Control Structures internal implementation

Control Structures

- Programs have 4 basic control structures:
  - Sequential
  - Selection
  - Repetition

- Control statements are essential for any programming language

Control structures

Assembly language implementation of

- Selection/Branching (if, if-else, switch-Case)
- Iteration/Loops (while, do-while, for)

Changing Execution Order

- Statement Execution order → determined by EIP
  - Sequential by default
  - can be changed via some instructions called jump instructions.

- 2 types of jumps
  - Unconditional (JMP label)
  - Conditional (Jc label c= condition)
    - Jumps if condition is true

- EIP's Re-definition

The EIP register contains the address of the **next** instruction to be executed if no branching/looping is done.

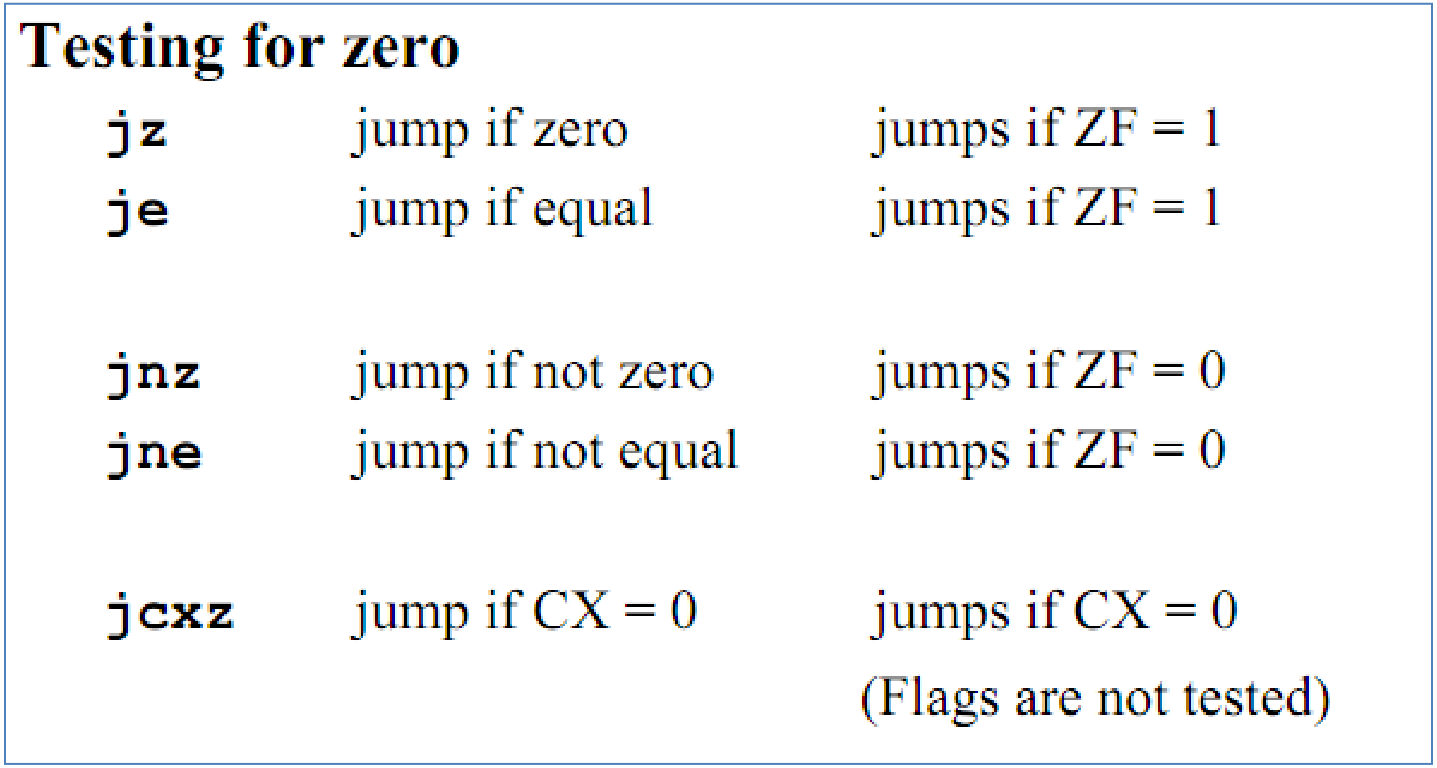
.

Conditional Jumps

- 2 types
  - Jumps checking Flags
  - Jump based on Comparison
    - of Unsigned Values
    - of Signed Values

Jumps checking Flags

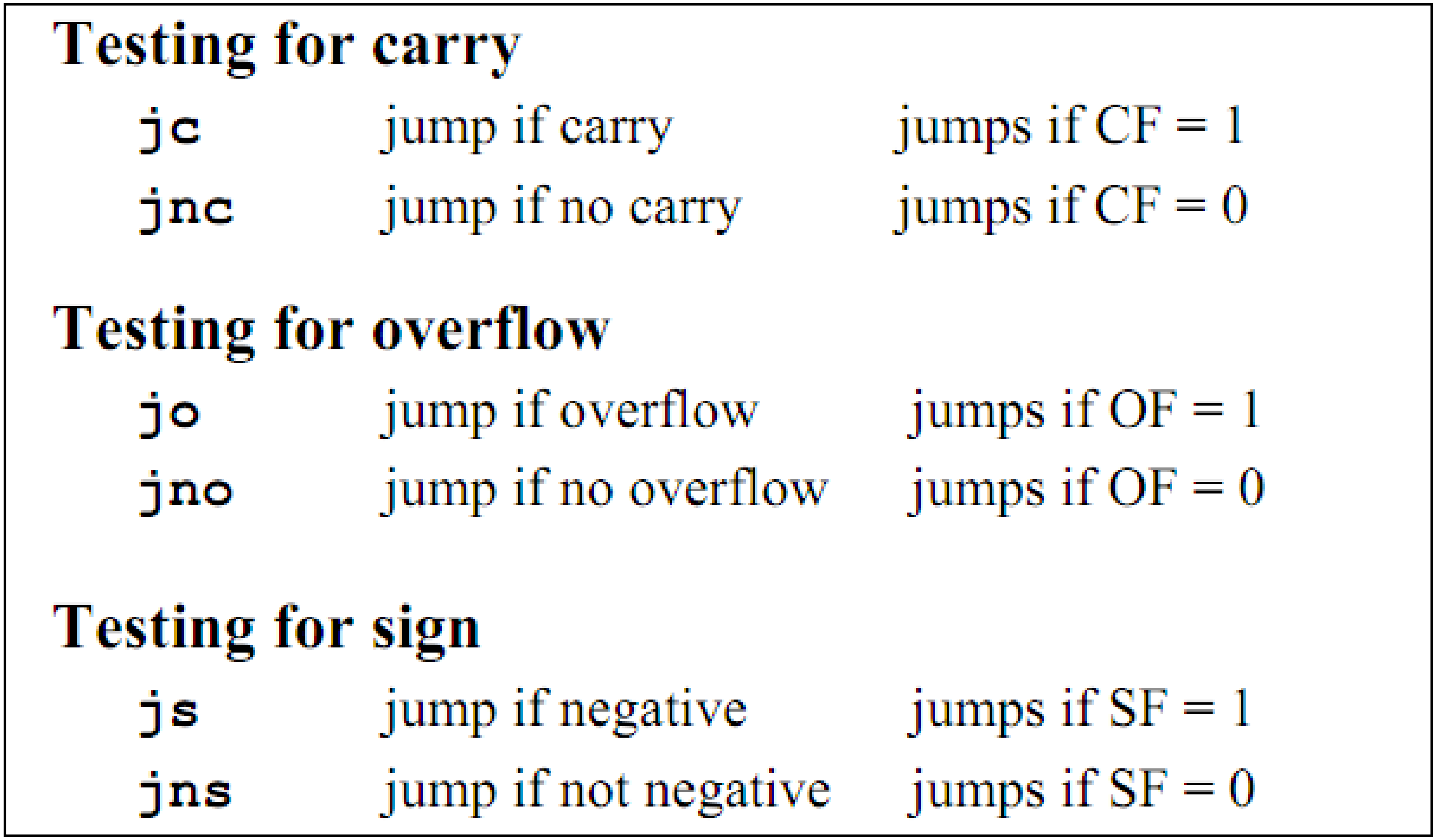
- Can jump conditionally at any place (label) via checking Flag values



Testing for zero

`jz`	jump if zero	jumps if ZF = 1
`je`	jump if equal	jumps if ZF = 1
`jnz`	jump if not zero	jumps if ZF = 0
`jne`	jump if not equal	jumps if ZF = 0
`jcxz`	jump if CX = 0	jumps if CX = 0 (Flags are not tested)

Jumps checking Flags...



Testing for carry

`jc`	jump if carry	jumps if CF = 1
`jnc`	jump if no carry	jumps if CF = 0

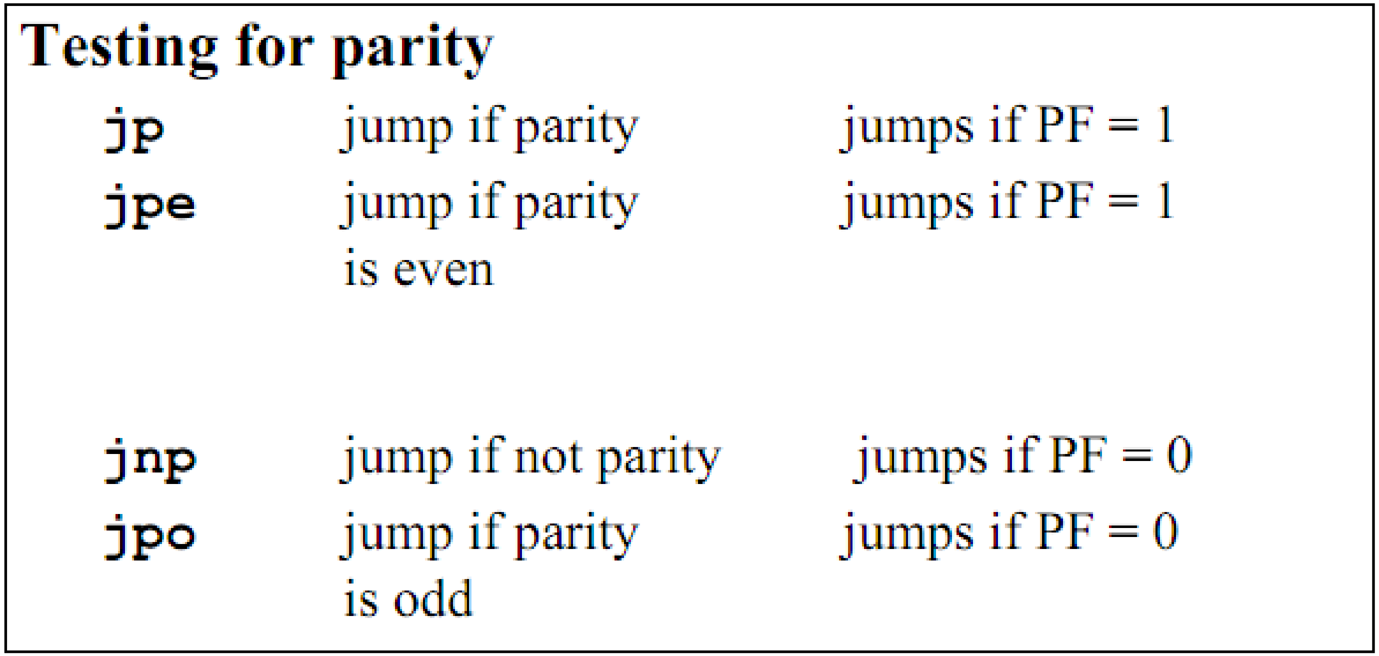
Testing for overflow

`jo`	jump if overflow	jumps if OF = 1
`jno`	jump if no overflow	jumps if OF = 0

Testing for sign

`js`	jump if negative	jumps if SF = 1
`jns`	jump if not negative	jumps if SF = 0

Jumps checking Flags ...

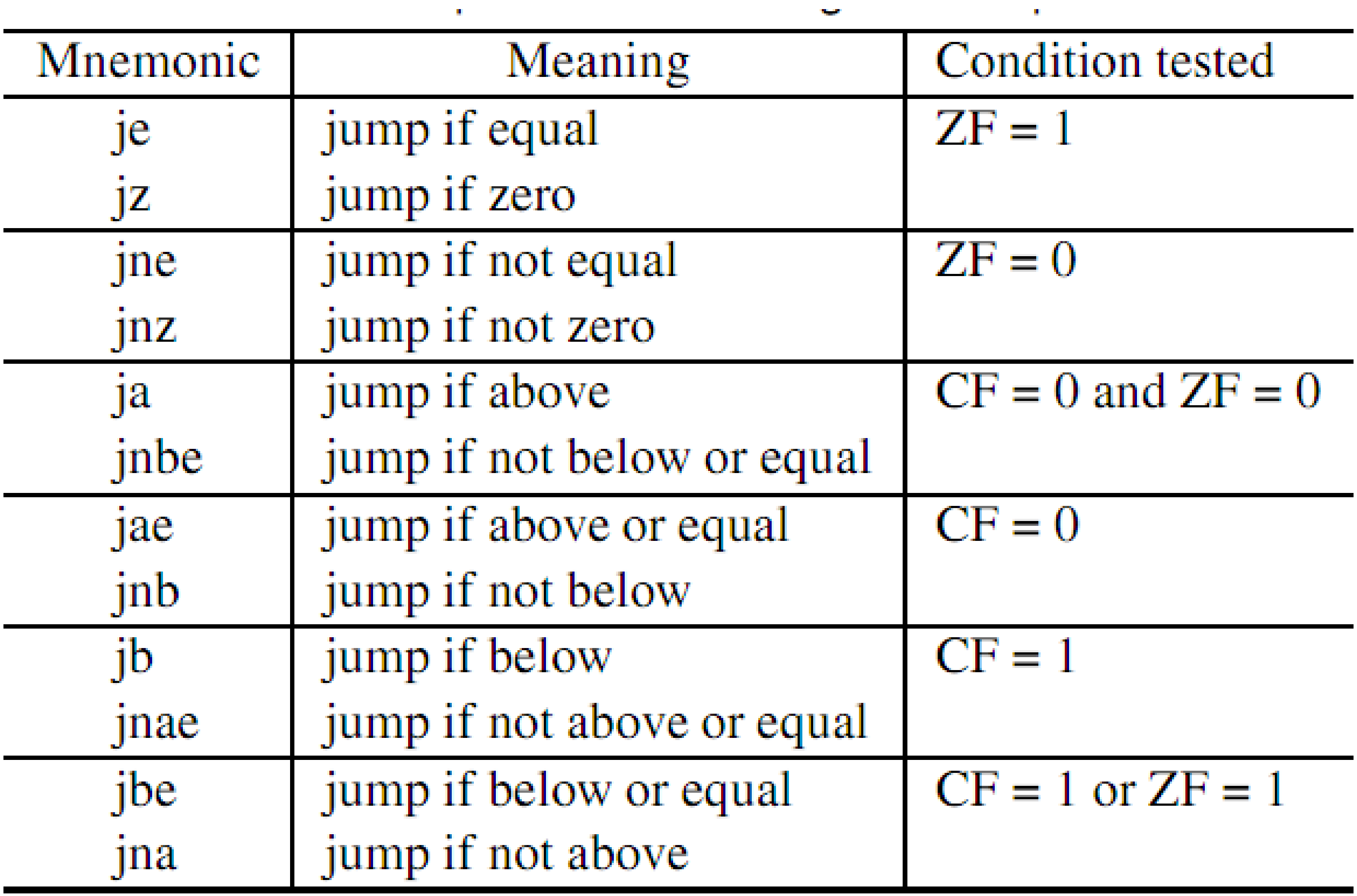


Testing for parity

`jp`	jump if parity	jumps if PF = 1
`jpe`	jump if parity is even	jumps if PF = 1
`jnp`	jump if not parity	jumps if PF = 0
`jpo`	jump if parity is odd	jumps if PF = 0

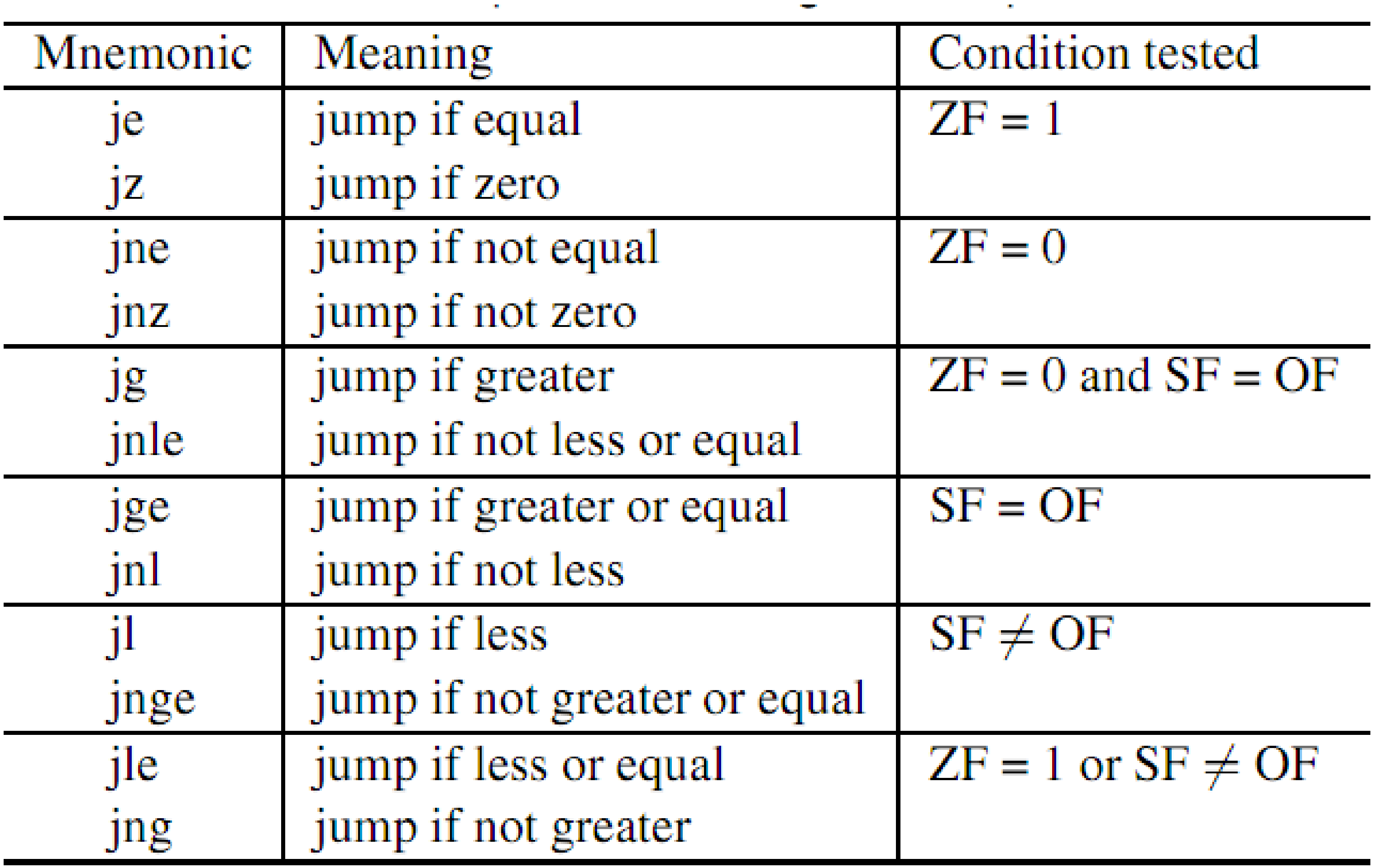
Jump based on Comparison (of Unsigned Values) used after CMP instruction

CMP = **SUB** but non-destructive operation just flags affected



Mnemonic	Meaning	Condition tested
je jz	jump if equal jump if zero	ZF = 1
jne jnz	jump if not equal jump if not zero	ZF = 0
ja jnbe	jump if above jump if not below or equal	CF = 0 and ZF = 0
jae jnb	jump if above or equal jump if not below	CF = 0
jb jnae	jump if below jump if not above or equal	CF = 1
jbe jna	jump if below or equal jump if not above	CF = 1 or ZF = 1

Jump based on comparison (of Signed Values)



Mnemonic	Meaning	Condition tested
je	jump if equal	ZF = 1
jz	jump if zero	
jne	jump if not equal	ZF = 0
jnz	jump if not zero	
jg	jump if greater	ZF = 0 and SF = OF
jnle	jump if not less or equal	
jge	jump if greater or equal	SF = OF
jnl	jump if not less	
jl	jump if less	SF ≠ OF
jnge	jump if not greater or equal	
jle	jump if less or equal	ZF = 1 or SF ≠ OF
jng	jump if not greater	

Three Structures of Program

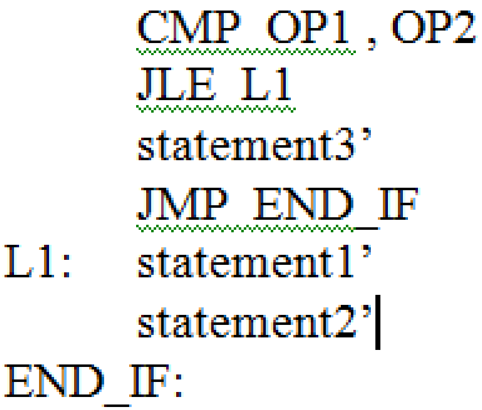
- Sequences
  - sequential execution of instructions
- Decision
  - Branching based on some condition
  - 2/multiple branches
- Loop
  - Repeated execution of instructions based on decision/condition

1:-Decisions

- Branch to one from two possible execution paths based on some condition.
- Implemented via IF-THEN-ELSE in HLL

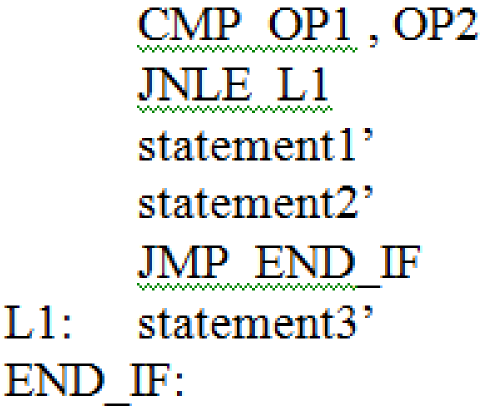
Example 1

```
IF(OP1 <= OP2)THEN
        statement1
        statement2
ELSE
        statement3
ENDIF
```



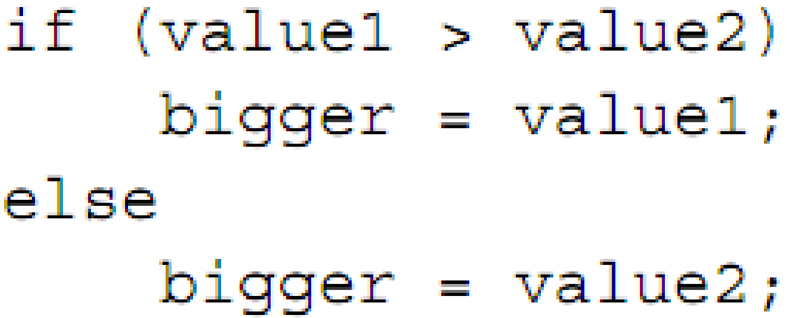
```
        CMP OP1 , OP2
        JLE L1
        statement3’
        JMP END_IF
L1:     statement1’
        statement2’
END_IF:
```

More Logical

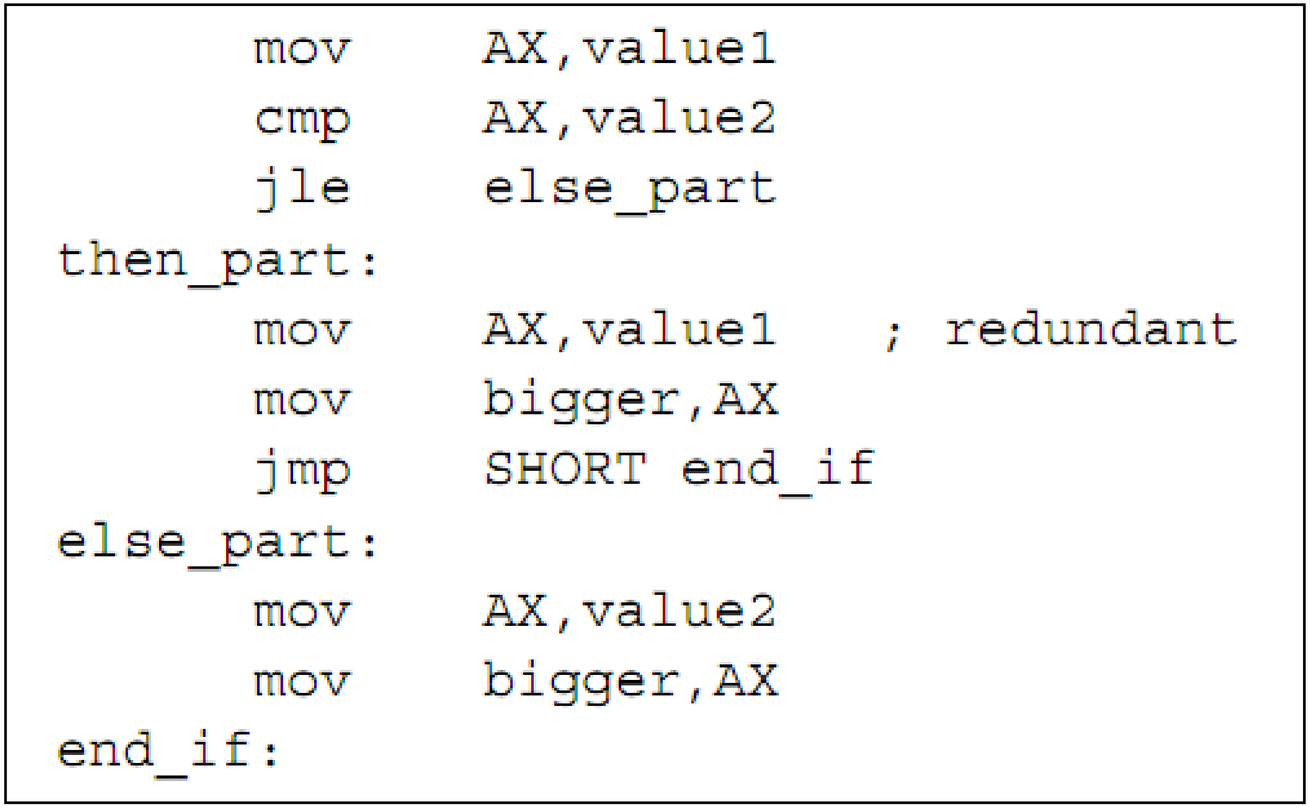


```
        CMP OP1 , OP2
        JNLE L1
        statement1’
        statement2’
        JMP END_IF
L1:     statement3’
END_IF:
```

Decision



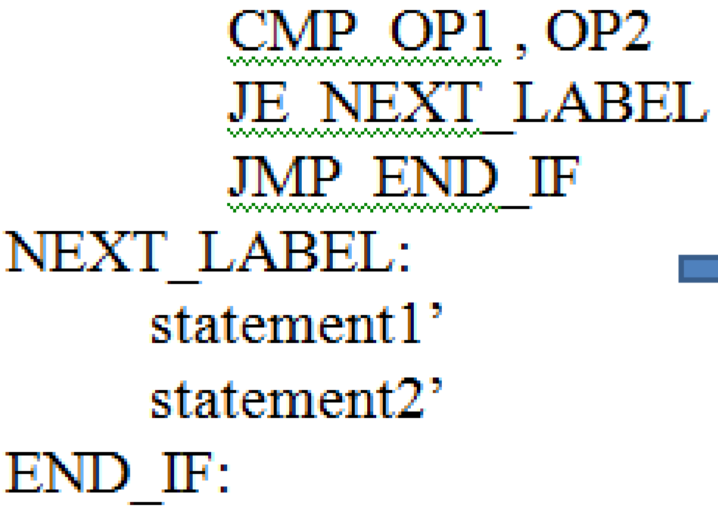
```
if (value1 > value2)
    bigger = value1;
else
    bigger = value2;
```



```
        mov     AX,value1
        cmp     AX,value2
        jle     else_part
then_part:
        mov     AX,value1    ; redundant
        mov     bigger,AX
        jmp     SHORT end_if
else_part:
        mov     AX,value2
        mov     bigger,AX
end_if:
```

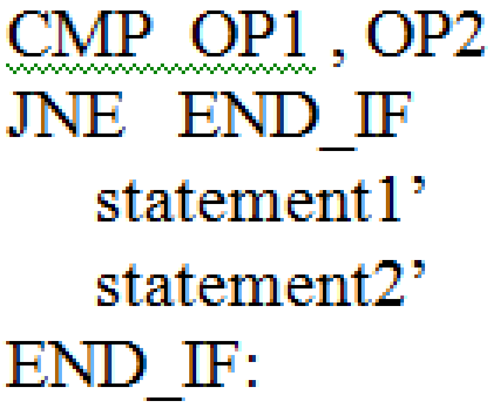
Example 2

```
IF(OP1 == OP2)THEN
        statement1
        statement2
ENDIF
```



```
        CMP  OP1 , OP2
        JE  NEXT_LABEL
        JMP  END_IF
NEXT_LABEL:
      statement1'
      statement2'
END_IF:
```

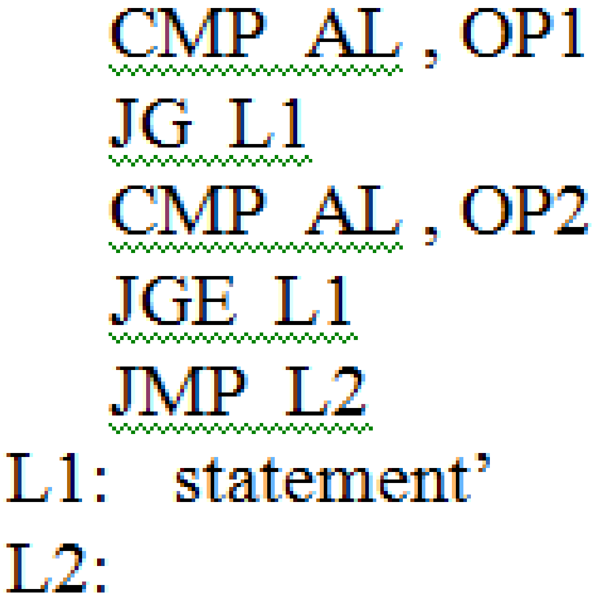
More better
condition reversed



```
CMP  OP1 , OP2
JNE   END_IF
   statement1'
   statement2'
END_IF:
```

Example 3

```
IF((AL > OP1) OR (AL >= 0P2))THEN
        statement
ENDIF
```



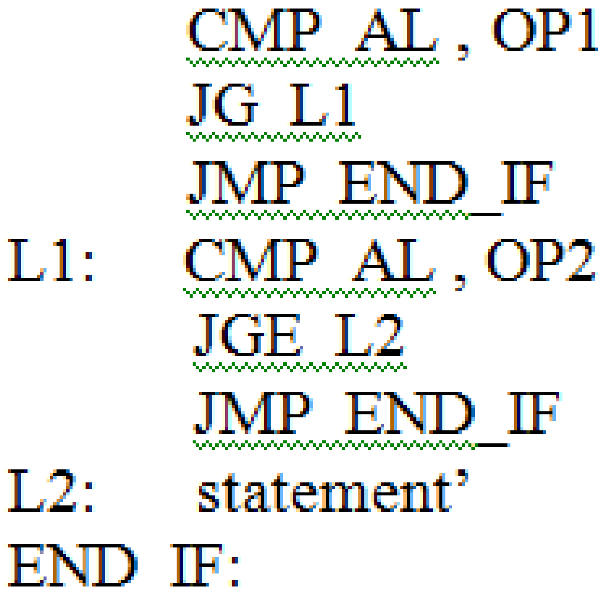
```
      CMP  AL , OP1
      JG  L1
      CMP  AL , OP2
      JGE  L1
      JMP  L2
L1:   statement’
L2:
```

Example 4

IF((AL > OP1) AND (AL >= OP2))THEN

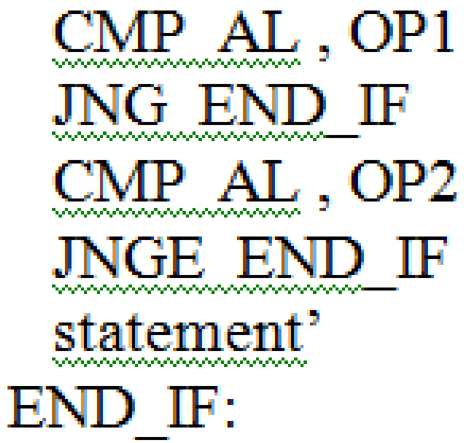
statement’

ENDIF



```
        CMP  AL , OP1
        JG  L1
        JMP  END_IF
L1:     CMP  AL , OP2
        JGE  L2
        JMP  END_IF
L2:     statement’
END_IF:
```

Reversing Conditions
Jumping to END_IF if
reversing Conditions true



```
  CMP  AL , OP1
  JNG  END_IF
  CMP  AL , OP2
  JNGE  END_IF
  statement’
END_IF:
```

Quiz 3

1)implement in Assembly

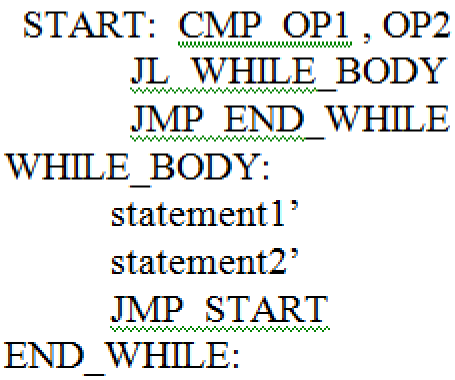
```
IF  ((a==b) AND (X>Y) )OR (S<=T)
        c=d;
ELSE
        b=b+1;
```

2-LOOPS

- Program loop consists of three components
  - Initialization
  - Loop termination test
  - Body of the loop
- Three permutations of these components
  - Thus three types of loops
    - While
    - Do ---While
    - Loop ---EndLoop

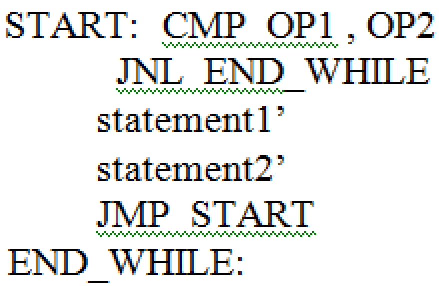
Example

```
WHILE(OP1 < OP2)DO
        statement1
        statement2
ENDWHILE
```



```
 START:  CMP  OP1 , OP2
         JL  WHILE_BODY
         JMP  END_WHILE
WHILE_BODY:
        statement1'
        statement2'
        JMP  START
END_WHILE:
```

Reversing condition and
Exiting while reverse condition true



```
START:  CMP  OP1 , OP2
         JNL  END_WHILE
        statement1'
        statement2'
        JMP  START
END_WHILE:
```

2.2-DO---WHILE loop

- Test for termination condition at end
- Loop body executes at least once

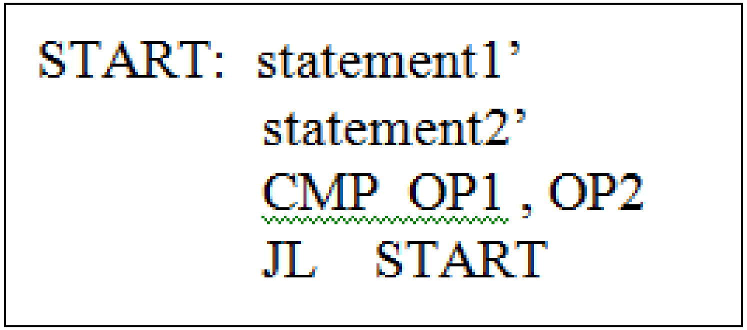
```
    DO
            Statements
    WHILE (Condition)
```

- Generally implemented in assembly as

```
LP:         Loop Body
    IF Termination-Condition THEN GOTO LP
```

Example

```
DO
        statement1
        statement2
WHILE(OP1 < OP2)
```



```
START:  statement1’
        statement2’
        CMP  OP1 , OP2
        JL    START
```

2.3-LOOP---ENDLOOP

- Test termination condition in between
- C/C++ Does not directly support such a loop
- It takes following form

```
LOOP
    Loop Body
ENDLOOP
```

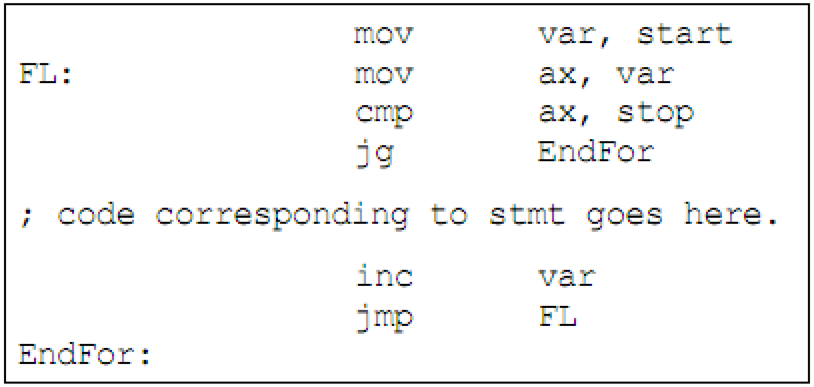
No explicit termination condition (handled by IF and GOTO)

2.4-FOR Loop

- Special form of while loop
- Number of repetitions fixed
- Used to process arrays most of the time

FOR var=start; var<=stop; var++)

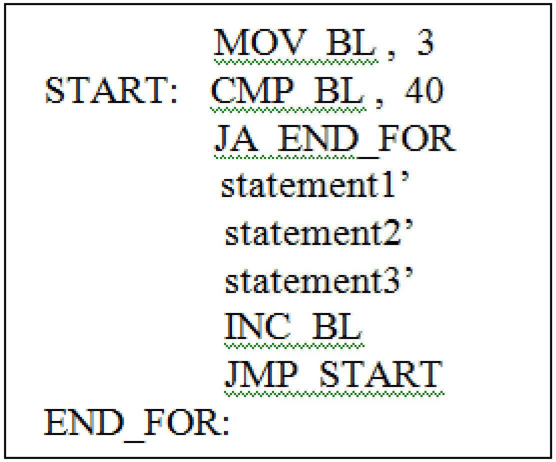
statements;



```
                        mov      var, start
FL:                     mov      ax, var
                        cmp      ax, stop
                        jg       EndFor
; code corresponding to stmt goes here.
                        inc      var
                        jmp      FL
EndFor:
```

Example 1

```
for( i = 3 ; i <= 40 ; i++ )
 {
        statement1 ;
        statement2 ;
        statement3 ;
 }
```

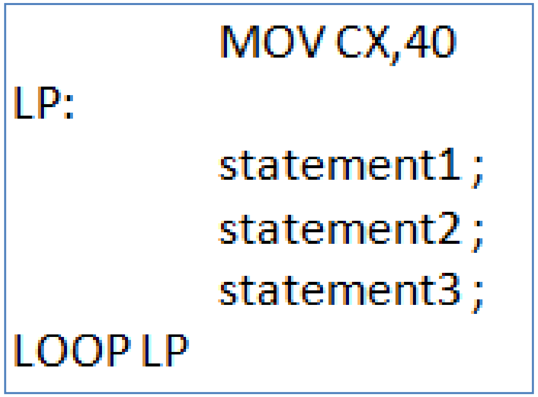


```
            MOV  BL , 3
START:      CMP  BL , 40
            JA  END_FOR
             statement1’
             statement2’
             statement3’
             INC  BL
             JMP  START
END_FOR:
```

If number of iterations known then
Do not simulate FOR Loop,
Instead Use LOOP instruction
→ See next slide

No of iterations known

```
for( i = 0 ; i <= 39 ; i++ )
 {
        statement1 ;
        statement2 ;
        statement3 ;
 }
```



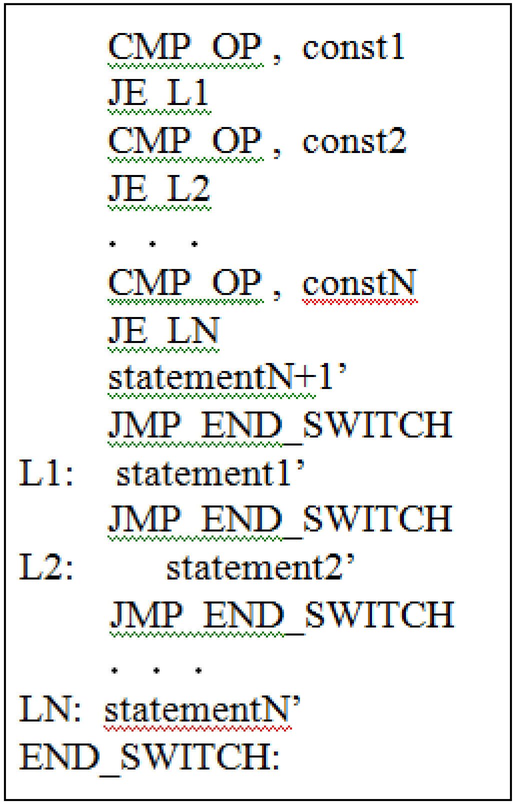
```
         MOV CX,40
LP:
         statement1 ;
         statement2 ;
         statement3 ;
LOOP LP
```

The LOOP instruction decrements CX and transfer control to the beginning of its loop if CX ≠ 0; otherwise the next sequential instruction in the program is executed. If CX = 0 before the loop it is decremented to -1 at the end of the first iteration of the loop. This -1 is treated as the unsigned number 65535, thus the loop will iterate 65536 times.

2.5-Switch Statement

Multiple selection conditions

```
Switch (OP)
 {
    case  const1:   statement1 ;
          break ;
    case  const2:   statement2 ;
          break ;
          .  .  .
    case  constN:   statementN ;
          break ;
   default:         statementN+1 ;
}
```



```
       CMP  OP ,  const1
       JE  L1
       CMP  OP ,  const2
       JE  L2
       .   .   .
       CMP  OP ,  constN
       JE  LN
       statementN+1'
       JMP  END_SWITCH
L1:     statement1'
       JMP  END_SWITCH
L2:        statement2'
       JMP  END_SWITCH
       .   .   .
LN:  statementN'
END_SWITCH:
```

CMP Instruction

- Compares the destination operand to the source operand
  - Nondestructive subtraction of source from destination (destination operand is not changed)
- Syntax: CMP *destination, source*
- Example: destination == source

```
mov al,5
cmp al,5                ; Zero flag set
```

- Example: destination < source

```
mov al,4
cmp al,5                ; Carry flag set
```

CMP Instruction

- Example: destination > source

```
mov al,6
cmp al,5                    ; ZF = 0, CF = 0
```

(both the Zero and Carry flags are clear)

J*cond* Instruction

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

- Specific jumps:

  JB, JC - jump to a label if the Carry flag is set

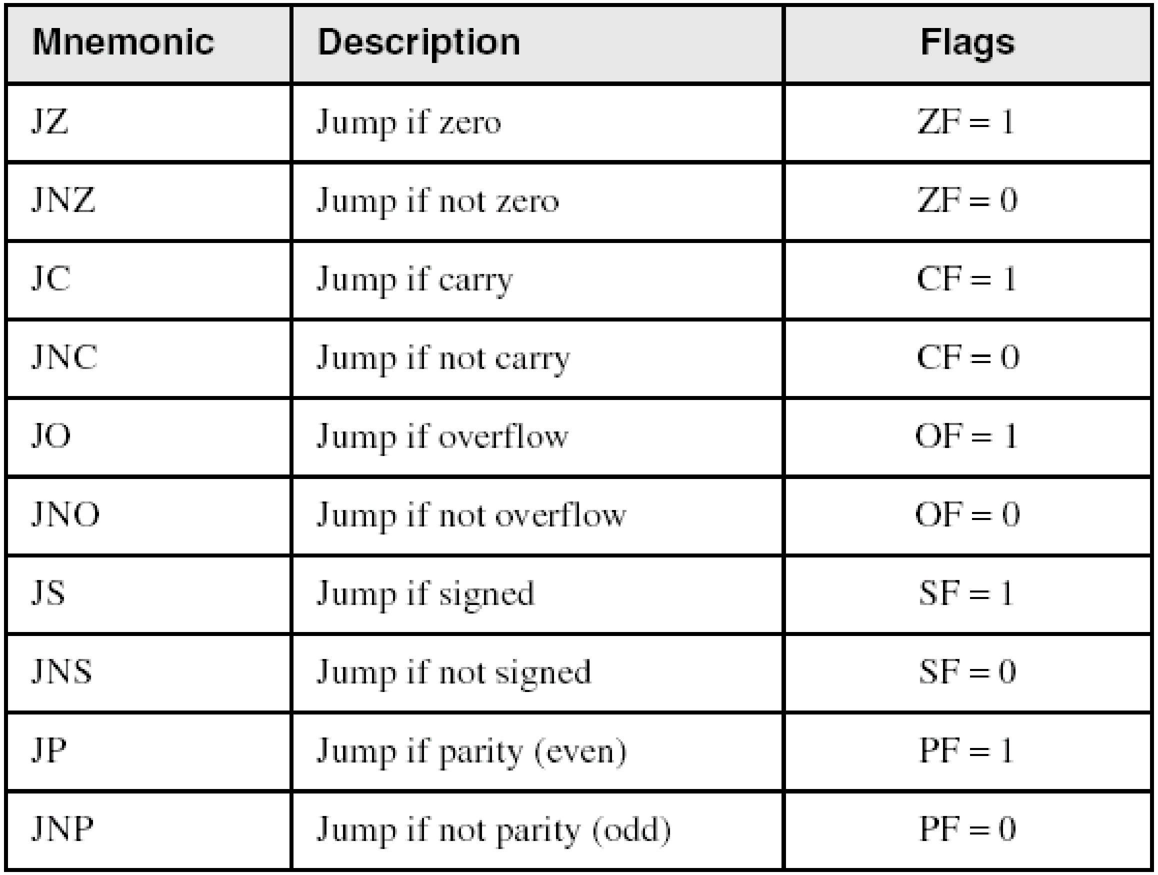
  JE, JZ - jump to a label if the Zero flag is set

  JS - jump to a label if the Sign flag is set

  JNE, JNZ - jump to a label if the Zero flag is clear

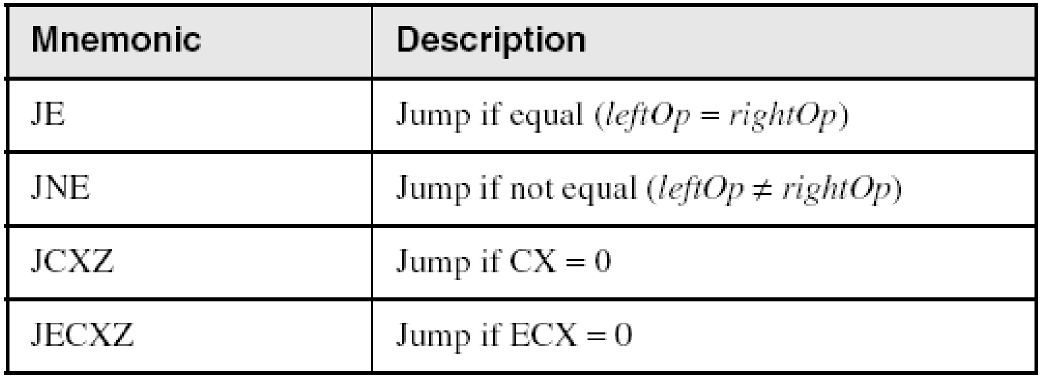
  JECXZ - jump to a label if ECX = 0

Jumps Based on Specific Flags



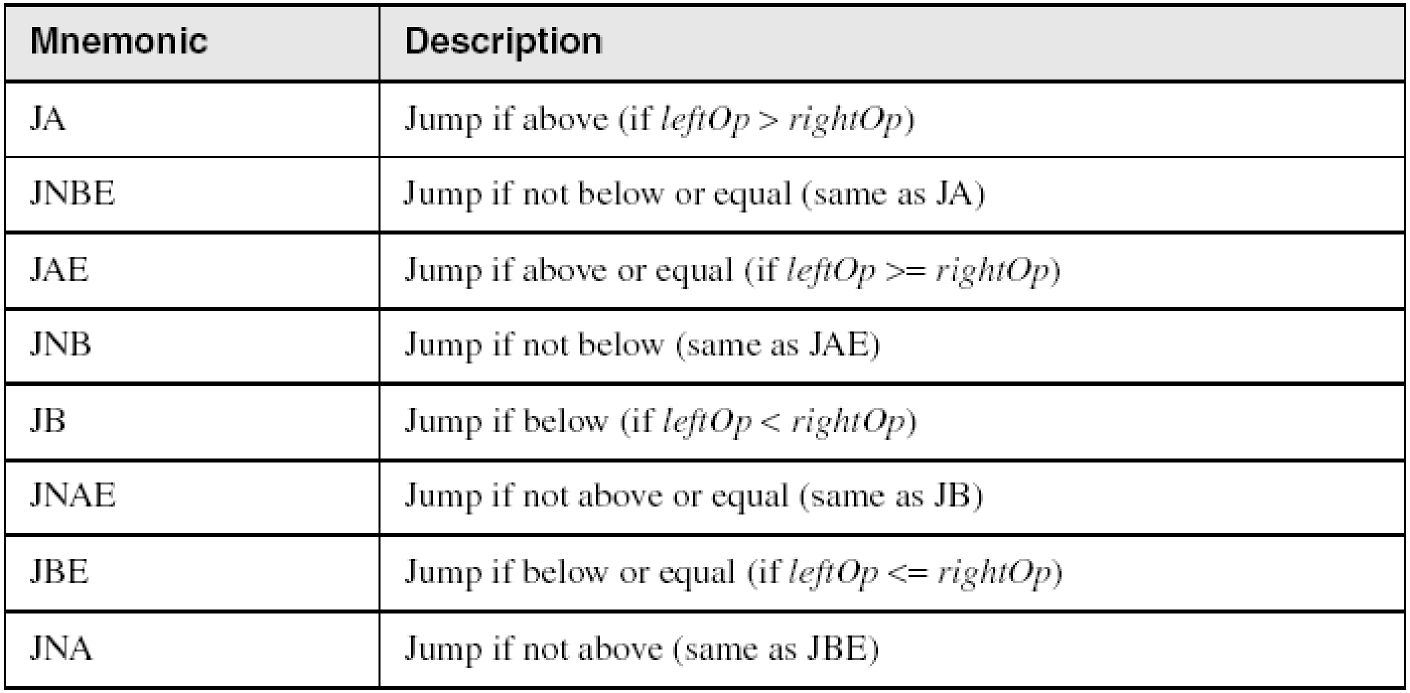
Mnemonic	Description	Flags
JZ	Jump if zero	ZF = 1
JNZ	Jump if not zero	ZF = 0
JC	Jump if carry	CF = 1
JNC	Jump if not carry	CF = 0
JO	Jump if overflow	OF = 1
JNO	Jump if not overflow	OF = 0
JS	Jump if signed	SF = 1
JNS	Jump if not signed	SF = 0
JP	Jump if parity (even)	PF = 1
JNP	Jump if not parity (odd)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

Jumps Based on Equality



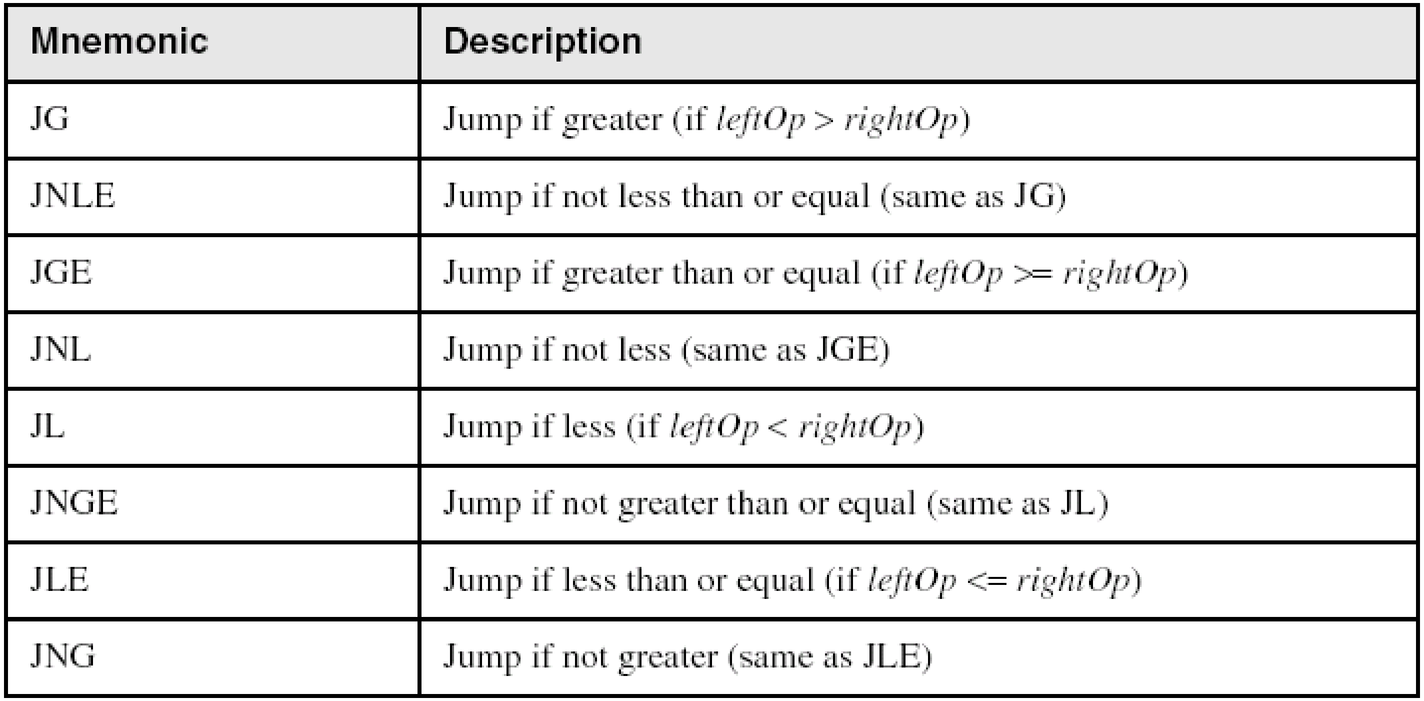
Mnemonic	Description
JE	Jump if equal (*leftOp* = *rightOp*)
JNE	Jump if not equal (*leftOp* ≠ *rightOp*)
JCXZ	Jump if CX = 0
JECXZ	Jump if ECX = 0

Jumps Based on Unsigned Comparisons



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)

Jumps Based on Signed Comparisons



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)

Applications

- Task: Jump to a label if unsigned EAX is greater than EBX
- Solution: Use CMP, followed by JA

```
cmp eax,ebx
ja  Larger
```

- Task: Jump to a label if signed EAX is greater than EBX
- Solution: Use CMP, followed by JG

```
cmp eax,ebx
jg  Greater
```

Applications

- Jump to label L1 if unsigned EAX is less than or equal to Val1

```
cmp eax,Val1
jbe L1          ; below or equal
```

- Jump to label L1 if signed EAX is less than or equal to Val1

```
cmp eax,Val1
jle L1
```

Applications

- Compare unsigned AX to BX, and copy the larger of the two into a variable named Large

```
          mov Large,bx
          cmp ax,bx
          jna Next
          mov Large,ax
      Next:
```

- Compare signed AX to BX, and copy the smaller of the two into a variable named Small

```
          mov Small,ax
          cmp bx,ax
          jnl Next
          mov Small,bx
      Next:
```

Applications

- Jump to label L1 if the memory word pointed to by ESI equals Zero

```
cmp WORD PTR [esi],0
je  L1
```

- Jump to label L2 if the doubleword in memory pointed to by EDI is even

```
test DWORD PTR [edi],1
jz   L2
```

Your turn . . .

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:
```

(There are multiple correct solutions to this problem.)

Your turn . . .

Implement the following pseudocode in assembly language. All values are unsigned:

```
if( ebx <= ecx
    && ecx > edx )
{
  eax = 5;
  edx = 6;
}
```

```
    cmp ebx,ecx
    ja  next
    cmp ecx,edx
    jbe next
    mov eax,5
    mov edx,6
next:
```

(There are multiple correct solutions to this problem.)

Your turn . . .

Implement the following loop, using unsigned 32-bit integers:

```
while( ebx <= val1)
{
    ebx = ebx + 5;
    val1 = val1 - 1
}
```

```
top: cmp ebx,val1               ; check loop condition
     ja  next                   ; false? exit loop
     add ebx,5                  ; body of loop
     dec val1
     jmp top                    ; repeat the loop
next:
```

Thanks!