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:
  
  \[
  \text{NOT} \ \text{reg} \\
  \text{NOT} \ \text{mem}
  \]

- Example:
  
  \[
  \text{mov} \ al, 00111011b \\
  \text{not} \ al
  \]

  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:
  
  \[
  \text{AND} \ \text{destination}, \ \text{source}
  \]

  The following operand combinations are permitted:

  \[
  \begin{array}{ccc}
  \text{AND} & \text{reg}, & \text{reg} \\
  \text{AND} & \text{reg}, & \text{mem} \\
  \text{AND} & \text{reg}, & \text{imm} \\
  \text{AND} & \text{mem}, & \text{reg} \\
  \text{AND} & \text{mem}, & \text{imm}
  \end{array}
  \]
The operands can be 8, 16, or 32 bits, and they must be the same size.

- Example:

  | mov al, 00110111b |
  | 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.
• Example:

\[
\begin{align*}
\text{mov } dl, &00111011b \\
\text{or } dl, &11110000b
\end{align*}
\]

❖ Applications:

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

\[
\begin{align*}
\text{mov } al, &'A' ;al = 0100 0001b \\
\text{or } al, &00100000b ;al = 0110 0001b
\end{align*}
\]

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

\[
\begin{align*}
\text{mov } al,6 ;al = 0000 0110b \\
\text{or } al, &00110000b ;al = 0011 0110b
\end{align*}
\]

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.

\[
\begin{align*}
\text{or } al, al \\
\text{jnz } \text{IsNotZero} ; \text{jump if not zero}
\end{align*}
\]

4. **XOR Instruction**:

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

• Example:

\[
\begin{align*}
\text{mov } dl, &00111011b \\
\text{xor } dl, &11110000b
\end{align*}
\]

\[
\begin{array}{c|c|c|c|c}
  x & y & x \oplus y \\
\hline
  0 & 0 & 0 \\
  0 & 1 & 1 \\
  1 & 0 & 1 \\
  1 & 1 & 0 \\
\end{array}
\]
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
  ```

  ![XOR Instruction Example](image)

  - 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)
```
between
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

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

2. Jump Based on Equality and value of CX:

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JE</td>
<td>Jump if equal ((leftOp = rightOp))</td>
</tr>
<tr>
<td>JNE</td>
<td>Jump if not equal ((leftOp \neq rightOp))</td>
</tr>
<tr>
<td>JCXZ</td>
<td>Jump if CX = 0</td>
</tr>
<tr>
<td>JECXZ</td>
<td>Jump if ECX = 0</td>
</tr>
</tbody>
</table>

3. Jump Based on unsigned comparison:

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JA</td>
<td>Jump if above ((leftOp &gt; rightOp))</td>
</tr>
<tr>
<td>JNBE</td>
<td>Jump if not below or equal ((same as JA))</td>
</tr>
<tr>
<td>JAE</td>
<td>Jump if above or equal ((leftOp \geq rightOp))</td>
</tr>
<tr>
<td>JNB</td>
<td>Jump if not below ((same as JAE))</td>
</tr>
<tr>
<td>JB</td>
<td>Jump if below ((leftOp &lt; rightOp))</td>
</tr>
<tr>
<td>JNAE</td>
<td>Jump if not above or equal ((same as JB))</td>
</tr>
<tr>
<td>JBE</td>
<td>Jump if below or equal ((leftOp \leq rightOp))</td>
</tr>
<tr>
<td>JNA</td>
<td>Jump if not above ((same as JBE))</td>
</tr>
</tbody>
</table>
4. Jump Based on signed comparison:

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JG</td>
<td>Jump if greater (if leftOp &gt; rightOp)</td>
</tr>
<tr>
<td>JNLE</td>
<td>Jump if not less than or equal (same as JG)</td>
</tr>
<tr>
<td>JGE</td>
<td>Jump if greater than or equal (if leftOp ≥ rightOp)</td>
</tr>
<tr>
<td>JNL</td>
<td>Jump if not less (same as JGE)</td>
</tr>
<tr>
<td>JL</td>
<td>Jump if less (if leftOp &lt; rightOp)</td>
</tr>
<tr>
<td>JNGE</td>
<td>Jump if not greater than or equal (same as JL)</td>
</tr>
<tr>
<td>JLE</td>
<td>Jump if less than or equal (if leftOp ≤ rightOp)</td>
</tr>
<tr>
<td>JNG</td>
<td>Jump if not greater (same as JLE)</td>
</tr>
</tbody>
</table>

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

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

```plaintext
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:

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

```asm
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:

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

This is one possible implementation . . .

```asm
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:
```
Lab work:  
**Exercise 1: Boolean Calculator**

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

1. \( x \text{ AND } y \)  
2. \( x \text{ OR } y \)  
3. \( x \text{ XOR } y \)  
4. \( \text{NOT } x \)

**Solution:**

```assembly
.model small  
.386  
.stack 100h  
data
x dd Offffh  
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, 0dh  
err_msg db 0ah, 0dh, "Invalid Operation", 0ah, 0dh, 0dh, 0dh
.code
main:  
mov ax, @data  
mov db, 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
```

```
if (al > bl) OR (bl > cl)
X = 1;
```
Note:
The number you entered will be stored in al using this service:
Service 01h: DOS get character function

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

Exercise 2:

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.

```assembly
.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
    cmp x,y
    jg msg1
    jmp msg3
    cmp x,y
    jl msg2
    jmp msg3
    mov al,0
    int 21h
    mov al,1
    int 21h
```

Exercise 3:

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

```assembly
mov ah,0: int 21h
mov al,0

; --- compare
mov bx,ax
mov dl,al

cmp bx,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
```

```assembly
.model small
.386
.stack 100h
.data
msg1 db 'Enter a number: ',0,'$'
msg2 db 'The number is odd',0,'$'
msg3 db 'The number is even',0,'$'
newLine db 0Ah,0Dh,'$'
.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 outputs a letter grade for 10 numbered grades according to the following table:

<table>
<thead>
<tr>
<th>Numbered Grade</th>
<th>Letter Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>90-100</td>
<td>A</td>
</tr>
<tr>
<td>80-90</td>
<td>B</td>
</tr>
<tr>
<td>70-80</td>
<td>C</td>
</tr>
<tr>
<td>60-70</td>
<td>D</td>
</tr>
<tr>
<td>0-59</td>
<td>F</td>
</tr>
<tr>
<td>other</td>
<td>Not Valid</td>
</tr>
</tbody>
</table>

The grades are 74, 109, 91, 86, 40, 76, 72, -6, 65, 94.

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

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