If statements

Processing control structures

If statements

Generating code for ifs

Notes on code generation

Directly generating binary code

• The \(<op>\) in j<op> is dependent on the type of comparison you are doing in \(<bool_expr>\)
• When you generate JUMP instructions, you should also generate the appropriate LABELs
• But you may not put the LABEL into the code immediately
  • e.g., the OUT label (when should you create this? When should you put this in code?)
  • Instead, pass LABEL around to routine which does need to generate it
• Remember: labels have to be unique!

• Recall difference between assembly code and machine code
• Assembly code must be processed by assembler; machine code directly executable
• One job of assembler: decide actual addresses to jump to instead of labels
• So what happens if we generate binary directly!
• Need to insert JMP instructions before knowing where the label will be
• Solution: backpatching
  • Store offset of JMP instruction in semantic record
  • When label is created, access JMP instruction and “patch up” jump target
Processing Loops

Generating code for while loops

while <bool_expr> do
    <stmt_list>
end

LOOP:
    <bool_expr>
    j<!op> OUT
    <stmt_list>
OUT:

Generating code: for loops

for (<init_stmt>; <bool_expr>; <incr_stmt>)
    <stmt_list>
end

continue and break statements

for (<init_stmt>; <bool_expr>; <incr_stmt>)
    <stmt_list>
end

• Execute init_stmt first
• Jump out of loop if bool_expr is false
• Execute incr_stmt after block, jump back to top of loop
• Question: Why do we have the INCR label?

• Continue statements: skip past rest of block, perform incr_stmt and restart loop
• Break statements: jump out of loop (do not execute incr_stmt)
• Caveats:
  • Code for stmt_list is generated earlier—where do we jump?
  • Keep track of “loop depth” as you descend through AST
Switch/case statements

```plaintext
switch (<expr>)
  case <const_list>: <stmt_list>
  case <const_list>: <stmt_list>
  ...
  default: <stmt_list>
end
```

Deciding where to jump

- Problem: do not know which label to jump to until switch expression is evaluated
- Use a jump table: an array indexed by case values, contains address to jump to
  - If table is not full (i.e., some possible values are skipped), can point to a default clause
  - If default clause does not exist, this can point to error code
- Problems
  - If table is sparse, wastes a lot of space
  - If many choices, table will be very large

Jump table example

Consider the code:
```
<table>
<thead>
<tr>
<th>Case x is</th>
<th>Jump to</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0010)</td>
<td>JUMP 0010</td>
</tr>
<tr>
<td>(0017)</td>
<td>JUMP 0017</td>
</tr>
<tr>
<td>(0192)</td>
<td>JUMP 0192</td>
</tr>
<tr>
<td>(0198)</td>
<td>JUMP 0198</td>
</tr>
<tr>
<td>(1000)</td>
<td>JUMP 1000</td>
</tr>
<tr>
<td>(1050)</td>
<td>JUMP 1000</td>
</tr>
</tbody>
</table>
```

- Jump table has 6 entries:
- Table only has one Unnecessary row (for choice 4)

Do a binary search

Consider the code:
```
<table>
<thead>
<tr>
<th>Case x is</th>
<th>Jump to</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0010)</td>
<td>JUMP 0010</td>
</tr>
<tr>
<td>(0017)</td>
<td>JUMP 0017</td>
</tr>
<tr>
<td>(0192)</td>
<td>JUMP 0192</td>
</tr>
<tr>
<td>(0198)</td>
<td>JUMP 0198</td>
</tr>
<tr>
<td>(1000)</td>
<td>JUMP 1000</td>
</tr>
<tr>
<td>(1050)</td>
<td>JUMP 1000</td>
</tr>
</tbody>
</table>
```

- Jump table has 6 entries:
- Perform a binary search on the table. If the entry is found, then jump to that offset. If the entry isn’t found, jump to others clause. O(log n) time, n is the size of the table, for each jump.
Linear search example

Consider the code:

(xxxx) Is offset of local
Code start from the
Jump instruction

Case x is
(0010) When 0: stmts
(0017) When 1: stmts
(0192) When 2: stmts
(1050) When others stmts;

If there are a small number of choices, then do an in-line linear search. A straightforward way to do this is generate code analogous to an IF THEN ELSE.

If (x == 0) then stmts1;
Elseif (x == 1) then stmts2;
Elseif (x == 2) then stmts3;
Else stmts4;

O(n) time, n is the size of the table, for each jump.

Dealing with jump tables

switch (<expr>)
  case <const_list>: <stmt_list>
  case <const_list>: <stmt_list>
  ...
  default: <stmt_list>
end

  <expr>
  <code for jump table>
  LABEL0:
  <stmt_list>
  LABEL1:
  <stmt_list>
  ...
  DEFAULT:
  <stmt_list>
  OUT:

  - Generate labels, code, then build jump table
  - Put jump table after generated code
  - Why do we need the OUT label?
  - In case of break statements

Wednesday, February 23, 2011