EL213: Computer Org. & Assembly Language Lab

# Lab#07: Conditional Processing

## Agenda

* Boolean and Comparison Instructions
  + AND Instruction
  + OR Instruction
  + XOR Instruction
  + NOT Instruction
  + TEST Instruction
  + CMP Instruction
* Conditional Jumps
  + Jcond Instruction
  + Jump Based on Specific Flags
  + Jump Based on Equality
  + Jump Based on Unsigned Comparisons
  + Jump Based on Signed Comparisons
* BT (Bit Test) Instruction
* Conditional Loop Instructions
  + LOOPZ & LOOPE Instructions
  + LOOPNZ & LOOPNE Instructions

## Boolean and Comparison Instructions

### AND Instruction

The AND instruction performs a boolean (bitwise) AND operation between each pair of matching bits in two operands and places the result in the destination operand:

AND destination,source

The following operand combinations are permitted:

AND reg,reg

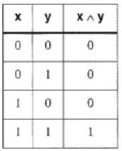
AND reg,mem

AND reg,imm

AND mem, reg

AND mem,imm

The operands can be 8, 16, or 32 bits, and they must be the same size.

****

***Truth Table***

The AND instruction is often used to clear selected bits and preserve others. In the following

example, the upper four bits are cleared and the lower four bits are unchanged:



**Converting Characters to Upper Case**

The AND instruction provides an easy way to translate a letter from lowercase to uppercase. If we compare the ASCII codes of capital A and lowercase a, it becomes clear that only one bit is different:

0 1 1 0 0 0 0 1 = 61h ('a')

0 1 0 0 0 0 0 1 = 41h (' A' )

The rest of the alphabetic characters have the same relationship. If we AND any character with 110111111 binary, all bits are unchanged except for bit 5, which is cleared. In the following example, all characters in an array are converted to uppercase

Include irvine32.inc

.data

array BYTE "hello",0

.code

main PROC

mov ecx,LENGTHOF array

mov esi,OFFSET array

dec ecx

Ll:

and BYTE PTR [esi], 11011111b

inc esi

loop Ll

mov edx, OFFSET array

Call WriteString

Call Crlf

;Alternate method, read character by character

;mov ecx, LENGTHOF array

;mov esi, OFFSET array

;L2:

;mov al, [esi]

;Call WriteChar

;inc esi

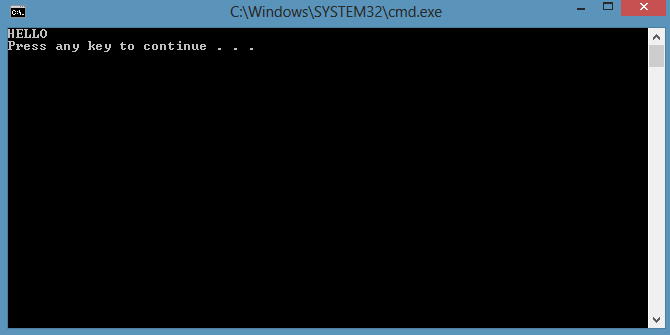
;loop L2

exit

main ENDP

END main

**Output**



***Flags:*** The AND instruction always clears the Overflow and Carry flags. It modifies the Sign, Zero flags according to the value of the destination operand.

### OR Instruction

The OR instruction performs a boolean OR operation between each pair of matching bits in two operands and places the result in the destination operand:

OR destination, source

The OR instruction uses the same operand combinations as the AND instruction:

OR reg, reg

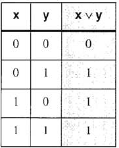
OR reg,mem

OR reg,imm

OR mem,reg

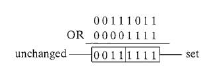
OR mem,imm

The operands can be 8, 16, or 32 bits, and they must be the same size.



***Truth Table***

The OR instruction is often used to set selected bits and preserve others. In the following figure, 3Bh is ORed with OFh. The lower four bits of the result are set and the high four bits are unchanged:



The OR instruction can be used to convert a byte containing an integer between 0 and 9 into an ASCII digit. To do this, you must set bits 4 and 5. If, for example, AL =05h, you can OR it with 30h to convert it to the ASCII code for the digit 5 (35h):

Include irvine32.inc

.data

val byte 5

.code

main PROC

or val,30h

movzx eax,val

call WriteChar

call Crlf

call WriteDec

call Crlf

call WriteHex

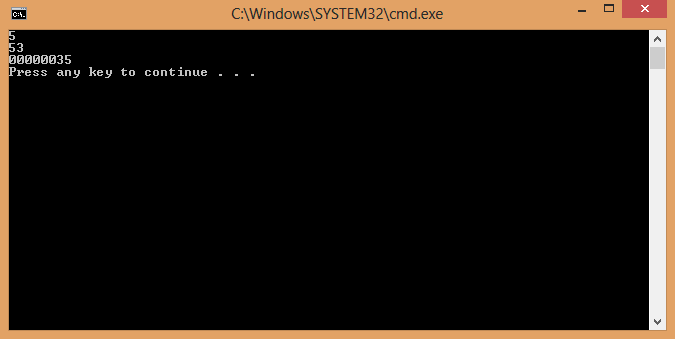
call Crlf

exit

main ENDP

END main

**Output**



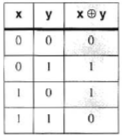
***Flags:*** The OR instruction always clears the Overflow and Carry flags. It modifies the Sign, Zero flags according to the value of the destination operand.

### XOR Instruction

The XOR instruction performs a boolean exclusive-OR operation between each pair of matching bits in two operands, and stores the result in the destination operand:

XOR destination, source

The operands can be 8, 16, or 32 bits.



***Truth Table***

***Flags:*** The XOR instruction always clears the Overflow and Carry flags. It modifies the Sign. Zero . and Parity flags according to the value of the destination operand.

### NOT Instruction

The NOT instruction toggles all bits in an operand. The result is called the one's complement. The following operand types are permitted:

NOT reg

NOT mem

For example, the one's complement of F0h is 0Fh:

mov al,11110000b

not al ; AL = 00001111b

***Flags:*** No flags are affected by the NOT instruction

;for Boolean instructions

XOR EAX,EAX

XOR EBX,EBX

mov al,10101010b

mov bl,01010101b

call DumpRegs

AND bl,al

call DumpRegs

OR bl,al

call DumpRegs

NOT al

call DumpRegs

;call WriteBin

### TEST Instruction

The TEST instruction performs an implied AND operation between each pair of matching bits in two operands and sets the flags accordingly. The only difference between TEST and AND is that TEST does not modify the destination operand. The TEST instruction permits the same operand combinations as the AND instruction. TEST is particularly valuable for finding out if individual bits in an operand are set.

**Example:** ***Testing Multiple Bits***

The TEST instruction can check several bits at once. Suppose we want to know if either bit 0 or bit 3 is set in the AL register. We can use the following instruction to find this out:

test al,00001001b ; test bits 0 and 3

From the following example data sets, we can infer that the Zero flag is set only when all tested bits are clear:

0 0 1 0 0 1 0 1 <- input value

0 0 0 0 1 0 0 1 <- test value

0 0 0 0 0 0 0 1 <- result : ZF = 0

0 0 1 0 0 1 0 0 <- input value

0 0 0 0 1 0 0 1 <- test value

0 0 0 0 0 0 0 0 <- result: ZF = 1

***Flags:*** The TEST instruction always clears the Overflow and Carry flags. It modifies the Sign, Zero, and Parity flags in the same way as the AND instruction.

### CMP Instruction

It compares the destination operand to the source operand by subtracting source operand from a destination operand. No operand is modified.

Remember these combinations for comparison:

mov al,6

cmp al,5 ; ZF=0, CF=0 ; if(al>5)

mov al,4

cmp al,5 ; Carry flag set ; if (al<5)

mov al,5

cmp al,5 ; Zero flag set ; if(al==5)

CMP for Signed Integers is given below

mov al,5

cmp al,-2 ;Sign flag == Overflow flag ;if(al > -2)

mov al,-1

cmp al,5 ;Sign flag != Overflow flag ;if(al < 5)

## Conditional Jumps

### Jcond Instruction

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

*For example*

* JE/ JZ: Jump to a label if the ZF=1 (set)
* JS: Jumps to a label if the SF=1(set)
* JNE/ JNZ: Jump to a label if the ZF=0 (clear)
* JECXZ: Jumps to a label if ECX = 0 (default used by Loop instruction)

cmp al, 0

jz L1 ;jump if ZF = 1

.

.

L1:

### Jump Based on Specific Flags

There conditional jump instructions that act on the basis of the status flags are as given below.



### Jump Based on Equality

Equity based JUMP Based Instructions are given below.



### Jump based on Unsigned Comparisons



### Jump Based on Signed Comparisons



**Scanning an Array**

; Scan an array for the first nonzero value.

INCLUDE Irvine32.inc

.data

intArray SWORD 0,0,0,0,1,20,35,-12,66,4,0

noneMsg BYTE "A non-zero value was not 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 ; compare value to zero

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] ; otherwise, display it

call WriteInt

jmp quit

notFound:

mov edx,OFFSET noneMsg ; display "not found" message

call WriteString

quit:

call crlf

exit

main ENDP

END main

## BT (Bit Test) Instruction

BT instruction copies bit n from an operand into the Carry flag.

BT bitBase, n

The first operand, called the bitBase is not changed.

Example: jump to label L1 if bit 8 is set in the AX register:

Include irvine32.inc

.data

msg\_s byte "Eight bit is set",0

msg\_c byte "Eight bit is clear",0

.code

main PROC

Mov AX, 0000000100000000b

bt AX,8 ; CF = bit 8

call dumpregs

jc L1 ; jump if Carry

mov edx, offset msg\_c

call writestring

call crlf

L1:

mov edx, offset msg\_s

call writestring

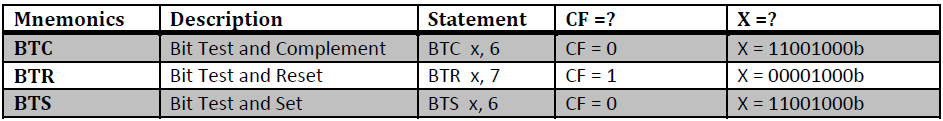
call crlf

exit

main ENDP

END main

For example x = 10001000b



## Conditional Loop Instructions

### LOOPZ & LOOPE Instructions

The LOOPZ (loop if zero) instruction permits a loop to continue while the Zero flag is set and the unsigned value of ECX is greater than zero.

The LOOPE (loop if equal) instruction is equivalent to LOOPZ. Below is the execution logic of LOOPZ and LOOPE:

ECX = ECX - 1

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

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

**Syntax**

LOOPZ destination

LOOPE destination

TITLE LOOPZ / LOOPE

INCLUDE Irvine32.inc

.data

.code

main PROC

XOR EAX,EAX

XOR EBX,EBX

XOR ecx,ecx

XOR ebx,ebx

mov ebx,11d

mov ecx,11d

mov edx,ebx

L1:

mov eax,ecx

call WriteInt

call Crlf

sub ebx,ebx ; sets => ZF=1

mov ebx,edx

LOOPZ L1 ;loop until ECX>0

mov ecx , 11 ; loop counter for LOOPE

L2:

mov eax,ecx

neg eax

call WriteInt

call Crlf

sub ebx,ebx ; sets => ZF=1

mov ebx,edx

LOOPE L2

exit

main ENDP

END main

### LOOPNZ & LOOPNE Instructions

The LOOPNZ (loop if not zero) instruction is the counterpart of LOOPZ. The loop continues while the unsigned value of ECX is greater than zero and the Zero flag is clear.

The LOOPNE (loop if not equal) instruction is equivalent to LOOPNZ. Below is the execution logic of LOOPNZ and LOOPNE:

ECX = ECX - 1

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

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

INCLUDE Irvine32.inc

.data

.code

main PROC

XOR EAX,EAX

XOR EBX,EBX

XOR ecx,ecx

XOR ebx,ebx

mov ebx,11d

mov ecx,11d

mov edx,ebx

L1:

mov eax,ecx

call WriteInt

call Crlf

LOOPNZ L1 ; loop until ECX>0

mov ecx , 11 ; loop counter for LOOPE

L2:

mov eax,ecx

neg eax

call WriteInt

call Crlf

LOOPNE L2

exit

main ENDP

END main