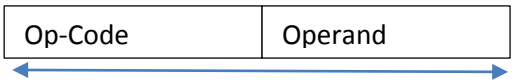
Chapter 5. Assembly Language Programming

5.1. Instruction format

An instruction is a command 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, 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 an 8-bit (or 16 bit) address. In some instructions, the operand is implicit.



Instruction format

5.5.1. Instruction Word Size

The 8085 instruction set is classified into the following three groups according to word size or byte size.

1. one-byte instructions

2. two-byte instructions

3. three-byte instructions

i) One Byte instruction

The one byte instruction includes the opcode and the operand in the same byte.

Eg.	MOV A, B	RET
	ADD B	SUB C
	RAR	ANA B
	CMA	HLT

ii) Two Byte instruction

A two-byte instruction has the first byte specifying the operation code and the second byte as the operand.

Eg. MVI A, 32H

ANI 04H

IN 01H

OUT 03H etc.

iii) Three Byte instruction

In a 3-byte instruction, the first byte specifies the opcode, and the following two byte specify the 16-bit address. The second byte is the low order address and the third byte is the higher order address.

Example:

LDA 2050H JZ 2025H

STA 2060H JNZ 2040H

JMP 2030H CALL 6000H etc.

5.2. 8085 Instruction set

An instruction is a binary bit pattern which performs a specific function in a system. The entire group of instructions of a system is called the *instruction set*. Instruction set determines what functions the microprocessor can perform with single instruction. The instruction in microprocessor 8085 can be classified into five functional categories

i. Data transfer (copy) operations
ii. Arithmetic operations
iii. Logical operations
iv. Branching operations
v. Machine-control operations

4.5.1. Data Transfer Instruction

This group of instructions copy data from a location called a source to another location called a destination, without modifying the contents of the source. Following are the notation used

1. MOV Instruction (1 byte instruction): copy from source to destination

Syntax: MOV Rd, Rs; MOV Rd, M, MOV M, Rs:

This instruction copies the contents of the source register or the memory into the destination register or in the memory location. The contents of source will not alter. If one of the operands is a memory location, its location is specified by the contents of the HL pair.

Example:

MOV B, A ➔ this instruction will copy the contents of register A into B register (B←A)

MOV M, B ➔This instruction will copy the contents of B register into memory location whose address is specified by HL registers.

2. MVI Instruction (2 byte instruction): Move immediate 8-bit data

Syntax: MVI Rd, data (8); MVI M, data(8)

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 HL registers.

Example; MVI Rd, data (8)➔ This instruction directly loads a specified register with an 8-bit data given within instructions.

MVI M, data (8)➔ This instruction directly loads an 8-bit data given within the instruction into a memory location. The memory location is specified by the contents of HL register pair.

3. LXI (3 byte instruction): Load register pair immediate

Syntax: LXI Rp, data (16)

The instruction loads 16-bit data specified within the instruction into register pair or stack pointer. The Rp is 16-bit register pair such as BC, DE, HL or 16-bit stack pointer.

Example:

LXI B, 1020H➔ This instruction will load 10H into B register and 20H into C register.

4. LDA (3 byte instruction): Load accumulator direct.

Syntax: LDA 16-bit address

This instruction copies the contents of the memory location whose address is given within the instruction into the accumulator.

Example:

LDA 2000H➔This instruction will copy the contents of memory location 2000H into the A register. i.e. A←[2000].

5. LDAX (1 Byte instruction): Load accumulator indirect

Syntax: LDAX Rp

This instruction copies the contents of memory location whose address is specified by the register into the accumulator. The Rp is BC or DE register pair. The register is used as a memory pointer.

Example:

LDAX D➔ this instruction copies the contents of memory location specified by DE register pair.

Note: LDAX H is invalid instruction. We rather use MOV A, M

6. STA instruction (3 byte instruction): Store accumulator direct

Syntax: STA 16-bit address

This instruction stores the contents of A register into the memory location whose address is directly specified within the instruction. The contents of A register remain unchanged.

Example:

STA 2000H➔ This instruction will store the contents of A register to memory location 2000H. ([2000H]←A)

7. STAX instruction (1 Byte instruction)

Syntax: STAX Rp

This instruction copies the contents of accumulator into the memory location whose address is specified by the specified register pair. The Rp is BC or DE register pair. This register pair is used as a memory pointer.

Example:

STAX B➔ This instruction will copy the contents of A register to the memory location specified by BC register pair.

Note: STAX H is invalid instruction. To store the accumulator content to the address pointed by HL pointer, we use the instruction MOV M, A

8. LHLD (3 byte instruction): Load H and L register direct

Syntax: LHLD 16-bit address

This instruction copies the contents of the memory location given within the instruction into the L register and the contents of next memory location into the H register.

Example:

Suppose [2500] =30H, [2501] = 60H

LHLD 2500H➔ this instruction will copy the contents of memory location 2500H. ie. Data 30H into L register and the contents of memory location 2501H i.e. data 60H into H register.

9. SHLD (3 byte instruction): Store H and L register direct.

Syntax: SHLD 16-bit address

This instruction stores the contents of L register in the memory location given within the instruction and contents of H register at address next to it.

Example:

Suppose: H = 30H, L =60H

SHLD 2500 ➔ This instruction will copy the contents of L register at address 2500H and the contents of H register at address 2501H

10. XCHG (1 byte instruction): Exchange H and L with D and E

Syntax: XCHG

This instruction exchanges the contents of the register H with that of D and of L with that of E.

Example:

Suppose DE= 2040H, HL = 7080H

XCHG➔ this instruction will load the data into registers as follows: H= 20H, L = 40H, D = 70H and E =80H

11. PUSH (1 byte instruction): Push register pair onto stack

Syntax: PUSH Rp, PUSH PSW

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 higher order register (B, H, 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:

Suppose: SP = 2000H, DE= 1050H

PUSH D ➔ This instruction will decrement the stack pointer (2000H) by one (SP = 1FFFH) and copies the contents of D register (10H) into the memory location 1FFFH. It then decrements the stack pointer again by one (SP = 1FFEH) and copies the contents of E register (50H) into the memory location 1FFEH.

PUSH PSW➔ This instruction decrements stack pointer by one and copies the accumulator contents into the memory location pointed by stack pointer. It then decrements the stack pointer again by one and copies the flag register into the memory pointed by the stack pointer.

12. POP (1 byte instruction): Pop off stack to register pair.

Syntax: POP Rp, POP PSW

The contents of the memory location pointed out by the stack pointer register are copied to the low-order register (C, E, L, status flag) 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:

Suppose: SP= 2000H, [2000H] =30H, [2001H] =50H

POP B ➔ This instruction will copy the contents of memory location pointed by stack pointer, 2000H (i.e. data 30H) into C register. It will then increment the stack pointer by one, 2001H and will copy the contents of memory location pointed by stack pointer, 2100H (i.e. data 50H) into B register, ad increment the stack pointer again by one.

13. OUT (2 byte instruction): Output data from accumulator to a port with 8-bit address.

Syntax: Out 8-bit port address

This instruction sends the contents of accumulator to the output port whose address is specified within the instruction.

Example:

Suppose: A = 40H

OUT 50H➔ This instruction will send the contents of accumulator (40H) to the output port whose address is 50H

14. IN (2 byte instruction): Input data to accumulator from a port with 8-bit address

Syntax: IN 8-bit port address

This instruction copies the data at a port whose address is specified in the instruction into the accumulator.

Example:

Suppose: Port address =80H, data stored at port address 80H , [80H]= 10H.

IN 80H ➔ this instruction will copy the data stored at address 80H, i.e. data 10H in the accumulator.

4.5.2. Arithmetic Instruction

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

1. ADD (1 byte instruction): Add register or memory to accumulator

Syntax: ADD R; ADD 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.

Example:

1. Suppose A = 20H, C =30H

ADD C ➔A ←A + C hence, A = 50H

2. Suppose: A= 20H, HL =2050H, [2050] = 10H (i.e. content of add. 2050 is 10H)

ADD M ➔ This instruction will add the contents of memory location pointed by HL register pair, 2050H, i.e. data 10H with the content of accumulator i.e. data 20H and it will store the result, 30H in the accumulator.

2. ADC (1 byte instruction): Add register to accumulator with carry

Syntax: ADC R; ADC M

The content 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.

Example:

Suppose: Carry flag = 1, A =50H, C = 20H

ADD C ➔ This instruction will add the contents of C (20H) register to the contents of accumulator (50H) with carry (1) and it will store result, 71H (50H + 20H + 1= 71H) in the accumulator.

3. ADI (2 byte instruction): Add immediate 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.

Example: ADI 30H ➔ A = A+ 30H

4. ACI (2 byte instruction): Add immediate to accumulator with carry

Syntax: ACI 8-bit data

This instruction adds 8-bit data given within the instruction to the contents of accumulator with carry and store the result in the accumulator

Example: ACI 30H➔ A = A+30H+CY

5. DAD (1 byte instruction): Add register pair to H and L registers.

Syntax: DAD register pair/SP

This instruction adds the contents of the specified register pair to the contents of the HL register pair and store the result in the HL register pair. The register pair is 16 bit register pair such as BC, DE, HL or stack pointer. Only higher order register is to be specified for register pair within the instruction

Example:

Suppose: DE = 1020H, HL = 2050H

DAD D ➔This instruction will add the contents of DE register pair, 1020H to the contents of HL register pair, 2050H. It will store result, 3070H in the HL register pair.

6. DAA (1 byte instruction): Decimal adjust accumulator

Syntax: DAA

The contents of the accumulator are changed from a binary value to two 4-bit binary coded decimal (BCD) digits.

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:

MVI A, 12H ADI 39H DAA	Initially 12+39=4B	After execution 12+39=51 in BCD form.

7. SUB (1 byte instruction): Subtract Register or memory from accumulator

Syntax: SUB R; SUB M

The content of the operand (register or memory) are subtracted from the content 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.

Example:

1. SUB B	2. SUB M
Suppose the Data at B register is 10H and A is 20H. Initially After execution B= 10H, A=20H A=10H, B=10H.	Suppose HL = 1020H, A = 50H, [1020H]=10H This instruction will subtract the contents of memory location pointed by HL register, 1020H, i.e. data 10H from the contents of accumulator. i.e. data 50H and store the result (40H) in accumulator

8. SBB (1 byte instruction): Subtract source and borrow from accumulator.

Syntax: SBB R, SBB 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.

Example:

Suppose: Carry flag = 1, C = 20H, A = 40H

SBB C ➔ This instruction will subtract the contents of C register (40H) and carry flag (1) from the content of accumulator (40H). It will store the result (40H – 20H-1H= 1FH) in the accumulator.

9. SUI (2 byte instruction): Subtract immediate from accumulator

Syntax: SUI 8-bit data

This instruction subtracts an 8-bit data given within the instruction from the contents of the accumulator and stores the result in the accumulator.

Example:

SUI 30H. (Subtract 30H from A).

Initially After execution

A=80H, A=50H.

10. SBI (2 byte instruction): Subtract immediate 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.

Example:

Suppose: Carry flag =1, A = 50H

SBI 20H➔ A =50H-20H-1= 2FH

11. INR (1 byte instruction): Increment register or memory by 1.

Syntax: INR R; INR M

The contents of 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 C. (Increment the content of C by 1).

Suppose the Data at C register is 10H.

Initially After execution

C= 10H C=11H.

12. INX (1 byte instruction): Increment register pair by 1

Syntax: INX Rp

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

Example:

INX SP (Increment the content of Stack pointer pair by 1).

INX B. (Increment the content of BC pair by 1).

Suppose the Data at BC register is 1010H and SP is C200H

Initially	After execution
BC= 1010H	BC=1011H.
SP=C200H	SP=C201H.

13. DCR (1 byte instruction): Decrement register or memory by 1.

Syntax: DCR R; DCR M

The content of designated register or memory is 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:

Suppose: HL = 2050, [2050] =21H

DCR M ➔ [2050] = 21-1=20H

14. DCX (1 byte instruction): Decrement register pair by 1.

Syntax: DCX Rp

This instruction decrements the contents of register pair by one. The result is stored in the same register pair. The Rp is register pair such as BC, DE, HL or stack pointer (SP). Only higher order register is to be specified within the instruction.

Example:

Suppose: DE = 1020H

DCX D ➔DE = 1020-1= 101FH

4.5.2. Logical Instruction

1. CMP (1 byte instruction): Compare register or memory with accumulator.

Syntax: CMP R; CMP 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 C (Compare the content of C with A).

Suppose the Data at C register is 17H.

Initially	After execution
C= 10H, A=17H	A=17H, C=17H.

Flags Affected: S=0, Z=0,P=0, Cy=reset, AC=reset.

2. CPI (2 byte instruction): Compare immediate 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 10H (Compare the content of C with A).

Initially	After execution

A=17H A=17H.

Flags Affected: S=0, Z=0, P=0, Cy=reset, AC=reset.

3. ANA (1 byte instruction): Logical AND register or memory with accumulator

Syntax: ANA R; ANA 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 C (AND the content of C with A).

Suppose the Data at C register is 10H.

Initially After execution

C= OFH, A=AAH A=0AH, C=0FH.

Flags Affected: S, Z, P are modified CY=reset, AC=set.

4. ANI (2 byte instruction): Logical AND immediate with accumulator

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 10H (AND 10H with A).

Initially After execution

A=10H A=10H

Flags Affected: S, Z, P are modified Cy=reset, AC=set.

5. XRA (1 byte instruction): EX-OR register or memory with accumulator

Syntax: XRA R; XRA 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 C (XOR the content of C with A).

Suppose the Data at C register is 17H.

Initially	After execution
C= 17H, A=10H	A=07H, C=17H.

Flags Affected: S, Z, P are modified CY=reset, AC=reset.

6. XRI (2 byte instruction): EX-OR immediate 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

7. ORA (1 byte instruction): Logical OR register or memory with accumulator

Syntax: ORA R; ORA 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

8. ORI (2-byte instruction): Logical OR immediate with accumulator.

Syntax: ORI R; ORI M

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 10H (OR 10H with A).

Initially After execution

A=30H A=30H

Flags Affected: S, Z, P are modified CY=reset, AC=set.

9. CMA (1 byte instruction): Complement accumulator

Syntax: CMA

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

10. RLC (1 byte instruction): Rotate accumulator left

Syntax: RLC

This instruction rotates the contents of the accumulator left by one position. Bit D7 is placed in D0 as well as in CY.

Example: A = 01010111 (57H) and CY = 1

RLC ➔ After executing the instruction the accumulator contents will be (10101110) AEH and carry flag will reset.

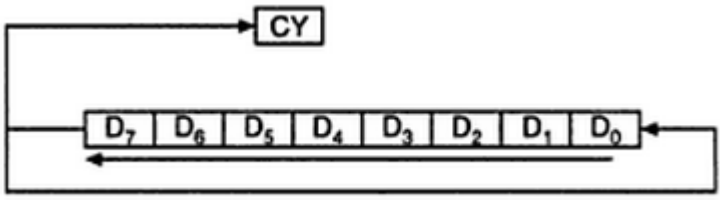


Figure 4-8: Pictorial representation of RLC

11. RRC (1 byte instruction): Rotate accumulator right.

Syntax: RRC

This instruction rotates the contents of the accumulator right by one position. Bit D0 is placed in D7 as well as in CY.

Example: A = 1001 1010 (9AH) and CY =1

RRC ➔ After executing this instruction the accumulator contents will be (0100 1101) 4DH and carry flag will reset.

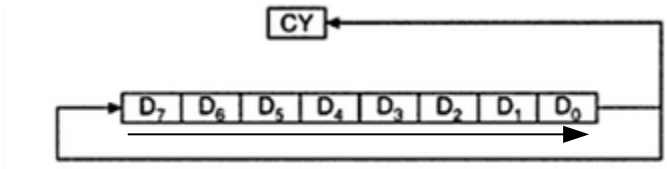


Figure 4-9: Pictorial representation of RRC

12. RAL (1 byte instruction): Rotate accumulator left through carry

Syntax: RAL

This instruction rotates the contents of the accumulator left by one position. Bit D7 is placed in CY and CY is placed in D0.

Example: A = 10101101 (ADH) and CY =0

RAL ➔ After execution of the instruction accumulator contents will be (0101 0110) 5AH and carry flag will set.

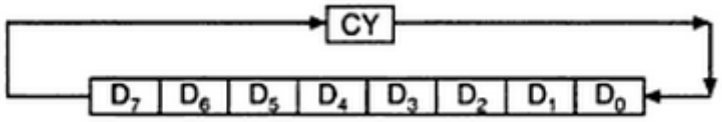


Figure 4-10: Pictorial representation of RAL

13. RAR (1 byte instruction): Rotate accumulator right through carry

Syntax: RAR

This instruction rotates the contents of the accumulator right by one position. Bit D0 is placed in CY and CY is placed in D7.

Example: A = 1010 0011 (A3H) and CY =0

RAR➔ After executing the instruction accumulator contents will be (0101 0001) 51H and carry flag will set.

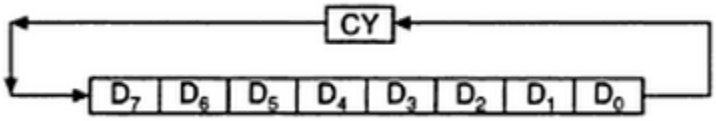


Figure 4-11: Pictorial representation of RAR

14 CMC (1 byte instruction): Complement carry

Syntax: CMC

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

15. STC (1 byte instruction): Set Carry

Syntax: STC

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

4.5.3. Branching Instruction

These instructions are also called control transfer instruction because with these instructions the control of operation is transferred from one location to another. The control transfer may be conditional or unconditional.

1. JMP (3 byte instruction): 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

2. Jump conditionally: Jump if condition is true.

This instruction causes a jump to an address given in the instruction if the desired condition occurs in the program before the execution of the instruction. Table 5-1 shows the possible condition for jumps.

Table 5-1: Conditional Jumps

Opcode	*Description*	*Condition for jump*
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*

3. CALL (3 byte instruction): 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.

4. Call conditionally: Call if condition is true.

This instruction calls the subroutine at the given address if a specified condition is satisfied. Before call it stores the address of instruction next to the call on the stack and decrement stack pointer by two. Table 5-2 shows the possible condition for calls.

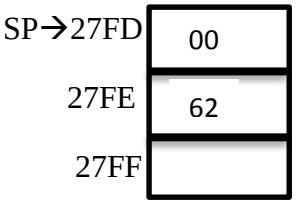
Table 5-2: Conditional Calls

Opcode	*Description*	*Condition for call*
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*

5. RET (1 byte instruction): Return from subroutine unconditionally.

This instruction pops the return address (address of the instruction next to CALL in the main program) from the stack and loads program counter with this return address. Thus transfer program control to the instruction next to call in the main program.

Example: If SP = 27FDH and contents of the stack are as shown then.



RET➔ This instruction will load the Pc with 6200H and it will transfer the program control to the address 6200H. It will also increment the stack pointer by two.

4.5.4. Machine Control Instruction

Opcode	Operand	Explanation of Instruction	Description
NOP	**none**	No operation	No operation is performed. The instruction is fetched and decoded. However no operation is executed. Example: NOP
HLT	**none**	Halt and enter wait state	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
DI	**none**	Disable interrupts	The interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled. No flags are affected. Example: DI
EI	**none**	Enable interrupts	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
RIM	**none**	Read interrupt mas	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 interpretations shown in fig. 4-11 Example: RIM
SIM	**none**	Set interrupt mask	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 shown in fig. 4.12 Example: SIM

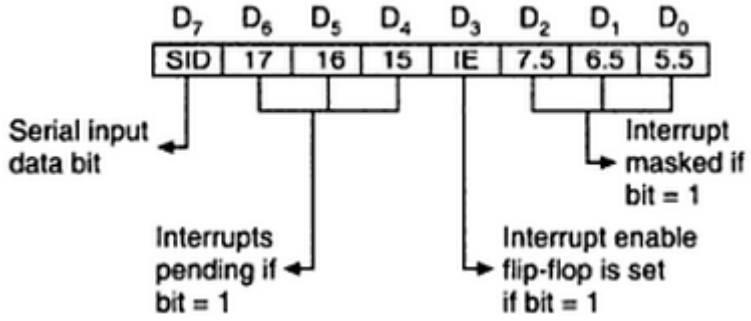


Figure 4-12: Format of RIM instruction

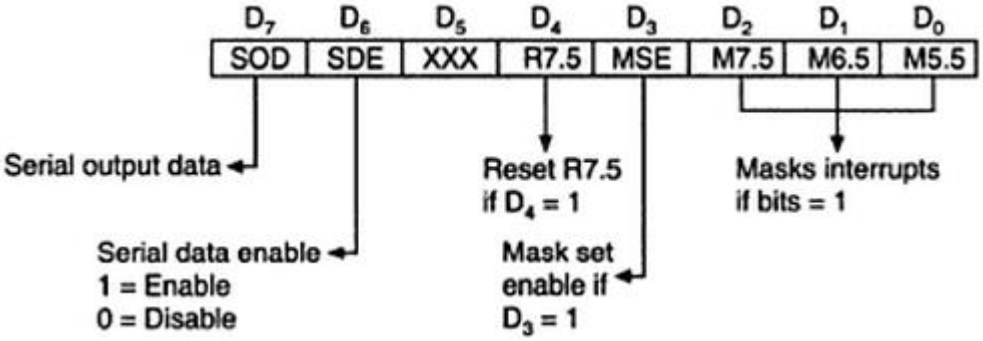


Figure 4-13: Format of SIM instruction

5.3. 8085 Programming practice

1. Write a program in 8-bit Microprocessor to store 60h, BAh, 7Ch and 10h in the memory location starting from 2000h. Add these data and store the result in 3000h and carry flag in 5001h. Explain all the steps (2066 – 5 marks)

Solution:

```
MVI A, 60H
STA 2000H
MVI A, BAH
STA 2001H
MVI A, 7CH
STA 2002H
```

```
MVI A, 10H
STA 2003H
MVI C, 04H
MVI B, 00H        ; clear B register to store the carry from sum.
XRA A             ; clear the accumulator contents
LXI H, 2000H      ; load HL register with starting address of array 2000h
LOOP: ADD M       ; add the memory content with accumulator
JNC AGAIN
INR B
AGAIN: INX H
DCR C
JNZ LOOP
STA 5000H
MOV A, B
STA 5001H
HLT
```

2. Write a program in 8-bit Microprocessor to store 45h, A0h, B5h, and 15h in the memory location stating from 4000h. Add these data and store the result in 5000h and carry flag in 5001h (2069- 10 marks)

Solution: Refer answer 1.

3. Write a program in 8085 microprocessor to subtract 16 bit number at 2000h from a 16 bit number at 2010h and store the result at 2020h (2070-5 marks)

Solution:

```
LDA 2000H
MOV B, A
LDA 2010H
SUB B
STA 2020H

LDA 2001H
MOV B, A
LDA 2011H
SBB B
STA 2021H
HLT
```

4. Write an assembly language program to add two 16-bit numbers (3467h and ACDCh) (2067- 5 marks).

Solution:

Note: The result of addition is stored in HL register. (L-lower byte, H-higher byte)

```
LXI B, 3467H
```

```
LXI D, ACDCH
MOV A, C
ADD E
MOV L, A
MOV A, B
ADC D
MOV H, A
HLT
```

5. Write an assembly language program to multiply 05h and 06h. Explain all the steps. (2066- 5 marks)

Solution:

```
MVI B, 06H
MVI C, 05H
MVI A, 00H
UP: ADD B
DCR C
JNZ UP
HLT
```

6. Write an assembly language program to multiply two numbers (2069- 5 marks)

Solution: Refer to answer 5.

7. Write a program in 8-bit microprocessor to store 68h, B3h, C0H, and 11h in the memory location starting from 3000h. Move these data and store in the memory location starting from 3200h. (2068- 10 marks)

Solution:

```
MVI A, 68H
STA 3000H
MVI A, B3H
STA 3001H
MVI A, C0H
STA 3002H
MVI A, 11h
STA 3003H

MVI C, 04H
LXI H, 3000H
LXI D, 3200H
UP: MOV A, M
```

```
STAX D
INX H
INX D
DCR C
JNZ UP
HLT
```

8. Write an assembly language program to add 16-bit number (2068-5 marks)

Solution: Refer answer 4

9. Write an assembly language program to subtract two 16-bit numbers.

Solution: Refer answer 3

10. Write a program in 8-bit microprocessor to multiply two 16-bits numbers and store in the memory location starting from 3500h. Save the carry bits in the location starting from 3600h. (2065 – 10 marks)

Solution: Try yourself

11. Write a program in 8-bit microprocessor to multiply two 16-bit numbers (ABCDh and 1234h) and store in the memory location starting from 3000h. (2067- 10 marks)

Solution: Try yourself

Some Important practice questions:

1. A list of 50 numbers is stored in memory, starting at 6000H. Find number of negative, zero and positive numbers from this list and store these results in memory locations 7000H, 7001H, and 7002H respectively.
Source Program:

```
      LXI H, 6000H          : Initialize memory pointer
      MVI C, 00H            : Initialize number counter
      MVI B, 00H            : Initialize negative number counter
      MVI E, 00H            : Initialize zero number counter
      MVI D, 00H            ;Initialize positive number counter
BEGIN:MOV A, M              : Get the number
      CPI 00H               : If number = 0
      JZ ZERONUM            : Goto zeronum
      ANI 80H               : If MSB of number = 1i.e. if
      JNZ NEGNUM            number is negative goto NEGNUM
      INR D                 : otherwise increment positive number counter
      JMP LAST
ZERONUM:INR E               : Increment zero number counter
```

```
    JMP LAST
NEGNUM:INR B          : Increment negative number counter
LAST:INX H            : Increment memory pointer
    INR C             : Increment number counter
    MOV A, C
    CPI 32H           : If number counter = 5010 then
    JNZ BEGIN         : Store       otherwise check next number
    LXI H, 7000       : Initialize memory pointer.
    MOV M, B          : Store       negative number.
    INX H
    MOV M, E          : Store       zero number.
    INX H
    MOV M, D          : Store positive number.
    HLT               : Terminate execution
```

2. **Write a program to separate even numbers from given numbers.**

Statement: Write an assembly language program to separate even numbers from the given list of 50 numbers and store them in the another list starting from 2300H. Assume starting address of 50 number list is 2200H.

Source Program:

```
LXI H, 2200H : Initialize memory pointer l
LXI D, 2300H : Initialize memory pointer2
MVI C, 32H : Initialize counter
BACK:MOV A, M : Get the number
ANI 0lH : Check for even number
JNZ SKIP : If ODD, don't store
MOV A, M : Get the number
STAX D : Store the number in result list
INX D : Increment pointer 2
SKIP: INX H : Increment pointer l
DCR C : Decrement counter
JNZ BACK : If not zero, repeat
HLT : Stop
```

7. A sequence of 20 signed numbers are stored at memory location 6000h onwards. You need to write an 8085 assembly language program to transfer all the negative numbers to memory location 5050h onwards and positive numbers to location 4040h onwards.

```
LXI H, 6000H
LXI B, 5050H        ; Address for negative number
LXI D, 4040H        ; Address for positive number

UP: MOV A, M        ; fetch the data
RLC
JNC POSITIVE
RRC
STAX B
INX B
```

```
JMP NEXT

POSITIVE: RRC
STAX D
INX D                ; Access for next memory location

NEXT:
INX H
MOV A, L
CPI 14H
JNZ UP
HLT
```

8. Write a program to find the smallest number in a given array data. The array is stored in the memory from 9200H onwards. Store the result at the memory location 9300H. The length of array is stored as a first element of an array and then data array starts.

Program:

```
LXI H, 9200H
MVI C, M
INX H
MVI A, FFH
UP: CMP M
JC NEXT       ; A< M then to next
MOV A, M
NEXT: INX H
DCR C
JNZ UP
STA 9300H
HLT
```

5.4. Programming with Intel 8086 microprocessor

Assembly Instruction Format

Programs in assembly language are represented by certain words representing the operation of instruction. Thus programming gets easier. These words (usually two-to-four letter) is used to represent each instruction are called Mnemonics. Assembly language statements are generally written in a standard form that has four fields.

- Label field
- Instruction, Mnemonic or Op-code field
- Operand field
- Comment field

Let us consider a simple example to add two numbers

Label	Mnemonic	Operand	Comment
Start:	MVI	A, 10h	; Move 10h into accumulator
	MVI	B, 20h	; Move 20h into register B
	ADD	B	;Add the contents of register B with accumulator

Thus, we see the ease with the assembly language rather than machine language.

Assembler:

An assembler is a program which translates assembly language program into machine language program or object code. Assembly language program are called source codes and machine language programs are known as object codes.

A translator converts source codes to object codes and then into executable formats. The process of converting source code into object code is called *compilation* and assembler does it. The process of converting object codes into executable formats is called *linking* and linker does it.

A **macro-assembler** is an assembler which allows user to define sequence of instruction as micro. The Microsoft Macro Assembler (MASM) is an x86 assembler that uses the Intel syntax for MS-DOS and Microsoft Windows.

Assembler can be linked with two or more than two assembler called *linking assemblers*.

Assembler directives:

There are some instruction in the assembly language program which are not a part of processor instruction set. These instructions are instruction to the assembler, linker, and loader. These are referred to as pseudo-operations or as assembler directives. The assembler directives are executed by an assembler at assembly time, not by a CPU at run time.

The commonly used assembler directives are:

ASSUME

The 8086, at any time, can directly address four physical segments which include a code segment, a data segment, a stack segment and an extra segment. The 8086 may contain a number of logical segments. The ASSUME directive assigns a logical segment to a physical segment at any time. That is, the ASSUME directive tells the assembler what addresses will be in the segment registers at execution.

Example: ASSUME CS: code, DS: Data, SS: stack

.CODE

This directive provides shortcut in definition of the code segment.

.DATA

This directive provides shortcut in definition of the data segment.

.STACK

This directive provides shortcut in definition of stack segment. General format for this directive is as shown below:

.STACK [size]

Example:

.STACK 100 ; this reserves 100 byte for the stack operation.

DB, DW, DD, DQ, and DT: These directives are used to define different types of variables, or to set aside one or more storage locations of corresponding data type in memory. Their definition are as follows:

DB- Define Byte

DW – Define Word

DD- Define Double word

DQ- Define Quad word

DT- Define Ten Byte

Example:

AMOUNT DB 10H, 20H, 30H, 40H ; Declare array of 4 bytes named AMOUNT

```
MES DB ‘WELCOME          ; Declare array of 7 bytes and initialize with ASCII codes for
                         ; letters in WELCOME
```

DUP:

The DUP directive can be used to initialize several locations and to assign values to these locations.

Format: Name Data_Type NUM DUP (value)

Example:

```
TABLE DW 10 DUP (0)      ; Reserve an array of 10 words of memory and initialize all
                          ; 10 words with 0. Array is named TABLE
```

END:

The END directive is put after the last statement of a program to tell the assembler that this is the end of the program module. The assembler ignores any statement after an END directive.

Example:

```
PORT EQU 80
```

LENGTH:

It is an operator which tells the assembler to determine the number of elements in some named data items such as a string or array

Example:

```
MOV BX, LENGTH STRING1        ; Loads the length of string in BX
```

.MODEL

This directive provides short-cuts in defining segments. It is initializes memory model before defining any segment. The memory model can be SMALL, MEDIUM, COMPACT or LARGE. We can choose the memory model based on our requirement by referring following table

Model	**Code segments**	**Data segments**
Small	One	One
Medium	Multiple	One
Compact	One	Multiple
Large	Multiple	multiple

OFFSET:

It is an operator which tells the assembler to determine the offset or displacement of a named data item (variable) from the start of segment which contains it.

Example

MOV AX, OFFSET MES1 ; Loads the offset of variable, MES1 in AX register.

ORG

The assembler uses a location counter to account for its relative position in a data or code segment.

Example:

ORG 100H, set the location counter to 100h

PROC and ENDP:

PROC: The procedures in the programs can be defined by PROC directive. The procedure name must be present, must be unique, and must follow naming conventions for the language. After the PROC directive the term NEAR or FAR are issued to specify the type of the procedure.

Example:

FACT PROC FAR ; Identifies the start of a procedure named FACT and tells the assembler that the procedure is far (in a segment with a different name from that which contains the instruction which calls the procedure).

ENDP: ENDP directive is used along with the PROC directive. ENDP defines the end of the procedure.

Types of Assembler

One Pass Assembler:

This assembler reads the assembly language program once and translates the assembly language program to machine code. This assembler has the program of defining forward references. This means that a branching instruction using an address that appears later in the program must be defined by the programmer after the program is assembled.

Two Pass Assembler:

This assembler scans the assembly language program twice. In the first pass, the assembler generates the table of the symbols. A symbol table consists of labels with addresses assigned to them. In this way labels can be used for JUMP statements, and no address calculation has to be

done by the user. On the second pass, the assembler translates the assembly language program into the machine code. The two pass assembler is thus easier to use.

5.5. Instruction sets

The 8086 instructions can be grouped according to the function they perform. These groups are

1. Data Transfer Group:

a) MOV Instruction

It is a general purpose instruction to transfer byte or word from register to register, register to memory or from memory to register

Syntax: MOV destination, source

Examples:

```
MOV BX, 592FH      ; Load the immediate number 592FH in BX
MOV CL, [357A]     ; Copy the contents of memory location at a displacement of 357AH from
                     data segment base, into CL register
MOV DS, CX         ; Copy word from CX register to data segment
```

b) IN instruction: Input a byte or word from port

The IN instruction will copy data from a port to the accumulator. If an 8-bit port is read, the data will go the AL and if a 16-bit port is read the data will go to AX.

Examples:

```
IN AL, 0C8H  ; Input a byte from port 0C8H to AL
IN AX, 34H   ; Input a word from port 34H to AX
```

c) OUT instruction: Output a byte or word to a port

The OUT instruction copies a byte from AL or a word from AX to the specified port.

Examples:

```
OUT 3BH, AL        ; copy the contents of AL to port 3BH
OUT 2CH, AX        ; copy the contents of AX to port 2CH
```

d) LEA Instruction: Load Effective Address

This instruction determines the offset of the variable or memory location names as the source and loads this address in the specified 16-bit register.

Syntax: LEA register, source

LEA CX, TOTAL ; Load CX with offset of TOTAL in DS

LEA BP, SS: STACK_TOP ; Load BP with offset of STACK_TOP in SS

LEA AX, [BX] [DX] ; Load AX with EX = [BX] + [DI]

2. Arithmetic and Logical Group

a) ADD (Add)/ ADC (Add with Carry):

These instructions add a number from some source to a number from some destination and put the result in the specified register. The add with carry instruction, ADC, also add the status of the carry flag into the result.

Syntax: ADD destination, source / ADC destination, source

Example:

ADD AL, 74H ; AL ← AL + 74H

ADC CL, BL ; CL ← CL + BL + CY

ADD DX, BX ; DX ← DX + BX + CY

b) SUB (subtract)/ SBB (Subtract with borrow) instruction

These instructions subtract the number in the source from the number in the destination and put result in the destination. The SBB, instruction also subtract the status of the carry flag from the result.

Syntax: SUB destination, source/ SBB destination, source

Examples:

SUB AL, 0F0H ; AL ← AL- 0F0H

SBB DL, CL ; DL ← DL- CL-CY

SBB DX, BX ; DX ← DX – BX -CY

c) INC: Increment destination (add by 1)

The INC instruction adds 1 to the specified destination. The destination may be a register or memory location.

Example:

```
INC AL      ; Add 1 to contents of AL

INC BX      ; Add 1 to contents of BX
```

d) DEC: Decrement destination (subtract by 1)

The DEC instruction subtract 1 from the specified destination. The destination may be a register or a memory location.

Examples:

```
DEC AL      ; Subtracts 1 from the contents of AL

DEC BX      ; Subtract 1 from the contents of BX
```

e) MUL Instruction

This instruction multiplies an unsigned byte from source and unsigned byte in AL register or unsigned word from source and unsigned word in AX register.

Examples:

```
MUL BL      ; AL x BL result in AX

MUL BX      ; AX x BX, result high word in DX low word in AX.
```

f) DIV instruction: DIV source

This instruction is used to divide an unsigned word by a byte or to divide an unsigned double word by a word.

Examples:

```
DIV CL      ; Word in AX/byte in CL, quotient in AL, remainder in AH.
```

g) AND instruction:

This instruction logically ANDs each bit of the source byte or word with the corresponding bit in the destination and stores result in the destination.

Examples:

```
AND BL, AL  ; AND byte in AL with byte in BL
```

h) OR instruction:

The instruction logically ORs each bit of the source byte or word with the corresponding bit in the destination and stores result in the destination.

Examples:

OR BL, AL ; OR byte in AL with byte in BL

OR CX, 00F0H; OR 00F0H with byte in CX

i) XOR instruction: XOR destination, source

This instruction logically XORs each bit of the source byte or word with the corresponding bit in the destination and stores result in the destination.

Examples:

XOR BL, AL ; XOR byte in AL with byte in BL

j) NOT instruction: NOT destination

The Not instruction inverts each bit of a byte or a word. The destination can be register or a memory location.

Examples:

NOT AL

NOT CX

k) CMP instruction:

The comparison instruction (CMP) compares a byte/ word from the specified source with a byte/ word from the specified destination. The source and destination both must be byte or word. The source may be an immediate number, a register, or a memory location. The destination may be a register or a memory location. However the source and destination both can't be a register or a memory locations. Source and destination remain unchanged, only flags are updated.

Examples:

CMP BL, 01H ; Compare immediate number 01H with byte in BL

CMP CX, BX ; Compare word in BX with word in CX

l) DAA instruction: Decimal Adjust Accumulator.

It is used to make sure that the result of adding two BCD numbers is adjusted to be a correct BCD number. It only works on AL register.

If low nibble of AL > 9 or AF = 1 then:

- AL = AL + 6
- AF = 1

If AL > 9Fh or CF = 1 then:

- AL = AL + 60h
- CF = 1

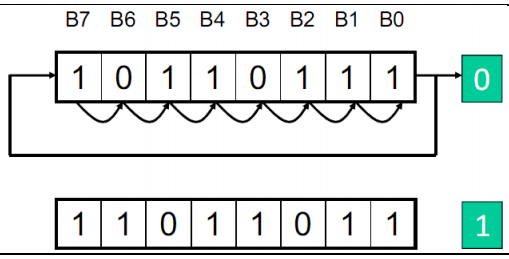
M) AAA instruction: ASCII Adjust for Addition.

The AAA instruction is executed after an ADD instruction that adds two ASCII coded operand to give a byte of result in AL. The AAA instruction converts the resulting contents of AI to an unpacked decimal digit.
Eg.
ADD CL, DL; [CL] = 32H = ASCII for 2, [DL] = 35H = ASCII for 5, Result [CL] = 67H
MOV AL, CL; move ASCII result into AL since AAA adjust only [AL]
AAA ; [AL] = 07, unpacked BCD for 7

N) ROR instruction: ROR destination, count

This instruction rotates all bits in a specified byte or word to the right some number of bit positions. LSB is placed as a new MSB and a new CF. If the number of bits desired to be shifted is 1, then the immediate number 1 can be written in Count. However, if the number of bits to be shifted is more than 1, then the count is put in CL register.

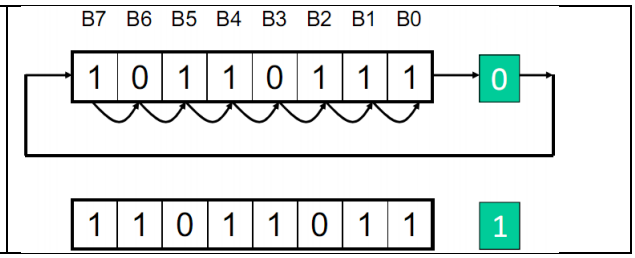
Examples:

ROR BL, 01 ROR AL, CL	

o) RCR Instruction: RCR destination, count.

This instruction rotates all of the bits in a specified word or byte some number of bit positions to the right along with the carry flag. LSB is placed as a new carry and previous carry is placed as a new MSB

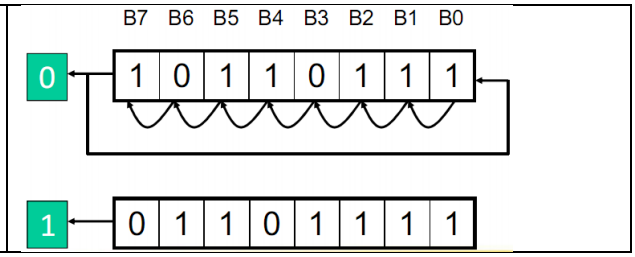
```
RCR CX, 1
MOV CL, 04H
RCR AL, CL
RCR AL, CL
```



p) ROL Instruction: ROL destination, count

This instruction rotates all bits in a specified byte or word to the left some number of bit positions. MSB is placed as a new LSB and a new CF.

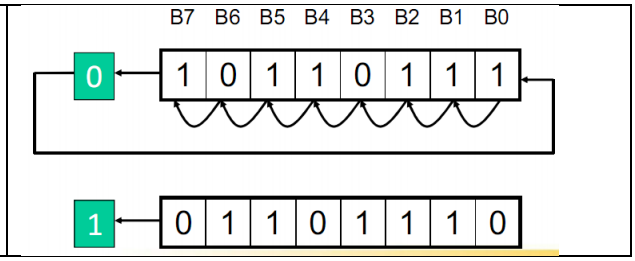
```
ROL CX, 1
MOV CL, 03H
ROL BL, CL
```



O) RCL Instruction: RCL Des, Count:

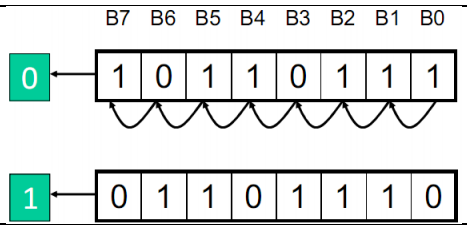
It rotates bits of byte or word left, by count. MSB is transferred to LSB and also to CF. If the number of bits desired to be shifted is 1, then the immediate number 1 can be written in Count. However, if the number of bits to be shifted is more than 1, then the count is put in CL register.

```
RCL AX, 1
RCL BL, CL
```



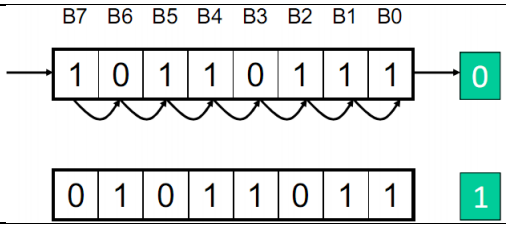
P) SHL Instruction: SHL Des, Count

It shift bits of byte or word left, by count. It puts zero(s) in LSBs. MSB is shifted into carry flag. If the number of bits desired to be shifted is 1, then the immediate number 1 can be written in Count. However, if the number of bits to be shifted is more than 1, then the count is put in CL register.

SHL CX, 1 MOV CL, 05H SHL AX, CL	

Q) SHR Instruction: SHR destination, count

This instruction shifts each bit in the specified destination to the right and 0 is stored at MSB position. The LSB is shifted into the carry flag.

SHR CX, 1 MOV CL,05H SHR AX, CL	

3. Branching Instruction

a) JMP Label:

This instruction is used for unconditional jump from one place to another.

b) Jxx Label (Conditional Jump):

All the conditional jumps follow some conditional statements or any instruction that affects the flag.

Mnemonic	**Meaning**	**Jump Condition**
JA	Jump if Above	CF = 0 and ZF = 0
JAE	Jump if Above or Equal	CF = 0
JB	Jump if Below	CF = 1
JBE	Jump if Below or Equal	CF = 1 or ZF = 1
JC	Jump if Carry	CF = 1
JE	Jump if Equal	ZF = 1

JNC	Jump if Not Carry	CF = 0
JNE	Jump if Not Equal	ZF = 0
JNZ	Jump if Not Zero	ZF = 0
JPE	Jump if Parity Even	PF = 1
JPO	Jump if Parity Odd	PF = 0
JZ	Jump if Zero	ZF = 1

c) CALL Instruction: CALL Des:

This instruction is used to call a subroutine or function or procedure.

The address of next instruction after CALL is saved onto stack.

d) RET:

It returns the control from procedure to calling program.

Every CALL instruction should have a RET.

e) LOOP: Loop Des:

This is a looping instruction. The number of times looping is required is placed in the CX register. With each iteration, the contents of CX are decremented. ZF is checked whether to loop again or not.

4. Stack Instruction:

a) PUSH Instruction: PUSH Operand

This instruction pushes the contents of the specified register/memory location on to the stack. The stack pointer is decremented by 2, after each execution of this instruction.

Example

PUSH AX

PUSH DX

PUSH [5000H]

b) POP Instruction: POP Des

This instruction when executed, loads the specified register/memory location with the contents of the memory location of which the address is formed using the current stack segment and stack pointer. The stack pointer is incremented by 2.

Example

POP AX

POP DS

POP [5000H]

5.6. Software Interrupt

Software interrupts are like hardware interrupts which are generated by the program itself. From the interrupt number, the CPU derives the address of the Interrupt service routine which must be executed.

- Software interrupts in assembly language can be treated as calls to subroutines of other programs which are currently running on the computer.
- One of the most famous software interrupt is Interrupt No. 21H, which branches in the operating system, and permits the use of PC-DOS functions defined there.
- The function required to be performed by DOS is specified in AH prior to the interrupt.
- The functions return and accept values in various registers.
- AN interrupt is called using the instruction INT followed by the interrupt number

For example: INT 21H

Function number	Description	Explanation
1	Keyboard Input (echoed)	Waits until a character is typed at the keyboard and then puts the ASCII code for that character in register AL and echoed to screen
2	Display Output	Prints the character whose ASCII code is in DL
8	Keyboard Input (No echo)	Waits until a character is typed at the keyboard and then puts the ASCII code for that character in register AL and NOT echoed to screen
9	Display String	Prints a series of characters stored in memory starting with the one in the address given in DX (relative to DS).Stop when the ASCII code for $ is encountered

5.7. INT 21 Instruction

INT 21h is a common function. The INT 21H instruction in 8086 is a software interrupt to vector 21H. It is used for input/output operations.

AH	Description	AH	Description
01	Read character from STDIN	02	Write character to STDOUT
05	Write character to printer	06	Console Input/Output
07	Direct char read (STDIN), no echo	08	Char Read from (STDIN), no echo
09	Write string to STDOUT	0A	Buffered input
0B	Get STDIN status	0C	Flush buffer for STDIN
0D	Disk reset	0E	Select default drive
19	Get current default drive	25	Set Interrupt vector
2A	Get system date	2B	Set system date
2C	Get system time	2D	Set system time
2E	Set verify flag	30	Get DOS version
35	Get interrupt vector		
36	Get free disk space	39	Create subdirectory
3A	Remove subdirectory	3B	Set working directory
3C	Create file	3D	Open file
3E	Close file	3F	Read file
40	Write file	41	Delete file
42	Seek file	43	Get/Set file attributes
47	Get current directory	4C	Exist program
4D	Get return code	54	Get verify flag
56	Rename file	57	Get/Set file date

** STDIN – Standard input

**STDOUT-Standard output

5.8. 8086 Programs

1. Write an assembly language to add two 8-bit numbers.

```
; 8-bit addition -8086

.model small
.stack 100h
.data
        n1 db 56h
        n2 db 98h
        sum dw ?
.code
begin : mov ax, @data
mov ds, ax
```

```
mov ah, 00
mov al, n1              ; al =56h
add al, n2              ; al = 56+98= EEH CY =0
jnc nocarry
inc ah
nocarry: mov sum, ax

mov ah, 4ch             ; service number
int 21h                 ; exit to dos
end begin
```

2. Write a program to display the character “Assembly language programming”

```
; program to display a string - 8086
.model small
.stack 100h
.data
        text db 'Assembly language programming$'
.code
begin:  mov ax, @data
        mov ds,ax

        mov dx, offset text     ; dx<== start address of text
        mov ah, 09h             ; display message text
        int 21h

        mov ah, 4ch     ;service number
        int 21h         ;to exit to DOS
        end begin
```

3. Write a program to add 16-bit numbers

```
; 16-bit addition -8086
.model small
.stack 100h
.data
        n1 dw 1234h
        n2 dw 6789h
        sum dw 2 dup(?)

.code
begin:  mov ax,@data
        mov ds,ax

        mov bx,0h
        mov ax, n1      ; ax = 1234
        add ax, n2      ;ax = 1234+6789
```

```
        mov sum, ax
        adc bx, 0
        mov sum[2], bx

        mov ah, 4ch    ;service number
        int 21h        ;exit to DOS
        end begin
```

4. Write a program to multiply 8-bit numbers.

```
; 8-bit multiplication -8086
.model small
.stack 100h
.data
        n1 db 56h
        n2 db   78h
        product dw ?

.code
begin: mov ax, @data
        mov ds, ax

        mov ah,00
        mov al,n1       ;al=56
        mul n2          ; 56*78 =>ax
        mov product,ax


        mov ah, 4ch    ;service number
        int 21h        ;exit to DOS
        end begin
```

5. Write a program in 8086 to multiply 16-bit numbers

```
.model small
.stack 100h
.data
        n1 dw 7895h
        n2 dw 0ffffh
        product dw 2 dup (?)
.code
begin: mov ax, @data
        mov ds, ax

        mov dx, 0
        mov ax, n1      ; ax= 7895
        mul n2          ; 7895*FFFF=dx:ax
```

```
        mov product, ax
        mov product [2], dx

        mov ah, 4ch     ; service number
        int 21h         ;exit DOS
```

6. Program to find the largest number from an array

```
; largest number -8086
.model small
.stack 100h
.data
        n db  34h, 67h, 89h, 12h, 55h, 90h, 0cdh, 23h, 67h, 88h
        count dw 10
        largest db ?
.code
begin: mov ax, @data
        mov ds, ax

        mov si, 0
        mov cx, count
        mov bl, n[si]
        dec cx
again:  inc si
        cmp bl, n[si]
        jnc noswap
        mov bl, n[si]
noswap: loop again
        mov largest, bl

        mov ah, 4ch
        int 21h
        end begin
```

7. Write a program to read keyboard input and display it to monitor.

Solution:

```
TITLE Read Keyboard Input and Display it to Monitor
.model small
.stack 100h
.data
.code

begin:      mov ax, @data       ; [loads the address of data
            mov ds, ax          ; segment in DS]
back:       mov ah, 01          ;  read character
            int 21h
            cmp al, '0'
            jz last
```

```
            jmp back
last:       mov ah, 4ch         ; [ Exit
            int 21h             ; to DOS]
end begin
```

8. Write a program to reverse a string "I like assembly language programming"
Solution: Try yourself

7. Write an assembly language to display a string "Assembly language coding is difficult" using 16 bit microprocessor code. Assume any necessary data. (2065- 5 marks)
Solution: Refer answer 2
8. Write an assembly language program to display a string "This is a test program" using 16 bit microprocessor code. Assume any necessary data. (2069 -5 marks)
Solution: Refer answer 2
9. Write an assembly language program to display a string "Microprocessor programming is a fun" using 16 bit microprocessor code. Assume any necessary data. (2068 – 5 marks)
Solution: Refer answer 2
10. Write an assembly language program to display a string "I want to know more about microprocessor "using 16 bit microprocessor code. Assume any necessary data. (2066 – 5 marks)
Solution: Refer answer 2.
11. Write an assembly language program to display a string "I like programming in the assembly language "using 16 bit microprocessor code. Assume any necessary data (2067- 5 marks)
Solution: Refer answer 2
12. Write an assembly language program for 8086 to read a string from keyboard and display each work in separate line. The length of input string can be up to 60 characters. (2070 – 10 marks)
Solution: Try yourself