Homework-1: IN210 Assembly Language and Computer Organization, Rung-Bin Lin
March 21, 2025, Computer Science and Engineering; International Bachelor Program in Informatics, Yuan Ze University

Computing n! with RISC-V ISA

Due: April 3, 2025

Problem Description

Modify the RISCV code given on page 44 of the lecture materials of Chapter 2 so that it can be repetitively executed to compute n! for different *n's*. Besides, the program should also count the instructions executed to compute n!. The count should only Include the instructions on page 44. For this part, you may employ the concept of basic blocks given on page 33 of the lecture materials of Chapter 2. In addition to practicing the RISCV instructions to compute *n*!, you will learn how to use Application Binary Interface (ABI) to carry out input/output task. Note that the given number *n* should not be too large such that *n*! will be too large to be represented in 32 bits.

You should use RISC-V instruction set simulator **Jupiter** (with I, F, M extensions) to develop and execute the assembly code. The Jupiter simulator can be downloaded from GitHub https://github.com/andrescv/Jupiter. To learn how to use RISC-V ISA to write a program that can be executed on Jupiter, you can study the code **Fibonacci.s** which can be downloaded from https://github.com/andrescv/Jupiter/blob/main/examples/fibonacci.s . You need to learn how the **assembler**, **directives**, **Ecalls**, etc. are used to develop the assembly code. In particular, **Ecalls** implement a set of ABI's (Application Binary Interface) that can be used by an assembly program to interact with an operating system to read data from a keyboard and write data onto a monitor. Refer to https://jupitersim.gitbook.io/jupiter/assembler/ecalls for Ecalls and https://jupitersim.gitbook.io/jupiter/assembler/directives for directives.

It is very easy to use Jupiter. Simply download the related files in its web site. Decompress the file and store the decompressed data in a new folder. For Windows version, find a file named **Jupiter.bat** in the **bin** folder. Doubly click this file to invoke Jupiter. Then, you can write, assemble, execute, and debug the code directly on Jupiter.

Input Format

When the code is executed on Jupiter, just follows the instruction presented on the monitor to provide input data. Each input line should contain only one data item (this restriction is imposed by the simulator).

Output Format

The output should be exactly the same as the example (except name and ID#) given below, starting from the banner Jupiter till the given input number -2. However, your name and student ID number should be correctly presented.

Homework-1: IN210 Assembly Language and Computer Organization, Rung-Bin Lin
March 21, 2025, Computer Science and Engineering; International Bachelor Program in Informatics, Yuan Ze University

Late Assignment is allowed till April 6. Eight points will be deducted for each delayed day. Such a graceful period is used to encourage you doing your best to complete your homework rather than delaying your homework.

What Should Be Handed In:

- Assembly code for the whole program. A comment should start with # at the beginning of the comment. **You should write as many comments as you can.** For example, each line of code has a comment. The file name of the assembly code should be **sID.s** where ID is your student ID number. A valid file name should look like s1091111.s . There is a header before the RISCV code. The header is formed by putting a number of comment lines together. The header should include the title of homework, your name, student ID number, and your answers to the following questions (see example below) :
  (a). How many hours did you spend for this homework?
  (a). Who has helped you solve the coding problems?
  (c). Do you copy someone's code? If yes, give the name of code owner and the number of lines of the code you copy.
- Some clips like the one shown in the example of input and output below. Save the clips as a file called **sID.pdf**, where ID is your student ID number. A valid file name for an output clip should look like s1091111.pdf. You can paste the clips into a WORD file and then create a PDF file from the WORD file. Only a PDF file is accepted.
- If you compress a file, only .zip file is acceptable.
- The homework will not be graded if you do not follow the above rules.

Other information:
https://github.com/riscv-non-isa/riscv-asm-manual/blob/master/riscv-asm.md
https://shakti.org.in/docs/risc-v-asm-manual.pdf
https://github.com/andrescv/Jupiter
https://jupitersim.gitbook.io/jupiter/assembler/directives

Code Example

```
##############################################################################
# Title: Fibonnaci Example
# Name: Rung-Bin Lin
# Student ID Number: s123456
# (a). How many hours did you spend for this homework?
# ANS: 18 hours
# (b). Who has helped you solve the coding problems?
# ANS: None
# (c). Do you copy someone's code?
# If yes, give the names of code owners and the number of lines of the code you copy.
# ANS: Mary Sun, Henry Lin. The number of lines of the code copied is 30.
##############################################################################
```

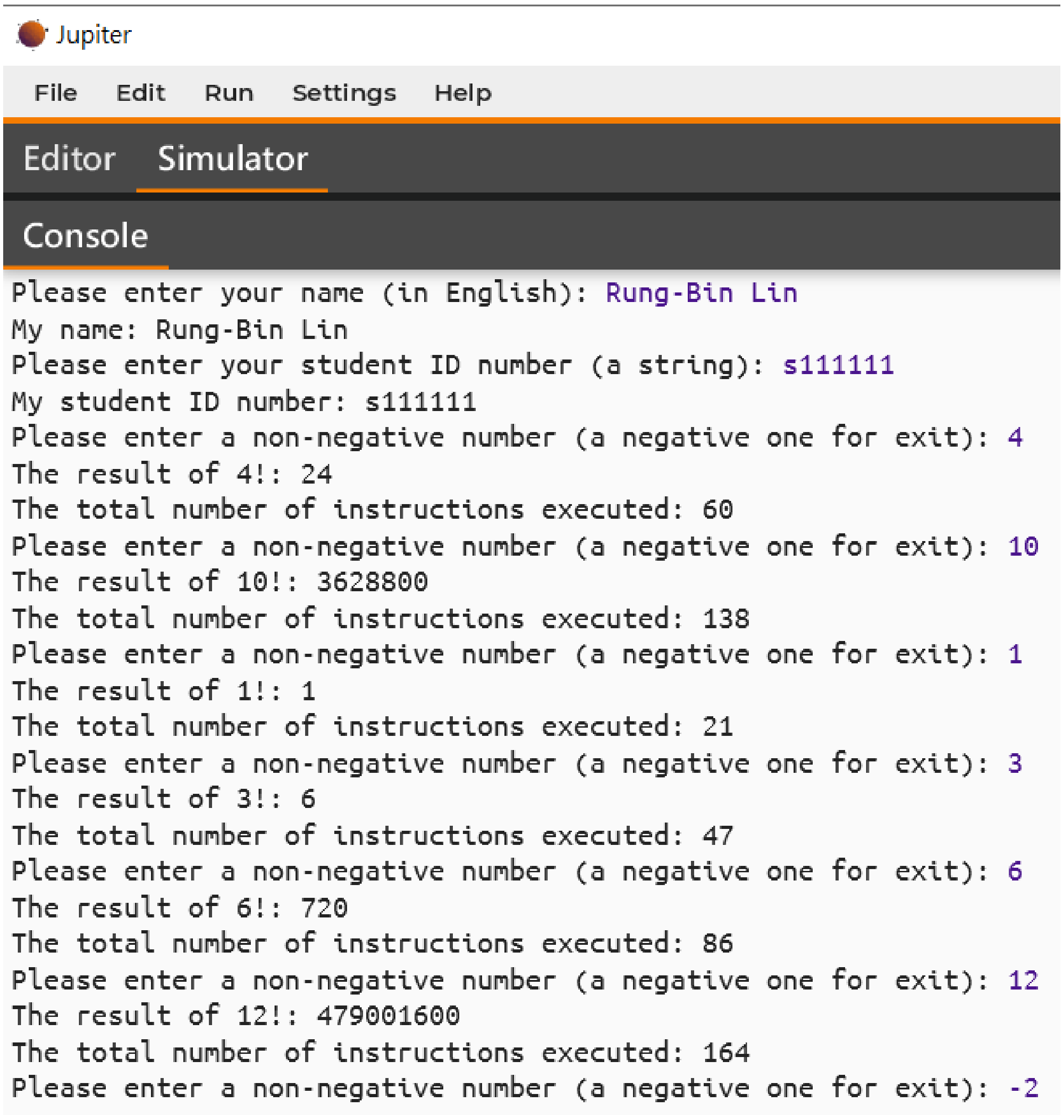
For n=7, the code does the following calculations:

f(1) = f(1) + f(0);
f(2) = f(1) + f(0) = 1 + 0 = 1
f(3) = f(2) + f(1) = 1 + 1 = 2
f(4) = f(3) + f(2) = 2 + 1 = 3
f(5) = f(4) + f(3) = 3 + 2 = 5
f(6) = (f5) + f(4) = 5 + 3 = 8
f(7) = f(6) + f(5) = 8 + 5 = 13

```
.globl __start
.rodata
   msg1: .string "Please enter a number: "
   msg2: .string "The "
   msg3: .string " fibonnaci number is: "

.text
__start:
   li t0, 0    # f(0) = 0
   li t1, 1    # f(1) = 1
   # prints msg1
   li a0, 4
   la a1, msg1
   ecall
   # reads an int and moves it to register t3
   li a0, 5    # the input number is stored in a0
   ecall
   mv t3, a0 # Move the input to t3, i.e., t3=a0
   # prints a newline
   li a0, 11
   li a1, '\n'
   ecall
   # prints msg2
   li a0, 4
   la a1, msg2
   ecall
   # prints the int value in t3
   li a0, 1
   mv a1, t3
   ecall
   # fibonnaci program
fib:
   beq t3, x0, finish    #check whether t3 is equal to zero
   add t2, t1, t0         # f(n) = f(n-1)+f(n-2) if not yet reach zero,
   mv t0, t1              # make f(n-1) into f(n-2)
   mv t1, t2              # make (n) into f(n-1)
   addi t3, t3, -1        # up date the number of times f(n) = f(n-1)+f(n-2)    yet to be performed
   j fib
finish:
   # prints msg3
   li a0, 4
   la a1, msg3
   ecall
   # prints the result in t0
   li a0, 1
   mv a1, t0
   ecall
   # ends the program with status code 0
   li a0, 10
   ecall
```

Example of Input and Output from a RISC-V Program for Computing n!



```
Please enter your name (in English): Rung-Bin Lin
My name: Rung-Bin Lin
Please enter your student ID number (a string): s111111
My student ID number: s111111
Please enter a non-negative number (a negative one for exit): 4
The result of 4!: 24
The total number of instructions executed: 60
Please enter a non-negative number (a negative one for exit): 10
The result of 10!: 3628800
The total number of instructions executed: 138
Please enter a non-negative number (a negative one for exit): 1
The result of 1!: 1
The total number of instructions executed: 21
Please enter a non-negative number (a negative one for exit): 3
The result of 3!: 6
The total number of instructions executed: 47
Please enter a non-negative number (a negative one for exit): 6
The result of 6!: 720
The total number of instructions executed: 86
Please enter a non-negative number (a negative one for exit): 12
The result of 12!: 479001600
The total number of instructions executed: 164
Please enter a non-negative number (a negative one for exit): -2
```