Decision Structures - Code Generation

Lecture 24
Intel Manual, Vol. 1, Sections 3.4.3, 8.1.4
Intel Manual, Vol. 2, Chapter 3, jcc

Robb T. Koether
Hampden-Sydney College
Mon, Apr 20, 2009
Outline

1. Conditional Branches
2. The CMPNE Tree
   - Integer Comparisons
   - Floating-Point Comparisons
3. Jump Trees
4. Assignment
Branching on a conditional expression will be done in two parts.

- Evaluate the expression, leaving either true (1) or false (0) on the stack.
- Test the value on the stack and branch on true.

It would be more efficient to test the original expression and branch immediately.

However, by separating the two tests, it will be simpler to write the code.
Conditional Branches

- Recall that the `if` statement is of the form
  \[
  \text{if } (\text{expr}) \text{ stmt}
  \]
  where \(\text{expr}\) is either zero (false) or nonzero (true).
- Thus, we must compare \(\text{expr}\) to 0.
- First, we generate code for a \texttt{CMPNE} tree that will leave 1 on the stack if \(\text{expr}\) is nonzero.
The CMPNE Tree

Integer Expression

Floating-point Expression
To compare two integers, we must
- Pop them off the stack into registers.
- Compare the registers.
- Push 0 or 1 onto the stack.

There is only one compare instruction: `cmp`.

It sets various flags, depending on the result of the comparison.
The relevant flags are

- CF - carry flag
- ZF - zero flag
- SF - sign flag
- OF - overflow flag
Conditional Jumps

- We then do a conditional jump.
- A conditional jump checks the flags to decide whether or not to jump.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>je</td>
<td>jump ==</td>
<td>ZF = 1</td>
</tr>
<tr>
<td>jne</td>
<td>jump !=</td>
<td>ZF = 0</td>
</tr>
<tr>
<td>jl</td>
<td>jump &lt;</td>
<td>SF ≠ OF</td>
</tr>
<tr>
<td>jge</td>
<td>jump &gt;=</td>
<td>SF = OF</td>
</tr>
<tr>
<td>jg</td>
<td>jump &gt;</td>
<td>ZF = 0 and SF = OF</td>
</tr>
<tr>
<td>jle</td>
<td>jump &lt;=</td>
<td>ZF = 1 or SF ≠ OF</td>
</tr>
</tbody>
</table>
Integer Comparison for !=

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov $1,%ecx</td>
<td>Move true to ecx</td>
</tr>
<tr>
<td>pop %eax</td>
<td>Pop right operand</td>
</tr>
<tr>
<td>pop %edx</td>
<td>Pop left operand</td>
</tr>
<tr>
<td>cmp %eax,%edx</td>
<td>Compare operands</td>
</tr>
<tr>
<td>jne Ln</td>
<td>Jump if left != right</td>
</tr>
<tr>
<td>dec %ecx</td>
<td>Change ecx to false</td>
</tr>
</tbody>
</table>

Ln:

push %ecx  # Push T/F result
Floating-Point Comparisons

- When we compare two floating-point numbers, they are already on the FPU stack, in $\text{st}(0)$ and $\text{st}(1)$.
- The instruction $\text{fucompp}$
  - Compares $\text{st}(0)$ to $\text{st}(1)$.
  - Sets condition codes $C0$ and $C3$ in the FPU status word.
  - Pops both operands off the stack.
The result of the comparison is determined by $C_0$ and $C_3$.

<table>
<thead>
<tr>
<th>Result</th>
<th>$C_0$</th>
<th>$C_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$st(0) &gt; st(1)$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$st(0) &lt; st(1)$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$st(0) = st(1)$</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Floating-Point Comparisons

- To perform a conditional jump based on a floating-point comparison, we must first move the FPU status word to the EFLAGS register.

- The instruction
  
  \[ \text{fnstsw} \ %ax \]

  stores the FPU status word in ax (16 bits).

- The instruction

  \[ \text{sahf} \]

  stores ah in the EFLAGS register.
Floating-Point Comparisons

- The `sahf` instruction maps
  - C0 → CF.
  - C3 → ZF.
- Thus, we want to use conditional jumps that check CF and ZF.
Conditional Jumps

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>je</td>
<td>jump ==</td>
<td>ZF = 1</td>
</tr>
<tr>
<td>jne</td>
<td>jump !=</td>
<td>ZF = 0</td>
</tr>
<tr>
<td>jb</td>
<td>jump &lt;</td>
<td>CF = 1</td>
</tr>
<tr>
<td>jae</td>
<td>jump &gt;=</td>
<td>CF = 0</td>
</tr>
<tr>
<td>ja</td>
<td>jump &gt;</td>
<td>ZF = 0 and CF = 0</td>
</tr>
<tr>
<td>jbe</td>
<td>jump &lt;=</td>
<td>ZF = 1 or CF = 1</td>
</tr>
</tbody>
</table>
Floating-Point Comparisons

Floating-Point Comparison for !=

mov $1, %ecx  # Move true to ecx
fucompp  # Pop left operand
fnstsw %ax  # Store status word
sahf  # Move ah to eflags
jne Ln  # Jump if lft != rgt
dec %ecx  # Change ecx to false

Ln:
push %ecx  # Push T/F result
One little complication is that `fucompp` compares `st(0) and st(1)` in the reverse order from what we would expect.

Therefore, the meanings of `<` and `>` are reversed and the meanings of `<=` and `>=` are reversed.

So we reverse the roles of `ja` and `jb`, and `jae` and `jbe`. 
Jump Trees

Jump Trees Syntax Tree

JUMP Syntax Tree

JUMPT Syntax Tree

JUMP

BLABEL 6

JUMPT

BLABEL 6

CMPNE
e.mode

NUM 0

e e.mode
A **JUMP** tree generates the following code.

**Code for JUMP**

```
jmp L8
```
A JUMPT tree must pop a boolean value off the stack and jump only if it is 1.

Code for JUMPT

<table>
<thead>
<tr>
<th>pop</th>
<th>%eax</th>
<th># Pop boolean value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmp</td>
<td>$1,%eax</td>
<td># Compare to true</td>
</tr>
<tr>
<td>je</td>
<td>L6</td>
<td># Jump if true</td>
</tr>
</tbody>
</table>
Assignment

Homework

- Read the Intel Manual, Vol. 1, Sections 3.4.3, 8.1.4.
- Read about the instructions `fucompp, fnstsw, sahf`.