Project 10: IR Code Generation, Part 3

Finish IR Code Generation
Optional Extension: Peephole Optimizer

Files:
- Generator.java
- Peephole.java -- “dummy stub”
- tst/ -- Contains all of the p9 tests plus more
- Main.java
- Main.jar
- makefile
- runAll
- IR.java
- <others> -- unchanged

An Optimization in “genAssignStmt”

Want to reduce temporaries
Watch for special case:
  “Lefthand side is a simple variable”
  ...and avoid calling genLValue()

Example:
- PCAT Source:
  \[
  x := a + b;
  \]

Before (in project 9):
  \[
  \begin{align*}
  t5 & := & \&x & \quad \text{genLValue} & \quad \downarrow t5 \\
  t6 & := & a + b & \quad \text{genExpr} & \quad \downarrow t6 \\
  *t5 & := & t6 & \quad \text{genAssignStmt}
  \end{align*}
  \]

With Optimization:
  \[
  \begin{align*}
  t6 & := & a + b & \quad \text{genExpr} & \quad \downarrow t6 \\
  x & := & t6 & \quad \text{genAssignStmt}
  \end{align*}
  \]

If the lefthand side is not a simple variable...
Call genLValue() and generate “store” instruction.
### Optimization #2

Goal: Reduce temporaries associated with constants!

**PCAT Source:**

\[(a + b) \ast 17\]

**Before (in project 9):**

\[
\begin{align*}
  t4 & := a + b \\
  t5 & := 17 \\
  t6 & := t4 \ast t5
\end{align*}
\]

**With Optimization:**

\[
\begin{align*}
  t4 & := a + b \\
  t5 & := t4 \ast 17
\end{align*}
\]

---

**E.Code**

**E.Place**

**Before:**

`genExpr()`

- creates a temporary
- generates code to move the value into the temporary
- returns the temporary

**With Optimization:**

`genExpr()`

When called on a constant...

- will return the value directly
Runtime Errors

runtimeError1:
  Heap allocation failed.

runtimeError2:
  Pointer is NIL. (during dereferencing)

runtimeError3:
  Read statement failed.

runtimeError4:
  Array index is out of bounds.
  \[ 0 \leq \text{index} < N \]

runtimeError5:
  In an array constructor, the count is \( \leq 0 \).
  \[
  \begin{align*}
  a & := \text{ATYPE} \{ \{1, 2, 3, 4\}\}; \\
  a & := \text{ATYPE} \{ \{100 \text{ of } 0, 200 \text{ of } -1\}\}; \\
  a & := \text{ATYPE} \{ \{i*10 \text{ of } -1, 3, x+y, k \text{ of } 0\}\};
  \end{align*}
  \]

Boilerplate

Canned, fixed material inserted into the SPARC output target file.

Layout of Arrays and Records

Every data value is 1 word long (4 bytes)
Record Sizes and Field Offsets

Each field is 4 bytes.
Compute and set: \texttt{RecordType.size} and \texttt{FieldDecl.offset}
Must take a look at \texttt{TypeDecl.compoundType}.

```
fieldDecls
  size 12

FieldDecl id “f1”
  type “f1”
  next 0
  offset 0

FieldDecl id “f2”
  type “f2”
  next 4
  offset 4

FieldDecl id “f3”
  type “f3”
  next NULL
  offset 8
```

Dealing With L-Values

```
x Variable
a[...expr...] ArrayDeref
r.fieldName RecordDeref
```
Can be used as
\begin{itemize}
  \item L-Value \texttt{genLValue()}
  \item R-Value \texttt{genValueOf()}
\end{itemize}
Dealing With L-Values

\[
\begin{align*}
\text{Variable} & : \text{Can be used as L-Value} \\
\text{ArrayDeref} & : \text{L-Value} \\
\text{RecordDeref} & : \text{R-Value}
\end{align*}
\]

How we deal with...
Variable
ArrayDeref
RecordDeref

Will differ depending on whether it is used as...

L-Value
Generate code to move an \textit{address} into a temp.

R-Value
Generate code to move a \textit{value} into a temp.

Idea:
Eliminate \texttt{genVariable}, \texttt{genArrayDeref}, \texttt{genRecordDeref}.
Include code directly in \texttt{genLValue} and \texttt{genValueOf}
... since it will be slightly different in each.

genValueOf (ValueOf p,...) {
    lv = p.lValue;
    if lv instanceof Variable {
        ...
    } else if lv instanceof RecordDeref {
        ...
    } else if lv instanceof ArrayDeref {
        ...
    }
}

genLValue (LValue p) {
    if p instanceof Variable {
        ...
    } else if p instanceof RecordDeref {
        ...
    } else if p instanceof ArrayDeref {
        ...
    }
}
Record Dereferencing (as an L-Value)

Source:

<table>
<thead>
<tr>
<th>Expr</th>
<th>Assign Stmt</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxxx . f3 := b + c</td>
<td></td>
</tr>
</tbody>
</table>

Record Deref

Could be any L-Value

IR Code:

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>0</td>
</tr>
<tr>
<td>f2</td>
<td>4</td>
</tr>
<tr>
<td>f3</td>
<td>8</td>
</tr>
</tbody>
</table>

* t9 := t10

gExpr genExpr

t9 genIdent
t10 genIdent

gLValue genLValue
gAssignStmt genAssignStmt
Record Dereferencing (as an L-Value)

**Source:**

```
xxxx. f3 := b + c;
```

**IR Code:**

```
t8 := *t7
if t8 = 0 then goto runtimeError2 (int)
t9 := t8 + 8
*t9 := t10
```

Example

**Source:**

```
y. f3 := b + c;
```

**IR Code:**

```
t7 := &y
t8 := *t7
if t8 = 0 then goto runtimeError2 (int)
t9 := t8 + 8
t10 := b + c
*t9 := t10
```
Example

Source:

\[ y . f3 := b + c ; \]

IR Code:

\[
\begin{align*}
t7 & := \&y \\
t8 & := *t7 \\
if \ t8 = 0 \ then \ goto \ runtimeError2 \ (int) \\
t9 & := t8 + 8 \\
t10 & := b + c \\
*t9 & := t10
\end{align*}
\]

Example

Source:

\[ y . f3 := b + c ; \]

IR Code:

\[
\begin{align*}
t7 & := \&y \\
t8 & := *t7 \\
if \ t8 = 0 \ then \ goto \ runtimeError2 \ (int) \\
t9 & := t8 + 8 \\
t10 & := b + c \\
*t9 & := t10
\end{align*}
\]

Optimization: Watch for special case

RecordDeref.lValue is a simple variable!

\[
\begin{align*}
if \ y = 0 \ then \ goto \ runtimeError2 \ (int) \\
t9 & := y + 8 \\
t10 & := b + c \\
*t9 & := t10
\end{align*}
\]
genLValue

genLValue (LValue p) {
    if p instanceof Variable {
        generate "t3 := &x"
        return t3
    } else if p instanceof RecordDeref {
        ...
        ...
    } else if p instanceof ArrayDeref {
        ...
        ...
    }
}

call genLValue
generate LoadIndirect instruction
Do the same optimization for ArrayDeref

© Harry H. Porter, 2006
Record Deref Optimization #2

Source:

```plaintext
xxxx . f1 := b + c ;
```

IR Code:

```plaintext
xxxx
  t8 := *t7
  if t8 = 0 then goto runtimeError2 (int)
  t9 := t8 + 0
  t10 := b + c
  *t9 := t10

Ughhh!!!
```

A field with offset = 0

Optimization:

Eliminate the ADD instruction
Avoid creating the temporary (t9)
Just return this variable
Example

Source:
\[
y.f1 := a + b;
\]

IR Code:
\[
\text{if } y = 0 \text{ then goto runtimeError2}
\text{t10 := a + b}
\text{*y := t10}
\]

Both optimizations apply!

Array Dereferencing (as an L-Value)

Source:
\[
xxxx \ [ \ i*3 \ ] := b + c ;
\]
Array Dereferencing (as an L-Value)

Source:

```
xxxx[i*3] := b + c;
```

The "index" expr

Array Deref

Could be any L-Value

IR Code:

```
genLValue(xxxx) genLValue

t22
genExpr(i*3) t24
genAssignStmt

t26
*t25 := t26
```

```
a[i*3]
r.f[i*3]
a[j*7][i*3]
```
Array Dereferencing (as an L-Value)

Source:

```plaintext
xxxx [ i*3 ] := b + c ;
```

IR Code:

```plaintext
xxxx
t23 = *t22
if t23 = 0 then goto runtimeError2 (int)
t24 := i * 3
t24 := i * 3
t24 := i * 3
t24 := i * 3
t25 := *t23
if t24 >= t25 then goto runtimeError4 (int)
t25 := t24 * 4
t25 := t25 + 4
t25 := t23 + t25
t26 := b + c
*t25 := t26
```

Dealing With R-Values: ArrayDeref and RecordDeref

Source:

```plaintext
z := a[i*3] + 7 ;
```
Dealing With R-Values:
ArrayDeref and RecordDeref

Source:
\[ z := a[i*3] + 7; \]

IR Code:
\[
\begin{align*}
& \text{genLValue} \quad \text{Expr} \quad \text{LValue} \quad \text{ValueOf} \\
& t26 := *t25 \\
& t27 := t26 + 7 \\
& *t24 := t27
\end{align*}
\]

Code from genLValue:
... gets address into some variable. Just add a “LoadIndirect”

Activation Record Layout

procedure foo (x_1, x_2, \ldots, x_M)
\begin{align*}
& \quad \text{var } y_1, y_2, \ldots, y_N; \\
& \quad \text{begin} \\
& \quad \quad \ldots \text{bar}(z_1, z_2, \ldots, z_P) \ldots \\
& \quad \end{align*}
\]

Frame of “foo”

Frame of “foo”’s caller

© Harry H. Porter, 2006
New Fields:
- VarDecl.offset
- Formal.offset
- Body.frameSize

Method “printOffsets()” has been added to IR.java

Computation of frameSize...
- numberOfLocals
- maxNumberOfArgsUsed

Constants in Generator.java

<table>
<thead>
<tr>
<th>static</th>
<th>final</th>
<th>int</th>
<th>INITIAL_VARIABLE_OFFSET</th>
<th>= -4;</th>
</tr>
</thead>
<tbody>
<tr>
<td>static</td>
<td>final</td>
<td>int</td>
<td>VARIABLE_OFFSET_INCR</td>
<td>= -4;</td>
</tr>
<tr>
<td>static</td>
<td>final</td>
<td>int</td>
<td>INITIAL_FORMAL_OFFSET</td>
<td>= +68;</td>
</tr>
<tr>
<td>static</td>
<td>final</td>
<td>int</td>
<td>FORMAL_OFFSET_INCR</td>
<td>= +4;</td>
</tr>
<tr>
<td>static</td>
<td>final</td>
<td>int</td>
<td>REGISTER_SAVE_AREA_SIZE</td>
<td>= +64;</td>
</tr>
<tr>
<td>static</td>
<td>final</td>
<td>int</td>
<td>DISPLAY_REG_SAVE_AREA_OFFSET</td>
<td>= +64;</td>
</tr>
</tbody>
</table>

Don’t Forget:
If frame size is not a multiple of 8....
then add 4 (the optional, unused alignment word)
to make it a multiple of 8.

New IR Instruction: alloc

t3 := alloc (n)

Allocate “n” bytes on the heap
Set “result” to a pointer to the memory
Save addr in t3
...or set to zero if problems.
Will call “calloc” from library.
Used for
- Array Constructors
- Record Constructors
Source:

\[ r = \text{RecType} \{ f3 := XXXX ; f1 := YYYY ; f2 := ZZZZ \} ; \]

FieldInits can be in any order
Record Construction

Source:
\[ r = \text{RecType} \{ f_3 := XXXX ; f_1 := YYYY ; f_2 := ZZZZ \} ; \]

FieldInits can be in any order

IR Code:
\[
XXX
XXX
\]
\[
YYYY
YYYY
\]
\[
ZZZZ
ZZZZ
\]
\[ r := t_1 \]

One per FieldInit, in the order of the RecordConstructor (not the order of the RecordType).
Record Construction

Source:
\[ r = \text{RecType} \{ \text{f3:= XXXX ; f1:= YYY ; f2:= ZZZZ} \}; \]

IR Code:
\[
\begin{align*}
t1 &:= \text{alloc}(12) \\
\text{if } t1 = 0 \text{ then goto runtimeError1 (int)} \\
t2 &:= t1 + 8 \\
*\text{t2} &:= \text{tx} \\
t2 &:= t1 + 0 \\
*\text{t2} &:= \text{ty} \\
t2 &:= t1 + 4 \\
*\text{t2} &:= \text{tz} \\
r &:= t1
\end{align*}
\]

FieldInits can be in any order.

Array Constructors

type A is array of \ldots;

\[ A \{ \{ \text{count}_1 \text{ of } \text{expr}_1, \ldots, \text{count}_K \text{ of } \text{expr}_K \} \} \]

optional (default = 1)

Expressions:
\[
\begin{align*}
a &:= \text{MyArr} \{\{ 1000 \text{ of } -1 \}\}; \\
b &:= \text{MyArr} \{\{ 101, 102, 103, 104 \}\}; \\
c &:= \text{MyArr} \{\{ i*4 \text{ of } \text{foo}(i), k \text{ of } \text{bar}(x) \}\};
\end{align*}
\]
Repeated K times

Initialize $count_i$ elements to $value_i$

Repeated K times

$t1 := 0$
$tCnt_i := \ldots \text{genExpr}(\text{countExpr}_i)\ldots$
if $tCnt_i \leq 0$ goto runtimeError5
$t1 := t1 + tCnt_i$
$tVal_i := \ldots \text{genExpr}(\text{valueExpr}_i)\ldots$
$t2 := t1*4$
$t2 := t2 + 4$
$t2 := \text{alloc}(t2)$
if $t2 = 0$ goto runtimeError1
$\text{Label}_i$: $t3 := t2$
$t4 := tCnt_i$
$t2 := t2 + 4$
$t2 := tVal_i$
$t4 := t4 - 1$
if $t4 > 0$ goto $\text{Label}_i$
t1 := 0
\[ t\text{Cnt}_i := \ldots \text{genExpr(countExpr}_i)\ldots \] 
if t\text{Cnt}_i <= 0 goto runtimeError5 
t1 := t1 + t\text{Cnt}_i 
tVal_i := \ldots \text{genExpr(valueExpr}_i)\ldots 

\[\ldots\]

At this point, t1 = N
\[ t2 := t1*4 \]
\[ t2 := t2 + 4 \]
\[ t2 := \text{alloc}(t2) \]
if t2 = 0 goto runtimeError1 
t3 := t2 
\[ *t2 := t1 \]
\[ t4 := t\text{Cnt}_i \]
Label\_i: 
\[ t2 := t2 + 4 \]
\[ *t2 := tVal_i \]
\[ t4 := t4 - 1 \]
if t4 > 0 goto Label\_i

String Constants

Source:
write ("hello");
write ("i = ", i);

SPARC Target Code:
str1: .asciz "hello"
str2: .asciz "i = 
...

New fields: StringConst.nameOfConstant and StringConst.next
You must set stringList to point to this list.

It is easier to add the next string to the front of the list.
The strings will naturally print in reverse order.
New Procedure Names

Source:

```
procedure foo1 (...) is
    procedure bar (...) is
procedure foo2 (...) is
    procedure bar (...) is
```

SPARC Target:

```
foo1:  save
...  
bar:   save
...  
foo2:  save
...  
bar:   save
...  
```

Just change “id”:

```
ProcDecl
id
id
```

```
CallStmt
id
myProc
```

```
newProcName()
"bar"
"p17_bar"
```

© Harry H. Porter, 2006
New Procedure Names

Source:

```plaintext
procedure foo1 (...) is
  ...
procedure bar (...) is
  ...
procedure foo2 (...) is
  ...
procedure bar (...) is
```

SPARC Target:

```plaintext
foo1:  save
  ...
bar:   save
  ...
foo2:  save
  ...
bar:   save
  ...
```

"bar"
```
newProcName()
```

"p17_bar"

Just change “id”:

```
ProcDecl
id
```

```
CallStmt
id
```

myProc

```
"p17_bar"
```

Benefits:

User-defined names show through.
No conflicts with other assembly symbols:

```
main    float1
str1    float2
str2    ...
Label_1  runtimeError1
Label_2  runtimeError2
...
```

For IR.call, we used a pointer to the ProcDecl:

```plaintext
mov   %l3,%o1
call  p17_bar
nop
mov   %o0,%l4
```
Numerical Constants

Integers

IR Code:
\[ x := x + 5 \]

SPARC:
\[
\begin{align*}
  ld & \ldots, %l4 \\
  add & %l4, 5, %l4 \\
  st & %l4, \ldots
\end{align*}
\]

The value can be included as a literal in the instruction stream... No problem.

Floating Point Literals:

IR Code:
\[ y := y + 5.67 \]

The value cannot be included as a literal... Must have a constant!

\[
\begin{align*}
  float4: & .single \ 0r5.67 \\
  \ldots \\
  set & \text{float4}, %l5 \\
  ldf & [ %l5 ], %f0 \\
  fadd & %f_, %f0, %f_ \\
  \ldots
\end{align*}
\]

Must build this list.
(in reverse order)
Must give each RealConst a name.

```java
String newFloatName () { ... }
```

© Harry H. Porter, 2006
Read and Write Statements

**New IR instructions:**
- `readInt x`
- `readFloat x`
- `writeInt y`
- `writeFloat y`
- `writeBoolean y`
- `writeString s`
- `writeNewLine`

- `writeBoolean b`  
  Will print either  
  - `true`  
  - `false`  
  depending on the value of “b”

Source:
```
read (i, j, x, y);
```

IR Code:
```
t3 := &i
readInt t3
t4 := &j
readInt t4
t5 := &x
readFloat t5
t6 := &y
readFloat t6
t3
```

These could be very complex L-Values, e.g., `r.a[i*foo(j)]`
Source:

```plaintext
write ("Hello there");
write ("value=", i*j, ...);
```

IR Code:

```plaintext
writeString str1
writeNewLine
writeString str2
t17 := i * j
writeInt t17...
writeNewLine
```

Source:

```plaintext
write ("ans=", (xxx or xxx));
```

IR Code:

```plaintext
writeString str3
xxx
xxx
xxx
xxx
xxx
t18 := 1
writeBoolean t18
writeNewLine
```
Peephole Patterns

Pattern:
goto L
L:

Action:
Delete the goto instruction (but keep the label)

Pattern:
if ... goto L1
goto L2
L1:

Action:
Replace with
if not(...) goto L2
L1:

Negating Comparisons:

\[
\begin{array}{c}
= & \rightarrow & \neq \\
< & \rightarrow & \geq \\
& & \\
\geq & \rightarrow & < \\
\geq & \rightarrow & < \\
\end{array}
\]
Peephole Patterns

Pattern: goto L
          L:
Action: Delete the goto instruction (but keep the label)

Pattern: if ... goto L1
          goto L2
          L1:
Action: Replace with
          if not(...) goto L2
          L1:

Pattern: goto L
          <anything except a label>
Action: Delete the second instruction
         Restart (without advancing)
         to eliminate a series of instructions

Negating Comparisons:
=  !
>  !
<  !
#  !
$  !

Other Peephole Patterns

Pattern: x := 4 * 7
Action: Replace with
         x := 28
Other Peephole Patterns

Pattern:
\[ x := 4 \times 7 \]

Action:
*Replace with*
\[ x := 28 \]

Patterns:
\[ x := z + 0 \]
\[ y := w \times 1 \]
\[ a := b \div 1.0 \]
\[ c := d - 0 \]

Action:
*Replace with*
\[ x := z \]
\[ y := w \]
\[ a := b \]
\[ c := d \]

Other Patterns:
\[ x := 0 + z \]
\[ y := 0 \times w \]
\[ y := w \times 0 \]
\[ e := 0 - f \]
Ideas for Implementation

Look for pattern starting at p1.
If found...
  Modify list of instructions;
  Repeat without incrementing
If not found
  Increment all pointers.

What to hand in, if you do PEEPHOLE?
• Email Peephole.java
• Turn in a short write-up
  ...with an annotated output listing

Another Peephole Idea

Associate a “count” with each LABEL instruction.
Keep track of how many GOTOs branch to that label.

Sometimes we eliminate GOTOs
  ...Then must reduce the “count”

If a LABEL’s count goes to zero...
  Delete it!
This may make some instructions unreachable.

Make repeated passes over the IR
 instruction list, until a pass is made in
which no instructions are eliminated.
[This process must terminate.... Why?]
Another Peephole Idea

Associate a “count” with each LABEL instruction.
Keep track of how many GOTOs branch to that label.

Sometimes we eliminate GOTOs
...Then must reduce the “count”

If a LABEL’s count goes to zero...
Delete it!
This may make some instructions unreachable.

Make repeated passes over the IR instruction list, until a pass is made in which no instructions are eliminated.
[This process must terminate.... Why?]

Example

Count = 0
L3 : 
xxxx
xxxx
goto L4

Count = 1
L4 : 
yyyy
yyyy
yyyy
goto L5

Example

Count = 1
L4 : 
xxxx
xxxx
Another Peephole Idea

Associate a “count” with each LABEL instruction. Keep track of how many GOTOs branch to that label.

Sometimes we eliminate GOTOs ...Then must reduce the “count”

If a LABEL’s count goes to zero... Delete it!
This may make some instructions unreachable.

Make repeated passes over the IR instruction list, until a pass is made in which no instructions are eliminated. [This process must terminate.... Why?]

Example

```
xxxx
xxxx
goto L4

YYYY
YYYY
goto L5

L4:

Example

```

Count = 1

L4:

Example

```

Count = 1

L4:

Example

```
Another Peephole Idea

Associate a “count” with each LABEL instruction. Keep track of how many GOTOs branch to that label.

Sometimes we eliminate GOTOs
...Then must reduce the “count”

If a LABEL’s count goes to zero...
Delete it!
This may make some instructions unreachable.

Make repeated passes over the IR instruction list, until a pass is made in which no instructions are eliminated.

[This process must terminate.... Why?]

Example

```
x xxxx
L4
```

```
Count = 1
```

```
y yyy
goto L5
```

```
L4:
```

```
x xxxx
```

```
Example

```
x xxxx
goto L4
```

```
Count = 1
```

```
y yyy
goto L5
```

```
L4:
```

```
x xxxx
```

© Harry H. Porter, 2006
Another Peephole Idea

Associate a “count” with each LABEL instruction. Keep track of how many GOTOs branch to that label.

Sometimes we eliminate GOTOs
...Then must reduce the “count”

If a LABEL’s count goes to zero...
Delete it!
This may make some instructions unreachable.

Make repeated passes over the IR instruction list, until a pass is made in which no instructions are eliminated.
[This process must terminate.... Why?]

Example

\[
\text{Count} = 1
\]

L4:

Example

\[
\text{Count} = 0
\]

L4:
Another Peephole Idea

Associate a “count” with each LABEL instruction.
Keep track of how many GOTOs branch to that label.

Sometimes we eliminate GOTOs
...Then must reduce the “count”

If a LABEL’s count goes to zero...
Delete it!
This may make some instructions unreachable.

Make repeated passes over the IR instruction list, until a pass is made in which no instructions are eliminated.

[This process must terminate.... Why?]

Example

```plaintext
xxxx
xxxx

xxxx
xxxx
```