CS 536 Announcements for Wednesday, April 17, 2024

Last Time
- continue code generation
  - function declaration, call, and return
  - expressions
  - literals
  - assignment
  - I/O

Today
- wrap up code generation
  - tuple access
  - control-flow constructs
- introduce control flow graphs

Next Time
- optimization

P6 : Codegen class

Constants for registers and logical constants
  e.g., FP, SP, T0, T1

Methods to help automatically generate code

\[
generate(\text{opcode, \ldots\ args \ldots})
\]
  e.g., \(generate("add", "$t0", "$t0", "$t1")\)

writes out \(add\ \$t0, \$t0, \$t1\)

versions for fewer args as well

\[
generateIndexed(\text{opcode,\ arg1, \ arg2, \ offset})
\]
  e.g., \(generateIndexed("lw", "$t0", \$t1", -12)\)

writes out \(lw\ \$t0, -12($t1)\)

//\ genPush(reg) / genPop(reg)

nextLabel() – returns a unique string to use as a label

genLabel(L) – places a label
Code Generation for Tuple Access

Offset from base of tuple to certain field is known statically
- compiler can do the math for the slot address
- not true for languages with pointers!

Example

```plaintext
tuple Inner {
    logical hi.
    integer there.
    integer c.
}.

tuple Demo {
    tuple Inner b.
    integer val.
}.

void f{} []
    tuple Demo inst.

    ... = inst:b:c.

    inst:b:c = ... .
```
Control flow graphs

Kinds of control flow
- function calls
- selection
- repetition
- short-circuited operators

Control flow graph (CFG)
- important representation for program optimization
- helpful way to visualize source code

Example
Line1: li $t0, 4
Line2: li $t1, 3
Line3: add $t0, $t0, $t1
Line4: sw $t0, val
Line5: b Line2
Line6: sw $t0, 0($sp)
Line7: subu $sp, $sp, 4
Kinds of control flow in base

if exp [
  ...
] else [
  ...
]

What is needed at the assembly-code level

- branching
  - unconditional  
    b label
  - conditional  
    beq r1, src, label

- labels
Code generation for if statements

base code example:

```plaintext
if a == b [
    $ body of if
]
```

Code generation steps:

- get a label for end of construct
- generate code for expression
- generate conditional branch
- generate body of if
- place end-of-construct label

Code generation for if-else statements

base code example:

```plaintext
if a > b [
    $ body of if
] else [
    $ body of else
]```
Code generation for if-else statements (cont.)

base code:
   if a > b [  
      $ body of if  
   ]
   else [  
      $ body of else  
   ]

MIPS code outline:
   lw $t0, addr_a
   push $t0

   lw $t0, addr_b
   push $t0

   pop $t1
   pop $t0
   sgt $t0, $t0, $t1
   push $t0

   pop $t0
   beq $t0, FALSE, falseLabel
   .
   .
   b doneIfLabel

falseLabel:
   .
   .
   .

doneIfLabel:
Code generation for if-else statements (cont.)

Revisiting the CFG

lw $t0, addr_a
push $t0
lw $t0, addr_b
push $t0
pop $t1
pop $t0
sgt $t0, $t0, $t1
push $t0
pop $t0
beq $t0, FALSE, falseLabel
.
. # code for true branch
.
b doneIfLabel

falseLabel:
.
. # code for false branch
.

doneIfLabel:

Code generation for while statements

base code example:

while a == b [
    $ body of while
]

MIPS tips

It’s really easy to get confused with assembly

Some suggestions

- start simple: main procedure that prints the value 1
  - get procedure main to compile and run
  - function prologue and epilogue
  - trivial case of expressions: evaluating the constant 1, which pushes a 1 on the stack
  - printing: write << 1.
- then grow your compiler incrementally
  - expressions
  - control constructs
  - call/return

Create super simple test cases

- main procedure: print the value of some expression
- create more and more complicated expressions

Regression suite

- rerun all test cases to check whether you introduced a bug
- more suggestions
  - try writing desired assembly code by hand before having the compiler generate it
  - draw pictures of program flow
  - have your compiler put in detailed comments in the assembly code it emits