Administration

- Read: Appel 6-8
- Prelim March 6, 7:30-9:30PM

Where we are

IR expressions

- \texttt{CONST}(i) \ : \ the \ integer \ constant \ i
- \texttt{TEMP}(t) \ : \ a \ temporary \ register \ t. \ The \ abstract \ machine
  has \ an \ infinite \ number \ of \ these
- \texttt{OP}(e_1, e_2) \ : \ one \ of \ the \ following \ operations
  \begin{itemize}
  \item arithmetic: \texttt{ADD}, \texttt{SUB}, \texttt{MUL}, \texttt{DIV}, \texttt{MOD}
  \item \texttt{bit \ logic}: \texttt{AND}, \texttt{OR}, \texttt{XOR}, \texttt{LSHIFT}, \texttt{RSHIFT}, \texttt{ARSHIFT}
  \item \texttt{comparisons}: \texttt{EQ}, \texttt{NEQ}, \texttt{LT}, \texttt{GT}, \texttt{LEQ}, \texttt{GEQ}
  \end{itemize}
- \texttt{MEM}(e) \ : \ contents \ of \ memory \ locn \ w/ \ address \ e
- \texttt{CALL}(ef, e_0, e_1, \ldots) \ : \ result \ of \ fcn \ ef \ applied \ to \ arguments \ e_i
- \texttt{NAME}(n) \ : \ address \ of \ the \ statement \ or \ global \ data
  location \ labeled \ n \ (TBD)
- \texttt{ESEQ}(s, e) \ : \ result \ of \ e \ after \ stmt \ s \ is \ executed

IR statements

- \texttt{MOVE}(dest, e) \ : \ move \ result \ of \ e \ into \ dest
  \begin{itemize}
  \item dest = \texttt{TEMP}(t) \ : \ assign \ to \ temporary \ t
  \item dest = \texttt{MEM}(e) \ : \ assign \ to \ memory \ locn \ e
  \end{itemize}
- \texttt{EXP}(e) \ : \ evaluate \ e, \ discard \ result
- \texttt{SEQ}(s_1, \ldots, s_n) \ : \ execute \ each \ stmt \ s_i \ in \ order
- \texttt{JUMP}(e) \ : \ jump \ to \ address \ e
- \texttt{CJUMP}(e, l_1, l_2) \ : \ jump \ to \ statement \ named \ l_1 \ or \ l_2
  depending \ on \ whether \ e \ is \ true \ or \ false
- \texttt{LABEL}(n) \ : \ a \ labeled \ statement \ (may \ be \ used \ in \ NAME, CJUMP)

Translation

- Intermediate \ code \ generation \ is \ tree \ translation

Abstract \ syntax \ tree \ \Rightarrow \ IR \ tree

- Each \ subtree \ of \ AST \ translated \ to \ subtree \ in \ IR \ tree
- Translated \ version \ of \ AST \ subtree \ e \ is \ IR \ subtree \ \langle e \rangle
Translating if

\[
\text{if } (e) \text{ s} = \text{SEQ}(\text{CJUMP}(e, t, f), \text{LABEL}(t), \text{LABEL}(f))
\]

How to read IR trees

- Think of SEQ nodes as blocks of stmts

\[
\text{if } (e) \text{ s} = \text{CJUMP}(e, t, f)
\]

Translating if-else

\[
\text{if } (e) s_1 \text{ else } s_2 = \text{SEQ}(\text{CJUMP}(e, t, f), \text{LABEL}(t), \text{LABEL}(f))
\]

Translating while

\[
\text{while } (e) \text{ s} = \text{SEQ}(\text{LABEL}(\text{loop}), \text{CJUMP}(e, t, f), \text{LABEL}(t), \text{LABEL}(f), \text{JUMP}(\text{NAME}(\text{loop})))
\]

Spec → Implementation

abstract class Node { abstract IRnode translate(); …}

\[
\text{if } (e) s = \text{SEQ}(\text{CJUMP}(e, t, f), \text{LABEL}(t), \text{LABEL}(f))
\]

class IfNode { …
IRnode translate() {
    SeqNode ret = new SEQ();
    ret.append(new CJUMP(e.translate(), "t", "f"));
    ret.append(new LABEL("t"));
    ret.append(s.translate());
    ret.append(new LABEL("f"));
    return ret;
}

Syntax-directed translation

- Translation of any expression or statement expressed in terms of translations of subexpressions
- Can write down translations formally
  - precise specification of what compiler does
  - converts directly to an implementation
  - allows proof that compiler works correctly
Lecture 14 CS 412/413 Spring '01 -- Andrew Myers

Problem: multiple translations

\[ v = e \]

As expression:
\[
\begin{array}{c}
\text{MOVE} \\
\text{TEMP}(t_e) \\
\text{SEQ} \\
\text{TEMP}(t_e) \\
\text{ESEQ} \\
\text{MOVE} \\
\end{array}
\]

As statement:
\[
\begin{array}{c}
\text{MOVE} \\
\text{TEMP}(t_e) \\
\end{array}
\]

Translation functions

- \( \varepsilon[e] \) is IR expr node that computes the same value as expression \( e \) (Appel: Ex)
- \( \varepsilon[s] \) is IR expr node that computes the same value as statement \( s \) (Appel: Ex)
- \( \hat{s}[s] \) is IR stmt node with side-effects of \( s \) (Appel: Nx)

- For boolean expr \( e \), \( \varepsilon[e, l_1, l_2] \) is IR statement node that jumps to label \( l_1 \) if \( e \) evaluates to true and to \( l_2 \) if \( e \) evaluates to false (Appel: Cx)

Implementing translations

abstract class Node {
  ...
  abstract IRnode translateE();
  abstract IRnode translateS();
  abstract IRnode translateC(); ...
}

class Assignment {
  Expr variable, value;
  IRnode translateS() {
    return new MOVE(translateE(variable), translateE(value));
  }
  IRnode translateE() {
    TEMP t = freshTemp();
    return new ESEQ(new SEQ(new MOVE(t, value.translateE()), new MOVE(…)), t);
  }
}

Some examples so far

- \( \varepsilon[v] = \text{TEMP}(v) \) (for local variable only!)
- \( \varepsilon[e_1 + e_2] = \text{ADD}(\varepsilon[e_1], \varepsilon[e_2]) \)
- \( \varepsilon[v = e] = \text{MOVE}(\varepsilon[v], \varepsilon[e]) \)
- \( \varepsilon[v = e] = \text{ESEQ}(\text{SEQ}(\text{MOVE}(\text{TEMP}(t), \varepsilon[e]), \text{MOVE}(\varepsilon[v], \text{TEMP}(t))), \text{TEMP}(t)) \)
- \( \varepsilon[\text{if (e) s}] = \text{SEQ}(\text{CJUMP}(\varepsilon[e], t, f), \text{LABEL}(t), \text{CJUMP}(\varepsilon[e], t, f), \text{LABEL}(f)) \)
- \( \varepsilon[\text{if (e) s}] = \text{ESEQ}(\text{SEQ}(\text{…}), \text{TEMP}(t)) \)

Translating a function

- Function body is expression \( e \)
- Translate as statement \( \varepsilon[e_1 + e_2] \) ?
- How to translate return statement?
- Idea: introduce return value register \( \text{TEMP}(RV) \), function epilogue label
- Function body \( e \) translated as
  \( \text{SEQ}(\text{MOVE}(\text{TEMP}(RV), \varepsilon[e]), \text{LABEL}(\text{epilogue})) \)
- return \( e \) translated as \( \hat{s}[\text{return } e] = \text{SEQ}(\text{MOVE}(\text{TEMP}(RV), \varepsilon[e]), \text{JUMP}(\text{epilogue})) \)

The boolean operator problem

- How to translate expression \( e_1 \land e_2 \) ?
- How about \( \varepsilon[e_1 \land e_2] = \text{AND}(\varepsilon[e_1], \varepsilon[e_2]) \)?
- How about \( \varepsilon[e_1 \land e_2] = \text{ESEQ}(\text{SEQ}(\text{MOVE}(\text{TEMP}(x), 0), \text{CJUMP}(\varepsilon[e_1], t_1, no_set), \text{LABEL}(t_1), \text{CJUMP}(\varepsilon[e_2], t_2, no_set), \text{LABEL}(t_2), \text{MOVE}(\text{TEMP}(x), 1), \text{LABEL}(no_set), \text{TEMP}(x))) \)?
Current translation

- Bad IR: $s_{\text{if } (e_1 \& e_2) s} = \text{SEQ(CJUMP(\text{SEQ(MOVE(TMP(x), 0)), C\text{JUMP}(\text{SEQ(e_1, t, f)}, t, f), LABE}\text{L(t)}, \text{CJUMP(e_2, t, f), f}}), \text{LABEL(f))})$

- Better IR: $s = \text{SEQ(CJUMP(e_1), t, f), LABEL(t), CJUMP(e_2, t, f), LABEL(f))}$

Booleans via control

- Idea: representing boolean values via control flow rather than explicitly
- For boolean expr $e$, $c_{true, l_1, l_2}$ is IR statement node that jumps to label $l_1$ if $e$ evaluates to true and to $l_2$ if $e$ evaluates to false
  
  $c_{true, l_1, l_2} = \text{JUMP(NAME(l_1))}$
  $c_{false, l_1, l_2} = \text{JUMP(NAME(l_2))}$
  $c_{e_1 \Rightarrow e_2, l_1, l_2} = \text{CJUMP(EQ(\text{SEQ(e_1, l, f)}, l_1), l_2)}$

Efficient translations of if and &

"$c_{e, l_1, l_2}$ is IR statement node that jumps to label $l_1$ if $e$ evaluates to true and to $l_2$ if $e$ evaluates to false"

Can use to improve if translation:

$s_{\text{if } (e \& e_2) s} = \text{SEQ(c_{e_1}, t, f), LABEL(t), c_{e_2, t, f}, LABEL(f))}$

Now: $s_{\text{if } (e \& e_2) s} = \text{SEQ(c_{e_1, t_1, f}, c_{e_2, t_2, f}, LABEL(t_1), c_{e_2, t_2, f}, LABEL(f))}$

Progress

- Now have rules for transforming AST into intermediate representation
- Can apply this to AST of each function defn to get IR for function
- Intermediate representation has many features not found in real assembly code
  - arbitrarily deep expression trees vs. <5 deep
  - ability to perform statements with side-effects as part of an expression (ESEQ, CALL); undefined behavior
  - CJUMP is two-way jump rather than fall-through
- Next time: flattening IR (canonical form)