## 4.6. INTRODUCTION TO 8085

The *8085 is an 8-bit microprocessor* introduced by Intel in 1976. The 8085 was the follow-on processor to the very successful Intel 8080A processor. The 8085 got its name because it was Intel's first 5 volt microprocessor. The 8085 was 100% software compatible with the 8080A with increased systems performance. The initial 8085's were based on NMOS technology and the later "H" versions were based on HMOS technology. The 8085 used 6,500 transistors.

The *8085 incorporated all the features of the 8224 (clock generator)* and the 8228 (system controller) increasing the level of system integration. The 8085 along with and 8156 RAM and 8355/8755 ROM/PROM constituted a complete system. The 8085 used a multiplexed Data Bus and required the 825X-5 support chips. The address was split between the 8-bit address bus and 8-bit data bus. The on-chip address latch of 8155/8355/8755 memory chips allowed a direct interface with the 8085.

## 4.7. FEATURES OF 8085 MICROPROCESSOR

1. It is an 8 bit microprocessor.
2. It is manufactured with N-MOS technology.
3. It has 16-bit address bus and hence can address up to $2^{16}$ memory locations.

4. The first 8 lines of address bus and 8 lines of data bus are multiplexed.
5. Data bus is a group of 8 lines $D_0 - D_7$.
6. It supports external interrupt request.
7. A 16 bit program counter (PC)
8. A 16 bit stack pointer (SP)
9. Six 8-bit general purpose register arranged in pairs: BC, DE, HL.
10. It requires a signal +5V power supply and operates at 3.2 MHZ clock.
11. It is available in 40 pin DIP (Dual in line package).
12. 8085 is one address microprocessor.

## 4.8. INTERNAL ARCHITECTURE AND FUNCTIONAL BLOCK DIAGRAM OF 8085

The 8085 Architecture follows *the von Neumann architecture* shown in Fig. 4.4, with a 16-bit address bus, and an 8-bit data bus. The 8085 incorporated all the features of the *8224 (clock generator)* and the 8228 (system controller), increasing the level of system integration.

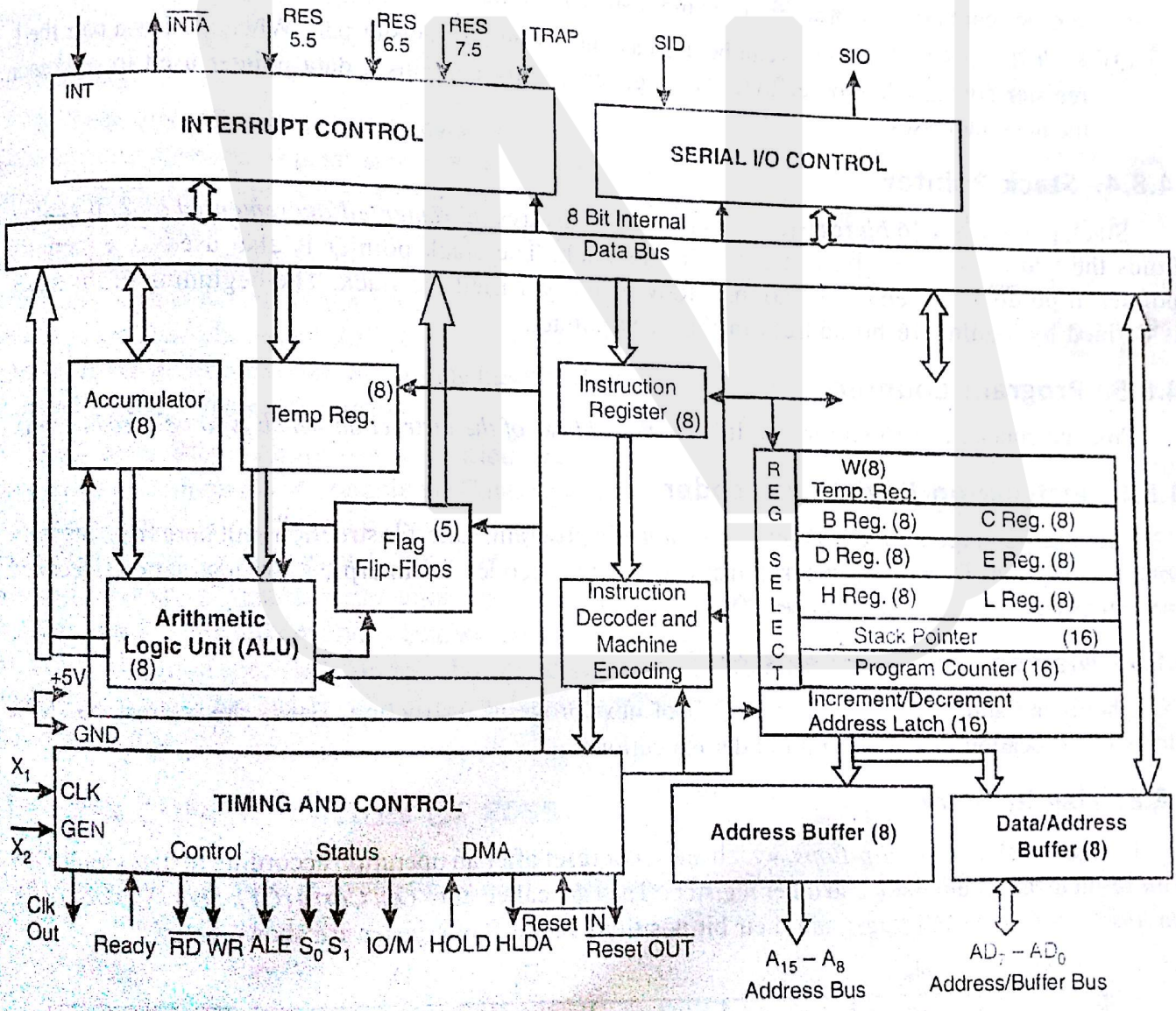



*Fig. 4.4. Functional Block Diagram of Microprocessor 8085*

### 4.8.1. Registers

The 8085 can access $2^{16}$ (= 65,536) individual 8-bit memory locations, or in other words, its address space is 64 kb. Unlike some other microprocessors of its era, it has a separate address space for up to $2^8$ (=256) I/O ports.

It also has a built in register array which are usually labeled *A (Accumulator), B, C, D, E, H, and L*. Further special-purpose registers are the *16-bit Program Counter (PC), Stack Pointer (SP), and 8-bit flag register F*.

### 4.8.2. Accumulator

The accumulator is an 8-bit register that is a part of arithmetic/logic unit (ALU). This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator. The accumulator is also identified as register A.

### 4.8.3. Register Pairs

(1) 8-bit B and 8-bit C registers can be used as one 16-bit BC register pair. When used as a pair, the C register contains low-order byte. Some instructions may use BC register as a data pointer.

(2) 8-bit D and 8-bit E registers can be used as one 16-bit DE register pair. When used as a pair the E register contains low-order byte. Some instructions may use DE register as a data pointer.

(3) 8-bit H and 8-bit L registers can be used as one 16-bit HL register pair. When used as a pair the L register contains low-order byte. HL register usually contains a data pointer used to reference memory addresses.

### 4.8.4. Stack Pointer

Stack pointer is a *16 bit* register. This register is always *incremented/decremented by* 2. It always holds the address of the top most entity in the stack. The stack pointer is also used as a memory pointer. It points to a memory location in R/W memory, called the stack. The beginning of the stack is defined by loading 16-bit address in the stack pointer.

### 4.8.5. Program Counter

Program counter is a 16-bit register. It holds *the address of the instruction which is to be executed next.*

### 4.8.6. Instruction Register/Decoder

Temporary storage for the *current instruction* of a program. Latest instruction sent here from memory prior to execution. Decoder then takes instruction and 'decodes' or interprets the instruction. Decoded instruction then passed to next stage.

### 4.8.7. Memory Address Register

It holds the address (received from PC) of next program instruction. Feeds the address bus with addresses of location of the program under execution.

### 4.8.8. Flag Register

The ALU includes *five flip-flops*, which are set or reset after an operation according to data conditions of the result in the accumulator and other registers. They are called *Zero (Z), Carry (CY), Sign (S), Parity (P), and Auxiliary Carry (AC) flags*; and their bit positions in the flag register are shown below.

| $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
|---|---|---|---|---|---|---|---|
| S | Z | U | AC | U | P | U | CY |

The most commonly used flags are Zero, Carry, and Sign. The microprocessor uses these flags to test data conditions.

**For example**, after an addition of two numbers, if the sum in the accumulator is larger than eight bits, the flip-flop uses to indicate a carry, called the Carry flag (CY) – is set to one.

When an arithmetic operation results in zero, the flip-flop called the Zero (Z) flag is set to one.

### 4.8.9. 8085 System Bus

Typical system uses a number of busses, collection of wires, which transmit binary numbers, one bit per wire. A typical microprocessor communicates with memory and other devices (input and output) using three busses: *Address Bus, Data Bus and Control Bus.*

**1. Address Bus:** One wire for each bit, therefore 16 bits = 16 wires. Binary number carried alerts memory to 'open' the designated box. Data (binary) can then be put in or taken out. The Address Bus consists of 16 wires, therefore 16 bits. Its "width" is 16 bits. A 16 bit binary number allows 216 different numbers, or 32000 different numbers, *i.e.*, 0000000000000000 up to 1111111111111111. Because memory consists of boxes, each with a unique address, the size of the address bus determines the size of memory, which can be used. To communicate with memory the microprocessor sends an address on the address bus, e.g. 0000000000000011 (3 in decimal), to the memory. The memory selects box number 3 for reading or writing data. Address bus is unidirectional, *i.e.*, numbers only sent from microprocessor to memory, not other way.

**2. Data Bus:** Data bus carries 'data', in binary form, between microprocessor and other external units, such as memory. Typical size is 8 or 16 bits. Size determined by size of boxes in memory and microprocessor size helps determine performance of microprocessor. The Data bus typically consists of 8 wires. Data bus used to transmit "data", i.e. information, results of arithmetic, etc. between memory and the microprocessor. Bus is bi-directional. Size of the data bus determines what arithmetic can be done. If only 8 bits wide then largest number is 11111111 (255 in decimal). Therefore, larger numbers have to be broken down into chunks of 255. This slows microprocessor. Data bus also carries instructions from memory to the microprocessor. Size of the bus therefore limits the number of possible instructions to 256, each specified by a separate number.

**3. Control Bus:** Control Bus has various lines which have specific functions for coordinating and controlling microprocessor operations. Control whether memory is being 'written to' (data stored in memory) or 'read from' (data taken out of memory) 1 = Read, 0 = Write. May also include clock line(s) for timing/synchronizing, 'interrupts', 'reset', etc. generally microprocessor has 10 control lines. The Control Bus carries control signals partly unidirectional, partly bi-directional. Control signals are things like "read or write". This tells memory that we are reading from a location, specified on the address bus, or writing to a location specified. Modern day microprocessors, like 80386, 80486 have much larger busses. Typically 16 or 32 bit busses, which allow larger number of instructions, more memory location, and faster arithmetic.

## 4.9. PIN CONFIGURATION OF 8085

Microprocessor 8085 is available in 40 pin DIP package shown in Fig. 4.5. The 8085 has some following properties these are:

1. Single + 5V Supply
2. 4 Vectored Interrupts (One is Non Maskable)
3. Serial In/Serial out Port

4. Decimal, Binary, and Double Precision Arithmetic
5. Direct Addressing Capability to 64K bytes of memory

### 4.9.1. Pin Function Description

The functions of different pins of 8085 are explained in the following manner.

#### $A_8 - A_{15}$ (Pin-21 to 28, Type-Output, 3 State)

**Address Bus:** The *most significant 8 bits of the memory address* or the 8 bits of the I/O address 3 stated during Hold and Halt modes.

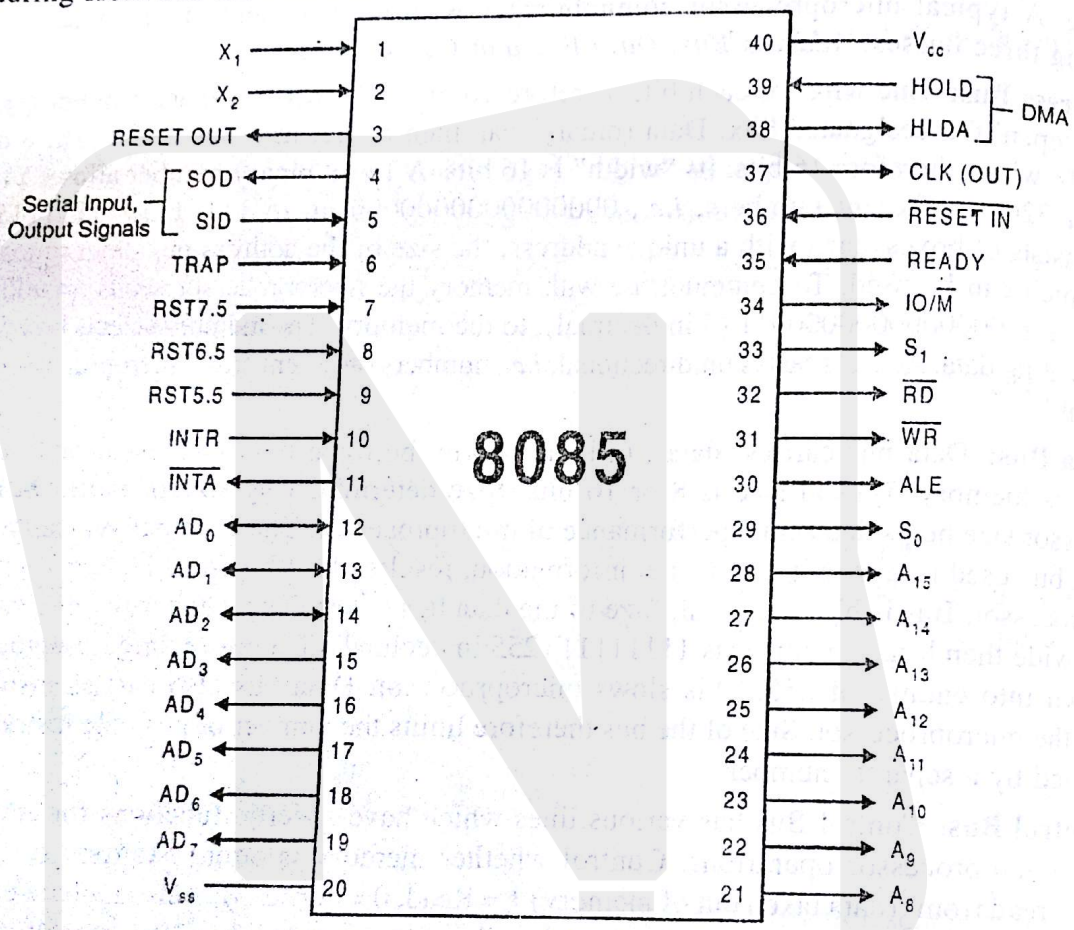



*Fig. 4.5. Pin Diagram of 8085 Microprocessor*

#### $AD_0 - AD_7$ (Pin-12 to 19, Type-Input/output, 3 state)

**Multiplexed Address/Data Bus:** Lower 8 bits of the memory address (or I/0 address) appear on the bus during the first clock cycle of a machine state. It then becomes the data bus during the second and third clock cycles. 3 stated during Hold and Halt modes.

#### ALE (Pin-30, Type-Output)

**Address Latch Enable:** It occurs during the first clock cycle of a machine state and enables the address to get latched into the on chip latch of peripherals. The falling edge of ALE is set to guarantee setup and hold times for the address information. ALE can also be used to strobe the status information. ALE is never 3stated.

### $S_0$, $S_1$ (Pin-29 and 33, Type-Output)

Data Bus Status of the bus cycle.

| $S_1$ | $S_0$ | Action |
|---|---|---|
| 0 | 0 | Halt |
| 0 | 1 | Write |
| 1 | 0 | Read |
| 1 | 1 | Fetch |

$S_1$ can be used as an advanced R/W status.

### $\overline{RD}$ (Pin 32, Type-Output, 3 State)

READ-indicates the selected memory or I/O device is to be read and that the Data Bus is available for the data transfer.

### $\overline{WR}$ (Pin 31, Type-Output, 3 State)

WRITE-indicates the data on the Data Bus is to be written into the selected memory or I/O location. Data is set up at the trailing edge of WR. Three stated during Hold and Halt modes.

### READY (Pin-35, Type-Input)

If Ready is high during a read or write cycle, it indicates that the memory or peripheral is ready to send or receive data. If Ready is low, the CPU will wait for Ready to go high before completing the read or write cycle.

### HOLD (Pin-39, Type-Input)

HOLD indicates that another Master is requesting the use of the Address and Data Buses. The CPU, upon receiving the Hold request. Will relinquish the use of buses as soon as the completion of the current machine cycle. Internal processing can continue. The processor can regain the buses only after the Hold is removed. When the Hold is acknowledged, the Address, Data, RD, WR, and IO/M lines are 3stated.

### HLDA (Pin-38, Type-Output)

HOLD ACKNOWLEDGE indicates that the CPU has received the Hold request and that it will relinquish the buses in the next clock cycle. HLDA goes low after the Hold request is removed. The CPU takes the buses one half clock cycle after HLDA goes low.

### INTR (Pin-10, Type-Input)

INTERRUPT REQUEST is used as a general purpose interrupt. It is sampled only during the next to the last clock cycle of the instruction. If it is active, the Program Counter (PC) will be inhibited from incrementing and an INTA will be issued. During this cycle a RESTART or CALL instruction can be inserted to jump to the interrupt service routine. The INTR is enabled and disabled by software. It is disabled by Reset and immediately after an interrupt is accepted.

### $\overline{INTA}$ (Pin-11, Type-Output)

INTERRUPT ACKNOWLEDGE is used instead of (and has the same timing as) RD during the Instruction cycle after an INTR is accepted. It can be used to activate the 8259 Interrupt chip or some other interrupt port.

*Restart Interrupts*

These three inputs have the same timing as I NTR except they cause an internal RESTART to be automatically inserted.

RST 7.5 (Pin-7, Type-Input) Possess Highest Priority

RST 6.5 (Pin-8, Type-Input)

RST 5.5 (Pin-9, Type-Input) Possess Lowest Priority

The priority of these interrupts is ordered as shown above. These interrupts have a higher priority than the INTR.

***TRAP (Pin-6, Type-Input)***

Trap interrupt is a non-maskable restart interrupt. It is recognized at the same time as INTR. It is unaffected by any mask or Interrupt Enable. It has the highest priority of any interrupt.

***$\overline{\text{RESET IN}}$ (Pin-36, Type-Input)***

Reset sets the Program Counter to zero and resets the Interrupt Enable and HLDA flip-flops. None of the other flags or registers (except the instruction register) are affected The CPU is held in the reset condition as long as Reset is applied.

***RESET OUT (Pin-3, Type-Output)***

This pin is used as a system RESET. The signal is synchronized to the processor clock.

***$X_1$, $X_2$ (Pin-1 and 2, Type-Input)***

Crystal or R/C network connections to set the internal clock generator X1 can also be an external clock input instead of a crystal. The input frequency is divided by 2 to give the internal operating frequency.

***CLK (Pin-37, Type-Output)***

Clock Output for use as a system clock when a crystal or R/ C network is used as an input to the CPU. The period of CLK is twice the X1, X2 input period.

***IO/$\overline{\text{M}}$ (Pin-34, Type-Output)***

IO/M indicates whether the Read/Write is to memory or I/O tristated during Hold and Halt modes.

***SID (Pin-5, Type-Input)***

Serial input data line, the data on this line is loaded into accumulator bit 7 whenever a RIM instruction is executed.

***SOD (Pin-4, Type-Output)***

Serial output data line. The output SOD is set or reset as specified by the SIM instruction.

***$V_{CC}$ (Pin-40)***

+5 volt supply.

***$V_{SS}$ (Pin-20)***

Ground Reference.

### 4.9.2. Signal Groups of Microprocessor 8085

Signals of 8085 are grouped in to six groups; these are shown in Fig. 4.6.

1. Power supply and frequency (+5V, GND, 3.2M Hz Crystal Oscillator.)
2. High order address buses ($A_{15}$–$A_8$).
3. Multiplexed Low order address bus and data buses. ($AD_7$–$AD_0$)
4. Control signals (ALE, S1, S0,IO/M,RD,WR, RESETOUT, CLK OUT)
5. Interrupts (TRAP, RST7.5, 6.5, 5.5, INTR).
6. Externally Initiating signals (Ready, HOLD, RESETIN) and Serial i/o signals.

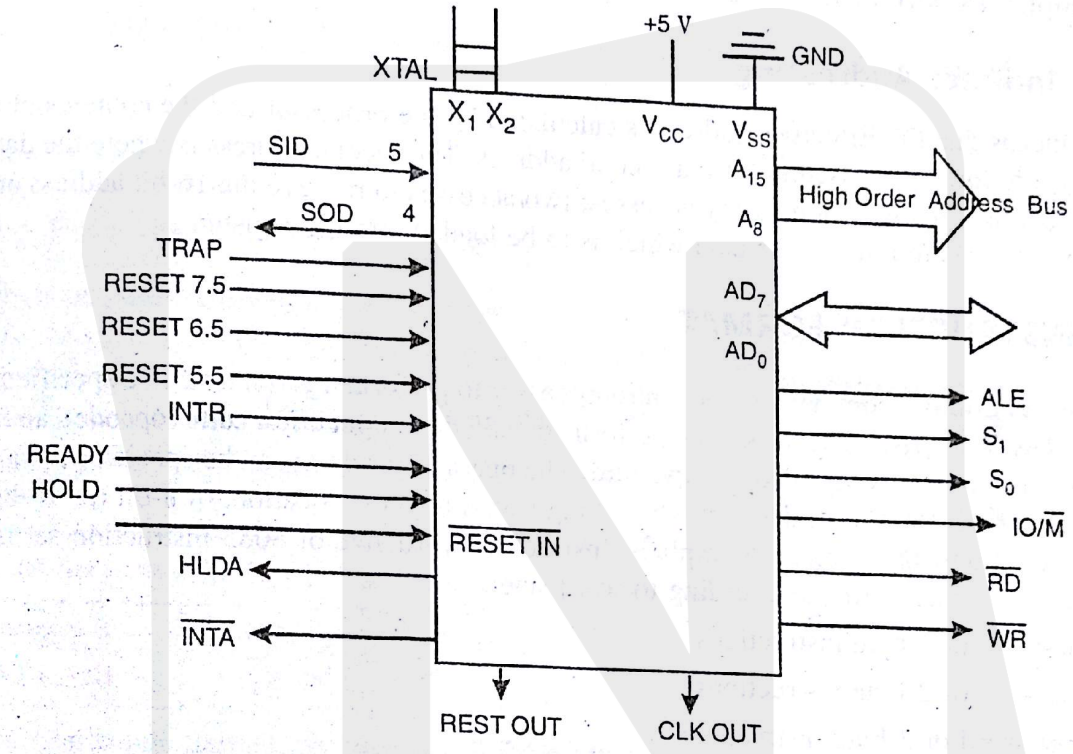



*Fig. 4.6. Different Signals of 8085*

## 4.10. ADDRESSING MODES OF 8085

The instructions MOV B, A or MVI A, 82H are to copy data from a source into a destination. In these instructions the source can be a register, an input port, or an 8-bit number (00H to FFH). Similarly, a destination can be a register or an output port. The sources and destination are operands. The various formats for specifying operands are called the addressing modes. For 8085, they are:

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

### 4.10.1. Immediate Addressing

Data is present in the instruction. Load the immediate data to the destination provided.

**Example:** MVI R, data

### 4.10.2. Register Addressing

Data is provided through the registers.

**Example:** MOV Rd, Rs

### 4.10.3. Direct Addressing

Used to accept data from outside devices to store in the accumulator or send the data stored in the accumulator to the outside device. Accept the data from the port 00H and store them into the accumulator or Send the data from the accumulator to the port 01H.

**Example:** IN 00H or OUT 01H

### 4.10.4. Indirect Addressing

This means that the Effective Address is calculated by the processor and the contents of the address (and the one following) is used to form a second address. The second address is where the data is stored. Note that this requires several memory accesses; two accesses to retrieve the 16-bit address and a further access (or accesses) to retrieve the data which is to be loaded into the register.

## 4.11. INSTRUCTION FORMAT

An **instruction** is a command to the microprocessor to perform a given task on a specified data. Each instruction has two parts: one is task to be performed, called the **operation code** (opcode), and the second is the data to be operated on, called the **operand.** The operand (or data) can be specified in various ways. It may include 8-bit (or 16-bit) data, an internal register, a memory location, or 8-bit (or 16-bit) address. In some instructions, the operand is implicit. Instruction word size of 8085 instruction set is classified into the following three groups according to word size:

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

In the 8085, "byte" and "word" are synonymous because it is an 8-bit microprocessor. However, instructions are commonly referred to in terms of bytes rather than words.

### 4.11.1. One-Byte Instructions

A 1-byte instruction includes the opcode and operand in the same byte. Operand(s) are internal register and are coded into the instruction. only 1 opcode

**For example :**

| Task | Opcode | Operand | Binary code | Hex code |
|---|---|---|---|---|
| Copy the contents of the Acc in the register C | MOV | C,A | 01001111 | 4F |
| Add the contents of register B to the contents of the Acc. | ADD | B | 10000000 | 80H |
| Invert each bit of the Acc. | CMA | | 00101111 | 2FH |

These instructions are 1-byte instructions performing three different tasks. In the first instruction, both operand registers are specified. In the second instruction, the operand B is specified and the accumulator is assumed. Similarly, in the third instruction, the accumulator is assumed to be the implicit operand. These instructions are stored in 8-bit binary format in memory; each requires one memory location.

**MOV rd, rs**

**rd <— rs copies contents of rs into rd.**

Coded as 01 ddd sss where ddd is a code for one of the 7 general registers which is the destination of the data, sss is the code of the source register.

**Example:** MOV A,B

Coded as 01111000 = 78H = 170 octal (octal was used extensively in instruction design of such processors).

**ADD r**

**A <— A + r**

### 4.11.2. Two-Byte Instructions

One op-code + 8-bit (address) data

In a two-byte instruction, the first byte specifies the operation code and the second byte specifies the operand. Source operand is a data byte immediately following the opcode. For example:

| Task | Opcode | Operand | Binary Code | Hex Code | |
|---|---|---|---|---|---|
| Load an 8-bit data byte in the accumulator. | MVI | A, data<br>Data code | 00111110<br>Hex code of data | 3E<br>2nd byte | 1st byte |

Assume that the data is 32 H.The assembly language instruction can be expressed as following.

| Mnemonics | Hex Code |
|---|---|
| MVI A,32H | 3E 32H |

The instruction would require two memory locations to store in memory.

**MVI r, data**

**r <— data**

**Example:** MVI A, 30H coded as 3EH 30H as two contiguous bytes.

### 4.11.3. Three-Byte Instructions

opcode + 16-bit data

In a three-byte instruction, the first byte specifies the opcode, and the following two bytes specify the 16-bit address. Note that the second byte is the low-order address and the third byte is the high-order address.

opcode + data byte + data byte

**For example:**

| Task | Opcode | Operand | Binary Code | Hex Code | |
|---|---|---|---|---|---|
| Transfer the program sequence to the memory location 2086H. | JMP | 2086H | 11000011<br>10000101<br>00100000 | C3<br>85<br>20 | 1st byte<br>2nd byte<br>3rd byte |

This instruction would require three memory locations to store in memory.

Three byte instructions – opcode + data byte + data byte

**LXI rp, data16**

rp is one of the pairs of registers BC, DE, HL used as 16-bit registers. The two data bytes are 16-bit data in L H order of significance.

**rp <— data16**

**Example:**

LXI H, 0520H coded as 21H 20H 50H in three bytes. This is also an immediate addressing.

**LDA addr**

**A <— (addr)** - Addr is a 16-bit address in L H order. Example: LDA 2134H coded as 3AH 34H 21H This is also an example of direct addressing.

## 4.12. INSTRUCTION SET OF 8085

Instruction set of 8085 can be divided into following categories

1. Data transfer instructions-MOV, LDA, MVI etc.
2. Arithmetic operation instructions-Add, subtract, increment and decrement.
3. Logical operation instructions-AND, OR, XOR and rotate.
4. Control transfer instructions-conditional, unconditional, call subroutine, return from subroutine and restarts.
5. Input/output operation instructions, Machine control instructions.
6. Other-setting/clearing flag bits, enabling/disabling interrupts, stack operations, etc.

### 4.12.1. Data Transfer Instructions

The data transfer instructions move data between registers or between memory and registers.

#### *MOV (Copy data from source to destination)*

**Syntax:** MOV Rd, Rs

This instruction copies the contents of the source M, Rs register into the destination register; the contents of Rd, M the source register are not altered. If one of the operands is a memory location, its location is specified by the contents of the HL registers.

**Example:** MOV B, C or MOV B, M

#### *MVI (Move Immediate 8-bit data)*

**Syntax:** MVI Rd, data

The 8-bit data is transferred to the destination register or M, data memory. If the operand is a memory location, its location is specified by the contents of the HL registers.

**Example:** MVI B, 57H or MVI M, 57H

#### *LDA (Load Accumulator Direct from Memory)*

**Syntax:** LDA 16-bit address

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

**Example:** LDA 2034H

### LHLD (Load H&L Registers Directly from Memory)

**Syntax:** LHLD 16-bit address

The instruction copies the contents of the memory location pointed out by the 16-bit address into register L and copies the contents of the next memory location into register H. The contents of source memory locations are not altered.

**Example:** LHLD 2040H

### LXI (Load Register Pair with Immediate data)

**Syntax:** LXI Reg. pair, 16-bit data

The instruction loads 16-bit data in the register pair designated in the operand.

**Example:** LXI H, 2034H or LXI H, XYZ

### LDAX (Load Accumulator from Address in Register Pair)

**Syntax:** LDAX B/D Reg. pair

The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory location into the accumulator. The contents of either the register pair or the memory location are not altered.

**Example:** LDAX B

### STA (Store Accumulator Direct in Memory)

**Syntax:** STA 16-bit address

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

**Example:** STA 4350H

### STAX (Store Accumulator in Address in Register Pair)

**Syntax:** STAX Reg. pair

The contents of the accumulator are copied into the memory location specified by the contents of the operand (register pair). The contents of the accumulator are not altered.

**Example:** STAX B

### SHLD (Store H and L Registers Directly in Memory)

**Syntax:** SHLD 16-bit address

The contents of register L are stored into the memory location specified by the 16-bit address in the operand and the contents of H register are stored into the next memory location by incrementing the operand. The contents of registers HL are not altered. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address.

**Example:** SHLD 2470H

### XCHG (Exchange H&L with D&E)

**Syntax:** XCHG none

The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E.

**Example:** XCHG

### XTHL (Exchange Top of Stack with H&L)

**Syntax:** XTHL none

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

**Example:** XTHL

### SPHL (Move content of H&L to Stack Pointer)

**Syntax:** SPHL none

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

**Example:** SPHL

## 4.12.2. Arithmetic Instructions

The arithmetic instructions are used to perform various arithmetic tasks such as addition, subtraction, division, multiplication, increment, or decrement etc.

### ADD (Add Register or Memory to Accumulator)

**Syntax:** ADD R

The contents of the operand (register or memory) are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition.

**Example:** ADD B or ADD M

### ADI (Add Immediate Data to Accumulator)

**Syntax:** ADI 8-bit data

The 8-bit data (operand) is added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition.

**Example:** ADI 45H

### ADC (Add Register to Accumulator with Carry)

**Syntax:** ADC R

The contents of the operand (register or memory) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition.

**Example:** ADC B or ADC M

### ACI (Add Immediate Data to Accumulator with Carry)

**Syntax:** ACI 8-bit data

The 8-bit data (operand) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition.

**Example:** ACI 45H

### DAD (Double Register Add; Add Content of Register Pair to H&L Register Pair)

**Syntax:** DAD Reg. pair

The 16-bit contents of the specified register pair are added to the contents of the HL register and the sum is stored in the HL register. The contents of the source register pair are not altered. If the result is larger than 16 bits, the CY flag is set. No other flags are affected.

**Example:** DAD H

### SUB (Subtract Register or Memory From Accumulator)

**Syntax:** SUB R

The contents of the operand (register or memory) are subtracted from the contents of the accumulator, and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction.

**Example:** SUB B or SUB M

### SUI (Subtract Immediate Data from Accumulator)

**Syntax:** SUI 8-bit data

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

**Example:** SUI 45H

### SBB (Subtract Source From Accumulator with Borrow)

**Syntax:** SBB R

The contents of the operand (register or memory) and the Borrow flag are subtracted from the contents of the accumulator and the result is placed in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction.

**Example:** SBB B or SBB M

### SBI (Subtract Immediate Data from Accumulator with Borrow)

**Syntax:** SBI 8-bit data

The 8-bit data (operand) and the Borrow flag are subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtraction.

**Example:** SBI 45H

### INR (Increment Register or Memory by One)

**Syntax:** INR R

The contents of the designated register or memory are incremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers.

**Example:** INR B or INR M

### DCR (Decrement Register or Memory by One)

**Syntax:** DCR R

The contents of the designated register or memory are M decremented by 1 and the result is stored i the same place. If the operand is a memory location, its location is specified by the contents of the HL register

**Example:** DCR B or DCR M

### INX (Increment Register Pair by One)

**Syntax:** INX R

The contents of the designated register pair are incremented by 1 and the result is stored in the same place

**Example:** INX H

### DCX (Decrement Register Pair by One)

**Syntax:** DCX R

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

**Example:** DCX H

### DAA (Decimal Adjust After Addition)

**Syntax:** DAA none

The contents of the accumulator are changed from a binary value to two 4-bit binary coded decimal (BCD) digits. This is the only instruction that uses the auxiliary flag to perform the binary to BCD conversion, and the conversion procedure is described below. S, Z, AC, P, CY flags are altered to reflect the results of the operation. If the value of the low-order 4-bits in the accumulator is greater than 9 or if AC flag is set, the instruction adds 6 to the low-order four bits. If the value of the high-order 4-bits in the accumulator is greater than 9 or if the Carry flag is set, the instruction adds 6 to the high-order four bits.

**Example:** DAA

## 4.12.3. Logical Instructions

This group of instructions performs logical (Boolean) operations on data in registers and memory and on condition flags. The logical operations are AND, OR and Exclusive OR etc.

### ANA (Logical AND of Register or Memory Data With Accumulator)

**Syntax:** ANA R

The contents of the accumulator are logically ANDed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is set.

**Example:** ANA B or ANA M

### ANI (Logical AND of Immediate Data With Accumulator)

**Syntax:** ANI 8-bit data

The contents of the accumulator are logically ANDed with the 8-bit data (operand) and the result is aced in the accumulator. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is set.

**Example:** ANI 86H

### ORA (Logical OR of Register or Memory Data with Accumulator)

**Syntax:** ORA R

The contents of the accumulator are logically ORed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset.

**Example:** ORA B or ORA M

### ORI (Logical OR of immediate data with Accumulator)

**Syntax:** ORI 8-bit data

The contents of the accumulator are logically ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset.

**Example:** ORI 86H

### XRA (Logical XOR of the register or memory data with Accumulator)

**Syntax:** XRA R

The contents of the accumulator are Exclusive ORed with M the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset.

**Example:** XRA B or XRA M

### XRI (Exclusive OR of Immediate Data with Accumulator)

**Syntax:** XRI 8-bit data

The contents of the accumulator are Exclusive ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset.

**Example:** XRI 86H

### CMP (Compare Register or Memory with Accumulator)

**Syntax:** CMP R

The contents of the operand (register or memory) are M compared with the contents of the accumulator. Both contents are preserved. The result of the comparison is shown by setting the flags of the PSW as follows:

**If (A) < (reg/mem) => carry flag is set**

**If (A) = (reg/mem) => zero flag is set**

**If (A) > (reg/mem) => carry and zero flags are reset**

**Example:** CMP B or CMP M

### CPI (Compare Immediate Data with accumulator)

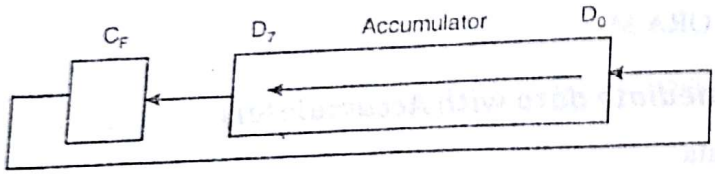
**Syntax:** CPI 8-bit data

The second byte (8-bit data) is compared with the contents of the accumulator. The values being compared remain unchanged. The result of the comparison is shown by setting the flags of the PSW as follows:

if (A) < data => carry flag is set
if (A) = data => zero flag is set
if (A) > data => carry and zero flags are reset

Example: CPI 89H

### RLC (Rotate Accumulator Left)

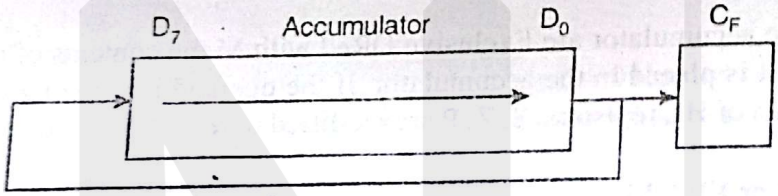



Syntax: RLC none

Each binary bit of the accumulator is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the Carry flag. CY is modified according to bit D7. S, Z, P, AC are not affected.

Example: RLC

### RRC (Rotate Accumulator Right)

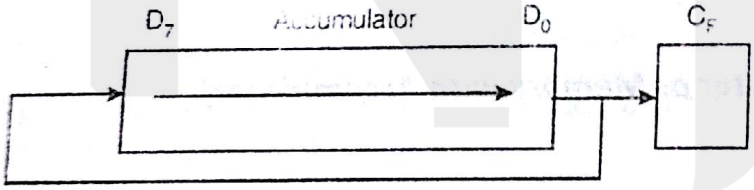



Syntax: RRC none

Each binary bit of the accumulator is rotated right by one position. Bit D0 is placed in the position of D7 as well as in the Carry flag. CY is modified according to bit D0. S, Z, P, AC are not affected.

Example: RRC

### RAL (Rotate Accumulator Left Through Carry)

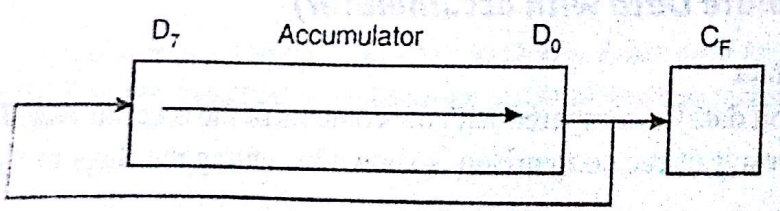



Syntax: RAL none

Each binary bit of the accumulator is rotated left by one position through the Carry flag. Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0. CY is modified according to bit D7. S, Z, P, AC are not affected.

Example: RAL

### RAR (Rotate Accumulator Right Through Carry)



**Syntax:** RAR none

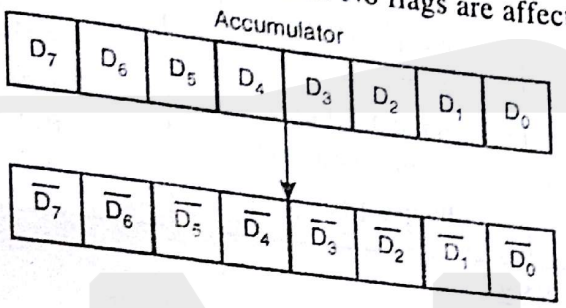
Each binary bit of the accumulator is rotated right by one position through the Carry flag. Bit D0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D7. CY is modified according to bit D0. S, Z, P, AC are not affected.

**Example:** RAR

### CMA (Complement Accumulator)

**Syntax:** CMA none

The contents of the accumulator are complemented. No flags are affected.




**Example:** CMA

### CMC (Complement Carry Flag)

**Syntax:** CMC none

The Carry flag is complemented. No other flags are affected.

**Example:** CMC

### STC (Set Carry Flag)

**Syntax:** STC none

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

**Example:** STC

## 4.12.4. Branch Instructions

The branching instructions alter normal sequential program flow, either unconditionally or conditionally.

### Unconditional Branching Instruction

**JMP (Jump unconditionally)**

**Syntax:** JMP 16-bit address

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand.

**Example:** JMP 2034H or JMP XYZ

### Conditional Branching Instructions

This kind of jump instructions are required where the status of the flags in the flag register need to meet a specific condition in the program. When the condition is met then the program sequence alters.

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW.

**Operand:** 16-bit address

Different types of conditional jump instructions are shown in the following Table.

| Opcode | Description | Flag status |
|---|---|---|
| JC | Jump if carry | CY=1 |
| JNC | Jump if not carry | CY=0 |
| JP | Jump if positive | S=0 |
| JM | Jump if minus | S=1 |
| JZ | Jump if zero | Z=1 |
| JNZ | Jump if not zero | Z=0 |
| JPE | Jump if parity even | P=1 |
| JPO | Jump if parity odd | P=0 |

**Example:** JZ 2034H or JZ XYZ

### CALL (Unconditional subroutine call)

**Syntax:** CALL 16-bit address

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

**Example:** CALL 2034H or CALL XYZ

### Conditional Subroutine CALL

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW as described below. Before the transfer, the address of the next instruction after the call (the contents of the program counter) is pushed onto the stack.

**Operand:** 16-bit address

Several types of conditional CALL instructions are shown in the following Table.

| Opcode | Description | Flag status |
|---|---|---|
| CC | Call if carry | CY=1 |
| CNC | Call if not carry | CY=0 |
| CP | Call if positive | S=0 |
| CM | Call if minus | S=1 |
| CZ | Call if zero | Z=1 |
| CNZ | Call if not zero | Z=0 |
| CPE | Call if parity even | P=1 |
| CPO | Call if parity odd | P=0 |

**Example:** CZ 2034H or CZ XYZ

### RET (Return from subroutine unconditionally)

**Syntax:** RET none

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

**Example:** RET

### Conditional Return from Subroutine

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

**Operand:** none

Various Conditional Return instructions are shown in the Table given below.

| Opcode | Description | Flag status |
|---|---|---|
| RC | Return if carry | CY=1 |
| RNC | Return if not carry | CY=0 |
| RP | Return if positive | S=0 |
| RM | Return if minus | S=1 |
| RZ | Return if zero | Z=1 |
| RNZ | Return if not zero | Z=0 |
| RPE | Return if parity even | P=1 |
| RPO | Return if parity odd | P=0 |

**Example:** RZ

### PCHL (Move the contents of HL to Program Counter)

**Syntax:** PCHL none

The contents of registers H and L are copied into the program counter. The contents of H are placed as the high-order byte and the contents of L as the low-order byte.

**Example:** PCHL

### RST (Special Restart Instruction Used with Interrupts)

**RST 0-7:** The RST instruction is equivalent to a 1-byte call instruction to one of eight memory locations depending upon the number. The instructions are generally used in conjunction with interrupts and inserted using external hardware. However these can be used as software instructions in a program to transfer program execution to one of the eight locations. The addresses are:

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

The 8085 has four additional interrupts and these interrupts generate RST instructions internally and thus do not require any external hardware. These instructions and their Restart addresses are:

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

### 4.12.5. Stack I/O Instructions

The following instructions affect the Stack and/or Stack Pointer.

#### *PUSH (Push Two bytes of Data onto the Stack)*

**Syntax:** PUSH Reg. pair

The contents of the register pair designated in the operand are copied onto the stack in the following sequence. The stack pointer register is decremented and the contents of the high order register (B, D, H, A) are copied into that location. The stack pointer register is decremented again and the contents of the low order register (C, E, L, flags) are copied to that location.

**Example:** PUSH B or PUSH A

#### *POP (Pop Two Bytes of Data from the Stack)*

**Syntax:** POP Reg. pair

The contents of the memory location pointed out by the stack pointer register are copied to the low order register (C, E, L, status flags) of the operand. The stack pointer is incremented by 1 and the contents of that memory location are copied to the high-order register (B, D, H, A) of the operand. The stack pointer register is again incremented by 1.

**Example:** POP H or POP A

### 4.12.6. I/O Instructions

#### *IN (Input Data to Accumulator from a Port with 8-bit Address)*

**Syntax:** IN 8-bit port address

The contents of the input port designated in the operand are read and loaded into the accumulator.

**Example:** IN 8CH

#### *OUT (Output Data from Accumulator to a Port with 8-bit Address)*

**Syntax:** OUT 8-bit port address

The contents of the accumulator are copied into the I/O port specified by the operand.

**Example:** OUT F8H

### 4.12.7. Machine Control Instructions

#### *EI (Enable Interrupt)*

*Syntax: EI none*

The interrupt enable flip-flop is set and all interrupts are enabled. No flags are affected. After a system reset or the acknowledgement of an interrupt, the interrupt enable flip-flop is reset, thus disabling the interrupts. This instruction is necessary to re-enable the interrupts (except TRAP).

*Example: EI*

### DI (Disable Interrupt)

**Syntax:** DI none

The interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled. No flags are affected.

**Example:** DI

### SIM (Set Interrupt Mask)

**Syntax:** SIM

This is a multipurpose instruction, used to implement the 8085 interrupts and serial data output. When microprocessor executes this instruction then 8-bit data of the accumulator is transferred to different fillip-flops. This instruction interprets the accumulator contents as follows.

| $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
|---|---|---|---|---|---|---|---|
| SOD | SOE | X | R7.5 | MSE | M7.5 | M6.5 | M5.5 |

**Description of symbols used :**

| Symbol | Description |
|---|---|
| SOD | Serial output data |
| SOE | Serial output enable |
| R7.5 | RST7.5 |
| M7.5, M6.5, M5.5 | Mask interrupts |

**Example:** SIM

### RIM (Read Interrupt Mask)

**Syntax:** RIM

This instruction is used to read the status of interrupt 7.5, 6.5, 5.5 and to read the serial data input bits on SID pin. The instruction loads 8-bits of the accumulator with following interpretation.

| $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
|---|---|---|---|---|---|---|---|
| SID | I7.5 | I6.5 | I5.5 | IE | M7.5 | M6.5 | M5.5 |

**Description of symbols used :**

| Symbol | Description |
|---|---|
| SID | Serial input data bit |
| I7.5, I6.5, I5.5 | Interrupt pending if Bit=1 |
| IE | Interrupt enable flip-flop is set if bit=1 |
| M7.5, M6.5, M5.5 | Interrupt masked if bit=1 |

**Example:** RIM

### HLT (Halt and Enter in Wait State)

**Syntax:** HLT none

The CPU finishes executing the current instruction and halts any further execution. An interrupt or reset is necessary to exit from the halt state.

**Example:** HLT

### NOP (No Operation)

**Syntax:** NOP none

No operation is performed. The instruction is fetched and decoded. However no operation is exec

**Example:** NOP

## SOLVED EXAMPLES

**Example 4.1.** *Program of adding two hexadecimal numbers 23H and 84H and saving the result i register.*

**Solution. Algorithm:**

1. Load the 1st number 23H in one register.
2. Load the 2nd number 84H in another register.
3. Add the contents of the two registers.
4. Save the result in any register.
5. End the program.

**Flow Chart**

The above steps can be represented in a pictorial format with the help of flowchart.

**Assembly Code**

```
MVI  A, 23H
MVI  B, 84H
ADD B
MOV C, A
HLT
```

**Example 4.2.** *Assembly language program which can perform addition, subtraction, NOT and OR function on two 8-bit data.*

**Solution. Program:**

```
MVI A, 20H
STA 2000H
MVI A, 10H
STA 2001H
LXI H, 2000H
MOV A, M
INX H
ADD M
STA 5000H
DCX M
MOV A, M
INX H
SUB M
STA 5001H
MOV B, M
```

```
DCX M
MOV A, M
ORA B
STA 5002H
MOV A, M
CMA
STA 5003H
DCX M
MOV A, M
CMA
STA 5004H
HLT
```

**Example 4.3.** *Assembly language program to add n- 8 bit numbers.*

**Solution. Program:**

```
MVI D, 00
MVI C, 05
LXI H, 8030
MOV A, M
DCR C
LOOP2 INX H
ADD M
JNC LOOP-1
INR D
LOOP-1 DCR C
JNZ LOOP2
STA 8056
MOV A, D
STA 8051
HLT
```

**Example 4.4.** *Assembly language program to subtract two 8 bit numbers.*

**Solution. Program:**

```
LXI H, 8501
MOV A, M
INX H
SUB M
JNC LOOP-1
INR C
LOOP-1 INX H
MOV M, A
MOV A, C
INX H
MOV M, C
RST5
```

**Example 4.5.** *Assembly language program to subtract two 16 bit numbers.*

**Solution.** **Program:**

```
LHLD 8100
XCHG
LHLD 8102
MOV A, E
SUB L
STA 8104
JNC LOOP
DCR D
LOOP MOV A, D
SUB H
STA 8105
RST5
```

**Example 4.6.** *Assembly language program to multiply two 8 bit numbers.*

**Solution.** **Program:**

```
LXI H, 8A00
MOV B, M
XRA A
MOV C, A
INX H
AHEAD ADD M
JNC LOOP-1
INR C
LOOP-1 DCR C
JNZ AHEAD
INX H
MOV M, A
INX H
MOV M, C
RST5
```