Environment : ripes v1.0.4

備註：若是 Load 上去在 jal, bnez 那幾行有錯誤的話，在尾端或逗號中間加個空格試看看

不同顏色筆跡代表的意義：

白色：instruction count

紅色：在哪一層 stack

綠色/藍色：該行指令中，右邊暫存器的值。或是迴圈中 i 和 j 的值。

---

1. bubble_sort.s

   Instruction Count : 498

```
.text
main:
    # printf("%s\n",str1);
    la a1, str1          1
    li a0, 4             2
    ecall                .

                         .
    la a1, endl          .
    li a0, 4             .
    ecall                6

    # initialize array
    jal ra, printArray   7
                         )99
    # sort array
    jal ra, sortArray    106
                         )+289
    la a1, str2          395
    li a0, 4             .
    ecall                -

    # initialize array
    jal ra, printArray   398 )99

    li a0, 10            497
    ecall                498  #
```

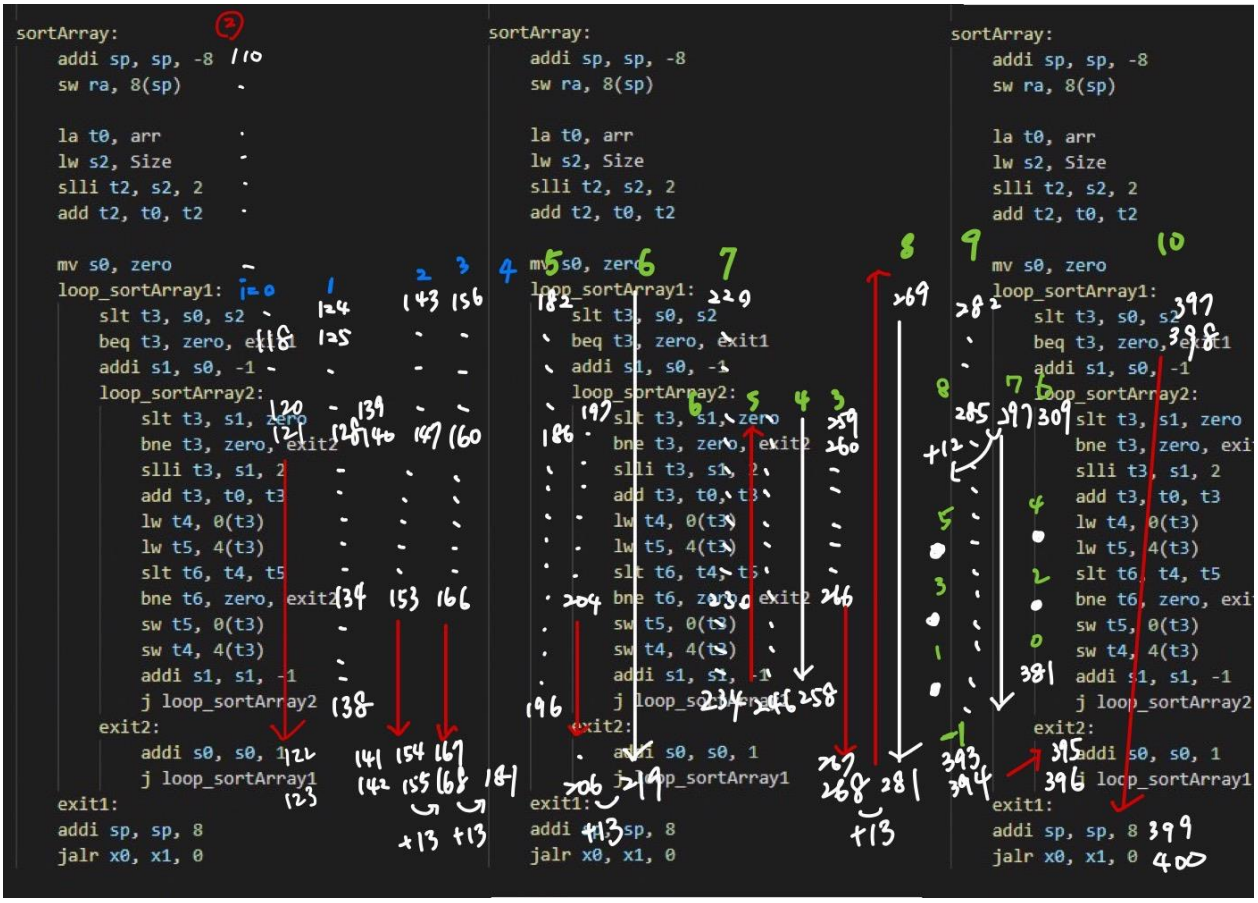
```
printArray:
    la t0, arr          8
    lw t1, Size         9
    slli t2, t1, 2      .
    add t2, t0, t2      .

    loop_iterateArray:
        lw t3, 0(t0)        12   21
        addi a0, x0, 1      .    .
        addi a1, t3, 0      .    .
        ecall               .    .

        la a1, spc          .    .
        li a0, 4            .    .
        ecall               .    .

        addi t0, t0, 4      .    .
        bltu t0, t2, loop_iterateArray   20   29 ... +8 = 101
                                         ①    ②    ③

    la a1, endl     102
    li a0, 4        103
    ecall           .
    ret             105
```



稍微修改了一下，下面是最後繳交的版本。

```
sortArray:
    la t0, arr
    lw s2, Size
    slli t2, s2, 2
    add t2, t0, t2

    mv s0, zero
    loop_sortArray1:
        slt t3, s0, s2
        beq t3, zero, exit1
        addi s1, s0, -1
        loop_sortArray2:
            slt t3, s1, zero
            bne t3, zero, exit2
            slli t3, s1, 2
            add t3, t0, t3
            lw t4, 0(t3)
            lw t5, 4(t3)
            slt t6, t4, t5
            bne t6, zero, exit2
            sw t5, 0(t3)
            sw t4, 4(t3)
            addi s1, s1, -1
            j loop_sortArray2
        exit2:
            addi s0, s0, 1
            j loop_sortArray1
    exit1:
    ret
```

少2步

⇒ total

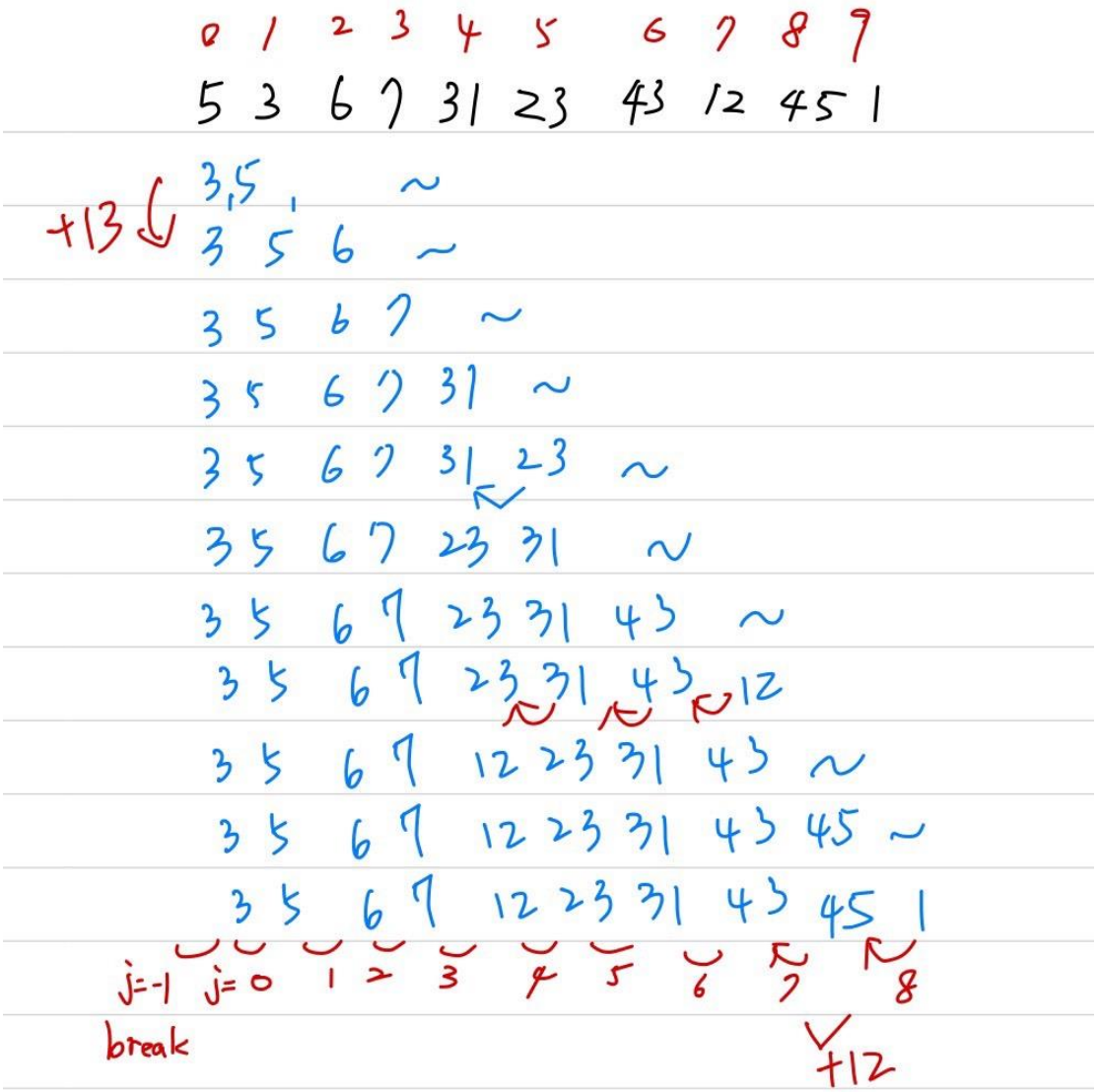
= 401
109
110

292 − 3

= 289 #

少一步

這張是在每個不同 i 的時候所要交換次數的統計：
由於每次不交換或是交換的步數都會一樣，所以想說在計算的時候將它紀錄下來，這樣每個相同動作就只要算一次，之後就不用對相同動作去 trace 每一次指令了。尤其在遞迴時蠻好用。



2. gcd.s

Instruction Count : 75

Stack Variables : 12

(我是看該層 stack 推了幾個 sw 算的)

```
.text
main:
        lw      a1,argument2    1
        lw      a0,argument1    2
        jal     ra, gcd         3

        lw      a1,argument1    49
        lw      a2,argument2    50
        jal     ra, printResult 51

        li      a0, 10          74
        ecall                   75

gcd:
        addi    sp,sp,-48       4 18 32
        sw      ra,44(sp)
        sw      s0,40(sp)
        addi    s0,sp,48
        sw      a0,-36(s0)
        sw      a1,-40(s0)
        lw      a5,-40(s0)
        bnez    a5,L4           11 25 39
        lw      a5,-36(s0)      40
        jal     ra,L5

L4:
        lw      a4,-36(s0)      12 26
        lw      a5,-40(s0)
        rem     a5,a4,a5
        mv      a1, a5
        lw      a0,-40(s0)
        jal     ra, gcd         17 31
        mv      a5,a0           48 47

L5:
        mv      a0,a5           47
        lw      ra,44(sp)       43
        lw      s0,40(sp)
        addi    sp,sp,48
        ret                     46
```

4×3 =12

```
printResult:
        mv      t0, a0          52
        mv      t1, a1
        mv      t2, a2

        la      a1, str1
        li      a0, 4
        ecall

        mv      a1, t1
        li      a0, 1
        ecall

        la      a1, str2
        li      a0, 4
        ecall

        mv      a1, t2
        li      a0, 1
        ecall

        la      a1, str3
        li      a0, 4
        ecall

        mv      a1, t0
        li      a0, 1
        ecall
        ret                     73
```

3. fibonacci.s

   Instruction Count : 863

   Stack Variables : 4*7 = 28

```
main:
        lw      a0,argument     1
        call    Fibonacci       2
                                )+845
        mv      a1, a0          847
        lw      a0, argument    .
        call    printResult     849
                                )
        li a0, 10               862
        ecall                   863

printResult:
        mv      t0, a0          850
        mv      t1, a1          .

        mv      a1, t0          .
        li      a0, 1           .
        ecall                   .

        la      a1, str1        .
        li      a0, 4           .
        ecall                   .

        mv      a1, t1          .
        li      a0, 1           .
        ecall                   .

        ret                     861
```

4*(8-1)

只是為了計算方便所以將三張疊在一起

```
Fibonacci:
        addi    sp,sp,-32
        sw      ra,28(sp)
        sw      s0,24(sp)
        sw      s1,20(sp)
        addi    s0,sp,32
        sw      a0,-20(s0)
        lw      a5,-20(s0)
        bnez    a5,.L4
        li      a5,0
        j       .L5
.L4:
        lw      a4,-20(s0)
        li      a5,1
        bne     a4,a5,.L6
        li      a5,1
        j       .L5
.L6:
        lw      a5,-20(s0)
        addi    a5,a5,-1
        mv      a0,a5
        call    Fibonacci
        mv      s1,a0
        lw      a5,-20(s0)
        addi    a5,a5,-2
        mv      a0,a5
        call    Fibonacci
        mv      a5,a0
        add     a5,s1,a5
.L5:
        mv      a0,a5
        lw      ra,28(sp)
        lw      s0,24(sp)
        lw      s1,20(sp)
        addi    sp,sp,32
        jr      ra
```

```
Fibonacci:
        addi    sp,sp,-32
        sw      ra,28(sp)
        sw      s0,24(sp)
        sw      s1,20(sp)
        addi    s0,sp,32
        sw      a0,-20(s0)
        lw      a5,-20(s0)
        bnez    a5,.L4
        li      a5,0
        j       .L5
.L4:
        lw      a4,-20(s0)
        li      a5,1
        bne     a4,a5,.L6
        li      a5,1
        j       .L5
.L6:
        lw      a5,-20(s0)
        addi    a5,a5,-1
        mv      a0,a5
        call    Fibonacci
        mv      s1,a0
        lw      a5,-20(s0)
        addi    a5,a5,-2
        mv      a0,a5
        call    Fibonacci
        mv      a5,a0
        add     a5,s1,a5
.L5:
        mv      a0,a5
        lw      ra,28(sp)
        lw      s0,24(sp)
        lw      s1,20(sp)
        addi    sp,sp,32
        jr      ra
```

```
Fibonacci
        a
        s
        s
        s
        a
        s
        l
        b
        l
        j
.L4:
        l
        l
        b
        l
        j
.L6:
        l
        a
        m
        c
        m
        l
        a
        m
        c
        m
        a
.L5:
        m
        l
        l
        l
        a
        j
```

main

我有將每次算 F(n)的步數記下來，所以之後每次 call 到都可以直接加。

F(1) ⇒ +20 1

F(0) ⇒ +17 0

F(2) ⇒ +58 1

F(3) ⇒ +104 2

F(4) ⇒ +183 3

F(5) ⇒ +310 5

F(6) ⇒ +514 8

F(7) ⇒ 13

```
printResult:
        mv      t0, a0
        mv      t1, a1

        mv      a1, t0
        li      a0, 1
        ecall

        la      a1, str1
        li      a0, 4
        ecall

        mv      a1, t1
        li      a0, 1
        ecall

        ret
```

Experience :

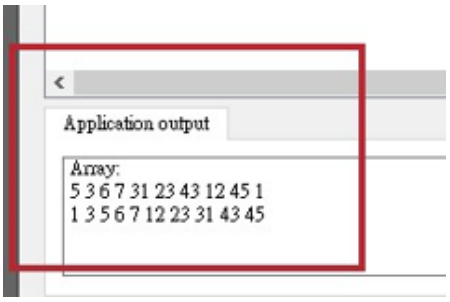
Some problems I met:

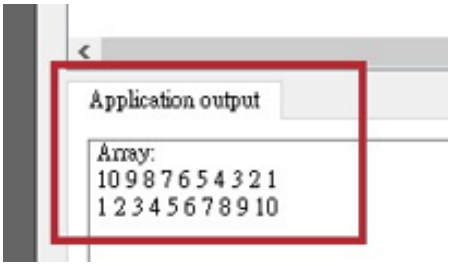
1. Not familiar with assembly code.
   This is my first time working on a HW with assembly code, so it was quite difficult to imagine how the code works. And I got stuck in logic statement every time I need to use it. For example, in my *bubble_sort* program, I needed to check if *j* is greater than or equal to 0, so the code should be like :
   slt t0, s0, zero (s0 is j, t0 is the result)
   When *j* is less than *0*, *t0* is *1*, so if *t0* == 1, I had to jump to another instruction; therefore the code should be like :
   bne t0, zero, exit (will jump to *exit* when *t0 != 0*)
   I wrote *beq* rather than *bne* at first, and it took me about an hour to find out that the program never does the swap operation.
2. Code converter doesn't perfectly match this Lab's environment.
   I tried to use *Compiler Explorer* to finish this Lab and I failed for the first time when writing *bubble_sort.s*, because the *for_loop* does not appear in the sample file, so I surfed on the web for the tutorial. The *global array* is not used in the sample file either, so I also spent some time looking for help online. The printResult part is different, too. I felt hopeless when bumping into errors at first because I couldn't find the information about the generated code.
   %hi(.LC1) (.LC1 is the string in the C code)
   Hence, I found out how to call *printf* function and finished the last 2 file(*gcd.s* and *fibonacci.s*) by modifying only those parts.
3. Need to compare lecture's ppt and the source found online.
   When iterating the array, I tried the code from Lecture's ppt but failed. Then I saw a post on *stack overflow* with the topic : *Venus RISC-V how to loop, compare, and print?* Then I successfully construct the code.
4. Cannot build the whole program on my own.
   I finished *bubble_sort.s* first, so I tried to write the *fibonacci.s* with *bubble_sort.s* and *factorial.s* as reference, but failed.(The program is executable but the result is wrong, 7th element is 28)
   So I turned to *Compiler Explorer* eventually.
5. Cannot declare same name for similar branches or function
   When implementing bubble sort, I use loop_sortArray for the nested loop and it reported error.

Some tips I discovered :

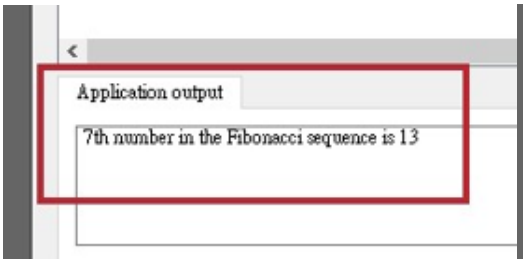
1. Syntax is not as scary as it looked.
   I had no idea why the syntax was so meaningless until I found a *cheatsheet* with some brief explanation.
   The instruction is in fact MEANINGFUL !
   blt : branch less than
   sw : store word
   lw : load word
   bne : branch not equal
   beq : branch equal
   slt : set less than
   jal : jump and link
   slli : shift left logical immediate
   la : load address...
   This is so important since it helps me debug faster.
2. Several possible ways of implementing branches(if..else)
   I learned 2 ways of implementing branches. When I want to compare two register's value, I can simply use *blt rs1, rs2, branch_name*.
   When I need to do logical compare, I have to use
   slt rd, rs1, rs2 and *bne(beq) rs1, rs2, branch_name*
   I think this will result in different instruction of a program.
3. Return is necessary, or Ripes will execute the program until keyboard interruption
   I'm not sure how much time did I spend until I found out this problem. At first, I add *ret* in my main function but in vail. Then I tried the method in sample file:
   li a1, 10 ecall. I put 0 rather than 10 and the program keeps running. I guess maybe the return function has this fixed syntax so 10 is unchangeable.

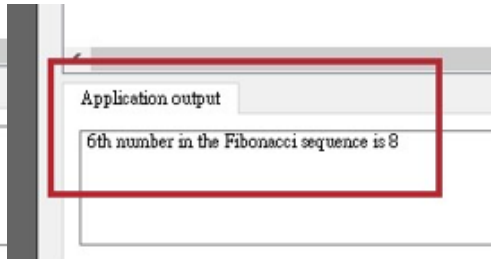
Bubble Sort :



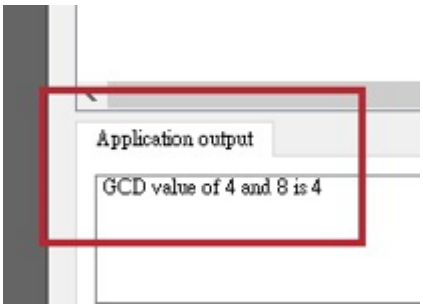


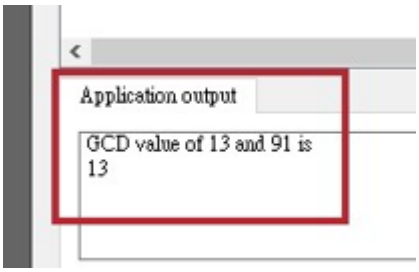
Fibonacci :





GCD :





Reference:

- https://web.eecs.utk.edu/~smarz1/courses/ece356/notes/assembly/
- https://hackmd.io/@xl86305955/CA_LAB1_R32I_Simulator
- https://stackoverflow.com/questions/59813759/how-to-use-an-array-in-risc-v-assembly
- http://csl.snu.ac.kr/courses/4190.307/2020-1/riscv-user-isa.pdf
- https://stackoverflow.com/questions/60430331/different-ways-to-traverse-arrays-in-risc-v
- https://stackoverflow.com/questions/60087133/venus-risc-v-how-to-loop-compare-and-print
- https://hackmd.io/@xl86305955/CA_LAB1_R32I_Simulator
- https://passlab.github.io/CSE564/notes/lecture03_ISA_Intro.pdf