Computer Organization Report

Lab 1：RISC-V assembly code programming

Submission date：2020/04/02

Department：土木系

Student ID：0511330

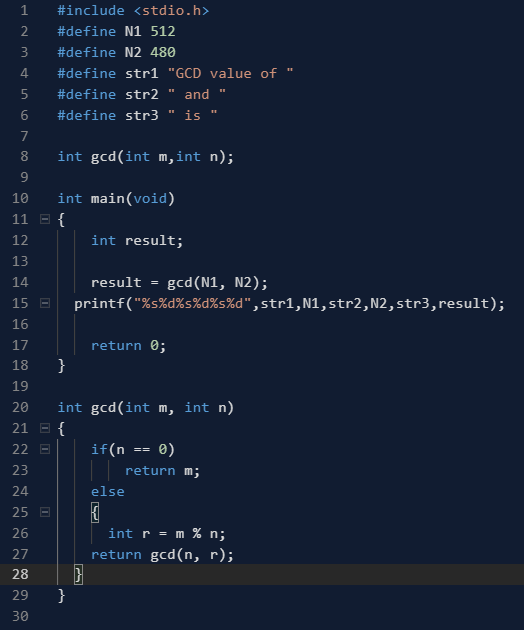
Name：劉紘華

1. **Introduction**

* RISC-V assembly language is one of **I**nstruction **S**et **A**rchitecture which is classified as a RISC ISA, i.e., Rudeced Instruction Set Computing ISA.
* In this Lab, we will use RISC-V simulator – **Ripes** to simulate the program and learn how to write RISC-V assembly code.
* We are required to convert C code to RISC-V assembly code, including GCD, Bubble Sort and Fibonacci Sequence.
* Link below is the official website where contains more information of **Ripes** simulator https://github.com/mortbopet/Ripes/wiki/Ripes-Introduction

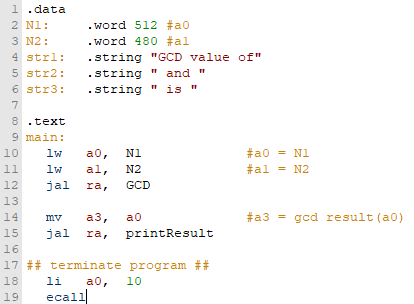
1. **Code Overview**
2. **Great Common Divisor (GCD)**

* **C code**

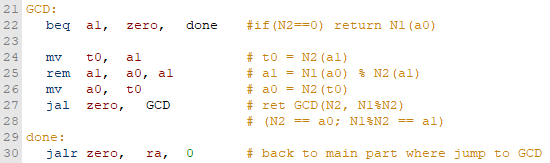
****

* **RISC-V assembly code**

**Main**

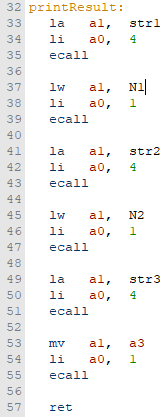
****

**GCD**

****

In fact, the RISC-V assembly code is easier than C code for GCD. We can recursively call GCD itself without any effort by using jal instruction. The condition of terminating the recursive part is when N2 = 0. It is quite straightforward to convert C to RISC-V.

**printResult**

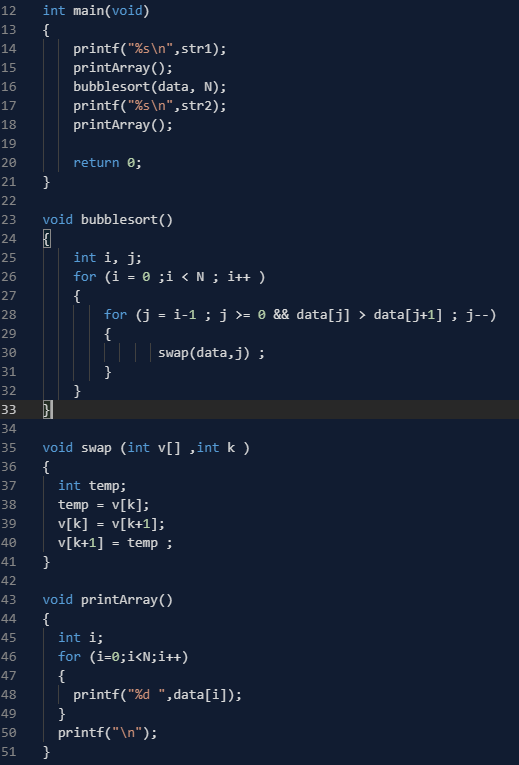
****

The tricky part in RISC-V assembly is the difference between printing String and word.

We use **load address** instruction (**la**) to print the string which we declare in “.data section” in the beginning. In contrast to printing the string, we use the **load word** instruction (**lw**) to print the word.

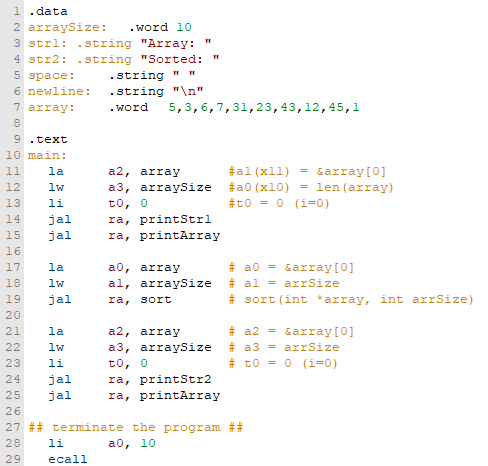
1. **Bubble Sort**

* **C code**

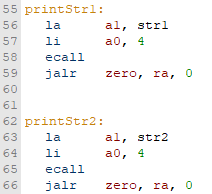
****

* **RISC-V assembly code**

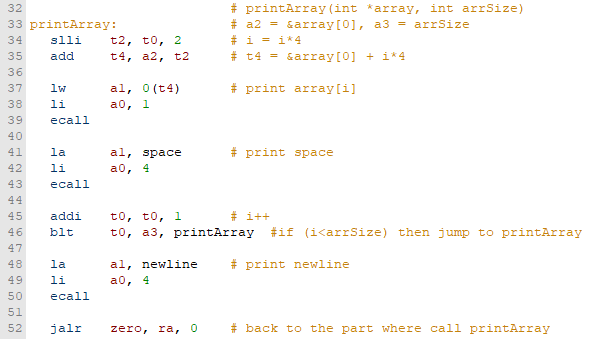
**main**

****

**printStr1** & **printStr2**

****

**printArray**

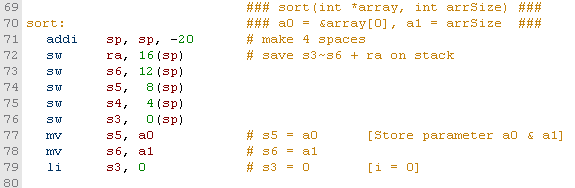
****

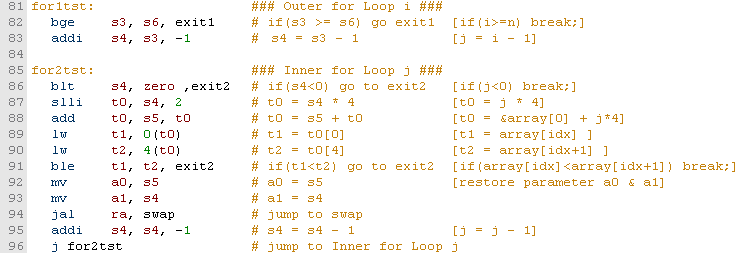
The difficult part in printArray is traversing the whole array. We have to distinguish between address and data. As shown in the figure above, the prototype of printArray function is passing the head of the array address (in Reg a2) and length of the array (in Reg a3). But why we store in a2 & a3 instead of a1 & a2? The reason is when we print the word or string, we have to load word or string address into Reg a1 and a2. For the convenience of accessing parameter, storing they in Reg. a2 & a3 is better.

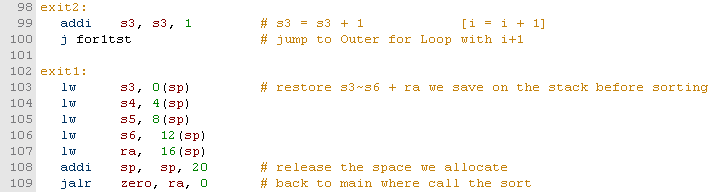
First, we initialize the value of Reg to with 0 and manipulate t0 as the **counter i** with increment by 1 when t0 is smaller than a1, i.e., **i** < len(array).

Next, Assign the value which equals to **i** multiplied by 4 to t2 (t2 = i\*4). Then we combine the value of t2 and a2, and assign it to t4 (t4 = t2 + a2). Notice that t4 here is the memory address. The final step is using loadword instruction to load the value from memory address t4 to Reg a1. Afterwards, we do the loop again until breaking the condition. ( I >= len(array))

**Sort**

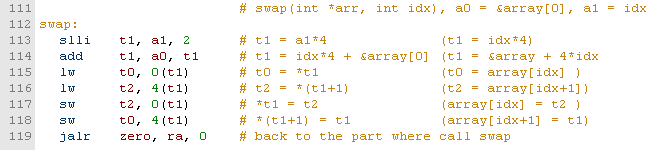






The sort part is a nested structure which is combination of 2 for loops. Counter i of Outer for loop begins from 0 to array length, which store in Reg s3. The counter j of inner for loop from i-1 decreases to 0, which store in Reg s4. Note that line 87~90 is the same way to access data stored in an array from memory address mentioned above (in **printArray**)**.** Array[idx] and Array[idx+1] will store in Reg s5 and s4. In line 92~93, we move these data into a0 & a1 passing arguments before swap.

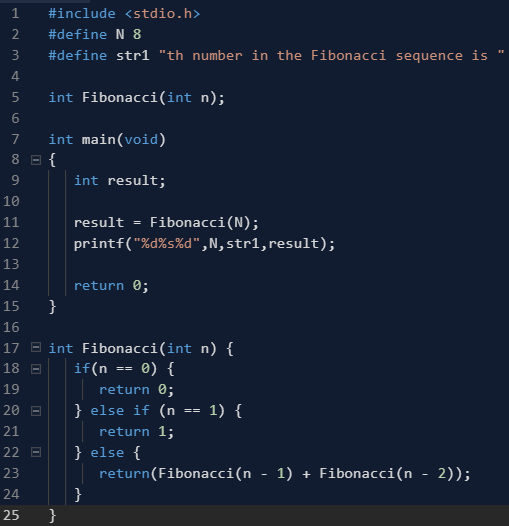
**Swap**

****

Parameters of array index and head of array address are in a1 and a0 respectively. Line 113~116 is the same way mentioned above to access data from array through the memory address.

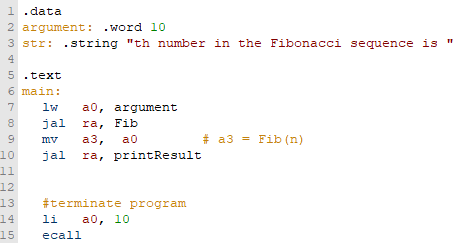
1. **Fibonacci Sequence**

* **C code**

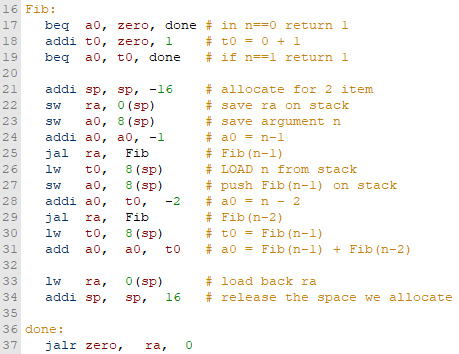
****

* **RISC-V assembly code**

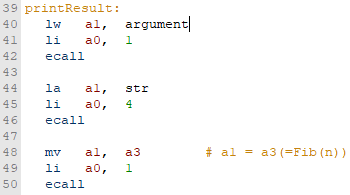
main

****

Fib

****

printResult

****

1. **Findings and Summary**

This is the first time to write a program which is relative to hardware. It is a really tough time for me at the beginning even making a simple function. Thanks to this **Ripes** simulator, I can check the value in each Register and figure out how all value passed in each steps. Honestly, after being familiar with basic rules and programming sense of RISC-V, it seems easier than C++ for me!