CS2422 Assembly Language & System Programming

October 17, 2006

Today’s Topics

• Conditional Processing
  – If…then…else
  – While…do; Repeat…until
Study Guide

• Suggested Order:
  – Section 6.1: Introduction
  – Section 6.3.1-6.3.4: Conditional Jumps
  – Section 6.5: Conditional Structures
  – Section 6.2: Boolean Operations
  (Optional): Sections 6.3.5, 6.4, 6.6, 6.7

CMP and Jcond Instruction

• The IF statement in C and PASCAL is converted into CMP and Jcond instructions in x86 Assembly:

If (X > op1)
Then
<...>
End If

CMP X, op1
JNG EndIf
<...>
EndIf:
CMP Instruction (1 of 3)

- 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
```

CMP Instruction (2 of 3)

- Example: destination > source
  (both the Zero and Carry flags are clear)

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

The comparisons shown so far were unsigned.
CMP Instruction (3 of 3)

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
```

Jcond Instruction

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

Examples:

– JB, JC jump to a label if the Carry flag is set
– JE, JZ jump to a label if the Zero flag is set
– JS jumps to a label if the Sign flag is set
– JNE, JNZ jump to a label if the Zero flag is clear
– JECXZ jumps to a label if ECX equals 0
## Jumps Based on Specific Flags

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>JZ</td>
<td>Jump if zero</td>
<td>ZF = 1</td>
</tr>
<tr>
<td>JNZ</td>
<td>Jump if not zero</td>
<td>ZF = 0</td>
</tr>
<tr>
<td>JC</td>
<td>Jump if carry</td>
<td>CF = 1</td>
</tr>
<tr>
<td>JNC</td>
<td>Jump if not carry</td>
<td>CF = 0</td>
</tr>
<tr>
<td>JO</td>
<td>Jump if overflow</td>
<td>OF = 1</td>
</tr>
<tr>
<td>JNO</td>
<td>Jump if not overflow</td>
<td>OF = 0</td>
</tr>
<tr>
<td>JS</td>
<td>Jump if signed</td>
<td>SF = 1</td>
</tr>
<tr>
<td>JNS</td>
<td>Jump if not signed</td>
<td>SF = 0</td>
</tr>
<tr>
<td>JP</td>
<td>Jump if parity (even)</td>
<td>PF = 1</td>
</tr>
<tr>
<td>JNP</td>
<td>Jump if not parity (odd)</td>
<td>PF = 0</td>
</tr>
</tbody>
</table>

## Jumps Based on Equality

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JE</td>
<td>Jump if equal (leftOp = rightOp)</td>
</tr>
<tr>
<td>JNE</td>
<td>Jump if not equal (leftOp ≠ rightOp)</td>
</tr>
<tr>
<td>JCXZ</td>
<td>Jump if CX = 0</td>
</tr>
<tr>
<td>JECXZ</td>
<td>Jump if ECX = 0</td>
</tr>
</tbody>
</table>
## Jumps Based on Unsigned Comparisons

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JA</td>
<td>Jump if above (if $leftOp &gt; rightOp$)</td>
</tr>
<tr>
<td>JNBE</td>
<td>Jump if not below or equal (same as JA)</td>
</tr>
<tr>
<td>JAE</td>
<td>Jump if above or equal (if $leftOp &gt;= rightOp$)</td>
</tr>
<tr>
<td>JNB</td>
<td>Jump if not below (same as JAE)</td>
</tr>
<tr>
<td>JB</td>
<td>Jump if below (if $leftOp &lt; rightOp$)</td>
</tr>
<tr>
<td>JNBE</td>
<td>Jump if not above or equal (same as JB)</td>
</tr>
<tr>
<td>JBE</td>
<td>Jump if below or equal (if $leftOp &lt;= rightOp$)</td>
</tr>
<tr>
<td>JNA</td>
<td>Jump if not above (same as JBE)</td>
</tr>
</tbody>
</table>

## Jumps Based on Signed Comparisons

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JG</td>
<td>Jump if greater (if $leftOp &gt; rightOp$)</td>
</tr>
<tr>
<td>JNLE</td>
<td>Jump if not less than or equal (same as JG)</td>
</tr>
<tr>
<td>JGE</td>
<td>Jump if greater than or equal (if $leftOp &gt;= rightOp$)</td>
</tr>
<tr>
<td>JNL</td>
<td>Jump if not less (same as JGE)</td>
</tr>
<tr>
<td>JL</td>
<td>Jump if less (if $leftOp &lt; rightOp$)</td>
</tr>
<tr>
<td>JNGE</td>
<td>Jump if not greater than or equal (same as JL)</td>
</tr>
<tr>
<td>JLE</td>
<td>Jump if less than or equal (if $leftOp &lt;= rightOp$)</td>
</tr>
<tr>
<td>JNG</td>
<td>Jump if not greater (same as JLE)</td>
</tr>
</tbody>
</table>
More Frequently Used Jcond Instructions

- JE (Equal)
- JNE (Not Equal)
- JG or JGE (Greater Than or Equal)
- JL or JLE (Less Than or Equal)
- Note: JG=JNLE, JGE=JNL, …etc.

Simple IF

- If (op1=op2) then <…> end if
- Two different approaches:

  CMP op1, op2
  JE True
  JMP EndIf
  True: <…>
  EndIf

  CMP op1, op2
  JNE False
  <…>
  False:
IF ... AND ...

If (X > op1) and (Y <= op2) and ...
Then <...
End If

CMP X, op1
JNG EndIf
CMP Y, op2
JNLE EndIf
CMP ...
...
<...
EndIf:

IF ... OR ...

If (X > op1) or (Y <= op2) or ...
Then <...
End If

CMP X, op1
JG True
CMP Y, op2
JLE True
CMP ...
...
JMP EndIf
True:
<...
EndIf:
Exercise

• Can you rewrite the last slide and remove the “JMP EndIf”?
• How about adding the “else” part?

A Strategy for Compound Conditions

• Figure out the “longest path” of the program.
• Then find the “short cuts” (or early jumps).
WHILE

DO WHILE(op1<op2)
  <...
END DO

While:
  CMP op1, op2
  JNL EndDo
  <...
  JMP While
EndDo:

REPEAT UNTIL

REPEAT
  <...
UNTIL(X = op1) or (Y > op2)

repeat:
  <...
  CMP X, op1
  JE EndIf
  CMP Y, op2
  JNG repeat
EndIf:
More Exercises

- Write the code for the CASE statement.
- AND conditions in REPEAT-UNTIL?

```
REPEAT
  <...
UNTIL (X=op1) and
  (Y>op2)
```

repeat:
  <...
  CMP ?
  ?
  ?
EndIf:

Flags and Jcond

*How do Jcond instructions decide which way to go?*

- They check the flags!
- Examples:
  - JE/JNE checks Zero flag.
  - JG/JL checks Sign flag.
- CMP instruction sets the flags.
Boolean and Comparison Instructions

- AND Instruction
- OR Instruction
- XOR Instruction
- NOT Instruction
- TEST Instruction
- CMP Instruction

AND Instruction

- Performs a Boolean AND operation between each pair of matching bits in two operands
- Syntax:
  \[
  \text{AND} \quad \text{destination}, \text{source}
  \]
  (same operand types as MOV)

\[
\begin{array}{c|c|c}
  x & y & \text{x \cdot y} \\
  \hline
  0 & 0 & 0 \\
  0 & 1 & 0 \\
  1 & 0 & 0 \\
  1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
  \text{clear} \quad \text{AND} \quad \text{set} \\
  \hline
  0 & 0 & 1 \\
  0 & 0 & 0 \\
  1 & 0 & 1 \\
  1 & 1 & 1 \\
\end{array}
\]

unchanged
OR Instruction

- Performs a Boolean OR operation between each pair of matching bits in two operands
- Syntax:

```
OR destination, source
```

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x ^ y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

OR is true (set) if either, or both, of the bits are true:

```
0 0 1 1 1 0 1 1
0 0 0 0 1 1 1 1
0 0 1 1 1 1 1 1
```

XOR Instruction

- Performs a Boolean exclusive-OR operation between each pair of matching bits in two operands
- Syntax:

```
XOR destination, source
```

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x ^ y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

XOR is a useful way to toggle (invert) the bits in an operand:

```
0 0 1 1 0 1 1
0 0 0 0 1 1 1
0 0 1 1 0 1 0
```

XOR is true (set) if either, but not both, of the bits are true:

```
0 0 1 1 1 1 1 1
0 0 0 0 1 1 1 1
0 0 1 1 0 1 0 0
```

XOR is a useful way to toggle (invert) the bits in an operand.
NOT Instruction

- Performs a Boolean NOT operation on a single destination operand
- Syntax:

  \[
  \text{NOT } \text{destination}
  \]

<table>
<thead>
<tr>
<th>(x)</th>
<th>(\neg x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

Application - An Example

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

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

• Performs a nondestructive AND operation between each pair of matching bits in two operands
• No operands are modified, but the Zero flag is affected.
• Example: jump to a label if either bit 0 or bit 1 in AL is set.

```
test al,00000011b
jnz ValueFound
```