Lab1 report 109550168 林慧旻

(I use the v.1.0.4 ripes)

1. bubble_sort:

First, it enters the main function. In main block then it jumps to other functions:

➔ loop: initialize the array.

➔ printResult: print the “Array: 5 3 6 7 31 23 43 12 45 1”

➔ bubble1: start sorting

➔ printResult: print the “Sorted: “1 3 5 6 7 12 23 31 43 45”

bubble_sort:

a. Bubble2 block: control the outer loop. If i is smaller than n,it keeps jumping to bubble2 itself and then bubble3; else, it breaks the loop by a ret instruction.
b. Bubble3 block : control the inner loop. If j is less than zero, it returns to bubble2. If arr[j+1] < arr[j], it jumps to swap block to swap.
c. Bubblw4 block: do j- -, and jump back to bubble3.

Swap:

Do swap for arr[j+1] and arr[j] which arr[j] is bigger than arr[j+1].

printResult:

Print the strings and the unsorted and sorted array.

(Detailed counting process can be checked in the file ”bubble_sort_count.s”.)

Q1. There are total 1341 instructions.

Q2. The maximum number of variables pushed in stack is 0. Because in this example, there is no recursion usage.

2. Fibonacci:

First, it enters the main function. In main block then it jumps to other functions:

➔ Fib: To start the Fibonacci calculation.

➔ printResult: it is similar to bubble_sort one, it is ued to print the

result.

a. Fib block: To check if a0 is zero, than it keeps going and stopped by ret instruction. If not, it jumps to fib_not_0 block.
b. Fib_not_0 block: To check if a0 is 1(If a0 – 1 equals to 0), than it keeps going and will be stopped by ret instruction. If not, it jumps to fib_not_01 block.
c. Fib_not_01 block: If a0 is neither 1 or 0, it will be in here, and it subtracts 1 from a0 then jumps to fib, and then it subtracts 2 from a0 then jumps to fib again.

(Detailed counting process can be checked in the file ”fibonacci_count.s”.)

Q1. There are total 559 instructions.
Q2. The maximum number of variables pushed in stack is 6*3 = 18
Because there are 6 layers in the fib_not_01 block, and 3 variables in each layer:

```
sw    ra, 0(sp)
sw    a0, 8(sp)
sw    a1, 16(sp)
```

3. gcd

First, it enters the main function. In main block then it jumps to other functions:

➔ gcd: to do gcd calculation.
➔ printResult: it is similar to the bubble_sort and the fibonacci one, it is ued to print the result.

a. Gcd block: In here, it judges that if a1 is zero or not. If it is, then ret instruction will bring it to the end; else, it will jump to ngcd block.
b. Ngcd block: Compute the new remainder, and then keeps jumping to gcd block.

(Detailed counting process can be checked in the file ”gcd_count.s”.)

Q1. There are total 66 instructions.
Q2. The maximum number of variables pushed in stack is 1*3 = 3
Because there are at least 3 layers in the gcd block, and variable in each layer: sw ra, 8(sp)

4. Experience

I think I have learned a lot in this homework. Since I've been more interested in hardware things than software ones, learning assembly language could help me enhance the recognition to a computer more realistically. However, I have to say it is really not an easy task to me. The most difficult part is to find a way to translate c code to risc-v language. Also, getting used to this new computer language is another challenge. The "grammar" of assembly language is quite a lot different from c, c++, or even Verilog. Not to mention the following heavy work that I have to count the number of the assembly instructions. I felt faint when doing this and I am still not sure if I made them correct or not(there may be still some slight mistake compared to the fully correct answer.) But still, to make a conclusion, I still finish this homework now. Thank you,TA, for reading this report!