EE-2003
Computer Organization & Assembly Language

Chapter No: 06

Conditional Processing

Bit-wise Operations

Status Flags (Revision)

- • The Zero flag is set when the result of an operation equals zero.
- • The Carry flag is set when an instruction generates a result that is too large (or too small) for the destination operand.
- • The Sign flag is set if the destination operand is negative, and it is clear if the destination operand is positive.
- • The Overflow flag is set when an instruction generates an invalid signed result.
- • The Parity flag is set when an instruction generates an even number of 1 bits in the low byte of the destination operand.
- • The Auxiliary Carry flag is set when an operation produces a carry out from bit 3 to bit 4.

NOT Instruction

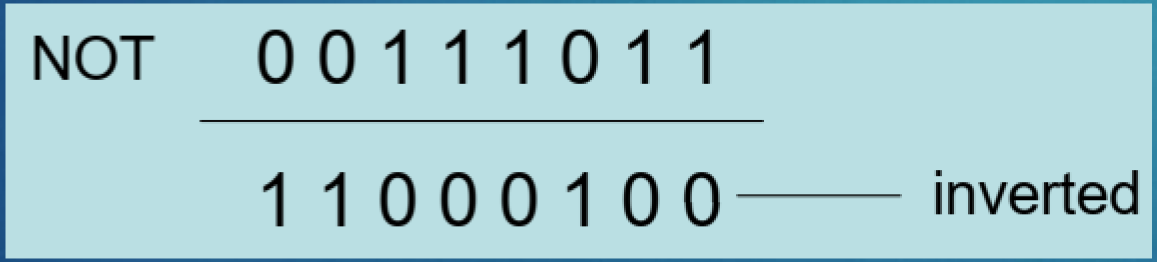
- Performs a bitwise Boolean **NOT** operation on a single destination operand
- Syntax: (no flag affected)

  NOT destination

- Example:

  mov al 11110000b

  NOT al

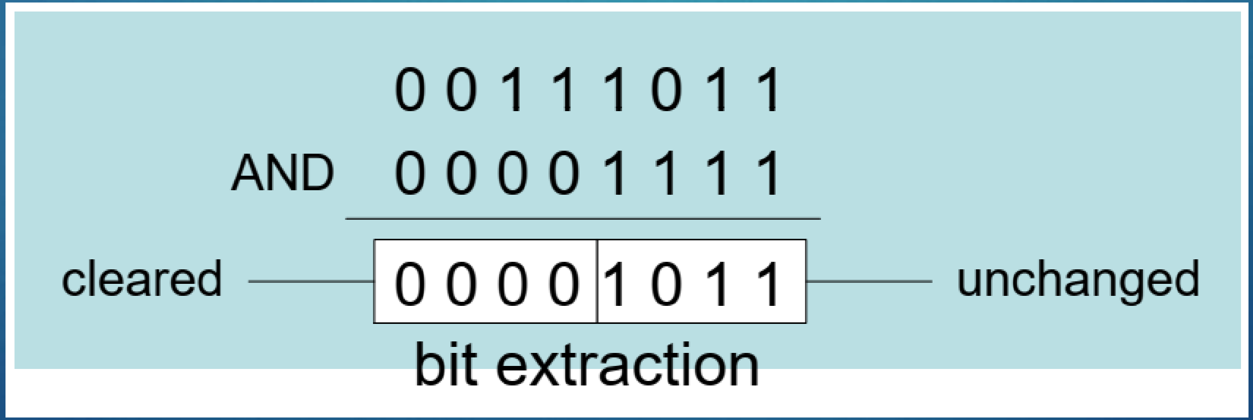


```
NOT   0 0 1 1 1 0 1 1
      _______________
      1 1 0 0 0 1 0 0 —— inverted
```

AND Instruction

- Performs a bitwise Boolean **AND** operation between each pair of matching bits in two operands
- AND instruction always clears Overflow and Carry flag. Also can modify Sign, Zero, and Parity in a way that is consistent with the value assigned to the destination operand.
- Syntax: AND destination, source
- Example:

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



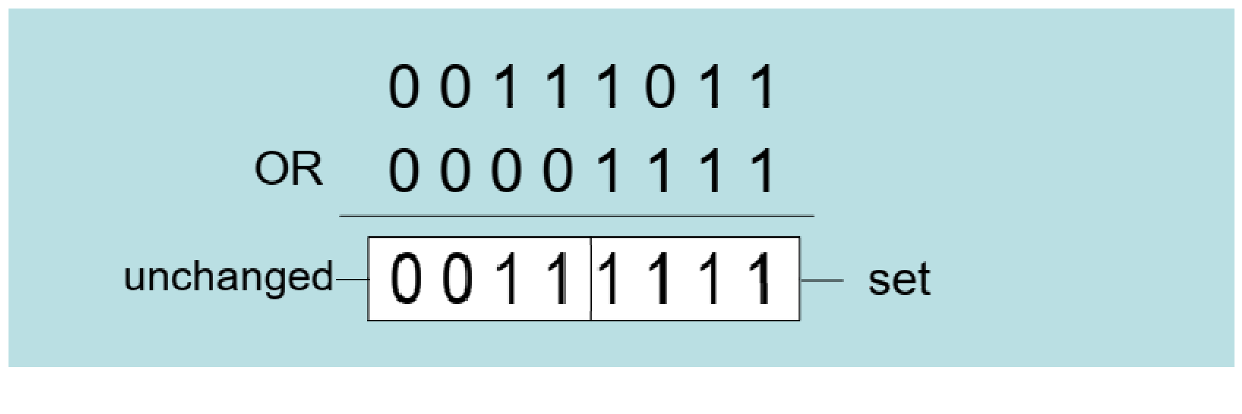
```
          0 0 1 1 1 0 1 1
    AND   0 0 0 0 1 1 1 1
          ---------------
cleared   0 0 0 0 1 0 1 1   unchanged
```

bit extraction

OR Instruction

- Performs a bitwise Boolean **OR** operation between each pair of matching bits in two operands
- Syntax: Clears Overflow, Cary . Modifies Sign, Zero, and Parity in a way that is consistent with the value assigned to the destination operand
- Syntax: OR destination, source
- Example:

```
mov   al, 00111011b
or    al, 00001111b
```

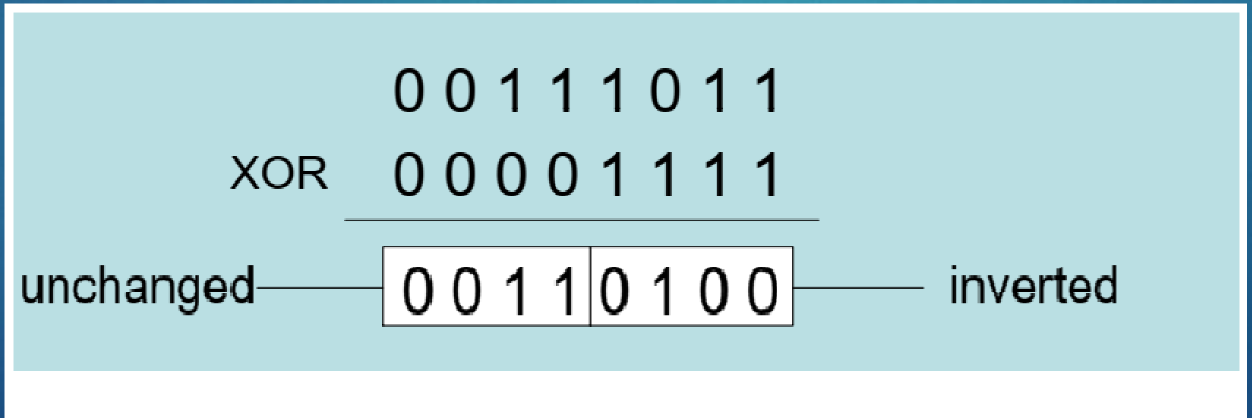


```
             0 0 1 1 1 0 1 1
       OR    0 0 0 0 1 1 1 1
            -----------------
unchanged —  0 0 1 1|1 1 1 1  — set
```

XOR Instruction

- Performs a bitwise Boolean X**OR** operation between each pair of matching bits in two operands
- The XOR instruction always clears the Overflow and Carry flags.
- Syntax: XOR destination, source
- Example:

```
mov   al, 00111011b
xor   al, 00001111b
```



```
            0 0 1 1 1 0 1 1
      XOR   0 0 0 0 1 1 1 1
          -----------------
unchanged — 0 0 1 1|0 1 0 0 — inverted
```

XOR is a useful way to invert the bits in an operand and data encryption

Applications

- Convert the character in AL to upper case
- **Solution:** Use the AND instruction to clear bit 5

```
mov   al , 'a'          ; AL = 01100001b
and   al , 11011111b    ; AL = 01000001b
```

- 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 = 00000110b
or    al , 00110000b    ; AL = 00110110b
```

- 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
```

Applications

- 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
```

Non-Destructive Instructions

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
- The TEST instruction always clears the Overflow and Carry flags
- 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
```

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
```

- Example: destination > source

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

The comparisons shown so far were unsigned

CMP Instruction

- The comparisons shown here are performed with signed integers
- Example: destination > source

```
mov al , 5
cmp al , -2                    ; Sign flag == Overflow flag
```

- Example: destination < source

```
mov al , -1
cmp al , 5                     ; Sign flag != Overflow flag
```

Conditions and Output

Unsigned	ZF	CF
destination < source	0	1
destination > source	0	0
destination = source	1	0

Signed	FLAGS
destination < source	SF != OF
destination > source	SF == OF
destination = source	ZF = 1

Setting and Clearing Individual Flags

```
and    al , 0                     ; set Zero
or     al , 1                     ; clear Zero
or     al , 80h                   ; set Sign
and    al , 7Fh                   ; clear Sign
stc                               ; set Carry
clc                               ; clear Carry

mov    al , 7Fh
inc    al                         ; set Overflow
or     eax , 0                    ; clear Overflow
```

BT (Bit Test) Instruction

- Copies n^{th} bit from an operand into the Carry flag
- Syntax:

```
BT reg/mem16 , reg16/reg32/imm
BT reg/mem32 , reg16/reg32/imm
```

- 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
```

- There are three more BT instructions:
  - BTC bitBase, n ; bit test and complement
  - BTR bitBase, n ; bit test and reset (clear)
  - BTS bitBase, n ; bit test and set

Conditional jumps

Conditional Structures

- There are no high-level logic structures such as if-then-else, in the IA-32 instruction set
  - But, you can use combinations of comparisons and jumps to implement any logic structure
- First, an operation such as CMP, AND or SUB is executed to modified the CPU flags
- Second, a conditional jump instruction tests the flags and changes the execution flow accordingly
- Example:

```
cmp eax,0                          and dl,10110000b
 jz L1    ; jump if ZF = 1         jnz L2          ; jump if ZF = 0
  . .                              ....


  L1:                                  L2:
```

Jcond Instructions

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

  Jcond Destination

- Four groups: (some are the same)
  - based on specific flag values
  - based on equality between operands
  - based on comparisons of unsigned operands
  - based on comparisons of signed operands

Jumps Based on Specific Flags

Flag	Instruction	Description	Flag Status
ZERO	JZ	Jump if zero	ZF = 1
	JNZ	Jump if not zero	ZF = 0
CARRY	JC	Jump if carry	CF = 1
	JNC	Jump if not carry	CF = 0
Over-Flow	JO	Jump if overflow	OF = 1
	JNO	Jump if not overflow	OF = 0
SIGN	JS	Jump if sign	SF = 1
	JNS	Jump if not sign	SF = 0
PARITY	JP	Jump if parity (even)	PF = 1
	JNP	Jump if not parity (odd)	PF = 0

Jumps Based on Equality

Instruction	Description
JE	Jump if equal (left OP = right OP)
JNE	Jump if not equal (left OP ≠ right OP)
JCXZ	Jump if CX = 0
JECXZ	Jump if ECX = 0

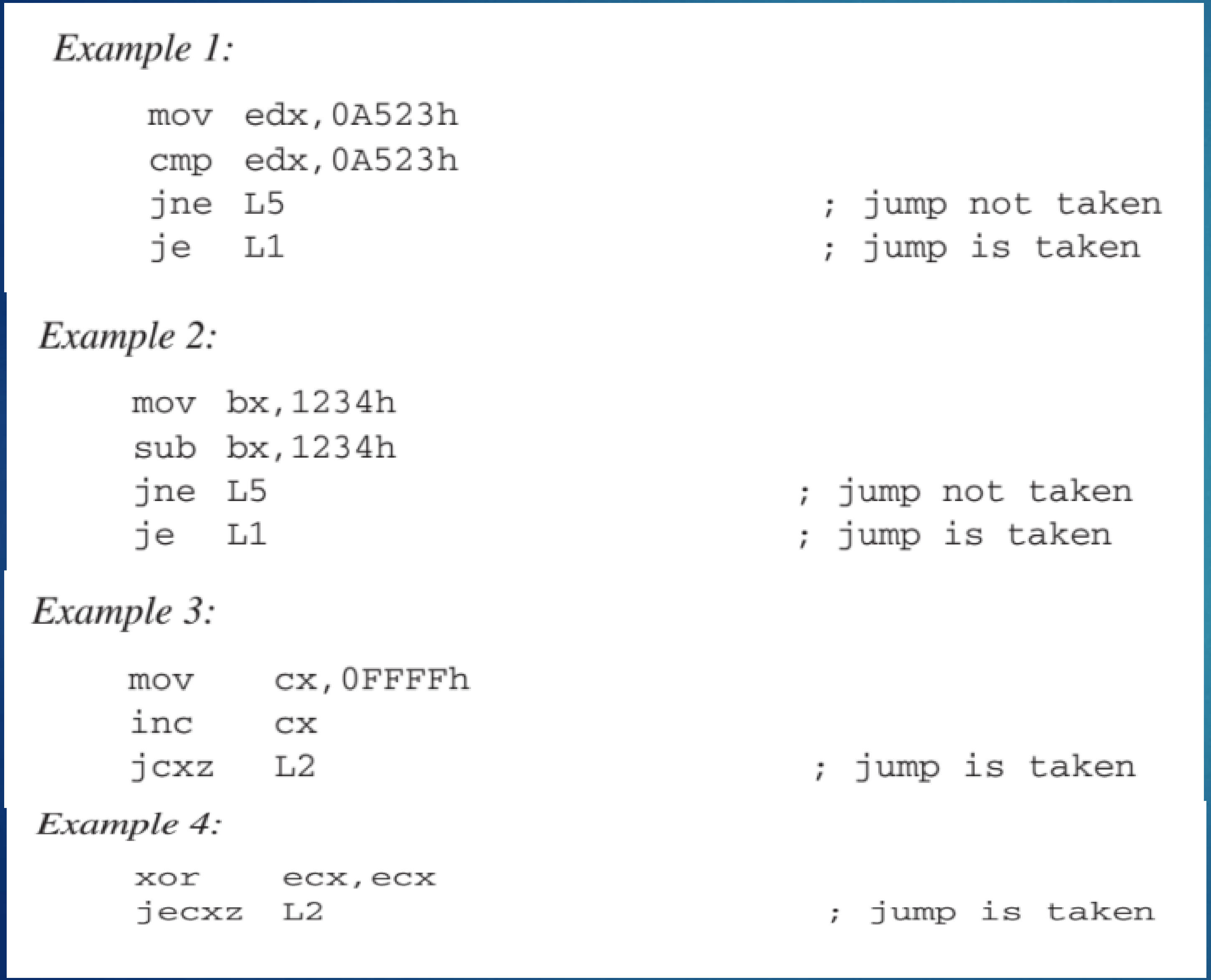
Jumps Based on Un-signed Comparison

Condition	Instruction	Description
$>$	JA	Jump if above $(left\ OP > right\ OP)$
	JNA	Jump if not above $(left\ OP \leq right\ OP)$
$\geq$	JAE	Jump if above and equal $(left\ OP \geq\ right\ OP)$
	JNAE	Jump if not above and equal $(left\ OP < right\ OP)$
$<$	JB	Jump if below $(left\ OP < right\ OP)$
	JNB	Jump if not below $(left\ OP \geq right\ OP)$
$\leq$	JBE	Jump if below and equal $(left\ OP \leq right\ OP)$
	JNBE	Jump if not below and equal $(left\ OP > right\ OP)$

Jumps Based on Signed Comparison

Condition	Instruction	Description
$>$	JG	Jump if greater $(left\ OP > right\ OP)$
	JNG	Jump if not greater $(left\ OP \leq right\ OP)$
$\geq$	JGE	Jump if greater and equal $(left\ OP \geq right\ OP)$
	JNGE	Jump if not greater and equal $(left\ OP < right\ OP)$
$<$	JL	Jump if less than $(left\ OP < right\ OP)$
	JNL	Jump if not less than $(left\ OP \geq right\ OP)$
$\leq$	JLE	Jump if less than and equal $(left\ OP \leq right\ OP)$
	JNLE	Jump if not less than and equal $(left\ OP > right\ OP)$

EXAMPLES



Example 1:

```
mov  edx,0A523h
cmp  edx,0A523h
jne  L5                          ; jump not taken
je   L1                          ; jump is taken
```

Example 2:

```
mov  bx,1234h
sub  bx,1234h
jne  L5                          ; jump not taken
je   L1                          ; jump is taken
```

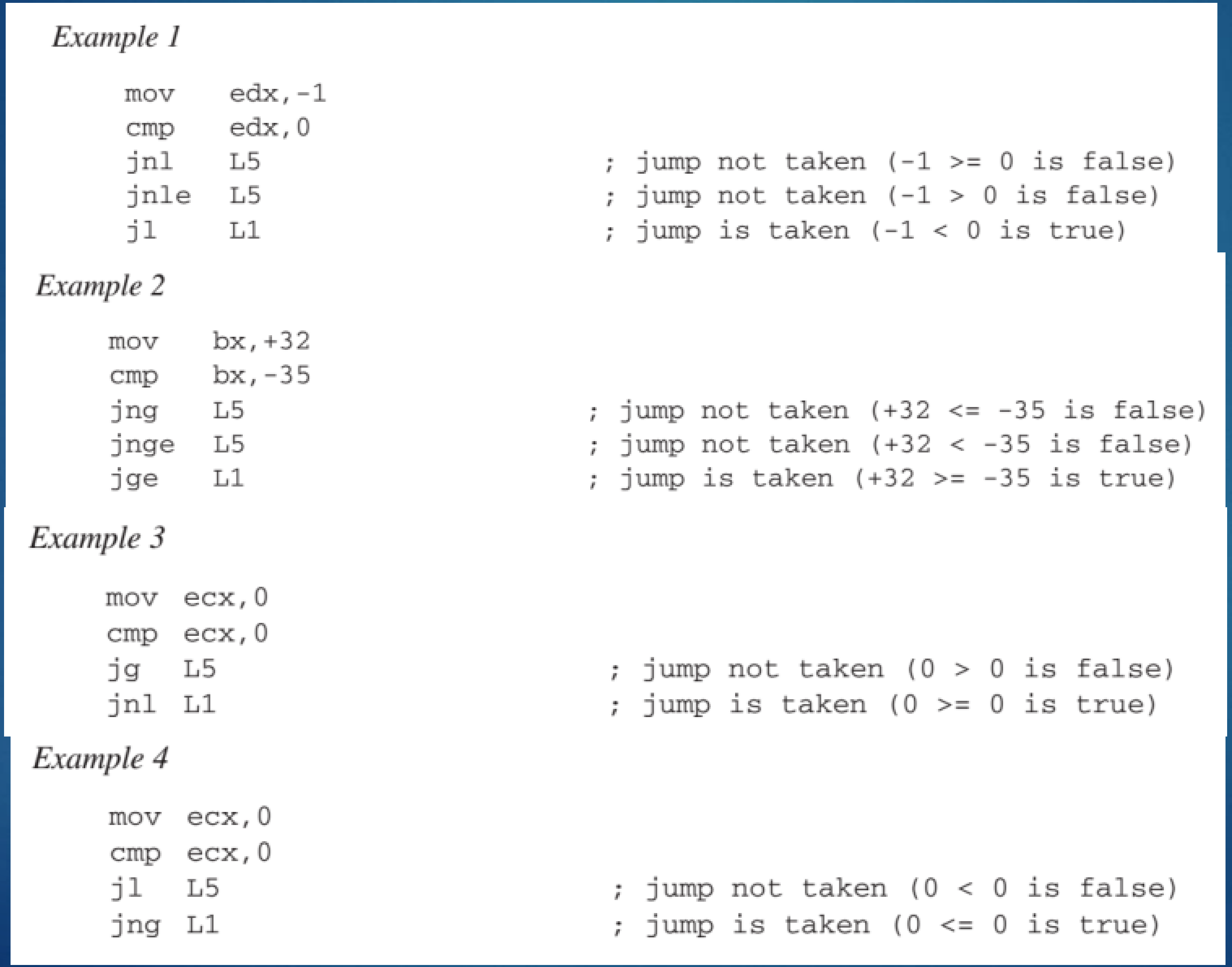
Example 3:

```
mov    cx,0FFFFh
inc    cx
jcxz   L2                        ; jump is taken
```

Example 4:

```
xor    ecx,ecx
jecxz  L2                        ; jump is taken
```

EXAMPLES (Signed CMP)



Example 1

```
    mov    edx,-1
    cmp    edx,0
    jnl    L5                        ; jump not taken (-1 >= 0 is false)
    jnle   L5                        ; jump not taken (-1 > 0 is false)
    jl     L1                        ; jump is taken (-1 < 0 is true)
```

Example 2

```
    mov    bx,+32
    cmp    bx,-35
    jng    L5                        ; jump not taken (+32 <= -35 is false)
    jnge   L5                        ; jump not taken (+32 < -35 is false)
    jge    L1                        ; jump is taken (+32 >= -35 is true)
```

Example 3

```
    mov  ecx,0
    cmp  ecx,0
    jg   L5                          ; jump not taken (0 > 0 is false)
    jnl  L1                          ; jump is taken (0 >= 0 is true)
```

Example 4

```
    mov  ecx,0
    cmp  ecx,0
    jl   L5                          ; jump not taken (0 < 0 is false)
    jng  L1                          ; jump is taken (0 <= 0 is true)
```

EXAMPLES

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

```
        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
- **Solution:**

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

Examples

- Find the first even number in an array of unsigned Integers
- **Solution:**

```
.date
intArray DWORD 7, 9, 3, 4, 6, 1
.code
…
    mov ebx, OFFSET intArray
    mov ecx, LENGTHOF intArray
L1:       test DWORD PTR [ebx], 1
          jz found
          add ebx, 4
          loop L1
…
Found:
```

Conditional Structures

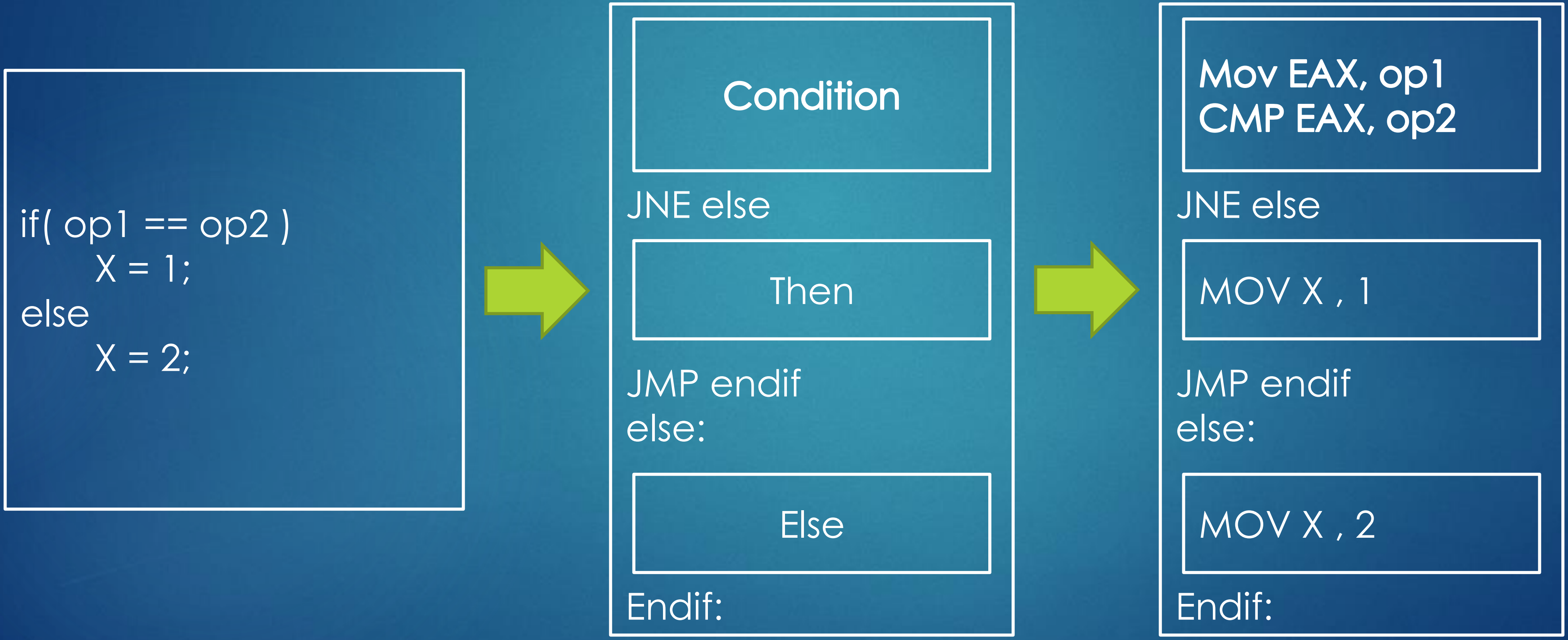
If Statements

- If Condition then Then else Else

```
    Condition
JNE esle
    Then
JMP endif
else:
    Else
endif:
```

Block-structured If Statements

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



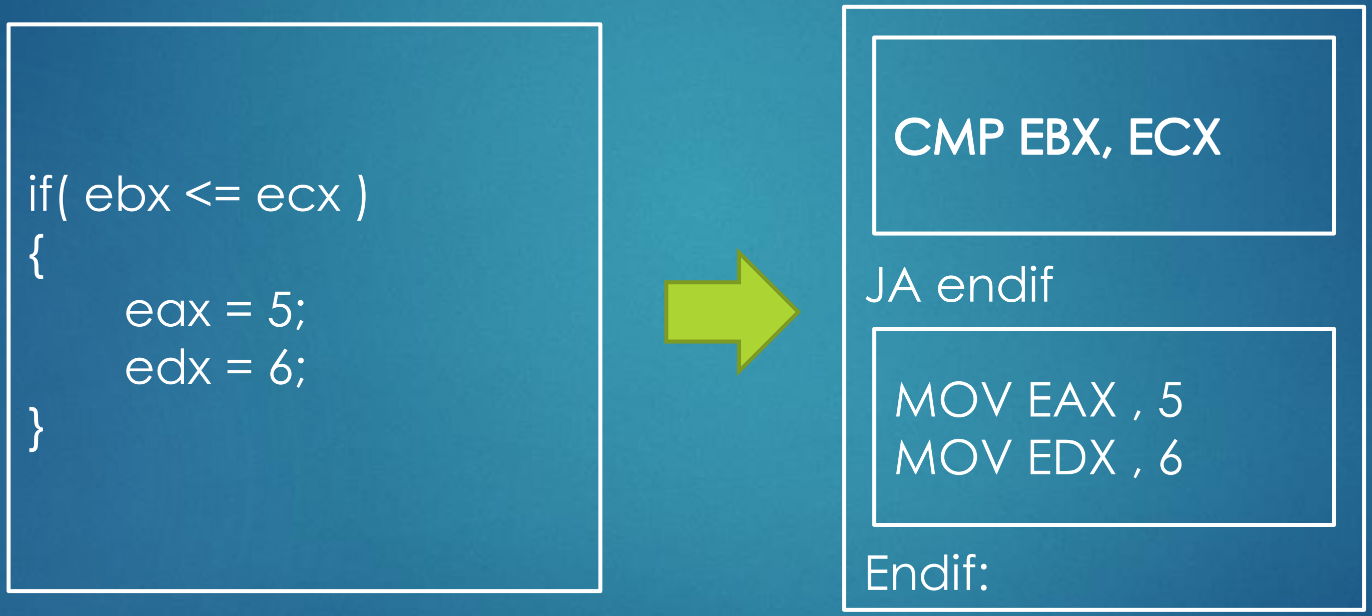
```
if( op1 == op2 )
        X = 1;
else
        X = 2;
```

```
Condition
JNE else
Then
JMP endif
else:
Else
Endif:
```

```
Mov EAX, op1
CMP EAX, op2
JNE else
MOV X , 1
JMP endif
else:
MOV X , 2
Endif:
```

Exercise

- Implement the following pseudocode in assembly language, all values are unsigned:

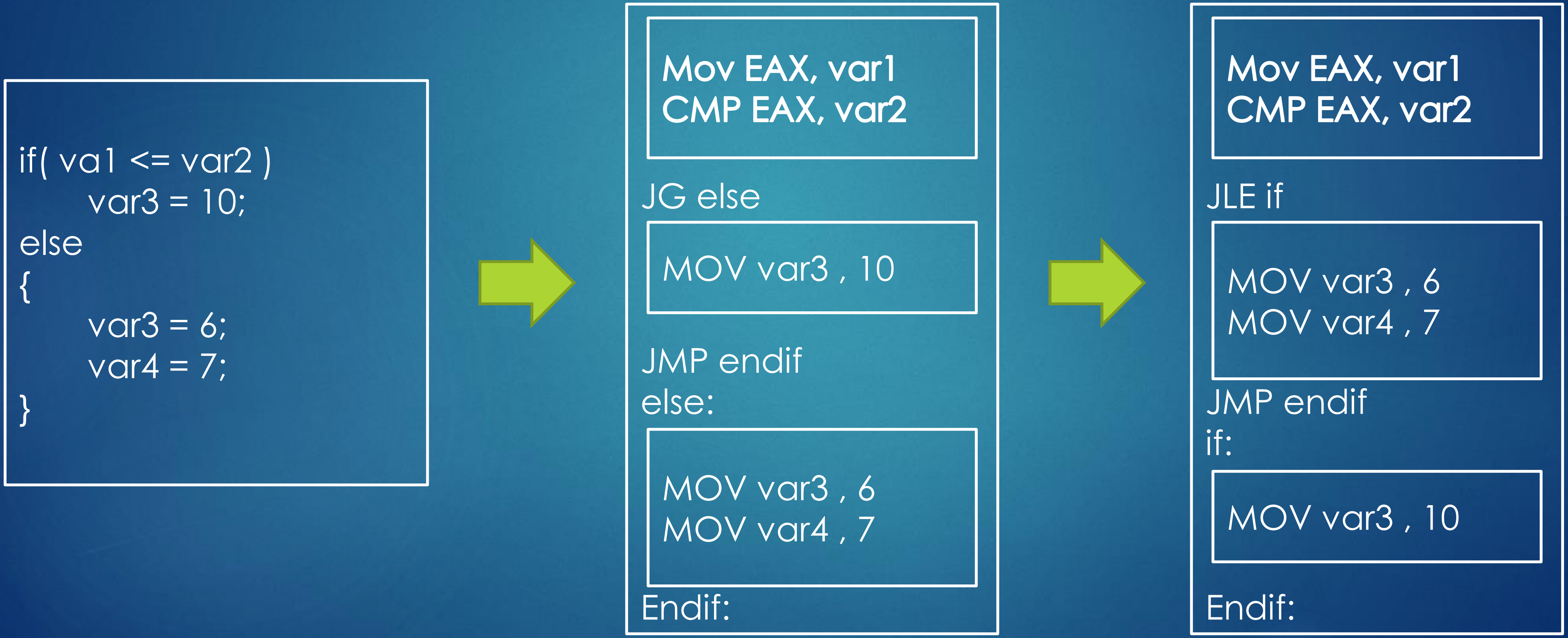


```
if( ebx <= ecx )
{
        eax = 5;
        edx = 6;
}
```

```
CMP EBX, ECX
JA endif
MOV EAX , 5
MOV EDX , 6
Endif:
```

Exercise

- Implement the following pseudocode in assembly language, all values are 32-bit signed integer



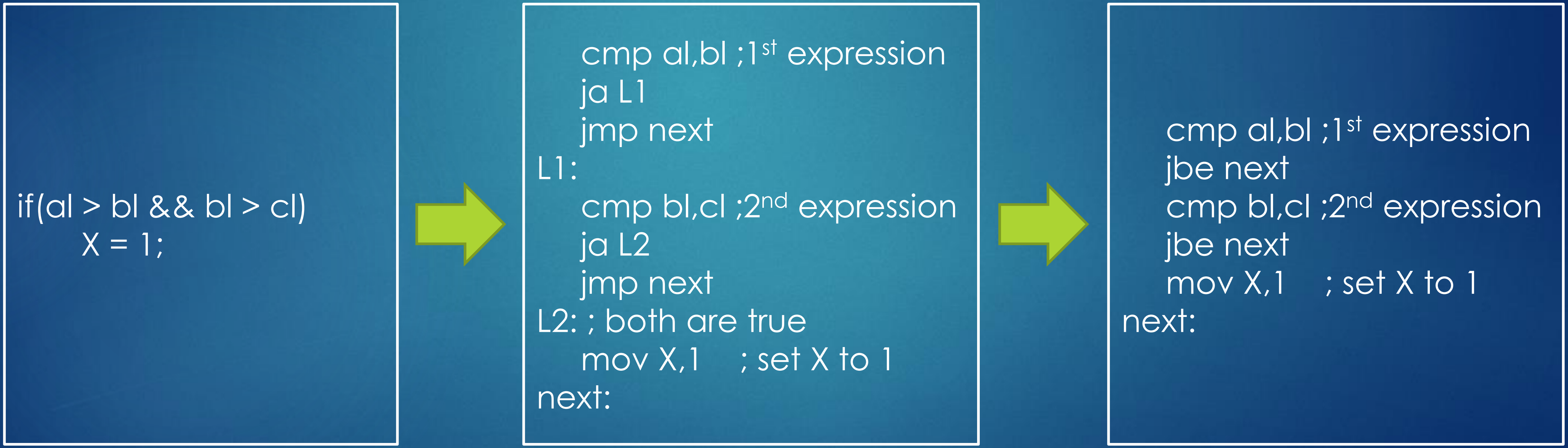
```
if( va1 <= var2 )
        var3 = 10;
else
{
        var3 = 6;
        var4 = 7;
}
```

```
Mov EAX, var1
CMP EAX, var2
JG else
MOV var3 , 10
JMP endif
else:
MOV var3 , 6
MOV var4 , 7
Endif:
```

```
Mov EAX, var1
CMP EAX, var2
JLE if
MOV var3 , 6
MOV var4 , 7
JMP endif
if:
MOV var3 , 10
Endif:
```

Compound Expression with AND

- When implementing the logical AND operator, consider that HLLs use short-circuit evaluation
- In the following example, if the first expression is false, the second expression is skipped



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

```
     cmp al,bl ;1st expression
     ja L1
     jmp next
L1:
     cmp bl,cl ;2nd expression
     ja L2
     jmp next
L2: ; both are true
     mov X,1     ; set X to 1
next:
```

```
     cmp al,bl ;1st expression
     jbe next
     cmp bl,cl ;2nd expression
     jbe next
     mov X,1     ; set X to 1
next:
```

Compound Expression with OR

- In the following example, if the first expression is true, the second expression is skipped

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

```
    cmp al,bl ;1st expression
    jbe L1
    jmp L2
L1:
    cmp bl,cl ;2nd expression
    jbe next
L2: ; both are true
    mov X,1     ; set X to 1
next:
```

```
    cmp al,bl ;1st expression
    ja L1
    cmp bl,cl ;2nd expression
    jbe next
L1:
    mov X,1     ; set X to 1
next:
```

EXERCISE

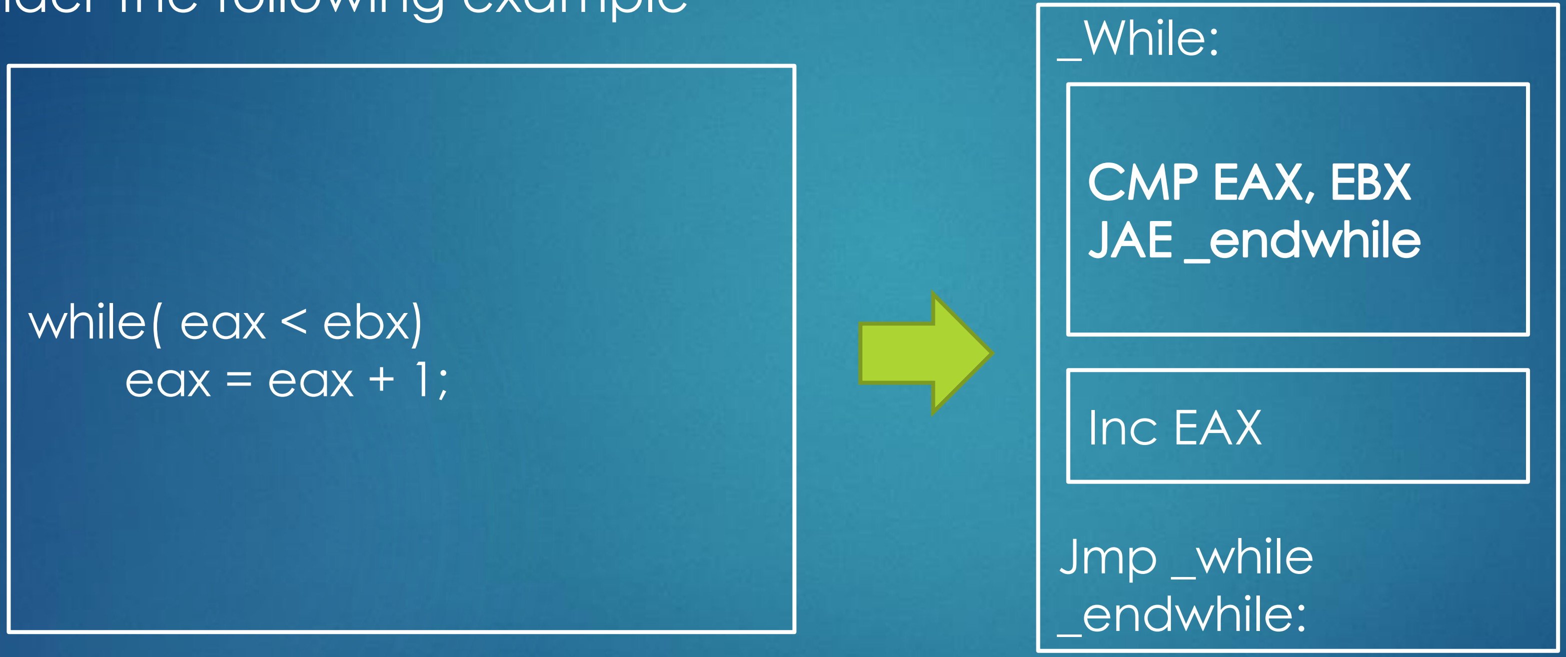
- Implement the following pseudocode in assembly language, all values are unsigned:

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

```
CMP EBX, ECX
JA endif
CMP ECX, EDX
JBE endif

MOV EAX , 5
MOV EDX , 6

Endif:
```

While Loop

- A WHILE loop is really an IF statement followed by the body of the loop followed by an unconditional jump to the top of the loop
- Consider the following example



```
while( eax < ebx)
      eax = eax + 1;
```

```
_While:
  CMP EAX, EBX
  JAE _endwhile
  Inc EAX
Jmp _while
_endwhile:
```

Exercise

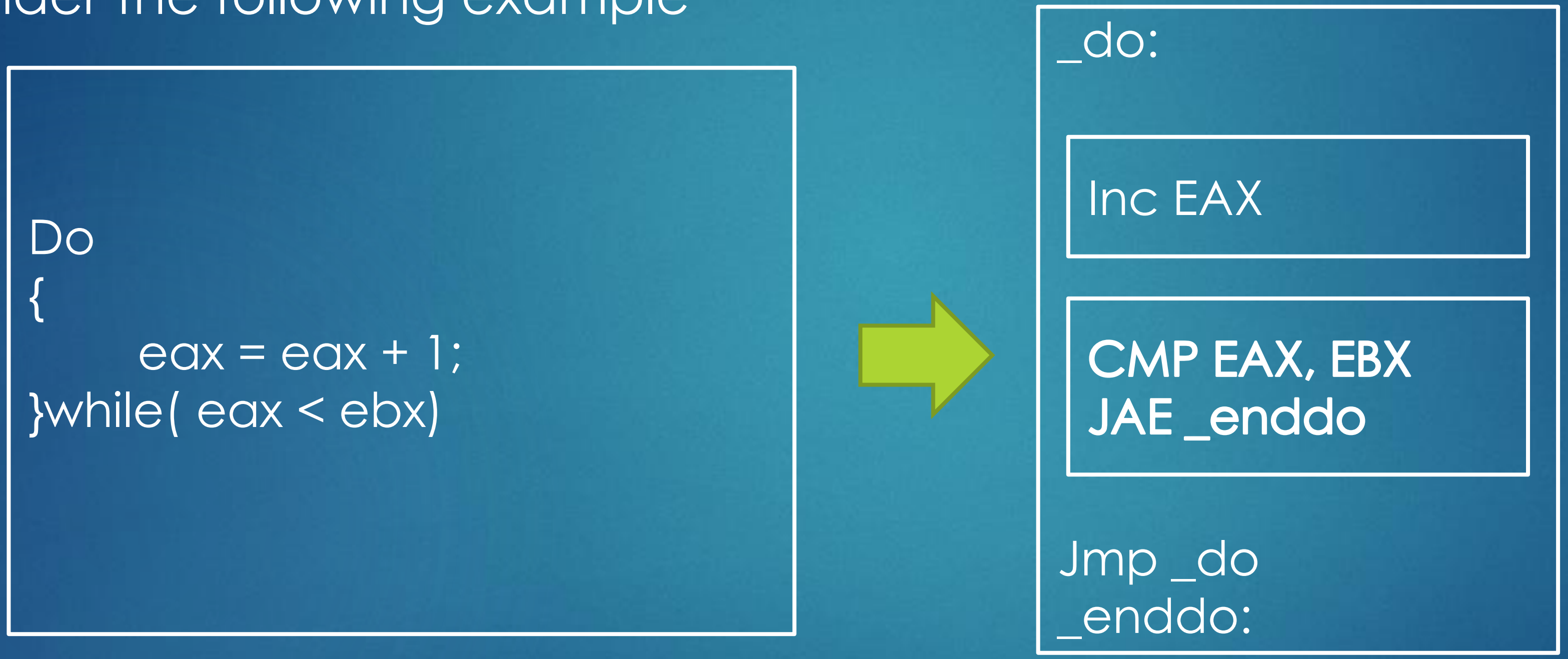
- Implement the following loop, using unsigned 32-bit integer

```
while( ebx <= val1)
{
        eax++;
        if (ebx == ecx)
                X=2;
        else
                X=3;
}
```

```
_While:

 CMP EBX, val1
 JA _endwhile

 INC EAX
 CMP EBX, ECX
 JNE else
 MOV X, 2
 Jmp _while
 else:
 MOV X, 3

Jmp _while
_endwhile:
```

Do Loop

- A DO loop is really an IF statement, here the body of the loop followed by an IF statement to unconditional jump
- Consider the following example



```
Do
{
        eax = eax + 1;
}while( eax < ebx)
```

```
_do:

    Inc EAX

    CMP EAX, EBX
    JAE _enddo

Jmp _do
_enddo:
```

LOOPZ and LOOPE Instruction

- **Syntax:**

  LOOPE destination

  LOOPZ destination

- **Logic:**
  - ECX ← ECX – 1
  - if ECX != 0 and ZF=1, jump to destination
- The destination label must be between -128 and +127 bytes from the location of the following instruction
- Useful when scanning an array for the first element that meets some condition

LOOPNZ and LOOPNE Instruction

- **Syntax:**

  LOOPNE destination

  LOOPNZ destination

- **Logic:**
  - ECX ← ECX – 1
  - if ECX != 0 and ZF=0, jump to destination
- The destination label must be between -128 and +127 bytes from the location of the following instruction
- Useful when scanning an array for the first element that meets some condition

Examples

- The following code finds the first positive value in an array:
- **Solution:**

```
.data
    array SWORD -3,-6,-1,-10,10,30,40,4
    sentinel SWORD 0
.code
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 from stack
loopnz next                              ; continue loop
jnz quit                                 ; none found
sub esi,TYPE array                       ; ESI points to value
quit:
```

Examples

- Locate the first nonzero value in the array. If none is found, let ESI point to the sentinel value
- **Solution:**

```
.data
    array SWORD 50 DUP (?)
    sentinel SWORD 0
.code
mov esi , OFFSET array
mov ecx , LENGTHOF array
next:
    cmp WORD PTR [esi] , 0                  ; check for zero
    pushfd                                  ; push flags on stack
    add esi , TYPE array
    popfd                                   ; pop flags from stack
loopz next                                  ; continue loop
jz quit                                     ; none found
sub esi,TYPE array                          ; ESI points to value
quit:
```