LAB MICROPROCESSOR

The internal structure of 8086 is divided into two functional unit. - BIU (Bus Interface Unit)
Execution unit

Function of BIU :-

i) To fetch the instruction or data from memory.
ii) Write the data to memory.
iii) Write data to I/O ports.
iv) Read data from the ports.

It has three functional parts.
1) Segment registers
2) Instruction pointer (IP)
3) Instruction queue.
↳ It stores the 8 bit of the instruction

There are four segment register :-
1) CS (code segment). 2) DS (Data Segment)
3) SS (Stack segment) 4) ES (Extra Segment)

1) CS → It is used for addressing a memory location in the code segment of the memory, where the executable program is stored.

2) DS → It consists of data used by the program and is accessed in the data segment by an offset address or the content of other register that holds the offset address.

3) SS → It handles the memory to store data and addresses during execution.

4) ES → It is additional data segment, which is used by the string to hold the extra destination data.

- PA = Seg X 10H + offset
  - (Physical address)

Ex The value of 4 segment register is 4042H and the value of IP register is 0580H. Find Physical address.

Sol
$$PA = (4042H \times 10H) + 0580H$$
$$= 40420H + 0580H = 409A0H.$$

40420
+ 0580
409A0

Ex The value of 4 Segment = 3860H
IP = 1735H (offset)

(In Hexadecimal after sum ≥ 16, carry will take)

$$PA = (3860H \times 10H) + 1735H$$
$$= 38600H + 1735H = 39B35H.$$

Ex The value of 4segment = 1005H and IP = 5555H

$$PA = (1005H \times 10H) + 5555H$$
$$= 10050H + 5555H = 155A5H.$$

1005 → 0001 0000 0000 0101
× 10 → 1010 0000
10050 → 0001 0000 0000 0101 0000
+ 5555 → + 0101 0101 0101 0101
155A5 → 0001 0101 0101 1010 0101

(EU) Execution Unit –
1) To instruct the Bus Interface Unit where to fetch the instruction & data from.
2) To decode the instructions.
3) To execute the instructions.

(iv) Execution unit contains the control circuitry to perform various internal operations.

Functional parts are
(i) General purpose registers
(ii) pointer and index register
(iii) ALU.
(iv) Flag registers
(v) Timing and control units

Flag Register - 16 bit register that behaves like a flip flop.
- It changes its status according to the result stored in the accumulator.
- It has 9 flags and divided into 2 groups - Conditional flags and Control flags.

Segment Registers	Offset Registers	Functions
CS (code)	IP	Address of the next instruction
DS (Data) (by default)	BX, DI, SI (stave pointer)	Address of data
SS (stack)	SP, BP (base pointer)	Address of the stack
ES (extra)	BX, DI, SI	Address of the destination data (for string operation)

Program - A Sequence of instructions

Instruction - A command in a program.

Assembly language - The language in which we write program in 8086.

Machine language Code - A program written in machine language.

→ Machine code will specify -
(i) What operation to be performed
(ii) What opcode is to be used
(iii) Whether operation is of 16 bit or 8 bit.
(iv) Whether operands are stored in registers or memory location.

→ Machine code of each instruction vary interms of no. of bytes.

Ex: ADD AX, BX
- AX → destination operand
- BX → Source operand
- AX, BX → operand
- ADD → opcode → OR Mnemonic

Addressing Modes

- It defines the way in which the operand of an instruction is specified.

Types :
(i) Register Addressing mode
(ii) Immediate Addressing mode
(iii) Memory Addressing mode
  - (a) Direct Addressing mode
  - (b) Indirect Addressing mode
  - (c) Indexed Addressing mode
  - (d) Base Addressing mode

↳ (c) Base Index Addressing mode

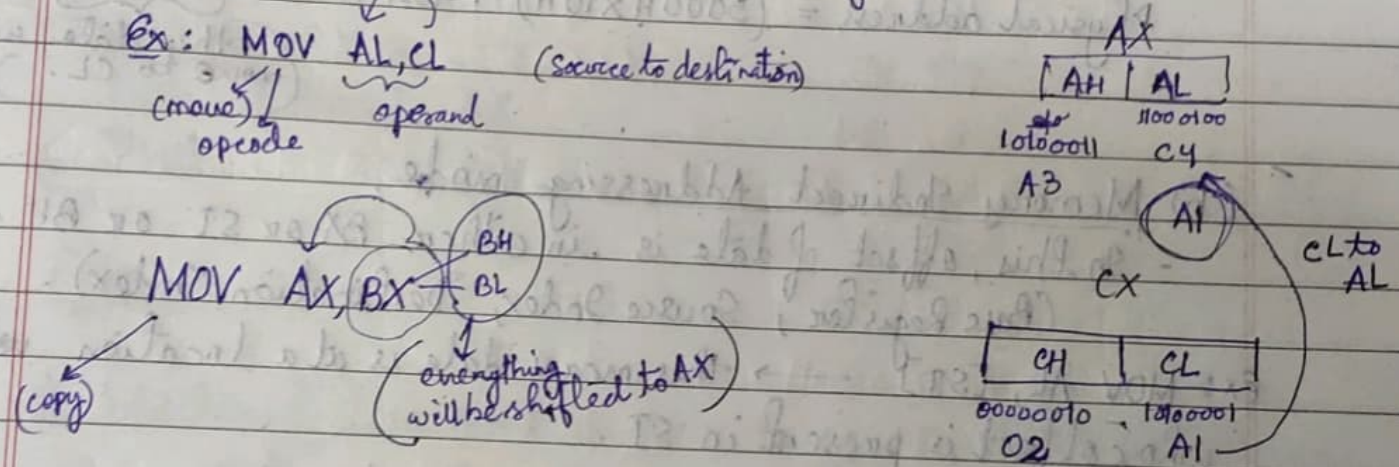
(iv) Relative Addressing mode
- ↳ (a) Relative Base Addressing mode
- ↳ (b) Relative Index Addressing mode
- ↳ (c) Relative Base Index Addressing mode

(v) Implied / Implicit Addressing mode

① Register Addressing Mode

- Here, data is stored in registers.
- Here, both the operands are registers.

Ex: MOV AL, CL (source to destination)
(move) opcode, AL, CL operand



② Immediate Addressing Mode

- Here, the source operand is a 8 bit or 16 bit data.
- Destination operand can never be an immediate data.

Ex: MOV AX, 2000H (✓) (MOV 3210H, AX ✗)
AX → location, 2000H → data

```
MOV AX, 0001H
MOV BX, 0002H
ADD BX, AX
```

③ (a) Memory Direct Addressing mode

- Here, a 16 bit address (offset) is directly specified in the instruction as a part of it.

Ex: MOV CL, [4321H].

↳ This means data is at a location whose offset is 4321 H.

Let us assume DS = 5000H. and its offset is 4321 H.

Physical address = (5000H × 10H) + 4321 H = 54321 H

(this data will move to CL.)

(b) Memory Indirect Addressing mode

- In this, offset of date is in either BX or SI or BH.
  (Base Register, Source Index, Destination Index).

Ex: MOV AL, [SI] ⟶ this means data is at a location whose offset is present in SI.

(c) Memory Base Addressing mode

- Here, the offset of the memory may be directly taken from one of the base registers. (BX/BP) specified by instructions.

Ex: MOV DL, [BX]

(d) Memory Indexed Addressing mode

- Here, the offset address may be directly taken from index registers.

If the register is SI then DX is the segment register.

and if the register is DI then EX is the segment register.

Ex: MOV EL [DI] or MOV DL [SI]

(e) Memory Base Index Addressing mode

-Here, the offset is the sum of base and index register.

Exp-1

Ex: MOV CH, [BX+SI].

Obj-1

Addition of two 16 bit numbers using immediate addressing mode:

```
MOV AX, 1234 H
MOV BX, 4321 H
ADD AX, BX
MOV [2,000H], AX
HLT
```

Objectives

1) Addition of two 16 bit numbers using immediate addressing mode.
2) Subtraction of two 16 bit numbers using direct addressing mode.

```
MOV AX, 1234 H
SUB AX, [2000H]  <-- (stores 2211)
MOV [2002H], AX
HLT
```

```
 1234
-2211
 F023
```

lab-1

Obj 5

```
MOV  AX, 0000H ; AX ← 0000
MOV  DS, AX    ; DS ← 0000
MOV  SI, 3000H ; SI ← 3000
MOV  BX, 0500H ; BX ← 0500
MOV  AX, [SI+BX] ; AX ← [3500H] data 1
MOV  DX, [SI+BX+0002H] ; DX ← [3502H]
ADD  AX, DX
MOV  [SI+BX+0004H], AX .
HLT
```

Obj 6

```
MOV  AX, 0000H   ; AX ← 0000
MOV  DS, AX      ; DS ← 0000
MOV  BX, [2000H] ; BX ← data 1 = 0002
MOV  CX, [2002H] ; CX ← data 2 = 0003
L1 : ADD AX, BX  ; AX ← AX + BX = 0000 + 0002
                 =0002
                 +0002 = 0004
                 +0002
                 =0006
DEC  CX          ; CX ← 0002, 0001, 0000
JNZ  L1
MOV  [2005H], AX ; [2005H] ← 0006H.
HLT
```

Decrement ←

Jump if no zero ←

Obj 7

```
MOV AX, 0000H   ; AX ← 0000
MOV DS, AX      ; DS ← 0000
MOV AX, [2000H] ; AX ← data 1 = 0007
MOV BX, [2002H] ; BX ← data 2 = 0002
MOV CX, 0000H   ; CX ← 0000

L1 : CMP AX, BX ; AX > BX
     JC  L2
     SUB AX, BX
     INC CX
     JNC L1
L2 : MOV [2004H], CX ; [2004H] ← 0003
     MOV [2006H], AX ; [2006H] ← 0001
     HLT
```

Exp-2

Obj 1 - Find the 2's complement of an 8-bit number.

```
MOV AX,0000H
MOV DS, AX
MOV AL, [3000H]
NOT AL
ADD AL, 01H
MOV [3001H], AL
HLT
```

Ob2 - Find the gray code of an 8-bit binary number.

```
MOV AX, 000H.        ; AX ← 0000
MOV DS, AX           ; DS ← 0000
MOV AL, [3000H] ;    ; AL ← 35
MOV BL, AL           ; BL ← 35
SHR BL, 01H          ; BL ← Shifted
XOR AL, BL           ; AL ← AL ⊕ B
MOV [3001H], AL      ; [3001H] ← AL.
HLT
```

CX → used only in counter
DX → used only in Multiplication & division of 16-bit no.

Two base register : BX
BB (stack)

If we multiply al with bl then it will stored in AX.

Exp-3

obj

1) Find the sum and average of N 16-bit numbers.
2) Count no. of 1's in an 8-bit number.
3) Move a block of 16-bit data from one location to other.

Code.

ax ÷ bl (8-bit) → stored in al → Quotient; ah → remainder

If we do 8-bit division then it will store in BL and if 16-bit then store in BX.

obj 1:

```
MOV AX, 1000H        ; AX ← 1000
MOV DS, AX           ; DS ← 1000
MOV SI, 2000H        ; SI ← 2000
MOV CX, [SI]         ; CL ← [2000] (the info/data that are located in SI will store in CL)
MOV BX, CX           ; BL ← [2000] (then the data will store in BL)
MOV DX, 0000H        ; DX ← 0000
MOV AX, 0000H        ; AX ← 0000
L2: INC SI           ; Increment SI
INC SI               ; Again increment SI
ADD AX, [SI]         ; the data located in SI will be added with AX.
JNC L1               ; Jump to L1 if there is no carry
INC DX               ; Increment DX
L1: DEC CX           ;
JNZ L2               ; Jump to L2 if there is no zero
INC SI               ; Increment SI
INC SI               ; Increment SI
MOV [SI], AX         ; SUM
DIV BX               ;
INC SI               ;
INC SI               ;
MOV [SI], AX         ;
INC SI               ;
INC SI               ;
MOV [SI], DX         ;
HLT
```

(Insert after MOV [SI], AX ; SUM:)

```
INC SI
INC SI
MOV [SI], DX
```

Margin notes:

This code is for upto FF.

C stores upto FF: CL

CX → CH: more than FF → store at CH

So write CX, BX instead of CL, BL

JNC: Jump if no carry

JNZ: Jump if no zero

16-bit

Multiplication: AX * BX = DX : AX

FFFF * FFFF = 0000 (DX) (higher bit) FFFE 0001 (AX)

BX (Divisor) | Dividend (DX:AX) | Q → AX

R → DX

Obj: MOV AX, 0000H
MOV DS, AX
MOV BX, 2000H
MOV AL, [BX]
MOV CL, 08H
MOV CH, 00H
LOOP2 : SHR AL, 01H
JNC LOOP1
INC CH
LOOP1 : DEC CL
JNZ LOOP2
INC BX
MOV [BX], CH
HLT

(modified) Obj 3 :
MOV AX, 0000H
MOV DS, AX → { MOV AX, 2000H ; MOV ES, AX
MOV SI, 2000H
MOV DI, 3000H
MOV CX, [SI]
BACK : INC SI
INC SI
MOV AX, DS : [SI]
MOV ES : [DI], AX

```
INC DI
INC DI
DEC CX
JNZ BACK
HLT
```

Exp-4

Obj 1 Find the largest and smallest number in each array of size N

```
MOV AX, 0000H
MOV DS, AX
MOV SI, 2000H
MOV DI, 3000H
MOV CL, [SI]      ; CL ← 02
XOR CH, CH        ; CH=00
INC SI            ; SI [2001] ← 11
MOV AL, [SI]      ; AL ← 11
DEC CX            ; CX ← 1
AGAIN: INC SI     ; SI [2002] ← 55
MOV BL, [SI]      ; BL ← 55
CMP AL, BL        ; AL < BL
JNC NEXT
MOV AL, BL
NEXT: LOOP AGAIN
MOV [DI], AL
HLT
```

2000 → 02 | 2001 → 11 | 2002 → 55

(to make CH → zero)

(compares two operands)

Obj 2 : Arrange the elements of a given array of size N in ascending and descending order:

```
MOV CH, 05H
L1: MOV CL, 05H
```

```
MOV SI, 3000H
L2: MOV AL, [SI]
MOV AH, [SI+1]
CMP AL, AH
JNC L3
JZ L3
MOV [SI], AH
MOV [SI+1], AL
L3: INC SI
DEC CL
JNZ L2
DEC CH
JNZ L1
HLT
```