3- Logic instruction:

AND, **OR**, **XOR**, and **NOT** Instructions

Mnemonic	Meaning	Format	Operation	Flags Affected
AND	Logical AND	AND D,S	(S) · (D) → (D)	OF, SF, ZF, PF, CF AF undefined
OR	Logical Inclusive-OR	OR D,S	(S) + (D) → (D)	OF, SF, ZF, PF, CF AF undefined
XOR	Logical Exclusive-OR	XOR D,S	(S) ⊕ (D) → (D)	OF, SF, ZF, PF, CF AF undefined
NOT	Logical NOT	NOT D	$(\bar{D}) \rightarrow (D)$	None

(a)

Destination	Source
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Accumulator	Immediate

(b)

Destination
Register
Memory

(c)

As shown in Fig. (a), the AND, OR, and XOR instructions perform their respective logic operations bit by bit on the specified source and destination operands, the result being represented by the final contents of the destination operand. Figure (b) shows the allowed operand combinations for the AND, OR, and XOR instructions. Fig. (c), shows the allowed operands for the NOT instruction, we see that this operand can be the contents of an internal register or a location in memory.

EX: Describe the result of executing the following sequence of instructions:

MOV AL , 01010101B

AND AL , 00011111B

OR AL , 11000000B

XOR AL , 00001111B

NOT AL

Here, B is used to specify a binary number.

Solution:

- The first instruction moves the immediate operand 01010101B = 55H into the AL register. This loads the data that are to be manipulated with the logic instructions.

- The next instruction performs a bit-by-bit **AND** operation of the contents of AL with immediate operand 00011111B = 1FH. This gives

  01010101B · 00011111B = 00010101B

  55H . 1F H = 15H

  This result is placed in destination register AL.

  (AL)= 00010101B=15H

Note that this operation has masked off the three most significant bits of AL.

- The third instruction performs a bit-by-bit logical OR of the present contents of AL with immediate operand 11000000B = C0H. This gives

  00010101B + 11000000B = 11010101B

  15H + C0H = D5H

  (AL) = 11010101B = D5H

This operation is equivalent to setting the two most significant bits of AL.

- The fourth instruction is an exc1usive-OR operation of the contents of AL with immediate operand 00001111B. We get

  11010101B ⊕ 00001111B = 11011010B

  D5H ⊕ 0FH = DA H

  (AL) = 11011010B = DAH

Note that this operation complements the logic state of those bits in AL that are 1s in the immediate operand.

- The last instruction, NOT AL, inverts each bit of AL. Therefore, the final contents of AL become

  (AL) = 11011010B = 00100101B = 25H

Figure below summarizes these results.

Instruction	(AL) binary	(AL) Hex
MOV AL,01010101B	0101 0101	55
AND AL,00011111B	0001 0101	15
OR AL,11000000B	1101 0101	D5
XOR AL,00O01111B	1101 1010	DA
NOT AL	0010 0101	25

Resetting (clearing) and Setting specific bits of an operand:

We may want to reset or set a specific bit without affecting the other bits. **AND** logic can be used to reset the bit and **OR** logic can be used to set the bit.

- **AND Logic:**

A common use of logic instructions is to mask a group of bits of a byte or word data. By mask, we mean to clear the bit or bits to zero. Remember that when a bit is ANDed with another bit that is at logic 0, the result is always 0. On the other hand. If bit is ANDed with a bit that is at logic 1, its value remains unchanged. Thus we see that bits that are to be masked must be set to 0 in the mask, which is the source operand, and those that are to remain unchanged are set to 1.

Ex:

AND AX, 000FH

The mask equals 000FH; therefore, it would mask off the upper l2 bits of the word of data destination AX. Let us assume that the original value in AX is FFFF H. Then executing the instruction performs the operation

0000 0000 0000 1111 B . 1111 1111 1111 1111 B = 0000 0000 0000 1111 B

000F H . FFFF H = 000F H

(AX)= 000F H

This shows that just the lower 4 bits in AX remain intact.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	X	X	X	X

- **OR Logic:**

The OR instruction can be used to set a bit or bits in a register or a storage location in memory to logic 1. If a bit is ORed with another bit that is 0, the value of the bit remains unchanged; however, if it is ORed with another bit that is 1, the bit becomes 1.

Ex: let us assume that we want to set bit D4 of the byte at the offset address [FLAGS] in the current data segment of memory to logic 1. This can be done with the following instruction sequence:

MOV AL, [FLAGS]

OR AL, 10H

MOV [FLAGS] , AL

First the value of the flags are copied into AL and the logic operation

(AL) = XXXX XXXX B + 000**1** 0000 B = XXX**1** XXXX B

Is performed. Finally, the new byte in AL, which has bit D4 set to 1, is written back to the memory location called FLAGS.

D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	X	X	X	X
0	0	0	1	0	0	0	0
X	X	X	1	X	X	X	X

Q/ write an instruction that will mask off all but bit 7 of the word of data stored at address DS:0100 H

Q/ write an instruction that will toggle the logic level of the most significant bit (D15) of the value in the upper byte of the accumulator register.