Faculty of Engineering
Computer Engineering Department
Islamic University of Gaza

Assembly Language Lab # 6
Boolean Instruction and conditional structured

Eng. Alaa.I.Haniya

Assembly Language Fundamentals

Objective:

To know more about Assembly Language Boolean Instructions and Conditional Structured.

❖ Boolean Instructions:

1. NOT Instruction:

The NOT instruction toggles (inverts) all bits in an operand. The result is called the **one's complement.**

• Syntax:

```
NOT reg
NOT mem
```

NOT

x	¬x
F	T
T	F

• Example:

```
mov al, 00111011b
not al
```

NOT 00111011
11000100 —— inverted

Flags: No flags are affected by the NOT instruction.

---

2. AND Instruction:

The AND instruction performs a Boolean (bitwise) AND operation between each pair of matching bits in two operands and places the result in the destination operand.

• Syntax:

```
AND destination, source
```

The following operand combinations are permitted:

```
AND reg,reg
AND reg,mem
AND reg,imm
AND mem,reg
AND mem,imm
```

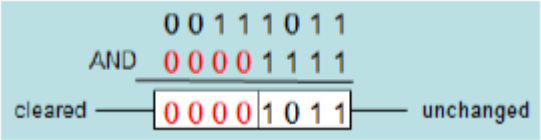
AND

x	y	x ∧ y
0	0	0
0	1	0
1	0	0
1	1	1

The operands can be 8, 16, or 32 bits, and they must be the same size.

- Example:

```
mov al, 00111011b
and al, 00001111b
```



❖ *Applications:*

1.Task: Convert the character in AL to upper case.
Solution: Use the AND instruction to clear bit 5.

```
mov al, 'a'           ; al = 0110 0001b
and al, 11011111b ;al = 0100 0001b
```

2.Task: Jump to a label if an integer is even.
Solution: AND the lowest bit with a 1. If the result is Zero, the number was even.

```
mov ax,wordVal
and ax,1              ; low bit set?
jz EvenValue          ; jump if Zero flag
set
```

3.Task: Convert an ASCII digit to binary?
Solution: Use the AND instruction to clear bits 4 to 7

```
mov al,36h      ;al = 0011 0110b
and al,0Fh      ;al = 0000 0110b = 6
```

3. OR Instruction:

The OR instruction performs a Boolean OR operation between each pair of matching bits in two operands and places the result in the destination operand.

- Syntax:

OR destination, source

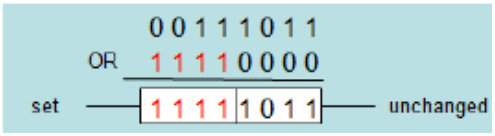
- The operand combinations are same as AND.
- The operands can be 8, 16, or 32 bits, and they must be the same size.

OR

x	y	x ∨ y
0	0	0
0	1	1
1	0	1
1	1	1

• Example:

```
mov dl, 00111011b
or dl, 11110000b
```



❖ *Applications:*

1.**Task**: Convert the character in AL to lower case.
Solution: Use the OR instruction to set bit 5.

```
mov al, 'A'          ;al = 0100 0001b
or al, 00100000b  ;al = 0110 0001b
```

2.**Task**: Convert a binary decimal byte into its equivalent ASCII decimal digit.
Solution: Use the OR instruction to set bits 4 and 5.

```
mov al,6 ;al = 0000 0110b
or al,00110000b     ;al = 0011 0110b
```

3.Task: Jump to a label if the value in AL is not zero.
Solution: OR the byte with itself, then use the JNZ (jump if not zero) instruction.

```
or al,al
jnz IsNotZero     ; jump if not zero
```

4. XOR Instruction:

XOR destination, source
The XOR instruction uses the same operand combinations and sizes as the AND and OR instructions.

• Example:

```
mov dl, 00111011b
xor dl, 11110000b
```

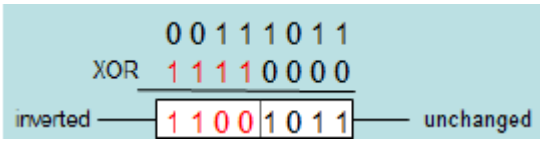
XOR

x	y	x ⊕ y
0	0	0
0	1	1
1	0	1
1	1	0

The XOR instruction performs a Boolean exclusive-OR operation between each pair of matching bits in two operands and stores the result in the destination operand.

• **Example**:

```
mov dl, 00111011b
xor dl, 11110000b
```



❖ *Application:*

1.Task: Reverse the case (Convert upper case to lower case and convert lower case to upper case)
Solution: Use the XOR instruction with 00100000b

```
mov al, 'A'          ;al = 0100 0001b
xor al, 00100000b ;al = 0110 0001b = 'a'
xor al, 00100000b ;al = 0100 0001b = 'A'
```

5. Test Instruction:

Performs a nondestructive AND operation between each pair of matching bits in two operands. No operands are modified, but the flags are affected.

• **Example**: jump to a label if either bit 0 or bit 1 in AL is set.

```
test al,00000011b
jnz ValueFound
```

• **Example**: jump to a label if neither bit 0 nor bit 1 in AL is set.

```
test al,00000011b
jz ValueNotFound
```

The six status flags are affected

- **Carry Flag:** Cleared by AND, OR, XOR and Test
- **Overflow Flag**: Cleared by AND, OR, XOR and Test
- **Sign Flag:** Copy of the sign bit in result
- **Zero Flag**: Set when result is zero
- **Parity Flag:** Set when parity in least-significant byte is even
- **Auxiliary Flag:** Undefined by AND, OR, XOR and Test

6. CMP Instruction:

- Compares the destination operand to the source operand.
- CMP (Compare) instruction performs a subtraction.

Syntax:

CMP destination, source

Computes :

- destination – source
- Destination operand is NOT modified.
- To check for equality, it is enough to check ZF flag

Flags: All six flags: OF, CF, SF, ZF, AF, and PF are affected.

7. BT (Bit Test) Instruction:

Copies bit n from an operand into the Carry flag

Syntax

BT bitBase, n

Example (jump to label L1 if bit 9 is set in the AX register)

```
bt AX,9 ; CF = bit 9
jc L1 ; jump if Carry set
```

8.Conditional structures:

Jcond Instruction:

A conditional jump instruction branches to a label when specific register or flag conditions are met.

• Examples:

- **JC** jump to a label if the Carry flag is set.
- **JE, JZ** jump to a label if the Zero flag is set.
- **JS** jumps to a label if the Sign flag is set.
- **JNE, JNZ** jump to a label if the Zero flag is clear.
- **JECXZ** jumps to a label if ECX equals 0.

The conditional jump instructions are divided into four groups:

1. Jump Based on flags values

Mnemonic	Description	Flags / Registers
JZ	Jump if zero	ZF = 1
JNZ	Jump if not zero	ZF = 0
JC	Jump if carry	CF = 1
JNC	Jump if not carry	CF = 0
JO	Jump if overflow	OF = 1
JNO	Jump if not overflow	OF = 0
JS	Jump if signed	SF = 1
JNS	Jump if not signed	SF = 0
JP	Jump if parity (even)	PF = 1
JNP	Jump if not parity (odd)	PF = 0

2. Jump Based on Equality and value of CX:

Mnemonic	Description
JE	Jump if equal (*leftOp* = *rightOp*)
JNE	Jump if not equal (*leftOp* ≠ *rightOp*)
JCXZ	Jump if CX = 0
JECXZ	Jump if ECX = 0

3.Jump Based on unsigned comparison:

Mnemonic	Description
JA	Jump if above (if *leftOp* > *rightOp*)
JNBE	Jump if not below or equal (same as JA)
JAE	Jump if above or equal (if *leftOp* ≥ *rightOp*)
JNB	Jump if not below (same as JAE)
JB	Jump if below (if *leftOp* < *rightOp*)
JNAE	Jump if not above or equal (same as JB)
JBE	Jump if below or equal (if *leftOp* ≤ *rightOp*)
JNA	Jump if not above (same as JBE)

4.Jump Based on signed comparison:

Mnemonic	Description
JG	Jump if greater (if *leftOp* > *rightOp*)
JNLE	Jump if not less than or equal (same as JG)
JGE	Jump if greater than or equal (if *leftOp* ≥ *rightOp*)
JNL	Jump if not less (same as JGE)
JL	Jump if less (if *leftOp* < *rightOp*)
JNGE	Jump if not greater than or equal (same as JL)
JLE	Jump if less than or equal (if *leftOp* ≤ *rightOp*)
JNG	Jump if not greater (same as JLE)

Notes:

Assembly language programmers can easily translate logical statements written in C++/Java into assembly language. For example:

```
if( op1 = = op2 )
  X = 1;
else
  X = 2;
```

```
        mov eax,op1
        cmp eax,op2
        jne L1
        mov X,1
        jmp L2
L1:     mov X,2
L2:
```

- Implement the following pseudocode in assembly language. All values are 32- bit signed integers:

```
if( var1 <= var2 )
  var3 = 10;
else
{
  var3 = 6;
  var4 = 7;
}
```

```
        mov eax,var1
        cmp eax,var2
        jle L1
        mov var3,6
        mov var4,7
        jmp L2
L1:     mov var3,10
L2:
```

Compound expression with AND:

```
if (al > bl) AND (bl > cl)
      X = 1;
```

This is one possible implementation . . .

```
        cmp al,bl       ; first expression...
        ja  L1
        jmp next
L1:
        cmp bl,cl       ; second expression...
        ja  L2
        jmp next
L2:             ; both are true
        mov X,1         ; set X to 1
next:
```

Compound expression with OR:

```
if (al > bl) OR (bl > cl)
  X = 1;
```

```
        cmp al,bl       ; is AL > BL?
        ja  L1   ; yes
        cmp bl,cl       ; no: is BL > CL?
        jbe next        ; no: skip next statement
L1:     mov X,1         ; set X to 1
next:
```

Lab work:
Excercise1: Boolean Calculator

Create a program that functions as a simple Boolean calculator for 32-bit integers.
The program Do the following functions:

1. x AND y	2. x OR y	3. x XOR y	4. NOT x

Solution:

```
   .model small
   .386
   .stack 100h
 .data
x     dd 0ffffh
y     dd 1111h
msg db "Enter 1:For and",0ah,0dh,
       "Enter 2:For or",0ah,0dh,
       "Enter 3:For xor",0ah,0dh,
       "Enter 4:For not",0ah,0dh,'$'

err_msg db 0ah,0dh,"Invalid Operartion",0ah,0dh,'$'
.code
main:
   mov ax,@data
   mov ds,ax

   mov ah,09h
   mov dx,offset msg
   int 21h

   mov ah,01h    ;to input operation type
   int 21h

   mov edx,x
   and al,0fh
   cmp al,1
   je L1
   cmp al,2
   je L2
   cmp al,3
   je L3
   cmp al,4
   je L4
```

```
    mov ah,09h
    mov dx,offset err_msg
    int 21h
    jmp L

    L1:
        and edx,y
        jmp L
    L2:
        or  edx,y
        jmp L
    L3:
        xor edx,y
        jmp L
    L4:
        not edx


    L:
        mov ah,4ch
        int 21h
end main
```

Note:
The number you entered will be stored in al using this service:
Service 01h: DOS get character function

```
mov ah,01h ; returns ASCII code of character to AL
int 21h
```

Excercise2:

Write an assembly language that compares the values x and y in memory and prints a message if the value in x is greater than y, less than y, or equal to y.

```
  .386
  .stack 100h
.data
x     db 25
y     db 30
msg1  db 'x is greater than y','$'
msg2  db 'x is equal to y','$'
msg3  db 'x is less than y','$'
new_line db 0ah,0dh,'$'
.code
main:
   mov ax,@data
   mov ds,ax
;---- user enter x
   mov ah,01
   int 21h
   and al,0fh
   mov x,al
```

```
;---- user enter y
   mov ah,01
   int 21h

   mov y,al

 ;--- compare
   mov bl,x
   mov dl,y

   cmp bl,dl
   jg  L1
   je  L2

   mov dx,offset msg3
   jmp L

   L1:
      mov dx,offset msg1
      jmp L

   L2:
      mov dx,offset msg2

   L:
      mov ah,09h
      int 21h

   mov ah,4ch
   int 21h
end main
```

Excercise3:

Write an assembly language program that allow user to input one digit number and determine if it is even or odd.

```
   .model small
   .386
   .stack 100h
  .data
msg1 db 'Enter a number: ','$'
msg2 db 'The number is odd','$'
msg3 db 'The number is even','$'
newLine db 0Ah,0Dh,'$'
  .code
main:
  mov ax,@data
  mov ds,ax
```

```
  ;---- print enter number
  mov ah,09h
  mov dx,offset msg1
  int 21h

  ;-- read from user
  mov ah,01h
  int 21h

 and al,0fh

 ;------compare ever by and 01h
  and al,01h
  jz EvenNum
  ;-----
    mov ah,09h
    mov dx,offset newLine
    int 21h

    mov dx,offset msg2
    int 21h
  jmp exit

EvenNum:
    mov ah,09h
    mov dx,offset newLine
    int 21h

   mov dx,offset msg3
   int 21h
exit:
  mov ah,4ch
  int 21h
end main
```

Homework:

1. Write an assembly code that output a letter grade for 10 numbered grades according to the following table:

Numbered Grade	Letter Grade
90-100	A
80-90	B
70-80	C
60-70	D
0-59	F
other	Not Valid

The grades are 74, 109, 91, 86, 40, 76, 72, -6, 65, 94. C,I,A,B,F,C,C,I,D,A

---

2. Write an assembly code to read a character from console and echo it.

---

3. Write an assembly language program that allow the user to input a character, if it is small letter; convert it to capital and print it, otherwise print it as the same as it was entered.