupt signal. In response to this interrupt signal, the microprocessor sends back an interrupt acknowledge signal to the I/O device indicating that it received the request (Fig. 7.8). It then suspends its job after executing the current instruction. It saves the contents



**Fig. 7.8**

of program counter and status to stack and jumps to the subroutine program. This subroutine program is called Interrupt Service Subroutine (ISS) program. The ISS saves the processor status into stack; and after executing the instruction for the data transfer, it restores the processor status and then returns to main program. The flow chart for this method of data transfer is shown in figure 7.9.



**Fig. 7.9**

As already discussed, several input/output devices may be connected to microprocessor using Interrupt Driven Data Transfer Scheme. Following interrupt request configuration may arise while interfacing the I/O devices to microprocessor.

1. Single Interrupt system
2. Multi Interrupt System

#### 1. Single Interrupt System

When only one interrupt line is available with the microprocessor and several I/O devices are to be connected, then the method is known as Single Interrupt System. Figure 7.10(a) shows the way to connect several devices to one active low input interrupt terminal (*INTR*) of the microprocessor. However, to connect the several I/O devices to active high interrupt terminal (*INTR*) is shown in figure 7.10(b). In the active low interrupt line of the microprocessor the devices are connected to *INTR* terminal through different open collector NOT gates; when any of the devices is active it provides a low

signal to  $\overline{INTR}$  enabling the interrupt line. Similarly, in the active high interrupt line the I/O devices are connected through an OR gate. When any of the device is high the output of OR gate sends a high signal to interrupt line (INTR).

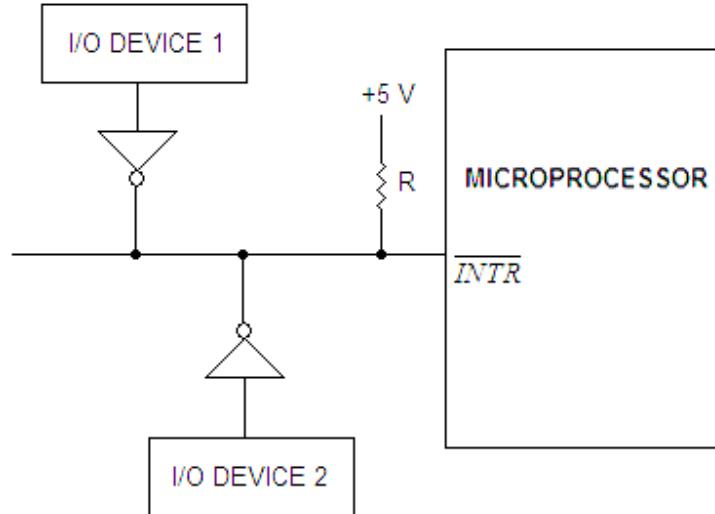


Fig. 7.10(a)

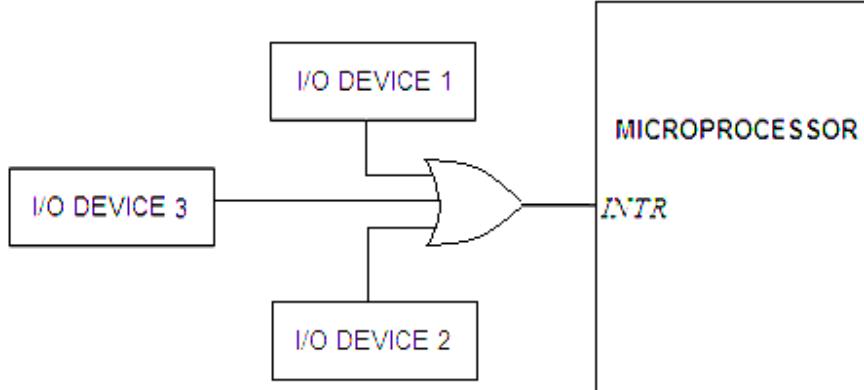
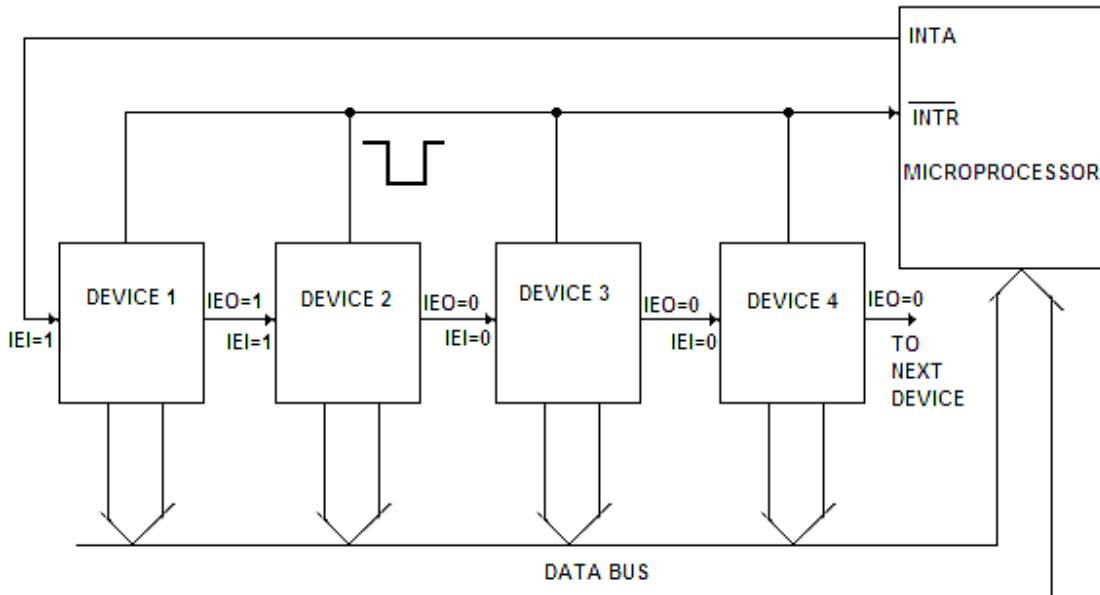


Fig. 7.10(b)

When the interrupt line is active in either of the two methods discussed above, then the microprocessor will not know which device has sent the interrupt signal. Three techniques are commonly used to solve the problem for the microprocessor that is requesting the interrupt and resolving any simultaneous requests by two or more devices. These are:

- Polling:** The interrupt signal from each device can be used to set one bit of a register wired as an input port. When an interrupt occurs, the ISS polls this port to see who requested service. A priority is automatically established by the order of polling. This technique is very simple but has the negative of degrading response time.
- Daisy Chain:** This technique has been shown in figure 7.11. In this method each I/O device has an Interrupt Enable Input (IEI) and an Interrupt Enable Output (IEO). An interrupt request can be made only if IEI is high. A serial

connection like a chain of all the I/O devices is made. The highest priority I/O device is placed at the first position followed by the lower priority devices in sequence. If any device sends an interrupt signal i.e. low to the interrupt line, then the INTR line of the processor is enabled. The interrupt acknowledge signal (INTA) is enabled in response to low INTR line. In figure 7.11, device 2 is requesting an interrupt causing its IEO to be low. This in turn disables devices 3 and 4. It may be noted that device 1 is still able to request an interrupt because it has a higher priority. The interrupt acknowledge signal can be used to reset IEO of the interrupting device.



**Fig. 7.11**

3. **Priority Interrupt Controller (PIC):** In this method several I/O devices may be connected to a single interrupt line through programmable interrupt controller (IC 8259). Up to 8 input/output devices may be connected to the microprocessor. If more than 8 I/O devices to be connected, more PICs (programmable interrupt controllers) are used in cascade. The details of PIC will be discussed in a later chapter.

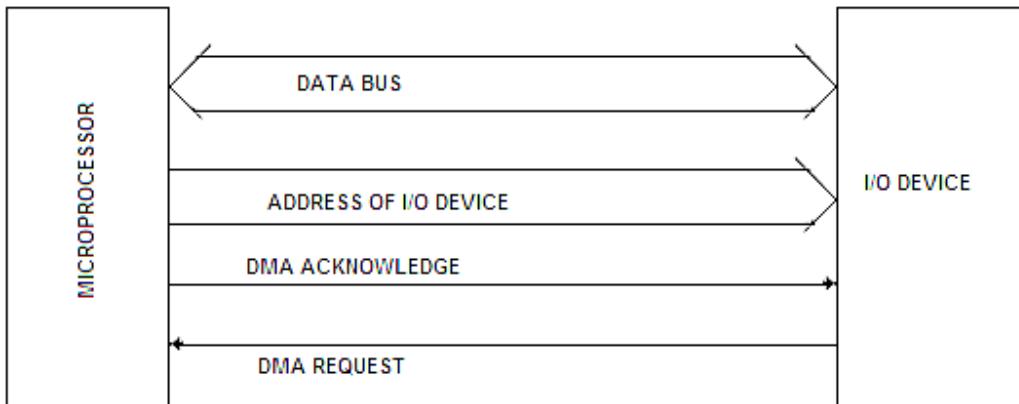
## 2. Multi Interrupt System

When the microprocessor has several interrupt terminals and one I/O device is to be connected to each interrupt terminal, then it is known as multi interrupt system. In this scheme, the number of I/O devices to be connected to the interrupt lines should be equal to or less than the number of interrupt terminals. In this way one device is connected to each level of interrupt. So when a device interrupts the microprocessor, it immediately knows which device has interrupted. Such an interrupt scheme is known as vectored interrupt.

### 7.2.3 Direct Memory Access (DMA) Data Transfer

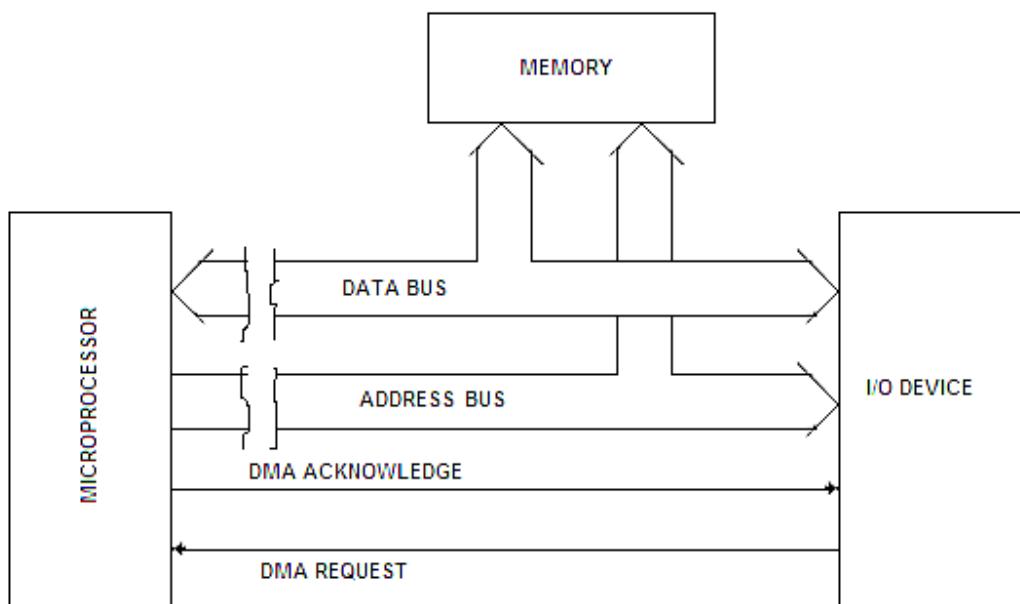
In programmed I/O or interrupt driven I/O methods of data transfer between the I/O devices and external memory is via the accumulator. For bulk data transfer from I/O

devices to memory or vice-versa, these two methods discussed above are time consuming and quite uneconomical even though the speed of I/O devices matches with the speed of microprocessor; since the data is first transferred to accumulator and then to concerned device. The Direct Memory Access (DMA) data transfer method is used for bulk data transfer from I/O devices to microprocessor or vice-versa. In this method I/O devices are



**Fig. 7.12**

allowed to transfer the data directly to the external memory without being routed through accumulator. For this the microprocessor relinquishes the control over the data bus and address bus, so that these can be used for transfer of data between the devices. For the data transfer using DMA process, a request to the microprocessor by the I/O device is sent and on receipt of such request, the microprocessor relinquishes the address and data buses and informs the I/O devices of the situation by sending Acknowledge signal as shown in figures 7.12 and 7.13. The I/O device withdraws the request when the data transfer between the I/O device and external memory is complete.

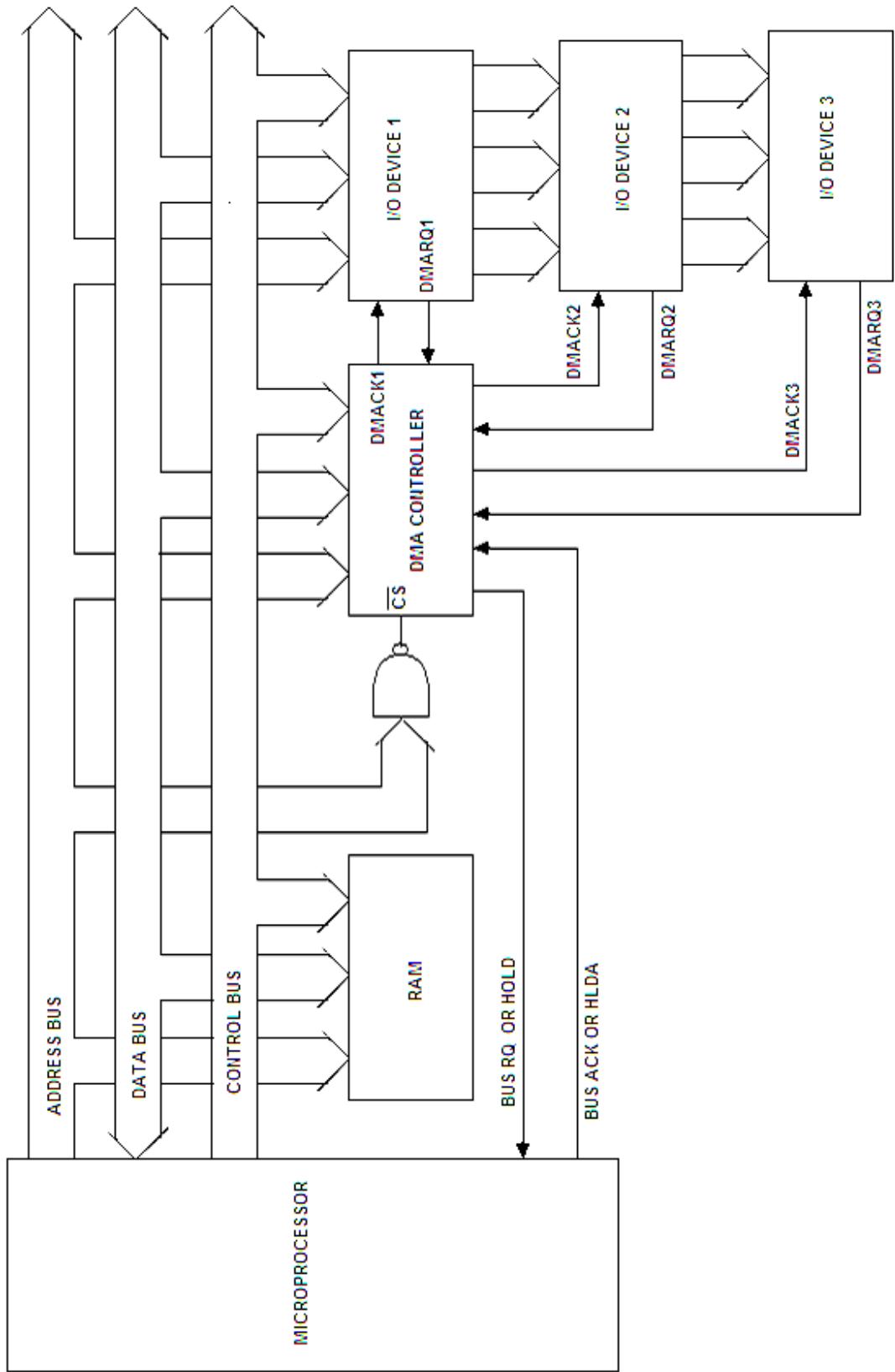


**Fig. 7.13**

It may be mentioned here that DMA transfer the data of the following types:

- Memory to I/O device
- I/O device to memory
- Memory to memory
- I/O device to I/O device

For transferring the data through DMA, an interfacing chip known as DMA Controller is used with the microprocessor that helps to generate the addresses for the data to be transferred from the I/O devices (Fig. 7.14). The peripheral device sends the request signal (DMARQ) to the DMA controller and the DMA controller in turn passes it to the microprocessor (HOLD signal). On receipt of the DMA request the microprocessor sends an acknowledge signal (HLDA) to the DMA controller. On receipt of this signal (HLDA) the DMA controller sends a DMA acknowledge signal (DMACK) to the I/O device. The DMA controller then takes over the control of the buses of microprocessor and controls the data transfer between RAM and I/O device. When the data transfer is complete, DMA controller returns the control over the buses to the microprocessor by disabling the HOLD and DMACK signals.



### 7.3 THE 8085 INTERRUPTS

As already discussed, in Interrupt Driven I/O data transfer methods the microprocessor gets interrupted by the I/O device when it is ready to transfer the data. The microprocessor suspends its job after executing the current instruction. It saves the contents of program counter to stack and jumps to the subroutine program. This subroutine program is called Interrupt Service Subroutine (ISS) program. The ISS saves the processor status into stack; and after executing the instruction for the data transfer, it restores the processor status and then returns to main program. The ISS executes the relevant set of instructions stored at a predetermined memory block. The Intel 8085 microprocessor has five hardware interrupt inputs on its chip. Besides these, the microprocessor has eight interrupt instruction in the instruction set. The microprocessor responds to both the hardware and software interrupts. In the following sections the details of both software and hardware interrupts will be discussed.

### 7.4 SOFTWARE INTERRUPTS

The 8085 microprocessor has eight software interrupts namely, RST 0, RST 1, RST 2, RST 3, RST 4, RST 5, RST 6 and RST 7.

The syntax for these interrupt instructions is given by:

**RST n**

where *n* varies from 0 to 7.

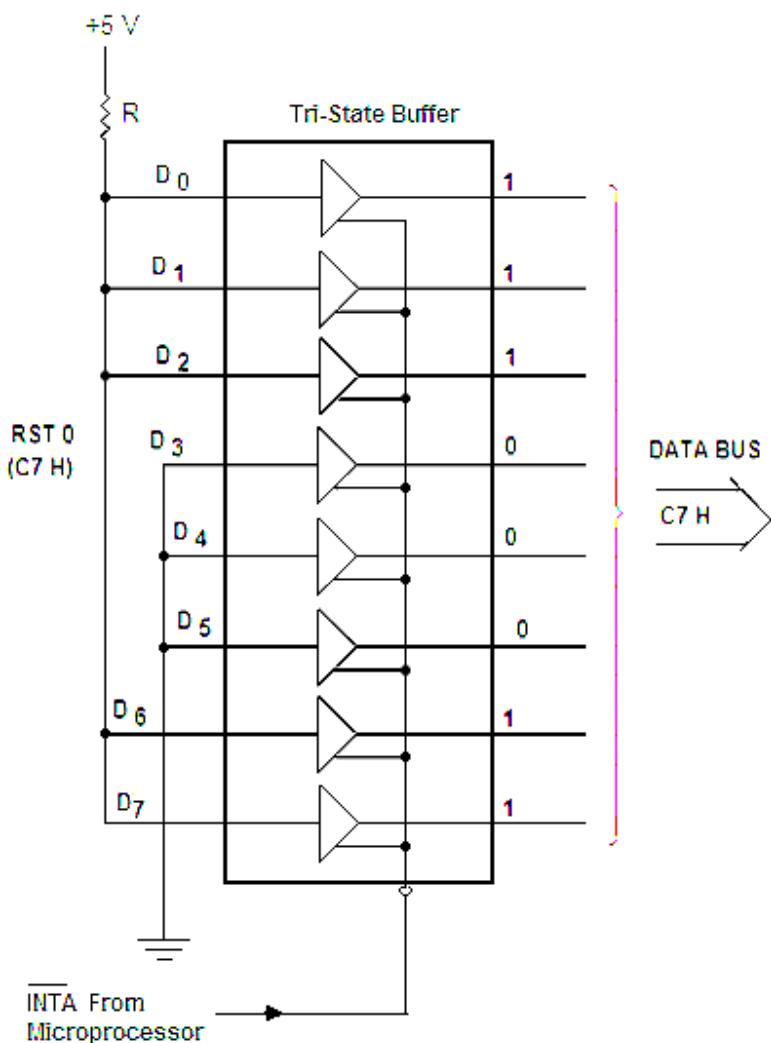
These instructions are single byte instructions. When an interrupt occurs, a CALL instruction to a predetermined location of the memory is executed. The effect of each restart instruction is shown in table 7. 4.

**Table 7.4**

| Instruction | Effect      | Op Code | Binary equivalent | Vector Location |
|-------------|-------------|---------|-------------------|-----------------|
| RST 0       | CALL 0000 H | C7      | 1100 0111         | 0000 H          |
| RST 1       | CALL 0008 H | CF      | 1100 1111         | 0008 H          |
| RST 2       | CALL 0010 H | D7      | 1101 0111         | 0010 H          |
| RST 3       | CALL 0018 H | DF      | 1101 1111         | 0018 H          |
| RST 4       | CALL 0020 H | E7      | 1110 0111         | 0020 H          |
| RST 5       | CALL 0028 H | EF      | 1110 1111         | 0028 H          |
| RST 6       | CALL 0030 H | F7      | 1111 0111         | 0030 H          |
| RST 7       | CALL 0038 H | FF      | 1111 1111         | 0038 H          |

These RST instructions are like vectors because they point to specific locations in the memory. The starting address of each instruction is called a **Vector Location**. The vector location for RST 0 is 0000 H and for RST 1 is 0008 H and so on. The vector locations of each instruction are 8 bytes apart. Therefore 8 bytes of instructions can be stored beginning at any vector location.

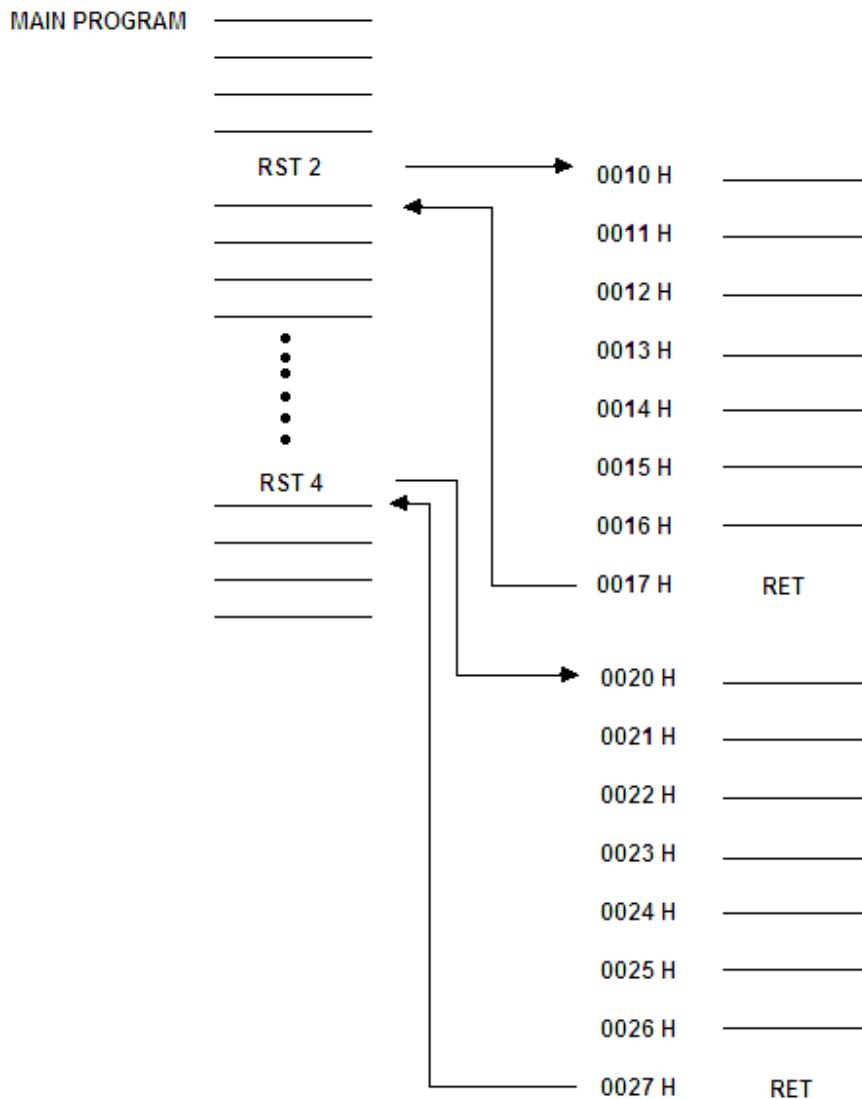
Figure 7.15 shows the hardware for the implementation of the software interrupt instruction. This circuit is for the implementation of an instruction RST0. In response to the interrupt request signal, the 8085 microprocessor sends the *INTA* (interrupt acknowledge) signal, which is used to enable the buffer. The hex code C7 H of RST 0 will be placed on the data bus.



**Fig. 7.15**

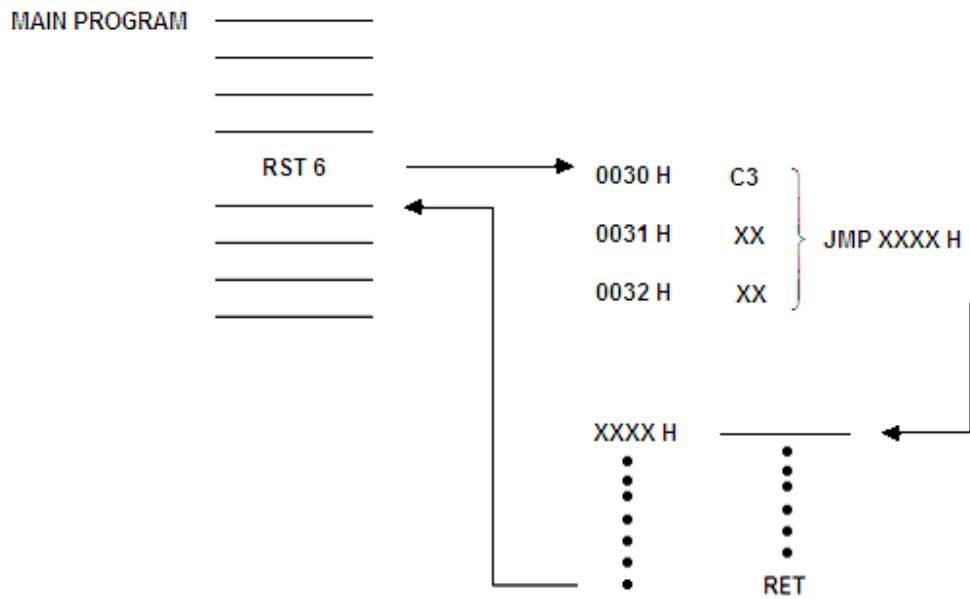
Figure 7.16 illustrates how software interrupt instructions are executed. Suppose in the main program RST2 instruction is given, and when this instruction is executed the contents of the program counter is pushed onto the stack. Then the program counter jumps to the address 0010 H. The subroutine located from 0010 H to 0017 H is executed, with the RET instruction as the last instruction of the subroutine program. So as the RET instruction is executed it returns to the main program.

Similarly, if another restart instruction RST4 is encountered in the main program, then the contents of the program counter are again pushed onto the stack. The program jumps to the subroutine program whose starting address is given by 0020 H. After the execution of this subroutine program it returns to the main program as shown in figure 7.16.



**Fig. 7.16**

From the above discussion it is clear that the software interrupt instructions are basically special kind of CALL instructions; when any of these instructions is executed it branches to the predetermined address. Notice that all these instructions are of one byte (ref. table 7.4) while the standard CALL instruction is of three bytes. The vector addresses of these 8 software interrupts are 8 bytes apart. So 8 bytes of instructions can be stored in the subroutine program associated with each of these instructions. But generally, the subroutine programs may require more than 8 bytes. For this reason the complete subroutine program are usually not stored in the vector locations, rather the vector locations are used to specify the starting address of the longer subroutine program i.e. in the vector location of these instructions JMP XXXX H may be stored. The XXXX H represents the starting address of the longer subroutine program. The last instruction of the program will be RET, so that after completing the subroutine program it jumps to the main program as shown in figure 7.17.



**Fig. 7.17**

The hardware solution to multi-interrupt problem is carried out by the circuit shown in figure 7.18. The multi-interrupt problem means the possibility of two requests arriving simultaneously. This circuit will accept eight interrupt request and work as per their priority and separate RST instruction for each request is generated. For the generation of all eight RST instructions, a 3-to-8 priority encoder (IC 74LS148) along with a tri-state octal buffer (IC 74LS244) is used as shown in figure 7.18.

The 3-to-8 priority encoder generates a 3 bit binary code corresponding to active input. If two or more inputs are active simultaneously, the highest numbered input will be encoded. The three bits are inverted by the inverted buffer and then applied to an octal buffer (IC 74LS244). The op code corresponding to the input will be generated which is latched on to the data bus if *INTA* signal is active.

The op codes for each RST instruction given in table 7.4 are reproduced below:

| Instruction | Op Code | Binary equivalent |
|-------------|---------|-------------------|
| RST 0       | C7      | 1100 0111         |
| RST 1       | CF      | 1100 1111         |
| RST 2       | D7      | 1101 0111         |
| RST 3       | DF      | 1101 1111         |
| RST 4       | E7      | 1110 0111         |
| RST 5       | EF      | 1110 1111         |
| RST 6       | F7      | 1111 0111         |
| RST 7       | FF      | 1111 1111         |

It may be noted from this op code table that D<sub>0</sub> to D<sub>2</sub> and D<sub>6</sub> to D<sub>7</sub> bits of the op codes are same for all the RST instructions (RST 0 to RST 7). The bits D<sub>3</sub> to D<sub>5</sub> are different and are in the sequence of 3 bit binary numbers for the RST 0 to RST 7. In general the code for RST instructions may be written as:

11CCC111

where CCC is

- 000 for RST 0
- 001 for RST 1
- 010 for RST 2
- 011 for RST 3
- 100 for RST 4
- 101 for RST 5
- 110 for RST 6
- 111 for RST 7

This sequence permits to use the 3-to-8 priority encoder. It can be understood that a signal say  $\overline{INT3}$  is low then  $A_0 A_1 A_2$  will be 100 which will be inverted by the inverter buffer as 011 ( $I_3 I_4 I_5$ ). Since the other lines  $I_0, I_1, I_2, I_6, I_7$  are all high, the octal buffer will provide the op code DF H (op code for RST 3) on the 8085 data bus as soon as the interrupt acknowledge signal ( $INTA$ ) is made low in response to the interrupt request signal. Similarly by activating other inputs of the priority encoder, the vector CALL instructions can be generated on the 8085 data bus.

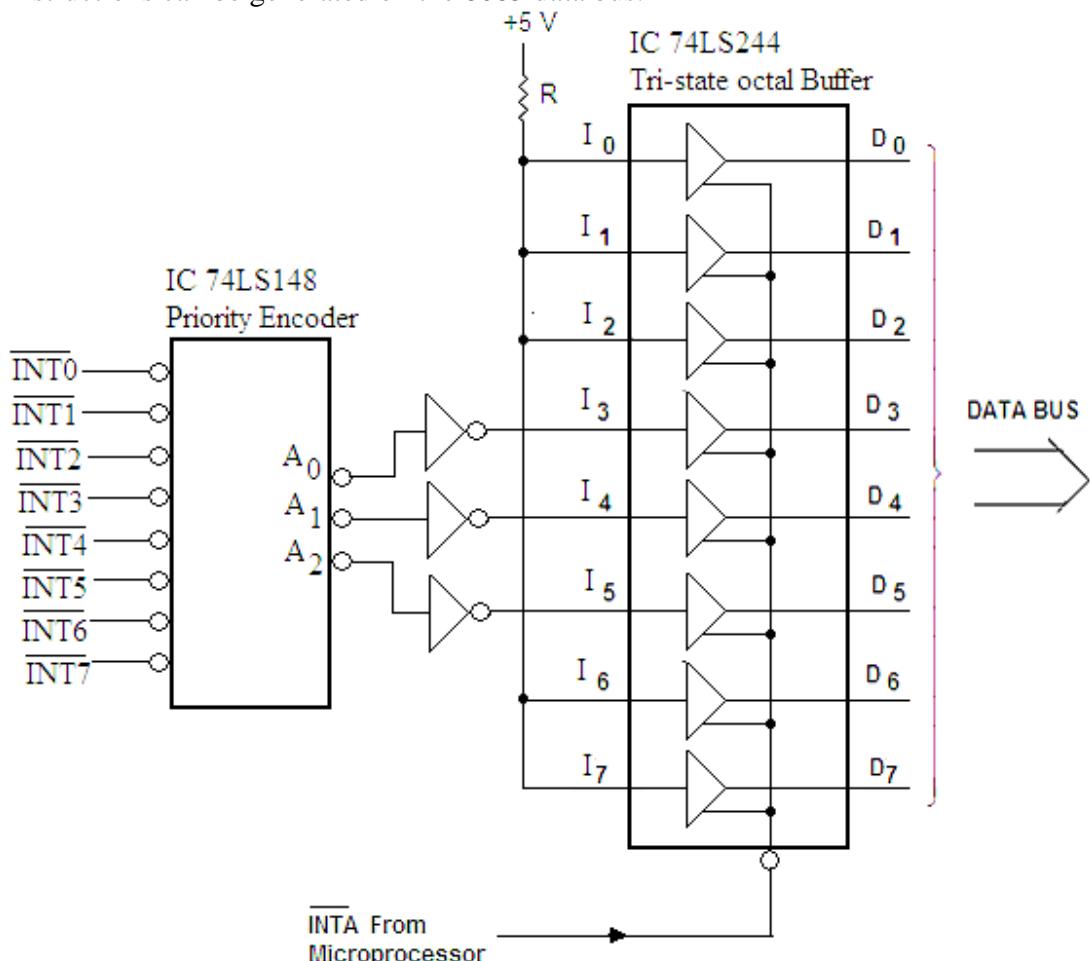


Fig. 7.18

## 7.5 HARDWARE INTERRUPTS

In addition of software interrupts discussed above the 8085 has five hardware interrupts also. For this, five hardware input pins are provided in the microprocessor. The hardware interrupts are initiated by an external device, by placing an appropriate signal at the interrupt pin of the microprocessor. The five interrupts RST 5.5, RST 6.5, RST 7.5, TRAP and INTR are shown in table 7.5. Out of these interrupts RST 5.5, RST 6.5, RST 7.5 and TRAP are vector interrupts and INTR is the non-vectored interrupt. In vector interrupts the microprocessor automatically sends the specific address to program counter in response to an interrupt request signal. However, in non-vectored interrupt, the interrupt device has to give the address of the interrupt service subroutine.

**Table 7.5**

| Interrupt | Priority | Vector Location |
|-----------|----------|-----------------|
| TRAP      | 1        | 0024 H          |
| RST 7.5   | 2        | 003C H          |
| RST 6.5   | 3        | 0034 H          |
| RST 5.5   | 4        | 002C H          |
| INTR      | 5        | -----           |

It may be noted from the table that TRAP has the highest priority, RST 7.5 next highest and so on. If two or more hardware interrupts are served at the same time then the microprocessor executes them in order of their priority level; i.e. TRAP is served first, then RST 7.5 and so on. When highest priority interrupt is executed, the lower priority interrupts remain pending rather they are known as pending interrupts.

The RST 7.5, RST 6.5 and RST 5.5 are the maskable interrupts and TRAP is non-maskable interrupt. The maskable means the prevention of any interrupt. Some times it may be required to prevent one or more interrupts when a certain task is being carried out by the microprocessor; for this the masking of these interrupts is done. The masking of any interrupt is carried out by an instruction known as SIM (Set Interrupt Mask). The SIM is one byte instruction. To find the status of the interrupts i.e. to know which interrupt is masked or which interrupt is pending, another instruction known as RIM (Read Interrupt Mask) is used. These two instructions RIM and SIM will be discussed in section 7.7.

## 7.6 INTERRUPT CONTROL CIRCUIT

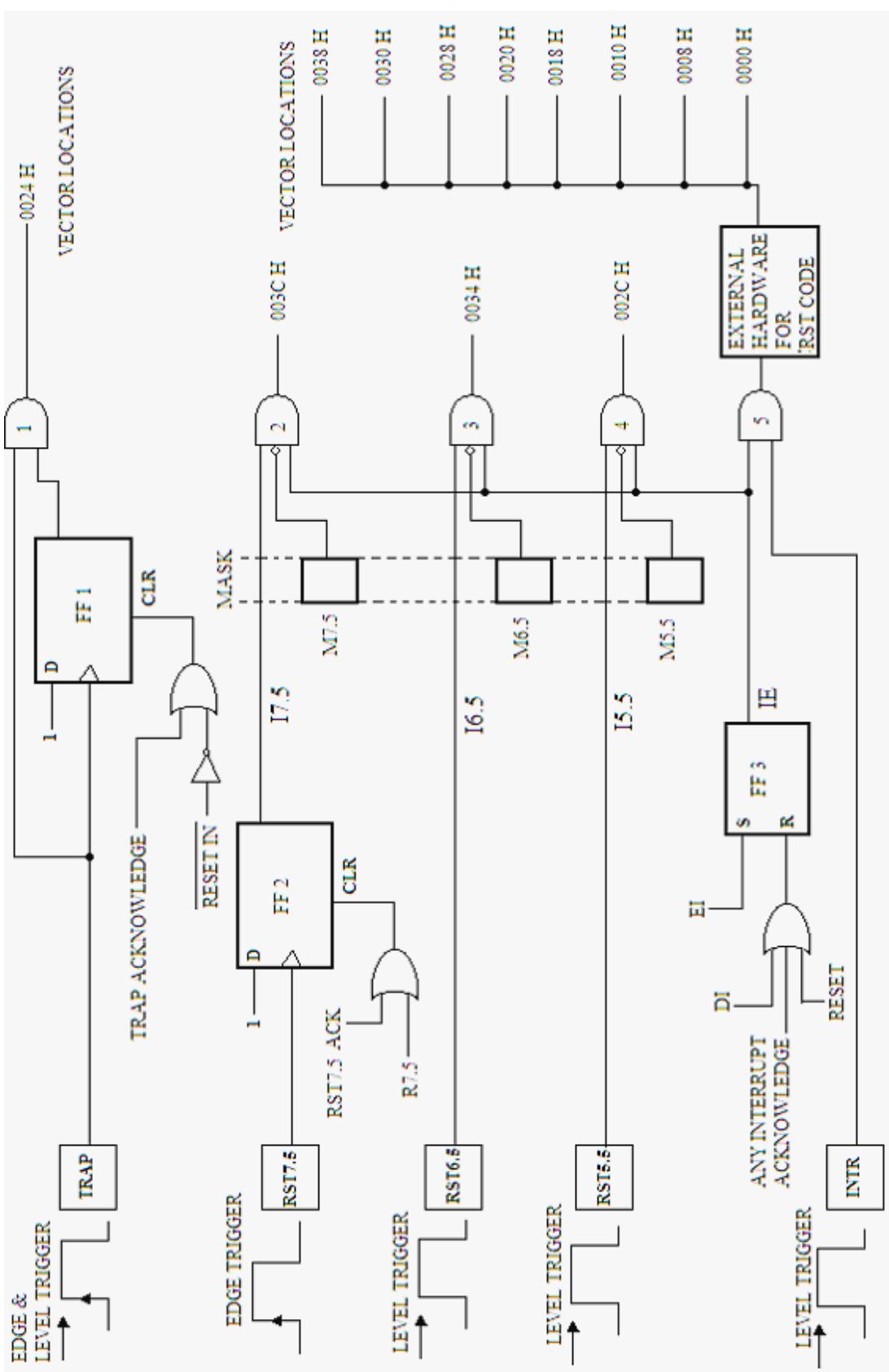
Figure 7.19 shows the interrupt control circuit with 8085 microprocessor. The TRAP is non-maskable interrupt i.e. it is neither be affected by any flag nor can be masked. It is used to handle very important functions. Since it is a highest priority interrupt, so it is used to take care of parity errors, power failure and other events that needs immediate attention. In the case of power failure, it may execute a routine to transfer the contents of the main memory to the back up memory (if any); and also for parity errors, the data may be corrected before carrying on. It is edge and level trigger which means the input has to go high and stays high. The rising edge and level triggered TRAP signal triggers the D flip-flop. The logic '1' at the output of D flip-flop and logic '1' of the TRAP input enable the AND gate to trigger the TRAP interrupt i.e. it calls the vector location 0024 H as shown in figure 7.19. Once the TRAP is recognized, further inputs at the TRAP will not be considered unless the interrupt is reset. The TRAP interrupt will be reset if the RESET signal is active (low) or internal TRAP

Acknowledge signal is high. After the 8085 microprocessor recognizes a TRAP interrupt, it will send a high TRAP acknowledge signal which will reset the D flip-flop.

RST 7.5 is a maskable interrupt which can be enabled or disabled by using SIM instruction. This is an edge triggered interrupt. When a leading edge signal appears at the RST 7.5 pin of 8085 microprocessor, D flip-flop 2 will be set (ref. figure 7.19). The output of this flip-flop is labeled as I 7.5 which is known as pending interrupt. This is one input of AND gate 2. The AND gate 2 will not be enabled until other two inputs of the gates are high. The RST flip-flop will be reset either by having a high R 7.5 bit or by having a high RST Acknowledge signal. The interrupt Acknowledge signal resets it for future RST 7.5 interrupt.

RST 6.5 and RST 5.5 are also maskable interrupts which may be enabled or disabled using SIM instruction. Both are level triggered interrupts. When a high signal (constant voltage of +5 V) appears at RST 6.5 pin of the microprocessor, it will enable one pin of AND gate 3. Till RST 6.5 interrupt pin is high and other two pins of AND gate 3 are low, this interrupt is known as pending interrupt (I 6.5). Similarly, when a high signal appears on RST 5.5 pin of the microprocessor, it will enable one terminal of AND gate 4. This interrupt will be pending interrupt (I 5.5) till RST 5.5 pin is high and other two inputs of AND gate 4 are low.

It may be noted from the figure 7.19 that one input each of the AND gates 2, 3 and 4 is connected to IE signal (called interrupt enable flag). The second pin of these gates will be enabled if IE signal is high. The IE signal may be made to high by enabling



the EI (Enable Interrupt) signal, which may be enabled by a software instruction **EI**. This instruction will be discussed in the succeeding section.

The mask bits M 7.5, M 6.5 and M 5.5 for the interrupts may be set using the SIM instruction. To mask an interrupt, the corresponding mask bit has to be set. So to enable a pending interrupt, the corresponding mask bit should be made low (by SIM instruction) and interrupt enable flag should be made high (by EI instruction).

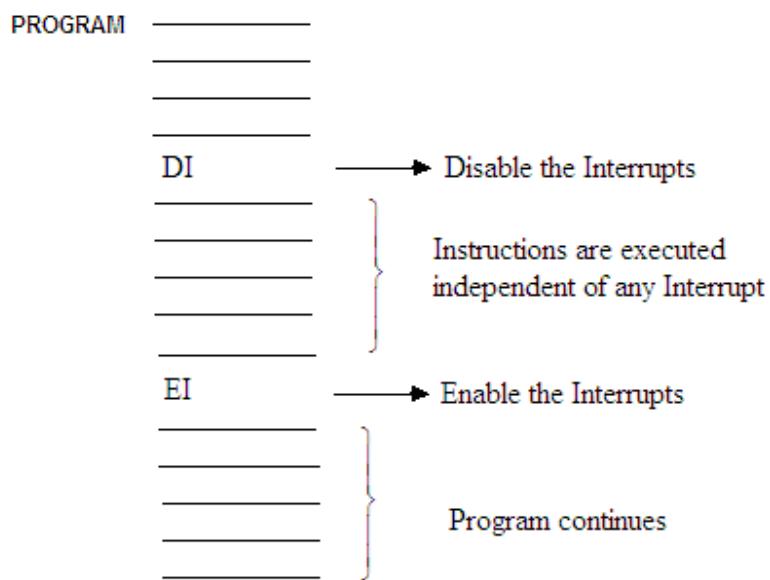
All the maskable interrupts may be disabled by either sending a low signal to *RESETIN* terminal or a high signal to ‘Any Interrupt Acknowledge’ terminal. The DI signal may also disable all the maskable interrupts. The DI (Disable Interrupts) is a software instruction.

All these hardware interrupts discussed above, once enabled will execute the corresponding hardware CALL instruction specified by their vector locations, as per their priority levels.

INTR is a lowest priority, maskable and level triggered interrupt. It uses handshaking. A high signal to INTR pin of the microprocessor will cause the current instruction to complete and will put the contents of the program counter into stack. The microprocessor will then generate a low *INTA* (interrupt acknowledge) signal. This signal is then used to enable a tristate buffer for the execution of hardware CALL instruction. It will execute the service subroutine program corresponding to any of the 8 software interrupts (RST 0 through RST 7).

## 7.7 INTERRUPT INSTRUCTIONS

Once the microprocessor recognizes any of the interrupts, it immediately disables all the interrupts except TRAP. This is done just to ensure that no further interrupts are recognized while the interrupt service subroutine (ISS) is being executed. Once the ISS is complete the program is required to enable the interrupts again. For this one byte instruction EI is introduced just before the RET instruction of ISS.



**Fig. 7.20**

Some portion of the main program may, sometimes be executed without being enabled the interrupt signals. For this the interrupts are to be disabled. The interrupts may be disabled by one byte instruction DI (Disable Interrupts). Later the interrupts may be enabled as shown in figure 7.20.

### EI and DI Instructions

The instruction

#### EI

stands for Enable Interrupt. When this instruction is executed, it produces a high EI signal (fig. 7.19), which sets the flip-flop 3 and produces a high signal to interrupt enable flag (IE). This way EI instruction enables all the interrupts except TRAP.

The instruction

#### DI

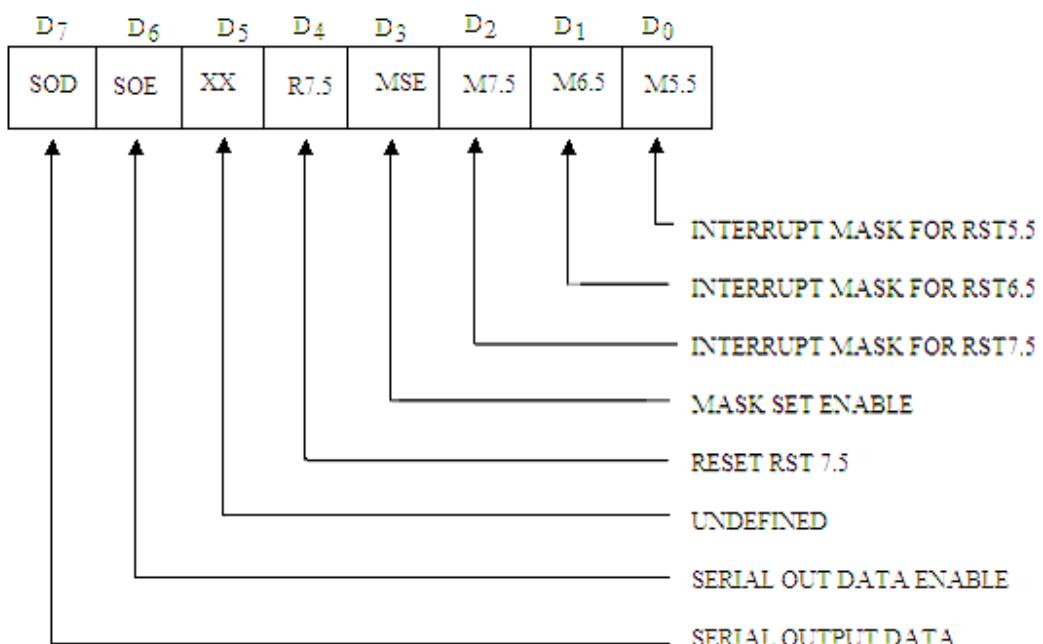
stands for Disable Interrupts. When this instruction is executed, it produces a high DI bit of flip-flop 3 (fig. 7.19). This resets the flip-flop and results a low IE (interrupt enable flag) signal. The low IE signal disables all the interrupts except TRAP.

### SIM and RIM Instructions

It is often required to selectively enable a few interrupts and disable others. The selectively enabling or disabling the interrupts can be done by an instruction

#### SIM

It stands for set interrupt masks. This instruction is used by loading the accumulator as shown in figure 7.21. The accumulator is loaded by *MVIA, data* (data bits are as per the requirements) instruction. The SIM instruction is then executed.



**Fig. 7.21**

The meaning of the different mask bits are given below:

The bits D<sub>0</sub> to D<sub>2</sub> are the mask bits (M 5.5 to M 7.5). A high to either of these bits represents that the particular interrupt is masked and a low, however, to either of these bits represents the enabling of that particular interrupt.

The bit D<sub>3</sub> is known as MSE (Mask Set Enable). When this bit is low, the mask bits D<sub>0</sub> to D<sub>2</sub> are ignored. A high to this bit indicates that the bits D<sub>0</sub> to D<sub>2</sub> are valid as described above.

The bit D<sub>4</sub> when high resets the flip-flop 2 (figure 7.19), in order to override RST 7.5 without servicing it.

D<sub>5</sub> is undefined bit.

The bits D<sub>6</sub> to D<sub>7</sub> are used for the serial transfer of data through the SOD line. The working of these pins will be described later.

Let us take an example to illustrate the function of SIM instruction. For example we have

MVI A, 0C H  
SIM

instructions in an assembly language program for 8085 microprocessor. When first instruction MVI A, 0C H is executed it will have the data (as shown in figure 7.22) for

| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 0              | 0              | 0              | 0              | 1              | 1              | 0              | 0              |

**Fig.7.22**

the execution of SIM. The SIM instruction after its execution will enable MSE signal (as D<sub>3</sub> bit is high). RST 7.5 interrupt is masked (bit D<sub>2</sub> is 1) and RST 5.5 and RST 6.5 interrupts are unmasked (enabled) as bits D<sub>0</sub> and D<sub>1</sub> are both 0. It will prevent the RST 7.5 interrupt from arriving at the final output.

The another interrupt instruction is

**RIM**

It stands for Read Interrupt Mask. This instruction will give the present status of the interrupt. This instruction loads the accumulator with 8 bit data whose details are given in figure 7.23.

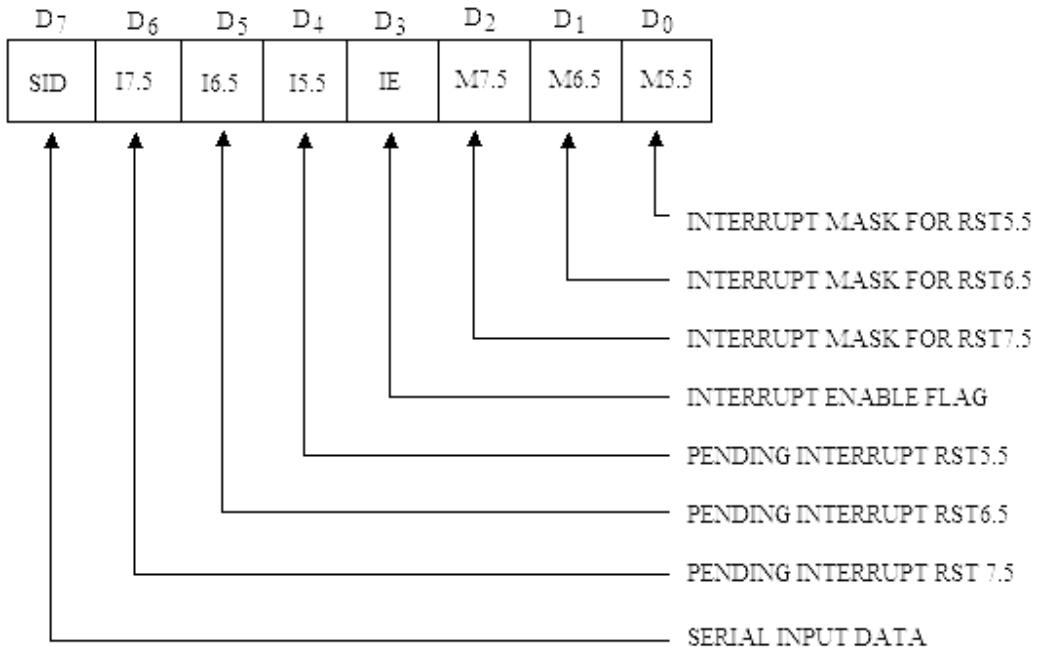
The meanings of the different bits for RIM are given below:

The bits D<sub>0</sub> to D<sub>2</sub> represent the masking of RST 5.5, RST 6.5 and RST 7.5 interrupts. A high to either of these bits represents that the particular interrupt is enabled; and a low to either of these bits represents that the particular interrupt is disabled.

The bit D<sub>3</sub> is known as interrupt enable flag (IE). When this bit is low, all the interrupts except TRAP are disabled and a high to this bit mask the bits D<sub>0</sub> to D<sub>2</sub> are ignored. A high to this bit indicates that the bits D<sub>0</sub> to D<sub>2</sub> are valid as described above.

The bits D<sub>4</sub> to D<sub>6</sub> represent the pending interrupts. A high to either of these bits represents that particular interrupt is pending; and a low to either of these bits represents that particular interrupt is not pending.

The bit D<sub>7</sub> is a serial input data and used for the serial input data through SID line. The working of this bit will be discussed later.



**Fig. 7.23**

Let us take an example to illustrate the function of RIM instruction. Suppose after the execution of RIM instruction, the accumulator contains 2A H as the data as shown in figure 7.24. The high to I 6.5, IE and M 6.5 bits indicate that RST 6.5 is a pending interrupt, the interrupt system is enabled and the RST 6.5 is currently masked.

| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 0              | 0              | 1              | 0              | 1              | 0              | 1              | 0              |

**Fig. 7.24**

**Example 7.1.** Write an assembly language program for 8085 microprocessor to enable RST 5.5 interrupt and disable RST 6.5 and RST 7.5 interrupts.

**Solution.** To enable RST 5.5, the bit D<sub>0</sub> (M 5.5) for the accumulator should be 0 (unmask); and for disabling RST 6.5 and RST 7.5, the bits D<sub>1</sub> and D<sub>2</sub> (M 6.5 and M 7.5) should be masked (1). Also MSE should be 1 (enable). The program will, therefore, be as given below:

|                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |
| SOD            | SOE            | XX             | R 7.5          | MSE            | M 7.5          | M 6.5          | M 5.5          |
| 0              | 0              | 0              | 0              | 1              | 1              | 1              | 0              |

=0E H

EI  
MVI A, 0E H  
SIM

**Example 7.2.** Write an assembly language program for 8085 microprocessor to enable all the interrupts.

**Solution.** The program for this case will be as given below. The bits D<sub>0</sub> to D<sub>2</sub> should be unmask and MSE should be enabled.

|                            |                            |                           |                              |                            |                              |                              |                              |
|----------------------------|----------------------------|---------------------------|------------------------------|----------------------------|------------------------------|------------------------------|------------------------------|
| D <sub>7</sub><br>SOD<br>0 | D <sub>6</sub><br>SOE<br>0 | D <sub>5</sub><br>XX<br>0 | D <sub>4</sub><br>R 7.5<br>0 | D <sub>3</sub><br>MSE<br>1 | D <sub>2</sub><br>M 7.5<br>0 | D <sub>1</sub><br>M 6.5<br>0 | D <sub>0</sub><br>M 5.5<br>0 |
|----------------------------|----------------------------|---------------------------|------------------------------|----------------------------|------------------------------|------------------------------|------------------------------|

=08 H

EI  
MVI A, 08 H  
SIM

**Example 7.3.** Write an assembly language program for 8085 microprocessor to enable RST 5.5 and RST 6.5 interrupt and reset 7.5 interrupt.

**Solution.** The program for this case will be as given below. The bits D<sub>0</sub> and D<sub>1</sub> should be unmasked and MSE should be enabled. For resetting of RST 7.5, the bit D<sub>4</sub> (R 7.5) should also be 1.

|                            |                            |                           |                              |                            |                              |                              |                              |
|----------------------------|----------------------------|---------------------------|------------------------------|----------------------------|------------------------------|------------------------------|------------------------------|
| D <sub>7</sub><br>SOD<br>0 | D <sub>6</sub><br>SOE<br>0 | D <sub>5</sub><br>XX<br>0 | D <sub>4</sub><br>R 7.5<br>1 | D <sub>3</sub><br>MSE<br>1 | D <sub>2</sub><br>M 7.5<br>1 | D <sub>1</sub><br>M 6.5<br>0 | D <sub>0</sub><br>M 5.5<br>0 |
|----------------------------|----------------------------|---------------------------|------------------------------|----------------------------|------------------------------|------------------------------|------------------------------|

=1C H

EI  
MVI A, 1C H  
SIM

**Example 7.4.** Write an assembly language program for 8085 microprocessor to check if RST 5.5 is pending. If it is pending, enable it without affecting any other interrupt else return to main program..

**Solution.** The program for this problem is given below. The RIM instruction will check if RST 5.5 is a pending interrupt. For this bit pattern of the accumulator will be checked. If bit D<sub>4</sub> is 1, the RST 5.5 is pending interrupt otherwise not.

#### For RIM

|                       |                         |                         |                         |                      |                         |                         |                         |
|-----------------------|-------------------------|-------------------------|-------------------------|----------------------|-------------------------|-------------------------|-------------------------|
| D <sub>7</sub><br>SID | D <sub>6</sub><br>I 7.5 | D <sub>5</sub><br>I 6.5 | D <sub>4</sub><br>I 5.5 | D <sub>3</sub><br>IE | D <sub>2</sub><br>M 7.5 | D <sub>1</sub><br>M 6.5 | D <sub>0</sub><br>M 5.5 |
|-----------------------|-------------------------|-------------------------|-------------------------|----------------------|-------------------------|-------------------------|-------------------------|

#### For SIM

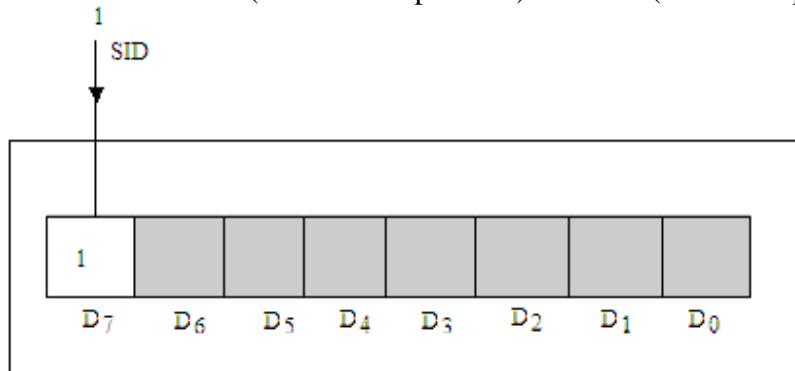
|                       |                       |                      |                         |                       |                         |                         |                         |
|-----------------------|-----------------------|----------------------|-------------------------|-----------------------|-------------------------|-------------------------|-------------------------|
| D <sub>7</sub><br>SOD | D <sub>6</sub><br>SOE | D <sub>5</sub><br>XX | D <sub>4</sub><br>R 7.5 | D <sub>3</sub><br>MSE | D <sub>2</sub><br>M 7.5 | D <sub>1</sub><br>M 6.5 | D <sub>0</sub><br>M 5.5 |
|-----------------------|-----------------------|----------------------|-------------------------|-----------------------|-------------------------|-------------------------|-------------------------|

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
|       | RIM       |         | ; Read Interrupt mask   |
|       | MOV B,    | A       | ; Mask information is moved to B-reg.                         |
|       | ANI       | 10 H    | ; Check if RST 5.5 is pending.                                |
|       | JNZ       | NXT     | ; Jump to NXT if it is pending interrupt.                     |
|       | EI        |         | ; Enable interrupts.  |
|       | RET       |         | ; RST 5.5 is not pending return to main program.              |
| NXT   | MOV A,    | B       | ; Get bit pattern (RST 5.5 is pending).                       |
|       | ANI       | 0E H    | ; Enable RST 5.5 by not masking D <sub>0</sub> bit (For SIM). |
|       | ORI       | 08 H    | ; Enable MSE for SIM.   |

SIM  
 JMP                    ISS                    ; Jump to service subroutine for RS  
 T5.5.

## 7.8 SERIAL INPUT AND OUTPUT DATA TRANSFER

As already discussed in the architecture of 8085, two pins (Pin Nos. 4 and 5) are provided for SOD (Serial Out Data) and SID (Serial In Data) lines. These lines are used for serial data transfer. The data transfer to or from the SID or SOD lines is possible using the Instructions RIM (Read Interrupt Mask) and SIM (Set Interrupt Mask).

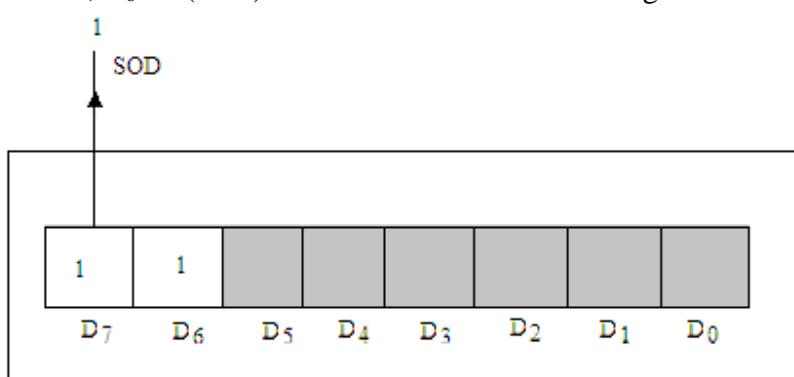


**Fig. 7.25**

The data on the SID line (Pin 5 of 8085) is loaded into accumulator at bit D<sub>7</sub> whenever a RIM instruction is executed. In other words a RIM instruction may be executed each time a new bit arrives at the SID input. For example, let a bit '1' arrives at the SID input. RIM instruction is now executed. After the execution of RIM instruction D<sub>7</sub> bit of the accumulator will be 1 as shown in figure 7.25.

Further to input 8 bit data serially through SID line, RIM instruction is executed 8 times and each time D<sub>7</sub> bit may be isolated and saved for the conversion of serial data into parallel data.

The SIM instruction sends the D<sub>7</sub> bit of the accumulator to the SOD line of 8085. For this transfer, D<sub>6</sub> bit (SOE) of the accumulator must be high as shown in figure 7.26.



**Fig. 7.26**

Suppose we wish to send a '0' bit to the SOD line, this can be done as:

MVI A, 40 H  
SIM

Similarly, to send a '1' bit to the SOD line, we use

MVI A, C0 H  
SIM

The rotate or other instructions may be used to convert 8 bit parallel data to serial data stream at the SOD output.

It may be noted from the above discussion that bit D<sub>7</sub> is used for SID line and this bit has nothing to do with the interrupt system. Similarly, bits D<sub>6</sub> and D<sub>7</sub> are used for SOD line and these bits have nothing to do with the interrupt system. So no new instructions are to be used for the serial transfer of data, i.e. interrupt instructions RIM and SIM may be used for this purpose also.

**Example 7.5.** Consider a switch is connected to SID line and an LED to SOD line of 8085. It is required to input the SID line via switch and output the switch status to the LED. In other words when the switch is ON or OFF the LED should glow or not glow. Write an assembly language program to accomplish this.

**Solution.** Program to accomplish the required work of the problem is given as:

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
| START | RIM       |         | ; Read Interrupt mask, Bit D <sub>7</sub> of A is SID. |
|       | ORI       | 40 H    | ; Enable SOE for SIM.                                  |
|       | SIM       |         | ; Output to LED.                                       |
|       | JMP       | START   | ; Jump to START.                                       |

**Example 7.6.** Write an assembly language program to generate a symmetrical square wave of known frequency at the SOD line of 8085 microprocessor.

**Solution.** To generate a square wave of certain frequency, SOD line should remain high for certain time and then low for the same amount of time. This is done by using the program given below. The time for the delay should be as per the frequency of the wave to be generated.

#### MAIN PROGRAM:

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
| START | MVI A,    | C0 H    | ; Enable SOE and SOD (D <sub>6</sub> and D <sub>7</sub> ) and disables all interrupts for SIM. |
|       | SIM       |         | ; Sends high signal to SOD line.   |
|       | CALL      | DELAY   | ; Delay is introduced for SOD to remains high for certain time.                                |
|       | MVI A,    | 40 H    | ; SOD is made low; and SOE and all interrupts are disabled for SIM.                            |
|       | SIM       |         | ; Sends low signal to SOD line.  |
|       | CALL      | DELAY   | ; Again delay is introduced for SOD to remain low for certain time.                            |
|       | JUMP      | START   | ; Repeats the process.   |

#### SUBROUTINE PROGRAM:

| Label | Mnemonics | Operand | Comments                                       |
|-------|-----------|---------|--|
| DELAY | LXI D,    | XXX H   | ; Loads DE register pair with a 16-bit number. |
| LOOP  | DCX D     |         | ; Decrements DE register pair by 1.            |

|          |   |  |
|----------|---|--|
| MOV A,   | E | ; Moves the contents of E register to accumulator.                               |
| ORA D    |   | ; ORing of the contents of D and E registers are performed to set the zero flag. |
| JNZ LOOP |   | ; If result is not zero than jump to loop.                                       |
| RET      |   | ; Go back to main program.   |

The 16-bit number XXX H loaded to DE register pair is as per the delay introduced in the program (discussed in chapter 4).

**Example 7.7.** Write an assembly language program to input an 8 bit data serially through SID line of 8085 microprocessor. Convert it to 8 bit parallel data and store this data to 2500 H memory location.

**Solution.** The program is given as:

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
| LOOP  | MVI B,    | 00 H    | ; Clears Register B.                                 |
|       | MVI C,    | 08 H    | ; Preset counter to 8.                               |
|       | RIM       |         | ; Get the bit through SID line.                      |
|       | ANI       | 80 H    | ; Isolate the bit received through SID line.         |
|       | ORA B     |         | ; Convert to parallel word.                          |
|       | RRC       |         | ; Rotate right.                                      |
|       | MOV B,    | A       | ; Save the accumulator contents to B register.       |
|       | DCR C     |         | ; Decrement the contents of C register.              |
|       | JNZ       | LOOP    | ; Go to LOOP if not zero.                            |
|       | STA       | 2500 H  | ; Store the parallel data to 2500 H memory location. |
|       | HLT       |         |  |

**Example 7.8.** An 8 bit data (say 32 H) is to be outputted serially through SOD line of 8085. An LED connected to SOD line, should glow or not glow each time a bit (1 or 0) is outputted through SOD line. A delay of 1 sec is to be introduced between each transfer of a bit. Write an assembly language program to implement this.

**Solution.** The program to output 8 bit data serially through SOD line is given below, which is self explanatory.

#### MAIN PROGRAM

| Label | Mnemonics | Operand | Comments                              |
|-------|-----------|---------|---------------------------------------|
|       | MVI A,    | 00 H    | ; Loads the data 32 H to accumulator. |
| LOOP  | MVI C,    | 08 H    | ; Preset counter to 8.                |
|       | RRC       |         | ; Rotate LSB to MSB.                  |

|        |       |  |
|--------|-------|--|
| MOV B, | A     | ; Save the accumulator contents to B register. |
| ANI    | 80 H  | ; Isolate SOD bit.                             |
| ORI    | 40 H  | ; Enable SOE bit for SIM.                      |
| SIM    |       | ; Sends the bit through SOD line.              |
| MOV A, | B     | ; Save the present contents to accumulator.    |
| CALL   | DELAY | ; Call the delay for 1 sec.                    |
| DCR C  |       | ; Decrement the contents of C register.        |
| JNZ    | LOOP  | ; Go to LOOP if not zero.                      |
| HLT    |       |  |

The delay subroutine program for 1 sec delay may be written as discussed in chapter 4.

#### SUBROUTINE PROGRAM:

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
| DELAY | LXI D,    | F424 H  | ; Loads DE register pair with a 16-bit number F424 H.                            |
| LOOP  | DCX D     |         | ; Decrements DE register pair by 1.  |
|       | MOV A,    | E       | ; Moves the contents of E register to accumulator.                               |
|       | ORA D     |         | ; ORing of the contents of D and E registers are performed to set the zero flag. |
|       | JNZ       | LOOP    | ; If result is not zero than jump to loop.                                       |
|       | RET       |         | ; Go back to main program.   |

**Example 7.9.** Write an assembly language program that is interrupted by applying a rising pulse at RST7.5 terminal manually. The program copies 256 bytes of data stored at memory locations starting at 3000 H to memory locations starting at 4000 H. The interrupt signal should, however, be able to introduce a delay of 1 sec. After the delay it should then move to main program. The main program for transferring the data is in a loop that repeats the same task again and again.

**Solution.** Here is the program that implements the given task.

#### MAIN PROGRAM

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
|       | EI        |         | ; Enable interrupts   |
|       | MVI A,    | 08 H    | ; Enable RST7.5, RST6.5 and RST5.5.                                   |
|       | SIM       |         |   |
| LOOP  | LXI H,    | 3000 H  | ; Load the H-L pair with the starting source address.                 |
|       | LXI D,    | 4000 H  | ; Load the D-E pair with the starting address of destination address. |

|     |        |      |  |
|-----|--------|------|--|
|     | MVI B, | FF H | ; load the B-reg. with the bytes of data.  |
| NXT | MOV A, | M    | ; Load the accumulator with the contents stored in memory location addressed by H-L register pair. |
|     | STAX D |      | ; Store the accumulator contents to the destination address given by D-E register pair.            |
|     | INX H  |      | ; Increments the H-L pair for next number.   |
|     | INX D  |      | ; Increments the D-E pair  |
|     | DCR B  |      | ; Decrement B.   |
| JNZ |        | NXT  | ; If not zero jump to NXT.   |
| JMP |        | LOOP | ; Jump to start.   |
|     | HLT    |      |  |

When the interrupt signal is enabled, it calls its vector location 003C H.

At the vector location say it is stored that JMP FFBD H i.e. monitor transfers the program to the memory location FFBD H. Now the user has to transfer from FFBD H to a memory location from where service subroutine for RST7.5 is written. The user transfers the program from FFBD H to 2000 H with the instructions JMP 2000 H. The location 2000 H indicates the starting address of interrupt service subroutine. The interrupt service subroutine is given as:

#### Interrupt Service Subroutine (at the starting address 2000 H )

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
|       | PUSH PSW  |         | ; Save accumulator and flag contents in stack.                                   |
|       | PUSH B    |         | ; Save the contents of B-C register pair in stack.                               |
|       | PUSH D    |         | ; Save the contents of D-E register pair in stack.                               |
| DELAY | LXI D,    | F424 H  | ; Loads DE register pair with a 16-bit number F424 H.                            |
| LOOP  | DCX D     |         | ; Decrements DE register pair by 1.  |
|       | MOV A,    | E       | ; Moves the contents of E register to accumulator.                               |
|       | ORA D     |         | ; ORing of the contents of D and E registers are performed to set the zero flag. |
|       | JNZ       | LOOP    | ; If result is not zero than jump to loop.                                       |
|       | POP D     |         | ; Restore the contents of D-E register pair from the stack.                      |
|       | POP B     |         | ; Restore the contents of B-C register pair from the stack.                      |

|         |   |
|---------|---|
| POP PSW | ; Restore the contents of Accumulator and flag contents form the stack. |
| EI      | ; Enable interrupts   |
| RET     | ; Go back to main program.  |

Program written in ISS is the program for delay of one second as already discussed.

### **PROBLEMS**

1. Discuss the memory Mapped I/O operation for the transfer of data from microprocessor to I/O devices and vice-versa.
2. Discuss the I/O Mapped I/O operation for the data transfer from microprocessor to I/O devices and vice-versa.
3. Give the comparison of memory mapped I/O and isolated I/O scheme fro the data transfer from microprocessor to I/O devices and vice-versa.
4. Why the problems arise when the data is transferred from the microprocessor to I/O devices and vice-versa? Mention various data transfer scheme.
5. Discuss programmed I/O data transfer scheme.
6. Describe Interrupt driven I/O data transfer scheme.
7. What do you mean by handshaking? How is it used in asynchronous data transfer between microprocessor and I/O devices?
8. What is DMA? Using block diagram explain how the data is transferred by a DMA controller.
9. Discuss DMA for 8085.
10. What is an interrupt? How data is transferred between the microprocessor and I/O devices.
11. What is the difference between software and hardware interrupts? Discuss software interrupts of 8085.
12. How is the INTR interrupt used in 8085?
13. Differentiate between
  - (i) Memory mapped I/O and I/O mapped I/O data transfer schemes.
  - (ii) Maskable and Non-maskable Interrupts
  - (iii) Hardware and Software interrupts
14. Explain briefly the following:
  - (i) DMA data transfer scheme
  - (ii) Interrupt driven data transfer scheme
15. What are interrupt terminals available in 8085 microprocessor? How SIM and RIM instructions are used to set and read the interrupts.
16. What are hardware interrupts? What is meant by Vectored interrupts?
17. What are the functions of SID and SOD pins of 8085? How RIM and SIM instructions are used to input a bit through SID line and output a bit through SOD line.
18. Explain RIM and SIM instructions.
19. Discuss EI and DI instructions.
20. Write down the procedure to mask the RST6.5 interrupt.
21. Explain RST  $n$  interrupt circuit of 8085.
22. Discuss the bit pattern for RIM instruction.

23. Discuss the bit pattern for SIM instruction.
24. Draw and explain the interrupt control circuit for 8085 microprocessor.
25. Mention interrupt instructions of 8085. Discuss any two of them.
26. How will you enable all the interrupts of 805?
27. How will you enable RST5.5 and disable RST6.5 and RST7.5?
28. Write an assembly program of 8085 to generate asymmetrical square wave at the SOD line of 8085.
29. Write an assembly language program to input 256 bytes of data serially through SID line of 8085 microprocessor. Convert the 256 bytes of data received serially to parallel data and store the data to memory locations starting at 2500 H.
30. Here are some instructions

MVI A, 1D H  
SIM

After SIM instruction is executed, which interrupts are masked.

**(Ans.: RST 5.5 and RST 7.5 are masked.)**

31. Here are some instructions

MVI A, XX H  
SIM

What should be the value of XX H in MVI instruction so that RST 5.5 and RST 6.5 are masked and all other don't care bits should be set to zero.

**(Ans.: XX H = 0B H)**

---

# 8

## Programmable Peripheral Interface (PPI) 8255A

---

The various methods of data transfer from the microprocessor to output devices or vice-versa has already been discussed in the preceding chapter. Special interface circuits, known as peripheral interface circuits are to be used for this purpose. The interfacing devices may be classified into two categories namely general purpose peripherals and special purpose peripherals. Basically the I/O devices to be connected to microprocessors are known as peripherals, these are printers, floppy drives, CRT and Cassette recorder etc. The general purpose peripherals are:

- Programmable Peripheral Interface (PPI)
- Programmable Interval Timer
- Programmable Interrupt Controller
- Programmable DMA Controller
- Programmable Communications Interface.

The special purpose peripherals used for interfacing a microprocessor to a specific type of I/O device are:

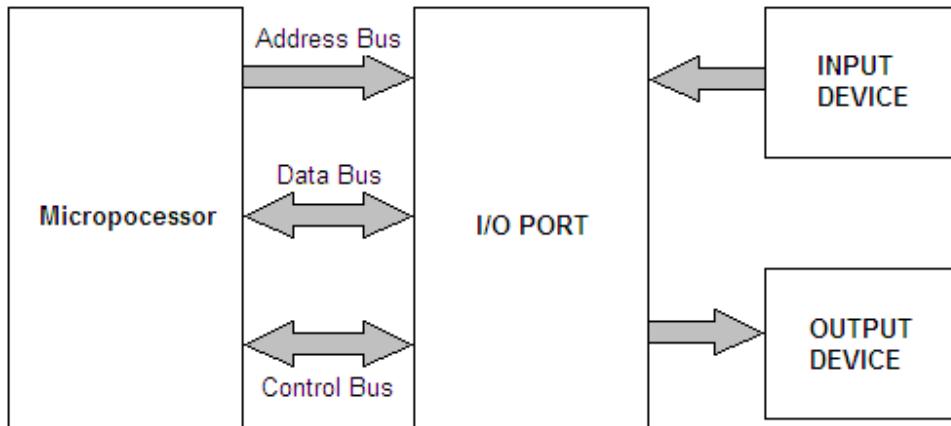
- Programmable Keyboard and Display Interface
- Programmable Hard Disk Controller
- Programmable Floppy Disk Controller

The present chapter will confine to the discussion of Programmable Peripheral Interface (PPI) IC 8255 A and its application.

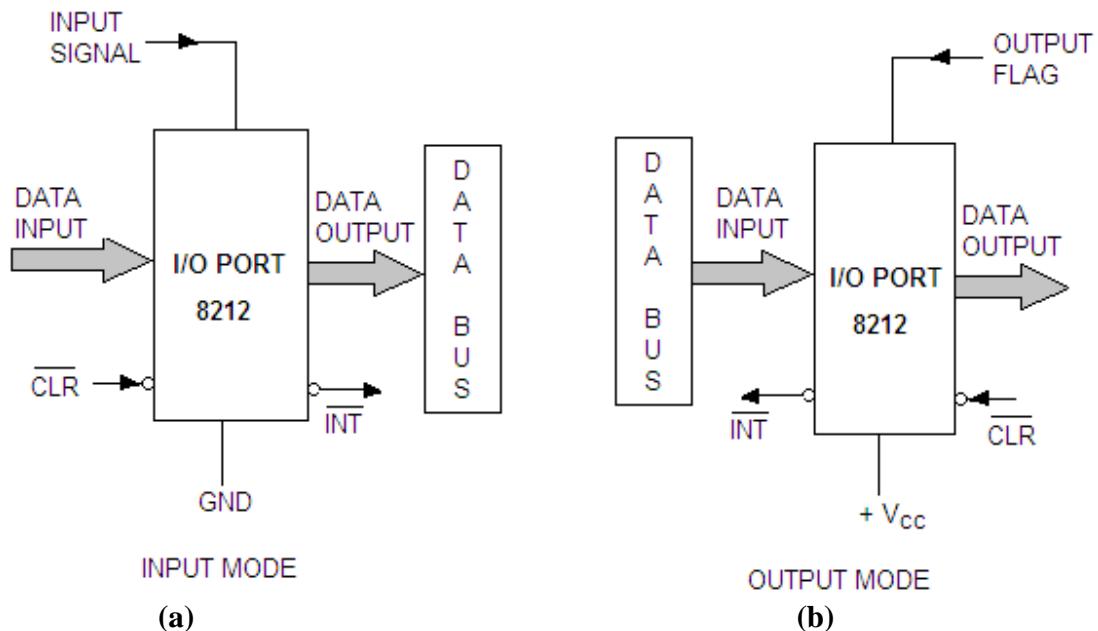
### 8.1 DETAILS OF PPI IC 8255A

The input/output devices are generally interfaced to the microprocessor through the input/output port as shown in figure 8.1. The input/output port is either non-programmable or programmable. A non-programmable port can either be connected in input mode or output mode, i.e. if both input and output devices are to be connected to the microprocessor two separate non-programmable ports are to be used, one for input device and other for the output device. INTEL 8212 is an 8-bit non-programmable I/O port. Figures 8.2 (a) and (b) show the interfacing of 8212 in input and output mode respectively. However, a programmable I/O port can be programmed to act either as an input port or an output port. The INTEL 8255A is a programmable port device. It is most

versatile Programmable Peripheral Interface which may be connected to almost any

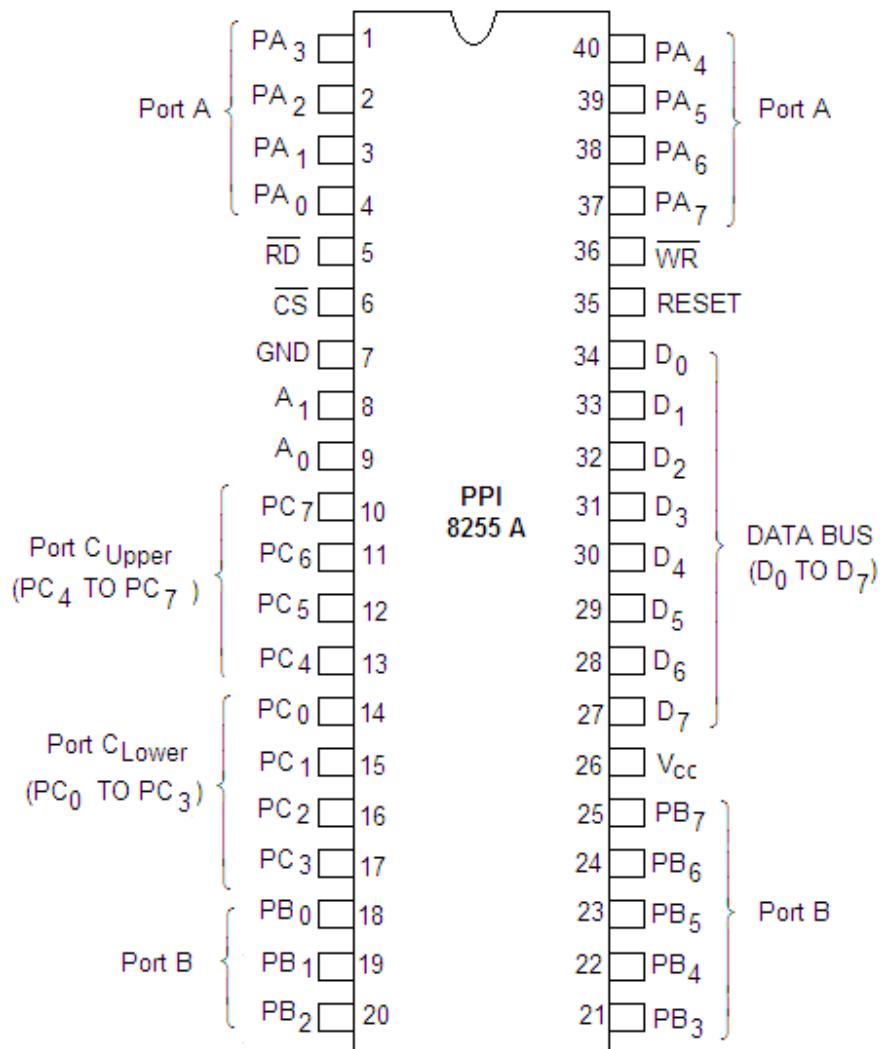


**Fig. 8.1**



**Fig. 8.2**

microprocessor. This IC is widely used and can be programmed to transfer the data to the input/output devices. It is a 40 pin dual in line IC package, whose pin configuration and block diagram are shown in figures 8.3 and 8.4 respectively.



**Fig. 8.3**

The IC 8255 A has three 8-bit ports:

Port-A

Port-B

and

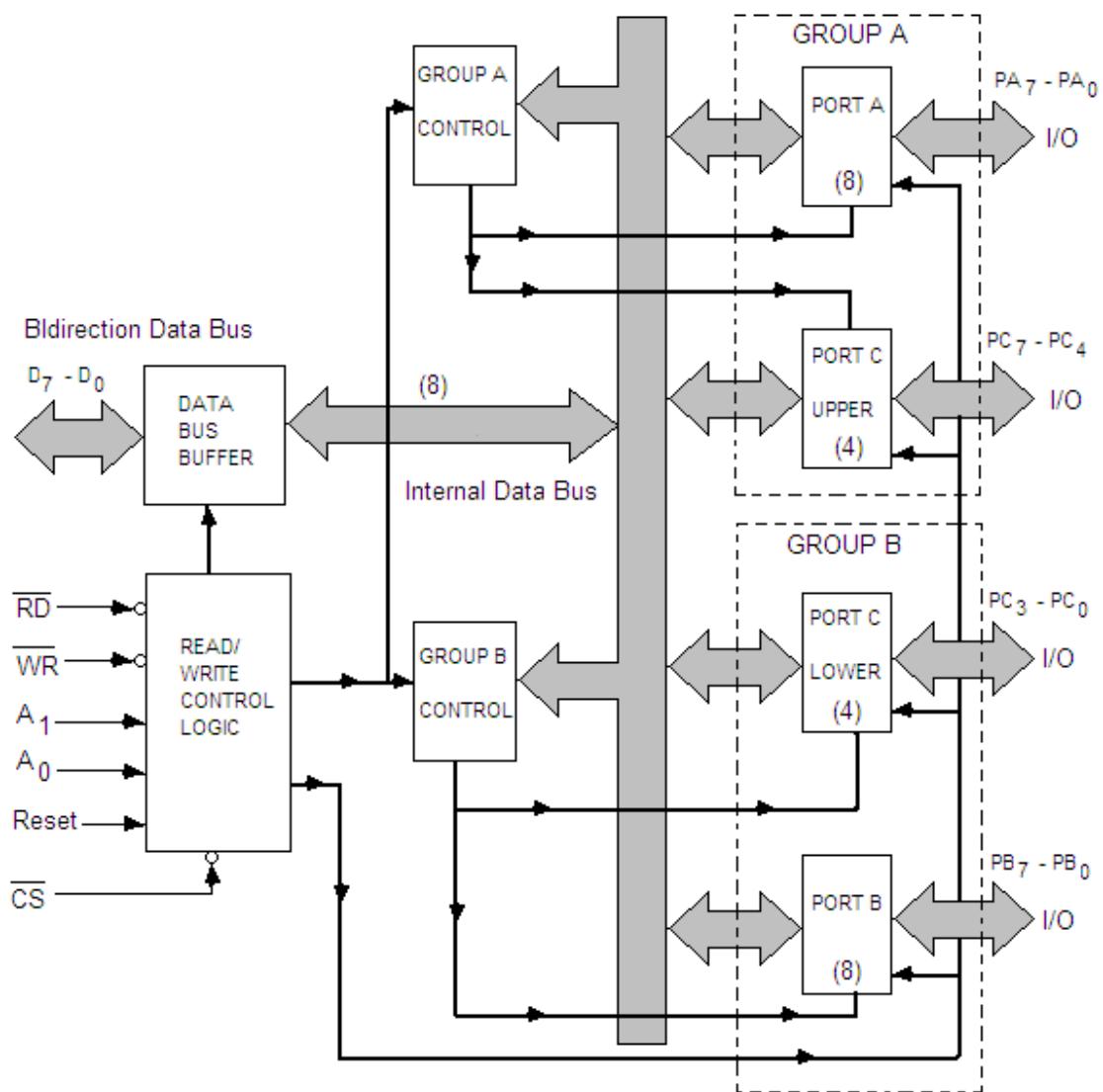
Port-C

The port-C can be used into two 4-bit ports represented as Port C<sub>Upper</sub> and Port C<sub>Lower</sub>.

Port-A (PA<sub>7</sub>-PA<sub>0</sub>) and Port-C<sub>Upper</sub> (PC<sub>7</sub>-PC<sub>4</sub>) together form Group A, whereas Port-B (PB<sub>7</sub>-PB<sub>0</sub>) and Port-C<sub>Lower</sub> (PC<sub>3</sub>-PC<sub>0</sub>) form Group B as shown in figure 8.5.

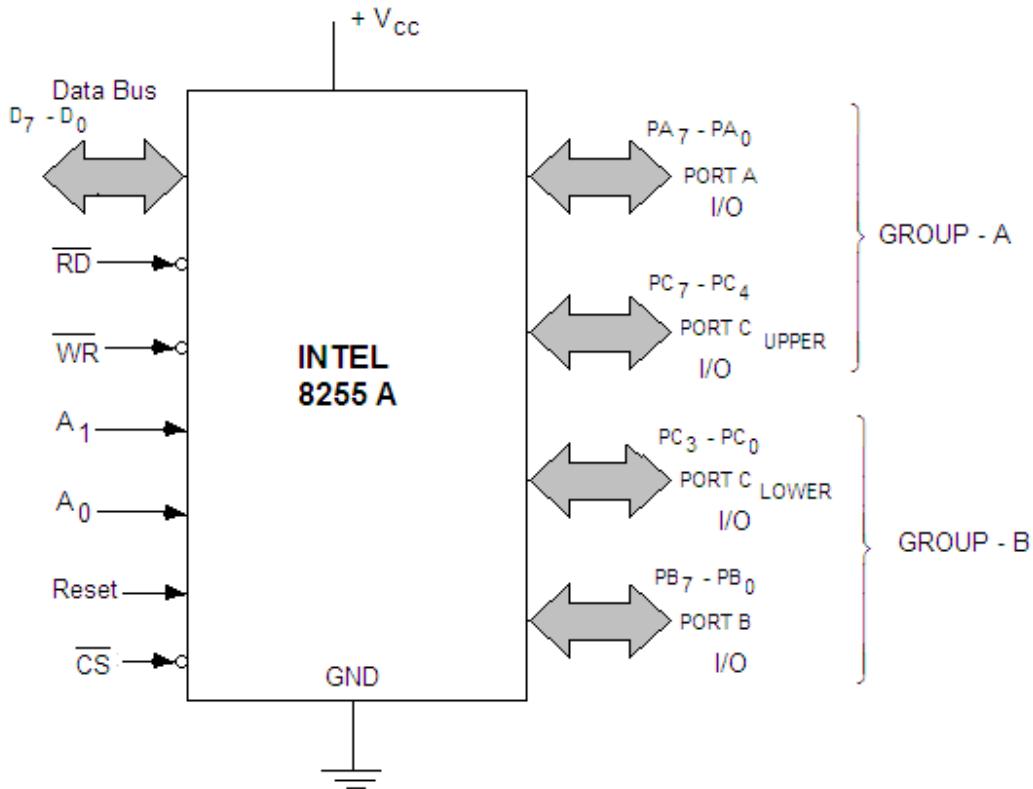
After deciding the configuration of 3 ports (which port is to be used as input port and which port is to be used as output port), a control word of the command has to be given to the microprocessor. An 8-bit control word is formed for this purpose which may be transferred to the control word register of 8255 through the accumulator.

The control register has 6 control lines RD, WR, RESET, A<sub>1</sub>, A<sub>0</sub> and CS whose functions are as given below:



**Fig. 8.4**

- (1) **RD (Read):** It is a read signal which is active low. When this signal goes low, it allows the microprocessor to read the data from the selected I/O ports of the 8255 PPI.
- (2) **WR (Write):** It is the write signal which is also active low. When this signal goes low, the microprocessor writes (loads) the data into the selected I/O ports or the control register of 8255.
- (3) **RESET (Reset):** This is a reset line which is active high. When a high signal appears on this line, it clears the control register and sets all the ports in the input mode.



**Fig. 8.5**

(4)  $A_0$  and  $A_1$ :

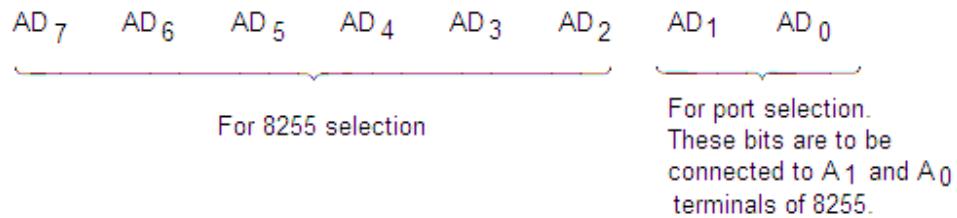
These lines  $A_0$  and  $A_1$  are used to address the three ports and the control word as shown in table 8.1. If the  $\overline{CS}$  (chip select) terminal of 8255 is low, the lines  $A_0$  and  $A_1$  decide the ports. The CPU can read or write into these registers by using  $\overline{RD}$  and  $WR$  signals.

**Table 8.1**

| $A_1$ | $A_0$ |                       |
|-------|-------|-----------------------|
| 0     | 0     | Port-A                |
| 0     | 1     | Port-B                |
| 1     | 0     | Port-C                |
| 1     | 1     | Control word register |

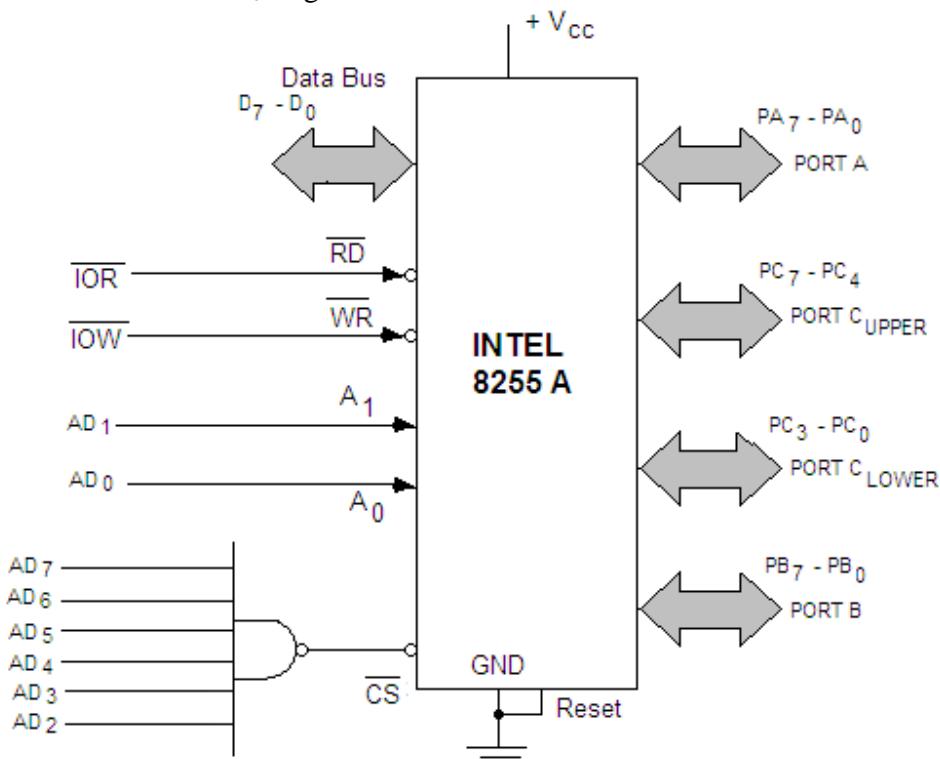
(5)  $\overline{CS}$  (Chip Select): It is an active low chip select terminal. It is used to select 8255 by applying low signal to  $\overline{CS}$  terminal. When a high signal appears on this line, the 8255 will not be selected, the data bus buffer that connects 8255 to the system data bus remains floated.

The 6 bits  $AD_2-AD_7$  of address data bus (low order address bus) of the microprocessor are decoded to provide  $\overline{CS}$ ; the remaining two bits  $AD_0-AD_1$  may be used for the selection of control register or any of the three ports as shown in figure 8.6. Since the six bits of address data bus are decoded for  $\overline{CS}$ , so as many as  $2^6 (= 64)$  PPI (8255) can be connected to any system. For the selection of any port of 8255  $AD_1$  and  $AD_0$  bits of the address data bus are connected to  $A_1$  and  $A_0$  terminals of 8255.



**Fig. 8.6**

Figure 8.7 shows the chip-select logic for 8255. From this logic diagram it is clear that if  $AD_7$ - $AD_2$  are all 0s then it enables  $\overline{CS}$  to select this chip. The port selection will depend on the bits  $AD_1$ - $AD_0$  as given in table 8.2.



**Fig. 8.7**

**Table 8.2**

| $AD_7$ | $AD_6$ | $AD_5$ | $AD_4$ | $AD_3$ | $AD_2$ | $AD_1$ | $AD_0$ | HEX ADDRESS | PORT NAME             |
|--------|--------|--------|--------|--------|--------|--------|--------|-------------|-----------------------|
| 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 00 H        | Port-A                |
| 0      | 0      | 0      | 0      | 0      | 0      | 0      | 1      | 01 H        | Port-B                |
| 0      | 0      | 0      | 0      | 0      | 0      | 1      | 0      | 02 H        | Port-C                |
| 0      | 0      | 0      | 0      | 0      | 0      | 1      | 1      | 03 H        | Control word Register |

Generally the microprocessor kits available with the laboratories have two 8255 connected with the system; 8255-I and 8255-II. Figure 8.8 shows the chip select logic for

the second 8255. If AD<sub>7</sub>-AD<sub>2</sub> are chosen as per this diagram then the second 8255 (8255-II) will be selected. The port selection will depend on the bits AD<sub>1</sub>-AD<sub>0</sub> as given in table 8.3. The M/S Vinytcs, New Delhi has adopted this logic in their microprocessor kits.

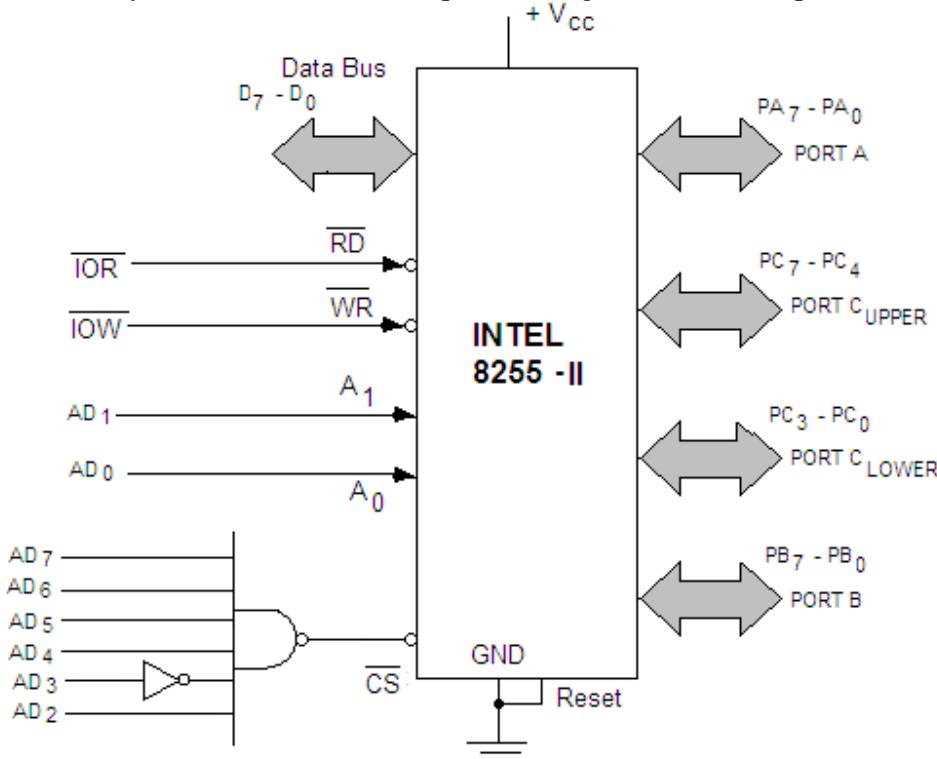


Fig. 8.8

Table 8.3

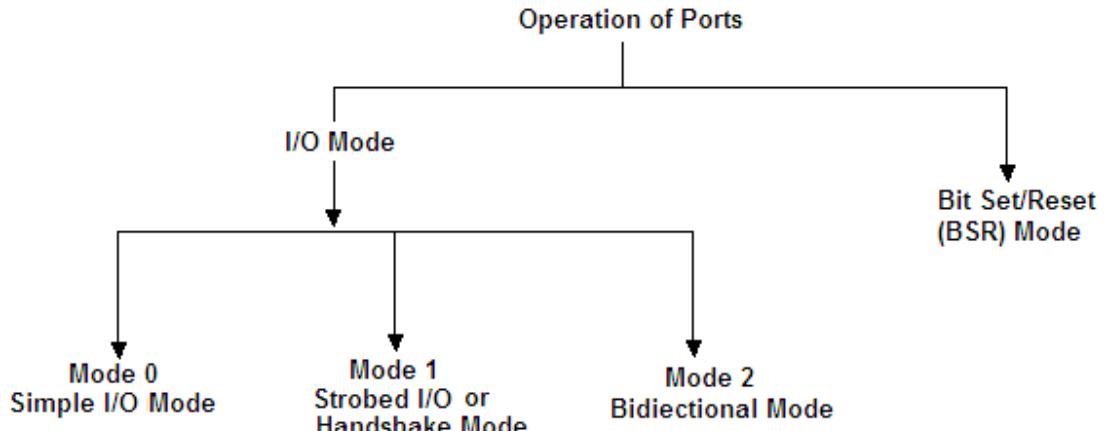
| AD <sub>7</sub> | AD <sub>6</sub> | AD <sub>5</sub> | AD <sub>4</sub> | AD <sub>3</sub> | AD <sub>2</sub> | $\overline{CS}$ |                 | HEX ADDRESS | PORT NAME of 8255-II  |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-------------|-----------------------|
|                 |                 |                 |                 |                 |                 | AD <sub>1</sub> | AD <sub>0</sub> |             |                       |
| 0               | 0               | 0               | 0               | 1               | 0               | 0               | 0               | 08 H        | Port-A                |
| 0               | 0               | 0               | 0               | 1               | 0               | 0               | 1               | 09H         | Port-B                |
| 0               | 0               | 0               | 0               | 1               | 0               | 1               | 0               | 0AH         | Port-C                |
| 0               | 0               | 0               | 0               | 1               | 0               | 1               | 1               | 0BH         | Control word Register |

It may be mentioned here that if the instruction OUT 03 H is executed then it transfers the contents of accumulator to the control word register of 8255-I. Similarly, if the instruction OUT OB H is executed then the accumulator contents will be transferred to control word register of 8255-II. For the control word, accumulator is loaded with the contents that are necessary to use the ports of 8255 as input port or output port. The procedure for the generation of control word will be discussed in a later section of this chapter. Now if 8255-I is initialized to use Port A as input port and ports B and C as output ports, then the execution of IN 00 H instruction will transfer the data from port A

of 8255-I to the accumulator. The execution of OUT 01 H instruction will transfer the accumulator contents to the port B of 8255-I. Similarly, 8255-II may be initialized by giving the proper control word to the control word register by using the instruction OUT 0B H. The IN or OUT instruction having the port address as 08, 09 or 0A may be used for the data transfer through 8255-II.

## 8.2 OPERATIONAL MODES OF 8255A

The operational modes of 8255 A PPI can be classified into two broad groups (fig. 8.9).



**Fig. 8.9**

### 1. I/O Mode

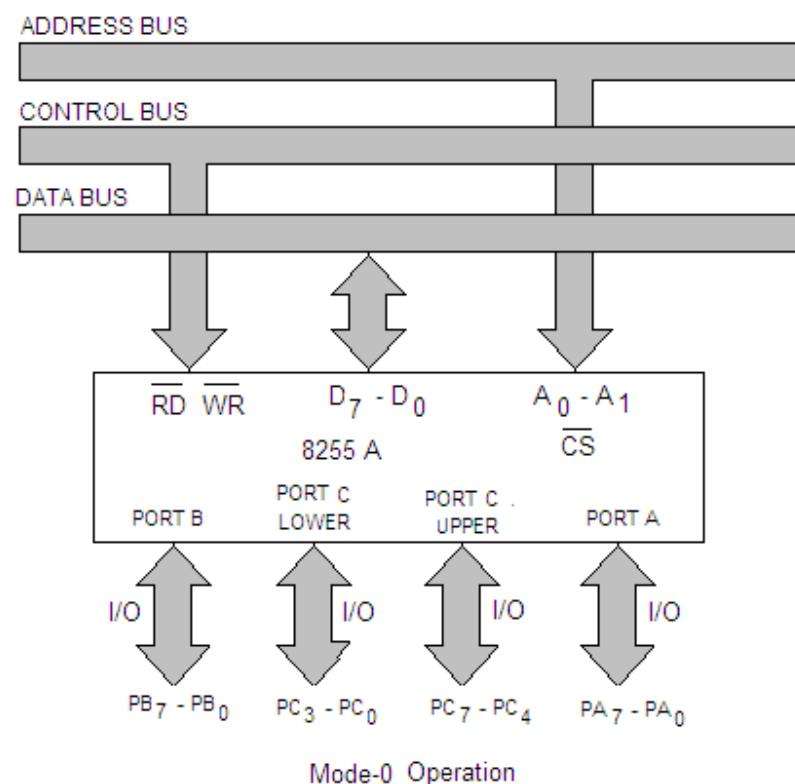
The input/output mode can further be subdivided into three groups:

- **Mode 0 – Simple Input/output mode**

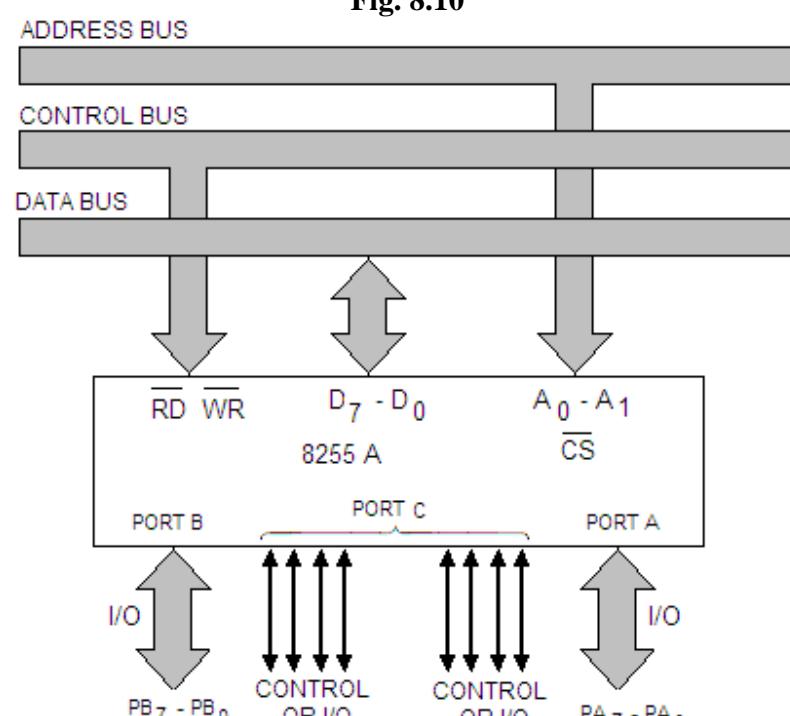
In this mode the 8 bit port A ( $PA_0-PA_7$ ) can be configured as **input or output port**. The port B ( $PB_0-PB_7$ ) can also be **configured, in the similar fashion, as input or output operation**. However there is flexibility for the port C. It can be divided into two 4 bit ports, the port  $C_{Lower}$  ( $PC_0-PC_3$ ) and port  $C_{Upper}$  ( $PC_4-PC_7$ ). Each of them can be set independently for input or output operation. In this way there are four ports (port-A, port-B,  $C_{Lower}$  and  $C_{Upper}$ ) and each of them can be set either as an **input port or an output port**. Here these ports are simple input or output ports i.e. these ports can work without handshaking. In this mode the **outputs are latched** whereas the **inputs are not latched**. The basic definition of this mode is shown in figure 8.10.

- **Mode 1 – Strobed Input/output or Handshake mode**

In this mode of **operation handshaking** is used for the input or output data transfer. As discussed earlier, there are two groups in 8255 PPI: Group A and Group B. Both these groups have **one 8-bit port and one 4-bit port**. Port-A and Port  $C_{Upper}$  form group A whereas Port-B and Port  $C_{Lower}$  form group B. The 8-bit port of each group can be programmed for input or output operation with **latched input and latched output facilities**. The bits of Port C are used for **handshaking**. Figure 8.11 shows the basic definition of this mode.



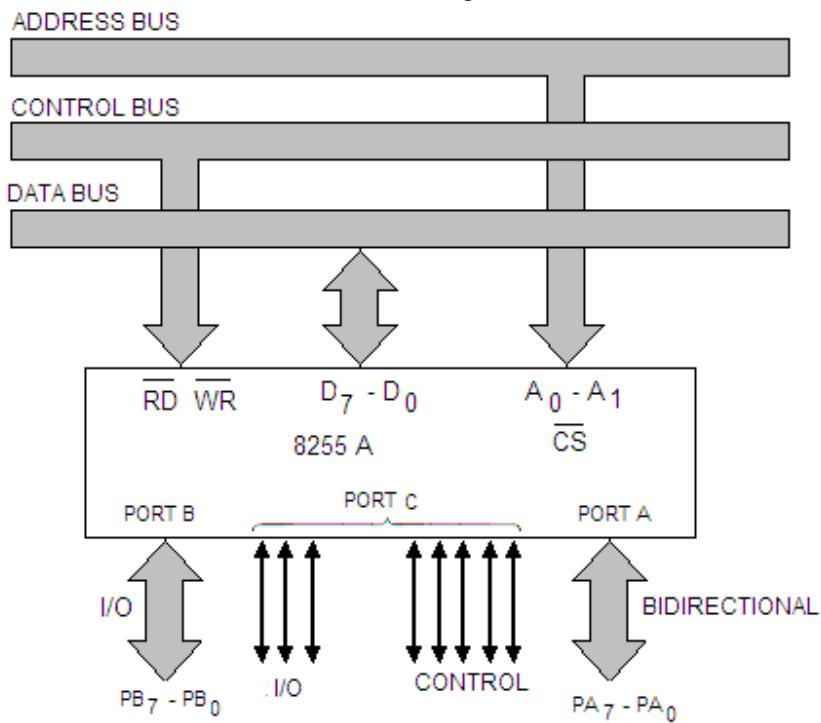
**Fig. 8.10**



**Fig. 8.11**

- **Mode 2 – Bidirectional Mode**

In this mode Port A can be programmed to operate as a **bidirectional port**. When Port A is programmed in this mode of operation, Port B can be used either in Mode 0 or Mode 1. For mode 2 operation PC<sub>3</sub> to PC<sub>7</sub> bits are used for handshaking. In this mode too both **inputs and outputs are latched**. The basic definition of this mode is shown in figure 8.12.



Mode-2 Operation

**Fig. 8.12**

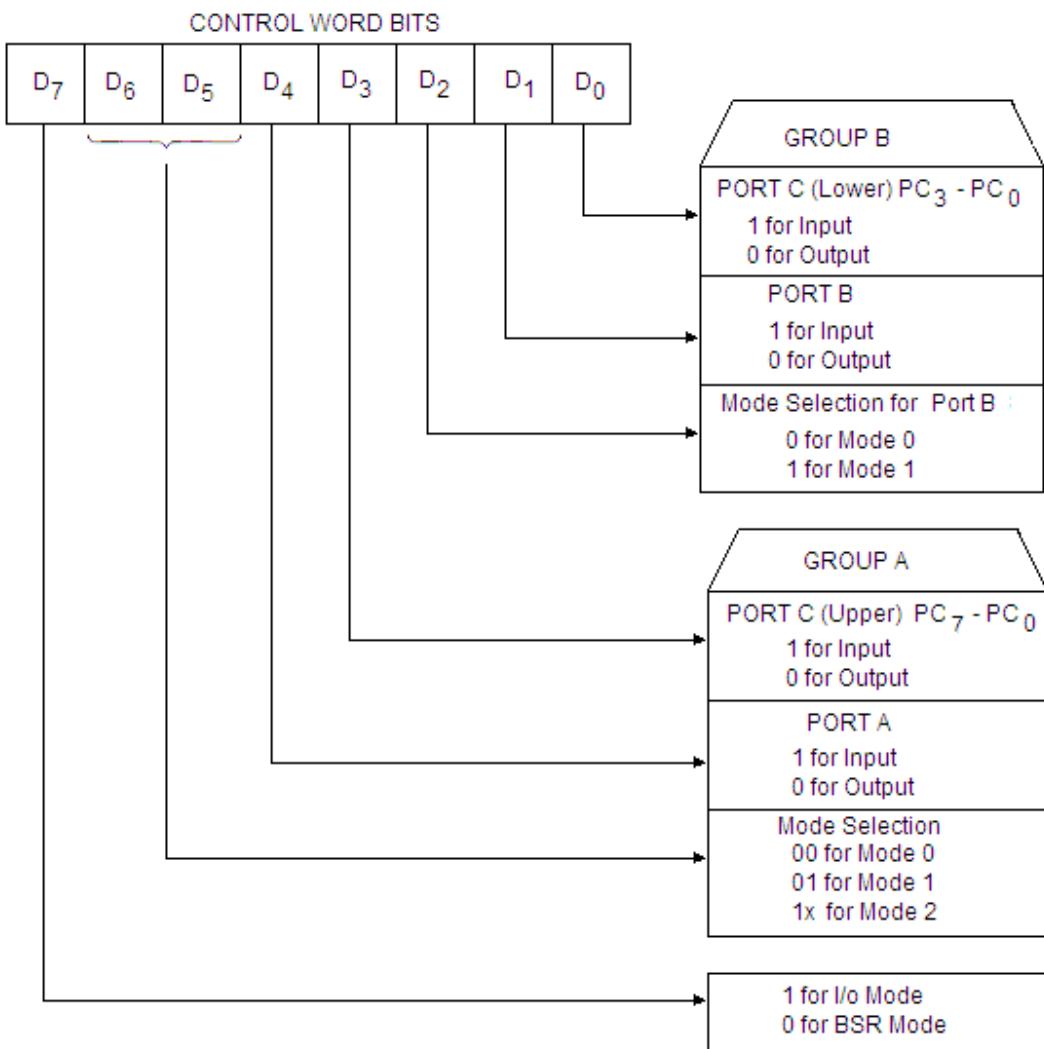
## 2. Bit Set/Reset Mode

The modes of operation of 8255 PPI may be selected by the software. The details of these modes will be discussed in the following sections.

## 8.3 CONTROL WORD FORMAT FOR 8255A

As discussed above, a port can be programmed to act as an input port or an output port. A control word is, therefore, to be formed for programming the ports of 8255A. The format for the control word is shown in figure 8.13. The control word, as per the requirement of the programmer, is written into the control word register of 8255A. No read operation of the control word is allowed. The description of the control bits is given below:

- Bit D<sub>0</sub>**      Sets Port C<sub>Lower</sub> as input or output port.  
To make Port C<sub>Lower</sub> as input port this bit is set to 1.  
To make Port C<sub>Lower</sub> as output port this bit is set to 0.
- Bit D<sub>1</sub>**      Sets Port B as input or output port.  
To make Port B as input port this bit is set to 1.  
To make Port B as output port this bit is set to 0.



**Fig. 8.13**

**Bit D<sub>2</sub>** This bit is for mode selection for the port B.

If this bit is set to 0, the port B will operate in mode 0.

If this bit is set to 1, the port B will operate in mode 1.

**Bit D<sub>3</sub>** Sets Port C<sub>Upper</sub> as input or output port.

To make Port C<sub>Upper</sub> as input port this bit is set to 1.

To make Port C<sub>Upper</sub> as output port this bit is set to 0.

**Bit D<sub>4</sub>** Sets Port A as input or output port.

To make Port A as input port this bit is set to 1.

To make Port A as output port this bit is set to 0.

**Bits D<sub>5</sub> and D<sub>6</sub>** These two bits are used to determine the I/O mode of port A.

These bits are defined for the various modes of port A as follows:

| D <sub>6</sub> | D <sub>5</sub> | Mode of port A |
|----------------|----------------|----------------|
| 0              | 0              | Mode 0         |
| 0              | 1              | Mode 1         |
| 1              | 0 or 1         | Mode 2         |

**Bit D<sub>7</sub>** This bit specifies either I/O function or bit set/reset function (BSR mode).

If this bit is set to 1 then the 8255 will work in I/O mode.

If this bit is set to 0 then the 8255 will work in BSR mode.

There are 16 combinations of control words for various configurations of the ports of 8255 for Mode 0 operations. These control words are shown in table 8.4.

**Table 8.4**

| Control Word Bits |                |                |                |                |                |                |                | Control Word | PORT A | PORT C <sub>Upper</sub> | PORT B | PORT C <sub>Lower</sub> |        |
|-------------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------------|--------|-------------------------|--------|-------------------------|--------|
| D <sub>7</sub>    | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |              |        |                         |        |                         |        |
| 1                 | 0              | 0              | 0              | 0              | 0              | 0              | 0              | 80 H         | Output | Output                  | Output | Output                  | Output |
| 1                 | 0              | 0              | 0              | 0              | 0              | 0              | 1              | 81 H         | Output | Output                  | Output | Input                   |        |
| 1                 | 0              | 0              | 0              | 0              | 0              | 1              | 0              | 82 H         | Output | Output                  | Input  | Output                  |        |
| 1                 | 0              | 0              | 0              | 0              | 0              | 1              | 1              | 83 H         | Output | Output                  | Input  | Input                   |        |
| 1                 | 0              | 0              | 0              | 1              | 0              | 0              | 0              | 88 H         | Output | Input                   | Output | Output                  |        |
| 1                 | 0              | 0              | 0              | 1              | 0              | 0              | 1              | 89 H         | Output | Input                   | Output | Input                   |        |
| 1                 | 0              | 0              | 0              | 1              | 0              | 1              | 0              | 8A H         | Output | Input                   | Input  | Output                  |        |
| 1                 | 0              | 0              | 0              | 1              | 0              | 1              | 1              | 8B H         | Output | Input                   | Input  | Input                   |        |
| 1                 | 0              | 0              | 1              | 0              | 0              | 0              | 0              | 90 H         | Input  | Output                  | Output | Output                  |        |
| 1                 | 0              | 0              | 1              | 0              | 0              | 0              | 1              | 91 H         | Input  | Output                  | Output | Input                   |        |
| 1                 | 0              | 0              | 1              | 0              | 0              | 1              | 0              | 92 H         | Input  | Output                  | Input  | Output                  |        |
| 1                 | 0              | 0              | 1              | 0              | 0              | 1              | 1              | 93 H         | Input  | Output                  | Input  | Input                   |        |
| 1                 | 0              | 0              | 1              | 1              | 0              | 0              | 0              | 98 H         | Input  | Input                   | Output | Output                  |        |
| 1                 | 0              | 0              | 1              | 1              | 0              | 0              | 1              | 99 H         | Input  | Input                   | Output | Input                   |        |
| 1                 | 0              | 0              | 1              | 1              | 0              | 1              | 0              | 9A H         | Input  | Input                   | Input  | Output                  |        |
| 1                 | 0              | 0              | 1              | 1              | 0              | 1              | 1              | 9B H         | Input  | Input                   | Input  | Input                   |        |

#### 8.4 PROGRAMMING IN MODE 0

As discussed earlier, the Ports A, B, and C can be configured as simple input or output ports by writing the appropriate control word in the control word register. In the control word D<sub>7</sub> is set to 1 as 8255A is to be used in simple I/O mode, D<sub>6</sub>, D<sub>5</sub> and D<sub>2</sub> are

all set to 0 as all the ports are to be used in mode 0. The remaining bits D<sub>4</sub>, D<sub>3</sub>, D<sub>1</sub> and D<sub>0</sub> determine if the corresponding ports are to be used as input port or output port (1 for input port and 0 for output port). Since these are four bits to decide which ports are to be used as input ports and which ports are to be used as output ports, so for this purpose, there will be 16 possible combinations of control words listed in table 8.4.

For example, the control word when the ports of 8255A PPI are to be used in mode 0 with Port A as input port, port B as output port, Port C<sub>Upper</sub> as output port and Port C<sub>Lower</sub> as input port is given as:

| D <sub>7</sub> | D <sub>6</sub>     | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub>     | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> | = 91 H |
|----------------|--------------------|----------------|----------------|--------------------|----------------|----------------|----------------|--------|
| 1              | 0                  | 0              | 1              | 0                  | 0              | 0              | 1              |        |
| A              | C <sub>Upper</sub> |                | B              | C <sub>Lower</sub> |                |                |                |        |

Fig. 8.14

The control word 91 H is to be loaded to the control word register (CWR) of 8255A through the OUT instruction. Let the control word 91 H is to be loaded to control word register of 8255-I connected to the system (port address is 03 H discussed earlier). It may be done with the following instructions:

MVI A, 91 H  
OUT 03 H

Now with OUT or IN instructions, the data may be transferred to or from the output devices. The proper port address is to be given with these instructions.

**Example 8.1.** Obtain the control word when the ports of 8255A are to be used in mode 0 with port-A as input port and port B and port C as output port.

**Solution.** The control word for this case is given as:

| D <sub>7</sub> | D <sub>6</sub>     | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub>     | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> | = 90 H |
|----------------|--------------------|----------------|----------------|--------------------|----------------|----------------|----------------|--------|
| 1              | 0                  | 0              | 1              | 0                  | 0              | 0              | 0              |        |
| A              | C <sub>Upper</sub> |                | B              | C <sub>Lower</sub> |                |                |                |        |

Fig. 8.15

**Example 8.2.** Obtain the control word when the ports of 8255A are to be used in mode 0 with port-A as output port and port B as input port and port C as output port.

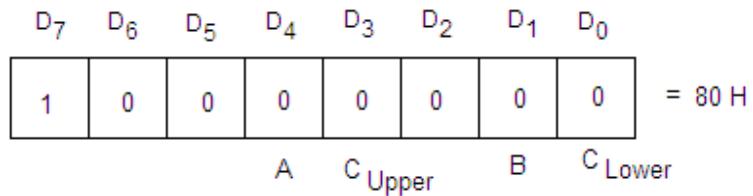
**Solution.** The control word for this case is given as:

| D <sub>7</sub> | D <sub>6</sub>     | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub>     | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> | = 82 H |
|----------------|--------------------|----------------|----------------|--------------------|----------------|----------------|----------------|--------|
| 1              | 0                  | 0              | 0              | 0                  | 0              | 1              | 0              |        |
| A              | C <sub>Upper</sub> |                | B              | C <sub>Lower</sub> |                |                |                |        |

Fig. 8.16

**Example 8.3.** Make control word when the ports of 8255A are to be used in mode 0 with all the ports as output ports.

**Solution.** The control word for this case is given as:



**Fig. 8.17**

**Example 8.4.** Write an assembly language program to generate a square wave of 1 KHz frequency using 8255A. The wave should be available at PA<sub>0</sub> terminal of Port-A.

**Solution.** The Program to perform the task given in the problem is shown below:

#### MAIN PROGRAM:

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
|       | MVI A,    | 80 H    | ; Initialize 8255-I to work all the ports as output ports.              |
|       | OUT       | 03 H    | ; Write the control word (80 H) in the control word register of 8255-I. |
| LOOP  | MVI A,    | 00 H    | ; Load 00 H to accumulator.   |
|       | OUT       | 00 H    | ; Send 00H (0V) to PA <sub>0</sub> .                                    |
|       | CALL      | DELAY   | ; Jump to delay subroutine to introduce a delay of 0.5 msec.            |
|       | MVI A,    | 01 H    | ; Load 01H to accumulator.  |
|       | OUT       | 00 H    | ; Send 01 H (5V) to PA <sub>0</sub> .                                   |
|       | CALL      | DELAY   | ; Jump to delay subroutine to introduce a delay of 0.5 msec.            |
|       | JMP       | LOOP    | ; Jump to loop to repeat the process.                                   |

#### SUBROUTINE PROGRAM:

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
| DELAY | LXI D,    | 003FH   | ; Loads DE register pair with a 16-bit number ( $63_{10}$ ).                     |
| LOOP1 | DCX D     |         | ; Decrements DE register pair by 1.  |
|       | MOV A,    | E       | ; Moves the contents of E register to accumulator.                               |
|       | ORA D     |         | ; ORing of the contents of D and E registers are performed to set the zero flag. |
|       | JNZ       | LOOP1   | ; If result is not zero than jump to LOOP1.                                      |
|       | RET       |         | ; Go back to main program.   |

The subroutine program introduces a delay of approximately 0.5 msec (discussed in section 4.3). The PA<sub>0</sub> terminal will be low for 0.5 msec and high for 0.5 msec. The total time will be 1 msec and thus a signal of 1 KHz will be generated at PA<sub>0</sub> of port-A. A CRO may be connected to this terminal to observe the wave.

**Example 8.5.** Write an assembly language program to glow 8-LEDs sequentially with a delay of one second. The LEDs are connected to 8 bits of port-B of 8255-II attached with the system.

**Solution.** The program to perform the task of the problem is given below:

### MAIN PROGRAM:

| Label  | Mnemonics | Operand | Comments   |
|--------|-----------|---------|--|
| REPEAT | MVI A,    | 80 H    | ; Initialize 8255-II to work all the ports as output ports.            |
|        | OUT       | 0B H    | ; Write the control word in the control word register of 8255-II.      |
|        | MVI A,    | 01 H    | ; Load 01 H to accumulator.  |
|        | OUT       | 09 H    | ; Send 01H (5V) to PB <sub>0</sub> for the glow of LED                 |
|        | CALL      | DELAY   | ; Jump to delay subroutine - to introduce a delay of 1 sec.            |
|        | RLC       |         | ; Rotate the contents of accumulator left for sequential glow of LEDs. |
|        | JMP       | REPEAT  | ; Jump to Repeat for the next LED to glow.                             |

### SUBROUTINE PROGRAM:

| Label   | Mnemonics        | Operand        | Comments  |
|---------|------------------|----------------|---|
| DELAY   | PUSH PSW         |                | ; Send the contents of Acc and flag register to stack.  |
| LOOP    | MVI C,<br>LXI D, | 02 H<br>F424 H | ; Load C register with 02 H data.<br>; Loads DE register pair with a 16-bit number (63) <sub>10</sub> . |
| LOOP1   | DCX D<br>MOV A,  | E              | ; Decrements DE register pair by 1.<br>; Moves the contents of E register to accumulator.               |
|         | ORA D            |                | ; ORing of the contents of D and E registers are performed to set the zero flag.                        |
| JNZ     | LOOP1            |                | ; If result is not zero than jump to LOOP1.   |
| DCR C   |                  |                | ; Decrement the contents of C register.   |
| JNZ     | LOOP             |                | ; If the result is not zero Jump to LOOP.   |
| POP PSW |                  |                | ; Acc and flag contents are popped from the stack.  |
| RET     |                  |                | ; Go back to main program.  |

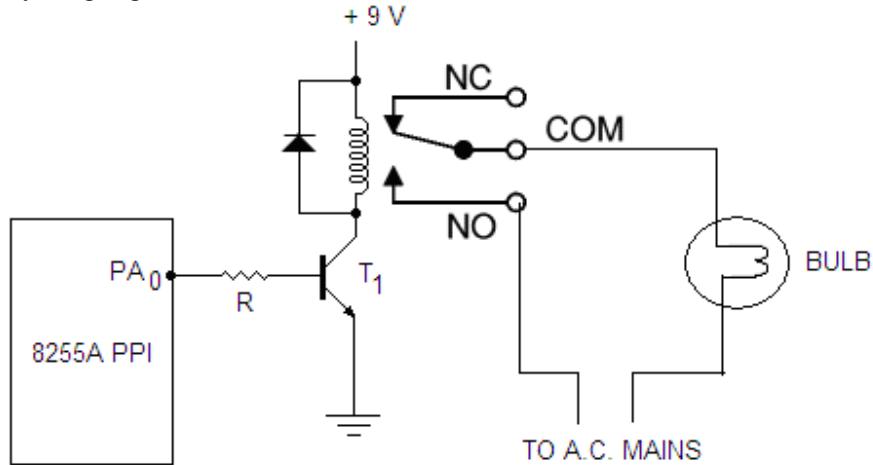
The subroutine program given here is already discussed in section 4.3.

In place of RLC in the main program RRC may be written, which will allow the LEDs to glow in the opposite sequence.

**Example 8.6.** Write a program in assembly language of 8085, to switch on an a.c. bulb after a delay of 1 Min. A relay circuit may be connected to PA<sub>0</sub> of 8255-I to switch on the bulb.

**Solution.** The relay circuit may be connected to PA<sub>0</sub> of 8255A as shown in figure 8.18. When PA0 is made high (through software), the transistor T<sub>1</sub> goes into saturation and thus energizes the relay. The normally open (N/O) terminals of the relay get connected

together and the bulb may be switched on. To provide high (logic 1) to PA<sub>0</sub>, the program in assembly language of 8085 is written as:



**Fig. 8.18**

#### MAIN PROGRAM:

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
|       | MVI A,    | 80 H    | ; Initialize 8255-I to work all the ports as output ports.       |
|       | OUT       | 03 H    | ; Write the control word in the control word register of 8255-I. |
|       | MVI A,    | 00 H    | ; Load 00 H to accumulator.                                      |
|       | OUT       | 00 H    | ; Send 00H (0V) to PA <sub>0</sub> for the bulb to remain OFF.   |
|       | CALL      | DELAY   | ; Jump to delay subroutine - to introduce a delay of 1 Min.      |
|       | MVI A,    | 01 H    | ; Load 01 to accumulator.  |
|       | OUT       | 00 H    | ; Send 01 (5V) to PA <sub>0</sub> for the bulb to ON.            |
|       | HLT       |         |  |

#### SUBROUTINE PROGRAM:

| Label | Mnemonics | operand | Comments  |
|-------|-----------|---------|---|
| DELAY | MVI C,    | 78 H    | ; Load C register with 78 H ( $120_{10}$ ) data so that the delay of 0.5sec is run 120 times (60sec). |
| LOOP  | LXI D,    | F424 H  | ; Loads DE register pair with a 16-bit number ( $63_{10}$ ).  |
| LOOP1 | DCX D     |         | ; Decrements DE register pair by 1.   |
|       | MOV A,    | E       | ; Moves the contents of E register to accumulator.  |
|       | ORA D     |         | ; ORing of the contents of D and E registers are performed to set the zero flag.                      |
| JNZ   |           | LOOP1   | ; If result is not zero than jump to LOOP1.   |

|       |      |   |
|-------|------|---|
| DCR C |      | ; Decrement the contents of C register.   |
| JNZ   | LOOP | ; If the result is not zero Jump to LOOP. |
| RET   |      | ; Go back to main program.                |

**Example 8.7.** Eight switches are connected to 8 bits of 8255 A (assume that the switches are debouncer switches) and 8 LEDs are connected to the 8 bits of Port B of 8255 A. Write an assembly language program that whenever any of the switch is depressed the corresponding LED glows. For example, if the switch number 3 is switched on, the LED connected to bit 3 should glow; similarly for the other.

**Solution.** The program to perform the task of the problem is given below:

**MAIN PROGRAM:**

| Label  | Mnemonics | Operand | Comments  |
|--------|-----------|---------|---|
|        | MVI A,    | 99 H    | ; Initialize 8255-I to make port A and port C as input ports and port B as output port. |
|        | OUT       | 03 H    | ; Write the control word in the control word register of 8255-I.                        |
| REPEAT | IN        | 00 H    | ; Send 01H (5V) to PB <sub>0</sub> for the glow of LED                                  |
|        | OUT       | 01 H    | ; Output the input data to glow the LED corresponding to depressed switch.              |
|        | JMP       | REPEAT  | ; Jump to Repeat.   |

## 8.5 PROGRAMMING IN MODE 1 (STROBED INPUT/OUTPUT)

As already discussed, in Mode 1 handshake (Strobes) signals are used for the data transfer between the microprocessor and I/O devices. The handshake signals are exchanged between the microprocessor and peripherals. Both group A and group B of 8255 can be made to operate in Mode 1. The two ports (Port A and Port B) can be configured either as input port or output port. Each port (Port A and Port B) uses three lines of port C as handshake signals (i.e. total 6 lines of port C are used for handshake signals). The remaining two lines of port C can be used for simple I/O operation.

Bits PC<sub>0</sub>, PC<sub>1</sub> and PC<sub>2</sub> of port C are used as handshake signals for port B in input and output modes. Bits PC<sub>3</sub>, PC<sub>4</sub> and PC<sub>5</sub> of port C are used as handshake signals for port-A in input mode only. The remaining bits of port C (PC<sub>5</sub> and PC<sub>6</sub>) are used as simple I/O operation as programmed by bit D<sub>3</sub> of the control word.

Bits PC<sub>3</sub>, PC<sub>6</sub> and PC<sub>7</sub> of port C are used as control (handshake) signals for port A in output mode. In this case bits PC<sub>4</sub> and PC<sub>5</sub> of port C are used as simple I/O operation as programmed by bit D<sub>3</sub> of the control word. The remaining bits PC<sub>0</sub>, PC<sub>1</sub> and PC<sub>2</sub> of port C are used for handshake signals for port B.

The combination of Mode 1 and Mode 0 operation is also possible. For example, when port A is programmed to operate in mode 1, the port B can be operated in mode 0.

### 8.5.1 Input Control Signals in Mode 1

Figure 8.19(a) shows the input configuration of both Port A and Port B to be used in mode 1. The control signals used for handshaking are also shown in this figure. Bits

$PC_3$ ,  $PC_4$  and  $PC_5$  of port C are used for handshaking when port A is used as input port; and bits  $PC_0$ ,  $PC_1$  and  $PC_2$  are used for handshaking when port B is also used for input port. Bits  $PC_6$  and  $PC_7$  are used for simple input/output operation. The control word for using Port A and Port B as input ports in mode 1 is shown in figure 8.19(b).

The functions of three control signals used for handshaking are given as:

**STB (Strobe Input)**

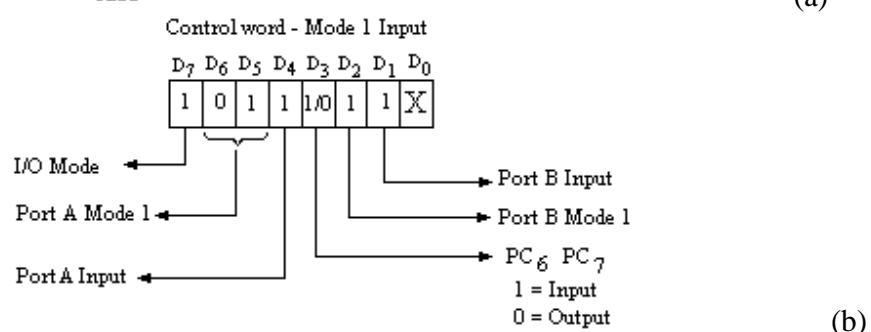
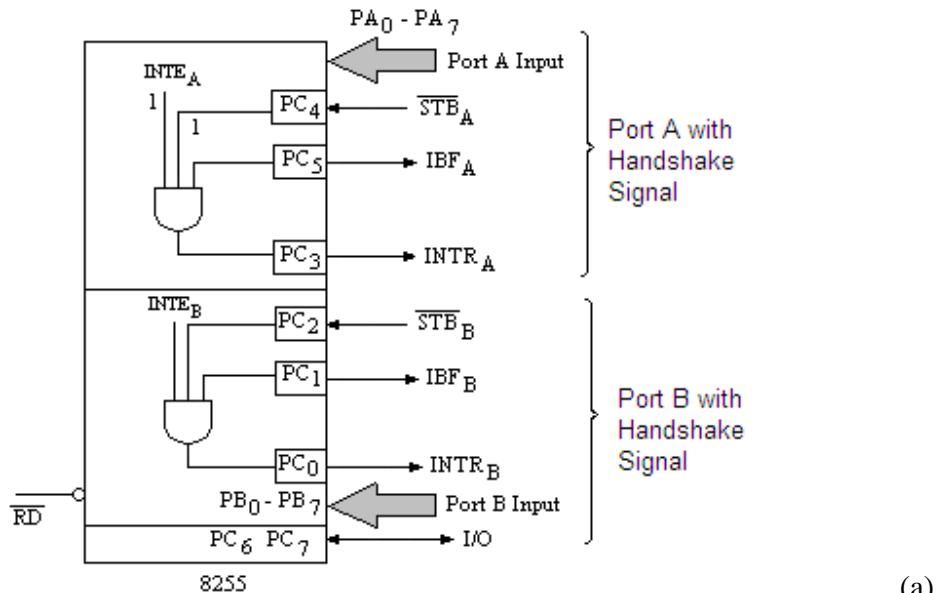
This is an active low signal generated by the peripheral device to indicate that it has transmitted a byte of data. In response to this signal, 8255 generates IBF (Input Buffer Full) and INTR (Interrupt Request) signals.

**IBF (Input Buffer Full)**

This is an acknowledge signal sent by 8255. A high to this signal indicates that the input latch has received the data byte.

**INTR (Interrupt Request)**

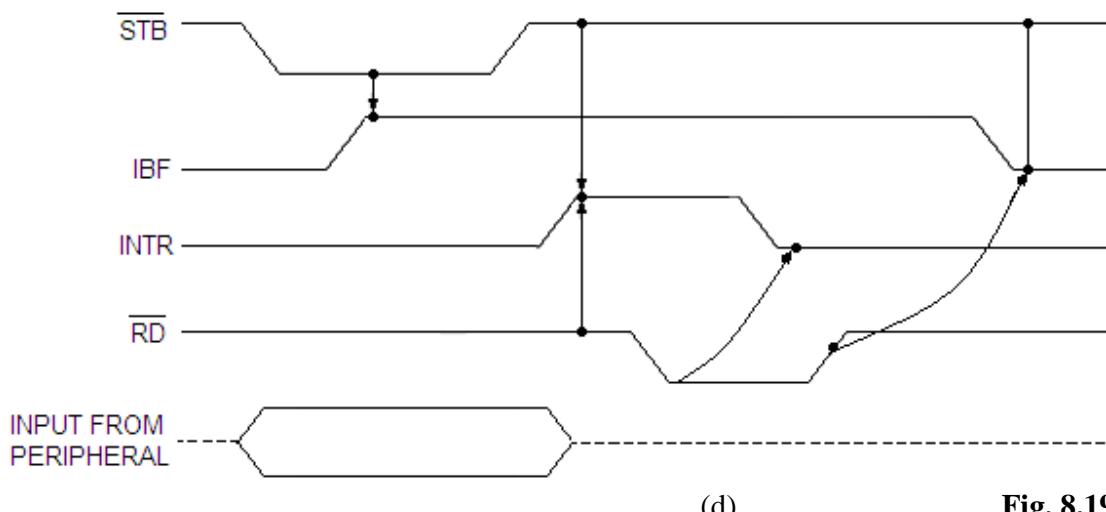
This is an output signal sent by 8255 which may be used to interrupt the microprocessor. This signal is generated when STB, IBF and INTE (Interrupt Enable) are all low (logic 0). At the falling edge of the  $\overline{RD}$  signal, this is reset.



| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub>    | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub>    |
|----------------|----------------|----------------|----------------|-------------------|----------------|----------------|-------------------|
| I/O            | I/O            | IBFA           | INTEA          | INTR <sub>A</sub> | INTEB          | IBFB           | INTR <sub>B</sub> |

Status Word - Mode 1 Input

(c)



(d)

**Fig. 8.19**

#### INTE (Interrupt Enable)

This is an internal flip-flop which is used to enable or disable the generation of INTR signal. The two flip-flops **INTE<sub>A</sub>** and **INTE<sub>B</sub>** are set/reset using the BSR mode. The bit PC<sub>4</sub> enables or disables INTE<sub>A</sub>; and PC<sub>2</sub> enables or disables INTE<sub>B</sub>.

Figure 8.19(c) shows the status word (i.e. the status of the signals INTR, INTEN and IBF signals etc.) which will be placed in the accumulator if the port C is read.

As shown in figure 8.19(d), the peripheral first loads data into the input port A or B by making **STB** input low. This low **STB** signal is generated by the peripheral device to indicate that it has transmitted a byte of data. The 8255A generates IBF and INTR in response to the active low **STB** signal.

#### 8.5.2 Output Control Signals in Mode 1

As already discussed Bits PC<sub>3</sub>, PC<sub>6</sub> and PC<sub>7</sub> of port C are used as control (handshake) signals for port A in output mode. In this case bits PC<sub>4</sub> and PC<sub>5</sub> of port C are used as simple I/O operation as programmed by bit D<sub>3</sub> of the control word. The remaining bits PC<sub>0</sub>, PC<sub>1</sub> and PC<sub>2</sub> of port C are used for handshake signals for port B.

Figure 8.20(a) shows the input configuration of both Port A and Port B to be used in mode 1. The control signals used for handshaking are also shown in this figure. Bits PC<sub>3</sub>, PC<sub>6</sub> and PC<sub>7</sub> of port C are used for handshaking when port A is used as output port; and bits PC<sub>0</sub>, PC<sub>1</sub> and PC<sub>2</sub> are used for handshaking when port B is also used for output port. Bits PC<sub>4</sub> and PC<sub>5</sub> are used for simple input/output operation. The control word for using Port A and Port B as input ports in mode 1 is shown in figure 8.20(b).

The functions of three control signals used for handshaking are given as:

**OBF (Output Buffer Full)** This is an active low signal; it activates when the microprocessor writes data into output latch of 8255. This

indicates to an output peripheral that a new data is ready to be read. It goes high again after the 8255 receives a  $\overline{\text{ACK}}$  signal from peripheral.

#### **$\overline{\text{ACK}}$ (Acknowledge)**

This is an input signal from a peripheral. It becomes active (low) when peripheral receives the data from 8255 ports.

#### **INTR (Interrupt Request)**

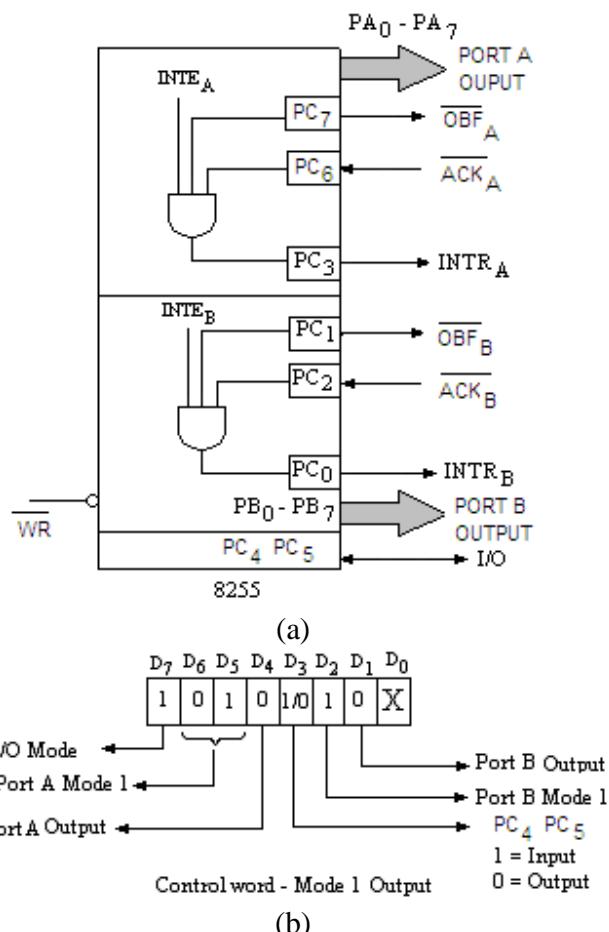
This is an output signal which is set at the rising edge of the  $\overline{\text{ACK}}$  signal. This signal can be used to interrupt the microprocessor. It requests for the next data byte for output. This signal is generated when  $\overline{\text{OBF}}$ ,  $\overline{\text{ACK}}$  and INTE (Interrupt Enable) are all high (logic 1). At the falling edge of the  $\overline{\text{WR}}$  signal this is reset.

#### **INTE (Interrupt Enable)**

This is an internal flip-flop to a port and required to be set to generate the INTR signal. The two flip-flops  $\text{INTE}_A$  and  $\text{INTE}_B$  are set/reset using the BSR mode.

Figure 8.20(c) shows the status word (i.e. the status of the signals INTR, INTE and OBF signals etc.) which will be placed in the accumulator if the port C is read.

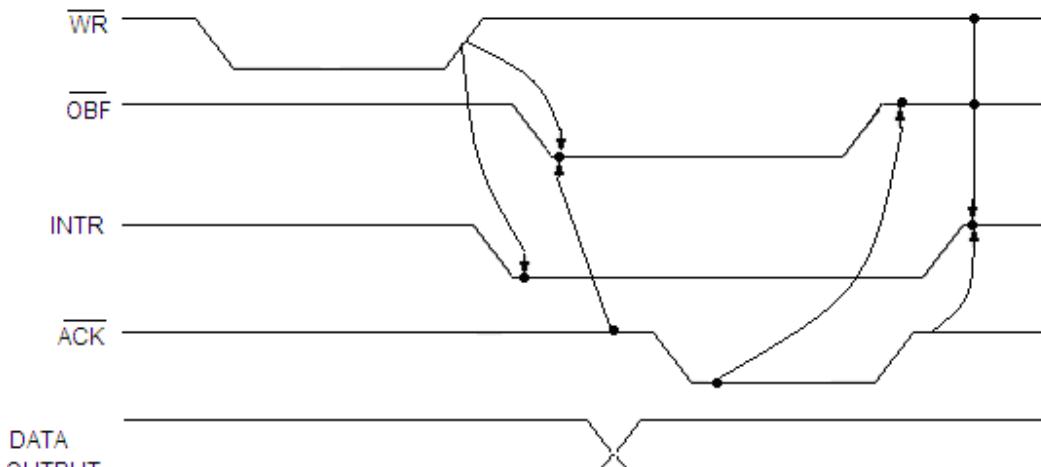
Figure 8.20(d) shows the timing diagram for strobed Output.



| D <sub>7</sub>   | D <sub>6</sub>    | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub>    | D <sub>2</sub>    | D <sub>1</sub>   | D <sub>0</sub>    |
|------------------|-------------------|----------------|----------------|-------------------|-------------------|------------------|-------------------|
| OBF <sub>A</sub> | INTE <sub>A</sub> | I/O            | I/O            | INTR <sub>A</sub> | INTE <sub>B</sub> | OBF <sub>B</sub> | INTR <sub>B</sub> |

Status Word - Mode 1 Output

(c)



(d) Fig. 8.20

**Example 8.8.** Obtain the control word for the following configuration of the ports of 8255 A for mode 1 operation:

Port A – as input port,

Mode for Port A – mode 1

Port B – input port

Mode for Port B – mode 1,

The remaining pins of port C<sub>Upper</sub> are to be used output pins.

**Solution.** The 8255 A is to be used in mode 1, and port A as input port so PC<sub>3</sub>, PC<sub>4</sub> and PC<sub>5</sub> pins of port C will be used as the handshake signals. For port B as input port the PC<sub>0</sub>, PC<sub>1</sub> and PC<sub>2</sub> pins of port C will be used as the handshake signals. The remaining pins of port C<sub>Upper</sub> will be used as simple input output mode. In the present example PC<sub>6</sub> and PC<sub>7</sub> are to be used as output pins. The control word for the same may be given as (fig. 8.21):

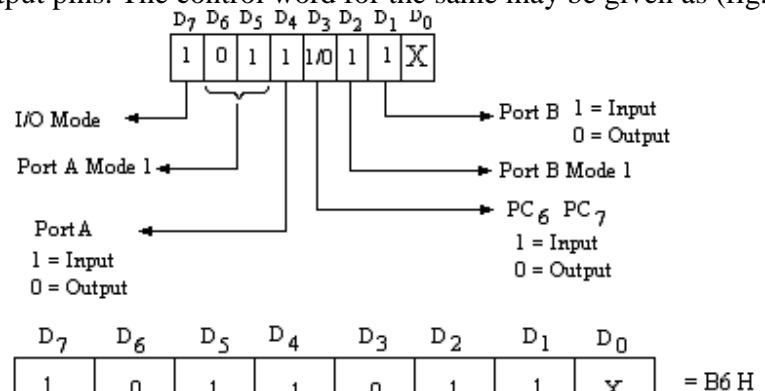


Fig. 8.21

**Example 8.9.** Obtain the control word for the following configuration of the ports of 8255 A for mode 1 operation:

Port A – as output port,

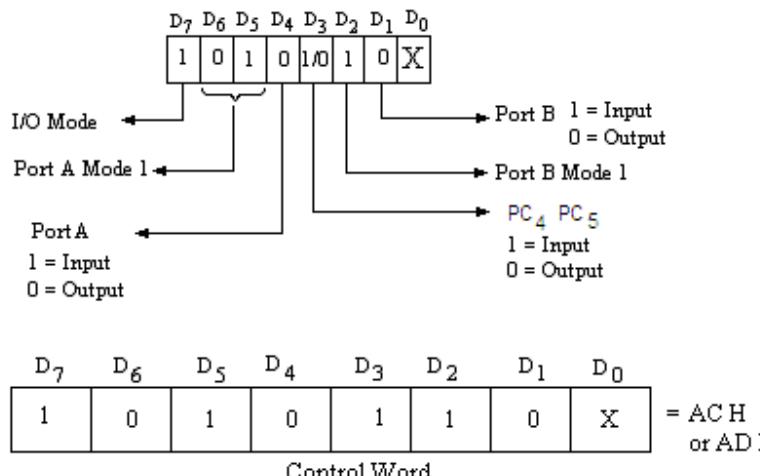
Mode for Port A – mode 1

Port B – output port,

Mode for Port B – mode 1,

The remaining pins PC<sub>4</sub> and PC<sub>5</sub> of port C are to be used input pins.

**Solution.** The 8255 A is to be used in mode 1, and port A as output port so PC<sub>3</sub>, PC<sub>6</sub> and PC<sub>7</sub> pins of port C will be used as the handshake signals. For port B as output port the PC<sub>0</sub>, PC<sub>1</sub> and PC<sub>2</sub> pins of port C will be used as the handshake signals. The remaining pins of port C will be used as simple input output mode. In the present example PC<sub>4</sub> and PC<sub>5</sub> are to be used as input pins. The control word for the same may be given as (fig. 8.22):



**Fig. 8.22**

**Example 8.10.** Obtain the control word for the following configuration of the ports of 8255 A:

Port A – as input port,

Mode for Port A – mode 1

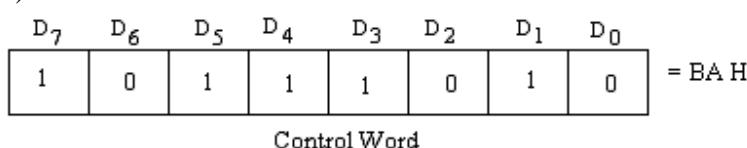
Port B – input port,

Mode for Port B – mode 0,

Port C<sub>Lower</sub> – output port

The remaining pins PC<sub>6</sub> and PC<sub>7</sub> of port C are to be used input pins.

**Solution.** The 8255 A is to be used in mode 1 and mode 0, and port A as input port in mode 1 so PC<sub>3</sub>, PC<sub>6</sub> and PC<sub>7</sub> pins of port C will be used as the handshake signals. The port B is to be used as input port in mode 0. The Port C<sub>Lower</sub> is used as output port and remaining bits of port C (PC<sub>6</sub>, PC<sub>1</sub>) as input pins. The control word for the same may be given as (fig. 8.23):



**Fig. 8.23**

## 8.6 PROGRAMMING IN MODE 2 (STROBED BIDIRECTIONAL BUS I/O)

In mode 2 operation of 8255A, port A can be used as bidirectional 8 bit I/O bus. The port B can operate in mode 0 or mode 1.

When port A is programmed to operate in mode 2 (bidirectional I/O bus), the five bits of port C ( $PC_3-PC_7$ ) are used as control (handshake) signals for port A. This is illustrated in figure 8.24. In this case port B can be programmed in mode 0 or mode 1. If port B is used in mode 0, the remaining bits of port C ( $PC_0-PC_2$ ) are used as input or output. However, if port B is used in mode 1,  $PC_0-PC_2$  bits are used as handshake signals for port B.

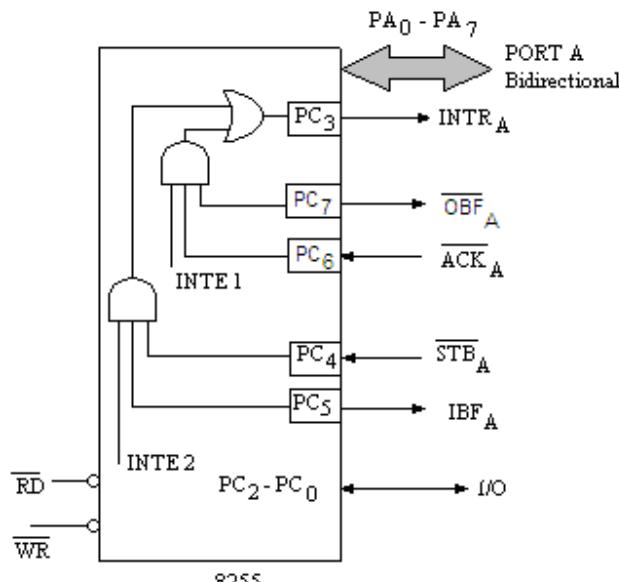


Fig. 8.24

The functions of five control signals used for handshaking of port A are given as:

- |                                 |   |
|---------------------------------|---|
| <b>STB (Strobe Input)</b>       | This is an active low signal generated by the peripheral device to indicate that it has transmitted a byte of data. In response to this signal, 8255 generates IBF (Input Buffer Full) and INTR (Interrupt Request) signals.  |
| <b>IBF (Input Buffer Full)</b>  | This is an acknowledge signal sent by 8255. A high to this signal indicates that the input latch has received the data byte.  |
| <b>INTR (Interrupt Request)</b> | This is an output signal which is set at the rising edge of the $\overline{ACK}$ signal. This signal can be used to interrupt the microprocessor. It requests for the next data byte for output. This signal is generated when $\overline{OBF}$ , $\overline{ACK}$ and INTE (Interrupt Enable) are all high (logic 1). At the falling edge of the $\overline{WR}$ signal this is reset. |
| <b>INTE (Interrupt Enable)</b>  | This is an internal flip-flop to a flip-flop to a port and required to be set to generate the INTR signal. The two flip-flops $INTE_1$ and $INTE_2$ are set/reset using the BSR mode.   |

**OBF (Output Buffer Full)** This is an active low signal; it activates when the microprocessor writes data into output latch of 8255. This indicates to an output peripheral that a new data is ready to be read. It goes high again after the 8255 receives a ACK signal from peripheral.

If all the possible combinations are considered, then there will be four configurations of the 8255A in mode 2. These combinations with their corresponding control words are shown in figures 8.25 to 8.28. The status word in mode 2 is shown in figure 8.29, which will be placed in the accumulator if the port C is read.

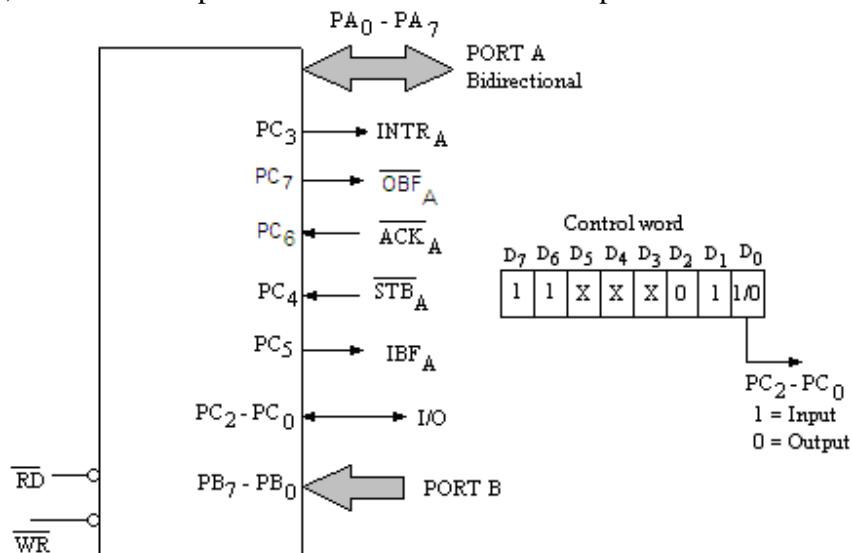


Fig. 8.25

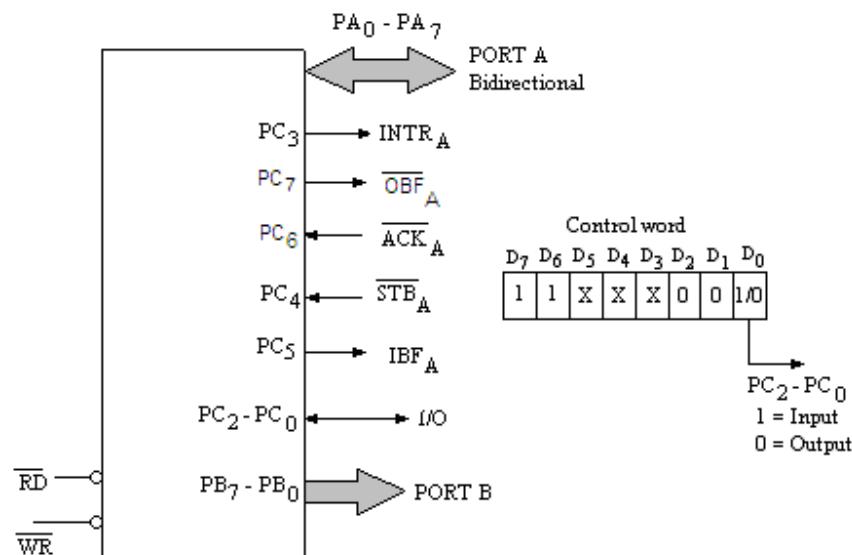
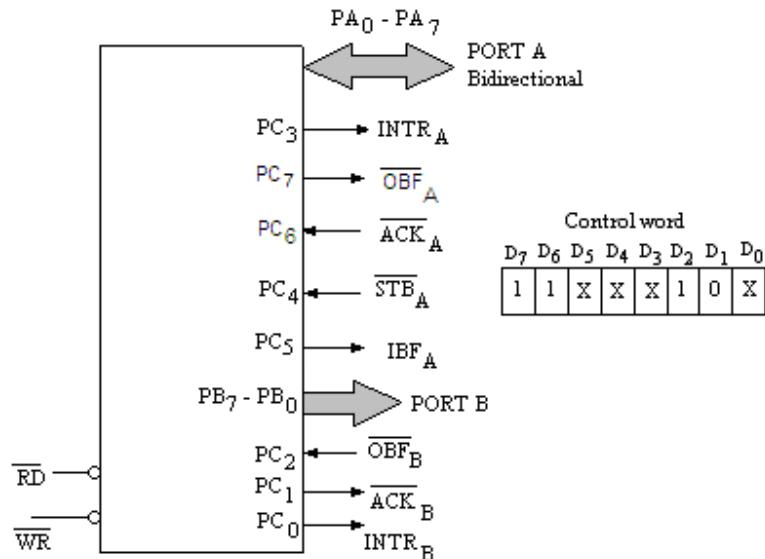
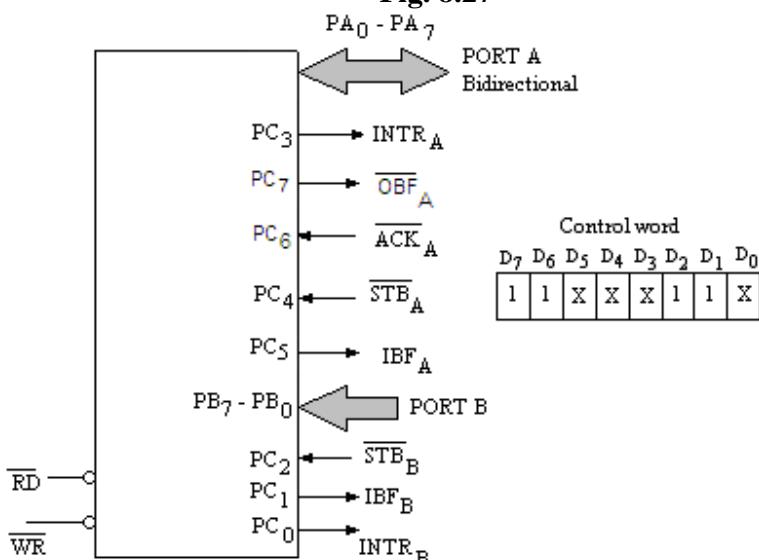


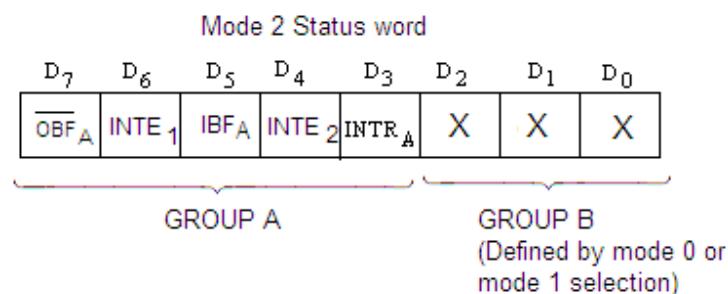
Fig. 8.26



**Fig. 8.27**



**Fig. 8.28**



**Fig. 8.29**

**Example 8.11.** Obtain the control word for the following configuration of the ports of 8255 A:

*Port A – as bidirectional,*  
*Mode for Port A – mode 2*  
*Port B – input port,*  
*Mode for Port B – mode 0,*  
*Port C<sub>Lower</sub> – input port*

**Solution.** . It may be remembered that the 8255 A is used in mode 2 only for port A. When port A is used to operate in mode 2; port B can be used either in mode 0 or mode 1. The pins of port C (PC<sub>3</sub> to PC<sub>7</sub>) are used for the control signals for port A. For port B to operate in mode 0, Port C<sub>Lower</sub> can be used as input or output pins as per the D<sub>0</sub> bit of the control word. On the other hand for the port B to operate in mode 1, PC<sub>0</sub>, PC<sub>1</sub> and PC<sub>2</sub> bits will be used for control signals.

The control word for this case is given as (fig. 8.30):

| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> | = C3 H |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| 1              | 1              | X              | X              | X              | 0              | 1              | 1              |        |

Control Word

**Fig. 8.30**

**Example 8.12.** Obtain the control word for the following configuration of the ports of 8255 A:

*Port A – as bidirectional,*  
*Mode for Port A – mode 2*  
*Port B – input port,*  
*Mode for Port B – mode 1,*

**Solution.**

The control word for this case is given as (fig. 8.31):

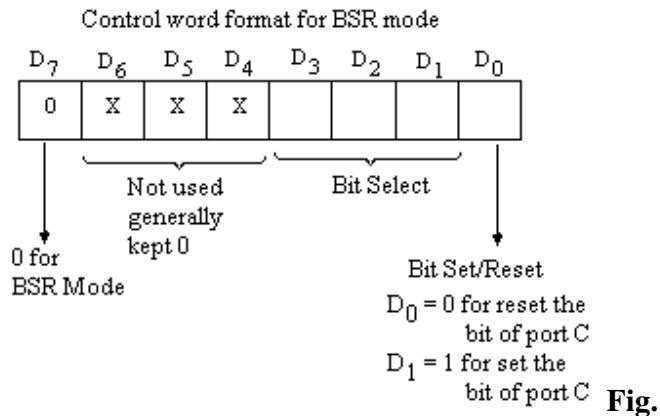
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> | = C6 H |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| 1              | 1              | X              | X              | X              | 1              | 1              | X              |        |

Control Word

**Fig. 8.31**

## 8.7 BIT SET/RESET (BSR) MODE

As discussed in the earlier sections of this chapter that 8255A PPI can be used in bit set/reset (BSR) mode. To use the PPI in BSR mode D<sub>7</sub> bit of the control word should be set to zero. However, for other mode of operations this bit should be set to 1. In BSR mode eight bits of Port C can be set or reset by writing the control word in the control word register. If a control word with bit D<sub>7</sub> as 0 is recognized as a BSR mode then it does not alter any previously transmitted control word with D<sub>7</sub> as 1; thus I/O operations are not affected by a BSR control word. In this mode of operation individual bit of port C can be set or reset or in other words a BSR control word affects only one bit of port C. The control word format for this mode of operation is shown in figure 8.32.



**Fig. 8.32**

The bits of the control word are defined as:

- Bit D<sub>7</sub>** This bit should be zero, to use 8255A in bit set/reset mode.
- Bits D<sub>4</sub>-D<sub>6</sub>** These bits are don't care bits, which may be kept 0s or 1s but generally kept 0.
- Bits D<sub>1</sub>-D<sub>3</sub>** The bits D<sub>3</sub>, D<sub>2</sub> and D<sub>1</sub> are known as bit select pins. Table 8.5 shows which bit of port C will be affected by the D<sub>3</sub>, D<sub>2</sub> and D<sub>1</sub> bits of bit select pins.

**Table 8.5**

| D3 | D2 | D1 | Bit of port C   |
|----|----|----|-----------------|
| 0  | 0  | 0  | PC <sub>0</sub> |
| 0  | 0  | 1  | PC <sub>1</sub> |
| 0  | 1  | 0  | PC <sub>2</sub> |
| 0  | 1  | 1  | PC <sub>3</sub> |
| 1  | 0  | 0  | PC <sub>4</sub> |
| 1  | 0  | 1  | PC <sub>5</sub> |
| 1  | 1  | 0  | PC <sub>6</sub> |
| 1  | 1  | 1  | PC <sub>7</sub> |

#### Bit D<sub>0</sub>

This is known as Set reset bit. To set a bit of port C, D<sub>0</sub> bit of the control word should be 1; and to reset a bit of port C, D<sub>0</sub> of the control word should be 0.

For example to reset PC<sub>3</sub> (bit 3 of port C), the control word format will be given as:

|                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |
| 0              | 0              | 0              | 0              | 0              | 1              | 1              | 0              |

= 06 H  
Bit 3 (PC<sub>3</sub>)

Similarly, to set PC<sub>3</sub>, the control word format is given as:

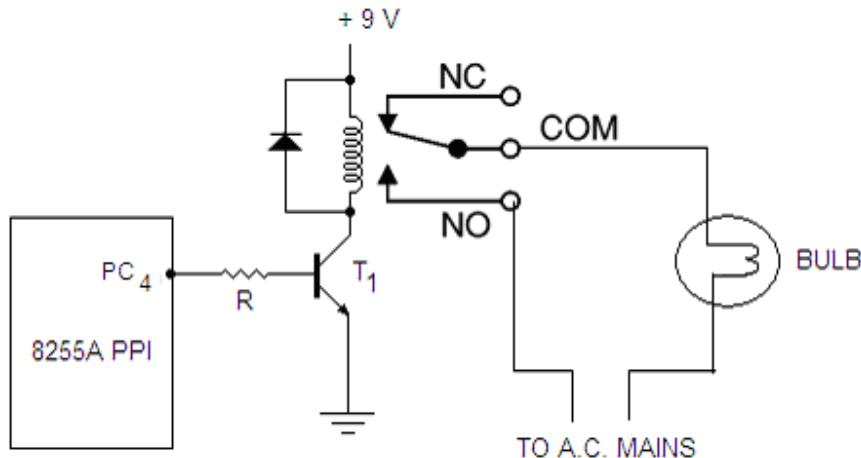
|                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |
| 0              | 0              | 0              | 0              | 0              | 1              | 1              | 1              |

= 07 H  
Bit 3 (PC<sub>3</sub>)

**Example 8.13.** A relay circuit, to switch on or off an a.c. mains bulb, is connected to PC<sub>4</sub> (4<sup>th</sup> bit of port C). The 8255A is connected microprocessor whose address for control

word is 03 H. Using BSR mode it is desired to switch on and off the bulb after a regular interval of time. Write a program in assembly language of 8085, to implement the above.

**Solution.** The relay circuit may be connected to PC<sub>4</sub> of 8255A as shown in figure 8.33. To switch on and off the bulb regularly using BSR, the program in assembly language of 8085 is written as:



**Fig. 8.33**

The control word in BSR mode to reset PC<sub>4</sub> bit is:

|                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> | = 08 H |
| 0              | 0              | 0              | 0              | 1              | 0              | 0              | 0              |        |

The control word in BSR mode to set PC<sub>4</sub> bit is:

|                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> | = 09 H |
| 0              | 0              | 0              | 0              | 1              | 0              | 0              | 1              |        |

#### MAIN PROGRAM:

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
|       | LXI SP,   | XXXX H  | ; Initialize stack pointer.   |
| START | MVI A,    | 08 H    | ; Load byte in accumulator to reset PC <sub>4</sub> .                       |
|       | OUT       | 03 H    | ; Reset PC <sub>4</sub> (the bulb will be off).                             |
|       | MVI A,    | 08 H    | ; Load 08 H to accumulator.   |
|       | OUT       | 03 H    | ; Reset PC <sub>4</sub> (the bulb will be off).                             |
|       | CALL      | DELAY   | ; Jump to delay subroutine - to introduce a regular delay of constant time. |
|       | MVI A,    | 09 H    | ; Load 09 to accumulator to set PC <sub>4</sub> .                           |
|       | OUT       | 03 H    | ; Set PC <sub>4</sub> (the bulb will be on).                                |
|       | CALL      | DELAY   | ; Jump to delay subroutine - to introduce a regular delay of constant time. |
| JMP   | START     |         | ; Jump to repeat the process.   |

#### SUBROUTINE PROGRAM:

| Label | Mnemonics | Operand | Comments |
|-------|-----------|---------|----------|
|-------|-----------|---------|----------|

|       |        |        |  |
|-------|--------|--------|--|
| DELAY | LXI D, | XXXX H | ; Loads DE register pair with a 16-bit number.                                   |
| LOOP  | DCX D  |        | ; Decrements DE register pair by 1.  |
|       | MOV A, | E      | ; Moves the contents of E register to accumulator.                               |
|       | ORA D  |        | ; ORing of the contents of D and E registers are performed to set the zero flag. |
|       | JNZ    | LOOP   | ; If result is not zero than jump to LOOP1.                                      |
|       | RET    |        | ; Go back to main program.   |

XXXX H may be taken any hexadecimal number for introducing any desired delay.

**Example 8.14.** Write a program in assembly language of 8085 to send 01H, 02H, 03H . . . FFH data to port A of 8255A PPI with a time delay of 1msec to each output.

**Solution.** The program to implement the above is given as:

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
|       | LXI SP,   | XXXX H  | ; Initialize stack pointer.                                      |
|       | MVI A,    | 80 H    | ; Initialize 8255-I to work all the ports as output ports.       |
|       | OUT       | 03 H    | ; Write the control word in the control word register of 8255-I. |
|       | MVI A,    | 00 H    | ; Load 00 H to accumulator.                                      |
|       | MVI B,    | FF H    | ; Load FF H to B-reg (last data).                                |
| LOOP  | INR A     |         | ; Increment acc.   |
|       | OUT       | 00 H    | ; Send the number to port A.                                     |
|       | CALL      | DELAY   | ; Jump to delay subroutine - to introduce a delay of 1 msec.     |
|       | DCR B     |         | ; Decrement B.   |
|       | JNZ       | LOOP    | ; If B is not zero jump to LOOP.                                 |
|       | HLT       |         |  |

#### SUBROUTINE PROGRAM:

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
| DELAY | PUSH PSW  |         | ; Push the program status word to stack.   |
|       | PUSH B    |         | ; Push the contents of B-C reg pair to stack.                                    |
|       | LXI D,    | XXXX H  | ; Loads DE register pair with a 16-bit number to introduce a delay of 1msec.     |
| LOOP  | DCX D     |         | ; Decrements DE register pair by 1.  |
|       | MOV A,    | E       | ; Moves the contents of E register to accumulator.                               |
|       | ORA D     |         | ; ORing of the contents of D and E registers are performed to set the zero flag. |

|         |      |   |
|---------|------|---|
| JNZ     | LOOP | ; If result is not zero than jump to LOOP1.         |
| POP B   |      | ; Pop the contents from stack to B-C reg pair back. |
| POP PSW |      | ; Pop program status word from stack.               |
| RET     |      | ; Go back to main program.                          |

**Example 8.15.** Write a program in assembly language of 8085 that inputs 256 bytes of data from port B of 8255A PPI with a time delay of 1msec to each input and store these bytes of data at memory locations starting at 2500 H.

**Solution.** The program to implement the above is given as:

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
| LOOP  | LXI SP,   | XXXX H  | ; Initialize stack pointer.   |
|       | MVI A,    | 82 H    | ; Initialize 8255-I to work port B as input port and ports A and C as output ports. |
|       | OUT       | 03 H    | ; Write the control word in the control word register of 8255-I.                    |
|       | LXI H,    | 2500 H  | ; Initialize H-L reg pair with first address of destination location.               |
|       | MVI C,    | FF H    | ; Load FF H to C reg. as pointer.   |
|       | IN        | 01 H    | ; Input from port B.  |
|       | MOV M,    | A       | ; Load the accumulator content to memory location.                                  |
|       | CALL      | DELAY   | ; Jump to delay subroutine - to introduce a delay of 1 msec.                        |
|       | INX H     |         | ; Increment H-L pair.   |
|       | DCR C     |         | ; Decrement C.  |
|       | JNZ       | LOOP    | ; If C is not zero jump to LOOP.  |
|       | HLT       |         |   |

#### SUBROUTINE PROGRAM:

The delay subroutine program is the same as given in the previous example.

**Example 8.16.** Suppose 4096 bytes of data are stored in 2000 H to 2FFF H memory locations. Write a program in assembly language of 8085 that outputs these bytes of data to port A of 8255A PPI with a time delay of 1msec to each output.

**Solution.** The program to implement the above is given as:

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
|       | LXI SP,   | XXXX H  | ; Initialize stack pointer.                                      |
|       | MVI A,    | 80 H    | ; Initialize 8255-I to work all the ports as output ports.       |
|       | OUT       | 03 H    | ; Write the control word in the control word register of 8255-I. |
|       | LXI H,    | 19FF H  | ; Initialize H-L reg pair as pointer.                            |

|      |            |  |  |
|------|------------|--|--|
|      | INX H      |  | ; Increment H-L reg. pair.                                   |
|      | MOV A, M   |  | ; Load the data to accumulator.                              |
| LOOP | OUT 00 H   |  | ; Output the data to port A.                                 |
|      | CALL DELAY |  | ; Jump to delay subroutine - to introduce a delay of 1 msec. |
|      | CPI 2F H   |  | ; Compare accumulator contents to 2F H.                      |
|      | JNZ LOOP   |  | ; If not zero jump to LOOP.                                  |
|      | MOV A, L   |  | ; Load L-reg data to accumulator.                            |
|      | CPI FF H   |  | ; Compare with FF H.   |
|      | JNZ LOOP   |  | ; If not zero jump to LOOP.                                  |
|      | HLT        |  |  |

#### SUBROUTINE PROGRAM:

The delay subroutine program is the same as given in example 8.14.

**Example 8.17.** A peripheral device is sending serial data to  $D_7$  bit of port B of 8255A at an interval of 1msec. Write a program in assembly language of 8085 that converts 8 bit serial data stream to 8 bit parallel word, which is then sent to port A of 8255A.

**Solution.** The program to implement the above is given as:

| Label | Mnemonics      | Operand | Comments  |
|-------|----------------|---------|---|
|       | LXI SP, XXXX H |         | ; Initialize stack pointer.   |
|       | MVI A, 82 H    |         | ; Initialize 8255-I to work port B as input port and ports A and C as output ports. |
|       | OUT 03 H       |         | ; Write the control word in the control word register of 8255-I.                    |
|       | MVI B, 00 H    |         | ; Store 00 H in register B.   |
|       | MVI C, 07 H    |         | ; Store 07 H in register C as counter.  |
|       | IN 01 H        |         | ; Inputs a byte through port B.   |
|       | ORA B          |         | ; ORing for taking 7 <sup>th</sup> bit.   |
|       | RAR            |         | ; Change the position.  |
|       | MOV B, A       |         | ; Store to B register.  |
|       | CALL DELAY     |         | ; Jump to delay subroutine - to introduce a delay of 1 msec.                        |
|       | DCR C          |         | ; Decrement C.  |
|       | JNZ LOOP       |         | ; If not zero jump to LOOP.   |
|       | OUT 00 H       |         | ; Outputs the data to port A.   |
|       | HLT            |         |   |

#### SUBROUTINE PROGRAM:

The delay subroutine program is the same as given in example 8.14.

**Example 8.18.** Suppose 256 bytes of data are stored in the memory locations 2000 H to 20FF H. Write a program in assembly language of 8085 that converts each of these bytes into serial data stream. The serial data stream is sent (bit by bit) through  $D_0$  bit of port A of 8255A at the rate of approximately 1000 Bits/sec. (i.e. 1msec time interval between two bits).

**Solution.** The program to implement the above is given as:

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>   |
|--------------|------------------|----------------|---|
|              | LXI SP,          | XXXX H         | ; Initialize stack pointer.   |
|              | MVI A,           | 80 H           | ; Initialize 8255-I to work all the ports as output ports.                |
|              | OUT              | 03 H           | ; Write the control word in the control word register of 8255-I.          |
| LOOP         | LXI H,<br>INX H  | 1FFF H         | ; Initialize the H-L register pair.<br>; Increment the H-L register pair. |
|              | MOV A,           | M              | ; Store the byte in accumulator.  |
|              | MVI B,           | 00 H           | ; Store 00 H in register B.   |
| AGAIN        | OUT              | 00 H           | ; Outputs the bit through D <sub>0</sub> bit of port A.                   |
|              | CALL             | DELAY          | ; Jump to delay subroutine - to introduce a delay of 1 msec.              |
|              | RAR              |                | ; Rotate right for the next bit.  |
|              | DCR B            |                | ; Decrement B.  |
|              | JNZ              | AGAIN          | ; If not zero jump to AGAIN.  |
|              | MOV A,           | L              | ; Move the contents of L reg. to accumulator.                             |
|              | CPI              | FF H           | ; Compare with FF H   |
|              | JNZ              | LOOP           | ; If not equal jump to LOOP.  |
|              | HLT              |                |   |

### SUBROUTINE PROGRAM:

The delay subroutine program is the same as given in example 8.14.

**Example 8.19.** Write a program in assembly language of 8085 that inputs a byte from port A of 8255A and determines if the decimal equivalent of the byte is even or odd. If byte is even, then send ASCII E (45 H) to port B and send ASCII O (4F H) to port B if the byte is odd.

**Solution.** The program to implement the above is given as:

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>   |
|--------------|------------------|----------------|---|
|              | MVI A,           | 90 H           | ; Initialize 8255-I to work port A as input port and ports B and C as output ports. |
|              | OUT              | 03 H           | ; Write the control word in the control word register of 8255-I.                    |
|              | IN               | 00 H           | ; Inputs the byte through port A.   |
|              | ANI              | 01 H           | ; ANDed with 01 H to check D <sub>0</sub> bit of the Byte.                          |
|              | JNZ              | ODD            | ; If not zero (odd byte) jump to ODD.   |
|              | MVI A,           | 45 H           | ; Load accumulator for ASCII E.   |
|              | JMP              | END            | ; Jump to END.  |
| ODD          | MVI A,           | 4F H           | ; Load accumulator for ASCII O.   |

|     |     |      |  |
|-----|-----|------|--|
| END | OUT | 01 H | ; Output ASCII E or ASCII O to port B. |
|     |     |      | HLT                                    |

**Example 8.20.** Write a program in assembly language of 8085 that inputs a byte from port A of 8255A and determines if the decimal equivalent of the byte is even or odd. If byte is even, then send ASCII E (45 H) to D<sub>0</sub> bit of port B in serial fashion; if odd send ASCII O (4F H) to D<sub>0</sub> bit of port B in serial fashion. The time interval between two consecutive bits should be 1msec.

**Solution.** The program to implement the above is given as:

| Label  | Mnemonics | Operand | Comments  |
|--------|-----------|---------|---|
|        | MVI A,    | 90 H    | ; Initialize 8255-I to work port A as input port and ports B and C as output ports. |
|        | OUT       | 03 H    | ; Write the control word in the control word register of 8255-I.                    |
|        | IN        | 00 H    | ; Inputs the byte through port A.   |
|        | ANI       | 01 H    | ; ANDed with 01 H to check D <sub>0</sub> bit of the Byte.                          |
|        | JNZ       | ODD     | ; If not zero (odd byte) jump to ODD.   |
|        | MVI A,    | 45 H    | ; Load accumulator for ASCII E.   |
|        | JMP       | END     | ; Jump to END.  |
| ODD    | MVI A,    | 4F H    | ; Load accumulator for ASCII O.   |
| END    | MVI C,    | 08 H    | ; Store 08 H to C register.   |
| REPEAT | OUT       | 01 H    | ; Output ASCII E or ASCII O to D <sub>0</sub> bit of port B.                        |
|        | CALL      | DELAY   | ; Call Delay subroutine to introduce a time delay of 1msec.                         |
|        | RAR       |         | ; Rotate right.   |
|        | DCR C     |         | ; Decrement C.  |
|        | JNZ       | REPEAT  | ; If not zero jump to REPEAT.   |
|        | HLT       |         |   |

### SUBROUTINE PROGRAM:

The delay subroutine program is the same as given in example 8.14.

### PROBLEMS

1. Using the functional block diagram of 8255A PPI, explain its detail.
2. Mention various modes of operations of 8255A PPI; explain its working in simple I/O mode.
3. Draw the schematic block diagram of 8255A and explain the function of each block.
4. Mention various modes of operations of 8255A and explain it working in BSR mode. Explain also its control word format.

5. Draw and explain the control word format of 8255A. What will be control word if 8255 is used in simple I/O mode with port A as input port, port B as output port and port C<sub>Lower</sub> as input port and Port C<sub>Upper</sub> as output port?
6. Discuss the mode 1 output configuration of 8255. Discuss also its control word, control signals, timing diagram and status word.
7. Discuss the mode 1 Input configuration of 8255. Discuss also its control word, control signals, timing diagram and status word.
8. Explain how 8255 can be used in mode 2. Give the format of the control word when port A is used to operate in mode 2 and port B is operated in mode 1.
9. Give the format of the control word when port A is used to operate in mode 2 and port B is operated in mode 0. Port C<sub>Lower</sub> is used as simple input or output.
10. Write an assembly language program of 8085A to output the contents of B-register to port A of 8255 A PPI.
11. Eight LEDs are connected to port B of 8255 A. write an assembly language program of 8085 to glow LEDs 0, 2, 4, 6 for 1 sec and then glow LEDs 1, 3, 5, 7 for the same time and repeat.
12. Obtain the control word when the ports of 8255A are to be used in mode 0 with port A as input port, port B as output port, and port C<sub>Lower</sub> as output port and port C<sub>upper</sub> as input port.

**(Ans. 98 H)**

13. Make a control word when the ports of 8255 A are to be used in mode 0 with all the ports as input ports.

**(Ans. 9B H)**

14. Obtain the control word for the following configuration of the ports of 8255 A for Mode 0 operation:  
Port A and port B as output ports and port C<sub>Lower</sub> as output port; and port C<sub>upper</sub> as input port.

**(Ans. 88 H)**

15. Obtain the control word for the following configuration of the ports of 8255 A for mode 1 operation:

Port A – as input port,  
Mode for Port A – mode 1  
Port B – input port  
Mode for Port B – mode 1,

The remaining pins of port C<sub>Upper</sub> are to be used input pins.

**(Ans. BE H or BF H)**

15. Obtain the control word for the following configuration of the ports of 8255 A for mode 1 operation:

Port A – output port in mode 1,  
Port B – output port in mode 1,  
The remaining pins PC<sub>4</sub> and PC<sub>5</sub> of port C are to be used output pins.

**(Ans. A4 H or A5 H)**

16. Obtain the control word for the following configuration of the ports of 8255 A:

Port A – as bidirectional,  
Mode for Port A – mode 2  
Port B – input port,

Mode for Port B – mode 0,  
Port C<sub>Lower</sub> –output port.

(Ans. C2 H)

17. Write an assembly language program of 8085A to generate square wave of 100 Hz frequency using 8255A PPI. The wave should be available at PB<sub>0</sub> terminal of port B. Give the program of delay subroutine also.
  18. Generate square wave of 100 Hz using BSR mode of 8255A. The wave should be available at PC4 terminal (D<sub>4</sub> bit of port C).
  19. Write a program in 8085 assembly language to blink an LED at a regular interval of time. Assume LED is connected to PA<sub>0</sub> (D<sub>0</sub> bit of port A of 8255A PPI).
  20. Write a program in 8085 assembly language to blink an LED at a regular interval of time using Bit Set/Reset (BSR) mode. Assume LED is connected to PC<sub>0</sub> (D<sub>0</sub> bit of port C of 8255A PPI).
-

# 9

## Programmable Interval Timer/Counter: 8253

---

In many industrial applications precise time delays are needed. In discrete systems the time delays are generated using electronic timer based on logic gates and circuits or mechanical timers. However, in microprocessor based systems the time delays can be generated using software. The generation of accurate time delays by software control is very commonly used method as discussed in the earlier chapters of this book. In this method the processor has to wait for some time to generate a time delay, as microprocessor becomes unnecessarily busy in looping. The programmable interval timer/counter chip 8253 solves this problem. It is one of the most supporting chips for 8085 microprocessor. It is used for real time applications. It can generate accurate time delays and wave forms using the software control. This chip includes three identical 16-bit pre-settable down counters that can operate independently, as each counter has a clock input, gate input and one output. The counters can count in binary or BCD. In this chapter a detailed discussion on this chip and its applications will be carried out.

### 9.1 INTEL PROGRAMMABLE INTERVAL TIMER 8253

Intel 8253 Programmable Interval Timer/Counter is an NMOS 24 pin plastic dual in line package (DIP). It operates at +5 volt d.c. source. It can generate accurate time delays and wave forms using the software control. This chip includes three identical 16-bit pre-settable down counters that can operate independently; namely counter 0, counter 1 and counter 2. The counters can count in binary or BCD. Each counter has a clock input, gate input and one output. The gate input enables the counting process. The counting may be started by applying the gate signal. The 8253 works at maximum clock frequency of 2 MHz. There is another programmable interval time which is upgraded version of 8253. It has pin compatible and software compatible with 8253. The 8254 can operate with high clock frequency up to about 8 MHz. This chip 8254 is compatible to 8086, 8088 and other microprocessors. However, 8253 is compatible to 8085 microprocessor. We shall discuss the details of the chip 8253.

The 8253 can operate on the following six 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 three counters of this chip can operate independently in any of the six modes.

## 9.2 BLOCK DIAGRAM OF 8253

Figures 9.1 and 9.2 show the pin diagram and functional block diagram of 8253. It has a data bus to be connected to the data bus of the microprocessor, a control word register, five control signals and three 16-bit presettable down counters namely:

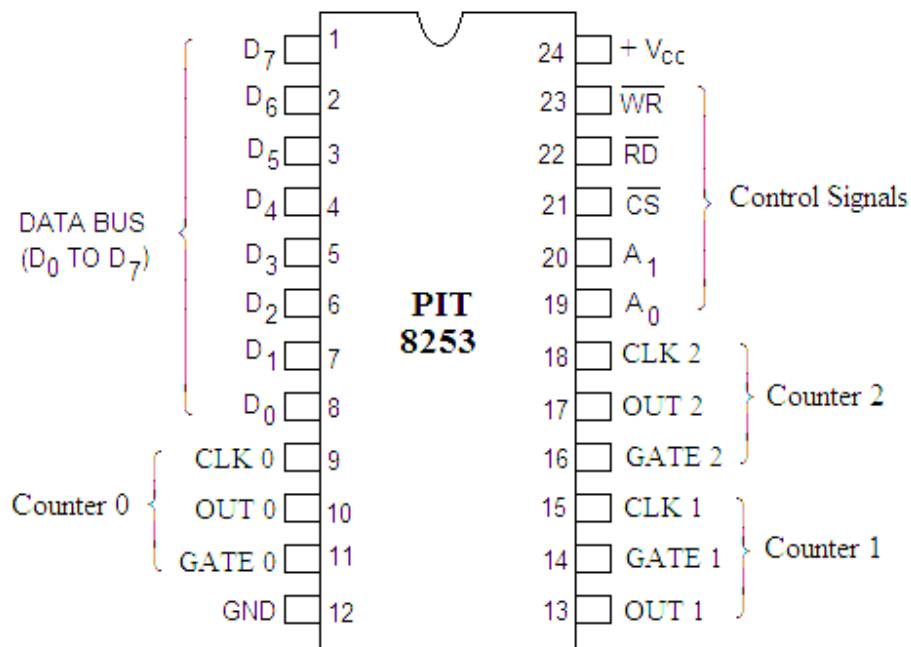
Counter 0

Counter 1

Counter 2.

### Control word Register:

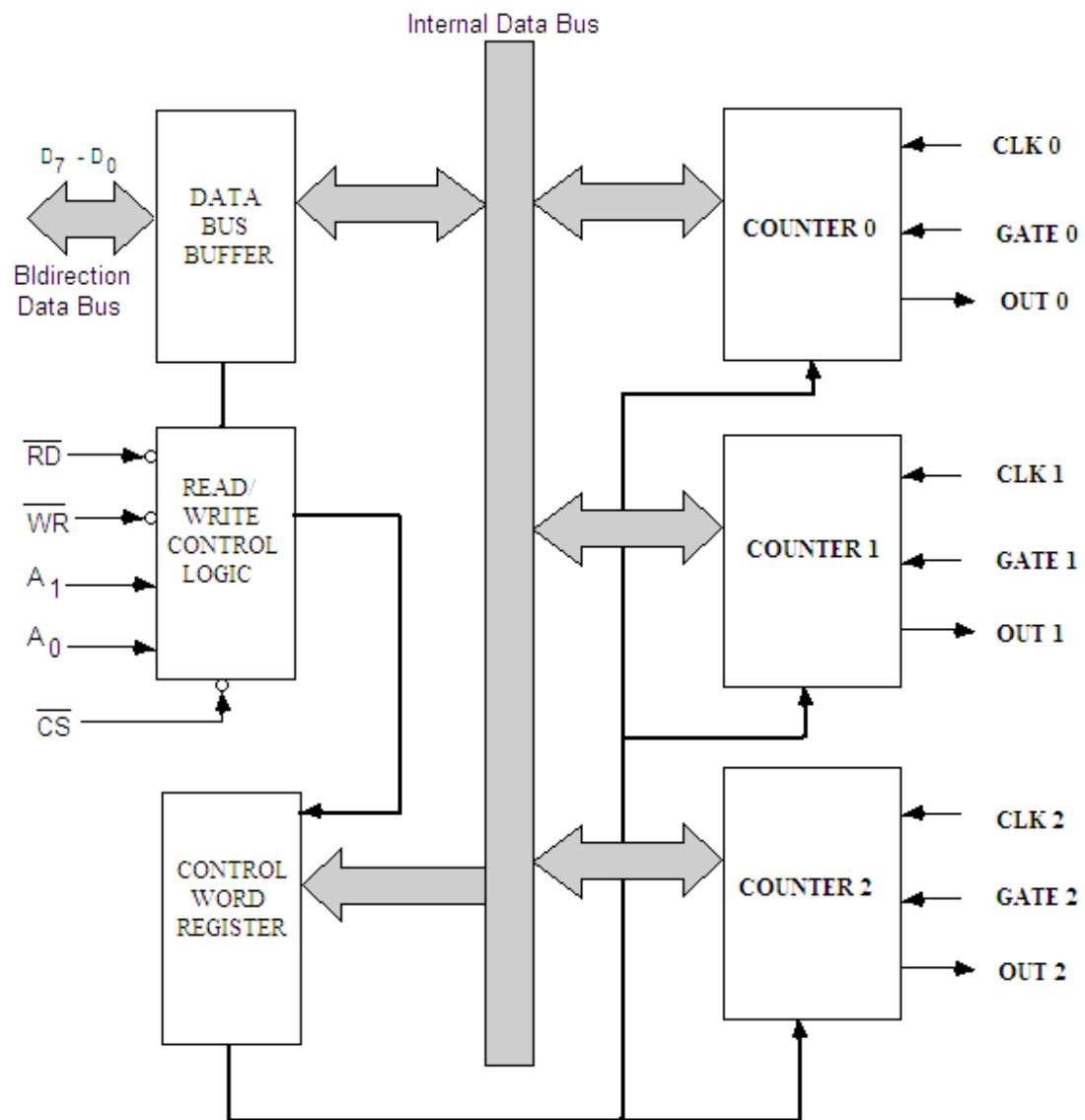
The control word register of 8253 is accessed when  $A_0$  and  $A_1$  terminals are at logic 1. It is used to write a command word which specifies the counter to be used, its mode, and either Read or Write operation,



**Fig. 9.1**

The description of pins of 8253 Programmable Interval Timer (PIT) is given as follows:

- Pin Nos. 1-8: form the bidirectional data bus buffer ( $D_7$  to  $D_0$ ) to be connected to the control bus of the microprocessor.
- Pin Nos. 9-11: are provided for counter 0. Pin 9 is CLK 0 (Clock input terminal for counter 0), Pin 10 is OUT 0 (Output terminal for counter 0) and Pin 11 is GATE 0 (Gate input terminal for counter 0).
- Pin No. 12: is the ground terminal.
- Pin Nos. 13-15 are allotted to counter 1. Pin 13 is OUT 1 (Output terminal for counter 1), Pin 14 is GATE 1 (Gate input terminal for counter 1) and pin 15 is CLK1 (Clock input for counter 1).



**Fig. 9.2**

- Pin Nos. 16-18 are for counter 2. Pin 16 is GATE 2 (Gate input terminal for counter 2), Pin 17 is OUT 2 (Output terminal for counter 2) and pin 18 is CLK 2 (Clock input for counter 2).
- Pin Nos. 19-20 are A<sub>0</sub> and A<sub>1</sub> terminals respectively. These terminals are used to select any one of the three counters and the control word register of 8253. These lines are to be connected to the address bus. The counters and control word register are selected as given in table 9.1.

**Table 9.1**

| <b>A<sub>0</sub></b> | <b>A<sub>1</sub></b> | <b>Selection for Counters</b> |
|----------------------|----------------------|-------------------------------|
| 0                    | 0                    | Counter 0 is selected.        |
| 0                    | 1                    | Counter 1 is selected.        |
| 1                    | 0                    | Counter 2 is selected.        |
| 1                    | 1                    | Counter 3 is selected.        |

- Pin No. 21 is  $\overline{CS}$  (Chip select terminal). It is an active low signal which allows enabling the device.
- Pin No. 22 is  $\overline{RD}$  (Read terminal). It is also an active low signal. A low to this pin informs the 8253; that the CPU is ready to receive the data in the form of count value.
- Pin No. 23 is  $\overline{WR}$  (Write terminal). When this pin is low, the microprocessor outputs data in the form of mode operation or loading of counter.
- Pin No. 24 is  $+V_{CC}$  (+5 V) terminal.

### 9.3 LOGICS FOR COUNTERS

As already discussed, the 8253 has five control signals  $\overline{CS}$ ,  $\overline{RD}$ ,  $\overline{WR}$ ,  $A_1$  and  $A_0$ . The table 9.2 shows the action of different combination of these control signals.

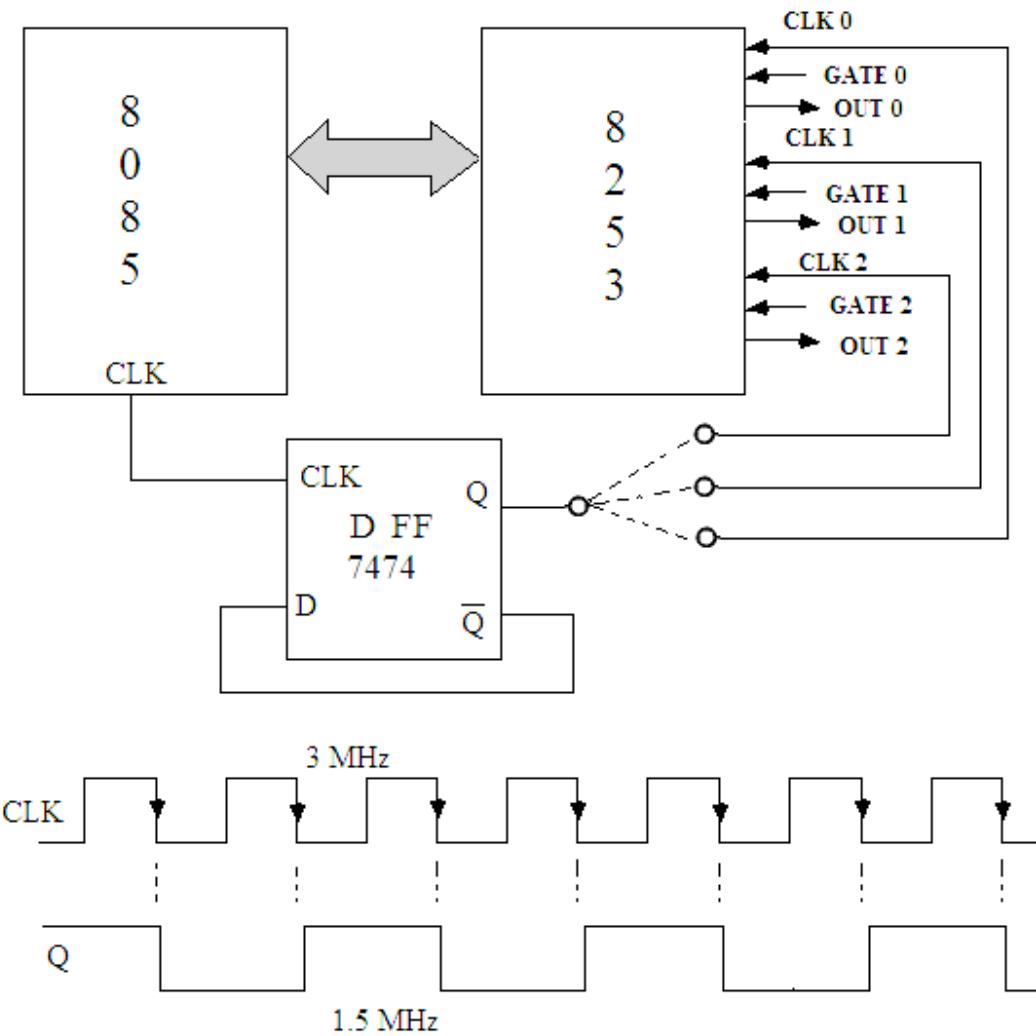
**Table 9.2**

| $\overline{CS}$ | $\overline{RD}$ | $\overline{WR}$ | $A_1$ | $A_0$ | Actions              |
|-----------------|-----------------|-----------------|-------|-------|----------------------|
| 0               | 1               | 0               | 0     | 0     | Load counter 0       |
| 0               | 1               | 0               | 0     | 1     | Load counter 1       |
| 0               | 1               | 0               | 1     | 0     | Load counter 2       |
| 0               | 1               | 0               | 1     | 1     | Write Mode Word      |
| 0               | 0               | 1               | 0     | 0     | Read counter 0       |
| 0               | 0               | 1               | 0     | 1     | Read counter 1       |
| 0               | 0               | 1               | 1     | 0     | Read counter 2       |
| 0               | 0               | 1               | 1     | 1     | No operation 3 state |
| 0               | 1               | 1               | X     | X     | No operation 3-state |
| 1               | X               | X               | X     | X     | Disable 3-state      |

Note: X indicates undefined state, i.e. it does not matter whether it is 0 or 1.

From this table it is clear that when  $\overline{CS} = \overline{WR} = 0$  and  $\overline{RD} = 1$  with different combination of  $A_1$  and  $A_0$ ; any counter can be selected and loaded with 16 counter values. Once the gate is enabled by applying a high signal (logic 1) to it, counter values starts decrementing at each of the trailing edge of the clock pulse. However, if  $\overline{CS} = \overline{RD} = 0$  and  $\overline{WR} = 1$ , the counter read operation is performed by the different counters.

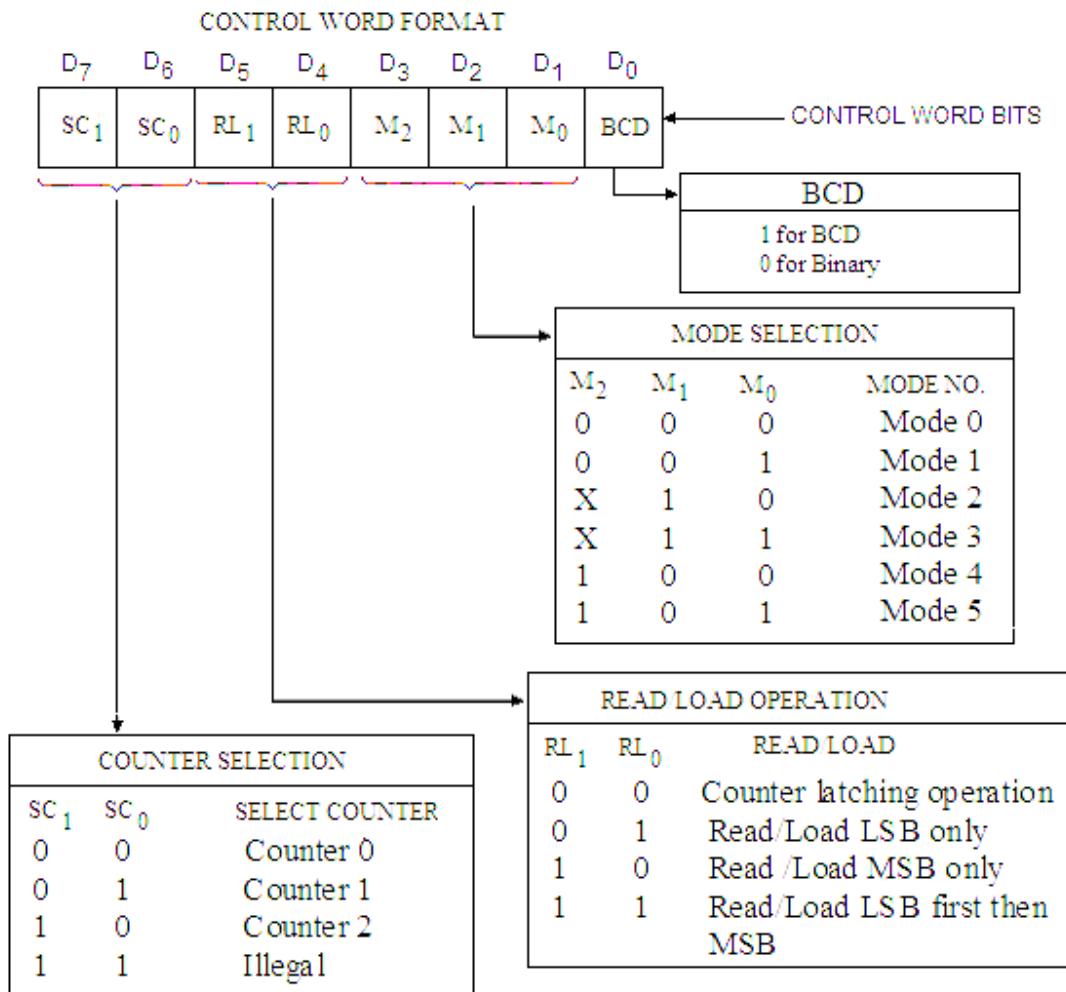
The clock pulse is to be applied to CLK terminal of each counter of 8253. Though the clock signal of 3 MHz frequency is available with 8085 microprocessor but it can not directly be applied to the CLK terminal of each counter. Since 8253 can work with a maximum of 2 MHz clock frequency. So the clock frequency available with 8085 microprocessor is divided by a factor of 2 using an edge triggered D-flip flop (7474) as shown in figure 9.3. So 1.5 MHz clock frequency available at the output of D-flip flop can be used as CLK input of the counters of 8253. In some microprocessor kits available in the market for use in laboratories, the CLK terminals of the counters are left open so that any clock signal of desired frequency may be connected to these terminals.



**Fig. 9.3**

## **9.4 CONTROL WORD FORMAT OF 8253**

As already discussed, any of the three counters can be programmed to operate in any of the 6 modes. The control word decides the selection of counter, its mode of operation, loading, sequence of the count and selection of binary or BCD counting. The format for the control word of 8253 is shown in figure 9.4. The control word is written into the control word register of 8253, as per the requirement.



**Fig. 9.4**

The description of the bits of control word format is given as:

**Bit D<sub>0</sub>** Sets the counting of the counter in binary or BCD.

To make the counting in BCD this bit is set to 1, and to make the counting in binary this bit is set to 0.

**Bits D<sub>1</sub>-D<sub>3</sub>** These three bits are called mode selection bits M<sub>0</sub>, M<sub>1</sub> and M<sub>2</sub> respectively; and are used for mode selection of the counters which are given in table 9.3.

**Table 9.3**

| D <sub>3</sub><br>(M <sub>2</sub> ) | D <sub>2</sub><br>(M <sub>1</sub> ) | D <sub>1</sub><br>(M <sub>0</sub> ) | Mode   |
|-------------------------------------|-------------------------------------|-------------------------------------|--------|
| 0                                   | 0                                   | 0                                   | Mode 0 |
| 0                                   | 0                                   | 1                                   | Mode 1 |
| X                                   | 1                                   | 0                                   | Mode 2 |
| X                                   | 1                                   | 1                                   | Mode 3 |
| 1                                   | 0                                   | 0                                   | Mode 4 |
| 1                                   | 0                                   | 1                                   | Mode 5 |

**Bits D<sub>4</sub>-D<sub>5</sub>**

These bits are known as Read/Load bits (RL<sub>0</sub> and RL<sub>1</sub>) and are used for read/load of the count values in the counters. Their operations are given in table 9.4.

**Table 9.4**

| D <sub>5</sub><br>(RL <sub>1</sub> ) | D <sub>4</sub><br>(RL <sub>0</sub> ) | Mode                            |
|--------------------------------------|--------------------------------------|---------------------------------|
| 0                                    | 0                                    | Counter latching operation      |
| 0                                    | 1                                    | Read/Load LSB only              |
| 1                                    | 0                                    | Read /Load MSB only             |
| 1                                    | 1                                    | Read/Load LSB first then<br>MSB |

**Bits D<sub>6</sub>-D<sub>7</sub>**

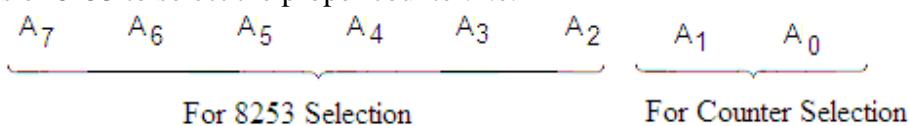
These bits (SC<sub>0</sub> and SC<sub>1</sub>) are used for counter selection as given in table 9.5.

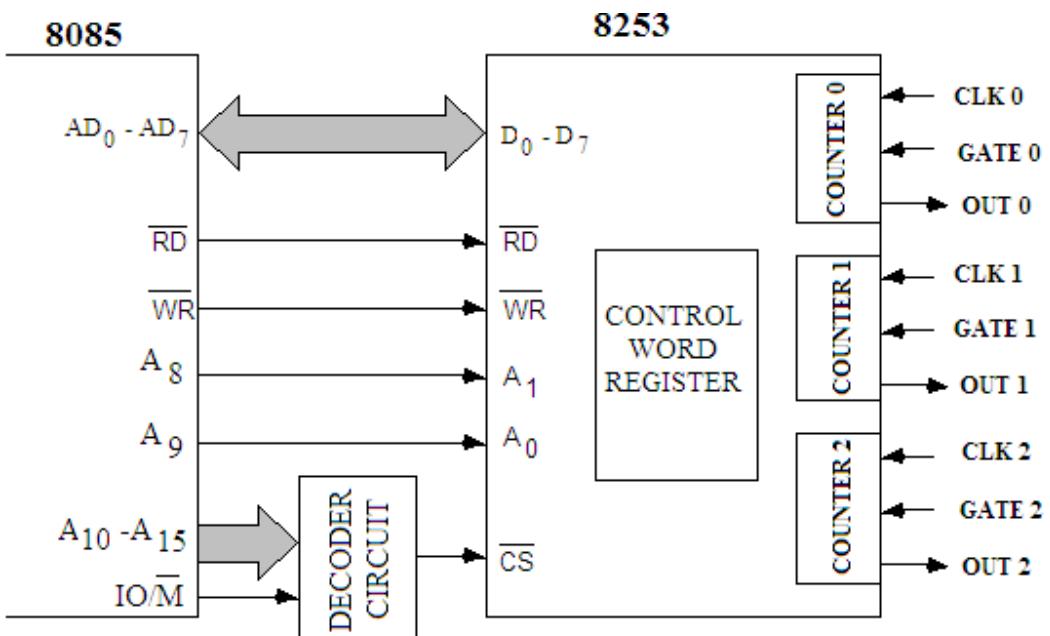
**Table 9.5**

| D <sub>7</sub><br>(SC <sub>1</sub> ) | D <sub>6</sub><br>(SC <sub>0</sub> ) | Select counter |
|--------------------------------------|--------------------------------------|----------------|
| 0                                    | 0                                    | Counter 0      |
| 0                                    | 1                                    | Counter 1      |
| 1                                    | 0                                    | Counter 2      |
| 1                                    | 1                                    | Illegal        |

**9.5 INTERFACING AND PROGRAMMING OF 8253**

Figure 9.5 shows interfacing of 8253 with the 8085 system bus. The eight data lines D<sub>0</sub> to D<sub>7</sub> of 8253 are connected to the data bus (Address data bus AD<sub>0</sub>-AD<sub>7</sub>) of the microprocessor. The chip is selected with  $\overline{CS}$  terminal. So higher order address bus of the system is used for decoder circuit to select the chip. The A<sub>8</sub> and A<sub>9</sub> (two LSBs of the address bus) terminals of the microprocessor address bus are connected to A<sub>0</sub> and A<sub>1</sub> terminals of 8253 to select the proper counter. i.e.



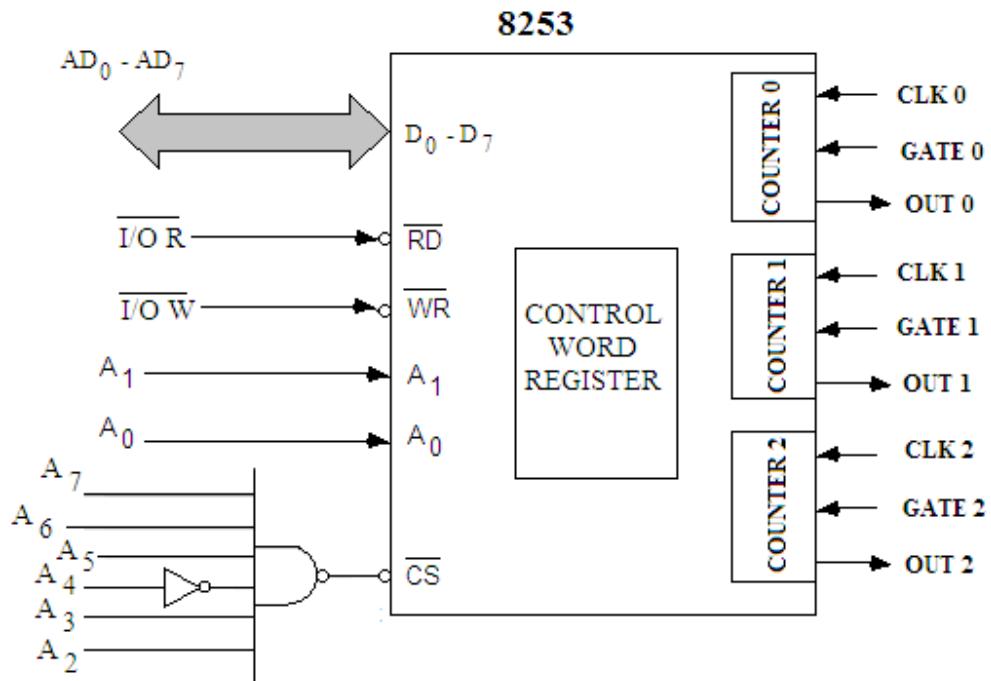


**Fig. 9.5**

Figure 9.6 shows the chip-select logic for 8253. From this logic diagram it is clear that if  $A_2 - A_3$ ,  $A_5 - A_7$  are all at logic 0 and  $A_3$  is at logic 1, then it enables  $\overline{CS}$  to select this chip. The counter selection will depend on the bits  $A_1 - A_0$  as shown in table 9.6.

**Table 9.6**

| $A_7$ | $A_6$ | $A_5$ | $A_4$ | $A_3$ | $A_2$ | $A_1$ | $A_0$ | HEX ADDRESS | Counter Selection     |
|-------|-------|-------|-------|-------|-------|-------|-------|-------------|-----------------------|
| 0     | 0     | 0     | 1     | 0     | 0     | 0     | 0     | 10 H        | Counter 0             |
| 0     | 0     | 0     | 1     | 0     | 0     | 0     | 1     | 11 H        | Counter 1             |
| 0     | 0     | 0     | 1     | 0     | 0     | 1     | 0     | 12 H        | Counter 2             |
| 0     | 0     | 0     | 1     | 0     | 0     | 1     | 1     | 13 H        | Control word Register |



**Fig. 9.6**

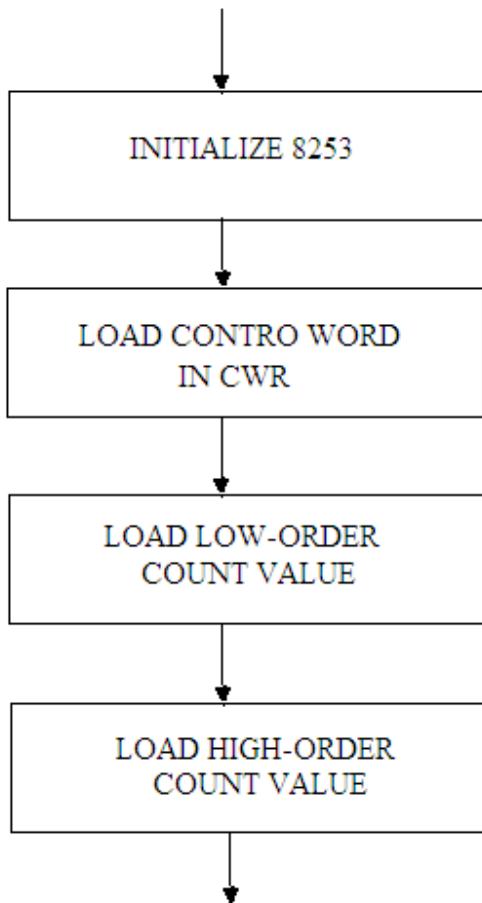
As usual with IN and OUT instructions, the 8085 duplicates the port address on the address bus and data bus. The OUT instruction loads the control word in the control word register. As discussed above we may choose the hexadecimal address as

|      |                                  |
|------|----------------------------------|
| 10 H | for counter 0                    |
| 11 H | for counter 1                    |
| 12 H | for counter 2                    |
| 13 H | for control word register (CWR). |

The OUT 13 H will load the accumulator content to the control word register. The accumulator is therefore, to be loaded to the proper control word (as decided by the control word format) before OUT 13 H instruction. This way 8253 is initialized. To load the count values to the counter number initialized by the OUT 13 instruction, again OUT instruction is used having the proper control word.

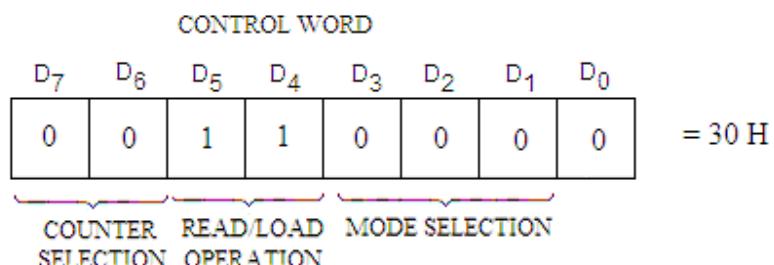
The IN instruction with proper port address will read the count values of the counter without stopping the counting. The 8253 can thus be programmed through write operations to generate various types of wave forms (which will be discussed in the succeeding section of this chapter) by loading the proper count values to any of the counters used in different modes.

So for programming the 8253 we proceed as given in the flow chart (figure 9.7).



**Fig. 9.7**

For example to load counter 0 in mode 0 for the count value 1639 H in binary, we follow the program given below. The control word, to initialize the 8253 to load the given count value, first LS byte and then MS byte to the counter 0 in mode 0, is shown in figure 9.8.



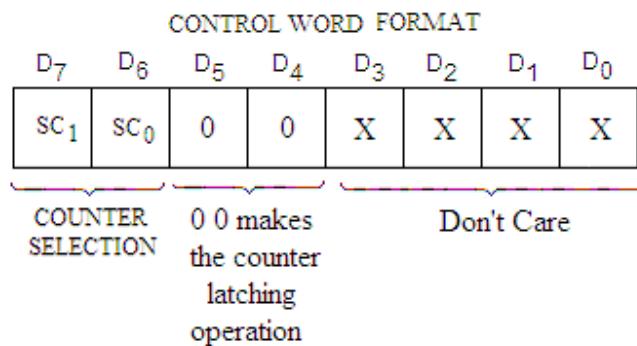
**Fig. 9.8**

**Program:**

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
|       | MVI A,    | 30 H    | ; Accumulator is loaded with the control word 30 H to Initialize 8253. |

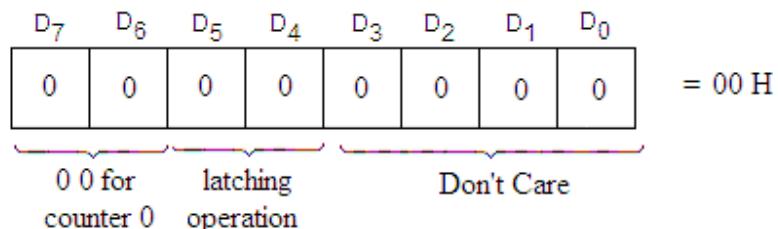
|        |      |  |
|--------|------|--|
| OUT    | 13 H | ; Write the control word in the control word register of 8253.         |
| MVI A, | 39 H | ; Load accumulator with LS byte (39) of the count value.               |
| OUT    | 10 H | ; First LS byte of the count value is loaded to the counter 0 of 8253. |
| MVI A, | 16 H | ; Load accumulator with MS byte (16) of the count value.               |
| OUT    | 10 H | ; MS byte of the count value is then loaded to the counter 0 of 8253.  |
| .....  |      |  |
| .....  |      |  |
| .....  |      |  |

The count value of any counter of 8253 can also be read without stopping the counting. The bit pattern for the control word (figure 9.9) for latching operation is as follows:



**Fig. 9.9**

The control word for latching operation for counter 0 is 00 H as shown in figure 9.10.



**Fig. 9.10**

So the following program can be used to read the count value of counter 0 while counting is in program.

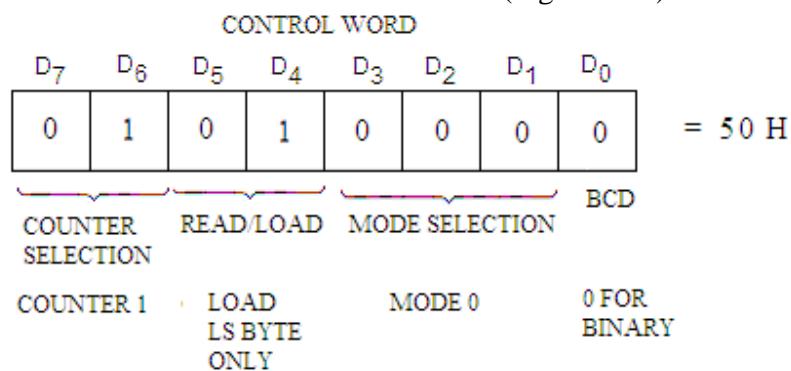
#### Program:

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
|       | MVI A,    | 00 H    | ; Control word 00 H to read the count value of the counter 0 is loaded to accumulator. |
|       | OUT       | 13 H    | ; The control word is loaded in the control word register of 8253.                     |

|        |      |   |
|--------|------|---|
| IN     | 10 H | ; First read the LS byte of the count value.        |
| MOV B, | A    | ; Load this value to B-register.                    |
| IN     | 10 H | ; Read the MS byte of the count value.              |
| MOV C, | A    | ; Load this value in C-register.                    |
| MVI A, | 16 H | ; Load accumulator with MS byte of the count value. |
| .....  |      |   |
| .....  |      |   |

**Example 9.1.** Initialize 8253 to load counter 1 in mode 0 with 8bit binary number 06 H.

**Solution.** To initialize 8253 we have the control word as (Figure 9.11):



**Fig. 9.11**

The program for the same is written as:

**Program:**

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
|       | MVI A,    | 50 H    | ; Accumulator is loaded with the control word 50 H to Initialize 8253. |
|       | OUT       | 13 H    | ; Write the control word in the control word register of 8253.         |
|       | MVI A,    | 06 H    | ; Load accumulator with LS byte (06 H) only of the count value.        |
|       | OUT       | 11 H    | ; 06 H of the count value is loaded to the counter 1 of 8253.          |
|       | HLT       |         | ; Stop processing.   |

**Example 9.2.** Initialize 8253 to load counter 0 in mode 1 with 1632 H BCD number.

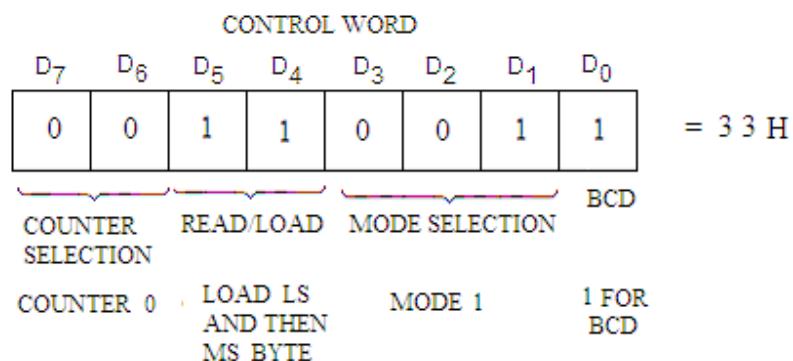
**Solution.** To initialize 8253 we have the control word as (Figure 9.12):

The program for the same is written as:

**Program:**

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
|       | MVI A,    | 33 H    | ; Accumulator is loaded with the control word 33 H to Initialize 8253. |
|       | OUT       | 13 H    | ; Write the control word in the control word register of 8253.         |

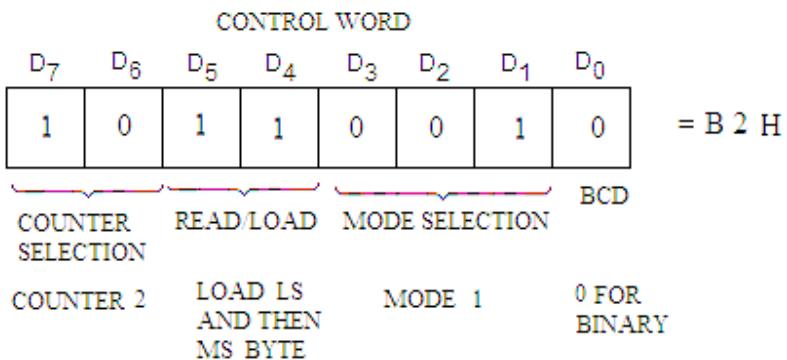
|        |      |   |
|--------|------|---|
| MVI A, | 32 H | ; Load accumulator first with LS byte (32 H) of the count value 1632 H. |
| OUT    | 10 H | ; 32 H of the count value is loaded to the counter 0 of 8253.           |
| MVI A, | 16 H | ; Load accumulator then with MS byte (16 H) of the count value.         |
| OUT    | 10 H | ; 16 H of the count value is loaded to the counter 0 of 8253.           |
| HLT    |      | ; Stop processing.  |



**Fig. 9.12**

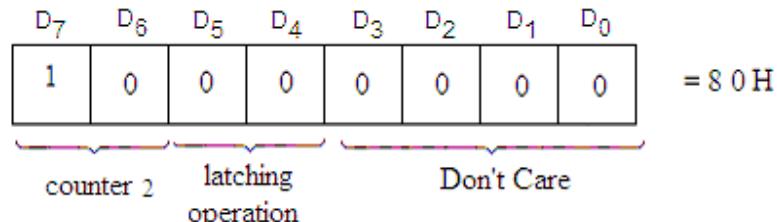
**Example 9.3.** Initialize 8253 to load counter 2 in mode 1 with a count value  $5000_{10}$  in mode 0. Read the count value on the fly.

**Solution.** To initialize 8253 we have the control word as (Figure 9.13):



**Fig. 9.13**

The control word for latching operation for counter 2 is 80 H as shown in figure 9.14.



**Fig. 9.14**

The program for the same is written as:

The hexadecimal equivalent of  $5000_{10}$  is  $1388\text{ H}$ .

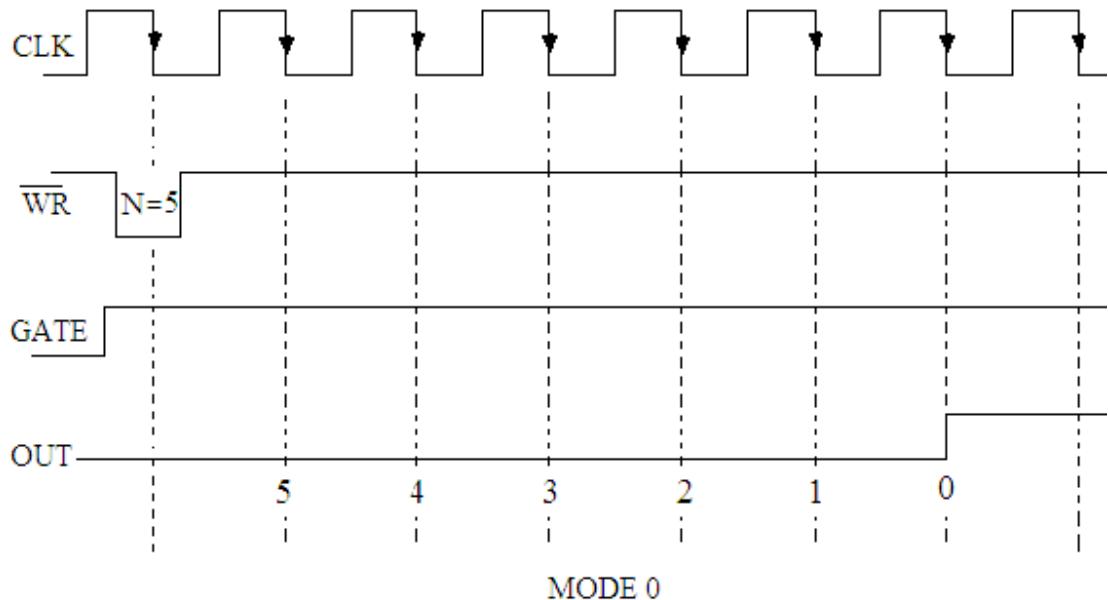
**Program:**

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>  |
|--------------|------------------|----------------|--|
| LOOP         | MVI A,           | B2 H           | ; Accumulator is loaded with the control word B2 H to Initialize 8253.                 |
|              | OUT              | 13 H           | ; Write the control word in the control word register of 8253.                         |
|              | MVI A,           | 88 H           | ; Load accumulator first with LS byte (88 H) of the count value 1632 H.                |
|              | OUT              | 12 H           | ; 88 H of the count value is loaded to the counter 2 of 8253.                          |
|              | MVI A,           | 13 H           | ; Load accumulator then with MS byte (13 H) of the count value.                        |
|              | OUT              | 12 H           | ; 16 H of the count value is loaded to the counter 2 of 8253.                          |
|              | MVI D,           | 00 H           | ; Store 00 H to D-register.  |
|              | MVI A,           | 80 H           | ; Control word 80 H to read the count value of the counter 2 is loaded to accumulator. |
|              | OUT              | 13 H           | ; The control word is loaded in the control word register of 8253.                     |
|              | IN               | 12 H           | ; First read the LS byte of the count value.   |
|              | MOV B,           | A              | ; Load this value to B-register.   |
|              | IN               | 12 H           | ; Read the MS byte of the count value.   |
|              | ORA B            |                | ; Logical OR the LS and MS byte to set zero flag.                                      |
|              | JNZ              | LOOP           | ; If not zero jump to LOOP.  |
|              | HLT              |                | ; Stop processing.   |

## 9.6 PROGRAMMING OF 8253 IN MODE 0: INTERRUPT ON TERMINAL COUNT

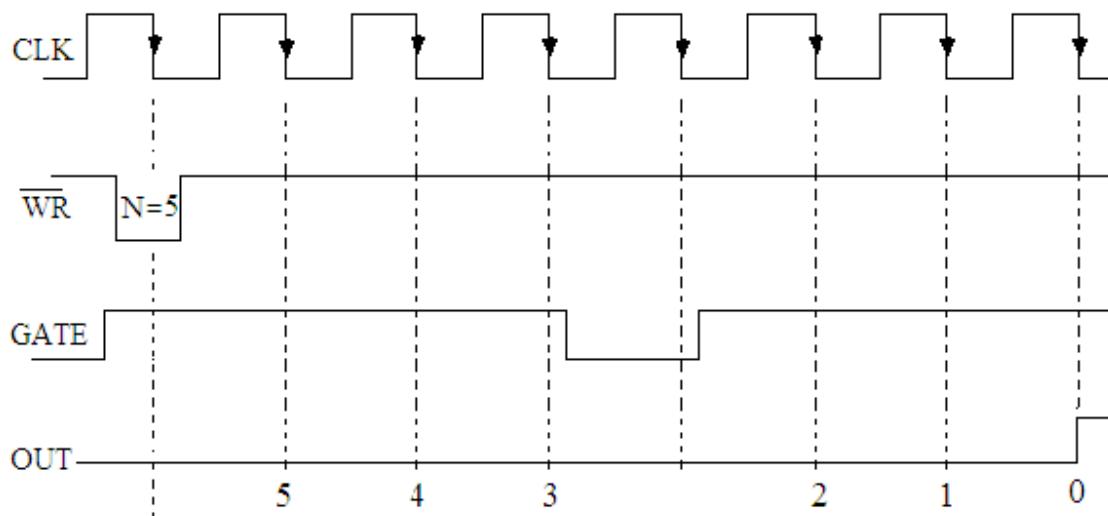
The MODE 0 is interrupt on terminal count. The output of the desired counter goes low on setting the mode, and goes high after the loaded counts are complete. It is used for the generation of precise time delay under software control. After the selection of desired counter, a suitable count value is loaded in the counter. The GATE terminal of the selected counter is made high from low. The down counting of the counter will be started. At the beginning of the counter, OUT terminal becomes low and remains low till the counting is stopped at the zero count value. On reaching the terminal count (i.e. count value becomes zero), the OUT terminal becomes high (logic 1) and remains high until and unless the selected counter is reloaded or a new count value is loaded or the mode of operation is changed. Further, if the count value is reloaded in count register while the counting is in progress, then after LS byte of the count value is written, the current counting is stopped; and after MS byte of the count value is written, the counting restarts

with the new count value. The timing diagram for mode 0 operation is shown in figure 9.15.



**Fig. 9.15**

If during the counting operation, GATE terminal is made low from high, the counting will be suspended. If the GATE terminal is again made high the counting may be resumed at the trailing edge of the clock pulse from the value where the counting was suspended. This is depicted in figure 9.16. The OUT terminal can be used to interrupt the microprocessor.

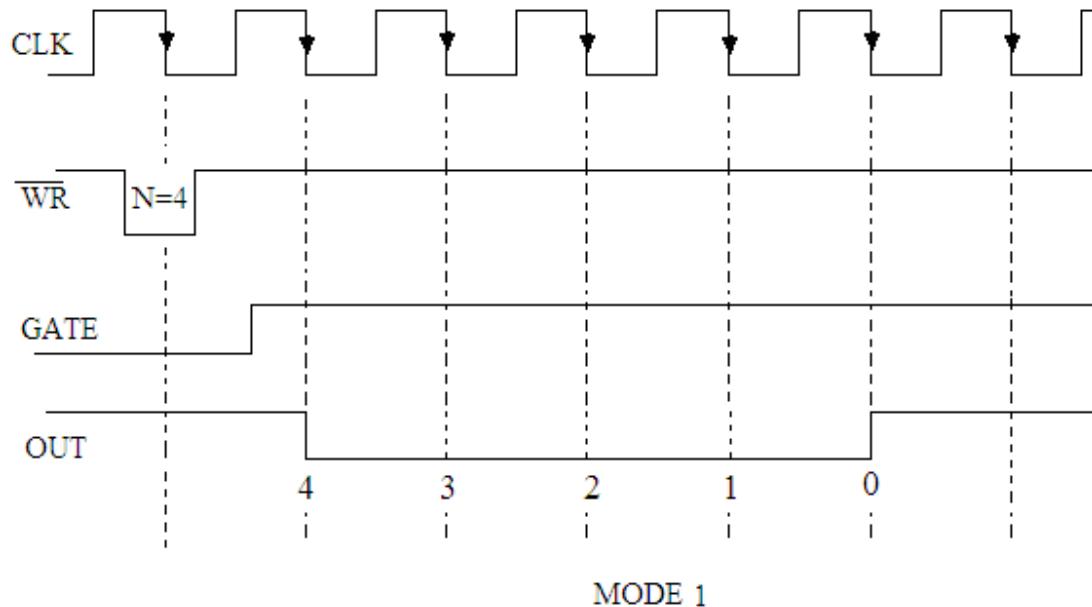


**Fig. 9.16**

#### 9.7 PROGRAMMING OF 8253 IN MODE 1: PROGRAMMABLE ONE SHOT

The MODE 1 is Programmable or retriggerable one shot. In this mode the OUT terminal of the counter goes low on the triggering of corresponding GATE input (low to high transition); which (OUT terminal) goes high on terminal count. The OUT terminal remains low for the count value loaded to the counter.

Initially control word and count values are loaded to the device register. A leading edge trigger signal is applied to the GATE input of the selected counter. At the next trailing edge of the clock the OUT terminal goes low. The down counting of the counter will now start and at the terminal count (i.e. count value becomes zero) the OUT terminal becomes high. In other word we can say that the OUT terminal remains low for the number of counts loaded in the selected counter. In this mode the counter thus be used as a triggered mono (or one) shot of desired width. This width can be varied by the count value loaded for the selected counter. It is illustrated in figure 9.17. The changing signal from high to low to GATE terminal does not affect the signal of OUT terminal, once the counting is started.



MODE 1

Fig. 9.17

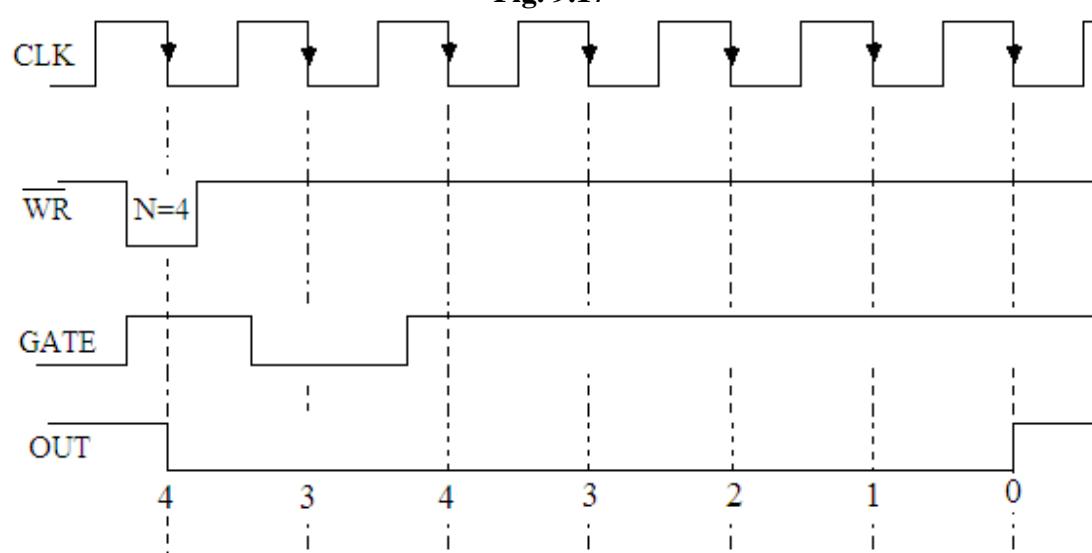


Fig. 9.18

If the signal at the GATE terminal is changed from low to high when the counting is in progress, then the counting will be stopped for exactly one clock pulse. After one

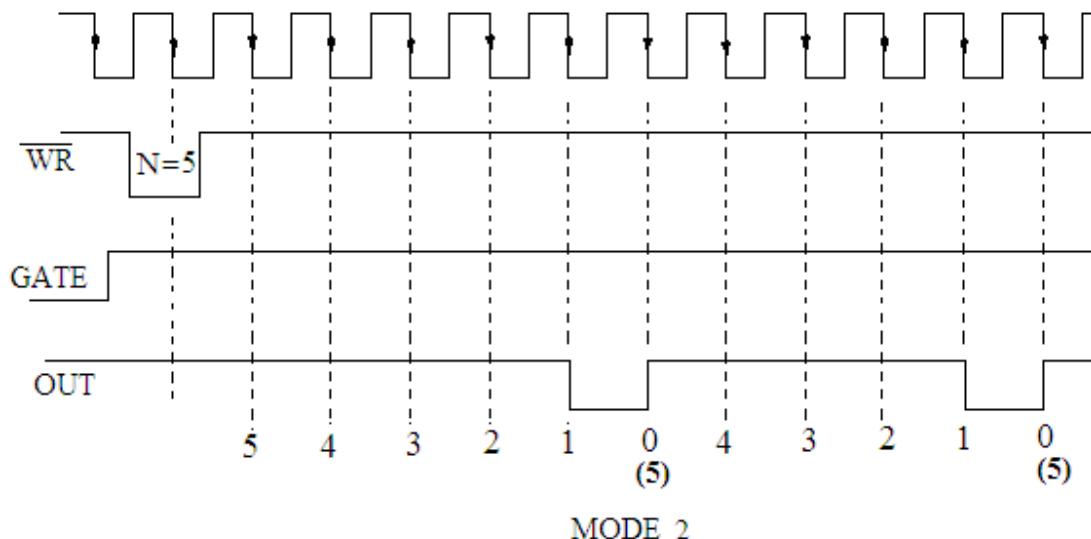
clock pulse, the recounting will be started from the beginning i.e. the counter will be reloaded with the same count value and the down counting will be started again and the OUT terminal goes low for the same count value. Thus it is also called retriggerable one shot. This situation is shown in figure 9.18.

A new count may be loaded to the counter while OUT terminal is low; it will not affect the width of the low OUT signal until the next trigger is applied to the GATE.

### 9.8 PROGRAMMING OF 8253 IN MODE 2: RATE GENERATOR

Any counter of 8253 may be programmed to operate in mode 2 as rate generator which is also called as divide by N counter.

Initially control word for mode 2 and count value N are loaded into the selected counter. The down counting of the counter will be started as soon as a high signal to the GATE terminal of the counter is applied. The OUT terminal stays high for (N-1) clock pulses and becomes low for exactly one clock pulse. After this OUT signal goes high again for (N-1) clock pulses and low for one pulse as the count value N is reloaded into the selected counter. This process continues as long as the GATE terminal is high. It, therefore, works as divide by N counter. Further, if the counter is working in this mode and a new count value is loaded, the present period will not be affected but the subsequent period will be as per the new count value. The timing diagram is shown in figure 9.19.



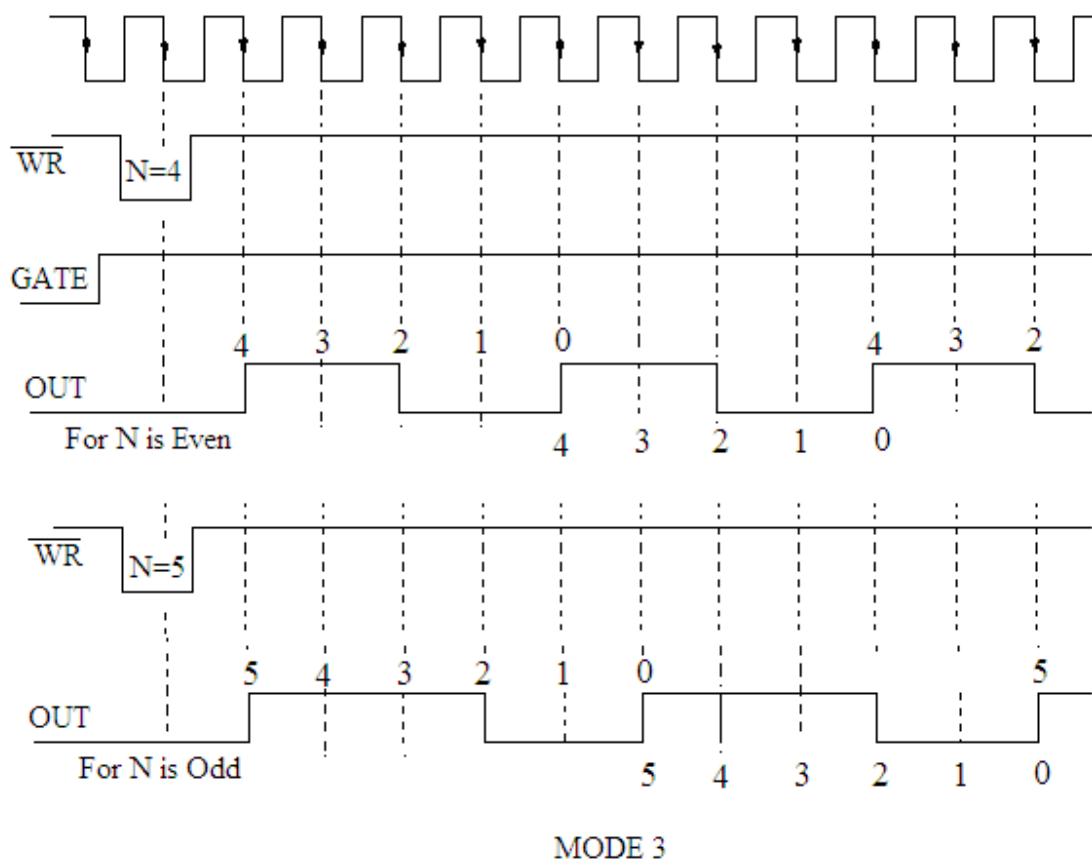
**Fig. 9.19**

The GATE input when low, will force the OUT terminal to high. When the GATE input goes high, the counter will start from the initial count. Thus the GATE can be used to synchronize the counter.

### 9.9 PROGRAMMING OF 8253 IN MODE 3: SQUARE WAVE GENERATOR

When 8253 operate in mode 3, the counter acts as a square wave generator. This is similar to mode 2 with the difference that the OUT terminal of the selected counter in mode 3 remains high for half the count value and is low for the other half, if the count value is an even number. However, if the count value is an odd value, OUT terminal remains high for  $\left(\frac{N+1}{2}\right)$  clock pulses and is low for  $\left(\frac{N-1}{2}\right)$  clock pulses.

When control word for mode 3 and count value N are loaded into the selected counter, a high signal at the GATE terminal starts the down counting. The OUT terminal remains high for  $\left(\frac{N}{2}\right)$  counts and is low for the next  $\left(\frac{N}{2}\right)$  counts, if the count value N is even; and for odd value of N, the OUT signal is high for  $\left(\frac{N+1}{2}\right)$  pulses and is low for  $\left(\frac{N-1}{2}\right)$  pulses as mentioned above. The count value is reloaded with full count and the process is repeated till the GATE is high. Figure 9.20 shows the timing diagram for this mode of operation for both even and odd count value.

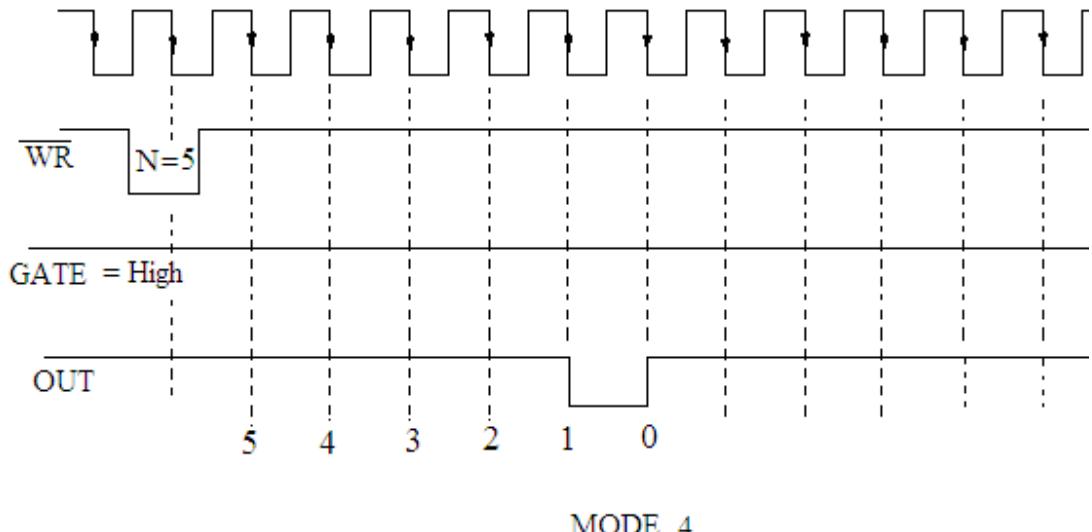


The mode of operation of 8253 generates a square wave of period specified by the count value loaded to the counter. Any change in count value becomes effective, after the present counts are over.

#### **9.10 PROGRAMMING OF 8253 IN MODE 4: SOFTWARE TRIGGERED STROBE**

In this mode of operation software controlled delayed negative pulse of one clock period duration is generated with and without synchronization. After the mode of operation of 8253 is set, the OUT terminal will be high, and the down counting of the counter will start as soon as the count value is loaded into the counter. On terminal count,

the OUT terminal goes low for one clock pulse then will go high again as shown in figure 9.21.

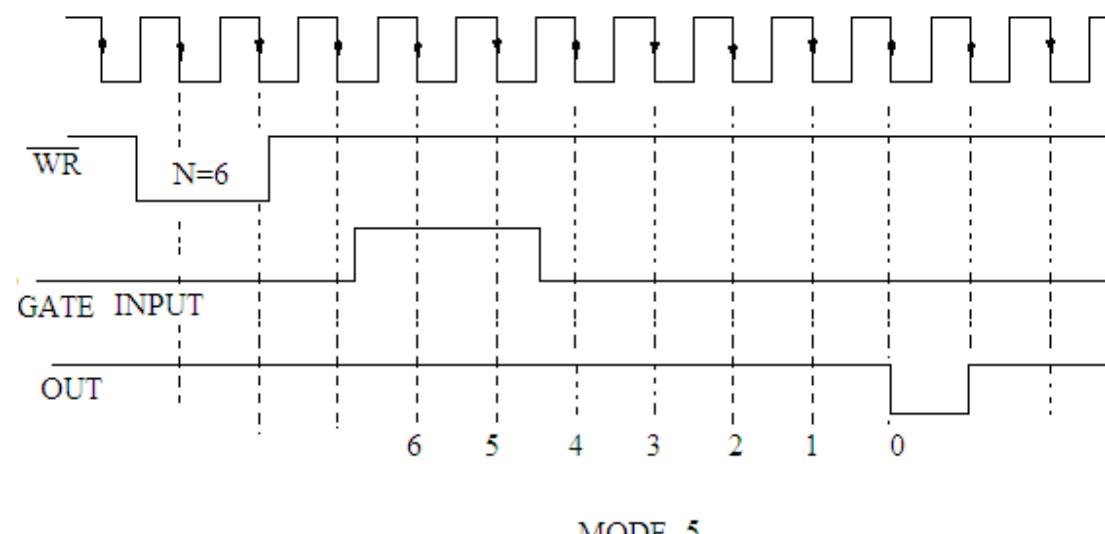


**Fig.9.21**

If the count register is reloaded between the output pulses, the present period will not be affected, but the subsequent period will reflect the new count value. The counter will be inhibited when the GATE terminal goes low. The GATE input can, therefore, be used for synchronization. The generation of strobe signal at OUT terminal is triggered by loading the count value in the counter that is why this mode is called software triggered mode.

#### 9.11 PROGRAMMING OF 8253 IN MODE 5: HARDWARE TRIGGERED STROBE

In this mode of operation a delayed negative pulse of one clock period duration is generated if a positive going trigger input is applied at the GATE terminal. After the mode of operation of 8253 is set, and the count value is loaded into the counter OUT



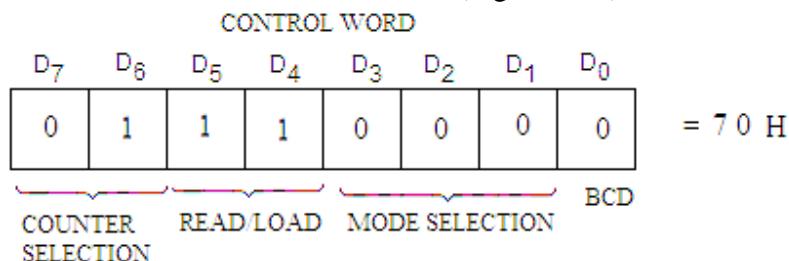
**Fig. 9.22**

terminal will be initially high. As the rising edge signal is applied to GATE terminal, the down counting of the counter will be started at the first trailing edge of the clock pulse after the rising edge of the GATE input. On terminal count, the OUT terminal goes low for one clock pulse then will go high again. The timing diagram for this mode of operation is shown in figure 9.22. As the low to high transition of the GATE terminal causes triggering hence this mode is referred to as hardware triggered strobe.

If the GATE input is made low to high again the counter is reloaded by the same count value. The down counting of the count value once again started and on the terminal count the OUT terminal goes low for one clock period.

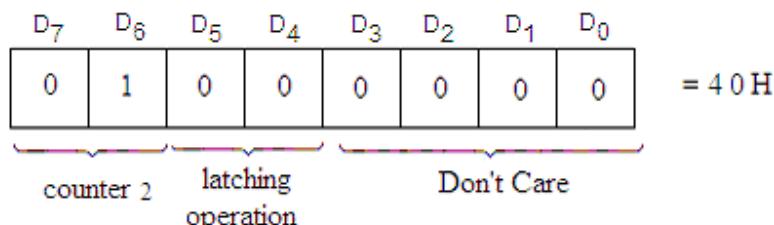
**Example 9.4.** An LED is connected to PA<sub>0</sub> of 8255 PPI. It should show ON and OFF regularly with a delay of one second. The delay should be introduced with 8253 used in mode 0 of counter 1. Assume the clock frequency applied to the counter of 8253 is 2 MHz.

**Solution.** The control word to initialize 8253 is (Figure 9.23):



**Fig. 9.23**

The control word for latching operation for counter 2 is 80 H as shown in figure 9.24.



**Fig. 9.24**

Since the frequency of the clock signal connected to CLK terminal is 2 MHz, so the time of one clock pulse is 0.5  $\mu$  sec. If a count value of 50000<sub>10</sub> (C350 H) is loaded to the counter, then a delay of 25 msec may be introduced when the counter is used in mode 0. Further to introduce a total time delay of 1 sec, then the subroutine program used for 25 msec may be executed 40 (28 H) times. Thus the program having the subroutine is given below:

#### Main Program:

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
|       | LXI SP,   | XXXX H  | ; Initialize stack pointer.  |
|       | MVI A,    | 80 H    | ; Initialize 8255 PPI with all the ports as output ports.          |
|       | OUT       | 03 H    | ; The control word is loaded to the control word register of 8255. |
|       | MVI B,    | 00 H    | ; Load the register B with 00 H.                                   |

|       |        |       |  |
|-------|--------|-------|--|
|       | MOV A, | B     | ; 00 H is stored in accumulator.   |
| LOOP1 | OUT    | 00 H  | ; 00 is send to port A of 8255 so that LED is OFF.                                 |
|       | MVI C, | 28 H  | ; C register is used as counter so that the delay subroutine is executed 40 times. |
| LOOP  | CALL   | DELAY | ; Delay program for the delay of 25 msec is called.                                |
|       | DCR C  |       | ; Decrement C.   |
|       | JNZ    | LOOP  | ; If the content in C becomes zero, jump to LOOP.                                  |
|       | MOV A, | B     | ; Move the content of B in accumulator.  |
|       | CMA    |       | ; Complement the accumulator contents.   |
|       | MOV B, | A     | ; Complemented content of accumulator is stored in B register.                     |
|       | JMP    | LOOP1 | ; Jump to LOOP1 to change the state of LED.  |

**Subroutine Program:**

|       |        |      |   |
|-------|--------|------|---|
| DELAY | MVI A, | 70 H | ; Accumulator is loaded with the control word 70 H to Initialize counter 1 of 8253.           |
|       | OUT    | 13 H | ; Write the control word in the control word register of 8253.                                |
|       | MVI A, | 50 H | ; Load accumulator first with LS byte (50 H) of the count value C350 H.                       |
|       | OUT    | 11 H | ; 50 H of the count value is loaded to the counter 1 of 8253.                                 |
|       | MVI A, | C3 H | ; Load accumulator then with MS byte (C3 H) of the count value.                               |
|       | OUT    | 11 H | ; C3 H of the count value is loaded to the counter 1 of 8253.                                 |
| AGAIN | MVI A, | 40 H | ; Control word 40 H is loaded to accumulator to read on fly the count value of the counter 1. |
|       | OUT    | 13 H | ; The control word is loaded in the control word register of 8253.                            |
|       | IN     | 11 H | ; First read the LS byte of the count value.  |
|       | MOV D, | A    | ; Load this value to D-register.  |
|       | IN     | 11 H | ; Read the MS byte of the count value.  |
|       | ORA D  |      | ; Logical OR the LS and MS byte to set zero flag.   |

|     |       |                              |
|-----|-------|------------------------------|
| JNZ | AGAIN | ; If not zero jump to AGAIN. |
|     | RET   | ; Return to main program.    |

**Example 9.5.** The OUT terminal of counter 0 of 8253 is connected to RST 7.5, which is used to interrupt the microprocessor to clear the memory locations 2100 H to 21FF H. Use counter 0 of 8253 in mode 0 to enable RST 7.5 after a delay of 32.5 msec. Assume the clock frequency applied to the counter 0 of 8253 is 2 MHz.

**Solution.** The control word to initialize counter 0 of 8253 is:

|                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |        |
| 0              | 0              | 1              | 1              | 0              | 0              | 0              | 0              | = 30 H |

The accumulator contents for SIM to enable RST 7.5, RST 6.5 and RST 5.5 are given by:

|                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |        |
| SOD            | SOE            | X              | R7.5           | MSE            | M7.5           | M6.5           | M5.5           |        |
| 0              | 0              | 0              | 0              | 1              | 0              | 0              | 0              | = 08 H |

The clock frequency used for counter 0 is 2 MHz, so time for one clock pulse is  $0.5 \mu\text{sec}$ . To introduce a delay of 32.5 msec, count value of 65000 ( $65000 \times 0.5 \mu\text{sec} = 32.5 \text{ msec}$ ) is to be loaded to the counter 0. The hexadecimal equivalent of  $65000_{10}$  is FDE8 H.

#### Main Program:

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
|       | EI        |         | ; Enable interrupts.  |
|       | MVI A,    | 08 H    | ; Bit pattern is loaded to accumulator to enable interrupts.                                  |
|       | SIM       |         | ; Enable interrupts.  |
|       | MVI A,    | 30 H    | ; Accumulator is loaded with the control word 30 H to Initialize counter 0 of 8253 in mode 0. |
|       | OUT       | 13 H    | ; Write the control word in the control word register of 8253.                                |
|       | MVI A,    | E8 H    | ; Load accumulator first with LS byte (E8 H) of the count value FDE8 H.                       |
|       | OUT       | 10 H    | ; E8 H of the count value is loaded to the counter 0 of 8253.                                 |
|       | MVI A,    | FD H    | ; Load accumulator then with MS byte (FD H) of the count value.                               |
|       | OUT       | 10 H    | ; FD H of the count value is loaded to the counter 0 of 8253.                                 |
| HERE  | JMP       | HERE    | ; Wait for interrupt.   |

As soon as the counting is over, the OUT terminal of counter 0 becomes high which enables RST 7.5. The RST 7.5 interrupts the microprocessor and the program jumps to its vector location 003C H. At this vector location say it is stored JMP FFBD H. So the monitor will transfer the program from FFBD H. Now the user transfers the program from FFBD H to a memory location where service subroutine program for RST 7.5 is

stored. The program in service subroutine will be stored to clear the memory location 2100 H to 21FF H.

#### Interrupt Service Subroutine:

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
|       | LXI H,    | 2100 H  | ; H-L pair is loaded with 2100 H, starting address of the memory location.    |
|       | MVI C,    | FF H    | ; Load FF H to C-register which is used as counter.                           |
|       | XRA A     |         | ; Clear the accumulator.  |
| LOOP  | MOV M,    | A       | ; Clear the memory location.  |
|       | INX H     |         | ; Increment the H-L register pair.  |
|       | DCR C     |         | ; Decrement the content of C-register for next byte.                          |
|       | JNZ       | LOOP    | ; If the contents of C-register are not zero then jump to LOOP for next byte. |
|       | EI        |         | ; Enable interrupts.  |
|       | RET       |         | ; Return.   |

**Example 9.6.** A positive going pulse is applied (figure 9.25) to the GATE terminal of counter 1 of 8253 used in mode 0. The RST 7.5 is used to interrupt the microprocessor. Write a program to measure the width of this pulse which is of approximately 1 second. The clock signal applied to CLK terminal of counter 1 is 10 KHz. The width should be stored in terms of counts in memory locations 2100 H and 2101 H.

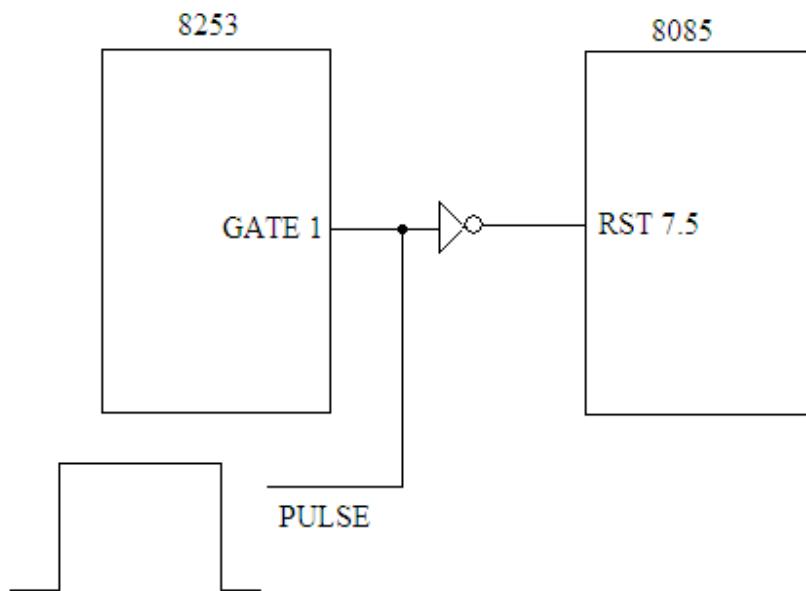


Fig. 9.25

**Solution.** The control word to initialize counter 1 of 8253 is:

|                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> | = 70 H |
| 0              | 1              | 1              | 1              | 0              | 0              | 0              | 0              |        |

The accumulator contents for SIM to enable RST 7.5, RST 6.5 and RST 5.5 are given by:

|                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |        |
| SOD            | SOE            | X              | R 7.5          | MSE            | M 7.5          | M 6.5          | M 5.5          |        |
| 0              | 0              | 0              | 0              | 1              | 0              | 0              | 0              | = 08 H |

The clock frequency used for counter 1 is 10 KMHz, so time for one clock pulse is 0.1 msec. This will also be measurement accuracy. Let us initialize the counter 1 with the count value corresponding to 2 second (1 sec for high and 1 sec for low). The count value will be 20000<sub>10</sub>, (as 20000x0.1 msec = 2 sec). The hexadecimal equivalent of 20000<sub>10</sub> is 4E20 H. So 4E20 H will be loaded as count value in counter 1.

To measure the pulse width the counter 1 is initialized in mode 0 in which signal (High pulse) on GATE terminal enables the counter; and RST 7.5 is disabled. As soon as the pulse ends, the counting will be stopped and RST 7.5 will interrupt the microprocessor. The interrupt service subroutine will store the required number of counts in 2100 H and 2101 H memory locations. The required number of counts will be obtained by subtracting the remaining counts from the initial counts.

The program is therefore given by:

#### Main Program:

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>   |
|--------------|------------------|----------------|---|
|              | EI               |                | ; Enable interrupts.  |
|              | MVI A,           | 08 H           | ; Bit pattern is loaded to accumulator to enable interrupts.                                  |
|              | SIM              |                | ; Enable interrupts.  |
|              | MVI A,           | 70 H           | ; Accumulator is loaded with the control word 70 H to Initialize counter 1 of 8253 in mode 0. |
|              | OUT              | 13 H           | ; Write the control word in the control word register of 8253.                                |
|              | MVI A,           | 20 H           | ; Load accumulator first with LS byte (20 H) of the count value 4E20 H.                       |
|              | OUT              | 11 H           | ; 20 H of the count value is loaded to the counter 1 of 8253.                                 |
|              | MVI A,           | 4E H           | ; Load accumulator then with MS byte (4E H) of the count value.                               |
|              | OUT              | 11 H           | ; 24E H of the count value is loaded to the counter 1 of 8253.                                |
| HERE         | JMP              | HERE           | ; Wait for interrupt.   |

#### Interrupt Service Subroutine:

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>  |
|--------------|------------------|----------------|--|
|              | LXI H,           | 2100 H         | ; H-L pair is loaded with 2100 H, starting address of the memory location. |
|              | IN               | 11 H           | ; Read the LS byte of the remaining counts.                                |

|        |      |  |
|--------|------|--|
| MOV D, | A    | ; Store it to D-register.  |
| IN     | 11 H | ; Read the MS byte of the remaining counts.  |
| MOV E, | A    | ; Store it to E-register. So the remaining counts are stored in D-E register pair. |
| MVI A, | 20 H | ; Initial maximum LS byte (20 H) is stored in accumulator.                         |
| SUB D  |      | ; Subtract the remaining LS byte from maximum LS byte.                             |
| MOV M, | A    | ; Store the counts in 2100 H.  |
| INX H  |      | ; Increment the H-L register pair.   |
| MVI A, | 4E H | ; Initial maximum MS byte is stored in accumulator.                                |
| SBB E  |      | ; Subtract with borrow the remaining MS byte from the maximum MS byte.             |
| MOV M, | A    | ; Store the counts in 2101 H.  |
| EI     |      | ; Enable interrupts.   |
| RET    |      | ; Return.  |

**Example 9.7.** The  $D_0$  bit of 8255 A PPI is connected trigger input (GATE terminal) of counter 2 of 8253 as shown in figure 9.26. The counter 2 is used in mode 1. The count to be loaded to the counter 2 is 0A H. Write a program so that a wave, having low for 10 pulse duration and repeats, is generated. The CRO may be connected to the OUT terminal of counter 2 to see the wave shape.

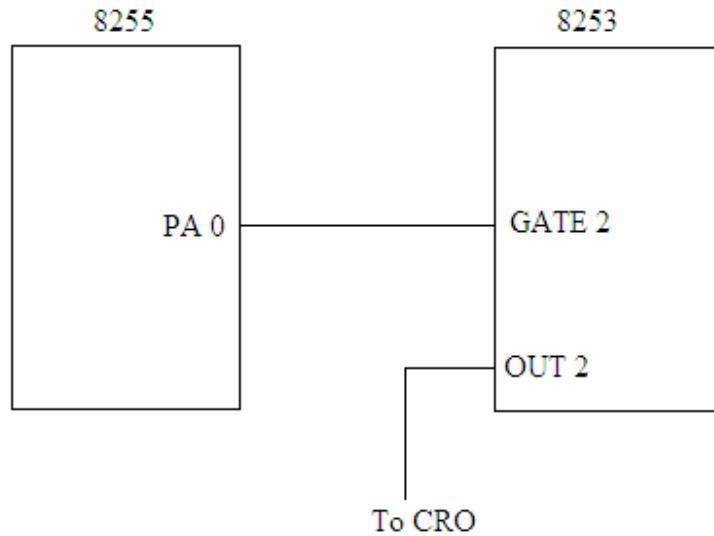


Fig. 9.26

**Solution.** The control word to initialize counter 0 of 8253 is:

|       |       |       |       |       |       |       |       |          |
|-------|-------|-------|-------|-------|-------|-------|-------|----------|
| $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ | $= A2 H$ |
| 1     | 0     | 0     | 1     | 0     | 0     | 1     | 0     |          |

**Main Program:**

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>   |
|--------------|------------------|----------------|---|
|              | MVI A,           | 30 H           | ; Accumulator is loaded with the control word 30 H to Initialize counter 2 of 8253 in mode 1. |
|              | OUT              | 13 H           | ; Write the control word in the control word register of 8253.                                |
|              | MVI A,           | 0A H           | ; Load accumulator 0A H as count value.   |
|              | OUT              | 12 H           | ; 0A H of the count value is loaded to the counter 2 of 8253.                                 |
|              | MVI A,           | 80 H           | ; Initialize 8255 A PPI with all the ports to use as output ports.                            |
|              | OUT              | 03 H           | ; 80 H is loaded to control word register of 8255.  |
|              | MVI A,           | 00 H           | ; Store 00 H to accumulator.  |
|              | OUT              | 00 H           | ; PA0 is zero.  |
|              | MVI A,           | 01             | ; Store 01 H to accumulator to make PA0 as 1.   |
|              | OUT              | 00 H           | ; PA0 is 1.   |
|              | CALL             | DELAY          | ; Call a delay program to introduce a small delay.  |
|              | JMP              | LOOP           | ; Jump to LOOP for repetition.  |

**DELAY SUBROUTINE:**

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>                                  |
|--------------|------------------|----------------|--|
| DELAY        | MVI C,           | FF H           | ; Store FF H to C register to use it as counter. |
| LOOP         | DCR C            |                | ; Decrement counter.                             |
|              | JNZ              | LOOP           | ; If count is not zero jump to LOOP.             |
|              | RET              |                | ; Return to main program.                        |

**Example 9.8.** Write a program that will count the approximate number of T-states in a program given below. The SOD line of 8085 is connected to the GATE terminal of counter 1 of 8253. The 8253 is used in mode 1. The crystal of 8085 is of 2 MHZ which is directly connected to CLK input of the counter. The number of T-states used is to be stored in memory locations 2200 H and 2201 H.

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> |
|--------------|------------------|----------------|
|              | LHLD             | 2101 H         |
|              | XCHG             |                |
|              | LHLD             | 2103 H         |
|              | MVI C,           | 00 H           |
|              | DAD D            |                |
|              | JNC              | NXT            |
|              | INR C            |                |
| NXT          | SHLD             | 2105 H         |

$MOVA, C$   
 $STA 2107H$   
 $HLT$

**Solution.** The control word to initialize counter 1 of 8253 in mode 1 is:

|                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> | = 72 H |
| 0              | 1              | 1              | 1              | 0              | 0              | 1              | 0              |        |

Let us load FFFF H as count value in counter 2.

#### Main Program:

| Label | Mnemonics     | Operand | Comments  |
|-------|---------------|---------|---|
|       | MVI A,        | 72 H    | ; Accumulator is loaded with the control word 72 H to Initialize counter 1 of 8253 in mode 1. |
|       | OUT           | 13 H    | ; Write the control word in the control word register of 8253.                                |
|       | MVI A,        | FF H    | ; Load accumulator with low byte of count value.  |
|       | OUT           | 11 H    | ; Low byte FF H of the count value is loaded to the counter 1 of 8253.                        |
|       | MVI A,        | FF H    | ; Load accumulator with High byte of count value.   |
|       | OUT           | 11 H    | ; High byte FF H of the count value is loaded to the counter 1 of 8253.                       |
|       | CALL MODPRG   |         | ; Call the modified program.  |
|       | LXI H, 2200 H |         | ; H-L pair is loaded with 2200 H, starting address of the memory location.                    |
|       | IN            | 11 H    | ; Read the LS byte of the remaining counts.   |
|       | MOV D,        | A       | ; Store it to D-register.   |
|       | IN            | 11 H    | ; Read the MS byte of the remaining counts.   |
|       | MOV E,        | A       | ; Store it to E-register. So the remaining counts are stored in D-E register pair.            |
|       | MVI A,        | FF H    | ; Initial maximum LS byte (FF H) is stored in accumulator.                                    |
|       | SUB D         |         | ; Subtract the remaining LS byte from maximum LS byte.  |
|       | MOV M,        | A       | ; Store the counts in 2100 H.   |
|       | INX H         |         | ; Increment the H-L register pair.  |
|       | MVI A,        | FF H    | ; Initial maximum MS byte is stored in accumulator.   |
|       | SBB E         |         | ; Subtract with borrow the remaining MS byte from the maximum MS byte.                        |

|        | MOV M,    | A       | ; Store the counts in 2101 H.                      |
|--------|-----------|---------|--|
|        | EI        |         | ; Enable interrupts.                               |
|        | HLT       |         | ; Stop processing.                                 |
| Label  | Mnemonics | Operand | Comments   |
| MODPRG | MVI A,    | C0 H    | ; Control word for enabling SOE and SOD.           |
|        | SIM       |         | ; Set SOD line.                                    |
|        | IN        | 11 H    | ; Read the LS byte of the remaining counts.        |
|        | MOV D,    | A       | ; Store it to D-register.                          |
|        | LHLD      | 2101 H  | ;  |
|        | XCHG      |         | ;  |
|        | LHLD      | 2103 H  | ;  |
|        | MVI C,    | 00 H    | ; Given program                                    |
|        | DAD D     |         | ;  |
|        | JNC       | NXT     | ;  |
|        | INR C     |         | ;  |
| NXT    | SHLD      | 2105 H  | ;  |
|        | MOV A,    | C       | ;  |
|        | STA       | 2107 H  | ;  |
|        | MVI A,    | 40 H    | ; Control word for enabling SOE and resetting SOD. |
|        | SIM       |         | ; Reset SOD line.                                  |
|        | RET       |         | ; Return to main program.                          |

**Example 9.9.** Write a program to generate a negative pulse of approximately one T state after every 4 msec using 8253. Consider 1 MHz clock signal is applied to CLK terminal of the counter 1 of 8253.

**Solution.** A negative pulse of approximately one T state is to be generated. This is the problem of rate generator, so 8253 may be used in mode 2. The counter 1 is to be initialized for this purpose. The GATE terminal is kept high. Further, clock frequency is 1 MHz so the time for one T state is 1  $\mu$ sec.

The number of counts to be loaded for 2 msec delay is given by:

$$= \frac{4 \text{ msec}}{1 \mu \text{sec}} = 4000_{10}.$$

The number 4000 may be counted in BCD.

The control word to initialize counter 1 of 8253 in mode 2 is:

|                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> | = 7A H |
| 0              | 1              | 1              | 1              | 0              | 1              | 0              | 1              |        |

The program is, therefore, written as:

**Main Program:**

| Label | Mnemonics | Operand | Comments |
|-------|-----------|---------|----------|
|-------|-----------|---------|----------|

|        |      |   |
|--------|------|---|
| MVI A, | 7A H | ; Accumulator is loaded with the control word 7A H to Initialize counter 1 of 8253 in mode 2. |
| OUT    | 13 H | ; Write the control word in the control word register of 8253.                                |
| MVI A, | 00 H | ; Load accumulator with low byte of count value.  |
| OUT    | 11 H | ; Low byte 00 H of the count value is loaded to the counter 1 of 8253.                        |
| MVI A, | 40 H | ; Load accumulator with High byte of count value.   |
| OUT    | 11 H | ; High byte 40 H of the count value is loaded to the counter 1 of 8253.                       |
| HLT    |      | ; Stop processing.  |

**Example 9.10.** Use counter 1 of 8253 in mode 2 as divide-by-8 counter.

**Solution.** The GATE terminal is kept high.

The control word to initialize counter 1 of 8253 in mode 2 is:

|                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> | = 54 H |
| 0              | 1              | 0              | 1              | 0              | 1              | 0              | 0              |        |

The program is written as:

**Main Program:**

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
|       | MVI A,    | 54 H    | ; Accumulator is loaded with the control word 54 H to Initialize counter 1 of 8253 in mode 2. |
|       | OUT       | 13 H    | ; Write the control word in the control word register of 8253.                                |
|       | MVI A,    | 08 H    | ; Load accumulator with count value 08 H.   |
|       | OUT       | 11 H    | ; Count Value is loaded to the counter 1 of 8253.   |
|       | HLT       |         | ; Stop processing.  |

**Example 9.11.** Write a program to generate a square wave of 3 KHz frequency using counter 1 of 8253. Consider 1.5 MHz clock signal is applied to CLK terminal of the counter.

**Solution.** To generate square wave, we use the 8253 in mode 3. The count value to be loaded to the counter 1 is given by:

$$= \frac{1.5\text{MHz}}{3\text{KHz}} = 500_{10}.$$

So load 0500 H in BCD to counter 1. The GATE terminal of counter 1 kept high.

The control word to initialize counter 1 of 8253 in mode 2 is:

|                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> | = 77 H |
| 0              | 1              | 1              | 1              | 0              | 1              | 1              | 1              |        |

The program is, therefore, written as:

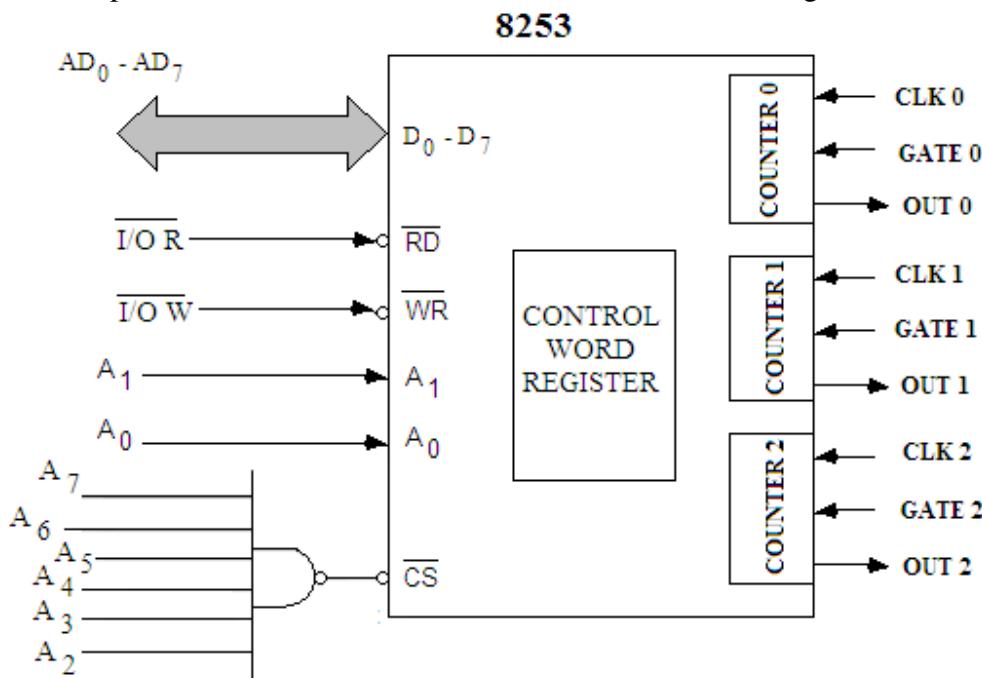
**Main Program:**

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>   |
|--------------|------------------|----------------|---|
|              | MVI A,           | 77 H           | ; Accumulator is loaded with the control word 77 H to Initialize counter 1 of 8253 in mode 3. |
|              | OUT              | 13 H           | ; Write the control word in the control word register of 8253.                                |
|              | MVI A,           | 00 H           | ; Load accumulator with low byte of count value.  |
|              | OUT              | 11 H           | ; Low byte 00 H of the count value is loaded to the counter 1 of 8253.                        |
|              | MVI A,           | 50 H           | ; Load accumulator with High byte of count value.   |
|              | OUT              | 11 H           | ; High byte 50 H of the count value is loaded to the counter 1 of 8253.                       |
|              | HLT              |                | ; Stop processing.  |

### **PROBLEMS**

1. Draw the schematic block diagram of programmable interval timer (PIT) 8253 and explain its detail..
2. Discuss the control word format for 8253.
3. Describe Read/Write control logic for 8253.
4. Explain the different modes of 8253.
5. Discuss different applications of 8253.
6. Discuss how 8253 can be used as rate generator.
7. Discuss how 8253 can be used as programmable mono shot.
8. Discuss how 8253 can be used as square wave generator.
9. Discuss how 8253 can be used as software triggered strobe.
10. Discuss how 8253 can be used as hardware triggered strobe.
11. Explain how to generate a delay using 8253 in mode 0.
12. Explain how the count value loaded to a counter of 8253 is read while the counting is in progress.
13. Write a program to generate a square wave of 1 KHz using 8253 in mode 3. The CLK terminal of the counter used is connected to a clock frequency of 1 MHz.
14. Assume that CLK terminal of the counter 0 of 8253 is connected to clock frequency of 1 KHz. It is desired to generate square wave of 1 Hz frequency using this counter. Write a program to implement this.
15. Set up 8253 as a square wave generator of 1 msec time period, if the input frequency connected to CLK terminal of counter 0 is 1 MHz.
16. Write a program to generate a negative pulse of approximately one T-state after every 1 msec. Consider 2 MHz clock frequency is applied to the CLK terminal of the counter 1.
17. Connect PC<sub>0</sub> of 8255 A PPI to the trigger input (GATE 0) of counter 0 of 8253 used in mode 1. Load the count 0024 H (binary). Write a program to generate a wave having low for 36 pulses and repeat.

18. The OUT terminal of counter 0 of 8253 is connected to RST 7.5. It is used to interrupt the microprocessor to set SOD line of 8085 after a delay of 40 msec. Use counter 0 in mode 0 and clock frequency connected to the CLK terminal is 2 MHz.
19. An LED is connected to SOD line of 8085. The should glow ON/OFF regularly with an interval of 1 sec. The delay should be introduced by counter 2 of 8253 used in mode 0. Consider clock frequency connected to the CLK terminal is 10 KHz.
20. Use 8253 in mode 3 as a square wave generator. Use count value as 10 H (BCD). The counter 1 is to be used for this purpose.
21. Use counter 0 of 8253 to generate a square wave of 10 KHz and simultaneously counter 1 as divide by N counter. ( $N = 10$  H BCD). The clock frequency connected to the CLK terminals is 1 MHz.
22. Use counter 0 of 8253 as a simple counter in mode 0. After certain delay, read the counts when the counting is in progress.
23. The 8253 is interfaced with 8085 microprocessor as shown in figure 9.27. Specify the port addresses of the three counters and control word register.



**Fig. 9.27**

(Ans.: 00 H for counter 0  
01 H for counter 1  
02 H for counter 2  
03 H for CWR )

24. Specify the control words for 8253:
  - (i) to program counter 1 in mode 5 having a 16-bit count value in BCD;
  - (ii) to count the counts on fly for the counter 1.

(Ans.: (i) 7B H, (ii) 40 H)
25. Write a program to use counter 0 of 8253 in mode 0 for BCD operation with an initial count value  $3648_{10}$ , and counter 2 in mode 3 for binary operation with an initial count value FF H.

**(Ans.: Program**   **MVI A, 31 H**  
                 **OUT 13 H**  
                 **MVI A, 96 H**  
                 **OUT 13 H**  
                 **MVI A, 48 H**  
                 **OUT 10 H**  
                 **MVI A, 36 H**  
                 **OUT 10 H**  
**MVI A, FF H**  
                 **OUT 12 H**  
                 **HLT**)

26. Explain what will be the meaning of the following program for 8253;

(i)    MVI A, 94 H  
        OUT 13 H  
        MVI A, FF H  
        OUT 12 H  
        HLT

(ii)   MVI A, 38 H  
        OUT 13 H  
        MVI A, 04 H  
        OUT 10 H  
        MVI A, 02 H  
        OUT 10 H  
        HLT

**(Ans.: (i) Counter 1 in mode 2 with binary counting,**  
**(ii) Counter 0 in mode 4 with binary counting)**

27. Write a program to use counter 0 of 8253 in mode 4 for BCD operation with an initial count value  $0224_{10}$ .
28. Write a program to use counter 1 of 8253 in mode 5 for Binary operation with an initial count value  $2AB6_{10}$ .
-

# 10

## Programmable Keyboard and Display Interface: 8279

---

The entering of program or data in the microprocessor base systems is most commonly done by a keyboard. Hence, keyboard is the most versatile input device. The keyboards with hexadecimal or ASCII characters are basically a combination of switches placed in a matrix with rows and columns. Similarly, display devices are to be connected to the system to output the data or the result, and the most common output device is seven segment display. The keyboard has to be constantly scanned to detect a key-press. The display, however, has to be supplied with the data to hold it ready. If the CPU is required to do all these operations itself, it will be heavily burdened and will have lesser time for other processes. These operations are, therefore, carried out by another device so that CPU is relieved from all these burdens. For this purpose, Intel 8279 keyboard/display interface is designed to directly connect to 8085 microprocessor. It performs both keyboard scan and output display operations repetitively and very short time of CPU is utilized for the data transfer between the device and CPU. This chapter will entirely deal with the details of this programmable keyboard and display device 8279.

### **10.1 INTEL PROGRAMMABLE KEYBOARD/DISPLAY INTERFACE 8279**

Intel 8279 Programmable Key Board/Display Interface is available in the form of 40 pin IC in plastic dual in line package (DIP). It has been designed to interface the key board (an input device) and display device (an output device) with microprocessor. The 8279 constantly scans to detect a key press and transmit the information of characteristics of the key press to the CPU. It also displays or outputs the data received from CPU to the display devices. These two operations keyboard scan and display are performed repetitively and independently without utilizing the time of CPU except for relatively short time when the data is actually transferred to and from the burden of scanning the keyboard or refreshing the display repetitively.

There are three input modes:

Scanned keyboard mode

Scanned Sensor matrix mode

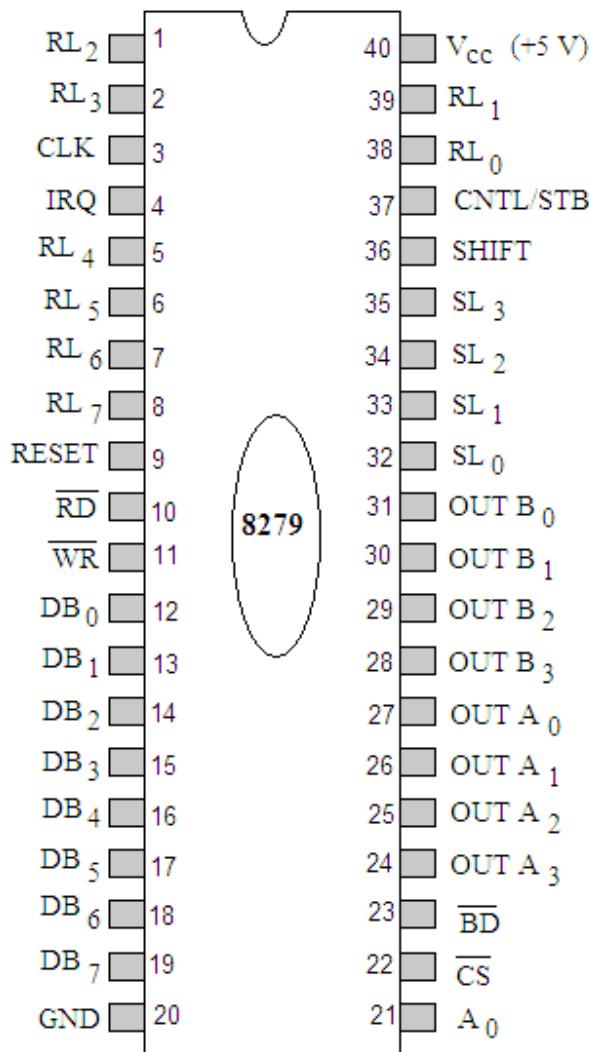
Strobed input mode.

The keyboard can provide a scanned interface to 64 contact key matrix or array of sensors or a strobed interface keyboard. Key depressions can be 2 key lock out or N-key rollover. Keyboard entries are debounced and strobed in an 8-character FIFO (First In First Out) and set the interrupt lines. If more than 8 characters are entered, overrun

interrupt status is set. The display provides a scanned display interface for LED, incandescent or other popular display technologies.

## 10.2 BLOCK DIAGRAM OF 8279

The pin diagram, logic diagram and functional block diagram of this 40 pin 8279 IC are depicted in figures 10.1, 10.2 and 10.3 respectively.



**Fig. 10.1**

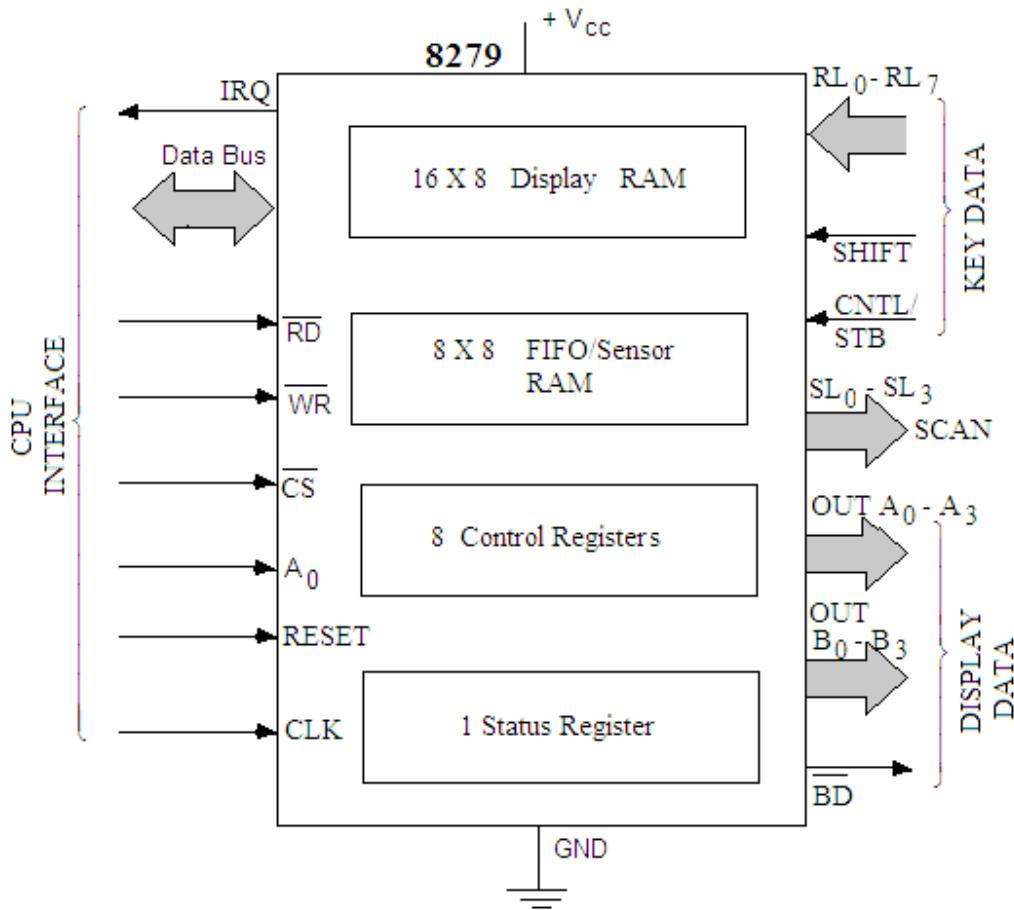
The description of pins of 8279 Programmable keyboard/display interface is given as follows:

DB<sub>0</sub>-DB<sub>7</sub>: These pins form the bidirectional 8-bit data bus. The commands are also transmitted on this data bus.

CLK: It is a clock terminal to be connected to the external clock terminal of the system clock. It is used to generate internal timing. Maximum frequency to be used is 3 MHz.

CS : This is a chip select terminal. A low signal to this terminal enables the chip for programming, reading the keyboard, etc.

A<sub>0</sub>: It is buffer address pin. A low signal on this pin indicates data and a high on this pin indicates the command.



**Fig. 10.2**

**IRQ:** It is an interrupt request output terminal. Interrupt request, becomes 1 when a key is pressed, data is available.

**SL<sub>0</sub>-SL<sub>3</sub>:** These four scan lines are used to scan the key switch or sensor matrix and the display digits. Scan line outputs scan both the keyboard and displays.

**RL<sub>0</sub>-RL<sub>7</sub>:** These are 8 Return line inputs which are connected to the scan lines through the key or sensor switches. These lines have internal pull-ups to keep high until a switch closure pulls one of the lines low.

**SHIFT:** This is an input terminal used in the scanned matrix keyboard modes. The SHIFT input status is stored along with the key position on key closure. SHIFT connects to shift key on keyboard.

**CNTL/STB:** This is control and strobe line, connected to the control key on the keyboard. It is high until a switch closure pulls it low.

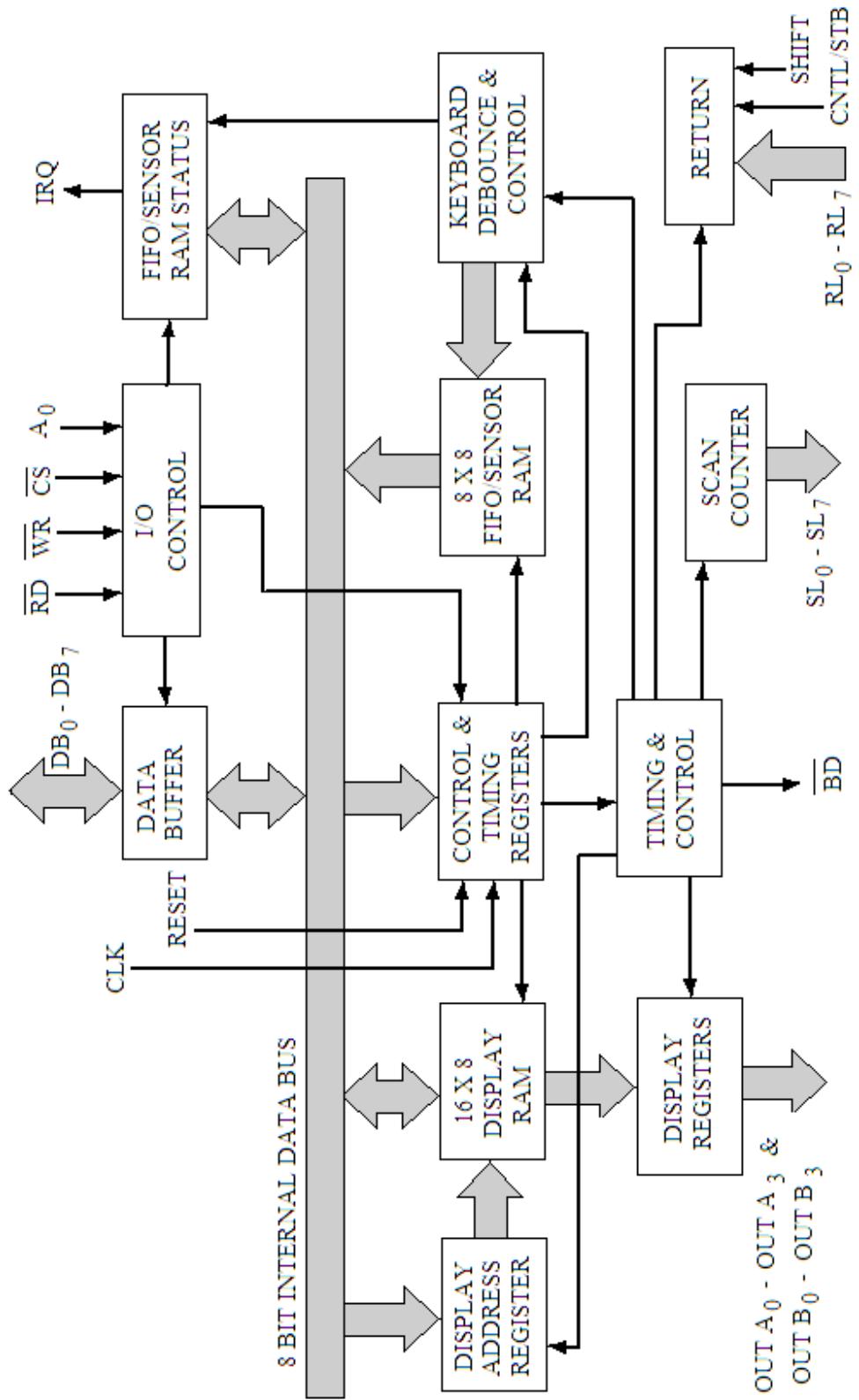


Fig. 10.3

|  |   |
|--|---|
| SHIFT:   | This is an input terminal used in the scanned matrix keyboard modes. The SHIFT input status is stored along with the key position on key closure.   |
| $\overline{RD}$ :  | It is read input terminal and is connected to microprocessor's RD signal. It reads data/status registers.   |
| $\overline{WR}$ :  | It is write input terminal and is connected to microprocessor's write terminal.   |
| OUT A <sub>0</sub> –OUT A <sub>3</sub> &<br>OUT B <sub>0</sub> -OUT B <sub>3</sub> : | These are two 4-bit ports. The display output is through two 4-bit ports (OUT A <sub>0</sub> –OUT A <sub>3</sub> and OUT B <sub>0</sub> -OUT B <sub>3</sub> ). These two ports can be combined to form an 8 bit port. |
| $\overline{BD}$ :  | This is a blanking display terminal, used to blank the display during the digit switching or by a display blanking command.   |

### 10.3 FUNCTIONAL DESCRIPTION OF 8279

The functional description of programmable keyboard and display interface 8279 will now be discussed with reference to figures 10.2 and 10.3. The 8279 has mainly been divided into four sections:

- Keyboard Section
- Scan Section
- Display Section
- Microprocessor Interface Section

#### Keyboard Section

The keyboard section includes Return buffer, keyboard debounce and control. Return buffers are to buffer the 8-input lines (RL<sub>0</sub>-RL<sub>7</sub>). In the keyboard mode, these lines are scanned, looking for the key closure in that row. If the debounce circuit detects a closed switch, it waits about 10 msec to check whether the switch remains closed. If it does, the address of the switch in the matrix in addition to other information is transferred to the FIFO.

The block FIFO/Sensor RAM and Status contains dual function 8 x 8 RAM. It will act as a FIFO in keyboard or strobed input modes. Each new entry is written into successive RAM positions and each is then read in the order of entry. FIFO status keeps track of the number of characters in the FIFO and whether it is full or empty. Too many reads or writes will be recognized as an error. The status can be read by  $\overline{RD}$  and  $\overline{CS}$  low and A<sub>0</sub> high. The status logic also provides an IRQ signal when the FIFO is not empty.

#### Scan Section

The scan section has a scan counter and four scan lines SL<sub>0</sub>-SL<sub>7</sub> which are used to scan the key switch or sensor matrix and display digits. This counter can be operated in two ways: Encoded mode and Decoded mode. In the encoded mode, the counter provides a binary count which can be decoded to provide the scan lines for the keyboard or display. However, in case of decoded mode, the scan counter itself is a decoder providing 1 of 4 scan.

#### Display Section

This section contains the 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 nibbles being displayed. The read/write addresses are programmed by CPU

command. They can also be set to auto increment after each read or write. The display RAM can be directly read by the CPU after the correct mode and address is set. These are two 4-bit ports. The display output is through two 4-bit ports (OUT A<sub>0</sub>-OUT A<sub>3</sub> and OUT B<sub>0</sub>-OUT B<sub>3</sub>). These two ports can be combined to form an 8 bit port. The data from these lines are synchronized to the scan lines (SL<sub>0</sub>-SL<sub>3</sub>) for the multiplexed digit displays. The two ports may be blanked independently by the blanking display terminal  $\overline{BD}$ . This section also includes 16 x 8 display RAM and the processor can read from or write into any of these registers.

### **Microprocessor Interface Section**

The microprocessor interface sections contains 8-bit bidirectional data lines (DB<sub>0</sub>-DB<sub>7</sub>), one interrupt request line (IRQ), one address buffer line A<sub>0</sub>, and five lines ( $\overline{RD}$ ,  $\overline{WR}$ ,  $\overline{CS}$ , RESET, CLK) for interfacing. When a high signal appears on A<sub>0</sub> terminal, the command or status registers inside the 8279 can be accessed i.e. it works as command word or status. A low, however, on this line indicates that the data register, e.g. display RAM or the FIFO/Sensor RAM can be accessed. The interrupt request line IRQ becomes high whenever data entries are stored in the FIFO. This signal is used to interrupt the microprocessor to indicate the availability of the data. The data and commands are communicated between the microprocessor and the 8279 through the data bus (DB<sub>0</sub>-DB<sub>7</sub>). The RESET pin resets the 8279, when a high signal appears on this pin. The two signals  $\overline{RD}$  and  $\overline{WR}$  enable the data bus to either send data to external bus or receive it from the external bus.

### **10.4 KEYBOARD SCAN**

As already discussed, the 8279 provides 4 scan lines (SL<sub>0</sub>-SL<sub>3</sub>) and 8 return lines (RL<sub>0</sub>-RL<sub>7</sub>). The scan lines can be operated either in encoded mode or in decoded mode.

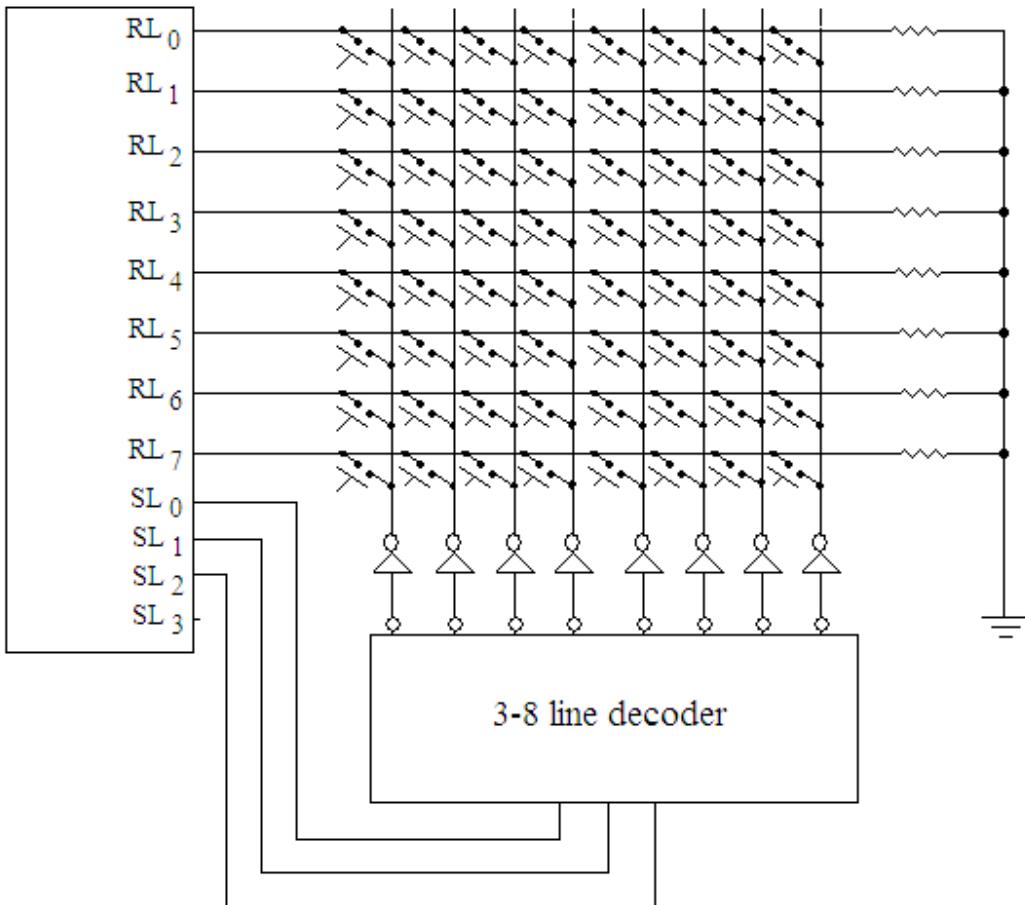
#### **Encoded Mode:**

In this encoded mode, four scan lines (SL<sub>0</sub>-SL<sub>3</sub>) can be used to generate 16 lines with the help of external decoder. Usually 3 to 8 line decoder is used with the scan lines. The most significant scan line SL<sub>3</sub> is not recommended to use in keyboard decoder. So with three scan lines (SL<sub>0</sub>-SL<sub>2</sub>) and a 3 to 8 line decoder, 8 decoded scan lines are generated. These 8 decoded scan lines in conjunction with the 8 return lines can form an 8 x 8 keyboard matrix as shown in figure 10.4. Two more extra lines SHIFT and CNTL (control) can also provide four more different combinations as shown in table 10.1.

**Table 10.1**

| SHIFT | CNTL |
|-------|------|
| 0     | 0    |
| 0     | 1    |
| 1     | 0    |
| 1     | 1    |

8279

**Fig. 10.4**

With these four combinations and  $8 \times 8$  matrix  $4 \times 8 \times 8 = 256$  character definitions are possible.

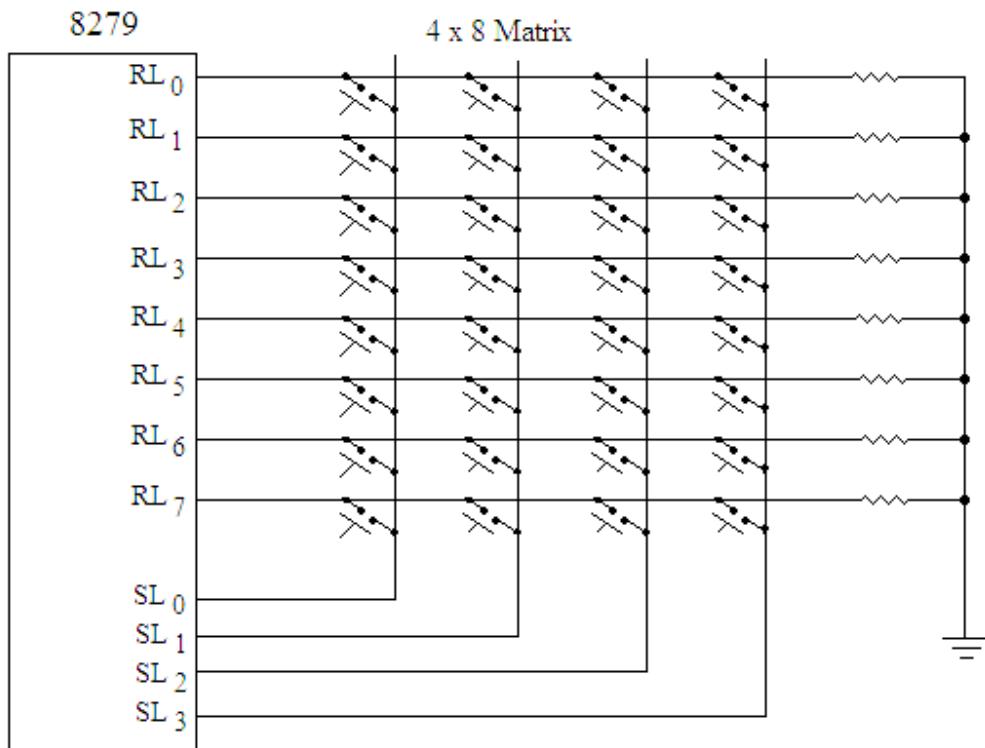
#### **Decoded Mode:**

In the decoded mode, an internal decoder is used to provide all the four scan lines (SL<sub>0</sub>-SL<sub>3</sub>) as the decoded lines as shown in table 10.2.

**Table 10.2**

| SL <sub>3</sub> | SL <sub>2</sub> | SL <sub>1</sub> | SL <sub>0</sub> |
|-----------------|-----------------|-----------------|-----------------|
| 0               | 0               | 0               | 1               |
| 0               | 0               | 1               | 0               |
| 0               | 1               | 0               | 0               |
| 1               | 0               | 0               | 0               |

These four decoded lines with 8 return lines form the keyboard matrix as shown in figure 10.5. The SHIFT and CNTL lines in combination with the keyboard matrix can provide  $4 \times 4 \times 8$  128 character definitions.

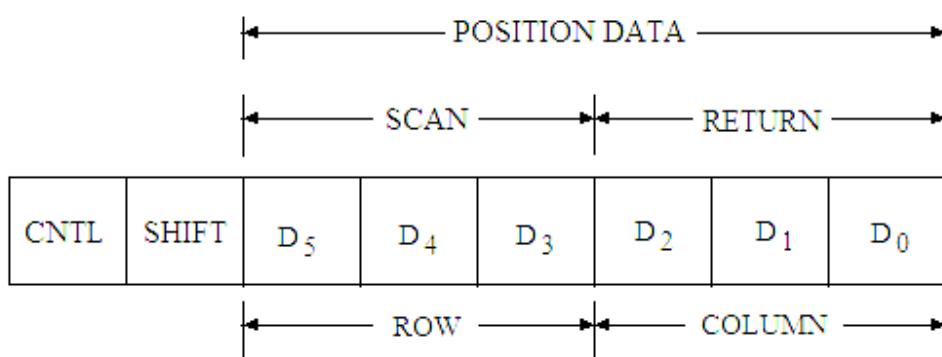


**Fig. 10.5**

The disadvantage of the decoded scan is that the number of combinations of the scan lines as given in table is only four. Hence, only four rows can be used in case of keyboard and only four digits can be used for the display.

### 10.5 SCANNED KEYBOARD

In this mode, when a key is pressed, a unique 6 bit data is generated characteristic of the key position. An 8-bit word is formed, with the 6-bit position data for the key pressed and two bits for control (CNTL) and SHIFT lines. The format for the scanned keyboard mode is shown in figure 10.6. The scan counter has three scan bits (D<sub>5</sub>-D<sub>3</sub>) as



**DATA FORMAT FOR SCANNED KEYBOARD MODE**

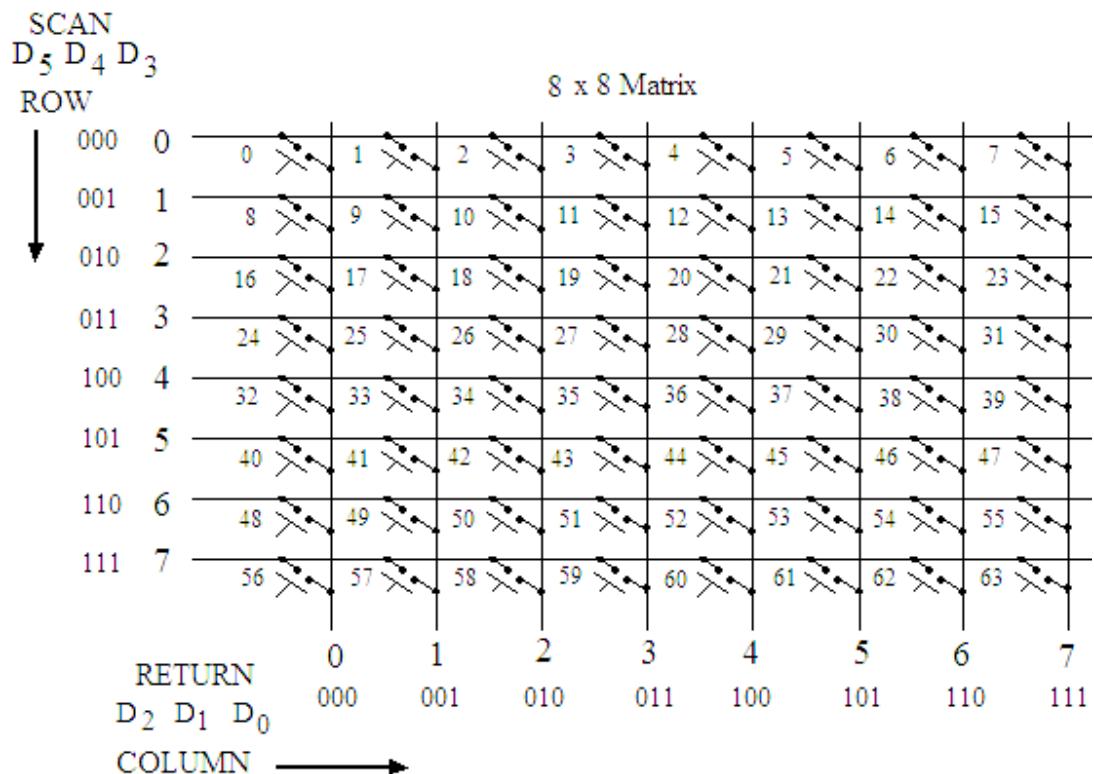
**Fig. 10.6**

000 to 111 for the row on which the pressed key is located. The column counter also has three bits (D<sub>2</sub>-D<sub>0</sub>) as 000 to 111 for column on which the pressed key is located. Figure

10.7 shows the  $8 \times 8$  keyboard matrix that has the 8 return lines and 8 scan lines. It has 64 key positions (0 to 63) which generate 8 bit data word for the key pressed. The 8 bit data word generated for a key pressed may be understood if we consider CNTL and SHIFT lines are 00. Suppose a key number 15 is pressed. The row line corresponding to the key number 15 is 1 (three bit binary is 001) and the column line for this pressed key is 7 (three bit binary is 111) so the 8 bit data formed for this pressed key becomes:

00 001 111

which is the binary equivalent of 15 (0F H). Similarly, the 8 bit data word for the key pressed 32 is 00 10 000 (20 H).



**Fig. 10.7**

The 8-bit data thus formed corresponding to the key pressed is stored in FIFO (first in first out) RAM inside the 8279. As soon as the 8-bit word is stored in FIFO RAM, IRQ terminal of 8279 goes high. This terminal is connected to one of the hardware interrupt line of CPU. With the high IRQ, the interrupt line of the CPU will be activated and the service subroutine program of the hardware interrupt will read the data from the FIFO RAM. As the data is read from the FIFO RAM, the IRQ terminal goes low; and it becomes high again if FIFO RAM contains further data.

The scanned keyboard mode has further two alternative ways of operation:

- Two-key lockout
- N-key rollover

#### 10.5.1 Two-key Lockout

The mechanical keys used in the keyboard have a problem. When a key is pressed the contacts bounce back and forth and finally settle down after a small time. Due to this contact bounce problem multiple entries may be made for the same key. This can be avoided either by using the debouncing circuit using flip-flops etc. or by making the device to wait for few milliseconds after the key is pressed. In the 8279, second method is used for debouncing which is the built in feature of this IC.

In the two-key lockout operation, if two keys are depressed within the bounce cycle, it is called a simultaneous depression and neither key will be recognized until one of the keys is released. The last key released will be recognized and its corresponding 8-bit code will be entered in FIFO RAM.

If after the first key is depressed, and no other key is found to be depressed within next two scans, then it is taken as a single key depression and the code for the depressed key will be entered in FIFO RAM.

If after the first key depression, one or more additional key depression is detected within the next two scans, then there will be following two possibilities:

If all keys are released before the first pressed key, then the first key data will be entered in FIFO RAM.

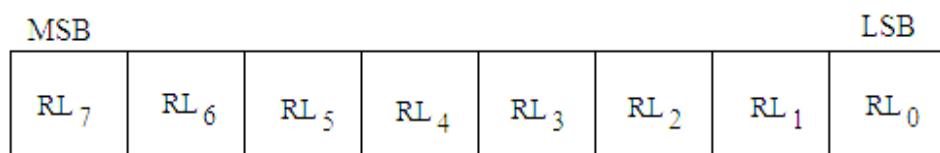
If first key pressed is released before others, then the press key will be entirely ignored.

### 10.5.2 N-key Rollover

In this mode, each key depression is treated independent. If simultaneous key depression occurs then keys are recognized and entered in FIFO RAM according to the order of the key pressed. In fact when a key is pressed the debounce circuit inside the 8279 waits for two scans then checks if this key is still pressed. If this key is still pressed, the code for the key depressed is entered into FIFO RAM.

### 10.6 SCANNED SENSOR MATRIX

As discussed earlier, the keyboard matrix size is 8 x 8 in encoded scan lines and 4 x 8 in decoded scan lines. In this mode, the keys are placed in the form of matrix either in 8 x 8 encoded scan lines or 4 x 8 decoded scan lines, the scan lines form the columns and return lines form the rows of the keyboard matrix. The key status (open or closed) is stored in RAM which can be addressed by the CPU. The data on each of the eight lines enter directly in eight columns of sensor RAM; and each switch position maps to specific sensor RAM positions. The SHIFT and CNTL lines are not considered as inputs. The format for each row of the sensor RAM is shown in figure 10.8. The logic circuits can also be connected to the return lines which will be triggered by the scan lines. The debouncing circuit is not provided in this mode. It therefore has the advantage that the CPU knows how long the sensor was closed. The IRQ line goes high if a sensor value is found to have changed at the end of sensor matrix scan. The IRQ line is cleared by the first data read operation if the auto increment flag is set to zero or by the End Interrupt Command if the auto increment flag is set to one.



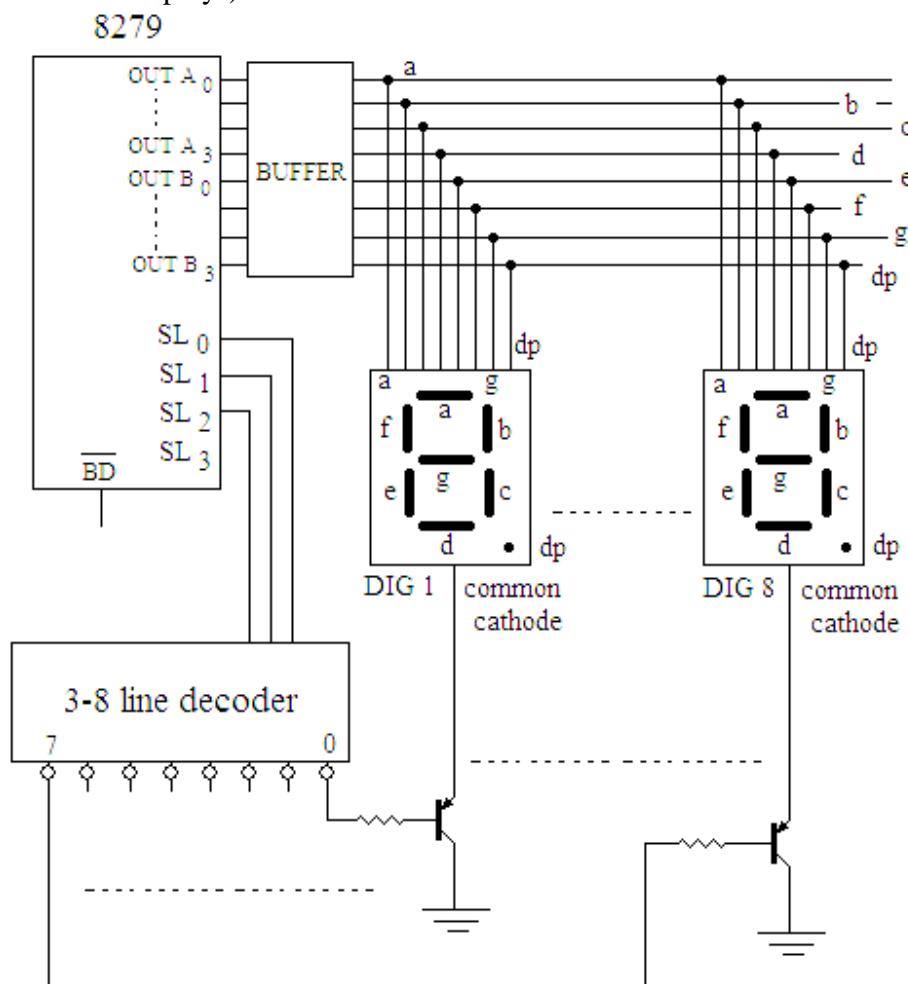
**Fig. 10.8**

## 10.7 STROBED INPUT

In this mode, the data is accepted from the return lines and go to FIFO RAM and entered at the rising edge of CNTL/STB line pulse. The data placed on the return lines can come from any source. The data is stored in the same format as shown in figure 10.8. Each scan line would lead to an 8-bit word.

## 10.8 DISPLAY INTERFACE

The interfacing of keyboard with the 8279 has been discussed in the preceding sections. The interfacing of display devices with the 8279 will now be discussed. Generally seven segment display devices are connected with 8279 using the multiplexing technique. In the multiplexing technique the seven segment code is sent to all the displays simultaneously, but the particular segment to be illuminated is only grounded (in case of common cathode displays).



**Fig. 10.9**

In the 8279, eight output terminals, OUT A<sub>0</sub>-A<sub>3</sub> and OUT B<sub>0</sub>-B<sub>3</sub> are provided for the purpose of interfacing the seven segment displays. If a 4-to-16 line decoder is used with the scan lines (SL<sub>0</sub>-SL<sub>3</sub>), then a maximum of 16 display devices may be connected to the 8279. The internal FIFO RAM of 8279 can hold 8-bit data for 16 digits. If a 3-to-8

line decoder is used with three scan lines ( $SL_0$ - $SL_2$ ), then a maximum of 8 displays may be connected to this IC. All the segments of display devices are connected in parallel (i.e. a's segment of all the displays are tight together, similarly for b's to g's and dp's segments). These segments are then connected to the output terminals OUT  $A_0$ - $A_3$  and OUT  $B_0$ - $B_3$ , as shown in figure 10.9. The OUT  $A_0$ , OUT  $A_1$ , OUT  $A_2$  and OUT  $A_3$  lines of the 8279 are connected to segments a, b, c and d respectively. The OUT  $B_0$ , OUT  $B_1$ , OUT  $B_2$  and OUT  $B_3$  lines of the 8279 are connected to segments e, f, g and dp respectively. The decoded outputs of scan lines provide 0 to 7 outputs in a periodic fashion, to select one digit of the display devices at a time. The  $\overline{BD}$  line is used to blank all display digits.

When a given bit is 1, the corresponding segment is switched ON. For example the key code for the alphabet 'C' is obtained if the segments a, d, e and f are high. The data code for the alphabet 'C' is 93 as shown in figure 9.10.

| $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |                       |
|-------|-------|-------|-------|-------|-------|-------|-------|-----------------------|
| $A_3$ | $A_2$ | $A_1$ | $A_0$ | $B_3$ | $B_2$ | $B_1$ | $B_0$ | Out terminals of 8279 |
| d     | c     | b     | a     | dp    | g     | f     | e     | Segments of display   |
| 1     | 0     | 0     | 1     | 0     | 0     | 1     | 1     | = 93 H                |

Fig. 10.10

### 10.9 DISPLAY MODES

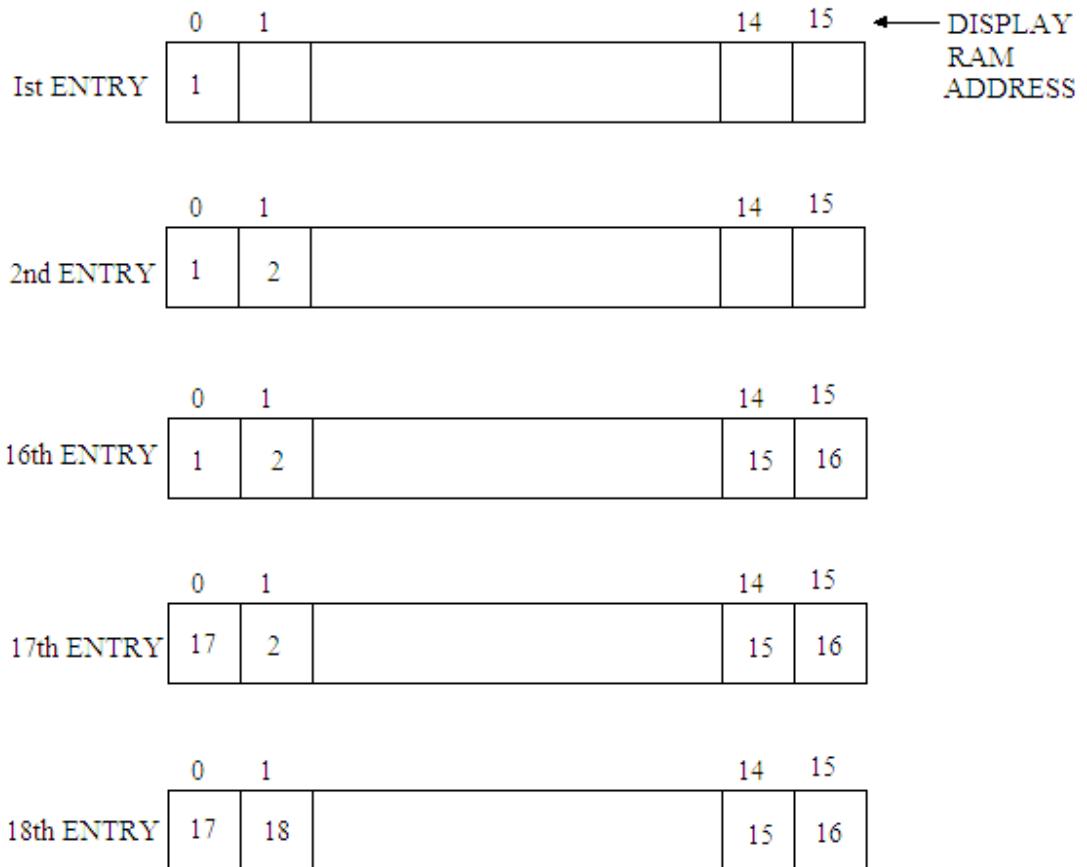
The interfacing of display devices with 8279 has been discussed in the earlier section. Further there are following two options for the display formats in the display modes of 8279:

- Left Entry Mode (Type Writer Mode)
- Right Entry Mode (Calculator Mode)

#### 10.9.1 Left Entry Mode (Type Writer Mode)

In the left entry mode, the first location of the display RAM data is treated as the segment for the left most digit and second location of the display is treated as the segment data for the second digit from the left and so on. This is similar to typing the paper with

the type writer. In this mode there is auto-increment facility as in the type writer; the



**Fig. 10.11**

carriage advances one step during typing. The left entry mode with auto increment facility is illustrated in figure 10.11 for 16 digits to be displayed on the display devices. From this figure it is clear that the first entry goes to the address 0 (first one of sixteen) for one word RAM and to the left most display position. The second entry goes to address 1 and the second display position and so on. The 16<sup>th</sup> entry goes to the address 15 of the display RAM and position 16 of the display. The 17<sup>th</sup> entry fill the left most position again.

There is a command (the details of which will be discussed in the next section) which allows entering data at an arbitrary address location of the display RAM. Figure 10.12 illustrates the result of a command that is used to display the next data to the 6<sup>th</sup> position using 8-digit display. The command word given here is 10010110 which contains 8 bits, the three most significant bits 100 represents the “write display RAM”: the next bit 1 is for auto increment and the next four bits 0110 (6<sup>th</sup>) are for the position at which it should start filling. This command does not lead to any undesirable result.

|                     |   |   |   |   |   |   |   |   |                       |
|---------------------|---|---|---|---|---|---|---|---|-----------------------|
|                     | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | ← DISPLAY RAM ADDRESS |
| Ist ENTRY           | 1 |   |   |   |   |   |   |   |                       |
| 2nd ENTRY           |   | 1 | 2 |   |   |   |   |   |                       |
| COMMAND<br>10010110 | 1 | 2 |   |   |   |   |   |   |                       |
| 3rd ENTRY           | 1 | 2 |   |   |   |   | 3 |   |                       |
| 4th ENTRY           | 1 | 2 |   |   |   |   | 3 | 4 |                       |

Fig. 10.12

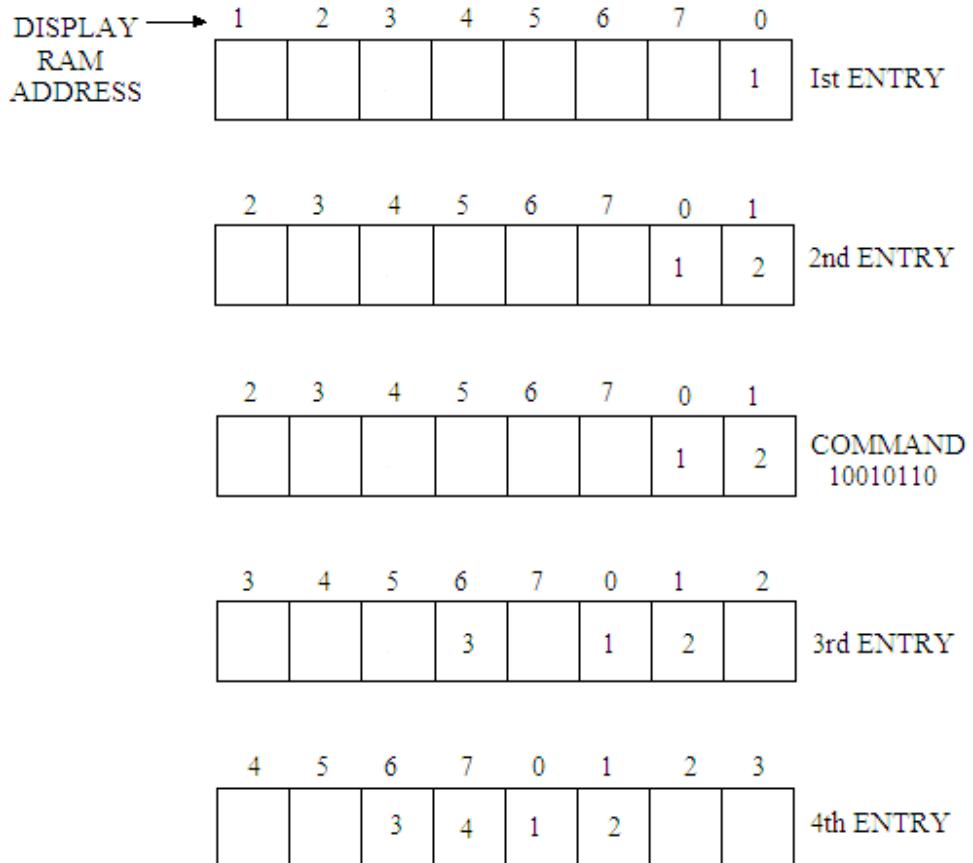
#### 10.9.2 Right Entry Mode (Calculator Mode)

The right entry mode is also known as calculator mode. In this mode the first location of the display RAM indicates the display data for the right most digit. Therefore, on the display, the data appears to start from the right and shift towards left as the digits move left in the calculator when the data is entered in to it. Figure 10.13 shows how the data are entered in this mode for 16 digits display with auto increment. The first character appears at the right display position. When a second character enters, the first character shifts towards the left by one place on the display. Similarly at the third entry, both the characters move by one place left each and the third character again takes the original right most position. It has been observed that a given character entered at a certain display position continues to remain there; but in case of right entry mode at every new entry each existing character moves one place to the left and finally the left most character shifts off the end, and is lost.

|                             |  |            |
|-----------------------------|--|------------|
| DISPLAY →<br>RAM<br>ADDRESS | 1    2                          14    15    0<br>        | 1st ENTRY  |
|                             | 2    3                          15    0    1<br>         | 2nd ENTRY  |
|                             | 3    4                          0    1    2<br>          | 3rd ENTRY  |
|                             | 0    1                          13    14    15<br>1    2 | 16th ENTRY |
|                             | 1    2                          14    15    0<br>2    3  | 17th ENTRY |
|                             | 2    3                          15    0    1<br>3    4   | 18th ENTRY |

**Fig. 10.13**

Figure 10.14 shows the entering of the data in right entry mode with auto increment for 8 digit display using the command word for entering data at an arbitrary address location of the display RAM. The results obtained for entering data at an arbitrary address location in this mode are unexpected. Therefore, a starting display RAM address 0 and sequential entry are recommended in this right entry mode.



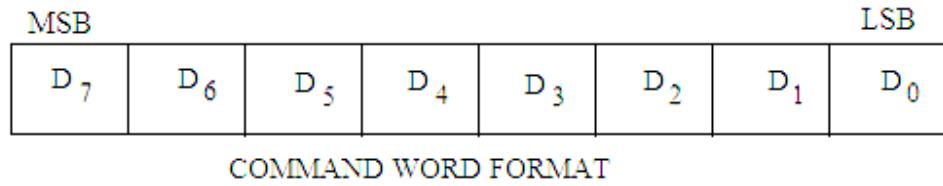
**Fig. 10.14**

### 10.10 PROGRAMMING OF 8279

The 8279 is a programmable keyboard and display interface device so it may be programmed for the desired operation. There are following 8 commands that can be used in the 8279:

- Keyboard/Display Mode Set
- Program Clock
- Read FIFO/Sensor RAM
- Read Display RAM
- Write Display RAM
- Display Write Inhibit/Blanking
- Clear
- End Interrupt/Error Mode Set

The command word, whose format is shown in figure 10.15, is sent on the data bus with  $\overline{CS}$  low and  $A_0$  high. The word is loaded to the 8279 using the write operation.



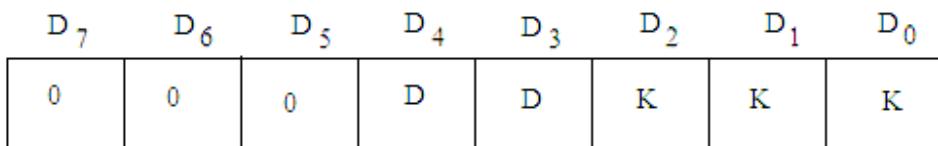
**Fig.10.15**

In this command word format, the three most significant bits selects the various operations which are given as:

| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | Function                       |
|----------------|----------------|----------------|--------------------------------|
| 0              | 0              | 0              | Keyboard/Display Mode Set      |
| 0              | 0              | 1              | Program Clock                  |
| 0              | 1              | 0              | Read FIFO/Sensor RAM           |
| 0              | 1              | 1              | Read Display RAM               |
| 1              | 0              | 0              | Write Display RAM              |
| 1              | 0              | 1              | Display Write Inhibit/Blanking |
| 1              | 1              | 0              | Clear                          |
| 1              | 1              | 1              | End Interrupt/Error Mode Set   |

#### 10.10.1 Keybaord/Display Mode Set

This command sets up the operation of keyboard and display mode. The command bit pattern for the same is given as (figure 10.16):



**Fig. 10.16**

**D D** represents the display mode and **K K K** represents the keyboard mode.  
**D D** in D<sub>4</sub> D<sub>3</sub> bit positions has the following four display mode options:

| D <sub>4</sub> | D <sub>3</sub> | Display Option                                   |
|----------------|----------------|--|
| 0              | 0              | Eight 8-bit character display with left entry    |
| 0              | 1              | Sixteen 8-bit character display with left entry  |
| 1              | 0              | Eight 8-bit character display with right entry   |
| 1              | 1              | Sixteen 8-bit character display with right entry |

**K K K** in D<sub>2</sub> D<sub>1</sub> D<sub>0</sub> bit positions has the following four display mode options:

| D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> | Keyboard Option                            |
|----------------|----------------|----------------|--|
| 0              | 0              | 0              | Encoded Scan Keyboard with 2-key lockout   |
| 0              | 0              | 1              | Decoded Scan Keyboard with 2-key lockout   |
| 0              | 1              | 0              | Encoded Scan Keyboard with N-key roll over |
| 0              | 1              | 1              | Decoded Scan Keyboard with N-key roll over |
| 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         |

The command word to set decoded scan keyboard with 2-key lockout and to have eight 8-bit characters with right entry in the display mode can be shown as:

| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> | = 11 H |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| 0              | 0              | 0              | 1              | 0              | 0              | 0              | 1              |        |

**Fig. 10.17**

If the address of the command port for 8279 is FF H, then by executing the following instructions, keyboard and display mode will be set as per the requirement discussed above:

```
MVI A, 11 H
OUT FF H
```

### 10.10.2 Program Clock

The different keys of the keyboard are scanned one by one in a sequence to detect a key press by the 8279; and also key debouncing is implemented by this device. All these functions require the timing signal. The 8279, therefore, needs an internal clock. The frequency of this clock should be around 100 KHz. But the clock of the system (i.e. the frequency of the external clock terminal of 8085) is 3 MHz. This system clock or any other external clock signal of known frequency may be applied to the CLK terminal (pin no. 3) of 8279. The 8279 internally provides an arrangement so that the clock applied at the pin no. 3 may be divided by a known factor to get the clock frequency of about 100 KHz. The frequency division is possible by the software.

| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 0              | 0              | 1              | P              | P              | P              | P              | P              |

**Fig. 10.18**

A command word is generated to get the required clock frequency after division of the system clock or external clock connected to pin no. 3 of 8279. The command word format is given in figure 10.18.

The five bits ( $D_0$  to  $D_4$ ) of the command word define the scale factor. These five bits P P P P P may be set 0 0 0 0 0 to 1 1 1 1 1 whose decimal equivalents are 0 to 31. So the scale factor may be chosen up to 31.

The scale factor will be given by:

$$\text{Scale Factor} = \frac{\text{System-frequency}}{100 \text{ KHz}}$$

If the external clock frequency applied to pin 3 of 8279 is 2 MHz, then the scale factor will be:

$$\begin{aligned}\text{Scale Factor} &= \frac{2 \text{ MHz}}{100 \text{ KHz}} \\ &= 20\end{aligned}$$

The binary equivalent of 20 in five bits is 1 0 1 0 0 (represent P P P P P) and the command word will be given by (figure 10.19):

| $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ | = 34 H |
|-------|-------|-------|-------|-------|-------|-------|-------|--------|
| 0     | 0     | 1     | 1     | 0     | 1     | 0     | 0     |        |

**Fig. 10.19**

Similarly, if the system clock frequency of 3 MHz is applied to 8279, then the scale factor is given by:

$$\begin{aligned}\text{Scale Factor} &= \frac{3 \text{ MHz}}{100 \text{ KHz}} \\ &= 30\end{aligned}$$

The binary equivalent of 30 is 1 1 1 1 0 and the command word will be given by (figure 10.20):

| $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ | = 3E H |
|-------|-------|-------|-------|-------|-------|-------|-------|--------|
| 0     | 0     | 1     | 1     | 1     | 1     | 1     | 0     |        |

**Fig. 10.20**

The following instructions after execution will set the program clock to 100 KHz, if the system clock frequency of 3 MHz is applied to 8279. The command for this is 3E H as discussed above.

```
MVI A, 3E H
OUT FF H
```

The address of the command port for 8279 is assumed as FF H.

### 10.10.3 Read FIFO/Sensor RAM

The command word for setting up the 8279 to read FIFO/sensor RAM is shown in figure 10.21.

| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 0              | 1              | 0              | A <sub>I</sub> | X              | A              | A              | A              |

**Fig. 10.21**

The bits D<sub>7</sub> D<sub>6</sub> D<sub>5</sub> represent for command word 2 (0 1 0).

A<sub>I</sub> represents Auto Increment.

X is don't care.

A A A (D<sub>2</sub> D<sub>1</sub> D<sub>0</sub>) represent RAM address bits.

In the sensor matrix mode, the address bits A A A represent on the 8 rows of the sensor RAM. If A<sub>I</sub> is set to 1, the successive Read operation is performed from the subsequent row of the sensor RAM. In the keyboard mode, auto increment bit A<sub>I</sub> and address bits A A A are irrelevant and the 8279 will automatically drive the data for each subsequent read (A<sub>0</sub> = 0) in the same sequence in which the data first entered in FIFO RAM.

If the data is to be read from the 3<sup>rd</sup> row of the sensor RAM, then the command word will be as:

| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 0              | 1              | 0              | 1              | 0              | 0              | 1              | 1              |

= 53 H

**Fig. 10.22**

#### 10.10.4 Read Display RAM

The command word for setting up the 8279 to read display RAM is shown in figure 10.23.

| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 0              | 1              | 1              | A <sub>I</sub> | A              | A              | A              | A              |

**Fig. 10.23**

The bits D<sub>7</sub> D<sub>6</sub> D<sub>5</sub> represent for command word 3 (0 1 1).

A<sub>I</sub> represents Auto Increment.

A A A A (D<sub>3</sub> D<sub>2</sub> D<sub>1</sub> D<sub>0</sub>) select one of the 16 rows of display RAM that is to be read.

If the auto increment bit A<sub>I</sub> is set to 1, then the row address will automatically be incremented after each of the Read to display RAM. The command word 0 1 1 1 0 0 0 0 (= 70 H) represents a read from the display RAM.

#### 10.10.5 Write Display RAM

To display information, the seven segment data is to be written in the internal display buffer. To enable writing seven segment data into display buffer, the write display RAM command has to be written into the command word register. The command word for the same is shown in figure 10.24.

| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 1              | 0              | 0              | A <sub>I</sub> | A              | A              | A              | A              |

**Fig. 10.24**

The bits D<sub>7</sub> D<sub>6</sub> D<sub>5</sub> represent for command word 4 (1 0 0).  
A<sub>I</sub> represents Auto Increment.  
A A A A (D<sub>3</sub> D<sub>2</sub> D<sub>1</sub> D<sub>0</sub>) is the address of the digit that the CPU writes into the buffer. Four bit address is provided to select any one of the digits.

After the write command with A<sub>0</sub> = 1, all the subsequent writes with A<sub>0</sub> = 0 will be to the display RAM

For example, to write the segment codes 63 H and B5 H in the first and second locations of the display RAM, the command will be given as:

| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> | = 90 H |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| 1              | 0              | 0              | 1              | 0              | 0              | 0              | 0              |        |

**Fig. 10.25**

The following instructions will write the segment codes 63 H and B5 H in the first and second locations of the display RAM:

```

MVI A, 90 H
OUT FF H
MVI A, 63 H
OUT FE H
MVI A, B5 H
OUT FE H

```

The first two instructions MVI A, 90 H and OUT FF H writes the command word in the command word register (FF H is the address of the command word register with A<sub>0</sub> is 1). The next two instructions MVI A, 63 H and OUT FE H writes the segment code 63 H in the first location of the display RAM (OUT FE H is the address for the same with A<sub>0</sub> = 0); and the last two instructions MVI A, B5 H and OUT FE H writes the segment data B5 H in the next location of display RAM as the in the command word auto increment A<sub>I</sub> bit is set to 1.

### 10.10.6 Display Write Inhibit/Blanking

The control word for Display Write Inhibit/Blanking is shown in figure 10.26.

| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 1              | 0              | 1              | X              | IW             | IW             | BL             | BL             |

**Fig. 10.26**

The bits D<sub>7</sub> D<sub>6</sub> D<sub>5</sub> represent for command word 5 (1 0 1).

X represents don't care may be taken as 0.

IW Inhibit Write Flag.

BL Blank Display Flag.

The display write inhibit control word inhibits writing to either left most 4 bits of the display or right most 4 bits. By setting D<sub>3</sub> (IW) bit to 1, the left most significant 4 bits of the display will be masked or inhibited and similarly, by setting D<sub>2</sub> (IW) bit to 1, the right most significant 4 bits will be masked or inhibited.

Similarly, the blank flag (BL) blanks half of the output pins. By setting D<sub>1</sub> (BL) bit to 1, the left most significant 4 bits will be blanked (or turned off); and by setting D<sub>0</sub> (BL) bit to w, the right most significant 4 bits will be blanked.

### 10.10.7 Clear

The command word to clear the display, FIFO or both is shown in figure 10.27.

| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 1              | 1              | 0              | C <sub>D</sub> | C <sub>D</sub> | C <sub>D</sub> | C <sub>F</sub> | C <sub>A</sub> |

**Fig. 10.27**

The bits D<sub>7</sub> D<sub>6</sub> D<sub>5</sub> represent for command word 6 (1 1 0).

C<sub>A</sub> Clears All (it clears both display RAM and FIFO).

C<sub>F</sub> Clears FIFO Status and resets the IRQ line.

C<sub>D</sub> C<sub>D</sub> C<sub>D</sub> Clears all rows of the display RAM to a selectable blanking code.

The selectable blanking code C<sub>D</sub> C<sub>D</sub> C<sub>D</sub> is defined as follows:

| C <sub>D</sub> | C <sub>D</sub> | C <sub>D</sub> |  |
|----------------|----------------|----------------|--|
| 0              | X              |                | All display RAM locations become 00000000 (all zeros).     |
| 0              | 0              |                | AB becomes 00100000 (20 H).                                |
| 1              | 1              |                | All ones 11111111 (FF H).<br>Enables clear display when 1. |

If  $C_F$  bit is set to 1, it clears FIFO and the display RAM status, and sets address pointer to 000; it also resets the IRQ line.

The bit  $C_A$  has the combined effect of CD and CF; if it is set to 1 it clears the display RAM and also clears FIFO status.

#### 10.10.8 End Interrupt/Error Mode Set

The command word to end interrupt/error mode set is shown in figure 10.28.

| $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 1     | 1     | 1     | E     | X     | X     | X     | X     |

Fig. 10.28

The bits  $D_7 D_6 D_5$  represent for command word 6 (1 1 0).

The bit  $D_4$  represents Error (E) which may be programmed to set the error mode or clear the IRQ line.

The bits  $D_3 - D_0$  are represented by X, means don't care and these may be considered as 0s.

For the sensor matrix mode this command lowers the IRQ line and enables writing into RAM.

For the N-key rollover if the bit E is programmed to 1 the chip will be operated in the special error mode.

#### 10.11 STATUS REGISTER (IN OPERATION)

The status word gives the information about how many characters are in the FIFO RAM and whether an error has occurred. The status word can be read by the CPU when  $A_0 = 1$  or in other words we can say that the status word can be read by the command register (IN FF H in the present case if FF H is the address of the command register).

| $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
|-------|-------|-------|-------|-------|-------|-------|-------|
| $D_U$ | $S/E$ | O     | U     | F     | N     | N     | N     |

Fig. 10.29

The bit  $D_7$  represents  $D_U$  - Display Unavailable.

The bit  $D_6$  represents S/E - Sensor Closure/Error Flag for multiple closure.

The bit  $D_5$  represents O- Over-run error.

The bit  $D_4$  represents U- Under-run error.

The bit  $D_3$  represents F- FIFO Full.

The bits  $D_2$  to  $D_0$  represent NNN as the number of key codes in FIFO RAM.

In this status word the first three bits  $D_2$  to  $D_0$  labeled as NNN identify the number of key codes or characters that are currently present in the FIFO.

The next bit F (D<sub>3</sub>) indicates whether FIFO is Full or Empty. If F = 1, then it means FIFO is full. The FIFO is empty is F = 0.

The next two bits U and O (D<sub>4</sub> and D<sub>5</sub>) are related to the under-run or over-run errors. Over-run error indicates that an attempt was made to enter the key code or character in FIFO RAM when it was already full. The other error under-run means the processor attempted to read the FIFO when it was empty.

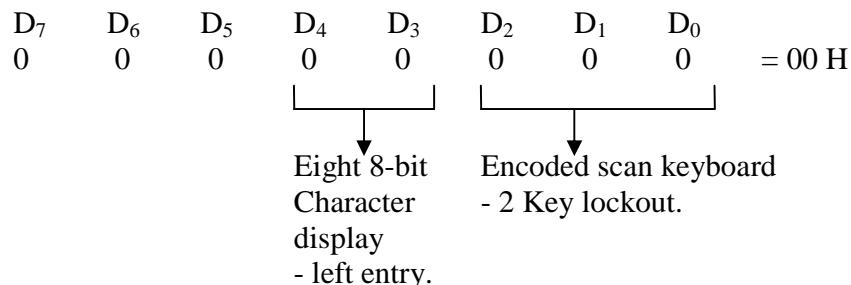
In the special error mode, the bit S/E (D<sub>6</sub>) is showing that an error flag gives an indication whether a simultaneous multiple closure error has occurred.

The most significant bit D<sub>U</sub> (D<sub>7</sub>) of the status word stands for display unavailable. When the clear display command is send, the 8279 clears the display and clearing requires some very small time. In this duration the display is unavailable and the data can not be read or written in display RAM. The bit D<sub>U</sub> is 1 when clearing is going on and it is automatically reset when display RAM becomes available again for writing.

### **Read Operation**

To read FIFO RAM for the keyboard data following steps are carried out:

**Step I** Send Keyboard/display command word to command register. Say:



**Step II** A read FIFO RAM command is written into the command register as:

|                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |        |
| 0              | 1              | 0              | A <sub>I</sub> | X              | A              | A              | A              |        |
| 0              | 1              | 0              | 0              | 0              | 0              | 0              | 0              | = 40 H |

(A<sub>I</sub> and A A A are irrelevant in sensor matrix mode).

**Step III** Read the status word by using IN instruction and port address is for A<sub>0</sub>= 0. Further, status word is checked, if D<sub>0</sub> bit is 1. If it is 1, which indicates one character or key code is available in the first location of FIFO RAM.

**Step IV** Read FIFO RAM command word is entered into the command register.

**Step V** Now the key data can be read from the data register with A<sub>0</sub>= 0.

Using these steps program may be written as given below:

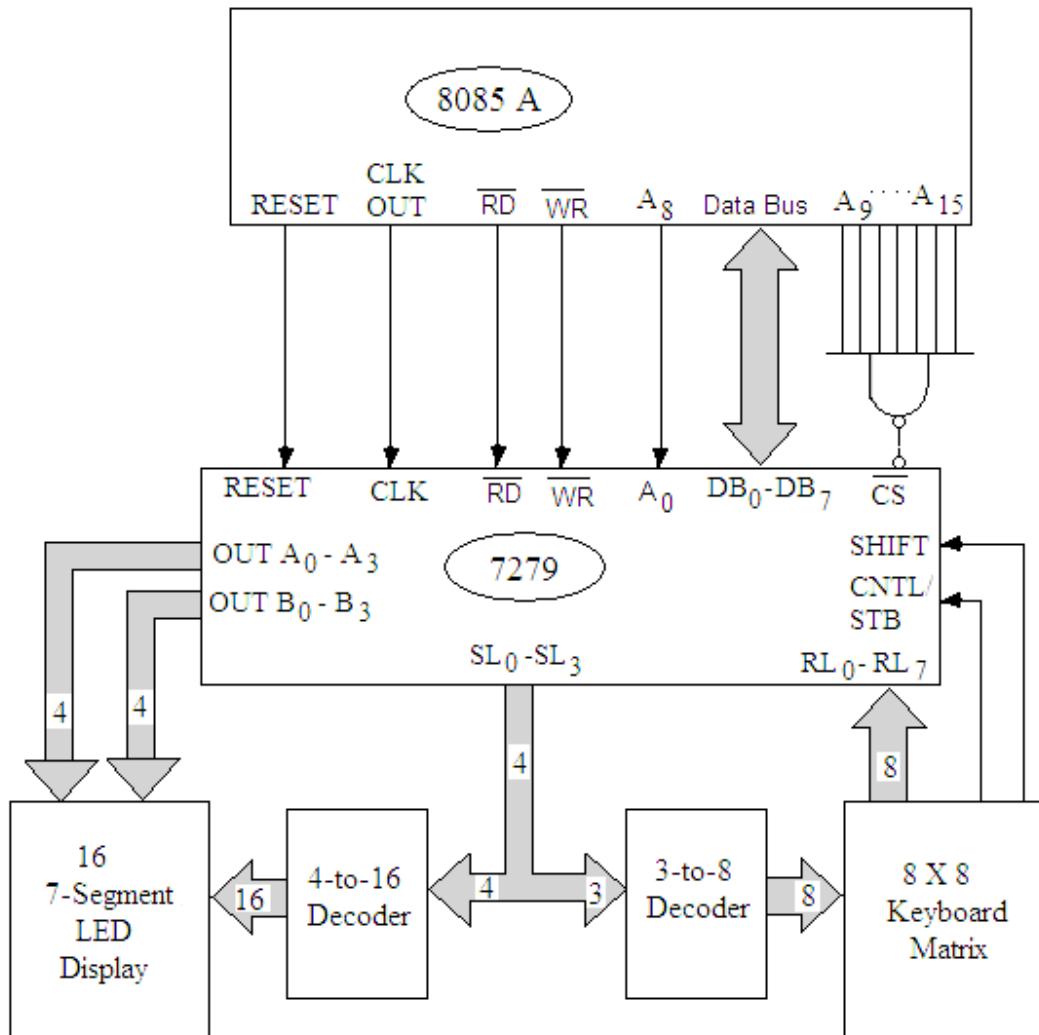
### **Program:**

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>   |
|--------------|------------------|----------------|---|
|              | MVI A,           | 00 H           | ; Encoded scan, 2 key lockout mode.                     |
|              | OUT              | FF H           | ; Keyboard display code is written in command register. |
| LOOP         | IN               | FF H           | ; Read the status word.                                 |
|              | ANI              | 01 H           | ; Check if D <sub>0</sub> is 1.                         |

|        |      |   |
|--------|------|---|
| JZ     | LOOP | ; If $D_0 = 0$ then read the status word again till $D_0$ is 1. |
| MVI A, | 40 H | ; Load FIFO RAM command word                                    |
| OUT    | FF H | ; in command word register.                                     |
| IN     | FE H | ; Read the data register to take the key code.                  |

### 10.12 INTERFACING OF 8279 WITH 8085

Since the 8279 is an I/O device, so it can be used as memory mapped I/O device or I/O mapped I/O device. The interfacing of this IC with 8085 microprocessor is shown in figure 10.29.



**Fig. 10.29**

The description of the connections of this IC with 8085 microprocessor is given below:

- The 8 data lines ( $DB_0-DB_7$ ) are connected to the data bus of the microprocessor.

- The RESET signal of 8279 is connected to the RESET OUT terminal of the CPU.
- The active low terminals  $\overline{RD}$  and  $\overline{WR}$  of 8279 are connected to active low terminals  $\overline{IOR}$  and  $\overline{IOW}$  of the CPU.
- The CLK signal of this IC may be connected to either an external clock signal or the CLK OUT terminal of the processor.
- The buffer address terminal  $A_0$  of 8279 is connected to the  $A_8$  terminal of the address bus of the microprocessor. A low signal on this pin indicates data word and a high on this pin indicates the command word.
- The output of the address decoder circuit is connected to active low terminal  $\overline{CS}$  of the IC 8279. The  $A_9-A_{15}$  terminals of the address bus of the microprocessor are connected to the inputs of the decoder circuit. These connections show that FF H is the address of the command register and FE H is the address of the data register.
- The Scan lines  $SL_0-SL_2$  of the 8279 are connected to a 3 to 8 line decoder, the outputs of which are connected to the  $8 \times 8$  keyboard matrix. The eight return lines  $RL_0-RL_7$  along with the CNTL (control) and SHIFT terminals are also connected with the keyboard matrix.
- The sixteen 7-segment LED displays are also connected to this IC 8279 through the OUT  $A_0-A_3$  and OUT  $B_0-B_3$  terminals. The common terminals of the digit display are connected to the Scan lines  $SL_0-SL_3$  through 4-to-16 line decoder.

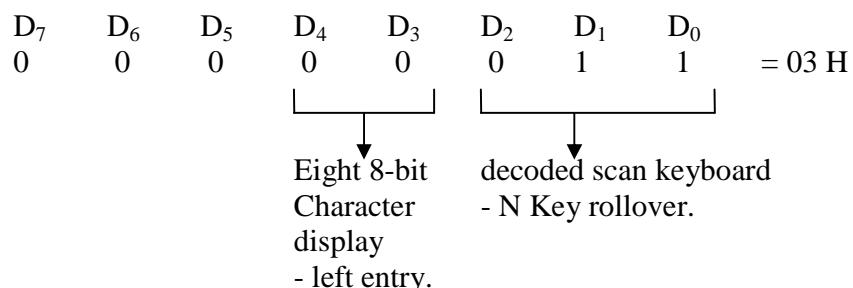
**Example 10.1.** A  $4 \times 4$  keyboard matrix is to be interfaced with 8085 microprocessor using 8279. Give the necessary hardware for the same. Also give the software to enter the hex code of the key pressed and store this code to memory location 2500 H.

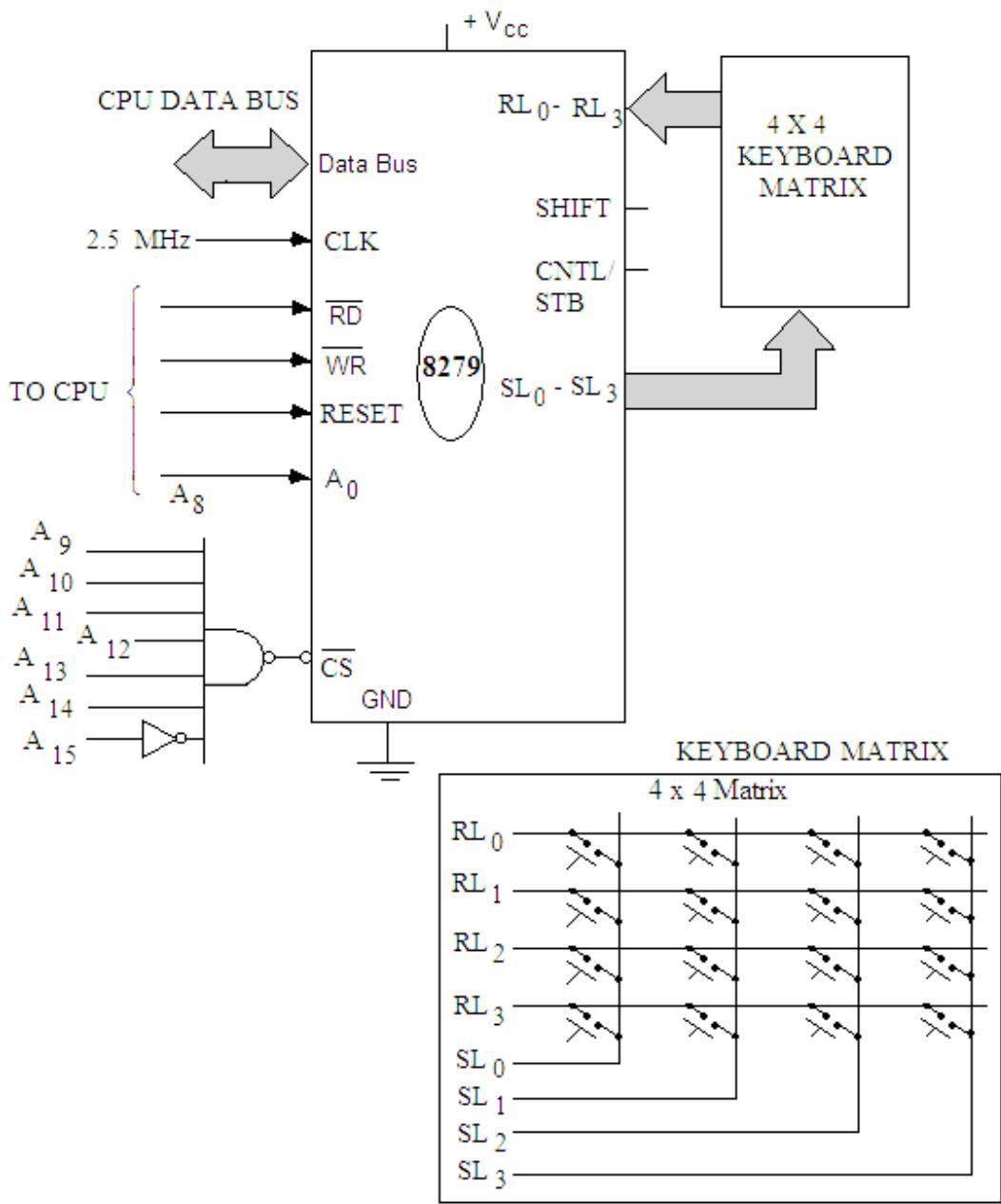
Given frequency of the clock connected to the CLK terminal of 8279 is 2.5 MHz. Assume Decoded scan mode with N-key-roll over. Let control port address is 81 H and data port address is 80 H.

**Solution.** The hardware needed to interface  $4 \times 4$  keyboard matrix with 8085 microprocessor is given in figure 10.30.

The initialization steps for providing the software are given below:

1. Send Keyboard/display command word to command register. Say:





**Fig. 10.30**

2. The frequency of the clock is to be set around 100 KHz from 2.5 MHz. This frequency is to be divided by 25 generating the program clock word as:

|                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |        |
| 0              | 0              | 1              | P              | P              | P              | P              | P              | = 39 H |
| 0              | 0              | 1              | 1              | 1              | 0              | 0              | 1              |        |

11001 for PPPPP is 25.

3. Generate clear command word as:

|                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |        |
| 1              | 1              | 0              | 0              | 0              | 0              | 1              | 1              | = C3 H |

It will clear all.

4. A read FIFO RAM command is written into the command register as:

|                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |        |
| 0              | 1              | 0              | A <sub>I</sub> | X              | A              | A              | A              | = 40 H |
| 0              | 1              | 0              | 0              | 0              | 0              | 0              | 0              |        |

(A<sub>I</sub> and A A A are irrelevant in sensor matrix mode).

5. Check the status word if the key is pressed.

The program for the same is written as:

**Program:**

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
|       | MVI A,    | 03 H    | ; Decoded scan, N key roll-over mode.                               |
|       | OUT       | 81 H    | ; Keyboard display code is written in command register.             |
|       | MVI A,    | 39 H    | ; Program clock   |
|       | OUT       | 81 H    | ; is set to 100 KHz.  |
|       | MVI A,    | C3 H    | ; Clear FIFO.   |
|       | OUT       | 81 H    | ; Clear command word is loaded to command register.                 |
| LOOP  | IN        | 81 H    | ; Read the status word.   |
|       | ANI       | 07 H    | ; Check if key pressed.   |
|       | JZ        | LOOP    | ; If not then read the status word again till the key is pressed.   |
|       | MVI A,    | 40 H    | ; Load FIFO RAM command word  |
|       | OUT       | 81 H    | ; in command word register.   |
|       | IN        | 80 H    | ; Read the data register to take the key code.                      |
|       | STA       | 2500 H  | ; Store the Hex code for the key pressed in memory location 2500 H. |
|       | HLT       |         | ; Stop processing.  |

**Example 10.2.** An 8279 is to be initialized with the following requirements:

Keyboard encoded scan mode with N key rollover.

External clock frequency is 2 MHz.

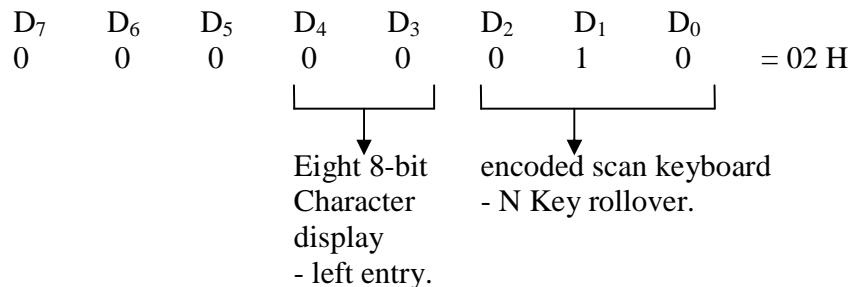
Control port address is FF H and

Data port address is FE H.

The IRQ line of the 8279 is connected to the RST 7.5 interrupt of 8085A. Write an interrupt service routine to store the hex code of the key pressed in 2501 memory location.

**Solution.** The initialization steps for providing the software are given below:

1. Send Keyboard/display command word to command register. Say:



2. The frequency of the clock is to be set around 100 KHz from 2 MHz. This frequency is to be divided by 20 generating the program clock word as:

|                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |        |
| 0              | 0              | 1              | P              | P              | P              | P              | P              | = 34 H |
| 0              | 0              | 1              | 1              | 0              | 1              | 0              | 0              |        |

10100 for PPPPP is 20.

4. Generate clear command word as:

|                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |        |
| 1              | 1              | 0              | 0              | 0              | 0              | 1              | 1              | = C3 H |

It will clear all.

4. A read FIFO RAM command is written into the command register as:

|                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |        |
| 0              | 1              | 0              | A <sub>I</sub> | X              | A              | A              | A              | = 40 H |
| 0              | 1              | 0              | 0              | 0              | 0              | 0              | 0              |        |

(A<sub>I</sub> and A A A are irrelevant in sensor matrix mode).

5. Check the status word if the key is pressed.

#### Program:

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
|       | MVI A,    | 02 H    | ; Encoded scan, N key roll-over mode.                   |
|       | OUT       | FF H    | ; Keyboard display code is written in command register. |
|       | MVI A,    | 34 H    | ; Program clock   |
|       | OUT       | FF H    | ; is set to 100 KHz.                                    |
|       | MVI A,    | C3 H    | ; Clear FIFO.   |
|       | OUT       | FF H    | ; Clear command word is loaded to command register.     |
|       | EI        |         | ; Enable interrupts.                                    |
|       | MVI A,    | 0B H    | ; Enable RST 7.5.                                       |
|       | SIM       |         | ; Set interrupt mask.                                   |

In this case when key code is available in FIFO RAM the IRQ line becomes high and enables RST 7.5 interrupt. The program will jump to the vector location 003C H of this interrupt, from where it jumps to the service routine.

003C H JMP ISR ; Jump to service routine.

|     |        |        |   |
|-----|--------|--------|---|
| ISR | MVI A, | 40 H   | ; Load FIFO RAM command word  |
|     | OUT    | FF H   | ; in command word register.   |
|     | IN     | FE H   | ; Read the data register to take the key code.                      |
|     | STA    | 2501 H | ; Store the Hex code for the key pressed in memory location 2500 H. |
|     | RET    |        | ; Return  |

**Example 10.3.** An 8279 keyboard/display interface is used to drive eight 7-segment display in multiplexed mode. Write a program to initialize 8279 to display a message 'HELLO 07'.

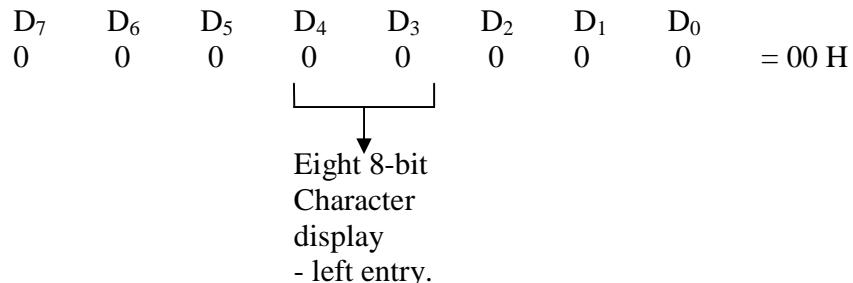
External clock frequency is 1.5 MHz.

Control port address is 19 H and

Data port address is 18 H.

**Solution.** The initialization steps for providing the software are given below:

- Send Keyboard/display command word to command register. Say:



- The frequency of the clock is to be set around 100 KHz from 1.5 MHz. This frequency is to be divided by 15 generating the program clock word as:

|                |                |                |                |                |                |                |                |       |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|-------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |       |
| 0              | 0              | 1              | P              | P              | P              | P              | P              |       |
| 0              | 0              | 1              | 0              | 1              | 1              | 1              | 1              | = 2FH |

01111 for PPPPP is 15.

- Generate clear command word as:

|                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |        |
| 1              | 1              | 0              | 1              | 0              | 0              | 1              | 1              | = D3 H |

It will clear all.

- Display RAM command is written into the command register as:

|                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |        |
| 1              | 0              | 0              | A <sub>I</sub> | A              | A              | A              | A              |        |
| 1              | 0              | 0              | 1              | 0              | 0              | 0              | 0              | = 90 H |

(A<sub>I</sub>=1 for auto increment and 0000 for A A A A as address bits).

#### Program:

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
|       | MVI A,    | 00 H    | ; 8 bit character display mode.                         |
|       | OUT       | 19 H    | ; Keyboard display code is written in command register. |

|     |        |        |   |
|-----|--------|--------|---|
|     | MVI A, | 2F H   | ; Program clock   |
|     | OUT    | 19 H   | ; is set to 100 KHz.                                    |
|     | MVI A, | D3 H   | ; Clear FIFO.   |
|     | OUT    | 19 H   | ; Clear command word is loaded to command register.     |
| UP  | IN     | 19 H   | ; Read the status word.                                 |
|     | ANI    | 80 H   | ; Check if display available.                           |
|     | JNZ    | UP     | ; If not available, read again.                         |
|     | LXI H, | 2100 H | ; Point to the data in address location.                |
|     | MVI C, | 08     | ; C is used counter for eight characters of 'HELLO 07'. |
|     | MVI A, | 90 H   | ; Write display command                                 |
|     | OUT    | 19 H   | ; The command word is loaded in command register.       |
| REP | MOV A, | M      | ; Write the segment code in                             |
|     | OUT    | 18 H   | ; display RAM.  |
|     | INX H  |        | ; Increment H-L pair to point the next code.            |
|     | DCR C  |        | ; Decrement C.  |
|     | JNZ    | REP    | ; If C is not zero jump to REP.                         |
|     | HLT    |        | ; Stop processing.                                      |

Enter hex codes for HELLO 07 in the memory locations starting at 2100 H.

| Location | Hex code for |
|----------|--------------|
| 2100 H   | H            |
| 2101 H   | E            |
| 2102 H   | L            |
| 2103 H   | L            |
| 2104 H   | O            |
| 2105 H   | - (Blank)    |
| 2106 H   | 0            |
| 2107 H   | 7            |

**Example 10.4.** An 8 X 8 keyboard matrix (64 keys) and 8 common cathode seven segment displays are interfaced with 8085 through 8279. Write a program to initialize 8279 to display a message 'DISPLAY' on pressing key "0".

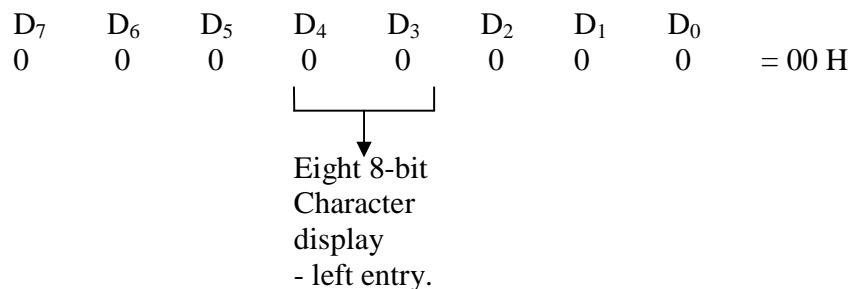
External clock frequency is 2.0 MHz.

Control port address is 05 H and

Data port address is 04 H.

**Solution.** The initialization steps for providing the software are given below:

- Send Keyboard/display command word to command register. Say:



2. The frequency of the clock is to be set around 100 KHz from 2.0 MHz. This frequency is to be divided by 20 generating the program clock word as:

|                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |        |
| 0              | 0              | 1              | P              | P              | P              | P              | P              |        |
| 0              | 0              | 1              | 1              | 0              | 1              | 0              | 0              | = 34 H |

10100 for PPPPP is 20.

3. Generate clear command word as:

|                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |        |
| 1              | 1              | 0              | 1              | 0              | 0              | 1              | 1              | = D3 H |

It will clear all.

4. Display RAM command is written into the command register as:

|                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |        |
| 1              | 0              | 0              | A <sub>I</sub> | A              | A              | A              | A              |        |
| 1              | 0              | 0              | 1              | 0              | 0              | 0              | 0              | = 90 H |

(A<sub>I</sub>=1 for auto increment and 0000 for A A A A as address bits).

### Program:

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
|       | MVI A,    | 00 H    | ; 8 bit character display mode.                         |
|       | OUT       | 05 H    | ; Keyboard display code is written in command register. |
|       | MVI A,    | 34 H    | ; Program clock   |
|       | OUT       | 05 H    | ; is set to 100 KHz.                                    |
|       | MVI A,    | D3 H    | ; Clear FIFO.   |
|       | OUT       | 05 H    | ; Clear command word is loaded to command register.     |
| UP    | IN        | 05 H    | ; Read the status word.                                 |
|       | ANI       | 80 H    | ; Check if display available.                           |
|       | JNZ       | UP      | ; If not available, read again.                         |
| UP1   | IN        | 05      | ; Read the status word again if the "0" is pressed.     |
|       | ANI       | 07 H    | ; Check for 0.  |
|       | JZ        | UP1     | ; If zero, key "0" is not pressed repeat.               |
|       | MVI A,    | 40 H    | ; Load FIFO command word                                |
|       | OUT       | 05 H    | ; in command word register.                             |
| REP   | IN        | 04 H    | ; Read the data register to take the key code.          |

|        |          |  |
|--------|----------|--|
| CPI    | 00 H     | ; Key “0” is pressed or not.                           |
| JNZ    | REP      | ; If not repeat.                                       |
| LXI H, | 2100 H   | ; Point to the data in address location.               |
| MVI C, | 07       | ; C is used counter for eight characters of ‘DISPLAY’. |
| MVI A, | 90 H     | ; Write display command                                |
| OUT    | 05 H     | ; The command word is loaded in command register.      |
| LOOP   | MOV A, M | ; Write the segment code in display RAM.               |
|        | OUT 04 H |  |
|        | INX H    | ; Increment H-L pair to point the next code.           |
|        | DCR C    | ; Decrement C.   |
|        | JNZ LOOP | ; If C is not zero jump to REP.                        |
|        | HLT      | ; Stop processing.                                     |

Enter hex codes for DISPLAY in the memory locations starting at 2100 H.

| Location | Hex code for |
|----------|--------------|
| 2100 H   | D            |
| 2101 H   | I            |
| 2102 H   | S            |
| 2103 H   | P            |
| 2104 H   | L            |
| 2105 H   | A            |
| 2106 H   | Y            |

**Example 10.5.** An 8279 keyboard/display interface is used to drive sixteen 7-segment display in multiplexed mode. Write a program to initialize 8279 to display a blinking message “CONGRATULATIONS”.

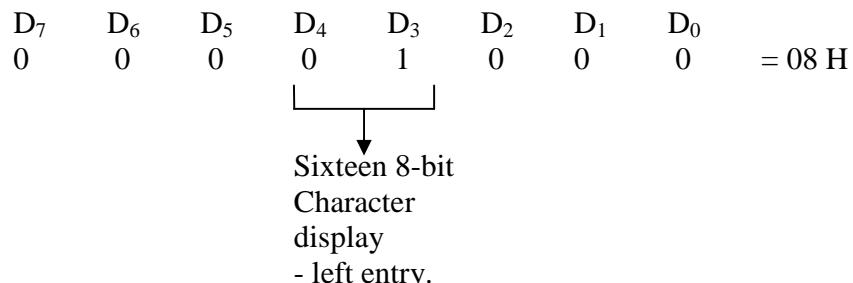
External clock frequency is 1.5 MHz.

Control port address is 19 H and

Data port address is 18 H.

**Solution.** The initialization steps for providing the software are given below:

- Send Keyboard/display command word to command register. Say:



- The frequency of the clock is to be set around 100 KHz from 1.5 MHz. This frequency is to be divided by 15 generating the program clock word as:

|                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |        |
| 0              | 0              | 1              | P              | P              | P              | P              | P              | = 2F H |
| 0              | 0              | 1              | 0              | 1              | 1              | 1              | 1              |        |

01111 for PPPPP is 15.

3. Generate clear command word as:

|                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |        |
| 1              | 1              | 0              | 1              | 0              | 0              | 1              | 1              | = D3 H |

It will clear all.

4. Display RAM command is written into the command register as:

|                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |        |
| 1              | 0              | 0              | A <sub>I</sub> | A              | A              | A              | A              |        |
| 1              | 0              | 0              | 1              | 0              | 0              | 0              | 0              | = 90 H |

(A<sub>I</sub>=1 for auto increment and 0000 for A A A A as address bits).

#### Program:

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
|       | MVI A,    | 08 H    | ; 8 bit character display mode.                               |
|       | OUT       | 19 H    | ; Keyboard display code is written in command register.       |
|       | MVI A,    | 2F H    | ; Program clock   |
|       | OUT       | 19 H    | ; is set to 100 KHz.  |
|       | MVI A,    | D3 H    | ; Clear FIFO.   |
|       | OUT       | 19 H    | ; Clear command word is loaded to command register.           |
| UP    | IN        | 19 H    | ; Read the status word.                                       |
|       | ANI       | 80 H    | ; Check if display available.                                 |
|       | JNZ       | UP      | ; If not available, read again.                               |
| LOOP  | LXI H,    | 2100 H  | ; Point to the data in address location.                      |
|       | MVI C,    | 0F      | ; C is used counter for 15 characters of 'CONGRATULATIONS'.   |
|       | MVI A,    | 90 H    | ; Write display command                                       |
|       | OUT       | 19 H    | ; The command word is loaded in command register.             |
| REP   | MOV A,    | M       | ; Write the segment code in display RAM.                      |
|       | OUT       | 18 H    |   |
|       | INX H     |         | ; Increment H-L pair to point the next code.                  |
|       | DCR C     |         | ; Decrement C.  |
|       | JNZ       | REP     | ; If C is not zero jump to REP.                               |
|       | CALL      | DELAY   | ; Introduce some delay using a Delay subroutine.              |
|       | CALL      | CLEAR   | ; A subroutine to clear the display is used.                  |
|       | CALL      | DELAY   | ; Delay is introduced so that display is clear for some time. |
|       | JUMP      | LOOP    | ; Jump to repeat the display.                                 |

HLT ; Stop processing.

Enter hex codes for “CONGRATULATIONS” in the memory locations starting at 2100 H.

| Location | Hex code for |
|----------|--------------|
| 2100 H   | C            |
| 2101 H   | O            |
| 2102 H   | N            |
| 2103 H   | G            |
| 2104 H   | R            |
| 2105 H   | A            |
| 2106 H   | T            |
| 2107 H   | U            |
| 2108 H   | L            |
| 2109 H   | A            |
| 210A H   | T            |
| 210B H   | I            |
| 210C H   | O            |
| 210D H   | N            |
| 210E H   | S            |

### PROBLEMS

1. Draw the function block diagram of Programmable Keyboard/Display interface 8279 and also discuss the function of each block.
2. Discuss three input modes of Keyboard scan of 8279.
3. Describe Scanned keyboard mode of 8279 with 2-key lockout and N-key rollover.
4. Discuss Scanned sensor matrix mode for 8279.
5. Discuss encoded mode of Keyboard scan of 8279.
6. Discuss decoded mode of Keyboard scan of 8279.
7. Describe left entry mode for display of 8279.
8. Name 8 command words of 8279 and discuss the command word format for the program clock.
9. Discuss the command word format for Read display RAM.
10. Discuss the command word format for keyboard/display mode set.
11. Discuss the command word format for Read FIFO/Sensor RAM.
12. Discuss the command word format for Display write Inhibit/blanking.
13. Discuss the command word format for End Interrupt/Error mode set.
14. Explain status register of 8279. How the status word can be read.
15. Discuss how 8279 can be interfaced with 8085A.
16. Mention various steps to be carried out for the initialization of 8279 with following specifications:

Keyboard encoded scan mode with N key rollover.  
External clock frequency is 2 MHz.  
Control port address is 11 H and  
Data port address is 10 H.

17. An 8279 keyboard/display interface is used to drive sixteen 7-segment display in multiplexed mode. Write a program to initialize 8279 to display a blinking message “HAPPY BIRTH DAY”.  
External clock frequency is 2.0 MHz.  
Control port address is 05 H and  
Data port address is 04 H.
  18. An 8 x 8 keyboard matrix (64 keys) and 8 common cathode seven segment displays are interfaced with 8085 through 8279. Write a program to initialize 8279 to display a message ‘PLEASE’ on pressing key “0”.  
External clock frequency is 2.5 MHz.  
Control port address is FF H and  
Data port address is FE H.
  19. An 8279 keyboard/display interface is used to drive sixteen 7-segment display in multiplexed mode. Write a program to initialize 8279 to display a message ‘BEST WISHES’.  
External clock frequency is 2.5 MHz.  
Control port address is 01 H and  
Data port address is 00 H.
  20. An 8 x 8 keyboard matrix is to be interfaced with 8085 microprocessor using 8279. Give the necessary hardware for the same. Also give the software to enter the hex code of the key pressed and store this code to memory location 2500 H.  
Given frequency of the clock connected to the CLK terminal of 8279 is 2.5 MHz.  
Assume Decoded scan mode with N-key-roll over. Let control port address is 19 H and data port address is 18 H.
-

# 11

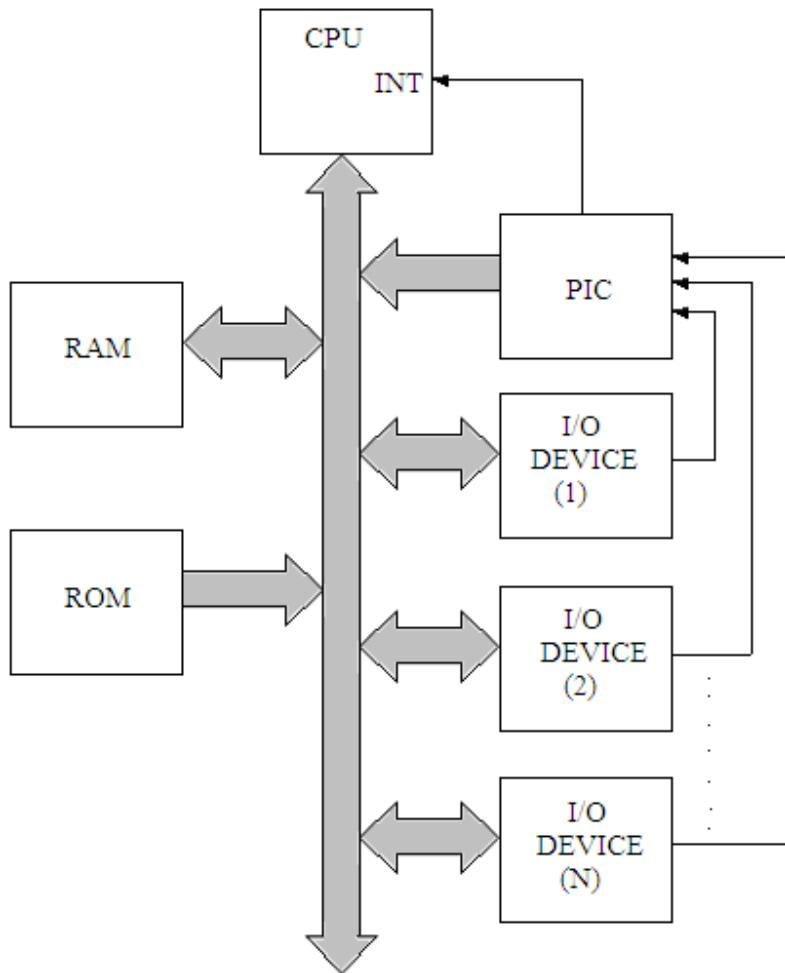
## Programmable Interrupt Controller: 8259

---

This chapter will confine to the detailed discussion on Programmable Interrupt Controller (PIC) – 8259. This device is also called **priority interrupt controller**. It is designed to work with Intel 8080A, 8085A, 8086 and 8088 microprocessors. It works as an overall manager in an interrupt driven system environment. This device can **handle 8 external interrupts and the starting address** of the interrupts service routine can be vectored to any location in the memory map unlike the software and hardware interrupts which point to the predetermined starting address. The 8259 can be set to accept level triggered or edge triggered interrupts. The interrupt can be expanded to 64 interrupt inputs by cascading many 8259 device.

### 11.1 PROGRAMMABLE INTERRUPT CONTROLLER 8259

In an earlier chapter it has been discussed that the 8085A has four hardware interrupt terminals namely TRAP, RST 5.5, RST 6.5 and RST 7.5. When the I/O device sends an interrupt signal to the CPU, the CPU completes the current instruction and branches to the service routine of the interrupt. After the execution of the ISR, it returns to the main program. In addition to these inputs, the CPU can also be interrupted through its INTR input. When an interrupt input signal INTR is send to the microprocessor, the CPU then send an interrupt acknowledge signal INTA to the external device. In response to this acknowledge signal the op code of the CALL instruction is placed on the data bus. The CPU reads the op code and sends another INTA signal. This signal is used to place the low order eight bits address of the CALL instruction on to the data bus. After reading the low order address the CPU sends another INTA signal. It then places the high order eight bit address on to the data bus. The CPU then reads and executes the interrupt service routine available in that CALL address. The number of I/O devices that may be used in an interrupt driven environment may be made greater than four if an external device is used. The external device should be capable of accepting interrupt requests and generating unique CALL instructions for the different interrupt inputs. This device will be used to interrupt the microprocessor on the INTR line. A programmable interrupt controller (PIC) is such a device (ref. figure 11.1) which accepts interrupt requests from a number of I/O devices, resolves the priority of servicing the requests, and issues an interrupt on the INTR input of the 8085A as discussed above.



**Fig. 11.1**

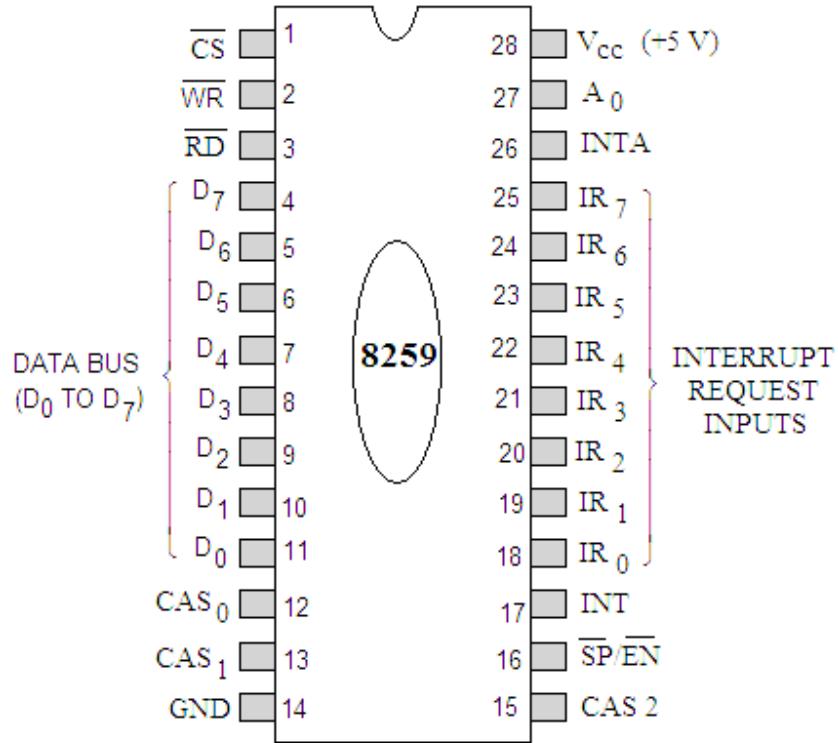
## 11.2 BLOCK DIAGRAM OF 8259

The 8259 is a programmable interrupt controller which uses NMOS technology. It is available in **28 pin plastic dual in line package (DIP)**. It requires one power supply of **+5 V** but does not require any internal or external clock. A single PIC can accept interrupt requests from **eight I/O devices**, resolve priority among them and communicate to the microprocessor. Interrupt requests from all the I/O devices can individually be masked and a suitable priority mode can be selected by programming. Built in expandability has also been provided to cascade **9 such Programmable Interrupt Controller devices (8259s)** to serve up to **64 I/O devices**. Figure 11.2 shows the pin diagram of this IC.

The description of the pins of 8259 programmable interrupt controller is given as:

Pin No. 1: This pin is chip select terminal ( $\overline{CS}$ ). It is active low. When a low signal is applied to this terminal, the device is chosen to work.

Pin No. 2: This is write input terminal ( $\overline{WR}$ ), which is also active low. A low to this input enables the CPU to write Initialization Control Word (ICW) and Operation Command Word (OCW) to the 8259A.



**Fig. 11.2**

Pin No. 3: This is read input terminal ( $\overline{RD}$ ). A low to this pin enables the 8259A to send the status of the Interrupt Request Register (IRR), In Service Register (ISR), the Interrupt Mask Register (IMR) or the BCD of the interrupt level on the data bus.

Pin Nos. 4-11: These pins (D<sub>0</sub> to D<sub>7</sub>) form the bidirectional data bus to be connected to the data bus of the system.

Pin Nos. 12, 13 &15: These pins (CS<sub>0</sub> to CS<sub>2</sub>) are known as cascade lines. These lines are used to cascade a number of 8259A.

Pin No. 14: This is a common ground terminal to be connected to the common terminal of the system or the supply.

Pin No. 16: This is slave program/enable buffer ( $\overline{SP}/\overline{EN}$ ). As already discussed more than one 8259A can be used with the system to expand the priority interrupt schemes up to 64 levels. In such cases one 8259A acts as the master and the others act as slaves. A high on this pin designates the 8259A as the master and a low to this pin designates as slave.

- Pin No. 17: This is an interrupt output terminal (INT) to be directly connected to the interrupt terminal (INTR) of the CPU. When an interrupt signals is applied to the any of the interrupt request terminals of the 8259, an INT signal is generated for the CPU.
- Pin Nos. 18 to 25: These are 8 interrupt request inputs ( $IR_0$  to  $IR_7$ ).
- Pin No. 26: This is an interrupt acknowledge input signal ( $\overline{INTA}$ ), connected to the  $\overline{INTA}$  terminal of the CPU. A low acknowledgement signal is sent by the CPU to the 8259, whenever CPU receives an interrupt signal.
- Pin No.27: This pin  $A_0$  is the input signal used in conjunction with the signals  $\overline{WR}$  and  $\overline{RD}$  to write the commands into the various status registers of the device. This terminal can directly be connected to one of the address lines.
- Pin No. 28: This is supply pin ( $+V_{CC}$ ) connected to the positive 5 volts.

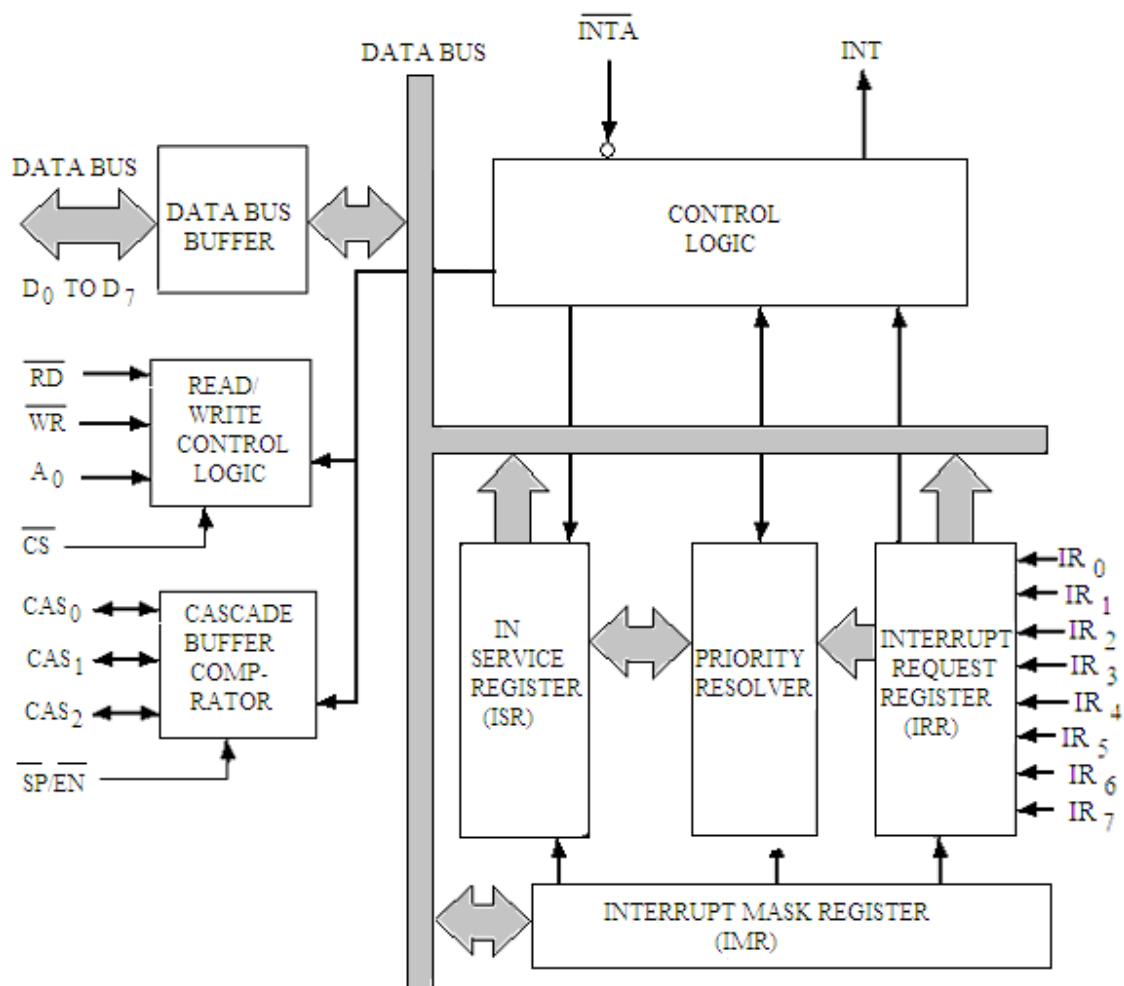


Fig. 11.3

Figure 11.3 shows the internal block diagram of the 8259A. The 8259A contains the following four sections:

1. Interrupt and Control Logic Section
2. Data Bus Buffer
3. Read/Write Control Logic Section
4. Cascade Buffer/Comparator Section

## **1. Interrupt and Control Logic Section**

This section contains the following five sections:

- **Interrupt Request Register (IRR)**
- **In-Service Register (ISR)**
- **Priority Resolver**
- **Interrupt Mask Register (IMR)**
- **Control Logic Section**

### **Interrupt Request Register (IRR)**

The interrupt request register (IRR) is used to store all the interrupt levels which are requesting services. It has **8-interrupt lines (IR<sub>0</sub> to IR<sub>7</sub>)**. When any of these lines become high, the corresponding mask bit is checked and if it enabled, then the corresponding bit in the interrupt request register (IRR) is set.

### **In-Service Register (ISR)**

The in-service register (ISR) is used to store information of all the interrupt levels which are currently being serviced.

### **Priority Resolver**

This block of Interrupt and Control Logic Section determines the priorities of the bits set in the IRR. This block determines the priorities as dictated by the priority mode set by the **Operation Command Words (OCWs)**. The bit corresponding to the highest priority interrupt is set in the ISR during the **INTA** input. It is a priority interrupt controller, this means, even while executing an interrupt, it will accept and service a higher priority; but it will reject a lower priority interrupt. The priority resolver does the job of judging whether to allow another interrupt to be executed in the middle of executing one interrupt service routine.

### **Interrupt Mask Register (IMR)**

The interrupt mask register (IMR) stores the bits of the interrupt lines to be masked. The IMR operates on the ISR. In fact this register can be programmed by an operation command word (OCW) to store the bits of the interrupt lines to be masked. An interrupt which is masked by software will not be recognized and serviced even if it sets the corresponding bit in the IRR.

### **Control Logic Section**

After the interrupt request priorities are resolved by the priority resolver, this block control logic sends an interrupt signal through its *INT* signal to the CPU. This terminal *INT* is connected to the *INTR* terminal of the microprocessor. The microprocessor responds to this request by putting an interrupt acknowledge signal *INTA* to the input terminal *INTA* of the 8259A. The PIC then places the op code of CALL instruction on the data bus. This is read by CPU, it places two additional *INTA* signals on the data bus. When the CPU receives the *INTA* signal out of these two additional *INTA* signals, it places the low order byte of the CALL address on the data bus. After receiving the last *INTA* signal, it places the high order byte of the CALL address on the data bus. The CALL address is the vector memory location for the interrupt; this address is placed in the control register during the initialization of 8259.

## 2. Data Bus Buffer

This is a three state bidirectional 8-bit data bus buffer is used to interface the 8259A to the system data bus. The control words and status information are transferred through this data bus buffer.

## 3. Read/Write Control Logic Section

The function of this Read/Write Control Logic section is to accept the commands from the CPU. It contains the initialization command word (ICW) registers and the operation command word (OCW) registers which store the various control formats for device operation. This section also accepts Read commands from the CPU to allow the CPU to read status words. The four pins are connected to this block whose functions are given below:

### **CS (Chip Select)**

This pin is chip select terminal, which is active low. A low signal to this terminal selects this device to work.

### **WR (Write)**

This is write input terminal, which is also active low. A low to this input enables the CPU to write Initialization Control Word (ICW) and Operation Command Word (OCW) to the 8259A.

### **RD (Read)**

A low to this pin enables the 8259A to send the status of the Interrupt Request Register (IRR), In Service Register (ISR), the Interrupt Mask Register (IMR) or the BCD of the interrupt level on the data bus.

### **A<sub>0</sub>**

This is the input signal used in conjunction with the signals *WR* and *RD* to write the commands into the various status registers of the device. This terminal can directly be connected to one of the address lines.

## 4. Cascade Buffer/Comparator Section

It is well known that more than one 8259A can be used with the system to expand the priority interrupt schemes up to 64 levels. In such cases one 8259A acts as the master and the others act as slaves. The necessary control signals for cascade operations are generated with this block. A high on the Slave Program/Enable Buffer (*SP/EN*) pin designates the 8259A as the master and a low to this pin designates as slave. The 8259A

can be set as master or a slave by the  $\overline{SP}/\overline{EN}$  pin in the non-buffer mode, or by software in buffer mode of operation. In the non-buffer mode  $\overline{SP}/\overline{EN}$  pin is used as an output to enable the data bus buffer of the system. In addition to  $\overline{SP}/\overline{EN}$  pin, there are three more pins ( $CAS_0 - CAS_2$ ) associated with the cascade buffer/comparator section, whose functions are described below:

For a master, these pins ( $CAS_0 - CAS_2$ ) are outputs, and for slave these are input pins. When 8259 is a master, the CALL op code is generated by the master in response to first  $INTA$  signal. The address for vector locations will be released by the slave 8259. Every 8259 has the identification code of three bits to  $CAS_0 - CAS_2$  lines so that one out of eight possible slave may be selected by the master. Basically, the slave 8259s accept the identification signals as inputs and compare the code put out by the master with code assigned to them during initialization. The slave thus selected then puts out the address of the interrupt service routine during the two additional  $INTA$  signals sent by CPU.

### 11.3 INTERFACING OF 8259A WITH 8085A

Figure 11.4 shows the interfacing connection of 8259A with 8085A microprocessor in I/O mapped I/O mode. The output of address decoder is connected to

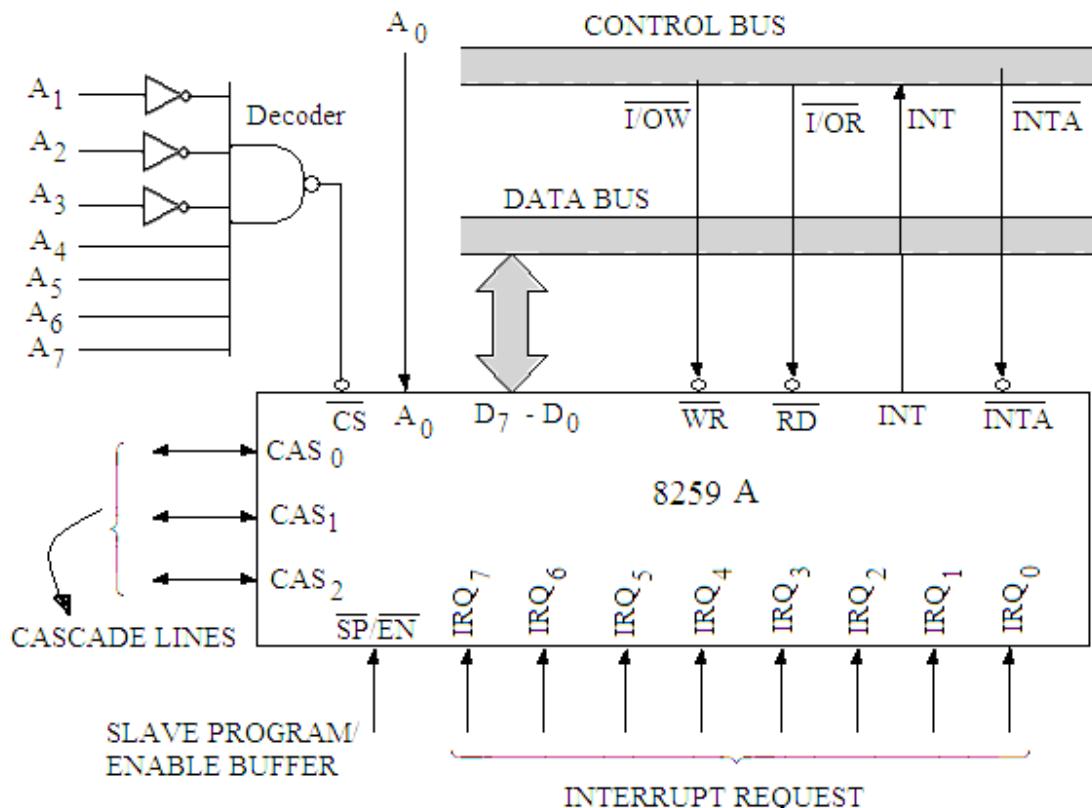


Fig. 11.4

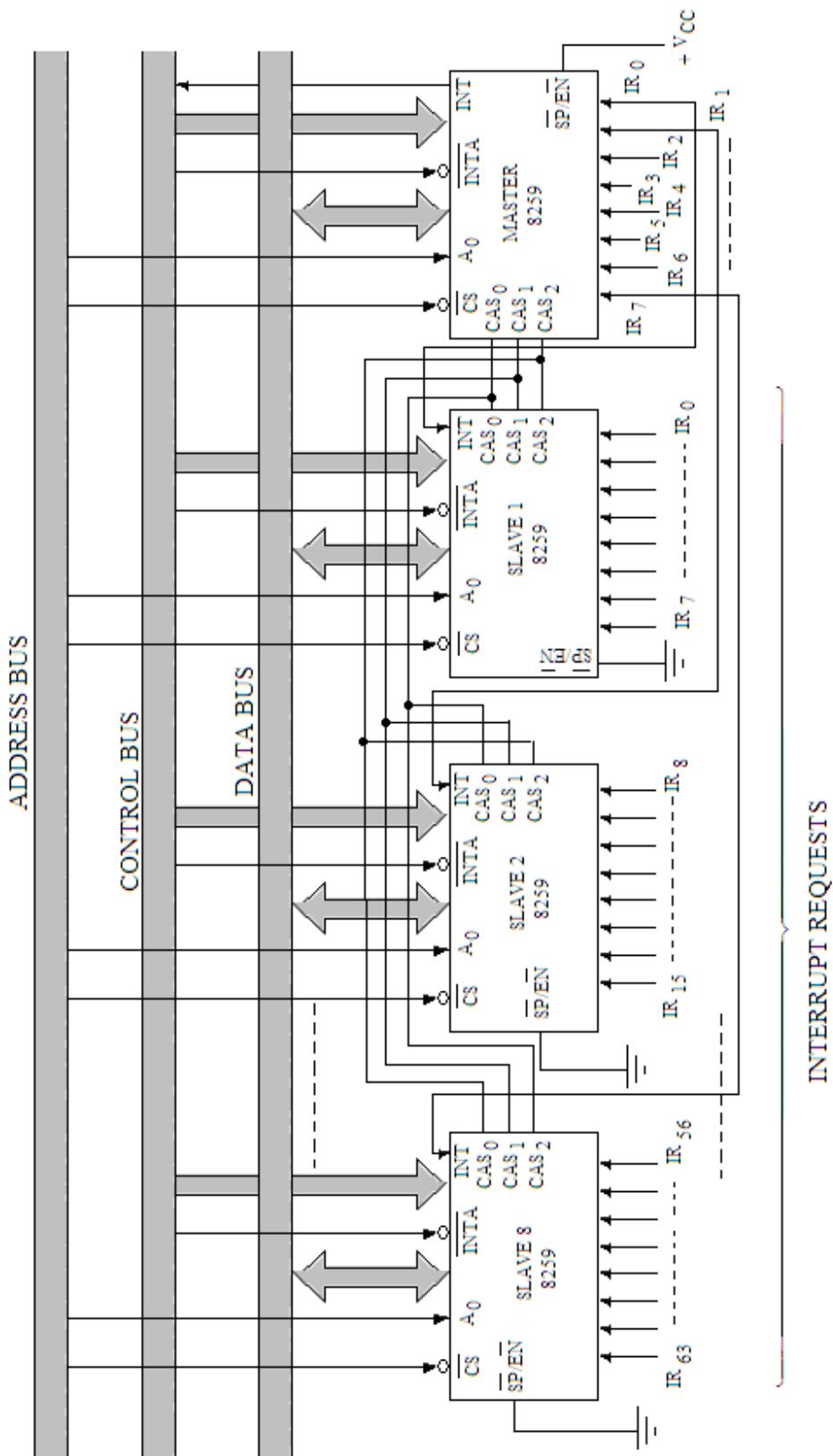


Fig.11.5

the  $\overline{CS}$  input of the 8259A. The  $A_0$  line of the IC is directly connected to the address lines of the system. The  $\overline{RD}$  and  $\overline{WR}$  signals of the IC are connected to  $I/OR$  and  $I/OW$  terminals respectively. Further,  $INT$  terminal of the 8259 is connected to the  $INTR$  terminal of the processor. The  $\overline{INTA}$  output of the processor is connected to the  $\overline{INTA}$  terminal of the processor. The terminal  $\overline{SP/EN}$  is connected to +5 V supply since only one 8259 is used in the system. If many 8259s are cascaded, then this terminal is connected to the ground terminal. The cascaded pins ( $CAS_0 - CAS_2$ ) are left open. The eight interrupt request pins  $IR_0 - IR_7$  are available for the I/O devices to send the interrupt signals. If any of these interrupt request pins are not used, they may be connected to the ground so that the noise pulse can not create any interrupt.

The cascading of many 8259s with the system is shown in figure 11.5. The  $\overline{SP/EN}$  terminal of the master PIC is connected to + VCC, whereas  $\overline{SP/EN}$  terminals of all the slave PICs are connected to ground. The  $INT$  output of each slave PICs are connected to one of the interrupt lines ( $IR_0 - IR_7$ ) of the master 8259 i.e. 8 INT terminals of 8 slave 8259s are connected to 8 interrupt request lines ( $IR_0 - IR_7$ ). When interrupt request line of a slave is activated, the slave PIC in turn activates one of the interrupt request lines of the master 8259, which in turn interrupts the microprocessor. After the  $\overline{INTA}$  is received from the microprocessor, the master enables the corresponding  $CAS_0 - CAS_2$  lines to release the vector address on the data bus in the next two  $\overline{INTA}$  signals.

The output of the address decoder is connected to the  $\overline{CS}$  of the 8259 PIC as shown in figure 11.4. It communicates with the CPU with the bidirectional bus. From this figure 11.4, it is clear that two I/O port addresses F0 H and F1 H are chosen for the 8259 with  $A_0 = 0$  and  $A_0 = 1$ . It may be noted here that each 8259A (interfaced with 8085A) has its own I/O address which is defined by the address decoder.

#### 11.4 VECTORING DATA FORMATS FOR 8259

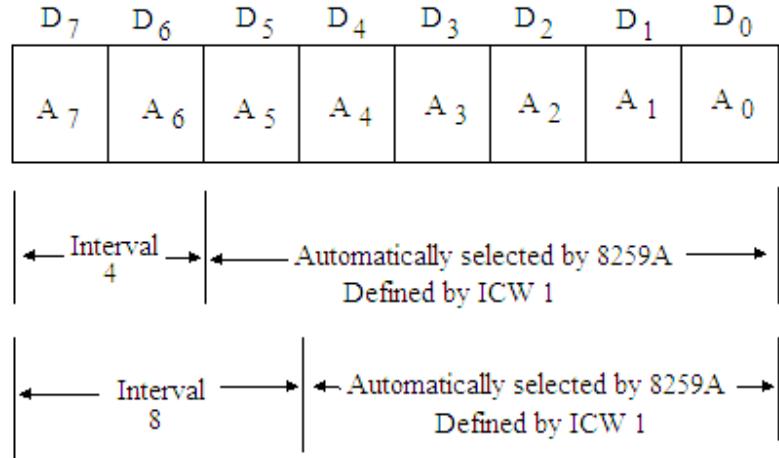
The eight interrupt levels generate CALLs to eight equally spaced locations in the memory. Through the programming, these locations can be spaced at an interval of 4 or 8 locations. The format of a byte released in response to the first  $\overline{INTA}$  for any IR levels is shown in figure 11.6.

| $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ | CALL CODE<br>= (CD H) |
|-------|-------|-------|-------|-------|-------|-------|-------|-----------------------|
| 1     | 1     | 0     | 0     | 1     | 1     | 0     | 1     |                       |

Fig. 11.6

It is the op code for the CALL instruction (CD H). The format of the byte in response to the second  $\overline{INTA}$  for different IR level is shown in figure 11.7, the details of which are given in table 11.1(a) and (b). These tables show for both 4 and 8 interval spacing. For spacing interval of four, the 8259 automatically inserts the bits  $D_0 - D_4$  and  $D_5 - D_7$  as specified while programming the 8259 through ICW 1 (first initialization command word, which will be discussed in the next section). For this interval bits  $D_4 - D_2$  specify the interrupt's number and the rest two bits  $D_1 - D_0$  have been set to 00. However,

for spacing interval of eight, the 8259 automatically inserts the bits D<sub>0</sub> – D<sub>5</sub> and D<sub>6</sub> – D<sub>7</sub> as specified while programming the 8259 through ICW 1. For the interval of 8, the bits D<sub>5</sub>-D<sub>3</sub> specify the interrupt's number and the rest two bits D<sub>2</sub>-D<sub>0</sub> have been set to 000.



**Fig. 11.7**

**Table 11.1(a)**

| IR | INTERVAL = 4   |                |                |                |                |                |                |                |
|----|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
|    | D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |
| 7  | A <sub>7</sub> | A <sub>6</sub> | A <sub>5</sub> | 1              | 1              | 1              | 0              | 0              |
| 6  | A <sub>7</sub> | A <sub>6</sub> | A <sub>5</sub> | 1              | 1              | 0              | 0              | 0              |
| 5  | A <sub>7</sub> | A <sub>6</sub> | A <sub>5</sub> | 1              | 0              | 1              | 0              | 0              |
| 4  | A <sub>7</sub> | A <sub>6</sub> | A <sub>5</sub> | 1              | 0              | 0              | 0              | 0              |
| 3  | A <sub>7</sub> | A <sub>6</sub> | A <sub>5</sub> | 0              | 1              | 1              | 0              | 0              |
| 2  | A <sub>7</sub> | A <sub>6</sub> | A <sub>5</sub> | 0              | 1              | 0              | 0              | 0              |
| 1  | A <sub>7</sub> | A <sub>6</sub> | A <sub>5</sub> | 0              | 0              | 1              | 0              | 0              |
| 0  | A <sub>7</sub> | A <sub>6</sub> | A <sub>5</sub> | 0              | 0              | 0              | 0              | 0              |

**Table 11.1(b)**

| IR | INTERVAL = 8   |                |                |                |                |                |                |                |
|----|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
|    | D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |
| 7  | A <sub>7</sub> | A <sub>6</sub> | 1              | 1              | 1              | 0              | 0              | 0              |
| 6  | A <sub>7</sub> | A <sub>6</sub> | 1              | 1              | 0              | 0              | 0              | 0              |
| 5  | A <sub>7</sub> | A <sub>6</sub> | 1              | 0              | 1              | 0              | 0              | 0              |

|   |                |                |   |   |   |   |   |   |
|---|----------------|----------------|---|---|---|---|---|---|
| 4 | A <sub>7</sub> | A <sub>6</sub> | 1 | 0 | 0 | 0 | 0 | 0 |
| 3 | A <sub>7</sub> | A <sub>6</sub> | 0 | 1 | 1 | 0 | 0 | 0 |
| 2 | A <sub>7</sub> | A <sub>6</sub> | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | A <sub>7</sub> | A <sub>6</sub> | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | A <sub>7</sub> | A <sub>6</sub> | 0 | 0 | 0 | 0 | 0 | 0 |

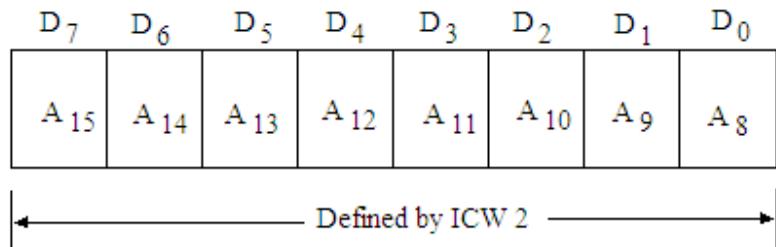
From this table it is clear that the interrupt requests IR<sub>2</sub> and IR<sub>4</sub> for an interval of eight is given by:

|                              | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |
|------------------------------|----------------|----------------|----------------|----------------|----------------|----------------|
| IR2 for an interval of 8 is  | 0              | 1              | 0              | 0              | 0              | 0              |
| IR 4 for an interval of 8 is | 1              | 0              | 0              | 0              | 0              | 0              |

Similarly these interrupt requests for an interval of 4 is:

|                              | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |
|------------------------------|----------------|----------------|----------------|----------------|----------------|----------------|
| IR2 for an interval of 4 is  | 0              | 1              | 0              | 0              | 0              | 0              |
| IR 4 for an interval of 4 is | 1              | 0              | 0              | 0              | 0              | 0              |

Figure 11.8 shows the format of the byte released by the 8259A in response to the third INTA. The bits D<sub>7</sub>-D<sub>0</sub> represent the high order byte of the interrupt vector address (A<sub>8</sub>-A<sub>15</sub>) of all the interrupts; which are specified in ICW 2.



**Fig. 11.8**

It may be noted here that the 8259A does not allow each interrupt input to have its own independent address. Instead, a base address can be programmed with four or eight interval spacing as discussed above. For example, if the base address is 0100 H and the interval spacing is four, then eight restart locations will exist as shown in table 11.2. Normally, instructions to jump to the appropriate interrupt service routine would be stored in these locations and the 32 bytes of memory from 0100 H to 011F H referred to as jump table.

**Table 11.2**

| Interrupt request | Restart address |
|-------------------|-----------------|
| IR0               | 0100 H          |
| IR1               | 0104 H          |
| IR2               | 0108 H          |
| IR3               | 010C H          |
| IR4               | 0110 H          |
| IR5               | 0114 H          |
| IR6               | 0118 H          |
| IR7               | 011C H          |

**Example 11.1.** For initialization of 8259 the address space available is from E010 H to E040 H. What should be the jump table, if interval spacing between each restart interrupt is four?

**Solution.** The address space available for the restart instruction is from E010 H to E040 H, so the restart address for the first interrupt request will start from E020 H, as the five bits of the starting address must be 00000 (ref. Table 11.1a). The jump table is therefore given in table 11.3.

**Table 11.3**

| Interrupt request | Restart address |
|-------------------|-----------------|
| IR0               | E020 H          |
| IR1               | E024 H          |
| IR2               | E028 H          |
| IR3               | E02C H          |
| IR4               | E030 H          |
| IR5               | E034 H          |
| IR6               | E038 H          |
| IR7               | E03C H          |

## 11.5 INITIALIZATION OF 8259

There are two types of command words in 8259A. These are:

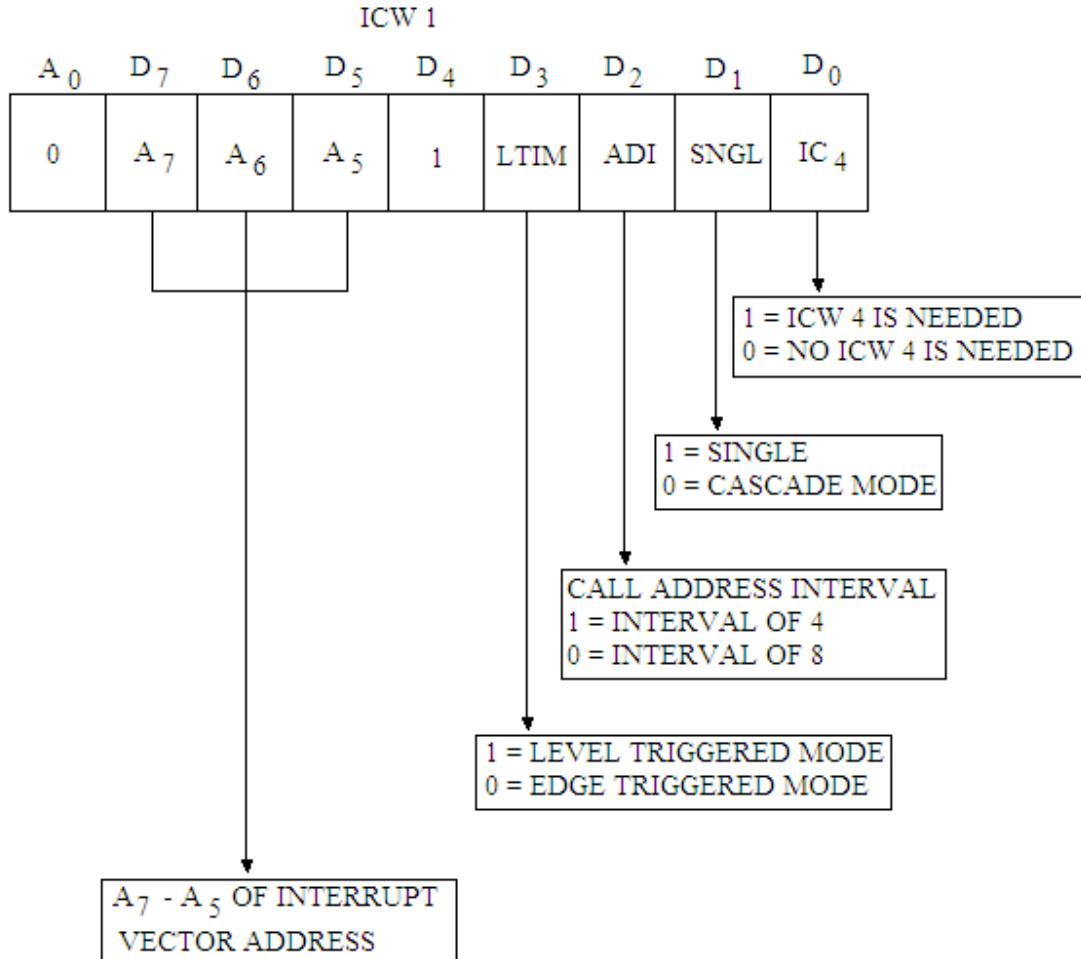
- Initialization Command Words (ICWs)
- Operation Command Words (OCWs)

## 11.6 INITIALIZATION COMMAND WORDS (ICWs)

There are four command words ICW 1, ICW 2, ICW 3 and ICW 4. The ICW 1 and ICW 2 commands are essential for any 8259A system. However, ICW 3 and ICW 4 commands are optional. The ICW 3 is used if the system has any slave 8259A. The ICW 4 command is used for special operations like special fully nested mode. The microprocessor programs the 8259A by loading the initialization command word (ICWs) and operation command words (OCWs). Each 8259A in the system is first initialized by loading a set of these ICWs in a sequence. The OCWs can be loaded any time after initialization.

### Initialization Command Word -1 (ICW 1)

The format for first initialization command word (ICW 1) is shown in figure 11.9. When a byte is sent to the 8259A with  $A_0 = 0$ , and  $D_4 = 1$ , it is interpreted as initialization



**Fig. 11.9**

command word-1 (ICW 1). The interpretations of the bits of ICW 1 command word are given below:

- Bit D<sub>0</sub>** This bit gives the information if the command word ICW 4 is needed.  
If D<sub>0</sub> = 1, ICW 4 is needed.  
If D<sub>0</sub> = 0, ICW 4 is not needed i.e. all functions selected by ICW 4 are cleared. This sets the 8259A in the 8085A mode without AEOI (automatic End of Interrupt) and non-buffered operation; and no ICW 4 would be sent with the initialization.
- Bit D<sub>1</sub>** This bit gives the information if the 8259A is to be used in single or cascaded mode.  
If D<sub>1</sub> = 1, single 8259A is used and no slave 8259A is used.  
If D<sub>1</sub> = 0, 8259A is used in cascaded mode.
- Bit D<sub>2</sub>** This bit gives the information about vector address.

If  $D_2 = 1$ , Interval for vector address is 4.

If  $D_2 = 0$ , Interval for vector address is 8.

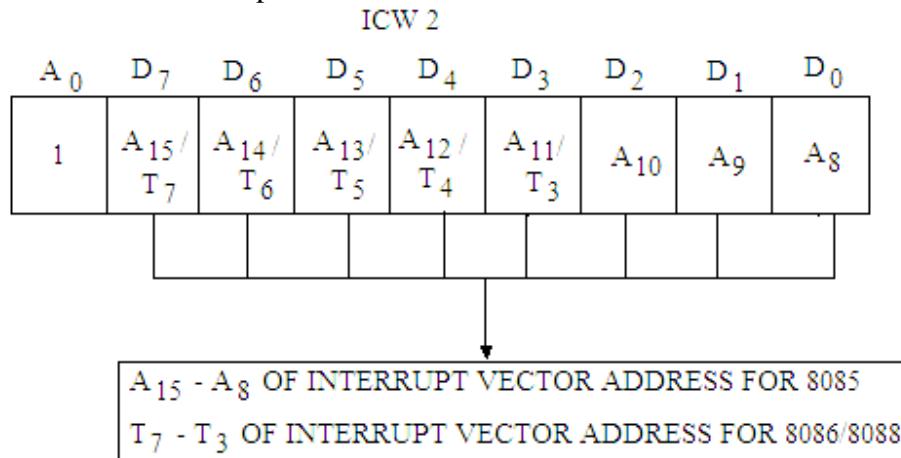
- Bit D<sub>3</sub>** The bit  $D_3$  gives the information about interrupt request inputs whether it is edge triggered or level triggered.  
 If  $D_3 = 1$ , Interrupt request input is level triggered.  
 If  $D_3 = 0$ , Interrupt request input is edge triggered.

- Bit D<sub>4</sub>** This bit is 1 for ICW 1, as discussed earlier.

- Bits D<sub>5</sub> to D<sub>7</sub>** These bits give the interrupt vector address (for 8085A only) i.e. these are address bits of the CALL instruction.

### Initialization Command Word -2 (ICW 2)

A write command following ICW 1, with  $A_0 = 0$ , is interpreted as ICW 2. The format for this command word is shown in figure 11.10. This gives information regarding Interrupt Vector Address. The bits D<sub>7</sub>-D<sub>0</sub> represent the high order byte of the interrupt vector address (A<sub>8</sub>-A<sub>15</sub>) for 8085 microprocessor. However, T<sub>7</sub>-T<sub>3</sub> represent the vector address for 8086/8088 microprocessors.



**Fig. 11.10**

### Initialization Sequence of 8259A

Once ICW1 is loaded, the following initialization procedure is carried out internally:

- The edge sense circuit is reset. Since 8259A interrupts are edge sensitive, they are reset.
- IMR is cleared.
- IR7 input is assigned the lowest priority.
- Slave mode address is set to 7.
- Special mask mode is cleared and status read is set to IRR.
- If IC<sub>4</sub> = 0, all functions of ICW4 are set to zero. Master/slave bit is ICW 4 is used in the buffered mode only.

**Example 11.2.** Initialize 8259A to interface eight level triggered inputs. The restart address of first interrupt request is E020 H. There is a spacing interval of four byte between each interrupt request. No cascading of 8259A is made. Let the port address is F0 H and F1 H for  $A_0 = 0$  and  $A_0 = 1$  respectively.

**Solution.** The single 8259A is to be initialized, so initialization command words ICW 1 and ICW 2 are to be written as given below:

### ICW 1

| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub>  |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|-----------------|--------|
| A <sub>7</sub> | A <sub>6</sub> | A <sub>5</sub> | 1              | LTM            | ADI            | SNGL           | IC <sub>4</sub> |        |
| 0              | 0              | 1              | 1              | 1              | 1              | 1              | 0               | = 3E H |

In ICW 1, D<sub>0</sub> = 0 as no ICW 4 is required; D<sub>1</sub> = 1 as single 8259 is used, D<sub>2</sub> = 1 as spacing interval of 4 is required, D<sub>3</sub> = 1 as these are level triggered inputs, D<sub>7</sub> D<sub>6</sub> D<sub>5</sub> are 001 as 20 H is the address of the first interrupt request (0010 0000).

### ICW 2

| D <sub>7</sub>  | D <sub>6</sub>  | D <sub>5</sub>  | D <sub>4</sub>  | D <sub>3</sub>  | D <sub>2</sub>  | D <sub>1</sub> | D <sub>0</sub> |        |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|----------------|----------------|--------|
| A <sub>15</sub> | A <sub>14</sub> | A <sub>13</sub> | A <sub>12</sub> | A <sub>11</sub> | A <sub>10</sub> | A <sub>9</sub> | A <sub>8</sub> |        |
| 1               | 1               | 1               | 0               | 0               | 0               | 0              | 0              | = E0 H |

ICW 2 gives the address E0 H (base address) 1110 0000.

Program:

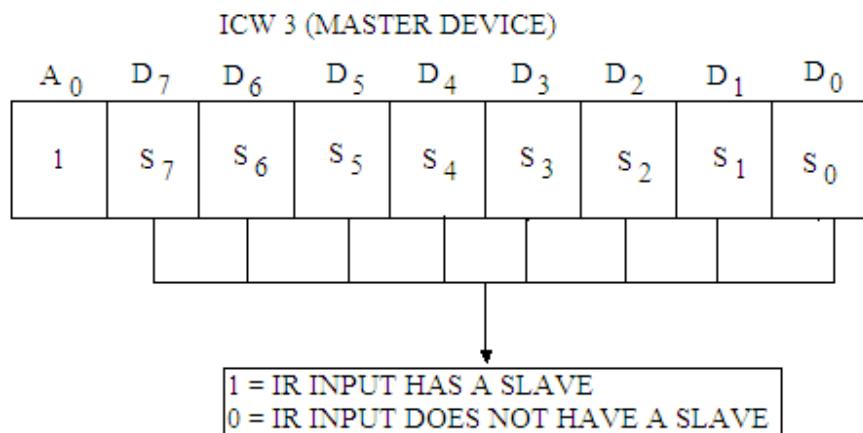
```
MVI A, 3E H      ; ICW 1 code is loaded to acc.
OUT F0 H        ; F0 H is the port address for A0 = 0.
MVI A, E0 H      ; Base address is given by ICW 2.
OUT F1 H        ; ICW 2 port address for A0 = 1.
```

### Initialization Command Word -3 (ICW 3)

As already discussed, ICW 1 and ICW 2 are compulsory command words in initialized sequence of 8259A where as ICW 3 and ICW 4 are optional. The command word ICW 3 is read only when cascading of 8259s is used, which is denoted by SNGL (Bit D<sub>1</sub>) of ICW 1. The ICW 3 is further classified into following two modes:

Master Mode ICW 3

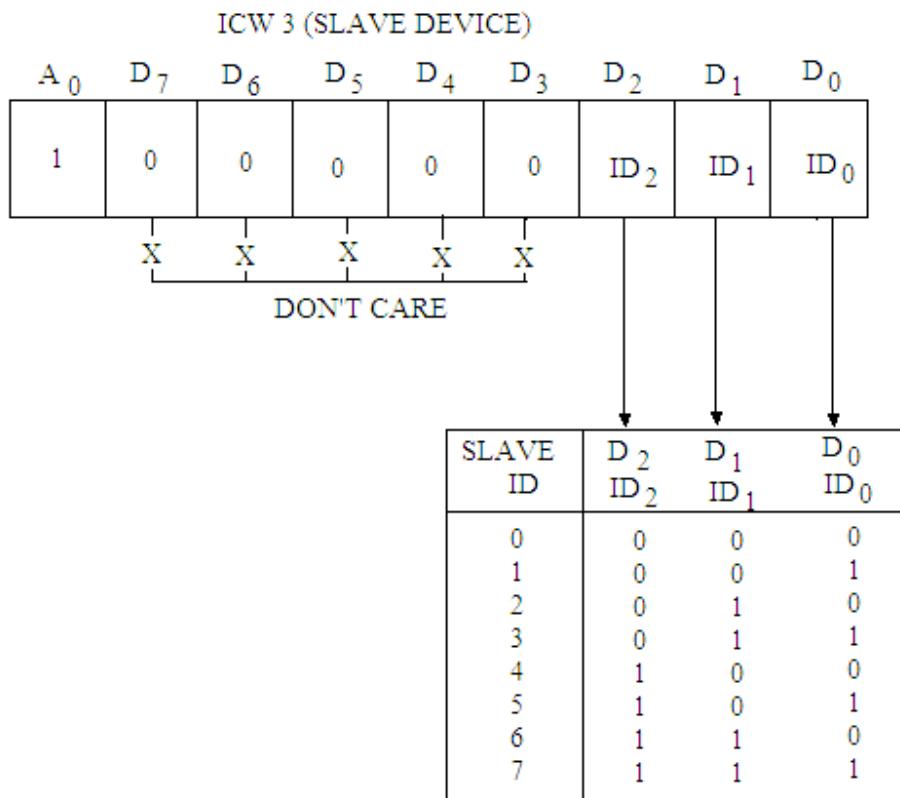
Slave Mode ICW 3.



**Fig. 11.11**

The ICW 3 in Master and Slave modes are shown in figure 11.11 and 11.12 respectively. Each bit in ICW 3 of master mode is used to specify whether it has a slave 8259 attached to its corresponding interrupt request (IRQ) inputs. Suppose  $S_0$  bit is 1. It indicates that 8259 is connected to interrupt request lines  $IRQ_0$ . The command word will be loaded to the 8 bit register and the functions of this register are:

- In the master mode (either when  $\overline{SP}/\overline{EN}$  is high, or in buffered mode when  $M/S = 1$  in ICW 4) a 1 set for each slave in the system. The master then will release byte 1 of the CALL sequence and will enable the corresponding slave to release byte 2 and byte 3 through the cascaded lines.
- In the slave mode (either when  $\overline{SP}/\overline{EN}$  is low or if  $BUF = 1$  and  $M/S = 0$  in ICW 4) bits  $D_2-D_0$  identify the slave. The slave compares its cascade inputs with these bits and if they are equal, bytes 2 and 3 of the CALL sequence are released by it on the data bus.



**Fig. 11.12**

#### Initialization Command Word -4 (ICW 4)

This command word is loaded only if  $IC_4$  bit of the command word ICW 1 is 1. That is if  $IC_4 = 1$  (of ICW 1) then command word -4 (ICW 4) is used otherwise it is

neglected. The format of this command word is shown in figure 11.13. The bits of this command word identify the following operations:

- $\mu$ PM**      This specifies if 8080/8085A or 8086/8088 operation environment is used.  
If  $\mu$ PM = 1, 8086/8088 microprocessor operation is selected.  
If  $\mu$ PM = 0, then the 8080/8085A operation is selected.
- AEOI**      If this bit is 1, then the automatic end of interrupt mode is selected, otherwise normal end of interrupt mode.
- M/S**      If M/S = 1, then 8259A is master.  
If M/S = 0, then 8259A is slave.  
If BUF = 0, then M/S bit is neglected.

ICW 4

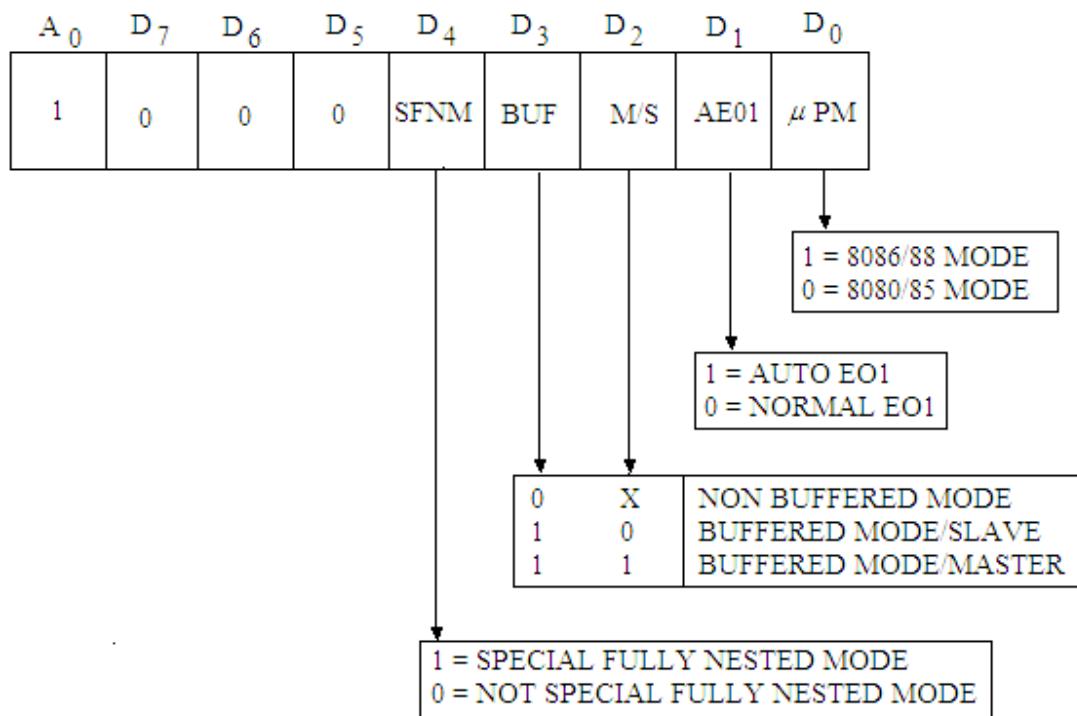
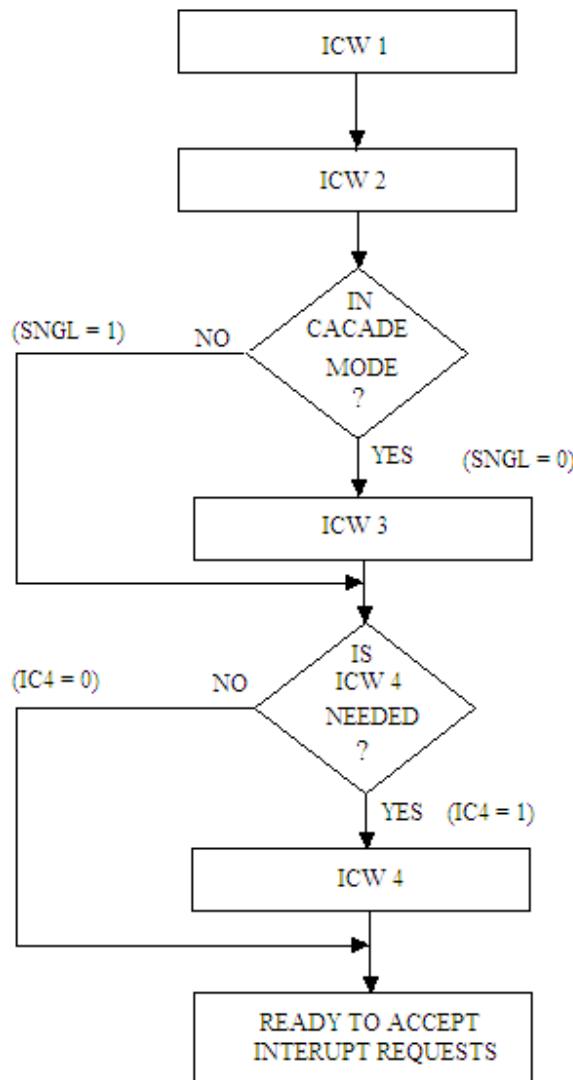


Fig. 11.13

- BUF**      If BUF = 1, then buffered mode is selected. In buffered mode  $\overline{SP}/\overline{EN}$  acts as enable output and Master/Slave is determined by M/S bit of ISW 4.
- SFNM**      If SFNM = 1, then fully nested mode is selected.  
If SFNM = 0, then fully nested mode is not selected.

Figure 11.14 shows the flow chart for the initialization of 8259A.



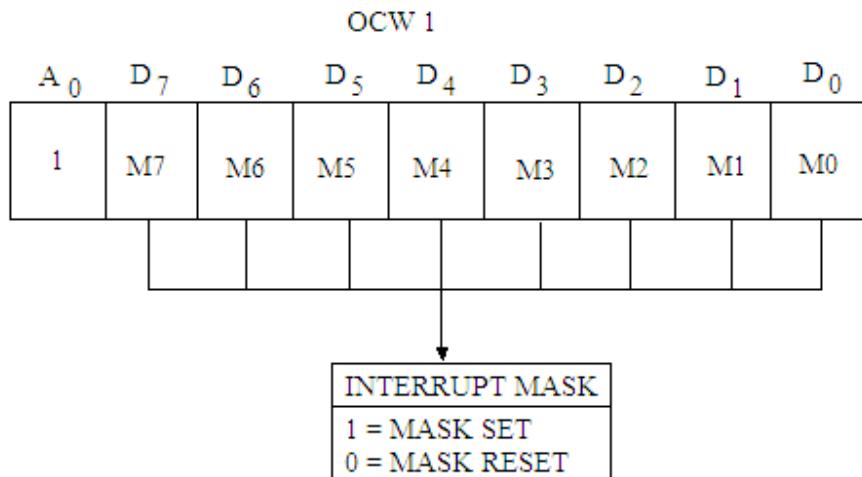
**Fig. 11.14**

### 11.7 OPERATION COMMAND WORDS (OCWs)

Once 8259A is initialized using initialization command words (ICWs), the 8259 is ready to accept interrupt requests. For this purpose, different operation command words (OCWs) are loaded into operation command word registers. There are three operation command words (OCWs) namely: OCW 1, OCW 2 and OCW 3. Now the discussion will be made of the operation command words.

#### Operation Command Word-1 (OCW 1)

This operation command word is written to mask or unmask an interrupt request input. The format for OCW 1 is shown in figure 11.15. For this word  $A_0$  must be high. In this format it is clear if  $M = 0$ , then the particular interrupt request is to be unmasked or enabled, however, if  $M = 1$ , it is masked or inhibited.



**Fig. 11.15**

For example, in order to enable the interrupts IR6, IR4 and IR1 then the bit pattern of this operation command word will be:

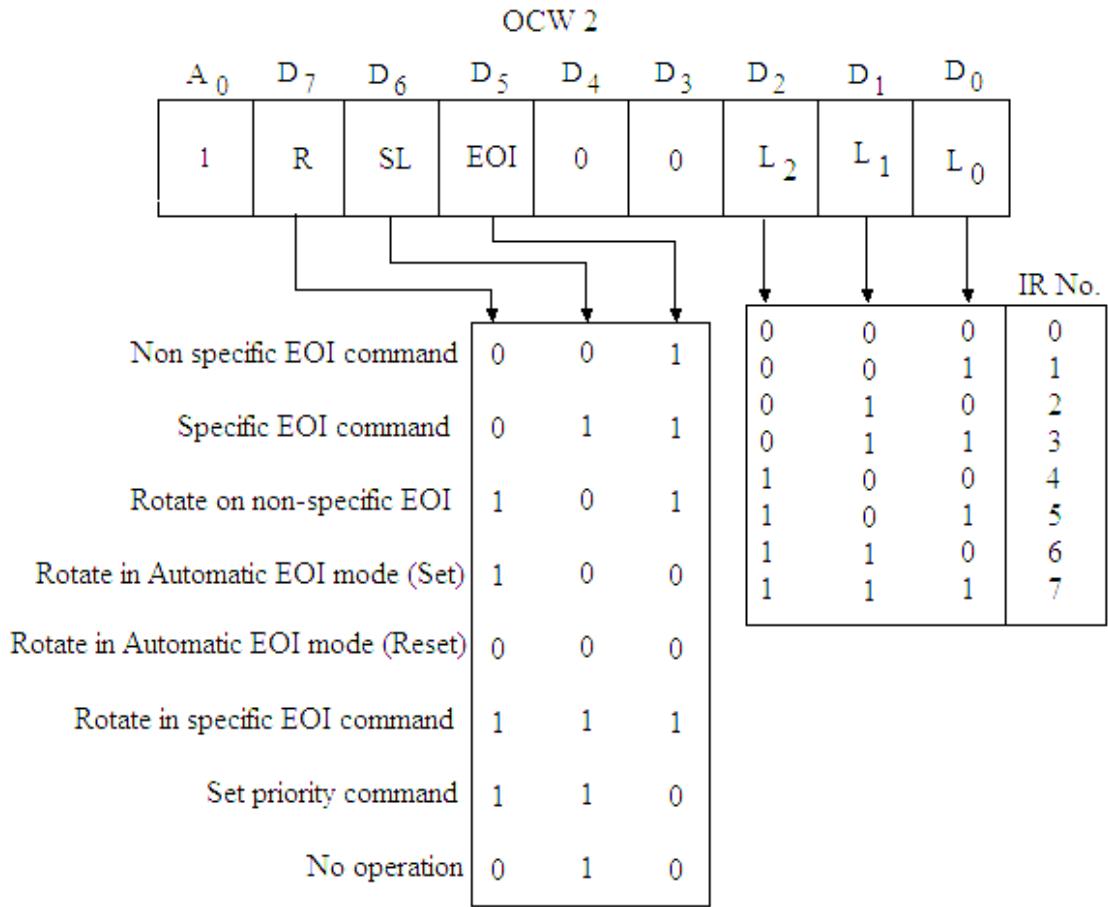
|       |       |       |       |       |       |       |       |
|-------|-------|-------|-------|-------|-------|-------|-------|
| $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
| 0     | 1     | 0     | 1     | 0     | 0     | 1     | 0     |

### Operation Command Word-2 (OCW 2)

A write command with  $A_0 = 0$  and  $D_4 D_3 = 00$  is interpreted as OCW 2, which is used to control the Rotate and End of interrupt mode. This is also called End of Interrupt Command. The OCW 2 is mainly used to reset a bit in the in service register (ISR). The format for this operation command word is shown in figure 11.16.

R, SL and EOI – bits control the Rotate, End of interrupt modes and the combination of two. The bits  $L_2$ ,  $L_1$  and  $L_0$  determine the interrupt level acted upon when the SL bit is active.

When an interrupt request is acknowledged, the 8259A determines the interrupt of highest priority and sets its corresponding bit in the ISR (in service register). The vector address corresponding to this interrupt is then put out. The bit in ISR remains set until an End of Interrupt (EOI) command through OCW 2 is issued by the CPU. At the end of the service routine, the corresponding bit in ISR is to be reset; otherwise other priority interrupts will not interrupt. The 8259A can respond to the interrupt signal of lower priority once the bit of the current IR bit in ISR is reset. The operation command word-2 (OCW 2) can issue the End of Interrupt (EOI) in the following three forms:



**Fig. 11.16**

### Non Specific EOI Command

In fully nested mode, the highest level in the ISR would necessarily correspond to the last interrupt acknowledged and services. In such a case, a non-specific end of interrupt command (EOI) may be issued by the CPU by the OUT instruction to the 8259A with  $A_0 = 0$ . For this bit pattern in the format of OCW 2 is given by:

|       |       |       |       |       |       |       |       |
|-------|-------|-------|-------|-------|-------|-------|-------|
| $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
| 0     | 0     | 1     | 0     | 0     | 0     | 0     | 0     |

The EOI bit ( $D_5$ ) of the operation command word-2 (OCW 2) is set to 1. This resets the highest priority in service bit of ISR.

### Specific EOI Command

If the fully nested mode is not used, the 8259 may not be able to determine the last interrupt acknowledged. In such case, a specific EOI command will have to be issued by the CPU by the OUT instruction to the 8259A with  $A_0 = 0$ . For this bit pattern in the format of OCW 2 is given by:

|       |       |       |       |       |       |       |       |
|-------|-------|-------|-------|-------|-------|-------|-------|
| $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
| 0     | 1     | 1     | 0     | 0     | $L_2$ | $L_1$ | $L_0$ |

In this bit pattern SL and EOI bits must be set to 1 and the encoded three bit value of the specific bit is written for  $L_2, L_1, L_0$ .

For example, in order to reset the bit 3 of the in service register corresponding to IR3, then the bit pattern of the OCW 2 will be given by:

|                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |
| 0              | 1              | 1              | 0              | 1              | 0              | 0              | 0              |

= 68 H

This operation command OCW 2 must be issued twice in case of cascaded mode; one for master and the other for slave.

#### Automatic End of Interrupt (AEOI)

If this mode is set, no command is necessary to reset the highest priority in service bit in the ISR. The bit in ISR will be reset automatically at the trailing edge of the third INTA signal sent by the CPU. This AEOI mode can only be used for a master 8259A and not for a slave. The AEOI is actuated by writing ICW 4 with D<sub>1</sub> bit to 1. At the time of writing the ICW 4, the A<sub>0</sub> line should be 1.

#### Operation Command Word-3 (OCW 3)

The format for OCW 3 is shown in figure 11.17. This word is used to read the status of the In Service Register (ISR) and Interrupt Request Register (IRR); and to set or reset the special mask and polled modes.

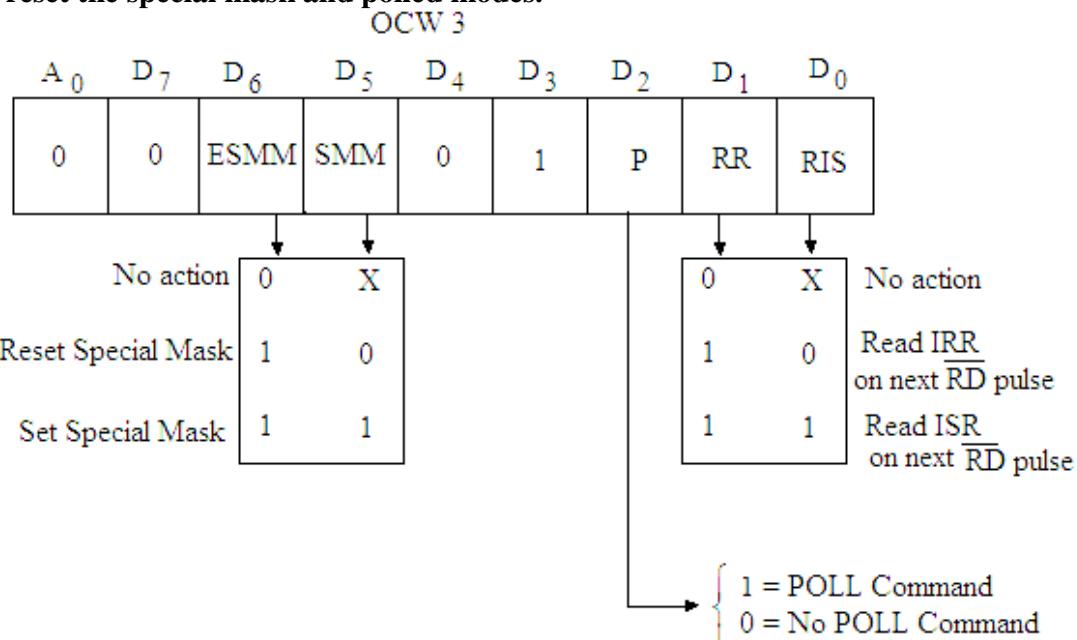


Fig. 11.17

### 11.8 INTERRUPT MODES OF 8259A

There are following modes of operations of 8259A:

- Fully Nested Mode
- Rotating Priority Mode
- Special Mask Mode
- Polled Mode

#### 11.8.1 Fully Nested Mode (FNM)

The fully nested mode is the general purpose mode in which all interrupt requests are arranged from the highest to lowest priority level. In general, IR<sub>0</sub> has the highest

priority and  $IR_7$  has the lowest priority. This is the default mode setting after initialization. The 8259 continues to operate in FNM until the mode is changed by OCWs. As discussed above, when an interrupt request is acknowledged, the 8259 determines the highest priority interrupt and set its corresponding bit in in-service register (ISR). The vector address corresponding to this interrupt is then put out. The bit in ISR remains set until an End of Interrupt (EOI) command through OCW 2 is issued by the CPU. At the end of the service routine, the corresponding bit in ISR is reset using OCW2.

### 11.8.2 Rotating Priority Mode

Though by default the priorities of the inputs are assigned as  $IR_0$  as the highest priority and  $IR_7$  as the lowest priority. But the priorities may be changed using either by automatic rotation or by specific rotation.

#### Automatic Rotation

This mode is used when all the interrupts are assigned equal priority. In this mode, currently executing interrupt is given the lowest priority after completing its service routine. The next interrupt level is assigned the highest priority. For example, the interrupt request  $IR_4$  and  $IR_6$  are being serviced as it can be read from the status of ISR (figure 11.18). As usual  $IR_4$  is assigned highest priority and will be executed first. When the interrupt service routine for  $IR_4$  is executed, this bit will be reset after EOI command as discussed earlier. Now in this mode of automatic rotation the interrupt  $IR_4$  will have the lowest property and the next interrupt will have the highest priority as shown in figure 11.19.

( $IR_6$ ,  $IR_4$  Being serviced:  $IR_4$  is active)

|                 | IS7               | IS6 | IS5 | IS4 | IS3 | IS2 | IS1                | IS0 |
|-----------------|-------------------|-----|-----|-----|-----|-----|--------------------|-----|
| ISR Status      | 0                 | 1   | 0   | 1   | 0   | 0   | 0                  | 0   |
| Priority Status | 7                 | 6   | 5   | 4   | 3   | 2   | 1                  | 0   |
|                 | ↑ Lowest Priority |     |     |     |     |     | ↑ Highest Priority |     |

Before Rotation

Fig. 11.18

(After EOI from  $IR_4$  routine)

|                 | IS7            | IS6 | IS5 | IS4 | IS3 | IS2 | IS1 | IS0 |
|-----------------|----------------|-----|-----|-----|-----|-----|-----|-----|
| ISR Status      | 0              | 1   | 0   | 0   | 0   | 0   | 0   | 0   |
| Priority Status | 2              | 1   | 0   | 7   | 6   | 5   | 4   | 3   |
|                 | After Rotation |     |     |     |     |     |     |     |

Fig. 11.19

Automatic rotation mode on an AEOI is set by loading an OCW 2 with R = 1, SL = 0 and EOI = 0 and is reset by loading an OCW 2 with R = 0, SL = 0, and EOI = 0.

#### Specific Rotation

In this, the programmer can change the priority by programming the lowest priority. This mode is set by CPU by issuing an OUT instruction with A0 = 0 and other bit pattern as given below:

|                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |
| 1              | 1              | 0              | 0              | 0              | L <sub>2</sub> | L <sub>1</sub> | L <sub>0</sub> |

The bits D<sub>2</sub>-D<sub>0</sub> specify the interrupt level that is to be assigned lowest priority. The specific rotation can be accomplished by using the Rotation of specific EOI in OCW2 as R = 1, SL = 1 and EOI = 1.

#### 11.8.3 Special Mask Mode

It may sometimes be desirable to selectively enable lower priority interrupts. Usually, if an EOI command is not given to the 8259A, the in service bit of the last serviced interrupt is not reset. As a result all lower priority interrupts are kept disabled.

This special mask mode can be set by making ESMM and SMM bits as 1 in OCW 3. When a mask bit is set in OCW 1, all further interrupts at that level are inhibited; while interrupts on all other levels that are not masked are enabled. It is thus possible to selectively enable interrupts by programming the mask register. This mode can be cleared by loading an OCW 3 with ESSM = 1 and SMM = 0.

#### 11.8.4 Polled Mode

In this mode the INT output of 8259A is not used. It is either not connected to the INTR input of 8085A or the system interrupts are disabled by software. The 8259A assign the priorities IR<sub>0</sub> through IR<sub>7</sub> inputs and provides a status port that can be polled. The status byte has the form as shown in figure 11.20.

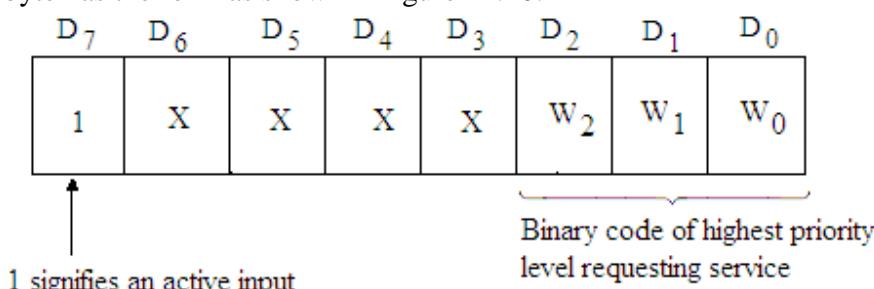


Fig. 11.20

#### 11.9 STATUS READ OPERATION OF 8259

The status of the Interrupt Request Register (IRR), the In-Service Register (ISR) and the Interrupt Mask Register (IMR) of the 8259 may be read by using appropriate read command. These read commands are described below:

##### Reading IRR Status

Just before reading the IRR (interrupt request register), OCW 3 must be written by making RR (read request) as 1 and RIS (read ISR) as 0. At the time of writing this operation command word, address line A<sub>0</sub> and read signal must be 0. Internally OCW 2 and OCW 3 are recognized by sensing the D<sub>4</sub> and D<sub>3</sub> bits as:

|                |                |
|----------------|----------------|
| D <sub>4</sub> | D <sub>3</sub> |
| 0              | 0              |

for OCW 2

1 0 for OCW 3.

After writing the OCW 3, the interrupt request status can be read by using the IN instruction.

IN F0 H if F0 H is the port address with  $A_0 = 0$ .

### Reading ISR Status

For reading ISR status, OCW 3 is initially written with RR = 1 and RIS = 1. The In-service register content can be read using the IN instruction with  $A_0 = 0$ .

IN F0 H if F0 H is the port address with  $A_0 = 0$ .

### Reading IMR Status

The contents of Interrupt Mask Register (IMR) can be read just by making the  $A_0 = 1$ . The OCW 3 is not written just before reading.

IN F1 H if F1 H is the port address with  $A_0 = 1$ .

**Example 11.3.** Assume that initialization command words have been written to 8259A (single) having the port address 20 H and 21 H for  $A_0 = 0$  and  $A_0 = 1$  respectively. Now write proper operation control words for a fully nested priority structure. Mask IR1, IR4 and IR6; enable ISR for a read. What code should be used at the end of interrupt service routine?

**Solution.** The initialization commands have already been written. The operation command words are written as:

#### OCW 1 $A_0 = 1$

|                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |
| M7             | M6             | M5             | M4             | M3             | M2             | M1             | M0             |
| 0              | 1              | 0              | 1              | 0              | 0              | 1              | 0              |

 $= 52 \text{ H}$ 

In this OCW 1, M1, M4 and M6 are 1 as IR1, IR4 and IR6 are masked.

For reading ISR status, OCW 3 is initially written with RR = 1 and RIS = 1.

#### OCW 3

|                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |
| 0              | 0              | ESMM           | SMM            | 1              | P              | RR             | RIS            |
| 0              | 0              | 0              | 0              | 1              | 0              | 1              | 1              |

 $= 0B \text{ H}$ 

In fully nested mode, the highest level in the ISR would necessarily correspond to the last interrupt acknowledged and services. In such a case, a non-specific end of interrupt command (EOI) may be issued by the CPU by the OUT instruction to the 8259A with  $A_0 = 0$ . For this bit pattern in the format of OCW 2 is given by:

|                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |
| 0              | 0              | 1              | 0              | 0              | 0              | 0              | 0              |

 $= 20 \text{ H}$ 

The EOI bit (D<sub>5</sub>) of the operation command word-2 (OCW 2) is set to 1. This resets the highest priority in service bit of ISR.

#### PROGRAM:

|             |  |
|-------------|--|
| MVI A, 52 H | ; OCW 1 is loaded to accumulator.                            |
| OUT 21 H    | ; 21 H is the port address for $A_0 = 1$ .                   |
| MVI A, 0B H | ; OCW 3 is loaded to accumulator to enable the ISR for read. |
| OUT 20 H    | ; 20 H is the port address for $A_0 = 0$ ,                   |
| IN 20 H     | ; Read ISR   |

Each interrupt service routine must end by giving the non-specific EOI command which resets in service bit (ISR bit).

```
MVI A, 20 H
OUT 20 H
RET
```

**Example 11.4.** Write Initializing command word to program 8259A (single) with port addresses C0 H and C1 H for  $A_0 = 0$  and  $A_0 = 1$  respectively. Let the starting address of the interrupt request is 7000 H with interval spacing of 4. The interrupts are edge triggered. Interrupts 1 and 3 are masked.

**Solution.** The single 8259A is to be initialized, so initialization command words ICW 1 and ICW 2 are to be written as given below:

#### ICW 1

|                |                |                |                |                |                |                |                 |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|-----------------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub>  |
| A <sub>7</sub> | A <sub>6</sub> | A <sub>5</sub> | 1              | LTM            | ADI            | SNGL           | IC <sub>4</sub> |
| 0              | 0              | 0              | 1              | 0              | 1              | 1              | 0               |

= 16 H

In ICW 1, D<sub>0</sub> = 0 as no ICW 4 is required; D<sub>1</sub> = 1 as single 8259 is used, D<sub>2</sub> = 1 as spacing interval of 4 is required, D<sub>3</sub> = 0 as these are edge triggered inputs, D<sub>7</sub> D<sub>6</sub> D<sub>5</sub> are 000 as 00 H is the address of the first interrupt request (0000 0000).

#### ICW 2

|                 |                 |                 |                 |                 |                 |                |                |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|----------------|----------------|
| D <sub>7</sub>  | D <sub>6</sub>  | D <sub>5</sub>  | D <sub>4</sub>  | D <sub>3</sub>  | D <sub>2</sub>  | D <sub>1</sub> | D <sub>0</sub> |
| A <sub>15</sub> | A <sub>14</sub> | A <sub>13</sub> | A <sub>12</sub> | A <sub>11</sub> | A <sub>10</sub> | A <sub>9</sub> | A <sub>8</sub> |
| 1               | 1               | 1               | 0               | 0               | 0               | 0              | 0              |

= E0 H

ICW 2 gives the address 70 H (base address) 0111 0000.

The operation command word is written as:

#### OCW 1      A<sub>0</sub> = 1

|                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |
| M <sub>7</sub> | M <sub>6</sub> | M <sub>5</sub> | M <sub>4</sub> | M <sub>3</sub> | M <sub>2</sub> | M <sub>1</sub> | M <sub>0</sub> |
| 0              | 0              | 0              | 0              | 1              | 0              | 1              | 0              |

= 0A H

In this OCW 1, M<sub>1</sub> and M<sub>3</sub> are 1 as IR1 and IR3 are masked.

Program:

```
MVI A, 16 H      ; ICW 1 code is loaded to acc.
OUT C0 H        ; C0 H is the port address for A0 = 0.
MVI A, 70 H      ; Base address is given by ICW 2.
OUT C1 H        ; ICW 2 port address for A0 = 1.
MVI A, 0A H      ; OCW 1 for masking.
OUT C1 H        ; Port address for A0 = 1.
```

**Example 11.5.** After initializing 8259A with port addresses C0 H and C1 H for  $A_0 = 0$  and  $A_0 = 1$  respectively. Give the instruction sequence to read ISR status and IMR status.

**Solution.** After initialization of 8259A, OCW 3 is written with RR = 1 and RIS = 1, to enable reading of ISR status (this is necessary for reading the status of ISR). So OCW 3 is given as:

#### OCW 3

| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| 0              | 0              | ESMM           | SMM            | 1              | P              | RR             | RIS            |        |
| 0              | 0              | 0              | 0              | 1              | 0              | 1              | 1              | = 0B H |

The instruction sequence is then given for the reading the status of ISR and IMR as:  
Program:

```
MVI A, 0B H      ; OCW 3 code is loaded to acc. to enable the
                   reading of ISR status.
OUT C0 H         ; C0 H is the port address for A0 = 0.
IN C0 H          ; Read the status of ISR.
IN C1 H          ; Read the status of IMR.
```

**Example 11.6.** An interrupt service routine written for 8259A having port addresses F0 H and F1 H, ends with the instruction sequence given as:

```
MVI A, 22 H
OUT F0 H
RET
```

How does the 8259A interpret these instructions?

**Solution.** First instruction MVI A, 22 H will show the bit format for OCW 2 is given by:

| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 0              | 0              | 1              | 0              | 0              | 0              | 1              | 0              |

This resets the IR2 in service bit of ISR. This is a non-specific end of interrupt command (EOI) may be issued by the CPU by the OUT instruction to the 8259A with A<sub>0</sub> = 0.

## PROBLEMS

1. Draw the block diagram of programmable interrupt controller (PIC) 8259A and discuss its various blocks.
2. Describe how the interfacing of 8259A with 8085A microprocessor is done.
3. Discuss how the cascading of 8259A is carried out. How many 8259s may be cascaded? What is the function of master 8259A in the cascading mode?
4. Discuss how initialization of 8259 is carried out.
5. Describe various Initialization command words in 8259A (PIC).
6. What do you understand by operation command words in 8259A? Discuss any one command word.
7. Explain the various interrupt modes of 8259A.
8. Discuss status read operation of 8259A.
9. Explain the vectoring data format of 8259A (PIC).
10. Discuss Rotating Priority mode of operation of 8259A.
11. Initialize 8259A to interface eight edge triggered inputs. The restart address of first interrupt request is 1020 H. There is a spacing interval of four byte between each interrupt request. No cascading of 8259A is made. Let the port address is C0 H and C1 H for A<sub>0</sub> = 0 and A<sub>0</sub> = 1 respectively.
12. Assume that initialization command words have been written to 8259A(single) having the port address F0 H and F1 H for A<sub>0</sub> = 0 and A<sub>0</sub> = 1 respectively. Now

- write proper operation control words for a fully nested priority structure. Mask IR2, IR3 and IR5; enable ISR for a read. What code should be used at the end of interrupt service routine?
13. Write Initializing command word to program 8259A (single) with port addresses 20 H and 21 H for  $A_0 = 0$  and  $A_0 = 1$  respectively. Let the starting address of the interrupt request is 8000 H with interval spacing of 4. The interrupts are edge triggered. Interrupts 0 and 4 are masked.
  14. Write a non-specific end of interrupt command (EOI) to reset IR3 in service bit of ISR4. Assume the port addresses 20 H and 21 H for 8259.
  15. After initializing 8259A with port addresses F0 H and F1 H for  $A_0 = 0$  and  $A_0 = 1$  respectively. Give the instruction sequence to read ISR status and IMR status.
  16. Explain the following initialization instructions:

```
MVI A, 3E H      ; ICW 1  
OUT 80 H        ; Initialize 8259A  
MVI A, E0 H      ; ICW 2  
OUT 81 H        ; Initialize 8259A
```

---

# 12

## Direct Memory Access Controller: 8257

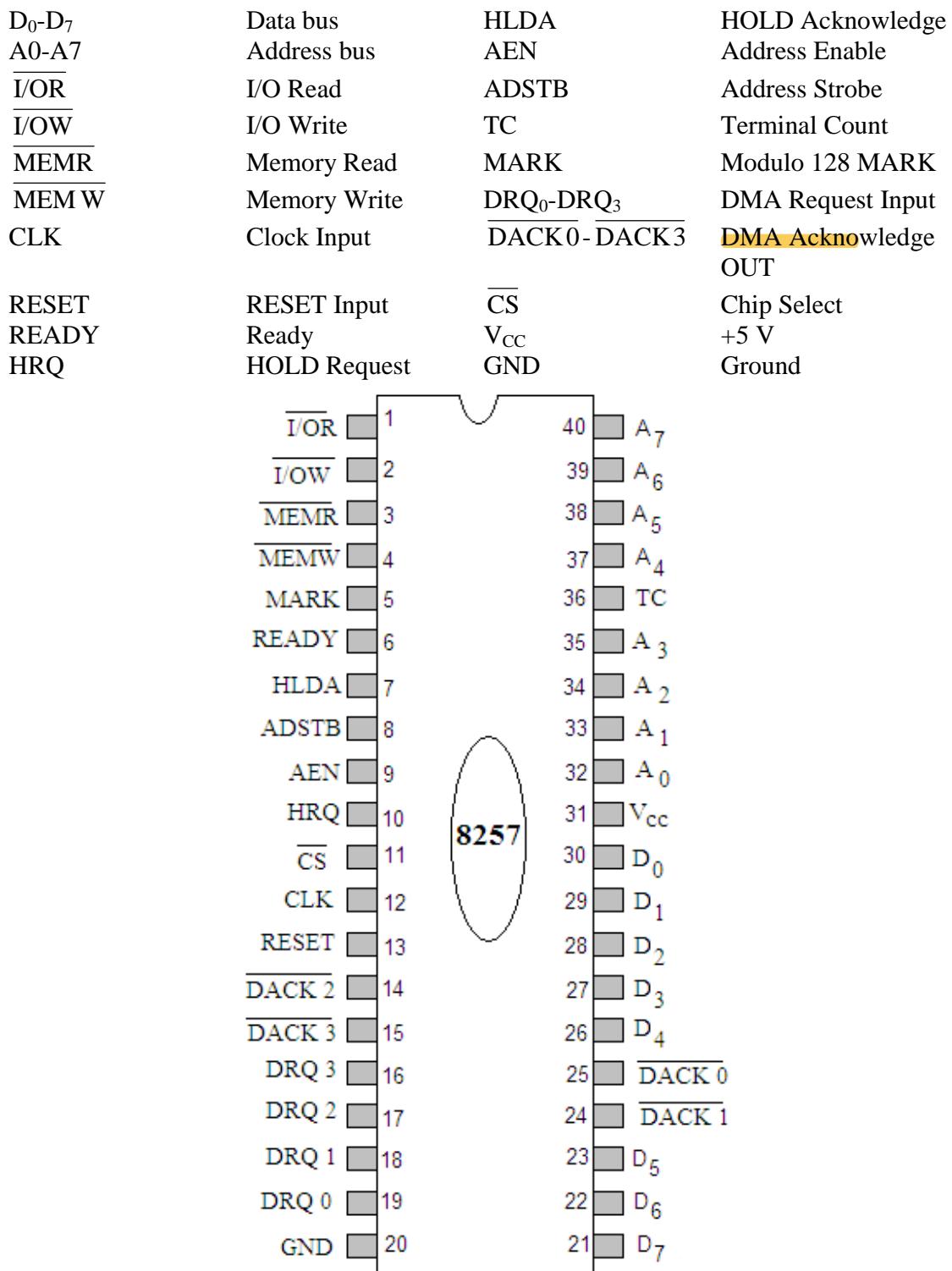
---

It has already been discussed in an earlier chapter of this book, that the Direct Memory Access (DMA) is the efficient way of transferring the data from the memory to the I/O devices or vice versa without involving the CPU. In DMA operation, the CPU relinquishes the control over the data bus and address bus, so that these can be used for transfer of data between the memory and I/O devices directly. For the data transfer using DMA process, a request to the microprocessor by the I/O device is sent and on receipt of such request, the microprocessor relinquishes the address and data buses and informs the I/O devices of the situation by sending Acknowledge signal. However, in the usual method of data transfer between the I/O devices and the memory, it involves the microprocessor. Each byte is routed through the accumulator, as a result of which it takes lot of time for transferring a large amount of data. For the purpose of DMA operation, a DMA controller is necessary to interface the peripheral to the system. Intel 8257 is a 4-channel programmable DMA controller used with 8085 and other microprocessors for this mode. In this chapter the details of the DMA controller 8257 will be discussed.

### 12.1 BLOCK DIAGRAM OF 8257

The 8257 Programmable DMA controller is available in the form of IC dual in line package. It consists of 40 pins and requires +5 volt d.c. supply for its operation. The 8257 enables to interface up to four I/O devices to the microprocessor for the data transfer between the memory and the I/O devices or vice versa using DMA. It has four DMA channels. Each channel consists of two 16 bit registers. One of these registers holds the address to be used in the next DMA cycle. The other register holds, in the least significant 14 bits, the total number of bytes to be transferred between the memory and peripherals. The two most significant bits of this register are set to indicate the operation to be performed by the device on that channel. A total of 16384 ( $2^{14} = 16$  K) bytes of data may directly be transferred under DMA control without any intervention of the microprocessor.

Figure 12.1 shows the pin diagram of 8257. The pin names of this IC are given below:



**Fig. 12.1**

The internal block diagram and schematic diagram of 8257, DMA Controller (DMAC) are shown in figures 12.2 and 12.3 respectively. It contains the following blocks:

- DMA Channels
- Data Bus Buffer
- Read/ Write Logic
- Control Logic
- Mode Set Register
- Status Word Register

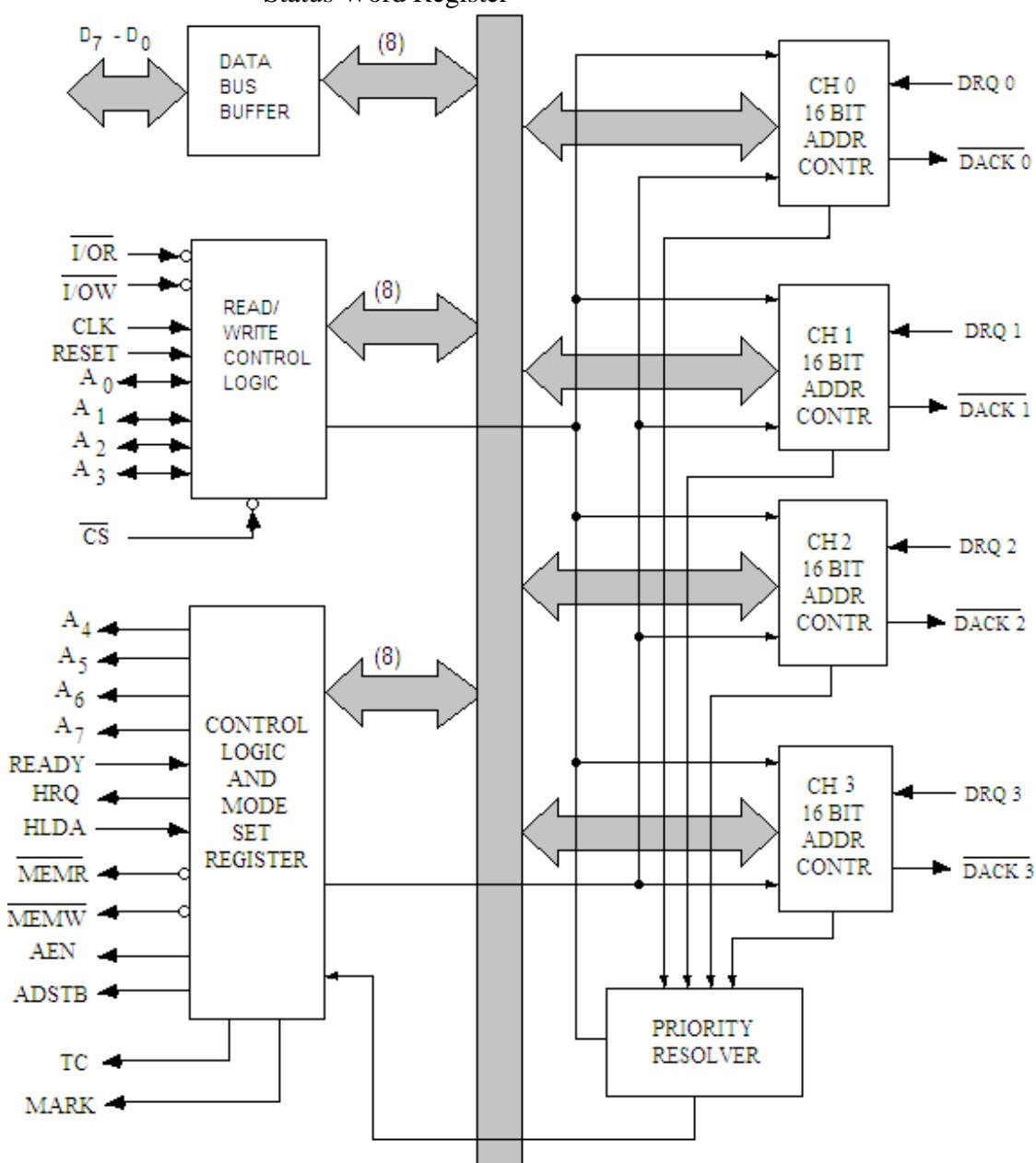
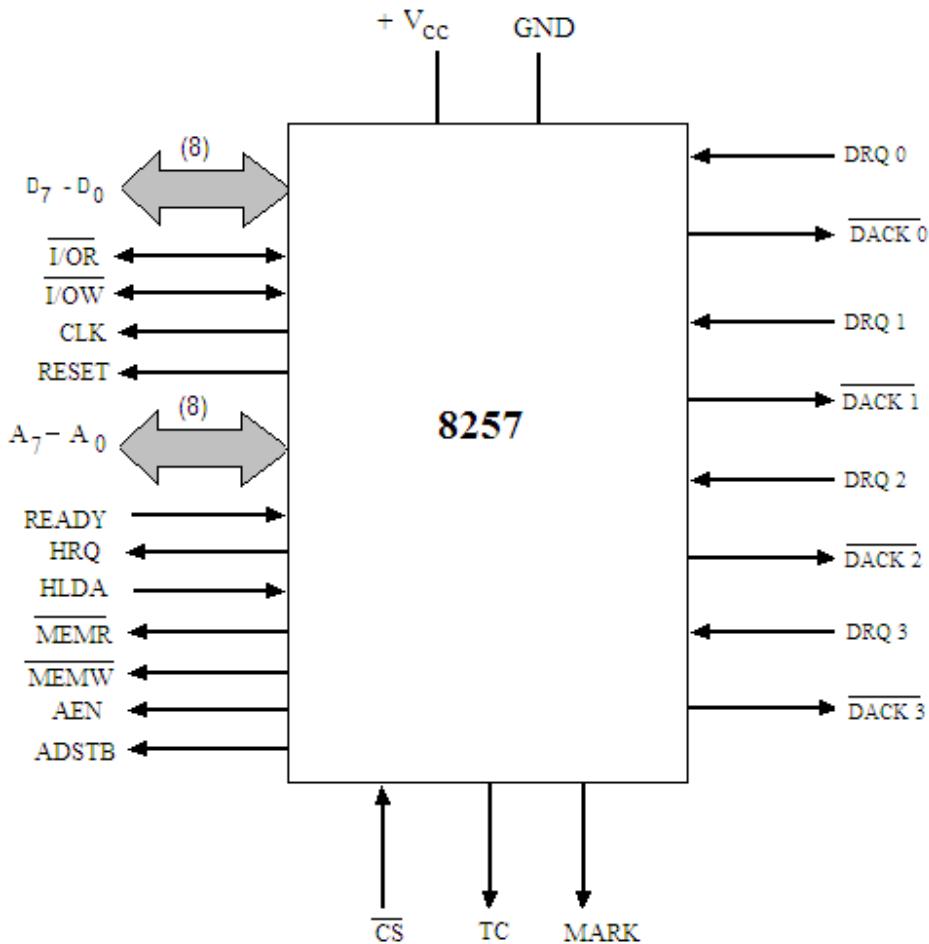


Fig. 12.2



**Fig. 12.3**

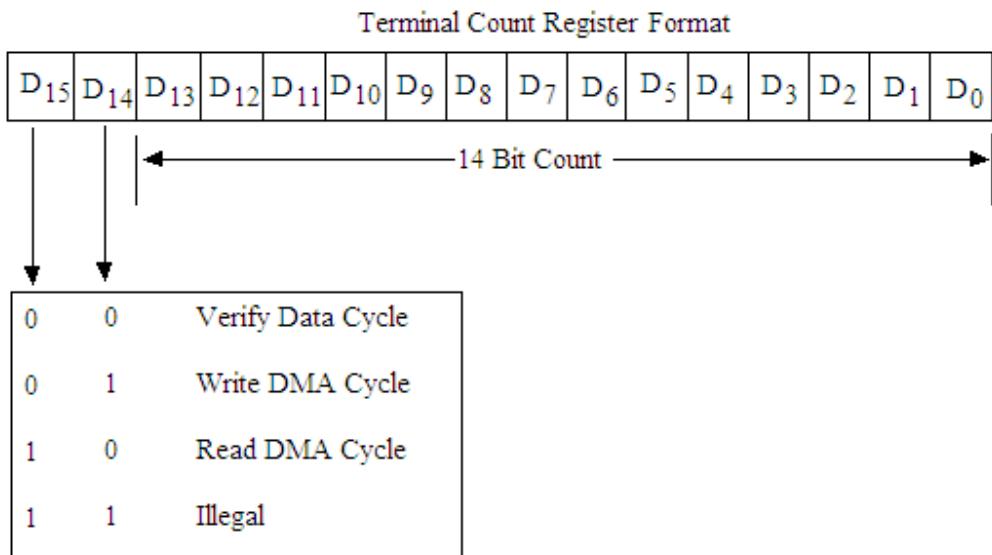
#### 12.1.1 DMA Channels

The 8257 provides 4 channels which can be connected to four separate I/O devices. These channels are designated as Channel 0 (CH-0), Channel 1 (CH-1), Channel 2 (CH-2) and Channel 4 (CH-4). Each of these channels has two 16-bit registers:

- DMA Address Register
- Terminal Count Register.

DMA Address Register contains the memory address from where the data transfer should begin. This address is loaded by the CPU.

The terminal count register is used to store the number of bytes to be transferred. In this 16 bit register, the least significant 14 bits are used to store the number of bytes to be transferred. In this way one DMA operation can transfer to a maximum of 16 K bytes. The most significant two bits of the terminal register represent the operation to be performed by the device. The operations to be performed are Write, Read and Verify. Figure 12.4 shows the format of the terminal count register of a DMA channel.



**Fig. 12.4**

- During DMA write operation, the data is transferred from the peripheral to memory.
- During DMA read operation, the data is transferred from memory to peripheral.
- DMA verify operation does not actually involve the transfer of data.

Each channel has two signals: DMA REQUEST Signal and DMA ACKNOWLEDGE Signal.

#### DMA REQUEST Signals (DRQ0 – DRQ3):

There are four DMA request signals (DRQ0 – DRQ3) in the 8257 connected one with each channel. Each channel accepts a DMA request through this pin from the peripheral. The DMA controller 8257 has priority facility. The channel 0 (CH 0) has the highest priority and the channel 3 (CH 3) has the lowest priority. As per the priority, a request is generated by the peripheral by making corresponding DRQ line high and holding it high until DMA operation is over.

#### DMA ACKNOWLEDGE Signals ( $\overline{\text{DACK}0}$ - $\overline{\text{DACK}3}$ ):

Likewise the DMA request signals, 8257 has the four DMA acknowledge signals ( $\overline{\text{DACK}0}$  -  $\overline{\text{DACK}3}$ ) connected one with each channel. As discussed above, each channel accept a DMA request through its DRQ line from the peripheral. After the receipt of the request channel issues an acknowledge signal through  $\overline{\text{DACK}}$  output of that channel. It is an active low signal and signifies that the request to be serviced for a DMA cycle is accepted. This signal is made high on completion of the transfer.

### **12.1.2 Data Bus Buffer**

This three state, bidirectional eight bit buffer ( $D_0-D_7$ ) interfaces the 8257 to the system data bus. When the CPU has the control over the system buses (Slave Mode), the  $D_0-D_7$  pins of the 8257 are used as input pins. The eight bits of data for a DMA address register and terminal count register or the mode set register are received on the data bus. In this slave mode, the CPU can also read eight bits of data at a time from the 16-bit DMA address and terminal count register or from the eight bit status register. However, in the master mode (for DMA cycle, when the CPU relinquishes the control over the control and data buses), the  $D_0-D_7$  pins are used for different purpose. At the beginning of each DMA cycle, most significant bits of DMA address register are put on these pins which are further latched to an external chip 8212 used as latch. These latched values are put on A8-A15 of the system address bus. The  $D_0-D_7$  bus is then released to handle the memory data transfer during the remainder of the DMA cycle.

### **12.1.3 Read/ Write Logic**

When the CPU is programming or reading one of the registers of 8257 (i.e. when the 8257 is in slave mode), the Read/ Write logic accepts the I/O Read ( $\overline{I/OR}$ ) or I/O Write ( $\overline{I/OW}$ ) signal. It then decodes the least significant four address bits ( $A_0-A_3$ ), and writes the contents of the data bus into the addressed register (if  $\overline{I/OW}$  is active) or places the contents of the addressed register onto the data bus (if  $\overline{I/OR}$  is active). During DMA cycle (i.e. when the 8257 is in master mode), the Read/ Write logic generates the I/O Read and Memory Write cycle (DMA Write cycle) and generates the Memory Read and I/O Write Cycle (DMA Read cycle).

#### **$\overline{I/OR}$ (I/O Read)**

This is an active low, bidirectional three-state pin. In the slave mode, it is an input signal which allows the 8-bit status register or the upper/lower byte of a 16-bit DMA address register or terminal count register to be read. In the master mode,  $\overline{I/OR}$  is a control output, which is used to access data from a peripheral during the DMA write cycle.

#### **$\overline{I/OW}$ (I/O Write)**

This is also an active low, bidirectional three-state pin. In the slave mode, it is an input signal which allows the contents of the data bus to be loaded into the 8-bit mode set register or the upper/lower byte of a 16-bit DMA address register or terminal count register. In the master mode,  $\overline{I/OW}$  signal is a control output which allows data to be output to a peripheral during a DMA read cycle.

#### **CLK (Clock input)**

This input is generally connected to an external clock or to the 8085A CLK output.

#### **RESET (Reset)**

This is an active high asynchronous input which disables all DMA channels and clears the mode set register. It also tri states all the control lines.

### **CS** (Chip Select)

This is an active low input which enables the I/O Read or I/O write input when the 8257 is being read or programmed in the slave mode. In the master mode, CS input is automatically disabled. This is done to prevent the 8257 from selecting itself when it puts out DMA addresses on the address bus.

### **A<sub>0</sub>-A<sub>3</sub>** (Address lines)

These least significant four address lines are bidirectional. In the slave mode, they are inputs which select one of the registers to be read or programmed. In the master mode, they are outputs which constitute the least significant four bits of the 16-bit address generated by the 8257.

#### **12.1.4 Control Logic**

This block controls the sequence of operations during all DMA cycles by generating the appropriate control signals and the 16-bit address that specifies the memory location to be accessed. The pins associated with the control logic are described below.

### **A<sub>4</sub>-A<sub>7</sub>** (Address lines)

These four address lines are the three state output lines. Bits 4-7 of the DMA address register are put on these address lines during the DMA cycles. So, the A<sub>0</sub>-A<sub>7</sub> of 8257 carry the least significant byte of DMA address during the execution of a DMA cycle.

### **READY**

This ready input signal can be used by the devices with slow access time in order to insert the wait states during DMA transfer. It has similar function to the Ready input pin of 8085.

### **HRQ (Hold Request)**

This output requests control of the system bus. In systems with only one 8257, this signal will be normally applied the HOLD input of the CPU. HRQ must confirm to specified setup and hold times.

### **HLDA (Hold Acknowledge)**

This is an input from the CPU that informs the 8257 that it has relinquished control of the buses and 8257 may take over the control.

### **MEMR** (Memory Read)

This is a three-state active low output which is used to read data from the addressed memory location during DMA Read cycles.

### **MEMW** (Memory Write)

This is also three state active low output which is used to write data into the addressed memory location during DMA Write cycles.

### ADSTB (Address Strobe)

This is an active high signal which is generated at the beginning of each DMA cycle and has a similar function as that of ALE pin of 8085A.

### AEN (Address Enable)

This is an output pin used to float the system data bus and the system control bus. It may also be used to float the system address bus. It may further be used to isolate the 8257 data bus from the system data bus to facilitate the transfer of the 8-bit most significant DMA address bits over the 8257 data I/O pins. When the 8257 is used in an I/O device structure, this AEN output is used to disable the selection of an I/O device when the DMA address is on the address bus. The I/O device selection should be determined by the DMA acknowledge output for the four channels.

### TC (Terminal Count)

This is an active high output. When this pin is high it indicates to the currently selected peripheral that the present DMA cycle is the last cycle for this data block. This output goes high, when the contents of the terminal count register of the selected channel are equal to zero. By programming the mode word, a channel can automatically be disabled after its last DMA cycle.

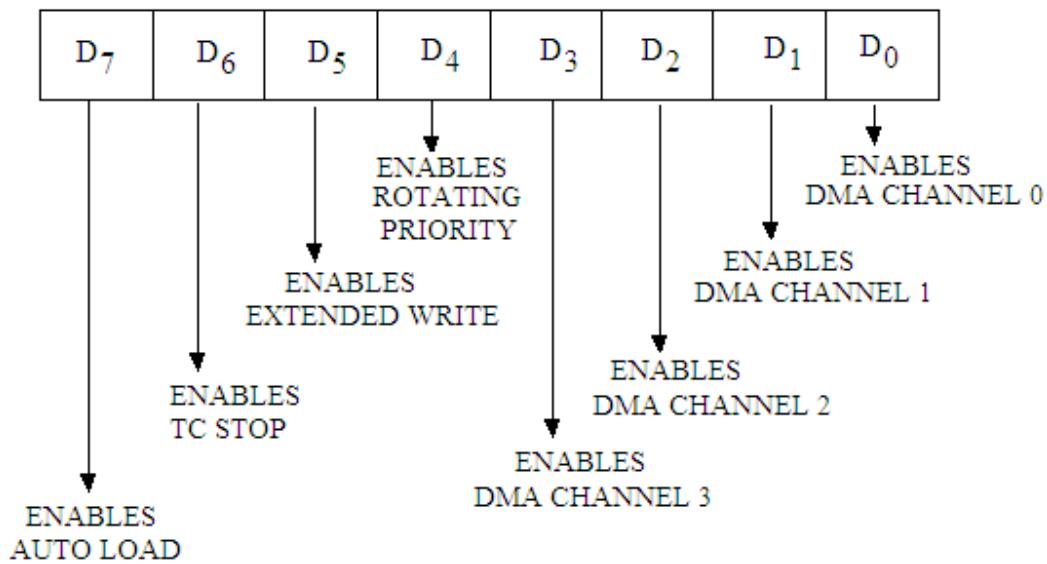
### MARK (Modulo 128 Mark)

This active high output notifies the selected peripheral that the current DMA cycle is the 128<sup>th</sup> cycle since the previous mark output. Mark always occurs at 128 (and all multiples of 128) cycles from the end of the data block. Only if the total number of DMA cycles (n) is evenly divisible by 128, mark will occur at 128 cycles from the beginning of the data block.

#### 12.1.5 Mode Set Register

The format of Mode Set Register of 8257 is shown in figure 12.5. The control word in the mode set register enables or disables channels and determines other functions. The mode set register can be accessed only for write operation. The various bits in this mode set register enable each of the four DMA channels and four different options for the 8257. The mode set register is cleared by the RESET input, thus disabling all options, inhibiting all channels. This mode set register is also used for programming the 8257 in the following options:

- Rotating Priority
- Extended Write
- TC Stop
- Auto Load

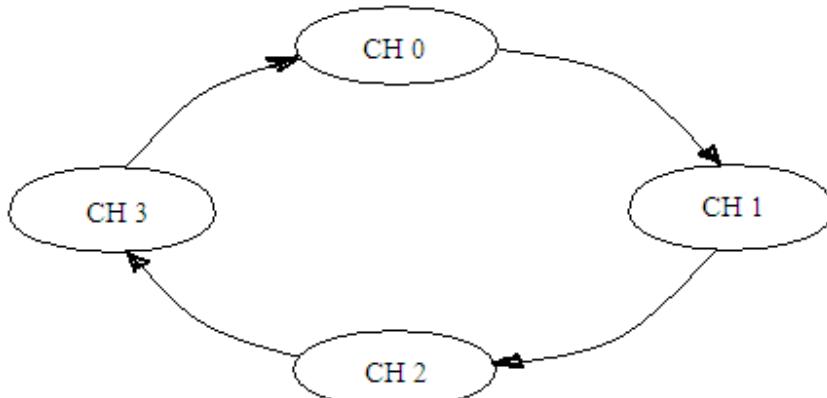


LOGIC 1 IN A BIT ENABLES OPTION/CHANNEL

**Fig. 12.5**

### Rotating Priority

The bit 4 (D<sub>4</sub>) of the Mode Set Register enables to set the rotating priority mode of the 8257. In the rotating priority mode, the priority of the channels has a circular sequence as shown in figure 12.6. After each DMA cycle the priority of channels has circular sequence. After each DMA cycle, the priority of each channel changes. The channel which had just been serviced will have the lowest priority (Figure 12.7).



**Fig. 12.6**

If the rotating priority bit is not set (D<sub>4</sub> = 0), each DMA channel has a fixed priority. In the fixed priority mode, channel 0 has the highest priority and the channel 3 has the lowest priority. If the rotating priority bit D<sub>4</sub> is set to 1, the priority of each channel changes after each DMA cycle (not each DMA request). Each channel moves up to the next highest priority assignment, while the channel which has just been serviced moves to the lowest priority assignment.

|                     | CHANNEL JUST SERVED | CH 0 | CH 1 | CH 2 | CH 3 |
|---------------------|---------------------|------|------|------|------|
| PRIORITY ASSIGNMENT | HIGHEST             | CH 1 | CH 2 | CH 3 | CH 0 |
|                     |                     | CH 2 | CH 3 | CH 0 | CH 1 |
|                     |                     | CH 3 | CH 0 | CH 1 | CH 2 |
|                     |                     | CH 0 | CH 1 | CH 2 | CH 3 |

Fig. 12.7

### Extended Write

The bit 5 ( $D_5$ ) of the mode set register is used for extended write option. The data transfer with in microcomputer systems takes place asynchronously which allow the use of various types of memory and I/O devices with different access times. If the device can not be accessed within the specified amount of time, it returns a ‘Not Ready’ signal on the READY input of the 8257 due to which one or more wait states are inserted in the internal sequencing. Some devices are fast enough to be accessed without the use of wait states. But if they generate their Ready response with the leading edge of the  $\overline{I/O W}$  or  $\overline{MEMW}$  signal (which generally occurs late in transfer sequence), they would normally cause the 8257 to enter a wait state because it does not receive Ready in time. For systems with these types of devices, the extended write operation provides alternative timing for the I/O and memory write signals which allows the devices to return on early Ready and prevents the unnecessary occurrence of wait states in the 8257.

### TC Stop

If the TC stop bit ( $D_6$ ) of the mode set register is set, a channel is disabled (i.e. its enable bit is reset) after the terminal count (TC) output goes high, thus automatically preventing further DMA operation on that channel. To continue or begin another DMA operation, the enable bit of that channel must be reprogrammed.

If the TC is not set, the occurrence of TC bit output has no effect on the channels enable bits. In that case, it will be responsibility of the peripheral to cease DMA requests in order to terminate a DMA operation.

### Auto Load

Bit  $D_7$  of the mode set register enables Auto Load mode. In some cases, it becomes necessary to perform DMA operation repeatedly. In case of a CRT monitor, the data has to repeatedly send to the monitor and this process is called refreshing. With the auto load option, the DMA controller permits such repetitive operation. This bit permits channel 2 to be used for repeat block operation. If the auto load mode is set ( $D_7 = 1$ ), the initial parameters of channel 2 are automatically copied onto channel 3. The channel 2 is programmed with the initial parameters for the first DMA block. When the DMA block of channel 2 is completed, the parameters stored in channel 3 are copied onto channel 2. The channel 2 is then serviced again with the same parameters for repetitive DMA operations. For chaining operation, channel 3 is loaded with a different set of parameters

after channel 2 is loaded; the new parameters are copied onto channel 2 during the update cycle and channel 2 is serviced with a new set of parameters.

### 12.1.6 Status Word Register

The eight bit status register is shown in figure 12.8. The status register can be read for the status of the terminal counts of the four channels. It also contains the update flag. The count bits (D<sub>0</sub>-D<sub>4</sub>) of the four channels are set when the terminal count output is high for that channel. The TC bits go low when the status word is read or when the 8257 receives a Reset input. The update flag is reset when:

- 8257 is reset or
- Auto load option in the mode set register is disabled or
- Update cycle gets completed.

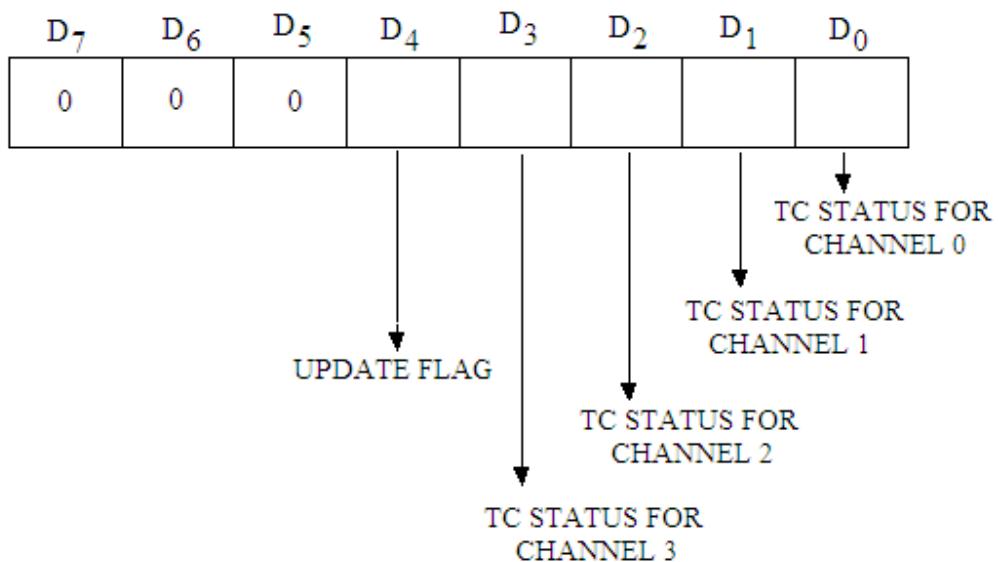


Fig.12.8

## 12.2 PROGRAMMING OF 8257

As discussed above, there are four pairs of channel registers in 8257; each pair consisting of a 16-bit Address Register and 16 bit Terminal Count Register i.e. one pair for each channel. The 8257 also includes two general registers one 8-bit Mode Set Register and other 8-bit Status Register. These various registers are accessed with the help of four input lines (A<sub>0</sub>-A<sub>3</sub>) and an internal F/L flip-flop (First/ Last Flip-flop). The address bit 3 (A<sub>3</sub>) signifies whether a channel register or the mode set register is being accessed. If A<sub>3</sub> = 0, then channel register is accessed and if A<sub>3</sub> = 1, then the mode set register is accessed. The other three address bits (A<sub>0</sub>-A<sub>2</sub>) specify which register is to be accessed. When these bits (A<sub>0</sub>-A<sub>2</sub>) are all zero, the mode set register or status register is being accessed. The 8257 distinguishes between the mode set register (which is write only operation) and the status operation (which is read only operation) by the I/OW and I/OR inputs respectively. Table 12.1 illustrates the status of A<sub>3</sub> and the control inputs for accessing the various registers. For channel registers, A<sub>2</sub> A<sub>1</sub> specify one of the four channels i.e. for channel 0, it is 00, for channel 1 it is 01, for channel 2 it 01 and for channel 3 it is 11. The bit A<sub>0</sub> specifies whether channel register or terminal count register

is to be accessed. If  $A_0 = 0$ , channel register is accessed and if  $A_0 = 1$ , terminal count register is accessed. The F/L flip-flop is reset by reset input. This flip-flop toggles for access of the 8257 registers and determines whether upper or lower order byte of a particular register is accessed. It requires that the reading of the address registers or the terminal count registers should be done in pairs; the lower order byte should be accessed first. The addressing of the registers is shown in table 12.2.

**Table 12.1**

| CONTROL INPUT                             | $\overline{CS}$ | $\overline{I/OW}$ | $\overline{I/OR}$ | $A_3$    |
|---|-----------------|-------------------|-------------------|----------|
| <b>Program Half of a Channel Register</b> | <b>0</b>        | <b>0</b>          | <b>1</b>          | <b>0</b> |
| <b>Read Half of a Channel Register</b>    | <b>0</b>        | <b>1</b>          | <b>0</b>          | <b>0</b> |
| <b>Program Mode Set Register</b>          | <b>0</b>        | <b>0</b>          | <b>1</b>          | <b>1</b> |
| <b>Read Status Register</b>               | <b>0</b>        | <b>1</b>          | <b>0</b>          | <b>0</b> |

**Table 12.2**

| ADDRESS INPUTS |       |       |       | State of  |                 | Register Accessed                             |
|----------------|-------|-------|-------|-----------|-----------------|---|
| $A_3$          | $A_2$ | $A_1$ | $A_0$ | FF<br>F/L | $\overline{CS}$ |   |
| 0              | 0     | 0     | 0     | 0         | 0               | LOB of DMA address CH 0                       |
| 0              | 0     | 0     | 0     | 1         | 0               | HOB of DMA address CH 0                       |
| 0              | 0     | 0     | 1     | 0         | 0               | LOB of Counts of Terminal Count Register CH 0 |
| 0              | 0     | 0     | 1     | 1         | 0               | HOB of Counts of Terminal Count Register CH 0 |
| 0              | 0     | 1     | 0     | 0         | 0               | LOB of DMA address CH 1                       |
| 0              | 0     | 1     | 0     | 1         | 0               | HOB of DMA address CH 1                       |
| 0              | 0     | 1     | 1     | 0         | 0               | LOB of Counts of Terminal Count Register CH 1 |
| 0              | 0     | 1     | 1     | 1         | 0               | HOB of Counts of Terminal Count Register CH 1 |
| 0              | 1     | 0     | 0     | 0         | 0               | LOB of DMA address CH 2                       |
| 0              | 1     | 0     | 0     | 1         | 0               | HOB of DMA address CH 2                       |
| 0              | 1     | 0     | 1     | 0         | 0               | LOB of Counts of Terminal Count Register CH 2 |
| 0              | 1     | 0     | 1     | 1         | 0               | HOB of Counts of Terminal Count Register CH 2 |
| 0              | 1     | 1     | 0     | 0         | 0               | LOB of DMA address CH 3                       |

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 1 | 0 | HOB of DMA address CH 3                       |
| 0 | 1 | 1 | 1 | 0 | 0 | LOB of Counts of Terminal Count Register CH 3 |
| 0 | 1 | 1 | 1 | 1 | 0 | HOB of Counts of Terminal Count Register CH 3 |
| 1 | 0 | 0 | 0 | 0 | 0 | Mode Set Register (Write Only)                |
| 1 | 0 | 0 | 0 | 0 | 0 | Status Register (Read Only)                   |

HOB – High order byte

LOB – Low order byte

The microprocessor can either read data from or write data into DMA address register and the terminal count register of each channel of 8257. The status register can be read and mode set register may be programmed. All these operations have been discussed above and carried out during the slave mode of 8257.

If the address decoder circuit for the chip select terminal of DMA controller 8257 shown in figure 12.9 is used, it will give the addresses of DMA registers and terminal count registers of all the four channels along with the address of the mode set registers. These addresses are given in table 12.3.

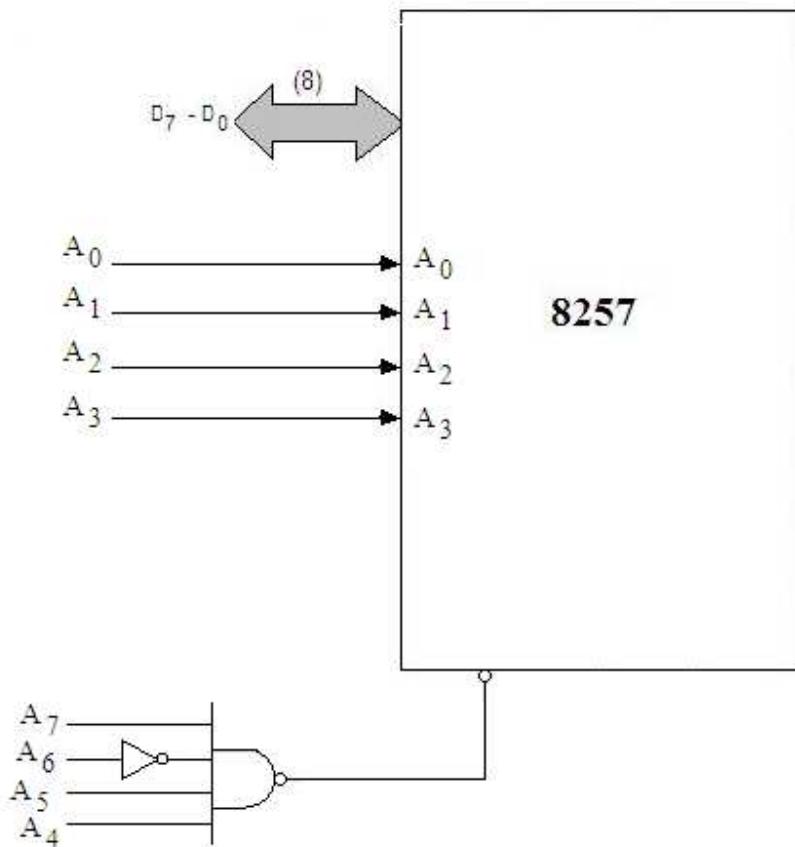


Fig. 12.9

**Table 12.3**

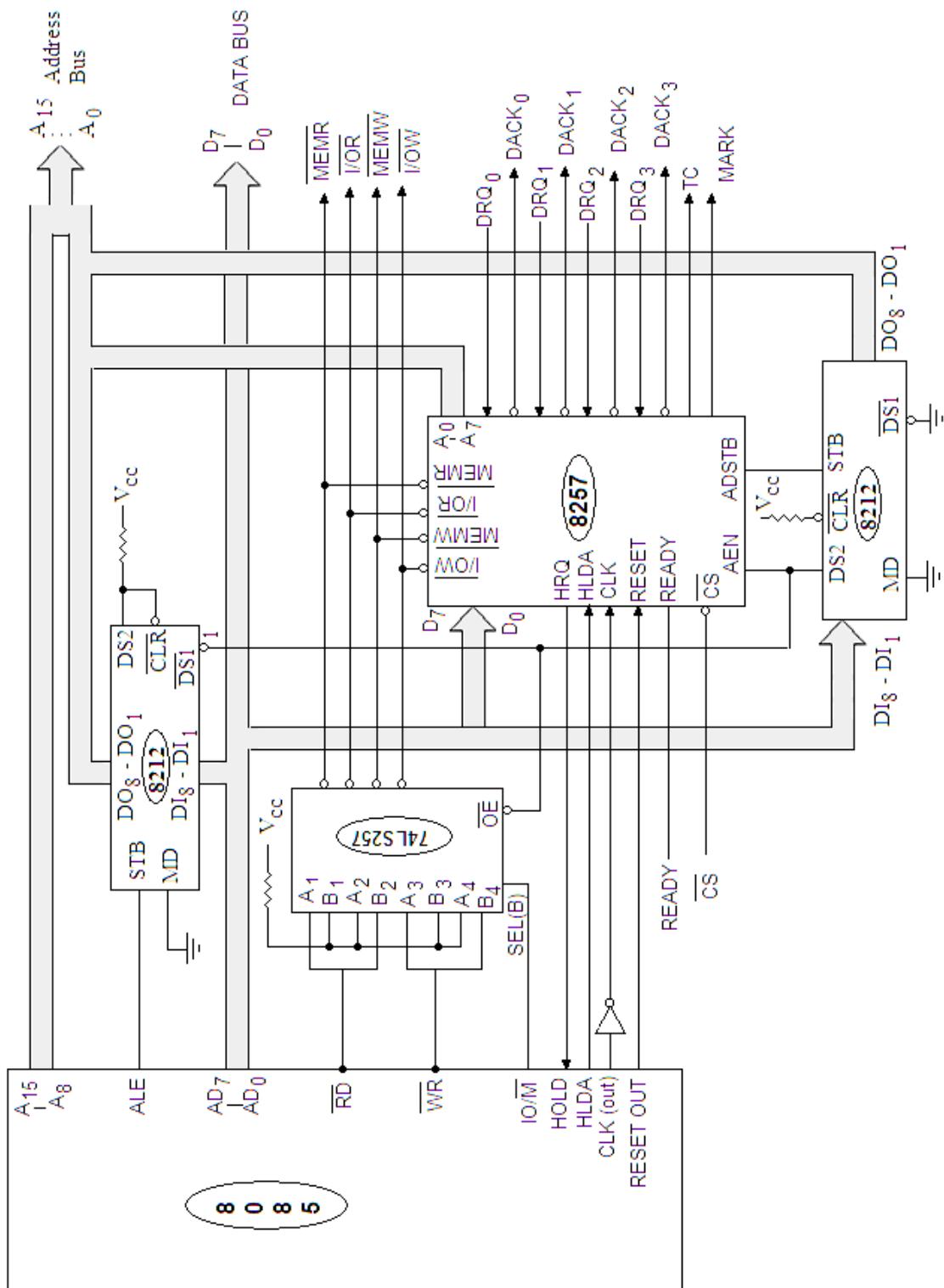
| <b>A<sub>7</sub></b> | <b>A<sub>6</sub></b> | <b>A<sub>5</sub></b> | <b>A<sub>4</sub></b> | <b>A<sub>3</sub></b> | <b>A<sub>2</sub></b> | <b>A<sub>1</sub></b> | <b>A<sub>0</sub></b> | <b>Address</b>          |
|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|-------------------------|
| 0                    | 1                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 70 H DMA address CH 0   |
| 0                    | 1                    | 0                    | 0                    | 0                    | 0                    | 0                    | 1                    | 71 H TC address CH 0    |
| 0                    | 1                    | 0                    | 0                    | 0                    | 0                    | 1                    | 0                    | 72 H DMA address CH 1   |
| 0                    | 1                    | 0                    | 0                    | 0                    | 0                    | 1                    | 1                    | 73 H TC address CH 1    |
| 0                    | 1                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 74 H DMA address CH 2   |
| 0                    | 1                    | 0                    | 0                    | 0                    | 1                    | 0                    | 1                    | 75 H TC address CH 2    |
| 0                    | 1                    | 0                    | 0                    | 0                    | 1                    | 1                    | 0                    | 76 H DMA address CH 3   |
| 0                    | 1                    | 0                    | 0                    | 0                    | 1                    | 1                    | 1                    | 77 H TC address CH 3    |
| 0                    | 1                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    | 78 H Mode Set Reg addr. |

The various registers have to be first initialized and then DMA operation will be started. The steps, to be followed for the DMA operation of 8257 (Master Mode), after its initialization, are given as:

- The I/O device sends a DMA request signal (DRQ) when it is ready to for the data transfer.
- The 8257 sends Hold Request signal (HRQ) high to the processor. It then enables the particular channel.
- In response to HRQ signal, the processor will relinquishes the system busses in the next cycle and will send a Hold Acknowledge (HLDA) signal to the 8257.
- In response to this HLDA signal from the microprocessor, the DMA controller will generate (DACK) DMA acknowledge signal (active low) through its Control Logic block as an acknowledgement to the requesting peripheral. At the same time the 8257 enables the Address Enable (AEN) signal which disables systems address lines (A<sub>0</sub>-A<sub>7</sub>). These address lines becomes the output lines. The low order byte of the memory location is placed of these lines. The address strobe (ADSTB) signal goes high when AEN is high and places higher byte of the memory location generated by 8212 on the address bus A<sub>15</sub>-A<sub>8</sub>.
- The data transfer continues till the terminal count is reached.

### 12.3 DMA INTERFACING CIRCUIT

Figure 12.10 shows the interfacing circuit of DMA controller 8257. In this circuit



**Fig. 12.10**

8212 is used to de-multiplex the 8085 bus to generate the low order address bus ( $A_7-A_0$ ). The 8257 has eight address lines, but requires sixteen address lines to address a memory location. The additional eight lines are generated by using the signal ADSTB to strobe a high order memory address into the 8212 from the data bus. This IC is not related to 8085 not to the DMA controller 8257, but only  $\overline{DS1}$  is controlled by AEN signal of 8257. The IC 74LS257 is a multiplexer which is used to generate control signals. The  $\overline{OE}$  terminal of this multiplexer is also controlled by AEN. The AEN output signal is used to disable (float) the system bus and control bus. This signal is also necessary to switch the 8257 from the slave mode to the master mode. The port addresses of the registers of 8257 are used as discussed above and circuit for the same is given in figure 12.9. The port addressed may be chosen as desired. The second 8212 connected to the DMA controller accepts the data during the DMA operation.

**Example 12.1** Initialize 8257 DMA controller to transfer 2K bytes of data stored in memory locations starting at 2101 H to floppy disk connected to the channel 1 of 8257. Assume that the address of the Mode Set Register is 78 H. The addresses of DMA address register for channel 1 and the Terminal Count Register are 72 H and 73 H respectively.

**Solution.** The format for Mode Set register is given below:

| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>1</sub> | D <sub>0</sub> |             |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|-------------|
| 0              | 1              | 0              | 0              | 0              | 1              | 0              | = 42 H      |
|                | ↓              |                |                |                |                |                | ↓           |
|                | Enable         |                |                |                |                |                | Enable CH 1 |

TC stop

Format for Terminal Count Register is given below:

| D <sub>15</sub> | D <sub>14</sub> | D <sub>13</sub> | D <sub>12</sub> | D <sub>11</sub> | D <sub>10</sub> | D <sub>9</sub> | D <sub>8</sub> | D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |          |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------|
| 1               | 0               | 0               | 0               | 0               | 1               | 1              | 1              | 1              | 1              | 1              | 1              | 1              | 1              | 1              | 1              | = 87FF H |

D<sub>15</sub> and D<sub>14</sub> as 10 indicate it is memory read operation and D<sub>10</sub> to D<sub>0</sub> as 1111111111 represents 2 K bytes of data to be transferred to the floppy.

For the initialization of 8257, following steps are to be followed:

- Load mode word in the Mode Set Register.
- Load counts to the terminal count register first LS byte and then MS byte.
- Load starting address of the memory location, from where the data is to be transferred to the floppy disk, to the DMA address register.

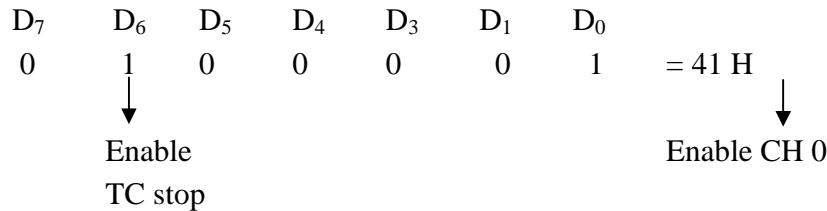
The initialization program is therefore given as:

|             |   |
|-------------|---|
| MVI A, 42 H | ; Load mode word in the Mode Set Register.                  |
| OUT 78 H    | ; Address of the Mode Set Register.                         |
| MVI A, FF H | ; Load LS byte of the count in the terminal count register. |
| OUT 73 H    | ; Address of TC register.                                   |

|             |   |
|-------------|---|
| MVI A, 87 H | ; Load MS byte of the count in the terminal count register.   |
| OUT 73 H    | ; Address of TC register.   |
| MVI A, 00 H | ; Load the LS byte of the starting address of the memory location from where the data is to be transferred. |
| OUT 72 H    | ; Address of the DMA address register.  |
| MVI A, 21 H | ; Load the MS byte of the starting address of the memory location.  |
| OUT 72 H    | ; Address of the DMA address register.  |

**Example 12.2** Initialize 8257 DMA controller to transfer 512 bytes of data from a peripheral to memory locations starting at 3000 H through channel 0. Assume that the address of the Mode Set Register is 28 H. Assume the addresses of DMA address register for channel 0 and the Terminal Count Register are 20 H and 21 H respectively.

**Solution.** The format for Mode Set register is given below:



Format for Terminal Count Register is given below:

|                 |                 |                 |                 |                 |                 |                |                |                |                |                |                |                |                |                |                |          |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------|
| D <sub>15</sub> | D <sub>14</sub> | D <sub>13</sub> | D <sub>12</sub> | D <sub>11</sub> | D <sub>10</sub> | D <sub>9</sub> | D <sub>8</sub> | D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |          |
| 0               | 1               | 0               | 0               | 0               | 0               | 0              | 1              | 1              | 1              | 1              | 1              | 1              | 1              | 1              | 1              | = 41FF H |
|                 |                 |                 |                 |                 |                 |                |                |                |                |                |                |                |                |                |                |          |

D<sub>15</sub> and D<sub>14</sub> as 01 indicate it is memory write operation and D<sub>8</sub> to D<sub>0</sub> as 11111111 represents 512 bytes of data to be transferred.

The initialization program is therefore given as:

|             |   |
|-------------|---|
| MVI A, 41 H | ; Load mode word in the Mode Set Register.  |
| OUT 28 H    | ; Address of the Mode Set Register.   |
| MVI A, FF H | ; Load LS byte of the count in the terminal count register.   |
| OUT 21 H    | ; Address of TC register.   |
| MVI A, 41 H | ; Load MS byte of the count in the terminal count register.   |
| OUT 21 H    | ; Address of TC register.   |
| MVI A, 00 H | ; Load the LS byte of the starting address of the memory location from where the data is to be transferred. |
| OUT 20 H    | ; Address of the DMA address register.  |
| MVI A, 30 H | ; Load the MS byte of the starting address of the memory location.  |

OUT 20 H ; Address of the DMA address register.

**Example 12.3** An I/O device which is associated with channel 2 of 8257 DMA controller is to be periodically refreshed with 4K bytes of data starting from 2200 H locations. Write the initialization routine for the DMA controller to be operated with auto-load and extended write operations. The addresses of DMA registers and terminal count registers of all the four channels along with the address of the mode set registers are given in table 12.3.

**Solution.** The format for Mode Set register is given below:

|                |                |                |                |                |                |                |                |                |        |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>1</sub> | D <sub>0</sub> | = A6 H |
| 1              | 0              | 1              | 0              | 0              | 1              | 1              | 0              | 0              |        |

D<sub>7</sub> = 1 indicates that the auto load enabled, and

D<sub>5</sub> = 1 indicates the extended write enabled.

D<sub>4</sub> and D<sub>3</sub> = 11, indicate that the channel 2 and channel 3 are enabled, which are needed for the auto load and extended write mode.

Format for Terminal Count Register is given below:

|                 |                 |                 |                 |                 |                 |                |                |                |                |                |                |                |                |                |                |          |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------|
| D <sub>15</sub> | D <sub>14</sub> | D <sub>13</sub> | D <sub>12</sub> | D <sub>11</sub> | D <sub>10</sub> | D <sub>9</sub> | D <sub>8</sub> | D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> | = 8FFF H |
| 1               | 0               | 0               | 0               | 1               | 1               | 1              | 1              | 1              | 1              | 1              | 1              | 1              | 1              | 1              | 1              |          |

D<sub>15</sub> and D<sub>14</sub> as 10 indicate it is memory read operation and D<sub>11</sub> to D<sub>0</sub> as 111111111111 represents 4096 (4K) bytes of data to be transferred.

In this case the starting address of the memory location is to be loaded to both DMA address registers of both channel 2 and channel 3 for auto load and extended write mode.

The initialization program is therefore given as:

|             |  |
|-------------|--|
| MVI A, A6 H | ; Load mode word in the Mode Set Register.                               |
| OUT 78 H    | ; Address of the Mode Set Register.                                      |
| MVI A, FF H | ; Load LS byte of the count in the terminal count register of channel 2. |
| OUT 75 H    | ; Address of TC register of channel 2.                                   |
| MVI A, 8F H | ; Load MS byte of the count in the terminal count register of channel 2. |
| OUT 75 H    | ; Address of TC register of channel 2.                                   |
| MVI A, FF H | ; Load MS byte of the count in the terminal count register of channel 3. |
| OUT 77 H    | ; Address of TC register of channel 3.                                   |
| MVI A, 8F H | ; Load MS byte of the count in the terminal count register of channel 3. |
| OUT 77 H    | ; Address of TC register of channel 3.                                   |
| MVI A, 00 H | ; Load the LS byte of the starting address of the memory location.       |

|             |  |
|-------------|--|
| OUT 74 H    | ; Address of the DMA address register channel 2.                   |
| MVI A, 22 H | ; Load the MS byte of the starting address of the memory location. |
| OUT 74 H    | ; Address of the DMA address register ch.2.                        |
| MVI A, 00 H | ; Load the LS byte of the starting address of the memory location. |
| OUT 76 H    | ; Address of the DMA address register channel 3.                   |
| MVI A, 22 H | ; Load the MS byte of the starting address of the memory location. |
| OUT 76 H    | ; Address of the DMA address register ch.3.                        |

## PROBLEMS

1. What do you understand by Direct Memory Access? What is the use of DMA controller for the data transfer from memory to I/O devices or vice-versa?
2. Draw the block diagram of 8257 DMA controller. Discuss the functions of its blocks.
3. How many I/O devices can be connected with the 8257? How many registers are there in 8257? Discuss these registers.
4. Give the format of terminal count register. Discuss the functions of the bits of TC register.
5. What is the function of the Read/ Write Logic block of the 8257 DMA controller?
6. Mention the function of each signal of the control logic block of 8257 DMA controller.
7. What is the function of Mode Set Register of the 8257? How 8257 is programmed in Auto Load option?
8. How 8257 is programmed in Rotating Priority option?
9. Give and discuss the format of the Status Word Register of 8257.
10. How the programming of 8257 DMA controller is done?
11. Give the address decoder circuit for addresses of various registers as C0 H to C8 H. The decoder circuit enables chip select terminal ( $\overline{CS}$ ) of the 8257.
12. What steps are carried out for initialization of 8257?
13. Draw and discuss the interfacing circuit of 8257 with 8085A microprocessor.
14. Initialize 8257 DMA controller to transfer 4K bytes of data stored in memory locations starting at 2501 H to an output device connected to the channel 2 of 8257. Assume that the address of the Mode Set Register is C8 H. The addresses of DMA address register for channel 2 and the Terminal Count Register are C4 H and C5 H respectively.

15. Initialize 8257 DMA controller to transfer 200 bytes of data stored in memory locations starting at 2000 H to an output device connected to the channel 0 of 8257. Assume that the address of the Mode Set Register is 28 H. The addresses of DMA address register for channel 0 and the Terminal Count Register are 20 H and 21 H respectively.
  16. The DMA controller 8257 should be initialized as follows:
    - DMA channel 2 must be enabled and terminal count stop bit is to be enabled.
    - 2500 H must be written in DMA address register channel 2.
    - 500 (decimal) must be written in Terminal counter of channel 2 and D14 and D15 are set for memory read operation.

Assume the address of the DMA address register for channel 2 is 94 H and the terminal count is 95 H; and the address of the mode set register is 98 H.
  17. Initialize 8257 DMA controller to transfer 8K bytes of data from a peripheral to memory locations starting at 21FF H through channel 1. Assume that the address of the Mode Set Register is 98 H. Assume the addresses of DMA address register for channel 1 and the Terminal Count Register are 92 H and 93 H respectively.
  18. An I/O device which is associated with channel 2 of 8257 DMA controller is to be periodically refreshed with 1K bytes of data starting from 2500 H locations. Write the initialization routine for the DMA controller to be operated with auto-load and extended write operations. The addresses of DMA registers and terminal count registers of all the four channels are C0 H to C7 H; and the address of the mode set register is CH H.
-

# 13

## Interfacing Data converters: A/D and D/A Converters

---

Sometimes the information available for processing in microprocessor based system is in digital form while in most of the cases it is available in analog form. For example, the outputs of digital voltmeter, digital frequency meter, digital clock and calculators etc. are available in digital form but most physical quantities such as temperature, pressure, light, voltage and current etc. gives information in analog form. It is often necessary to convert information in one form to another form for the purpose of interfacing with the system. For example, to design the microprocessor base temperature controller, the temperature of the device obtained from the transducer such as thermocouple or thermister is first converted to the digital form using D/A converter then interfaced with the microprocessor. Similarly, for plotting the output of a system on a curve plotter or X-Y recorder, the digital output is first converted to analog output with the help of digital to analog converter, the output of which drives a servomotor. So analog to digital (A/D) converters or digital to analog (D/A) converters are the interfacing devices with the system. In this chapter various types of A/D and D/A converters and their interfacing with the microprocessor will be discussed.

### 13.1 DIGITAL TO ANALOG CONVERTER

Digital to Analog (D/A) converter converts the digital information into analog form. The input may be of  $n$ -bit long having different voltage levels. So in the D/A converters, some method is to be used which can convert this voltage level of  $n$ -bits to its equivalent analog form. This can be accomplished by using different resistive networks. Following two types of resistive networks are basically used for this purpose:

1. Resistive Divider Network or weighted resistor network
2. Binary Ladder Network or R-2R network

The converter which comprises the resistive divider network is known as Resistive Divider D/A converter and the D/A converter which comprises the binary ladder network is known as Binary Ladder D/A converter. These converters will now be discussed.

#### 13.1.1 Resistive Divider D/A converter

As discussed above, the resistive divider D/A converter consists of a resistive divider network, so before discussing the complete circuit diagram of a resistive divider D/A converter, it is better to understand the working of resistive divider network. The resistive divider network changes each of the  $n$ -bit digital level into its equivalent analog

output. The discussion will now be made for the method of converting the  $n$ -bit digital input to its equivalent analog signal. A weight is assigned to each bit of  $n$ -bit digital input in such a way that the sum of weight must be equal to 1. In general, the binary weight assigned to LSB in an  $n$ -bit digital input is  $\frac{1}{2^n - 1}$ . The weights assigned to 2<sup>nd</sup> LSB, 3<sup>rd</sup> LSB, 4<sup>th</sup> LSB and so on are obtained by multiplying the weights of LSB to  $2^1 (=2)$ ,  $2^2 (=4)$ ,  $2^3 (=8)$ .... respectively. For instance, weights assigned to different bits of 4-bit binary input  $b_3 b_2 b_1 b_0$  are:

|  |                                      |
|--|--------------------------------------|
| Weight assigned to LSB ( $b_0$ bit) is                 | $\frac{2^0}{2^4 - 1} = \frac{1}{15}$ |
| Weight assigned to 2 <sup>nd</sup> LSB ( $b_1$ bit) is | $\frac{2^1}{2^4 - 1} = \frac{2}{15}$ |
| Weight assigned to 3 <sup>rd</sup> LSB ( $b_2$ bit) is | $\frac{2^2}{2^4 - 1} = \frac{4}{15}$ |
| Weight assigned to MSB ( $b_3$ bit) is                 | $\frac{2^3}{2^4 - 1} = \frac{8}{15}$ |

The sum of weights assigned to each bit of 4-bit digital input is 1 as  $\frac{1}{15} + \frac{2}{15} + \frac{4}{15} + \frac{8}{15} = 1$ .

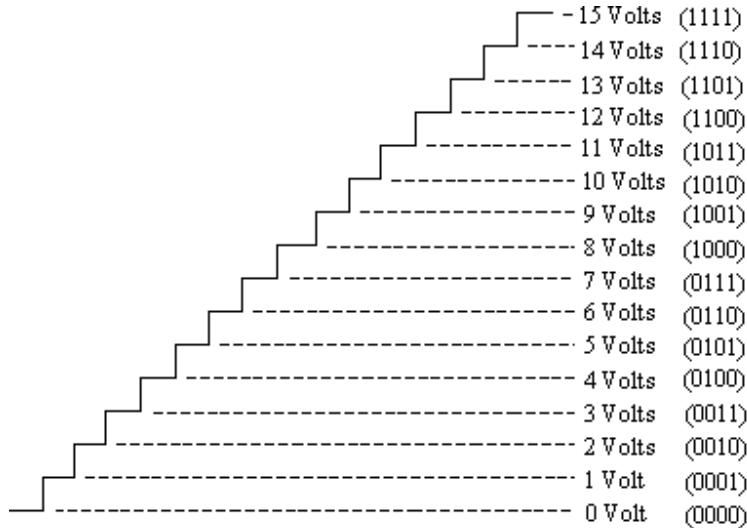
In a four bit binary system there will be 16 different possible input combinations, corresponding to which the analog signal will be obtained if it is assumed that a certain reference voltage ( $V_{REF}$ ) is applied whenever there is a 1 in binary bit. In a 4 bit digital system if  $V_{REF} = 15$  volts, the analog voltage available for each combination of binary input should be as given in table 13.1.

Table 13.1

| <b><math>b_3</math></b> | <b><math>b_2</math></b> | <b><math>b_1</math></b> | <b><math>b_0</math></b> | <b>Weight</b> | <b>Analog Voltage</b>      |
|-------------------------|-------------------------|-------------------------|-------------------------|---------------|----------------------------|
| 0                       | 0                       | 0                       | 0                       | 0/15          | $(0/15)V_{REF} = 0$ Volt   |
| 0                       | 0                       | 0                       | 1                       | 1/15          | $(1/15)V_{REF} = 1$ Volt   |
| 0                       | 0                       | 1                       | 0                       | 2/15          | $(2/15)V_{REF} = 2$ Volt   |
| 0                       | 0                       | 1                       | 1                       | 3/15          | $(3/15)V_{REF} = 3$ Volt   |
| 0                       | 1                       | 0                       | 0                       | 4/15          | $(4/15)V_{REF} = 4$ Volt   |
| 0                       | 1                       | 0                       | 1                       | 5/15          | $(5/15)V_{REF} = 5$ Volt   |
| 0                       | 1                       | 1                       | 0                       | 6/15          | $(6/15)V_{REF} = 6$ Volt   |
| 0                       | 1                       | 1                       | 1                       | 7/15          | $(7/15)V_{REF} = 7$ Volt   |
| 1                       | 0                       | 0                       | 0                       | 8/15          | $(8/15)V_{REF} = 8$ Volt   |
| 1                       | 0                       | 0                       | 1                       | 9/15          | $(9/15)V_{REF} = 9$ Volt   |
| 1                       | 0                       | 1                       | 0                       | 10/15         | $(10/15)V_{REF} = 10$ Volt |
| 1                       | 0                       | 1                       | 1                       | 11/15         | $(11/15)V_{REF} = 11$ Volt |
| 1                       | 1                       | 0                       | 0                       | 12/15         | $(12/15)V_{REF} = 12$ Volt |
| 1                       | 1                       | 0                       | 1                       | 13/15         | $(13/15)V_{REF} = 13$ Volt |
| 1                       | 1                       | 1                       | 0                       | 14/15         | $(14/15)V_{REF} = 14$ Volt |
| 1                       | 1                       | 1                       | 1                       | 15/15         | $(15/15)V_{REF} = 15$ Volt |

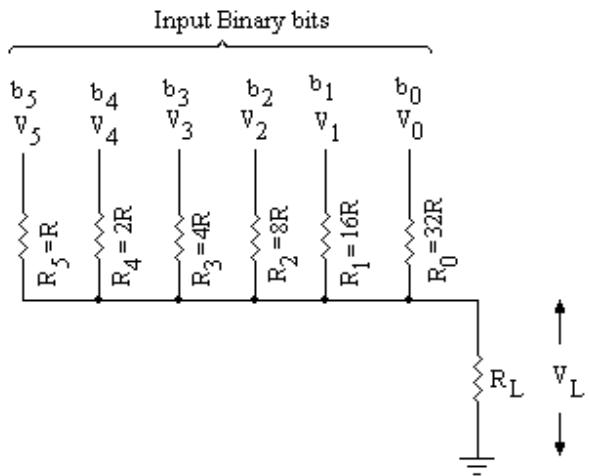
So the analog voltage for binary word = (weight of the binary word) x  $V_{REF}$

It may be noted from this table 13.1 that the analog voltage corresponding to binary equivalent is discrete step value as given in figure 13.1. The discrete step is of 1 volt if  $V_{REF}$  is assumed to be 15 volts in a four bit digital input. The step voltage (analog) will be dependent on the reference voltage. There will, however, be  $2^n$  steps in  $n$ -bit digital system.



**Fig. 13.1**

Resistive divider network is used for converting digital inputs to analog outputs. The network for 6 bit binary system shown in figure 13.2 is known as the weighted network, as the resistors are weighted inversely with their current values. The input binary bits are  $b_5 b_4 b_3 b_2 b_1 b_0$  where  $b_0$  is the LSB and  $b_5$  is MSB. These binary bits may be logic 0 or 1. Logic 0 may further be assumed as 0 volt and logic 1 as  $V_{REF}$ . So  $V_0, V_1, V_2, V_3, V_4$  and  $V_5$  are the input voltage levels which may be 0 volt or  $V_{REF}$  depending on the binary bits. The resistors  $R_0, R_1, R_2, R_3, R_4$  and  $R_5$  are connected to bits  $b_0, b_1, b_2, b_3, b_4, b_5$  respectively. It may be noted from this network that the resistor connected to the binary bit is half the value of resistor connected to the previous (lower) bit. Hence this network also called as the resistive divider network. Let  $R_L$  is the load resistance which is supposed to very high i.e. very much higher than the resistor  $R_0$ .



**Fig. 13.2**

Now the voltage  $V_L$  across the load resistance  $R_L$  can be obtained by using Millman's theorem. This theorem states that the voltage appearing at any node in a resistive network is equal to the sum of all the currents that would enter to the node divided by the sum of conductances connected to that node.

$$\begin{aligned}
 \text{Thus } V_L &= \frac{\frac{V_5}{R_5} + \frac{V_4}{R_4} + \frac{V_3}{R_3} + \frac{V_2}{R_2} + \frac{V_1}{R_1} + \frac{V_0}{R_0}}{\frac{1}{R_5} + \frac{1}{R_4} + \frac{1}{R_3} + \frac{1}{R_2} + \frac{1}{R_1} + \frac{1}{R_0}} \\
 &= \frac{\frac{V_5}{R} + \frac{V_4}{2R} + \frac{V_3}{4R} + \frac{V_2}{8R} + \frac{V_1}{16R} + \frac{V_0}{32R}}{\frac{1}{R} + \frac{1}{2R} + \frac{1}{4R} + \frac{1}{8R} + \frac{1}{16R} + \frac{1}{32R}} \\
 &= \frac{[32V_5 + 16V_4 + 8V_3 + 4V_2 + 2V_1 + V_0]}{32} \\
 &= \frac{V_0 + 2V_1 + 4V_2 + 8V_3 + 16V_4 + 32V_5}{63} \\
 &= \frac{1}{(2^6 - 1)} (2^0 V_0 + 2^1 V_1 + 2^2 V_2 + 2^3 V_3 + 2^4 V_4 + 2^5 V_5) \quad \dots(13.1)
 \end{aligned}$$

In this equation (13.1), the load resistance  $R_L$  is not considered as it is assumed to be large enough offering low (almost zero) conductance. From this equation it is clear that if the input binary bits are all 1 (in a six bit system) and reference voltage  $V_{REF} = 6.4$  volts (say), the  $V_L$  is given by:

$$V_L = \frac{1}{63} \times 63 V_{REF} = 6.4 \text{ volts}$$

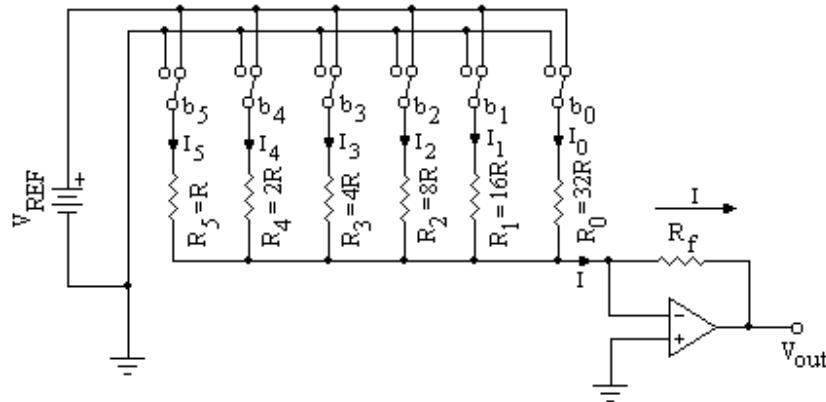
In general, the equation (13.1) for output voltage of  $n$ -bit binary digits is given as:

$$V_L = \frac{1}{(2^n - 1)} (2^0 V_0 + 2^1 V_1 + 2^2 V_2 + 2^3 V_3 + 2^4 V_4 + \dots + 2^{n-1} V_{n-1}) \quad \dots(13.2)$$

The output of this network is as per our requirement, and is proportional to the input binary data.

Using the network discussed above, a D/A converter (called binary weighted D/A converter or Resistive divider D/A converter) can be designed as given below. The schematic diagram of 6-bit D/A converter is shown in figure 13.3. It consists of the following major parts:

- (i)  $n$  switches, one for each bit applied to the input,
- (ii) A binary weighted resistive network which changes each of the digital level into equivalent binary weighted voltage or current.
- (iii) A reference voltage source  $V_{REF}$ .
- (iv) A summing amplifier that adds the currents flowing in the resistors of the network to develop a signal that is proportional to the digital input.



**Fig. 13.3**

In this circuit, one switch is connected to each binary bit. In fact these switches are such that when the binary bit is 0, the corresponding resistor of the network gets connected to the ground potential and when the binary bit is 1, the corresponding resistor of the network gets connected to the  $V_{REF}$  volt. The current flowing through any branch of the network will be the logical voltage (0volt or  $V_{REF}$  volts) divided by the corresponding resistor.

So the total current  $I$  will be given by (ref. fig. 11.3):

$$\begin{aligned} I &= \frac{V_5}{R_5} + \frac{V_4}{R_4} + \frac{V_3}{R_3} + \frac{V_2}{R_2} + \frac{V_1}{R_1} + \frac{V_0}{R_0} \\ &= \frac{V_5}{R} + \frac{V_4}{2R} + \frac{V_3}{4R} + \frac{V_2}{8R} + \frac{V_1}{16R} + \frac{V_0}{32R} \end{aligned}$$

Since the voltages  $V_5$  through  $V_0$  are either 0 or  $V_{REF}$  volts depending upon the bit value, so it is customary to take common voltage  $V_{REF}$  and bits are kept in place of voltages. So  $V_5$  is replaced by  $V_{REF} \cdot b_5$ ,  $V_4$  by  $V_{REF} \cdot b_4$  and so on; the bits  $b_5, b_4, b_3$  etc will be 0 or 1. The current  $I$  may, therefore, be represented as follows:

$$I = \frac{V_{REF}}{32R} [32b_5 + 16b_4 + 8b_3 + 4b_2 + 2b_1 + b_0]$$

$$= \frac{V_{REF}}{2^5.R} [2^0b_0 + 2^1b_1 + 2^2b_2 + 2^3b_3 + 2^4b_4 + 2^5b_5]$$

This is the equation of current  $I$  for 6 input bits. The general equation of current  $I$  for  $n$  input bits is given by:

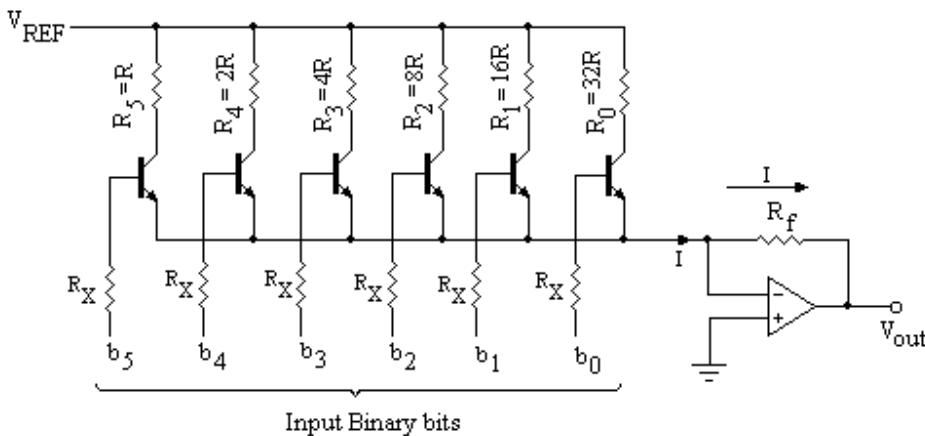
$$I = \frac{V_{REF}}{2^{n-1}.R} [2^0b_0 + 2^1b_1 + 2^2b_2 + 2^3b_3 + 2^4b_4 + \dots + 2^{n-1}b_{n-1}] \quad \dots(13.3)$$

The voltage at the output of operational amplifier will be given by:

$$V_{out} = -R_f.I$$

The resistor  $R_f$  is the feed back resistance in the operational amplifier. The output voltage of the operational amplifier is proportional to input binary data.

The switches connected in figure 13.3 can be replaced by the electronic switches (transistorized) as shown in figure 13.4. When the bit is at logic 1, the corresponding transistor conducts and the current flows through the collector resistor as required; and when the bit is at logic 0 the transistor goes into cutoff and no collector current flows.



**Fig. 13.4**

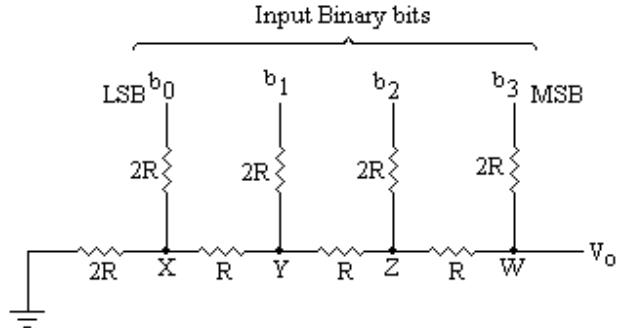
This D/A converter is economical and simple method to design but suffers the following serious drawbacks:

1. The network in this D/A converter is constructed using the precession resistors and resistors have different values. So it is difficult in practice to choose the resistors with accuracy and stability.
2. When the number of bits in the network is large, then the current from the source will be large enough. The current in the MSB branch (resistor) will be much larger than LSB branch. In a 10 bit D/A converter, the current in MSB branch will be 512 times larger than the MSB branch.

### 13.1.2 Binary Ladder D/A Converter

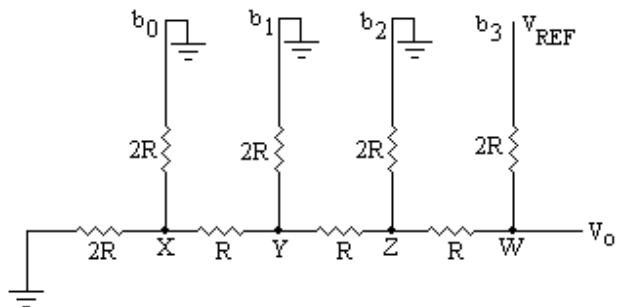
A more commonly used D/A converter is a binary ladder D/A converter, which removes the drawbacks discussed in resistive divider D/A converter. This type of D/A converter contains an R-2R resistive ladder network. The R-2R resistive ladder network will now be discussed, which gives the output a weighted sum of digital inputs. Such a ladder

network for 4-bit input is shown in figure 13.5. This network is constructed having only two resistor values i.e.  $R$  and  $2R$ . In this network  $b_0$ ,  $b_1$ ,  $b_2$  and  $b_3$  are the input binary bits and  $b_0$  is the LSB and  $b_3$  is MSB. Any of these bits will be at the ground potential when the corresponding bit is at logic 0 or at the reference potential ( $V_{REF}$ ) when the input bit is at logic 1.

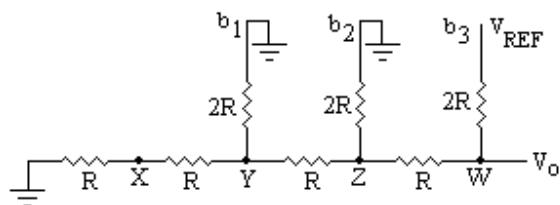


**Fig. 13.5**

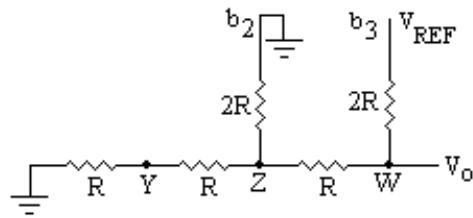
To examine the behaviour of this network, it is assumed that the bit  $b_3$  is at logic 1 (or  $V_{REF}$  potential) as shown in figure 13.6(a). The output voltage corresponding to MSB may be calculated as follows. The equivalent resistance at the point X is the parallel combination of two resistances each having the value of  $2R$ . So the equivalent resistance looking at point X and ground is  $R$  as shown in figure 13.6(b). At the point Y again there is a parallel combination of two  $2R$  resistances; the equivalent resistance looking at the point Y and ground is  $R$  as shown in figure 13.6(c). Similarly, one can find the equivalent resistance looking at the point Z and ground is  $R$  as shown in figure 13.6(d).



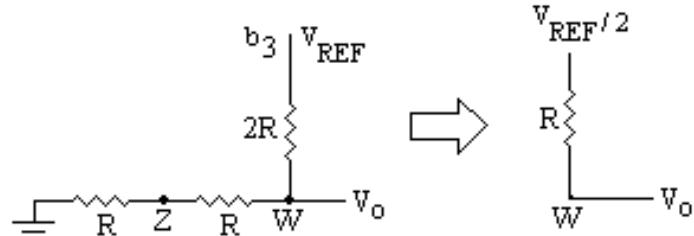
(a)



(b)



(c)



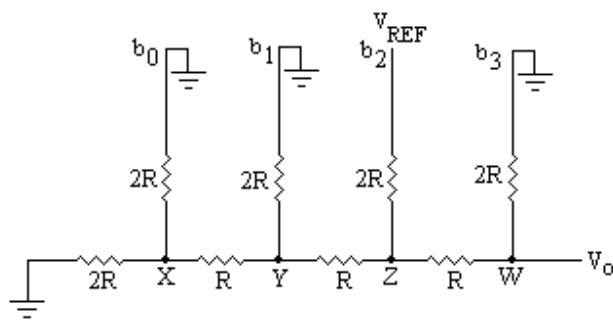
(d)

Fig. 13.6

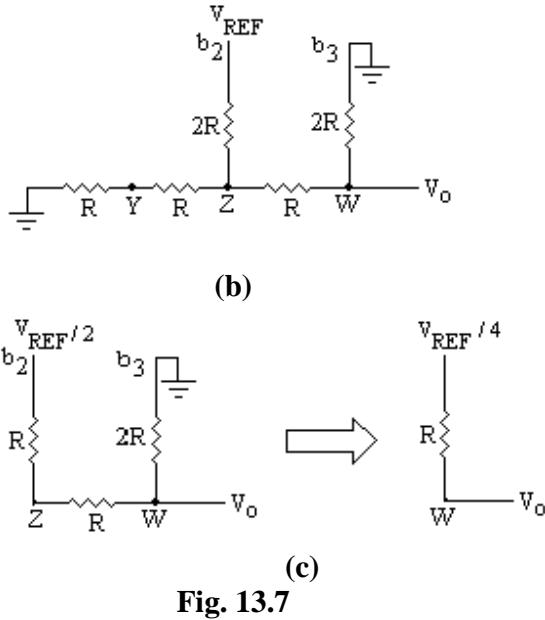
From figure 13.6(d) it is clear that the resistance looking at the point W and ground is  $2R$ , and the resistance looking towards the bit  $b_3$  is also  $2R$ . Thus the output voltage at the point W due to bit  $b_3$  (MSB) assumed at  $V_{REF}$  potential is given by:

$$V_0 = \frac{V_{REF} \times 2R}{(2R + 2R)} = \frac{V_{REF}}{2}$$

The output voltage  $V_0$  due to the binary input 1000 (only MSB is high) is half of the reference voltage having Thevenin's resistance  $R$  in series with it. Similarly one can calculate the output voltage due to the binary input 0100 (i.e. second MSB); the network for this case is shown in figure 13.7(a). The resistance looking at the point Y and ground is  $R$  as shown in figure 13.7(b). The resistance between the point Z and ground is  $2R$ . The voltage at point Z and ground is ( $V_{REF}/2$ ) have a Thevenin's resistance  $R$ , as shown in figure 13.7(c).



(a)



(c)  
**Fig. 13.7**

From this figure the output voltage  $V_0$  at the point W is given by:

$$V_0 = \frac{(V_{REF}/2)x2R}{(2R+2R)} = \frac{V_{REF}}{4}$$

So the output voltage due to second MSB (or for binary input 0100) is  $\frac{V_{REF}}{4}$  with Thevenin's resistance R in series with it.

It can further be shown that the output due to third MSB (for binary input 0010) is  $\frac{V_{REF}}{8}$ . And for LSB (0001 binary input) the output is  $\frac{V_{REF}}{16}$ . Each voltage source will have Thevenin's resistance R in series with the source. The total output voltage in analog form, due to all the inputs as 1 (for 1111) can easily be found by adding the outputs obtained for each bit as given below:

$$V_0 = \frac{V_{REF}}{2} + \frac{V_{REF}}{4} + \frac{V_{REF}}{8} + \frac{V_{REF}}{16}$$

It may be noted that  $\frac{V_{REF}}{2}$  is the voltage due to MSB,  $\frac{V_{REF}}{4}$  due to second MSB,  $\frac{V_{REF}}{8}$  for third MSB and  $\frac{V_{REF}}{16}$  for LSB. So to distinguish these voltages it is useful to write the bit positions along with  $V_{REF}$  as given below. So if the bit is 0 the voltage corresponding to that bit will be zero otherwise the voltage as discussed above.

$$\begin{aligned} V_0 &= \frac{V_{REF}xb_3}{2} + \frac{V_{REF}xb_2}{4} + \frac{V_{REF}xb_1}{8} + \frac{V_{REF}xb_0}{16} \\ &= \frac{V_{REF}}{16}[8xb_3 + 4xb_2 + 2xb_1 + 1xb_0] \end{aligned}$$

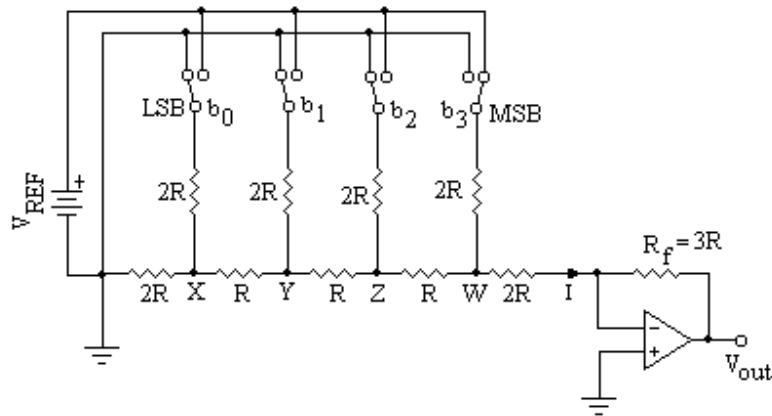
$$= \frac{V_{REF}}{2^4} [2^0 xb_0 + 2^1 xb_1 + 2^2 xb_2 + 2^3 xb_3] \quad \dots(13.4)$$

The equation (13.4) is the equation for voltage at the output of 4 bit binary ladder network. A general equation for the output of  $n$ -bit binary data can be given as follows:

$$V_0 = \frac{V_{REF}}{2^n} [2^0 xb_0 + 2^1 xb_1 + 2^2 xb_2 + 2^3 xb_3 + \dots + 2^{n-1} xb_{n-1}] \quad \dots(13.5)$$

The output of this network is proportional to the input binary data. So using this R-2R ladder network, a D/A converter (called binary ladder D/A converter) can be designed as given below. The schematic diagram of 4-bit D/A converter is shown in figure 13.8. It consists of the following major parts.

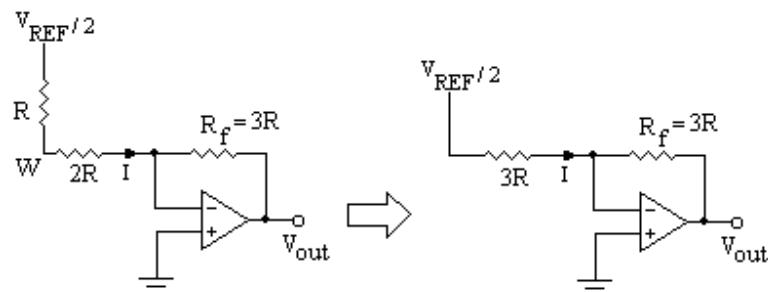
- (i)  $n$  switches, one for each bit applied to the input,
- (ii) A binary ladder network which changes each of the digital level into equivalent binary weighted voltage or current.
- (iii) A reference voltage source  $V_{REF}$ .
- (iv) A summing amplifier that adds the currents flowing in the resistors of the network to develop a signal that is proportional to the digital input.



**Fig. 13.8**

The output voltage  $V_{out}$  of this D/A converter due to MSB (1000 binary input) will be calculated as given below:

The voltage at the point W due to MSB is  $V_{REF}/2$  having a Thevenin's resistance  $R$  in series with it as discussed above and is shown in figure 13.9



**Fig. 13.9**

From this figure, the current  $I$  is given by:

$$I = \frac{V_{REF}}{2} \left( \frac{1}{3R} \right)$$

and the output voltage  $V_{out}$  is given by:

$$V_{out} = -I \cdot R_f$$

$$= -\frac{V_{REF}}{2} \left( \frac{1}{3R} \right) \cdot 3R = -\frac{V_{REF}}{2}$$

The output voltage is the same as calculated in equation 13.5, with the difference that it has a negative value because the operational amplifier is used in inverting configuration.

Note that the resistors in the ladder network are either  $R$  or  $2R$ . It is the ratio of resistances matters rather than the absolute value of resistances. Further the resistors do not cover a wide range of magnitude; it is, therefore, practically possible to get the precision in the ratio of their magnitudes. The temperature coefficients of these resistances can easily match. Because of these advantages, the ladder network is widely used in D/A converters.

### 13.2 PERFORMANCE CRITERIA FOR D/A CONVERTER

The D/A converters are available in the form of ICs with different specifications for their performances. So before discussing D/A converter ICs it will be better to discuss first the characteristics of the converters specified by the manufacturers. These specifications include:

1. Resolution
2. Accuracy
3. Monotonicity
4. Settling time

**1. Resolution:** As discussed above, the analog output of D/A converter is proportional to the digital input (binary data), so a perfect staircase is obtained if there is an LSB increment. The resolution is, therefore, a measure of quality of D/A converter, which is defined as the ratio of the LSB increment to the maximum output. For an  $n$ -bit D/A converter the resolution is given by:

The change in output due to LSB increment for  $n$ -bit digital input (Step size)

$$\begin{aligned} &= \text{Full scale output} / \text{No. of steps} \\ &= \frac{\text{Full scale output}}{2^n - 1} \end{aligned}$$

where  $(2^n - 1)$  is the number of steps for  $n$ -bit D/A converter.

$$\begin{aligned} \text{Percentage Resolution} &= \frac{\text{Full scale output} / (2^n - 1)}{\text{Full scale output}} \times 100\% \\ &= \frac{1}{2^n - 1} \times 100\% \end{aligned}$$

The step size for a 10 bit D/A converter, having full scale output voltage as 10 volts, is given by

$$= \frac{10}{2^{10} - 1} = \frac{10}{1023} = 9.8mV$$

And % Resolution

$$= 0.0978\%$$

**2. Accuracy:** Accuracy of a D/A converter is the closeness of the output analog voltage to the expected theoretical output. In a linear variation of analog output with digital input, the relative accuracy is the maximum deviation of the D/A output compared with the linear behaviour. It is expressed as a percent of a full-scale or maximum output voltage. For example, if a converter has a full scale output of 10 V and the accuracy is  $\pm 0.1\%$ , then the maximum error for any output voltage is  $(10V)(0.001) = 10 \text{ mV}$ . Ideally, the accuracy should be at most  $\pm \frac{1}{2}$  of an LSB.

For an 8 bit D/A converter, one LSB is  $\frac{1}{256} = 0.0039 = 0.39\%$  of full scale. The accuracy should be approximately  $\pm 0.2\%$ .

**3. Monotonicity:** A D/A converter is said to be monotonic if it gives an analog output voltage which increases regularly and linearly with increase in input digital signal. Such a quality of the converter is called as monotonicity. In order to demonstrate monotonicity of a D/A converter, a counter output is given as digital input to a D/A converter and the analog output is displayed on the CRO. Monotonicity then requires that the output waveform should be a perfect staircase waveform with steps equally spaced and of same magnitude. If the steps are missing or have varying magnitude, the D/A converter is defective.

**4. Settling Time:** After the application of digital input to a D/A converter, it takes about few nanoseconds to microseconds to produce the correct output. So the settling time is defined as the time the converter takes to give an output to settle within  $\pm \frac{1}{2}$  LSB of its final value. For example, if a D/A converter has a resolution of 10mV, the settling time is the measure of the time the converter takes to settle with in  $\pm 5\text{mV}$  of its final value. Figure 13.10 illustrates the settling time in a D/A converter. The settling time is important because it places a limit on how fast one can change the digital input. The settling time depends on the stray capacitance, saturation delay time, and other factors.

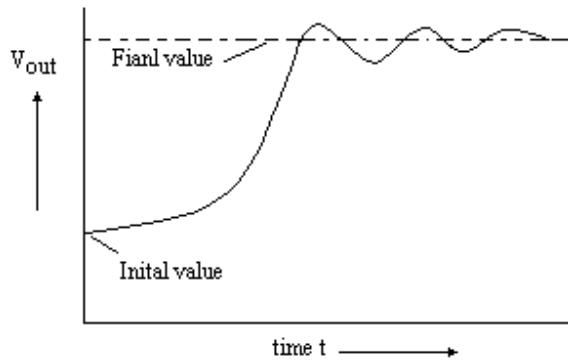
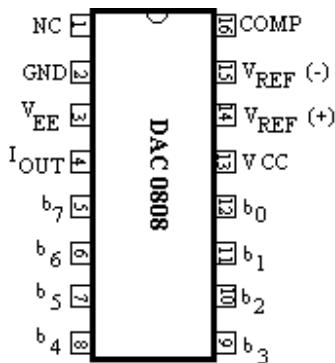


Fig. 13.10

### 13.3 D/A CONVERTER IC 0808

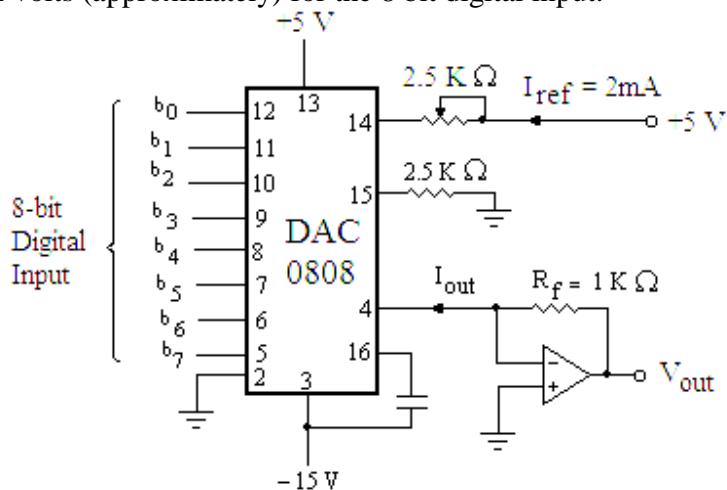
There are many commercially available D/A converter ICs. The IC 0808 is the most popular, inexpensive and widely used 8 bit D/A converter. It contains a reference current source, an R-2R binary ladder network and 8 transistor switches to steer the binary currents to the network. Figure 13.11 shows the pin configuration of this D/A converter IC 0808.



**Fig. 13.11**

In this IC, pins 5 through 12 are the 8 bit input data, so should be connected to input data bits. Pin 15 is to be connected to ground through a resistance. Pin 13 is to be connected to +5 volt supply. Pin 3 ( $V_{EE}$ ) is to be connected to -15 volts. Pin 4 is the output current of the ladder network should be connected to the operational amplifier. Pin 2 is the ground pin. The pin 16 is the frequency compensation pin, a capacitor between pin 16 and 3 is to be connected for this purpose.

A circuit diagram to get the analog output voltage corresponding to 8 bit digital input is shown in figure 13.12. A +5 V supply sets up a reference current of 2mA for the ladder. The output current  $I_{out}$  drives the operational amplifier to give final output between 0 and 2 volts (approximately) for the 8 bit digital input.



**Fig. 13.12**

There are many other commercially available D/A converter as given below:

DAC 0800 – A monolithic 8-bit high speed current output DAC.

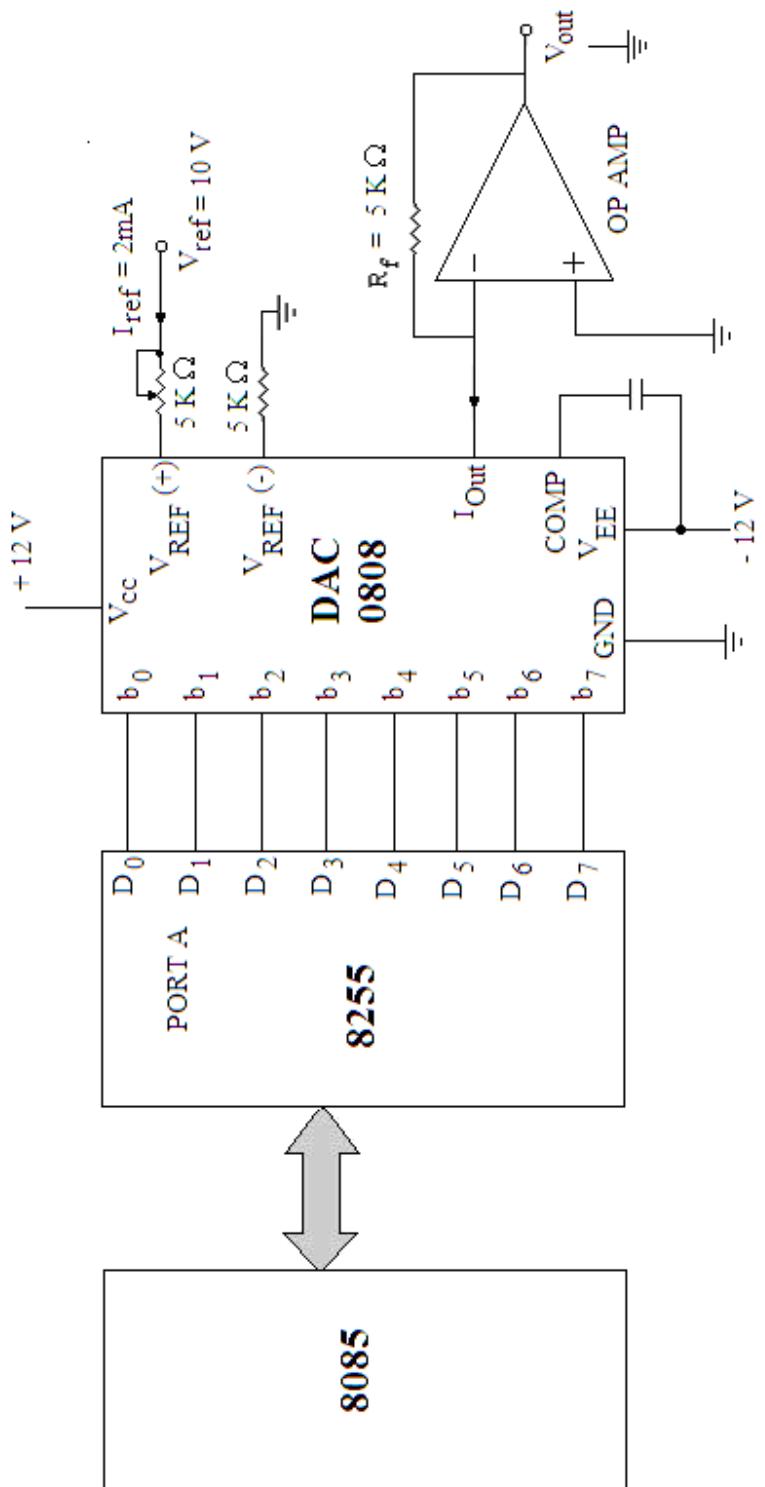
DAC 0806 and DAC 0807 – 8 bit monolithic D/A converters.

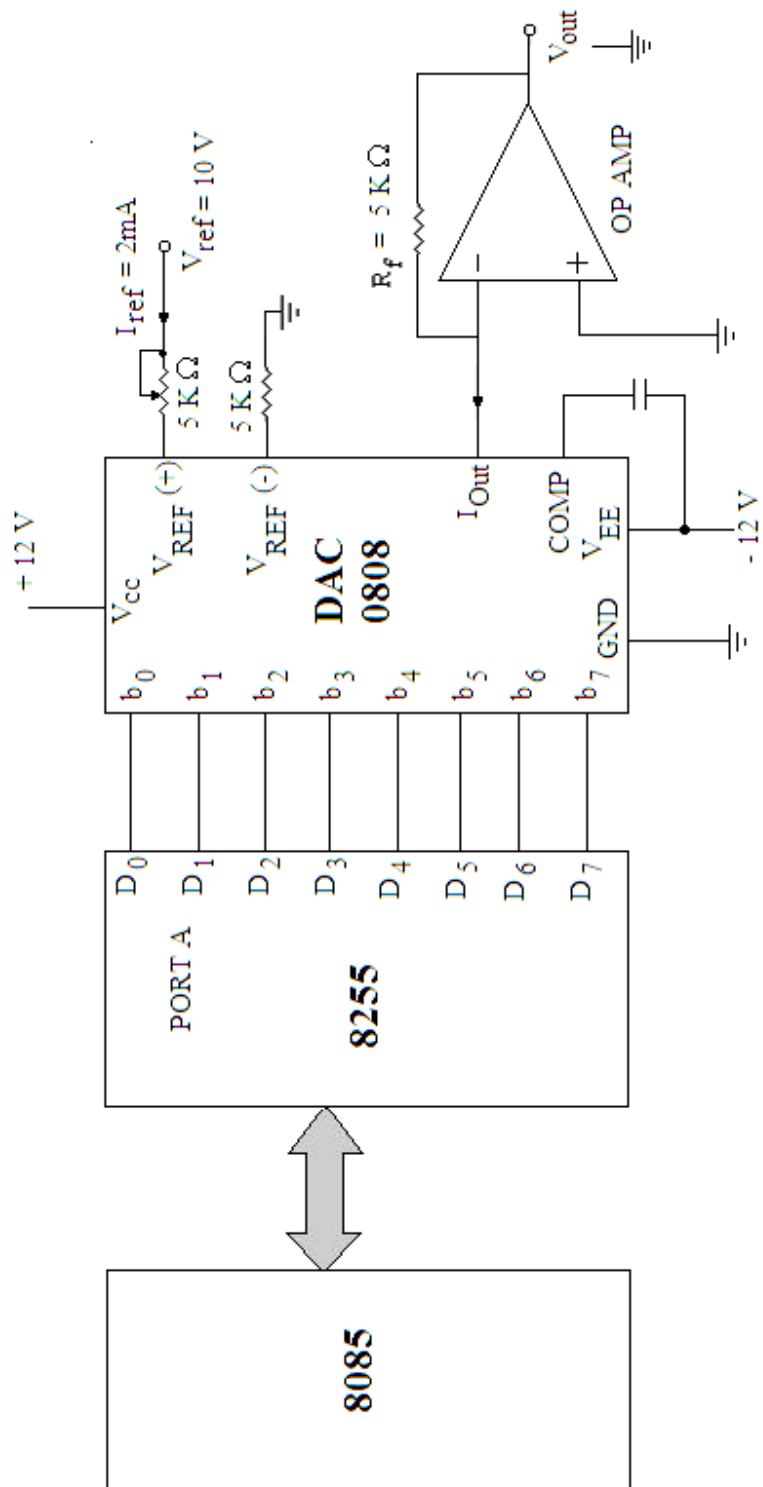
DAC 1000 and DAC 1008 – 10 bit microprocessor compatible advanced CMOS D/A converters.

DAC 1202 and DAC 1203 – Three-digit (BCD) D/A converter.

#### 13.4 INTERFACING OF D/A CONVERTER

For the interfacing of 0808 D/A converter with the microprocessor 8085A, the circuit shown in figure 13.12 may be connected to the system through the 8255 PPI (Programmable Peripheral Interface as shown in figure 13.13.





**Fig. 13.13**

In this circuit feedback resistor  $R_f$  is considered as  $5 \text{ K}\Omega$  and the reference voltage is taken as  $V_{REF} = 10 \text{ V}$ , so that we get the output current  $I_{Out}$  as:

$$I_{Out} = \frac{I_{REF}}{2^n} [2^0 xb_0 + 2^1 xb_1 + 2^2 xb_2 + 2^3 xb_3 + \dots + 2^{n-1} xb_{n-1}]$$

Here the value of  $n = 8$  as it is 8 bit D/A converter.

So  $I_{Out} = \frac{I_{REF}}{256} [2^0 xb_0 + 2^1 xb_1 + 2^2 xb_2 + 2^3 xb_3 + \dots + 2^{n-1} xb_{n-1}]$

and  $I_{REF} = \frac{V_{REF}}{R_{REF}} = \frac{10V}{5K\Omega} = 2mA$

and  $V_{OUT} = \frac{2mA}{256} [2^0 xb_0 + 2^1 xb_1 + 2^2 xb_2 + 2^3 xb_3 + \dots + 2^{n-1} xb_{n-1}] \times R_f$   
 $= \frac{10}{256} [2^0 xb_0 + 2^1 xb_1 + 2^2 xb_2 + 2^3 xb_3 + \dots + 2^{n-1} xb_{n-1}]$

If all the input bits ( $b_0 - b_7$ ) are 1 (FF H), then the output voltage (known as full scale output voltage) is given by:

$$V_{OUT} = \frac{255}{256} \times 10 \approx 10V$$

The output voltage corresponding to the input 1000 0000 (80 H) is given by:

$$V_{OUT} = \frac{128}{256} \times 10 = 5V$$

So we have the linear output corresponding to the binary inputs i.e.

| Digital Input | Output Voltage |
|---------------|----------------|
| 00 H          | 0 V            |
| 80 H          | 5 V            |
| FF H          | 10 V           |

These input output levels may be verified, if the following three programs are executed and the voltages at the outputs in the three cases are measured.

Program 1:

|             |  |
|-------------|--|
| MVI A, 80 H | ; 8255 is initialized with all the ports as output port.                                       |
| OUT 03 H    | ; Control word is written in control word register.  |
| MVI A, 00 H | ; Get A = 00 H, so that 00 H is applied to the input of D/A converter.                         |
| OUT 00 H    | ; 00 H is available at the Port A of 8255 so that it is applied to the input of D/A converter. |
| HLT         | ; Stop processing.   |

After execution of this program, if the voltage at the output of the circuit of D/A converter is measured we get 0 V. It verifies that 00 H is applied at the input of the converter, we get 0 V.

Program 2:

|             |  |
|-------------|--|
| MVI A, 80 H | ; 8255 is initialized with all the ports as output port. |
|-------------|--|

|             |  |
|-------------|--|
| OUT 03 H    | ; Control word is written in control word register.  |
| MVI A, 80 H | ; Get A = 80 H, so that 80 H is applied to the input of D/A converter.                         |
| OUT 00 H    | ; 80 H is available at the Port A of 8255 so that it is applied to the input of D/A converter. |
| HLT         | ; Stop processing.   |

After execution of this program, if the voltage at the output of the circuit of D/A converter is measured we get 5 V. It verifies that 80 H is applied at the input of the converter, we get 5 V.

Program 3:

|             |  |
|-------------|--|
| MVI A, 80 H | ; 8255 is initialized with all the ports as output port.                                       |
| OUT 03 H    | ; Control word is written in control word register.  |
| MVI A, FF H | ; Get A = FF H, so that FF H is applied to the input of D/A converter.                         |
| OUT 00 H    | ; FF H is available at the Port A of 8255 so that it is applied to the input of D/A converter. |
| HLT         | ; Stop processing.   |

After execution of this program, if the voltage at the output of the circuit of D/A converter is measured we get 10 V. It verifies that FF H is applied at the input of the converter, we get 10 V.

**Example 13.1.** *The input bits of the D/A converter is connected to the output pins of Port A of 8255, which is already connected with the microprocessor (ref. fig.13.13). Write a program to generate stair case voltage with ten steps. It should have the constant pulse duration.*

**Solution.** There should be 10 steps for the stair case voltage to be generated with the D/A converter. So the height of the stair case should be approximately 1 volt. Since FF H gives 10 volts, so 19 H should be decreased each time in 10 go.

When 19 H is subtracted from FF H in the first go we get E6 H. The output voltage corresponding to E6 H is given by:

$$V_{\text{out}} = \frac{230}{256} \times 10 = 8.984 \text{ volts}$$

Therefore, the subtraction of 19 H from FF H gives a decrement of 1 volt at the output. The program for the generation of stair voltage is given as:

#### PROGRAM

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
|       | MVI A,    | 80 H    | ; Initialize 8255-I to work all the ports as output ports.              |
|       | OUT       | 03 H    | ; Write the control word (80 H) in the control word register of 8255-I. |
| START | MVI A,    | FF H    | ; Load FF H to accumulator.   |

|                            | OUT       | 00 H    | ; Send FF H (10 V) to PA <sub>0</sub> .  |
|----------------------------|-----------|---------|--|
|                            | CALL      | DELAY   | ; Jump to delay subroutine to introduce a delay of constant pulse width.         |
|                            | MVI B,    | 00 H    | ; Use B-register as counter for 10 steps.  |
| REPEAT                     | SBI       | 19 H    | ; Subtract 19 H to calculate next weight for the output of D/A converter.        |
|                            | OUT       | 00 H    | ; Send the data to the output.   |
|                            | INR B     |         | ; Increment B-register.  |
|                            | CPI       | 0A H    | ; If 10 steps complete then  |
|                            | JZ        | END     | ; Jump to repeat the process.  |
|                            | PUSH PSW  |         | ; Save PSW   |
|                            | PUSH B    |         | ; Save B-C register pair.  |
|                            | CALL      | DELAY   | ; Jump to delay subroutine to introduce a delay of constant pulse width.         |
|                            | POP B     |         | ; Restore the contents of B-C pair.  |
|                            | POP PSW   |         | ; Restore the PSW.   |
|                            | JUMP      | REPEAT  | ; Jump to repeat to output for the next weight.                                  |
| END                        | MVI A,    | 00 H    | ; Store 00 H to the accumulator.   |
|                            | OUT       | 00 H    | ; Outputs for 00 H.  |
|                            | CALL      | DELAY   | ; Jump to delay subroutine to introduce a delay of constant pulse width.         |
|                            | JMP       | START   | ; Repeat for next cycle.   |
| <b>SUBROUTINE PROGRAM:</b> |           |         |  |
| Label                      | Mnemonics | Operand | Comments   |
| DELAY                      | LXI D,    | 0020 H  | ; Loads DE register pair with a 16-bit number.                                   |
| LOOP1                      | DCX D     |         | ; Decrements DE register pair by 1.  |
|                            | MOV A,    | E       | ; Moves the contents of E register to accumulator.                               |
|                            | ORA D     |         | ; ORing of the contents of D and E registers are performed to set the zero flag. |
| JNZ                        |           | LOOP1   | ; If result is not zero than jump to LOOP1.                                      |
|                            | RET       |         | ; Go back to main program.   |

In the subroutine program, DE register pair may be loaded with any other 16-bit number to change the pulse width. The exact time of the pulse width may be calculated as discussed in the delay programs. After the execution of this program, the output may be seen on the CRO. The output will be stair case wave.

**Example 13.2.** Write a program to generate square wave of 1 KHz frequency with a peak voltage of 2.5 volts. Use D/A converter (ref. fig. 13.13) for this purpose. The square wave should be available at the output of the converter.

**Solution.** It is required to generate square wave of 1 KHz frequency, the time period of such wave should be 1msec. During 1 msec the output should be high for 0.5 msec and low for the same time. Time delay can be introduced by using the following subroutine program:

#### SUBROUTINE PROGRAM:

| Label | Mnemonics | Operand | Comments  |
|-------|-----------|---------|---|
| DELAY | LXI D,    | 0040 H  | ;Loads DE register pair with a 16-bit number.                                   |
| LOOP  | DCX D     |         | ;Decrement DE register pair by one.   |
|       | MOV A, E  |         | ;Moves the contents of E register to accumulator.                               |
|       | ORA D     |         | ;ORing of the contents of D and E registers are performed to set the zero flag. |
| JNZ   | LOOP      |         | ;If result ≠ 0 jump to loop   |
| RET   |           |         | ;Go back to main program.   |

Total T-states used for the above sub routine program are given as:

| Mnemonics | T-states |
|-----------|----------|
| LXI       | 10       |
| DCX       | 5        |
| MOV       | 5        |
| ORA       | 4        |
| JNZ LOOP  | 10/7     |
| RET       | 10       |

24 T-states are used for the inner loop and  $10+7+10 = 27$  T-states are used for outer loop.

In this program the execution of loop is for 64 times (as  $0040\text{ H} = 64_{10}$ ). The condition for the check of zero flag can not be applied just after DCX instruction, since no flag gets affected with this instruction. So to check the zero flag ORA instruction affect the zero flag. The zero flag will be set if the contents of both D and E registers are zero.

The time delay introduced by the inner loop is:

$$T_{\text{LOOP}} = 64 \times 24 \times \text{Time of one T-state.}$$

If the system clock frequency is 3 MHz, then

$$\begin{aligned} T_{\text{LOOP}} &= 64 \times 24 \times \frac{1}{3} \mu\text{sec} \\ &= 512 \mu\text{sec} \\ &= 0.5 \text{ msec.} \end{aligned}$$

Delay introduced for outside loop is:

$$T_{out} = 27 \times 1 \times \frac{1}{3} \mu\text{sec}$$

$$= 9 \mu\text{sec}$$

So the total time delay introduced by the above subroutine program is given by:

$$T_{Delay} = 0.5 \text{ msec} + 9 \mu\text{sec}$$

$$\approx 0.5 \text{ msec}$$

Further, it is required that the voltage level of the output should be 2.5 volts, for which the digital input should be 40 H as 80 H gives 5 volts. So the main program is given as:

### MAIN PROGRAM

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>   |
|--------------|------------------|----------------|---|
| START        | MVI A,           | 80 H           | ; Initialize 8255-I to work all the ports as output ports.              |
|              | OUT              | 03 H           | ; Write the control word (80 H) in the control word register of 8255-I. |
|              | MVI A,           | 00 H           | ; Load 00 H to accumulator.   |
|              | OUT              | 00 H           | ; Send 00 H (00 V) to PA <sub>0</sub> .                                 |
|              | PUSH PSW         |                | ; Save PSW  |
|              | CALL             | DELAY          | ; Jump to delay subroutine to introduce a delay of 0.5msec.             |
|              | POP PSW          |                | ; Restore the PSW.  |
|              | MVI A,           | 40 H           | ; Store 40 H to the accumulator.  |
|              | OUT              | 00 H           | ; Outputs for 40 H.   |
|              | PUSH PSW         |                | ; Save PSW  |
| LOOP         | CALL             | DELAY          | ; Jump to delay subroutine to introduce a delay of 0.5msec.             |
|              | POP PSW          |                | ; Restore the PSW.  |
|              | JMP              | START          | ; Repeat the process for next cycle.                                    |

During the execution of this program the wave shape at output of the D/A converter can be checked. It will be a square wave of 1 KHz frequency.

**Example 13.3.** Write a program to generate a triangular wave form using 8255 and DAC 0808 (ref. fig. 13.13).

**Solution.** The program for the same is given below:

### MAIN PROGRAM

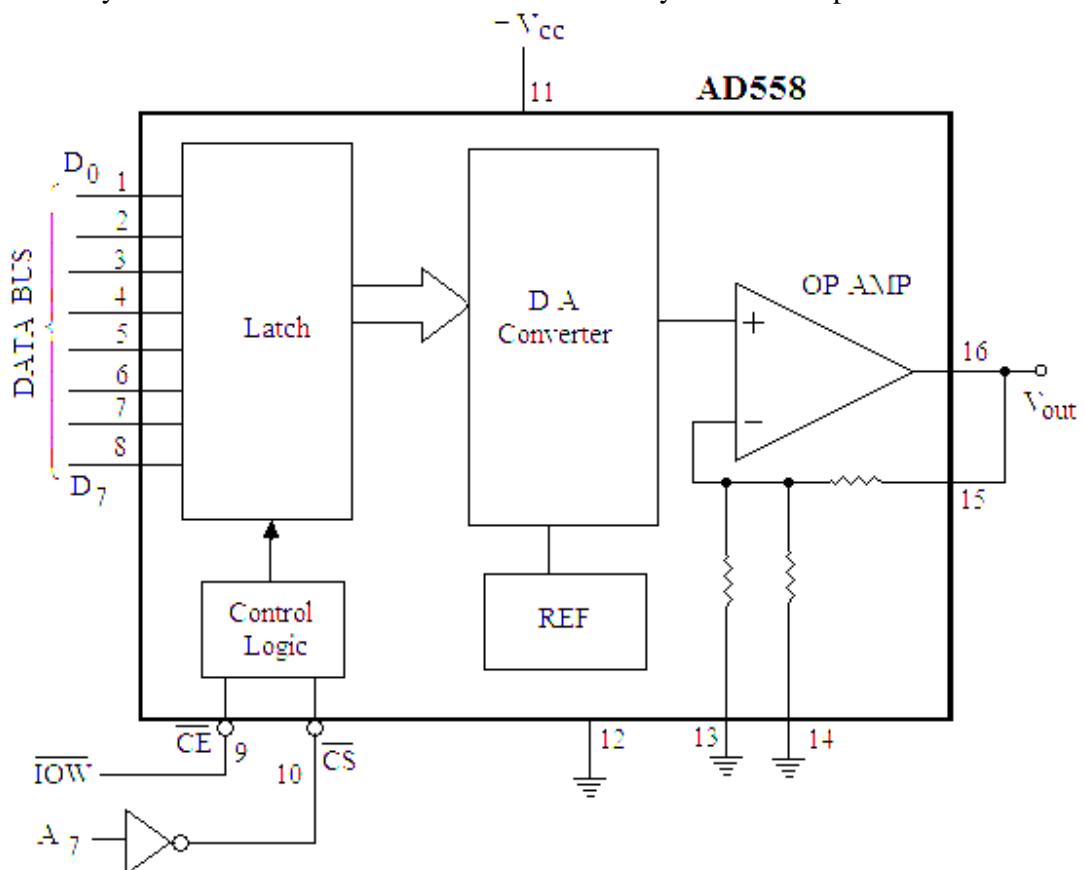
| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>   |
|--------------|------------------|----------------|---|
| LOOP         | MVI A,           | 80 H           | ; Initialize 8255-I to work all the ports as output ports.              |
|              | OUT              | 03 H           | ; Write the control word (80 H) in the control word register of 8255-I. |
|              | MVI A,           | FF H           | ; Load FF H to accumulator.   |
|              | OUT              | 00 H           | ; Send FF H (10 V) to PA <sub>0</sub> .                                 |
|              | INR A            |                | ; Increment accumulator.  |
|              | OUT              | 00 H           | ; Output corresponding to this weight is available (i.e. increasing)    |
|              | POP PSW          |                | ; Restore the PSW.  |
|              | MVI A,           | 40 H           | ; Store 40 H to the accumulator.  |
|              | OUT              | 00 H           | ; Outputs for 40 H.   |
|              | PUSH PSW         |                | ; Save PSW  |
| NEXT         | CALL             | DELAY          | ; Jump to delay subroutine to introduce a delay of 0.5msec.             |
|              | POP PSW          |                | ; Restore the PSW.  |
|              | JMP              | LOOP           | ; Repeat the process for next cycle.                                    |

|        |       |        |   |
|--------|-------|--------|---|
|        | CPI   | FF H   | voltage is available at the output of the converter). |
|        | JZ    | LOOP 1 | ; Compare if output has become 10 Volts.              |
| LOOP 1 | JMP   | LOOP   | ; If yes go to LOOP 1.                                |
|        | DCR A |        | ; If no go to LOOP.                                   |
|        | OUT   | 00 H   | ; Decrement accumulator.                              |
|        | JZ    | LOOP   | ; Outputs for decrement data.                         |
|        | JMP   | LOOP   | ; If accumulator content becomes 0, go to LOOP.       |
|        |       |        | ; Else to LOOP.                                       |

During the execution of this program the wave shape at output of the D/A converter can be checked. It will be triangular wave.

### 13.5 MICROPROCESSOR COMPATIBLE D/A CONVERTER

Now a days microprocessor compatible D/A converters are also available, which are designed for the purpose of directly interfacing with the microprocessor without the use of external latch. Among such converters, AD558 is very simple, inexpensive and commonly used D/A converter to be connected directly to the microprocessor.



**Fig. 13.14**

Figure 13.14 shows the block diagram of this D/A converter, which includes internally a latch and output op amp. It can be operated with + 4.5 V to 16.5 V d.c. power

supply. Two pins  $\overline{CS}$  (chip select) and  $\overline{CE}$  (chip enable) are provided with the chip to interface with the microprocessor 8080 or 8085. For the interfacing with 8085  $\overline{CE}$  may be connected to  $IOW$  of the processor and  $\overline{CS}$  may be connected to the  $A_7$  bit of the address line. The other pins of the address lines may be assumed as 0, so that 80 H will be the port address of this converter. When both the signals  $\overline{CS}$  and  $\overline{CE}$  are at logic 0, the latch is transparent means the input is transferred to the D/A block of this converter. When either of  $\overline{CS}$  or  $\overline{CE}$  goes to logic 1, the input is latched in the register and held until both control signal go to logic 0.

The programs given in the above examples (solved) can be executed with this converter by using the proper port address.

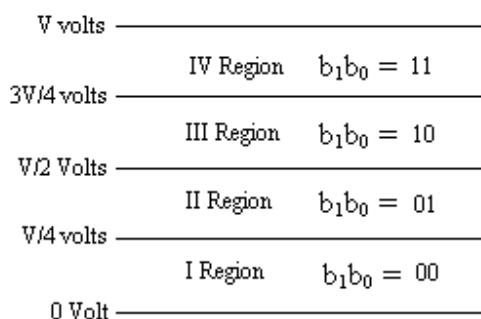
### 13.6 ANALOG TO DIGITAL CONVERTER

Generally the information to be processed by the digital systems is in the analog form. So before applying such signals to the digital systems it is necessary to convert the signal into its equivalent digital form. The method with the help of which the analog signal is converted to digital form is known as analog to digital (A/D) converter. The A/D converter is more complex and difficult than the D/A converter. Followings are the different methods for A/D converter, which will be discussed in the next sections.

- (i) Simultaneous A/D converter
- (ii) Successive approximation D/A converter
- (iii) Counter or Digital Ramp type A/ D converter
- (iv) Single slope D/A converter
- (v) Dual slope D/A converter

### 13.7 SIMULTANEOUS A/D CONVERTER

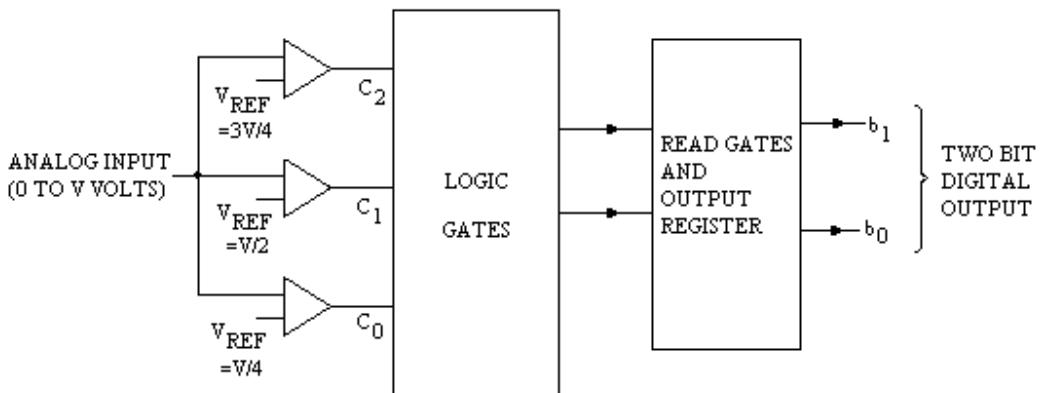
This is the fastest and simplest method of converting an analog signal to digital signal. It utilizes the parallel differential comparators; the input analog voltage is compared by these comparators with known voltages called as reference voltages. The comparators gives the low output (logic 0) when the input is less than the reference voltage and gives the high output (logic 1) when the input analog voltage exceeds the reference voltage. This method of conversion is also called as Flash or parallel type A/D converter.



**Fig. 13.15**

For the conversion of analog voltage ranging between 0 to  $V$  volts into two bit digital output, three comparators (in general  $2^n - 1$  comparators where  $n$  is the number of

bits) are required. The input analog voltage is converted to the 4 (in general  $2^n$ ) equal regions as shown in figure 13.15. If the analog voltage is lying in the first region, then the binary bits ( $b_1, b_0$ ) are 00, similarly to second, third and forth regions the binary bits are 01, 10 and 11 respectively. The reference voltages to the three comparators  $C_0, C_1, C_2$  should be  $V/4, V/2, 3V/4$  respectively as shown in figure 13.16. The output of the three comparators should be connected to the logic gates to produce the desired binary output. The read gates and output registers are used to read the digital output.



**Fig. 13.16**

Referring to figures 13.15 and 13.16, if the input analog voltage exceeds the reference voltage to any comparator, the comparator gives high output (logic 1); if on the other hand if the input analog voltage is less than the reference voltage of the comparator, it gives low output (logic 0). In this way if all the comparators give low output, the analog input voltage must be between 0 and  $V/4$  volts (I region) and digital binary output should be 00. If the  $C_0$  is high and  $C_1$  and  $C_2$  are low, the input must be between  $V/4$  and  $V/2$  volts (II region) and digital binary output should be 01. If  $C_0$  and  $C_1$  are high and  $C_2$  is low, the input must be between  $V/2$  and  $3V/4$  volts (III region) and digital binary output should be 10. Finally if all the comparators give high outputs, the input must lies  $3V/4$  and  $V$  volts (IV region) and digital binary output should be 11. Table 13.2 summarizes outputs of the comparators.

**Table 13.2**

| Input voltage   | Comparator output |       |       | Binary output |       |
|-----------------|-------------------|-------|-------|---------------|-------|
|                 | $C_2$             | $C_1$ | $C_0$ | $b_1$         | $b_0$ |
| 0 to $V/4$      | 0                 | 0     | 0     | 0             | 0     |
| $V/4$ to $V/2$  | 0                 | 0     | 1     | 0             | 1     |
| $V/2$ to $3V/4$ | 0                 | 1     | 1     | 1             | 0     |
| $3V/4$ to $V$   | 1                 | 1     | 1     | 1             | 1     |

By drawing the K-maps (figure 13.17), the expressions for  $b_0$  and  $b_1$  are obtained as:

$$b_1 = C_1 \quad \text{and} \quad b_0 = C_1 \cdot \overline{C}_0$$

| $C_0$ | $C_2 C_1$              | 00 | 01 | 11 | 10 |
|-------|------------------------|----|----|----|----|
| 0     | b <sub>1</sub> = $C_1$ | 0  | 1  | 1  | 0  |
| 1     |                        | 0  | 1  | 0  | 1  |

| $C_0$ | $C_2 C_1$                              | 00 | 01 | 11 | 10 |
|-------|--|----|----|----|----|
| 0     | b <sub>0</sub> = $C_2 + C_1 \bar{C}_0$ | 0  | 0  | 1  | 1  |
| 1     |  | 1  | 0  | 0  | 0  |

(a)

(b)

Fig. 13.17

These expressions may be realized using the gates as shown in figure 13.18. The output may be reset by applying high signal to the reset line and to read the data a high signal is applied to the read line.

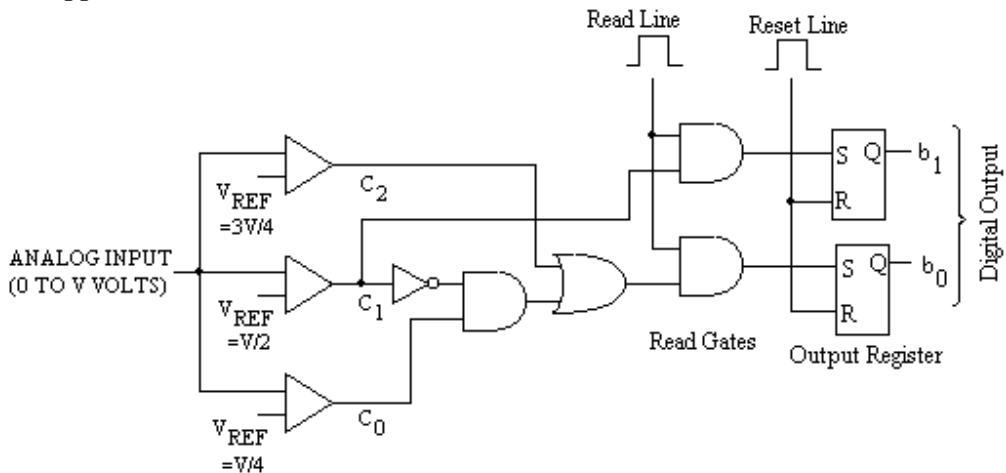


Fig. 13.18

For the conversion of analog input voltage (0 to V volts) into three bit binary output we proceed in the similar method as for the two bits output. For the three bits outputs the input voltages are divided into 8 (as  $2^3 = 8$ ) equal regions and 7 (as  $2^3 - 1 = 7$ ) comparators are to be used. So the logic circuit to be designed should have seven inputs (output of the seven comparators) and three outputs. The output of comparators and the corresponding binary output are shown in table 13.3.

Table 13.3

| Input voltage | Comparator output |                |                |                |                |                |                | Binary output  |                |                |
|---------------|-------------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
|               | C <sub>6</sub>    | C <sub>5</sub> | C <sub>4</sub> | C <sub>3</sub> | C <sub>2</sub> | C <sub>1</sub> | C <sub>0</sub> | b <sub>2</sub> | b <sub>1</sub> | b <sub>0</sub> |
| 0 to V/8      | 0                 | 0              | 0              | 0              | 0              | 0              | 0              | 0              | 0              | 0              |
| V/8 to V/4    | 0                 | 0              | 0              | 0              | 0              | 0              | 1              | 0              | 0              | 1              |
| V/4 to 3V/8   | 0                 | 0              | 0              | 0              | 0              | 1              | 1              | 0              | 1              | 0              |
| 3V/8 to V/2   | 0                 | 0              | 0              | 0              | 1              | 1              | 1              | 0              | 1              | 1              |
| V/2 to 5V/8   | 0                 | 0              | 0              | 1              | 1              | 1              | 1              | 1              | 0              | 0              |
| 5V/8 to 3V/4  | 0                 | 0              | 1              | 1              | 1              | 1              | 1              | 1              | 0              | 1              |
| 3V/4 to 7V/8  | 0                 | 1              | 1              | 1              | 1              | 1              | 1              | 1              | 1              | 0              |
| 7V/8 to V     | 1                 | 1              | 1              | 1              | 1              | 1              | 1              | 1              | 1              | 1              |

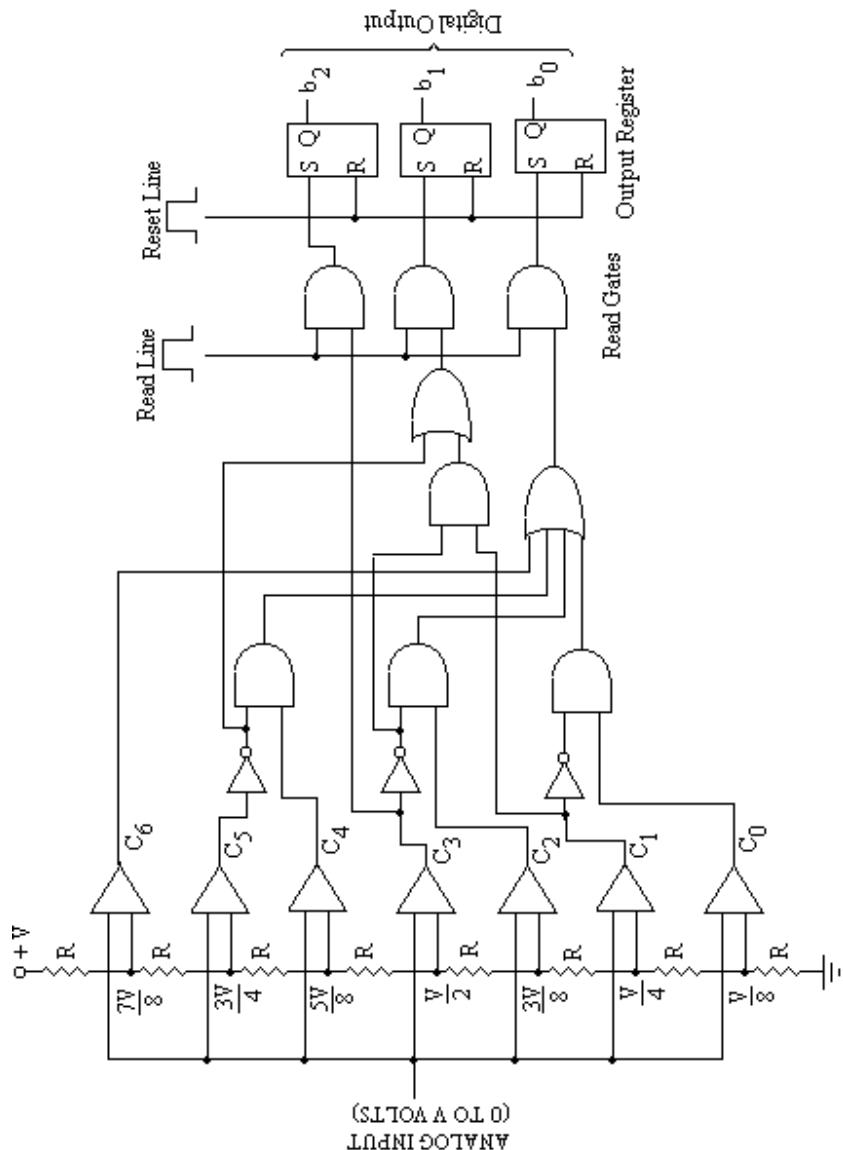
The expressions of the output bits can easily be obtained by examining the table 13.3.

The bit  $b_2$  gives the high output whenever the output of the comparator  $C_3$  is high. So  $b_2 = C_3$ .

The bit  $b_1$  is high whenever the output of comparator  $C_1$  is high and output of  $C_3$  is low, or whenever the output of comparator  $C_5$  is high. So  $b_1 = C_1 \cdot \bar{C}_3 + C_5$ .

Similarly, the expression for bit  $b_0$  can be obtained as:

$$b_0 = C_0 \cdot \bar{C}_1 + C_2 \cdot \bar{C}_3 + C_4 \cdot \bar{C}_5 + C_6$$



**Fig. 13.19**

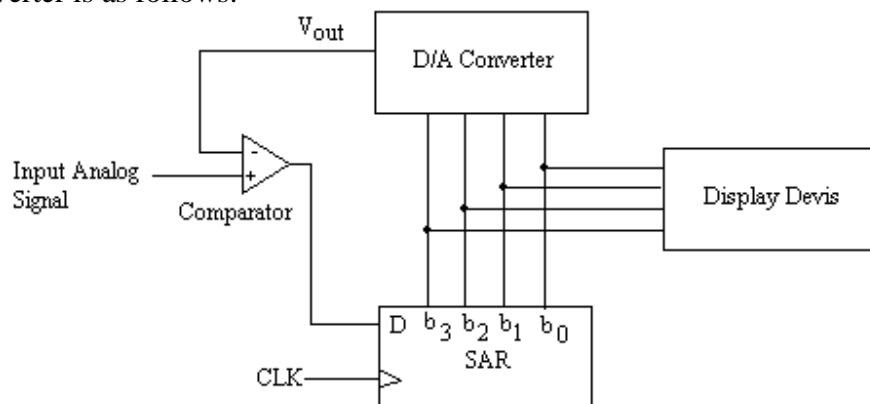
These expressions may be realized using the gates as shown in figure 13.19. The output may be reset by applying high signal to the reset line and to read the data a high signal is applied to the read line.

The design of a simultaneous A/D converter is quite straight forward and relatively easy to understand. However, the design becomes complicated as the number

of bits is increased, since the number of comparators to be used increases drastically. This method has highest speed of conversion.

### 13.8 SUCCESSIVE APPROXIMATION A/D CONVERTER

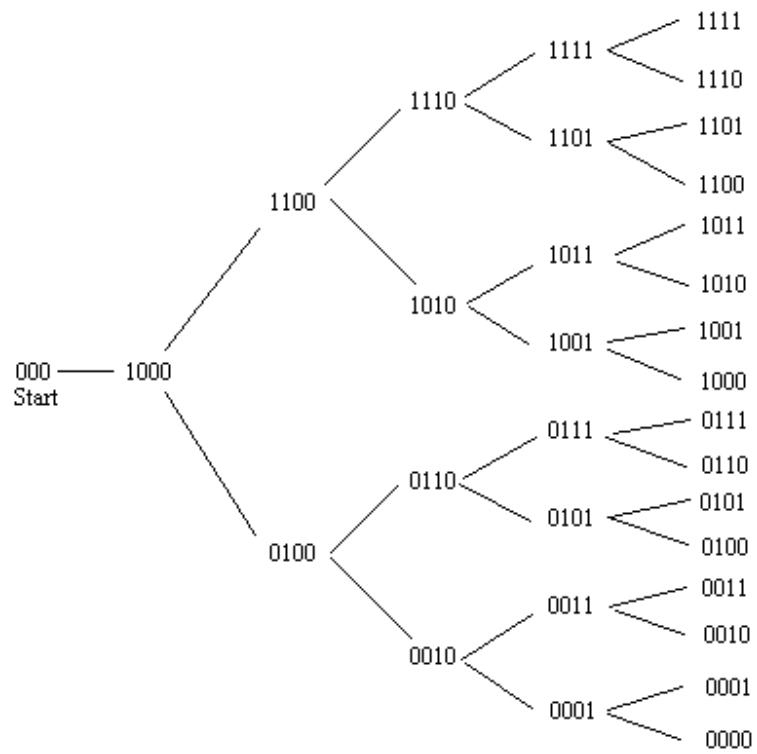
Simultaneous A/D converter has the very fast conversion time but becomes unwieldy when the required digital bits are more. The successive approximation method is most useful and commonly used method. The block diagram four bit successive approximation A/D converter is shown in figure 13.20. It consists of a D/A converter, successive approximation register (SAR) and a comparator. The basic principle of this A/D converter is as follows:



**Fig. 13.20**

In this type of converter, the bits of D/A converter are enabled one by one, starting with the most significant bit (MSB). The analog output of the D/A converter corresponding to the enabled bit is compared with the input analog voltage. The comparator gives the output low if the input analog voltage is less than the output of the D/A converter and it gives the high out if the input analog voltage is more than the output of the D/A converter. The low output of the comparator resets the corresponding bit of SAR, on the other hand if the comparator's output is high then that bit is retained in SAR. In this way the output of D/A converter are compared with the input voltage for all the bits starting with the most significant bit.

Thus the successive approximation method is the process of approximating the analog voltage bit by bit starting with MSB. This process is shown in figure 13.21.



**Fig. 13.21**

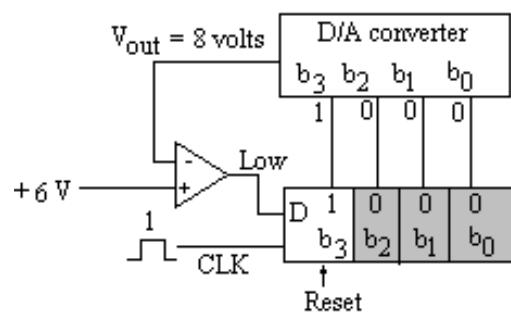
In order to understand the operation of this type of A/D converter, we will take a specific example of a four-bit conversion. Figure 13.22 (a through d) shows the step-by-step conversion of a given analog input voltage (say 6 volts). It is further assumed that D/A converter has the following output characteristics:

$$V_{out} = 8 \text{ volts for bit 3 (MSB or } b_3)$$

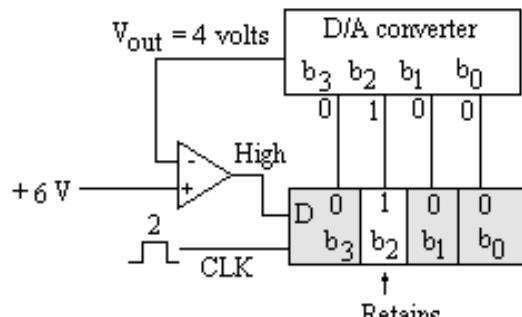
$$V_{out} = 4 \text{ volts for bit 2 (2^{nd} MSB or } b_2)$$

$$V_{out} = 2 \text{ volts for bit 1 (3^{rd} MSB or } b_1)$$

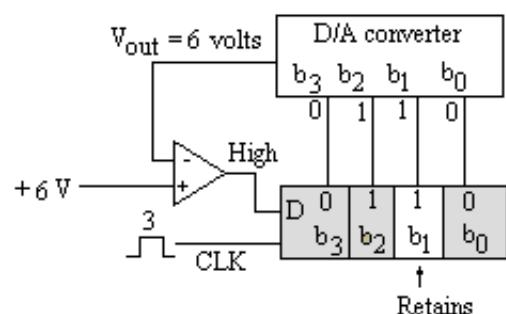
$$V_{out} = 1 \text{ volt for bit 0 (LSB or } b_0)$$



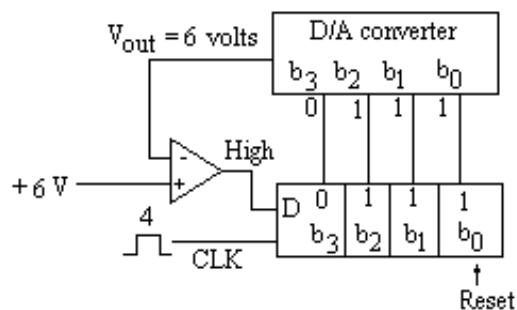
(a)



(b)



(c)



(d)

**Fig. 13.22**

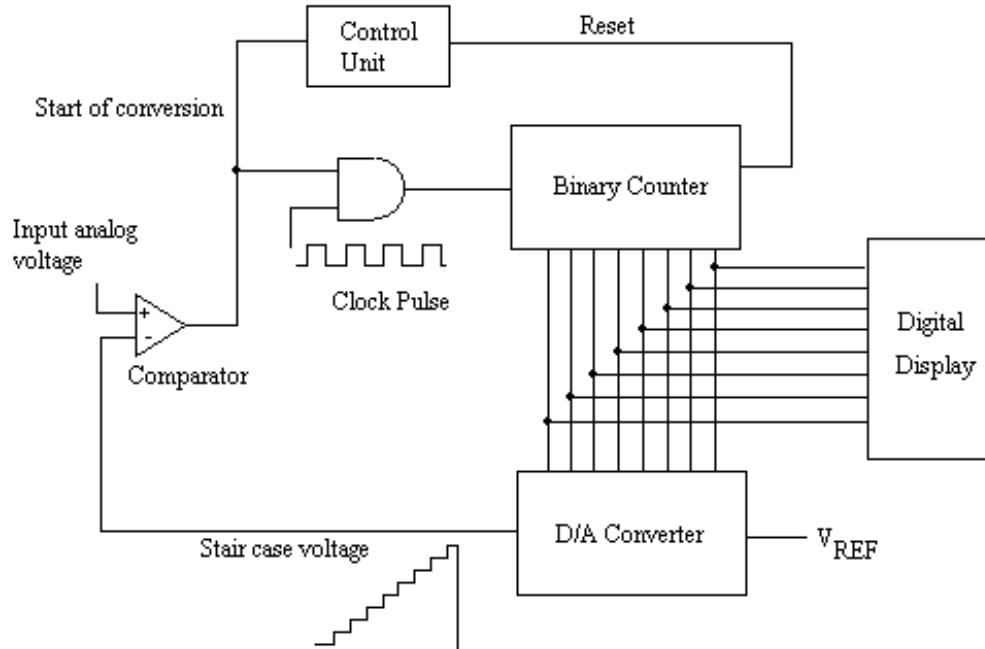
It is clear from these figures that after completing the conversion cycle. The binary code 0110 is retained in SAR, which is binary value of the input voltage (6Volts). It is finally displayed on the display devices.

### 13.9 COUNTER OR DIGITAL RAMP TYPE A/D CONVERTER

Another method of converting the analog signal to digital one is the counter or digital ramp type A/D converter which utilizes a binary counter to count a continuous pulse of standard width and height from a clock. The standard clock pulses are passed through a gate which is open for some time to allow these pulses to go to the input of counter. Normally the gate is closed and as soon as the start signal is applied a stair case voltage is initiated. This voltage is increased linearly with the increase of the binary counts in the counter. The gate remains open for the time the linear stair case voltage

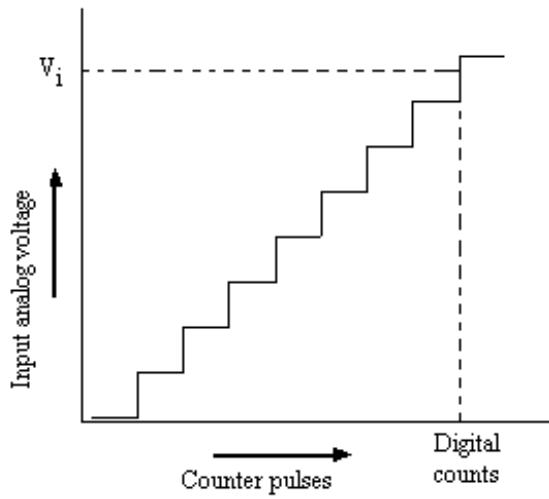
becomes equal to the input analog voltage. The counter records the number of clock pulses which is proportional to the input analog voltage.

Figure 13.23 shows the schematic diagram of this type of A/D converter. The analog signal, to be converted to its equivalent digital output, is applied to one input of an operational amplifier being used as a comparator. When a start of conversion pulse is applied to the control unit it resets the binary counter and opens the gate. The counter



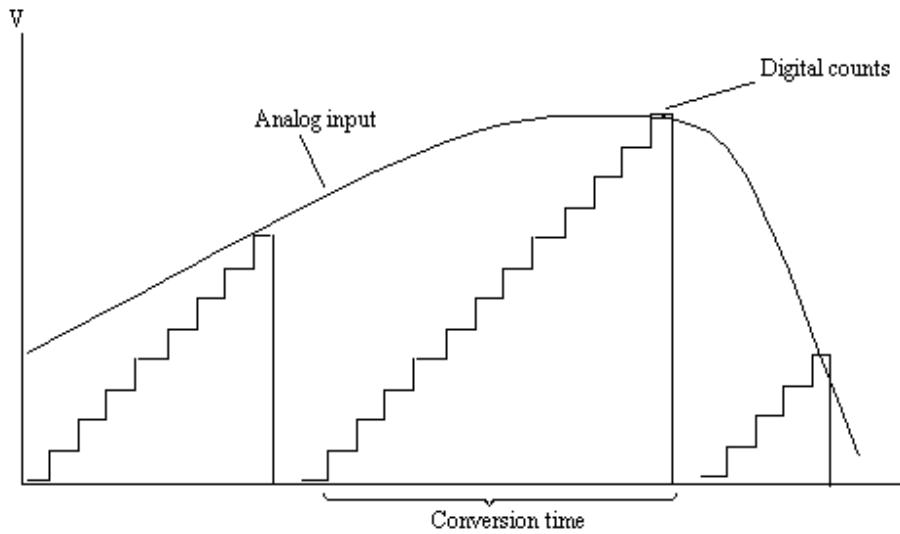
**Fig. 13.23**

starts counting the clock pulses which are of standard width and height. The output of the counter is fed to a D/A converter which produces an analog output (stair case voltage) in response to the digital signal (output of the counter) as its input. This analog output voltage is fed to the reference input of the comparator. So long as the input analog signal is greater than the stair case voltage the comparator provides the high output to the gate, the gate remains open and the clock pulses are allowed to reach to the input of the counter. These pulses are counted by the counter thus continuously increasing the digital output. The moment the analog output of D/A converter (stair case voltage) exceeds the input analog voltage, the comparator provides a low output disabling the gate and the counter stops counting. The binary number stored in the counter represents the digital output voltage corresponding to the input analog voltage. The digital output is displayed on the display devices.



**Fig. 13.24**

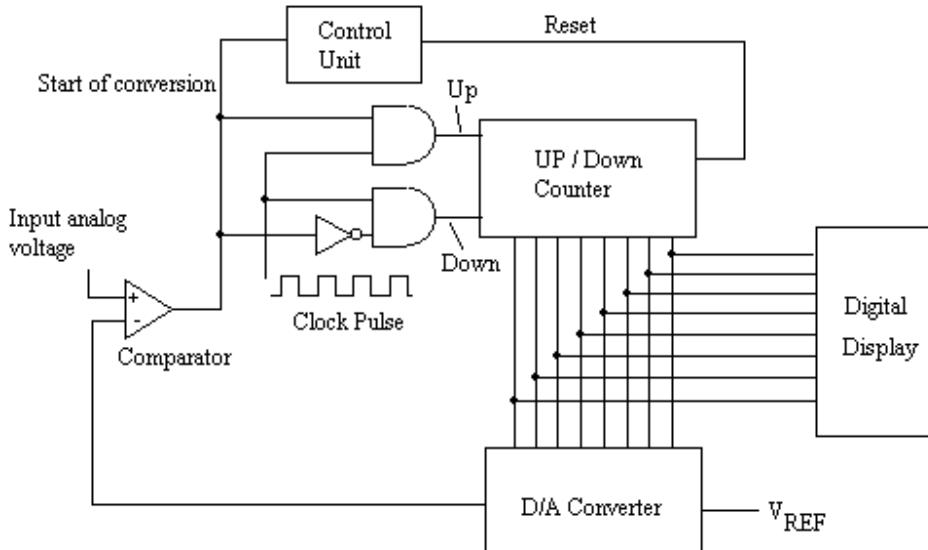
For a steady input the digital output is as shown in figure 13.24. The output is represented by the number of clock pulses counted by the counter till the stair case voltage becomes equal to the input voltage. This method of conversion is slow; as for maximum input, the counter has to count from zero to maximum number of states for the comparison. For each conversion cycle the counter is to be reset and counting starts from beginning. The time of conversion is not important in d.c. or slow varying signals as the output waveform gives a good representation from which the input waveform can be constructed as shown in figure 13.25. But if the conversion time and the signal transient time are comparable the reconstructed digital output will not be correct. In this case it is necessary to reduce the conversion time by using faster D/A converter.



**Fig. 13.25**

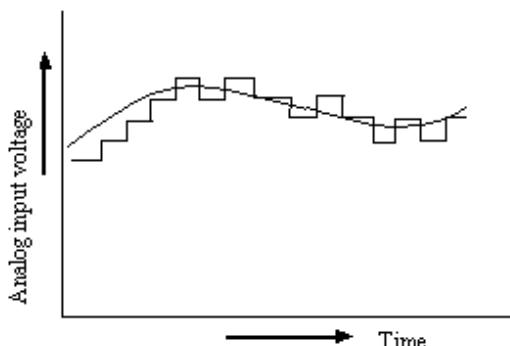
A modification to this converter is possible if the resetting of the counter is avoided each time. For this purpose an up/down counter may be used in place of up

counter. The circuit shown in figure 13.26 illustrates this modification in which an



**Fig. 13.26**

up/down binary counter is used and the converter proceeds without resetting. The circuit is almost the same as the counter or digital ramp type A/D converter. The up/Down counter is operated by up or down signals from the control unit. The digital to analog converter output controls the output of the comparator. Till the D/A converter output is less than the analog input voltage, the up signal is enabled and the counter counts in forward direction. When the analog input falls, the down signal is enabled and the counter starts reverse counting giving an output corresponding to new analog input as shown in figure 13.27.



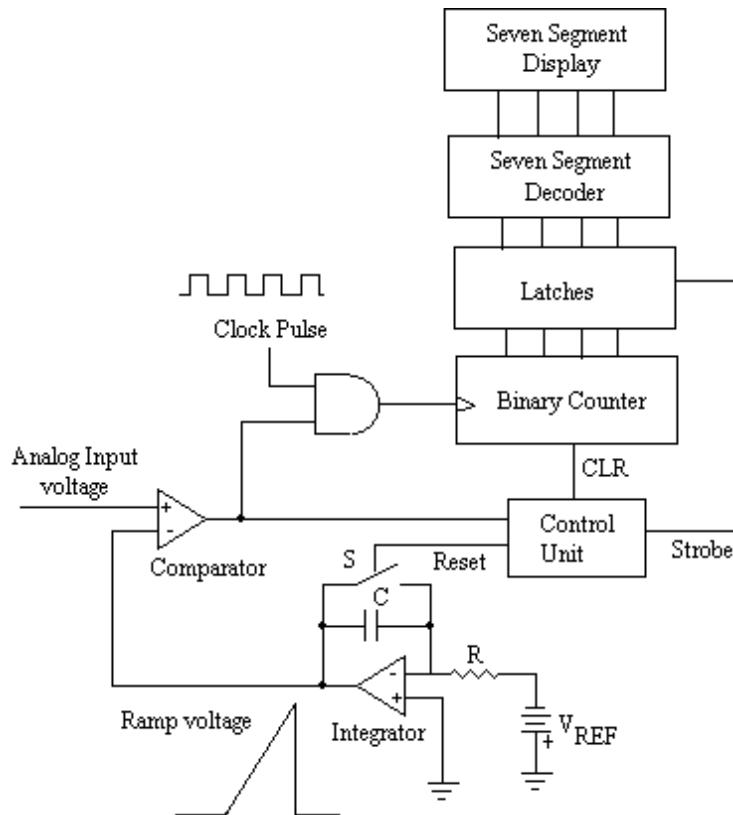
**Fig. 13.27**

### 13.10 SINGLE SLOPE A/D CONVERTER

This type of method is similar to counter or digital ramp type A/D converter. In this type of A/D converter also, a gate whose period is proportional to the amplitude of the analog sample is generated. For the generation of gate, the input analog voltage is compared with the output of an integrator. The output of integrator is a ramp voltage of constant slope. The standard clock pulses are passed through the gate and are counted by the counter. The gate remains open for the time proportional to the input analog signal. The recorded number of pulses is, therefore, the required digital output of the analog signal.

The schematic block diagram of such an analog to digital converter is shown in figure 13.28. Initially a reset pulse is applied which clears the counter and resets the integrator. The integrator produces a linearly rising ramp voltage, whose slope will depend on the values of the resistance R and capacitor C. The input analog voltage is compared by a comparator with the ramp voltage. As long as the integrator output is smaller than the input analog voltage, the comparator output is high. This high output enables the AND gate. The standard clock pulses are, therefore, allowed to pass through the gate which will be counted by the counter. When the ramp voltage becomes greater than the input analog voltage, the comparator changes the state thereby disabling the AND gate. The counter stops counting. It can easily be seen that the gate duration is linearly related to the magnitude of the input analog signal. Hence the count accumulated in the counter is a digital representation of the input analog voltage.

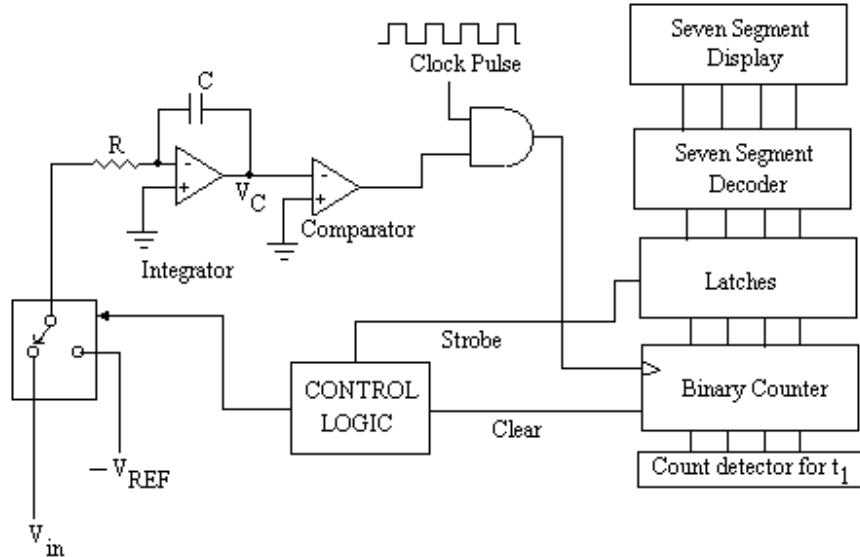
It may be mentioned here that the precision in the proportionality between the gate duration and the magnitude of the input analog signal depends upon the linearity of the ramp voltage obtained at the output of the operational amplifier. So the overall accuracy will depend upon the stability of reference source, the off-set of the operational amplifier, the frequency stability of the clock as well as the values of resistance R and capacitance C.



**Fig. 13.28**

### 13.11 DUAL SLOPE A/D CONVERTER

In single slope A/D converter, the accuracy of the converter depends on the linearity of the ramp voltage generated by the integrator. The linearity of ramp voltage, however, depends on the accuracy of the values of Resistance R and capacitance C of the integrator, whose values may vary with time and temperature. The dual slope analog to digital converter utilizes two different ramps, one for fixed time and other for fixed slope. It is very popular and widely used D/A converter because it has the slowest conversion time and relatively low cost. This method offers good accuracy, good linearity, and very good noise rejection characteristics.



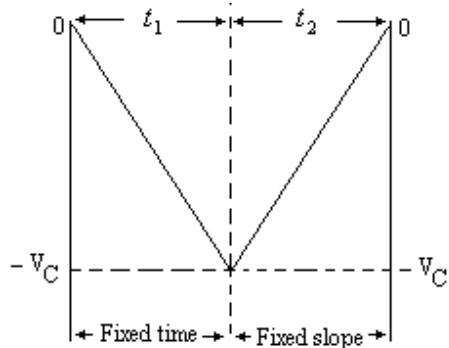
**Fig. 13.29**

The logic diagram of the dual slope A/D converter is given in figure 13.29. This converter is similar to that of the single slope A/D converter. In this converter, the integrator forms two different ramps, one for fixed time and other for fixed slope. The capacitor of the integrator is first charged with constant current obtained from input analog voltage for fixed time then the capacitor is discharged for fixed slope through other constant current obtained from a reference voltage source. The basic operation of this converter can be understood as follows:

This converter consists of standard clock pulses applied to the gate. The gate allows the pulses to the input of the counter which counts these pulses. Initially all the counters are reset to 0's and ramp too is reset to zero. Now the control logic allows switch S to connect the input analog voltage  $V_{in}$  to the integrator circuit. A constant current equal to  $\frac{V_{in}}{R}$  flows through the capacitor C as the inverting input of the operational amplifier of the integrator is at virtual ground. The capacitor C will charge linearly with this constant current. This results a negative going ramp at the output of the integrator. The comparator's output will be positive which allows the clock pulse to pass through the AND gate to the input of the counter. This ramp is allowed for fixed time say  $t_1$ . The actual time  $t_1$  is determined by the count detector. The voltage  $V_C$  at the output of the integrator is given by:

$$\begin{aligned}
 V_C &= -\frac{1}{R.C} \int_0^{t_1} V_{in} dt \\
 &= -\frac{1}{R.C} \cdot V_{in} \cdot t_1
 \end{aligned} \quad \dots (13.6)$$

The counter when reaches the fixed count at  $t_1$ , the control logic generate a pulse to clear the counter to zero and the switch S connects the integrator input to a negative reference voltage ( $-V_{REF}$ ). The capacitor C of the integrator starts discharging linearly due to the constant current from  $-V_{REF}$ . The integrator thus produces a positive going ramp beginning at  $-V_C$  and increases steadily till it reaches to 0 volt as shown in figure 11.29. At this time the counter is counting. The conversion cycle ends when  $V_C = 0$  volt; the comparator produces the low state, which disables the gate and counter stops counting.



**Fig. 13.30**

Let  $t_2$  is the time when the output of integrator becomes zero, so the output of the integrator is given by:

$$V_C = \frac{V_{REF}}{R.C} \cdot t_2 \quad \dots (13.7)$$

Since the integrator's output beginning at 0 volt and integrates down to  $-V_C$  and then integrate back to 0 volt, so the equations (13.6) and (13.7) may be equated as:

$$\begin{aligned}
 \frac{V_{in}}{R.C} \cdot t_1 &= \frac{V_{REF}}{R.C} \cdot t_2 \\
 \text{or} \quad V_{in} &= V_{REF} \cdot \frac{t_2}{t_1}
 \end{aligned} \quad \dots (13.8)$$

In this equation  $V_{REF}$  and time  $t_1$  are constants, so

$$V_{in} \propto t_2 \quad \dots (11.9)$$

This equation is independent of the values of resistance R and capacitance C. Further at the end of conversion cycle, the counts measured by the counter are proportional to the input analog signal are latched to display on the display devices.

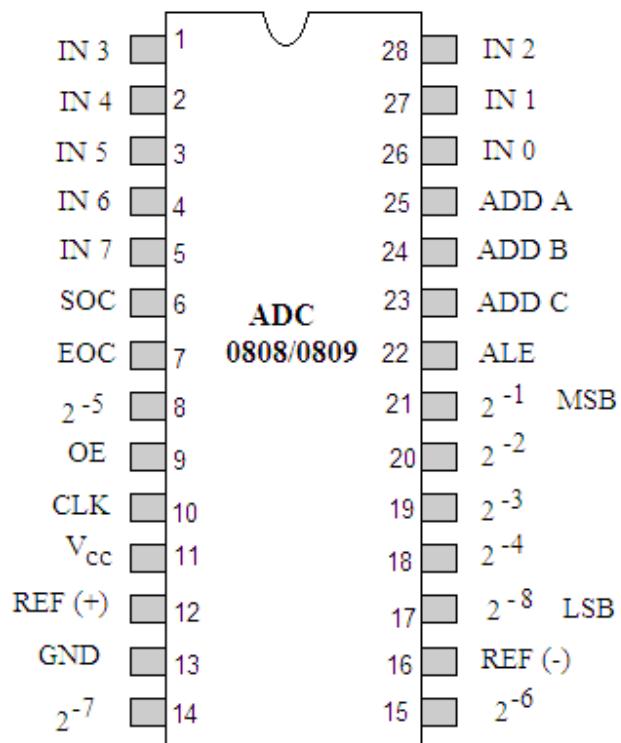
### 13.12 ADC 0808/0809

The ADC 0808/0809 is an 8-bit A/D converter with 8-channel multiplexer which can be interfaced with the microprocessor. It is the most popular, inexpensive and widely

used A/D converter. It is a monolithic CMOS device which uses the method of successive approximation A/D converter. It does not require external zero and full scale adjustment. The device operates with + 5 V d.c. supply. The conversion time is 100  $\mu$ sec at clock frequency of 640 KHz. The 8-channel multiplexer can directly access any of the 8 single ended analog signals.

The ADC 0808 / 0809 offers following features:

- High speed
- High accuracy
- Minimal temperature dependence
- Excellent long term accuracy and repeatability
- Consume minimal power

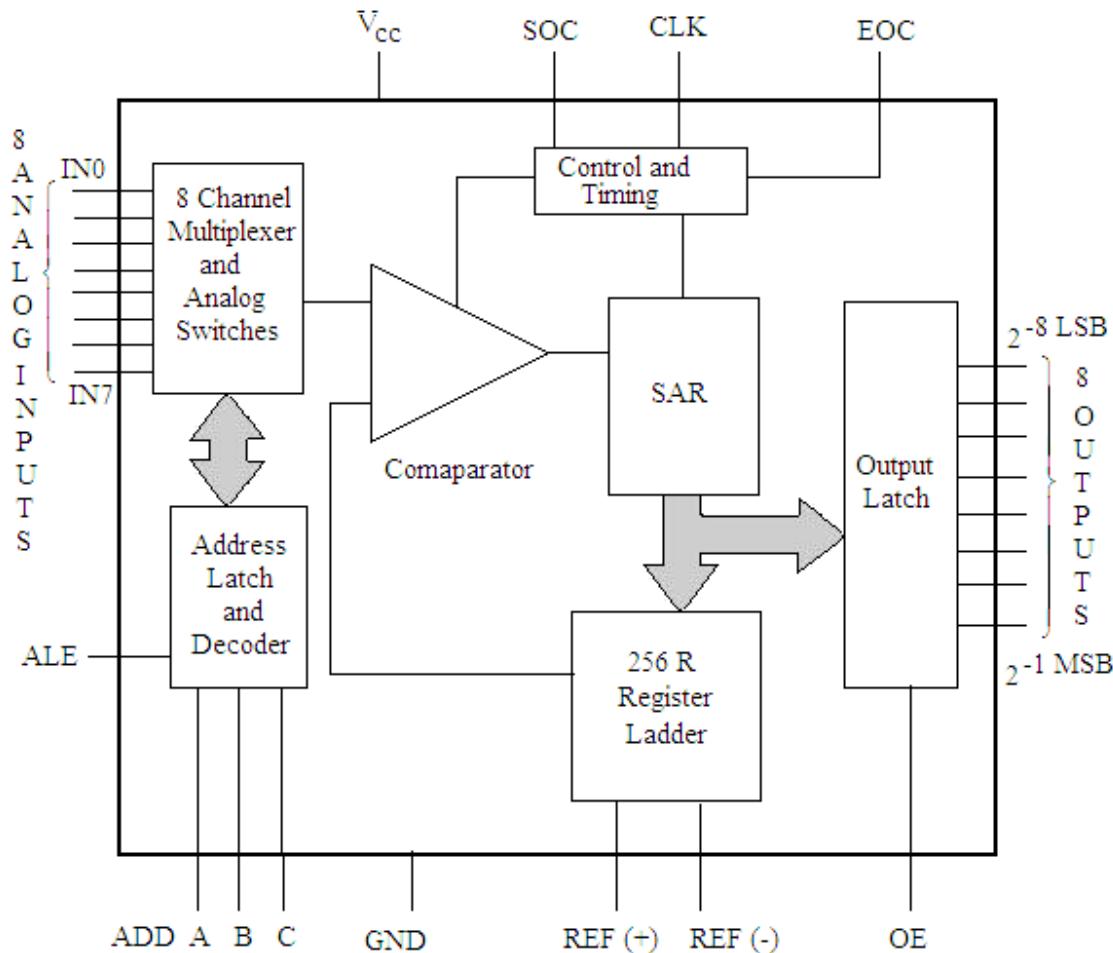


**Fig. 13.31**

The pin diagram and schematic block diagrams of this IC ADC 0808 / 0809 are shown in figures 13.31 and 13.32 respectively. The pin description of this IC is given as:

- Pins 26-28 and 1-5 – are 8 input channels IN0 – IN7 (multiplexed), which are selected by the three (address) lines ADD A, ADD B and ADD C.
- Pin 6 SOC – Start of conversion pin. A high to this pin starts the conversion.
- Pin 7 EOC – End of conversion. This an output terminal and gives high signal when the conversion ends.
- Pins 21-18, 8, 15-14 & 17 – 8-bit output (latch) pins to be connected to the port of 8255.

- Pin 9 OE – Output enable pin which is connected to the positive supply.
- Pin 10 CLK – Clock terminal. It is to be connected either to the external clock of frequency between 10 to 1280 KHz or the clock out frequency of the 8085A after dividing it by a factor of four may be connected to this pin.
- Pin 11 V<sub>CC</sub> – Supply pin (+5 V).
- Pin 12 Ref (+) – Reference pin which is to be connected to the + supply.



**Fig. 13.32**

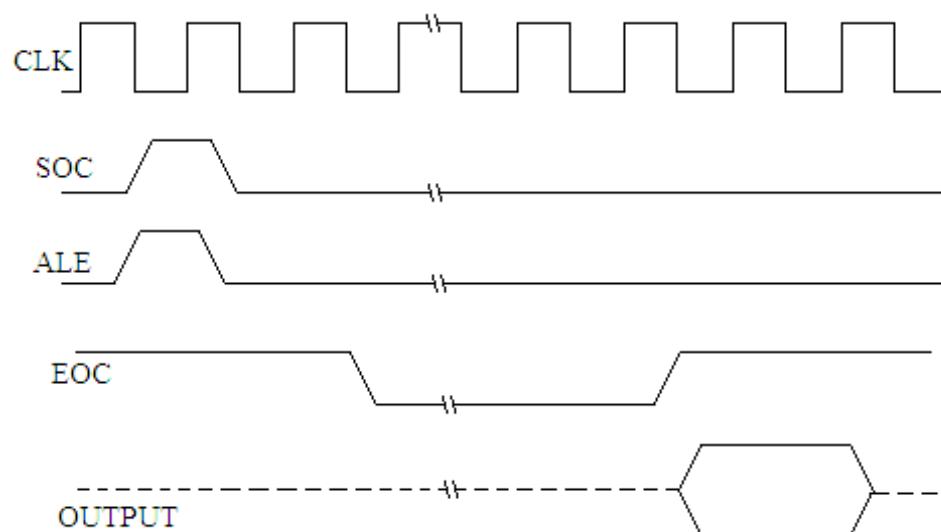
- Pin 13 GND – Ground terminal.
- Pin 16 Ref (-) – Reference pin to be connected to the ground.
- Pins 25-23 – Address pins (ADD A, ADD B and ADD C). These pins are used to select the channels as given below (Table 13.4):

Table 13.4

| Channels Selected | Address lines |   |   |
|-------------------|---------------|---|---|
|                   | C             | B | A |
| IN0               | 0             | 0 | 0 |

|     |   |   |   |
|-----|---|---|---|
| IN1 | 0 | 0 | 1 |
| IN2 | 0 | 1 | 0 |
| IN3 | 0 | 1 | 1 |
| IN4 | 1 | 0 | 0 |
| IN5 | 1 | 0 | 1 |
| IN6 | 1 | 1 | 0 |
| IN7 | 1 | 1 | 1 |

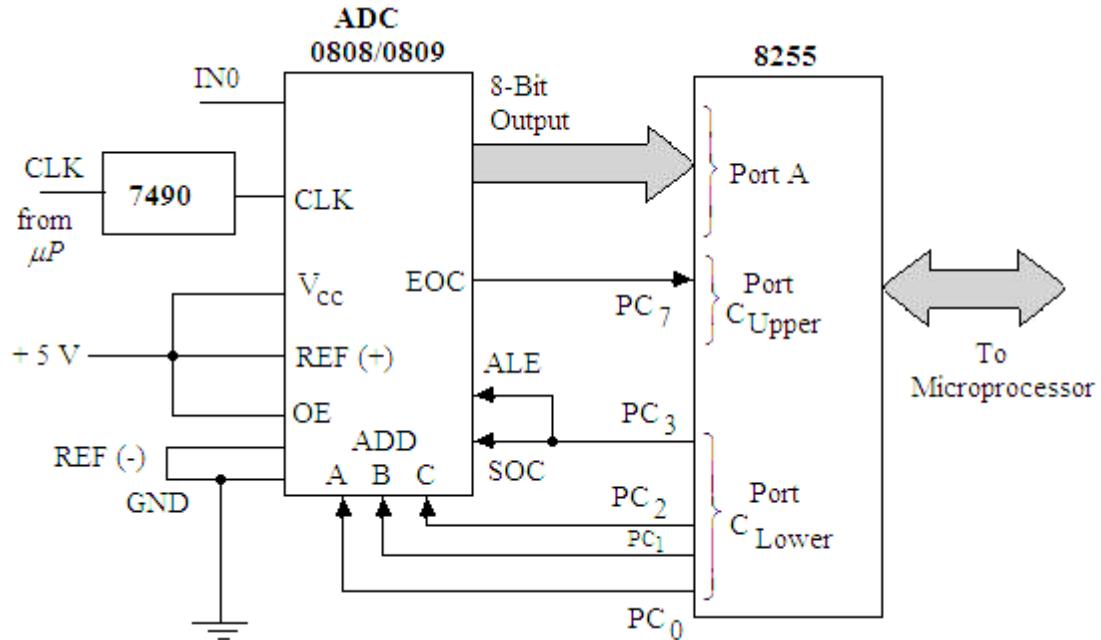
The timing diagram of this IC is shown in figure 13.33.



**Fig. 13.33**

The interfacing connection of ADC 0808 / 0809 with the microprocessor 8085A is shown in figure 13.34. In this circuit an analog voltage, whose equivalent digital value

is to be obtained, is applied to IN0. The output terminals of the ADC are connected to



**Fig. 13.34**

Port A of 8255 PPI. This port is to be used as the input port. The PC<sub>7</sub> terminal of the Port C<sub>Upper</sub> is connected to the End of Conversion terminal of the ADC. This port is also used as the input port. The port C<sub>Lower</sub> is used as the output port. The three terminals PC<sub>0</sub> to PC<sub>2</sub> are connected to the three address pins of the ADC 0808 / 0809 for the selection of input channel. The PC<sub>3</sub> terminal is connected to the SOC (Start of Conversion) and the ALE terminals of the converter. The OE and REF (+) terminals are connected to the + 5V supply. The assembly language program is given below:

In the above circuit diagram, the channel used is IN0, so for its selection we have:

| C               | B               | A               |        |
|-----------------|-----------------|-----------------|--------|
| PC <sub>2</sub> | PC <sub>1</sub> | PC <sub>0</sub> |        |
| 0               | 0               | 0               | = 00 H |

To enable the start of conversion first PC<sub>3</sub> is made high and then low without changing the selection of channel, we proceed as follows:

| PC <sub>3</sub> | PC <sub>2</sub> | PC <sub>1</sub> | PC <sub>0</sub> |        |
|-----------------|-----------------|-----------------|-----------------|--------|
| 1               | 0               | 0               | 0               | = 08 H |
| 0               | 0               | 0               | 0               | = 00 H |

### PROGRAM

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
|       | MVI A,    | 98 H    | ; Initialize 8255-I to work Port A and Port C <sub>Upper</sub> as input ports, and port C <sub>Lower</sub> as output port. |

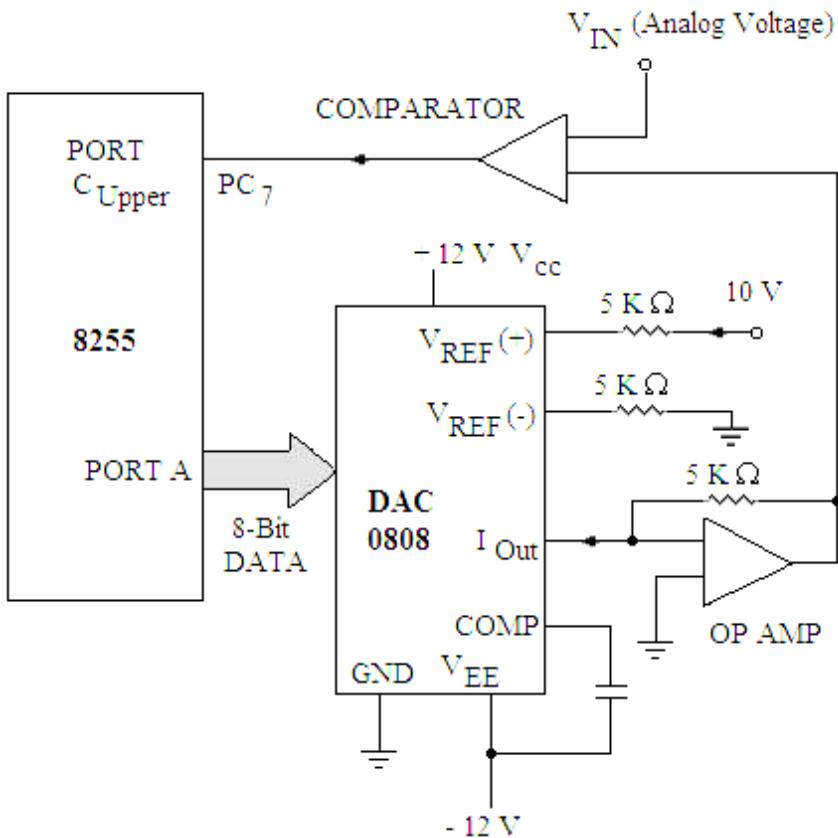
|      |        |        |   |
|------|--------|--------|---|
|      | OUT    | 03 H   | ; Write the control word in the control word register of 8255-I.  |
|      | MVI A, | 00 H   | ; Load accumulator with 00H.  |
|      | OUT    | 02 H   | ; 00H is sent to Port C <sub>Lower</sub> to select IN0 channel.   |
|      | MVI A, | 08 H   | ; Load 08 H to accumulator.   |
|      | OUT    | 02 H   | ; ALE and SOC are enabled (high).   |
|      | MVI A, | 00 H   | ; Load 00 H to accumulator.   |
|      | OUT    | 02 H   | ; ALE and SOC will be low. A pulse is applied from high to low for the conversion through PC <sub>3</sub> without affecting the channel selected. |
| READ | IN     | 02 H   | ; Read End of conversion (PC <sub>7</sub> ).  |
|      | RAL    |        | ; Rotate left to check if PC <sub>7</sub> is one.   |
|      | JNC    | READ   | ; If not 1 READ again.  |
|      | IN     | 00 H   | ; Read digital output of the A/D converter.   |
|      | STA    | 2100 H | ; Store the result in 2100 H memory location.   |
|      | HLT    |        | ; Stop processing.  |

After the execution of this program, the values stored in the memory location 2100 may be checked.

| Analog Voltage | Digital output |
|----------------|----------------|
| 0 V            | 00 H           |
| 1 V            | 34 H           |
| 1.4 V          | 47 H           |
| 2 V            | 6D H           |
| 2.8 H          | 90 H           |
| 3 V            | 94 H           |
| 4 V            | CE H           |
| 5 V            | FF H           |

### 13.13 A/D CONVERTER USING D/A CONVERTER AND SOFTWARE

The A/D converter can also be realized using the D/A converter and the software. For this purpose the software is used to implement successive approximation A/D converter. Figure 13.35 shows the interfacing connection for this. The input analog voltage is applied to a comparator. This voltage is compared by the output of the D/A converter. The output of the comparator is sensed by the microprocessor through the PC<sub>7</sub> of the 8255. A digital input is applied to the D/A converter, through port A of 8255.



**Fig. 13.35**

The microprocessor first applies 00 H as the digital input to the DAC 0808 and checks through PC<sub>7</sub> for high signal. If a high pulse does not appear at the output terminal of the comparator, the microprocessor increments the digital input by 1 and the process is repeated till a high signal appears at the output of the comparator. As a high signal appears the microprocessor stops successive approximation and the corresponding digital input value is stored in the memory location. This digital value is equivalent to the analog signal applied at V<sub>IN</sub> terminal of the comparator. The software for the same is given as:

#### PROGRAM

| Label | Mnemonics | operand | Comments  |
|-------|-----------|---------|---|
|       | LXI SP,   | XXXX H  | ; Initialize stack pointer.   |
|       | MVI A,    | 88 H    | ; Initialize 8255-I to work Port A, Port B and port C <sub>Lower</sub> as output ports, and Port C <sub>Upper</sub> as input ports. |
|       | OUT       | 03 H    | ; Write the control word in the control word register of 8255-I.  |
|       | MVI A,    | 00 H    | ; Load accumulator with digital input to send it to D/A converter.  |
| LOOP  | OUT       | 00 H    | ; The digital input is sent to D/A converter through Port A.  |
|       | PUSH PSW  |         | ; Save PSW  |

|     |         |        |  |
|-----|---------|--------|--|
|     | CALL    | DELAY  | ; Call delay subroutine program.   |
|     | IN      | 02 H   | ; Check the output of the comparator through PC <sub>7</sub> , Read PC <sub>7</sub> bit. |
|     | RAL     |        | ; Rotate left to check PC <sub>7</sub> , if it is high.                                  |
|     | JC      | NXT    | ; If is high go to NXT.  |
|     | POP PSW |        | ; Else get the digital value from the stack.   |
|     | INR A   |        | ; Increment digital value.   |
|     | JMP     | LOOP   | ; Jump to LOOP.  |
| NXT | POP PSW |        | ; Get the digital value from the stack.  |
|     | STA     | 2100 H | ; Store the result in 2100 H memory location.  |
|     | HLT     |        | ; Stop processing.   |

#### SUBROUTINE PROGRAM:

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
| DELAY | LXI D,    | 0002 H  | ; Loads DE register pair with a 16-bit number.                                   |
| LOOP1 | DCX D     |         | ; Decrements DE register pair by one.  |
|       | MOV A,    | E       | ; Moves the contents of E register to accumulator.                               |
|       | ORA D     |         | ; ORing of the contents of D and E registers are performed to set the zero flag. |
| JNZ   | LOOP1     |         | ; If result ≠ 0 jump to loop1  |
| RET   |           |         | ; Go back to main program.   |

After the execution of this program, the values stored in the memory location may be checked for the input applied voltage.

| Analog Input | Digital output |
|--------------|----------------|
| 0 V          | 00 H           |
| 2.5 V        | 40 H           |
| 5 V          | 80 H           |
| 10 V         | FF H           |

#### PROBLEMS

1. Discuss the resistive divider D/A converter. Find the general expression for the output voltage of a resistive divider network.
2. Using the resistive divider network draw the circuit of a 6 bit D/A converter and explain its operation. What are the drawbacks of this D/A converter?
3. Show that the outputs of a binary weighted resistor network are directly proportional to the binary inputs.
4. Draw the schematic diagram of a resistive divider D/A converter. Explain its operation. Mention the drawbacks of this converter.

5. A 5 bit resistive divider network has 0 volts full scale output, find the output voltage for a binary input 10101. (Ans. 6.774 V)
6. A 6 bit resistive divider D/A converter has resistance of  $100\text{ K}\Omega$  in MSB branch. The reference voltage is 15 V. The resistance in the feed back path of the operational amplifier is  $39\text{ K}\Omega$ . What is the output voltage for the binary input 101101? (Ans. – 8.22V)
7. For a 6 bit resistive divider network, the reference voltage is 10 V, find the following:  
 (i) Full Scale output voltage.  
 (ii) The analog output voltage for a digital input of 010011.  
 (iii) The output voltage change due to least significant bit. (Ans.:10V, 3.02 V, 0.16 V)
8. Draw the schematic diagram of a binary ladder D/A converter. Explain its operation. Mention its merits and demerits.
9. Find the expressions for the output due to MSB and second MSB of a 4-bit binary ladder network.
10. Discuss the binary ladder D/A converter. Find the general expression for the output voltage of a binary ladder network.
11. What are the performance criteria for the D/A converter? Discuss their importance while selecting a D/A converter.
12. For a 6-bit binary ladder D/A converter the input levels are  $0 = 0\text{ V}$  and  $1 = 10\text{ V}$ , find  
 (i) The output voltages caused by each bit.  
 (ii) The output voltage corresponding to an input of 101101.  
 (iii) The full scale output voltage of the ladder. (Ans. (i) -5V, -2.5 V, -1.25 V, -0.625 V, -0.3125 V, -0.15625 V (ii) – 7.03125 V (iii) – 9.84 V)
13. For a 5-bit binary ladder D/A converter the input levels are  $0 = 0\text{ V}$  and  $1 = 10\text{ V}$ , find the output voltage corresponding to binary input of (i) 10111 (ii) 01101. (Ans. – 7.1875 V, – 4.0625 V)
14. What is the step size (or resolution in volts) of a 10 bit D/A converter, if the full scale output is +10 volts? Find the percentage resolution also. (Ans. 9.78 mV, 0.0978%)
15. How many bits are required at the input of a D/A converter to achieve a resolution of 15mV, if the full scale output is 15 volts? (Ans. 10 bits)
16. Give the details of D/A converter IC 0808. Using this IC draw a circuit diagram to get the analog output voltage corresponding to 8 bit digital input.
17. Describe how the interfacing of the D/A converter is done with the microprocessor 8085A. Give its circuit diagram.
18. Discuss the simultaneous A/D converter to convert 0 to V volts analog voltage to 3 bit digital output. Draw the logic diagram also. What are the disadvantages of this type of A/D converter?
19. Give the details of the microprocessor compatible D/A converter IC AD558.
20. Draw a logic diagram to convert 0 to V volt analog voltage to its equivalent 2 bit digital output using simultaneous A/D converter.

21. Describe the successive approximation method for A/D conversion.
  22. Draw a schematic diagram of counter or digital ramp type A/D converter. Explain its operation.
  23. Describe the modified counter or digital ramp type A/D converter with neat diagram.
  24. Draw a schematic diagram of a D/A converter. Explain its operation.
  25. Describe single slope A/D converter with its logic diagram. Mention its merits and demerits.
  26. Describe Dual slope A/D converter with its logic diagram.
  27. Give the details of A/D converter IC 0801.
  28. The input bits of the D/A converter are connected to the output pins of Port A of 8255, which is already connected with the microprocessor (ref. fig. 13.13). Write a program to generate staircase voltage with ten steps. It should have the constant pulse duration.
  29. Write a program to generate square wave of 2 KHz frequency with a peak voltage of 2.5 volts. Use D/A converter (ref. fig. 13.13) for this purpose. The square wave should be available at the output of the converter.
  30. Give the details of A/D converter IC ADC0808/0809. Using this IC draw a circuit diagram to get the digital value of the input analog voltage.
  31. Describe how the interfacing of the A/D converter is done with the microprocessor 8085A. Give its circuit diagram.
  32. Write a program to get the digital output of an analog signal applied to ADC 0808/0809 interfaced with the 8085A. The input analog signal is applied to the channel 3 of this IC.
  33. Explain how the A/D converter can be designed using D/A converter and the software. For this purpose the software is used to implement successive approximation A/D converter.
-

# 14

## Serial Communication and Programmable Communication Interface

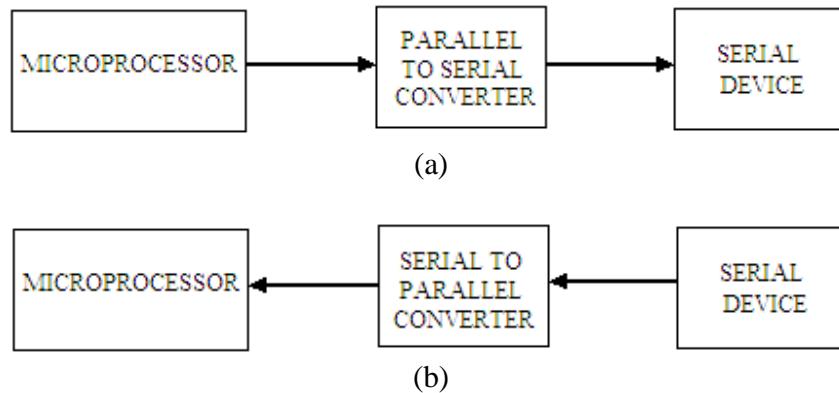
---

In continuation to the series of interfacing devices, the discussion will now be made on serial data communication and programmable communication interface, since the serial data transmission is always preferred for long distance transmission. Though the serial data transmission can be made through the SID (serial input data) and SOD (serial out data) lines of 8085A microprocessor, but it requires suitable software for the data transfer. Moreover, most of the microprocessors do not have the provision of SID and SOD lines. Apart from this, it can not offer high baud rate. Hence this chapter will deal the detailed discussion on serial communication and most powerful programmable communication interface IC 8251.

### 14.1 SERIAL DATA COMMUNICATION

There are two types of data communication, Parallel and Serial data communications. In parallel data communication each bit of a word is to be transmitted on a separate line along with a common ground line. In the 8085A microprocessor the data to be transmitted is of 8 bits. This parallel data transmission is not recommended for long distance communication as large number of data lines is needed. In serial data transmission, data is transmitted bit by bit either from the microprocessor to I/O devices or vice-versa. There are certain input/output devices in the microprocessor based systems, which only accept the serial data. The common I/O devices for such a data communication are CRT terminals, printers, cassette recorders etc. The microprocessors thus have the provision for transferring the data in the serial fashion. In such I/O devices, a parallel to serial conversion is required for the data transfer from the microprocessor to the peripheral device (figure 14.1a). Similarly, a serial to parallel conversion is required in transmitting the data from the peripheral device to the microprocessor (figure 14.1b).

As already discussed, the 8085A microprocessor has the provision of SID and SOD lines for the serial data transfer between the microprocessor and I/O devices. But it requires the software. Moreover, most of the microprocessors do not have the provision of these SID and SOD lines. Apart from this, it can not offer high baud rate. So, serial data communication is generally used for the long distance communication with high baud rate. Further, it requires less number of lines for the data transmission.

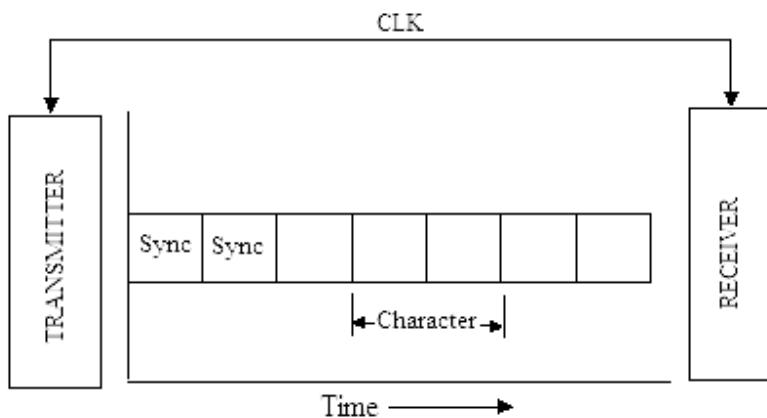


**Fig. 14.1**

The serial communication occurs either in the following two formats:

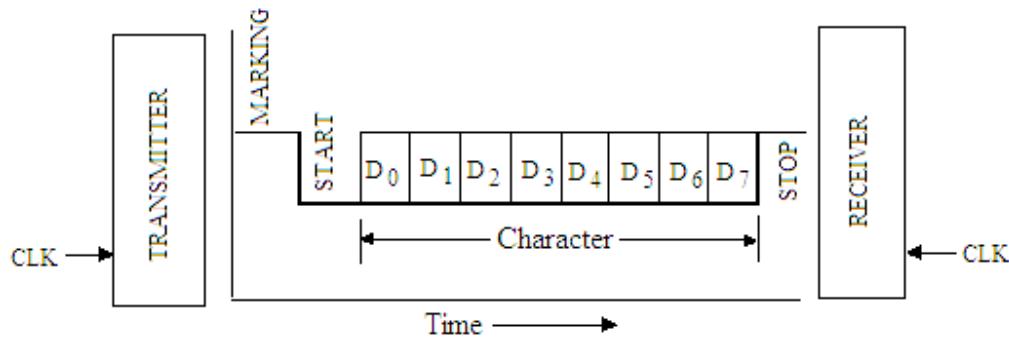
- Synchronous
- Asynchronous

In **synchronous** format, a receiver and a transmitter are **synchronized**. In this format the data is divided into fairly large blocks, typically 1000s of bytes. A block of characters is transmitted along with the **synchronization information**. The synchronizing signals are added once before each block. The synchronization characters mark the beginning of the transmission. This format is generally used for high baud rate (more than 20 K bits/second). The baud rate is defined as the number of bits transmitted or received per second. Figure 14.2 illustrates this format.



**Fig. 14.2**

In **asynchronous** format, timing signals are added to each character of data. A **START bit is added at the beginning** of each character. A **STOP bit is added at the end** of the character data. A low signal is used for the START bit and a high signal is used for STOP bit. When no data is being transmitted, a receiver stays high, called **MARK**. The process of adding start and stop bits to a character is known as **FRAMING**. The asynchronous format is generally used for low speed transmission. This format is illustrated in figure 14.3.



**Fig. 14.3**

The serial mode of data transfer can be divided into three groups namely:

Simplex Method

Duplex Method

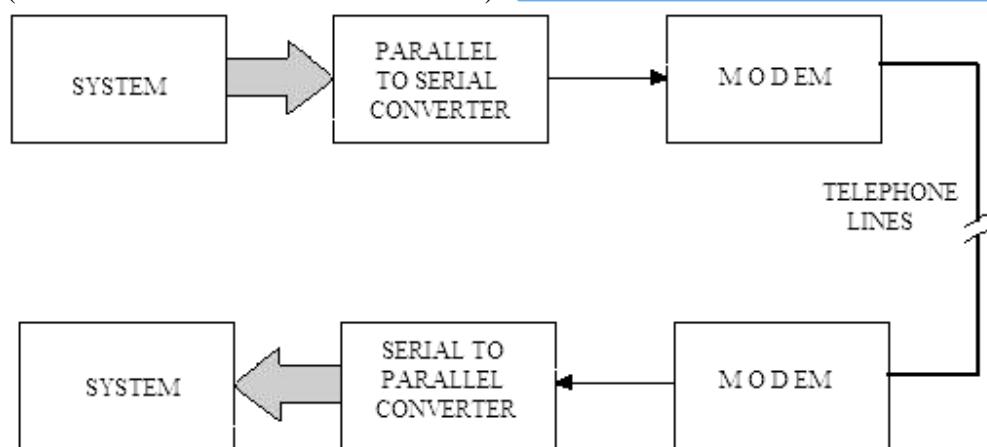
Half Duplex Method

In **simplex** method data transfer takes place in unidirectional i.e. either from the system to the I/O devices or from I/O devices to the system, of course through the serial link. A typical example is the transmission from a system to a printer.

In the **duplex** method data transfer takes place in both the directions. However, if the transmission goes one way at a time, it is called half duplex; if it goes both ways simultaneously, it is called full duplex. The walkie talkie is an example of half duplex and the communication between the computers is an example of full duplex.

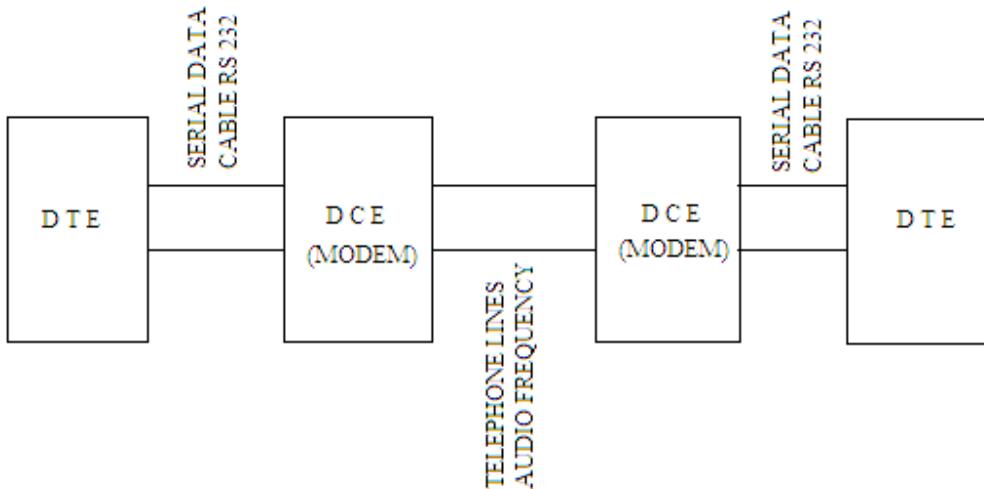
#### 14.2 MODEM

One of the most common applications of the serial transmission would be data transmission between large computers. For such serial digital transmission the existing telephone lines are used. The telephone lines are designed to carry analog signals in the range of 40 Hz to 4 KHz. These lines will not readily support digital signals. Therefore, the digital data is to be converted into audio frequency format. A device that can convert the digital value in the form of the audio signal has to be used because the basic binary bits can not be placed directly on these lines. Such a device is known as **MODEM (MODULATOR- DEMODULATOR)**. The modem is to be used at both the ends; at the



**Fig. 14.4**

receiving end as well as at the transmitting end. The modem at the transmitting end converts digital signal into analog signal format (audio frequency range) and at the receiving end the modem converts audio signal back to the digital data.



**Fig. 14.5**

Modems often use frequency shift keying technique for the conversion. It converts a 1 level signal into an audio signal of 1200 Hz and 0 level into audio signal of 2200 Hz. These converted audio signals are modulated on the carrier and then transmitted to the telephone lines. However at the receiving end another modem is used to convert the audio signal back to digital form. Figure 14.4 shows the block diagram of such transmission system. The originators and receptors of the digital data are called Data Terminal Equipments (DTE) and the modem units are called Data Communication Equipments (DCE). Figure 14.5 shows the modem connections using telephone lines.

### 14.3 SERIAL COMMUNICATION STANDARD

There are primarily two standards for transmitting the information in serial form. All the serial I/O devices work on either of these standards.

- Current Loop
- Voltage standard

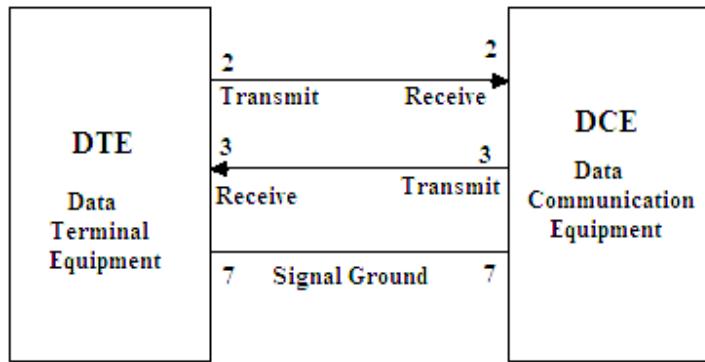
In a current loop method, the current flows through the lines for the logic 1 and no current flow for the logic 0.

The two standards for the current loop are:

20 milliampere loop  
60 milliampere loop.

A current loop reduces noise pickup and is suitable for long distance transmission.

The other standard is the voltage signal. When the data is transmitted as voltage, the commonly used standard is known as RS 232C. It is defined in reference to DTE (Data Terminal Equipment) and DCE (Data Communication Equipment) – terminal and Modem as shown in figure 14.6, which illustrates the connections of DTE to DCE. This standard was developed before the existence of TTL logics; its voltage levels are not compatible with TTL logic levels.

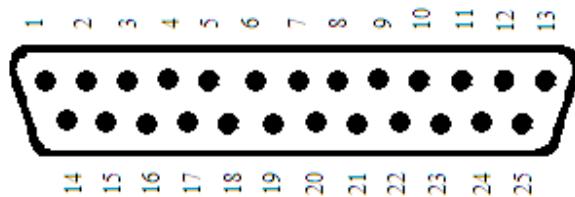


**Fig. 14.6**

### RS 232C

Figure 14.7 shows the [RS 232C 25-pin connector](#). The signals are divided into four groups namely:

- Data Signals
- Control Signals
- Timing Signals
- Ground



**Fig. 14.7**

The pin assignments are given below:

- |         |   |
|---------|---|
| Pin 1:  | Protective Ground                               |
| Pin 2:  | Transmitted Data (T x D) → DCE                  |
| Pin 3:  | Received Data (R x D) → DTE                     |
| Pin 4:  | Request to Send (RTS) → DCE                     |
| Pin 5:  | Clear to Send (CTS) → DTE                       |
| Pin 6:  | Data Set Ready → DTE                            |
| Pin 7:  | Signal Ground                                   |
| Pin 8:  | Received Line Signal Detector                   |
| Pin 9:  | Reserved for Data Set Testing                   |
| Pin 10: | Reserved for Data Set Testing                   |
| Pin 11: | Unused  |
| Pin 12: | Secondary Received Line Signal Detector         |
| Pin 13: | Secondary Clear to send                         |
| Pin 14: | Secondary Transmitted Data                      |
| Pin 15: | Transmission Signal Element Timing (DCE Source) |

|         |   |
|---------|---|
| Pin 16: | Secondary Received Data                     |
| Pin 17: | Receiver Signal Element Timing (DCE Source) |
| Pin 18: | Unused                                      |
| Pin 19: | Secondary Request to Send                   |
| Pin 20: | DCE ← Data Terminal Ready (DTR)             |
| Pin 21: | Signal Quality Detector                     |
| Pin 22: | Ring Indicator                              |
| Pin 23: | Data Signal Rate Selector (DTE/DCE Source)  |
| Pin 24: | Transmit Signal Element Timing (DTE Source) |
| Pin 25: | Unused                                      |

Technically the RS232C has -3V to -12V for logic '1' and +3V to +12V for logic '0'. This is negative logic. Because of incompatibility with TTL logic, voltage translator called line drivers and line receivers are required to interface TTL logic with RS232 signals. As shown in figure 14.6, the minimum interface requires three lines: pins 2, 3 and 7. These lines are defined in relation to the DTE; the terminal transmits on pin 2 and receives on pin 3. On the other hand, the DCE transmits on pin 3 and receives on pin 2.

The comparative study of synchronous and Asynchronous modes of data transfer to and from the serial I/O devices is given in table 14.1.

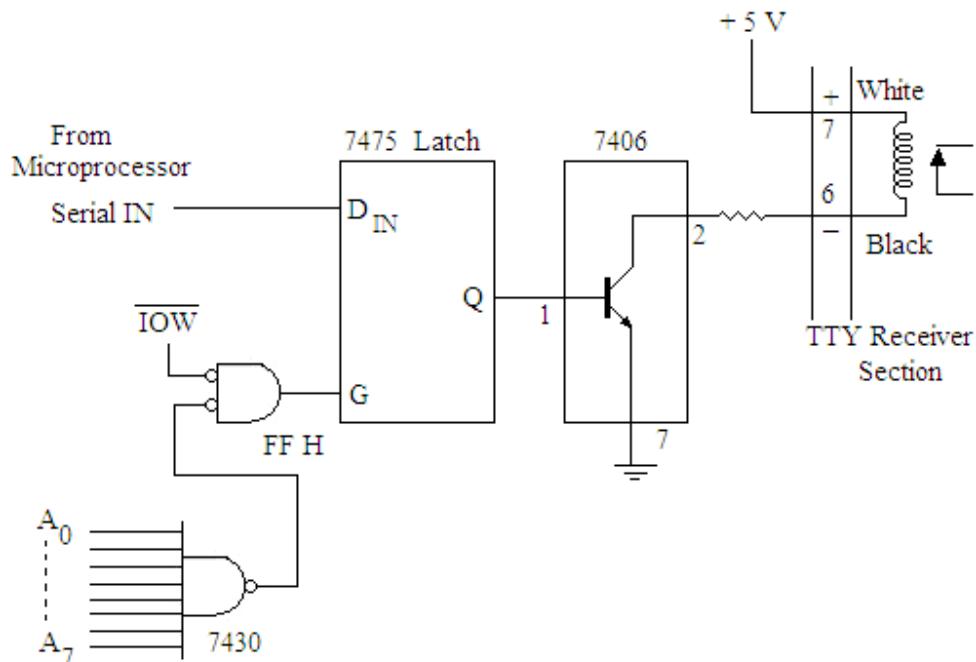
**Table 14.1**

| Sr.<br>No. | Format                      | Serial Transmission                              |   |
|------------|-----------------------------|--|---|
|            |                             | Synchronous                                      | Asynchronous  |
| 1.         | Data format                 | Groups of characters                             | One character at a time.                              |
| 2.         | Framing                     | Synchronous characters are sent with each group. | START and STOP bit/bits are sent with each character. |
| 3.         | Speed                       | High speed, 20 K bits/sec or more.               | Low, Less than 20 K bits/sec.                         |
| 4.         | Distance for communication. | Recommended for long distance communication.     | Recommended for small distance communication.         |
| 5.         | Implementation              | Hardware.  | Hardware or Software.                                 |
| 6.         | Data direction              | Simplex, half and full duplex.                   | Simplex, half and full duplex.                        |

The computer communicates with the serial I/O devices either in Asynchronous or Synchronous modes. Here the discussion will be made on the Asynchronous mode of serial data transfer, as it is both hardware and software implemented.

#### 14.4 ASYNCHRONOUS SOFTWARE APPROACH

In this case, the framing of each character is done by introducing START and STOP bits using the software. The data in parallel form is available with the microprocessor to be transmitted to serial I/O devices. The data is then converted to serial form using the software and sent to the serial I/O devices through an interfacing circuit. The interfacing circuit is to be designed for the microprocessor 8085A microprocessor is shown in figure 14.8. This circuit consists of D-type flip-flop (IC 7475) whose device address is chosen to be FF H. The address bus is decoded using an 8-input NAND gate (IC 7430), and combined with the control signal  $\overline{IOW}$  to clock the latch. When the instruction OUT FF H is executed, the clock goes high and bit at the  $D_{IN}$  terminal is transferred to output Q of the flip-flop. The bit is latched as the clock goes low. If the Q is high 20 mA current flows through the TTY (Tele Type) circuit.

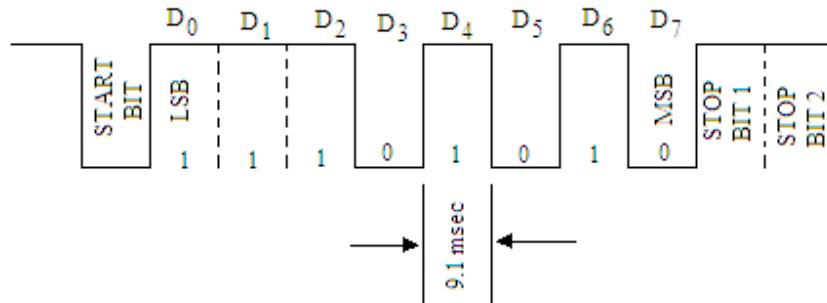


**Fig. 14.8**

Let us consider, a message is to be transmitted to teletype (TTY) using the 8085A microprocessor. The interfacing circuit, shown in figure 14.8, is used for this transmission. The characters of the message are stored in the memory locations starting at 2101 H. Each character includes framing information (START and STOP bits). Each character has 11 bits for the transmission; 8 bits for ASCII character, one START bit and two STOP bits. Let the first character of the message is W (ASCII code for W is 57 H). Figure 14.9 shows a stream of eleven bits for this first ASCII character of the message. The character bits are transmitted beginning with the least significant bit (LSB)  $D_0$ . The

bit time, the delay between two successive bits, is determined by the transmission rate (baud rate).

ASCII CODE FOR W = 47 H  
= 01010111



**Fig. 14.9**

A typical (TTY) sends (or receives) 10 ASCII characters per second. If each character has 11 bits, then TTY transmits 110 bits/second.

$$\text{TTY transmission rate} = 110 \text{ bits/second}$$

$$\begin{aligned}\text{Time for each bit} &= \frac{1 \text{ sec}}{110} \\ &= 9.1 \text{ msec.}\end{aligned}$$

Therefore, microprocessor should send bit by bit information (including framing bits) to TTY to transmit the bit a time interval of 9.1 msec.

For the transmission the necessary software for 8085A is given as:

### PROGRAM

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
|       | LXI SP,   | XXXX H  | ; Initialize Stack pointer.  |
|       | LXI H,    | 2101 H  | ; Initialize H-L register pair for index pointer.  |
|       | MVI A,    | 01 H    | ; Set up MARK bit as 1.  |
|       | OUT       | FF H    | ; Goes to the output of Flip-flop, whose address is FF H.                                  |
| NXT   | MOV A,    | M       | ; Accumulator is loaded with the character of the message.                                 |
|       | CPI       | 0D H    | ; Compare if the character is Carriage return (0D H is the ASCII Code of carriage return). |
|       | MOV B,    | A       | ; Save the character in register B.  |
|       | JZ        | END     | ; If the character is Carriage return, jump to END.  |

|     |             |               |   |
|-----|-------------|---------------|---|
|     | CALL        | SERIAL        | ; Else Call subroutine for converting into the serial form of the character.  |
|     | INX H       |               | ; Point to next character of the message.   |
| END | JMP<br>CALL | NXT<br>SERIAL | ; Jump for next character.<br>; Call subroutine for converting into the serial form of the character 'Carriage return'. |
|     | MVI B,      | 0A H          | ; Load the character of Line feed (0A H is the ASCII code for line feed).   |
|     | CALL<br>HLT | SERIAL        | ; Transmit line feed serially.<br>; Stop Processing.  |

### SUBROUTINE PROGRAM:

This is the subroutine for the conversion of bit of a character into serial form and then transmits to a TTY with a baud rate of 110.

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>   |
|--------------|------------------|----------------|---|
| SERIAL       | MVI C,           | 08 H           | ; Load register C with 08 H, which is used as counter.                              |
|              | MVI A,           | 00 H           | ; Load START bit.   |
|              | OUT              | FF H           | ; Send the start bit at the output of the flip-flop.                                |
|              | CALL             | DELAY          | ; Call delay subroutine which introduces a time delay of 9.1 msec after START bit.  |
|              | MOV A,           | B              | ; Load the character into the accumulator from register B.                          |
| NXT1         | OUT              | FF H           | ; Send the bit.   |
|              | CALL             | DELAY          | ; Call delay subroutine which introduces a time delay of 9.1 msec between each bit. |
|              | RRC              |                | ; Rotate right for the serial conversion.   |
|              | DCR C            |                | ; Decrement C.  |
|              | JNZ              | NXT1           | ; If rotated 8 times then jump to NXT1.   |
|              | MVI A,           | 01 H           | ; Introduce STOP bit.   |
|              | OUT              | FF H           | ; STOP is sent to the output of the flip-flop.                                      |
|              | CALL             | DELAY          | ; Call delay subroutine which introduces a time delay of 9.1 msec.                  |

CALL            DELAY ; Call delay subroutine which introduces a time delay of 9.1 msec.

RET            ; Go back to main program.

Delay Program may be written for the time delay of 9.1 msec.

Similar, circuit and software may be used for the reception of the messages in the serial fashion.

The RIM and SIM instructions can also be used for the transmission and reception of data in serial fashion.

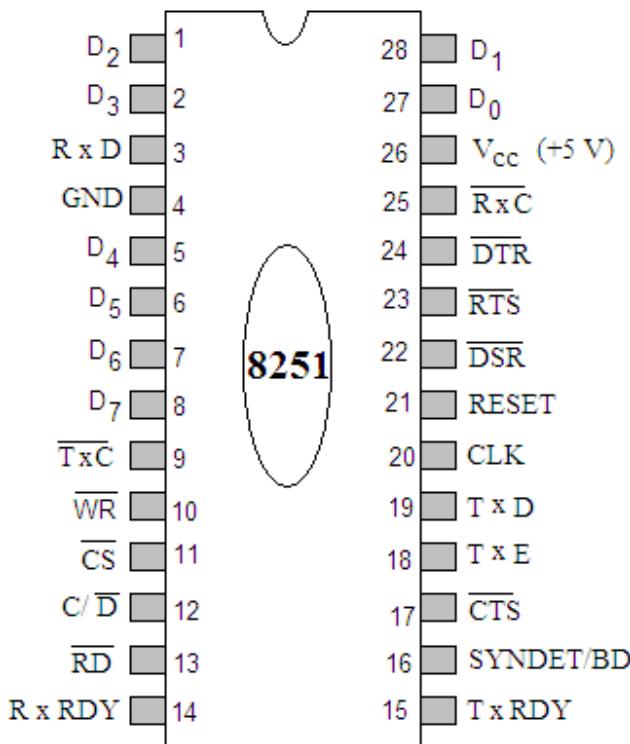
#### **14.5 PROGRAMMABLE COMMUNICATION INTERFACE**

Many types of UART (Universal Asynchronous Receiver Transmitter) and **USART** (**Universal Synchronous/Asynchronous Receiver Transmitter**) have been developed for data transfer between serial I/O device and the microcomputer in the form of the chips. The USART is most powerful and commonly used LSI chip. It provides a programmable serial communication interface between the microprocessor and serial I/O devices. The data transfer between the microprocessor and USART takes place in the parallel form. On the other hand the data transfer between the I/O devices and the USART is in the serial form. The USART has to be initialized before the data transfer takes place. The desired data format and synchronization method for the data transfer are specified during initialization by command byte written by the microprocessor for the USART. Thus an USART works as parallel to serial converter from microprocessor to I/O devices and vice-versa. Here we shall discuss the details of the popular USART chip 8251.

#### **14.6 BLOCK DIAGRAM OF 8251A**

The 8251A is a programmable communication interface available in the form of IC dual in line package. It consists of 28 pins and requires + 5 V d.c. supply for its operation. Figure 14.10 shows the pin diagram of 8251A. The pin details are as given below:

|                                |                                       |                 |                      |
|--------------------------------|---------------------------------------|-----------------|----------------------|
| D <sub>0</sub> -D <sub>7</sub> | Data bus (8 bits)                     | RxC             | Receiver Clock       |
| C/D                            | Control or data to be written or read | RxD             | Receiver Data        |
| <u>RD</u>                      | Read Data                             | RxRDY           | Receiver Ready       |
|                                | Command                               | TxRDY           | Transmitter Ready    |
| <u>WR</u>                      | Write Data or Control Command         | DSR             | Data Set Ready       |
|                                |                                       | DTR             | Data Terminal Ready  |
| <u>CLK</u>                     | Clock Pulse (TTL)                     | SYNDET/BD       | Sync/Break Detect    |
| <u>CS</u>                      | Chip Enable                           | RTS             | Request to Send Data |
| RESET                          | RESET Input                           | CTS             | Clear to Send Data   |
| <u>TxC</u>                     | Transmitter Clock                     | TxE             | Transmitter Empty    |
| TxD                            | Transmitter Data                      | V <sub>CC</sub> | + 5 V                |
|                                |                                       | GND             | Ground               |



**Fig. 14.10**

The functional block diagram of 8251A is shown in figure 14.11. It includes the following four sections:

- **Read/Write Control Logic**
- **Transmitter Section**
- **Receiver Section**
- **Modem Control**

#### 14.6.1 Read/Write Control Logic

This section includes 6-input signals:  $\overline{CS}$ ,  $C/\overline{D}$ ,  $\overline{WR}$ ,  $RESET$  and  $CLK$ ; and three buffer registers:

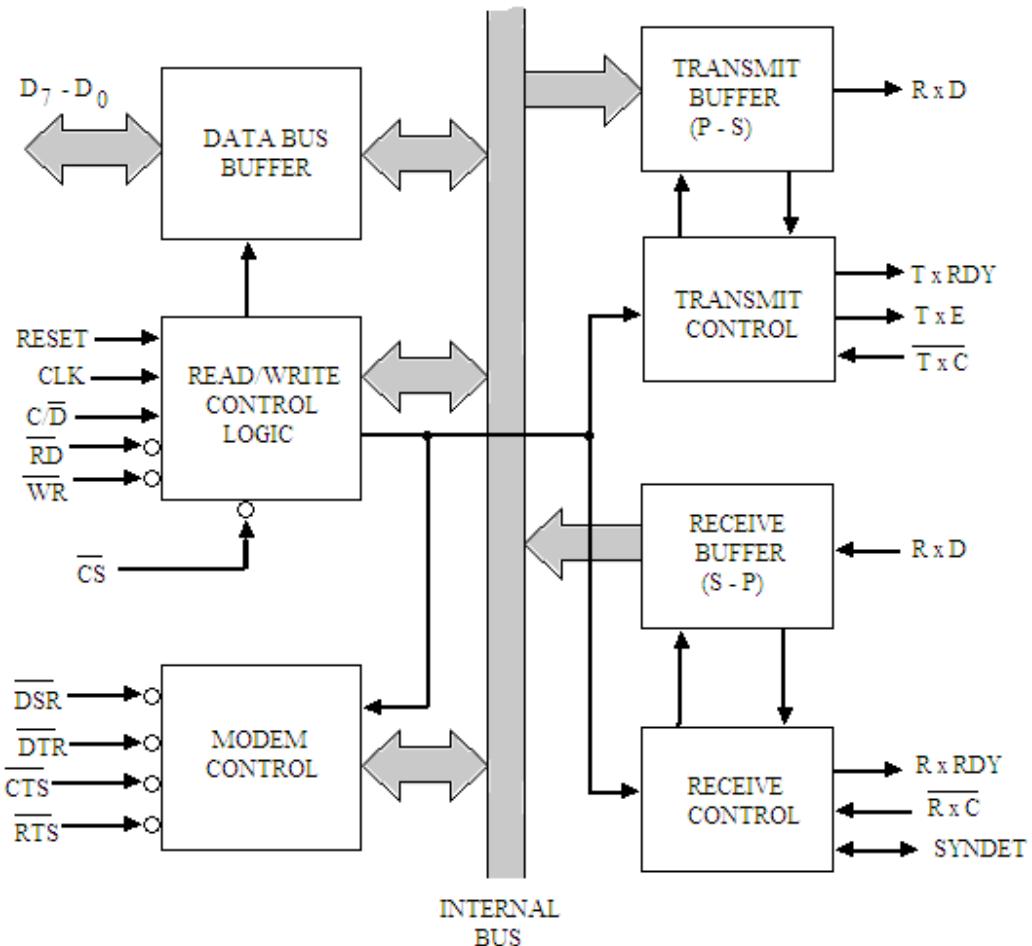
- **Control Register**
- **Status Register**
- **Data Bus Buffer Register**

$\overline{CS}$  : It is a Chip Select terminal. A low to this terminal selects the 8251A for communication with the microprocessor. This is connected to a decoded address bus.

$C/\overline{D}$  : It is a Control/Data pin. A high on this terminal addresses the control register or status register; whereas a low addresses the data bus buffer.

$\overline{WR}$  : It is an active low Write signal. When a low signal is applied to this terminal, the microprocessor either writes in the control word register or

sends output to the data buffer. This pin is connected to either  $\overline{IOW}$  or  $\overline{MRMW}$ .



**Fig. 14.11**

**RD** : It is an active low Read signal. When this terminal is low, the microprocessor either reads the status from the status register or accepts data from the data buffer. This is connected to either  $\overline{IOR}$  or  $\overline{MEMR}$ .

**RESET** : It is a Reset input pin. When this terminal is high, it resets the 8251 and forces it in the idle mode.

**CLK** : This is the Clock input, usually connected with the system clock.

The details of the three registers are as:

#### **Control Register:**

This is a 16-bit register and contains the control word with two independent bytes.

The first byte is called the Mode Instruction Word, and

the second byte is called the Command Instruction Word.

This register can be accessed as an output port when **C/D** pin is high.

#### **Status Register:**

This input register checks the Ready status of a peripheral. This register is addressed as an input port when  $C/D$  terminal is high. It has the same port address as the control register.

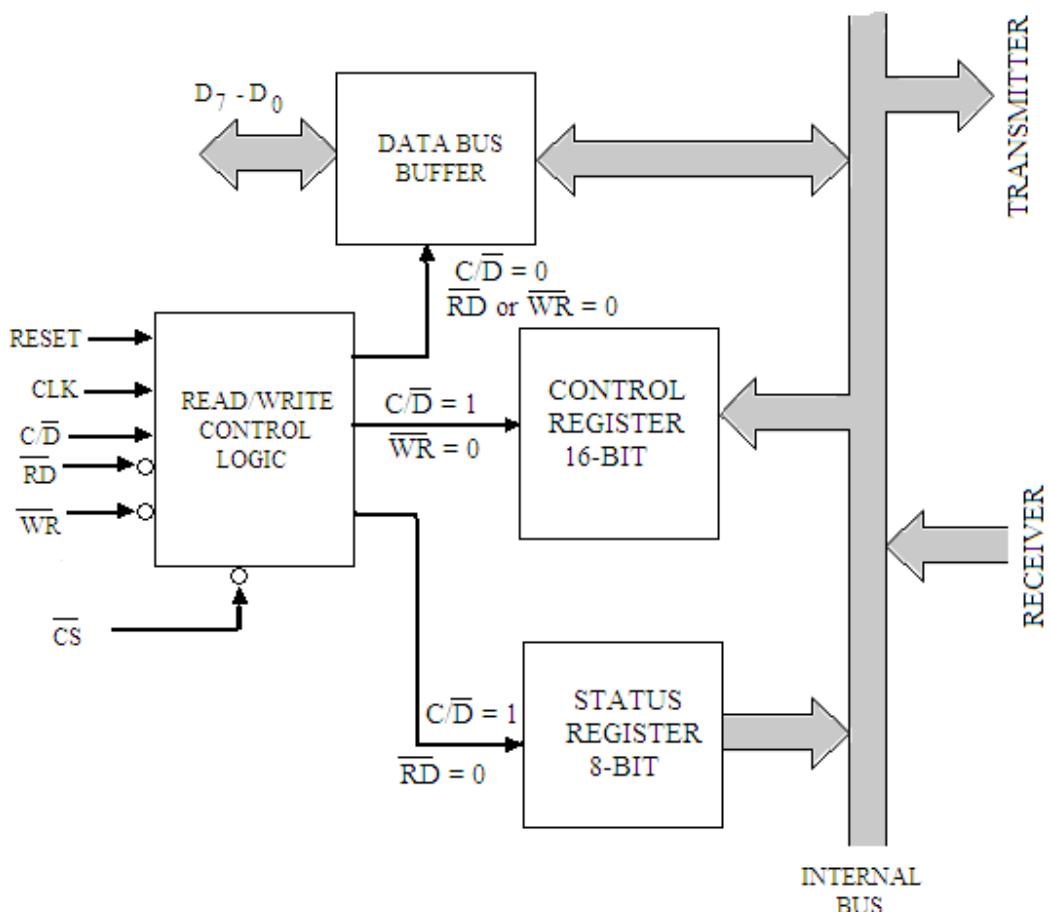
### Data Bus Buffer Register

This is a bidirectional register and can be addressed as an input port and an output port when  $C/D$  pin is low. Table 14.2 gives the summary of interfacing and control signals.

**Table 14.2**

| $\overline{CS}$ | $C/D$ | $\overline{RD}$ | $\overline{WR}$ | Functions  |
|-----------------|-------|-----------------|-----------------|--|
| 0               | 1     | 1               | 0               | Microprocessor writes instruction in USART control register. |
| 0               | 1     | 0               | 1               | Microprocessor reads from USART status register.             |
| 0               | 0     | 0               | 1               | Microprocessor outputs data to USART data buffer.            |
| 0               | 0     | 0               | 1               | Microprocessor accepts data from USART data buffer.          |
| 1               | X     | X               | X               | USART not selected.  |

Figure 14.12 shows the expanded version of this section.



**Fig. 14.12**

### 14.6.2 Transmitter Section

The expanded block diagram of the transmitter section is shown in figure 14.13. It consists of the following registers:

- Transmitter Buffer Register – It holds 8-bit data.
- Serial Output Register – Converts 8-bits into a stream of serial bits.
- Transmitter Control Logic – It directs the output register to send the serial data at the output register through TxD terminal.

Three output signals and one input signal are also associated with the transmitter section. These signals are described as:

*TxD* : It is **Transmit Data** terminal. The serial bits are transmitted on this line.

*TxC* : This pin is **Transmitter Clock**. This controls the rate at which bits are to be transmitted by the 8251A. The clock frequency can be 1, 16 or 64 times of the baud.

*TxRDY* : This is **Transmitter Ready** pin. When this output terminal is high, it indicates that the buffer is empty and the 8251 is ready to accept a byte. It can be used either to interrupt the microprocessor or to indicate the status. This signal becomes reset when the data byte is loaded into the transmitter buffer register.

*TxE* : This is **Transmitter Empty** signal used as output terminal. A high on this terminal indicates that the output register is empty and becomes reset when a byte is transferred from the buffer register to the output register.

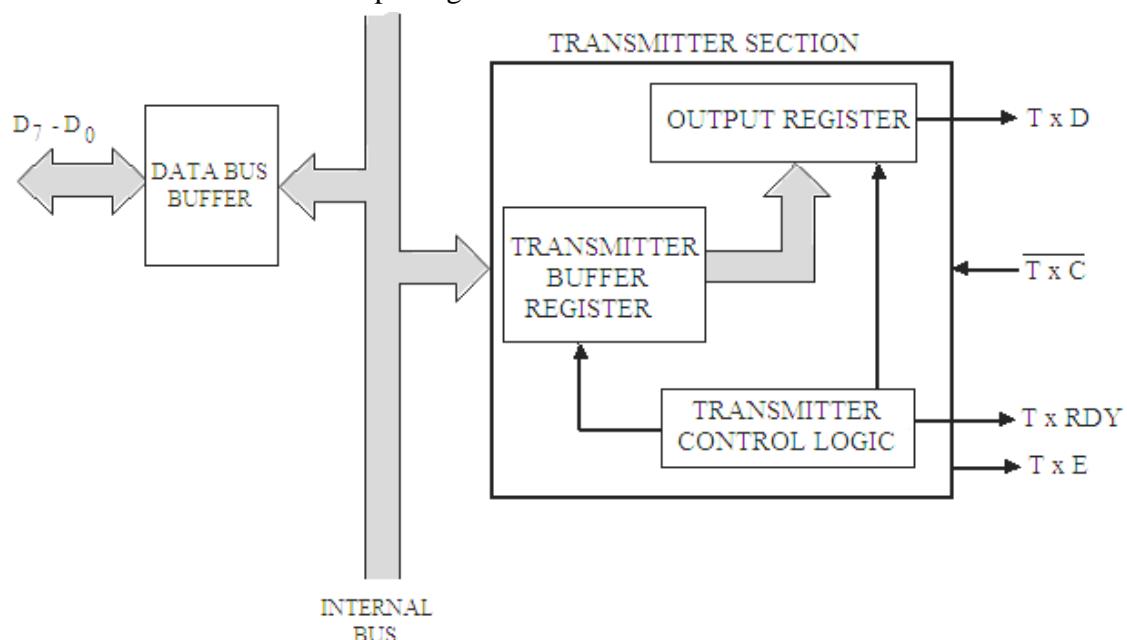


Fig. 14.13

### 14.6.3 Receiver Section

The expanded block diagram of the receiver section is shown in figure 14.14. It consists of the following registers:

- Receiver Buffer Register
- Serial Input Register
- Receiver Control Logic

The receiver section accepts serial data on the Rx D terminal and converts it to parallel data. When the Rx D line goes low, the control logic assumes that it is a START bit and waits for half a bit time, and samples the line again. If the line is still low, the input register accepts the next coming bits and forms a character. The character is then loaded to the buffer register. Subsequently, the parallel byte is transferred to the microprocessor when a request is made. Following signals are associated with the receiver section which are described below:

- RxD* : It is **Receive Data** terminal. The serial bits are received on this line and converted to a parallel byte in the receiver input register.
- RxC* : This pin is **Receiver Clock**. This controls the rate at which bits are received by the 8251A. In the asynchronous mode, the clock can be set to 1, 16 or 64 times of the baud.
- RxRDY* : This is **Receiver Ready** pin. When this output terminal is high, the 8251A has a character in the buffer register and is ready to transfer it to the microprocessor. It can be used either to indicate the status or to interrupt the microprocessor.

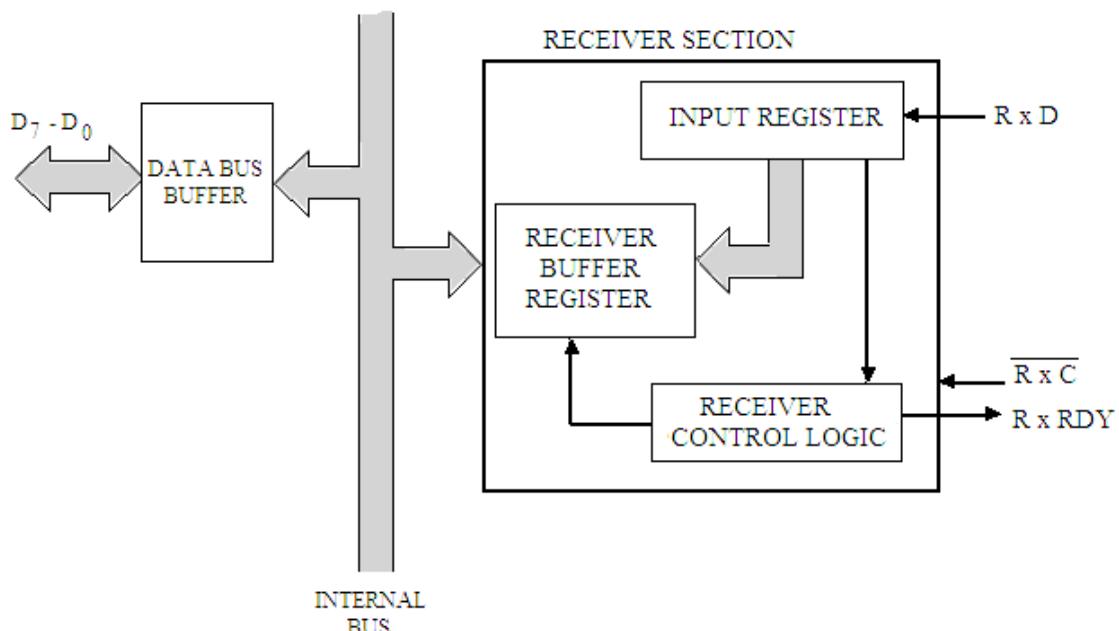


Fig. 14.14

### 14.6.4 Modem Control

The modem control section of the 8251A provides two input signals  $\overline{DSR}$  (Data Set Ready) and  $\overline{CTS}$  (Clear to Send); and two output control signals  $DTR$  (Data Terminal Ready) and  $RTS$  (Request to Send) to handle DTE and DCE. These signals are described as:

- $\overline{DSR}$  (Data Set Ready):** This is an active low input terminal used by the modem to indicate that it is ready for communication.
- $\overline{CTS}$  (Clear to Send):** This active low input terminal is used by the modem to signal the DTE that the communication channel is clear and it can send out the serial data.
- DTR (Data Terminal Ready):** This output signal is used by the 8251 to signal the modem to indicate that the terminal is ready to communicate.
- RTS (Request to Send):** This output signal is used by the 8251A to signal to modem that it has data to be transmitted.

#### 14.7 INTERFACING OF 8251A

Figure 14.15 shows the interfacing connection of 8251A with the microprocessor. The eight data lines of 8251A are to be connected to the data bus of the microprocessor.

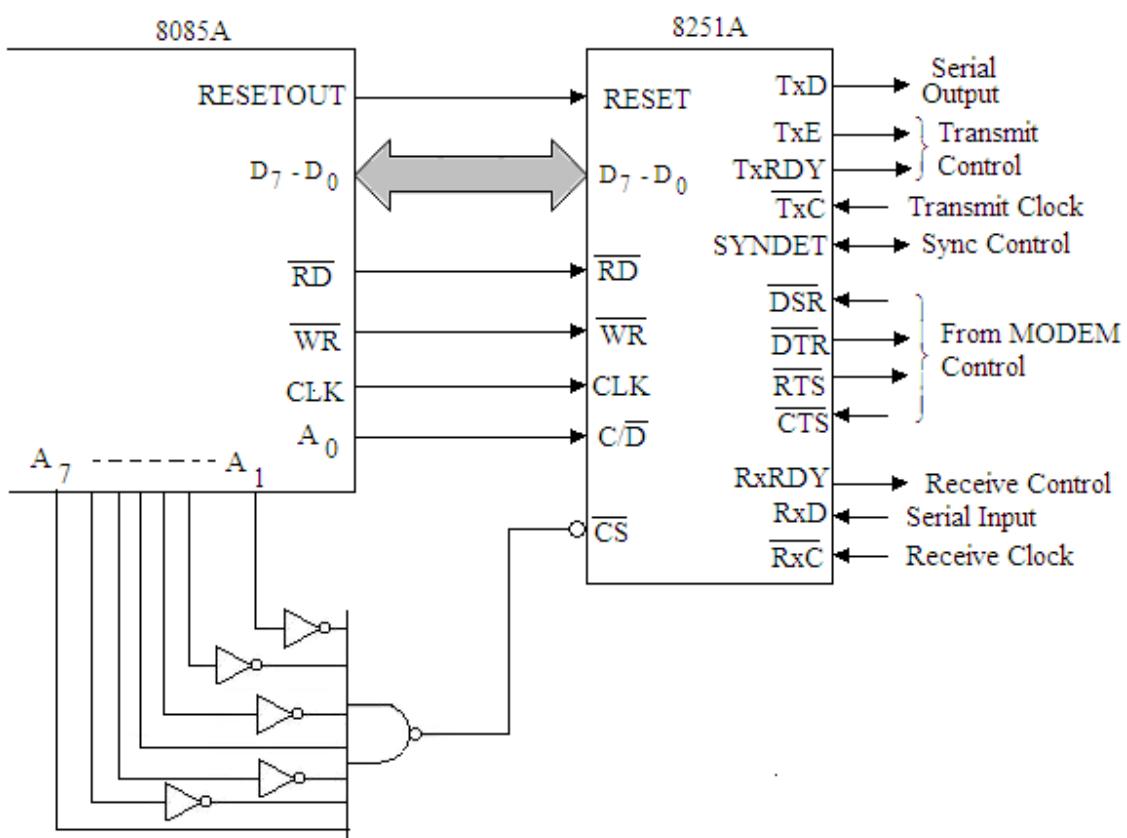


Fig. 14.15

The  $\overline{RD}$  and  $\overline{WR}$  lines of 8251A are to be connected to the  $\overline{RD}$  and  $\overline{WR}$  of the control lines of 8085A microprocessor respectively. The CLK pin of the 8251A is to be connected to the CLK out terminal of the microprocessor. The RESET pin is connected to the RESETOUT of the microprocessor. The terminal  $C/\overline{D}$  is used to select the internal registers either control register or data register. So it is connected to the  $A_0$  address line of the microprocessor. The chip select terminal  $\overline{CS}$  of the PCI 8251A is to be connected to the output of an address decoder circuit. The address decoder circuit uses the address lines  $A_1$  to  $A_7$  of the microprocessor. From the decoder circuit used in this figure, the chip select terminal of 8251A will be enabled when  $A_7$  and  $A_4$  are high and all other inputs are low. The port addresses for reading the data word, writing the data word, reading the status word and writing the control word will be as shown in table 14.3.

**Table 14.3**

| <b><math>A_0</math></b> | <b><math>\overline{RD}</math></b> | <b><math>\overline{WR}</math></b> | <b>Function</b>    | <b>Port Address</b> |
|-------------------------|-----------------------------------|-----------------------------------|--------------------|---------------------|
| 0                       | 0                                 | 1                                 | Read Data Word     | 90 H                |
| 0                       | 1                                 | 0                                 | Write Data Word    | 90 H                |
| 1                       | 0                                 | 1                                 | Read Status Word   | 91 H                |
| 1                       | 1                                 | 0                                 | Write Control Word | 91 H                |

## 14.8 PROGRAMMING OF 8251A

It has already been discussed that there is a 16 bit control register in a 8251A which contains two independent bytes (words). The first byte (word) is known as mode word which tells about the initialization parameters like mode (Asynchronous or Synchronous), baud, stop bits and parity bits etc. The second byte (word) is known as command word which tells about enabling the transmitter and receiver sections. The readiness of the peripherals can also be checked by reading the status word.

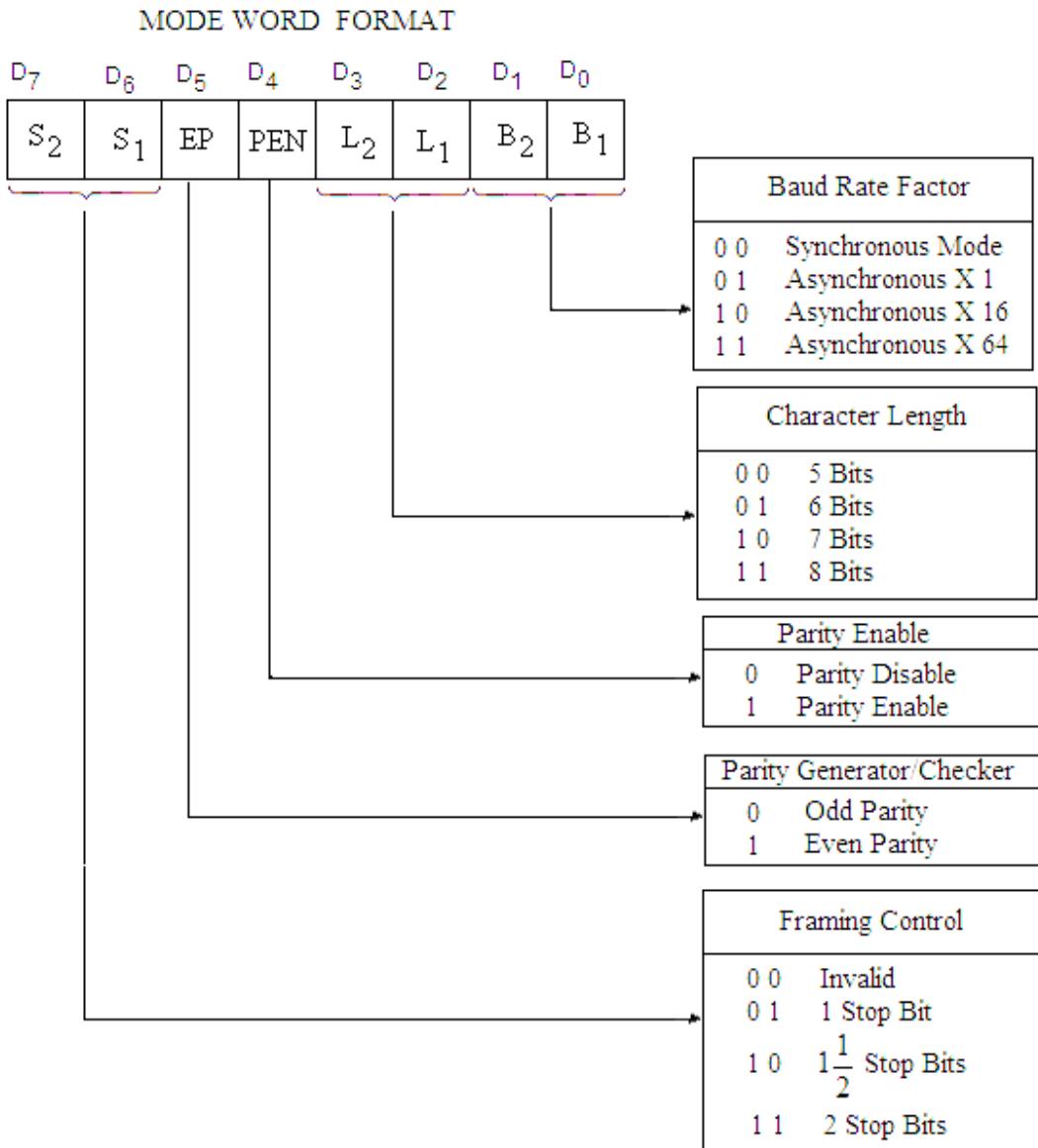
### 14.8.1 Initialization of 8251A in Asynchronous Mode

To initialize 8251A in asynchronous mode, a certain sequence of control words must be followed. After a reset operation (through system RESET or through instruction) a mode word must be written into the control register followed by a command word. Any control word written into the control register immediately after the mode word will be interpreted as a command word that means a command word can be changed anytime during the operation. However, the 8251A should be reset prior to writing a new mode word, and it can be reset using internal reset bit ( $D_6$ ) in the command word.

Figure 14.16 shows the format of the mode word. The bits of this mode word are described below:

Bits  $D_1-D_0$ :

These bits program the baud rate factor. These bits are set to 00 for synchronous operation. However, for asynchronous operation these bits specify the factor by which the transmit/receive clocks  $\overline{TxC}$  and  $\overline{RxC}$ , exceeds the baud rate. The other clock inputs CLK to the 8251A is used to generate the internal device timing and must simply be greater than 30 times the transmitter/receiver baud rate.



**Fig. 14.16**

Bits D<sub>3</sub>-D<sub>2</sub>:

These bits specify the number of data bits in the character is to be sent or received.

Bit D<sub>4</sub>:

This bit enables the parity. If this bit is 0, the parity is disabled.

Bit D<sub>5</sub>:

This bit selects the even or odd parity. If this bit is 0, odd parity is selected. The even parity is selected if this bit is 1.

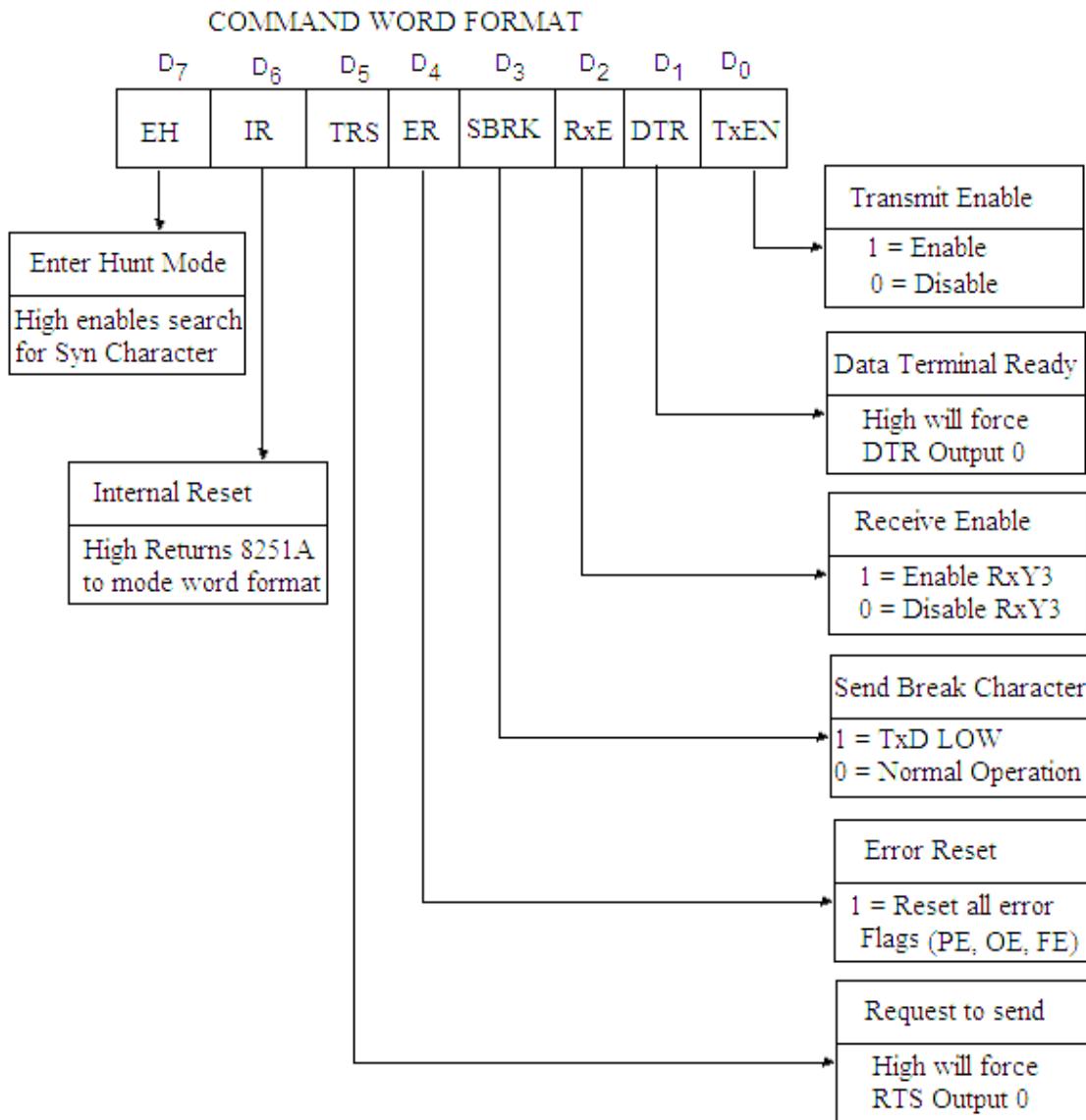
Bit D<sub>6</sub>-D<sub>7</sub>:

These bits specify the number of stop bits.

### Command Word

Once the function definition of the 8251A has been programmed by the mode word instruction, the device is ready for data communication. The command word

instruction controls the actual operation of the selected format. Functions such as Enable Transmit/ Receive, Error Reset and modem control are provided by the command word instruction. The format of the command word is shown in figure 14.17.



**Fig. 14.17**

The bits of the command word are described below:

- |                      |   |
|----------------------|---|
| Bit D <sub>0</sub> : | This bit enables the Transmitter.                                   |
| Bit D <sub>1</sub> : | This bit controls the Data Terminal Ready.                          |
| Bit D <sub>2</sub> : | This bit enables the Receiver.                                      |
| Bit D <sub>3</sub> : | This bit makes the transmitter to send continuous break characters. |

- Bit D<sub>4</sub>: This resets the error flags. It resets Parity Error flag (PE), Over Run Error flag (OE) and Framing Error flag (FE).
- Bit D<sub>5</sub>: This bit controls the request to send to RTS pin of the device. A high at this bit makes the RTS pin to become active.
- Bit D<sub>6</sub>: It is used as an internal reset. A high to this bit resets the device, so that a new mode word can be entered.
- Bit D<sub>7</sub>: It is used in synchronous mode. It enables the receiver to look for the synchronous data.

### Programming Sequence

When programming the 8251A in asynchronous mode, following sequence must be followed:

1. Reset (either internal or external)
2. Mode Instruction (specify the asynchronous mode)
3. Command Instruction

The resetting of the 8251A can be done by loading the control register with a command word whose D<sub>6</sub> bit is high. This bit reset the device. The command word is, therefore, loaded as:

|             |   |
|-------------|---|
| MVI A, 40 H | ; Command word whose D <sub>6</sub> bit is high is loaded to accumulator.                     |
| OUT 91 H    | ; Command word is loaded to control register whose port address is 91 H as discussed earlier. |

Now the mode instruction word is loaded to the control register. The mode word is formed as per the required modes of transmission. For example, for asynchronous transmission with 7 data bits, 2 stop bit and odd parity; also a 16 X clock is used. For this the format is shown in figure 14.18 and mode word will be given as:

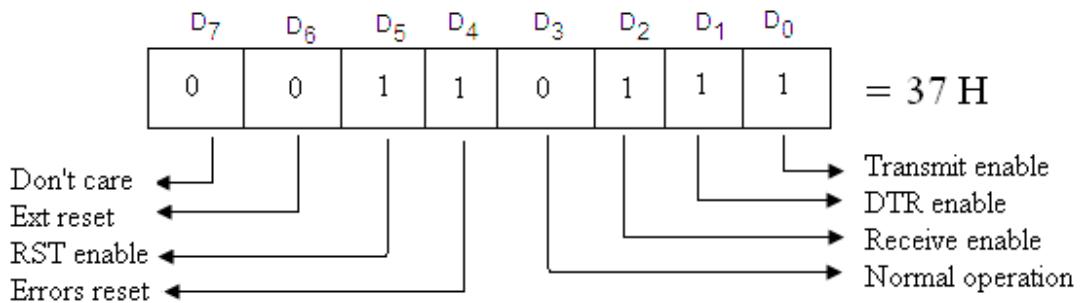
| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub>         | D <sub>2</sub> | D <sub>1</sub>    | D <sub>0</sub> |
|----------------|----------------|----------------|----------------|------------------------|----------------|-------------------|----------------|
| 1              | 1              | 0              | 1              | 1                      | 0              | 1                 | 0              |
| Two Stop Bits  |                | Odd parity     |                | Character of<br>7 bits |                | Baud Rate<br>X 16 |                |

= DA H

Fig. 14.18

|             |  |
|-------------|--|
| MVI A, DA H | ; Mode word is loaded to accumulator.        |
| OUT 91 H    | ; Mode word is loaded into control register. |

The command word with RTS, error reset and DTR enable with be given as shown in figure 14.19.



**Fig. 14.19**

The command word to be loaded in the control register is given below:

```
MVI A, 37 H ; Command word is loaded to accumulator.  
OUT 91 H ; Command word is loaded into control  
register.
```

For initializing the 8251A a dummy mode is sent before resetting it. The resetting is done using command word with  $D_6$  bit as 1. The mode word followed by the command word is sent. With all these requirements the initialization program of 8251A is written as:

```
MVI A, 00 H  
OUT 91 H ; Dummy mode word.  
MVIA, 40 H ; Resetting of 8251A by using the command  
word with  $D_6$  bit as high.  
  
OUT 91 H ; Mode word is loaded to accumulator.  
MVI A, DA H ; Mode word is loaded into control register.  
OUT 91 H ; Command word is loaded to accumulator.  
MVI A, 37 H ; Command word is loaded into control  
register.
```

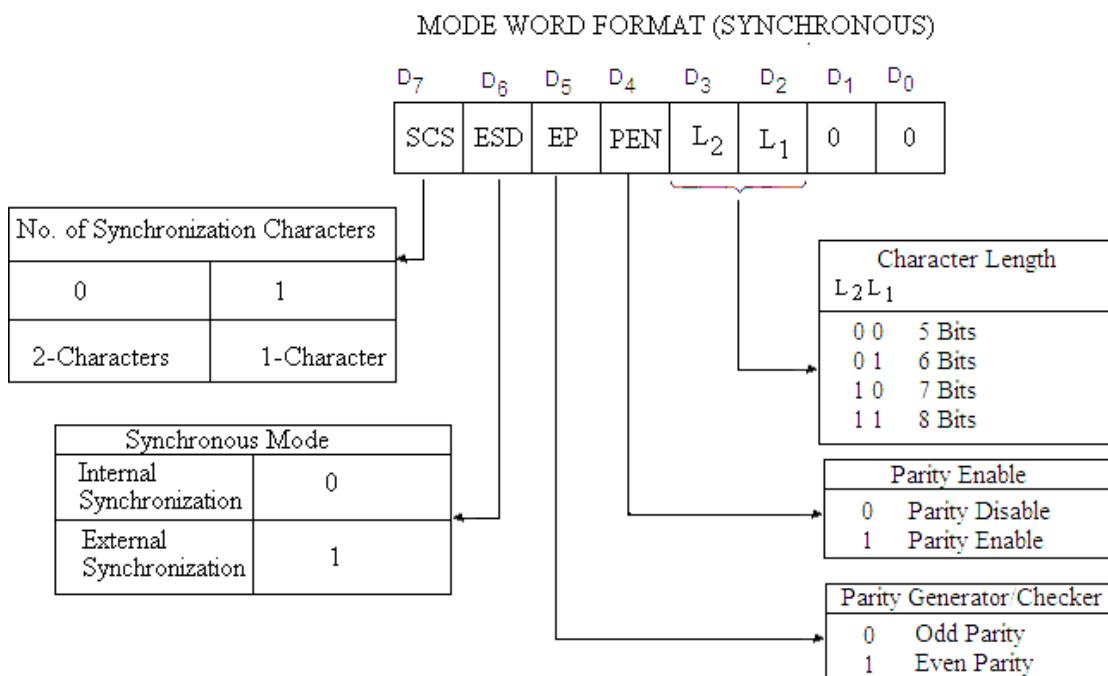
#### 14.8.2 Initialization of 8251A in Synchronous Mode

When programming the 8251A in asynchronous mode, following sequence must be followed:

1. Reset (either internal or external)
2. Mode Instruction (specify the Synchronous Mode and number of synchronizing characters)

3. One or two synchronizing character
4. Command Instruction

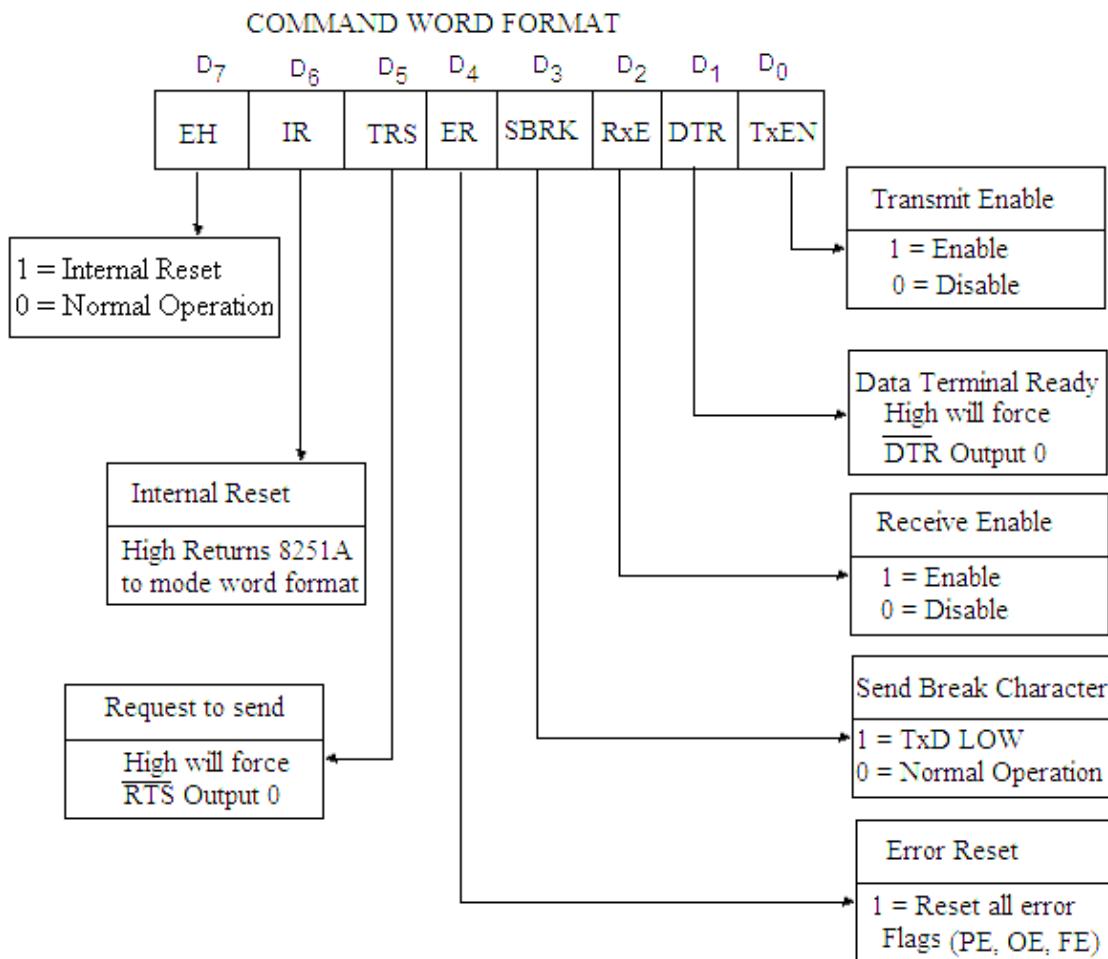
For initializing of the 8251A in synchronous mode the mode instruction word is to be written only after resetting the PCI as in the case of asynchronous mode. Then with one or two sync characters, a command word is written in the control register. The mode word format in synchronous operation is shown in figure 14.20. As discussed earlier the D<sub>1</sub> and D<sub>0</sub> bits program the baud rate factor which are set 00 for synchronous operation.



**Fig. 14.20**

The format to be used for command instruction format for the synchronous operation is the same as for the asynchronous operation which is shown in figure 14.21. Once the functional definition of the 8251A has been programmed by the mode instruction and the Sync characters are loaded (in synchronous mode) then the device is ready to be used for data communication. The command instruction controls the actual operation of the selected format. Functions such as Enable Transmit/ Receive, Error Reset etc are provided by the command instruction.

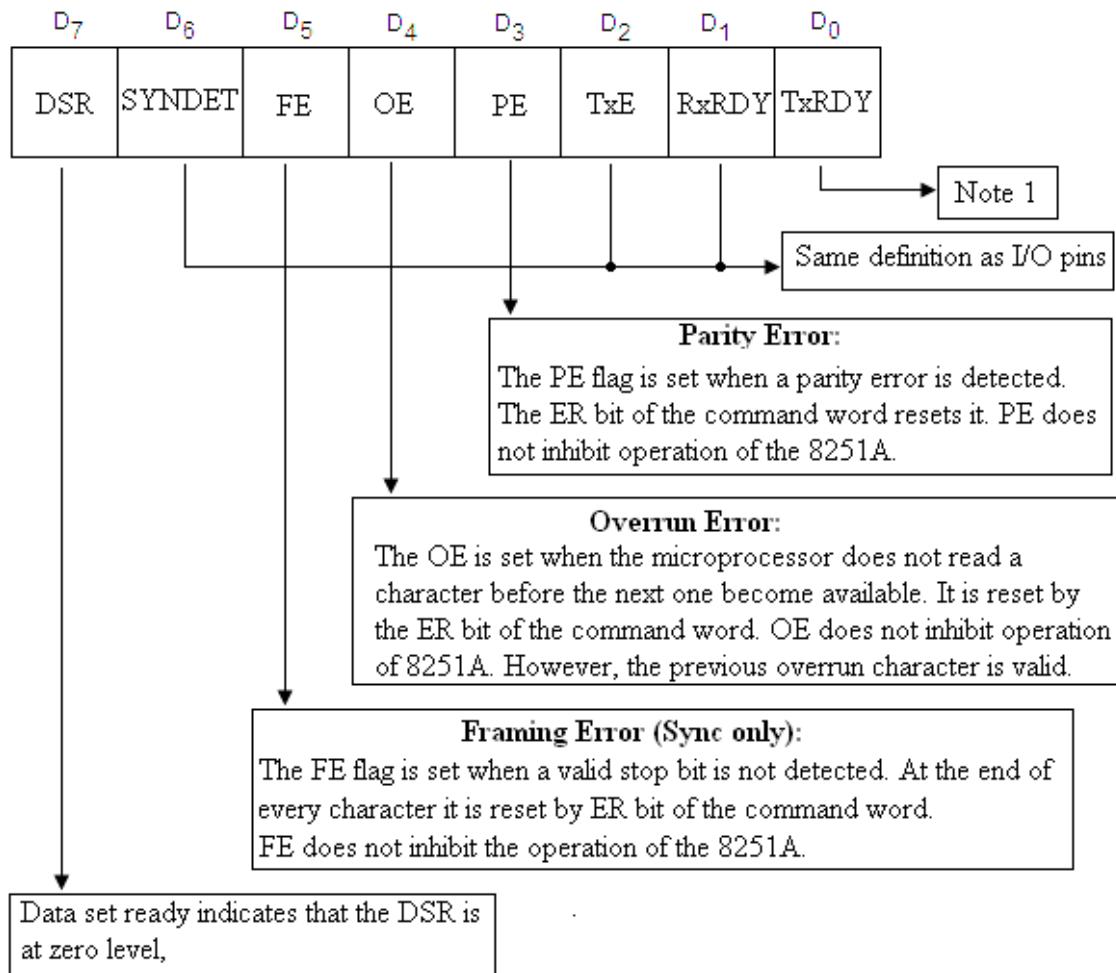
Once the mode instruction has been written into the 8251A and Sync characters inserted, then all further “control writes” ( $C / \bar{D} = 1$ ) will load a command instruction. A reset operation (internal or external) will return the 8251A to the mode instruction format.



**Fig. 14.21**

In data communication system, it is necessary to examine the status of the active device to ascertain if errors have occurred or other conditions that require the processor's attention. The 8251A has facilities that allow the programmer to read the status of device at any time during functional operation. A normal read command is issued by the CPU with  $C / \bar{D} = 1$  to accomplish the function.

The microprocessor can output the data to be transmitted only after checking the status TxRDY (D<sub>0</sub>) bit of status word as shown in figure 14.22. The sync characters will automatically be inserted if the buffer of the transmitter becomes empty at any time.



**Note:** TxDY status bit has different meaning from TxRDY status pin. The former is not conditioned by CTS and TxEN; the latter is conditioned by both CTS and TxEN.

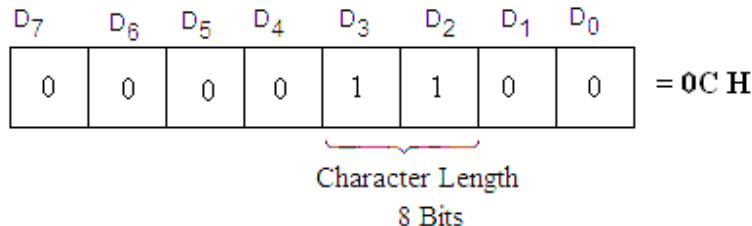
|                 |   |  |
|-----------------|---|--|
| TxDY status bit | : | DB buffer empty                        |
| TxDY pin out    | : | DB buffer empty. (CTS = 0). (TxEN = 1) |

**Fig. 14.22**

**Example 14.1** Write a subroutine program to transmit serially 256 bytes of data stored in the memory locations starting at 3000 H. The bytes are to be transmitted in synchronous mode (without parity) with two sync characters using 8251A. Consider 90 H and 91 H are the port addresses for control/status register and data register respectively.

**Solution.** The mode word format for the given problem is shown in figure 14.23.

### MODE WORD FORMAT (SYNCHRONOUS)



**Fig. 14.23**

The subroutine program may then be given as:

### SUBROUTINE PROGRAM

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>   |
|--------------|------------------|----------------|---|
| STATUS       | LXI H,           | 3000 H         | ; Initialize H-L register pair for index pointer.                             |
|              | MVI C,           | FF H           | ; Register C to be used as counter.   |
|              | MVI A,           | 00 H           | ; Dummy word  |
|              | OUT              | 91 H           |   |
|              | MVI A,           | 40 H           | ; Command word for resetting 8251A.   |
|              | OUT              | 91 H           | ; Command word loaded to the control register.                                |
|              | MVI A,           | 0C H           | ; Mode word 0C H is loaded to the accumulator.                                |
|              | OUT              | 91 H           | ; The mode word for two sync character is loaded to the control register.     |
|              | IN               | 90 H           | ; Read the status word.   |
|              | ANI              | 01 H           | ; Mask all the bits except $D_0$ for checking the status.                     |
|              | JZ               | STATUS         | ; If $D_0$ bit is zero jump to STATUS to read again.                          |
|              | MOV A,           | M              | ; Loads the character to be transmitted to the accumulator.                   |
|              | OUT              | 91 H           | ; Character is transmitted.   |
|              | DCR              | C              | ; Decrements the data in C register for next byte for transmission.           |
|              | JNZ              | STATUS         | ; If all bytes are not transmitted then jump to STATUS to read the next byte. |
|              | RET              |                | ; Returns to main program.  |

## **PROBLEMS**

1. What are the advantages and disadvantages of serial I/O data transfer over the parallel I/O data transfer?
  2. Name three methods of serial mode of data transfer and explain them in brief.
  3. Draw and explain the block diagram of Programmable Communication Interface 8251A.
  4. Discuss the Transmission section of 8251A.
  5. Discuss the Receiver section of 8251A.
  6. Explain the working principle of RS 232 interface.
  7. Explain how Programmable Communication Interface 8251A is interface with the CPU.
  8. Draw the schematic diagram of interfacing 8251A with 8085 microprocessor. The connections should be such that the port address for control register and data registers are 71 H and 70 H respectively.
  9. Explain the mode word format of 8251A.
  10. Explain the command word format of 8251A.
  11. Explain how the 8251A is initialized in Asynchronous mode.
  12. Explain how the 8251A is initialized in synchronous mode.
  13. Explain the following terms related to 8251A:  
 $C/D$ ,  $\overline{RxC}$ ,  $RxD$ ,  $\overline{DSR}$ ,  $\overline{TxC}$ ,  $TxD$ ,  $\overline{DTR}$ ,  $\overline{CTS}$
  14. Explain the following terms related to 8251A:  
 $RxRDY$ ,       $TxRDY$ ,       $TxEMPTY$
-

# 15

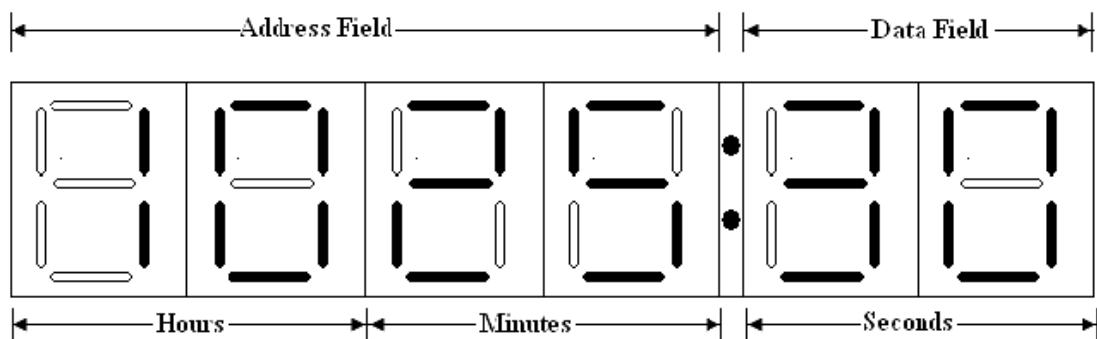
## Applications of Microprocessor

---

After the study of all the details of microprocessor 8085, including the assembly language programming and peripheral devices we are now in a position to design some microprocessor based systems. Real time clock with on/off timer, running light, washing machine control, water level control etc. are some designs will be discussed in this chapter. These designs are supposed to be very interesting and useful. The assembly language programming of these designs have been verified on M/S Vinytis Kit

### 15.1 REAL TIME CLOCK WITH ON/OFF TIMER

In the present section of this chapter, the details of the assembly language programming of the design of Real Time Clock with On/Off Timer will be discussed. In this microprocessor based design six FNDs of the microprocessor kit (four of address field and two of data field) have been used to display the current time. The four 7-segments of the address field would display hours and minutes of the current time and two 7-segments of data field would display the seconds of the current time. For example, current time is 10:25:30, the address and data field of the kit would display as shown in Fig. 15.1. It would then continuously update the current time after every second. In other words, it will work as the real time clock (digital clock).



**Fig. 15.1**

For the working of real time clock, the software prepared was checked on the Vinytis kit (VMC-8506) and found to work satisfactorily. Two monitor programs (stored in Kit's ROM /EPROM) at the locations 0347 H (for clearing the display) and 05D0 H (displaying the contents of memory of memory locations 2050 H through 2055 H in the address and data fields respectively) have been used in the program. Please note that before calling the display routine, registers A and B are required to be initialized with either 00 or 01 to indicate to the monitor program as to where the contents of above mentioned memory locations are to be displayed (e.g. address field or data field) and whether a dot is to be displayed at the end of address field or not. The assembly language program was executed to switch ON an electrical appliance after a preset time period. This program was proved to be very useful microprocessor based system. This type of microprocessor based system has the useful requirement in research laboratories. An electronic circuit was also designed which was interfaced with the 8085A microprocessor through the programmable peripheral Interface (PPI) 8255A. The PPI-8255A was available with the microprocessor kit.

## **ASSEMBLY LANGUAGE PROGRAM**

The assembly language program for the universal timer and for switching ON electrical device is given below:

### **PROGRAM**

#### **Main Program**

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>  |
|--------------|------------------|----------------|--|
|              | LXI SP,          | 27FF H         | ;Initialize Stack Pointer.                                       |
|              | MVI A,           | 80H            | ;Control word for 8255-I.  |
|              | OUT              | 03H            | ;Work all the ports of 8255-I as output port .                   |
|              | CALL             | 0347 H         | ;Clears the display. It is the program stored in ROM of the kit. |
| AA           | XRA              | A              | ;Clears the accumulator  |
|              | MOV B,           | A              | ;Clears the B register also.                                     |

|    |                            |                        |   |
|----|----------------------------|------------------------|---|
|    | LXI H,                     | 2050 H                 | ;Initialize H-L pair where the current time is stored.  |
|    | CALL                       | 05D0 H                 | ;Displays the current time in address field. It is the program stored in ROM of the Kit.  |
|    | MVI A,<br>MVI B,<br>LXI H, | 01 H<br>00 H<br>2054 H | ;Stores 01 H in accumulator.<br>;Stores 00 H in B register.<br>;Initialize H-L register pair with 2054 H.                                       |
|    | CALL                       | 05D0 H                 | ;Displays the current time in data field.   |
|    | CALL                       | ONOFF                  | ;Calls subroutine program to switch on the electrical appliance.  |
|    | LXI H,                     | 2055 H                 | ;Initialize H-L register pair with 2055 H.  |
|    | MOV A,                     | M                      | ;Moves the least significant digit (LSD) of current seconds to the accumulator.   |
|    | ADI                        | 01 H                   | ;Add 01 to the accumulator.   |
|    | CPI                        | 0A H                   | ;Compares with 0A H.  |
|    | JZ                         | RR                     | ;If Acc.=0A H, then jump to RR, indicating that LSD of seconds has become 9. Else goes to next statement.                                       |
|    | MOV M,                     | A                      | ;Moves the accumulator contents to memory location addressed by H-L register pair i.e. current seconds are stored back in the memory locations. |
| DD | MVI B,                     | 02 H                   | ;Stores 02 H in accumulator.  |
| YY | LXI D,                     | FA00 H                 | ;Intialize D-E register pair with FA00H.  |
|    | CALL                       | DELAY                  | ;Calls the delay program.   |
|    | DCR                        | B                      | ;Decrement B- register.   |
|    | JNZ                        | YY                     | ;If B≠0 then jump to YY.  |
|    | JMP                        | AA                     | ;Jump to AA for the display of current time.  |
| RR | MVI A,                     | 00 H                   | ;LSD of current time becomes 0, which is stored in accumulator.   |
|    | MOV M,                     | A                      | ;Stores it to the memory location of LSD of seconds   |

|    |        |      |  |
|----|--------|------|--|
|    |        |      | whose address is given in H-L register pair.   |
|    | DCX    | H    | ;Decrement H-L register pair.  |
|    | MOV A, | M    | ;Moves MSD of the current second to accumulator.   |
|    | ADI    | 01H  | ;Add 01 H to it  |
|    | CPI    | 06 H | ;Compaers if MSD of the current seconds is 06.   |
|    | JZ     | UU   | ;If Acc.=06 then jump to UU, indicating that MSD of seconds has become 6. Else goes to next statement. |
|    | MOV M, | A    | ;If A≠06 then stores to the memory location addressed by the H-L register pair.                        |
| UU | JMP    | DD   | ;Jump to DD for 1 sec.delay.   |
|    | MVI A  | 00 H | ;Store 00 to accumulator.  |
|    | MOV M, | A    | ;Stores 00 to the memory location of MSD of seconds addressed by the H-L register pair.                |
|    | DCX    | H    | ;Decrement H-L register pair.  |
|    | MOV A, | M    | ;Moves the LSD of the current minutes to accumulator.  |
|    | ADI    | 01 H | ;Add 01 H to the LSD of the current minutes.   |
|    | CPI    | 0A H | ;Compares it with 0A H.  |
|    | JZ     | VV   | ;If LSD of minutes is 0A H, then jump to VV else to the next instruction.                              |
|    | MOV M, | A    | ;Stores it to the memory location of LSD of minutes addressed by H-L register pair.                    |
| VV | JMP    | DD   | ;Jump to DD for 1 sec delay.   |
|    | MVI A, | 00 H | ;Stores 00 H to accumulator.   |
|    | MOV M, | A    | ;Stores 00 to the memory location of LSD of minutes addressed by H-L register pair.                    |
|    | DCX    | H    | ;Decrement H-L register pair.  |

|    |        |      |  |
|----|--------|------|--|
|    | MOV A, | M    | ;Moves the MSD of the current minutes to accumulator.  |
|    | ADI    | 01 H | ;Add 01 H to the MSD of the current minutes.   |
|    | CPI    | 06 H | ;Compaers if MSD of the current minutes is 06.   |
|    | JZ     | SS   | ;If Acc.=06 then jump to SS, indicating that MSD of minutes has become 6. Else goes to next statement. |
|    | MOV M, | A    | ;If Acc≠06 then stores to the memory location addressed by the H-L register pair.                      |
|    | JMP    | DD   | ;Jump to DD for delay of 1 sec.  |
| SS | MVI A  | 00 H | ;Store 00 to accumulator.  |
|    | MOV M, | A    | ;Stores 00 to the memory location of MSD of minutes addressed by the H-L register pair.                |
|    | DCX    | H    | ;Decrement H-L register pair.  |
|    | MOV A, | M    | ;Moves the LSD of the current hrs. to accumulator.   |
|    | ADI    | 01 H | ;Add 01 H to the LSD of the current hrs.   |
|    | * CPI  | 03 H | ;Compares it with 03 H.  |
|    | JZ     | LL   | ;If LSD of hrs is 03 H, then jump to LLelse to the next instruction.                                   |
|    | MOV M, | A    | ;Stores it to the memory location of LSD of hrs. addressed by H-L register pair.                       |
|    | CPI    | 0A H | ;It is also compared by 0AH.   |
|    | JZ     | WW H | ;If it is 0A H, then jump to EE else go to next instruction.   |
|    | MOV M, | A    | ;Stores it to memory locations addressed by H-L register pair.   |
|    | JMP    | DD   | ;Jump to DD for delay of 1sec.   |
| LL | DCX    | H    | ;Decrement H-L register pair.  |

|     |        |      |   |
|-----|--------|------|---|
|     | MOV A, | M    | ;Moves the contents of memory location addressed by H-L register pair to Acc. |
| **  | CPI    | 01 H | ;Compares it with 01 H.   |
|     | JZ     | GG   | ;If it is 01 H, then jump to GG H.  |
|     | INX    | H    | ;Increment the H-L register pair.   |
|     | MOV A, | M    | ;Moves the contents of M <sub>H-L</sub> to the Acc.                           |
|     | ADI    | 01 H | ;Add 01 H to it.  |
|     | CPI    | 0A H | ;Compares it with 0A H.   |
|     | JZ     | WW   | ;If it is 0A H then jump to WW.   |
|     | MOV M, | A    | ;Else stores the accumulator contents in M <sub>H-L</sub> .                   |
|     | JMP    | DD   | ;Jumps to DD for the delay of 1sec.   |
| GG  | MVI A, | 00 H | ;Stores 00 H to Acc.  |
|     | MOV M, | A    | ;Moves 00 H to the memory location addressed by H-L register pair.            |
|     | INX    | H    | ;Increments the H-L register pair.  |
| *** | MVI A, | 01 H | ;Loads Acc. to 01 H.  |
|     | MOV M, | A    | ;Moves the Acc. Contents to M <sub>H-L</sub> .                                |
|     | JMP    | DD   | ;Jumps to DD for the delay of 1sec.   |
| WW  | MVI A, | 00 H | ;Stores 00 H to accumulator.  |
|     | MOV M, | A    | ;Stores it to M <sub>H-L</sub> .  |
|     | DCX    | H    | ;Decrements the H-L register pair.  |
|     | MOV A, | M    | ;Moves M <sub>H-L</sub> to accumulator.                                       |
|     | ADI    | 01 H | ;Add 01 H to it.  |
|     | MOV M, | A    | ;Acc. Contents are loaded to M <sub>H-L</sub> .                               |
|     | JMP    | DD   | ;Jump for delay of 1sec.  |

### Delay Subroutine Program

| Label | Mnemonics | Operand | Comments                      |
|-------|-----------|---------|-------------------------------|
| DELAY | DCX       | D       | ;Decrement D-E register pair. |

|        |       |  |
|--------|-------|--|
| MOV A, | D     | ;Moves the contents stored in M <sub>D-E</sub> to Acc.               |
| ORA    | E     | ;The contents of A and E registers are ORed bit by bit.              |
| JNZ    | DELAY | ;If the result is not zero then jump to DELAY else next instruction. |
| RET    |       | ;Return to main program.   |

### Subroutine Program to check the time (ON time) after every second

| Label | Mnemonics | Operand | Comments   |
|-------|-----------|---------|--|
| ONOFF | LXI H,    | 2055 H  | ;Initialize H-L register pair with 2055 H.   |
|       | LXI D,    | 205C H  | ;Initialize D-E register pair with 205C H.   |
|       | MVI B,    | 06 H    | ;Stores 06 H to B-register.  |
|       | LDAX D    |         | ;Loads the contents of M <sub>D-E</sub> to the accumulator.                                |
|       | CMP       | M       | ;Compares it with M <sub>H-L</sub> .   |
|       | RNZ       |         | ;Return if not zero.   |
|       | DCX       | H       | ;Decrements H-L register pair.   |
|       | DCX       | D       | ;Decrements D-E register pair.   |
|       | DCR       | B       | ;Decrements the contents of B-register.  |
|       | JNZ       | AGAIN   | ;Jump if not zero to AGAIN.  |
| AGAIN | MVI A,    | 01 H    | ;Stores 01 H to Acc.   |
|       | OUT       | 00 H    | :D0 bit of port A of 8255 becomes high to energies the relay (to switch on the appliance). |
|       | RET       |         | ;Returns to main program.  |

#### **DATA**

|        |   |                |   |
|--------|---|----------------|---|
| 2050 H | - | MSD of hrs.    | <br>Current Time |
| 2051 H | - | LSD of hrs.    |   |
| 2052 H | - | MSD of Minutes |   |
| 2053 H | - | LSD of Minutes |   |
| 2054 H | - | MSD of Seconds |   |
| 2055 H | - | LSD of Seconds |   |

|        |   |   |
|--------|---|---|
| 2056 H | - | 01H (if wish to switch ON the electrical appliance) |
| 2057 H | - | MSD of Hrs.   |
| 2058 H | - | LSD of Hrs.   |
| 2059 H | - | MSD of Minutes                                      |
| 205A H | - | LSD of Minutes                                      |
| 205B H | - | MSD of Seconds                                      |
| 205C H | - | LSD pf Seconds                                      |

Time For On Electrical  
Appliance

This program written in assembly language is self-explanatory. The comment column will help in understanding the operation of the project. The current time is stored in the beginning before the start of the program in the memory locations starting at 2050 H to 2055 H. In the memory location 2056 H, 01H is stored if we wish to switch on the electrical device connected with the PPI-8255 A; otherwise any number can be stored in this memory location. If 01H is stored in the memory location 2056H, then the time for switching ON the electrical appliance is stored in the memory location starting at 2057 H to 205C H. Suppose current time, when we wish to start the program is:

|      |      |      |
|------|------|------|
| Hrs. | Min. | Sec. |
| 02   | 25   | 30   |

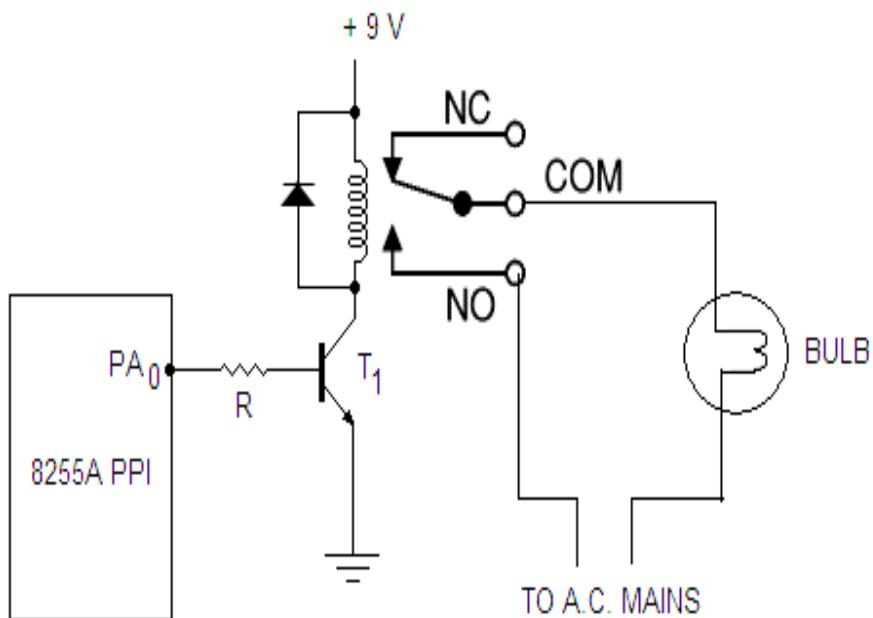
And we wish to switch on the electrical appliance at:

|      |      |      |
|------|------|------|
| Hrs. | Min. | Sec. |
| 04   | 23   | 00   |

The data to be stored in the memory locations starting at 2050 to 205C is given below:

| Memory Location | Data | Current Time                   | Time when electrical appliance to ON |
|-----------------|------|--------------------------------|--------------------------------------|
| 2050 H          | 00 H | 02 Hrs.                        |                                      |
| 2051 H          | 02 H |                                |                                      |
| 2052 H          | 02 H | 25 Minutes                     |                                      |
| 2053 H          | 05 H |                                |                                      |
| 2054 H          | 03 H | 30 Seconds                     |                                      |
| 2055 H          | 00 H |                                |                                      |
| 2056 H          | 01 H | 01 for switch ON the appliance |                                      |
| 2057 H          | 00 H | 04 Hrs.                        |                                      |
| 2058 H          | 04 H |                                |                                      |
| 2059 H          | 02 H | 23 Minutes                     |                                      |
| 205A H          | 03 H |                                |                                      |
| 205B H          | 00 H | 00 Seconds                     |                                      |
| 205C H          | 00 H |                                |                                      |

A relay circuit to be connected to the Programmable Peripheral Interface IC 8255-I is shown in Fig. 15.2.



**Fig.15.2**

This circuit consists of a relay, a diode and transistor. The base of the transistor is connected to PA<sub>0</sub> of 8255 although a 10 k resistance. A diode IN4007 is connected in parallel with coil of 12V relay. The electrical appliance is connected to the a.c.mains through the N/O pins of the relay.

The working of this circuit may be explained as:

When a high signal is available through the software to a PA<sub>0</sub> (D<sub>0</sub> bit of port A) of 8255-I, the transistor goes into saturation. The relay is energized. The N/O terminals of the relay get connected and electrical appliance become ON.

When the program is executed, the current time is displayed in the address and data fields of the microprocessor kit. The current time is updated after every second by the programming. For this LSD (Least significant Digit) of the seconds is transferred to accumulator, which is added with 1. The contents after addition are compared with 0A H (decimal number 10). In other words LSD of the second is continuously increased (after every one second) by 1 to reach to 0A H. When it reaches to 0A H then program will jump to MSD (Most Significant Digit) of the seconds after storing 00H in the memory location 2050 H.

Now MSD of the second is moved into accumulator and then increased by 1 which then compared with 06H.

When MSD of the second reaches to 06H, it will make 00H to its corresponding location. The program will then jump to LSD of the minutes. The program will execute in such a way the minutes will be checked after every one minute and it will be updated in the address field so that it displays up to 59H. After that, it becomes 00 H.

In the similar fashion, Hrs. will be checked after every hour so that it displays the updated current time.

To switch ON the electrical appliance at the predetermined time the subroutine program was developed. The subroutine program is executed after every second to check if the current time is equal to the time when the electrical appliance is to switched ON. When the current time is equal to the ON time (required time), 8255A sends a high signal to PA<sub>0</sub> (D<sub>0</sub> bit of Port A). The high signal energies the relay and appliance becomes ON. After switching ON the appliance the program will, however, continue displaying the updated time.

The observations were made to check the ON time and found to work fine. The working of the real time clock was also very accurate. It was proved to be satisfactory assembly language programming of the 8085A.

Further, to change the real time clock to display the timing in address and data field in 24 Hrs., i.e. after 12:59:59 it should display 13:00:00 onwards, rather than 01:00:00 onwards. So certain changes in the main program should be incorporated. The instructions marked by stars (\*, \*\*, \*\*\*) be replaced as given below:

- \* Replace the instruction CPI 03 H by CPI 05 H
- \*\* Replace the instruction CPI 01 H by CPI 02 H
- \*\*\* Replace the instruction MVI A, 01 H by MVI A, 00 H.

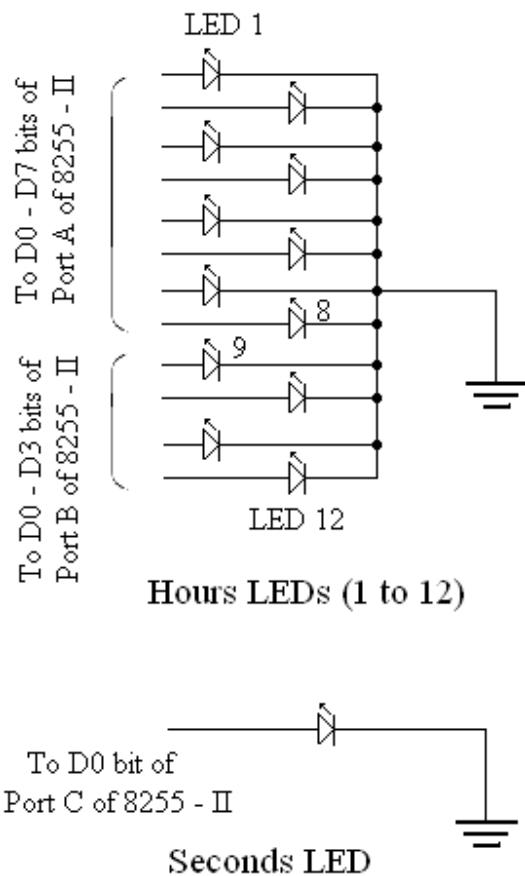
## 15.2 MICROPROCESSOR BASED LED DIAL CLOCK

Here the design of microprocessor based LED dial clock has been discussed which runs using microprocessor 8085. It works purely on the basis of the software; and some hardware has also been used. The software is prepared in the assembly language of 8085A microprocessor. This program was tested on Vinytis 8085  $\mu$ p kit and found to work very fine.

In this dial clock 73 LEDs are used which are interfaced with the different ports of two 8255 PPI ICs available on  $\mu$ p kit. Out of 73 LEDs, 12 LEDs of Amber color are used to show hours, 60 LEDs of red color are used to show minutes and one green LED alternately glows for half second showing that the seconds are continuously being counted by the clock. All these LEDs are connected in a circle in the form of a dial of a wall clock. The beauty of this project is the software part; and the students of Electronics discipline would definitely like to design this project or this idea may be used in other

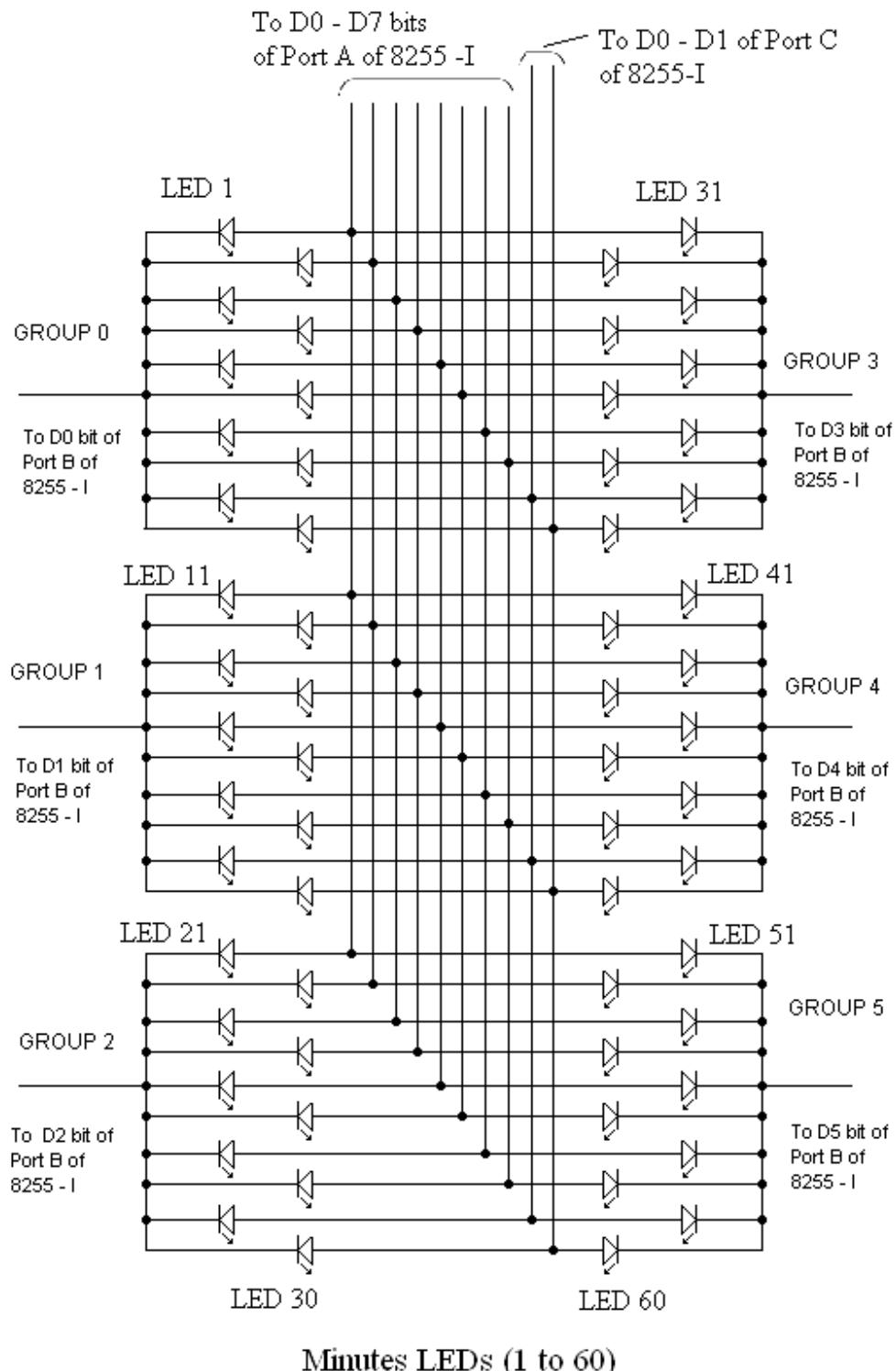
projects also. It is worthwhile to mention that no electronic circuit was used, except the LEDs are connected to the two PPI-8255 ICs available with the microprocessor kit.

Figures 15.3 and 15.4 show the circuit diagrams to be interfaced with the  $\mu$ -Kit. The cathodes of all the 12 Amber colored Hours LEDs are connected to the ground. The Anodes of these LEDs are connected to the Port A and Port B of 8255 -II, the anodes of LEDs showing Hrs 1 to 8 are connected respectively to D0 –D7 Bits of Port A of 8255-II and the anodes of LEDs showing Hours 9 to 12 are connected respectively to D0 – D3 bits of Port B of 8255 – II. Look – up table 15.1 provides Logic 1 to Hours LEDs 1-8. However, logic 1 to Hours LEDs 9 – 12 are provided using the software. The bit D0 of Port C lower of 8255 – II is connected to the anode of Green LED (showing second) and cathode of this LED is also grounded.



**Fig. 15.3**

The minutes red colored 60 LEDs are divided into 6 Groups (Group 0 to Group 5) and each group contains 10 LEDs. The cathodes of each group are connected together. The D0 – D5 bits of Port B of 8255 – I are connected to the cathodes (Common Points) of Group 0 to Group 5 LEDs respectively. The ground signal is provided to the cathode pin of each Group through the software using Look up table 15.2.



**Fig. 15.4**

The anodes of 10 LEDs (1 to 10) of each of 6 Groups are connected in parallel, i.e. LED No. 1 of each Group is connected together, and similarly LED No 2 of each

Group is connected together and so on. In this way 10 different Anode leads (1 to 10) are finally available to be connected to different pins of Port A and Port C of 8255 – I. D0 – D7 bits of Port A of 8255 – I are connected to 1 – 8 Anodes leads and 9 – 10 Anode leads are connected to D0 – D1 bits of Port C of 8255– I respectively. Look up table III provides Logic 1 to 1 – 8 Anode Leads; and to the 9 – 10 Anode leads the Logic 1, is provided by the software.

**Table – 15.1**

| <b>Memory Location</b> | <b>Data Stores</b> | <b>Binary Equivalent</b> | <b>Comments<br/>(Port A of 8255-II)</b> |
|------------------------|--------------------|--------------------------|---|
| 2101 H                 | 01H                | 0000 0001                | Provides Logic 1 to D0 bit (Hrs.1)      |
| 2102 H                 | 02H                | 0000 0010                | Provides Logic 1 to D1 bit (Hrs.2)      |
| 2103 H                 | 04H                | 0000 0100                | Provides Logic 1 to D2 bit (Hrs.3)      |
| 2104 H                 | 08H                | 0000 1000                | Provides Logic 1 to D3 bit (Hrs.4)      |
| 2105 H                 | 10H                | 0001 0000                | Provides Logic 1 to D4 bit (Hrs.5)      |
| 2106 H                 | 20H                | 0010 0000                | Provides Logic 1 to D5 bit (Hrs.6)      |
| 2107 H                 | 40H                | 0100 0000                | Provides Logic 1 to D6 bit (Hrs.7)      |
| 2108 H                 | 80H                | 1000 0000                | Provides Logic 1 to D7 bit (Hrs.8)      |

**Table – 15.2**

| <b>Memory Location</b> | <b>Data Stores</b> | <b>Binary Equivalent</b> | <b>Comments<br/>(Port B of 8255 - I)</b> |
|------------------------|--------------------|--------------------------|--|
| 2200 H                 | FEH                | 1111 1110                | Provides Logic 0 to D0 bit (Group 0)     |
| 2201 H                 | FDH                | 1111 1101                | Provides Logic 0 to D1 bit (Group 1)     |
| 2202 H                 | FBH                | 1111 1011                | Provides Logic 0 to D2 bit (Group 2)     |
| 2203 H                 | F7H                | 1111 0111                | Provides Logic 0 to D3 bit (Group 3)     |
| 2204 H                 | EFH                | 1110 1111                | Provides Logic 0 to D4 bit (Group 4)     |
| 2205 H                 | DFH                | 1101 1111                | Provides Logic 0 to D5 bit (Group 5)     |

**Table – 15.3**

| <b>Memory Location</b> | <b>Data Stores</b> | <b>Binary Equivalent</b> | <b>Comments<br/>(Port A of 8255 – I)</b> |
|------------------------|--------------------|--------------------------|--|
| 2300 H                 | 01H                | 00000001                 | Provides Logic 1 to D0 bit               |
| 2301 H                 | 02H                | 00000010                 | Provides Logic 1 to D1 bit               |
| 2302 H                 | 04H                | 00000100                 | Provides Logic 1 to D2 bit               |
| 2303 H                 | 08H                | 00001000                 | Provides Logic 1 to D3 bit               |
| 2304 H                 | 10H                | 00010000                 | Provides Logic 1 to D4 bit               |
| 2305 H                 | 20H                | 00100000                 | Provides Logic 1 to D5 bit               |
| 2306 H                 | 40H                | 01000000                 | Provides Logic 1 to D6 bit               |
| 2307 H                 | 80H                | 10000000                 | Provides Logic 1 to D7 bit               |

The software provides a signal continuously to Green LED so that it remains ON and OFF alternately for half a second. The minutes 60 LEDs glow one by one after every one minute. Port B of 8255 – I provides a low voltage (0 volt) to the cathodes of one block for 10 minutes and to the anode of these LEDs (1 to 10) are provided positive voltage (logic 1) alternately for one minute through D0 – D7 bits of Port A and D0 – D2 bits of Port B of 8255 – I. However, Hours' 12 Amber colored LEDs glow one by one after every an Hour time, as positive voltage (logic 1) is provided to the anodes of these LEDs after every an Hour. As discussed earlier positive voltage to Hours LEDs (1 to 8) are provided by D0 – D7 bits of Port A of 8255 – II and to Hours LEDs (9 to 12 ) are provided by D0 – D3 bits of Port B of 8255 – II after every an hour time. All the LEDs glow in a sequence as the minutes and hours hands move in an analog clock.

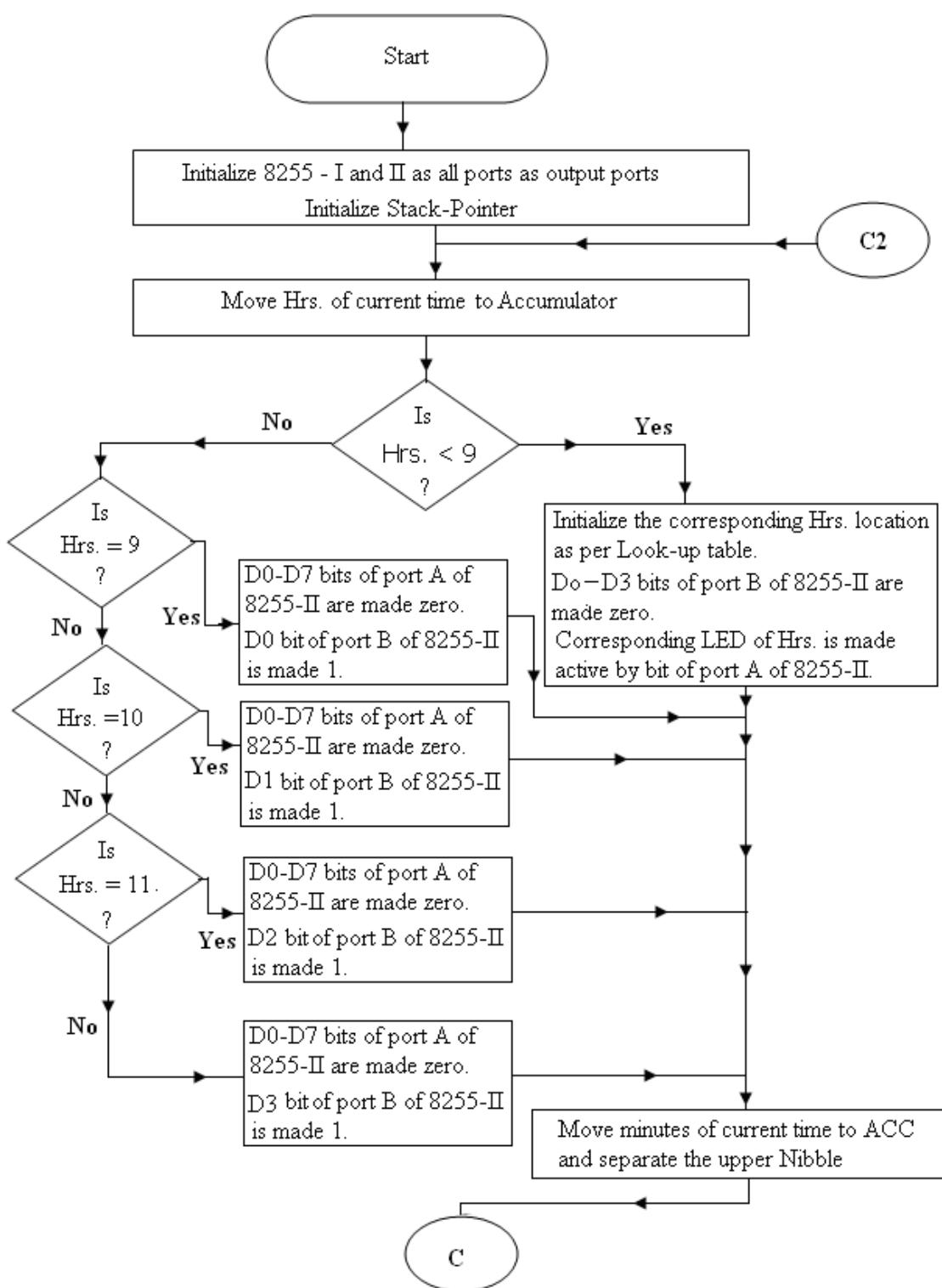
The current time is stored in the beginning in some memory locations. Hours, minutes and seconds of the current time are stored in the memory locations say 2000 H to 2002 H respectively. Figures 15.5 show the flow chart of the program. The software in the assembly language of 8085A for providing appropriate signal to different LEDs is given below:

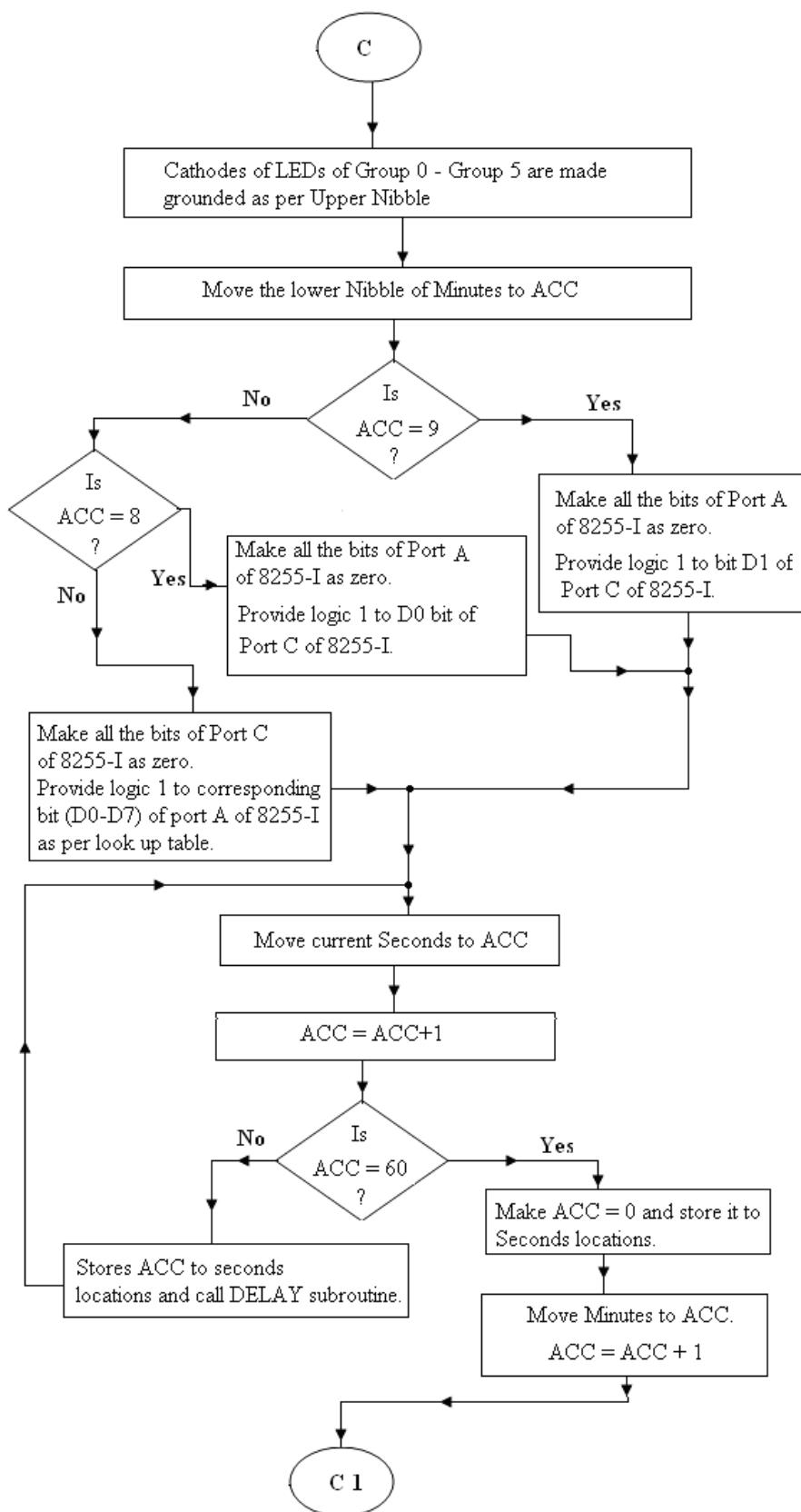
## **Main Program**

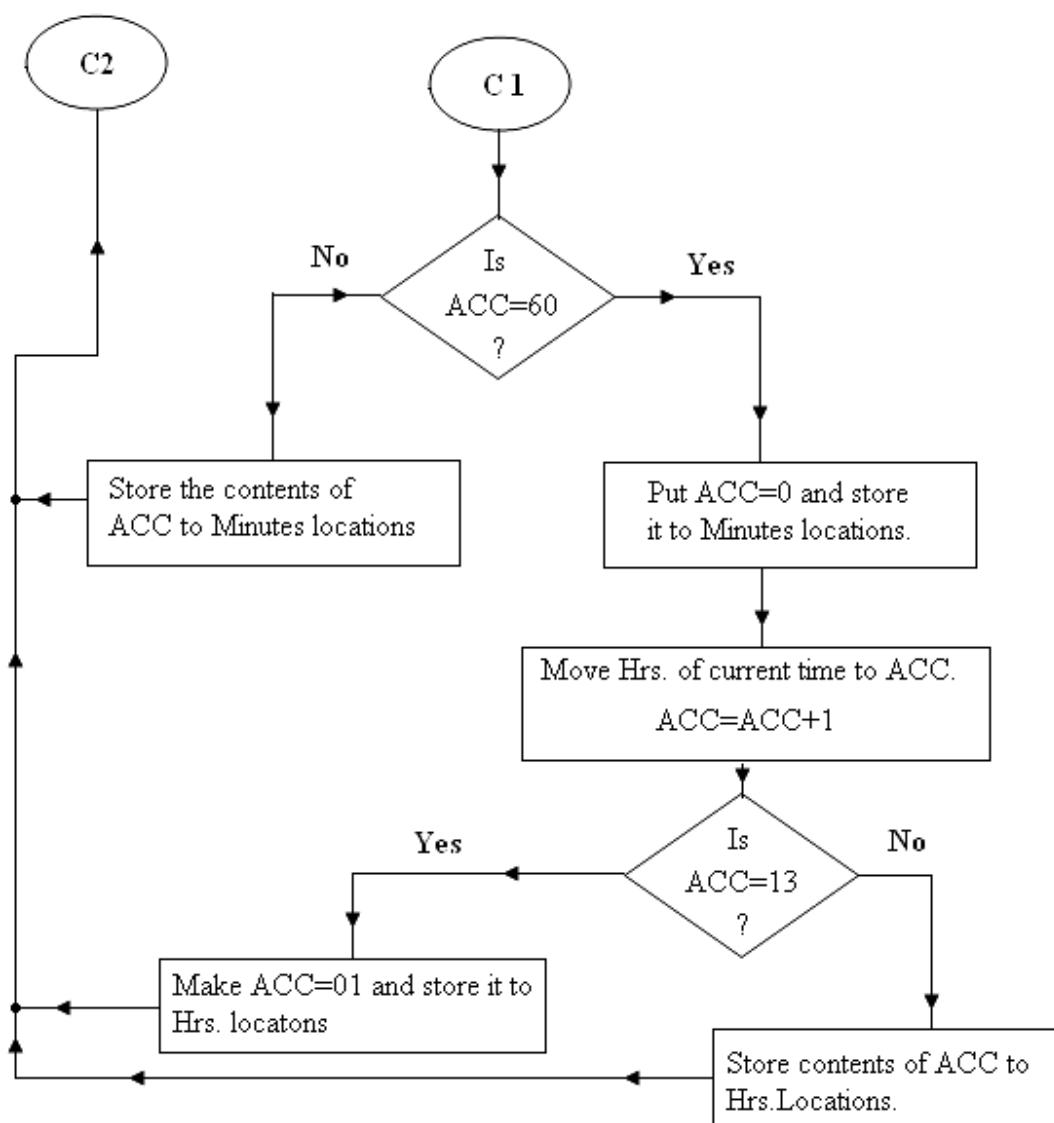
### **MEMORY LOCATIONS WHERE CURRENT TIME IS STORED**

|       |   |                 |
|-------|---|-----------------|
| 2000H | - | CURRENT HRS     |
| 2001H | - | CURRENT MINUTES |
| 2002H | - | CURRENT SECONDS |

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>  |
|--------------|------------------|----------------|--|
|              | LXI SP,          | XXXX H         | ;Initialize Stack Pointer.   |
|              | MVI A,           | 80H            | ;Control word for 8255-I.  |
|              | OUT              | 03H            | ;Works all the ports of 8255-I as output port                          |
|              | OUT              | 0B H           | ;Works all the ports of 8255-II as output port.                        |
| START        | LXI H,           | 2000 H         | ;Loads the H-L register pair with the address of Hrs. of current time. |
|              | MOV A,           | M              | ;Moves the Hrs. of current time to accumulator.                        |
|              | CPI              | 09 H           | ;It is compared with 09 H.   |







**Fig. 15.5**

|        |      |   |
|--------|------|---|
| JC     | PT0  | ;If ACC<9, then jump to PT0.                        |
| JZ     | PT1  | ;If ACC=9, then jump to PT1.                        |
| CPI    | 10 H | ;Accumulator contents are again compared with 10 H. |
| JZ     | PT2  | ;If ACC=10 then jump to PT2.                        |
| CPI    | 11 H | ;If ACC=11 then jump to PT3.                        |
| MVI A, | 00 H |   |

|     |                       |                     |  |
|-----|-----------------------|---------------------|--|
|     | OUT                   | 08 H                | ;Provides logic 0 to all the bits of Port A of 8255-II.  |
|     | MVI A,<br>OUT         | 08 H<br>09 H        | ;Provides logic 1 to D3 bit of port B of 8255-II.<br>;Jumps to PT5.                                      |
| PT1 | JMP<br>MVI A,<br>OUT  | PT5<br>00 H<br>08 H | ;Provides logic 0 to all the bits of port A of 8255-II.  |
|     | MVI A,<br>OUT         | 01 H<br>09 H        | ;Provides logic 1 to D0 bit of port B of 8255-II.<br>;Jumps to PT5.                                      |
| PT2 | JUMP<br>MVI A,<br>OUT | PT5<br>00 H<br>08 H | ;Provides logic 0 to all the bits of port A of 8255-II.  |
|     | MVI A,<br>OUT         | 02 H<br>09 H        | ;Provides logic 1 to D1 bit of Port B of 8255-II.<br>;Jumps to PT5.                                      |
| PT3 | JMP<br>MVI A,<br>OUT  | PT5<br>00 H<br>08 H | ;Provides logic 0 to all the bits of port A of 8255-II.  |
|     | MVI A,<br>OUT         | 04 H<br>09 H        | ;Provides logic 1 to D2 bit of port B of 8255-II.<br>;Jumps to PT5.                                      |
| PT0 | JMP<br>MOV E,         | PT5<br>M            | ;Hrs. of current time is less than 9 which is loaded to E register.                                      |
|     | MVI A,<br>OUT         | 00 H<br>09 H        | ;Provides logic 0 to all the bits of port B of 8255-II.  |
|     | MVI D,<br>LDAX D      | 21 H                | ;Loads the Acc with the data as per the table 15.1 in respect of the current time, which is less than 9. |
|     | OUT                   | 08 H                | ;Provides logic w to one of the bits of port A of 8255-II as per the look up table 15.1.                 |
| PT5 | INX H                 |                     | ;H-L register pair will indicate the minutes of the current time.  |

|      |               |      |   |
|------|---------------|------|---|
|      | MOV A,        | M    | ;Moves current minutes to ACC.  |
|      | ANI           | F0 H | ;Make lower Nibble of ACC to zero.  |
| LOOP | MVI B,<br>RLC | 04 H | ;Rotate ACC contents four times to shift higher Nibble of ACC to lower Nibble and lower which are zero is shifted to higher Nibble. |
|      | DCR           | B    |   |
|      | JNZ           | LOOP |   |
|      | MOV E,        | A    | ;ACC contents are loaded to E register.   |
|      | INR           | D    |   |
|      | LDAX D        |      |   |
|      | OUT           | 01 H | ;Provides logic 0 to the cathodes of one group from group 0 to group 5 as per the look up table 15.2.                               |
|      | MOV A,        | M    |   |
|      | ANI           | 0F H | ;Provides logic 0 to higher Nibble of ACC.  |
|      | CPI           | 09 H | ;Compared with 09.  |
|      | JZ            | AK   | ;If ACC=09 then jump to AK.   |
|      | CPI           | 08 H | ;Compared with 08.  |
|      | JZ            | AK1  | ;If ACC=08 then jump to AK1.  |
|      | MVI A.        | 00 H |   |
|      | OUT           | 02 H | ;Provides logic 0 to all the bits of Port C of 8255-I.  |
|      | MOV E,        | A    |   |
|      | INR           | D    |   |
|      | LDAX D        |      | ;Provides logic 1 to one of the bits of D0-D7 of port A of 8255-I as per the look-up table 15.3.                                    |
|      | OUT           | 00 H |   |
|      | JMP           | AK2  | ;Jump to AK2.   |
| AK1  | MVI A,        | 00 H |   |
|      | OUT           | 00 H | ;Provides logic 0 to all the bits of Port A of 8255-I.  |
|      | MVI A,        | 01 H |   |
|      | OUT           | 02 H | ;Provides logic 1 to D0 bit of port C of 8255-I.  |

|     |        |       |  |
|-----|--------|-------|--|
|     | JMP    | AK2   | ;Jump to AK2.  |
| AK  | MVI A, | 00 H  |  |
|     | OUT    | 00 H  | ;Provides logic 0 to all the bits of Port A of 8255-I.           |
|     | MVI A, | 02 H  |  |
|     | OUT    | 02 H  | ;Provides logic 1 to D1 bit of port C of 8255-I.                 |
| AK2 | INX H  |       |  |
| PT7 | MOV A, | M     | ;Moves current seconds to ACC.                                   |
|     | ADI    | 01 H  | ;ACC=ACC+1   |
|     | DAA    |       | ;Decimal adjust the Accumulator.                                 |
|     | CPI    | 60 H  | ;Is ACC=60.  |
|     | JZ     | PT6   | ;If yes jump to PT6.   |
|     | MOV M, | A     | ;Else stores the current seconds to its corresponding locations. |
|     | CALL   | DELAY | ;Calls subroutine program for one second dealay.                 |
| PT6 | JMP    | PT7   | ;Jumps to PT7.   |
|     | MVI A, | 00 H  |  |
|     | MOV M, | A     | ;Stores 00 to Memory locations of current seconds.               |
|     | DCX H  |       |  |
|     | MOV A, | M     | ;Minutes of the current time is loaded to the accumulator.       |
|     | ADI    | 01 H  | ;ACC=ACC+1   |
|     | DAA    |       | ;Decimal adjust the Accumulator.                                 |
|     | CPI    | 60 H  | ;Is ACC=60.  |
|     | JZ     | PT8   | ;If yes jump to PT8.   |
|     | MOV M, | A     | ;Stores minutes to the Minutes locations.                        |
|     | JMP    | START | ;Jump to START to glow minutes and Hours.                        |
| PT8 | MVI A, | 00 H  |  |
|     | MOV M, | A     | ;Stores 00 to Memory locations of current minutes.               |
|     | DCX H  |       |  |
|     | MOV A, | M     | ;Hrs. of the current time is loaded to the accumulator.          |
|     | ADI    | 01 H  | ;ACC=ACC+1   |

|     |                  |           |  |
|-----|------------------|-----------|--|
|     | DAA              |           | ;Decimal adjust the Accumulator.                 |
|     | CPI              | 13 H      | ;Is ACC=13.                                      |
|     | JZ               | PT9       | ;If yes jump to PT9.                             |
|     | MOV M,           | A         | ;Stores Hrs. to the Hrs. locations.              |
|     | JMP              | START     | ;Jump to START to glow minutes and Hours.        |
| PT9 | MVI A,<br>MOV M, | 01 H<br>A | ;Stores Hrs.=01 in Hrs, locations instead of 13. |
|     | JMP              | START     | ;Jump to START to glow minutes and Hours.        |

### **Delay Subroutine Program**

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>   |
|--------------|------------------|----------------|---|
| DELAY        | MVI A,<br>OUT    | 01 H<br>0A H   | ;Seconds LED glows for half second.                                 |
| NXT1         | LXI D,<br>DCX    | FA00 H<br>D    | ;Decrement D-E register pair.                                       |
|              | MOV A,           | D              | ;Moves the contents stored in M <sub>D-E</sub> to Acc.              |
|              | ORA              | E              | ;The contents of A and E registers are ORed bit by bit.             |
|              | JNZ              | NXT1           | ;If the result is not zero then jump to nxt1 else next instruction. |
|              | MVI A,<br>OUT    | 00 H<br>0A H   | ;Seconds LED remains off for half second.                           |
| NXT2         | LXI D,<br>DCX    | FA00 H<br>D    | ;Decrement D-E register pair.                                       |
|              | MOV A,           | D              | ;Moves the contents stored in M <sub>D-E</sub> to Acc.              |

|     |      |   |
|-----|------|---|
| ORA | E    | ;The contents of A and E registers are ORed bit by bit.             |
| JNZ | NXT2 | ;If the result is not zero then jump to nxt1 else next instruction. |
| RET |      | ;Return to main program.  |

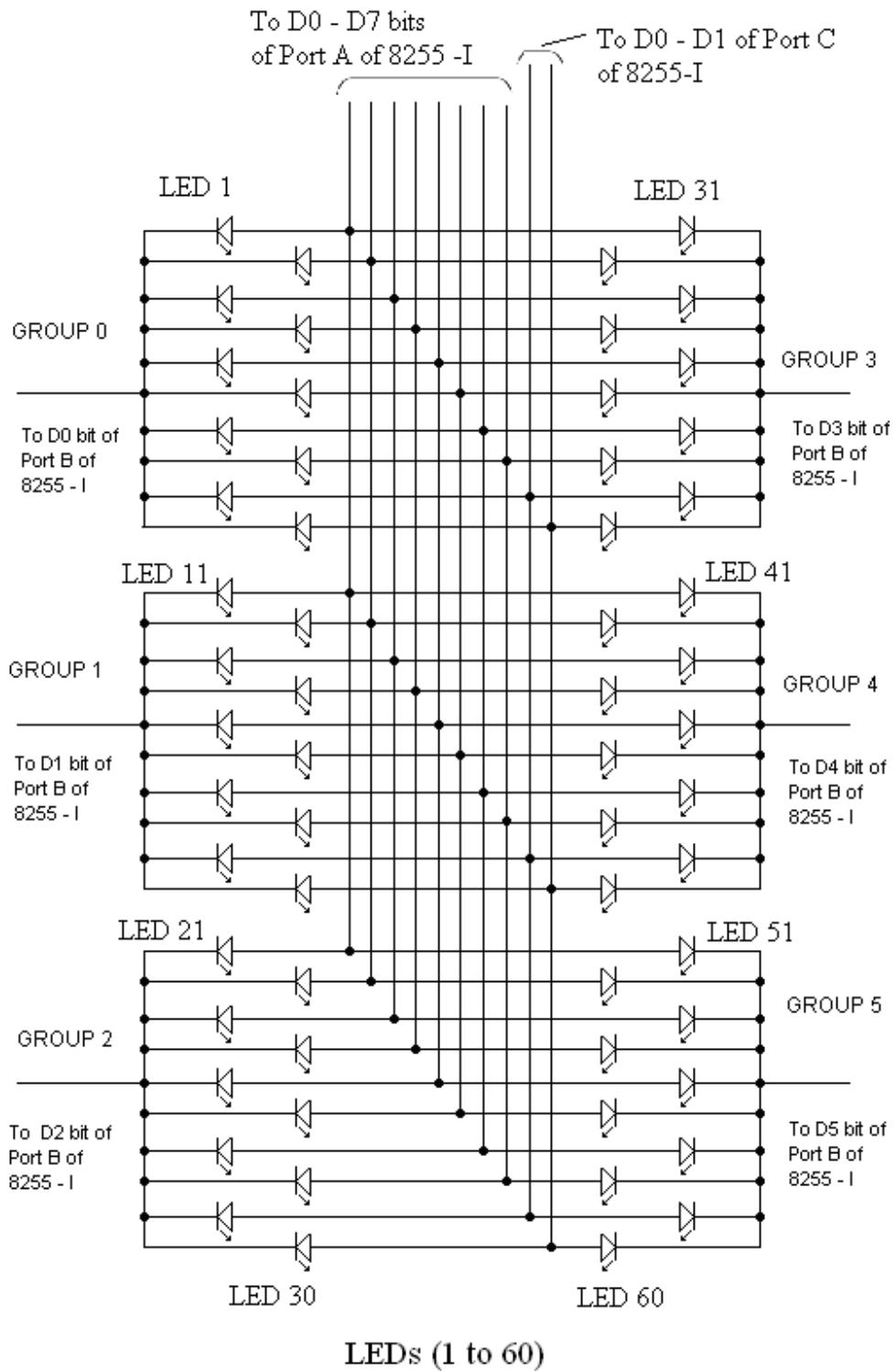
### 15.3 DESIGN OF MICROPROCESSOR BASED RUNNING LIGHT

The details discussed in the foregoing section of this chapter, may also be used to design the running light. In this project it is expected that 60 LEDs should glow in a sequence one after another in a preset time say one second. The LEDs are connected as shown in figure 15.6. The logic used in this project is the same as in discussed in the microprocessor based dial clock. The assembly language program is self explanatory which is given below. The look-up table for the same is given in table 15.4 and 15.5.

| Label | Mnemonics        | Operand        | Comments   |
|-------|------------------|----------------|--|
| START | MVI A,<br>OUT    | 80H<br>03H     | ;Control word for 8255-I.<br>;Works all the ports of 8255-I as output port |
|       | MVI A,<br>STA    | 00 H<br>2054 H | ;00 is loaded to accumulator.<br>;Store 00 to some location say 2054 H     |
|       | STA              | 2055 H         | and 2055 H   |
|       | LXI H,           | 2054 H         | ;Loads the H-L register pair with the address 2054 H.                      |
|       | MOV E,           | M              | ;Data of 2054 is loaded to the E register.                                 |
|       | MVI D,<br>LDAX D | 21 H           | ;Data of one memory location 2100 H to 2105 H is loaded to accumulator.    |
|       | OUT              | 01 H           | ;Cathode of one block (1 – 5) will be low.                                 |
|       | INX H            |                | ;H-L register pair will incremented.                                       |
|       | MOV A,           | M              | ;Moves the data in next location will be loaded to accumulator.            |
|       | CPI<br>JZ        | 09 H<br>AK     | ;Compared with 09.<br>;If ACC=09 then jump to AK.                          |
|       | CPI              | 08 H           | ;Compared with 08.   |

|      |        |        |  |
|------|--------|--------|--|
|      | JZ     | AK1    | ;If ACC=08 then jump to AK1.                                     |
|      | MVI A. | 00 H   |  |
|      | OUT    | 02 H   | ;Provides logic 0 to all the bits of Port C of 8255-I.           |
|      | MOV E, | M      |  |
|      | INR    | D      |  |
|      | LDAX D |        |  |
|      | OUT    | 00 H   |  |
|      | JMP    | AK2    | ;Jump to AK2.  |
| AK1  | MVI A, | 00 H   |  |
|      | OUT    | 00 H   | ;Provides logic 0 to all the bits of Port A of 8255-I.           |
|      | MVI A, | 02 H   |  |
|      | OUT    | 02 H   | ;Provides logic 1 to D1 bit of port C of 8255-I.                 |
|      | JMP    | AK2    | ;Jump to AK2.  |
| AK   | MVI A, | 00 H   |  |
|      | OUT    | 00 H   | ;Provides logic 0 to all the bits of Port A of 8255-I.           |
|      | MVI A, | 01 H   |  |
|      | OUT    | 02 H   | ;Provides logic 1 to D0 bit of port C of 8255-I.                 |
| AK2  | MOV A, | M      | ;Moves current value to ACC.                                     |
|      | ADI    | 01 H   | ;ACC=ACC+1   |
|      | CPI    | 0A H   | ;Is ACC=0A H.  |
|      | JZ     | RR     | ;If yes jump to RR.  |
|      | MOV M, | A      | ;Else stores the current seconds to its corresponding locations. |
| PP   | MVI B, | 02 H   | ;02 is loaded to B register for the loop.                        |
| YY   | LXI D, | FA00 H |  |
| NXT1 | DCX    | D      | ;Decrement D-E register pair.                                    |
|      | MOV A, | D      | ;Moves the contents stored in M <sub>D-E</sub> to Acc.           |
|      | ORA    | E      | ;The contents of A and E registers are ORed bit by bit.          |

|    |        |       |   |
|----|--------|-------|---|
|    | JNZ    | NXT1  | ;If the result is not zero then<br>jump to nxt1 else next<br>instruction. |
| RR | DCR    | B     | ;Decrement B register   |
|    | JNZ    | YY    |   |
|    | JMP    | START | ;Jumps to START.  |
|    | MVI A, | 00 H  |   |
|    | MOV M, | A     | ;Stores 00 to current<br>Memory locations.                                |
|    | DCX H  |       |   |
|    | MOV A, | M     | ;Current value is loaded to<br>the accumulator.                           |
| UU | ADI    | 01 H  | ;ACC=ACC+1  |
|    | CPI    | 06 H  | ;Is ACC=06.   |
|    | JZ     | UU    | ;If yes jump to UU.   |
|    | MOV M, | A     | ;Stores in memory location.   |
|    | JMP    | PP    | ;Jump to PP for delay.  |
|    | MVI A, | 00 H  |   |
|    | MOV M, | A     | ;Stores 00 to Memory<br>locations of current.                             |
|    | JMP    | PP    | ;Jump to PP for delay.  |



**Fig. 15.6**

**Table – 15.4**

| <b>Memory Location</b> | <b>Data Stores</b> | <b>Binary Equivalent</b> | <b>Comments<br/>(Port B of 8255 - I)</b> |
|------------------------|--------------------|--------------------------|--|
| 2100 H                 | FEH                | 1111 1110                | Provides Logic 0 to D0 bit (Group 0)     |
| 2101 H                 | FDH                | 1111 1101                | Provides Logic 0 to D1 bit (Group 1)     |
| 2102 H                 | FBH                | 1111 1011                | Provides Logic 0 to D2 bit (Group 2)     |
| 2103 H                 | F7H                | 1111 0111                | Provides Logic 0 to D3 bit (Group 3)     |
| 2104 H                 | EFH                | 1110 1111                | Provides Logic 0 to D4 bit (Group 4)     |
| 2105 H                 | DFH                | 1101 1111                | Provides Logic 0 to D5 bit (Group 5)     |

**Table – 15.5**

| <b>Memory Location</b> | <b>Data Stores</b> | <b>Binary Equivalent</b> | <b>Comments<br/>(Port A of 8255 – I)</b> |
|------------------------|--------------------|--------------------------|--|
| 2200 H                 | 01H                | 00000001                 | Provides Logic 1 to D0 bit               |
| 2201 H                 | 02H                | 00000010                 | Provides Logic 1 to D1 bit               |
| 2202 H                 | 04H                | 00000100                 | Provides Logic 1 to D2 bit               |
| 2203 H                 | 08H                | 00001000                 | Provides Logic 1 to D3 bit               |
| 2204 H                 | 10H                | 00010000                 | Provides Logic 1 to D4 bit               |
| 2205 H                 | 20H                | 00100000                 | Provides Logic 1 to D5 bit               |
| 2206 H                 | 40H                | 01000000                 | Provides Logic 1 to D6 bit               |
| 2207 H                 | 80H                | 10000000                 | Provides Logic 1 to D7 bit               |

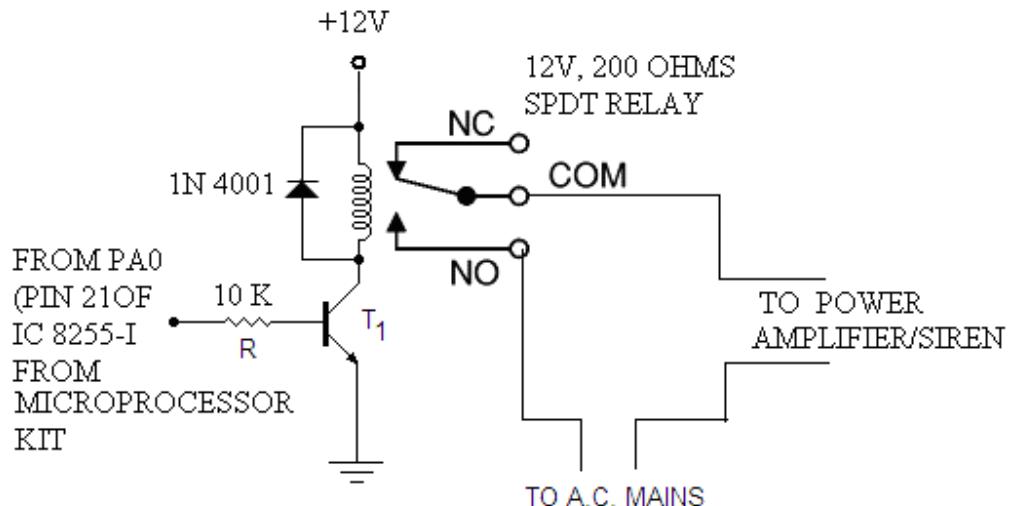
#### **15.4 MICROPROCESSOR BASED AUTOMATIC SCHOOL BELL SYSTEM**

This is an effective and useful project for educational institutes. In most of educational institutes, the peon rings the bell after every period (usually a period is of 40 minutes duration). The peon has to depend on his wrist watch or clock and sometimes he would even forget to ring the bell in time. In the present system, the human error has been eliminated. Every morning, when the school starts, someone has just to switch on the system and it will thereafter work automatically.

The automatic microprocessor based school bell system presented here has been tested on a Vinytis' microprocessor -8085 kit (VMC-8506). The kit displays the period number on two most significant digits of address field and minutes of the period elapsed on the next two digits of the address field. The data field of the kit displays seconds continuously.

The idea used here is very simple. The programmable peripheral interfacing (PPI) IC INTEL 8255-I available with the kit has been used. The bit 0 of port A (PA0) is connected to the base of a transistor through a resistance as shown in figure 15.7. It is used to energize the relay when PA0 pin of 8255-I is high. The siren, hooter or any bell voice system with an audio amplifier of proper wattage (along with 2 to 3 loudspeakers)

may be installed in the school campus. The relay would get energized after every 40 minutes for a few seconds (say 6 Seconds).



**Fig. 15.7**

The assembly language program and data used for the purpose are given below in the mnemonic and machine code. The program is self explanatory.

| Address | Opcode   | Label | Mnemonic & Operand | Comments   |
|---------|----------|-------|--------------------|--|
| 20FC H  | 3E 80    |       | MVI A, 80 H        | ;Initialize 8255-1 as output port.                   |
| 20FE H  | DE 03    |       | OUT 03 H           |  |
| 2100 H  | 31 FF 27 |       | LXI SP, 27FF H     | ;Initialize the stack pointer.                       |
| 2103 H  | CD 47 03 |       | CALL 0347 H        | ;Clear the display.                                  |
| 2106 H  | C3 69 21 |       | JMP TT             | ;Jump to ring the bell.                              |
| 2109 H  | AF       | AA    | XRA A              | ;Put A = 0.  |
| 210A H  | 47       |       | MOV B, A           | ;Put B=0.  |
| 210B H  | 21 50 05 |       | LXI H, 2050 H      | ;Initialize address of the display.                  |
| 210E H  | CD D0 05 |       | CALL 05D0 H        | ;To display period no. and minutes to address field. |
| 2111 H  | 3E 01    |       | MVI A, 01 H        | ;Put A = 01.   |
| 2113 H  | 06 00    |       | MVI B, 00 H        | ;Put A = 00.   |
| 2115 H  | 21 54 20 |       | LXI H, 2054 H      | ;Initialize seconds locations.                       |
| 2118 H  | CD D0 05 |       | CALL 05 D0 H       | ;Displays seconds in data field.                     |

|        |          |    |                |  |
|--------|----------|----|----------------|--|
| 211B H | 21 55 20 |    | LXI H, 2055 H  | ;Initialize address of LSD of current seconds.     |
| 211E H | 7E       |    | MOV A, M       | ;Moves LSD of current seconds to accumulator.      |
| 211F H | C6 01    |    | ADI 01 H       | ;Add 1 to accumulator                              |
| 2121 H | FE 0A    |    | CPI 0A H       | ;Compares LSD of seconds with 0A (decimal no. 10). |
| 2123 H | CA 36 21 |    | JZ RR          | ;If LSD of seconds completes 09 then jump to RR.   |
| 2126 H | 77       |    | MOV M, A       | ;Moves the ACC contents to 2055 H location.        |
| 2127 H | 06 02    | DD | MVI B, 02 H    | ;Delay program                                     |
| 2129 H | 11 00 FA | YY | LXI D, FA 00 H | for one second.                                    |
| 212C H | CD 00 25 |    | CALL 2500 H    |  |
| 212F H | 05       |    | DCR B          |  |
| 2130 H | C2 29 21 |    | JNZ YY         |  |
| 2133 H | C3 09 21 |    | JMP AA         | ;Jump to AA for display the time.                  |
| 2136 H | 3E 00    | RR | MVI A, 00 H    | ;Put A=00.   |
| 2138 H | 77       |    | MOV M, A       | ;Store ACC to memory location.                     |
| 2139 H | 2B       |    | DCX H          | ;Decrement the H-L register pair content.          |
| 213A H | 7E       |    | MOVE A, M      | ;Moves the MSD of second to ACC.                   |
| 213B H | C6 01    |    | ADI 01 H       | ;Add 01 to ACC.                                    |
| 213D H | FE 06    |    | CPI 06 H       | ;Compares MSD of seconds with 06H.                 |
| 213F H | CA 46 21 |    | JZ UU          | ;If seconds completes 59 jump to UU.               |
| 2142 H | 77       |    | MOV M, A       | ;Moves Acc. contents to memory location.           |
| 2143 H | C3 27 21 |    | JMP DD         | ;Jumps for delay of 1 second.                      |
| 2146 H | 3E 00    | UU | MVI A, 00 H    | ;Put A=0 after completing 59 Sec.                  |
| 2148 H | 77       |    | MOV M, A       |  |
| 2149 H | 2B       |    | DCX H          |  |
| 214A H | 7E       |    | MOV A, M       | ;Moves LSD of current minutes to accumulator.      |

|        |          |    |               |  |
|--------|----------|----|---------------|--|
| 214B H | C6 01    |    | ADI 01 H      | ;Adds 01 to ACC.                                       |
| 214D H | FE 0A    |    | CPI 0A        | ;Compares ACC to 0A.                                   |
| 214F H | CA 56 21 |    | JZ VV         | ;Jumps to VV if LSD of minutes completes 09.           |
| 2152 H | 77       |    | MOV M, A      | ;Moves ACC contents to memory location.                |
| 2153 H | C3 27 21 |    | JMP DD        | ;Jumps for delay of 1 second.                          |
| 2156 H | 3E 00    | VV | MVI A, 00 H   |  |
| 2158 H | 77       |    | MOV M, A      |  |
| 2159 H | 2B       |    | DCX H         | ;Decrement the contents of H-L register pair.          |
| 215A H | 7E       |    | MOV A, M      | ;Moves MSD of minutes to ACC.                          |
| 215B H | C6 01    |    | ADI 01 H      | ;Add 1 to it.  |
| 215D H | FE 04    |    | CPI 04 H      | ;Compares ACC contents with 04 H.                      |
| 215F H | CA 66 21 |    | JZ SS         | ;If minutes 40 then jumps to SS.                       |
| 2162 H | 77       |    | MOV M, A      |  |
| 2163 H | C3 27 21 |    | JMP DD        | ;Jumps for delay of 1 second.                          |
| 2166 H | 3E 04    | SS | MVI A, 04 H   | ;Put A=4.  |
| 2168 H | 77       |    | MOV M, A      |  |
| 2169 H | AF       | TT | XRA A         | ;Put A=0.  |
| 216A H | 47       |    | MOV B, A      | ;Put b=0.  |
| 216B H | 21 50 20 |    | LXI H, 2050 H |  |
| 216E H | CD D0 05 |    | CALL 05 D0 H  | ;Displays the period No. and minutes in address field. |
| 2171 H | 3E 01    |    | MVI A, 01 H   | ;Put A=01.   |
| 2173 H | 06 00    |    | MVI B, 00 H   | ;Put B=00.   |
| 2175 H | 21 54 20 |    | LXI H, 2054 H |  |
| 2178 H | CD D0 05 |    | CALL 05 D0 H  | ;Displays the seconds in data field.                   |
| 217B H | 3E 01    |    | MVI A, 01 H   |  |
| 217D H | D3 00    |    | OUT 00 H      | ;Excite 8255-I to energize the relay (rings the bell). |
| 217F H | 21 55 20 |    | LXI H, 2055 H |  |

|        |          |                |   |
|--------|----------|----------------|---|
| 2182 H | 3E 00    | MVI A, 00 H    | ; Puts 00 to 2055 H<br>to<br>2052 H                         |
| 2184 H | 77       | MOV M, A       |   |
| 2185 H | 2B       | DCX H          |   |
| 2186 H | 77       | MOV M, A       |   |
| 2187 H | 2B       | DCX H          |   |
| 2188 H | 77       | MOV M, A       |   |
| 2189 H | 2B       | DCX H          |   |
| 218A H | 77       | MOV M, A       |   |
| 218B H | 2B       | DCX H          |   |
| 218C H | 7E       | MOV A, M       | ;Brings the LSD of current period number to ACC.            |
| 218D H | C6 01    | ADI 01 H       | ;Add 1 to it.   |
| 218F H | FE 0A    | CPI 0A H       | ;Compares with 0A.  |
| 2191 H | CA 98 21 | JZ XX          | ;If LSD of period number complete 09 then jump to XX.       |
| 2194 H | 77       | MOV M, A       | ;Else stores it to memory location.                         |
| 2195 H | C3 A0 21 | JMP XY         | ;Jump to XY.  |
| 2198 H | 3E 00    | XX             | MVI A, 00 H   |
| 219A H | 77       | MOV M, A       | ;Puts A=00.<br>;Stores MSD of period number to accumulator. |
| 219B H | 2B       | DCX H          |   |
| 219C H | 7E       | MOV A, M       | ;Stores MSD of period number to accumulator.                |
| 219D H | C6 01    | ADI 01 H       | ;Adds 1 to it.  |
| 219F H | 77       | XXX            | MOV M, A  |
| 21A0 H | 06 02    | XY             | ; Delay program for one second.                             |
| 21A2 H | 11 00 FA | XYZ            |   |
| 21A5 H | CD 00 25 | LXI D, FA 00 H |   |
| 21A8 H | 05       | CALL 2500 H    |   |
| 21A9 H | C2 A2 21 | DCR B          |   |
| 21AC H | AF       | JNZ XYZ        |   |
| 21AD H | 47       | XRA A          |   |
| 21AE H | 21 50 20 | MOV B, A       |   |
| 21B1 H | CD D0 05 | LXI H, 2050 H  | ;Program to display the period number minutes and seconds.  |
| 21B4 H | 3E 01    | CALL 05D0 H    |   |
| 21B6 H | 06 00    | MVI A, 01 H    |   |
| 21B8 H | 21 54 20 | MVI B, 00 H    |   |
| 21BB H | CD D0 05 | LXI H, 2054 H  |   |
| 21BE H | 21 55 20 | CALL 05D0 H    |   |
|        |          | LXI H, 2055 H  |   |

|        |          |             |   |
|--------|----------|-------------|---|
| 21C1 H | 7E       | MOV A, M    | ;Stores LSD of current seconds to ACC.      |
| 21C2 H | C6 01    | ADI 01 H    | ;Adds 1 to it.                              |
| 21C4 H | FE 06    | CPI 06 H    | ;Compares it with 06.                       |
| 21C6 H | C2 9F 21 | JNZ XXX     | ;If not 06 then jump to XXX.                |
| 21C9 H | 3E 00    | MVI A, 00 H | ;Put A=0 .                                  |
| 21CB H | D3 00    | OUT 00 H    | ;Output to 8255-I to de-energize the relay. |
| 21CD H | C3 09 21 | JMP AA      | ;Repeat for the next period.                |

### **DELAY SUBROUTINE AT MEMORY LOCATION 2500 H**

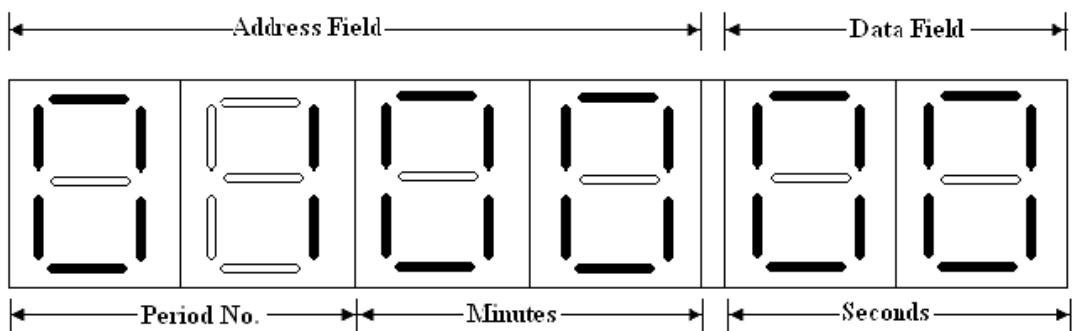
|        |          |      |          |
|--------|----------|------|----------|
| 2500 H | 1B       | NEXT | DCX D    |
| 2501 H | 7A       |      | MOV A, D |
| 2502 H | B3       |      | ORA E    |
| 2503 H | C2 00 25 |      | JNZ NEXT |
| 2506 H | C9       |      | RET      |

### **DATA TO BE FILLED BEFORE THE EXECUTION OF THE PROGRAM**

|        |    |                                 |
|--------|----|---------------------------------|
| 2050 H | 00 | MSD OF PERIOD NO. IS STORED 00. |
| 2051 H | 00 | LSD OF PERIOD NO. IS STORED 00. |
| 2052 H | 00 | MSD OF MINUTES IS STORED 00.    |
| 2053 H | 00 | LSD OF MINUTES IS STORED 00.    |
| 2054 H | 00 | MSD OF SECONDS IS STORED 00.    |
| 2055 H | 00 | LSD OF SECONDS IS STORED 00.    |

First, the program and data are entered in the microprocessor kit. When the program is run a bell sound will be heard for few seconds (say 6 seconds). This will be repeated after every 40 minutes. The period number will be displayed in the two higher digits of the address field. The minutes and seconds of the time elapsed will be displayed in the rest two digits of lower digits of address field and two digits of the data field.

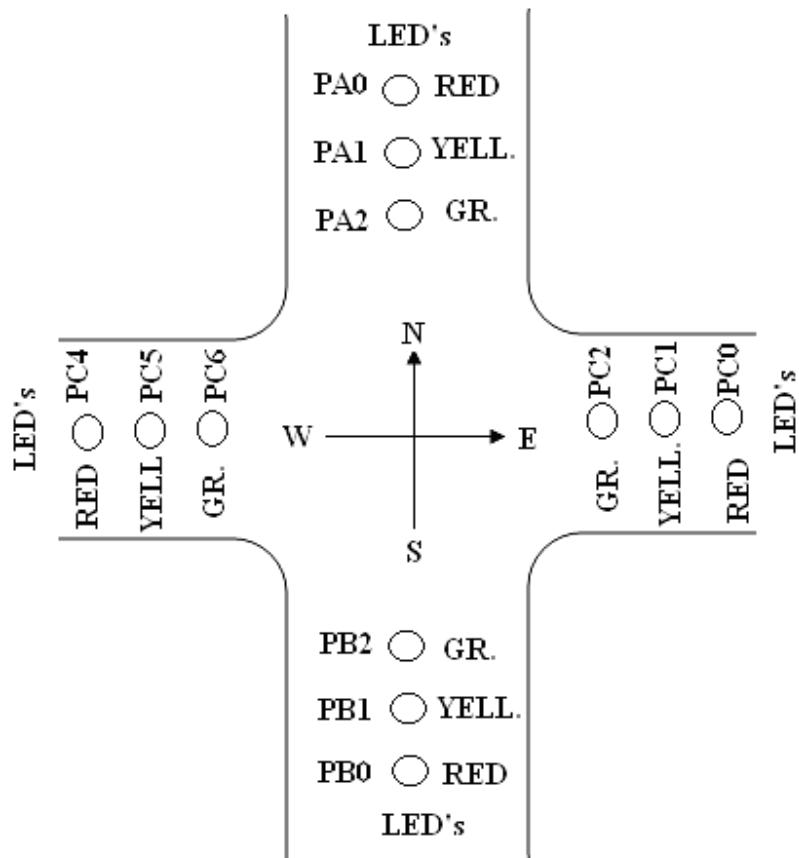
The address and data fields of the microprocessor kit would display the period number and elapsed minutes and seconds in the address and data fields as given below (fig. 15.8). This figure indicate that period No. is 1, minutes and seconds elapsed are 0, 0 respectively.



**Fig. 15.8**

### 15.5 MICROPROCESSOR BASED TRAFFIC LIGHT

The traffic light controller, which is generally seen at the crossing of the roads, may be designed using the assembly language programming of the microprocessor 8085A. The traffic light controller suggested in this section uses four sets of three LEDs (red, yellow and green) arranged at the four sides of the crossing as shown in figure 15.9. The microprocessor kit (M/S Vinytis) was used to design the traffic light controller.



**Fig. 15.9**

The ports of PPI 8255-I (Port A, Port B, Port C<sub>Lower</sub> and Port C<sub>Upper</sub>) available with the kit were used as the output ports. The Red, Yellow and Green LEDs of North are connected to PA<sub>0</sub>, PA<sub>1</sub> and PA<sub>2</sub> (bits 0, 1 and 2 of port A) pins of 8255-I respectively. The bits 0, 1, 2 of port B (PB<sub>0</sub>, PB<sub>1</sub> and PB<sub>2</sub>) were connected to the red, yellow and green LEDs of South respectively. Similarly, PC<sub>0</sub>, PC<sub>1</sub> and PC<sub>2</sub> of port C<sub>Lower</sub> and PC<sub>4</sub>, PC<sub>5</sub> and PC<sub>6</sub> of port C<sub>Upper</sub> were connected to the red, yellow and green LEDs of East and West sides.

Now the following four points occurs for the traffic to control:

- (1) First traffic to east west direction is allowed for some time (delay time of 60 seconds say) and traffic to north and south side is not allowed for the sane delay time. This is possible if Red LEDs of north and south (PA<sub>0</sub> and PB<sub>0</sub> both are high) and Green LEDs of East and West (PC<sub>2</sub> and PC<sub>6</sub> are also high) glow.
- (2) The yellow lights of east-west and north-south sides are allowed to glow to clear the traffic of these directions for some time (say 20 seconds). The green LEDs of east and west, and red LEDs of north and south are to be remained off for the same time. This condition will be satisfied if PC<sub>1</sub> and PC<sub>5</sub>, and PA<sub>1</sub> and PB<sub>1</sub> are high; and PA<sub>0</sub> and PB<sub>0</sub>, PC<sub>2</sub> and PC<sub>6</sub> are low.
- (3) Now the traffic to north and south side is allowed for the delay time of 60 seconds and traffic to east west direction is not allowed for the sane delay time. This is possible if green LEDs of north and south (PA<sub>2</sub> and PB<sub>2</sub> both are high) and red LEDs of East and West (PC<sub>0</sub> and PC<sub>4</sub> are also high) glow.
- (4) Again the yellow lights of east-west and north-south sides are allowed to glow to clear the traffic of these directions for some time (say 20 seconds). The green LEDs of north and south, and red LEDs of east and west are to be remained off for the same time. This condition will be satisfied if PC<sub>1</sub> and PC<sub>5</sub>, and PA<sub>1</sub> and PB<sub>1</sub> are high.

The assembly language program for the above conditions is given below, which is self explanatory:

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>                                    |
|--------------|------------------|----------------|--|
|              | LXI SP,          | XXXX H         | ;Initialize the stack pointer.                     |
|              | MVI A,           | 80H            | ;Control word for 8255-I.                          |
|              | OUT              | 03H            | ;Works all the ports of 8255-I as output ports.    |
| AGAIN        | MVI A,           | 01 H           | ;01 is loaded to accumulator.                      |
|              | OUT              | 00 H           | ;PA <sub>0</sub> is high (red LED of north glows). |
|              | OUT              | 01 H           | ;PB <sub>0</sub> is high (red LED of south glows). |

|        |          |  |
|--------|----------|--|
| MVI A, | 44 H     | ;PC2 and PC6 are high<br>(green LEDs of east and west glows).                          |
| OUT    | 02 H     |  |
| CALL   | DELAY I  | ; Delay program of 60 seconds.   |
| MVI A, | 22 H     |  |
| OUT    | 02 H     | ; PC <sub>1</sub> and PC <sub>5</sub> are high<br>(Yellow LEDs of east and west glow). |
| MVI A, | 02 H     |  |
| OUT    | 00 H     | ; PA <sub>1</sub> is high (yellow LED of north glows).                                 |
| OUT    | 01 H     | ; PB <sub>1</sub> is high (yellow LED of south glows).                                 |
| CALL   | DELAY II | ; Delay program of 20 seconds.   |
| MVI A, | 11 H     |  |
| OUT    | 02 H     | ; PC <sub>0</sub> and PC <sub>4</sub> are high (red LEDs of east and west glow).       |
| MVI A, | 04 H     |  |
| OUT    | 00 H     | ; PA <sub>2</sub> is high (green LED of north glows).                                  |
| OUT    | 01 H     | ; PB <sub>2</sub> is high (green LED of south glows).                                  |
| CALL   | DELAY I  | ; Delay program of 60 seconds.   |
| MVI A, | 22 H     |  |
| OUT    | 02 H     | ; PC <sub>1</sub> and PC <sub>5</sub> are high<br>(Yellow LEDs of east and west glow). |
| MVI A, | 02 H     |  |
| OUT    | 00 H     | ; PA <sub>1</sub> is high (yellow LED of north glows).                                 |
| OUT    | 01 H     | ; PB <sub>1</sub> is high (yellow LED of south glows).                                 |
| CALL   | DELAY II | ; Delay program of 20 seconds.   |
| JMP    | AGAIN    | ; Jump to repeat the program from the beginning.                                       |

Subroutine programs to introduce delay for 60 seconds and 20 seconds are given below:

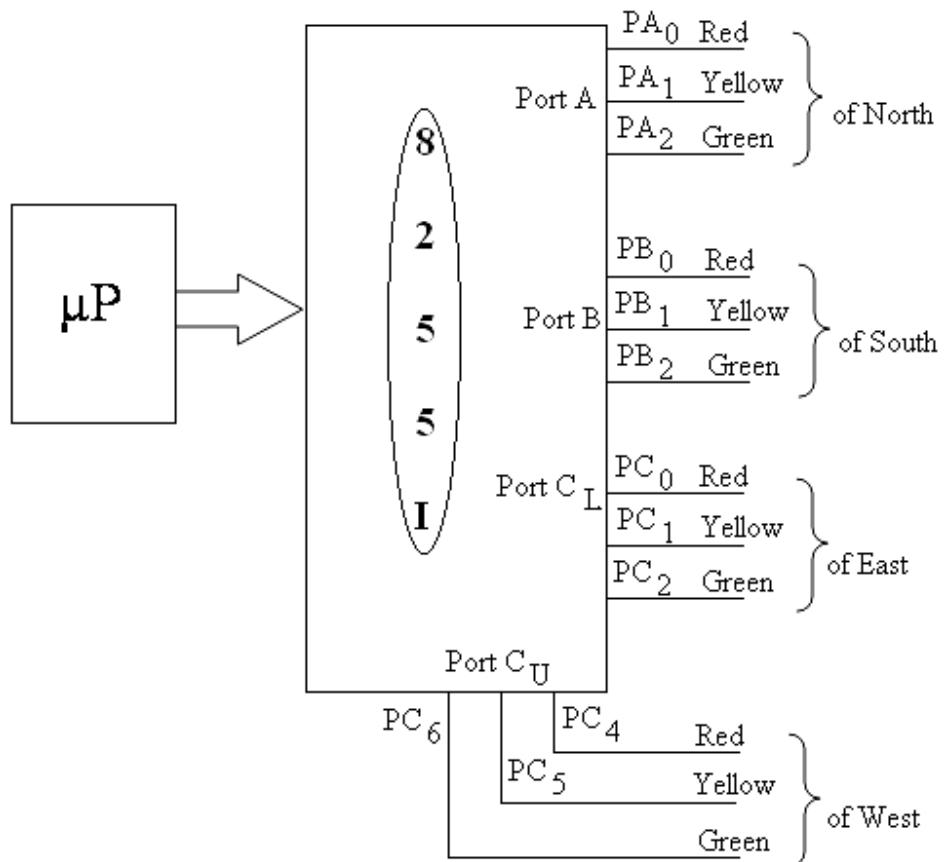
### Subroutine Program for DELAY I

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>   |
|--------------|------------------|----------------|---|
| DELAY I      | MVI B,           | 78 H           | ;78 H (120 decimal number) is loaded to B register for the loop.        |
| YY<br>NXT1   | LXI D,<br>DCX    | FA00 H<br>D    | ;Decrement D-E register pair.   |
|              | MOV A,           | D              | ;Moves the contents stored in M <sub>D-E</sub> to Acc.                  |
|              | ORA              | E              | ;The contents of A and E registers are ORed bit by bit.                 |
|              | JNZ              |                | NXT1;If the result is not zero then jump to nxt1 else next instruction. |
|              | DCR              | B              | ;Decrement B register   |
|              | JNZ              | YY             |   |
|              | RET              |                | ; Returns to main program.  |

### Subroutine Program for DELAY II

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>   |
|--------------|------------------|----------------|---|
| DELAY II     | MVI B,           | 28 H           | ;28 H (40 decimal number) is loaded to B register for the loop. |
|              | JMP              | YY             | ; Jump to YY of DELAY I program.                                |

Figure 15.10 shows the LED connections to 8255-I available with the microprocessor kit.

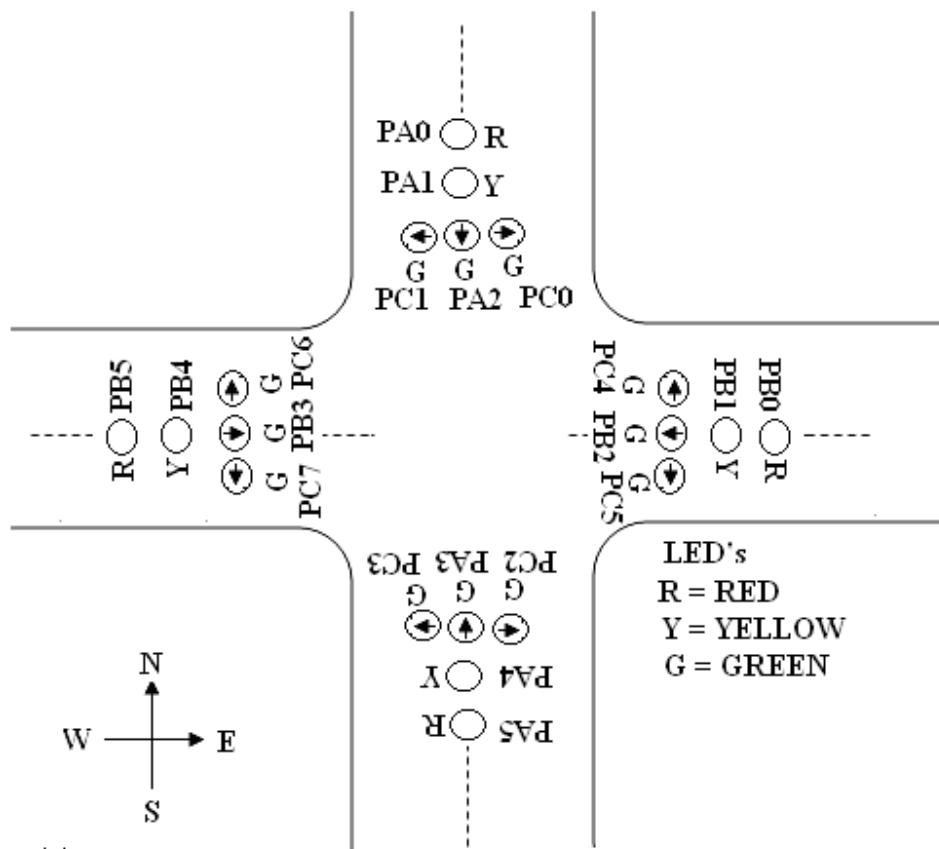


**Fig. 15.10**

### 15.5.1 Another Design of Microprocessor Based Traffic Light

Here another design of Traffic light controller is given in which there are more degree of freedom for the traffic. Further, the display of time delay (count down) on the microprocessor kit is also shown. In this design, the divider lines at the four path ways are shown (ref. figure 15.11). At each divider line of four path ways near the crossing, five LEDs (one red, one yellow and three green) are installed on four pillars. These LEDs are connected to the ports of PPI 8255-I available with the microprocessor kit. All the ports of 8255 are used as output ports. The status of LEDs will indicate the direction of the traffic at the crossing.

In this design the delay of 90 seconds are allowed for the normal traffic and say 10 seconds delay is introduced for yellow light to clear the traffic, before the next traffic direction. The delay will be displayed on the data field with down counting. The assembly language programming of the design is given below. The program is simple and self-explanatory.



**Fig. 15.11**

**Main Program:**

**LOOK UP TABLE:**

|        |    |       |
|--------|----|-------|
| 2050 H | 16 | BLANK |
| 2051H  | 16 | BLANK |
| 2052 H | 16 | BLANK |
| 2053 H | 16 | BLANK |

| Label | Mnemonics                | Operand              | Comments   |
|-------|--------------------------|----------------------|--|
|       | LXI SP,<br>MVI A,<br>OUT | XXXX H<br>80H<br>03H | ;Initialize the stack pointer.<br>;Control word for 8255-I.<br>;Works all the ports of 8255-I as output ports. |
| START | MVI A,                   | 21 H                 | ; 21 H is loaded to accumulator.   |

|        |          |   |
|--------|----------|---|
| OUT    | 00 H     | ;PA0 and PA5 are high (red LEDs of north and south glow).                         |
| MVI A, | 0C H     | ; 0C H is loaded to accumulator.  |
| OUT    | 01 H     | ; PB2 and PB3 are high (green LEDs of East and west glow).                        |
| MVI A, | 60 H     | ; 60 H is loaded to acc.  |
| OUT    | 02 H     | ; PC5 and PC6 are high (Traffic from west to north and east to south is allowed). |
| CALL   | DELAY I  | ; Calls the delay subroutine program (DELAY I).                                   |
| CALL   | DISPLAY  | ; Calls the Display subroutine program for the display of delay time.             |
| MVI A, | 21 H     | ;21 H is loaded to accumulator.   |
| OUT    | 00 H     | ;PA0 and PA5 are high (red LEDs of north and south remain glow).                  |
| MVI A, | 12 H     | ; 12 H is loaded to accumulator.  |
| OUT    | 01 H     | ; PB1 and PB4 are high (yellow LEDs of East and west glow).                       |
| MVI A, | 00 H     | ; 00 H is loaded to acc.  |
| OUT    | 02 H     | ; PC0 to PC7 are low.   |
| CALL   | DELAY II | ; Calls the delay subroutine program (DELAY II).                                  |
| CALL   | DISPLAY  | ; Calls the Display subroutine program for the display of delay time.             |
| MVI A, | 21 H     | ; 21 H is loaded to accumulator.  |
| OUT    | 00 H     | ;PA0 and PA5 are high (red LEDs of north and south glow).                         |
| OUT    | 01 H     | ; PB0 and PB5 are high (red LEDs of East and west glow).                          |
| MVI A, | 96 H     | ; 96 H is loaded to acc.  |
| OUT    | 02 H     | ; PC1, PC2, PC4 and PC7 are high (Traffic from west                               |

|        |          |   |
|--------|----------|---|
|        |          | to south and east to north is allowed).   |
| CALL   | DELAY I  | ; Calls the delay subroutine program (DELAY I).                                   |
| CALL   | DISPLAY  | ; Calls the Display subroutine program for the display of delay time.             |
| MVI A, | 12 H     | ; 12 H is loaded to accumulator.  |
| OUT    | 00 H     | ; PA1 and PA4 are high (yellow LEDs of north and south glow).                     |
| OUT    | 01 H     | ; PB1 and PB4 are high (yellow LEDs of East and west glow).                       |
| MVI A, | 00 H     | ; 00 H is loaded to acc.  |
| OUT    | 02 H     | ; PC0 to PC7 are low.   |
| CALL   | DELAY II | ; Calls the delay subroutine program (DELAY II).                                  |
| CALL   | DISPLAY  | ; Calls the Display subroutine program for the display of delay time.             |
| MVI A, | 0C H     | ; 0C H is loaded to accumulator.  |
| OUT    | 00 H     | ; PA2 and PA3 are high (green LEDs of north and south glow).                      |
| MVI A, | 21 H     | ; 21 H is loaded to accumulator.  |
| OUT    | 01 H     | ; PB0 and PB5 are high (red LEDs of East and west glow).                          |
| MVI A, | 09 H     | ; 09 H is loaded to acc.  |
| OUT    | 02 H     | ; PC0 and PC3 are high (Traffic from north to east and south to west is allowed). |
| CALL   | DELAY I  | ; Calls the delay subroutine program (DELAY I).                                   |
| CALL   | DISPLAY  | ; Calls the Display subroutine program for the display of delay time.             |
| MVI A, | 12 H     | ; 12 H is loaded to accumulator.  |

|                       |                          |   |
|-----------------------|--------------------------|---|
| OUT                   | 00 H                     | ;PA1 and PA4 are high<br>(yellow LEDs of north and south glow).   |
| MVI B,<br>OUT         | 21 H<br>01 H             | ; 21 H is loaded to acc.<br>; PB0 and PB5 are high (red LEDs of East and west glow).                                    |
| MVI A,<br>OUT<br>CALL | 00 H<br>02 H<br>DELAY II | ; 00 H is loaded to acc.<br>; PC0 to PC7 are low.<br>; Calls the delay subroutine program (DELAY II).                   |
| CALL                  | DISPLAY                  | ; Calls the Display subroutine program for the display of delay time.   |
| MVI A,                | 21 H                     | ; 21 H is loaded to accumulator.  |
| OUT                   | 00 H                     | ;PA0 and PA5 are high (red LEDs of north and south glow).   |
| OUT                   | 01 H                     | ; PB0 and PB5 are high (red LEDs of East and west glow).  |
| MVI A,<br>OUT         | 96 H<br>02 H             | ; 99 H is loaded to acc.<br>; PC1, PC2, PC4 and PC7 are high (Traffic from north to west and south to east is allowed). |
| CALL                  | DELAY I                  | ; Calls the delay subroutine program (DELAY I).   |
| CALL                  | DISPLAY                  | ; Calls the Display subroutine program for the display of delay time.   |
| MVI A,                | 12 H                     | ; 12 H is loaded to accumulator.  |
| OUT                   | 00 H                     | ;PA1 and PA4 are high (yellow LEDs of north and south glow).  |
| OUT                   | 01 H                     | ; PB1 and PB4 are high (yellow LEDs of East and west glow).   |
| MVI A,<br>OUT<br>CALL | 00 H<br>02 H<br>DELAY II | ; 00 H is loaded to acc.<br>; PC0 to PC7 are low.<br>; Calls the delay subroutine program (DELAY II).                   |

|      |         |   |
|------|---------|---|
| CALL | DISPLAY | ; Calls the Display subroutine program for the display of delay time. |
| JMP  | START   | ; Jump to START.  |

### **Subroutine Program for DELAY I:**

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>  |
|--------------|------------------|----------------|--|
| DELAY I      | MVI A,           | 09 H           | ; 09 H is loaded to accumulator.                         |
|              | STA              | 2054           | ; Store it to memory location 2054 H.                    |
|              | MVI A,           | 00 H           |  |
|              | STA              | 2055 H         | ; for display of delay of 90 seconds for normal traffic. |
|              | RET              |                | ; Returns to main program.                               |

### **Subroutine Program for DELAY II:**

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>  |
|--------------|------------------|----------------|--|
| DELAY II     | MVI A,           | 01 H           | ; 01 H is loaded to accumulator.   |
|              | STA              | 2054           | ; Store it to memory location 2054 H.  |
|              | MVI A,           | 00 H           |  |
|              | STA              | 2055 H         | ; for display of delay of 10 seconds for clearing the traffic during yellow light at the crossing. |
|              | RET              |                | ; Returns to main program.   |

### **Subroutine Program for the display of Delay time (count down):**

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>   |
|--------------|------------------|----------------|---|
| DISPLAY      | CALL             | 0347 H         | ; Clears the display. It is the program stored in ROM of the kit. |
| AA           | XRA              | A              | ; Clears the accumulator  |
|              | MOV B,           | A              | ; Clears the B register also.                                     |
|              | LXI H,           | 2050 H         | ; Initialize H-L pair where the current delay time is stored.     |

|         |                            |                        |  |
|---------|----------------------------|------------------------|--|
|         | CALL                       | 05D0 H                 | ;Displays the current time in address field. It is the program stored in ROM of the Kit. Here blank is displayed in the address field. |
|         | MVI A,<br>MVI B,<br>LXI H, | 01 H<br>00 H<br>2054 H | ;Stores 01 H in accumulator.<br>;Stores 00 H in B register.<br>;Initialize H-L register pair with 2054 H.                              |
|         | CALL                       | 05D0 H                 | ;Displays the current delay time in data field (90 seconds or 10 seconds).   |
|         | LXI H,                     | 2055 H                 | ;Initialize H-L register pair with 2055 H.   |
|         | MOV A,                     | M                      | ;Moves the least significant digit (LSD) of seconds to the accumulator.  |
|         | CPI<br>JNZ                 | 00 H<br>PROCEED        | ;Compares with 00 H.<br>;If Acc.is not 00 H, then jump to PROCEED.   |
|         | DCX                        | H                      | ;Decrement H-L register pair.  |
|         | MOV A,                     | M                      | ;Moves MSD of the seconds to accumulator.  |
|         | CPI<br>RZ                  | 00 H                   | ;Compaers it with 00.<br>;Return if both are zero.   |
| PROCEED | MVI B,                     | 02 H                   | ; Program for delay of 1 sec starts. It stores 02 H in accumulator.  |
| YY      | LXI D,                     | FA00 H                 | ;Intialize D-E register pair with FA00H.   |
| NXT     | DCX                        | D                      | ;Decrement D-E register pair.  |
|         | MOV A,                     | D                      | ;Moves the contents stored in M <sub>D-E</sub> to Acc.   |
|         | ORA                        | E                      | ;The contents of A and E registers are ORed bit by bit.  |
|         | JNZ                        | NXT                    | ;If the result is not zero then jump to NXT else next instruction.   |
|         | DCR<br>JNZ<br>LXI H,       | B<br>YY<br>2055 H      | ; End of 1 sec delay.<br>;Initialize H-L register pair with 2055 H.  |

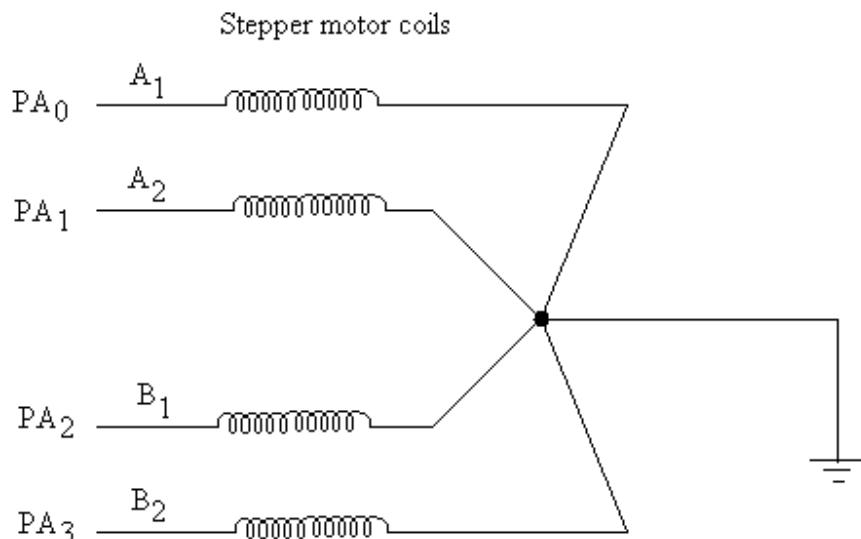
|    |        |      |   |
|----|--------|------|---|
|    | MOV A, | M    | ;Moves the least significant digit (LSD) of seconds to the accumulator. |
|    | CPI    | 00 H | ;Compares with 00 H.  |
|    | JZ     | BB   | ;If Acc.=00 H, then jump to BB. Else goes to next statement.            |
|    | SUI    | 01 H | ; Subtract 01 from the acc.   |
|    | MOV M, | A    | ; Store it to memory location.  |
|    | JMP    | AA   | ; Jump to AA for the display.   |
| BB | MVI A, | 09 H | ; Move 09 to Acc.   |
|    | MOV M, | A    | ; It is stored in the memory location.                                  |
|    | DCX H  |      | ; Decrement of H-L register pair.                                       |
|    | MOV A, | M    | ; Contents of this location is moved to Acc.                            |
|    | CPI    | 00 H | ; Compared with 00.   |
|    | JNZ    | CC   | ; If not zero jump to CC.   |
|    | INX    | H    |   |
|    | MVI A, | 00 H | ; Get back 0 in the same location.                                      |
|    | MOV M, | A    |   |
|    | JMP    | AA   | ; Jump to AA for the display.   |
| CC | SUI    | 01 H | ; Subtract 01 from the location.  |
|    | MOV M, | A    | ; After subtraction store to the memory location.                       |
|    | JMP    | AA   | ; Jump to AA for the display.   |

### 15.6 MICROPROCESSOR BASED STEPPER MOTOR CONTROL

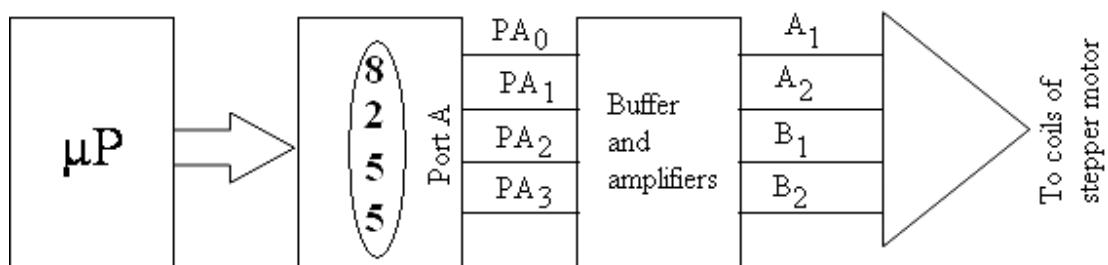
In this section the design of microprocessor based stepper motor control will be discussed. The stepper motor has four windings such that the motor rotates in precise steps from one fixed position to another. A stepper motor is an electromechanical device which converts electrical pulses into discrete mechanical movements. The shaft or spindle of a stepper motor rotates in discrete step increments when electrical command pulses are applied to it in the proper sequence. The motor's rotation has several direct relationships to these applied input pulses. The sequence of the applied pulses is directly related to the direction of motor shafts rotation. The speed of the motor shafts rotation is directly related to the frequency of the input pulses and the length of rotation is directly related to the number of input pulses applied. A stepper motor can be a good choice whenever controlled movement is required. They can be used where the control rotation angle, speed, position and synchronism are needed. Because of the inherent advantages

stepper motors have found their place in many different applications. Some of these include printers, plotters, high end office equipment, hard disk drives, medical equipments, fax machines, automotive and many more.

The stepper motor is to be interfaced with microprocessor by 8255A PPI and some buffers/ amplifiers. The four windings (coils) of the stepper motor are shown in figure 15.12. However, figure 15.13 shows the block diagram of the interfacing connections for the stepper motor. One set of coils of the stepper motor is energized using 12 V d. c. supply in the form of pulse. After energizing one set of coil windings some time delay is provided through software, then the supply is given to other set of windings. The speed of the motor will be governed by the delay introduced in the system.



**Fig. 15.12**



**Fig. 15.13**

For the clockwise movement of the stepper motor, it is energized in the sequence:

**A<sub>1</sub> A<sub>2</sub>,      A<sub>2</sub> B<sub>1</sub>,      B<sub>1</sub> B<sub>2</sub>,      B<sub>2</sub> A<sub>1</sub>** and then repeats.

Similarly, in the anti-clock wise movement of the stepper motor, it is energized in the sequence:

**A<sub>1</sub> A<sub>2</sub>,      B<sub>2</sub> A<sub>1</sub>,      B<sub>1</sub> B<sub>2</sub>,      A<sub>2</sub> B<sub>1</sub>** and then repeats.

Number of steps or movements, for the stepper motor to be moved, are loaded in the register C. For example the stepper motor to be moved for 20 steps then its equivalent hexadecimal number 14 H is to be loaded this register before the start of the run. Similarly, if the stepper motor is to be rotated in clock wise direction then load 00 in

register B or load 01 in register B if the motor is to be moved in anti-clock wise direction. The flow chart for the movement of the stepper motor is shown in figure 15.14.

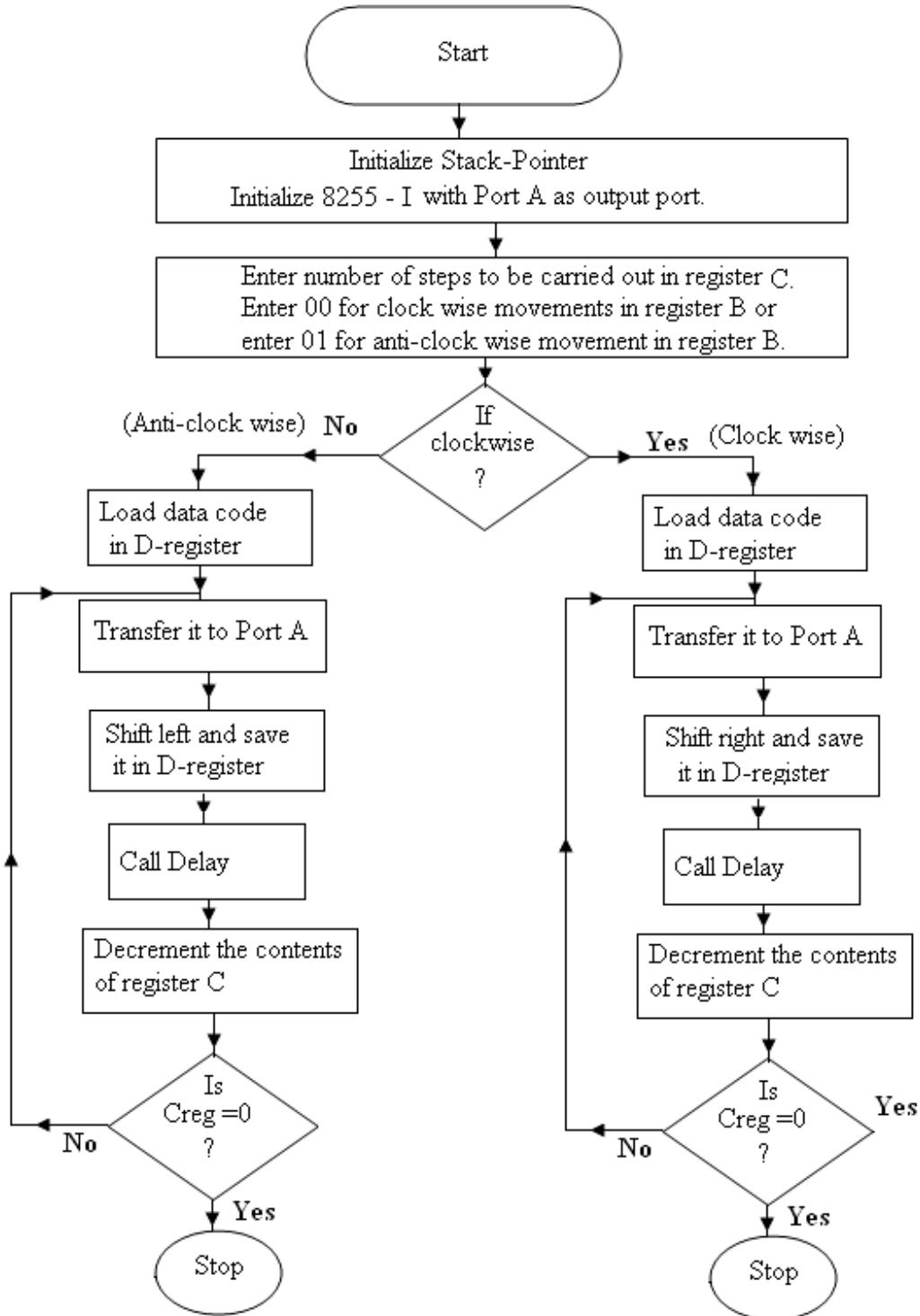
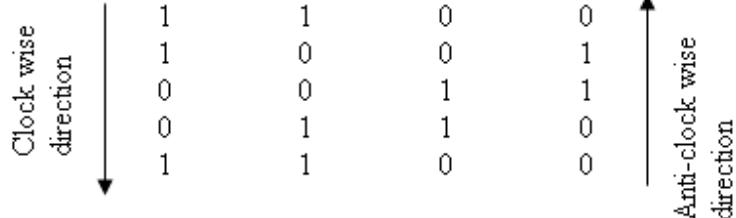


Fig. 15. 14

Further, the bits ( $PA_0$ ,  $PA_1$ ,  $PA_2$  and  $PA_3$ ) of port A of 8255A are to be connected to  $A_1$ ,  $A_2$ ,  $B_1$  and  $B_2$  coils of the motor respectively. The movements of the motor in clockwise or anti-clock wise direction should be as:

| $B_2$  | $B_1$  | $A_2$  | $A_1$  |
|--------|--------|--------|--------|
| $PA_3$ | $PA_2$ | $PA_1$ | $PA_0$ |
| 1      | 1      | 0      | 0      |
| 1      | 0      | 0      | 1      |
| 0      | 0      | 1      | 1      |
| 0      | 1      | 1      | 0      |
| 1      | 1      | 0      | 0      |



This is possible if CC H is loaded to the D-register and then data is rotated clockwise or anti-clock wise direction with RRC or RLC instructions as shown below:

| C       | C (CC H in D-register) |
|---------|------------------------|
| 1 1 0 0 | 1 1 0 0                |
| 1 0 0 1 | 1 0 0 1                |
| 0 0 1 1 | 0 0 1 1                |
| 0 1 1 0 | 0 1 1 0                |
| 1 1 0 0 | 1 1 0 0                |



The assembly language program for the above description is given below which is self explanatory:

| Label   | Mnemonics | Operand | Comments  |
|---------|-----------|---------|---|
| LXI SP, |           | XXXX H  | ;Initialize the stack pointer.  |
| MVI A,  |           | 80H     | ;Control word for 8255-I.   |
| OUT     |           | 03H     | ;Works all the ports of 8255-I as output ports.                                   |
| MVI C,  |           | 14 H    | ;14 H (20 steps for movement) is loaded to C-register. It is used as counter.     |
| MVI B,  |           | YY H    | ; YY may be 00 H or 01 H for clockwise or anti-clock wise direction respectively. |
| MOV A,  |           | B       | ; Move the contents of B-register to accumulator.                                 |

|                  |                         |           |   |
|------------------|-------------------------|-----------|---|
|                  | CPI                     | 01 H      | ; Check if movement is clockwise or anti-clock wise.                              |
|                  | JNZ                     | CLKWISE   | ; If condition is satisfied then clockwise direction else anti-clock wise.        |
| LOOP1            | MVI D,<br>MOV A,        | CC H<br>D | ; Move CC H to D-register.<br>; This data is moved to accumulator.                |
|                  | OUT                     | 00 H      | ; Data is sent to coils of the motor through Port A of 8255A.                     |
|                  | RLC                     |           | ; Data is rotated left for anti-clock wise movement of the motor.                 |
|                  | MOV D,                  | A         | ; After rotation data is stored in D register.                                    |
|                  | PUSH PSW                |           | ; Push Acc content and flag content to Stack.                                     |
|                  | PUSH D                  |           | ; Push content of D-E register pair to Stack.                                     |
|                  | PUSH B                  |           | ; Push content of B-C register pair to Stack.                                     |
|                  | CALL                    | DELAY     | ; Call subroutine program for delay as per the requirement.                       |
|                  | POP B                   |           | ; Get back the content of B-C register pair from the Stack.                       |
|                  | POP D                   |           | ; Get back the content of D-E register pair from the Stack.                       |
|                  | POP PSW                 |           | ; Get back the Acc and flag content from the Stack.                               |
|                  | DCR                     | C         | ; Decrement in number of counts.  |
|                  | JNZ                     | LOOP1     | ; If counts are not 0 then jump to LOOP1 for next movement, else stop processing. |
| CLKWISE<br>LOOP2 | HLT<br>MVI D,<br>MOV A, | CC H<br>D | ; Move CC H to D-register.<br>; This data is moved to accumulator.                |

|          |       |   |
|----------|-------|---|
| OUT      | 00 H  | ; Data is sent to coils of the motor through Port A of 8255A. |
| RRC      |       | ; Data is rotated right for clock wise movement of the motor. |
| MOV D,   | A     | ; After rotation data is stored in D register.                |
| PUSH PSW |       | ; Push Acc content and flag content to Stack.                 |
| PUSH D   |       | ; Push content of D-E register pair to Stack.                 |
| PUSH B   |       | ; Push content of B-C register pair to Stack.                 |
| CALL     | DELAY | ; Call subroutine program for delay as per the requirement.   |
| POP B    |       | ; Get back the content of B-C register pair from the Stack.   |
| POP D    |       | ; Get back the content of D-E register pair from the Stack.   |
| POP PSW  |       | ; Get back the Acc and flag content from the Stack.           |
| DCR      | C     | ; Decrement in number of counts.                              |
| JNZ      | LOOP2 | ; If counts are not 0 then jump to LOOP2 for next movement.   |
| HLT      |       | ; stop processing.  |

Subroutine programs to introduce delay for required time may be written as discussed earlier.

### 15.7 MICROPROCESSOR BASED WASHING MACHINE CONTROLLER

The automatic washing machines available in the market are microprocessor based. Here conditional sequences are considered which are controlled by microprocessor along with interfacing devices. It basically involves trigger inputs as well as specific delays. In other words there are some trigger inputs which leads the washing machine to work with certain time delays as well.

The port A and Port B of 8255-I are used for the purpose of control of certain functions of washing machine. Let the bits D0 and D1 of port A are used for introducing time delay for certain time as given below:

|    |    |
|----|----|
| D1 | D0 |
|----|----|

|   |   |   |
|---|---|---|
| 0 | 0 | No time delay required.                                   |
| 0 | 1 | Delay 1 for certain time delay (say for t1)               |
| 1 | 0 | Delay 2 for certain other time delay (say for t2 seconds) |
| 1 | 1 | Not used  |

D2 bit of the Port A is used for the tub motor rotating at the drying speed.

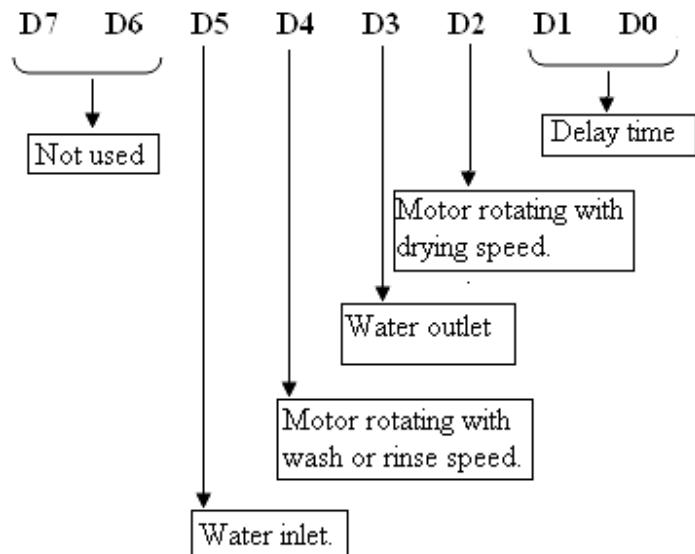
D3 bit of the port A is used for water outlet.

D4 bit of this port is used for the tub motor rotating at wash or rinse speed.

D5 bit of this port is used for water inlet.

D6 and D7 bits are not used.

The devices connected to the Port A of 8255-I for the purpose of controlling events during the operation of washing machine are as given in figure 15. 15



**Fig. 15.15**

Consider a water level sensor connected with a D1 bit of port B of 8255-I. The port B is used as the input port which will indicate if the water is filled to the specified level.

The sequence of events to be undertaken during the operation of washing machine is as given below:

1. The tub is filled with water up to the specified level with the water inlet. The level of the water will be sensed by the water level sensor.
  2. The tub is rotated at wash or rinse speed for a fixed time  $t_1$ .
  3. The tub is allowed to empty for a fixed time  $t_2$ .
  4. The tub is filled again with water up to a specified level whose level will be sensed with water level sensor.
  5. The tub is rotated again with wash or rinse speed for the fixed time  $t_1$ .

6. The tub is allowed to empty again for a fixed time t2.
7. Switch on dryer for time t1.

Figure 15.16 shows the flow chart for the sequence of events to be carried out during the washing machine operation.

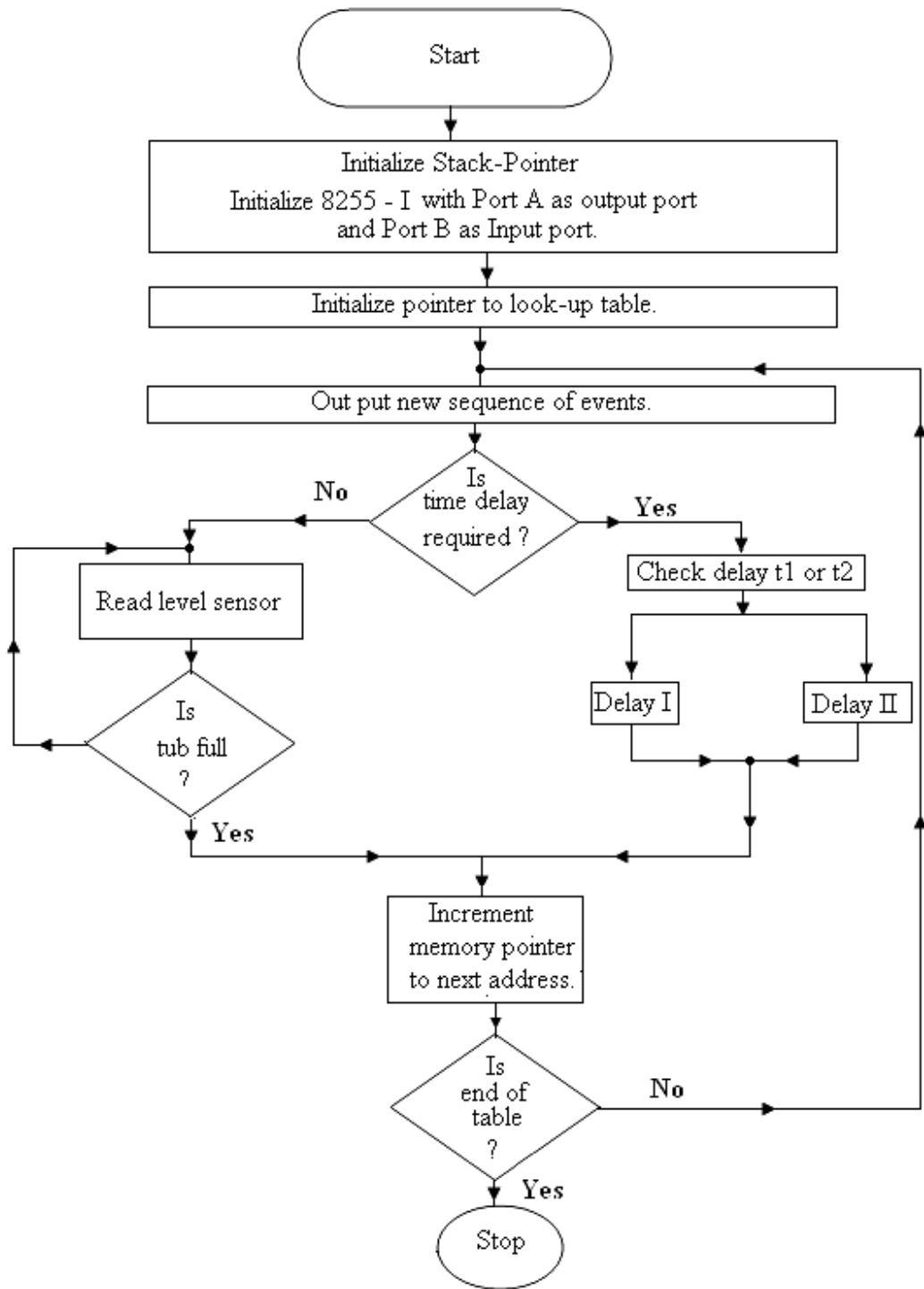
The assembly language program for the above mentioned sequence of events is given below:

| <b>Label</b> | <b>Mnemonics</b>  | <b>Operand</b>   | <b>Comments</b>   |
|--------------|---|--|---|
|              | LXI SP,<br>MVI A,<br>OUT  | XXXX H<br>82 H<br>03 H   | ;Initialize the stack pointer.<br>;Control word for 8255-I.<br>;Works port A of 8255-I as output ports and port B as input port.  |
|              | LXI H,  | 2500 H   | ; Initialize H-L register pair which indicate address of the look-up table.   |
| NEXT 1       | MOV A,<br><br>OUT<br><br>ANI<br>JZ<br><br>ANI<br><br>JZ<br>CALL | M<br><br>00 H<br><br>03 H<br>NEXT<br><br>02 H<br><br>SB<br>DELAY I | ; Get the data from the address of the look-up table.<br>; The accumulator contents go the port A of 8255-I.<br>; Check if delay required.<br>; If the delay is not required jump to NEXT.<br>; Check if delay-II is required.<br>; If yes jump to SB.<br>; Call delay program of t1 seconds. |
| SB           | JMP<br>CALL<br>JMP  | PT1<br>DELAY II<br>PT1   | ; Jump to PT1.<br>; Call delay program of t2.<br>; Jump to PT1.   |
| NEXT         | INX   | H  | ; Get the address of next data from the look-up table.  |
| LOOP         | IN<br><br>SUB   | 01 H<br><br>M  | ; Get the input from the water level sensor.<br>; Check if the water level is equal to the required level.  |
| PT1          | JNZ<br>INX<br><br>MOV A,<br>CPI                                 | LOOP<br>H<br><br>M<br>00 H   | ; If no jump to LOOP.<br>; Get the address of next data from the look-up table.<br>; Data is moved to Acc.<br>; Data is compared with 00.   |

|     |        |                                 |
|-----|--------|---------------------------------|
| JNZ | NEXT 1 | ; If not zero jump to<br>NEXT1. |
| HLT |        | ; Else Halt.                    |

### **Subroutine program for DELAY I:**

| <b>Label</b> | <b>Mnemonics</b> | <b>Operand</b> | <b>Comments</b>  |
|--------------|------------------|----------------|--|
| DELAY I      | LXI D,           | FFFF H         |  |
| LOOP 1       | DCX              | D              | ;Decrement D-E register pair.  |
|              | MOV A,           | D              | ;Moves the contents stored in M <sub>D-E</sub> to Acc.               |
|              | ORA              | E              | ;The contents of A and E registers are ORed bit by bit.              |
|              | JNZ              | LOOP 1         | ;If the result is not zero then jump to DELAY else next instruction. |
|              | RET              |                | ;Return to main program.   |



**Fig. 15.16**

## Subroutine program for DELAY II

| Label    | Mnemonics | Operand | Comments  |
|----------|-----------|---------|---|
| DELAY II | LXI D,    | DEEE H  |   |
| LOOP 2   | DCX       | D       | ; Decrement D-E register pair.  |
|          | MOV A,    | D       | ; Moves the contents stored in M <sub>D-E</sub> to Acc.               |
|          | ORA       | E       | ; The contents of A and E registers are ORed bit by bit.              |
|          | JNZ       | LOOP 2  | ; If the result is not zero then jump to DELAY else next instruction. |
|          | RET       |         | ; Return to main program.   |

## 15.8 MICROPROCESSOR BASED WATER LEVEL CONTROLLER

Water level controller means water is to be pump out from the underground water tank to the another water tank situated on the top of the building for the distribution of water to the other parts of the building. When the upper tank is empty or the level goes below to a certain level, the pump should be automatically switched on. Further, if the tank gets filled up to a certain level (top level) the pump should be switched off. The level of the water inside the upper water tank should be displayed on the seven segment units connected separately with the microprocessor kit. For this the tank to be filled is divided in to 8 equal regions which are say 10 cm apart. So at every 10 cm, a metallic probe is situated.

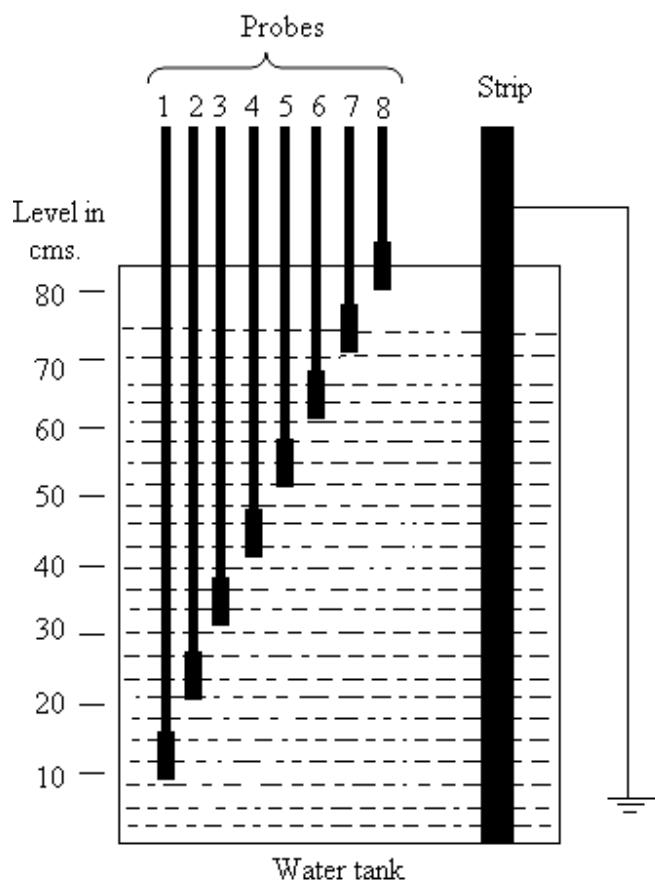
The arrangement of 8 probes which are to be immersed in the water tank is shown in figure 15.17. For the controller 8 metallic probes connected to +5 V d.c. supply through resistance are immersed in the upper tank. The difference in heights between the probes is equal to 10 cm (say). Besides the probe, another metallic strip is also immersed in the tank, which is grounded. The eight probes are also connected to the inputs of 8 inverters as shown in figure 15.18. The outputs of the inverters are connected to 8 bits of port A of 8255-I, which is used as input port. When any of the probes is immersed in the water (or water level is below the probe), the output of the corresponding inverter will provide a logic 0 to its bit of port A. However, if the water level touches the probe or above the probe, then it provides logic 1 to the corresponding bit of port A. The status of water level in the upper tank can directly be read by the microprocessor through the input statement. For example, if only one probe is immersed in the water, then port A of 8255-I will read it 01, similarly for the others. Table 15.4 shows the data to be read out by 8255-I, when different probes are immersed in the water.

When the water level of the tank is below 10 cm, the pump is automatically switched on; however, when its level reaches to 80 cm, the pump will be switched off.

Figure 15.19 shows the connection to the two segment display units connected to the port B of 8255-I for displaying the level of water in the upper tank in centimeters i.e. if the water level is between 1 and 2 probe, then the display unit will display 10, similarly for the other levels (ref. table 15.4).

**Table 15.4**

| Sr.No. | Probe number/ numbers immersed in water | Data to be read out by the microprocessor |                | Level in cms. |
|--------|---|---|----------------|---------------|
|        |   | in Binary                                 | in Hexadecimal |               |
| 1.     | 1                                       | 0000 0001                                 | 01 H           | 10            |
| 2.     | 1 and 2                                 | 0000 0011                                 | 03 H           | 20            |
| 3.     | 1 to 3                                  | 0000 0111                                 | 07 H           | 30            |
| 4.     | 1 to 4                                  | 0000 1111                                 | 0F H           | 40            |
| 5.     | 1 to 5                                  | 0001 1111                                 | 1F H           | 50            |
| 6.     | 1 to 6                                  | 0011 1111                                 | 3F H           | 60            |
| 7.     | 1 to 7                                  | 0111 1111                                 | 7F H           | 70            |
| 8.     | 1 to 8                                  | 1111 1111                                 | FF H           | 80            |



**Fig. 15.17**

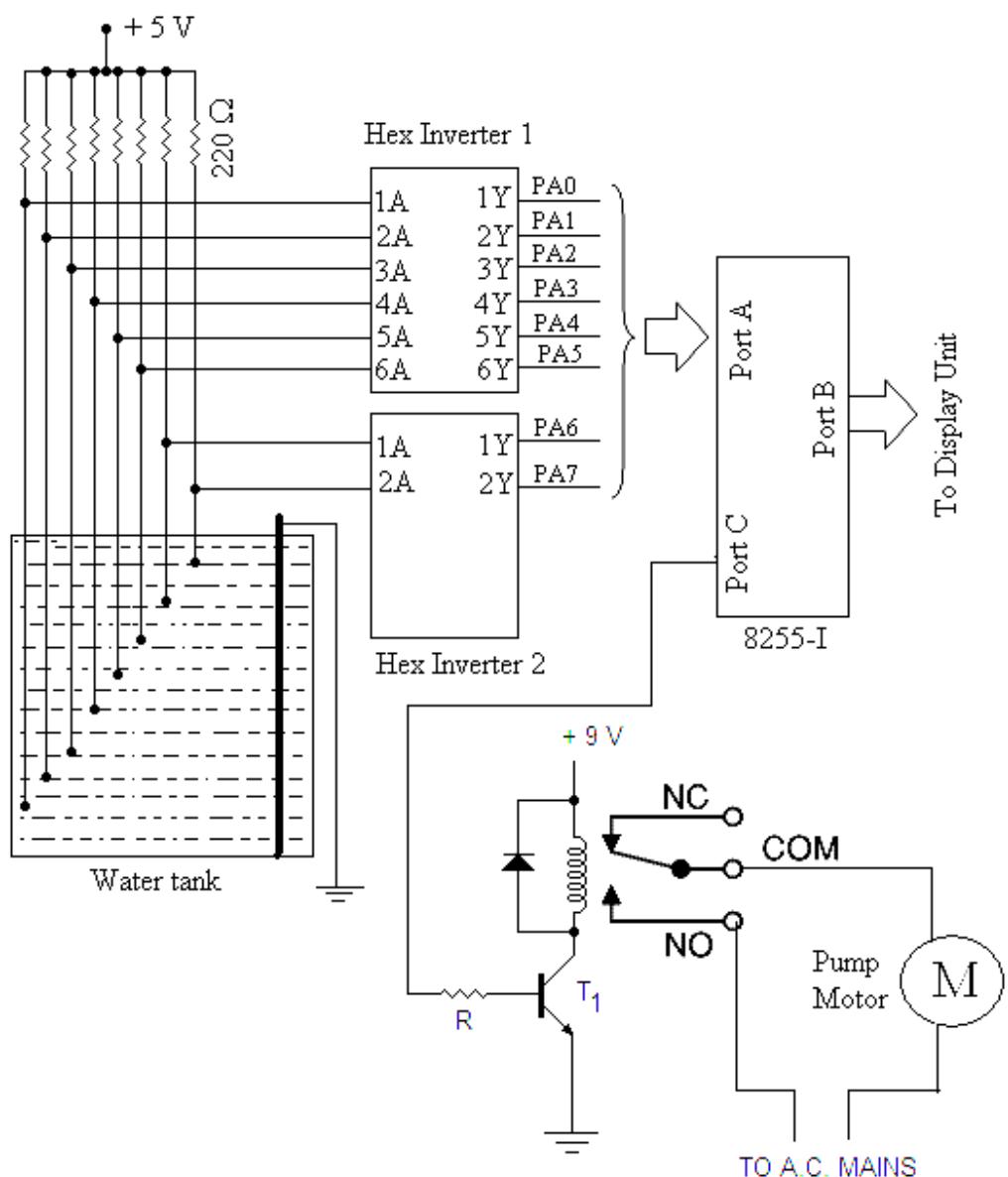
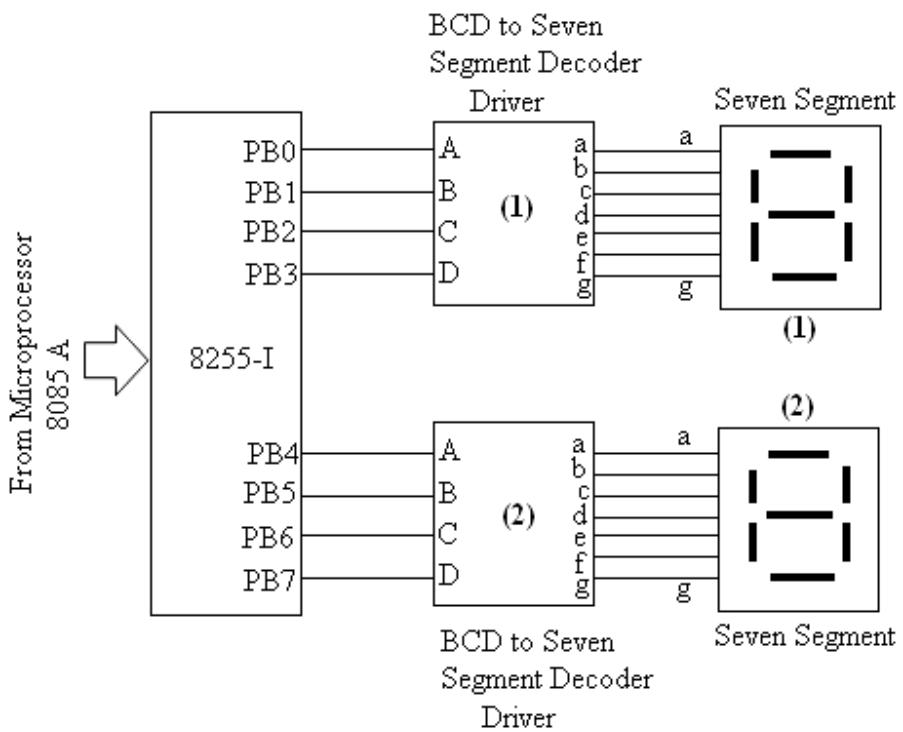


Fig. 15.18



**Fig. 15.19**

The assembly language program for the design of water level controller with the above mentioned conditions is given below, which is self explanatory.

| <b>Label</b> | <b>Mnemonics</b>         | <b>Operand</b>       | <b>Comments</b>   |
|--------------|--------------------------|----------------------|---|
|              | LXI SP,<br>MVI A,<br>OUT | XXXX H<br>90H<br>03H | ;Initialize the stack pointer.<br>;Control word for 8255-I.<br>;Works port A of 8255-I as input port and other ports as output ports. |
| START        | IN                       | 00 H                 | ; Read the water level through input port A.  |
|              | LXI H,                   | 25FF H               | ; Intilise the H-L register pair (starting address of the look up table).   |
|              | MOV C,                   | M                    | ; Data is moved to C-register which is used as counter.   |
| AGAIN        | INX                      | H                    | ; Increment the H-L register pair.  |
|              | CMP                      | M                    | ; Compare the two levels.   |
|              | JZ                       | PT1                  | ; If two levels are equal, then jump to PT1.  |
|              | DCR                      | C                    | ; Decrement the counts.   |

|     |        |       |  |
|-----|--------|-------|--|
|     | JNZ    | AGAIN | ; If counts are not zero then jump to AGAIN.               |
| PT1 | INR    | H     | ; Increment the content of H-register.                     |
|     | MOV A, | M     | ; Get the water level in Accumulator.                      |
|     | OUT    | 01 H  | ; Move it to Port B of 8255-I for the display.             |
|     | CPI    | 10 H  | ; Compare this data with 10 cm.                            |
|     | JNC    | NXT   | ; If the water is more than 10 cm, then move it to NXT.    |
|     | PUSH   | PSW   | ; Else save the Acc contents in the stack.                 |
|     | MVI A, | 01 H  | ; 01 H is loaded to the accumulator.                       |
|     | OUT    | 02 H  | ; PC0 is high (Switch on the motor for lifting the water). |
|     | POP    | PSW   | ; Get back the accumulator contents from the stack.        |
| NXT | CPI    | 80 H  | ; Compared the data with 80 cm.                            |
|     | JC     | START | ; If the data is less than 60, then jump to START.         |
|     | MVI A, | 00 H  | ; 00 H is loaded to the accumulator.                       |
|     | OUT    | 02 H  | ; PC0 is low (Switch off the motor).                       |
|     | CALL   | DELAY | ; Calls delay program.                                     |
|     | JMP    | START | ; Jump to start.   |

Delay Program may be written as discussed in earlier program as per the requirement.

### LOOK UP TABLE

|        |               |
|--------|---------------|
| 25FF H | 09 H (Counts) |
| 2600 H | 00 H          |
| 2601 H | 01 H          |
| 2602 H | 03 H          |
| 2603 H | 07 H          |
| 2604 H | 0F H          |
| 2605 H | 1F H          |
| 2606 H | 3F H          |
| 2607 H | 7F H          |
| 2608 H | FF H          |

### **Water level in Cms.**

|        |      |          |
|--------|------|----------|
| 2700 H | 00 H | (00 Cm.) |
| 2701 H | 10 H | (10 Cm.) |
| 2702 H | 20 H | (20 Cm.) |
| 2703 H | 30 H | (30 Cm.) |
| 2704 H | 40 H | (40 Cm.) |
| 2705 H | 50 H | (50 Cm.) |
| 2706 H | 60 H | (60 Cm.) |
| 2707 H | 70 H | (70 Cm.) |
| 2708 H | 80 H | (80 Cm.) |

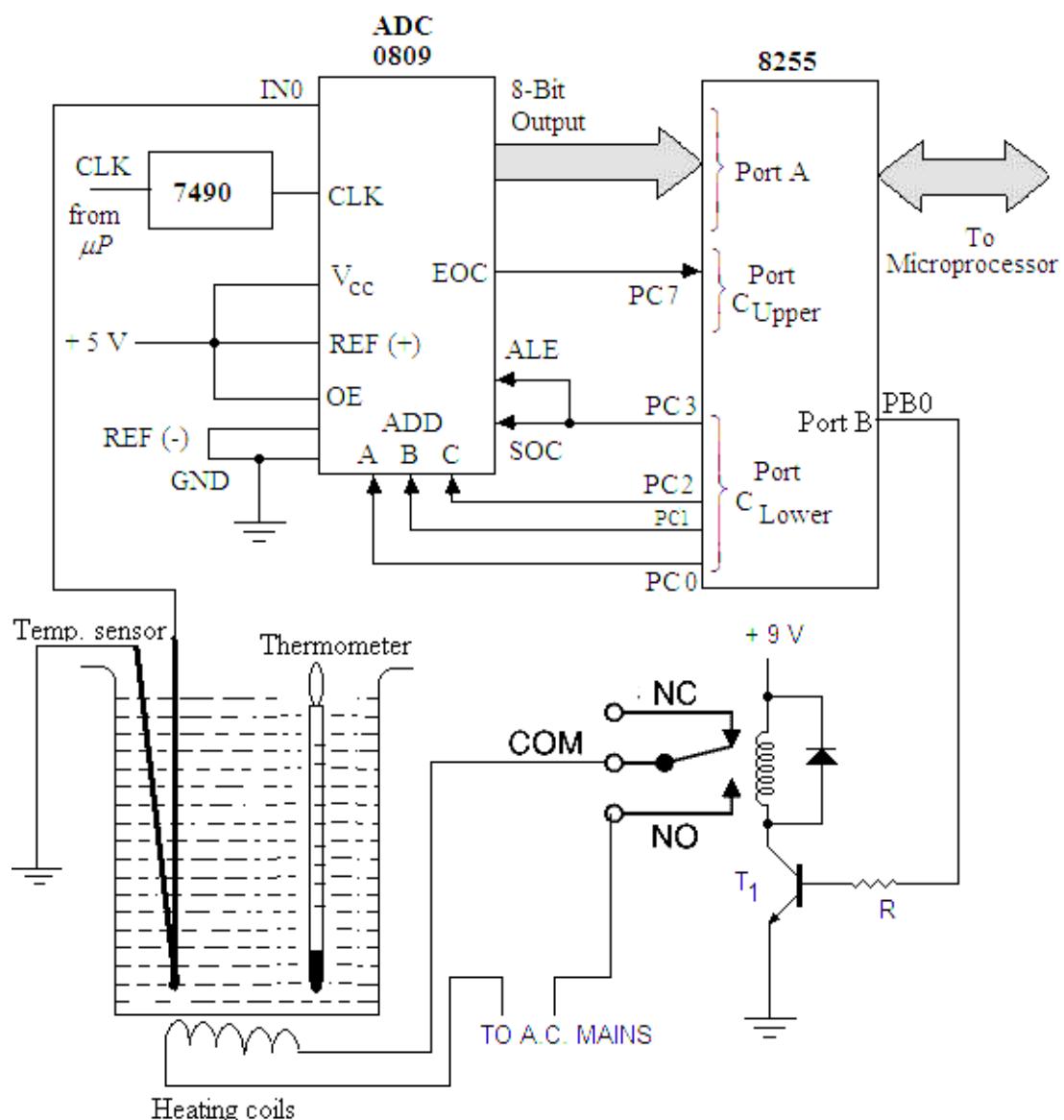
### **15.9 MICROPROCESSOR BASED TEMPERATURE CONTROLLER**

In this section a very simple design of temperature controller is being discussed. The temperature of water bath or of any other device is to be controlled or maintained constant. This is possible if the a.c. mains supplied to the device is switched off if the temperature of the device is more than the required temperature; and it is switched on if the temperature is less than the required temperature. A temperature sensor may be used to sense the temperature of the device. The sensor (say a thermocouple) may provide the thermo e.m.f. (d.c. signal) corresponding to the temperature. The d.c. signal may be converted to the digital signal with the help of A/D converter IC 0809. The output of the A/D converter can be read by the microprocessor through the 8255 PPI.

The arrangement for the control is shown in figure 15.20. Channel 0 of A/D converter is used to convert the sensor voltage to equivalent digital signal. This digital signal can be read out by the microprocessor 8085A through port A (used as input port) of 8255-I provided on the microprocessor kit. Port  $C_{Lower}$  of 8255 is used as output port to send the SOC (Start of Conversion) signal and other bits for the channel selection of A/D converter 0809. For the selection of channel 0 ALE (Address Latch Enable) and SOC are made high by the output port  $C_{Lower}$  of 8255. Port  $C_{Upper}$  of 8255 is used as output port. Before reading the input data (digital value) by the microprocessor, EOC (End of Conversion) signal PC7 is checked for logic 1. The bit PB0 of port B (used as output port) of 8255 may be used to control the heating arrangement of the device through the relay system.

Before running the assembly language program, it is to be calibrated as per the sensor used in this arrangement. Note down the temperature and digital value (in Hexadecimal) which may be stored in some memory locations as a look-up table. This look-up table may be used for the display of the temperature on the address/data field of the microprocessor kit using its monitor program. The digital data of the temperature of the device to be maintained constant, may be saved in a memory location say 2500 H. By comparing the current data (of temperature) with the saved (or required) data, the electrical heating system may be switched on/off.

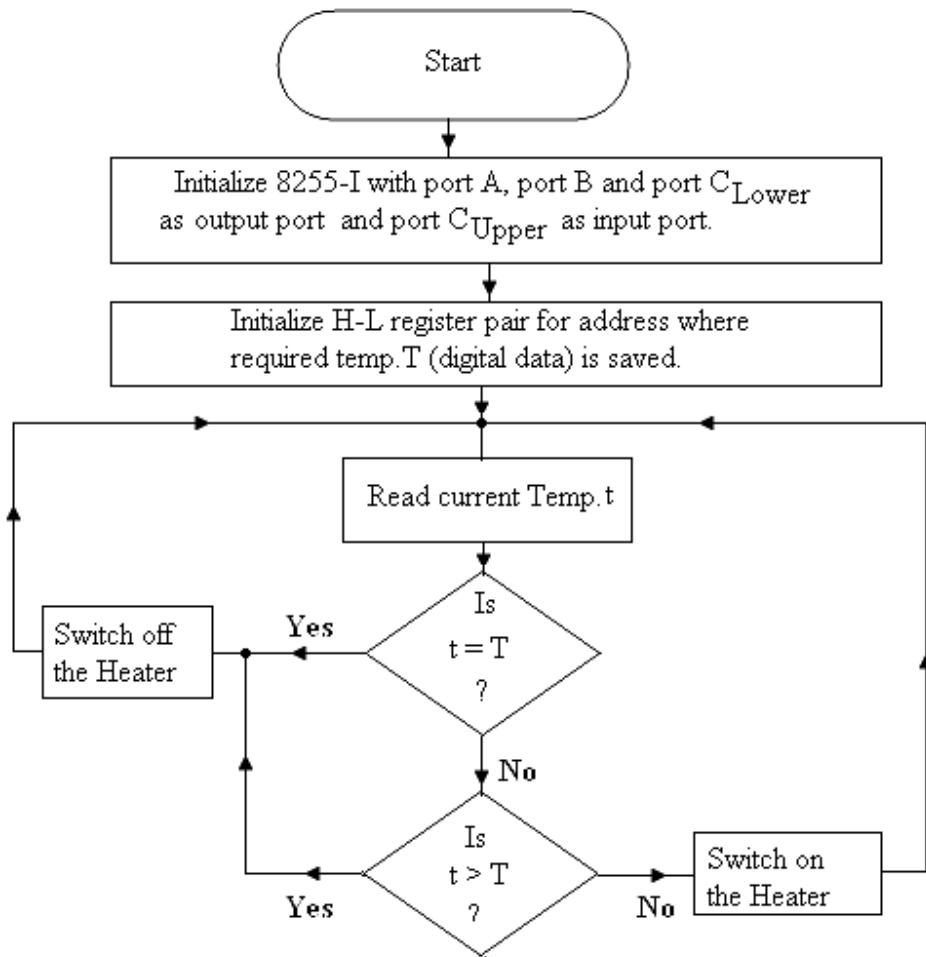
Figure 15.21 shows the flow chart for the controller.



**Fig. 15.20**

### PROGRAM

| Label  | Mnemonics | operand | Comments   |
|--------|-----------|---------|--|
|        | MVI       | A, 98 H | ; Initialize 8255-I to work Port A and Port C <sub>Upper</sub> as input ports, and port C <sub>Lower</sub> as output port. |
|        | OUT       | 03 H    | ; Write the control word in the control word register of 8255-I.   |
| LXI H, |           | 2500 H  | ; Initialize H-L register pair for address where required temp. T is saved.  |



**Fig. 15.21**

|       |     |         |   |
|-------|-----|---------|---|
| START | MVI | A, 00 H | ; Load accumulator with 00H.  |
|       | OUT | 02 H    | ; 00H is sent to Port C <sub>Lower</sub> to select channel 0.   |
|       | MVI | A, 08 H | ; Load 08 H to accumulator.   |
|       | OUT | 02 H    | ; ALE and SOC are enabled (high).   |
|       | MVI | A, 00 H | ; Load 00 H to accumulator.   |
|       | OUT | 02 H    | ; ALE and SOC will be low. A pulse is applied from high to low for the conversion through PC <sub>3</sub> without affecting the channel selected. |
| READ  | IN  | 02 H    | ; Read End of conversion (PC <sub>7</sub> ).  |
|       | RAL |         | ; Rotate left to check if PC <sub>7</sub> is one.   |
|       | JNC | READ    | ; If not 1 READ again.  |
|       | IN  | 00 H    | ; Read digital output of the A/D converter (Temp. t).   |

|       |        |        |   |
|-------|--------|--------|---|
|       | STA    | 2501 H | ; Store the result in 2501 H memory location. |
|       | CMP    | M      | ; Compare the two temperatures for equality.  |
|       | JNZ    | NEXT   | ; If not equal then jump to NEXT.             |
| LOOP1 | MVI A, | 00 H   | ; Switch off the heater.                      |
|       | OUT    | 01 H   | ; PB0 is low.                                 |
|       | JMP    | START  | ; Jump to START.                              |
| NEXT  | SUB    | M      | ; Check if $t > T$ ?                          |
|       | JNC    | LOOP1  | ; If yes then jump to LOOP1.                  |
|       | MVI A, | 01 H   | ; Switch on the heater.                       |
|       | OUT    | 01 H   | ; PB0 is high.                                |
|       | JMP    | START  | ; Jump to START.                              |

## PROBLEMS

1. How will you design the digital clock on the microprocessor kit? Hours and minutes should be displayed on the address field and seconds should be displayed on the data field. Monitor programs of the kit may be used for the display of time. Write program in assembly language of 8085.
2. Design the digital clock which can display the time in 24 hours form on the microprocessor kit. Write program in assembly language of 8085.
3. In problem 1, on time should also be introduced. Write program in assembly language of 8085.
4. Discuss the design of microprocessor based LED dial clock. Write program in assembly language of 8085.
5. Discuss the design of microprocessor based running light in which some LEDs can run in a sequence. Write program in assembly language of 8085.
6. Discuss the design of microprocessor based school bell system. Write program in assembly language of 8085.
7. Discuss the design of microprocessor based Traffic light (simple arrangement). Write program in assembly language of 8085.
8. Discuss the design of microprocessor based Traffic light (complicated arrangement) in which traffic is allowed in left-right direction also. Write program in assembly language of 8085.
9. In the above design of microprocessor based traffic light, the delay time should also be displayed on the address and data field of the microprocessor kit. Write program in assembly language of 8085.
10. Discuss the design of microprocessor based Stepper Motor controller. Write program in assembly language of 8085.
11. Write program in assembly language of 8085 to control the different function of washing machine.

12. Discuss the water level controller which can pump out the water from the underground tank to the tank placed on the roof of the building. The motor of the pump should be switched on if the water level in the upper tank is less than certain level. The pump should be switched off when the water level in the upper tank is completely filled. Program for same should be written in assembly language of the 8085 microprocessor.
  13. Discuss the design of microprocessor based Temperature controller which can maintain the temperature of a device constant. Write program in assembly language of 8085.
-

## Appendix - I

### Instruction codes of 8085 Microprocessor in Alphabetical Order

| Mne-monics | Ope-<br>rand | Op<br>code | Bytes | T-<br>States | Status |   |   |   |   | Flags<br>Affected |
|------------|--------------|------------|-------|--------------|--------|---|---|---|---|-------------------|
|            |              |            |       |              | C      | Y | A | C | Z |                   |
| ACI        | Data         | CE         | 2     | 7            | x      | x | x | x | x | All               |
| ADC        | A            | 8F         | 1     | 4            | x      | x | x | x | x | All               |
| ADC        | B            | 88         | 1     | 4            | x      | x | x | x | x | All               |
| ADC        | C            | 89         | 1     | 4            | x      | x | x | x | x | All               |
| ADC        | D            | 8A         | 1     | 4            | x      | x | x | x | x | All               |
| ADC        | E            | 8B         | 1     | 4            | x      | x | x | x | x | All               |
| ADC        | H            | 8C         | 1     | 4            | x      | x | x | x | x | All               |
| ADC        | L            | 8D         | 1     | 4            | x      | x | x | x | x | All               |
| ADC        | M            | 8E         | 1     | 7            | x      | x | x | x | x | All               |
| ADD        | A            | 87         | 1     | 4            | x      | x | x | x | x | All               |
| ADD        | B            | 80         | 1     | 4            | x      | x | x | x | x | All               |
| ADD        | C            | 81         | 1     | 4            | x      | x | x | x | x | All               |
| ADD        | D            | 82         | 1     | 4            | x      | x | x | x | x | All               |
| ADD        | E            | 83         | 1     | 4            | x      | x | x | x | x | All               |
| ADD        | H            | 84         | 1     | 4            | x      | x | x | x | x | All               |
| ADD        | L            | 85         | 1     | 4            | x      | x | x | x | x | All               |
| ADD        | M            | 86         | 1     | 7            | x      | x | x | x | x | All               |
| ADI        | Data         | C6         | 2     | 7            | x      | x | x | x | x | All               |

| Mne-monics | Ope-<br>rand | Op<br>code | Bytes | T-<br>States | Status<br>CY AC Z S P | Flags<br>affected  |
|------------|--------------|------------|-------|--------------|-----------------------|--------------------|
| ANA        | A            | A7         | 1     | 4            | 0 1 x x x             | CY is 0<br>AC is 1 |
| ANA        | B            | A0         | 1     | 4            | 0 1 x x x             | CY is 0<br>AC is 1 |
| ANA        | C            | A1         | 1     | 4            | 0 1 x x x             | CY is 0<br>AC is 1 |
| ANA        | D            | A2         | 1     | 4            | 0 1 x x x             | CY is 0<br>AC is 1 |
| ANA        | E            | A3         | 1     | 4            | 0 1 x x x             | CY is 0<br>AC is 1 |
| ANA        | H            | A4         | 1     | 4            | 0 1 x x x             | CY is 0<br>AC is 1 |
| ANA        | L            | A5         | 1     | 4            | 0 1 x x x             | CY is 0<br>AC is 1 |
| ANA        | M            | A6         | 1     | 7            | 0 1 x x x             | CY is 0<br>AC is 1 |
| ANI        | Data         | E6         | 2     | 7            | 0 1 x x x             | CY is 0<br>AC is 1 |
| CALL       | Label        | CD         | 3     | 18           | — — — — —             | None               |
| CC         | Label        | DC         | 3     | 18/9         | — — — — —             | None               |
| CM         | Label        | FC         | 3     | 18/9         | — — — — —             | None               |
| CMA        |              | 2F         | 1     | 4            | — — — — —             | None               |
| CMC        |              | 3F         | 1     | 4            | x — — — —             | Only<br>CY         |
| CMP        | A            | BF         | 1     | 4            | x x x x x             | All                |
| CMP        | B            | B8         | 1     | 4            | x x x x x             | All                |
| CMP        | C            | B9         | 1     | 4            | x x x x x             | All                |
| CMP        | D            | BA         | 1     | 4            | x x x x x             | All                |
| CMP        | E            | BB         | 1     | 4            | x x x x x             | All                |
| CMP        | H            | BC         | 1     | 4            | x x x x x             | All                |
| CMP        | L            | BD         | 1     | 4            | x x x x x             | All                |
| CMP        | M            | BE         | 1     | 7            | x x x x x             | All                |

| Mne-monics | Ope-<br>rand | Op<br>code | Bytes | T-<br>States | Status |    |   |   |   | Flags<br>Affected |
|------------|--------------|------------|-------|--------------|--------|----|---|---|---|-------------------|
|            |              |            |       |              | CY     | AC | Z | S | P |                   |
| CNC        | Label        | D4         | 3     | 18/9         | —      | —  | — | — | — | None              |
| CNZ        | Label        | C4         | 3     | 18/9         | —      | —  | — | — | — | None              |
| CP         | Label        | F4         | 3     | 18/9         | —      | —  | — | — | — | None              |
| CPE        | Label        | EC         | 3     | 18/9         | —      | —  | — | — | — | None              |
| CPI        | Data         | FE         | 2     | 7            | x      | x  | x | x | x | All               |
| CPO        | Label        | E4         | 3     | 18/9         | —      | —  | — | — | — | None              |
| CZ         | Label        | CC         | 3     | 18/9         | —      | —  | — | — | — | None              |
| DAA        |              | 27         | 1     | 4            | x      | x  | x | x | x | All               |
| DAD        | B            | 09         | 1     | 10           | x      | —  | — | — | — | Only CY flag      |
| DAD        | D            | 19         | 1     | 10           | x      | —  | — | — | — | Only CY flag      |
| DAD        | H            | 29         | 1     | 10           | x      | —  | — | — | — | Only CY flag      |
| DAD        | SP           | 39         | 1     | 10           | x      | —  | — | — | — | Only CY flag      |
| DCR        | A            | 3D         | 1     | 4            | —      | x  | x | x | x | All Expt. CY      |
| DCR        | B            | 05         | 1     | 4            | —      | x  | x | x | x | All Expt. CY      |
| DCR        | C            | 0D         | 1     | 4            | —      | x  | x | x | x | All Expt. CY      |
| DCR        | D            | 15         | 1     | 4            | —      | x  | x | x | x | All Expt. CY      |
| DCR        | E            | 1D         | 1     | 4            | —      | x  | x | x | x | All Expt. CY      |
| DCR        | H            | 25         | 1     | 4            | —      | x  | x | x | x | All Expt. CY      |
| DCR        | L            | 2D         | 1     | 4            | —      | x  | x | x | x | All Expt. CY      |
| DCR        | M            | 35         | 1     | 10           | —      | x  | x | x | x | All Expt. CY      |
| DCX        | B            | 0B         | 1     | 6            | —      | —  | — | — | — | None              |
| DCX        | D            | 1B         | 1     | 6            | —      | —  | — | — | — | None              |

| Mne-monics | Ope-<br>rand  | Op<br>code | Bytes | T-<br>States | Status<br>CY AC Z S P | Flags<br>affected |
|------------|---------------|------------|-------|--------------|-----------------------|-------------------|
| DCX        | H             | 2B         | 1     | 6            | — — — — — —           | None              |
| DCX        | SP            | 3B         | 1     | 6            | — — — — — —           | None              |
| DI         |               | F3         | 1     | 4            | — — — — — —           | None              |
| EI         |               | FB         | 1     | 4            | — — — — — —           | None              |
| HLT        |               | 76         | 1     | 5            | — — — — — —           | None              |
| IN         | Port<br>Addr. | DB         | 2     | 10           | — — — — — —           | None              |
| INR        | A             | 3C         | 1     | 4            | — X X X X             | All Expt.<br>CY   |
| INR        | B             | 04         | 1     | 4            | — X X X X             | All Expt.<br>CY   |
| INR        | C             | 0C         | 1     | 4            | — X X X X             | All Expt.<br>CY   |
| INR        | D             | 14         | 1     | 4            | — X X X X             | All Expt.<br>CY   |
| INR        | E             | 1C         | 1     | 4            | — X X X X             | All Expt.<br>CY   |
| INR        | H             | 24         | 1     | 4            | — X X X X             | All Expt.<br>CY   |
| INR        | L             | 2C         | 1     | 4            | — X X X X             | All Expt.<br>CY   |
| INR        | M             | 34         | 1     | 10           | — X X X X             | All Expt.<br>CY   |
| INX        | B             | 03         | 1     | 6            | — — — — — —           | None              |
| INX        | D             | 13         | 1     | 6            | — — — — — —           | None              |
| INX        | H             | 23         | 1     | 6            | — — — — — —           | None              |
| INX        | SP            | 33         | 1     | 6            | — — — — — —           | None              |
| JC         | Label         | DA         | 3     | 10/7         | — — — — — —           | None              |
| JM         | Label         | FA         | 3     | 10/7         | — — — — — —           | None              |
| JMP        | Label         | C3         | 3     | 10           | — — — — — —           | None              |
| JNC        | Label         | D2         | 3     | 10/7         | — — — — — —           | None              |

| Mne-monics | Oper-<br>rand | Op<br>code | Bytes | T-<br>States | Status<br>CY AC Z S P | Flags<br>affected |
|------------|---------------|------------|-------|--------------|-----------------------|-------------------|
| JNZ        | Label         | C2         | 3     | 10/7         | — — — — —             | None              |
| JP         | Label         | F2         | 3     | 10/7         | — — — — —             | None              |
| JPE        | Label         | EA         | 3     | 10/7         | — — — — —             | None              |
| JPO        | Label         | E2         | 3     | 10/7         | — — — — —             | None              |
| JZ         | Label         | CA         | 3     | 10/7         | — — — — —             | None              |
| LDA        | Addr.         | 3A         | 3     | 13           | — — — — —             | None              |
| LDAX       | B             | 0A         | 1     | 7            | — — — — —             | None              |
| LDAX       | D             | 1A         | 1     | 7            | — — — — —             | None              |
| LHLD       | Addr.         | 2A         | 3     | 16           | — — — — —             | None              |
| LXI        | B             | 01         | 3     | 10           | — — — — —             | None              |
| LXI        | D             | 11         | 3     | 10           | — — — — —             | None              |
| LXI        | H             | 21         | 3     | 10           | — — — — —             | None              |
| LXI        | SP            | 31         | 3     | 10           | — — — — —             | None              |
| MOV        | A, A          | 7F         | 1     | 4            | — — — — —             | None              |
| MOV        | A, B          | 78         | 1     | 4            | — — — — —             | None              |
| MOV        | A, C          | 79         | 1     | 4            | — — — — —             | None              |
| MOV        | A, D          | 7A         | 1     | 4            | — — — — —             | None              |
| MOV        | A, E          | 7B         | 1     | 4            | — — — — —             | None              |
| MOV        | A, H          | 7C         | 1     | 4            | — — — — —             | None              |
| MOV        | A, L          | 7D         | 1     | 4            | — — — — —             | None              |
| MOV        | A, M          | 7E         | 1     | 7            | — — — — —             | None              |
| MOV        | B, A          | 47         | 1     | 4            | — — — — —             | None              |

| Mne-monics | Ope-rand | Op code | Bytes | T-States | Status CY AC Z S P | Flags affected |
|------------|----------|---------|-------|----------|--------------------|----------------|
| MOV        | B, B     | 40      | 1     | 4        | — — — — —          | None           |
| MOV        | B, C     | 41      | 1     | 4        | — — — — —          | None           |
| MOV        | B, D     | 42      | 1     | 4        | — — — — —          | None           |
| MOV        | B, E     | 43      | 1     | 4        | — — — — —          | None           |
| MOV        | B, H     | 44      | 1     | 4        | — — — — —          | None           |
| MOV        | B, L     | 45      | 1     | 4        | — — — — —          | None           |
| MOV        | B, M     | 46      | 1     | 7        | — — — — —          | None           |
| MOV        | C, A     | 4F      | 1     | 4        | — — — — —          | None           |
| MOV        | C, B     | 48      | 1     | 4        | — — — — —          | None           |
| MOV        | C, C     | 49      | 1     | 4        | — — — — —          | None           |
| MOV        | C, D     | 4A      | 1     | 4        | — — — — —          | None           |
| MOV        | C, E     | 4B      | 1     | 4        | — — — — —          | None           |
| MOV        | C, H     | 4C      | 1     | 4        | — — — — —          | None           |
| MOV        | C, L     | 4D      | 1     | 4        | — — — — —          | None           |
| MOV        | C, M     | 4E      | 1     | 7        | — — — — —          | None           |
| MOV        | D, A     | 57      | 1     | 4        | — — — — —          | None           |
| MOV        | D, B     | 50      | 1     | 4        | — — — — —          | None           |
| MOV        | D, C     | 51      | 1     | 4        | — — — — —          | None           |
| MOV        | D, D     | 52      | 1     | 4        | — — — — —          | None           |
| MOV        | D, E     | 53      | 1     | 4        | — — — — —          | None           |
| MOV        | D, H     | 54      | 1     | 4        | — — — — —          | None           |
| MOV        | D, L     | 55      | 1     | 4        | — — — — —          | None           |

| Mne-monics | Ope-<br>rand | Op<br>code | Bytes | T-<br>States | Status<br>CY AC Z S P | Flags<br>affected |
|------------|--------------|------------|-------|--------------|-----------------------|-------------------|
| MOV        | D, M         | 56         | 1     | 7            | — — — — — —           | None              |
| MOV        | E, A         | 5F         | 1     | 4            | — — — — — —           | None              |
| MOV        | E, B         | 58         | 1     | 4            | — — — — — —           | None              |
| MOV        | E, C         | 59         | 1     | 4            | — — — — — —           | None              |
| MOV        | E, D         | 5A         | 1     | 4            | — — — — — —           | None              |
| MOV        | E, E         | 5B         | 1     | 4            | — — — — — —           | None              |
| MOV        | E, H         | 5C         | 1     | 7            | — — — — — —           | None              |
| MOV        | E, L         | 5D         | 1     | 4            | — — — — — —           | None              |
| MOV        | E, M         | 5E         | 1     | 7            | — — — — — —           | None              |
| MOV        | H, A         | 67         | 1     | 4            | — — — — — —           | None              |
| MOV        | H, B         | 60         | 1     | 4            | — — — — — —           | None              |
| MOV        | H, C         | 61         | 1     | 4            | — — — — — —           | None              |
| MOV        | H, D         | 62         | 1     | 4            | — — — — — —           | None              |
| MOV        | H, E         | 63         | 1     | 4            | — — — — — —           | None              |
| MOV        | H, H         | 64         | 1     | 4            | — — — — — —           | None              |
| MOV        | H, L         | 65         | 1     | 4            | — — — — — —           | None              |
| MOV        | H, M         | 66         | 1     | 7            | — — — — — —           | None              |
| MOV        | L, A         | 6F         | 1     | 4            | — — — — — —           | None              |
| MOV        | L, B         | 68         | 1     | 4            | — — — — — —           | None              |
| MOV        | L, C         | 69         | 1     | 4            | — — — — — —           | None              |
| MOV        | L, D         | 6A         | 1     | 4            | — — — — — —           | None              |
| MOV        | L, E         | 6B         | 1     | 4            | — — — — — —           | None              |

| Mne-monics | Ope-<br>rand | Op<br>code | Bytes | T-<br>States | Status<br>CY AC Z S P | Flags<br>affected  |
|------------|--------------|------------|-------|--------------|-----------------------|--------------------|
| MOV        | L, H         | 6C         | 1     | 4            | — — — — —             | None               |
| MOV        | L, L         | 6D         | 1     | 4            | — — — — —             | None               |
| MOV        | L, M         | 6E         | 1     | 7            | — — — — —             | None               |
| MOV        | M, A         | 77         | 1     | 7            | — — — — —             | None               |
| MOV        | M, B         | 70         | 1     | 7            | — — — — —             | None               |
| MOV        | M, C         | 71         | 1     | 7            | — — — — —             | None               |
| MOV        | M, D         | 72         | 1     | 7            | — — — — —             | None               |
| MOV        | M, E         | 73         | 1     | 7            | — — — — —             | None               |
| MOV        | M, H         | 74         | 1     | 7            | — — — — —             | None               |
| MOV        | M, L         | 75         | 1     | 7            | — — — — —             | None               |
| MVI        | A, data      | 3E         | 2     | 7            | — — — — —             | None               |
| MVI        | B, data      | 06         | 2     | 7            | — — — — —             | None               |
| MVI        | C, data      | 0E         | 2     | 7            | — — — — —             | None               |
| MVI        | D, data      | 16         | 2     | 7            | — — — — —             | None               |
| MVI        | E, data      | 1E         | 2     | 7            | — — — — —             | None               |
| MVI        | H, data      | 26         | 2     | 7            | — — — — —             | None               |
| MVI        | L, data      | 2E         | 2     | 7            | — — — — —             | None               |
| MVI        | M, data      | 36         | 2     | 10           | — — — — —             | None               |
| NOP        |              | 00         | 1     | 4            | — — — — —             | None               |
| ORA        | A            | B7         | 1     | 4            | 0 0 x x x             | CY is 0<br>AC is 0 |
| ORA        | B            | B0         | 1     | 4            | 0 0 x x x             | CY is 0<br>AC is 0 |
| ORA        | C            | B1         | 1     | 4            | 0 0 x x x             | CY is 0<br>AC is 0 |

| Mne-monics | Ope-<br>rand  | Op<br>code | Bytes | T-<br>States | Status<br>CY AC Z S P | Flags<br>affected  |
|------------|---------------|------------|-------|--------------|-----------------------|--------------------|
| ORA        | D             | B2         | 1     | 4            | 0 0 x x x             | CY is 0<br>AC is 0 |
| ORA        | E             | B3         | 1     | 4            | 0 0 x x x             | CY is 0<br>AC is 0 |
| ORA        | H             | B4         | 1     | 4            | 0 0 x x x             | CY is 0<br>AC is 0 |
| ORA        | L             | B5         | 1     | 4            | 0 0 x x x             | CY is 0<br>AC is 0 |
| ORA        | M             | B6         | 1     | 7            | 0 0 x x x             | CY is 0<br>AC is 0 |
| ORI        | Data          | F6         | 2     | 7            | 0 0 x x x             | CY is 0<br>AC is 0 |
| OUT        | Port<br>Addr. | D3         | 2     | 10           | — — — — —             | None               |
| PCHL       |               | E9         | 1     | 6            | — — — — —             | None               |
| POP        | B             | C1         | 1     | 10           | — — — — —             | None               |
| POP        | D             | D1         | 1     | 10           | — — — — —             | None               |
| POP        | H             | E1         | 1     | 10           | — — — — —             | None               |
| POP        | PSW           | F1         | 1     | 10           | x x x x x             | All                |
| PUSH       | B             | C5         | 1     | 12           | — — — — —             | None               |
| PUSH       | D             | D5         | 1     | 12           | — — — — —             | None               |
| PUSH       | H             | E5         | 1     | 12           | — — — — —             | None               |
| PUSH       | PSW           | F5         | 1     | 12           | x x x x x             | All                |
| RAL        |               | 17         | 1     | 4            | x — — — —             | Only<br>CY         |
| RAR        |               | 1F         | 1     | 4            | x — — — —             | Only<br>CY         |
| RC         |               | D8         | 1     | 12/6         | — — — — —             | None               |
| RET        |               | C9         | 1     | 10           | — — — — —             | None               |
| RIM        |               | 20         | 1     | 4            | — — — — —             | None               |
| RLC        |               | 07         | 1     | 4            | x — — — —             | Only<br>CY         |

| Mne-monics | Ope-<br>rand | Op<br>code | Bytes | T-<br>States | Status<br>CY AC Z S P | Flags<br>affected |
|------------|--------------|------------|-------|--------------|-----------------------|-------------------|
| RM         |              | F8         | 1     | 12/6         | — — — — — —           | None              |
| RNC        |              | D0         | 1     | 12/6         | — — — — — —           | None              |
| RNZ        |              | C0         | 1     | 12/6         | — — — — — —           | None              |
| RP         |              | F0         | 1     | 12/6         | — — — — — —           | None              |
| RPE        |              | E8         | 1     | 12/6         | — — — — — —           | None              |
| RPO        |              | E0         | 1     | 12/6         | — — — — — —           | None              |
| RRC        |              | 0F         | 1     | 4            | x — — — —             | Only CY           |
| RST        | 0            | C7         | 1     | 12           | — — — — — —           | None              |
| RST        | 1            | CF         | 1     | 12           | — — — — — —           | None              |
| RST        | 2            | D7         | 1     | 12           | — — — — — —           | None              |
| RST        | 3            | DF         | 1     | 12           | — — — — — —           | None              |
| RST        | 4            | E7         | 1     | 12           | — — — — — —           | None              |
| RST        | 5            | EF         | 1     | 12           | — — — — — —           | None              |
| RST        | 6            | F7         | 1     | 12           | — — — — — —           | None              |
| RST        | 7            | FF         | 1     | 12           | — — — — — —           | None              |
| RZ         |              | C8         | 1     | 12/6         | — — — — — —           | None              |
| SBB        | A            | 9F         | 1     | 4            | x x x x x             | All               |
| SBB        | B            | 98         | 1     | 4            | x x x x x             | All               |
| SBB        | C            | 99         | 1     | 4            | x x x x x             | All               |
| SBB        | D            | 9A         | 1     | 4            | x x x x x             | All               |
| SBB        | E            | 9B         | 1     | 4            | x x x x x             | All               |
| SBB        | H            | 9C         | 1     | 4            | x x x x x             | All               |

| Mne-monics | Ope-<br>rand | Op<br>code | Bytes | T-<br>States | Status<br>CY AC Z S P | Flags<br>affected |
|------------|--------------|------------|-------|--------------|-----------------------|-------------------|
| SBB        | L            | 9D         | 1     | 4            | x x x x x             | All               |
| SBB        | M            | 9E         | 1     | 7            | x x x x x             | All               |
| SBI        | Data         | DE         | 2     | 7            | x x x x x             | All               |
| SHLD       | Addr.        | 22         | 3     | 16           | — — — — —             | None              |
| SIM        |              | 30         | 1     | 4            | — — — — —             | None              |
| SPHL       |              | F9         | 1     | 6            | — — — — —             | None              |
| STA        | Addr.        | 32         | 3     | 13           | — — — — —             | None              |
| STAX       | B            | 02         | 1     | 7            | — — — — —             | None              |
| STAX       | D            | 12         | 1     | 7            | — — — — —             | None              |
| STC        |              | 37         | 1     | 4            | x — — — —             | Only CY           |
| SUB        | A            | 97         | 1     | 4            | x x x x x             | All               |
| SUB        | B            | 90         | 1     | 4            | x x x x x             | All               |
| SUB        | C            | 91         | 1     | 4            | x x x x x             | All               |
| SUB        | D            | 92         | 1     | 4            | x x x x x             | All               |
| SUB        | E            | 93         | 1     | 4            | x x x x x             | All               |
| SUB        | H            | 94         | 1     | 4            | x x x x x             | All               |
| SUB        | L            | 95         | 1     | 4            | x x x x x             | All               |
| SUB        | M            | 96         | 1     | 7            | x x x x x             | All               |
| SUI        | Data         | D6         | 2     | 7            | x x x x x             | All               |
| XCHG       |              | EB         | 1     | 4            | — — — — —             | None              |
| XRA        | A            | AF         | 1     | 4            | 0 0 x x x             | CY & AC Zero      |
| XRA        | B            | A8         | 1     | 4            | 0 0 x x x             | CY & AC Zero      |

| Mne-monics | Ope-<br>rand | Op<br>code | Bytes | T-<br>States | Status |    |   |   |   | Flags<br>affected |
|------------|--------------|------------|-------|--------------|--------|----|---|---|---|-------------------|
|            |              |            |       |              | CY     | AC | Z | S | P |                   |
| XRA        | C            | A9         | 1     | 4            | 0      | 0  | x | x | x | CY & AC Zero      |
| XRA        | D            | AA         | 1     | 4            | 0      | 0  | x | x | x | CY & AC Zero      |
| XRA        | E            | AB         | 1     | 4            | 0      | 0  | x | x | x | CY & AC Zero      |
| XRA        | H            | AC         | 1     | 4            | 0      | 0  | x | x | x | CY & AC Zero      |
| XRA        | L            | AD         | 1     | 4            | 0      | 0  | x | x | x | CY & AC Zero      |
| XRA        | M            | AE         | 1     | 7            | 0      | 0  | x | x | x | CY & AC Zero      |
| XRI        | Data         | EE         | 2     | 7            | 0      | 0  | x | x | x | CY & AC Zero      |
| XTHL       |              | E3         | 1     | 16           | —      | —  | — | — | — | None              |

## Appendix - II

### Instruction codes of 8085 Microprocessor in Hexadecimal Order

| <b>Hex Code</b> | <b>Mnemonics</b> | <b>Hex Code</b> | <b>Mnemonics</b> |
|-----------------|------------------|-----------------|------------------|
| 00              | NOP              | 27              | DAA              |
| 01              | LXI B            | 28              | ---              |
| 02              | STAX B           | 29              | DAD H            |
| 03              | INX B            | 2A              | LHLD             |
| 04              | INR B            | 2B              | DCX H            |
| 05              | DCR B            | 2C              | INR L            |
| 06              | MVI B            | 2D              | DCR L            |
| 07              | RLC              | 2E              | MVI L            |
| 08              | ---              | 2F              | CMA              |
| 09              | DAD B            | 30              | SIM              |
| 0A              | LDAX B           | 31              | LXI SP           |
| 0B              | DCX B            | 32              | STA              |
| 0C              | INR C            | 33              | INX SP           |
| 0D              | DCR C            | 34              | INR M            |
| 0E              | MVI C            | 35              | DCR M            |
| 0F              | RRC              | 36              | MVI M            |
| 10              | ---              | 37              | STC              |
| 11              | LXI D            | 38              | ---              |
| 12              | STAX D           | 39              | DAD SP           |
| 13              | INX D            | 3A              | LDA              |
| 14              | INR D            | 3B              | DCX SP           |
| 15              | DCR D            | 3C              | INR A            |
| 16              | MVI D            | 3D              | DCR A            |
| 17              | RAL              | 3E              | MVI A            |
| 18              | ---              | 3F              | CMC              |
| 19              | DAD D            | 40              | MOV B, B         |
| 1A              | LDAX D           | 41              | MOV B, C         |
| 1B              | DCX D            | 42              | MOV B, D         |
| 1C              | INR E            | 43              | MOV B, E         |
| 1D              | DCR E            | 44              | MOV B, H         |
| 1E              | MVI E            | 45              | MOV B, L         |
| 1F              | RAR              | 46              | MOV B, M         |
| 20              | RIM              | 47              | MOV B, A         |
| 21              | LXI H            | 48              | MOV C, B         |
| 22              | SHLD             | 49              | MOV C, C         |
| 23              | INX H            | 4A              | MOV C, D         |
| 24              | INR H            | 4B              | MOV C, E         |
| 25              | DCR H            | 4C              | MOV C, H         |

|    |       |    |          |
|----|-------|----|----------|
| 26 | MVI H | 4D | MOV C, L |
|----|-------|----|----------|

| Hex Code | Mnemonics | Hex Code | Mnemonics |
|----------|-----------|----------|-----------|
| 4E       | MOV C, M  | 75       | MOV M, L  |
| 4F       | MOV C, A  | 76       | HLT       |
| 50       | MOV D, B  | 77       | MOV M, A  |
| 51       | MOV D, C  | 78       | MOV A, B  |
| 52       | MOV D, D  | 79       | MOV A, C  |
| 53       | MOV D, E  | 7A       | MOV A, D  |
| 54       | MOV D, H  | 7B       | MOV A, E  |
| 55       | MOV D, L  | 7C       | MOV A, H  |
| 56       | MOV D, M  | 7D       | MOV A, L  |
| 57       | MOV D, A  | 7E       | MOV A, M  |
| 58       | MOV E, B  | 7F       | MOV A, A  |
| 59       | MOV E, C  | 80       | ADD B     |
| 5A       | MOV E, D  | 81       | ADD C     |
| 5B       | MOV E, E  | 82       | ADD D     |
| 5C       | MOV E, H  | 83       | ADD E     |
| 5D       | MOV E, L  | 84       | ADD H     |
| 5E       | MOV E, M  | 85       | ADD L     |
| 5F       | MOV E, A  | 86       | ADD M     |
| 60       | MOV H, B  | 87       | ADD A     |
| 61       | MOV H, C  | 88       | ADC B     |
| 62       | MOV H, D  | 89       | ADC C     |
| 63       | MOV H, E  | 8A       | ADC D     |
| 64       | MOV H, H  | 8B       | ADC E     |
| 65       | MOV H, L  | 8C       | ADC H     |
| 66       | MOV H, M  | 8D       | ADC L     |
| 67       | MOV H, A  | 8E       | ADC M     |
| 68       | MOV L, B  | 8F       | ADC A     |
| 69       | MOV L, C  | 90       | SUB B     |
| 6A       | MOV L, D  | 91       | SUB C     |
| 6B       | MOV L, E  | 92       | SUB D     |
| 6C       | MOV L, H  | 93       | SUB E     |
| 6D       | MOV L, L  | 94       | SUB H     |
| 6E       | MOV L, M  | 95       | SUB L     |
| 6F       | MOV L, A  | 96       | SUB M     |
| 70       | MOV M, B  | 97       | SUB A     |
| 71       | MOV M, C  | 98       | SBB B     |
| 72       | MOV M, D  | 99       | SBB C     |
| 73       | MOV M, E  | 9A       | SBB D     |
| 74       | MOV M, H  | 9B       | SBB E     |

| <b>Hex Code</b> | <b>Mnemonics</b> | <b>Hex Code</b> | <b>Mnemonics</b> |
|-----------------|------------------|-----------------|------------------|
| 9C              | SBB H            | C3              | JMP              |
| 9D              | SBB L            | C4              | CNZ              |
| 9E              | SBB M            | C5              | PUSH B           |
| 9F              | SBB A            | C6              | ADI              |
| A0              | ANA B            | C7              | RST 0            |
| A1              | ANA C            | C8              | RZ               |
| A2              | ANA D            | C9              | RET              |
| A3              | ADA E            | CA              | JZ               |
| A4              | ANA H            | CB              | ---              |
| A5              | ANA L            | CC              | CZ               |
| A6              | ANA M            | CD              | CALL             |
| A7              | ANA A            | CE              | ACI              |
| A8              | XRA B            | CF              | RST 1            |
| A9              | XRA C            | D0              | RNC              |
| AA              | XRA D            | D1              | POP D            |
| AB              | XRA E            | D2              | JNC              |
| AC              | XRA H            | D3              | OUT              |
| AD              | XRA L            | D4              | CNC              |
| AE              | XRA M            | D5              | PUSH D           |
| AF              | XRA A            | D6              | SUI              |
| B0              | ORA B            | D7              | RST 2            |
| B1              | ORA C            | D8              | RC               |
| B2              | ORA D            | D9              | ---              |
| B3              | ORA E            | DA              | JC               |
| B4              | ORA H            | DB              | IN               |
| B5              | ORA L            | DC              | CC               |
| B6              | ORA M            | DD              | ---              |
| B7              | ORA A            | DE              | SBI              |
| B8              | CMP B            | DF              | RST 3            |
| B9              | CMP C            | E0              | RPO              |
| BA              | CMP D            | E1              | POP H            |
| BB              | CMP E            | E2              | JPO              |
| BC              | CMP H            | E3              | XTHL             |
| BD              | CMP L            | E4              | CPO              |
| BE              | CMP M            | E5              | PUSH H           |
| BF              | CMP A            | E6              | ANI              |
| C0              | RNZ              | E7              | RST 4            |
| C1              | POP B            | E8              | RPE              |
| C2              | JNZ              | E9              | PCHL             |

| <b>Hex Code</b> | <b>Mnemonics</b> | <b>Hex Code</b> | <b>Mnemonics</b> |
|-----------------|------------------|-----------------|------------------|
| EA              | JPE              | F5              | PUSH PSW         |
| EB              | XCHG             | F6              | ORI              |
| EC              | CPE              | F7              | RST 6            |
| ED              | ---              | F8              | RM               |
| EE              | XRI              | F9              | SPHL             |
| EF              | RST 5            | FA              | JM               |
| F0              | RP               | FB              | EI               |
| F1              | POP PSW          | FC              | CM               |
| F2              | JP               | FD              | ---              |
| F3              | DI               | FE              | CPI              |
| F4              | CP               | FF              | RST 7            |

**Total: 246 Instructions**

### Appendix - III

#### Instruction Set of 8085

|   | Lower order Nibble |         |         |         |         |          |         |         | Higher order Nibble |         |         |         |         |         |        |        |
|---|--------------------|---------|---------|---------|---------|----------|---------|---------|---------------------|---------|---------|---------|---------|---------|--------|--------|
|   | 0                  | 1       | 2       | 3       | 4       | 5        | 6       | 7       | 8                   | 9       | A       | B       | C       | D       | E      | F      |
| 0 | NOP                | LXI B   | STAX B  | INX B   | INR B   | DCR B    | MVI B   | RLO     | -                   | DAD B   | LDAX B  | DCX B   | INRC    | DCRC    | MVIC   | RRC    |
| 1 | -                  | LXI D   | STAX D  | INX D   | INR D   | DCR D    | MVI D   | RAL     | -                   | DAD D   | LDAX D  | DCX D   | INRE    | DCRE    | MVIE   | RAR    |
| 2 | RIM                | LXI H   | SHLD    | INX H   | INR H   | DCR H    | MVI H   | DAA     | -                   | DAD H   | LHD     | DCX H   | INRL    | DCRL    | MVL    | CMA    |
| 3 | SIM                | LXI SP  | STA     | INX SP  | INR M   | DCR M    | MVI M   | STC     | -                   | DAD SP  | LDA     | DCX SP  | INRA    | DCRA    | MVA    | CMC    |
| 4 | MOV B,B            | MOV B,C | MOV B,D | MOV B,E | MOV B,H | MOV B,L  | MOV B,M | MOV B,A | MOV C,B             | MOV C,C | MOV C,D | MOV C,E | MOV C,H | MOV C,L | MOV CM | MOV CA |
| 5 | MOV D,B            | MOV D,C | MOV D,D | MOV D,E | MOV D,H | MOV D,L  | MOV D,M | MOV D,A | MOVE B              | MOVE C  | MOVE D  | MOVE E  | MOVE H  | MOVE L  | MOVE M | MOVE A |
| 6 | MOV H,B            | MOV H,C | MOV H,D | MOV H,E | MOV H,H | MOV H,L  | MOV H,M | MOV H,A | MOV L,B             | MOV L,C | MOV L,D | MOV L,E | MOV LH  | MOV LL  | MOV LM | MOV LA |
| 7 | MOV M,B            | MOV M,C | MOV M,D | MOV M,E | MOV M,H | MOV M,L  | HLT     | MOV M,A | MOV A,C             | MOV A,D | MOV A,E | MOV AH  | MOV AL  | MOV AM  | MOV AA | MOV AA |
| 8 | ADD B              | ADD C   | ADD D   | ADD E   | ADD H   | ADD L    | ADD M   | ADD A   | ADC B               | ADC D   | ADC E   | ADC H   | ADC H   | ADC L   | ADC M  | ADC A  |
| 9 | SUB B              | SUB C   | SUB D   | SUB E   | SUB H   | SUB L    | SUB M   | SUB A   | SBB B               | SBB C   | SBB D   | SBB E   | SBB H   | SBB L   | SBB M  | SBB A  |
| A | ANA B              | ANA C   | ANA D   | ANA E   | ANA H   | ANAL     | ANAM    | ANAA    | XRA B               | XRA C   | XRAD    | XRA E   | XRA H   | XRAL    | XRAM   | XRA A  |
| B | ORA B              | ORA C   | ORA D   | ORA E   | ORA H   | ORAL     | ORAM    | ORAA    | CMP B               | CMP C   | CMPD    | CMP E   | CMP H   | CMP L   | CMP M  | CMP A  |
| C | RNZ                | PQP B   | JNZ     | PUSH B  | CNZ     | PUSH B   | ADI     | RST 0   | RZ                  | RET     | JZ      | -       | CZ      | CALL    | ACI    | RST 1  |
| D | RNC                | POP D   | JNC     | OUT     | CNC     | PUSH D   | SUJ     | RST 2   | RC                  | -       | JC      | IN      | CC      | -       | SBI    | RST 3  |
| E | RPO                | POP H   | JPO     | XTHL    | CPO     | PUSH H   | ANI     | RST 4   | RPE                 | PCHL    | JPE     | XCHG    | CPE     | -       | XRI    | RST 5  |
| F | RP                 | POP PSW | JP      | DI      | CP      | PUSH PSW | QRI     | RST 6   | RM                  | SPHL    | JM      | EI      | CM      | -       | CPI    | RST 7  |