Assignment 7
Due tonight at midnight
Solution class files have been posted

Assignment 8
Posted tomorrow
Due next Tuesday at midnight

Today:
Canonical form IR transformation
Assignment 8 overview

Why canonical form?
“Clean” IR form will simplify machine code generation
   Eliminate embedded ESEQ’s
   Standardize SEQ form

Want match to hardware
   1 way branches, not 2

To simplify remainder
   Convert IR to canonical form
4 major changes for canonical:
1. All code into basic blocks
   Start with label
   End with jump/cjump
2. False label always follows cjump
   Mirrors real branch instructions
3. No embedded side effects
   No eseqs
4. Orderly seqs
   Only left child is seq

1. Basic blocks

Basic block is linear stretch of code
   No branches in
   No branches out
   All statements in block computed

Starts with label
Ends with JUMP/CJUMP

To obtain basic blocks from statements
   Start at beginning of function
   Expect to see label
   Add code to existing block until either new
   label encountered...
   Close previous block
   Append jump at end
   or until jump/cjump encountered
   Close this block
generate IR and canonicalize

L0: MOVE T1,0
    MOVE T2,0
    JUMP L3
L1: MOVE T2,+(T2,T1)
    L2: MOVE T1,+(T1,1)
    L3: CJUMP T1,4,LT,L1,L4
L4: MOVE %0,T2
    JUMP L5:

first block: beginning to JUMP

L0: MOVET2,+(T2,T1)
L1: MOVET1,+(T1,1)
L2: CJUMP T1,4,LT,L1,L4
L4: MOVE %0,T2
JUMP L5:

second block L1,MOVE

L0: MOVE T1,+(T1,1)
L3: CJUMP T1,4,LT,L1,L4
L4: MOVE %0,T2
JUMP L5:
2. Canonical jumps

Always want false label of cjump to immediately follow cjump statement

Reflects machine code
Five possible forms for cjumps, corresponding canonicalizing strategies:

a) Next statement is false label
   No rearrangement necessary for canonical
b) Next statement is true label
   Swap comparison op
   Swap true/false labels in cjump
c) False is not next, not false target for
   any other cjumps
   Reposition false block as next block
d) True label is not next, not false target for
   anyone
   Swap comparison op
   Swap true/false labels in cjump
   Reposition true (now false) block as next block
e) Neither true/false block available for
   repositioning (already exist as false target)
   Generate new false label
   Followed by jump to original false target
   Position this basic block as next

Example

IR transformation of if (a == b) t else f :

cjump a==b, T,F
label T
t
jump X
label F
f
label X
To canonicalize (note case b)...

Swap condition, true/false labels in cjump

cjump a! = b, F,T
label T
t
jump X
label F
t
label X

Canonical transformation of cjumps
e xample of trace generation

Trace: A sequence of statements that could be consecutively executed

Moving False block immediately after cjump constructs a trace

3. Eliminating ESeqs (Overview)

Intuition:

First move ESeq to top of expression, via transformations based on identities

When Eseq at top, turn into Seq
Some example identities:

\[ \text{ESeq}(s_1, \text{ESeq}(s_2, e)) = \text{ESeq}(\text{Seq}(s_1, s_2), e) \]

\[ \text{Mem}(\text{ESeq}(s, e)) = \text{ESeq}(s, \text{Mem}(e)) \]

Many others; ask if interested in more details (not necessary for us)

Consider function call (in Lake):

\[ \text{print}(\text{row}[r] ? "0" : ".") \]

Parse tree looks like:

```
Call
  ID print
  Conditional
    row[r]
      "0"
      "="
```

IR example:

```
Call
  Name print
  ESeq
    CJumpNEQ T=L1 F=L2
      row[r]
        0
    L1:
      MOVE(TEMP, '0')
      JUMP L3
    L2:
      MOVE(TEMP, ".")
    L3:
      TEMP
```
multiple ESeq’s generate Seq’s
BinExpr transformation:
Move ESeq to top
replaced by expr of ESeq

Can do same transform for right

but there's a catch…

Consider code:

```c
int a;
int f()
{
    return g(a, g2());
}
```
IR for call g(a, g2()) is:

```
  | call |
  |     |
  | g   |
  |     |
  | mem |
  | ESeq|
  |     |
  | a   |
  |     |
  | call|
  |     |
  | reg 00|
  |     |
  | g2  |
```

simple canonical transformation is

```
  | call |
  |     |
  | g   |
  |     |
  | mem |
  | ESeq|
  |     |
  | a   |
  |     |
  | call|
  |     |
  | reg 00|
  |     |
  | g2  |
  | Seq |
  |     |
  | call|
  |     |
  | g2  |
  |     |
  | mem |
  |     |
  | a   |
```

Original IR implied
read memory of a
call g2
call g1

New IR does
call g2
read memory of a
call g1

Change to order of evaluation:
If g2 mutates a, semantics not preserved in transformation!
C/C++ guarantee order between statements
Side effect of statement occurs before side effect of later statement
But no guarantee at expression level
Compiler defined ordering
Unpredictable behavior
Pond guarantees order of expressions
Requires more work
May generate slower code
Predictable, principled behavior

Transformation for preserving evaluation order:
Given ESeq in right
‘simple’ transformation for right only allowed if s commutes with left
Expressions e1, e2 commute iff ordering does not affect evaluation
If expressions don’t commute, must compute left first to preserve evaluation order
Save result in a temporary

order preserving canonical IR is

```
<table>
<thead>
<tr>
<th>Seq</th>
</tr>
</thead>
<tbody>
<tr>
<td>move</td>
</tr>
<tr>
<td>t</td>
</tr>
<tr>
<td>mem</td>
</tr>
<tr>
<td>a</td>
</tr>
<tr>
<td>call</td>
</tr>
<tr>
<td>call</td>
</tr>
<tr>
<td>g</td>
</tr>
<tr>
<td>t</td>
</tr>
<tr>
<td>reg0</td>
</tr>
</tbody>
</table>
```

11
Always safe to use order preserving transformation
   But may generate more verbose code
   Want to use simpler when safe

Use conservative approximation of commutative
   Returns true only if certainly commutative

Most conservative definition:
   If any storage modified by statement, cannot commute with expression

Slight improvement:
   Consider whether modified storage may be referred to in expression

 Called *may alias* problem

Example:
   A constant commutes with any statement.
   Different degrees of precision achievable.
   *Alias analysis* provides various algorithms for achieving maximum precision.

Advanced topic; project?
4. Flattening seqs
Seq allowed as left child of Seq
Seq allowed at top
Seq disallowed anywhere else
Simple transformation
Based on identity:

Seq(s1, Seq(s2, s3)) = Seq(Seq(s1, s2), s3)

Assignment 8:
Canonical form transformations.

1. Written problems: identify basic blocks, canonicalize cjump by hand
2. Programming problems:
   a. Create list of basic blocks
   b. Canonicalize cjump
   c. Eliminate ESeqs (done for you)
   d. Clean up Seqs (done for you)

Programming:

a. Identify basic blocks:
   Define IRFunction.findBlocks
b. Canonicalize cjump:
   Define IRFunction.traceBlocks
c. Canonicalizing ESeqs:
   IRESeq.clean (already defined)
d. Cleaning up Seqs:
   IRSeq.clean (already defined)
Assignment 8:

Out tomorrow

More details next class