**Lab Session 07**



**Conditional Processing**



**Boolean Instructions**

* **AND**

Boolean AND operation between a source operand and destination operand.

**Syntax:** *AND reg, reg*

*AND reg, mem*

*AND reg, imm*

*AND mem, reg*

*AND mem, imm*

* **OR**

Boolean OR operation between a source operand and destination operand.

**Syntax:** *OR reg, reg*

*OR reg, mem*

*OR reg, imm*

*OR mem, reg*

*OR mem, imm*

* **XOR**

Boolean XOR operation between a source operand and destination operand.

**Syntax:** *XOR reg, reg*

*XOR reg, mem*

*XOR reg, imm*

*XOR mem, reg*

*XOR mem, imm*

* **NOT**

Boolean NOT operation on a destination operand.

**Syntax:** *NOT reg*

*NOT mem*

* **TEST**

Similar to AND operation, except that instead of affecting any operands it sets the FLAGS appropriately.

**Syntax:** *TEST reg, reg*

*TEST reg, mem*

*TEST reg, imm*

*TEST mem, reg*

*TEST mem, imm*

**EXAMPLE # 01:**

.code

mov al, 10101110b ; Clear only bit 3

and al, 11110110b ; AL = 10100110

mov al, 11100011b ; set bit 2

or al, 00000100b ; AL = 11100111

mov al, 10110101b ; 5 bits means odd parity

xor al, 0 ; PF = 0 (PO)

mov al, 10100101b ; 4 bits means even parity

xor al, 0 ; PF = 1 (PE)

mov al, 11110000b

not al ; AL = 00001111b

mov al, 00100101b

test al, 00001001b ; ZF = 0

mov al, 00100101b

test al, 00001000b ; ZF = 1

call DumpRegs

exit

**Set Operations (using Boolean instructions)**

* **Set Complement**

The complement of a set can be achieved through NOT instruction.

* **Set Intersection**

The intersection of two sets can be achieved through AND instruction.

* **Set Union**

The union of two sets can be achieved through OR instruction.

**EXAMPLE # 02:**

.data

setA DWORD 10000000 00000000 00000000 00000111b

setB DWORD 10000001 01010000 00000111 01100011b

msg1 BYTE "A intersection B is: ", 0

msg2 BYTE "A union B is: ", 0

msg3 BYTE "Complement of A is: ", 0

.code

mov eax, setA

and eax, setB ; A intersection B

mov edx, OFFSET msg1

call WriteString

mov ebx, TYPE DWORD

call WriteBinB

call Crlf

mov eax, setA

or eax, setB ; A union B

mov edx, OFFSET msg2

call WriteString

mov ebx, TYPE DWORD

call WriteBinB

call Crlf

mov eax, setA

not eax ; A complement

mov edx, OFFSET msg3

call WriteString

mov ebx, TYPE DWORD

call WriteBinB

exit

**CMP instruction**

CMP (compare) instruction performs an implied subtraction of a source operand from a destination operand for comparison.

For unsigned operands:

* Destination < source ZF = 0 CF = 1
* Destination > source ZF = 0 CF = 0
* Destination = source ZF = 1 CF = 0

For signed operands:

* Destination < source SF ! = OF
* Destination > source SF = OF
* Destination = source ZF = 1

**EXAMPLE # 03:**

.code

mov ax, 5

cmp ax, 10 ; ZF = 0 and CF = 1

mov ax, 1000

cmp ax, 1000 ; ZF = 1 and CF =0

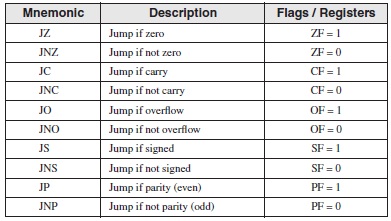
mov si, 106

cmp si, 0 ; ZF = 0 and CF = 0

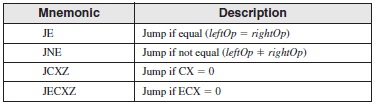
exit

**Conditional Jumps**

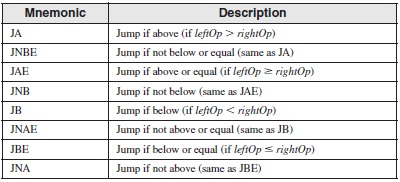
* **Jumps based on Flag values**

****

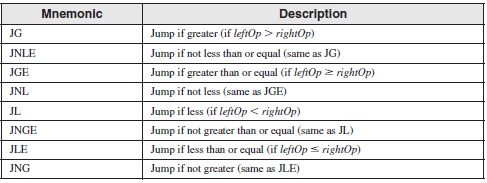
* **Jumps based on Equality**

****

* **Jumps based on unsigned comparisons**

****

* **Jumps based on signed comparisons**

****

**EXAMPLE # 04:**

; This program compares and finds larger of the two integers

.data

var1 DWORD 250

var2 DWORD 125

larger DWORD ?

.code

mov eax, var1

mov larger, eax

mov ebx, var2

cmp eax, ebx

jae L1

mov larger, ebx

L1:

exit

**EXAMPLE # 05:**

; This program compares and finds smallest of the three integers

.data

var1 DWORD 50

var2 DWORD 25

var3 DWORD 103

msg BYTE "The smallest integer is: ", 0

.code

mov eax, var1

cmp eax, var2

jbe L1

mov eax, var2

L1:

cmp eax, var3

jbe L2

mov eax, var3

L2:

mov edx, OFFSET msg

call WriteString

call WriteDec

exit

**EXAMPLE # 06:**

; The following program continues a loop until an alphanumeric key is pressed

.data

char BYTE ?

.code

L1:

mov eax, 10 ; create 10ms delay

call Delay

call ReadKey ; reads a key input

jz L1 ; repeat if no key is pressed

mov char, al ; saves the character

**Conditional Loop Instructions**

**LOOPZ and LOOPE Instructions**

The LOOPZ (loop if zero) instruction works just like the LOOP instruction except that it has one additional condition: the Zero flag must be set in order for control to transfer to the destination label.

The syntax is

LOOPZ *destination*

They perform the following tasks:

ECX = ECX - 1

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

Otherwise, no jump occurs, and control passes to the next instruction.

.code

main proc

mov eax,2

mov ecx,5

l1:

sub eax,1

dec eax

mov eax,ecx

loopz l1

**LOOPNZ and LOOPNE Instructions**

The LOOPNZ (loop if not zero) instruction is the counterpart of LOOPZ.

The syntax is

LOOPNZ *destination*

They perform the following tasks:

ECX = ECX - 1

if ECX > 0 and ZF = 0, jump to destination

Otherwise, nothing happens, and control passes to the next instruction.

.code

main proc

mov eax,3

mov ecx,5

l1:

sub eax,1

loopnz l1