8086 Microprocessor

- The 8086 is a first 16-bit microprocessor developed by Intel (HIgh Performance bulK CMOS)
- It is designed using the HMOS technology and contain approximately 29,000 transistors.

Features -

1. It operates on single +5V power supply.
2. The 8086 has 20 bit address lines, hence it can address 1M byte memory locations.
3. It operates with 5MHz clock frequency.
4. It has 16 multiplexed address/data bus which reduces the number of pins.
5. The 8086 performs arithmetic, & logical operations on bit, byte, word including multiply and divide operations.
6. The 8086 can generate 16 bit I/O address hence it can access 2^{16} = 65535 I/O ports.
7. The 8086 work with register, direct, immediate, register indirect, based and indexed addressing modes.
8. It consist of four segments CS (Code Segment), DS (Data Segment), ES (Extra Segment) and SS (Stack Segment)
9. The 8086 supports multiprogramming.

Architecture of 8086

The 8086 is internally divided into two separate functional units. These are Bus Interface Unit (BIU) & Execution Unit(EU)

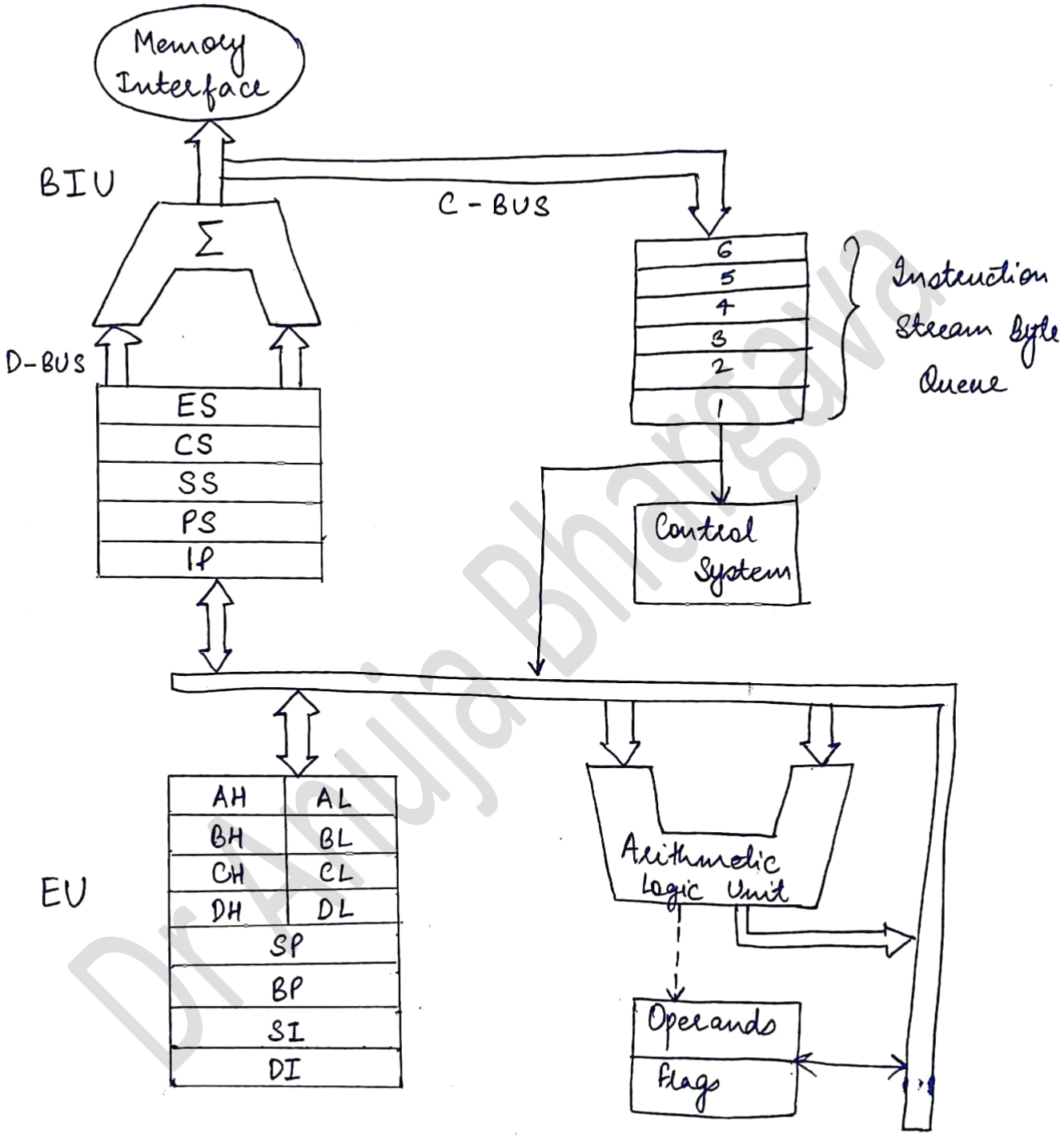


Fig : Internal Architecture of 8086 Microprocessor

1. Bus Interface Unit (BIU):

- The BIU fetches instruction, reads data from memory and ports, and writes data to memory and I/O ports.
- The EU executes instructions that have already been fetched by the BIU.
- The BIU and EU work independently.
- The BIU handles all transfer of data and addresses on the buses for the execution unit.
- The BIU consist of instruction pointer, segment register, instruction queue and address generation / bus control circuitary to provide functions such as fetching and queuing of instruction and bus control.

(i) The Queue

→ The queue is a first in first out group of register in which upto 6 bytes of instruction code are prefetched from memory ahead of time.

→ The BIU can be fetching instruction bytes while the EU is decoding an instruction which does not required use of buses.

→ When EU is ready for its next instruction, it simply reads the instruction from the queue in BIU.

→ If an instruction such as jump or CALL encountered, the BIU will reset the queue and began refilling after passing the new instruction to the EU.

→ The process of fetching next instruction while current instruction is executing is known as pipelining. This scheme greatly speeds up processing.

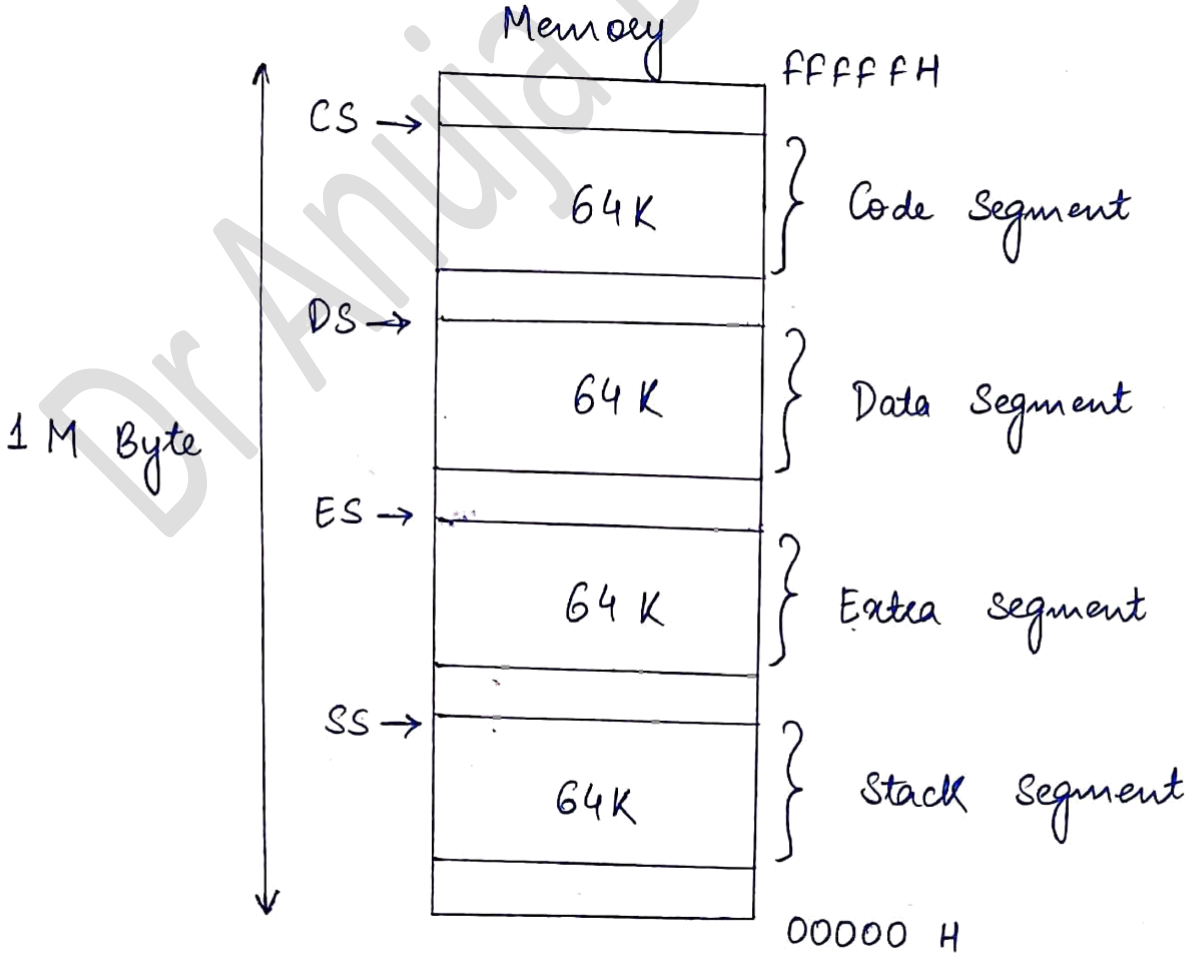
(ii) Segment Register :

→ The BIU has four 16 bit registers. These are Code Segment (CS), Data Segment (DS), Stack Segment (SS) and Extra Segment (ES) registers.

→ The 8086 uses memory segmentation

★ 1 M bytes of memory is divided into segments with maximum size of segment 64 K byte.

★ The 8086 can directly address four segments at a particular time.



* Thus a location with in a segment can be addressed using 16 bits.
* These segment register used to hold the upper 16 bits of the starting address of four memory segments.
* The part of segment starting address stored in segment register is called as segment base.

Code Segment (CS) Register

→ The code segment holds the program code.
→ The upper 16 bits of the starting address of coded segment is loaded in CS Register.
→ CS register is of 16 bit wide.

eg If CS = 3456 H, that means code segment starts from Memory location 34560 H (3456 * 10), which is of 20 bit address.

Data Segment (DS) Register

→ During program execution, the microprocessor access a memory location for read or write operation.
→ In 8086, there will be a separate data segment where the data will reside.
→ The upper 16 bits of the starting address of data segment is loaded in DS register.
→ The data segment holds the data constraint needed by the program.

Stack Segment (SS) Register

→ The upper 16-bits of the starting address of Stack Segment is loaded in SS Register.

(iii) Instruction Pointer (IP)

→ It points to the instruction to be fetched and executed next.

→ The IP register holds the 16-bit address of the next code byte with in the code segment.

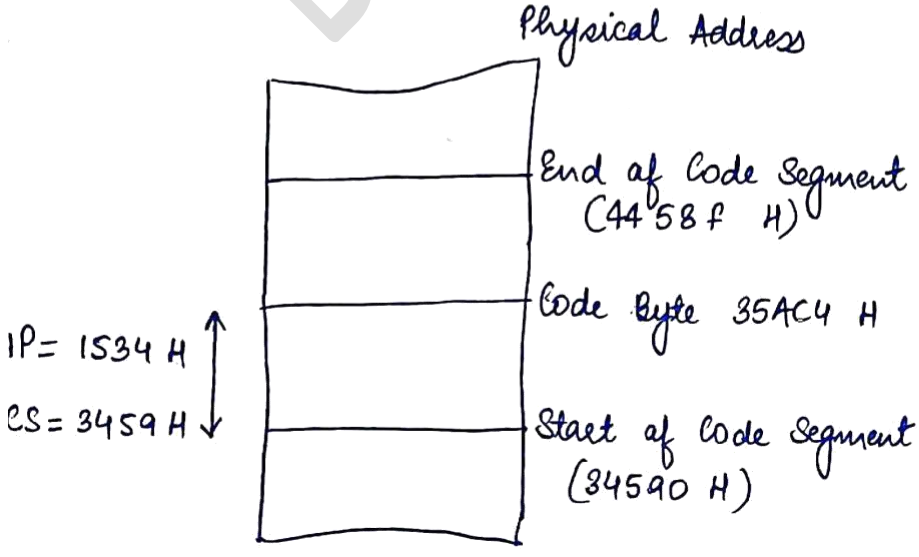
→ The value contained in IP is referred as offset, because the value must be offset from (added to) the segment base address in CS to produce the required 20-bit physical address.

Generation of 20-bit address ★

How 20 bit address is generated by BIU using the 16 bit IP.

Assume that IP content is 1534 H. This is called offset value or effective address.

Let the content of CS be 3459 H. This is called segment base value.



$$\begin{array}{r} CS \quad 34590 \\ IP + \quad 1534 \\ \hline 35AC4 \end{array}$$

- The CS Register points to the base or start of current code segment. The IP contains the offset from base address to the next instruction byte to be fetched.
- Here, the content of CS register are shifted left four bit position before the content of IP are added to it.

Eg Content of CS = 3459 H

Shifted by four position = 34590 H

∴ Code Segment starts at 34590 H

Now, fetch an instruction from memory location which is 1534 H bytes away from the beginning of code segment.

∴ 34590 H + 1534 H (using adder in BIU)

sends out as 35AC4 H as physical address on 20-bit address pins.

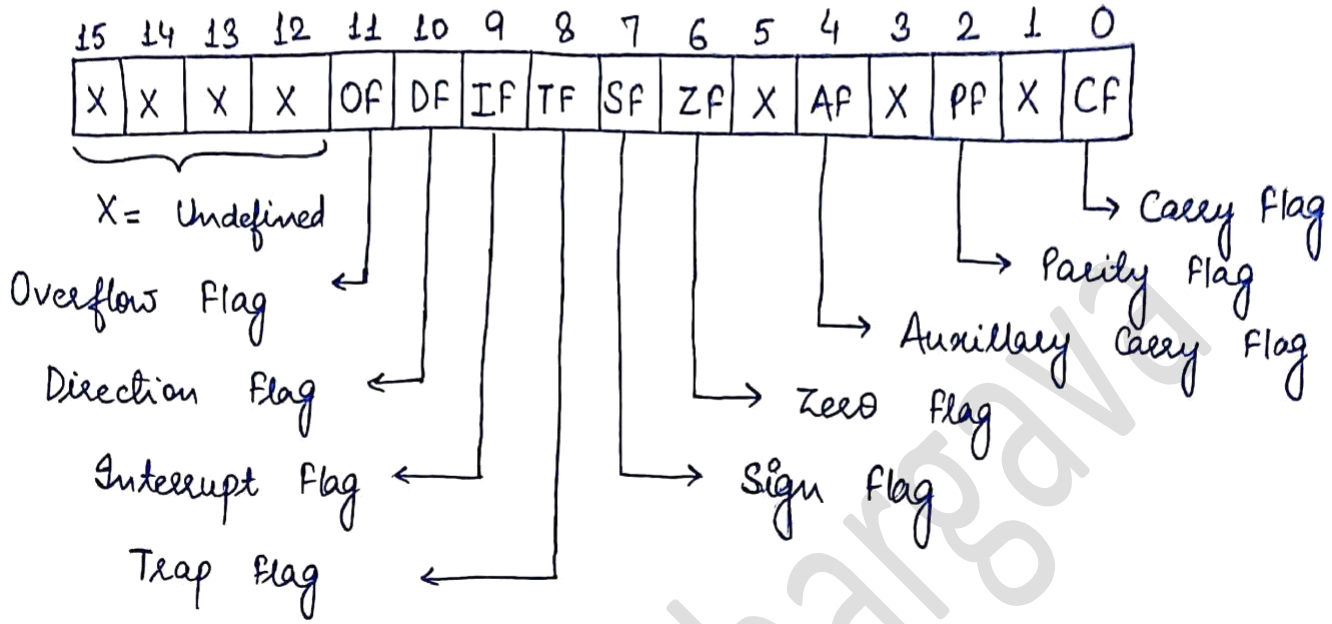
- To access data in memory the 8086 must produce a 20 bit physical address. It does this by adding a 16-bit value called the effective address to one of the four segment base.

(2) Execution Time :- (EU)

- It is responsible for executing the instruction fetched by BIU.
- It consist of Instruction Decoder (ID), Control Unit (CU), Arithmetic Logic Unit (ALU), general purpose register, flag register and other registers.

Flag Register :-

- The 8086 PSW contains 16 bits, but 7 of them are not in used. Each bit in PSW is called a flag.
- A flag is a flip-flop which will be in 1 or 0 state.



- The 8086 flags can be divided into two types
  (1) Conditional flag (2) Control flag

(1) Conditional Flag

→ It stores the data condition according to the result of the previous operation in ALU.

→ There are six conditional flags

(i) CF (Carry Flag)

It is set to 1, if there is a carry out of MSB during addition operation and borrow in overall subtraction operation

(ii) ZF (Zero Flag)

It is set to 1, whenever result stored in register or memory location is zero.

(iii) AF (Auxilliary Carry Flag)

It is set to 1, if there is a carry out of bit 3 during addition or borrow during subtraction.

(iv) SF (Sign Flag)

It is set to 1, if the result is negative.

It is set to 0, if the result is positive.

Note :- This flag is meaningful only if we are working with signed number.

(v) PF (Parity Flag)

After arithmetic operation, the 8086 count the numbers of 1's in the result.

It is set to 1, if there are even number of 1's.

It is set to 0, if there are odd numbers of 1's.

(vi) OF (Overflow Flag)

It is set to 1, if overflow occurs i.e result is out of range. The processor gives a wrong answer sometimes, when it is working with signed numbers. If we add two positive numbers, the result is expected to be positive. But in some cases result is negative, which indicates wrong answer. In such cases OF is set to 1.

(2) Control Flag

→ It is used to control certain operations of the processor.

→ There are three control flags

(i) IF (Interrupt Enable Flag)

→ If it is set to 1, a maskable interrupt can be recognized by the CPU, otherwise these interrupts are ignored.

→ The INTR is maskable interrupt of 8086.

→ If the IF is reset to 0, the 8086 does not respond to the request on INTR.

→ If the IF is set to 1, the 8086 responds to the request on INTR pin.

→ The IF bit has no effect on NMI, which is a non-maskable interrupt.

(ii) TF (Trap Flag)

→ When this flag is set to 1, the 8086 enters in a 'single stepping' mode.

→ To detect the error in the programs, it is necessary to run the programs one instruction at a time. This process is called 'single stepping' through a program.

→ But there is no instruction in 8086 to directly set the TF flag to 1, or reset the TF flag to 0.

→ The instructions given below are used to set TF to 1.

```
PUSH F              : Push flag on stack
MOV BP, SP          : Copy SP to BP
OR WORD PTR, 0100H: Set TF bit
POP F               : Restore flag register
```

→ The instruction given below are used to reset TF to 0.

PUSH F	:	Push flags on stack
MOV BP, SP	:	Copy SP to BP
AND WORD PTR, 0FEFF H	:	Reset TF bit
POP F	:	Restore flag register

(iii) DF (Direction Flag)

→ It is used by string manipulation instruction.

→ The content of SI (Source Indexed Register) and DI (Destination Indexed Register) is automatically incremented or decremented by setting the DF.

→ Automatic increment of SI and DI is done if DF is 0.
Automatic decrement of SI and DI is done if DF is 1.

Numerical.

eg1 Calculate flag status.

1) ADD AX, BX

Before Execution

AX = 2345 H 0010 0011 0100 0101

BX = 3219 H + 0011 0010 0001 1001

0101 0101 0101 1110

After Execution

AX = 555E H

BX = 3219 H

SF = 0 , ZF = 0 , PF = 0 , CF = 0 , AF = 0 , OF = 0

2) ADD BX, CX

Before Execution

BX = 1234 H

CX = 7F2E H

```
  0001  0010  0011  0100
+ 0111  1111  0010  1110
-----------------------
  1001  0001  0110  0010
```

After Execution

BX = 9162 H

CX = 7F2E H

SF = 1, ZF = 0, PF = 0, CF = 0, AF = 1, OF = 1

3) ADD BL, CH

Before Execution

BL = 04 H

CH = FC H

```
    0000  0100
+   1111  1100
-------------
1   0000  0000
```

After Execution

BL = 00 H

CH = FC H

SF = 0 , ZF = 1 , PF = 1, CF = 1 , AF = 1, OF = 0

General Purpose Registers

- The EU has four general purpose registers labelled as AX, BX, CX and DX. These registers are used to store 16 bit data.
- These registers are divided into two 8 bit portions.
  - AX is divided as AH and AL.
  - BX is divided as BH and BL.
  - CX is divided as CH and CL.
  - DX is divided as DH and DL.

Other Pointers

SP - Stack Pointer

It points to the top of the stack and is used for stack operation.

BP - Base Pointer

It is used as base register for accessing the stack memory

Advantage of Segment Register

The advantage of using segment register for segmenting 1 M byte memory space are as follows

(1) It facilitates the use of separate areas of a program code, its data and stack. So that program will be fetched from CS, while program data will be stored in ES and DS.

(2) Since SS area of memory is different from CS and DS, there is no possibility of stack getting overlapped.

(3) It has Multiprogramming environment.

(4) It allows memory capacity of 1M byte even though the address associated with individual instructions are only 16 bit.

(5) By structuring memory into separate area for separate operations, the programs are shorter, faster and more structured.

8086 Addressing Modes :-

- The way in which an operand is specified in the instruction or the various methods of accessing data in the execution of an instruction is called addressing modes.
- The 8086 addressing modes can be classified into five major categories.

1. Immediate Addressing Mode.

→ 8 or 16 bit data can be specified as part of instruction. The immediate operand can only be the source operand. Operand is specified with in the instruction itself.

eg (i) MOV CL, 03H : Move the 8-bit data 03H in CL.

(ii) MOV DX, 1234H : Move the 16-bit data 1234H in DX.

(iii) ADD AL, 03H : The 8-bit data 03H is added with AL and the result is stored in AL.

(iv) ADD AX, 1234H : The 16-bit data 1234H is added with AX and result is stored in AX.

2. Register Addressing Mode

→ The source and destination operand are specified in registers.

→ The operand can be 8-bit or 16-bit wide.

b) Indirect addressing mode.

→ In this, effective address is calculated from the content of registers.

→ IAM, subdivided into five categories

(i) Register Indirect Addressing Mode.

In this mode, EA is provided in an index register.

eg: MOV [DI], 1234 H : destination operand is a memory location specified using register indirect add. mode.

MOV AX, [BX] : source operand is a memory location specified using register indirect addressing mode.

(ii) Based Addressing with Displacement Mode.

In this mode, EA is sum of 8 or 16 bit displacement and the content of base register.

$$EA = \begin{Bmatrix} (BX) \\ (BP) \end{Bmatrix} + \begin{Bmatrix} \text{8 bit displacement} \\ \text{16 bit displacement} \end{Bmatrix}$$

eg: MOV [BX-2], 1234 H :

MOV AX, [BX + 200]

MOV [BX-5], AX

(iii) Indexed Addressing with Displacement Mode

In this case, effective address is sum of 8 or 16 bit displacement plus the content of index register (SI or DI)

eg MOV [DI + 2345], 1234

MOV AX, [SI +45]

(iv) Based Indexed Addressing Mode

In this, effective address is sum of base register (BX or BP) and indexed register (SI or DI) both of which are specified in instruction.

eg MOV [BX + DI], 1234

MOV AX, [SI +BX]

(v) Based Indexed Addressing with displacement mode

In this case, effective address is sum of 8 bit or 16 bit displacement plus base register (BX or BP) plus index register (SI or DI).

eg MOV [DI + BX + 37 H], AX

MOV AL, [BX + SI + 278H]