Experiment Number : (4)

Addressing Modes

Object:

To study the method of addressing modes to access any location of memory.

Theory:

We learn from the last experiment how to access the memory and we known Physical Address = Segment * (10H) + Offset. The instructions is divided into two part the first part is called opcode and the second part is called operand for example MOV AX,1234H the opcode is (MOV) and there operand is (AX, 1234H), always all instruction have one opcode and either one or two or three operand, above instruction have three operand and there (AX, 12, 34). The size of all registers is 16 bits and the size of the address bus is 20 bits (We can access memory of 2^{20} = 1048576 = 1 Mbytes) therefore we cannot access all memory location by use register only. Therefore we have main five methods to addressing the memory and there:

1. Direct Addressing Mode:

These method represented by give the segment to one of the segment registers and give the address directly inside [ ] symbols.

MOV REG, [Address]

Example: let you have DS = 0200H, CS = 0100H, IP = 0000H, CX=BEEDH, and you execute the instruction MOV CX, [1234H]. How these instruction is executed and what are the memory mapping of it?

- The starting address of your instruction is:
  PA = CS*10H+IP = 0100 * 10 + 0000 = 1000H.
- The starting address of your data is:
  PA = DS*10H+1234 = 200*10+1234 = 03234H
  See Fig. (4-1).

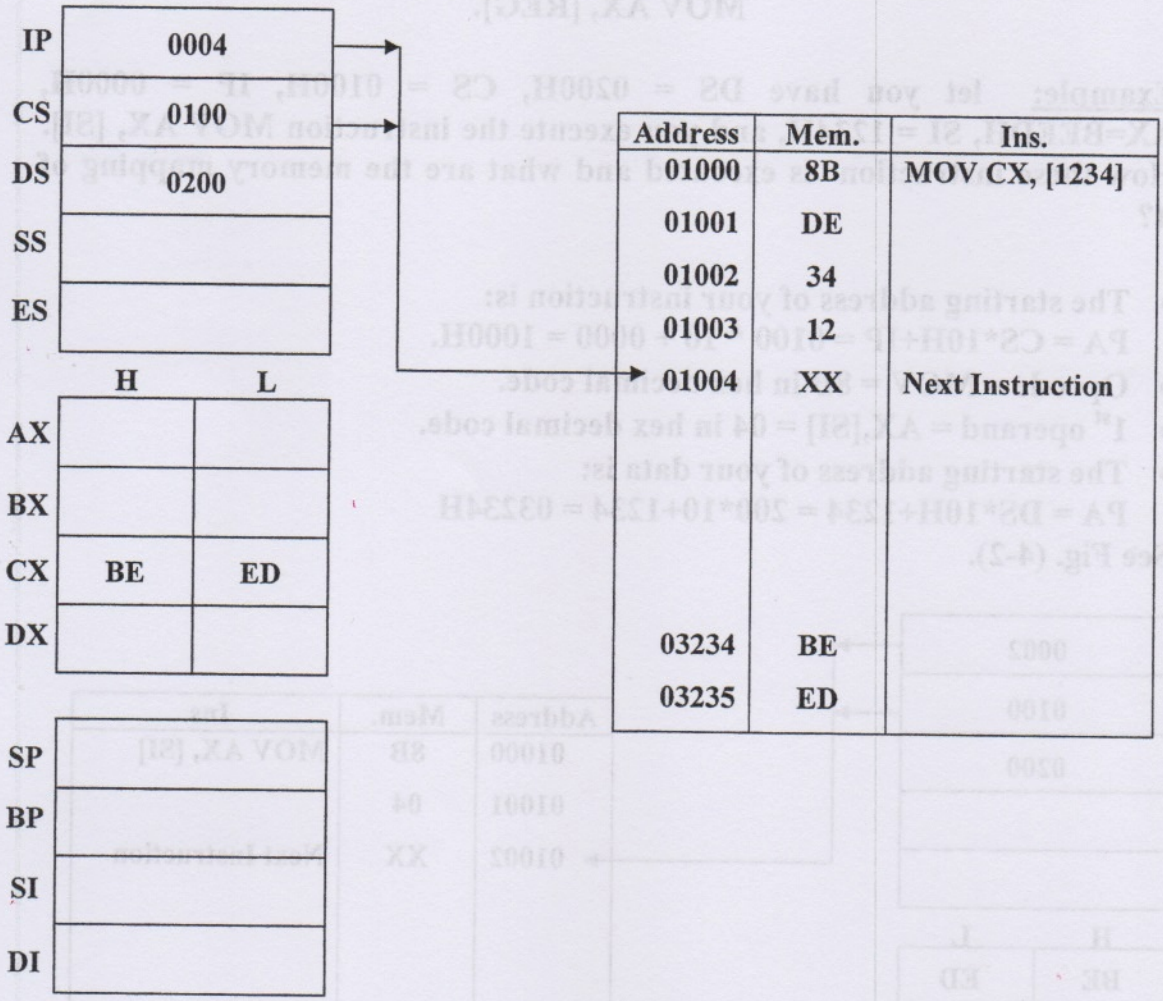

Fig. (4-1)
Memory mapping for direct addressing mode

2. Register Indirect Addressing Mode:

These method is the same as the direct addressing mode method but the address is put to the one of the register (BX, SI, DI).

MOV AX, [REG].

Example: **let you have DS = 0200H, CS = 0100H, IP = 0000H, AX=BEEDH, SI = 1234H, and you execute the instruction MOV AX, [SI]. How these instruction is executed and what are the memory mapping of it?**

- **The starting address of your instruction is:**
  PA = CS*10H+IP = 0100 * 10 + 0000 = 1000H.
- **Opcode = MOV = 8B in hex decimal code.**
- **1st operand = AX,[SI] = 04 in hex decimal code.**
- **The starting address of your data is:**
  PA = DS*10H+1234 = 200*10+1234 = 03234H

See Fig. (4-2).

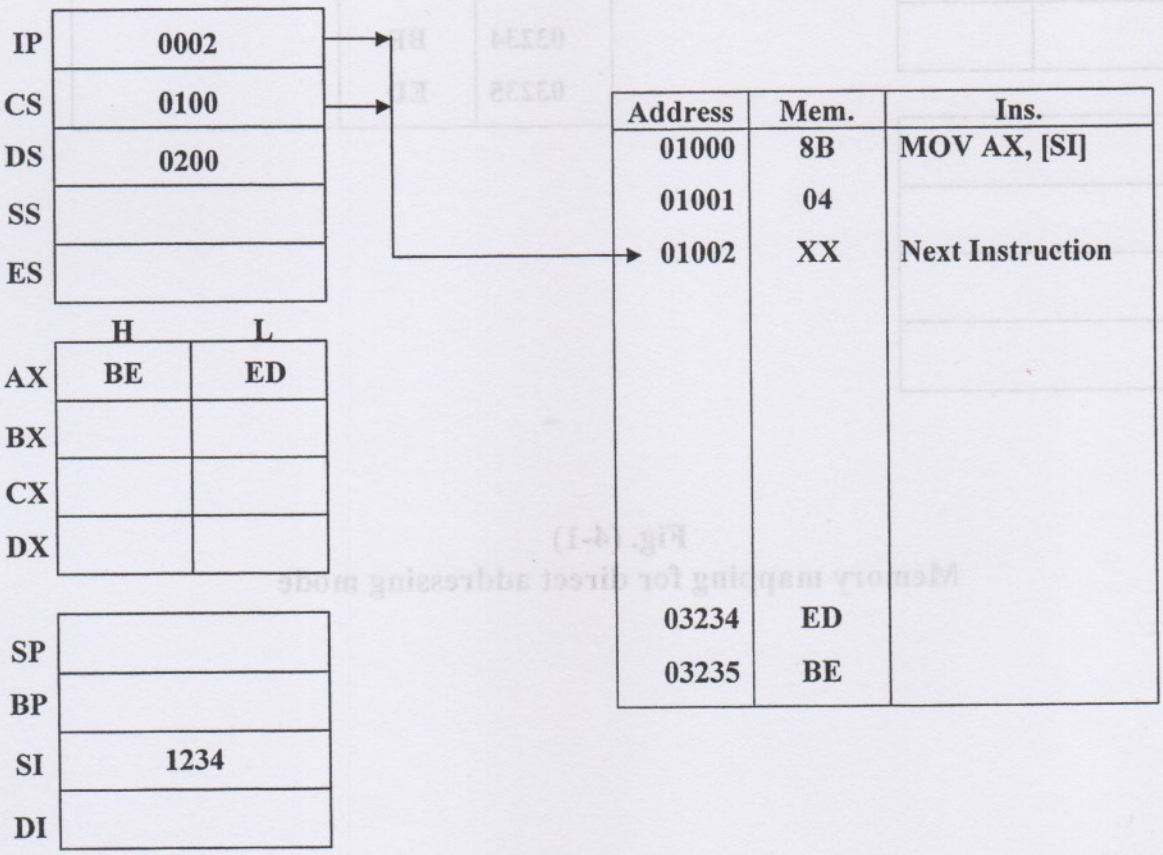

Fig. (4-2)
Memory mapping for register indirect addressing mode

3. Base Addressing Mode:

This method of addressing is represented by put the segment into DS and divided the offset into two part, the first part is put into register BX and last part either put directly or use one of (SI, DI) registers, so offset = BX+SI or DI.

MOV [BX]+1234H,AL

Example: let you have DS = 0200H, CS = 0100H, IP = 0000H, AX=BEEDH, BX = 1000H, and you execute the instruction MOV [BX]+1234H,AL. How these instruction is executed and what are the memory mapping of it?

- **The starting address of your instruction is:**
  PA = CS*10H+IP = 0100 * 10 + 0000 = 1000H.
- **The starting address of your data is:**
  PA = DS*10H+BX+1234H = 200*10+1000+1234 = 04234H

See Fig. (4-3).

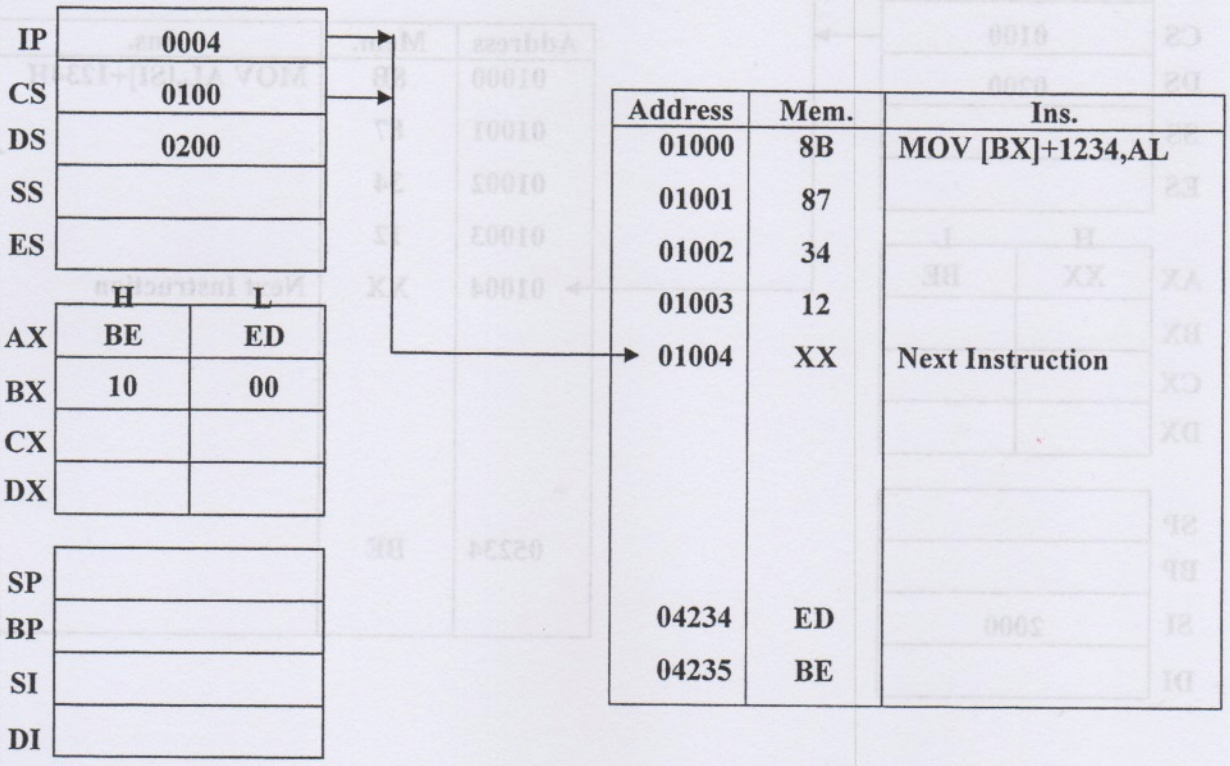

Fig. (4-3)
Memory mapping for base addressing mode

4. Index Addressing Mode:

This method of addressing mode is a simple as the last method of addressing mode (base addressing mode) except change the register BX to SI.

MOV AL,[SI]+1234H

Example: let you have DS = 0200H, CS = 0100H, IP = 0000H, AL=BEH, SI = 2000H, and you execute the instruction MOV AL,[SI]+1234H. How these instruction is executed and what are the memory mapping of it?

- **The starting address of your instruction is:**
  PA = CS*10H+IP = 0100 * 10 + 0000 = 1000H.
- **The starting address of your data is:**
  PA = DS*10H+SI+1234H = 200*10+2000+1234 = 05234H

See Fig. (4-4).

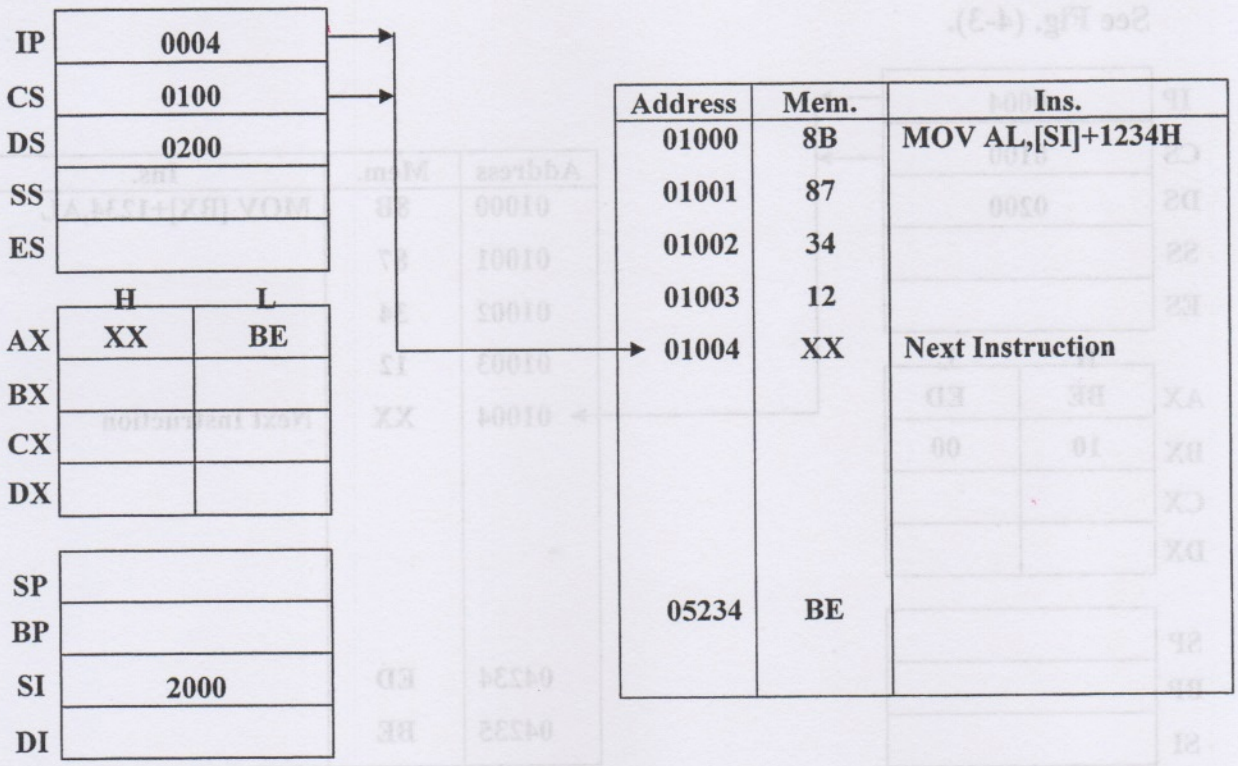

Fig. (4-4)
Memory mapping for index addressing mode

5. Base Index Addressing Mode:

This method we use the base register and source index register and other immediate offset to set the total offset of memory, (I.E. Offset = SI+BX+ immediate offset).

MOV AH,[SI]+[BX]+1234H.

Example: let you have DS = 0200H, CS = 0100H, IP = 0000H, AL=BEH, SI = 2000H, and you execute the instruction MOV AL,[SI]+1234H. How these instruction is executed and what are the memory mapping of it?

- **The starting address of your instruction is:**
  PA = CS*10H+IP = 0100 * 10 + 0000 = 1000H.
- **The starting address of your data is:**
  PA = DS*10H+SI+BX+1234H = 200*10+2000+1234 = 05234H

See Fig. (4-5).

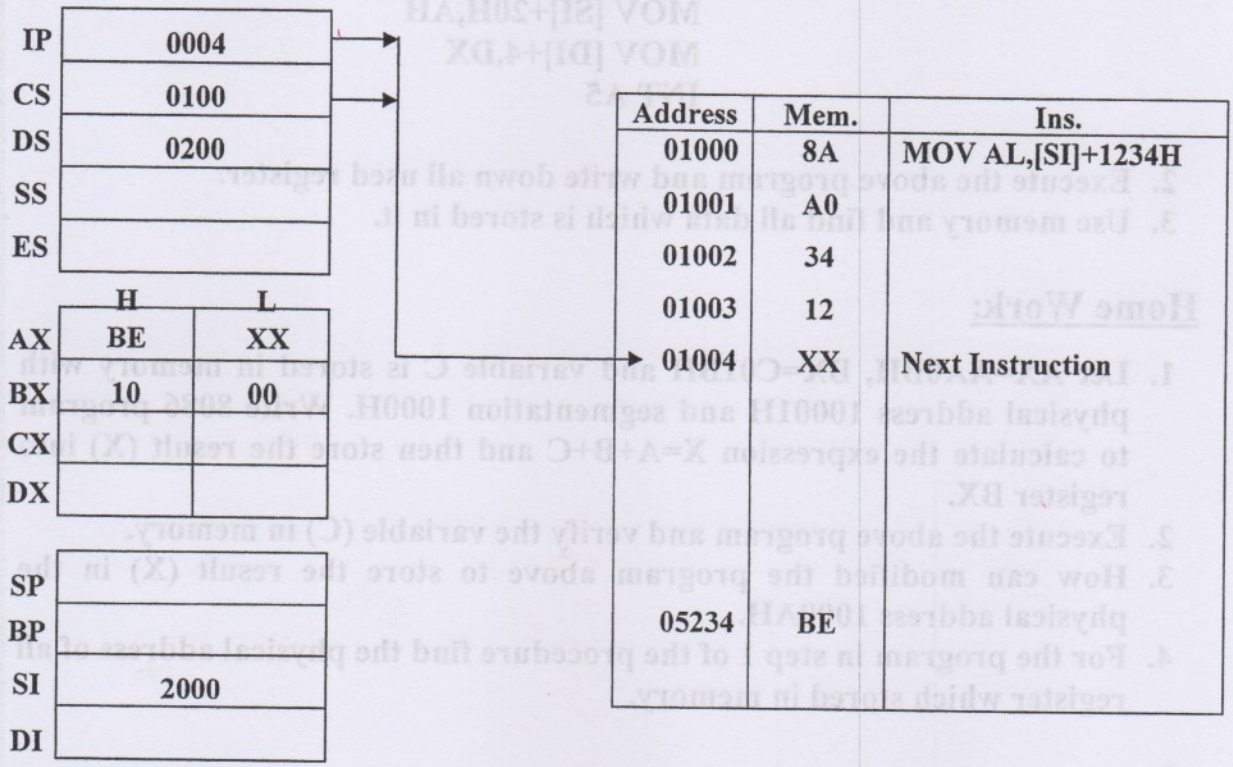

Fig. (4-5)
Memory mapping for index addressing mode

Procedure:

1. **Write the 8086 program to the 8086 trainer:**

```
MOV AX,1000H
MOV DS,AX
MOV AX,4000H
MOV SS,AX
MOV BX,0600H
MOV BP,0700H
MOV DX,96E2H
MOV  SI,0040H
MOV DI,00E0H
MOV AL,AAH
MOV [BX]+SI,AL
MOV [BX]+DI,AH
MOV [SI]+20H,AH
MOV [DI]+4,DX
INT A5
```

2. **Execute the above program and write down all used register.**
3. **Use memory and find all data which is stored in it.**

Home Work:

1. **Let AX=AA0BH, BX=C01BH and variable C is stored in memory with physical address 10001H and segmentation 1000H. Write 8086 program to calculate the expression X=A+B+C and then store the result (X) into register BX.**
2. **Execute the above program and verify the variable (C) in memory.**
3. **How can modified the program above to store the result (X) in the physical address 1000AH.**
4. **For the program in step 1 of the procedure find the physical address of all register which stored in memory.**

Experiment Number : (5)

Arithmetic Instructions

Object:

To learn what are the visible arithmetic instructions and how use these instruction in our program.

Theory:

There are three type of mathematical instructions supported:

1. First Group:

- **Add the second operand to the first without carry (The carry are neglect) the result is store in the first operand:**

(First operand = first operand + second operand)
ADD REG, IMMEDIATE
ADD MEMORY, IMMEDATE
ADD REG, REG
ADD REG, MEMORY
ADD MEMORY, REG

- **Add the second operand to the first with pervious state carry the result is store in the first operand:**

(First operand = first operand + second operand + carry)
ADC REG, IMMEDIATE
ADC MEMORY, IMMEDATE
ADC REG, REG
ADC REG, MEMORY
ADC MEMORY, REG

- **Subtract the second operand from first without borrow the result is store in the first operand:**

(First operand = first operand - second operand)
SUB REG, IMMEDIATE
SUB MEMORY, IMMEDATE
SUB REG, REG
SUB REG, MEMORY
SUB MEMORY, REG

- **Subtract the second operand from the first with borrow the result is store in the first operand:**

(First operand = first operand - second operand - borrow)
SBB REG, IMMEDIATE
SBB MEMORY, IMMEDATE
SBB REG, REG
SBB REG, MEMORY
SBB MEMORY, REG

- **Compare the second operand to the first and this result is only to set the flag register:**

CMP REG, IMMEDIATE
CMP MEMORY, IMMEDATE
CMP REG, REG
CMP REG, MEMORY
CMP MEMORY, REG

2. Second Group:

- **Multiply the second operand with accumulator (AX) for unsigned number and the result is store in the accumulator (AX):**

AX = AL * operand **when operand is 8 bits.**
DXAX = AX * operand **when operand is 16 bits.**
MUL REG
MUL MEMORY

- **Multiply the second operand with accumulator (AX) for signed number and the result is store in the accumulator (AX):**

AX = AL * operand **when operand is 8 bits.**
DXAX = AX * operand **when operand is 16 bits.**
IMUL REG
IMUL MEMORY

- **Divided the accumulator (AX) to the second operand for unsigned number and the result is store in the accumulator (AX):**

AL = AX / operand **when operand is 8 bits.**
AH = remainder
AX = DXAX / operand **when operand is 16 bits.**
DX = remainder
DIV REG
DIV MEMORY

- **Divided the accumulator (AX) to the second operand for signed number and the result is store in the accumulator (AX):**

AL = AX / operand **when operand is 8 bits.**
AH = remainder
AX = DXAX / operand **when operand is 16 bits.**
DX = remainder
IDIV REG
IDIV MEMORY

3. Third Group:

- **Increase the operand by (1):**

Operand = operand + 1.
INC REG
INC MEMORY

- **Decrease the operand by (1):**

Operand = operand - 1.
DEC REG
DEC MEMORY

Procedure:

1. **Write a 8086 program to calculate the results of the expression (C=A+B), if A=0AH, B=06H. store the result into register CL.**
2. **Execute the program and give the result (with flag).**
3. **Modified the above program to store the result into memory location which physical address is (11100H), and write the results.**

Home Work:

1. **How can you modified the program in the procedure to calculate the following expression:**
   - **C = A+A-B.**
   - **$C = A^2 + B$.**
   - **C = B/A.**
2. **Write a program to store the reminder of the division (C = B/A) to the memory location which is the physical address is 14141H.**

Experiment Number : (6)

Logical Instructions

Object:

To understand the logical instructions and how can use it properly.

Theory:

The logical instructions are:

- **AND: Logical AND between all bits of two operands. These rules apply:**

0	AND	0	=	0
0	AND	1	=	0
1	AND	0	=	0
1	AND	1	=	1

As you see we get 1 only when both bits are 1. The result is stored in the first operand.
(First operand = First operand AND second operand)

```
AND  REG, IMMEDIATE
AND  MEMORY, IMMEDATE
AND  REG, REG
AND  REG, MEMORY
AND MEMORY, REG
```

- **TEST: The same as AND but for flag only.**
  (i.e. this instruction not change the first or second operand only change the flag register).
  These instruction are used to make decision during program execution. These instruction affect these flag only:

  CF, ZF, SF, OF, PF, AF.

```
TEST  REG, IMMEDIATE
TEST  MEMORY, IMMEDATE
TEST  REG, REG
TEST  REG, MEMORY
TEST MEMORY, REG
```

- **OR: Logical OR between all bits of two operands. These rules apply:**

0	OR	0	=	0
0	OR	1	=	1
1	OR	0	=	1
1	OR	1	=	1

As you see we get 1 every time when at least one of this bits is 1. The result is stored in the first operand.
(First operand = First operand OR Second operand).
OR REG, IMMEDIATE
OR MEMORY, IMMEDATE
OR REG, REG
OR REG, MEMORY
OR MEMORY, REG

- **XOR: Logical Exclusive OR (XOR) between all bits of two operands. These rules apply:**

0	XOR	0	=	0
0	XOR	1	=	1
1	XOR	0	=	1
1	XOR	1	=	0

As you see we get 1 every time when bits are different from each other. The result is stored in first operand.
(First operand = First operand XOR Second operand).
XOR REG, IMMEDIATE
XOR MEMORY, IMMEDATE
XOR REG, REG
XOR REG, MEMORY
XOR MEMORY, REG

- **NOT: Logical NOT which reverse each bit of operand. These rules apply:**

NOT	0	=	1
NOT	1	=	0

The result is stored in the operand and this instruction doesn't affect of the flag.

NOT REG
NOT MEMORY

NOTE:
REG: AX, BX, CX, DX, AH, AL. BH, BL, CH, CL, DH, DL, SI, DI, SP, BP.
MEMORY: [BX], [BX+SI=7], ETC…
IMMEDIATE: 5, -24, 3FH, 10001101B, ETC…

Example:

Write 8086 program to calculate the result of the logical expression
X=(A ⊕ B)+AB
Where A = 1010110101011000, B = 0000000011111111
Store the result into location DT1

Solution:

```
.DATA
DT1 DW 0
.CODE
MOV AX, @ DATA
MOV DS, AX
MOV SI, OFFSET DT1
MOV AX, 1010110101011000B
MOV BX, 0000000011111111B
MOV CX, AX
MOV DX, BX
XOR AX, BX
AND CX, DX
OR AX, CX
MOV [SI], AX
RET
```

Procedure:

1. Write 8086 program to evaluate the logical expression

   $X = AB + AC + (B \oplus C)$.

2. A = 5FH, B = 10H, C = 10010010B.
3. Store the results into location DT.
4. Execute above program and write down the results.

Home work:

1. Write 8086 program to find the parity even of the DT1 and DT2, store the results in the location DT3.

   DT1 = (1, 2, 3, 4, 5, 6)D.

   DT2 = (6, 7, 8, 9, 10, 11)D.
2. How can modify above program to find the odd parity?
3. How can modify the program in step 1 to store the results into stack?
4. Write a 8086 program to design half adder if you have A=15H, B=FAH. Store the results (Result of addition & carry) into location DT.

Experiment Number : (7)

Loop Instruction & Counter

Object:

To study the loop instruction and know the facilities of the counter.

Theory:

The loop instruction is one of the important instructions which is control of the flow of the program, and to avoid repeated the instruction more than one. The register which is used to loop instruction is CX.

The loop procedure is:

1. **Put the value of the number of repeated instructions to the register CX.**
2. **Kept the address of the first instruction which is looped.**
3. **At the end of the looped instructions put LOOP Label, where Label is the address of the first looped instruction.**

The structure of the loop instruction is:

```
.CODE
  |
  |
  |
MOV CX, Number of Loop
  |
  |
  |
L1: MOV --------
  |
  |
  |
LOOP L1
  |
  |
  |
RET
```

Example1:

Write 8086 program to count from 0 to 15D.

```
.CODE
MOV CX, 000FH
MOV AL, 0
L1: INC AL
LOOP L1
RET
```

Or

```
.CODE
MOV CX, 000FH
MOV AL, 0
L1: ADD AL, 1
LOOP L1
RET
```

Procedure:

1. **Write 8086 program to fill the memory with (0A) starting at the physical address 10000H to the physical address 10010H.**
2. **Execute the program above and write the results.**
3. **Write a 8086 program to add (05) to the above memory locations (in step 1).**

Home Work:

1. **Write a 8086 program to store the number from 0 to 50D to the memory starting at the physical address 10000H**

2. **Write a 8086 program to store the number (0 – 20)D instep of 5 to the stack and then write the results.**

3. **Find the value of the registers SP & SS and then calculate the physical address of the stack at the end of stored data.**

4. **Write a 8086 program to store the number from (16 to 0)H to memory starting at the physical address 10000H.**