Phases of a compiler

Intermediate Representation (IR): specification and generation

Figure 1.6, page 5 of text
backpatching if
6.7.3 Backpatching Flow-of-Control statements

<table>
<thead>
<tr>
<th>S → if (B) M1 S1</th>
<th>N else M2 S2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>backpatch(B.truelist, M1.instr)</td>
</tr>
<tr>
<td></td>
<td>backpatch(B.falselist, M2.instr)</td>
</tr>
<tr>
<td></td>
<td>temp = merge(S1.nextlist, N.nextlist)</td>
</tr>
<tr>
<td></td>
<td>S.nextlist = merge(temp, S2.nextlist)</td>
</tr>
</tbody>
</table>

M → ε

M.instr = nextinstr

N → ε

N.nextlist = makelist(nextinstr)

gen('goto _')

Diagram:
- **true**
  - **B**
  - **M1.instr** → **S1** → **N**
- **false**
  - **M2.instr** → **S2**
Example 6.24 - extended
if (x < 100 || x > 200 && x != y) S1 else S2

Let's extend the Boolean expression example from part 1 by embedding that expression into an if-then-else statement (using the textbook syntax, not alpha syntax).
Example 6.24 - extended
if \((x < 100 \; \text{||} \; x > 200 \; \&\& \; x \neq y)\) S1 else S2

100: if \(x < 100\) goto ___
101: goto 102
102: if \(x > 200\) goto 104
103: goto ___
104: if \(x \neq y\) goto ___
105: goto ___

truelist = \{100,104\}
falselist = \{103,105\}

Let's remember where we left off...
Example 6.24 - extended

if (x < 100 || x > 200 && x != y) S1 else S2

100: if x < 100 goto ___
101: goto 102
102: if x > 200 goto 104
103: goto ___
104: if x != y goto ___
105: goto ___
106: instruction for S1
107: instruction for S1
108: instruction for S1
109: instruction for S1
110: instruction for S1
111: goto ___
112: instruction for S2
113: instruction for S2
114: instruction for S2

In the example above
we have not spelled out what S1 and S2 are.

Let's assume S1 requires 5 instructions
and S2 requires 3 instructions.

truelist = \{100,104\}
falselist = \{103,105\}
Example 6.24 - extended
if (x < 100 || x > 200 && x != y) S1 else S2

100: if x < 100 goto 106
101: goto 102
102: if x > 200 goto 104
103: goto 112
104: if x != y goto 106
105: goto 112
106: instruction for S1
107: instruction for S1
108: instruction for S1
109: instruction for S1
110: instruction for S1
111: goto ___
112: instruction for S2
113: instruction for S2
114: instruction for S2
115: instruction for S2

truelist = {100,104}
falselist = {103,105}
nextlist = {111}

Embedded in the context of this if-then-else statement we can backpatch truelist and falselist from the Boolean expression, and we introduce nextlist.
backpatching while
6.7.3 Backpatching Flow-of-Control statements

The end-of-rule actions for a while statement are shown on the next slide.

Exercise:
Extend example 6.24 as a while statement where the body of the while requires 5 instructions.

while $(x < 100 || x > 200 && x != y)$ S1

Show how backpatching works in the instruction array.
# 6.7.3 Backpatching Flow-of-Control statements

| S -> while M1 | backpatch(S1.nextlist, M1.instr) 
| (B) M2 S1 | backpatch(B.truelist, M2.instr) 
| | S.nextlist = B.falselist 
| | gen('goto' M1.instr) 
| M -> ε | M.instr = nextinstr 

**Diagram:**
- **M1.instr**
- **B**
- **M2.instr**
- **S1**
- **true**
- **false**
- **S.next**