Assignment 5

Code Generation

(Due: May/28/2008)

In programming assignment 5, we will use the parser and the type checker implemented in the programming assignment #4 as a base to generate real instructions for C-- programs. I would highly encourage you to use your own type checker as a base. However, if your type checker is not working, or is missing major semantic information gathering capability, you may use the package we provided in E3

The target machine model is the MIPS architecture. SPIM (A MIPS simulator) will be used to verify the correctness of the generated code. The output from your compiler will be MIPS assembly code rather than MIPS machine binary. You can run the assembly code on spim for correctness verification, or run it on xspim, an interactive version of spim with X interface, to debug your code. One sample assembly code (NOT optimized) output for the factorial program is included in the appendix.

Please refer to the SPIM document, "SPIM S20: A MIPS R2000 Simulator" for the details on the MIPS assembly instruction set, directives, and calling conventions.

You should generate comments (comments in assembler files begin with a sharp-sign #, everything from the sharp-sign to the end of the line is ignored) with the code that identify which program statement corresponds to the generated assembly instructions. (**FOR TEACHER**: Don’t know is that necessary?)

**In assignment#5, you need to produce and demonstrate correct code for the following features:**

1) Assignment statements

2) Arithmetic expressions

3) Control statements: while, if-then-else

4) Parameterless procedure calls

5) Read and Write function calls

More features (as listed below) will be implemented in assignment #6.

6) Short-circuit boolean expressions

7) Variable initializations

8) Procedure and function calls with parameters

9) For loops

10) Multiple dimensional arrays

11) Implicit type conversions

PS: For variable initialization, we support only simple constant initializations, such as

Int I=1;

Float a=2.0;

**How to handle Read and Write?**

Read and Write will be translated into SPIM system calls. SPIM provides

a small set of OS-like services through the "syscall" instructions.

For example, a write function call

write("enter a number");

will be translated as follows:

First, the string "enter a number" will be placed in the data segment such as

.data

m1: .asciiz "enter a number"

Then the generated code will be as follows:

li $v0 4 # syscall 4 means a call to print\_str;

la $a0 m1 # the address of the string is passed by register $a0 (i.e. register $r4)

syscall

A read function call such as

n = read()

will be translated as follows:

li $v0 5 # syscall 5 means read\_int; the returned result will be in $v0

syscall

The integer value that has been read will return in $v0 (i.e. register $r2). So you need to copy the result from $v0 to a register or store it to a local variable such as:

move $r9, $v0 # if n is allocated to $r9

or

sw $v0, -4($fp) # if n is a local variable not allocated to a reg

Please refer to the sample code in the appendix. More detailed description on system calls can be found in the SPIM document, "SPIM S20: A MIPS R2000 Simulator".

**Appendix I Sample output from a C-- compiler**

int result;

int fact(int n)

{

if (n == 1)

{

return n;

}

else

{

return (n\*fact(n-1));

}

}

int main()

{

int n;

write("Enter a number:");

n = read();

if (n > 1)

{

result = fact(n);

}

else

{

result = 1;

}

write("The factorial is ");

write(result);

}

**Sample un-optimized code from a C-- compiler**

.data

\_result: .word 0

.text

fact:

# prologue sequence

sw $ra, 0($sp)

sw $fp, -4($sp)

add $fp, $sp, -4

add $sp, $sp, -8

lw $2, \_framesize\_of\_fact

sub $sp,$sp,$2

sw $8,32($sp)

sw $9,28($sp)

sw $10,24($sp)

sw $11,20($sp)

sw $12,16($sp)

sw $13,12($sp)

sw $14,8($sp)

sw $15,4($sp)

\_begin\_fact:

\_Ltest\_1:

lw $8, 8($fp)

li $9, 1

seq $10, $8, $9

beqz $10, \_Lexit\_1

lw $9, 8($fp)

move $v0, $9

j \_end\_fact

j \_Lelse\_1

\_Lexit\_1:

lw $8, 8($fp)

li $11, 1

sub $12, $8, $11

addi $sp, $sp, -4

sw $12, 4($sp)

jal fact

addi $sp, $sp, 4

move $12, $v0

lw $11, 8($fp)

mul $8, $11, $12

move $v0, $8

j \_end\_fact

\_Lelse\_1:

# epilogue sequence

\_end\_fact:

lw $8,32($sp)

lw $9,28($sp)

lw $10,24($sp)

lw $11,20($sp)

lw $12,16($sp)

lw $13,12($sp)

lw $14,8($sp)

lw $15,4($sp)

lw $ra, 4($fp)

add $sp, $fp, 4

lw $fp, 0($fp)

jr $ra

.data

\_framesize\_of\_fact: .word 32

.data

.text

main:

# prologue sequence

sw $ra, 0($sp)

sw $fp, -4($sp)

add $fp, $sp, -4

add $sp, $sp, -8

lw $2, \_framesize\_of\_main

sub $sp,$sp,$2

sw $8,32($sp)

sw $9,28($sp)

sw $10,24($sp)

sw $11,20($sp)

sw $12,16($sp)

sw $13,12($sp)

sw $14,8($sp)

sw $15,4($sp)

\_begin\_main:

li $v0, 4

la $a0, \_m2

syscall

li $v0, 5

syscall

sw $v0, -4($fp)

\_Ltest\_3:

lw $8, -4($fp)

li $9, 1

sgt $10, $8, $9

beqz $10, \_Lexit\_3

addi $sp, $sp, -4

lw $9, -4($fp)

sw $9, 4($sp)

jal fact

addi $sp, $sp, 4

move $9, $v0

sw $9, \_result

j \_Lelse\_3

\_Lexit\_3:

li $9, 1

sw $9, \_result

\_Lelse\_3:

li $v0, 4

la $a0, \_m4

syscall

li $v0, 1

lw $a0, \_result

syscall

# epilogue sequence

\_end\_main:

lw $8,32($sp)

lw $9,28($sp)

lw $10,24($sp)

lw $11,20($sp)

lw $12,16($sp)

lw $13,12($sp)

lw $14,8($sp)

lw $15,4($sp)

lw $ra, 4($fp)

add $sp, $fp, 4

lw $fp, 0($fp)

li $v0, 10

syscall

.data

\_framesize\_of\_main: .word 36

\_m2: .asciiz "Enter a number:"

\_m4: .asciiz "The factorial is "

**Appendix II Installation of the MIPS simulator**

SPIM official site : http://pages.cs.wisc.edu/~larus/spim.html

**installation**

1. Download http://www.cs.wisc.edu/~larus/SPIM/pcspim.zip

2. Decompress the file

3. Click the setup.exe

**run**

1. run Pcspim

2. click “File->Open” to choose your assembly program

3. click “Simulator->Go” to run your code

**Note for 64-bit users:** The first time you run Spim it may complain about a missing exception handler (exceptions.s). If you see this message, open Settings, look under "Load exception file", and change the path to the following (or something similar):

C:\Program Files (x86)\PCSpim\exceptions.s

**Linux:**

**installation**

Use “apt-get install spim” to get and install spim

Another method is use the work stations. SPIM is already installed in the workstations linux1 bsd[3 – 5] (address : linux1.cs.nctu.edu.tw & bsd[3-5].cs.nctu.edu.tw)

**run**

1. Enter the spim console : spim

2. load the MIPS assembly program : load “filename”

3. run the program : run

**Additional Notes:**

a) You may assume the identifier names will not exceed 256 characters. However, the number of distinct identifiers should not be limited.

b) In the Homework5 directory you may find the following files:

1) src/lexer3.l the sample lex program that you may start with

2) src/header.h contains AST data structures

3) src/Makefile

4) src/parser.y template YACC file with incomplete production rules

5) src/functions.c functions

6) pattern/\*.c test data files

7) tar.sh packaging script file

(TO\_DO)

**Submission requirements:**

1) DO NOT change the executable name (parser).

2) Use the script file “tar.sh” to wrap up your assignment works into a single file. Then upload your packaged file to e3.

Usage: ./tar.sh source\_directory studentID1\_studentID2 (all student IDs in your team) version\_number

Example: ./tar.sh hw 9912345\_9912346 ver1

Output: 9912345\_9912346\_ver1.tar.bz2 (submit this file)

3) We grade the assignments on the linux1 server. Before summiting your assignment, you should make sure your version works correctly on linux1.

4) Do not submit one pass compilation approach in assignment 5; this assignment is based on AST.

Use a separate e-mail to inform TAs about the students in your group.

The TA responsible for this homework assignment is: