UNIT- III INSTRUCTION SET OF INTEL 8085

An instruction is a command given to the computer to perform a specified operation on given data. Instructions have been classified into following groups:

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

1.Data Transfer Group. Instructions which are used to transfer data from one register to another register, from memory to register or register to memory, come under this group. Examples are: MOV, MVI, LXI, LDA, STA, etc.

Example: MOV A, B
LDA 2000

2. Arithmetic Group. It performs arithmetic operations addition, subtraction, increment/decrement of content of a register or memory. Examples are: ADD, SUB, INR, DAD, DCR, etc.

3.Logical Group. It performs logical operations such as AND, OR, compare, rotate, etc. Examples are: ANA, XRA, ORA, CMP, RAL, etc.

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

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

INSTRUCTION AND DATA FORMATS

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

The various techniques to specify data for instructions are:

- **i.** 8-bit/16-bit data may be directly given in instruction itself
- **ii.** The address of memory location, I/O port/device, where data resides, may be given in the instruction itself
- **iii.** In some instructions only one register is specified. The content of the specified register is one of the operands. The another operand is in accumulator

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

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

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

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

1) Single-Byte Instruction
2) Two-Byte Instruction
3) There-Byte Instruction

STATUS FLAGS

There is a set of five flip-flops which indicate status (conditions) arising after the execution of arithmetic and logic instructions.

SYMBOLS AND ABBREVIATIONS

Symbol/Addressing	**Meaning**
addr	16-bit address of memory location
data	8-bit data
data 16	16-bit data
r, r1, r2,	one of the registers A, B, C, D, E, H or L
A, B, C, D, H, L	8-bit register
A	Accumulator
H-L	Register pair H-L
B-C	Register pair B-C
D-E	Register pair D-E
PSW	Program Status Word
M	Memory whose address is in H-L pair
H	Appearing at the end of a group of digits specifies hexadecimal. Eg. 2500H
rp	One of the register pairs Example : B represents B-C pair, B is the higher order and C lower order, D represents D-C pair, D is higher order register and E is the lower order register pair. H represents H-L pair, H is the higher order register and L is the lower order register. SP represents 16-bit stack

Pointer. SPH is high order 8-bits, and SPL is the low order 8 bits

rh	The higher order register of a register pair
rl	The low order register of a register pair
PC	16-bit Program Counter, PCH is high order 8 bits and PCL Low order 8 bits of register PC.
CS	Carry Status
[ ]	The contents of the register identified with in a bracket
[[ ]]	The content of memory location whose address is in the Register pair identified within brackets.
^	AND operation
V	OR operation
←	Move data in the direction of arrow
↔	Represents Exchange operation

INTEL 8085 INSTRUCTIONS

Some of Intel 8085 instructions are frequently, some occasionally and some seldom used by the programmer. It is not necessary that one should learn all of the instructions to understand simple programs.

DATA TRANSFER GROUP (GROUP I)

1. MOV r1, r2

(Move data: move the content of the one register to another)

The content of register r2 is moved to register r1, for example, the instruction MOV A,B move the content of register B to register A. the instruction MOV B,A moves the content of register A to register B. the time for the execution of this instruction is 4 clock period. One clock period is called state. No flag is affected.

[r1] ← [r2]

2. MOV r.M : (Move the content of memory to register**)**

[r] ← [[H-L]]

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

Example:

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

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

3. MOV M, r: [move the content of register to memory]

[[H-L]] ← [r]

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

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

[r]← data.

Here the 1st byte of the instruction is its opcode. The 2nd byte of the instruction is the data which is moved to register r .For example, the instruction

MVI A,05 moves 05 to register A. The opcode is written as 3E,05

The opcode for MVI A is 3E and 05 is the data which is to be moved to register A.

5. MVI M, data. (Move immediate data to memory whose address is in HL pair).

[[H-L]] ← data.

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

Example

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

HLT Halt

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

6. LXI rp , data 16. (load register pair immediate).

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

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

7. LDA addr. (Load accumulator direct).

[A] ← [addr]

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

8. STA addr. (Store accumulator direct)

[addr] ← [A]

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

9. LHLD addr. (load H-L pair direct).

[L] ← [addr], [H] ← [addr+1].

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

10. SHLD addr. (store H-L pair direct)

[addr] ← [L], [addr+1] ← [H]

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

11. LDAX rp. (LOAD accumulator indirect)

[A] ← [[rp]]

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

12. STAX rp, (Store accumulator indirect)

[[rp]] ← [A]

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

13. XCHG. (exchange the content of H-L with D-E pair)

[H-L] ↔ [D-E]

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

ARITHMETIC GROUP (GROUP II)

1. ADD r. (add register to accumulator).

[A]←[A] + [r].

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

ADD B - The content of B register is added with content of accumulator and the result is stored in accumulator.

2. ADD M. (add memory to accumulator).

[A]←[A] + [[H-L]].

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

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

[A] ← [A] + [r] + [CS].

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

4.ADC M. (add memory with carry to accumulator).

[A] ← [A] + [[H-L]] + [CS]

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

5.ADI data. (Add immediate data to accumulator)

[A] ← [A] + data.

The immediate data is added to the content of the accumulator. The 1st byte of the instruction is its opcode. The 2nd byte of the instruction is data, and it is added to content of the accumulator. The sum is placed in the accumulator. For example, the instruction ADI 08 will add 08 to the content of the accumulator and place the result in the accumulator.

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

[A] ← [A] + data + [CS]

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

7.DAD rp. (add register pair to H-L pair)

[H-L] ← [H-L] + [rp] .

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

8.SUB r.(subtract register from accumulator)

[A] ← [A] - [r].

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

Eg. SUB B – The content of B register is subtracted from content of accumulator and the result or sum is stored in accumulator.

9. SUB M. (subtract memory from accumulator).

[A] ← [A] -- [[H-L]]

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

10.SBB r (subtract register from accumulator with borrow)

[A] ← [A] -- [r] -- [CS].

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

11.SUI data. (subtract immediate data from accumulator)

[A] ← [A] -- data.

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

12.SBI data.(subtract immediate data from accumulator with borrow)

[A] ← [A] – data -- [CS].

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

13. INR r (increment register content)

[r] ← [r] + 1.

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

14. INR M (increment memory content)

[[H-L]] ← [[H-L]] + 1

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

15.DCR r (decrement register content)

[r] ← [r] - 1

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

16.DCR M. (decrement memory content)

[[H-L]] ← [[H-L]] -- 1

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

17.INX rp, (increment register pair)

[rp] ← [rp] + 1

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

18. DCX rp (decrement register pair)

[rp] ← [rp] - 1

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

19. DAA (Decimal Adjust Accumulator)

This instruction is used in the program after ADD, ADI, ACI and ADC instructions. After the execution of ADD, ADC instructions, the result is in hexadecimal form and it is placed in the accumulator. The DAA instruction operates on this result and gives the final result in the decimal system.

LOGICAL GROUP (GROUP III)

The instructions of this group perform AND, OR, EX-OR operation, compare and rotate or complement of the data stored in memory or register.

1. **ANA r. (AND register with Accumulator)**

   [A] ← [A] ^ [r]

   The content of register r is ANDed with the content of the accumulatore and the result is placed in the accumulator.

ANA C , AND content of C register with accumulator and the result is stored in accumulator.

2. **ANA M. (AND memory with the Accumulator)**

   [A] ← [A] ^ [[H-L]]

   The content of memory location addressed by H-L pair is ANDed with the accumulator and the result is placed in the accumulator.

3. **ANI data. (AND immediate data with Accumulator)**

   [A] ← [A] ^ data

   The data which is the second byte of the instruction is ANDed with the content of the accumulator and the result is placed in the Accumulator.

4. **ORA r (OR register with Accumulator)**

   [A] ← [A] v [r]

   The content of register r is ORed with the content of the accumulator and the result is placed in the accumulator.

   ORA C , The content of Register C is Logically ORed with content of accumulator and the result is stored in accumulator.

5. **ORA M . (OR memory with Accumulator).**

   [A] ← [A] v [[H-L]]

   The content of memory location addressed by H-L pair is ORed with the content of the accumulator and the result is placed in the accumulator.

6. **ORI data. (OR immediate data with accumulator)**

   [A] ← [A] v data.

   The data which is the second byte of the instruction which is ORed with the content of accumulator and the result is placed in the accumulator.

7. **XRA r . (EXCLUSIVE –OR register with the accumulator)**

$[A] \leftarrow [A] \oplus [r]$

The content of register r is EXCLUSIVE-ORed with the content of accumulator and the result is stored in accumulator.

XRA C , The content of C register is logically EX-ORed with content of accumulator and the result is stored in accumulator.

8. XRA M. (EXCLUSIVE-OR memory with Accumulator)

$[A] \leftarrow [A] \oplus [[H\text{-}L]]$

The content of memory location addressed by H-L pair is EXCLUSIVE-ORed with the content of accumulator and the result is placed in accumulator.

9. XRI data. (EXCLUSIVE-OR immediate data with accumulator)

$[A] \leftarrow [A] \oplus \text{data}$

The data which is the second byte of the instruction which is EXCLUSIVE-ORed with the content of accumulator and the result is placed in the accumulator.

10. CMA (Complement the accumulator).

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

The one's complement of the accumulator which is obtained and the result is placed in the accumulator.

11. CMC (Complement the carry status)

$[CS] \leftarrow [\overline{CS}]$

The carry flag is complemented.

12. STC (Set carry status)

$[CS] \leftarrow 1$

The status flag CS is set to 1.

13. CMP r. (Compare register with accumulator).

$[A] - [r]$

The content of register r is subtracted from the content of accumulator. But the result of the subtraction is discarded and the content of accumulator remain unchanged.

14. CMP M. (Compare memory with accumulator)

[A] - [[H-L]]

The content of memory location addressed by H-L pair is subtracted from the content of the accumulator and the result of subtraction is discarded and the content of accumulator remain unchanged.

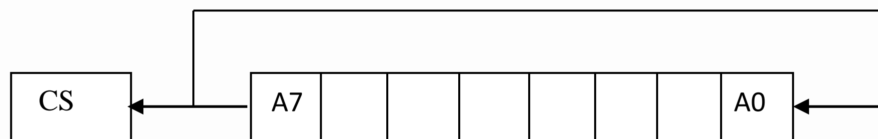
15. CPI data. (Compare immediate data with accumulator).

[A] – data

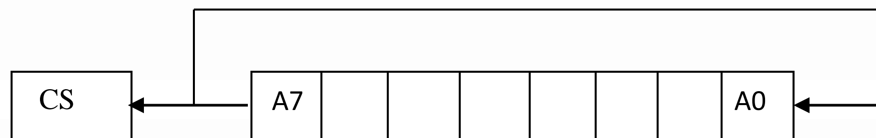
The data which is the second byte of the instruction is subtracted from the content of accumulator and the result of the subtraction is discarded. The content of accumulator remains unchanged.

ROTATE INSTRCTIONS

The rotate instructions rotate the content of accumulator to left or right with carry or without carry. In rotate instructions the default operand is accumulator. There are four rotate instructions, they are RLC, RRC,RAL and RAR. The following diagram shows accumulator with carry.



1. RLC (Rotate accumulator left)

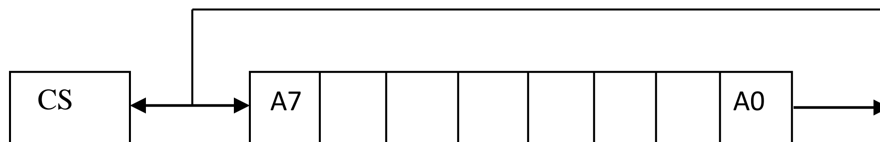


Carry status Accumulator

$[A_{n+1}] \leftarrow [A_n]$, $[A_0] \leftarrow [A_7]$, $[CS] \leftarrow [A_7]$

The Content of accumulator is rotated left by one bit. The seventh bit of accumulator is moved to carry bit as well as the zero bit of the accumulator . Only CS flag is affected.

2. **RRC (Rotate accumulator right)**

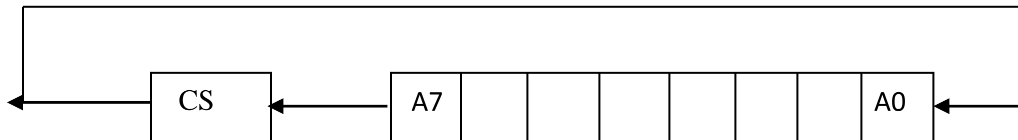


CARRY STATUS ACCUMULATOR

$[A_7] \leftarrow [A_0]$, $[CS] \leftarrow [A_0]$, $[A_n] \leftarrow [A_{n+1}]$

The content of accumulator is rotated right by one bit. The zero bit of the accumulator is moved to the seventh bit as well as carry bit. Only CS flag is affected.

3. **RAL (Rotate accumulator left through carry)**

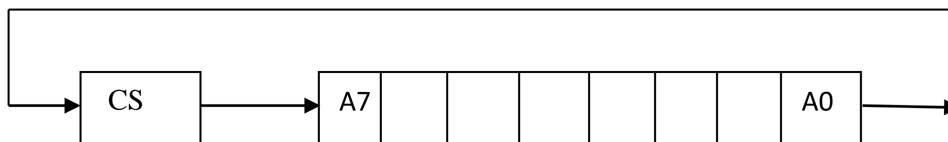


CARRY STATUS ACCUMULATOR

$\mathbf{[A_{n+1}]} \leftarrow [A_n]$, $[CS] \leftarrow [A_7]$, $[A_0] \leftarrow [CS]$

The content of accumulator is rotated left one bit through carry. The seventh bit of the accumulator is moved to carry and carry bit is moved to the zero bit of the accumulator. Here only CS flag is affected.

4. **RAR (Rotate accumulator right through carry)**



CARRY STATUS ACCUMULATOR

$[A_n]$ $\leftarrow [A_{n+1}]$, $[CS] \leftarrow [A_0]$, $[A_7] \leftarrow [CS]$

The content of the accumulator is rotated right one bit through carry. The zero bit of the accumulator is moved to carry and the carry bit is moved to the seventh bit of the accumulator. Only CS flag is affected.

BRANCH GROUP (GROUP IV)

The instructions of this group change the normal sequence of the program. There are two types of branch instructions: conditional and unconditional. The conditional branch instructions transfer the program to specified label when certain condition is specified. The unconditional branch instructions transfer the program to specified label unconditionally.

Unconditional JUMP

JMP addr (label). (Jump to instruction specified by address)

[PC] ← Label.

2nd and 3rd byte of instruction give address of label where program jumps. The address of label is address of memory location for next instruction to be executed. The program jumps to the instruction specified by the address(label) unconditionally.

(Same syntax for unconditional and conditional JUMP instruction)

Conditional JUMP

Conditional Jump addr (label) : After execution of conditional jump the program jumps to instruction specified by label if specified condition is fulfilled. The execution of conditional jump takes 3 machine cycles: 10 states. If condition is not true, only 2 machine cycle; 7 states are required for execution of instruction.

(i) **JZ addr (label).** (jump if result is zero)
Syntax : [PC] ← Label
The program jumps to instruction specified by label if result is zero. Here the result after the execution of preceding instruction is under consideration.

(ii) **JNZ addr (label).** (jump if result is not zero)
Syntax : [PC] ← Label
The program jumps to instruction specified by label if result is non-zero.

(iii) **JC addr (label).** (jump if result is zero)
Syntax : [PC] ← Label
The program jumps to instruction specified by label if there is a carry. Here the carry after the execution of preceding instruction is under consideration.

(iv) **JNC addr (label).** (jump if result is no carry)
[PC] ← Label,
The program jumps to instruction specified by label if there is no-carry.

(v) **JP Addr (label).** (jump if result is plus)
[PC] ← Label
The program jumps to instruction specified by label if result is plus

(vi) **JM addr (label).** (jump if result is minus)
[PC] ← Label
If the result is minus, the program jumps to instruction specified by label.

(vii) **JPE addr (label).** (jump if even parity)
[PC] ← Label, jump if even parity: the parity status P=1,
If the result contains even number of 1's, the program jumps to instruction specified by label.

(viii) **JPO addr (label).** (jump if odd parity)
[PC] ← Label, jump if odd parity: the parity
If the result contains odd number of 1's, the program jumps to instruction specified by label.

UNCONDITONAL CALL

CALL addr (label) : (Unconditional CALL: call the subroutine identified by address)

[[SP] – 1] ← [PCH], Save the address of next instruction of program in stack.
[[SP] – 2] ← [PCL],
[SP] ← [SP]-2
[PC] ← Addr (label)

CALL instruction is used to call a subroutine. Before control is transferred to subroutine, address of next instruction of main program is saved in stack. The content of stack

pointer is decremented by two to indicate new stack top. Then the program jumps to subroutine starting at address specified by label.

(Same syntax for unconditional and conditional CALL instructions)

CONDITIONAL CALL

CALL addr (label)

[[SP] – 1] ← [PCH], [[SP] ← [PCL],
[PC] ← Addr (label), [SP] ← [SP] – 2.

If the condition is true and program calls the specified subroutine, the execution of call instruction takes 5 machine cycles; 18 states. If condition is not true, only 2 machine cycles; 9 states are required for execution of instruction.

(i) CC addr (label) Call, if carry status CS=1.
(ii) CNC addr (label) Call, if carry status CS=0.
(iii) CZ addr (label) Call, if result is zero, zero status Z=1.
(iv) CNZ addr (label) Call, if result is non-zero, zero status Z=0.
(v) CP addr (label) Call, if result is plus, status S=0.
(vi) CM addr (label) Call, if result is minus, status S=1.
(vii) CPE addr (label) Call, if even parity, parity status P=1.
(viii) CPO addr (label) Call, if odd parity, parity status P=0.

UNCONDITIONAL RETURN

RET : (Return from subroutine when condition is satisfied)

[PCL] ← [[SP]],
[PCH] ← [[SP] + 1],
[SP] ← [SP] + 2.

RET instruction is used at end of subroutine. Before execution of subroutine address of next instruction of main program is saved in stack. The execution of RET instruction brings back the saved address from stack to program counter. The content of stack pointer is incremented by two to indicate new stack top. Then the program jumps to instruction of main program next to CALL instruction which called the subroutine.

(Same syntax is for conditional and unconditional RET instruction)

CONDITIONAL RETURN

RET (Return from subroutine when condition is satisfied)

$[PCL] \leftarrow [[SP]]$,
$[PCH] \leftarrow [[SP] +1]$,
$[SP] \leftarrow [SP] +2$.

If condition is true and program returns from subroutine, the execution of conditional return instruction takes 3 machine cycle, 12 states. If condition is not true only one machine cycle, 6 state are required.

(i)	RC	addr (label)	Return from subroutine, if carry status CS=1.
(ii)	RNC	addr (label)	Return from subroutine, if carry status CS=0.
(iii)	RZ	addr (label)	Return from subroutine, if result is zero, zero status Z=1.
(iv)	RNZ	addr (label)	Return from subroutine, if result is non-zero.
(v)	RP	addr (label)	Return from subroutine, if result is plus, status S=0.
(vi)	RM	addr (label)	Return from subroutine, if result is minus, status S=1.
(vii)	RPE	addr (label)	Return from subroutine, if even parity, parity status P=1.
(viii)	RPO	addr (label)	Return from subroutine, if odd parity, parity status P=0.

RST n (Restart)

$[[SP] - 1] \leftarrow [PCH]$, $[[SP] - 2] \leftarrow [PCL]$,
$[SP] \leftarrow [SP] - 2$, $[PC] \leftarrow$ 8 times n.

Restart is a one-word CALL instruction. The content of PC is saved in stack. The program jumps to instruction starting at restart location. The address of restart location is 8 times n.
The restart instruction and location are as follows:

Instruction	Opcode	Restart Locations
RST 0	C7	0000
RST 1	CF	0008
RST 2	D7	0010
RST 3	DF	0018
RST 4	E7	0020
RST 5	EF	0028
RST 6	F7	0030
RST 7	FF	0038

PCHL. (Jump to address specified by H-L pair)

[PC] ← [H-L], [PCH] ← [H], [PCL] ← [L]

The contents of H-L pair are transferred to PC. The contents of H are moved to high order 8-bits of PC. The contents of L are transferred to low order 8-bits of PC.

STACK, I/O AND MACHINE CONTROL GROUP (GROUP V)

1. IN port-address. (Input to accumulator from I/O port)

[A] ← [Port]
The data available on port is moved to accumulator. After IN, address of port is specified. The 2nd byte of instruction contains address of port. Address of port is an 8-bit address. For ex., IN 01. The address of port B of an I/O port 8255.1 of a microprocessor kit is 01.

2. OUT port-address. (Input to accumulator from I/O port).

[Port] ← [A]
The content of accumulator is moved to port specified by its address. After OUT, port address is specified. The 2nd byte of instruction contains address of port. Address of port is an 8-bit address. For ex., OUT 00. The address of port A of an I/O port 8255.1 of a microprocessor kit is 00.

3. PUSH rp. (Push content of register pair to stack)
[[SP] – 1] ← [rp],
[[SP] – 2] ← [rl],
[SP] ← [SP] – 2

The content of register pair rp is pushed into the stack.

4.PUSH PSW. (Push processor status word)
[[SP] – 1] ← [A],
[[SP] – 2] ← PSW (Processor Status Word),
[SP] ← [SP] – 2
The content of accumulator is pushed into the stack. The contents of status flags are also pushed into stack. The content of register SP is decremented by 2 to indicate new stack top.

5. POP rp. (Pop content of register pair, which was saved, from the stack)
[rl] ← [SP],
[rp] ← [[SP] + 1],
[SP] ← [SP] + 2

The content of register pair, which was saved earlier is moved from stack to register pair.

6. POP PSW. (Pop processor status word)
PSW ← [[SP]],
[A] ← [[SP] + 1],
[SP] ← [SP] + 2

The PSW, which was saved earlier during the execution of program is moved from stack PSW. The content of accumulator which was also saved is moved from stack to accumulator.

7. HLT (Halt)
The execution of instruction HLT stops the microprocessor. The registers and status flags remain unaffected.

8. XTHL. (Exchange stack-top with H-L)
[L] ↔ [[SP]]
[H] ↔ [[SP] +1]
The contents of register L are exchanged with byte of stack-top. The contents of H exchanged with byte below stack-top.

9. SPHL (Move the contents of H-L pair to stack pointer)
[H-L] → [SP]
The contents of H-L pair are transferred to SP register.

10. EI (Enable Interrupt)

When this instruction is executed the interrupts are enabled.

11. DI (Enable Interrupt)

When this instruction is executed the interrupts are disabled.

12. SIM (Set Interrupt Masks)

When this instruction is executed bits 0-5 of accumulator are used in programming the restart interrupt masks. Bits 6-7 of accumulator are used in making serial output SOD.

13. RIM (Read Interrupt Masks)

When this instruction is executed, accumulator is loaded with pending interrupts, the restart interrupt masks and contents of serial input SID.

14.NOP (No Operation)

No operation is performed when this instruction is executed. The registers and flags remain unaffected.

ASSEMBLY LANGUAGE

Writing of program in machine language is very difficult, tiresome, boring and error. Hence to facilitate programmers easily understandable languages have been developed. Assembly language is one of them. A programmer easily write a program in alphanumeric symbols instead of zeros and ones.

Examples are: ADD for addition, SUB for subtraction, CMP for comparison etc. Such symbols are called **Mnemonics.** A program written in mnemonics is known as **assembly language program.** The writing of a program in assembly language is much easier and faster as compared to the writing of a program in machine language.

Both assembly language and machine language are microprocessor specific. A microprocessor-specific language is known as a **low-level language.** When a program is written in a language other than machine language, a computer cannot understand.

A program which translates an assembly language program into a machine language program is called an **assembler.** A **self-assembler/resident assembler** is an assembler which runs on microcomputer for which it produces object codes. A **cross-assembler** is an assembler that runs on a computer other than that for which it produces object codes.

Disassembler is a software aid to convert a machine language program to an assembly language program. Its role is complementary to that of an assembler. It is useful in situations when a program is available in machine language and it is to be converted to assembly language for better understanding.

ONE-PASS AND TWO-PASS ASSEMBLER

An assembler which goes through an assembly language program only once is known as **one-pass assembler.the** Such an assembler must have some technique to take the forward references. The assembly labels that may appear later on in the program . Such labels are called as forward references.

The assembler which goes through an assembly language program twice is called as a two pass assembler. Such an assembler does not face difficulty with forward references. During the first pass it collects all labels. During second pass it produces the machine code for each instruction and assigns addresses to each of them. It assigns addresses to labels by counting their positions from the starting address.

One-pass assembler is faster as it goes through a program only once. Its disadvantage is that it does not provide many feature s as a two pass assembler can. The two pass assemblers are commonly used.

An assembly language program for addition of two numbers placed in two consecutive memory locations 2501H and 2502H is given below:

Example:

Mnemonics	operands	Comments
LXI	H, 2501H	Get the address of 1st number in H-L Pair
MOV	A, M	Get the 1st number in accumulator
INX	H	Increment content of H-L pair
ADD	M	Add the 1st and 2nd number
STA	2503H	Store sum in 2503
HLT		Stop

DATA
2501 – 49H
2502 – 56H
RESULT
2503 – 9F

The programmer first of all writes program in assembly language and then fills up machine code corresponding to each mnemonic. This is called **hand assembly.** The codes are written in hexadecimal system.

Advantages:

1. Computation time is less.
2. Faster to produce result.

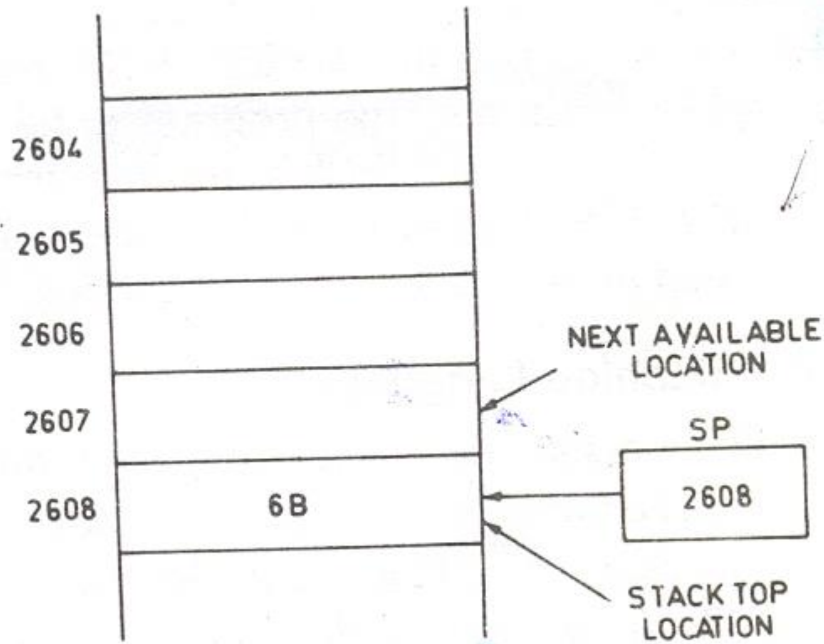
Disadvantages:

1. Programming is difficult and time consuming.
2. Not portable.
3. Each statement in high-level language corresponds to many assembly language instructions.

STACK

During execution of program sometimes it becomes necessary to save contents of certain registers because the registers are required for some other operation in subsequent steps. These contents are moved to certain memory locations by PUSH operation. Then these registers are used for other operations. After completing operations those contents which were saved in memory are transferred back to registers by POP operation. Memory locations for this purpose is set by the programmer at the beginning. The set of memory locations kept for this purpose is called **stack.** The last memory location of the occupied portion of the stack is called **stack-top.** A special 16-bit register known as **stack pointer** holds the address of stack-top. Any area of RAM can be used as stack. The stack pointer is initialized in the beginning of the program by LXI SP or SPHL instruction.

Data are stored in stack on **LIFO (Last-In-First-Out)** memory. Stack is faster than memory access. SP register holds the address of stack-top location. PUSH operation moved the contents of register to stack. POP operation is used to transfer the contents from stack to register. The following diagram shows the stack before PUSH and after PUSH operation. Also diagram shows before POP and after POP operation.



Stack and stacktop location.

The following diagram shows the stack before and after push operation.

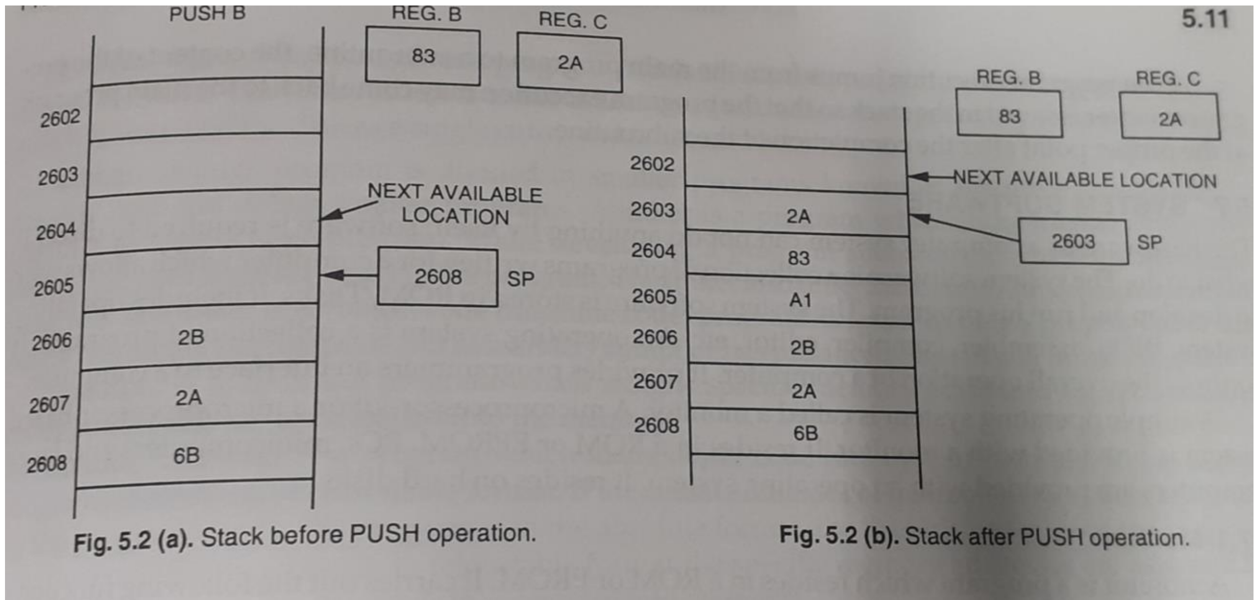


Fig. 5.2 (a). Stack before PUSH operation.

Fig. 5.2 (b). Stack after PUSH operation.

The following diagram shows the stack before and after pop operation

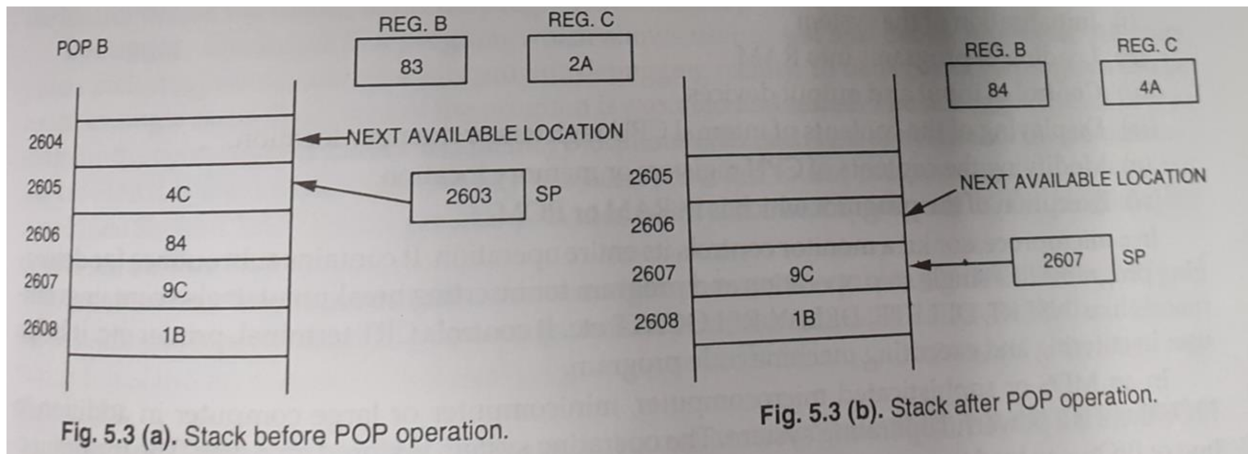


Fig. 5.3 (a). Stack before POP operation.

Fig. 5.3 (b). Stack after POP operation.

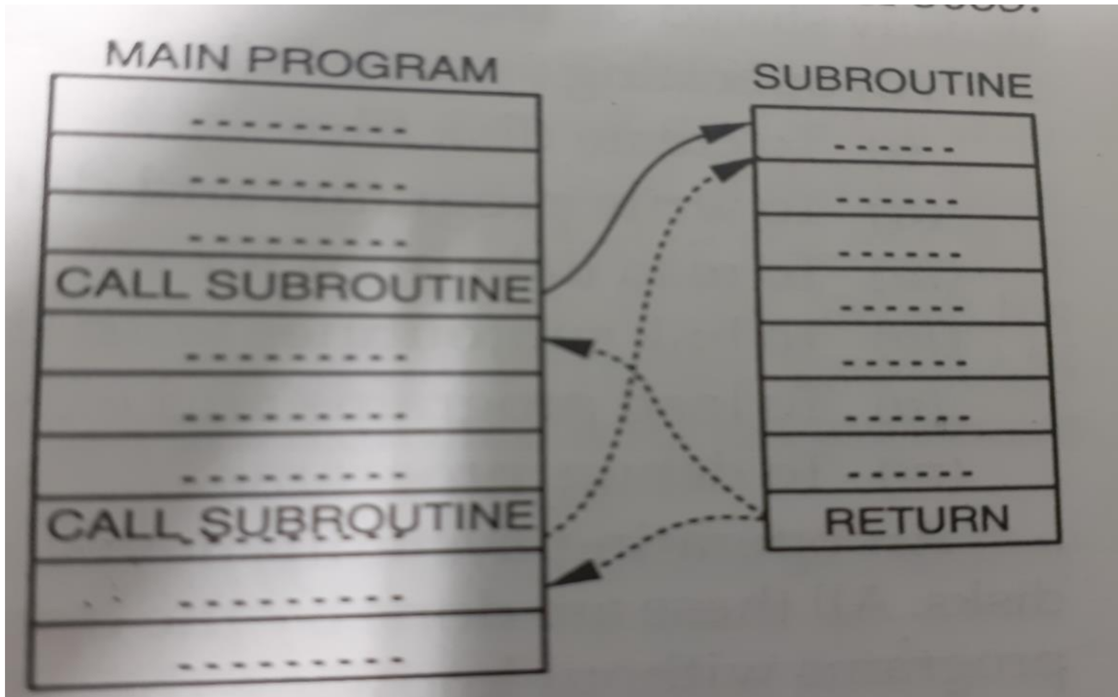
SUBROUTINES

While writing the program certain operations may occur several times and they are not available as individual instruction. The program for such operations are repeated again and again in main program. The simple example of such operation is multiplication. The Intel 8085 does not provide multiplication instruction.

The concept of subroutines is used to avoid the repetition of smaller programs. The small program for a particular task is called subroutine. Subroutines are written separately and stored in memory. They are called at various points of main program by CALL instruction where they are required. The last instruction of subroutine is RET. After completion of subroutine, main program begins from instruction immediately following the CALL instruction.

There may be more than one subroutine in a program. The technique of subroutine saves memory locations. The diagram shows CALL-RETURN structure.

When program execution jumps from the main program to a subroutine, the content of PC is saved in the stack so that the program execution may come back to main program at proper point after the completion of the subroutine. The diagram shows the CALL RETURN structure of subroutine.



MACROS

A sequence of instructions to which a name is assigned is called a **macro**. The name of the macro is used in assembly language programming. The macro facility is available with many assemblers.

Example 1

COMPLE	**MACRO**	**ADDRESS**
	LXI	H,ADDRESS
	MOV	A,M
	CMA	
	ENDM	

In above example COMPLE is the name of the macro. MACRO is written in the beginning of the definition Here ADDRESS is the parameter. ENDM is used to end of the macro. Suppose if we write COMPLE 2500H in an assembly language program , the assembler will replace this macro by the following sequence of instructions in the program

```
        LXI     H,2500H
        MOV     A,M
        CMA
```

Example 2

```
SHIFT   MACRO
        ADD A
        ENDM
```

SHIFT is name of macro. If SHIFT is written in program, assembler will replace it by ADD A. This instruction adds the content of accumulator to itself. This addition will result in logical shifting of content of accumulator left by one bit.

Once a sequence of instructions is written and macro name is assigned to it, macro name can be used frequently in program. It makes programs easier to read and understand. Macros and subroutines are similar. A subroutine requires CALL and RETURN instruction whereas macros do not. The macros execute faster than subroutines. Macros are used for short sequences of instructions whereas subroutines for longer ones, preferably for 10 instructions and more.