ADDRESSING MODES OF 8086

Every instruction of a program has to operate on a data. The method of specifying the data to be operated by the instruction is called addressing. The 8086 has 12 addressing modes and they can be classified into following five groups.

1. Register addressing 2. Immediate addressing	Group I	: Addressing modes for register and immediate data
3. Direct addressing 3. Register indirect addressing 5. Based addressing 6. Indexed addressing 7. Based index addressing 8. String addressing	Group II	: Addressing modes for memory data
9. Direct IO port addressing 10. Indirect IO port addressing	Group III	: Addressing modes for IO ports
11. Relative addressing	Group IV	: Relative addressing mode
12. Implied addressing	Group V	: Implied addressing mode

Note : 1. *The "register" or "register + constant" enclosed by square brackets in the operand field of instructions refer to the method of effective address calculation of memory. The 16-bit constant enclosed by square brackets in the operand field of instructions refer to the effective address of memory data. The 8-bit/16-bit constants which are not enclosed by square brackets in the operand field refer to immediate data.*

2. *The term MA used in the symbolic description of instructions refer to physical memory address of data segment memory and* MA_S *refer to physical memory address of stack segment memory and* MA_E *refer to physical memory address of extra segment memory.*

3. *The register/memory enclosed by brackets in symbolic description refer to the content of register/memory.*

4. *For hexa-decimal constant (data/address) the letter H is included at the end of 8-bit/16-bit constants (data/address), and the numeral 0 is included in the front of hexadecimal constant starting with A through F.*

Register Addressing

In register addressing, the instruction will specify the name of the register which holds the data to be operated by the instruction.

Examples *:*

a) MOV CL,DH **(CL) ← (DH)**

The content of 8-bit register DH is moved to another 8-bit register CL.

b) MOV BX,DX **(BX) ← (DX)**

The content of 16-bit register DX is moved to another 16-bit register BX.

Immediate Addressing

In immediate addressing mode, an 8-bit or 16-bit data is specified as part of the instruction.

Examples :

a) MOV DL,08H **(DL) ← 08_H**

The 8-bit data (08_H) given in the instruction is moved to DL-register.

b) MOV AX,0A9FH **(AX) ← $0A9F_H$**

The 16-bit data ($0A9F_H$) given in the instruction is moved to AX-register.

Direct Addressing

*In direct addressing, an unsigned 16-bit displacement or signed 8-bit displacement will be specified in the instruction. The displacement is the **E**ffective **A**ddress(EA) or offset. In case of 8-bit displacement, the effective address is obtained by sign extending the 8-bit displacement to 16-bit.*

The 20-bit physical address of memory is calculated by multiplying the content of DS-register by 16_{10} (or 10_H) and adding to effective address. When segment override prefix is employed, the content of segment register specified in the override prefix will be used for segment base address calculation instead of DS-register.

Examples :

a) MOV DX,[08H]

EA = 0008_H (sign extended 8-bit displacement)

BA = (DS) $\times 16_{10}$; MA = BA + EA

(DX) $\leftarrow$ (MA) or DL $\leftarrow$ (MA)

DH $\leftarrow$ (MA+1)

*The **E**ffective **A**ddress(EA) is obtained by sign extending the 8-bit displacement given in the instruction to 16-bit. The segment **B**ase **A**ddress(BA) is computed by multiplying the content of DS by 16_{10}. The **M**emory **A**ddress(MA) is computed by adding the **E**ffective **A**ddress (EA) to segment **B**ase **A**ddress(BA).*

The content of memory whose address is calculated as explained above is moved to DL-register and the content of next memory location is moved to DH-register.

b) MOV AX, [089DH]

$EA = 089D_H$; $BA = (DS) \times 16_{10}$; $MA = BA + EA$

$(AX) \leftarrow (MA)$ or $(AL) \leftarrow (MA)$

$(AH) \leftarrow (MA+1)$

*Here the 16-bit displacement given in the instruction is the effective address. The segment **B**ase **A**ddress(BA) is computed by multiplying the content of DS by 16_{10}. The **M**emory **A**ddress(MA) is computed by adding the **E**ffective **A**ddress(EA) to segment **B**ase **A**ddress(BA).*

The content of memory whose address is calculated as explained above is moved to AL-register and the content of next memory location is moved to AH-register.

Register Indirect Addressing

*In register indirect addressing, the name of the register which holds the **E**ffective **A**ddress(EA) will be specified in the instruction. The register used to hold the effective address are BX, SI and DI. The content of DS is used for segment base address calculation. When segment override prefix is employed, the content of segment register specified in the override prefix will be used for base address calculation instead of DS-register.*

The base address is obtained by multiplying the content of segment register by 16_{10}. The 20-bit physical address of memory is computed by adding the effective address to base address.

Examples : **a) MOV CX, [BX]**

EA = (BX) ; BA = (DS) $\times$ 16_{10} ; MA = BA + EA

(CX) $\leftarrow$ (MA) or (CL) $\leftarrow$ (MA)

(CH) $\leftarrow$ (MA+1)

*The content of BX is the **E**ffective **A**ddress(EA). The segment **B**ase **A**ddress(BA) is computed by multiplying the content of DS by 16_{10}. The **M**emory **A**ddress(MA) is obtained by adding BA and EA.*

The content of memory whose address is calculated as explained above is moved to CL-register and the content of next memory location is moved to CH-register.

b) MOV AX,[SI]

EA = (SI) ; BA = (DS) $\times$ 16_{10} ; MA = BA + EA

(AX) $\leftarrow$ (MA) or (AL) $\leftarrow$ (MA)

(AH) $\leftarrow$ (MA + 1)

*The content of SI is the **E**ffective **A**ddress (EA). The segment **B**ase **A**ddress(BA) is computed by multiplying the content of DS by 16_{10}. The memory address is obtained by adding BA and EA.*

The content of memory whose address is calculated as explained above is moved toAL-register and the content of next memory location is moved to AH-register.

Based Addressing

*In this addressing mode, the BX or BP-register is used to hold a base value for effective address and a signed 8-bit or unsigned 16-bit displacement will be specified in the instruction. The displacement is added to base value in BX or BP to obtain the **E**ffective **A**ddress(EA). In case of 8-bit displacement, it is sign extended to 16-bit before adding to base value.*

When BX is used to hold base value for EA, the 20-bit physical address of memory is calculated by multiplying the content of DS by 16_{10} adding to EA.

When BP is used to hold base value for EA, the 20-bit physical address of memory is calculated by multiplying the content of SS by 16_{10} and adding to EA.

Example :

MOV AX, [BX+08H]

0008_H 08_H ; EA = (BX) + 0008_H

BA = (DS) $\times$ 16_{10} ; MA = BA + EA

(AX) $\leftarrow$ (MA) or (AL) $\leftarrow$ (MA)

(AH) $\leftarrow$ (MA+1)

*The effective address is calculated by sign extending the 8-bit displacement given in the instruction to 16-bit and adding to the content of BX-register. The **B**ase **A**ddress(BA) is obtained by multiplying the content of DS by 16_{10}. The **M**emory **A**ddress(MA) is obtained by adding BA and EA.*

The content of memory whose address is calculated as explained above is moved to AL-register and the content of next memory is moved to AH-register.

Indexed Addressing

*In this addressing mode, the SI or DI-register is used to hold an index value for memory data and a signed 8-bit displacement or unsigned 16-bit displacement will be specified in the instruction. The displacement is added to index value in SI or DI-register to obtain the **E**ffective **A**ddress (EA). In case of 8-bit displacement it is sign extended to 16-bit before adding to index value.*

*The 20-bit memory address is calculated by multiplying the content of **D**ata **S**egment (DS) by 16_{10} and adding to EA.*

Example :

MOV CX, [SI+0A2H]

$FFA2_H \xleftarrow{\text{sign extend}} A2_H$; EA = (SI) + $FFA2_H$

BA = (DS) × 16_{10} ; MA = BA + EA

(CX) ← (MA) or (CL) ← (MA)

(CH) ← (MA + 1)

*The effective address is calculated by sign extending the 8-bit displacement given in the instruction to 16-bit and adding to the content of SI-register. The **B**ase **A**ddress (BA) is obtained by multiplying the content of DS by 16_{10}. The **M**emory **A**ddress(MA) is obtained by adding BA and EA.*

The content of memory whose address is calculated as explained above is moved to CL-register and the content of next memory is moved to CH-register.

Based Indexed Addressing

In this addressing mode, the effective address is given by sum of base value, index value and an 8-bit or 16-bit displacement specified in the instruction. The base value is stored in BX or BP-register. The index value is stored in SI or DI-register. In case of 8-bit displacement, it is sign extended to 16-bit before adding to base value. This type of addressing will be useful in addressing two dimensional arrays where we require two modifiers.

When BX is used to hold base value for EA, the 20-bit physical address of memory is calculated by multiplying the content of DS by 16_{10} and adding to EA.

When BP is used to hold the base value for EA, the 20-bit physical address of memory is obtained by multiplying the content of SS-register by 16_{10} and adding it to EA.

Examples : **MOV DX, [BX+SI+0AH]**

$000A_H \xleftarrow{\text{sign extend}} 0A_H$; $EA = (BX) + (SI) + 000A_H$

$BA = (DS) \times 16_{10}$; $MA = BA + EA$

$(DX) \leftarrow (MA)$ or $(DL) \leftarrow (MA)$

$(DH) \leftarrow (MA + 1)$

*The **E**ffective **A**ddress(EA) is calculated by sign extending the 8-bit displacement given in the instruction to 16-bit and adding it to the content of BX and SI-register. The **B**ase **A**ddress(BA) is obtained by multiplying the content of DS by 16_{10}. The 20-bit **M**emory **A**ddress(MA) is obtained by adding BA and EA.*

The content of memory whose address is calculated as explained above is moved to DL-register and the content of the next memory location is moved to DH-register.

STRING ADDRESSING

*This addressing mode is employed in string instructions to operate on string data. In string addressing mode, the **E**ffective **A**ddress(EA) of source data is stored in SI-register and the EA of destination data is stored in DI-register.*

The segment register used for calculating base address for source data is DS and can be overridden. The segment register used for calculating base address for destination is ES and cannot be overridden.

*This addressing mode also supports auto increment/decrement of index registers SI and DI depending on **D**irection **F**lag(DF). If DF=1, then the content of index registers are decremented to point to next byte/word of the string after execution of a string instruction. If DF=0, then the content of index registers are incremented to point to previous byte/word of the string after execution of a string instruction. (For word operand, the content of index registers are incremented/decremented by two and for byte operand, the content of index registers are incremented/decremented by one.)*

MOVS BYTE

$EA = (SI)$; $BA = (DS) \times 16_{10}$; $MA = BA + EA$

$EA_E = (DI)$; $BA_E = (ES) \times 16_{10}$; $MA_E = BA_E + EA_E$

$(MA_E) \leftarrow (MA)$

If DF = 1, then $(SI) \leftarrow (SI) - 1$ and $(DI) \leftarrow (DI) - 1$

If DF = 0, then $(SI) \leftarrow (SI) + 1$ and $(DI) \leftarrow (DI) + 1$

This instruction move a byte of string data from one memory location to another memory location. The address of source memory location is calculated by multiplying the content of DS by 16_{10} and adding to SI. The address of destination memory location is calculated by multiplying the content of ES by 16_{10} and adding to DI.

After the move operation if DF = 1, then the content of index registers DI and SI are decremented by one. If DF = 0, then the content of index registers DI and SI are incremented by one.

Direct IO Port Addressing

This addressing mode is used to access data from standard I O-mapped devices or ports. In the direct port addressing mode, an 8-bit port address is directly specified in the instruction.

Example :

IN AL, [09H]

$PORT_{addr} = 09_H$

(AL) ← (PORT)

The content of the port with address 09_H is moved to AL-register.

Indirect IO Port Addressing

This addressing mode is used to access data from standard IO-mapped devices or ports. In the indirect port addressing mode, the instruction will specify the name of the register which holds the port address. In 8086, the 16-bit port address is stored in DX-register.

Example :

OUT [DX], AX

$PORT_{addr}$ **= (DX) ; (PORT) ← (AX)**

The content of AX is moved to the port whose address is specified by DX-register.

Relative Addressing

In this addressing mode the effective address of a program instruction is specified relative to the **I**nstruction **P**ointer (IP) by an 8-bit signed displacement.

Example :

JZ 0AH

$000A_H$ ← sign extend $0A_H$

If ZF = 1, then (IP) ← (IP) + $000A_H$; $EA_C = (IP) + 000A_H$

$BA_C = (CS) \times 16_{10}$; $MA_C = BA_C + EA_C$

Note : Suffix C refers to code memory

If ZF = 1, then the program control jumps to a new code address as calculated above. If ZF = 0, then the next instruction of the program is executed.

Implied Addressing

In implied addressing mode, the instruction itself will specify the data to be operated by the instruction.

Example :

CLC - Clear carry ; CF ← 0

Execution of this instruction will clear the **C**arry **F**lag(CF).

MEMORY ADDRESS CALCULATION IN 8086 USING DEFAULT SEGMENT REGISTER

S.No.	Addressing mode	Effective address EA	Physical address MA/MA_s
1.	[BX + SI]	EA = (BX) + (SI)	MA = (DS) × 16_{10} + EA
2.	[BX + SI + disp8]	disp8 $\xrightarrow{\text{sign extend}}$ disp16 EA = (BX) + (SI) + disp16	MA = (DS) × 16_{10} + EA
3.	[BX + SI + disp16]	EA = (BX) + (SI) + disp16	MA = (DS) × 16_{10} + EA
4.	[BX + DI]	EA = (BX) + (DI)	MA = (DS) × 16_{10} + EA
5.	[BX + DI + disp8]	disp8 $\xrightarrow{\text{sign extend}}$ disp16 EA = (BX) + (DI) + disp16	MA = (DS) × 16_{10} + EA
6.	[BX + DI + disp16]	EA = (BX) + (DI) + disp16	MA = (DS) × 16_{10} + EA
7.	[BP + SI]	EA = (BP) + (SI)	MA_S = (SS) × 16_{10} + EA

8.	[BP + SI + disp8]	disp8 $\xrightarrow{\text{sign extend}}$ disp16 EA = (BP) + (SI) + disp16	$MA_S = (SS) \times 16_{10} + EA$
9.	[BP + SI + disp16]	EA = (BP) + (SI) + disp16	$MA_S = (SS) \times 16_{10} + EA$
10.	[BP + DI]	EA = (BP) + (DI)	$MA_S = (SS) \times 16_{10} + EA$
11.	[BP + DI + disp8]	disp8 $\xrightarrow{\text{sign extend}}$ disp16 EA = (BP) + (DI) + disp16	$MA_S = (SS) \times 16_{10} + EA$
12.	[BP + DI + disp16]	EA = (BP) + (DI) + disp16	$MA_S = (SS) \times 16_{10} + EA$
13.	[SI]	EA = (SI)	$MA = (DS) \times 16_{10} + EA$
14.	[SI + disp8]	disp8 $\xrightarrow{\text{sign extend}}$ disp16 EA = (SI) + disp16	$MA = (DS) \times 16_{10} + EA$
15.	[SI + disp16]	EA = (SI) + disp16	$MA = (DS) \times 16_{10} + EA$
16.	[DI]	EA = (DI)	$MA = (DS) \times 16_{10} + EA$
17.	[DI + disp8]	disp8 $\xrightarrow{\text{sign extend}}$ disp16 EA = (DI) + disp16	$MA = (DS) \times 16_{10} + EA$

18.	[DI + disp16]	EA = (DI) + disp16	MA = (DS) × 16_{10} + EA
19.	[disp16]	EA = disp16	MA = (DS) × 16_{10} + EA
20.	[BP + disp8]	disp8 $\xrightarrow{\text{sign extend}}$ disp16 EA = (BP) + disp16	MA_S = (SS) × 16_{10} + EA
21.	[BP + disp16]	EA = (BP) + disp16	MA_S = (SS) × 16_{10} + EA
22.	[BX]	EA = (BX)	MA = (DS) × 16_{10} + EA
23.	[BX + disp8]	disp8 $\xrightarrow{\text{sign extend}}$ disp16 EA = (BX) + disp16	MA = (DS) × 16_{10} + EA
24.	[BX + disp16]	EA = (BX) + disp16	MA = (DS) × 16_{10} + EA

Note : *Segment registers used in address calculation can be modified using segment override prefix.*
MA → Memory address of data segment ; MA_s → Memory address of stack segment.

Segment Override Prefix

The **Segment Override Prefix** says that if we want to use some other segment register than the default segment for a particular code, then it is possible. It can simply be one by mentioning the segment that is to be used before the address location or the offset register containing that address. By doing so, the machine, i.e. the 8086 microprocessor, while calculating the effective address will consider the mentioned segment for calculating the effective address rather than opting for the default one.

The syntax of doing so, as mentioned earlier is by mentioning the segment just before the address location and proceeded by a colon. The following abbreviations for each segment register are used for this purpose:

- Stack Segment Register - SS
- Data Segment Register - DS
- Code Segment Register - CS
- Extra Segment Register - ES

Let us take the following examples to further understand this concept:

```
MOV AX , [BX]
```

This is a normal instruction without any segment overriding. Hence the effective address will be calculated by using the default segment itself. Therefore, the effective address for the above-mentioned instruction:

Effective address = DS X 10H + content of BX register

```
MOV AX, SS : [BX]
```

Here, in this case, the Stack segment register is used as a prefix for the offset **BX**. So, instead of **DS**, which is the default segment register for **BX**, the **SS** will be used for finding the effective address location. Therefore, the effective address in the above-mentioned equation will be:

Effective address = SS X 10H + content of BX register