UNIT 4

Assembly Language Programming

Programming with Intel 8085/86 Microprocessor

Programming with Intel 8085 microprocessor

Instruction Format:

On the basis of size they occupy, the instruction formats are:

- **One byte instruction:**

✓It includes the op-code and operand in the same byte (all register transfer). E.g. MOV A, B

- **Two byte instruction:**

✓The first byte specifies op-code and second byte specifies the operand. E.g. MVI A, 32H

- **Three byte instruction:**

✓The first byte specifies op-code and following two byte specifies 16-bit operand. E.g. LXI B, 2032H

Addressing Modes

- Each instruction performs an operation on the specified data.
- An operand must be specified for an instruction to be executed. The operand may be in the general purpose registers, accumulator or a memory location.
- The method in which the operand is specified in an instruction is called addressing mode.
- To perform any corresponding instructions to the microprocessor, In each instruction, programmer has to specify 3 things:
  - ✓ Operation to be performed.
  - ✓ Address of source of data.
  - ✓ Address of destination of result.

Types

- **Intel 8085 uses the following addressing modes:**
  1. *Direct Addressing Mode*
  2. *Register Addressing Mode*
  3. *Register Indirect Addressing Mode*
  4. *Immediate Addressing Mode*
  5. *Implicit Addressing Mode*

Direct Addressing:

- The effective address of the operand is specified directly in the instruction, 'or' Address of the operand is given in the instruction itself.
- Instructions using this mode may contain 2 or 3 bytes, with first byte as the Op-code followed by 1 or 2 bytes of address of data.
- Loading and storing in memory use 2 bytes of address while IN and OUT have one byte address.
- For example:
  - LDA 2035H (A←M [2035H], i.e. Load the memory content to the accumulator)
  - STA 2500H (M [2500H]←A)
  - IN 07H (A←port address 07H)

Register Addressing:

- register is the source of an operand for an instruction.
- similar to direct addressing.
- For example:
- MOV A, B (A←B) //MOV is the operation.
  //B is the source of data.
  //A is the destination.
- ADD B (A←A+B)

Register Indirect Addressing:

- The address of the operand is specified by register pair.
  - LDAX B (if B=23 and C=50 then A←M [2350H])
  - STAX D (if D=30 and E=10 then M [3010H]←A)
  - MOV A, M (M=HL; if H=68 and L=32, then A←M [6832H)

//Mov is the operation.

//M is the memory location specified by H-L register pair.

//A is the destination.

Immediate Addressing:

- simplest way of addressing.
- operate on immediate hexadecimal number.
- operand is present in instruction in this mode.
- used to define and use constants of set initial values of variables.
- operand may be 8 bit data or 16 bit data.
- For example:
- MVI B, 05H (B←05H) //MVI is the operation.
  //05 H is the immediate data (source).
  //A is the destination.

  - LXI B, 7A21H (B←7A and C←21)
  - ADI 72H (A←A+72H)

Implied or Implicit or Inherent Addressing:

- The instructions of this mode do not have operands.
- For example:

  EI (Enable Interrupt),

  STC (set the carry flag),

  NOP (No operation),

  CMA (complement A)

  //CMA is the operation.

  //A is the source.

  //A is the destination.

Instruction Types:

According to their functions, 8085 instructions can be classified as:

1) Data transfer
2) Arithmetic
3) Logical
4) Branching
5) Miscellaneous

1. Data transfer:

i. MOV

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

ii. MVI

Syntax: MVI Rd/M, 8-bit data

iii. LXI

Syntax: LXI Reg. pair, 16-bit data
e.g. LXI B, 8085

iv. LDA

Syntax: LDA 16-bit address
e.g. LDA 8085

A←M[8085]

v. STA

Syntax: STA 16-bit address
e.g. STA 8085

M [8085]←A

vi. LDAX

Syntax: LDAX B/D register pair

e.g. LDAX B

A←M [8050] (if B=80, C=50)

vii. STAX

Syntax: STAX B/D register pair
e.g. STAX B
M [8050]←A (if B=80, C=50)

viii. LHLD

Syntax: LHLD 16-bit address e.g. LHLD 8085

L←M [8085] H←M [8086]

ix. SHLD

Syntax: SHLD 16-bit address

e.g. SHLD 8085

M [8085]←L M [8086]←H

x. XCHG

Syntax: XCHG ; interchanges between D and H; and E and L

H←D and L←E
D←H and E←L

xi. IN

Syntax: IN 8-bit port address

e.g. IN 15; data received from port address 15 is transferred into Accumulator.

xii. OUT

Syntax: OUT 8-bit port address

e.g. OUT 15 ; content of Accumulator is transferred to port address 15.

2. Arithmetic Instructions

i) **ADD**

Syntax ADD R/M

e.g. ADD B ; A←A+B

ii) **ADC**

Syntax: ADC R/M

e.g. ADC B ; A←A+B+CY

iii) **ADI**

Syntax: ADI 8-bit data

e.g. ADI 45 ; A←A+45

iv) **ACI**

Syntax: ACI 8-bit data

e.g. ACI 45 ; A←A+45+CY

v) **SUB**

Syntax: SUB R/M

e.g. SUB B ; A←A-B

vi) **SBB**

Syntax: SBB R/M

e.g. SBB B ; A←A-B-CY

vii) **SUI**

Syntax: SUI 8-bit data

e.g. SUI 45 ; A←A-45

viii) **SBI**

Syntax: SBI 8-bit data

e.g. SBI 45; A←A-45-CY

ix) **DAD**

Syntax: DAD register pair; contents of register pair is added to HL and the sum is saved in HL pair.

e.g. DAD B ; HL←HL+BC

x) **INR**

Syntax: INR R/M

e.g. INR B ; B←B-1

xi) **DCR**

Syntax: DCR R/M

e.g. DCR C ; C←C-1

xii) **INX**

Syntax: INX register pair

e.g. INX H ; HL←HL+1

xiii) **DCX**

Syntax: DCX register pair

e.g. DCX B ; BC←BC-1

3) Logical instructions

i) **ANA**

Syntax: ANA reg./M

e.g. ANA B ; A←A ∧ B

ii) **ANI**

Syntax: ANI 8-bit data

e.g. ANI 23H ; A←A ∧ 23H

iii) **ORA**

Syntax: ORA Reg./M

e.g. ORA B ; A←A ∨ B

iv) **ORI**

Syntax: ORI 8-bit data

e.g. ORI 23H ; A←A ∨ 23H

v) **XRA**

Syntax: XRA Reg./M

e.g. XRA B ; A←A ⊕ B

vi) **XRI**

Syntax: XRI 8-bit data

e.g. XRI 23H ; A←A ⊕ 23H

vii) **CMA**

e.g. CMA ; complements the content of accumulator

viii) **CMP**

Syntax: CMP Reg./M

The contents of the operand (register/memory) are compared with the content of accumulator and both contents are preserved and the comparison is shown by setting the flags.

If A>Reg./M, CY=0, Z=0

If A=Reg./M, CY=0, Z=1

If A<Reg./M, CY=1, Z=0

ix) **CPI**

Syntax: CPI 8-bit data

If A>data, CY=0, Z=0

If A=data, CY=0, Z=1

If A<data, CY=1, Z=0

x) **RLC**

Each binary bit of the accumulator is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the carry flag (CY). Other flags are not affected.

Syntax: RLC

Before RLC: 10100111, CY=0

After RLC: 01001111; CY=1

xi) **RRC**

Syntax:

RRC

Before RLC: 10100111, CY=0

After RLC: 11010011; CY=1

xii) **RAL**

Each binary bit of the accumulator is rotated left by one position through the carry flag (CY). Other flags are not affected.

Syntax:

RAL

Before RAL: 10100111, CY=0

After RAL: 01001110; CY=1

xiii) **RAR**

Each binary bit of the accumulator is rotated right by one position through the carry flag. Other flags are not affected.

Syntax:

RAL

Before RAL: 10100111, CY=0

After RAL: 01010011; CY=1

xiv) **DAA (Decimal Adjust Accumulator)**

The contents of accumulator are changed from binary value to two binary-coded decimal (BCD) digits. This is the only instruction that uses the AC flag to perform binary to BCD conversion. All flags are affected.

4) Branching instructions

a) Jump instructions:

i) Unconditional jump:

The program sequence is transferred to the memory location specified by the 16-bit address without checking any condition.

Syntax: JMP 16-bit address e.g. JMP 8085

Conditional jump:

Op-code	Description	Flag status
JC	Jump on carry	CY=1
JNC	Jump on no carry	CY=0
JP	Jump on positive	S=0
JM	Jump on minus	S=1
JPE	Jump on even parity	P=1
JPO	Jump on odd parity	P=0
JZ	Jump on zero	Z=1
JNZ	Jump on non-zero	Z=0

b) Call instructions / Return instructions:

i) Unconditional call/return:

Syntax: CALL 16-bit address
:
:
: RET

ii) Conditional Call/return:

Op-code	Description	Flag status
CC/RC	Call/Return on carry	CY=1
CNC/RNC	Call/Return on no carry	CY=0
CP/RP	Call/Return on positive	S=0
CM/RM	Call/Return on minus	S=1
CPE/RPE	Call/Return on even parity	P=1
CPO/RPO	Call/Return on odd parity	P=0
CZ/RZ	Call/Return on zero	Z=1
CNZ/RNZ	Call/Return on non-zero	Z=0

c) **Restart instructions**

The RST instructions are equivalent to 1-byte call instructions to one of the eight memory locations on page 0. The instructions are generally used in conjunction with interrupts and inserted using external hardware. However, these can be used as software instructions in a program to transfer program execution to one of the eight locations.

Op-code	Restart Address (H)
RST 0	0000
RST 1	0008
RST 2	0010
RST 3	0018
RST 4	0020
RST 5	0028
RST 6	0030
RST 7	0038

MISCELLANEOUS GROUP INSTRUCTIONS:

1. **PUSH:**
   - Syntax: PUSH reg. pair e.g. PUSH B
2. **POP:**
   - Syntax: POP reg. pair e.g. POP H
3. **PCHL:**
   - The contents of registers H and L are copied to the program counter.
4. **SPHL:**
   - The contents of registers H and L are copied to the stack pointer register.
5. **XTHL:**
   - Exchange H and L with Top of Stack

6. HLT:

- Halt and enter wait state.
- The contents of the registers are unaffected during the HLT state.

7. NOP:

- No operation is performed.
- The instruction is fetched and decoded; however, no operation is executed.
- The instruction is used to fill in time delays or to delete and insert instructions while troubleshooting.

8. RIM:

- Read Interrupt Mask.
- Multipurpose instruction used to read the status of interrupts 7.5, 6.5, 5.5 and to read serial data input bit.
- Loads 8 bits in the accumulator with the following interpretations:

D7	D6	D5	D4	D3	D2	D1	D0
SID	**I7**	**I6**	**I5**	**IE**	**RST7.5**	**RST6.5**	**RST5.5**

- D7=serial input data bit
- D6, D5, D4=interrupts pending if bit=1
- D3=interrupt enable; flip-flop is set if bit=1
- D2, D1, D0=interrupt masked if bit=1

  e.g. After the execution of instruction RIM, the accumulator contained 49H. i.e.,

  (A): 49H = 0 1 0 0 1 0 0 1

- Interrupt is pending.
- Interrupt enable flip-flop is set.
- RST 7.5 and 6.5 are enabled. RST 5.5 masked.

9. SIM (Set Interrupt Mask):

- Multipurpose instruction and used to implement the 8085 interrupts (RST 7.5, 6.5, and 5.5) and serial data output.
- The instruction interprets the accumulator contents as follows:

D7	D6	D5	D4	D3	D2	D1	D0
SOD	**SDE**	**Xxx**	**R7.5**	**MSE**	**M7.5**	**M6.5**	**M5.5**

- D7=serial output data
- D6=serial data enable (1=enable and 0=disable)
- If D4=1, reset RST 7.5 flip-flop
- If D3=1, mask set enable
- D2, D1, D0=masks interrupts, if bits=1

- **SOD** – Serial Output Data: Bit D7 of the accumulator is latched into the SOD output line and made available to a serial peripheral if bit D6=1.
- **SDE** – Serial Data Enable: If this bit = 1, it enables the serial output. To implement serial output, this bit needs to be enabled.
- **XXX** – Don't care.
- **R7.5** – Reset RST 7.5: If this bit = 1, RST 7.5 flip-flop is reset. This is an additional control to reset RST 7.5.
- **MSE** – Mask Set Enable: If this bit is high, it enables the functions of bits D2, D1, D0.This is a master control over all the interrupt masking bits. If this bit is low, bits D2, D1, and D0 do not have any effect on the masks.
- **M7.5** – D2 = 0, RST 7.5 is enabled.
  = 1, RST 7.5 is masked or disabled.
- **M6.5** – D1 = 0, RST 6.5 is enabled.
  = 1, RST 6.5 is masked or disabled.
- **M5.5** – D0 = 0, RST 5.5 is enabled.
  = 1, RST 5.5 is masked or disabled.

TIME DELAY & COUNTERS

- **Counter:**
  - designed simply by loading an appropriate number into one of the registers.
  - Use of the INR or DCR instructions.
  - A loop is established to update a count, and each count is checked to determine whether it has reached the final number, if not the loop is repeated.
- **Time Delay:**
  - The loop causes the delay.
  - Depending upon the clock period of the system, the time delay occurred during looping.
  - The instructions within the loop use their own T-states so they need certain time to execute resulting delay.

- Suppose we have an 8085 microprocessor with 2MHZ clock frequency.
- Let us use the instruction MVI which takes 7 T-states.
- Clock frequency of system (f) = 2 MHz
- Clock period (T) = 1/f= 1/ ($2*10^{-6}$) = 0.5μs
- Time to execute MVI = 7 T-states * 0.5μs = 305μs
  - E.g.

	MVI C, FFH	7
LOOP:	DCR C	4
	JNZ LOOP	10/7

- Here register C is loaded with count FFH (255_{10}) by using MVI which takes 7 T-states.
- Next 2 instructions DCR and JNZ form a loop with a total of 14 (4+10) T-states.
- The loop is repeated 255 times until C=0.
- The time delay in loop (TC) with 2 MHz frequency is

  T1= (T * loop T- states * count)
- Where, Tl = time delay in loop

  T= system clock period=0.5μS

  Count = decimal value for counter
- Tl= ($0.5 * 10^{-6} * 14 * 255$) s

  = 1785 μs

- But JNZ takes only 7 T-states when exited from loop i. e. last count = 0.50 adjusted loop delay

$$Tl_{adj} = Tl\text{-}(3\text{ T-states clock period}) = 1785\mu s - 1.5\mu s$$
$$= 1783.5\mu s$$

- Total delay loop of program,

$$TD = \text{Time to execute outside code} + Tl_{adj} \text{ inside Loop}$$
$$= 7 * 0.5\mu s + 1783.5\mu s$$
$$= 1787\mu s \approx 1.8 \text{ ms}$$

- To increase the time delay beyond 1.8 ms for 2MHZ microprocessor, we need to use counter for register pair or loop within a loop.