Project4
Code Generation & Optimization
(DUE DATE: 2022/12/13)

Part 1. Code generation (70 pts)

We will use the parser and the type checker implemented in the previous programming assignment as a base to generate real instructions for C-- programs.

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 Linking Format), so we need to use some tools to convert our output.s to an executable in the ELF format. 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. One sample assembly code (NOT optimized) output for the factorial function is included in the appendix.

Some useful reference:

1. QEMU website and download site

http://wiki.qemu.org/Main_Page

2. A Guide to ARM64 / AArch64 Assembly on Linux with Shellcodes

https://modexp.wordpress.com/2018/10/30/arm64-assembly/

Grading requirements:

We will use qemu-aarch64 to test run your processed executables. Please see how_to for more details.

In this assignment, you need to produce and demonstrate the correct code for the following C— features. Then write down how/what you do on your report:

1) Assignment statements
2) Arithmetic expressions
3) Control statements: while, if-then-else
4) Parameterless procedure calls
5) Read and Write I/O calls (See Appendix I)

BONUS: More features (as listed below) 5pts each

(Please specify your works in the report)

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;
```

Part 2. Implement one optimization for your compiler (30 pts)

From Part 1, we are able to generate code of C-- programs. However, it's not optimized. In part 2, you should focus on one optimization technique & implement it in your parser.

Some of the most common optimizations:

• Function inlining
• Function cloning
• Constant folding
• Constant propagation
• Dead code elimination
• Loop-invariant code motion
• Common sub-expression elimination
• Data prefetching
• Loop unrolling

For Part 1 & 2, you should write a report.pdf about the work you've done. Please specify the optimization technique you used, then write on your report.

Additional Notes:

a) In the hw4 directory you may find the following files:

1) src/lexer.l	the lex program
2) src/header.h	contains AST data structures
3) src/Makefile	
4) src/parser.y	the parser program
5) src/functions.c	supporting functions
6) test/*.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 report.pdf and all your files as studentID_hw4.zip. Then upload to NTU Cool.
4) We grade the assignments on the QEMU installed on Ubuntu 16.04. Before summiting your assignment, you should make sure your version works fine on the environment.
5) You are free to modify Makefile. But make sure your make command works correctly.

Appendix I. 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 II Sample output from a C--/ARMv8 compiler

```
int n;
int fact()
{
    if (n == 1)
    {
        return n;
    }
    else

    {
        n =n-1;
        return (n*fact());
    }
}

int MAIN()
```

Because of the usage of our specific tools, main() is replaced by 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 w30, [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
```