Lecture 20

Code Generation Conclusion

CS 241: Foundations of Sequential Programs
Fall 2009

Troy Vasiga et al
University of Waterloo

Conventions

Remember we have:

- $0
- $1
- $2
- $3
- $30
- $31

plus

- $4
- $11

Code Structure

Prologue

Generated Code

Epilogue

Problems with println

Could create a println function in Epilogue that is called by the "standard" calling procedure.

Problems:

Fixing problem with println

A better solution:

More problems:

Declarations

Rule: dcls $\rightarrow$ dcls dcl BECOMES NUM SEMI

We can handle duplicate variable declaration errors here.

But we need a convention to store variables.
Three options.

Option A: Variables in Registers

One variable per register, stored somehow in symbol table.

Problems:

Option B: Variables in RAM using .word

Each variable x in WL program corresponds to label x in MIPS.

Problems:

Option C: Variables on Stack

- Suppose we have n variables
- In prologue,
- How to get n?
- Picture in RAM
- Where is the ith variable?
- What about \$30 changing?

Giving variables values

We have the ... dcl BECOMES NUM SEMI part of the rule.

Also, stmt $\rightarrow$ ID BECOMES expr SEMI

A note about parameters in $1 and $2

Booleans

The only spot where booleans are allowed are in control structures.

Rule: test $\rightarrow$ expr LT expr

Conventions:

MIPS code:

While loop

stmt $\rightarrow$ WHILE LPAREN test RPAREN LBRACE statements RBRACE

MIPS code:

Problems:

If statement

IF test ... statements ELSE statements

Two choices: what code to place first

Back to booleans

Rule: test $\longrightarrow$ expr GT expr

Rule: test $\longrightarrow$ expr NE expr

Finishing Booleans

Rule: test $\longrightarrow$ expr EQ expr

Rule: test $\longrightarrow$ expr GE expr
Rule: test $\longrightarrow$ expr LE expr

Optimization

Large and interesting topics: we will just skim.
Recall that we can have an infinite number of MIPS programs that are equivalent to a particular WL program. What are the "good" MIPS programs that we would like to generate?

- Reducing speed of instructions in actual code: hard
- Reducing number of statements (stack usages, register allocation)