CS480

Ch 8

(8.1 - 8.7 pp. 463-508)

Intermediate Code Generation

April 1, 2009

This will be on Monday's exam
Intermediate Code

- Ties front end to back end
- Can be machine/language independent

Parser $\rightarrow$ Static Checker $\rightarrow$ Intermediate Code Generator $\rightarrow$ (Optimizer) $\rightarrow$ Code Generator

- We don't really have distinct Intermediate and Machine code
Three Address Code

\[ x = y \ op \ z \]

- At most three addresses used
  - \( x = y \) is valid 3 Address Code
  - \( x = -y \) is valid 3 Address Code

- May need to break down expressions
  - \( x = y + z \times q; \)

- Our “Quads” are three address code
Types of Statements

- Assignment Statement
  \[ x = y \; \text{binop} \; z; \; x = \text{unop} \; y \]

- Unconditional Jumps: goto Label
- Conditional Jumps: if \((x \; \text{relop} \; y)\) goto Label
- Function Call: Foo \((a,b,c)\)
- Function Return: return \(y\)
- Indexed Assignments: \(x = a[i] \; ; \; a[i] = x\)
- Address and Pointer Assignments: \(x = \&a \; ; \; b = *a; \; *a = b\)
Syntax Directed Translation into 3 Address code

- Semantic Rules!
- Attributes? How? Where? When?

\[ S \rightarrow \text{id} = E \ | | \]

\[ E \rightarrow E_1 + E_2 | | \]

- What happens if \( E + E \) is deep in the parse tree?
- What if \( E_1 \) is an array? Pointer? The expression \((a + b)\) ?
void intExecute(int wOpcode, /*opcode of current quad to be executed */
            int wOperand1, /*operand1 value if necessary for opcode*/
            int wOperand2, /*operand2 value if necessary for opcode*/
            int wOperand3) /*operand3 value if necessary for opcode*/
{
    
    switch (wOpcode)
    {
    case OP_ADD:           gStack[wOperand3] = wOperand1 + wOperand2;
                          break;

    case OP_DEREFERENCE:     gStack[wOperand3] = gStack[wOperand1];
                        break;

    ...
Addressing Modes

```c
int intDecode(int wMode, int wAddress) {
    switch(wMode) {
    case IMMEDIATE:       return(wAddress); // 0
    case GLOBAL_LVALUE:   return(wAddress); // 1
    case GLOBAL_RVALUE:   return(gStack[wAddress]); // 2
    case LOCAL_LVALUE:    return(gAP + wAddress); // 3
    case LOCAL_RVALUE:    return(gStack[gAP + wAddress]); // 4
    }
}
```
Example

```c
int Foo(b)
{
  output(b);
}

int main ()
{
  int a;
  a = 100;
  a = a + 1;
  Foo (a);
}
```

```
<table>
<thead>
<tr>
<th>addr</th>
<th>Lvl</th>
<th>Insn</th>
<th>AM</th>
<th>Op1</th>
<th>AM</th>
<th>Op2</th>
<th>AM</th>
<th>Op3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>27</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>20</td>
<td>4</td>
<td>-3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>25</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>26</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>23</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>22</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>26</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>26</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>20</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>21</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>26</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>26</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>23</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>28</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>RTS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Various Statements

• Assignments

```c
int *a, x[10] p, b, c;
a = &b;
*p = a;
x[b] = c;
c = b++;
```
Short-Circuit Code

```c
if( x == 0 && ( y = x + 1) )
{
    //do something
}
```
Control Flow

• If

• If/else

• While

• How is for different from while?
Function Calls

int Foo(a, b)
int a; int *b;
{
  return a * *b;
}

...

Foo(1, &x);

...