Microprocessor and its Applications

UNIT-2

- **2.1 8085 Instruction set**
- **2.2 Addressing Modes and**
- **2.3 Assembly Language**

Introduction

An instruction is a command given to the microprocessor to perform a specific operation on the given data.
Each microprocessor is designed to execute a specific group of instructions. This group of instructions/operations is called an instruction set. This instruction set defines what the microprocessor can and cannot do.
A microprocessor executes instructions given by the user
Instructions should be in a language known to the microprocessor
Microprocessor understands the language of 0's and 1's only
This language is called Machine Language

Sequence of instructions written for a processor to perform a particular task is called a program. Program & data are stored in the memory. The microprocessor fetches one instruction from the memory at a time & executes it. It executes all the instructions of the program one by one to produce the final result. The necessary steps that a microprocessor carries out to fetch an instruction & necessary data from the memory & to execute it constitute an instruction cycle.

In other words, an instruction cycle is defined as the time required completing the execution of an instruction.
An instruction cycle consists of a fetch cycle and an execute cycle. The time required to fetch an opcode (fetch cycle) is a fixed slot of time while the time

required to execute an instruction (execute cycle) is variable which depends on the type of instruction to be executed.

Instruction cycle(IC) = Fetch cycle(FC) + Execute cycle(EC)

This is shown diagrammatically in the **Fig.2.1**

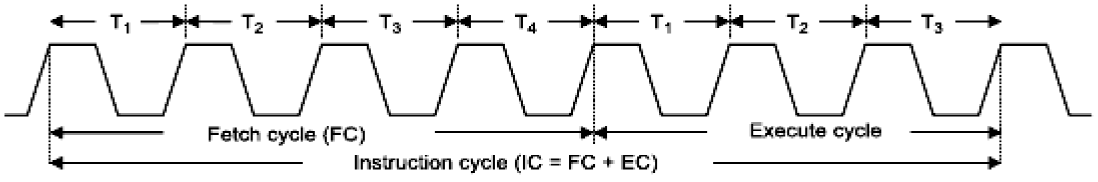


Figure 2.1 Instruction cycle

Machine cycle:

Machine cycle is defined as the time required for completing the operation of accessing either memory or I/O device. In the 8085, the machine cycle may consist of three to six T states. The T-state is defined as one sub-division of the operation performed in one clock period. These sub-divisions are internal states synchronized with the system clock. In every machine cycle the first operation is op-code fetch and the remaining will be read or write from memory or IO devices.

Fetch operation:

The first byte of an instruction is its op-code. 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 op-code is available, is sent to the memory. The memory places the op-code on the data bus so as to transfer it to the microprocessor. The entire operation of fetching an op-code takes three clock cycles.

Execute operation:

The op-code fetched from the memory goes to the instruction register (IR). From the instruction register it goes to the decoder circuitry which decodes the instruction. After the instruction is decoded, execution begins. If the operand is in general purpose registers execution is immediately performed.

The time taken for decoding and execution is one clock cycle. If an instruction contains data or operand and address which are still in the memory, the microprocessor 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 received from the memory are data or operand address instead of an op-code.** In some instructions write operation is performed. In write cycle data are sent from the microprocessor 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.

2.1 Instruction set of 8085

An Instruction is a command given to the microprocessor to perform a given task on specified data. Each instruction has two parts one is the task to be performed called the operation code (op-code) and the second is the data to be operated on, known as operand. The operand or data can be specified in various ways.

Instruction and data formats:

Since the format of a typical instruction is composed of two parts: an operation code or op-code and an operand. Every instruction needs an op-code to specify what the operation of the instruction is and then an operand that gives the appropriate data needed for that particular operation code.

According to the word or byte size the 8085 instructions are classified into three types. They are

(a) One byte (single) instructions.
(b)Two byte instructions.
(c) Three byte instructions.

One–byte instructions: An instruction with only opcode and do not require any data or address is called a one byte instruction.

Ex: 1. MOV C, A Hex code = 4FH (one byte)
2. ADD B Hex code = 80H (one byte)
3. CMA Hex code = 2FH (one byte)

Two–byte instructions: A two byte instruction is one which contains an 8-bit op-code and 8-bit operand (Data).

Ex: 1. MVI A, 09 Hex code = 3E, 09 (two bytes)
2. ADD B, 07 Hex code = 80, 07 (two bytes)
3. SUB A, 05 Hex code = 97, 05 (two bytes)

Three–byte instructions: A three byte instruction contains an opcode plus a 16 – bit address as operand.

Ex: 1.LXI H, 8509 Hex code = 21, 09, 85 (Three bytes)
2 .LDA 8509 Hex code = 3A, 09, 85 (Three bytes)
3. JMP 9567 Hex code = C3, 67, 95 (Three bytes)
4. STA 3525 Hex code = 32, 35, 25 (Three bytes)

DATA FORMATS: The 8085 is an 8-bit microprocessor which process only binary numbers. But it is very difficult to understand these binary numbers by a common user.
For e.g. 01001111

- Is a valid machine language instruction of 8085
- It copies the contents of one of the internal registers of 8085 to another

So, we have to code these binary numbers into different data formats. The commonly known data formats are ASCII, BCD, signed integers and unsigned integers. The ASCII code is a 7-bit alpha-numeric code that represents decimal numbers, English alphabets and certain special characters. The ASCII stands for ”American Standard code for Information Interchange”

The term BCD stands for binary coded decimal, used for decimal numbers from 0-9.An 8-bit register can store two BCD numbers. A signed integer is either a positive or a negative number. In 8085 microprocessor the most significant bit D_7 is used for the sign. Here 0 denotes positive sign and 1 denotes the negative sign. An integer without a sign can be represented by all the 8-bits in a microprocessor register. So, the largest number that can be processed at one time is FFH. The numbers larger than 8-bits like 16, 24, 32 bits can be processed by dividing them in groups of 8-bits.

CLASSIFICATION OF INSTRUCTIONS

An instruction is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions, called the instruction set, determines what functions the microprocessor can perform. The 8085 microprocessor instruction set has 74 operation codes that result in 246 instructions. This instruction set includes all the 8080A instructions plus two additional instructions namely SIM and RIM.

Example : a machine language to add two numbers

```
00111110            ;Copy value 2H in register A
00000010
00000110            ;Copy value 4H in register B
00000100
10000000            ;A = A + B
```

The instruction set of 8085 microprocessor is classified into five groups. They are:

1. Data transfer (copy) group
2. Arithmetic group
3. Logic group
4. Branch control group
5. Machine control and I/O group.

1- Data transfer (copy) instructions

The data transfer instructions are used to transfer data from one register to another register, from memory to register or register to memory but not from one memory location to another memory location. Actually this data transfer instruction copies the data from source to destination and the contents of the source are not altered. So, the data transfer instruction performs basically 'copy' operation.

Examples of data transfer instructions are MOV, MVI (Move Immediate), LXI (Load eXtended Immediate H-L Pair), LDA (Load Accumulator), STA (Store Accumulator), LHLD (Load H-L pair direct), SHLD (Store H-L pair direct), XCHG (Exchange the contents of H-L pair with D-E pair) etc…

Ex:
```
MVI A, 55H  ; Move the data 55H into Accumulator
MOV B, C    ; Copies the contents of 'C' register into 'B' register
IN 00H      ; Read the Input port(00H is the port address)
OUT 01H     ; write data to an output port(01H is the port address)
LXIH 8570H ; Load H-L pair by address 8570H.
```

In the 8085 microprocessor, data transfer instructions do not affect any flags.

2- Arithmetic Instructions

The arithmetic operations like addition, subtraction, increment and decrement are performed by the 8085 microprocessor using the following arithmetic instructions.
ADD, ADI (Add Immediate), SUB (Subtract), SUI (Subtract Immediate), INR (Increment), DCR (Decrement) etc…

The arithmetic operations Add and subtract are performed in relation to the contents of the accumulator. But, the increment or the decrement operations can be performed in any register.

Ex:	ADD B, C	; Add the contents of B register to the B register contents
	ADI 08	; Add the data 08 to the accumulator.
	SUB A, B	; Subtract the contents of B register from accumulator.
	SUI 05	;Subtract immediate the 8-bit data from accumulator
	INR B	; Increment the B register contents by one bit
	DCR C	; Decrement the C register contents by one bit.

Arithmetic instructions modify all the flags according to the data conditions of the result. The INR and DCR instructions affect all flags except the carry flag.

3- Logical Group of Instructions:

Since the microprocessor is a programmable logic chip, it can perform all the logic functions of the hard-wired logic through its instruction set. The 8085 processor can perform the logic instructions like, AND, OR, NOT (Complement) and X-OR (Exclusive OR) etc… The mnemonics of these instructions are given below.

ANA :	Logically AND the contents of a register
ANI :	Logically AND immediate the 8-bit data.
ORA :	Logically OR the contents of a register.
ORI :	Logically OR immediate the 8-bit data.
XRA :	Exclusive-OR the contents of a register.
XRI :	Immediate Exclusive-OR the 8-bit data
CMA :	Complement the accumulator

All the logic operations are performed in relation to the contents of the accumulator. The CMA instruction does not affect any flags. The executions of the logical instruction do not affect the contents of the operand register.

4- Branch Instructions

These instructions are very important because they allow the microprocessor to change the sequence of a program either conditionally or unconditionally. The conditional branch instructions transfer the program to the specified label when certain condition is satisfied. The unconditional branch instructions transfer the program to the specified location unconditionally.

We know that the microprocessor is a sequential machine. So, it executes machine codes from one memory location to the next. Branch instructions instruct the microprocessor to go to a different memory location and the processor continues executing machine codes from the new location. The address of the new locations either specified explicitly or provided by the microprocessor or some times by additional hardware. The Branch instructions are classified into three categories. They are

(a). Jump instructions
(b). Call and return instructions
(c). Restart instructions.

(a). Jump instructions

Jump instructions specify memory locations explicitly and they are 3-byte instructions. These Jump instructions are of two types. They are, Unconditional Jump and Conditional Jump.

Unconditional Jump:

When this instruction is executed the 16-bit address available immediately in the instruction is loaded into the program counter, so that the next sequence of instruction execution starts from this location. This Unconditional Jump instruction enables the programmer to create continuous loops.

JMP (16 bit address). So, this is a 3-byte instruction where the first byte is op-code and the second, third bytes specify memory address.

For example, the instruction JMP 8500H, instructs the microprocessor to go to the memory location8500H unconditionally. Sometimes, the jump location is specified using a label also.

Conditional Jump:

This instruction allows the microprocessor to make decision depending on certain conditions indicated by flags. The 8085 processor Jump instruction is associated with four flags. Namely Carry flag (CY), Zero flag (Z), Sign flag (S) and Parity flag (P). The following instructions shown in Table 2.1 transfer the program sequence to the memory location specified under the given conditions.

S. No	**Instruction**	**Description**
1	JC (16 bit Addr)	Jump on carry (if CY=1)
2	JNC (16 bit Addr)	Jump on no carry (if CY=0)
3	JZ (16 bit Addr)	Jump on Zero (if Z=1)
4	JNZ (16 bit Addr)	Jump on no Zero (if Z=0)
5	JP (16 bit Addr)	Jump on plus (if D_7=0)
6	JM (16 bit Addr)	Jump on minus (if D_7=1)
7	JPE (16 bit Addr)	Jump on Even Parity (if P=1)
8	JPO (16 bit Addr)	Jump on Odd Parity (if P=0)

Table 2.1 various conditional jump instructions

To understand the instructions, let us consider the instruction JC (16 bit address). The meaning of this instruction is, the microprocessor is instructed to jump to the specified 16 bit memory location if there exists a carry after the arithmetic operation else it will execute the next instruction in the sequence.

b) CALL and RETURN Instructions

The microprocessor uses the two instructions CALL and RETURN to implement subroutines. Here CALL instruction calls a subroutine program which is not a part of the main program and the RET instruction at the end of the subroutine program to return the control to the main program.

Ex: CALL (16 bit memory address)
RET

c) RESART (RST) Instruction

The 8085 processor provides eight RST instructions to transfer the program control to a specific location. These instructions are 1-byte instructions. The various RST instructions and their call locations are given in the following Table 2.2

S. No	Mnemonics	Hex code	Call location In Hex
1	RST 0	C7	0000
2	RST1	CF	0008
3	RST2	D7	0010
4	RST3	DF	0018
5	RST4	E7	0020
6	RST5	EF	0028
7	RST6	F7	0030
8	RST7	FF	0038

Table 4 Various RST instructions and their call location

5- Machine control and I/O Instructions

There are six basic machine control instructions. They are

1. EI (Enable Interrupt)
2. DI (Disable Interrupt)
3. NOP (No Operation)
4. SIM (Set Interrupt Mask)
5. RIM (Read Interrupt Mask)
6. HLT (Halt)

EI (Enable Interrupt): This is a one byte instruction used to enable the interrupt. This instruction is used to enable the interrupts when the microprocessor is reset or the interrupt enable flag is reset after interrupt acknowledge. This instruction takes one machine cycle with four states. The op-code is FBH.

DI (Disable Interrupt): This is a one byte instruction which resets the interrupt enable flag to disable all the interrupts except TRAP. It takes one machine cycle with four states. The op-code is F3H.

NOP (No Operation): when this instruction is executed, the microprocessor performs nothing. Microprocessor spends four states doing nothing. It is a one byte instruction whose op-code is 00H.This instruction is normally used to generate very small time delays of the order of few micro seconds. This NOP instruction is also very useful when we are required to insert a few instructions in the main program additionally.

SIM (Set Interrupt Mask): This instruction masks the interrupt as desired. This is a dual purpose instruction. The first purpose is to set or reset the mask of the maskable interrupt. The second purpose is to send the data out through the SOD pin at pin number 4 of the microprocessor.

RIM (Read Interrupt Mask): This instruction copies the status of the interrupts into the accumulator. It is also used to read the serial data through the SID pin

HLT (Halt): After execution of this instruction the microprocessor goes into the halt state. The processor can be restarted by a valid interrupt or by applying a RESET signal. The microprocessor takes 5T states to implement the halt instruction.

I/O instructions:
There are two important instructions to input the data into the microprocessor's accumulator through the input port and output the data from the accumulator to the output port. They are:

- IN (port address)
- OUT (port address)

This port address is an 8-bit address. In both these instructions the default register is Accumulator.

Ex: (i) IN 01H. This instruction will copy the contents into the Accumulator through the port whose address is 01H. It takes three machine cycles and takes 10 states. The op-code is DBH.

(ii)OUT 02H. This instruction sends the contents of Accumulator to the out-port whose address is 02H. It is a two byte instruction which requires 10 states. The op-code for this instruction is D3H.

DETAILED INSTRUCTION SET

DATA TRANSFER INSTRUCTIONS

Opcode	Operand	Description
Move from source to destination		
MOV	Rd, Rs M, Rs Rd, M	This instruction copies the contents of the source register into the destination register; the contents of 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
Move immediate 8-bit		
MVI	Rd, data M, data	The 8-bit data is stored in the destination register or 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
Load accumulator		
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
Load accumulator indirect		
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
Load register pair immediate		
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
Load H and L registers direct		
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

Store accumulator direct

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

Store accumulator indirect

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

Store H and L registers direct

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

Exchange H and L with D and E

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

Copy H and L registers to the stack pointer

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

Exchange H and L with top of stack

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

Push register pair onto stack

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 PSW

Pop off stack to register pair

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 PSW

Output data from accumulator to a port with 8-bit address

OUT 8-bit port address

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

Example: OUT F8H

Input data to accumulator from a port with 8-bit address

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

ARITHMATIC INSTRUCTIONS

Opcode	Operand	Description
Add register or memory to accumulator		
ADD	R M	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
Add register to accumulator with carry		
ADC	R M	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
Subtract register or memory from accumulator		
SUB	R M	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
Subtract source and borrow from accumulator		
SBB	R M	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
Subtract immediate from accumulator		
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

Subtract immediate from accumulator with borrow

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

Increment register or memory by 1

INR	R M	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

Increment register pair by 1

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

Decrement register or memory by 1

DCR	R M	The contents of the designated register or memory are decremented 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: DCR B or DCR M

Decrement register pair by 1

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

Decimal adjust accumulator

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

BRANCHING INSTRUCTIONS

Opcode	Operand	Description
Jump unconditionally		
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

Jump conditionally

Operand: 16-bit address

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.

Example: JZ 2034H or JZ XYZ

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

Unconditional subroutine call

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

Call conditionally

Operand: 16-bit address

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.

Example: CZ 2034H or CZ XYZ

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

Return from subroutine unconditionally

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

Return from subroutine conditionally

Operand: none

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.

Example: RZ

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

Load program counter with HL contents

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

Restart

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

LOGICAL INSTRUCTIONS

Opcode	Operand	Description
Compare register or memory with accumulator		
CMP	R M	The contents of the operand (register or memory) are 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
Compare immediate with accumulator		
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

Logical AND register or memory with accumulator

ANA	R M	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

Logical AND immediate with accumulator

ANI	8-bit data	The contents of the accumulator are logically ANDed 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 is reset. AC is set. Example: ANI 86H

Exclusive OR register or memory with accumulator

XRA	R M	The contents of the accumulator are Exclusive 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: XRA B or XRA M

Exclusive OR immediate with accumulator

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

Logical OR register or memory with accumulator

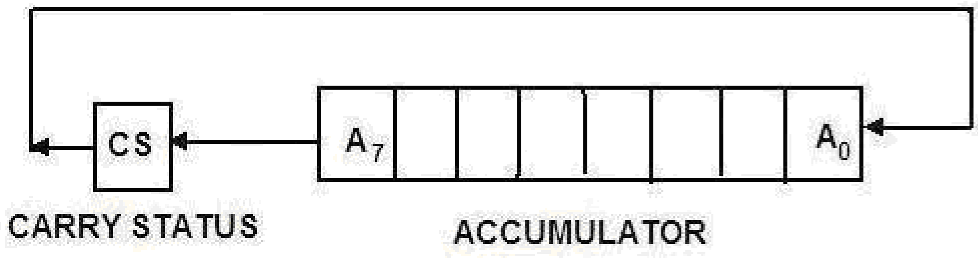
ORA	R M	The contents of the accumulator are logically ORed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the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

Logical OR immediate with accumulator

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

Rotate accumulator left

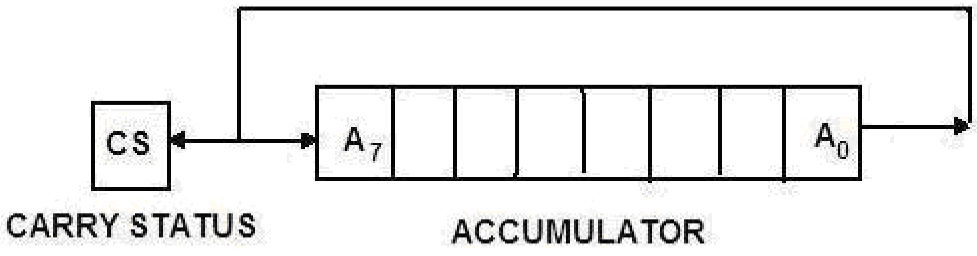
RLC	none	Each binary bit of the accumulator is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the Carry flag. CY is modified according to bit D7.S, Z, P, AC are not affected. Example: RLC



Schematic Diagram for RLC

Rotate accumulator right

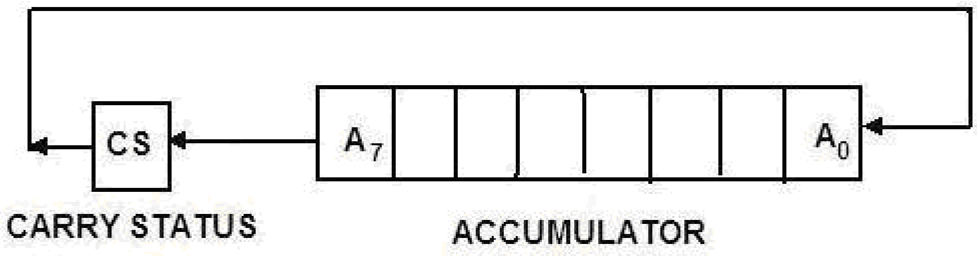
RRC	none	Each binary bit of the accumulator is rotated right by one position. Bit D0 is placed in the position of D7 as well as in the Carry flag. CY is modified according to bit D0.S, Z, P, AC are not affected. Example: RRC



Schematic Diagram for RRC

Rotate accumulator left through carry

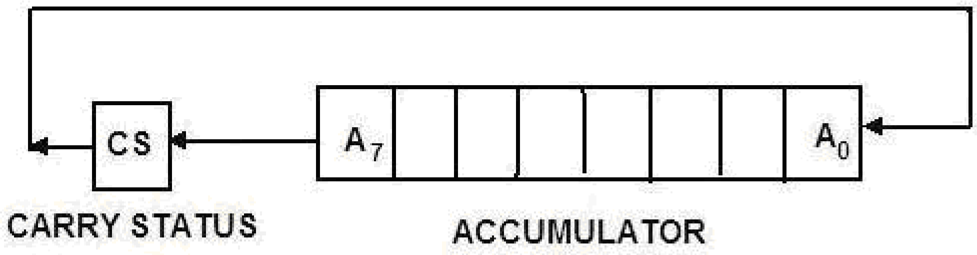
RAL	none	Each binary bit of the accumulator is rotated left by one position through the Carry flag. Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0.CY is modified according to bit D7. S, Z, P, AC are not affected. Example: RAL



Schematic Diagram for RAL

Rotate accumulator left through carry

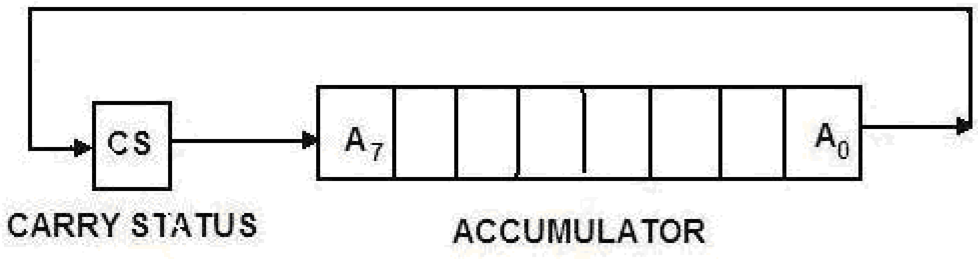
RAL	none	Each binary bit of the accumulator is rotated left by one position through the Carry flag. Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0.CY is modified according to bit D7. S, Z, P, AC are not affected. Example: RAL



Schematic Diagram for RAL

Rotate accumulator right through carry

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



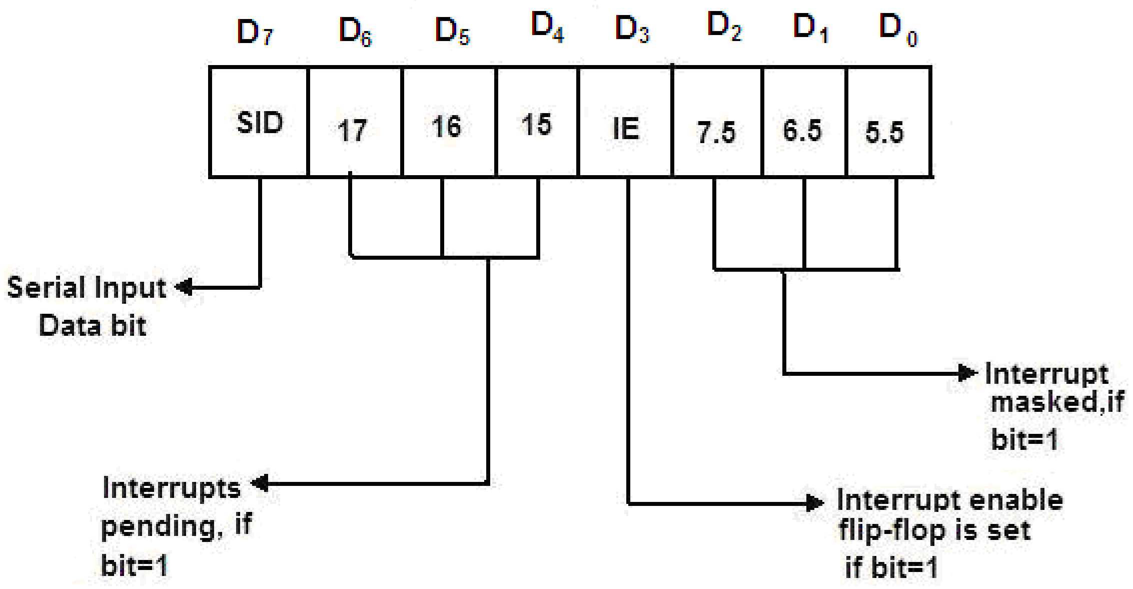
Schematic Diagram for RAR

CONTROL INSTRUCTIONS

Opcode	Operand	Description
No operation		
NOP	none	No operation is performed. The instruction is fetched and decoded. However no operation is executed. Example: NOP
Halt and enter wait state		
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
Disable interrupts		
DI	none	The interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled. No flags are affected. Example: DI
Enable interrupts		
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 reenable the interrupts. (except TRAP). Example: EI

Read interrupt mask

RIM none	This is a multipurpose instruction used to read the status of interrupts 7.5, 6.5, 5.5 and read serial data input bit. The instruction loads eight bits in the accumulator with the following interpretations.
	Example: RIM


D7
D6
D5
D4
D3
D2
D1
D0
SID
17
16
15
IE
7.5
6.5
5.5
Serial Input Data bit
Interrupts pending, if bit=1
Interrupt enable flip-flop is set if bit=1
Interrupt masked,if bit=1

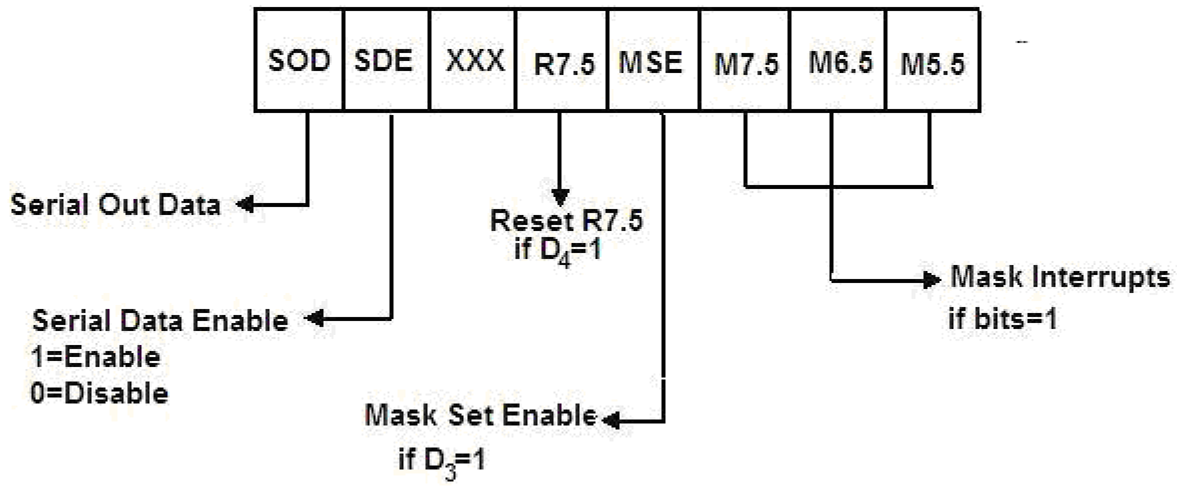
Read Interrupt Mask Register

Set interrupt mask

SIM none

This is a multipurpose instruction and used to implement the 8085 interrupts 7.5, 6.5, 5.5, and serial data output. The instruction interprets the accumulator contents as follows.

Example: SIM



Set Interrupt Mask Register

2.2 Addressing Modes of 8085

The microprocessor has different ways of specifying the data for the instruction. These are called “addressing modes”.

1. **Immediate addressing:**
2. **Register addressing:**
3. **Memory addressing**
    - **3.1. Direct addressing:**
    - **3.2. Indirect addressing:**
4. **Implied addressing:**

1. Immediate addressing:

Immediate data is transferred to address or register. All but two of the Immediate instructions use the accumulator as an Implied operand, these are: MVI and LXI
-The MVI (move Immediate) Instruction can move its immediate data to any of the working registers, including the accumulator, or to memory. Thus, the Instruction MVI D,OFFH moves the hexadecimal value FF to the D register.
-The LXI Instruction (load register pair immediate) is even more unusual in that its immediate data is a 16-bit value. This instruction is commonly used to load addresses into a register pair. Since it is necessary that a program must initialize the stack pointer; LXI is the instruction most commonly used for this purpose.

Example:
The instruction LXI SP,30FFH loads the stack pointer with the hexadecimal value 30FF

MVI A,20H. Transfer immediate data 20H to accumulator.
Number of bytes:
Either 2 or 3 bytes long.

1st byte is opcode.
2nd byte 8 bit data .

3rd byte higher byte data of 16 bytes.

2. Register addressing:

Quite a large set of instructions call for register addressing. With these instructions, you must specify one of the registers A through E, H or L as well as the operation code. With these instructions, the accumulator is implied as a second operand. For example, the instruction CMP E may be Interpreted as 'compare the contents of the E register with the contents of the accumulator.'
Most of the Instructions that use register addressing deal with 8-bit values.

Example MOV A,C :Transfer data from C register to accumulator.
Number of bytes: Only 1 byte long.
However, a few of these Instructions deal with 16-bit register pairs. For example, the PCHL Instruction exchanges the contents of the program counter with the contents of the Hand L registers.

3. Memory addressing:

3.1 Direct addressing:

For these data is transferred from direct address to other register or vice-versa.
Jump Instructions include a l6-bit address as part of the instruction.
Instructions that include a direct address require three bytes of storage:
1st byte is opcode.
2nd byte lower address.
3rd byte higher address.

Example,
-JMP 1000H causes a jump to the hexadecimal address 1000 by replacing the current contents of the program counter with the new value 1000
-LDA C200H .Transfer contents from C200H to Acc.

3.2 Indirect addressing:

A memory pointer register is used to store the address of the memory location (reference memory via a register pair). Thus, the Instruction MOV M ,C moves the

contents of the C register into the memory address stored in the Hand L register pair or vice-versa.
The instruction LDAX B loads the accumulator with the byte of data specified by the address in the B and C register pair.

5. Implied addressing:
These doesn't require any operand. The data is specified in Opcode itself.
Example: RAL: Rotate left with carry.
CMA : Complement Accumulator
STC (set carry flag)
These are single byte instruction or Opcode only.

2.3 Assembly Language

It is important to remember that a machine language and its associated assembly language are completely machine dependent.
In other words, they are not transferable from one microprocessor to a different one.
• For example, Motorola has an 8- bit microprocessor called the 6800.
– The 8085 machine language is very different from that of the 6800. So is the assembly language.
A program written for the 8085 cannot be executed on the 6800 and vice versa.

How does assembly language get translated into machine language?

There are two ways

1- There is "**hand assembly**".
The programmer translates each assembly language instruction into its equivalent hexadecimal code (machine language). Then the hexadecimal code is entered into memory.
2 – The other possibility is a program called an "**assembler**", which does the translation automatically.

This program enables the PC to receive commands in the form of abbreviations and convert them precisely into so called "executable file". The moment of compiling a program into machine language is crucial as this file, called HEX file, represents a series of binary numbers understandable to microprocessor/microcontrollers only. The program written in assembly language cannot be executed practically unless this file is loaded into the microcomputer/microcontroller memory. This is the moment when the last link in the chain (the programmer) appears on the scene. Programmer is a small device connected to a PC via some of the ports and has a socket for placing chip in.

Syntax of Assembly language

When writing a program in assembly language it is necessary to observe specific rules in order to enable the process of compiling into executable "HEX-code" to run without errors. These compulsory rules are called syntax and there are only some of them:

- Every program line may consist of a maximum of 255 characters;
- Every program line to be compiled, must start with a symbol, label, mnemonics or directive;
- Text following the mark ";" in a program line represents a comment ignored (not compiled) by the assembler; and
- All the elements of one program line (labels, instructions etc.) must be separated by at least one space character. For the sake of better clearness, a push button TAB on a keyboard is commonly used instead of it, so that it is easy to delimit columns with labels, directives etc. in a program.

Format of a typical Assembly language instruction is given below-

```
[Label:] Mnemonic [Operands] [;comments]
         HLT
         MVI A, 20H
         MOV M, A   ;Copy A to memory location whose
                     address is stored in register pair HL
LOAD:  LDA 2050H ;Load A with contents of memory
                     location with address 2050H
READ:  IN 07H      ;Read data from Input port with
                     address 07H
```

Steps to write a program

1-Analyze the problem

2-Develop program Logic

3-Write an Algorithm

4-Make a Flowchart

5-Write program Instructions using Assembly language of 8085

Program 8085 in Assembly language to add two 8-bit numbers and store 8-bit result in register C.

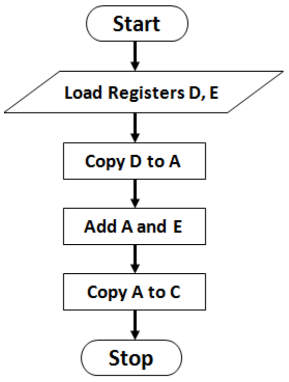
1. **Analyze the problem**
   - Addition of two 8-bit numbers to be done
2. **Program Logic**
   - Add two numbers
   - Store result in register C
   - Example

```
 10011001  (99H) A
+00111001  (39H) D
 11010010  (D2H) C
```

3. Write an Algorithm

	Translation to 8085 operations
1. Get two numbers	• Load 1st no. in register D • Load 2nd no. in register E
2. Add them	• Copy register D to A • Add register E to A
3. Store result	• Copy A to register C
4. Stop	• Stop processing

4. Make a Flowchart



- Load 1st no. in register D
- Load 2nd no. in register E

- Copy register D to A
- Add register E to A

- Copy A to register C

- Stop processing

Example :Program 8085*u*P in Assembly language to add two 8-bit numbers. Result can be more than 8-bits.

1. Analyze the problem
   - Result of addition of two 8-bit numbers can be 9-bit
   - Example

```
 10011001  (99H) A
+10011001  (99H) B
100110010 (132H)
```

   - The 9th bit in the result is called CARRY bit.

5. Write Assembly Language Program using instructions

1. Get two numbers	
a) Load 1st no. in register D b) Load 2nd no. in register E	MVI D, 2H MVI E, 3H
2. Add them	
a) Copy register D to A b) Add register E to A	MOV A, D ADD E
3. Store result	
a) Copy A to register C	MOV C, A
4. Stop	
a) Stop processing	HLT

Example :Program 8085*u*P in Assembly language to add two 8-bit numbers. Result can be more than 8-bits.

1. Analyze the problem
   - Result of addition of two 8-bit numbers can be 9-bit
   - Example

```
 10011001   (99H) A
+10011001   (99H) B
100110010 (132H)
```

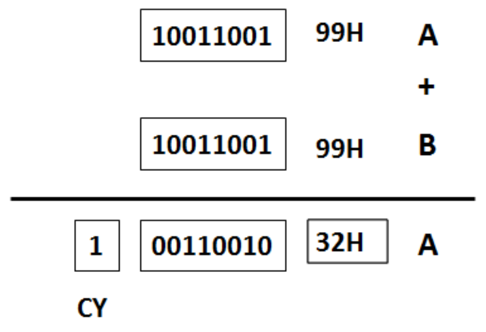
   - The 9th bit in the result is called CARRY bit.

How 8085 does it?

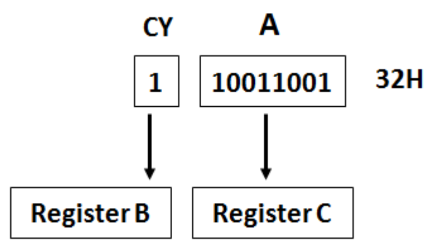
– Adds register A and B

– Stores 8-bit result in A

– SETS carry flag (CY) to indicate carry bit



Storing result in Register memory



Step-1 Copy A to C

Step-2

a) Clear register B

b) Increment B by 1

2. Program Logic

1. Add two numbers
2. Copy 8-bit result in A to C
3. If CARRY is generated
   – Handle it
4. Result is in register pair BC

3. Algorithm

1. Load two numbers in registers D, E
2. Add them
3. Store 8 bit result in C
4. Check CARRY flag
5. If CARRY flag is SET
   - Store CARRY in register B
6. Stop

Translation to 8085 operations

- Load registers D, E

- Copy register D to A
- Add register E to A

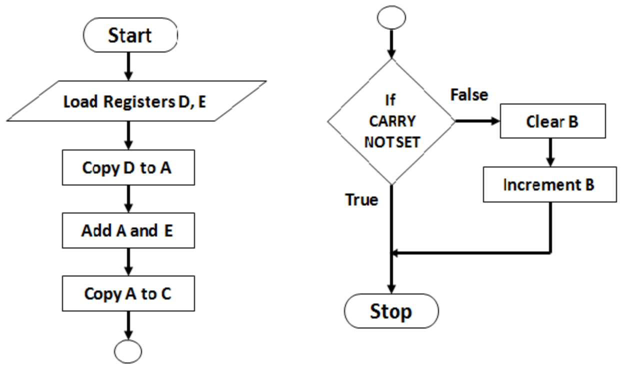
- Copy A to register C

- Use Conditional Jump instructions

- Clear register B
- Increment B

- Stop processing

4. Make a Flowchart



5. Assembly Language Program

• Load registers D, E		MVI D, 2H MVI E, 3H
• Copy register D to A • Add register E to A		MOV A, D ADD E
• Copy A to register C		MOV C, A
• Use Conditional Jump instructions		JNC END
• Clear register B • Increment B		MVI B, 0H INR B
• Stop processing	END:	HLT