1.2 EVOLUTION OF MICROPROCESSORS

The first microprocessor, Intel 4004, a 4-bit PMOS microprocessor was introduced in the year 1971 by Intel Corporation, U.S.A. After this a 4-bit microprocessor Intel 4040, an enhanced version of Intel 4004 was developed. Many other companies also developed 4-bit microprocessors. Examples are: Rockwell International's PPS-4, Toshiba's T3472, etc. 4-bit microcontrollers and 4-bit microprocessor-based systems were used in industrial control applications, calculators, instrumentation, commercial appliances, videogames, toys, etc. Calculator is not a programmable machine, and hence it is not a computer. Step by step technique is used for calculations. The user does not prepare a program to solve his problem. In 1972, Intel introduced the first 8-bit microprocessor, Intel 8008 which also uses PMOS technology. The microprocessors using PMOS technology were slow and not compatible with TTL logic. In 1973, Intel introduced a more powerful and faster 8-bit NMOS microprocessor, Intel 8080. The microprocessors using NMOS process technology are faster and compatible with TTL logic. The NMOS technology also provides higher density as compared to PMOS technology. The drawback of the 8080 was that it needed three power supplies. In 1975, Intel developed an improved 8-bit NMOS microprocessor, Intel 8085 which uses only one +5 V supply. It is an improved version of Intel 8080. It is still used in laboratory for students training. In the first course on microprocessor, the theory based on Intel 8085 is taught as it gives the basic concept of microprocessor, which are needed for further study of other courses on microprocessors, micro-controllers and computers.

8-bit microprocessors of other manufacturers are: Zilog's Z80 and Z800, National Semi-conductor's NSC800, Motorola's MC6800 and MC6809, MOS Technology's 6500 series, Rockwell International's PPS-8, RCA COSMAC (it uses CMOS technology), etc. 8-bit microprocessors and 8-bit microcontrollers are widely used in instrumentation, industrial control applications, etc. Some powerful 8-bit microprocessors were used in small general purpose computers. Generally, the memory addressing capacity of 8-bit microprocessors are 64 KB. The clock frequency for the most of 8-bit microprocessors lies in the range of 1 MHz to 6 MHz. The Z800 can address more memory i.e. 500 KB and it operates at 25 MHz. The 8-bit microprocessors use LSI technology and contain 5000 to 10000 transistors.

In the year 1978, Intel introduced a 16-bit microprocessor, Intel 8086. Other 16-bit microprocessors are: Intel 80186, Intel 8088, Intel 80188, Intel 80286, Zilog's Z8000, Motorola's 68000, 68010, 68012; Fairchild's 9440, National Semiconductor's PACE and INS8900, Texas Instruments' TMS9900 series, and so on. 12-bit microprocessors were also developed. Examples are : Toshiba's T3190, Intersil's IM6100, etc. The 80186 and 80188 are integrated microprocessors. Besides CPU they contain some additional components like programmable interrupt controller, two independent high-speed DMA channels, three programmable 16-bit timer/counters, clock generator, programmable memory and peripheral chip-select logic, programmable wait state generator and local bus controller. The 80286 has been designed for multiuser system. Besides CPU it contains integrated memory management unit, four-level memory protection and support for virtual memory and operating system. The 16-bit microprocessors have more powerful instruction set than that of 8-bit microprocessors. They use VLSI technology. They can directly address memory in the range of 1 MB to 16 MB. 16-bit microprocessors are designed to work in multiprocessor environment and use assembly languages and efficient high-level languages. They were used in complex industrial control applications, general purpose and portable computers, etc. In 1980s, personal computers were widely used. They used 16-bit microprocessors.

Table 1.1 Important Intel Microprocessors

Micro processor	*Year of Introduction*	*Wordlength*	*Memory addressing capacity*	*Pins*	*Clock*	*Remarks*
4004	1971	4-bit	1 KB	16	750 KHz	First Microprocessor
8085	1976	8-bit	64 KB	40	3-6 MHz	Popular 8-bit Microprocessor
8086	1978	16-bit	1 MB	40	5-10 MHz	Popular 16-bit Microprocessor
80486	1989	32-bit	4 GB real, 64 TB virtual	17 × 17 PGA	25-100 MHz	Contains MMU, cacge and, FPU, 1.2 million transistors,
Pentium	1993	32-bit	4 GB real, 32-bit address 64-bit data bus	237 PGA	60-200	Contains 2 ALUs, 2 Caches, FPU, 3.3 million transistors, 3.3 V and 5 V versions
Pentium Pro	1995	32-bit	64 GB real, 36-bit address bus	387 PGA	150-200 MHz	It is a data flow processor. It contains second-level cache also, 3.3 V
Celeron	1998	32-bit	—	—	2 GHz	Low-cost processor
Pentium 4	2000	32-bit	64 GB	423 PGA	1.3-3.2 GHz	Data flow architecture and SIMD instructions
Itanium	2001	64-bit	64 address lines	423 PGA	733 MHz-1.5 GHz	64-bit EPIC (Explicitly Parallel) Instruction Computing) processor
Core 2 Duo	2005	—	—	—	3 GHz	Two CPUs on a single chip
Core 2 Quad	2006	—	—	—	3 GHz	Quad CPUs on a single chip
Core 2 Extreme	2006 Quad core (QX Series)	—	—	—	3 GHz	Quad CPUs on a single chip
i7 965 XE	2009 Quad core	—	—	—	3.2 GHz	Quad CPUs on a single chip

Very popular personal computers PC and PC/XT used Intel 8088 microprocessor. The PC/AT used 80286 microprocessor. XT and AT stand for Extended Technology and Advanced Technology respectively.

After 1980 many manufacturers introduced 32-bit microprocessors. In the year 1985, Intel introduced a more powerful 32-bit microprocessor, Intel 80386 which became very popular and was widely used in desktop computers. In short it was called Intel 386 microprocessor. Other 32-bit Intel's microprocessors are: the 486, Pentium, Pentium Pro, Pentium MMX, Pentium II, Pentium II Xeon, Celeron, Pentium III and Pentium 4. Pentium II Xeon (1998) uses second-level cache memory 1 MB/2 MB. The Celeron (1998) is a low-cost 32-bit processor built on P6 (Pentium Pro) architecture. It is an entry-level 32-bit processor. Motorola's 68020, 68030, 68040 and 88100; Advanced Micro Devices' (AMD's) K5, K6-MMX and Athlon and Sempron; SUN's SPARC; Cyrix's 586, 686 and 6X86MX; National Semiconductor's 32032, 32332 and 32C532; Zilog's Z80000; Inmos' T414 and T800, etc. are 32-bit microprocessors. Motorola, IBM and Apple have jointly developed 32-bit RISC processors: PowerPC 601, 603 and 604. PowerPC 740 and 750 have been developed by IBM and Motorola. 32-bit microprocessors are widely used for desktop, portable and notebook computers; workstations and servers. Intel has also introduced 80960, a 32-bit RISC microprocessor for embedded control applications. 32-bit microprocessors contain a number of essential components of a computer besides CPU, such as MMU, FPU, on-chip cache memory, etc.

A number of 64-bit microprocessors have also been developed. Examples are : SUN's SPARC and ULTRASPARC, AMD's Athlon 64 and Opteron, R 12000; Intel-H.P's PA8000 series, HP's 8200 and 8500 series, etc. Intel's Itanium is a 64-bit EPIC (Explicitly Instructions Parallel Computing) processor. Table 1.1 shows the summary of microprocessors of Intel Corporation.

Now-a-days multicore processors have been developed. Two, four, eight or more CPUs are placed on a single-chip IC. Examples of multicore processors having 2 CPUs on a single chip (i.e. in an IC) are: Intel Core 2 Duo 8500, Intel Core 2 Duo E6850, Intel Core 2 Extreme X6800, Intel Core Duo T2700; Intel Core i3 2310M, Intel Atom D525, AMD Athlon 64 X2 6000+, AMD Turion X2 TL-60, AMD dual core Opteron etc. Examples of multicore processors having 4 CPUs on a single chip (single IC) are: Intel Core 2 Extreme QX9770, Intel Core 2 Extreme QX6850, Intel Core 2 Quad Q9550, Intel i7 965XE, Intel Core i5 2500K, AMD Opteron Barcelona, K8L, AMD Phenom II X4 9950 etc. Intel dual and quad core Xeon processors for servers are also available. The Dhrystones (integer performance rating) rating of Intel Core 2 Extreme QX6700 is 49153 MIPS, and Whetstones rating (both integer and floating-point rating) is 32,865 MFLOPS.

Number of transistors used were: 2300 in Intel 4004, 29000 in Intel 8086, 55 million in Pentium 4, 582 million in Intel Core 2 Extreme QX6850, and 820 million in Intel QX9770. Process technologies used are 45 nm and 32 nm.

1.3 EVOLUTION OF DIGITAL COMPUTERS

General-purpose electronic computers using valves were developed in 1940s. Earlier, successful general-purpose mechanical computers were developed in 1930s. Before 1930 mechanical calculators were built to perform addition, subtraction, multiplication and division. The first mechanical calculator was designed and built in the year 1623 by Mr. Wilhelm Schickhard, professor at University of Tubingen. This machine did not become popular at that time. A popular mechanical calculator, capable of performing

CHAPTER 3

MICROPROCESSOR ARCHITECTURE

INTRODUCTION

The microprocessor(s) is the central processing unit (CPU) of a computer. It is the heart of the computer. This chapter describes Intel 8085 as it is one of the most popular 8-bit microprocessors. The Intel Corporation has also developed a large number of general purpose and special purpose peripheral devices. These devices are very useful for the development of microprocessor-based system. The availability of a variety of support devices is also one of the causes of popularity of Intel microprocessors.

3.1 INTEL 8085

Intel 8085 is an 8-bit, NMOS microprocessor. It is a 40 pin I.C. package fabricated on a single LSI chip. The Intel 8085 uses a single $+5V_{d.c}$ supply for its operation. Its clock speed is about 3 MHz. The clock cycle is of 320 ns. The time for the clock cycle of the Intel 8085AH-2, version is 200 ns. It has 80 basic instructions and 246 opcodes. Fig. 3.1 shows the block diagram of Intel 8085. It consists of three main sections: an arithmetic and logic unit, a timing and control unit and a set of registers. These important sections are described in the subsequent sections.

3.1.1 ALU

The arithmetic and logic unit, ALU, performs the following arithmetic and logical operations:

(i) Addition
(ii) Subtraction
(iii) Logical AND
(iv) Logical OR
(v) Logical EXCLUSIVE OR
(vi) Complement (logical NOT)
(vii) Increment (add 1)
(viii) Decrement (subtract 1)
(ix) Left shift, Rotate left, Rotate right
(x) Clear etc.

3.1.2 Timing and Control Unit

The timing and control unit is a section of the CPU. It generates timing and control signals which are necessary for the execution of instructions. It controls data flow between CPU and peripherals (including memory). It provides status, control and timing signals which are required for the operation of memory and I/O devices. It controls the entire operations of the microprocessor and peripherals connected to it. Thus, it is seen that the control unit of the CPU acts as the brain of the computer system.

3.1.3 Registers

Figure 3.1 shows the various registers of Intel 8085. Registers are used by the microprocessor for temporary storage and manipulation of data and instructions. Data

remain in the registers till they are sent to the memory or I/O devices. In a large computer the number of registers is more and hence the program requires less transfer of data to/from the memory. In a small computer the number of registers is small due to limited size of the chip. Intel 8085 microprocessor has the following registers:

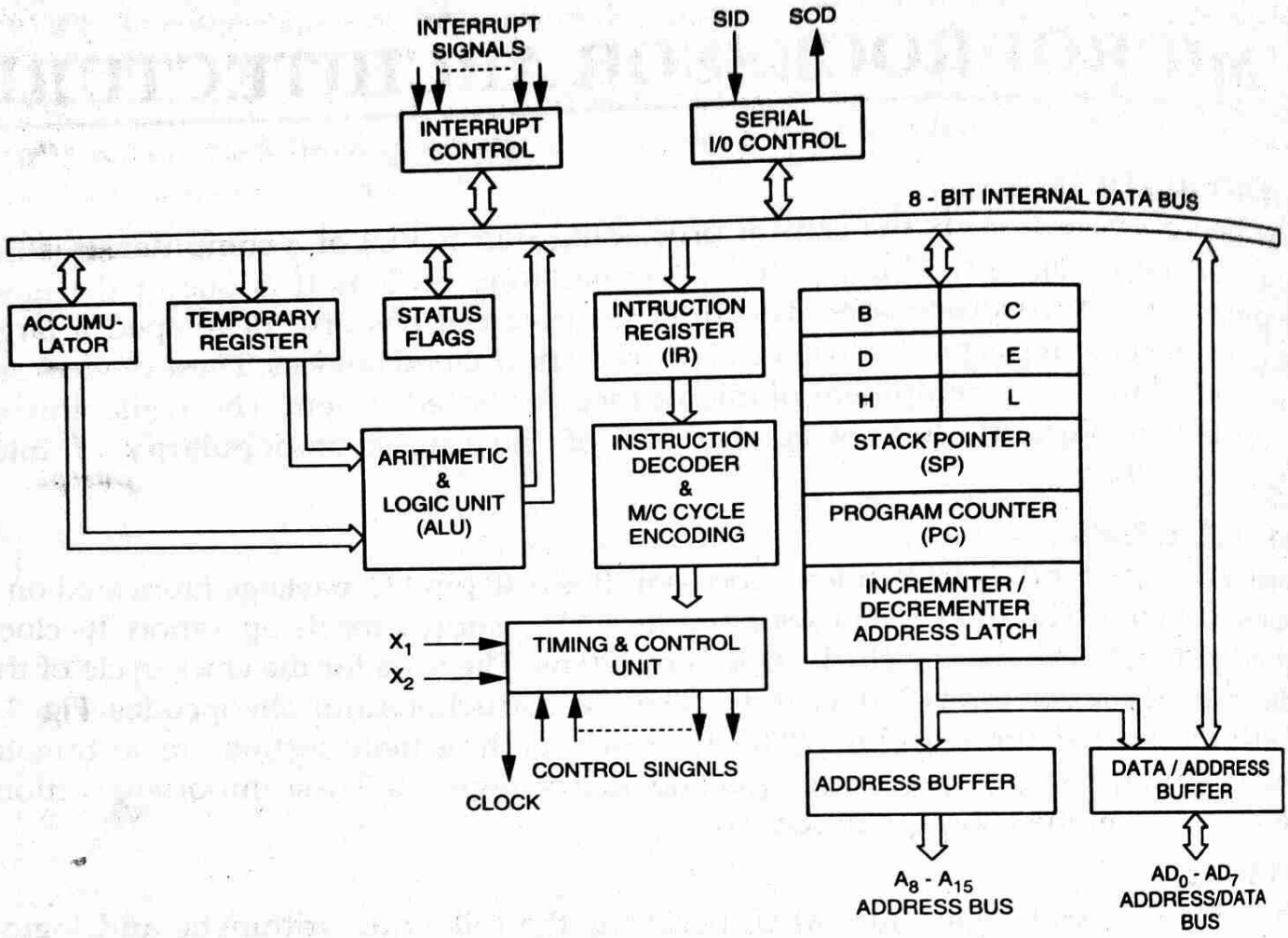

Fig. 3.1 Block diagram of Intel 8085

(i) One 8-bit accumulator (ACC) *i.e.* register *A*
(ii) Six 8-bit general purpose registers. These are *B*, *C*, *D*, *E*, *H* and *L*
(iii) One 16-bit stack pointer, *SP*
(iv) One 16-bit program counter, *PC*
(v) Instruction register
(vi) Temporary register

In addition to the above mentioned registers the 8085 microprocessor contains a set of five flip-flops which serve as flags (or status flags). A flag (or status flag) is a flip-flop which indicates some condition which arises after the execution of an arithmetic or logical instruction.

Accumulator (ACC). The accumulator is an 8-bit register associated with the ALU. The register 'A' in the 8085 is an accumulator. It is used to hold one of the operands of an arithmetic or logical operation. It serves as one input to the ALU. The other operand for an arithmetic or logical operation may be stored either in the memory or in one of the general-purpose registers. The final result of an arithmetic or logical operation is placed in the accumulator.

The above descriptions are true for general cases, not for some typical or exceptional cases. For example, there are some logical instructions which need only one operand. It is held in the accumulator. The result is placed in the accumulator. Such

instructions do not require any other register or memory location because there is no other operand. There is one typical instruction DAD rp, for 16-bit addition for which one of the 16-bit operands is kept in H-L pair and the other in the B-C or D-E pair. The result is placed in the H-L pair. See details for such cases in Chapter 4.

General-Purpose Registers. The 8085 microprocessor contains six 8-bit general-purpose registers. They are: B, C, D, E, H and L register. To hold 16-bit data a combination of two 8-bit registers can be employed. The combination of two 8-bit registers is known as a **register-pair**. The valid register pairs in the 8085 are: B-C, D-E and H-L. The programmer can not form a register-pair by selecting any two registers of his choice. The H-L pair is used to act as memory pointer and for this purpose it holds the 16-bit address of a memory location. The general-purpose registers and the accumulator are accessible to programmer. He can store data in these registers during writing his program.

Program Counter (PC). It is a 16-bit special-purpose register. It is used to hold the memory address of the next instruction to be executed. It keeps the track of memory addresses of the instructions in a program while they are being executed. The microprocessor increments the content of the program counter during the execution of an instruction so that it points to the address of the next instruction in the program at the end of the execution of an instruction.

Stack Pointer (SP). It is a 16-bit special function register. The **stack** is a sequence of memory locations set aside by a programmer to store/retrieve the contents of accumulator, flags, program counter and general-purpose registers during the execution of a program. Any portion of the memory can be used as stack. Since, the stack works on LIFO (last-in-first-out) principle, its operation is faster compared normal store/retrieve of memory locations. During the execution of a program sometimes it becomes necessary to save the contents of some registers which are needed for some other operations in the subsequent steps of the program. The contents of such registers are saved in the stack. Then the registers are used for some other operations. After completing the needed operations the contents which were saved in the stack are brought back to the registers. The contents of only those registers are saved, which are needed in the later part of the program. The stack pointer (SP) controls addressing of the stack. The SP holds the address of the top element of data stored in the stack.

The stack is defined and the stack pointer is initialized by the programmer at the beginning of a program which needs stack operation. Stack is also used by the microprocessor. For example, it stores the contents of program counter when it jumps to a subroutine using CALL instruction. Stack has been described in details in Chapter 5.

Instruction Register. The instruction register holds the opcode (operation code or instruction code) of the instruction which is being decoded and executed.

Temporary Register. It is an 8-bit register associated with the ALU. It holds data during an arithmetic/logical operation. It is used by the microprocessor. It is not accessible to programmer.

Flags. The Intel 8085 microprocessor contains five flip-flops to serve as status flags. The flip-flops are set or reset according to the conditions which arise during an arithmetic or logical operation.

The five status flags of Intel 8085 are:

(i) Carry Flag (CS)

(ii) Parity Flag (P)
(iii) Auxiliary Carry Flag (AC)
(iv) Zero Flag (Z)
(v) Sign Flag (S)

If a flip-flop for a particular flag is set, it indicates 1. When it is reset, it indicates 0.

Carry Flag (CS). After the execution of an arithmetic instruction if a carry is produced, the carry flag CS is set to 1, otherwise it is 0. The carry flag is set or reset in case of addition as well as subtraction. After the addition of two 8-bit numbers, if the sum is larger than 8 bits, a carry is produced; and the carry flag is set to 1. In case of subtraction, if borrow occurs, the carry flag is set to 1. The carry flag holds carry out of the most significant bit resulting from the execution of an arithmetic operation.

Parity Flag (P). The parity status flag P is set to 1, if the result of an arithmetic or logical operation contains even number of 1s. It is reset i.e. it is 0, if the result contains odd number of 1s.

Auxiliary Carry Flag (AC). The auxiliary carry flag AC holds carry out of the bit number 3 to the bit number 4 resulting from the execution of an arithmetic operation. The counting of bits starts from 0, and hence, the Bit No. 3 is actually the fourth bit from the least significant bit [See Fig. 3.2(b)].

Zero Flag (Z). The zero status flag Z is set to 1, if the result of an arithmetic or logical operation is 0. If the result is not zero, the flag is set to 0.

Sign Flag (S). The sign flag S is set to 1, if the result of an arithmetic or logical operation is negative. If the result is positive, the sign flag is set to 0.

The sign flag has its significance only when signed arithmetic operation is performed. To represent a signed number the most significant bit is reserved by the programmer to represent the sign of a number. In other words the MSB is used as a sign bit. It represents the sign of the number. If a number is negative, the sign bit is 1. For a positive number, the sign bit is 0. In case of 8-bit signed operation, the remaining 7 bits are used to represent the magnitude of a number. After execution of signed arithmetic operation, the MSB of the result represents its sign. The sign flag acquires the value of the MSB of the result following the execution of signed arithmetic operation. Hence, it represents the sign of the result.

For unsigned arithmetic operation, all the 8 bits are used to represent the magnitude of the number. After the execution of an arithmetic operation, all the 8 bits of the result represent its magnitude. Therefore, the sign flag has no significance in unsigned arithmetic operation. Also, for logical operation sign bit has no significance. Since, the sign flag is set or reset according to the MSB of the result, it is set or reset on the value of MSB of the result of logical operation also.

The above discussion also holds good for signed arithmetic operation of 16-bit, 32-bit or more. In case of 16-bit operation 15 bits are used to represent the magnitude of a number and 1 bit to represent its sign. In case of 32-bit operation 31 bits are used to represent the magnitude and 1 bit to represent its sign.

Figure 3.2 (b) shows the status flags of ADD operation. Take an example of the instruction ADD B. The execution of the instruction ADD B will add the content of the register B to the content of the accumulator. Suppose, the contents of the accumulator and register B are CB and E9 respectively. Now, CB and E9 are added and the result is 01, B4. As the accumulator is an 8-bit register B4 remains in the accumulator and there is a carry. The various status flags are shown in Fig. 3.2 (b).

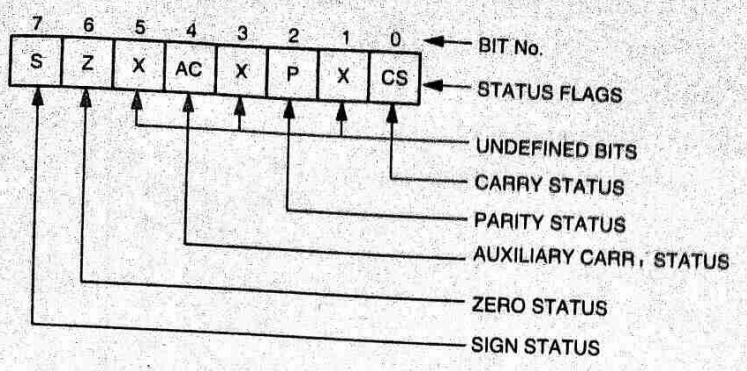

Fig. 3.2 (a) Status Flags of Intel 8085

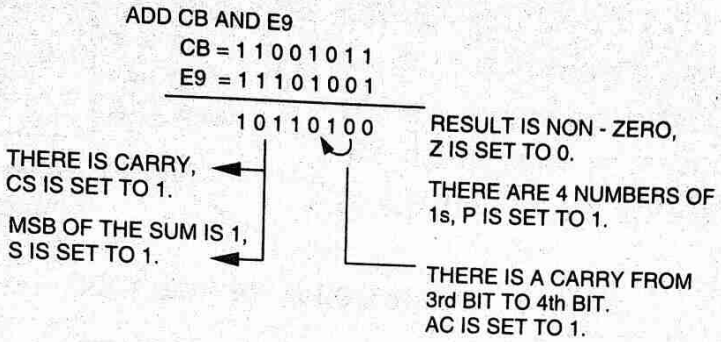

Fig. 3.2 (b) Status Flags for ADD operation

PSW. In Fig. 3.2 (a), five bits indicate the five status flags and three bits are undefined. The combination of these 8 bits is called **Program Status Word (PSW)**. PSW and the accumulator are treated as a 16-bit unit for stack operation.

3.1.4 Data and Address Bus

The Intel 8085 is an 8-bit microprocessor. Its data bus is 8-bit wide and hence, 8 bits of data can be transmitted in parallel from or to the microprocessor. The Intel 8085 requires a 16-bit wide address bus as the memory addresses are of 16 bits. The 8 most significant bits of the address are transmitted by the address bus, A-bus (pins A_8 to A_{15}). The 8 least significant bits of the address are transmitted by address/data bus, AD-bus (pins AD_0 - AD_7). The address/data bus transmits data and address at different moments. At a particular moment it transmits either data or address. Thus, the AD-bus operates in time shared mode. This technique is known as multiplexing. First of all 16-bit memory address is transmitted by the microprocessor; the 8 MSBs of the address on the A-bus and the 8 LSB_S of the address on AD-bus. Thus, the effective width of the address bus becomes 16-bit wide. Then the 8 LSBs of the address is latched either into the memory or external latch so that the complete 16-bit address remains available for further operation. The 8-bit AD-bus now becomes free, and it is available for data transmission. 2^{16} (= 65536 = 64 K, where 1 K = 1024) memory locations can be addressed directly by Intel 8085. Each memory location contains 1 byte of data.

3.1.5 Pin Configuration

Figure 3.3 shows the schematic diagram of Intel 8085. The description of various pins are as follows:

A_8 - A_{15} (output). These are address bus and are used for the most significant bits of the memory address or 8 bits of I/O address.

AD_0 - AD_7 (input/output). These are time multiplexed address/data bus *i.e.* they serve dual purpose. They are used for the least significant 8 bits of the memory address or I/O address during the first clock cycle of a machine cycle. Again they are used for data during second and third clock cycles.

ALE (output). It is an address latch enable signal. It goes high during first clock cycle of a machine cycle and enables the lower 8 bits of the address to be latched either into the memory or external latch.

IO/$\overline{M}$ (output). It is a status signal which distinguishes whether the address is for memory or I/O. When it goes high, the address on the address bus is for an I/O device. When it goes low, the address on the address bus is for a memory location.

S0, S_1 (output). These are status signals sent by the microproessor to distinguish the various types of operation given in Table 3.1.

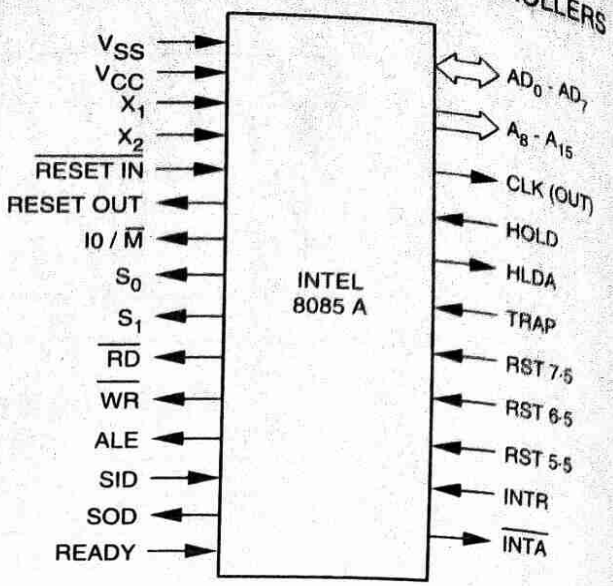

Fig. 3.3 Schematic Diagram of Intel 8085

Table 3.1 Status Codes for Intel 8085

S_1	S_0	*Operations*
0	0	HALT
0	1	WRITE
1	0	READ
1	1	FETCH

$\overline{RD}$ (output). When microprocessor reads data or codes from a memory location or an input device, it is called READ operation. $\overline{RD}$ is a signal sent by the microprocessor to the memory/input device to control READ operation. When it goes low, the selected memory or input device is read.

$\overline{WR}$ (output). When microprocessor sends data to a memory location or an output device, it is called WRITE operation. $\overline{WR}$ is a signal sent by the microprocessor to the memory/output device to control WRITE operation. When it goes low, the data which is on the data bus, is written into the selected memory or sent to the output device.

READY (input). It is a signal sent by an input or output device to the microprocessor. This signal indicates that the input or output device is ready to send or receive data. The microprocessor examines READY signal before it performs data transfer operation. A slow input or output device is connected to the microprocessor through READY line. When READY is high, it indicates that the input or output device is ready to send or receive data. When READY is low, the microprocessor waits till READY becomes high. The microprocessor examines the status of READY signal in the second clock cycle of the machine cycle.

HOLD (input). When another device of the computer system, requires address and data buses for data transfer, it sends HOLD signal to the microprocessor. After receiving the HOLD request, the microprocessor sends out a HLDA (HOLD Acknowledge) signal to the device. Then the microprocessor leaves the control over the buses as soon as the current machine cycle is completed. Internal processing may continue. The microprocessor regains the control over the buses after the HOLD signal is removed.

HLDA (output). It is a HOLD acknowledge signal sent out by the microprocessor after receiving the HOLD signal. It is sent to the device which has issued the HOLD signal. After the removal of the HOLD signal, the HLDA goes low, and thereafter the microprocessor takes over the buses.

INTR (input). It is an interrupt signal sent by an external device to the microporcessor. Through this line an external device informs microprocessor that it is ready to transfer data or to initiate certain operation. The 8085 microprocessor has 5 interrupt lines. The INTR is one of them. When it goes high, the microprocessor suspends the execution of its normal sequence of instructions. After completing the current instruction at hand, it attends the interrupting device. The microprocessor issues an interrupt acknowledge signal $\overline{\text{INTA}}$. Then it transfers data or takes any other action as required.

$\overline{\text{INTA}}$ (output). It is an interrupt acknowledge signal issued by the microprocessor after receiving an interrupt request from an external device. It is a low active signal.

RST 5.5, 6.5, 7.5 and TRAP (inputs). These are interrupts. When an interrupt is recognised the next instruction is executed from a fixed location in the memory as given below:

Line	*Location from which next instruction is picked up*
TRAP	0024
RST 5.5	002C
RST 6.5	0034
RST 7.5	003C

RST 7.5, RST 6.5 and RST 5.5 are the restart interrupts. Each of them has a programmable mask. The TRAP has the highest priority among interrupts. It is a nonmaskable interrupt. It is unaffected by any mask or interrupt enable. The order of priority of interrupts is as follows:

TRAP (highest priority)
RST 7.5
RST 6.5
RST 5.5
INTR (lowest priority).

RESET IN (input). It resets the program counter to zero. It also resets interrupt enable and HLDA flip-flops. It does not affect any other flag or register except the instruction register. The CPU is held in reset condition as long as RESET is applied.

RESET OUT (output). It indicates that the CPU is being reset.

X_1, X_2 (input). These are terminals to be connected to an external crystal oscillator which drives an internal circuitry of the microprocessor to produce a suitable clock for the operation of microprocessor.

CLK (output). It is a clock output for user, which can be used for other digital ICs. Its frequency is same at which processor operates.

SID (input). It is data line for serial input. The data on this line is loaded into the 7th bit of the accumulator when RIM instruction is executed.

SOD (output). It is a data line for serial output. The 7th bit of the accumulator is output on SOD line when SIM instruction is executed.

V_{CC} + 5 Volts supply
V_{SS} ground reference

3.1.6 Intel 8085 Instructions

A computer receives data from the user, processes data and sends the result back to the user. The computer simply performs a given task on specified data in response to certain instructions. An instruction is a command given to the computer to perform a specified operation on given data.

Intel 8085 microprocessor is widely used in India. Its instructions are discussed in detail in Chapter 4. A few of them are described here as they are needed in this chapter in subsequent sub-sections.

MOV r_1, r_2. The content of register r_2 is moved to register r_1. For example, the instruction MOV A, B moves the content of register B to register A. The instruction MOV B, A moves the content of register A to register B.

MOV M, r. The content of register *r* is moved to the memory location whose address is in H-L pair. For example, MOV M, A will transfer the content of the accumulator to the memory location specified by H-L pair.

MVI r, data. The data given just after an instruction are moved to the register *r*. For example, the instruction MVI A, 05 moves 05 to register A.

LXI rp, data 16-bit. 16-bit immediate data are moved to the register pair *rp*. For example, the instruction LXI H, 2400H moves 2400 to H-L pair.

LDA addr. The content of the memory location whose address is specified within the instruction itself is moved to the accumulator. For example, the instruction LDA 2400H moves the constent of the memory location 2400H to the accumulator.

STA addr. The content of the accumulator is moved to the memory location whose address is specified within the instruction itself. For example, the instruction STA 2500H moves the content of the accumulator to memory location 2500H.

ADD r. The content of the register *r* is added to the content of the accumulator. For example, the instruction ADD B adds the content of register B to the content of the accumulator and places the sum in the accumulator.

SUB r. The content of register *r* is subtracted from the content of the accumulator and the result is placed in the accumulator. For example, the instruction SUB C subtracts the content of register C from the content of the accumulator and places the result in the accumulator.

RAL. The content of the accumulator is rotated left one bit through carry.

INR r. The content of register *r* is incremented by one.

IN Port-address. This instruction transfers data from input device or port. The data available at the input port or device is moved to the accumulator.

OUT Port-address. This instruction transfers data from processor to the output port or device. The content of the accumulator is transferred to the output port or device.

3.1.7 Opcode and Operands

Each instruction contains two parts: operation code (opcode) and operand. The first part of an instruction which specifies the task to be performed by the computer is called *opcode*. The second part of the instruction is the data to be operated on, and it is called *operand*. The operand (or data) given in the instruction may be in various forms such as 8-bit or 16-bit data, 8-bit or 16-bit address, internal registers or a register or memory location. In some instructions the operand is implicit. When operand is a register it is understood that the data is the content of the register.

3.1.8 Instruction Word Size

A digital computer understands instructions written in binary codes (machine codes). The machine codes of all instructions are not of the same length. According to the word size the Intel 8085 instructions are classified into the following three types:

(1) 1-byte instruction
(2) 2-byte instruction

(3) 3-byte instruction

One-Byte instruction. Examples of one-byte instructions are:

MOV A, B. Move the content of register B to register A. 78H is the opcode for MOV A, B. Besides the operation to be performed the opcode also specifies the registers which contain operands (data). The opcode 78H can be written in the binary form as 01111000. The 1st two bits, *i.e.*, 01 are for MOV operation; the next three bits 111 are the binary code for register A and the last three bits 000 are the binary code for register B.

ADD B. Add the content of register B to the content of the accumulator. 80H is the opcode for the instruction ADD B. In this instruction one of the operands is register B (its content) which is indicated in the instruction itself. In this type of instruction (arithmetic group of instruction) it is assumed that the other operand is in the accumulator. The opcode 80H in the binary form is 10000000. The first five bits, i.e. 10000 specify the operation to be performed, i.e. ADD operation. The last three bits 000 are the code for register B for 8085 microprocessor.

RAL. Rotate the content of the accumulator left by one bit. The opcode for RAL is 17H. In this instruction operand is not given, it is implied. The operand is in the accumulator. The instruction has to operate on the content of the accumulator.

All the above three examples are only one byte long. All one-byte instructions contain informtaion regarding operands in the opcode itself.

Two-Byte Instruction. In a two-byte instruction the 1st byte of the instruction is its opcode and the 2nd byte is either data or address. Examples are:

(i) MVI B, 05 ; Move 05 to register B.

06, 05 ; MVI B, 05 in the code form.

The 1st byte 06 is the opcode for MVI B and the 2nd byte 05 is the data which is to be moved to register B.

(ii) IN 01 ; Read data at port B.

DB, 01 ; IN 01 in the code form.

DB is the opcode for the instruction IN and 01 is the address of a port (the port B of Intel 8255.1 of a microprocessor kit). A two-byte instruction is stored in two consecutive memory locations.

Three-Byte Instruction. In a three byte instruction the 1st byte of the instruction is its opcode and the 2nd and 3rd bytes are either 16-bit data or 16-bit address. Examples are:

(i) LXI H, 2400H ; Load H-L pair with 2400H.

21, 00, 24 ; LXI H, 2400H in the code form.

The 1st byte 21 is the opcode for the instruction LXI H. The 2nd byte 00 is 8 LSBs of the data (2400H), which is loaded into register L. The 3rd byte 24 is 8 MSBs of the data (2400H), which is loaded into register H.

(ii) LDA 2500H ; get the content of the memory location 2500H into accumulator.

3A, 00, 25 ; LDA 2500H in the code form.

The 1st byte 3A is the opcode for the instruction LDA. The 2nd byte 00 is 8 LSBs of the address of the memory location 2500H. The 3rd byte 25 is 8 MSBs of the address of the memory location 2500H. In this instruction the data is the content of the memory location 2500H. The data is to be loaded into the accumulator. A 3-byte instruction is stored in three consecutive memory locations.

3.2 INSTRUCTION CYCLE

An instruction is a command given to the computer to perform a specified operation on given data. To perform a particular task a programmer writes a sequence of instructions, called a program. Program and data are stored in the memory. The CPU fetches one instruction from the memory at a time and executes it. It executes all the instructions of a program one by one to produce to the final result.

The necessary steps that a CPU carries out to fetch an instruction and necessary data from the memory, and to execute it, constitute an *instruction cycle*. An instruction cycle consists of a *fetch cycle* and *execute cycle*, In fetch cycle a CPU fetches opcode (the machine code of an instruction) from the memory. The necessary steps which are carried out to fetch an opcode from the memory, constitute a *fetch cycle*. The necessary steps which are carried out to get data, if any, from the memory and to perform the specific operation specified in an instruction, constitute an *execute cycle*. The time required to fetch an opcode (FC) is a fixed slot of time while the time required to execute an instruction (EC) is variable which depends on the type of instruction to be executed. The total time required to execute an instruction is given by

$$IC = FC + EC$$

3.2.1 Fetch Operation

The 1st byte of an instruction is its opcode. An instruction may be more than one byte long. The other bytes are data or operand address. The program counter (PC) keeps the memory address of the next instruction to be executed. In the beginning of a fetch cycle the content of the program counter, which is the address of the memory location where opcode is available, is sent to the memory. The memory places the opcode on the data bus so as to transfer it to the CPU. The entire operation of fetching an opcode takes three clock cycles. A slow memory may take more time. In case of a slow memory the CPU has to wait till the memory sends the opcode. The clock cycle for which the CPU waits is called *wait cycle*. Most of the CPUs have been designed to introduce wait cycles to cope with slow memories.

3.2.2 Execute Operation

The opcode fetched from the memory goes to the data register, DR (data/address buffer in Intel 8085) and then to instruction register, IR. From the instruction register it goes to the decoder circuitry which decodes the instruction. The decoder circuitry is within the microprocessor. After the instruction is decoded, execution begins. If the operand is in the general purpose registers, execution is immediately performed. The time taken in decoding and execution is one clock cycle. If an instruction contains data or operand address which are still in the memory, the CPU has to perform some read operations to get the desired data. After receiving the data it performs execute operation. A read cycle is similar to a fetch cycle. In case of a read cycle the quantity

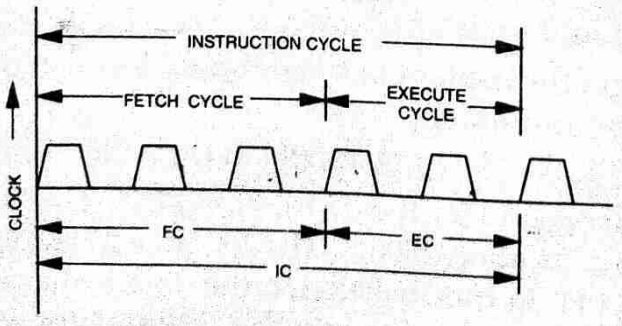

Fig. 3.4 Instruction cycle showing FC, EC, IC

received from the memory are data or operand address instead of an opcode. In some instructions write operation is performed. In write cycle data are sent from the CPU to the memory or an output device. Thus, we see that in some cases an execute cycle may involve one or more read or write cycles or both. Figures. 3.4 and 3.5 shows an instruction and fetch cycle respectively.

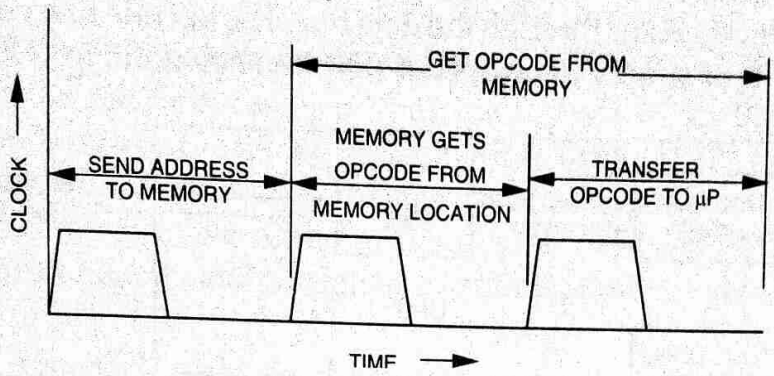

Fig. 3.5 Fetch Cycle

3.2.3 Machine Cycle and State

The necessary steps carried out to perform the operation of accessing either memory or I/O device, constitute a machine cycle. In other words necessary steps carried out to perform a fetch, a read or a write operation constitute a *Machine cycle.* The necessary steps include to send memory or I/O address; IO/$\overline{M}$, ALE, $\overline{RD}$ (or $\overline{WR}$) etc signals.

In a machine cycle one basic operation such as opcode fetch, memory read, memory write, I/O read or I/O write is performed. An instruction cycle consists of several machine cycles. The opcode of an instruction is fetched in the first machine cycle of an instruction cycle. Most of the single-byte instructions require only one machine cycle to fetch the opcode and execute the instruction. Two-byte and three-byte instructions require more than one machine cycle. Additional machine cycles are needed to read data from or to write data into the memory or I/O devices. Figures 3.6 shows instruction cycle for MVI *r*, data. It has two machine cycles: one for fetching opcode, and the other for reading data from the memory and to execute the instruction.

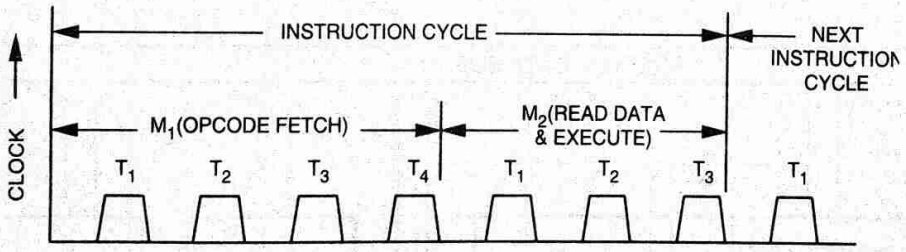

Fig. 3.6 Typical Instruction Cycle

One subdivision of an operation performed in one clock cycle is called a state or T-state. The subdivisions are internal states synchronised with the system clock. So, one clock cycle of the system clock is referred to as a state.

3.2.4 Instruction and Data Flow

A computer receives instructions and data both in binary form. A group of 8 bits is called a byte. Data and instructions for computers are specified in the byte form. The word length of a computer is the number of bits it handles at a time. The word length of an 8-bit microprocessor is of one byte, that of a 16-bit microprocessor is of two bytes and so on. Two kinds of words, namely instruction word (instruction code or opcode) and data word are processed during an instruction cycle. In the beginning of a fetch cycle the content of the program counter is transferred to a special register known as

memory address register, MAR or simply address register, AR; (address buffer in Intel 8085). The content of MAR is transferred to the memory through the address bus. By sending certain control signals to the memory the microprocessor also indicates that it wants to read the content of the memory. The decoder circuitry in the memory is activated and the memory understands what is to be done. Then the memory sends opcode to the microprocessor through the data bus. The opcode first comes in memory data register (MDR) or simply data register (DR) as shown in Fig. 3.7. In case of Intel

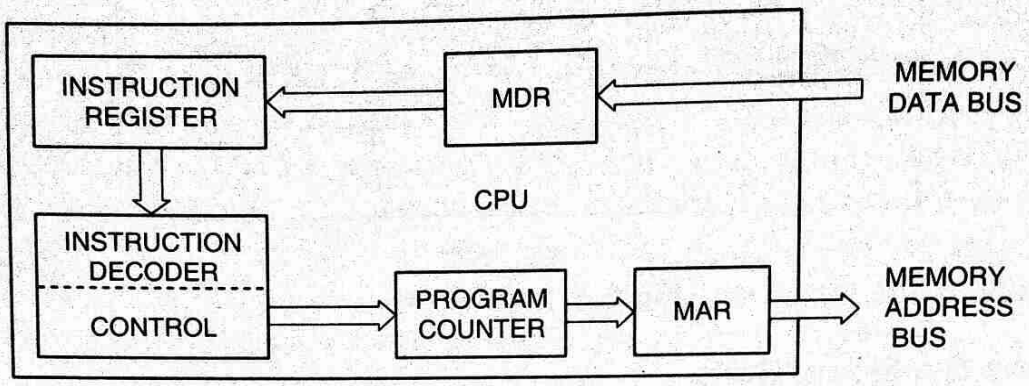

Fig. 3.7 Flow of Instruction word (opcode)

8085, there is a data buffer for this purpose. The operation code is then placed in the instruction register (IR). The instruction is decoded by instruction decoder and it is executed. Finally, the content of the program counter is incremented.

The execution of an instruction requires the flow of data word in the most of the instructions. The flow of a data word is shown in Fig. 3.8. A data word is received either from the memory or input device. The data word flows to the processor through the data bus and is placed in the accumulator or any other general purpose register depending upon the instruction. After the execution of an instruction the data is placed in a register or a memory location. After the execution of a program the result (data) is placed in the memory or sent to an output device. When a data word is written into the memory, it is also held in MDR (or DR or Data Buffer) until the write operation is complete.

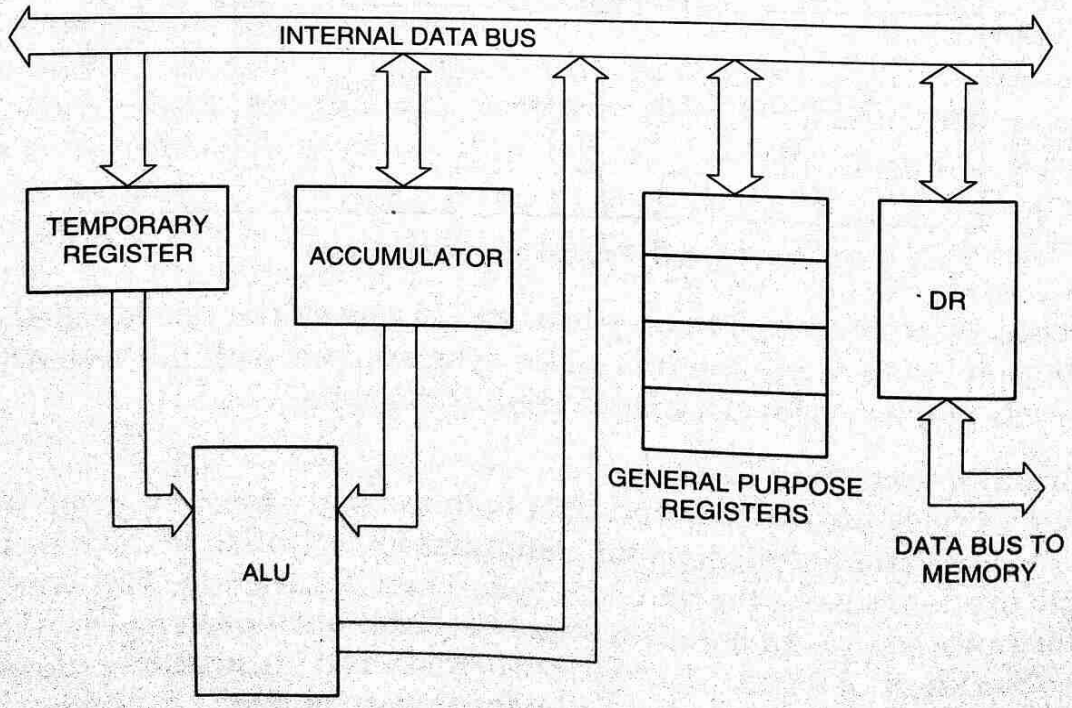

Fig. 3.8 Flow of Data Word

3.3 TIMING DIAGRAM

The necessary steps which are carried out in a machine cycle can be represented graphically. Such a graphical representation is called *timing diagram.* The timing diagram for opcode fetch, memory read, memory write, I/O read and I/O write will be discussed below.

3.3.1 Timing Diagram for Opcode Fetch Cycle

In a fetch cycle the microprocessor fetches the opcode of an instruction from the memory. Figures 3.9 shows the timing diagram for an opcode fetch cycle. T_1, T_2, T_3, and T_4 are consecutive four clock cycles. The microprocessor issues a low IO/$\overline{M}$ signal to indicate that it wants to make communication with the memory. Again the microprocessor sends out high S_0 and S_1 signals to indicate that it is going to perform fetch operation.

During the first clock cycle, T_1, the microprocessor sends out the address of the memory location where the opcode is available. The 16-bit memory address is sent through the address bus A and address/data bus AD. The 8 MSBs of the memory address are sent over the A bus, and 8 LSB_S of the memory address over AD bus. Since, the AD bus is needed to transfer data during subsequent clock cycles, it is used in time-multiplexed mode. Therefore, it has to be made available to carry data during T_2 and T_3. To accomplish this the microprocessor sends an address latch enable singal ALE to latch the 8 LSB_S of the memory address either in the memory or an external latch so that the complete 16-bit memory address may be available in the subsequent clock cycles. 16-bit memory address is needed by the memory to obtain the opcode from the given memory address. During T_2, AD bus becomes ready to carry data. In T_2 the microprocessor makes $\overline{RD}$ low. Now, memory gets the opcode from the specified memory location and places it on the data bus. During T_3, the opcode is placed in the instruction register, IR which is within the microprocessor. The memory is disabled

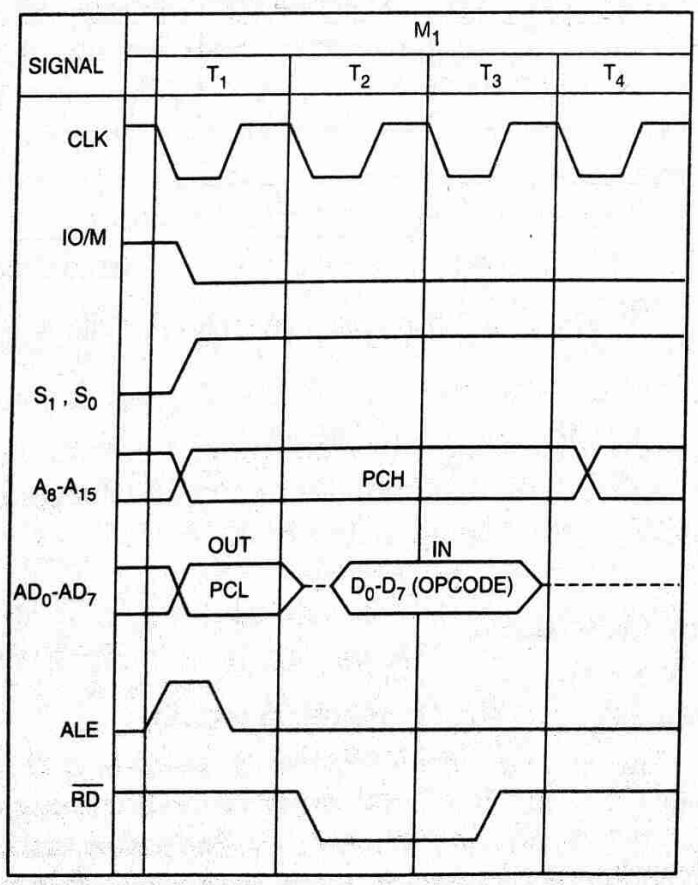

Fig. 3.9 Timing Diagram for Opcode Fetch Operation

when $\overline{RD}$ goes high during T_3. The fetch cycle is completed by T_3. The opcode is decoded in T_4. The microprocessor examines READY signal in T_2. If it is high, the microprocessor enters into T_3 state. If it is low, the microprocessor inserts a wait state in between T_2 and T_3.

If an instruction is one byte long, only one machine cycle as shown in Fig. 3.9, is required to fetch and execute the instruction. Examples of one byte long instructions are MOV, SUB, ADD, RAL, etc. As the operands are in the general purpose registers, the decoding of the operation code and its execution takes only one clock cylce, T_4. If an instruction is two or three bytes long, it requires more machine cycles. The first machine cycle M_1 is for fetching the opcode from the memory. Subsequent machine cycles M_2, M_3 etc. are required either to read data or address from the memory or I/O devices or to write data into the memory or I/O devices. Figure 3.10 shows the timing diagram for a two byte instruction MVI r, data. The first machine cycle M_1 shows fetch cycle, and M_2 a data read cycle.

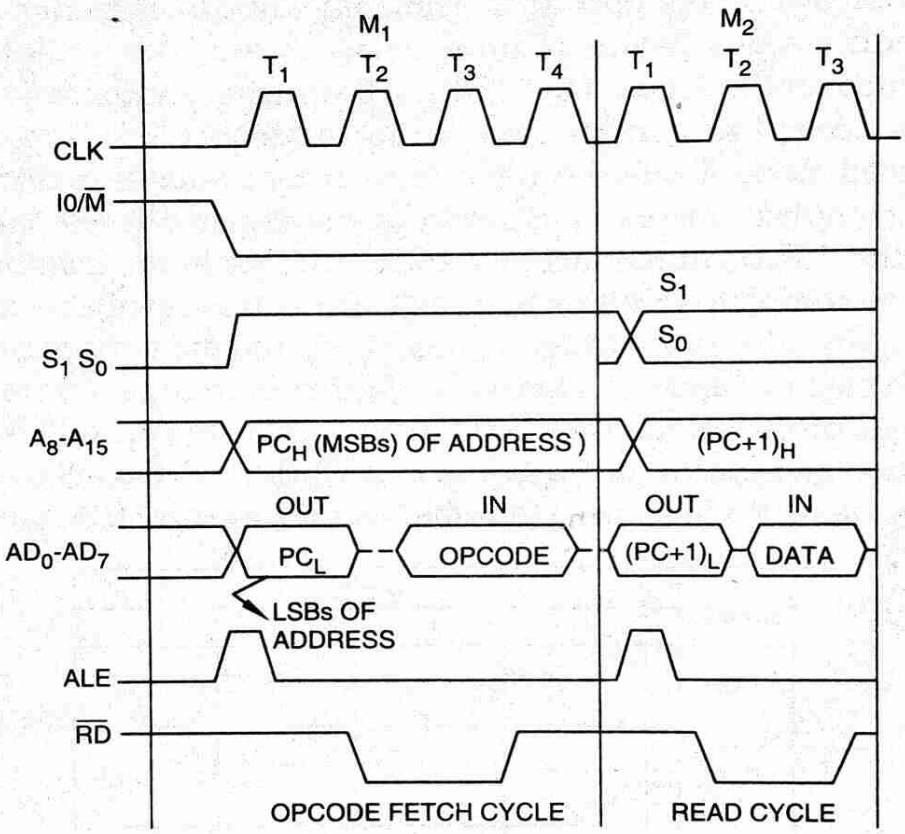

Fig. 3.10 Timing Diagram for MVI r, Data

3.3.2 Memory Read

In a memory read cycle the microprocessor reads the content of a memory location. The content is then placed either in the accumulator or any other register of the CPU. Now, let us take an example of two byte long instruction.

MVI A, 05

In the coded form it is written as

3E, 05

where 3E is opcode for MVI A instruction and 05 is data.

This instruction requires two machine cycles, M_1 and M_2. The first machine cycle M_1 is to fetch the operation code 3E from the memory. The timing diagram for opcode fetch operation has already been shown in Fig. 3.9. The second machine cycle M_2 is for reading the data (05) from the memory as shown in Fig. 3.11. It is a memory read cycle. IO/$\overline{M}$ goes low indicating that the address is for memory. S_1 and S_0 are set to 1 and 0

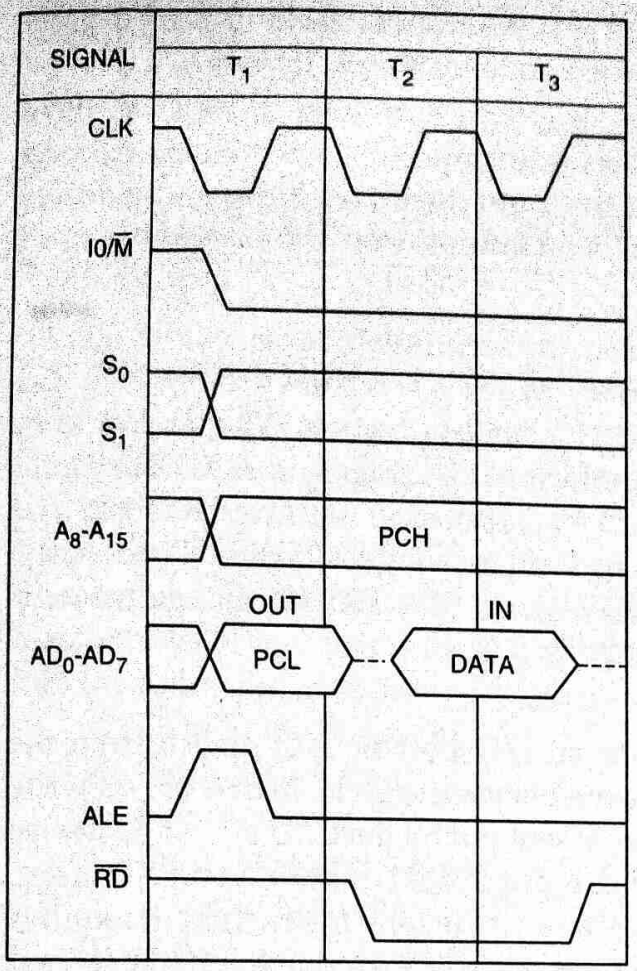

Fig. 3.11 Timing Diagram for Memory Read Operation

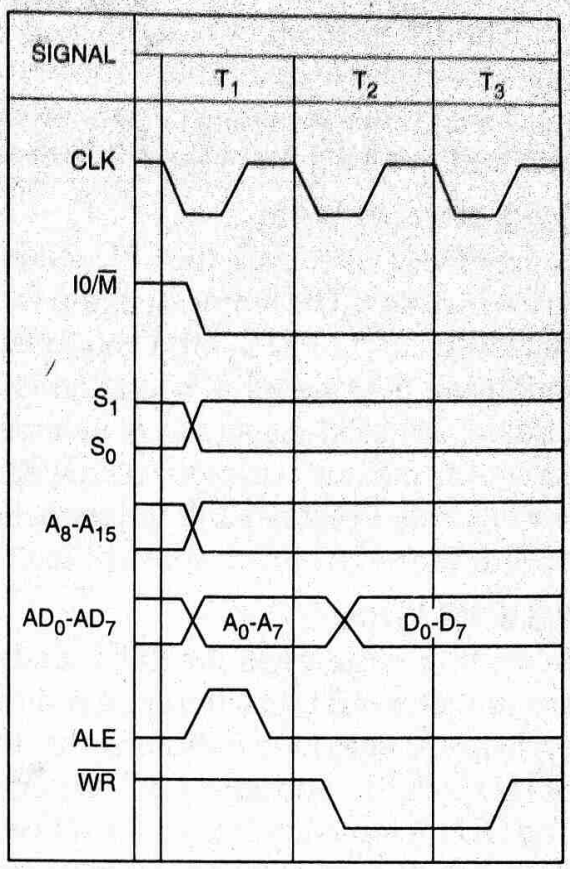

Fig. 3.12 Timing Diagram for Memory Write Operation

respectively for read operation, see Table 3.1. On the address lines $A_8 - A_{15}$, the 8 MSBs of the memory address of the data (05) are sent. During T_1, 8 LSBs of the memory address of the data are sent on $AD_0 - AD_7$ During T_2, 8 LSBs of the address is latched and $AD_0 - AD_7$ are made free for data transmission. $\overline{RD}$ goes low in T_2 to enable the memory for read operation. Now, data is placed on data bus. During T_3 the data enters into the CPU. In T_3, $\overline{RD}$ goes high and disables the memory. For MVI A, 05 instruction the data enters into the accumulator. The timing diagram for an opcode fetch cycle and a memory read cycle are same except the status signals S_1 and S_0. MR cycle consists of only three clock cycles.

Now, consider a three byte long instruction, for example; LXI H, 2500H. This instruction requires three machine cycles, one fetch cycle and two consecutive memory read cycles. The 1st machine cycle is to fetch the opcode, the 2nd machine cycle to read 8 LSBs of the 16-bit data (2500), and the 3rd machine cycle to read 8 MSBs of the 16-bit data (2500). The instruction LDA 2400H requires four machine cycles. The 1st machine cycle is to fetch the operation code, the 2nd machine cycle for reading the 8LSBs of the memory address (2400H), the third machine cycle for reading 8 MSBs of the memory address. In the fourth machine cycle the microprocessor sends the memory address 2400H to get its content into the accumulator.

3.3.3 I/O Read

In an I/O read cycle the microprocessor reads the data available at an input port or input device. The data is placed in the accumulator. An I/O read cycle is similar to memory read cycle. The only difference between a memory ready cycle and I/O read cycle is that signal $IO/\overline{M}$ goes high in case of I/O read. It indicates that the address on the A-bus is for an input device. In timing diagram all other signals will remain same as

shown in Fig. 3.11. In case of I/O device or I/O port the address is only 8-bits long and therefore, the address of I/O device is duplicated on both A and AD buses.

The IN instruction is used for I/O read. It is two byte long. It requires three machine cycles for execution. The first machine cycle is opcode fetch cycle, the second machine cycle is a memory read cycle to read the input device (or input port) address and the third machine cycle is I/O read cycle to read the data from the device or port.

3.3.4 Memory Write

In a memory write cycle the CPU sends data from the accumulator or any other register to the memory. The timing diagram for a memory write cycle is shown in Fig. 3.12. The status signals S_0 and S_1 are 1 and 0 respectively for write operation. $\overline{WR}$ goes low in T_2 indicating that the write operation is to be performed. During T_2 the AD-bus is not disabled as it is done in case of memory or I/O read operation. But the data to be sent out to the memory is placed on the AD-bus. As soon as $\overline{WR}$ goes high in T_3 the write operation is terminated. The instructions MOV M, A; STA 2500H etc. use memory write cycle.

3.3.5 I/O Write

In an I/O write cycle the CPU sends data to an I/O port or I/O device from the accumulator. An I/O write cycle is similar to a memory write cycle. In case of I/O write cycle IO/$\overline{M}$ goes high indicating that the address sent out by the CPU is for I/O device or I/O port. The address of an I/O port or device is duplicated on both A and AD buses. The OUT instruction is used for I/O write. It is a two byte long instruction. It requires three machine cycles. The first machine cycle is for opcode fetch operation, the second machine cycle is a memory read cycle for reading the I/O device address from the memory and the third machine cycle is an I/O write cycle for sending data to the I/O device.

PROBLEMS

1. What are the various registers of 8085? Discuss their function.
2. Discuss the function of ALU of 8085.
3. What are various status flags provided in 8085? Discuss their roles.
4. Discuss the function of the following signals of 8085:
   IO/$\overline{M}$, INTR, $\overline{INTA}$, HOLD, HLDA and READY.
5. Discuss the function of the following signals of 8085:
   $\overline{RD}$, $\overline{WR}$, ALE, S_0 and S_1.
6. Discuss instruction cycle, machine cycle and state.
7. Discuss fetch operation and execute operation.
8. Draw and explain the timing diagram for fetch operation.
9. Draw and explain the timing diagram for memory read operation.
10. Draw and explain the timing diagram of memory write operation.
11. Draw and explain the timing diagram for the instruction MVI r, data.
12. Draw and explain the timing diagram for I/O read operation.
13. Draw and explain the timing diagram for I/O write operation.
14. How many machine cycles are required by the following instructions of Intel 8085? Justify your answers.
    MOV r_1, r_2; MVI r, data; MOV r, M; LXI rp, data; LDA addr; and IN addr.
15. Explain the requirement of a program counter, stack pointer and status flags in the architecture of Intel 8085 microprocessor.

Figure 7.7 show the interfacing of an A/D converter to transfer data employing interrupt driven data transfer scheme. The microprocessor sends first the start of conversion signal, S/C to the A/D converter. Thereafter, the microprocessor executes its main program. A/D converter is a slow device compared to a microprocessor. It takes some time to convert analog signal to its equivalent digital quantity. When A/D converter completes the task of conversion, it makes an end of conversion signal, E/C high. The E/C signal is connected to an interrupt line of the microprocessor. When interrupt line goes high, the microprocessor takes all necessary steps to transfer data from the A/D converter. After completing the data transfer the micro-processor returns back to execute the main program that it was executing prior to the interrupt.

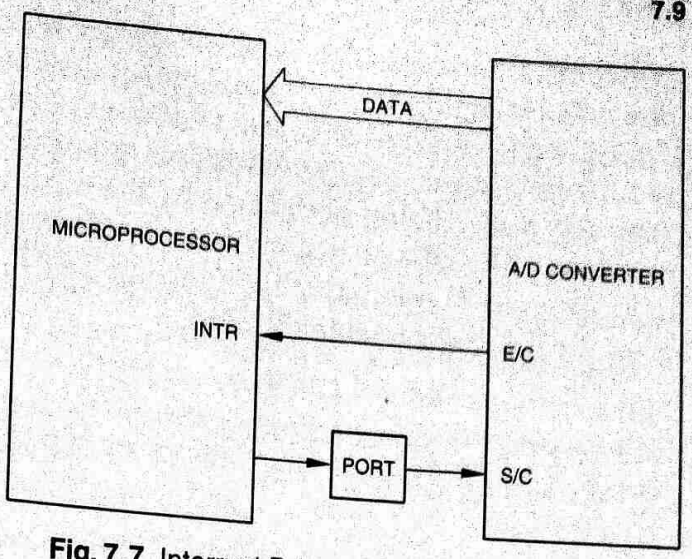

Fig. 7.7 Interrupt Driven Data Transfer Scheme for an A/D Converter

7.4.4 Multiple Interrupts

In a microcomputer system several I/O devices can use interrupt driven data transfer scheme. While interfacing I/O devices using interrupts the following situations may arise:

1. A microprocessor may have only one interrupt level and several I/O devices are to be connected to it.
2. A microprocessor may have several interrupt levels and one I/O device is to be connected to each interrupt level.

The schemes which are used to tackle such situations are described in the following subsections:

Several I/O Devices Connected to a Single Interrupt Level. When several I/O devices are to be connected to a single interrupt level, they are connected through interrupt controller, 8259. Up to 8 I/O devices can be connected to the microprocessor through an8259. If more than 8 I/O devices are to be connected, more 8259 ICs are used in series. The 8259 has been described in detail in Section 7.9.

One Device Connected to Each Level of Interrupt. When a microprocessor has several interrupt levels and the number of I/O devices is equal to or less than the interrupt levels, one device may be connected to each level of interrupt. In this scheme when a device interrupts the microprocessor, the microprocessor immediately knows which device has interrupted. The processor automatically transfers its program to a specific memory location that has been assigned to the interrupt level. It executes ISS for the device which has interrupted. Such an interrupt scheme is known as *vectored interrupt*.

7.5 INTERRUPTS OF INTEL 8085

The Intel 8085 has five interrupt inputs namely TRAP, RST 7.5, RST 6.5, RST 5.5 and INTR. The TRAP has the highest priority, followed by RST 7.5, RST 6.5 and RST 5.5. The INTR has the lowest priority. When interrupts are to be used, they are enabled by software using the instruction EI (Enable Interrupt) in the main program. Figure 7.8

shows the schematic diagram of interrupts of Intel 8085. The instruction EI sets the interrupt enable flip-flop to enable the interrupts. The use of the instruction EI enables all the interrupts. The instruction DI (Disable Interrupt) is used to disable interrupts. In certain situation it may be desired to prevent the occurrence of interrupts while a particular task is being performed by the microprocessor. This can be done using DI instruction. The DI instruction resets the interrupt enable flip-flop and disables all the interrupts except nonmaskable interrupt TRAP, see Fig. 7.8. The system RESET also resets the Interrupt Enable flip-flop.

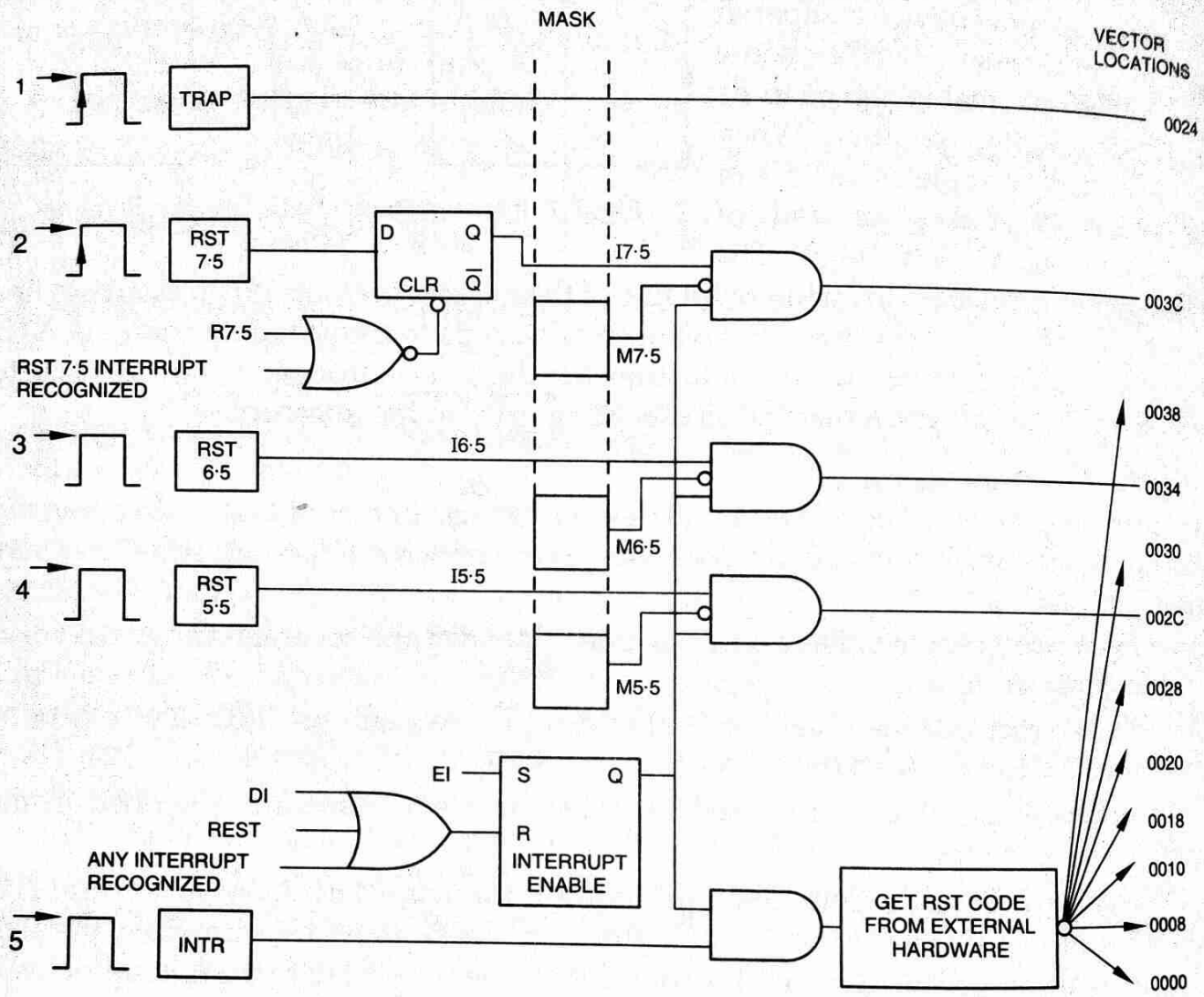

Fig. 7.8 Schematic Diagram of 8085 Interrupts

When an interrupt line goes high processor completes its current instruction and saves program counter on the stack. It also resets Interrupt Enable flip-flop before taking up ISS so that the occurrence of further interrupts by other devices is prevented during the execution of ISS. All the interrupts except TRAP are disabled by resetting the Interrupt Enable flip-flop.

The resetting of this flip-flop can be done in one of the three ways : by software using instruction DI, system reset or by recognition of an interrupt request.

Before the program returns back from ISS to the main program all the interrupts are to be enabled again. This is done using instruction EI in ISS before using the instruction RET.

At many occasions the programmer may like to prevent the occurrence of a few of several interrupts while microprocessor is performing certain tasks. This is done by masking off those interrupts which are not required to occur when certain task is being performed. The interrupts which can be masked off (*i.e.*, made ineffective) are called

maskable interrupts. Masking is done by software. The Intel 8085 has two categories of interrupts : maskable and nonmaskable. The TRAP is a nonmaskable interrupt. It need not be enabled. It cannot be disabled. It is not accessible to user. It is used for emergency situation such as power failure and energy shut-off. RST 7.5, RST 6.5 and RST 5.5 are maskable interrupts.

7.5.1 Hardware and Software Interrupts

Interrupts caused by I/O devices are called *hardware interrupt*. The normal operation of a microprocessor can also be interrupted by abnormal internal conditions or special instructions. Such an interrupt is called a *software interrupt*. RST n instructions of the 8085 are used for software interrupt. When RST n instruction is inserted in a program, the program is executed upto the point where RST n has been inserted. This is used in debugging of a program. See an example in Section 5.10.

The internal abnormal or unusual conditions which prevent the normal processing sequence of a microprocessor are also called exceptions. For example, divide by zero will cause an exception. Intel literatures do not use the term exception, whereas Motorola literatures use the term exception. Intel includes exception in software interrupt.

When several I/O devices are connected to INTR interrupt line, an external hardware is used to interface I/O devices. The external hardware circuit generates RST n codes to implement the multiple interrupt scheme. This will be discussed later on in this chapter.

7.5.2 Interrupts Call-Locations

When an interrupt occurs, the program is transferred to a specific memory location. Then the monitor transfers the program from the specific memory location to a memory location in RAM, from where the user can write the program for interrupt service sub-routine (ISS). For TRAP, RST 7.5, 6.5 and 5.5 the program is automatically transferred to specific memory locations without any external hardware. The necessary hardware is already provided within 8085. The specific memory locations for these interrupts are as follows:

Interrupt	*Call-Location in Hex*
TRAP	0024
RST 7.5	003C
RST 6.5	0034
RST 5.5	002C

An interrupt for which hardware automatically transfers the program to a specific memory location is known as *vectored interrupt*.

INTR CALL Locations. There are 8 numbers of CALL-locations for INTR interrupt. Table 7.4 shows CALL-locations, RST n instructions and corresponding hex-code. For INTR, external hardware is used to transfer program to specific CALL-location. The hardware circuit generates RST codes for this purpose. The INTR line is sampled by the microprocessor in the last state of the last machine cycle of each instruction. When INTR is high, the microprocessor saves the contents of the program counter on the stack and then sends an interrupt acknowledge signal, INTA to the external hardware. In response to INTA the external hardware generates a RST n code. When microprocessor receives this code, it transfers program to the corresponding CALL-location. Upto 8 number of I/O devices can be connected to INTR through an

external hardware. Priority can be assigned to I/O devices connected to INTR through external hardware or interrupt controller. The external hardware recognizes which I/O device has interrupted and it generates proper RST code that causes microprocessor to take up ISS for that particular I/O device. An external hardware can be built employing prority encoder and a tri-state buffer, see Ref. 9. Alternatively, Priority Interrupt Controller 8214 can be used. Programmable interrupt controller 8259 is more versatile, and persent trend is to use programmable interrupt controller. INTA is used to activate 8259 or some other hardware. The 8259 has been described later on in this chapter.

7.5.3 RST 7.5, 6.5 and 5.5

RST 7.5, RST 6.5 and RST 5.5 are maskable interrupts. These interrupts are enabled by software using instructions EI and SIM (Set Interrupt Mask). The execution of the instruction SIM enables/disables interrupts according to the bit pattern of the accumulator. Figure 7.9 shows accumulator contents for SIM instruction.

Bits 0 - 2 set/rest the mask bits of interrupt mask register for RST 5.5, 6.7 and 7.5. Bit 0 is for RST 5.5 mask, bit 1 for RST 6.5 mask and bit 2 for RST 7.5 mask. If a bit is set to 1 the corresponding interrupt is masked off (disabled). If it is set to 0, the corresponding interrupt is enabled. Bit 3 is set to 1 to make bits 0 - 2 effective. Bit 4 is an additional control for RST 7.5. If it is set to 1, the flip-flop for RST 7.5 is reset. Thus RST 7.5 is disabled regardless of whether bit 2 for RST 7.5 is 0 or 1.

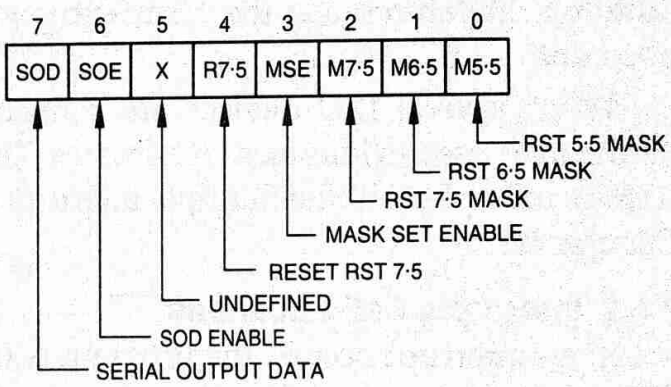

Fig. 7.9 Accumulator Content for SIM

Table 7.4 CALL-Locations and Hex-Codes for RST n

RST n	*Hex-Code*	*CALL-Locations*
RST 0	C7	0000
RST 1	CF	0008
RST 2	D7	0010
RST 3	DF	0018
RST 4	E7	0020
RST 5	EF	0028
RST 6	F7	0030
RST 7	FF	0038

Bits 6 and 7 are for serial data output. The instruction SIM is also used for serial data transmission. Bit 6 is to enable SOD. If SIM instruction is executed the content of the 7th bit of the accumulator is output on SOD line of the microprocessor. The content of bit 7 may be either high (1) or low (0).

The instruction DI disables all the interrupts. This is not always desired. When the processor is performing a particular task, it may be desired to prevent the occurrence of a few of the several interrupts. In such a situation the interrupts which are not desired to occur may be masked off using SIM instruction.

Triggering Levels. TRAP is edge as well as level triggered. This means that TRAP should go high and stay high until it is acknowledged. In this way false triggering caused by noise and transients is avoided. TRAP has also a flip-flop which is not shown in Fig. 7.8. The flip-flop is cleared when interrupt is acknowledged so that future interrupt may be entertained.

RST 7.5 is positive edge triggered interrupt. It can be triggered with a short duration pluse. RST 6.5 and 5.5 are level triggered interrupts. Triggering level for RST 6.5 and 5.5 has to be kept high until the microprocessor recognises these interrupts. If the processor does not recognise these interrupts immediately, their triggering level should be held by external hardware.

Example 1. Enable all the interrupts of Intel 8085.

The content of the accumulator for instructions SIM to enable RST 7.5, 6.5 and 5.5 are programmed as follows.

7	6	5	4	3	2	1	0
SOD	SOE	X	R7.5	MSE	M7.5	M6.5	M5.5
0	0	0	0	1	0	0	0 = 08

Bits 0, 1 and 2 are set to 0 to enable RST 7.5, 6.5 and 5.5. Bit 3 is set to make bits 0, 1 and 2 effective. Bit 4 is set to 0 to enable RST 7.5 as it is an additional control for RST 7.5.

Instructions

Mnemonic	*Operands*	*Comments*
EI		Enable all interrupts.
MVI	A, 08	Get accumulator bit pattern to enable RST 7.5, 6.5 and 5.5.
SIM		Enable RST 7.5, 6.5 and 5.5.

Instruction EI and SIM both are essential to enable RST 7.5, 6.5 and 5.5. In addition to RST 7.5, 6.5 and 5.5, the instruction EI also enables INTR.

Example 2. Enable RST 6.5 and disable RST 7.5 and 5.5.

The contents of the accumulator to enable RST 6.5 and disable RST 7.5 and 5.5 are:

7	6	5	4	3	2	1	0
SOD	SOE	X	R7.5	MSE	M7.5	M6.5	M5.5
0	0	0	1	1	1	0	1 = 1D

Bit 1 is set to 0 to enable RST 6.5.

Bits 0 and 2 are set to 1 to mask off (disable) RST 5.5 and 7.5 respectively. Bit 4 which is an addition control for RST 7.5 is set to 1 to disable RST 7.5.

Bit 3 is set to 1 to make bits 0, 1 and 2 effective.

Instructions

EI	Enable interrupts.
MVI A, 1D	Accumulator bit pattern to enable RST 6.5 and mask off RST 7.5 and 5.5.
SIM	Enable RST 6.5 and disable RST 7.5 and 5.5.

Example 3. Use RST 7.5 to interrupt Intel 8085.

For laboratory testing a very simple program, in which microprocessor is moving in a loop, has been considered. The interruption is made by applying a rising pulse at RST 7.5 terminal manually.

MAIN PROGRAM

Memory address	*Machine Codes*	*Labels*	*Mnemonics*	*Operands*	*Comments*
FC00	FB		EI		Enable all interrupts.
FC01	3E, 08		MVI	A, 08	Get accumulator bit pattern to enable RST 7.5, 6.5 and 5.5.

FC03	30		SIM		
FC04	21, 50, FC	LOOP	LXI	H, FC50	Memory address of 1st number
FC07	7E		MOV	A, M	Get 1st number.
FC08	23		INX	H	
FC09	86		ADD	M	Add 1st and 2nd number.
FC0A	32, 52, FC		STA	FC52	Sum in FC52
FC0D	C3, 04, FC		JMP	LOOP	Jump to LOOP

DATA
FC50 — 05
FC01 — 02
SUM
FC52 — 07

In the beginning of the program all interrupts have been enabled as explained in Example 1. There is a simple program from FC04 to FC0C, to add two numbers and to store their sum in the memory location FC03. The program moves in a loop and repeats the same task again and again.

Now, interrupt the microprocessor by applying a pulse to RST 7.5 line. For this purpose make contact of RST 7.5 line to ground terminal, then to 5 V supply. Now, the program will jump to the memory location 003C which is the specific memory location for RST 7.5. The manufacturer of the Vinytics microprocessor kit has given the following monitor program:

003C C3, BD, FF JMP FFBD

Therefore, the monitor transfers the program from 003C to the memory location FFBD. Now, the user has to transfer the program from FFBD to a memory location in RAM from which service subroutine for RST 7.5 has been written.

User's JMP Instruction

Transfer program from FFBD to FC20.

FFBD C3, 20, FC JMP FC20 ; Transfer program from FFBD to FC20.

FC20 is the starting address of the interrupt service subroutine.

Interrupt Service subroutine

Memory address	Machine Codes	*Mnemonics*	*Operands*	Comments
FC20	3A, 00, FD	LDA	FD00	Get content of FD00.
FC23	32, 30, FC	STA	FC30	Store it in FC30.
FC26	2A, 01, FD	LHLD	FD01	Get contents of FD01 and FD02.
FC29	22, 31, FC	SHLD	FC31	Store them in FC31 and FC32.
FC2C	FB	EI		Enable interrupts.
FC2D	C9	RET		Return to main program.

DATA
FD00 — 01
FD02 — 02
FD03 — 05

Result
FC30 — 01
FC31 — 02
FC32 — 05

The service subroutine is a simple program to transfer the contents of memory locations FD00, FD01 and FD02 to memory locations FC30, FC31 and FC32 respectively. When interrupt occurs the microprocessor transfers the program from main program to memory location 003C. Then the program is transferred to FFBD by the monitor. From FFBD the program is transferred to service subroutine. After performing the task of service subroutine the program goes back to the main program. The processor disables all the interrupts before it enters service subroutine. Therefore, at the end of service subroutine instruction EI is used to enable interrupts so that further interrupts will be recognised. After making interrupt see the result in FC30 – FC32.

Similarly, user can try with RST 6.5 and 5.5. The monitor program for these interrupts for Vinytics kit are:

0034 C3, B7, FF,	JMP	FFB7	It is for RST 6.5
002C C3, 09, 05	JMP	0509	It if for RST 5.5.

As RST 6.5 and 5.5 are level triggered, to make an interrupt the interrupt line has to be kept simply high till the processor recognises the interrupt. For laboratory testing as explained above the user may simply make contact of interrupt line to $5V_{d.c}$ terminal on the board.

Pending Interrupts. When one interrupt request is being served, other interrupt may occur resulting in a pending request. When more than one interrupts occur simultaneously, the interrupt having higher priority is served and the interrupts with lower priority remain pending. As 8085 contains several interrupt lines; an interrupt may occur while other interrupt is being served resulting in a pending interrupt. The 8085 has an instruction RIM using which the programmer can know the current status of the pending interrupts. This instruction gives the current status of only maskable interrupts. In case of INTR the processor reads its status in the last state of the last machine cycle of an instruction. When the processor returns to the main program after serving interrupt service subroutine it knows the status of INTR line as soon as it makes further execution of the main program.

Figure 7.10 shows accumulator contents after the execution of instruction RIM. Bits 0-2 are for interrupt masks; 1 = masked. Bit 3 indicates interrupt enable flag; 1 = enabled. Bits 4-6 indicate pending interrupts; 1 = pending. Bit 7-serial input data, if any.

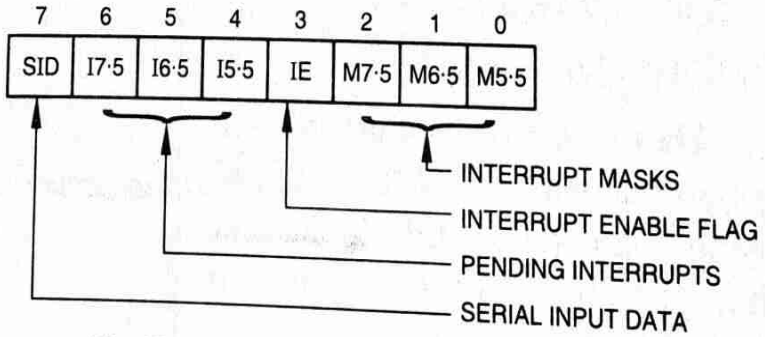

Fig. 7.10 Accumulator Content after the Execution of RIM

After executing the interrupt service subroutine the processor checks whether any other interrupt is pending using RIM instruction. If an interrupt is pending the processor executes its interrupt service subroutine before it returns to the main program.

7.6 INTERFACING DEVICES AND I/O DEVICES

INSTRUCTION SET OF INTEL 8085

4.1 INTRODUCTION

An instruction is a command given to the computer to perform a specified operation on given data. The instruction set of a microprocessor is the collection of the instructions that the microprocessor is designed to execute. The instructions described in this chapter are of INTEL 8085. These instructions are of Intel Corporation. They cannot be used by other microprocessor manufacturers. The programmer can write a program in assembly language using these instructions. These instructions have been classified into the following groups:

1. Data Transfer Group
2. Arithmetic Group
3. Logical Group
4. Branch Control Group
5. I/O and Machine Control Group.

Data Transfer Group. Instructions which are used to transfer data from one register to another register, from memory to register or register to memory, come under this group. Examples are: MOV, MVI, LXI, LDA, STA etc. When an instruction of data transfer group is executed, data is transferred from the source to the destination without altering the contents of the source. For example, when MOV A, B is executed the content of the register B is copied into the register A, and the content of register B remains unaltered. Similarly, when LDA 2500 is executed the content of the memory location 2500 is loaded into the accumulator. But the content of the memory location 2500 remains unaltered.

Arithmetic Group. The instructions of this group perform arithmetic operations such as addition, subtraction; increment or decrement of the content of a register or memory. Examples are: ADD, SUB, INR, DAD etc.

Logical Group. The instructions under this group perform logical operation such as AND, OR, compare, rotate etc. Examples are: ANA, XRA, ORA, CMP, RAL etc.

Branch Control Group. This group includes the instructions for conditional and unconditional jump, subroutine call and return, and restart. Examples are: JMP, JC, JZ, CALL, CZ, RST etc.

I/O and Machine Control Group. This group includes the instructions for input/output ports, stack and machine control. Examples are : IN, OUT, PUSH, POP, HLT etc.

4.2 INSTRUCTION AND DATA FORMATS

Intel 8085 is an 8-bit microprocessor. It handles 8-bit data at a time. One byte consists of 8 bits. A memory location for Intel 8085 microprocessor is designed to accommodate 8-bit data. If 16-bit data are to be stored, they are stored in consecutive memory locations. The address of a memory location is of 16 bits i.e. 2 bytes.

The various techniques to specify data for instructions are:

(i) 8-bit or 16-bit data may be directly given in the instruction itself.

(ii) The address of the memory location, I/O port or I/O device, where data resides, may be given in the instruction itself.

(iii) In some instructions only one register is specified. The content of the specified register is one of the operands. It is understood that the other operand is in the accumulator.

(iv) Some instructions specify two registers. The contents of the registers are the required data.

(v) In some instructions data is implied. The most instructions of this type operate on the content of the accumulator.

Due to different ways of specifying data for instructions, the machine codes of all instructions are not of the same length.

There are three types of the Intel 8085 instructions as described below:

(1) Single-Byte Instruction
(2) Two-Byte Instruction
(3) Three-Byte Instruction

This has already been discussed in Chapter 3; see details.

4.3 ADDRESSING MODES

Each instruction requires certain data on which it has to operate. It has already been explained that there are various techniques to specify data for instructions. These techniques are called *addressing modes*. Intel 8085 uses the following addressing modes:

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

4.3.1 Direct Addressing

In this mode of addressing the address of the operand (data) is given in the instruction itself.

Examples are:

(1) STA 2400 H — Store the content of the accumulator in the memory location 2400 H.

32, 00, 24 — The above instruction in the code form.

In this instruction 2400H is the memory address where data is to be stored. It is given in the instruction itself. The 2nd and 3rd bytes of the instruction specify the address of the memory location. Here, it is understood that the source of the data is accumulator.

(2) IN 02 — Read data from the port C.

DB, 02 — Instruction in the code form.

In this instruction 02 is the address of the port C of an I/O port from where the data is to be read. Here, it is implied that the destination is the accumulator. The 2nd byte of the instruction specifies the address of the port.

4.3.2 Register Addressing

In register addressing mode the operand is in one of the general purpose registers. The opcode specifies the address of the register(s) in addition to the operation to be performed. Examples are:

(1) MOV A, B	Move the content of register B to register A.
78	The instruction in the code form.
(2) ADD B	Add the content of register B to the content of register A.
80	The instruction in the code form.

In Example 1 the opcode for MOV A, B is 78H. Besides the operation to be performed the opcode also specifies source and destination registers. The opcode 78H can be written in binary form as 01111000. The first two bits, i.e. 01 are for MOV operation, the next three bits 111 are the binary code for register A, and the last three bits 000 are the binary code for register B.

In Example 2 the opcode for ADD B is 80H. In this instruction one of the operands is register B (its content is one of the data) which is indicated in the instruction itself. In this type of instruction (arithmetic group) it is understood that the other operand is in the accumulator. The opcode 80H in the binary form is 10000000. The first five bits, *i.e.* 10000 specify the operation to be performed, i.e. ADD. The last three bits 000 are the binary code for register B for 8085 microprocessor.

4.3.3 Register Indirect Addressing

In this mode of addressing the address of the operand is specified by a register pair. Examples are:

(1) LXI H, 2500 H	Load H-L pair with 2500 H.
MOV A, M	Move the content of the memory location, whose address is in H-L pair (*i.e.* 2500 H) to the accumulator
HLT	Halt.

In the above program the instruction MOV A, M is an example of register indirect addressing. For this instruction the operand is in the memory. The address of the memory is not directly given in the instruction. The address of the memory resides in H-L pair and this has already been specified by an earlier instruction in the program, i.e. LXI H, 2500 H.

(2) LXI H, 2500 H	Load the H-L pair with 2500 H.
ADD M	Add the content of the memory location, whose address is in H-L pair (i.e. 2500 H), to the content of the accumulator.
HLT	Halt.

In this program the instruction ADD M is an example of register indirect addressing.

4.3.4 Immediate Addressing

In immediate addressing mode the operand is specified within the instruction itself, examples are:

(1) MVI A, 05	Move 05 in register A.
3E, 05	The instruction in the code form.
(2) ADI 06	Add 06 to the content of the accumulator.
C6, 06	The instruction in the code form.

In these instructions the 2nd byte specifies data.

(3) LXI H, 2500 is an example of immediate addressing. 2500 is 16-bit data which is given in the instruction itself. It is to be loaded into H-L pair.

4.3.5 Implicit Addressing

There are certain instructions which operate on the content of the accumulator. Such instructions do not require the address of the operand. Examples are: CMA, RAL, RAR etc.

4.4 STATUS FLAGS

There is a set of five flip-flops which indicate status (conditions) arising after the execution of arithmetic and logic instructions. It has already been discussed in Section 3.1.3.

4.5 SYMBOLS AND ABBREVIATIONS

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

Symbol/Abbreviations	*Meaning*
addr	16-bit address of the memory location.
data	8-bit data.
data 16	16-bit data.
r, r_1, r_2	One of the registers A, B, C, D, E, H or L,
A, B, C, D, H, L	8-bit register
A	Accumulator
H-L	Register pair H-L
B-C	Register pair B-C
D-E	Register pair D-E
PSW	Program Status Word
M	Memory whose address is in H-L pair
H	Appearing at the end of a group of digits specifies hexadecimal, e.g. 2500H.
rp	One of the register pairs. The representation of a register pair is made as described below: B represents B-C pair, B is high order register and C low order register. D represents D-E pair, D is high order register and E is low order register. H represents H-L pair, H is high order register and L low order register. SP represents 16-bit stack pointer, SPH is high order 8 bits and SPL low order 8 bits of register SP.
rh	The high order register of a register pair.
rl	The low order register of a register pair.
PC	16-bit program counter, PCH is high order 8 bits and PCL low order 8 bits of register PC.
CS	Carry status.
[ ]	The contents of a register identified within bracket.
[[ ]]	The content of the memory location whose address is in the register pair identified within brackets.
∧	AND operation
∨	OR operation

∀ or ⊕	EXCLUSIVE-OR
←	Move data in the direction of arrow.
⇔	Exchange contents.

4.6 INTEL 8085 INSTRUCTIONS

Some of Intel 8085 instructions are frequently, some occasionally and some seldom used by the programmer. It is not necessary that one should learn all the instructions to understand simple programs. The beginner can learn about 15 to 20 important instructions such as MOV, MVI, LXI, LDA, LHLD, STA, SHLD, ADD, ADC, SUB, JMP JC, JNC, JZ, JNZ, INX, DCR, CMP etc., and start to understand simple programs given in Chapter 6. While learning programs he can understand new instructions which he has not learnt earlier.

The operation codes (opcodes) are given in Appendix II. The explanations of the most instructions are given in the subsequent subsections.

4.6.1 Data Transfer Group MOV r_1, r_2 (Move data; Move the content of the one register to another)

$[r_1] \leftarrow [r_2]$. States: 4. Flags: none. Addressing: register.Machine cycle: 1.

The content of register r_2 is moved to register r_1. For example, the instruction MOV A, B moves the content of register B to register A. The instruction MOV B, A moves the content of register A to register B. The time for the execution of this instruction is 4 clock period. One clock period is called *State*. No flag is affected.

MOV r, M. (Move the content of memory to register).

[r] ← [[H - L]]. States: 7. Flag none. Addressing: register indirect. Machine cycles: 2.

The content of the memory location, whose address is in H-L pair, is moved to register r.

Example

LXI H, 2000 H	Load H-L pair by 2000H.
MOV B, M	Move the content of the memory location 2000H to register B.
HLT	Halt.

In this example the instruction LXI H, 2000 H loads H-L pair with 2000 H which is the address of a memory location. Then the instruction MOV B, M will move the content of the memory location 2000H to register B.

MOV M, r. (Move the content of register to memory).

[[H-L]] ← [r]. States: 7. Flags: none. Addressing: reg. indirect. Machine cycles: 2.

The content of register *r* is moved to the memory location addressed by H-L pair. For example, MOV M, C moves the content of register C to the memory location whose address is in H-L pair.

MVI r, data. (Move immediate data to register).

[r] ← data. States: 7. Flags: none. Addressing: immediate. Machine cycle: 2.

The 1st byte of the instruction is its opcode. The 2nd byte of the instruction is the data which is moved to register *r*. For example, the instruction MVI A, 05 moves 05 to register A. In the code form it is written as 3E, 05. The opcode for MVI A is 3E and 05 is the data which is to be moved to regiser A.

MVI M, data. (Move immediate data to memory).

[[H - L]] ← data. States: 10. Flags: none. Addressing: immediate/reg. indirect. Machine cycle: 3.

The data is moved to memory location whose address is in H-L pair.

Example

LXI H, 2400H	Load H-L pair with 2400H.
MVI M, 08	Move 08 to the memory location 2400H.
HLT	Halt.

In the above example the instruction LXI H, 2400 H loads H-L pair with 2400 H which is the address of a memory location. Then the instruction MVI M, 08 will move 08 to memory location 2400H. In the code form it is written as 36, 08. The opcode for MVI M is 36 and 08 is the data which is to be moved to the memory location 2400H.

LXI rp, data 16. (Load register pair immediate).

[*rp*] ← data 16 bits, [*rh*] ← MSBs, [*rl*] ← 8 LSBs of data.

States: 10. Flags: none. Addressing: immediate. Machine cycles: 3.

This instruction loads 16-bit immediate data into register pair *rp*. This instruction is for register pair; only high order register is mentioned after the instruction. For example, H in the instruction LXI H stands for H-L pair. Similarly, LXI B is for B-C pair. LXI H, 2500H loads 2500H into H-L pair. H with 2500H denotes that the data 2500 is in hexadecimal. In the code form it is written as 21, 00, 25. The 1st byte of the instruction 21 is the opcode for LXI H. The 2nd byte 00 is 8 LSBs of the data and it is loaded into register L. The 3rd byte 25 is 8 MSBs of the data and it is loaded into register H.

LDA addr. (Load Accumulator direct).

[A] ← [*addr*]. States: 13. Flags: none. Addressing: direct. Machine cycles: 4.

The content of the memory location, whose address is specified by the 2nd and 3rd bytes of the instruction; is loaded into the accumulator. The instruction LDA 2400 H will load the content of the memory location 2400 H into the accumulator. In the code form it is written as 3A, 00, 24. The 1st byte 3A is the opcode of the instruction. The 2nd byte 00 is of 8 LSBs of the memory address. The 3rd byte 24 is 8 MSBs of the memory address.

STA Addr. (Store accumulator direct).

[*addr*] ← [A]. States: 13. Flags: none. Addressing: direct. Machine cycles: 4.

The content of the accumulator is stored in the memory location whose address is specified by the 2nd and 3rd byte of the instruction. STA 2000H will store the content of the accumulator in the memory location 2000H.

LHLD addr. (Load H-L pair direct).

[L] ← [*addr*], [H] ← [*addr* + 1]. States: 16. Flags: none. Addressing: direct. Machine cycles: 5.

The content of the memory location, whose address is specified by the 2nd and 3rd bytes of the instruction, is loaded into register L. The content of the next memory location is loaded into register H. For example, LHLD 2500H will load the content of the memory location 2500 H into register L. The content of the memory location 2501H is loaded into register H.

SHLD addr. (Store H-L pair direct)

[*addr*] ← [L], [*addr* + 1] ← [H]. States : 16. Flags: none. Addressing: direct. Machine cycles: 5.

The content of register L is stored in the memory location whose address is specified by the 2nd and 3rd bytes of the instruction. The content of register H is stored in the next memory location. For example, SHLD 2500H will store the content of register L in the memory location 2500H. The content of register H is stored in the memory location 2501H.

LDAX rp. (LOAD accumulator indirect)

[A] ← [[*rp*]]. States; 7. Flags; none. Addressing: register indirect. Machine cycles: 2.

The content of the memory location, whose address is in the register pair *rp*, is loaded into the accumulator. For example, LDAX B will load the content of the memory location, whose address is in the B-C pair, into the accumulator. This instruction is used only for B-C and D-E register pairs.

STAX rp. (Store accumulator indirect)

[[*rp*]] ← [A]. States: 7. Flags: none. Addressing: register indirect. Machine cycles: 2.

The content of the accumulator is stored in the memory location whose address is in the register pair *rp*. For example, STAX D will store the content of the accumulator in the memory location whose address is in D-E pair. This instruction is true only for register pairs B-C and D-E.

XCHG. (Exchange the contents of H-L with D-E pair)

[H-L]↔ [D-E]. States: 4. Flags: none. Addressing: register. Machine cycles: 1.

The contents of H-L pair are exchanged with contents of D-E pair.

4.6.2 Arithmetic Group

ADD r. (Add register to accumulator)

[A] ← [A] + [*r*]. States: 4. Flags: all. Addressing: register. Machine cycle: 1.

The content of register *r* is added to the content of the accumulator, and the sum is placed in the accumulator.

ADD M. (Add memory to accumulator)

[A] ← [A] + [[H-L]]. States: 7. Flags: all. Addressing: reg. indirect, Machine cycles: 2.

The content of the memory location addressed by H-L pair is added to the content of the accumulator. The sum is placed in the accumulator.

ADC r. (Add register with carry to accumulator.)

[A] ← [A] + [*r*] + [CS]. States: 4. Flags: all. Addressing: register. Machine cycles: 1.

The content of register *r* and carry status are added to the content of the accumulator. The sum is placed in the accumulator.

ADC M. (Add memory with carry to accumulator)

[A] ← [A] + [[H-L]] [CS]. States: 7. Flags: all. Addressing: reg. indirect. Machine cycles: 2.

The content of the memory location addressed by H-L pair and carry status are added to the content of the accumulator. The sum is placed in the accumulator.

ADI data. (Add immediate data to accumulator)

[A] ← [A] + data. States: 7. Flags: all. Addressing : immediate. Machine cycles: 2.

The immediate data is added to the content of the accumulator. The 1st byte of the instruction is its opcode. The 2nd byte of the instruction is data, and it is added to content of the accumulator. The sum is placed in the accumulator. For example, the

instruction ADI 08 will add 08 to the content of the accumulator and place the result in the accumulator. In code form the instruction is written as C6, 08.

ACI data. (Add with carry immediate data to accumulator)

[A] ← [A] + data + [CS]. States: 7. Flags: all. Addressing: immediate. Machine cycles: 2

The 2nd byte of the instruction (which is data) and the carry status are added to the content of the accumulator. The sum is placed in the accumulator.

DAD rp. (Add register paid to H-L pair)

[H-L] ← [H-L] + [*rp*]. States: 10. Flags: CS. Addressing: register. Machine cycles: 3.

The contents of register pair *rp* are added to the contents of H-L pair and the result is placed in H-L pair. Only carry flag is affected.

SUB r. (Subtract register from accumulator)

[A] ← [A] – [*r*]. States: 4 Flags: all. Addressing: register. Machine cycles: 1.

The content of register *r* is subtracted from the content of the accumulator, and the result is placed in the accumulator.

SUB M. (Subtract memory from accumulator).

[A] ← [A] – [[H-L]]. States: 7. Flags: all. Addressing: reg. indirect. Machine cycles: 2

The content of the memory location addressed by H-L pair is subtracted from the content of the accumulator. The result is placed in the accumulator.

SBB r. (Subtract register from accumulator with borrow).

[A] ← [A] – [*r*] – [CS]. States: 4. Flags: all. Addressing: register. Machine cycles: 1.

The content of register *r* and carry status are subtracted from the content of the accumulator. The result is placed in the accumulator.

SBB M. (Subtract memory from accumulator with borrow).

[A] ← [A] – [[H-L]] – [CS]. States: 7. Flags: all. Addressing: reg. indirect. Machine cycles: 2.

The content of the memory location addressed by H-L pair and carry status are subtracted from the content of the accumulator. The result is placed in the accumulator.

SUI data. (Subtract immediate data from accumulator)

[A] ← [A] – data. States: 7. Flags: all. Addressing: immediate. Machine cycles: 2.

The 2nd byte of the instruction is data. It is subtracted from the content of the accumulator. The result is placed in the accumulator. For example, the instruction SUI 05 will subtract 05 from the content of the accumulator and place the result in the accumulator. In the code form the above instruction is written as D6, 05.

SBI data. (Subtract immediate data from accumulator with borrow).

[A] ← [A] – data – [CS]. States: 7. Flags: all. Addressing: immediate. Machine cycles: 2

The data and carry status are subtracted from the content of the accumulator. The result is placed in the accumulator

INR r. (Increment register content)

[*r*] ← [*r*] + 1. States : 4. Flags: all except carry flag. Addressing : register. Machine cycle : 1.

The content of register *r* is incremented by one. All flags except CS are affected.

INR M. (Increment memory content)

[[H-L]] ← [[H-L]] + 1. States: 10. Flags: all except carry flag. Addressing: reg. indirect.

Machine cycles: 3.

The content of the memory location addressed by H-L pair is incremented by one. All flags except CS are affected.

DCR r. (Decrement register content) [r] ← εε [r] − 1. States: 4. Flags: all except carry flag, Addressing: register. Machine cycles: 1.

The content of register *r* is decremented by one. All flags except CS are affected.

DCR M. (Decrement memory content)

[[H-L]] ← [[H-L]] − 1. States: 10. Flags: all except carry flag. Addressing: reg. indirect. Machine cycles: 3.

The content of the memory location addressed by H-L pair is decremented by one. All flags except CS are affected.

INX rp. (increment register pair)

[*rp*] ← [*rp*] + 1. States: 6. Flags: none. Addressing: register. Machine cycles : 1.

The content of the register pair *rp* is incremented by one. No flag is affected.

DCX rp (Decrement register pair)

[*rp*] ← [*rp*] − 1. States: 6. Flags: none. Addressing: register. Machine cycles: 1.

The content of the register pair *rp* is decremented by one. No flag is affected.

DAA. (Decimal adjust accumulator)

States: 4. Flags: all. Machine cycle : 1.

The instruction DAA is used in the program after ADD, ADI, ACI, ADC, etc instructions. After the execution of ADD, ADC, etc instructions the result is in hexadecimal and it is placed in the accumulator. The DAA instruction operates on this result and gives the final result in decimal system. It uses carry and auxiliary carry for decimal adjustment. 6 is added to 4 LSBs of the content of the accumulator if their value lies in between A and F or the AC flag is set to 1. Similarly, 6 is also added to 4 MSBs of the content of the accumulator if their value lies in between A and F or the CS flag is set to 1. All status flags are affected. When DAA is used data should be in decimal numbers.

4.6.3 Logical Group

The instructions of this group perform AND, OR, EXCLUSIVE-OR operations; compare, rotate or take complement of data in register or memory.

ANA r. (AND register with accumulator)

[A] ← [A] <185 > [*r*]. States: 4. Flags : all. Addressing: register. Machine cycles: 1.

The content of register *r* is ANDed with the content of the accumulator, and the result is placed in the accumulator. All status flags are affected. The flag CS is cleared, *i.e.* it is set to 0. Auxiliary carry flag AC is set to 1.

ANA M. (AND memory with accumulator)

[A] ← [A] ∧ [[H-L]]. States: 7. Flags: all. Addressing: reg. indirect. Machine cycles: 2.

The content of the memory location addressed by H-L pair is ANDed with the accumulator. The result is placed in the accumulator. All flags are affected. The CS flag is set to 0 and AC to 1.