Conditional Processing

COE 205

Computer Organization and Assembly Language

Dr. Aiman El-Maleh

College of Computer Sciences and Engineering

King Fahd University of Petroleum and Minerals

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

TEST Instruction

- Bitwise AND operation between each pair of bits

  TEST *destination, source*

- 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`**
  - We only need to check the zero flag

```
; If zero flag => both bits 0 and 3 are clear
; If Not zero  => either bit 0 or 3 is set
```

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	¬X
F	T
T	F

- None of the Flags is affected by the NOT instruction

CMP Instruction

- 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
```

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
```

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

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 unsigned comparisons
  - Jumps based on signed comparisons
  - Jumps based on the value of CX or ECX

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

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

- 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:
```

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
```

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* >= *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)

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

Solution:

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

```
JB condition
CF = 1
```

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)

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

Solution:

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

```
JL condition
OF ≠ SF
```

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
```

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:
```

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, -1     ; if not found then esi = -1
found:
   ret              ; if found, esi = element address
search ENDP
```