Lecture 18+19

Code Generation for WLP4MPMP

CS 241: Foundations of Sequential Programs
Spring 2017

Troy Vasiga et al. (where Dan Holtby ∈ al)
University of Waterloo
Code Generation (A9/A10)

▶ Input:

▶ Output:

Number of different outputs:
Code Generation Issues

► Correctness:
  ▶ Outputs a MIPS assembly program that does the same thing as the input program does.

► Ease of writing compiler:
  ▶ WLP4 = designed to be easy
  ▶ C++ = designed to be hard (or at least it feels that way sometimes)

► Efficiency of the compiler:
  ▶ $O(n) = \text{good, } O(n^2) = \text{fail.}$
  ▶ Try to have a small coefficient, too.
  ▶ Pro Tip: valgrind can profile your code for you.
  ▶ Pro Tip: Racket can profile your code for you, too

► Efficiency of the compiled code:
  ▶ Different ways to measure
  ▶ There’s a contest to reduce the file size (that’s one measure)
  ▶ Your A9+10 code does not need to produce optimized code
Fundamental idea:

- traverse the parse tree and gather information
- bottom up. terminals are easy! \{\text{INT, ”42”}\} becomes:
  \begin{verbatim}
  lis $5
  .word 42
  \end{verbatim}
- (it doesn’t need to be register 5)
First rule

```
main → INT WAIN LPAREN dcl COMMA dcl RPAREN LBRACE dcls
statements RETURN expr SEMI ...
```

```
= preamble code + code(dcl₁) + code(dcl₂) + code(dcls) +
code(statements) + code(expr) + cleanup code
```

”+” is some sort of concatenation that needs to know ”how” to combine them.
And make sure registers don’t clash.
Three more simple rules

\[
\text{statements } \rightarrow \epsilon \\
\text{code(}\text{statements}\text{)} = """
\]

\[
\text{expr } \rightarrow \text{term} \\
\text{code(}\text{expr}\text{)} = \text{code(}\text{term}\text{)}
\]

\[
\text{term } \rightarrow \text{factor} \\
\text{code(}\text{term}\text{)} = \text{code(}\text{factor}\text{)}
\]

In general:
\[
\text{A } \rightarrow \text{ } \beta : \text{code(A)} = \sum_{s \in \beta} \text{code(s)}
\]
Recall that the input must be *semantically valid*. What do the WLP4 programs look like?

```c
int wain(int a, int b) {
    return a;
}
```

What do the equivalent MIPS programs look like?

What do the parse trees look like?
Changes to the symbol table
How to store variables

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 WLP4 program corresponds to label $x$ in MIPS.

Example:

Problems:
Option C: Variables on Stack

- Suppose we have \( n \) variables
- How to get \( n \)?
- Picture in RAM
- Where is the \( i \)th variable?
- What about $30 changing?
Some conventions

Remember we have:

▶ $0

▶ $1

▶ $2

▶ $3

▶ $30

▶ $31

plus

▶ $4

▶ $11

▶ $29
Code Structure

Prologue

Generated Code

Epilogue
One more rule (A9P2)

factor $\rightarrow$ LPAREN expr RPAREN

- What this gives us:
A9P3: Full expressions

- int wain(int a, int b) { return a+b; }
- int wain(int a, int b) { return a+b+a; }
A9P3: Parse Tree
A9P4: Printing

statements → PRINTLN LPAREN expr RPAREN SEMI

A first attempt:
return code(expr) +
    move $3 to $1 +
call print

Problems:
Where is print?

Solution:
`.import print in Prologue`

Problems:
How to not destroy $1?

Three possible solutions (two of which have problems):

▶

▶

▶
A special note about parameters in $1$ and $2$

THEY

AREN’T

(special, that is)
Other advice

- comments

- rule-based structure of your code
  - each grammar rule will represent one function/procedure/step in our compilation process
  - the parse tree indicates which subtrees need to produce the code before the “entire” code has been produced

- read the WHOLE assignment before beginning and PLAN ahead

- test, test, test
Declarations (A9P5)

Rule: \( dcls \rightarrow dcls \text{ dcl BECOMES NUM SEMI} \)
Giving variables values

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

Also, stmt → lvalue BECOMES expr SEMI
(and, for now, lvalue is just an ID: no pointers yet)
Booleans (A9P6)

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

Rule: test → expr LT expr

Conventions:

MIPS code:
While loop (A9P6)

statement → WHILE LPAREN test RPAREN LBRACE statements RBRACE

MIPS code:

Long-range problem:
If statement (A9P8)

statement → IF test ... statements ... ELSE ... statements ...

Two choices: what code to place first
Back to booleans (A9P7)

Rule: test $\rightarrow$ expr GT expr

Rule: test $\rightarrow$ expr NE expr
Finishing Booleans (A9P7)

Rule: test $\rightarrow$ expr $\text{EQ}$ expr

Rule: test $\rightarrow$ expr $\text{GE}$ expr

Rule: test $\rightarrow$ expr $\text{LE}$ expr