CS2422 Assembly Language & System Programming

October 19, 2006

Today’s Topics

• Sections 6.4 and 6.7: Conditional Processing
  – Conditional Loop
  – .IF, .WHILE, .REPEAT Directives
A Quick Review – CMP and Jcond Instruction

• The IF statement in C and PASCAL is converted into CMP and J\textit{cond} instructions in x86 Assembly:

\[
\begin{align*}
\text{If} \ (X > \text{op1})
\quad &\text{Then} \\
\quad &\langle\ldots\rangle \\
\quad &\text{End If}
\end{align*}
\]

\[
\begin{align*}
\text{CMP} \ X, \text{op1} \\
\text{JNG} \ \text{EndIf} \\
\langle\ldots\rangle \\
\text{EndIf:}
\end{align*}
\]

Rest of Chapter 6

• Conditional Loop Instructions.
• Finite State Machine (not covered today)
• .IF, .WHILE, and .REPEAT Directives
  (note: not Instructions)
Conditional Loop Instructions

• LOOPZ and LOOPE
• LOOPNZ and LOOPNE

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 ← 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.

LOOPNZ Example
The following code finds the first positive value in an array:

```
.data
array SWORD -3,-6,-1,-10,10,30,40,4
.code
  mov esi,OFFSET array
  mov ecx,LENGTHOF array
  sub esi,TYPE array
next:
  add esi, TYPE array
  test WORD PTR [esi],8000h ; test sign bit
  loopnz next ; continue loop
jnz quit ; none found
... ; ESI points to value
quit:
```
Using the .IF Directive

- Runtime Expressions
- Relational and Logical Operators
- MASM-Generated Code
- .REPEAT Directive
- .WHILE Directive

Runtime Expressions

- .IF, .ELSE, .ELSEIF, and .ENDIF can be used to evaluate runtime expressions and create block-structured IF statements.
- Examples:

  ```
  .IF eax>ebx
  mov edx,1
  .ELSE
  mov edx,2
  .ENDIF
  ```

  ```
  .IF eax>ebx && eax>ecx
  mov edx,1
  .ELSE
  mov edx,2
  .ENDIF
  ```

- MASM generates "hidden" code for you, consisting of code labels, CMP and conditional jump instructions.
Relational and Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>expr1 == expr2</code></td>
<td>Returns true when <code>expr1</code> is equal to <code>expr2</code>.</td>
</tr>
<tr>
<td><code>expr1 != expr2</code></td>
<td>Returns true when <code>expr1</code> is not equal to <code>expr2</code>.</td>
</tr>
<tr>
<td><code>expr1 &gt; expr2</code></td>
<td>Returns true when <code>expr1</code> is greater than <code>expr2</code>.</td>
</tr>
<tr>
<td><code>expr1 &gt;= expr2</code></td>
<td>Returns true when <code>expr1</code> is greater than or equal to <code>expr2</code>.</td>
</tr>
<tr>
<td><code>expr1 &lt; expr2</code></td>
<td>Returns true when <code>expr1</code> is less than <code>expr2</code>.</td>
</tr>
<tr>
<td><code>expr1 &lt;= expr2</code></td>
<td>Returns true when <code>expr1</code> is less than or equal to <code>expr2</code>.</td>
</tr>
<tr>
<td><code>!expr</code></td>
<td>Returns true when <code>expr</code> is false.</td>
</tr>
<tr>
<td><code>expr1 &amp; expr2</code></td>
<td>Performs logical AND between <code>expr1</code> and <code>expr2</code>.</td>
</tr>
<tr>
<td>`expr1</td>
<td>expr2`</td>
</tr>
<tr>
<td><code>expr1 ^ expr2</code></td>
<td>Performs bitwise XOR between <code>expr1</code> and <code>expr2</code>.</td>
</tr>
<tr>
<td><code>CARRY?</code></td>
<td>Returns true if the Carry flag is set.</td>
</tr>
<tr>
<td><code>OVERFLOW?</code></td>
<td>Returns true if the Overflow flag is set.</td>
</tr>
<tr>
<td><code>PARITY?</code></td>
<td>Returns true if the Parity flag is set.</td>
</tr>
<tr>
<td><code>SIGN?</code></td>
<td>Returns true if the Sign flag is set.</td>
</tr>
<tr>
<td><code>ZERO?</code></td>
<td>Returns true if the Zero flag is set.</td>
</tr>
</tbody>
</table>

Generated code:

```
.data
val1   DWORD 5
result DWORD ?
.code
mov eax,6
  .IF eax > val1
    mov result,1
  .ENDIF
```

MASM automatically generates an **unsigned** jump (JBE).
MASM-Generated Code

.data
val1    SDWORD  5
result  SDWORD  ?
.code
mov eax,6
.if eax > val1
mov result,1
.endif
@C0001:

Generated code:

mov eax,6
cmp eax,val1
ejle @C0001
mov result,1
@C0001:

MASM automatically generates a signed jump (JLE).

.REPEAT Directive

Executes the loop body before testing the loop condition associated with the .UNTIL directive.

Example:

; Display integers 1 – 10:
mov eax,0
.repeat
inc eax
call WriteDec
call Crlf
.until eax == 10
.WHILE Directive

Tests the loop condition before executing the loop body
The .ENDW directive marks the end of the loop.

Example:

```
; Display integers 1 - 10:
mov eax, 0
.WHILE eax < 10
    inc eax
    call WriteDec
    call Crlf
.ENDW
```

When to Use or Not to Use Directives?

- Directives make assembly language easier to write and to understand, by hiding tedious work. *(Food for thought: Wouldn’t it be even better to use C language?)*
- Don’t use directives if you want to have total control.