Lecture 17+18

Code Generation for WLM

CS 241: Foundations of Sequential Programs
Winter 2018

Troy Vasiga et al
University of Waterloo

Code Generation (A9/A10)

- Input:
  -
  -
- Output:

Number of different outputs:

Code Generation Issues

- Correctness
- Ease of writing compiler
- Efficiency of the compiler
- Efficiency of the compiled code

Code Generation via Syntax-Directed Translation

Fundamental idea:

- traverse the parse tree and gather information
- When generating the code for a node, it will rely on the code of children
- **Crucial point:** when discussing "combining code" in the following slides, we will use "code(X)+code(Y)" to indicate "put the code of X first followed by the code of Y".
- **Do not use string concatenation or its equivalent to do this! Your running time will blow up! Create the final code by a tree traversal!**

Initial conventions

Before we get into the actual code generation, we should map out a plan.

In particular, we should have some conventions which we will follow. These include:

- parameters to `wain` are in `$1` and `$2`; return value into `$3`
- we will use some registers for storing special constants
- we will use some registers to hold temporary values
- we will use some registers to hold other special non-constant values

First rule

procedure → INT ID LPAREN params RPAREN
LBRACE dcls statements RETURN expr SEMI RBRACE

Three more simple rules

statements $\rightarrow \epsilon$

expr $\rightarrow$ term

term $\rightarrow$ factor

In general, when $\alpha, \beta \in N$:
$\alpha \rightarrow \beta$

A9P2

Recall that the input must be *semantically valid*.
What do the WLM programs look like?

```
int wain(int a, int b) {
    return 0;
}
```

(where 0 could be any other number).

What does the equivalent MIPS code look like?

We will see that we will produce many more lines than this in our final solution, but more on that as we continue...

A9P3

Recall that the input must be *semantically valid*.
What do the WLM programs look like?

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

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

What to the parse trees look like?

What do the equivalent MIPS programs 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

Option C: Variables on Stack

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

A special note about parameters in $1 and $2

Some conventions

Remember we have:

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

plus

- \$4
- \$11
- \$29

Design decisions

Look for opportunities to generalize and abstract away details.
Specifically:

- `push`
- `pop`
- `getVariable`
- `...`

Code Structure

Prologue

Generated Code

Epilogue

Wain Structure

Prologue

Generated Code

Epilogue

One more rule (A9P4)

factor $\rightarrow$ LPAREN expr RPAREN

- What this gives us:

A9P4: Full expressions

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

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

A9P4: Parse Tree

A9P4: Pseudocode

A9P5: getchar

factor → pcall
pcall → ID LPAREN RPAREN

where ID must be `getchar`

Where is `getchar`?

Solution 1: put it as a procedure after `wain`

Solution 2: "inline" the procedure

Short-term solutions may not lead to long-term solutions

A9P6: `putchar`

statements $\rightarrow$ statements statement
statement $\rightarrow$ pcall SEMI
pcall $\rightarrow$ ID LPAREN expr RPAREN

Other advice

- comments everywhere!
- 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 (A10P1)

dcls $\rightarrow$ dcls dcl BECOMES NUM SEMI

Giving variables values (A10P1)

statement $\rightarrow$ ID BECOMES expr SEMI

Booleans (A10P2)

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

Rule: test $\rightarrow$ expr LT expr

Conventions:

MIPS code:

`while` loop (A10P2)

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

MIPS code:

Long-range problem:

`if` statement (A10P4)

statement $\rightarrow$ IF test ... statements ... ELSE ... statements ...

Two choices: what code to place first

Back to booleans (A10P3)

Rule: test $\rightarrow$ expr GT expr

Rule: test $\rightarrow$ expr NE expr

Finishing Booleans (A10P3)

Rule: test $\rightarrow$ expr EQ expr

Rule: test $\rightarrow$ expr GE expr

Rule: test $\rightarrow$ expr LE expr