EC 413 Computer Organization - Fall 2019
Lab 1: RISC-V Assembly Programming

Due: October 15th, 2019

1 Introduction and Background

This lab will introduce you to the RISC-V ISA. For this lab, you will write RISC-V RV32I assembly code and simulate it on the BRISC-V Simulator to show that it works.

1.1 RISC-V Assembly Code

For this lab we will only be using the 32-bit base RISC-V ISA (RV32I). This base ISA only supports simple integer operations with a handful of load, store, branch and jump instructions. Instructions for floating point and integer multiply are not included in RV32I. See the RISC-V Specification Document and the RISC-V Instruction Sheet in the class reading list (http://ascslab.org/courses/ec413/reading.html) to see which instructions are available in the RV32I base instruction set. Note that the instruction sheet and RISC-V spec both include instructions for extensions in addition to the base RV32I instructions. Do not use instructions from any extensions, they will not be supported by the simulator (or our hardware).

An example program (function_template.c), it's assembly (function_template.s) and it's disassembled binary (function_template.dump) have been included to show you how to follow the calling conventions used by RISC-V. These files are here for your reference. You will not need function_template.c to complete any of the problems. The function_template.s and function_template.dump files are only needed in Problem 1.

Pay attention to three things in the function_template.s file:

- How the stack pointer is managed
- How the frame pointer (s0) is managed
- How are function arguments and return values passed

To test your assembly, load it in the BRISC-V Simulator at the following link:
http://ascslab.org/research/briscv/simulator/simulator.html

A manual with directions for the Simulator can be found here:
http://ascslab.org/research/briscv/simulator/manual.html

2 Submission Requirements

Submit a zipped file named bu_id_lab1.zip with your assembly code in a directory named bu_id_lab1, where bu_id is replaced with your BU ID (including the 'U'). The code files must have the name bu_id_problem_x.s, where bu_id is replaced with your BU ID (including the 'U') and x is replaced with the problem number. You should submit a separate file for each problem. Submissions that do not follow these directions will not be graded.

3 Problems

Note that some problems ask you to test your function with only a single constant value, the TAs will test them with several. You should test your code with more than one input/constant value.

3.1 Problem 1

What real instructions are used to replace the "call" psudo-instruction. Describe all of them in detail. What are the register contents of the system before and after each instruction used in the call psudo-instruction. You only have to describe the state of registers that change. Submit a .txt or .PDF file for this problem.

3.2 Problem 2

(a) Write a function named "int_multiply" in RISC-V assembly to multiply two integers and return the result. The function must follow the calling conventions discussed in lecture and discussion sessions (see function_template.s). Your function must use the assembly equivalent of a for loop. Remember, there is no multiply instruction in RV32I.

The listing below shows equivalent C code (your output must be the same):

```
int int_multiply( int a, int b) {
  return a*b;
}
```

(b) Call your multiply function in a main function. Pass arguments of -2 and 2 to the function. Your main function must return the result of the multiplication. Your code must follow the calling conventions discussed in lecture and discussion sessions (see function_template.s).

The listing below shows equivalent C code (your output must be the same):

```
int int_multiply( int a, int b) {
  return a*b;
}

int main() {
  return int_multiply(-2, 2);
}
```

3.3 Problem 3

(a) Write a function to compute the factorial of a number using the assembly equivalent of a while loop. The function must be named "factorial". This function must call your multiply function. Your code must follow the calling conventions discussed in lecture and discussion sessions (see function_template.s).

The listing below shows equivalent C code (your output must be the same):

```
int int_multiply( int a, int b) {
  return a*b;
}

int factorial(int n) {
  int product = 1;
  while(n>0) {
    product = int_multiply(product, n);
    n--;
  }
  return product;
}
```

(b) Call your factorial function in a main function. Pass an argument of 8 to the function. The main function should return the result of the calculation. Your code must follow the calling conventions discussed in lecture and discussion sessions (see function_template.s).

The listing below shows equivalent C code (your output must be the same):

```
int int_multiply( int a, int b) {
  return a*b;
}

int factorial(int n) {
  int product = 1;
  while(n>0) {
    product = int_multiply(product, n);
    n--;
  }
  return product;
}

int main() {
  return factorial(8);
}
```

3.4 Problem 4

(a) Implement a non-recursive function to compute the n'th number in the Fibonacci sequence. The function must take in a single integer and return a single integer. The function must be

named "fib_nr" (nr for non-recursive) Your code must follow the calling conventions discussed in lecture and discussion sessions (see function_template.s).

The listing below shows equivalent C code (your output must be the same):

```
unsigned int fib_nr(unsigned int n) {
  unsigned int term1 = 0;
  unsigned int term2 = 1;
  unsigned int nextTerm;
  unsigned int i;

  for (i = 1; i <= n; ++i) {
    nextTerm = term1+term2;
    term1 = term2;
    term2 = nextTerm;
  }
  return term1;
}
```

(b) Call your fib_nr function in main with the following inputs: 1, 3, 8, 17. Place the return values of each function call in registers s2-s5. Your code must follow the calling conventions discussed in lecture and discussion sessions (see function_template.s).

The listing below shows (roughly) equivalent C code:

```
unsigned int fib_nr(unsigned int n) {
  unsigned int term1 = 0;
  unsigned int term2 = 1;
  unsigned int nextTerm;
  unsigned int i;

  for (i = 1; i <= n; ++i) {
    nextTerm = term1+term2;
    term1 = term2;
    term2 = nextTerm;
  }
  return term1;
}

int main(void) {
  // You must make sure the value in these variables is stored
  // in the register  associated with the variable name when
  // main returns.
  unsigned int s2, s3, s4, s5;
  s2 = fib_nr(1);
  s3 = fib_nr(3);
  s4 = fib_nr(8);
  s5 = fib_nr(17);
```

```
  return 0;
}
```

3.5 Problem 5

(a) Implement a recursive function to compute the n'th number in the Fibonacci sequence. The function must take in a single integer and return a single integer. The function must be named "fib_r" (r for recursive) Your code must follow the calling conventions discussed in lecture and discussion sessions (see function_template.s).

The listing below shows equivalent C code (your output must be the same):

```
unsigned int fib_r(unsigned int n) {
  if (n <= 1) {
    return n;
  } else {
    return fib_r(n-1)+fib_r(n-2);
  }
}
```

(b) Call your fib_r function in main with the following inputs: 1, 3, 8, 17. Place the return values of each function call in registers s2-s5. Your code must follow the calling conventions discussed in lecture and discussion sessions (see function_template.s).

The listing below shows (roughly) equivalent C code:

```
unsigned int fib_r(unsigned int n) {
  if (n <= 1) {
    return n;
  } else {
    return fib_r(n-1)+fib_r(n-2);
  }
}

int main(void) {
  // You must make sure the value in these variables is stored
  // in the register  associated with the variable name when
  // main returns.
  unsigned int s2, s3, s4, s5;
  s2 = fib_r(1);
  s3 = fib_r(3);
  s4 = fib_r(8);
  s5 = fib_r(17);

  return 0;
}
```

3.6 Problem 6

Answer the following performance analysis questions about the recursive and non-recursive Fibonacci functions implemented in problems 4 and 5. Place your answers in a .txt or .pdf file.

(a) How many instructions are in each function?

(b) What are the total number of executed instructions for each function with an input of 3.

(c) Which function uses the most registers?

(d) Which function uses the most memory (number of loads and stores)?

(e) Which function executes in the fewest number of cycles? Assume each instruction executes in 1 cycle.