

Conditional Processing

Computer Organization & Assembly Language Programming

Dr Adnan Gutub

aagutub 'at' uqu.edu.sa

[Adapted from slides of Dr. Kip Irvine: Assembly Language for Intel-Based Computers]

Most Slides contents have been arranged by Dr Muhamed Mudawar & Dr Aiman El-Maleh from Computer Engineering Dept. at KFUPM

Presentation Outline

- ❖ **Boolean and Comparison Instructions**
- ❖ Conditional Jumps
- ❖ Conditional Loop Instructions
- ❖ Translating Conditional Structures
- ❖ Indirect Jump and Table-Driven Selection
- ❖ Application: Sorting an Integer Array

AND Instruction

thường dùng để xóa 1 số bit được chọn

- ❖ Bitwise AND between each pair of matching bits

AND *destination, source* `des &= src`

- ❖ Following operand combinations are allowed

AND *reg, reg*

AND *reg, mem*

AND *reg, imm*

AND *mem, reg*

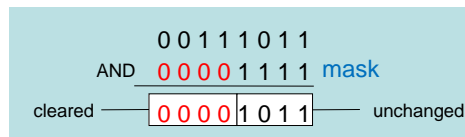
AND *mem, imm*

Operands can be
8, 16, or 32 bits
and they must be
of the same size

AND

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

- ❖ AND instruction is often used to
clear selected bits



Conditional Processing

Computer Organization & Assembly Language Programming

slide 3/55

Converting Characters to Uppercase

- ❖ AND instruction can convert characters to uppercase

`lower = upper + 32`

'a' = 0 1 **1** 0 0 0 0 1 'b' = 0 1 **1** 0 0 0 1 0

'A' = 0 1 **0** 0 0 0 0 1 'B' = 0 1 **0** 0 0 0 1 0

- ❖ Solution: Use the AND instruction to **clear bit 5**

```
mov ecx, LENGTHOF mystring
mov esi, OFFSET mystring
L1: and BYTE PTR [esi], 11011111b ; clear bit 5
    inc esi
    loop L1
```

Conditional Processing

Computer Organization & Assembly Language Programming

slide 4/55

OR Instruction

dùng để set (bật) một số bit được chọn

- ❖ Bitwise OR operation between each pair of matching bits

OR *destination, source* *des |= src*

- ❖ Following operand combinations are allowed

OR

OR *reg, reg*

OR *reg, mem*

OR *reg, imm*

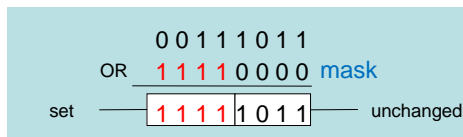
OR *mem, reg*

OR *mem, imm*

Operands can be
8, 16, or 32 bits
and they must be
of the same size

x	y	x ∨ y
0	0	0
0	1	1
1	0	1
1	1	1

- ❖ OR instruction is often used to
set selected bits



Conditional Processing

Computer Organization & Assembly Language Programming

slide 5/55

Converting Characters to Lowercase

- ❖ OR instruction can convert characters to lowercase

'A' = 01000001 'B' = 01000010

'a' = 01100001 'b' = 01100010

- ❖ Solution: Use the OR instruction to **set bit 5**

```

mov ecx, LENGTHOF mystring
mov esi, OFFSET mystring
L1: or  BYTE PTR [esi], 20h      ; set bit 5
    inc esi
    loop L1
    
```

20h = 100 000b

Conditional Processing

Computer Organization & Assembly Language Programming

slide 6/55

Converting Binary Digits to ASCII

- ❖ OR instruction can convert a binary digit to ASCII

0 = 00 **00** 0000 1 = 00 **00** 0001

'0' = 00 **11** 0000 '1' = 00 **11** 0001

- ❖ Solution: Use the OR instruction to **set bits 4 and 5**

`or al, 30h ; Convert binary digit 0 to 9 to ASCII`

- ❖ What if we want to convert an ASCII digit to binary?

- ❖ Solution: Use the AND instruction to **clear bits 4 to 7**

`and al, 0Fh ; Convert ASCII '0' to '9' to binary`

Conditional Processing

Computer Organization & Assembly Language Programming

slide 7/55

XOR Instruction

dùng đảo ngược bit 1<-->0

- ❖ Bitwise XOR between each pair of matching bits

`XOR destination, source` $des \wedge= src$

- ❖ Following operand combinations are allowed

XOR

`XOR reg, reg`

`XOR reg, mem`

`XOR reg, imm`

`XOR mem, reg`

`XOR mem, imm`

Operands can be
8, 16, or 32 bits
and they must be
of the same size

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

- ❖ XOR instruction is often used to **invert selected bits**

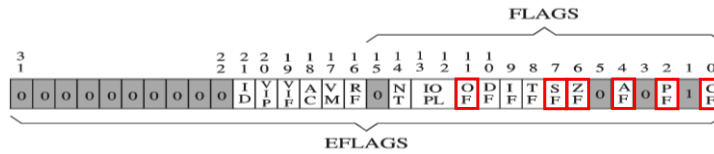
0 0 1 1 1 0 1 1
XOR 1 1 1 1 0 0 0 0 mask
inverted — 1 1 0 0 1 0 1 1 — unchanged

Conditional Processing

Computer Organization & Assembly Language Programming

slide 8/55

Affected Status Flags



The six status flags are affected

1. Carry Flag: **Cleared** by AND, OR, and XOR $CF=0$
2. Overflow Flag: **Cleared** by AND, OR, and XOR $OF=0$
3. Sign Flag: Copy of the **sign bit** in result $SF=\text{bit dấu của kết quả}$
4. Zero Flag: Set when result is **zero**
5. Parity Flag: Set when parity in least-significant byte is **even** $PF=1 \text{ nếu tổng số bit 1 của kq là chẵn}$
6. Auxiliary Flag: **Undefined** by AND, OR, and XOR

Conditional Processing

Computer Organization & Assembly Language Programming

slide 9/55

String Encryption Program

❖ Tasks:

- ❖ Input a message (string) from the user
- ❖ Encrypt the message
- ❖ Display the encrypted message
- ❖ Decrypt the message
- ❖ Display the decrypted message

❖ Sample Output

```
Enter the plain text: Attack at dawn.
Cipher text: «ççÄîä-Äç-îÄÿü-Gs mã hóa
Decrypted: Attack at dawn.
```

Conditional Processing

Computer Organization & Assembly Language Programming

slide 10/55

Encrypting a String

```
KEY      = 239                ; Can be any byte value
BUFMAX   = 128
.data
buffer   BYTE   BUFMAX+1 DUP(0)
bufSize  DWORD  BUFMAX
```

The following loop uses the XOR instruction to transform every character in a string into a new value

```

    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
    loop L1

```

TEST Instruction

- ❖ Bitwise AND operation between each pair of bits
- TEST destination, source** flag=des & src (ko lưu kết quả)
- ❖ The flags are affected similar to the AND Instruction
 - ❖ However, TEST does NOT modify the destination operand
 - ❖ TEST instruction can check several bits at once
- ✧ Example: Test whether bit 0 or bit 3 is set in AL
 - ✧ Solution: test al, 00001001b ; test bits 0 & 3
 mask
 - ✧ We only need to check the zero flag
- ZF=1
; If zero flag => both bits 0 and 3 are clear
- ; If Not zero => either bit 0 or 3 is set
ZF=0

NOT Instruction

- ❖ Inverts all the bits in a destination operand

NOT *destination*

- ❖ Result is called the **1's complement**

- ❖ Destination can be a register or memory

NOT *reg*

NOT *mem*

NOT 0 0 1 1 1 0 1 1
 1 1 0 0 0 1 0 0 ——— inverted

NOT

X	$\neg X$
F	T
T	F

- ❖ None of the Flags is affected by the NOT instruction

Conditional Processing

Computer Organization & Assembly Language Programming

slide 13/55

CMP Instruction

dùng để thiết lập các cờ

- ❖ CMP (Compare) instruction performs a **subtraction**

Syntax: **CMP *destination*, *source***

Computes: ***destination* - *source***

- ❖ Destination operand is NOT modified
- ❖ All six flags: OF, CF, SF, ZF, AF, and PF are affected
- ❖ CMP uses the same operand combinations as SUB
 - ❖ Operands can be 8, 16, or 32 bits and must be of the same size
- ❖ Examples: assume EAX = 5, EBX = 10, and ECX = 5

```
cmp eax, ebx      ; OF=0, CF=1, SF=1, ZF=0
cmp eax, ecx      ; OF=0, CF=0, SF=0, ZF=1
```

Conditional Processing

Computer Organization & Assembly Language Programming

slide 14/55

Unsigned Comparison

- ❖ CMP can perform unsigned and signed comparisons
 - ✧ The *destination* and *source* operands can be unsigned or signed
- ❖ For unsigned comparison, we examine ZF and CF flags

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

To check for equality, it is enough to check ZF flag

- ❖ CMP does a subtraction and CF is the **borrow** flag

CF = 1 if and only if **unsigned** destination < **unsigned** source

- ❖ Assume AL = 5 and BL = -1 = FFh

cmp al, bl ; Sets carry flag CF = 1

Conditional Processing

Computer Organization & Assembly Language Programming

slide 15/55

Signed Comparison

- ❖ For signed comparison, we examine SF, OF, and ZF

Signed Comparison	Flags
signed destination < signed source	SF ≠ OF
signed destination > signed source	SF = OF, ZF = 0
destination = source	ZF = 1

- ❖ Recall for subtraction, the overflow flag is set when ...

✧ Operands have different signs and result sign ≠ destination sign

- ❖ CMP AL, BL (consider the four cases shown below)

Case 1	AL = 80	BL = 50	OF = 0	SF = 0	AL > BL
Case 2	AL = -80	BL = -50	OF = 0	SF = 1	AL < BL
Case 3	AL = 80	BL = -50	OF = 1	SF = 1	AL > BL
Case 4	AL = -80	BL = 50	OF = 1	SF = 0	AL < BL

Conditional Processing

Computer Organization & Assembly Language Programming

slide 16/55

Next ...

- ❖ Boolean and Comparison Instructions
- ❖ **Conditional Jumps**
- ❖ Conditional Loop Instructions
- ❖ Translating Conditional Structures
- ❖ Indirect Jump and Table-Driven Selection
- ❖ Application: Sorting an Integer Array

Conditional Structures

- ❖ No high-level control structures in assembly language
- ❖ Comparisons and conditional jumps are used to ...
 - ✧ Implement conditional structures such as IF statements
 - ✧ Implement conditional loops
- ❖ Types of Conditional Jump Instructions
 - ✧ Jumps based on specific flags
 - ✧ Jumps based on equality
 - ✧ Jumps based on the value of CX or ECX
 - ✧ Jumps based on unsigned comparisons
 - ✧ Jumps based on signed comparisons

Jumps Based on Specific Flags

- ❖ Conditional Jump Instruction has the following syntax:

Jcond destination ; cond is the jump condition

- ❖ Destination

Destination Label

- ❖ Prior to 386

Jump must be within
-128 to +127 bytes
from current location

- ❖ IA-32

32-bit offset permits
jump anywhere in
memory

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

Conditional Processing

Computer Organization & Assembly Language Programming

slide 19/55

Nếu thỏa đk thì nhảy đến
des
Nếu ko thì thực hiện lệnh
kế tiếp

Jumps Based on Equality

Mnemonic	Description
JE	Jump if equal (<i>leftOp = rightOp</i>)
JNE	Jump if not equal (<i>leftOp ≠ rightOp</i>)
JCXZ	Jump if CX = 0
JECXZ	Jump if ECX = 0

- ❖ JE is equivalent to JZ

- ❖ JNE is equivalent to JNZ

- ❖ JECXZ

Checked once at the beginning

Terminate a loop if ECX is zero

```

jecxz L2 ; exit loop
L1: . . . ; loop body
loop L1
L2:
    
```

Conditional Processing

Computer Organization & Assembly Language Programming

slide 20/55

Examples of Jump on Zero

- ❖ Task: Check whether integer value in EAX is even

Solution: TEST whether the least significant bit is 0

If zero, then EAX is even, otherwise it is odd

```
test eax, 1      ; test bit 0 of eax
jz  EvenVal      ; jump if Zero flag is set
```

- ❖ Task: Jump to label L1 if bits 0, 1, and 3 in AL are all set

Solution:

```
and al, 00001011b ; clear bits except 0,1,3
cmp al, 00001011b ; check bits 0,1,3
je  L1            ; all set? jump to L1
```

Conditional Processing

Computer Organization & Assembly Language Programming

slide 21/55

Jumps 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 $leftOp \geq rightOp$)
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 $leftOp \leq rightOp$)
JNA	Jump if not above (same as JBE)

Task: Jump to a label if **unsigned** EAX is less than EBX

Solution:

```
cmp eax, ebx
jb  IsBelow
```

```
JB condition
CF = 1
```

Conditional Processing

Computer Organization & Assembly Language Programming

slide 22/55

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 $leftOp \geq rightOp$)
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 $leftOp \leq rightOp$)
JNG	Jump if not greater (same as JLE)

Task: Jump to a label if **signed** EAX is less than EBX

Solution:

```
cmp eax, ebx
jl  IsLess
```

JL condition
OF \neq SF

Conditional Processing

Computer Organization & Assembly Language Programming

slide 23/55

Compare and Jump Examples

Jump to L1 if **unsigned** EAX is **greater than** Var1

Solution:

```
cmp eax, Var1
ja L1
```

JA condition
CF = 0, ZF = 0

Jump to L1 if **signed** EAX is **greater than** Var1

Solution:

```
cmp eax, Var1
jg L1
```

JG condition
OF = SF, ZF = 0

Jump to L1 if **signed** EAX is **greater than or equal to** Var1

Solution:

```
cmp eax, Var1
jge L1
```

JGE condition
OF = SF

Conditional Processing

Computer Organization & Assembly Language Programming

slide 24/55

Computing the Max and Min

- ❖ Compute the **Max** of **unsigned** EAX and EBX

Solution:

```
mov Max, eax      ; assume Max = eax
cmp Max, ebx
jae done
mov Max, ebx      ; Max = ebx
done:
```

- ❖ Compute the **Min** of **signed** EAX and EBX

Solution:

```
mov Min, eax      ; assume Min = eax
cmp Min, ebx
jle done
mov Min, ebx      ; Min = ebx
done:
```

Conditional Processing

Computer Organization & Assembly Language Programming

slide 25/55

Application: Sequential Search

```
; Receives: esi = array address
;           ecx = array size
;           eax = search value
; Returns:  esi = address of found element

search PROC USES ecx
    jecxz notfound
L1:
    cmp [esi], eax ; array element = search value?
    je  found      ; yes? found element
    add esi, 4      ; no? point to next array element
    loop L1
notfound:
    mov esi, 0      ; if not found then esi = 0
found:
    ret             ; if found, esi = element address
search ENDP
```

Conditional Processing

Computer Organization & Assembly Language Programming

slide 26/55

BT Instruction

- ❖ BT = Bit Test Instruction
- ❖ Syntax:
 - BT *r/m16, r16*
 - BT *r/m32, r32*
 - BT *r/m16, imm8*
 - BT *r/m32, imm8*
- ❖ Copies bit *n* from an operand into the Carry flag
- ❖ Example: jump to label L1 if bit 9 is set in AX register

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

Conditional Processing

Computer Organization & Assembly Language Programming

slide 27/55

Next ...

- ❖ Boolean and Comparison Instructions
- ❖ Conditional Jumps
- ❖ **Conditional Loop Instructions**
- ❖ Translating Conditional Structures
- ❖ Indirect Jump and Table-Driven Selection
- ❖ Application: Sorting an Integer Array

Conditional Processing

Computer Organization & Assembly Language Programming

slide 28/55

LOOPZ and LOOPE

❖ Syntax:

LOOPE *destination*

LOOPZ *destination*

❖ Logic:

✧ $ECX = ECX - 1$

✧ if $ECX > 0$ and $ZF=1$, jump to *destination*

❖ Useful when scanning an array for the first element that does not match a given value.

LOOPNZ and LOOPNE

❖ Syntax:

LOOPNZ *destination*

LOOPNE *destination*

❖ Logic:

✧ $ECX \leftarrow ECX - 1$;

✧ if $ECX > 0$ and $ZF=0$, jump to *destination*

❖ Useful when scanning an array for the first element that matches a given value.

LOOPZ Example

The following code finds the first negative value in an array

```
.data
array SWORD 17,10,30,40,4,-5,8    kích thước 2bytes
.code
    mov esi, OFFSET array - 2    ; start before first
    mov ecx, LENGTHOF array    ; loop counter
L1:
    add esi, 2                    ; point to next element
    test WORD PTR [esi], 8000h    ; test sign bit
    loopz L1                    ; ZF = 1 if value >= 0
    jnz found                    ; found negative value
notfound:
    . . .                        ; ESI points to last array element
found:
    . . .                        ; ESI points to first negative value
```

label

value >= 0 nên
sign bit = 0, ZF=1

Conditional Processing

Computer Organization & Assembly Language Programming

slide 31/55

Your Turn ...

Locate the first zero value in an array

If none is found, let ESI be initialized to 0

```
.data
array SWORD -3,7,20,-50,10,0,40,4
.code
    mov esi, OFFSET array - 2    ; start before first
    mov ecx, LENGTHOF array    ; loop counter
L1:
    add esi, 2                    ; point to next element
    cmp WORD PTR [esi], 0        ; check for zero
    loopne L1                    ; continue if not zero
    JE Found
    XOR ESI, ESI
Found:
```

so sánh từng phần
tử với 0; nếu pt!=0
thì ZF=0, nếu
pt==0 thì ZF=1

Conditional Processing

Computer Organization & Assembly Language Programming

slide 32/55

Next ...

- ❖ Boolean and Comparison Instructions
- ❖ Conditional Jumps
- ❖ Conditional Loop Instructions
- ❖ **Translating Conditional Structures**
- ❖ Indirect Jump and Table-Driven Selection
- ❖ Application: Sorting an Integer Array

Block-Structured IF Statements

- ❖ IF statement in high-level languages (such as C or Java)
 - ✧ Boolean expression (evaluates to true or false)
 - ✧ List of statements performed when the expression is true
 - ✧ Optional list of statements performed when expression is false
- ❖ Task: Translate IF statements into assembly language
- ❖ Example:

```
if( var1 == var2 )  
    X = 1;  
else  
    X = 2;
```

```
mov eax,var1  
cmp eax,var2  
jne elsepart  
mov X,1  
jmp next  
elsepart:  
    mov X,2  
next:
```

Your Turn ...

- ❖ Translate the IF statement to assembly language
- ❖ All values are **unsigned**

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

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

There can be multiple correct solutions

Your Turn ...

- ❖ Implement the following IF in assembly language
- ❖ All variables are **32-bit signed** integers

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

```
mov eax,var1
cmp eax,var2
jle ifpart
mov var3,6
mov var4,7
jmp next
ifpart:

    mov var3,10
next:
```

There can be multiple correct solutions

Compound Expression with AND

- ❖ HLLs use **short-circuit evaluation** for logical AND
- ❖ If first expression is **false**, second expression is **skipped**

```
if ((a1 > b1) && (b1 > c1)) {X = 1;}
```

; One Possible Implementation ...

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

Conditional Processing

Computer Organization & Assembly Language Programming

slide 37/55

Better Implementation for AND

```
if ((a1 > b1) && (b1 > c1)) {X = 1;}
```

The following implementation uses less code

By reversing the relational operator, We allow the program to **fall through** to the second expression

Number of instructions is reduced from 7 to 5

```
    cmp al, bl      ; first expression...
    jbe next        ; quit if false
    cmp bl, cl      ; second expression...
    jbe next        ; quit if false
    mov X, 1        ; both are true
next:
```

Conditional Processing

Computer Organization & Assembly Language Programming

slide 38/55

Your Turn ...

- ❖ Implement the following IF 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:
```

Conditional Processing

Computer Organization & Assembly Language Programming

slide 39/55

Application: IsDigit Procedure

Receives a character in AL

Sets the Zero flag if the character is a decimal digit

```
if (al >= '0' && al <= '9') {ZF = 1;}
```

```
IsDigit PROC  
    cmp     al,'0'           ; AL < '0' ?  
    jb      L1              ; yes? ZF=0, return  
    cmp     al,'9'           ; AL > '9' ?  
    ja      L1              ; yes? ZF=0, return  
    test    al, 0            ; ZF = 1  
L1: ret  
IsDigit ENDP
```

Conditional Processing

Computer Organization & Assembly Language Programming

slide 40/55

Compound Expression with OR

- ❖ HLLs use **short-circuit evaluation** for logical OR
- ❖ If first expression is **true**, second expression is **skipped**

```
if ((a1 > b1) || (b1 > c1)) {X = 1;}
```

- ❖ Use **fall-through** to keep the code as short as possible

```
cmp al,b1      ; is AL > BL?
ja  L1         ; yes, execute if part
cmp bl,cl      ; no: is BL > CL?
jbe next       ; no: skip if part
L1: mov X,1     ; set X to 1
next:
```

Conditional Processing

Computer Organization & Assembly Language Programming

slide 41/55

WHILE Loops

A WHILE loop can be viewed as

IF statement followed by

The body of the loop, followed by

Unconditional jump to the top of the loop

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

This is a possible implementation:

```
top: cmp eax,ebx      ; eax < ebx ?
     jae next         ; false? then exit loop
     inc eax          ; body of loop
     jmp top          ; repeat the loop
next:
```

Conditional Processing

Computer Organization & Assembly Language Programming

slide 42/55

Your Turn ...

Implement the following loop, assuming **unsigned** integers

```
while (ebx <= var1) {  
    ebx = ebx + 5;  
    var1 = var1 - 1  
}
```

```
top: cmp ebx,var1      ; ebx <= var1?  
    ja next          ; false? exit loop  
    add ebx,5         ; execute body of loop  
    dec var1  
    jmp top           ; repeat the loop  
next:
```

Conditional Processing

Computer Organization & Assembly Language Programming

slide 43/55

Yet Another Solution for While

Check the loop condition at the end of the loop

No need for JMP, loop body is reduced by 1 instruction

```
while (ebx <= var1) {  
    ebx = ebx + 5;  
    var1 = var1 - 1  
}
```

```
    cmp ebx,var1      ; ebx <= var1?  
    ja next          ; false? exit loop  
top: add ebx,5         ; execute body of loop  
    dec var1  
    cmp ebx, var1     ; ebx <= var1?  
    jbe top           ; true? repeat the loop  
next:
```

Conditional Processing

Computer Organization & Assembly Language Programming

slide 44/55

Next ...

- ❖ Boolean and Comparison Instructions
- ❖ Conditional Jumps
- ❖ Conditional Loop Instructions
- ❖ Translating Conditional Structures
- ❖ Indirect Jump and Table-Driven Selection
- ❖ Application: Sorting an Integer Array

Indirect Jump

- ❖ Direct Jump: Jump to a Labeled Destination
 - ✧ Destination address is a **constant**
 - Address is encoded in the jump instruction
 - Address is an offset relative to EIP (Instruction Pointer)
- ❖ Indirect jump
 - ✧ Destination address is a **variable or register**
 - Address is stored in memory/register
 - Address is absolute
- ❖ Syntax: **JMP *mem32/reg32***
 - ✧ 32-bit absolute address is stored in *mem32/reg32* for FLAT memory
- ❖ Indirect jump is used to implement switch statements

Switch Statement

❖ Consider the following switch statement:

```
Switch (ch) {  
    case '0': exit();  
    case '1': count++; break;  
    case '2': count--; break;  
    case '3': count += 5; break;  
    case '4': count -= 5; break;  
    default : count = 0;  
}
```

❖ How to translate above statement into assembly code?

❖ We can use a sequence of compares and jumps

❖ A better solution is to use the indirect jump

Conditional Processing

Computer Organization & Assembly Language Programming

slide 47/55

Implementing the Switch Statement

```
case0:  
    exit  
case1:  
    inc count  
    jmp exitswitch  
case2:  
    dec count  
    jmp exitswitch  
case3:  
    add count, 5  
    jmp exitswitch  
case4:  
    sub count, 5  
    jmp exitswitch  
default:  
    mov count, 0  
exitswitch:
```

There are many case labels. How to jump to the correct one?

Answer: Define a **jump table** and use **indirect jump** to jump to the correct label

Conditional Processing

Computer Organization & Assembly Language Programming

slide 48/55

Jump Table and Indirect Jump

- ❖ Jump Table is an array of double words
 - ✧ Contains the case labels of the switch statement
 - ✧ Can be defined inside the same procedure of switch statement

```
jump table DWORD case0,  
               case1,  
               case2,  
               case3,  
               case4
```

} Assembler converts labels to addresses

- ❖ Indirect jump uses jump table to jump to selected label

```
movzx eax, ch           ; move ch to eax  
sub    eax, '0'         ; convert ch to a number  
cmp    eax, 4           ; eax > 4 ?  
ja     default          ; default case  
jmp    jump table[eax*4] ; Indirect jump
```

Conditional Processing

Computer Organization & Assembly Language Programming

slide 49/55

Next ...

- ❖ Boolean and Comparison Instructions
- ❖ Conditional Jumps
- ❖ Conditional Loop Instructions
- ❖ Translating Conditional Structures
- ❖ Indirect Jump and Table-Driven Selection
- ❖ Application: Sorting an Integer Array

Conditional Processing

Computer Organization & Assembly Language Programming

slide 50/55

Bubble Sort

❖ Consider sorting an array of 5 elements: 5 1 3 2 4

First Pass (4 comparisons)

	5	1	3	2	4	
Compare 5 with 1 and swap:	1	5	3	2	4	(swap)
Compare 5 with 3 and swap:	1	3	5	2	4	(swap)
Compare 5 with 2 and swap:	1	3	2	5	4	(swap)
Compare 5 with 4 and swap:	1	3	2	4	5	(swap)

largest

Second Pass (3 comparisons)

Compare 1 with 3 (No swap):	1	3	2	4	5	(no swap)
Compare 3 with 2 and swap:	1	2	3	4	5	(swap)
Compare 3 with 4 (No swap):	1	2	3	4	5	(no swap)

Third Pass (2 comparisons)

Compare 1 with 2 (No swap):	1	2	3	4	5	(no swap)
Compare 2 with 3 (No swap):	1	2	3	4	5	(no swap)

No swapping during 3rd pass ⇒ array is now sorted

Conditional Processing

Computer Organization & Assembly Language Programming

slide 51/55

Bubble Sort Algorithm

❖ Algorithm: Sort *array* of given *size*

```

bubbleSort(array, size) {
    comparisons = size
    do {
        comparisons--;
        sorted = true;    // assume initially
        for (i = 0; i < comparisons; i++) {
            if (array[i] > array[i+1]) {
                swap(array[i], array[i+1]);
                sorted = false;
            }
        }
    } while (! sorted)
}
    
```

Conditional Processing

Computer Organization & Assembly Language Programming

slide 52/55

Bubble Sort Procedure - Slide 1 of 2

```
;-----  
; bubbleSort: Sorts a DWORD array in ascending order  
;               Uses the bubble sort algorithm  
; Receives:  ESI = Array Address  
;           ECX = Array Length  
; Returns:   Array is sorted in place  
;-----  
  
bubbleSort PROC USES eax ecx edx  
outerloop:  
    dec  ECX          ; ECX = comparisons  
    jz   sortdone     ; if ECX == 0 then we are done  
    mov  EDX, 1       ; EDX = sorted = 1 (true)  
    push ECX          ; save ECX = comparisons  
    push ESI          ; save ESI = array address
```

Conditional Processing

Computer Organization & Assembly Language Programming

slide 53/55

Bubble Sort Procedure - Slide 2 of 2

```
innerloop:  
    mov  EAX,[ESI]  
    cmp  EAX,[ESI+4]   ; compare [ESI] and [ESI+4]  
    jle  increment     ; [ESI]<=[ESI+4]? don't swap  
    xchg EAX,[ESI+4]   ; swap [ESI] and [ESI+4]  
    mov  [ESI],EAX  
    mov  EDX,0         ; EDX = sorted = 0 (false)  
increment:  
    add  ESI,4         ; point to next element  
    loop innerloop     ; end of inner loop  
    pop  ESI           ; restore ESI = array address  
    pop  ECX           ; restore ECX = comparisons  
    cmp  EDX,1         ; sorted == 1?  
    jne  outerloop     ; No? loop back  
sortdone:  
    ret               ; return  
bubbleSort ENDP
```

Conditional Processing

Computer Organization & Assembly Language Programming

slide 54/55

Summary

- ❖ Bitwise instructions (AND, OR, XOR, NOT, TEST)
 - ✧ Manipulate individual bits in operands
- ❖ CMP: compares operands using implied subtraction
 - ✧ Sets condition flags for later conditional jumps and loops
- ❖ Conditional Jumps & Loops
 - ✧ Flag values: JZ, JNZ, JC, JNC, JO, JNO, JS, JNS, JP, JNP
 - ✧ Equality: JE(JZ), JNE (JNZ), JCXZ, JECXZ
 - ✧ Signed: JG (JNLE), JGE (JNL), JL (JNGE), JLE (JNG)
 - ✧ Unsigned: JA (JNBE), JAE (JNB), JB (JNAE), JBE (JNA)
 - ✧ LOOPZ (LOOPE), LOOPNZ (LOOPNE)
- ❖ Indirect Jump and Jump Table