Assignment 5

Code Generation

(Posted 12/21/2014, Due: 1/07/2014)

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 yet fully functional, you are welcome to use the package we provided in cieba (which will be released on 12/22).

The target machine model is the ARM architecture. QEMU (A machine emulator that can emulate ARM on PC/x86-based workstations) will be used to verify the correctness of the generated code. The output file named output.s from your compiler will be ARM assembly code rather than ARM machine code. However, the input executable for QEMU is in ELF(Executable and Linkable Format or Extensible Linkling Format), so we need to use some tools to convert our output.s to an executable in ELF. In this assignment, we have attached an instructional document, called ***how\_to***, which explains how to use tools to build needed ELF files and how to debug them efficiently. In order to reduce your efforts in building your test environment, we also provide a VirtualBox image which contains all the tools needed. One sample assembly code (NOT optimized) output for the factorial function is included in the appendix.

Some useful reference:

1. Procedure Call Standard for the ARM® Architecture

<http://infocenter.arm.com/help/topic/com.arm.doc.ihi0042e/IHI0042E_aapcs.pdf>

1. QEMU website and download site

<http://wiki.qemu.org/Main_Page>

**Grading requirements:**

We will use qemu-arm version 2.0.2 to test run your processed executables.

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

1) Assignment statements

2) Arithmetic expressions

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

4) Parameterless procedure calls

5) **Read** and **Write** I/O 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 external function calls.

For example:

**write("Enter a number\n");**

could be translated as follows:

First, the string "Enter a number\n" will be placed in the data segment such as:

.data

\_CONSTANT\_0: .ascii "Enter a number\n\000"

.align 2

Then the generated code will be as follows:

ldr r4, =\_CONSTANT\_0 # Load address of \_CONSTANT\_0 to r4

mov r0, r4 # move r4 to r0, r0 is used to pass parameter. It is used to pass the string label to \_write\_str.

bl \_write\_str #jump to \_write\_str

# **a=read();**

bl \_read\_int

mov r4, r0 # the read integer will be put in r0.

str r4, [fp, #-4]

# **b=fread();**

bl \_read\_float

vmov s16, s0 # the read float number will be put in s0.

vstr.f32 s16, [fp, #-8]

# **write(a);** a is an integer variable

ldr r4, [fp, #-4]

mov r0, r4 #r0 is used to pass the value you would like to write.

bl \_write\_int

# **write(b);** b is a floating point variable.

vldr.f32 s16, [fp, #-8]

vmov s0, s16 #s0 is used to pass the value you would like to write out.

bl \_write\_float

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

int n;

int fact()

{

if (n == 1)

{

return n;

}

else

{

n =n-1;

return (n\*fact());

}

}

Because of the usage of our specific tools, main() is replaced by MAIN().

int MAIN()

{

int result;

write("Enter a number:");

n = read();

n = n+1;

if (n > 1)

{

result = fact();

}

else

{

result = 1;

}

write("The factorial is ");

write(result);

write("\n");

}

Sample un-optimized code from a C--/ARM compiler

.data

\_g\_n: .word 0

.text

.text

\_start\_fact:

str lr, [sp, #0]

str fp, [sp, #-4]

add fp, sp, #-4

add sp, sp, #-8

ldr lr, =\_frameSize\_fact

ldr lr, [lr, #0]

sub sp, sp, lr

str r4, [sp, #4]

str r5, [sp, #8]

str r6, [sp, #12]

str r7, [sp, #16]

str r8, [sp, #20]

str r9, [sp, #24]

str r10, [sp, #28]

str r11, [sp, #32]

vstr.f32 s16, [sp, #36]

vstr.f32 s17, [sp, #40]

vstr.f32 s18, [sp, #44]

vstr.f32 s19, [sp, #48]

vstr.f32 s20, [sp, #52]

vstr.f32 s21, [sp, #56]

vstr.f32 s22, [sp, #60]

vstr.f32 s23, [sp, #64]

ldr r10, =\_g\_n

ldr r4, [r10,#0]

.data

\_CONSTANT\_1: .word 1

.text

ldr r5, =\_CONSTANT\_1

ldr r5, [r5, #0]

cmp r4, r5

mov r4, #0

moveq r4, #1

cmp r4, #0

beq \_elseLabel\_0

ldr r10, =\_g\_n

ldr r4, [r10,#0]

mov r0, r4

b \_end\_fact

b \_ifExitLabel\_0

\_elseLabel\_0:

ldr r10, =\_g\_n

ldr r4, [r10,#0]

.data

\_CONSTANT\_2: .word 1

.text

ldr r5, =\_CONSTANT\_2

ldr r5, [r5, #0]

sub r4, r4, r5

ldr r5, =\_g\_n

str r4, [r5, #0]

ldr r10, =\_g\_n

ldr r4, [r10,#0]

bl \_start\_fact

mov r5, r0

mul r4, r4, r5

mov r0, r4

b \_end\_fact

\_ifExitLabel\_0:

\_end\_fact:

ldr r4, [sp, #4]

ldr r5, [sp, #8]

ldr r6, [sp, #12]

ldr r7, [sp, #16]

ldr r8, [sp, #20]

ldr r9, [sp, #24]

ldr r10, [sp, #28]

ldr r11, [sp, #32]

vldr.32 s16, [sp, #36]

vldr.32 s17, [sp, #40]

vldr.32 s18, [sp, #44]

vldr.32 s19, [sp, #48]

vldr.32 s20, [sp, #52]

vldr.32 s21, [sp, #56]

vldr.32 s22, [sp, #60]

vldr.32 s23, [sp, #64]

ldr lr, [fp, #4]

mov sp, fp

add sp, sp, #4

ldr fp, [fp,#0]

bx lr

.data

\_frameSize\_fact: .word 64

.text

\_start\_MAIN:

str lr, [sp, #0]

str fp, [sp, #-4]

add fp, sp, #-4

add sp, sp, #-8

ldr lr, =\_frameSize\_MAIN

ldr lr, [lr, #0]

sub sp, sp, lr

str r4, [sp, #4]

str r5, [sp, #8]

str r6, [sp, #12]

str r7, [sp, #16]

str r8, [sp, #20]

str r9, [sp, #24]

str r10, [sp, #28]

str r11, [sp, #32]

vstr.f32 s16, [sp, #36]

vstr.f32 s17, [sp, #40]

vstr.f32 s18, [sp, #44]

vstr.f32 s19, [sp, #48]

vstr.f32 s20, [sp, #52]

vstr.f32 s21, [sp, #56]

vstr.f32 s22, [sp, #60]

vstr.f32 s23, [sp, #64]

.data

\_CONSTANT\_3: .ascii "Enter a number:\000"

.align 2

.text

ldr r4, =\_CONSTANT\_3

mov r0, r4

bl \_write\_str

bl \_read\_int

mov r4, r0

ldr r5, =\_g\_n

str r4, [r5, #0]

ldr r10, =\_g\_n

ldr r4, [r10,#0]

.data

\_CONSTANT\_4: .word 1

.text

ldr r5, =\_CONSTANT\_4

ldr r5, [r5, #0]

add r4, r4, r5

ldr r5, =\_g\_n

str r4, [r5, #0]

ldr r10, =\_g\_n

ldr r4, [r10,#0]

.data

\_CONSTANT\_6: .word 1

.text

ldr r5, =\_CONSTANT\_6

ldr r5, [r5, #0]

cmp r4, r5

mov r4, #0

movgt r4, #1

cmp r4, #0

beq \_elseLabel\_5

bl \_start\_fact

mov r4, r0

str r4, [fp, #-4]

b \_ifExitLabel\_5

\_elseLabel\_5:

.data

\_CONSTANT\_7: .word 1

.text

ldr r4, =\_CONSTANT\_7

ldr r4, [r4, #0]

str r4, [fp, #-4]

\_ifExitLabel\_5:

.data

\_CONSTANT\_8: .ascii "The factorial is \000"

.align 2

.text

ldr r4, =\_CONSTANT\_8

mov r0, r4

bl \_write\_str

ldr r4, [fp, #-4]

mov r0, r4

bl \_write\_int

.data

\_CONSTANT\_9: .ascii "\n\000"

.align 2

.text

ldr r4, =\_CONSTANT\_9

mov r0, r4

bl \_write\_str

\_end\_MAIN:

ldr r4, [sp, #4]

ldr r5, [sp, #8]

ldr r6, [sp, #12]

ldr r7, [sp, #16]

ldr r8, [sp, #20]

ldr r9, [sp, #24]

ldr r10, [sp, #28]

ldr r11, [sp, #32]

vldr.32 s16, [sp, #36]

vldr.32 s17, [sp, #40]

vldr.32 s18, [sp, #44]

vldr.32 s19, [sp, #48]

vldr.32 s20, [sp, #52]

vldr.32 s21, [sp, #56]

vldr.32 s22, [sp, #60]

vldr.32 s23, [sp, #64]

ldr lr, [fp, #4]

mov sp, fp

add sp, sp, #4

ldr fp, [fp,#0]

bx lr

.data

\_frameSize\_MAIN: .word 68

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 hw5 directory you may find the following files:

1) src/lexer3.l the lex program

2) src/header.h contains AST data structures

3) src/Makefile

4) src/parser.y

5) src/functions.c supporting functions

6) pattern/\*.c test data files

Submission requirements:

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

2) Your compiler should produce the output ARM code in a file called “output.s”.  
3) Compress all modules needed to generate your compiler. Then upload your packaged file to ceiba.

3) We grade the assignments on the QEMU installed on Ubuntu 14.04. Before summiting your assignment, you should make sure your version can be compiled by using “make” and works correctly on such environment.