Lab01: RISC-V Programming

109550164 徐聖哲

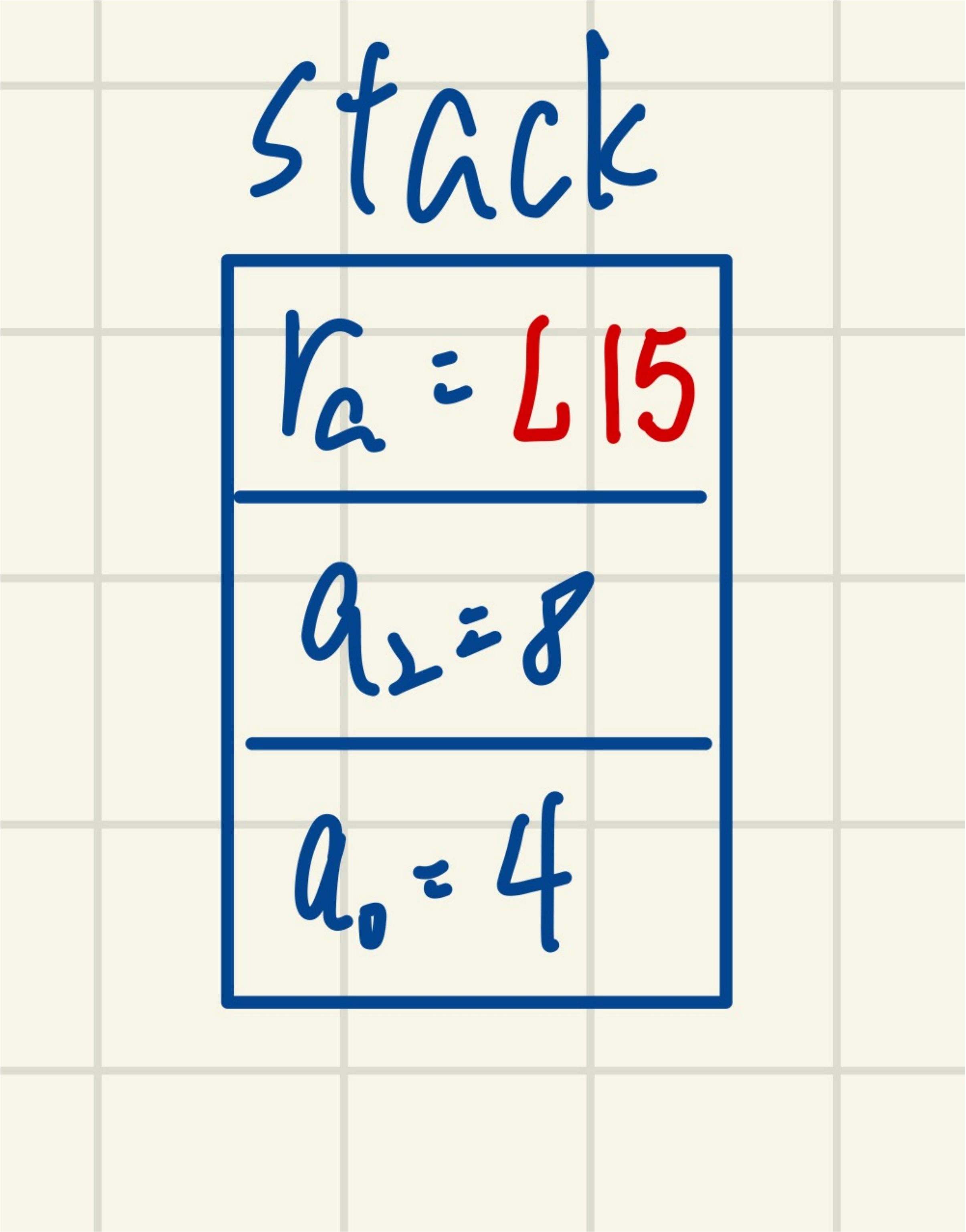
1. GCD

1. The total number of instruction in my gcd.s file is 46
   Beacause the value I use for testing is 4 and 8, so I only recursive one times
   the recursion happens from 5 to 18, as the line: beqz a0 exit iterate two times,
   then jump to 16 for return and pop stack

   For each function call, my program will count 9 instruction in each recurrsion
   When return each function call, my program will count 3 instruction each time
2. the maximum number of variable in hte stack is three, which is ilustrated on the picture below

```
main:
        lw   a2, argument2   # Load 8 from static data           # 1
        lw   a0, argument1    # Load 4 from static data          # 2
        lw   t0, 0                                               # 3
        jal  ra, gcd       # Jump-and-link to the 'gcd' label    # 4

        # Print the result to console
        mv   a1, a2                                              # 19
        lw   a0, argument1   # Load 8 from static data           # 20
        lw   a2, argument2    # Load 4 from static data          # 21
        jal  ra, printResult                                     # 22

        # Exit program
        li   a7, 10                                              # 45
        ecall                                                    # 46

gcd:
        beqz a0 exit                                             # 5  14
        addi sp, sp, -24                                         # 6
        sw   a0, 0(sp)                                           # 7
        sw   a2, 8(sp)                                           # 8
        sw   ra, 16(sp)                                          # 9
        lw   t1, 0(sp)                                           # 10
        rem  a0, a2, a0                                          # 11
        mv   a2, t1                                              # 12
        jal  ra gcd                                              # 13
        lw   ra, 16(sp)                                          # 16
        addi sp, sp, 24                                          # 17
        ret                                                      # 18
```

```
exit:   jr ra                                                    # 15

# --- printResult ---
# a1: GCD result
printResult:
        mv t0, a0                                                # 23
        mv t1, a1                                                # 24
        mv t2, a2                                                # 25

        la a0, str1    #print GCD value of 8                     # 26
        li a7, 4                                                 # 27
        ecall                                                    # 28
        mv a0, t0                                                # 29
        li a7, 1                                                 # 30
        ecall                                                    # 31

        la a0, str2    #print and  4                             # 32
        li a7, 4                                                 # 33
        ecall                                                    # 34
        mv a0, t2                                                # 35
        li a7, 1                                                 # 36
        ecall                                                    # 37

        la a0, str3    #print is answer                          # 38
        li a7, 4                                                 # 39
        ecall                                                    # 40
        mv a0, t1                                                # 41
        li a7, 1                                                 # 42
        ecall                                                    # 43

        ret                                                      # 44
```


stack
ra = L15
$a_1 = 8$
$a_0 = 4$

2. Bubble sort

1. The total instruction is 144
    - First, printarray and printResult both have 41 instructions
    - Second, if loop i which is sorted, it only has 9 instuctions
    - Third, if loop i isn’t sorted, loop j need to add 2 insturctions, and swap will also offer 4 instructions
    - Four, in this case, since a[0] and a[1] swap, so next loopj need 8 instruction before jump

    144 = 41(printarray) + 41(printResult) + 8(main) + 9 * 3(sorted) + 3(exit) + (10 + 2 + 4 + 8)(unsorted)

2. no need stack pointer, iI only use loop to implement bubble sort

```
main:
        la   s0, arr                                      # 1
        jal  ra, printarray                               # 2
        li   t0, 5           # n = 5                      # 45
        li   t1, 0           # i = 0                      # 46

        jal loopi                                         # 47

        exit: jal  ra, printResult                        # 100

        # Exit program
        li   a7, 10                                       # 143
        ecall                                             # 144

loopi:
        addi t1, t1, 1       # i++                        # 48  # 70  # 79  # 88  # 97
        blt  t1, t0, loopj   # if i < n, loopj            # 49  # 71  # 80  # 89  # 98
        j exit                                            # 99

loopj:
        addi t2, t1, -1      # j = i - 1                  # 50  # 64  # 72  # 81  # 90
        slli t3, t2, 2       # address of arr[j]          # 51  # 65  # 73  # 82  # 91
        add  s1, s0, t3                                   # 52  # 66  # 74  # 83  # 92
        lw   t5, 0(s1)       # t5 = arr[j]                # 53  # 67  # 75  # 84  # 93

        #slli t3, t2, 4       # address of arr[j+1]
        #add  s2, s0, t3
        lw   t6, 4(s1)       # t6 = arr[j+1]              # 54  # 68  # 76  # 85  # 94
        bltz t2, loopi       # j < 0, jump                # 55  # 69  # 77  # 86  # 95
        bge  t6, t5, loopi    # if arr[j] > arr[j+1]      # 56  # 78  # 87  # 96
        jal ra, swap                                      # 57
        addi t2, t2, -1      # j = j - 1                  # 62
        j loopj                                           # 63

swap:
        add t4, t5, x0       # temp = arr[j]              # 58
        sw t6, 0(s1)       # arr[j] = arr[j+1]            # 59
        sw t4, 4(s1)       # arr[j+1] = temp              # 60
        ret                                               # 61
#print unsorted array
printarray:
        la   a0, str1                                   # 3
        li   a7, 4                                      # 4
        ecall                                           # 5
        la   a0, endl                                   # 6
        li   a7, 4                                      # 7
        ecall                                           # 8
        lw   t0, 0(s0)                                  # 9
        mv   a0, t0                                     # 10
        li   a7, 1                                      # 11
        ecall                                           # 12
        la   a0, spac                                   # 13
        li   a7, 4                                      # 14
        ecall                                           # 15
        lw   t0, 4(s0)                                  # 16
        mv   a0, t0                                     # 17
        li   a7, 1                                      # 18
        ecall                                           # 19
        la   a0, spac                                   # 20
        li   a7, 4                                      # 21
        ecall                                           # 22
        lw   t0, 8(s0)                                  # 23
        mv   a0, t0                                     # 24
        li   a7, 1                                      # 25
        ecall                                           # 26
        la   a0, spac                                   # 27
        li   a7, 4                                      # 28
        ecall                                           # 29
        lw   t0, 12(s0)                                 # 30
        mv   a0, t0                                     # 31
        li   a7, 1                                      # 32
        ecall                                           # 33
        la   a0, spac                                   # 34
        li   a7, 4                                      # 35
        ecall                                           # 36
        lw   t0, 16(s0)                                 # 37
        mv   a0, t0                                     # 38
        li   a7, 1                                      # 39
        ecall                                           # 40
        la   a0, endl                                   # 41
        li   a7, 4                                      # 42
        ecall                                           # 43
        ret                                             # 44

#printresult
printResult:
        la   a0, str2                                    # 101
        li   a7, 4
        ecall
        la   a0, endl
        li   a7, 4
        ecall
        lw   t0, 0(s0)
        mv   a0, t0
        li   a7, 1
        ecall
        la   a0, spac
        li   a7, 4
        ecall
        lw   t0, 4(s0)
        mv   a0, t0
        li   a7, 1
        ecall
        la   a0, spac
        li   a7, 4
        ecall
        lw   t0, 8(s0)
        mv   a0, t0
        li   a7, 1
        ecall
        la   a0, spac
        li   a7, 4
        ecall
        lw   t0, 12(s0)
        mv   a0, t0
        li   a7, 1
        ecall
        la   a0, spac
        li   a7, 4
        ecall
        lw   t0, 16(s0)
        mv   a0, t0
        li   a7, 1
        ecall
        ret                                              # 142
```

3. Fibonacci sequence

1. The total instruction is 365, which is counted by store 1 to a register and ecall in the console
   The recurssion part of the instruction
   fib(0) : 2
   fib(1) : 2
   fib(2) = fib(1) + fib(0) : 6 + 2 + 2 + 4 +5 = 19
   fib(3) = fib(2) + fib(1) : 6 + 19 + 2 + 4 + 5 = 36
   fib(4) = fib(3) + fib(2) : 6 + 36 + 19 + 4 + 5 = 70
   fib(5) = fib(4) + fib(3) : 6 + 70 + 36 + 4 + 5 = 121
   fib(6) = fib(5) + fib(4) : 6 + 121+ 70+ 4 + 5 = 206
   fib(7) = fib(6) + fib(5) : 6 + 206+ 121 + 4 + 5 = 342

   other instruction include print is 23, thus total is 365

2. The maximum variable in my code is 17, because my code got to exit when n ≤ 1

```
fib:
        ble  a0, t1, exit
        addi sp, sp, -24
        sw   ra, 0(sp)
        sw   a0, 8(sp)

        addi a0, a0, -1
        jal, ra, fib
        sw   a0, 16(sp)
        lw   a0, 8(sp)

        addi a0, a0, -2
        jal  ra, fib
        lw   t0, 16(sp)

        #la a0, str4
        #li a7, 4
        #ecall
        #mv a0, t1
        #li a7, 1
        #ecall

        add  a0, a0, t0

        #la a0, str3
        #li a7, 4
        #ecall
        #mv a0, t1
        #li a7, 1
        #ecall
        lw   ra, 0(sp)
        addi sp, sp, 24
        ret

exit: ret
```

4. Experience

In lab1, riscv assembly code is really difficult. I spent three days finishing this lab, and encountered many problems.

First, I didn’t know how to control output in the console. It took me quite a while to figure out the meaning of the number after a7. Next, the way to control output string and word is different.

Second, sp (stack pointer) is also a big problem. I have to carefully set up ra (return address), or the recursion will crash. But, as soon as I understand the meaning of stack, it is quite easy to implement it in gcd.

Third, in bubble sort, the use of slli and array addresses also need to search for some resource. The trick of slli is quite awesome, it lets me easily access the arr[j] and arr[j+1]

Four. The assembly code is quite different from c, c++, html ... ,but it is fun to implement this homework after having a lot of effort.