Assignment 5

Code Generation

(Posted 12/13/2015, Due:12/31/2015)

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 12/13).

The target machine model is the ARMv8 architecture. QEMU (A machine emulator that can emulate ARMv8 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 ARMv8 assembly code rather than ARMv8 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 ARMv8® Architecture

http://infocenter.arm.com/help/topic/com.arm.doc.ihi0055b/IHI0055B\_aapcs64.pdf

1. QEMU website and download site

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

**Grading requirements:**

We will use qemu-aarch64 version 2.0.0 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 3

Then the generated code will be as follows:

ldr x9, =\_CONSTANT\_0 # Load address of \_CONSTANT\_0 to x9

mov x0, x9 # move x9 to x0, x0 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 w9, w0 # the read integer will be put in w0.

str w9, [x29, #-4]

# **b=fread();**

bl \_read\_float

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

str s16, [x29, #-8]

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

ldr w9, [x29, #-4]

mov w0, w9 #w0 is used to pass the value you would like to write.

bl \_write\_int

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

ldr s16, [x29, #-8]

fmov 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--/ARMv8 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--/ARMv8 compiler

.data

\_g\_n: .word 0

.text

.text

\_start\_fact:

str x30, [sp, #0]

str x29, [sp, #-8]

add x29, sp, #-8

add sp, sp, #-16

ldr x30, =\_frameSize\_fact

ldr x30, [x30, #0]

sub sp, sp, w30

str x9, [sp, #8]

str x10, [sp, #16]

str x11, [sp, #24]

str x12, [sp, #32]

str x13, [sp, #40]

str x14, [sp, #48]

str x15, [sp, #56]

str s16, [sp, #64]

str s17, [sp, #68]

str s18, [sp, #72]

str s19, [sp, #76]

str s20, [sp, #80]

str s21, [sp, #84]

str s22, [sp, #88]

str s23, [sp, #92]

# }

ldr x14, =\_g\_n

ldr w9, [x14,#0]

.data

\_CONSTANT\_1: .word 1

.align 3

.text

ldr w10, \_CONSTANT\_1

cmp w9, w10

cset w9, eq

cmp w9, #0

beq \_elseLabel\_0

# }

# return n;

ldr x14, =\_g\_n

ldr w9, [x14,#0]

mov w0, w9

b \_end\_fact

b \_ifExitLabel\_0

\_elseLabel\_0:

# }

# n =n-1;

ldr x14, =\_g\_n

ldr w9, [x14,#0]

.data

\_CONSTANT\_2: .word 1

.align 3

.text

ldr w10, \_CONSTANT\_2

sub w9, w9, w10

ldr x10, =\_g\_n

str w9, [x10, #0]

# return (n\*fact());

ldr x14, =\_g\_n

ldr w9, [x14,#0]

bl \_start\_fact

mov w10, w0

mul w9, w9, w10

mov w0, w9

b \_end\_fact

\_ifExitLabel\_0:

\_end\_fact:

ldr x9, [sp, #8]

ldr x10, [sp, #16]

ldr x11, [sp, #24]

ldr x12, [sp, #32]

ldr x13, [sp, #40]

ldr x14, [sp, #48]

ldr x15, [sp, #56]

ldr s16, [sp, #64]

ldr s17, [sp, #68]

ldr s18, [sp, #72]

ldr s19, [sp, #76]

ldr s20, [sp, #80]

ldr s21, [sp, #84]

ldr s22, [sp, #88]

ldr s23, [sp, #92]

ldr x30, [x29, #8]

mov sp, x29

add sp, sp, #8

ldr x29, [x29,#0]

RET x30

.data

\_frameSize\_fact: .word 92

.text

\_start\_MAIN:

str x30, [sp, #0]

str x29, [sp, #-8]

add x29, sp, #-8

add sp, sp, #-16

ldr x30, =\_frameSize\_MAIN

ldr x30, [x30, #0]

sub sp, sp, w30

str x9, [sp, #8]

str x10, [sp, #16]

str x11, [sp, #24]

str x12, [sp, #32]

str x13, [sp, #40]

str x14, [sp, #48]

str x15, [sp, #56]

str s16, [sp, #64]

str s17, [sp, #68]

str s18, [sp, #72]

str s19, [sp, #76]

str s20, [sp, #80]

str s21, [sp, #84]

str s22, [sp, #88]

str s23, [sp, #92]

# write("Enter a number:");

.data

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

.align 3

.text

ldr x9, =\_CONSTANT\_3

mov x0, x9

bl \_write\_str

# n = read();

bl \_read\_int

mov w9, w0

ldr x10, =\_g\_n

str w9, [x10, #0]

# n = n+1;

ldr x14, =\_g\_n

ldr w9, [x14,#0]

.data

\_CONSTANT\_4: .word 1

.align 3

.text

ldr w10, \_CONSTANT\_4

add w9, w9, w10

ldr x10, =\_g\_n

str w9, [x10, #0]

# }

ldr x14, =\_g\_n

ldr w9, [x14,#0]

.data

\_CONSTANT\_6: .word 1

.align 3

.text

ldr w10, \_CONSTANT\_6

cmp w9, w10

cset w9, gt

cmp w9, #0

beq \_elseLabel\_5

# }

# result = fact();

bl \_start\_fact

mov w9, w0

str w9, [x29, #-4]

b \_ifExitLabel\_5

\_elseLabel\_5:

# }

# result = 1;

.data

\_CONSTANT\_7: .word 1

.align 3

.text

ldr w9, \_CONSTANT\_7

str w9, [x29, #-4]

\_ifExitLabel\_5:

# write("The factorial is ");

.data

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

.align 3

.text

ldr x9, =\_CONSTANT\_8

mov x0, x9

bl \_write\_str

# write(result);

ldr w9, [x29, #-4]

mov w0, w9

bl \_write\_int

# write("\n");

.data

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

.align 3

.text

ldr x9, =\_CONSTANT\_9

mov x0, x9

bl \_write\_str

\_end\_MAIN:

ldr x9, [sp, #8]

ldr x10, [sp, #16]

ldr x11, [sp, #24]

ldr x12, [sp, #32]

ldr x13, [sp, #40]

ldr x14, [sp, #48]

ldr x15, [sp, #56]

ldr s16, [sp, #64]

ldr s17, [sp, #68]

ldr s18, [sp, #72]

ldr s19, [sp, #76]

ldr s20, [sp, #80]

ldr s21, [sp, #84]

ldr s22, [sp, #88]

ldr s23, [sp, #92]

ldr x30, [x29, #8]

mov sp, x29

add sp, sp, #8

ldr x29, [x29,#0]

RET x30

.data

\_frameSize\_MAIN: .word 92

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 ARMv8 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.