Unit - II

Instruction word size : (instruction format)

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

1) 1 - byte instruction
2) 2 - byte instruction
3) 3 - byte instruction

one-Byte instruction : Example of one-byte instructions are :

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

ADD B : Add the content of register B to the content of the accumulator. 80H is the opcode for the instruction ADD B. In this instruction one of the operands is register B (its content) which indicated in the instruction itself. In this type of instruction (arithmetic group of instruction) it is assumed that the other operand is in the accumulator. The opcode

80H in the binary form is 10000000. The first five bits i.e. 10000 specify the operation to be performed, i.e. ADD operation. The last three bits 000 are the code for register B for 8085 MP.

The above examples only one byte long. All one-byte instructions contain information regarding operands or in the opcode itself.

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

i) MVI B, 05 ; Move 05 to register B.
06, 05 ; MVI B, 05 in the code form
The 1st byte 06 is the opcode for MVI B and 2nd byte 05 is the data which is to be moved to register B.

ii) IN 01 ; Read data at port B
DB, 01 ; IN 01 in the code form
DB is the opcode for the instruction IN and 01 is the address of a port.
A two byte instruction is stored in two consecutive memory locations.

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

DATE : / /201
Page No. :

i) LXI H, 2400H; Load H-L pair with 2400H
21, 00, 24; LXIH, 2400H in the code form.
The 1st byte 21 is the opcode for the instruction LXI H. The 2nd byte 00 is LSBs of the data (2400H), which is loaded into register L. The 3rd byte 24 is 8 MSBs of the data (2400H), which is loaded into register H.

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

(4)

Addressing Modes :-

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

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

→ Direct Addressing

In this mode of addressing the address of the operand is given in the instruction itself. Examples are:

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

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

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

2) IN 02 Read data from the port C

DB, 02 Instruction in the code form.

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

accumulator. The 2nd byte of the instruction specifies the address of the port.

→ Register Addressing :-

In register addressing mode the operand is in one of the general purpose registers or accumulator. The opcode specifies the address of the register in addition to the operation to be performed.

Examples are :-

1) MOV A,B — Move the content of register B to register A.

78 — The instruction in the code form

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

→ Register Indirect Addressing :-

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

Example

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

In the above program the instruction MOV A,M

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

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

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

→ Immediate Addressing :-
In immediate addressing mode the operand is specified within the instruction itself, examples are

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

In these instructions the 2nd byte specifies data.

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

→ Implicit Addressing :- There are certain instructions which operate on the content of the accumulator. such instructions do not require the address of the operand. eg. CMA, RAL, RAR etc.

Classification of Instructions :-

An instruction is a command given to the computer to perform a specified operation on given data. The instruction set of a µP is the collection of the instructions that the µP is designed to execute. These instructions are of Intel corporation. They cannot be used by other µP manufacturers. The programmer can write a program in assembly language using these instructions.

These instructions have been classified into the following groups:-

1) Data Transfer group
2) Arithmetic group
3) Logical group
4) Branch control group
5) I/O and Machine control group.

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

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

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

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

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

4.6 INTEL 8085 INSTRUCTIONS

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

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

4.6.1 Data Transfer Group MOV r_1, r_2

(Move data; Move the content of the one register to another).

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

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

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

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

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

Example

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

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

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

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

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

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

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

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

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

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

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

Example

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

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

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

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

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

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

LDA addr. (Load Accumulator direct).

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

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

STA Addr. (Store accumulator direct).

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

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

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

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

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

SHLD addr. (Store H-L pair direct)

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

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

LDAX rp. (LOAD accumulator indirect)

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

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

STAX rp. (Store accumulator indirect)

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

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

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

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

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

4.6.2 Arithmetic Group

ADD r. (Add register to accumulator)

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

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

ADD M. (Add memory to accumulator)

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

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

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

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

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

ADC M. (Add memory with carry to accumulator)

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

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

ADI data. (Add immediate data to accumulator)

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

The immediate data is added to the content of the accumulator. The 1st byte of the instruction is its opcode. The 2nd byte of the instruction is data, and it is added to content of the accumulator. The sum is placed in the accumulator. For example, the instruction ADI 08 will add 08 to the content of the accumulator and place the result in the accumulator. In code form the instruction is written as C6, 08.

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

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

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

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

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

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

SUB r. (Subtract register from accumulator)

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

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

SUB M. (Subtract memory from accumulator).

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

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

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

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

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

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

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

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

SUI data. (Subtract immediate data from accumulator)

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

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

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

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

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

INR r. (Increment register content)

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

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

INR M. (Increment memory content)

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

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

DCR r. (Decrement register content)

[r] ← [r] – 1. States: 4. Flags: all except carry flag, Addressing: register. Machine cycles: 1.

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

DCR M. (Decrement memory content)

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

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

INX rp. (increment register pair)

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

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

DCX rp (Decrement register pair)

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

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

DAA. (Decimal adjust accumulator)

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

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

4.6.3 Logical Group

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

ANA r. (AND register with accumulator)

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

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

ANA M. (AND memory with accumulator)

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

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

ANI data. (AND immediate data with accumulator)

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

The 2nd byte of the instruction is data, and it is ANDed with the content of the accumulator. The result is placed in the accumulator. The CS flag is set to 0 and AC to 1.

ORA r. (OR register with accumulator)

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

The content of register *r* is ORed with the content of the accumulator. The result is placed in the accumulator. All status flags are affected. Carry and auxilary carry are cleared *i.e.* the CS and AC flags are set to 0.

ORA M. (OR memory with accumulator)

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

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

ORI data. (OR immediate data with accumulator)

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

The 2nd byte of the instruction is data, and it is ORed with the content of the accumulator. The result is placed in the accumulator. All status flags are affected. The CS and AC flags are set 0.

XRA r. (EXCLUSIVE - OR register with accumulator)

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

The content of register *r* is EXCLUSIVE - ORed with the content of the accumulator. The result is placed in the accumulator. All status flags are affected. The CS and AC flags are set to 0.

XRA M. (EXCLUSIVE - OR memory with accumulator)

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

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

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

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

The 2nd byte of the instruction is data, and it is EXCLUSIVE-ORed with the content of the accumulator. The result is placed in the accumulator. All flags are affected. The CS and AC flags are set to 0.

CMA. (Complement the accumulator)

[A] ← [$\overline{A}$]. States: 4. Flags: none. Machine cycles : 1. Addressing : implicit.

1's complement of the content of the accumulator is obtained, and the result is placed in the accumulator. To obtain the 1's complement of a binary number 0 is replaced by 1, and 1 by 0. For example, one's complement of 1100 is 0011.

CMC. (Complement the carry status)

[CS] ← [$\overline{CS}$]. States: 4. Flags: CS, Machine cycle: 1.

The CS flag is complemented. Other flags are not affected.

STC. (Set carry status)

[CS] ← 1. States: 4. Flags: CS. Machine cycles : 1.

The status flag CS is set to 1. Other flags are not affected.

CMP r. (Compare register with accumulator)

[A] − [r]. States 4. Flags: all. Addressing: register. Machine cycles: 1.

The content of register r is subtracted from the content of the accumulator and status flags are set according to the result of the subtraction. But the result is discarded. The content of the accumulator remains unchanged.

CMP M. (Compare memory with accumulator)

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

The content of the memory location addressed by H-L pair is subtracted from the content of the accumulator, and status flags are set according to the result of the subtraction. But the result is discarded. The content of the accumulator remains uncharged.

CPI data. (Compare immediate data with accumulator)

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

The 2nd byte of the instruction is data, and it is subtracted from the content of the accumulator. The status flags are set according to the result of subtraction. But the result is discarded. The content of the accumulator remains unchanged.

RLC. (Rotate accumulator left)

[A_{n+1}] ← [A_n], [A_0] ← [A_7], [CS] ← [A_7].

States: 4. Flags: CS. Machine cycles : 1. Addressing: implicit.

The content of the accumulator is rotated left by one bit. The seventh bit of the accumulator is moved to carry bit as well as to the zero bit of the accumulator. Only CS flag is affected. See Fig. 4.1

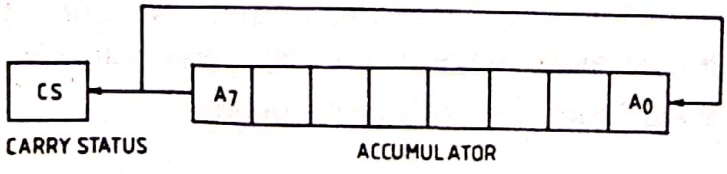

Fig. 4.1 Schematic Diagram for RLC.

RRC. (Rotate accumulator right)

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

States: 4. Flags: CS. Machine cycles : 1. Addressing: implicit.

The content of the accumulator is rotated right by one bit. The zero bit of the accumulator is moved to the seventh bit as well as to carry bit. Only CS flag is affected. See Fig. 4.2.

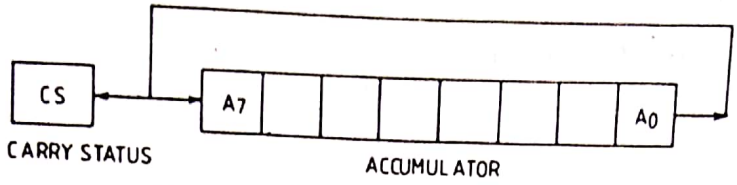

Fig. 4.2 Schematic Diagram for RRC.

RAL. (Rotate accumulator left through carry)

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

States: 4. Flags: CS. Machine cycles : 1. Addressing: implicit.

The content of the accumulator is rotated left one bit through carry. The seventh bit of the accumulator is moved to carry, and the carry bit is moved to the zero bit of the accumulator. Only carry flag is affected. See Fig. 4.3.

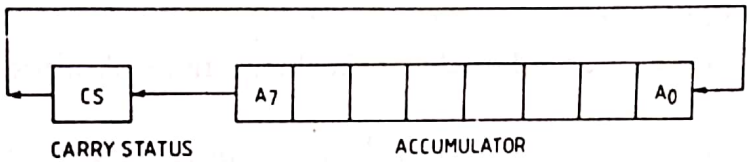

Fig. 4.3 Schematic Diagram for RAL.

RAR. (Rotate accumulator right through carry)

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

States : 4 Flags : CS. Machine cycle : 1. Addressing implicit.

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 to the seventh bit of the accumulator. Only CS flag is affected. See Fig. 4.4.

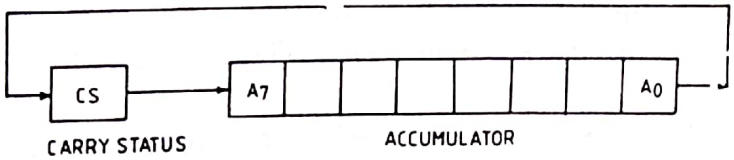

Fig. 4.4. Schematic Diagram for RAR.

4.6.4 Branch Group

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 the specified label when certain condition is statisfied. The unconditional branch instructions transfer the program to the specified label unconditionally.

JMP addr (label). (Unconditional jump: jump to the instruction specified by the address).

[PC] ← Label. States: 10. Flags: none. Addressing: immediate. Machine cycles: 3.

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

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

(*i*) **JZ addr (label).** (Jump if the result is zero)

[PC] ← address (label), jump if Z = 1. States: 7/10. Flags: none. Addressing: immediate. Machine cycles: 2/3.

The program jumps to the instruction specified by the address (label) if the result is zero (*i.e.* the zero status Z = 1). Here the result after the execution of the preceding instruction is under consideration.

(*ii*) **JNZ addr (label).** (Jump if the result is not zero)

[PC] ← address (label), jump if Z = 0. States: 7/10. Flags: none. Addressing : immediate. Machine cycles: 2/3.

The program jumps to the instruction specified by the address (label) if the result is non-zero (*i.e* the zero status Z = 0).

(*iii*) **JC addr (label).** (Jump if there is a carry)

[PC] ← address (label), jump if CS = 1. States: 7/10. Flags: none. Addressing : immediate. Machine cycles: 2/3.

The program jumps to the instruction specified by the address (label) if there is a carry (*i.e.* the carry status: CS = 1). Here the carry after the execution of the preceding instruction is under consideration.

(*iv*) **JNC addr (label).** (Jump if there is no carry)

[PC] ← address (label), jump if CS = 0. States: 7/10. Flags: none. Addressing : immediate. Machine cycles: 2/3.

The program jumps to the instruction specified by the address (label) if there is no carry (*i.e.* the carry states CS = 0).

(*v*) **JP addr (label).** (Jump if the result is plus)

[PC] ← address (label), jump if S = 0. States: 7/10. Flags: none. Addressing: immediate. Machine cycles: 2/3.

The program jumps to the instruction specified by the address (label) if the result is plus.

(*vi*) **JM addr (label).** (Jump if the result is minus)

[PC] ← address (label), jump if S = 1. States: 7/10. Flags: none. Addressing: immediate. Machine cycles: 2/3.

If the result is minus the program jumps to the instruction specified by the address (label).

(*vii*) **JPE addr (label).** (Jump if even parity)

[PC] ← address (label), jump if even parity: the parity status P = 1, States: 7/10. Flags: none. Addressing: immediate. Machine cycles: 2/3.

If the result contains even number of 1s, the program jumps to the instruction specified by the address (label).

(*viii*) **JPO addr (label).** (Jump if odd parity)

[PC] ← address (label), jump if odd parity ; the parity status P = 0, States: 7/10, Flags: none, Addressing: immediate, Machine cycles: 2/3.

If the result contains odd number of 1s, the program jumps to the instruction specified by the address (label).

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

[[SP] – 1] ← [PCH], Save the address of the next instruction of the program in the stack.

[[SP] – 2] ← [PCL],

[SP] ← [SP] – 2

[PC] ← addr (label)

States: 18. Flags: none, Addressing: immediate/reg. indirect. Machine cycles: 5.

CALL instruction is used to call a subroutine. Before the control is transferred to the subroutine, the address of the next instruction of the main program is saved in the stack. The content of the stack pointer is decremented by two to indicate the new stacktop. Then the program jumps to subroutine starting at address specified by the label.

Conditional CALL addr (label)

[[SP] – 1] ← [PCH], [[SP] – 2] ← [PCL],

[PC] ← addr (label), [SP] ← [SP] – 2.

States: 9/18. Flags: none. Addressing: immediate/reg. indirect. Machine cycles: 2/5. If the condition is true and program calls the specified subroutine, the execution of a conditional call instruction takes 5 machine cycles; 18 states. If condition is not true, only 2 machine cycles; 9 states are required for the execution of the instruction.

(*i*)	CC	addr (label)	Call subroutine if carry status CS = 1.
(*ii*)	CNC	addr (label)	Call subroutine if carry status CS = 0.
(*iii*)	CZ	addr (label)	Call subroutine if the result is zero; the zero status Z = 1.
(*iv*)	CNZ	addr (label)	Call subroutine if the result is not zero; the zero status Z = 0.
(*v*)	CP	addr (label)	Call subroutine if the result is plus; the sign status S = 0.
(*vi*)	CM	addr (label)	Call subroutine if the result is minus, the sign status S = 1.
(*vii*)	CPE	addr (label)	Call subroutine if even parity; the parity status P = 1.
(*viii*)	CPO	addr (label)	Call subroutine if odd parity; the parity status P = 0.

RET. (Return from subroutine)

[PCL] ← [[SP]],

[PCH]← [[SP] + 1],

[SP] ← [SP] + 2.

States: 10. Flags: none. Addressing: reg. indirect. Machine cycles: 3.

RET instruction is used at the end of a subroutine. Before the execution of a subroutine the address of the next instruction of the main program is saved in the stack. The execution of RET instruction brings back the saved address from the stack to the program counter. The content of the stack pointer is incremented by 2 to indicate the new stack top. Then the program jumps to the instruction of the main program next to CALL instruction which called the subroutine.

Conditional Return

[PCL] ← [[SP]], [PCH] ← [[SP] + 1],

[SP] ← [SP] + 2.

States: 6/12. Flags: none. Addressing: reg. indirect. Machine cycles: 1/3. If the condition is true and the program returns from the subroutine, the execution of a conditional return instruction takes 3 machine cycles, 12 states. If condition is not true only one machine cycle, 6 states are required.

(*i*) RC Return from subroutine if carry status CS = 1.
(*ii*) RNC Return from subroutine if carry status CS = 0.
(*iii*) RZ Return from subroutine if the result is zero; the zero status Z = 1.
(*iv*) RNZ Return from subroutine if the result is not zero; the zero status Z = 0.
(*v*) RP Return from subroutine if the result is plus; the sign status S = 0.
(*vi*) RM Return from subroutine if the result is minus, the sign status S = 1.
(*vii*) RPE Return from subroutine if even parity, the parity status P = 1.
(*viii*) RPO Return from subroutine if odd parity, the parity status P = 0.

RST n (Restart).
[[SP] – 1] ← [PCH], [[SP] – 2] ← [PCL],
[SP] ← [SP] – 2, [PC] ←8 times *n*.
States: 12. Flags: none, Addressing: reg. indirect. Machine cycles : 3.

Restart is a one-word CALL instruction. The content of the program counter is saved in the stack. The program jumps to the instruction starting at restart location. The address of the restart location is 8 times *n*. The restart instruction and locations 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]
States: 6. Flags: none. Addressing: register. Machine cycle : 1.

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

4.6.5 Stack, I/O and Machine Control Group

IN port-address. (Input to accumulator from I/O port)
[A] ← [Port]. States: 10. Flags: none. Addressing: direct. Machine cycles: 3.

The data available on the port is moved to the accumulator. After instruction IN, the address of the port is specified. The 2nd byte of the instruction contains the address of the port. The address of a port is an 8-bit address. For example, IN 01. The address of the port B of an I/O port 8255.1 of a microprocessor kit is 01.

OUT port-address. (Output from accumulator to I/O port)
[Port] ← [A]. States: 10. Flags: none. Addressing: direct. Machine cycles : 3.

The content of the accumulator is moved to the port specified by its address. After the OUT instruction, the port address is specified. The 2nd byte of the instruction contains the address of the

port. For example, OUT 00. The address of the port A of an I/O port 8255.1 of a microprocessor kit is 00.

PUSH rp. (Push the content of register pair to stack)

[[SP] − 1] ← [*rh*],

[[SP] − 2] ← [*rl*],

[SP] ← [SP] − 2.

States: 12. Flags: none. Addressing: register(source)/reg. indirect(destination), Machine cycles: 3.

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

PUSH PSW. (PUSH processor status word)

[[SP] − 1] ← [A]

[[SP] − 2] ← PSW (Program Status Word)

[SP] ← [SP] − 2.

States: 12, Flags: none. Addressing: register(source)/reg.indirect(destination), Machine cycles: 3.

The content of the accumulator is pushed into the stack. The contents of status flags are also pushed into the stack. The content of the register SP is decremented by 2 to indicate new stacktop.

POP rp. (Copy two bytes from the top of the stack into the specified register)

[*rl*] ← [[SP]]

[*rh*] ← [[SP] + 1]

[SP] ← [SP] + 2.

States: 10. Flags: none. Addressing: register(destination)/reg.indirect (source),Machine cycles: 3.

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

POP PSW. (Copy two bytes from the top of the stack into PSW and Accumulator)

PSW ← [[SP]]

[A] ← [[SP] + 1]

[SP] ← [SP] + 2.

States: 10. Flags: all. Addressing: reg. indirect. Machine cycles: 3.

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

HLT (Halt)

States: 5. Flags: none. Machine cycle : 1.

When this instruction is executed, any further program execution is stopped. The microprocessor remains in Halt state. An interrupt or reset is necessary to exit from the Halt state.

XTHL. (Exchange stack-top with H-L)

[L] ↔ [[SP]]

[H] ↔ [[SP]] + 1].

States: 16. Flags: none. Addressing: register indirect. Machine cycles: 5.

The contents of the register L are exchanged with the byte of the stack-top. The contents of the H register exchanged with the byte below the stack-top.

SPHL (Move the contents of H-L pair to stack pointer)

[H-L] → [SP].

States: 6. Flags: none. Addressing: register. Machine cycle: 1.

The contents of H-L pair are transferred to the SP register.

EI. (Enable interrupts)

States: 4. Flags: none, Machine cycle: 1.

When this instruction is executed the interrupts are enabled.

DI (Disable Interrups)

States: 4. Flags : none, Machine cycle: 1

When this instruction is executed interrups are disabled.

SIM (Set Interupt Masks)

States: 4. Flags: none, Machine cycle: 1.

When this instruction is executed bits 0-5 of the accumulator are used in programming the restart interrupt masks. Bits 6-7 of the accumulator are used in making serial output on SOD line. See details in Chapter 7, Section 7.5: Interrupts of Intel 8085.

RIM (Read Interrupt Mask)

States: 4. Flags: none. Machine cycle: 1.

When this instruction is executed, the accumulator is loaded with pending interrupts, the restart interrupt masks and the contents of SID. See details in Chapter 7, Section 7.5: Interrupts of Intel 8085.

NOP (No Operation)

States: 4. Flags: none. Machine Cycle: 1.

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

PROBLEMS

1. Classify 8085 instructions in various groups. Give examples of instructions for each group.
2. What are the various types of data formats for Intel 8085 instructions? Give examples for each type of data format.
3. Discuss various types of addressing modes of Intel 8085 with suitable examples.
4. Explain what operation will take place when the following instructions are executed: LXI rp, data; LDA addr, LHLD addr, STA addr, and SHLD addr.
5. Explain what operation is performed when the following instructions are executed: DAD rp, DAA, CMP r, CMP M, CMA, RAL, RAR, PUSH rp and POP rp.