Computer Organization and Assembly Language

Lecture 6 - Conditional Processing

What Are Booleans?

- Boolean values are either ***True*** or ***False.***
- These are usually represented by ***1*** for True and ***0*** for False.
- The most common Boolean operations are
  - AND
  - OR
  - XOR
  - NOT

Boolean and Comparison Instructions

- Using the conditional instructions to conditional loops and if-then–else structures requires an understanding of the flags registers.
- The flags register is affected by most instruction as a byproduct of the operation.
  - There are some instruction whose whole purpose is to change the flags register.
  - These include **CMP**, **AND**, **OR**, **XOR**, **NOT**, and **NEG**.

The Flags Register

- The Flags Register contain four flags of particular interest:
  - ***Zero flag*** (set when the result of an operation is zero).
  - ***Carry flag*** (set when the result of unsigned arithmetic is too large for the destination operand or when subtraction requires a borrow).
  - ***Sign flag*** (set when the high bit of the destination operand is set indicating a negative result).
  - ***Overflow flag*** (set when signed arithmetic generates a result which ifs out of range).

AND Operation

x	y	x ∧y
0	0	0
0	1	0
1	0	0
1	1	1

AND Instruction

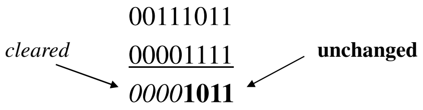
- The **AND** instruction performs a bit wise AND operation between corresponding bits in the two operands and places the result in the first operand.
- The format for the **AND** instruction is:

```
AND     reg, reg
AND     reg, mem
AND     reg, immed
AND     mem, reg
AND     mem, immed
```

reg, *mem*, and *immed* can be 8, 16, or 32 bits.

AND Instruction (continued)

- An example of ANDing:



00111011
<u>00001111</u>
*0000***1011**

cleared

unchanged

- The AND instruction can be used to clear selected bits in an operand while preserving the remaining bits. This is called ***masking***.

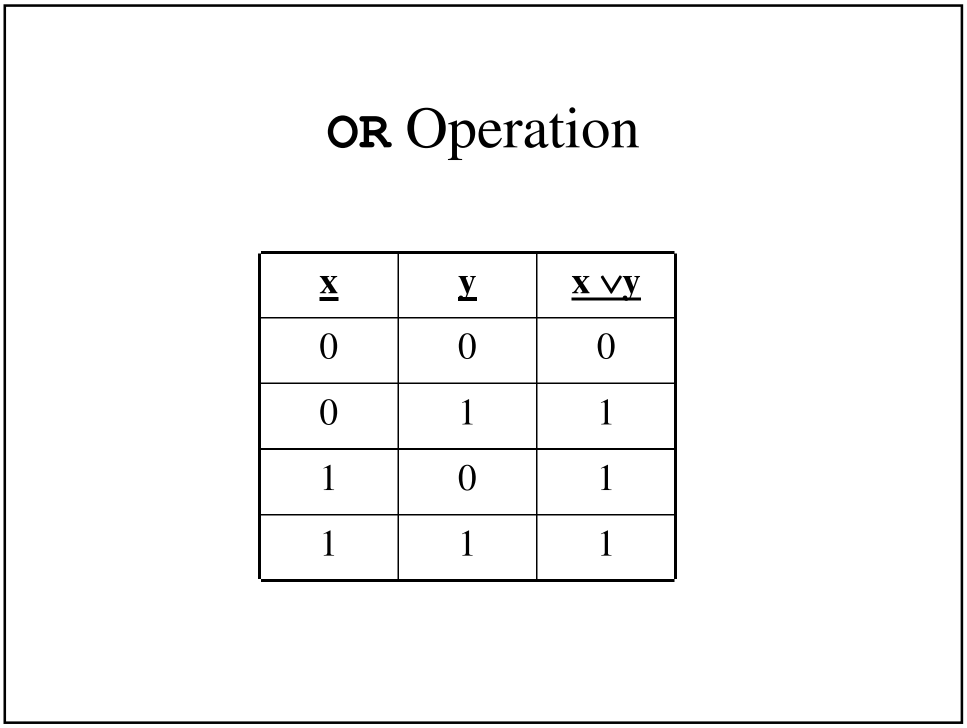
```
mov     al, 00111011b
and     al, 00001111b   ; AL = 00001011b
```

Converting Characters to Upper Case

- We convert lower case to upper case by clearing bit 5:

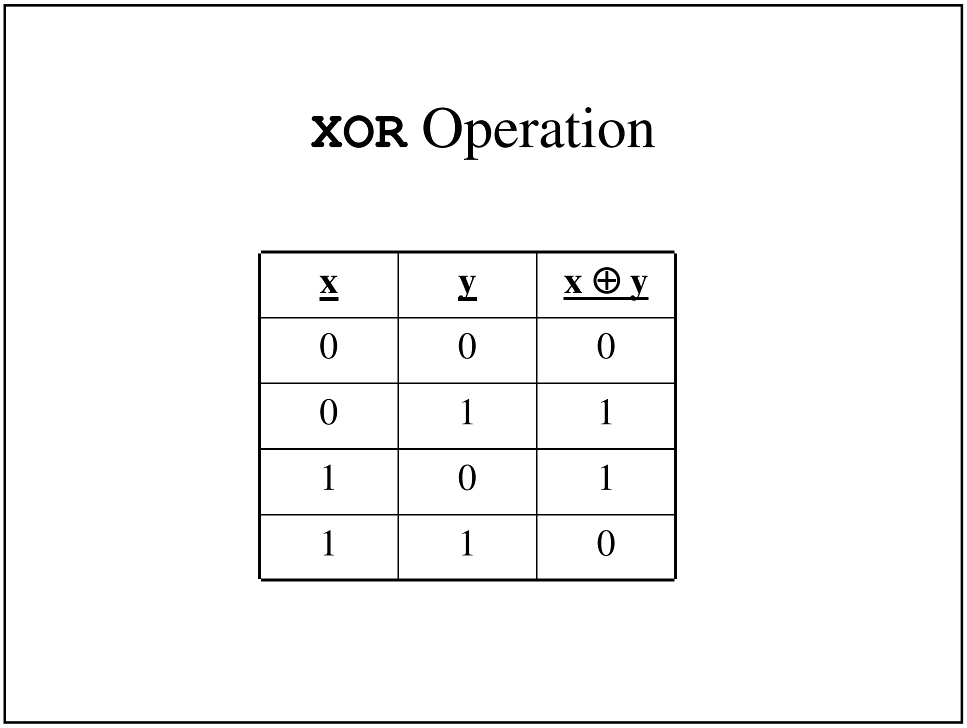
```
0  1  1  0  0  0  0  1  {'a' }
0  1  0  0  0  0  0  1  {'A' }
```

```
.data
array BYTE 50 DUP(?)
.code
      mov   ecx, LENGTHOF array
      mov   esi, OFFSET array
L1:
      and   byte ptr [esi], 11011111b
      inc   esi
      loop  L1
```



OR Operation

x	y	x ∨y
0	0	0
0	1	1
1	0	1
1	1	1



XOR Operation

x	y	x ⊕ y
0	0	0
0	1	1
1	0	1
1	1	0

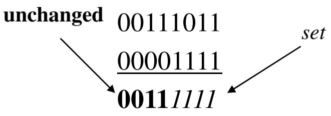
OR Instruction

- The **`OR`** instruction performs a bit wise OR operation between corresponding bits in the two operands and places the result in the first operand.
- The format for the **`OR`** instruction is:

```
OR      reg, reg
OR      reg, mem
OR      reg, immed
OR      mem, reg
OR      mem, immed
reg, mem, and immed can be 8, 16, or 32
  bits.
```

OR Instruction (continued)

- An example of ORing:



- The OR instruction can be used to set selected bits in an operand while preserving the remaining bits.

```
mov     al, 00111011b
or      al, 00001111b  ; AL = 001111111b
```

OR: Some Examples

- OR can be used to convert a one-digit value into its ASCII equivalent:

```
mov     dl, 5      ; binary value
or      dl, 30h    ; convert to ASCII
```

- ORing a value with itself preserves the value but sets to flags
  - ZF = 1 if AL = 0
  - SF=1 if AL < 0
  - SF = ZF = 0 if AL > 0

```
or al, al    ; sets the flags
```

XOR Operation

x	**y**	**x ⊕ y**	**(x ⊕ y) ⊕ y**
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

XOR Instruction

- The **XOR** instruction performs a bit wise Exclusive OR operation between corresponding bits in the two operands and places the result in the first operand.
- The format for the **XOR** instruction is:

```
XOR      reg, reg
XOR      reg, mem
XOR      reg, immed
XOR      mem, reg
XOR      mem, immed
reg, mem, and immed can be 8, 16, or 32
  bits.
```

XOR Instruction (continued)

- An example of XORing:

00111011

unchanged 00111111 *inverted*

00*000100*

- The XOR instruction can be used to reverse selected bits in an operand while preserving the remaining bits.

```
mov     al, 00111011b
and     al, 00001111b   ; AL = 00110100b
```

XOR Example: Checking the Parity Flag

- Parity flag indicates whether the lowest order byte of the result an arithmetic or bit wise operation has an even or odd number of 1s.
- Flag = 1 if parity is even; Flag = 0 if parity is odd.
- We want to find the parity of a number without changing its value:

```
mov al, 10110101b     ; 5 bits = odd parity
xor al, 0             ; Parity flag clear (PO)
mov al, 11001100b     ; 4 bits = even parity
xor al, 0             ; Parity flag set (PE)
```

XOR Example: 16-Bit Parity Flag

- You can check the parity of a 16-bit register by performing an exclusive-OR between the upper and lower bytes:

```
mov ax, 64C1h   ; 0110 0100 1100 0001
xor ah, al      ; Parity flag set (PE)
```

AND, **OR**, **XOR** and the Status Flags

- All three instructions affect the following flags, with the result determining their actual values:
  - Overflow
  - Sign
  - Zero
  - Parity
  - Carry
  - Auxiliary Carry

NOT Instruction

- The NOT instruction reverse all bits in an operand:

```
NOT     reg
NOT     mem
```

- Example:

```
mov    al, 11110000b
not    al                 ; AL = 0Fh
```

TEST Instruction

- The **TEST** instruction performs an implied AND operation between corresponding bits in the two operands and sets the flags without modifying either operand.
- The format for the **TEST** instruction is:

```
TEST    reg, reg
TEST    reg, mem
TEST    reg, immed
TEST    mem, reg
TEST    mem, immed
reg, mem, and immed can be 8, 16, or 32
  bits.
```

TEST Instruction: Examples

- The TEST instruciton can check several bits at once.
- If we wanted to know if either bit 0 or bit 3 is set in the AL register, we can use

```
test      al, 00001001b     ; test bits 0 and 3
```

```
0  0  1  0  0  1  0  1  ←    input value
0  0  0  0  1  1  0  1  ←    test value
0  0  0  0  0  0  0  1  ←    result: ZF = 0

0  0  1  0  0  0  1  0  ←    input value
0  0  0  0  1  1  0  1  ←    test value
0  0  0  0  0  0  0  0  ←    result: ZF = 1
```

`CMP` Instruction

- The CMP instruction sets the flags ***as if*** it had performed subtraction on the operand.
- Neither operand is changed.
- The CMP instruction takes the forms:

```
CMP reg, reg        CMP mem, reg
CMP reg, mem        CMP mem, immed
CMP reg, immed
```

`CMP` Results

CMP Results	**ZF**	**CF**
destination < source	0	1
destination > source	0	0
destination = source	1	0

CMP Results

CMP Results	**Flags**
destination < source	SF ≠ OF
destination > source	SF = OF
destination = source	ZF = 1

CMP Instruction : Examples

- Subtracting 5-10 requires a borrow:

```
mov     ax, 5
cmp     ax, 10    ; CF = 1
```

- Subtracting 1000 from 1000 results in zero.

```
mov     ax, 1000
mov     cx, 1000
cmp     cx, ax    ; ZF = 1
```

- Subtracting 0 from 105 produces a positive difference:

```
mov     si, 105
cmp     si, 0;    ZF = 0 and CF = 0
```

Setting & Clearing Individual Flags

- Setting and Clearing the Zero Flag

```
and    al, 0    ; Set Zero Flag
or     al, 1    ; Clear Zero Flag
```

- Setting and Clearing the Sign Flag

```
or     al, 80h  ; Set Sign Flag
and    al, 7fh  ; Clear Sign Flag
```

Setting & Clearing Individual Flags

- Setting and Clearing the Carry Flag

```
 stc        ; Set Carry Flag
 clc        ; Clear Carry Flag
```

- Setting and Clearing the Overflow Flag

```
mov    al, 7fH  ; AL = +127
inc    al       ; AL = 80H; OF = 1
or     eax, 0   ; Clear Overflow
                ; Flag
```

Conditional Structures – An Example

- Compare AL to Zero. Jump to L1if the zero flag was set by the comparison:

```
    cmp  al, 0
    jz       L1
    … …
L1:
```

Conditional Structures – Another Example

- Perform a bitwise AND on the DL register . Jump to L2 if the Zero flag is clear:

```
    and      dl, 10110000b
    jnz      L2
    … …
L2:
```

J*cond* Instruction

- A conditional jump instruction branches to a destination label when a flag condition is true.
- If the flag is false, the instruction immediately following the conditional jump is performed instead.
- The syntax is:

  J*cond destination*

Limitations of Conditional Jumps

- Microsoft Macro assembler limits jumps to a label within the current procedure and within –128 to +127 of the current address.
- To jump to another procedure, you must use a global label:

```
    jc   MyLabel   ; Jump if Carry
                   ; (flag is set)
  … …
MyLabel::
```

Examples of Conditional Jumps

- In all three cases, the jump is made:

```
mov   ax, 5
cmp   ax, 5
je    L1    ; jump if equal

mov   ax, 5
cmp   ax, 6
jl    L1    ; jump if less

mov   ax, 5
cmp   ax, 4 ; jump if greater
```

Jumps based on General Comparisons

Mnemonic	Description	Flags/Registers
JZ	Jump if zero	ZF = 1
JE	Jump if equal	ZF = 1
JNZ	Jump if not zero	ZF = 0
JNE	Jump if not equal	ZF = 0

Jumps based on General Comparisons

Mnenomic	Description	Flags/Registers
JC	Jump if carry	CF = 1
JNC	Jump if not carry	CF = 0
JCXZ	Jump if CX = 0	CX = 0
JECXZ	Jump if ECX = 0	ECX = 0

Jumps based on General Comparisons

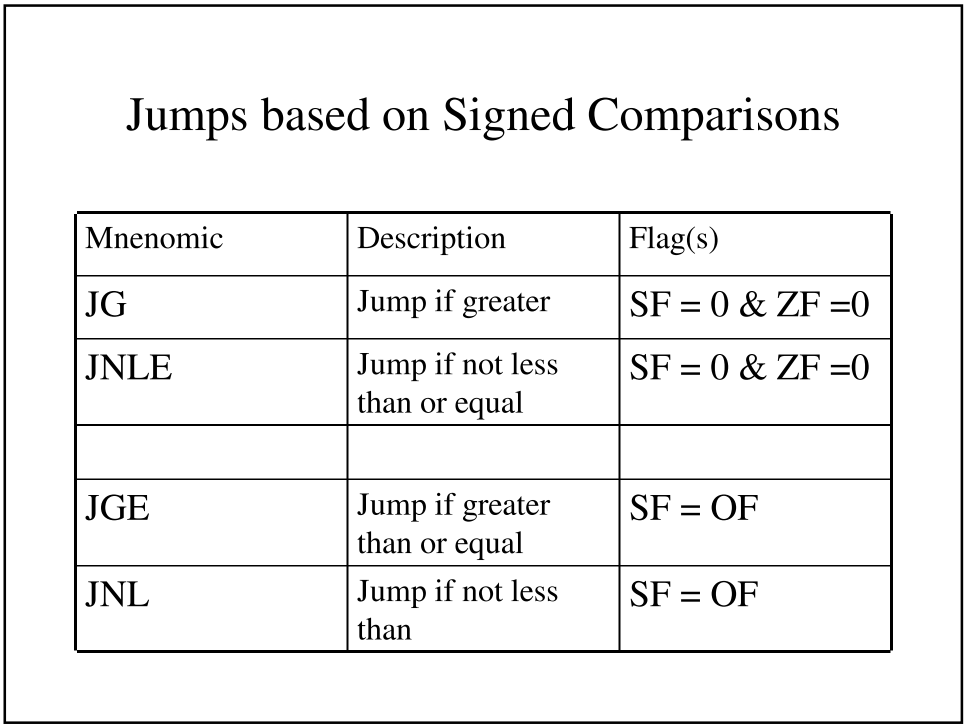
Mnenomic	Description	Flags/Registers
JP	Jump if Parity even	PF = 1
JNP	Jump if Parity odd	PF = 0

Jumps based on Unsigned Comparisons

Mnenomic	Description	Flag(s)
JA	Jump if above (op1 > op2)	CF = 0 & ZF = 0
JNBE	Jump if not below or equal	CF = 0 & ZF = 0
JAE	Jump if above or equal	CF = 0
JNB	Jump if not below	CF = 0

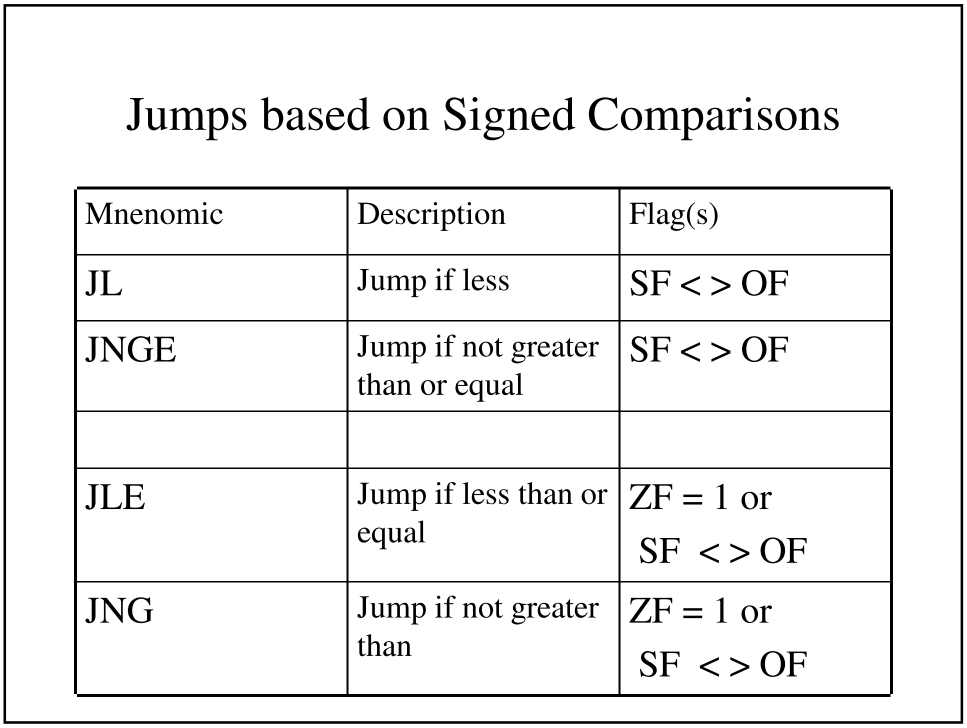
Jumps based on Unsigned Comparisons

Mnenomic	Description	Flag(s)
JB	Jump if below (op1 < op2)	CF = 1
JNAE	Jump if not above	CF = 1
JBE	Jump if below or equal	CF = 1 or ZF = 1
JNA	Jump if not above	CF = 1 or ZF = 1



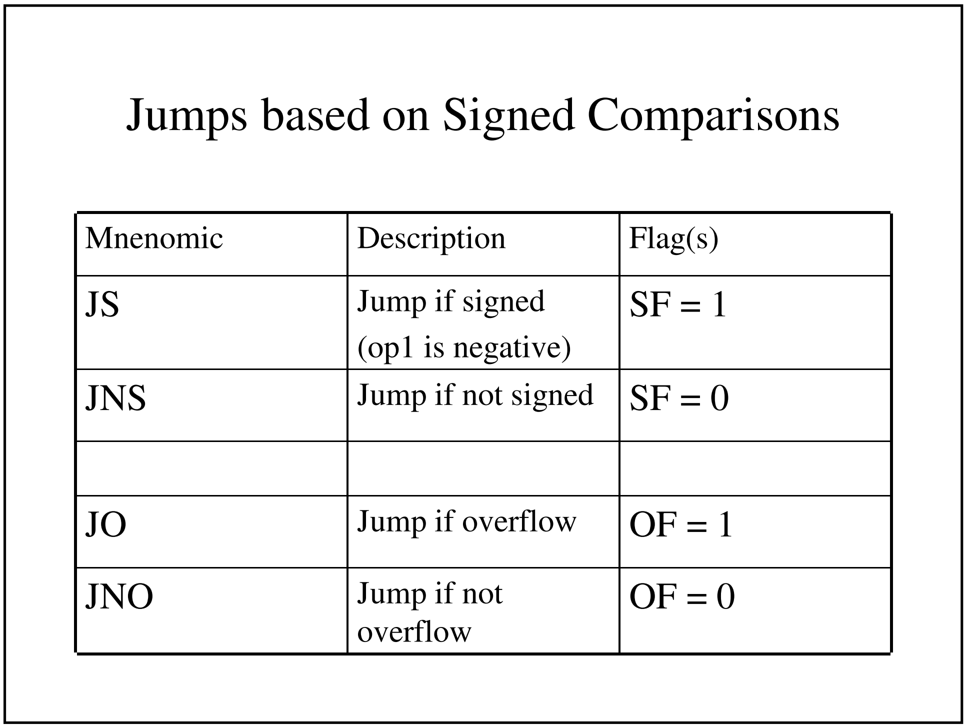
Jumps based on Signed Comparisons

Mnenomic	Description	Flag(s)
JG	Jump if greater	SF = 0 & ZF =0
JNLE	Jump if not less than or equal	SF = 0 & ZF =0
JGE	Jump if greater than or equal	SF = OF
JNL	Jump if not less than	SF = OF



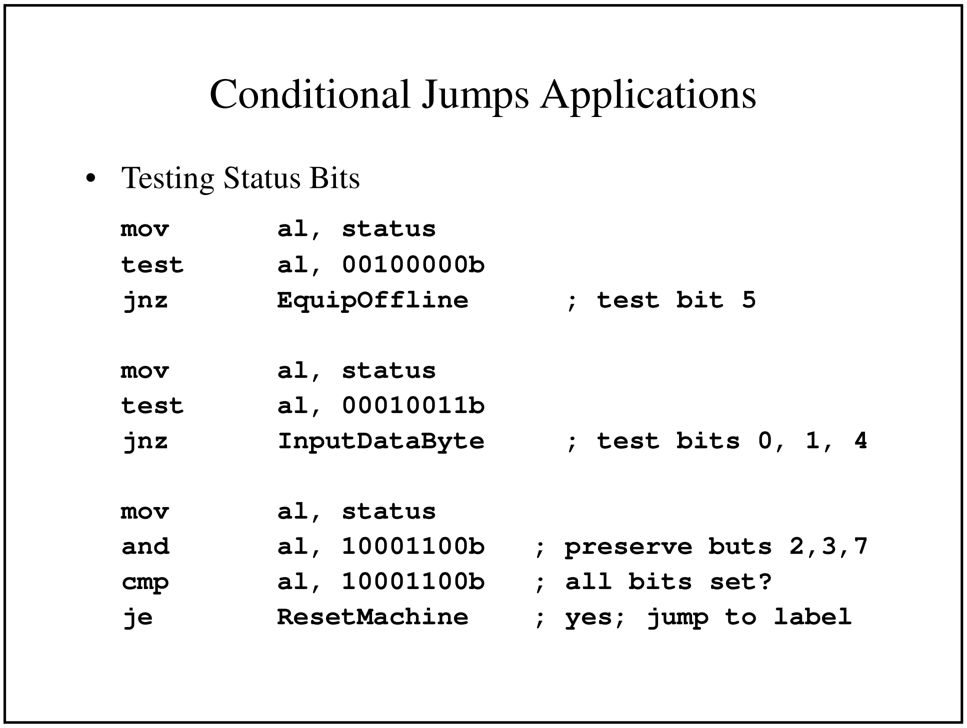
Jumps based on Signed Comparisons

Mnenomic	Description	Flag(s)
JL	Jump if less	SF < > OF
JNGE	Jump if not greater than or equal	SF < > OF
JLE	Jump if less than or equal	ZF = 1 or SF < > OF
JNG	Jump if not greater than	ZF = 1 or SF < > OF



Jumps based on Signed Comparisons

Mnenomic	Description	Flag(s)
JS	Jump if signed (op1 is negative)	SF = 1
JNS	Jump if not signed	SF = 0
JO	Jump if overflow	OF = 1
JNO	Jump if not overflow	OF = 0



Conditional Jumps Applications

- Testing Status Bits

```
mov      al, status
test     al, 00100000b
jnz      EquipOffline      ; test bit 5

mov      al, status
test     al, 00010011b
jnz      InputDataByte     ; test bits 0, 1, 4

mov      al, status
and      al, 10001100b   ; preserve buts 2,3,7
cmp      al, 10001100b   ; all bits set?
je       ResetMachine    ; yes; jump to label
```

Example – Larger of Two Integers

```
     mov    dx, ax      ; assume that AX is larger
     cmp    ax, bx      ; IF AX >= BX then
     jae    L1          ; jump to L1
     mov    dx, bx      ; else move BX to DX
L1:                     ; DX contains the larger
                        ; integer
```

Example – Smallest of Three Integers

```
.data
V1     WORD  ?
V2     WORD  ?
V3     WORD  ?
.code
       mov   ax, V1  ; assume that V1 is smallest
       cmp   ax, V2  ; IF AX <= V2 then
       jbe   L1      ; jump to L1
       mov   ax, V2  ; else move V2 to AX
L1:    cmp   ax, V3  ; if AX <= V3 then
       jbe   L2      ;   jump to L3
       mov   ax, V3  ; else move to V3 to AX
L2:                  ; smallest is in AX
```

Example – Scanning An Array

```
TITLE Scanning an Array (ArryScan.asm)
; Scan an array for the first nonzero value.
INCLUDE     Irvine32.inc

.data
intArray    SWORD 0, 0, 1, 20, 35, -12, 66, 4, 0
noneMsg     BYTE  "A nonzero value wasnt found", 0
.code
main  PROC
      mov   ebx, OFFSET intArray
                              ; point to the array
      mov   ecx, LENGTHOF intArray
                              ; loop counter
```

```
L1:   cmp   word ptr [ebx], 0
      jnz   found             ; found a value
      add   ebx, 2            ; point to next
      loop  L1                ; continue the loop
      jmp   notFound          ; none found

found:
      movsx eax, word ptr [ebx]
      call  WriteInt
      jmp   quit

notFound:         ; display “not found message”
      mov   edx, OFFSET noneMsg
      call  WriteString
```

```
quit: call  CrLf
      exit
main  ENDP
      END   main
```

Example – Encryption Program

```
TITLE Encryption Program

INCLUDE     Irvine32.inc
KEY = 239   ; Any value Between 1-255
BUFMAX = 128; Maximum buffer size

.data
sPrompt           BYTE  "Enter the plain text:
  ", 0
sEncrypt    BYTE  "Cypher text:           ", 0
sDeCrypt    BYTE  "Decrypted:             ", 0
buffer      BYTE  BUFMAX dup(0)
bufSize     DWORD ?
```

```
.code
main  PROC
  call      InputTheString
                         ; input the plain text
  call      TranslateBuffer
                         ; encrypt the buffer
  mov edx, OFFSET sEncrypt
                  ; display encrypted message
  call      DisplayMessage
  call      TranslateBuffer
                         ; decrypt the buffer
  mov edx, OFFSET sDecrypt
                  ; display decrypted message
  call      DisplayMessage
  exit
main  ENDP
```

```
;-------------------------------------------
InputTheString      PROC
;
; Asks the user to enter a string from the
; keyboard.  Saves the string and its length
; in variables
; Receives:  nothing
; Returns:   nothing
;-------------------------------------------
  pushad
  mov edx, OFFSET sPrompt ; display prompt
  callWriteString
  mov ecx, BUFMAX         ; maximum character count
  mov edx, OFFSET buffer  ; point to the buffer
  callReadString          ; input the string
  mov bufsize, eax        ; save the length
  callCrLf
  popad
  ret
InputTheString      ENDP
```

```
;----------------------------------------
DisplayMessage      PROC
;
; Displays the encrypted or decrypted
; message
; in variables
; Receives:  EDX points to the message
; Returns:   nothing
;----------------------------------------
  pushad
  callWriteString
  mov edx, OFFSET buffer  ; display the buffer
  callWriteString
  callCrLF
  callCrLf
  popad
  ret
DisplayMessage      ENDP
```

```
;----------------------------------------
TranslateBuffer     PROC
;
; Translate the sring by exclusive-ORing
; each byte with the same integer
; Receives:  nothing
; Returns:   nothing
;----------------------------------------
  pushad
  mov ecx, bufSize ; loop counter
  mov esi, 0       ; index 0 in buffer
L1:
  xor buffer[esi], KEY; translate a byte
  inc esi          ; point to next byte
  loopL1

  popad
  ret
TranslateBuffer     ENDP
  END main
```

LOOPZ and LOOPE Instructions

- **`LOOPZ`** (Loop if zero) and **`LOOPE`** (Loop if equal) let a loop continue if ZF = 1 & CX > 0 (First CX is decremented)
- The syntax is:

  `LOOPZ` *destination*

  `LOOPE` *destination*

LOOPZ and LOOPE Instructions : Example

- Example

```
.data
intarray WORD  1, 20, 35, 012, 66, 40, 0
ArraySize=($-intarray)/2
.code
  mov    ebx, offset intarray ; point to the array
  sub    ebx, 2              ; back up one position
  mov    ecx, ArraySize      ; repeat 100 times
next:
  add    ebx, 2              ; point to next entry
  cmp    word ptr [ebx], 0   ; compare value to zero
  loopz  next                ; loop while ZF 1, CX > 0
```

LOOPNZ and LOOPNE Instructions

- **`LOOPNZ`** (Loop if not zero) and **`LOOPNE`** (Loop if not equal) let a loop continue if ZF = 1 & CX > 0 (First CX is decremented)
- The syntax is:

  `LOOPZ` *destination*

  `LOOPE` *destination*

LOOPNZ - an Example

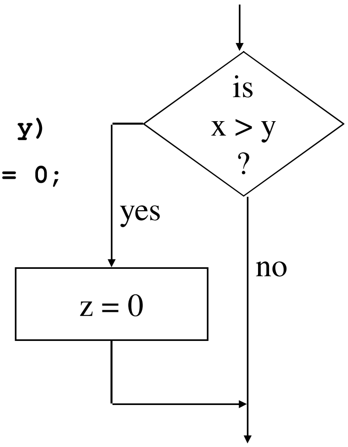
```
INCLUDE     Irvine32.inc

.data
array SWORD -3, -6, -1, -10, 10, 30, 40, 4
Msg   BYTE  " is a positive value", 0
sentine     SWORD 0

.code
main  PROC
  mov esi, OFFSET array
  mov ecx, LENGTHOF     array
```

```
next:
  test      WORD PTR [esi], 8000h ; test sign bit
  pushfd                 ; push flags on stack
  add esi, TYPE array
  popfd                  ; pop flags
  loopnz    next         ; continue loop
  jnz quit               ; none found
  sub esi, TYPE array    ; ESI points to value
quit:
  movzx     eax, word ptr [esi]     ; print value
  call      WriteDec
  mov       edx, OFFSET Msg
  call      WriteString
  exit
main  ENDP
  END main
```

Writing IF-THEN

In C++:

```
if  (x > y)
     z = 0;
```



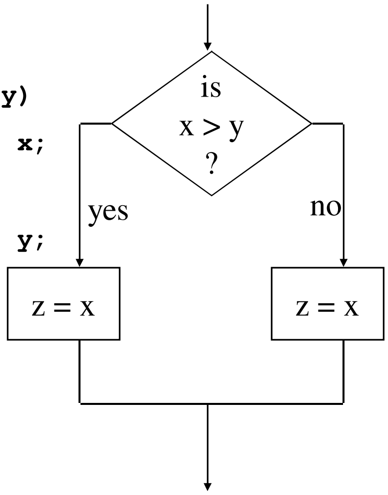
In Assembler

```
      mov ax, x
      cmp ax, y
      jng L1
      mov ax, 0
      mov z, ax
 L1:
```

Writing IF-THEN-ELSE

In C++:

```
if  (x > y)
      z = x;
else
      z = y;
```



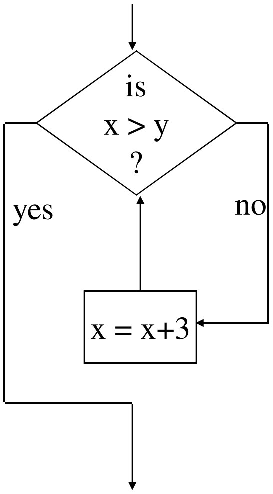
In Assembler

```
      mov ax, x
      cmp ax, y
      jng L1
      mov ax, x
      jmp L2
L1:
      mov ax, y
L2:
      mov z, ax
```

Writing WHILE loops

In C++:

```
while (x <= y)
    x = x + 3;
```



In Assembler

```
L1:
      mov ax, x
      cmp ax, y
      jg L2
      mov ax, x
      add x, 3
      jmp L1
L2:
```