Lab01 Report

109550053楊立嘉

Implementation

- summary table of all answer

	instruction count	max variable in stack
GCD	41	3
Fibonacci	363	18
Bubble sort	215	do not need to answer

- This implementation part is separated into three part : GCD, Fibonacci, and bubble sort. and each part also separated into two question that mentioned in Lab01_spec.pdf.

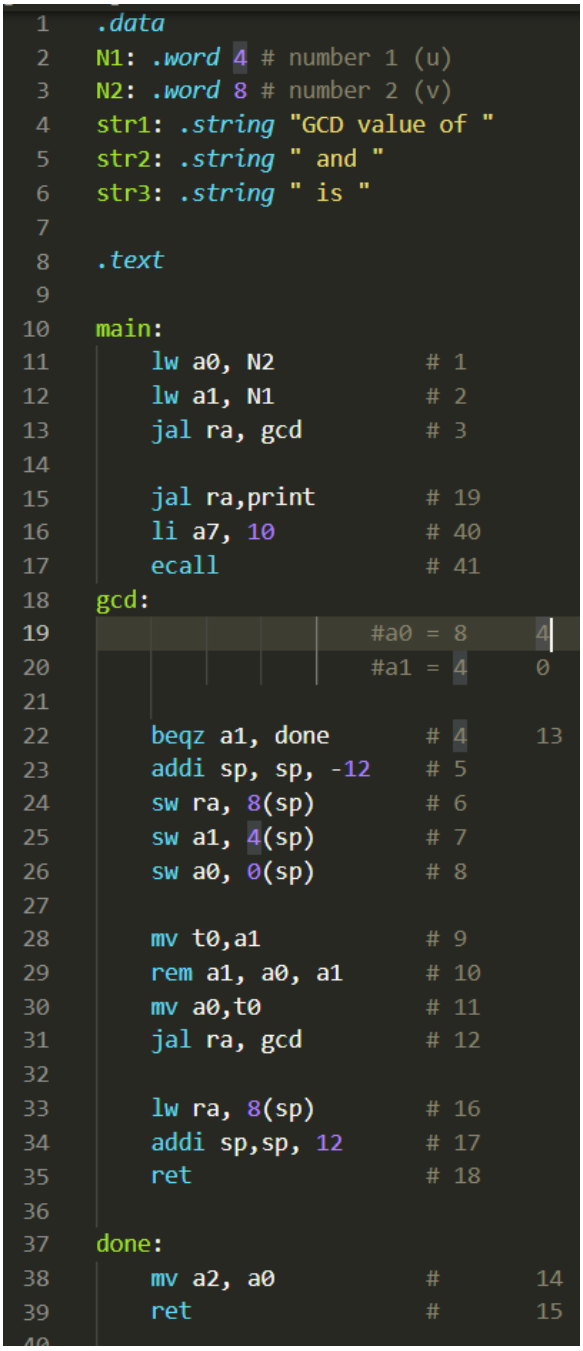
Part 1. GCD

Q1: How many instructions are actually executed?

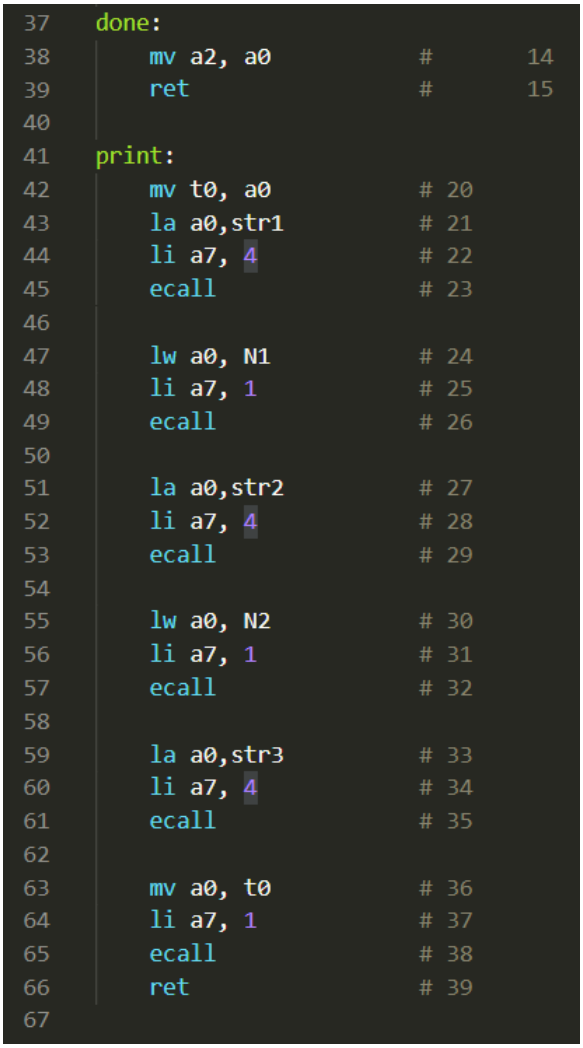
Because the two number `N1 = 4`, `N2 = 8` are small, instruction counts are small, so I use same method as TAs to show how I calculated the answer.

Note that numbers in line 19 means the value of `a0` when instructions at this column were executed. And numbers in line 20 have same idea. For example, instructions 4~18 were executed under the situation that `a0 = 8` and `a1 = 4`, and instructions 13~15 were executed under `a0 = 4` and `a1 = 0`.

The number of instructions needed is 41.



```
.data
N1: .word 4 # number 1 (u)
N2: .word 8 # number 2 (v)
str1: .string "GCD value of "
str2: .string " and "
str3: .string " is "

.text

main:
    lw a0, N2           # 1
    lw a1, N1           # 2
    jal ra, gcd         # 3

    jal ra,print        # 19
    li a7, 10           # 40
    ecall               # 41
gcd:
                        #a0 = 8     4
                        #a1 = 4     0

    beqz a1, done       # 4     13
    addi sp, sp, -12    # 5
    sw ra, 8(sp)        # 6
    sw a1, 4(sp)        # 7
    sw a0, 0(sp)        # 8

    mv t0,a1            # 9
    rem a1, a0, a1      # 10
    mv a0,t0            # 11
    jal ra, gcd         # 12

    lw ra, 8(sp)        # 16
    addi sp,sp, 12      # 17
    ret                 # 18

done:
    mv a2, a0           #       14
    ret                 #       15
```



```
done:
    mv a2, a0           #       14
    ret                 #       15

print:
    mv t0, a0           # 20
    la a0,str1          # 21
    li a7, 4            # 22
    ecall               # 23

    lw a0, N1           # 24
    li a7, 1            # 25
    ecall               # 26

    la a0,str2          # 27
    li a7, 4            # 28
    ecall               # 29

    lw a0, N2           # 30
    li a7, 1            # 31
    ecall               # 32

    la a0,str3          # 33
    li a7, 4            # 34
    ecall               # 35

    mv a0, t0           # 36
    li a7, 1            # 37
    ecall               # 38
    ret                 # 39

```

Q2: What is the maximum number of variable be pushed into the stack at the same time when your code execute?

For the given number `N1 = 4` and `N2 = 8`, the max number of variables in stack is 3, because three instructions 6,7,8. Where

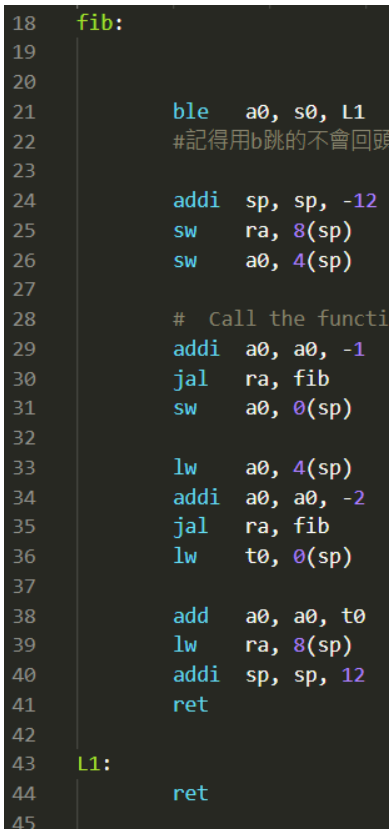
- 6th instruction store return address when `gcd` is called under `a0 = 8`, `a1 = 4`.
- line 25 store the first number for every `gcd` call. And 7th instruction store the first number `4` when `gcd` is called under `a0 = 8`, `a1 = 4`.
- line 26 store the second number for every `gcd` call. And 8th instruction store the first number `8` when `gcd` is called under `a0 = 8`, `a1 = 4`.
- note that I do not store the answer for every call in stack, because in GCD problem, we just need the answer of next layer ,but not next “two” layer like Fibonacci problem. So I do not need to save this answers, just cover the

previous answer when I have a new answer (even the answer will not change when go back to `N = i` from `N = i - 1`, in recursive concept, it is still a new answer for `N = i`). And because TAs said in hackmd that we need to minimize the answer, so I didn't open a stack space for saving answer of every `gcd` call, but store it just in `a2`.

Part 2. Fibonacci

Q1 : How many instructions are actually executed?

For Fibonacci, I calculated the answer 363 by hand, and following are calculating steps.



```
18  fib:
19
20
21          ble   a0, s0, L1
22          #記得用b跳的不會回頭
23
24          addi  sp, sp, -12
25          sw    ra, 8(sp)
26          sw    a0, 4(sp)
27
28          #  Call the functi
29          addi  a0, a0, -1
30          jal   ra, fib
31          sw    a0, 0(sp)
32
33          lw    a0, 4(sp)
34          addi  a0, a0, -2
35          jal   ra, fib
36          lw    t0, 0(sp)
37
38          add   a0, a0, t0
39          lw    ra, 8(sp)
40          addi  sp, sp, 12
41          ret
42
43  L1:
44          ret
45
```

1. Consider only `fib` and `L1` function only first.

   Note that `a0` is given number `N` and `s0` is set to be 1 for comparison with `N` between `main` label and `fib` label.

   1. Obviously, when `a0 = 0` and `a0 = 1` , we need 2 instructions, which are line 21 and line 44.
   2. When `a0=2`, we need $6 + 2 + 4 + 2 + 5 = 19$ instructions, where

- 6 means line 21 to 30
- 2 means the number needed before `jal` return, that is, the number needed when `a0=1`.
- 4 means line 31 to 35
- 2 means the number needed before `jal` return, that is, the number needed when `a0=0`.
- 5 means line 36 to 41

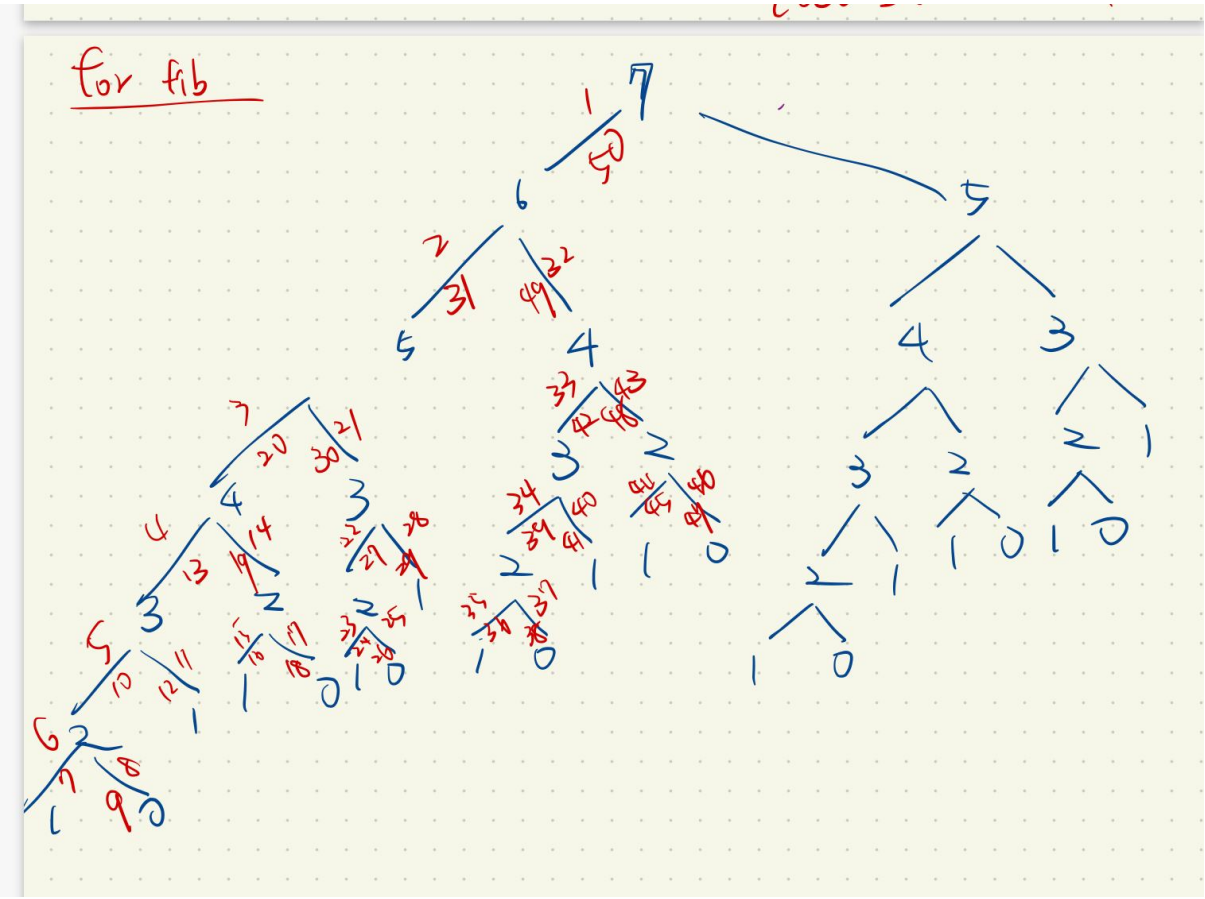
3. Follow same logic of above equation, we can get the following instruction counts table, and find that under the given number `a0 = 7`, we need 342 instructions at `fib` and `L1`.

n	fixed number = 6+4+5 = 15	jal in line 30 needed (when a0 = n-1)	jal in line 35 needed (when a0 = n-2)	total number needed when a0 = n
2	15	2	2	19
3	15	19	2	36
4	15	36	19	70
5	15	70	36	121
6	15	121	70	206
7	15	206	121	342

4. finally, I took the other instructions (`main, exit, and printResult`) into account, then I can got final answer is 363.

Q2 : What is the maximum number of variable be pushed into the stack at the same time when your code execute?

Use the same logic used to calculate instructions, I calculated the max number in stack by hand and use recursive method. Following are picture shows all fib call, return order and calculating steps.



1. When calling `a0 = 0` `fib` and `a0 = 1` `fib` , they branch to `L1` before using stack, so they have 0 variable in stack.

   So we get that 6th(+0) to 7th(-0) order and 8th(+0) to 9th(-0) order in the picture have nothing to do with the answer, written as $+0-0$ below.

2. When calling `a0 = 2` `fib`, we need 3 stack variables, which are line 25, 26 and 31.

   - line 25 store the current return address
   - line 26 store the current argument `N`
   - line 31 store the return value of `fib(N-1)`

   So we get for 5th to 10th order, the change of variables in stack is $+3+0-0+0-0-3$.

3. Similarly, When calling `a0 = 3` `fib`, the change of variables in stack is $+3(+3+0-0+0-0-3)(+0-0)-3$. Where

   - The first $+3$ and final -3 means the stack variables change ***in this layer***

- The $(+3+0-0+0-0-3)$ means the stack variables needed **when calling** `a0 = 2`
- The $(+0-0)$ means the stack variables needed **when calling** `a0 = 1`

4. So we can get the formula like this : the stack variables change when calling `a0 = i` is +3 (+ the stack variables change when calling `a0 = i-1`)(+the stack variables change when calling `a0 = i-2`.)

   So we can got the counts under different condition below :

   1. calling `a0 = 2` : $+3+0-0+0-0-3$
   2. calling `a0 = 3` : $+3(+3+0-0+0-0-3)(+0-0)-3$
   3. calling `a0 = 4` :

      $$+3\\(+3(+3+0-0+0-0-3)(+0-0)-3)\\(+3+0-0+0-0-3)\\-3$$

   4. calling `a0 = 5` :

      $$+3\\(+3(+3(+3+0-0+0-0-3)(+0-0)-3)(+3+0-0+0-0-3)-3)\\(+3(+3+0-0+0-0-3)(+0-0)-3)\\-3$$

   5. calling `a0 = 6` :

      $$+3\\(+3(+3(+3(+3+0-0+0-0-3)(+0-0)-3)(+3+0-0+0-0-3)-3)(+3(+3+0-0+0-0-3)(+0-0)-3)-3)\\(+3(+3(+3+0-0+0-0-3)(+0-0)-3)(+3+0-0+0-0-3)-3)\\-3$$

   6. calling `a0 = 7` :

      $$+3\\(+3(+3(+3(+3(+3+0-0+0-0-3)(+0-0)-3)(+3+0-0+0-0-3)-3)(+3(+3+0-0+0-0-3)(+0-0)-3)-3)(+3(+3(+3+0-0+0-0-3)(+0-0)-3)(+3+0-0+0-0-3)-3)-3)$$

$$(+3(+3(+3(+3+0-0+0-0-3)(+0-0)-3)(+3+0-0+0-0-3)-3)(+3(+3+0-0+0-0-3)(+0-0)-3)-3)-3$$

5. finally using the counts for `a0 = 7`, we can get the following stack variables change condition, and find that the maximum number is 18.

+3	3
(+3	6
(+3	9
(+3	12
(+3	15
(+3+0-0+0-0-3)	***18*** 15
(+0-0)-3)	12
(+3+0-0+0-0-3)	15 12
-3)	9
(+3	12
(+3+0-0+0-0-3)	15 12
(+0-0)-3)	9
-3)	6
(+3	9
(+3	12
(+3+0-0+0-0-3)	15 12
(+0-0)-3)	9
(+3+0-0+0-0-3)	12 9
-3)	6
-3)	3
\\ (+3	6
(+3	9
(+3	12
(+3+0-0+0-0-3)	15 12

(+0-0)-3)	9
(+3+0-0+0-0-3)	12 9
-3)	6
(+3	9
(+3+0-0+0-0-3)	12 9
(+0-0)-3)	6
-3)	3
\\ -3	0

Part 3. Bubble Sort

Q1: How many instructions are actually executed?

I choose size of array be 5, that is `5,3,6,7,31`. And I use same method as TAs to show how I calculated the answer.

Note that numbers in line 61, 97, 100, 105, 108, 116 are variables' value when the instructions at corresponding column were executed. Just like what I said in GCD part, for example, line 61 represents when line 62~77 were executed, when instructions 15~22 and 55~58 were executed , `t0` store `a[0]`. This lines just help myself to calculate answer, TAs can ignore it.

```
14  .text
15  main:
16      jal ra, printResult1            # 1
17
18      la  a7 data                     # 9
19      jal ra, printArray              # 10
20
21      la a7 data                      # 59
22      jal ra, bubblesort              # 60
23
24      jal ra, printResult2            # 156
25
26      la a7 data                      # 164
27      jal ra, printArray              # 165+49 = 214
28
29
30      # exit program, so set a7=10
31      li       a7, 10                 #214
32      ecall                           #215
33
34  printResult1:
35          la a0, str1                 # 2
36          li a7,4                     # 3
37          ecall                       # 4
38
39          la a0, newline              # 5
40          li a7, 4                    # 6
41          ecall                       # 7
42          ret                         # 8
```

```
44  printResult2:
45          la a0, str2                 # 157
46          li a7,4                     # 158
47          ecall                       # 159
48
49          la a0, newline              # 160
50          li a7, 4                    # 161
51          ecall                       # 162
52          ret                         # 163
53
54  printArray:
55      mv t0, a7                       # 11
56      lw t1, N                        # 12
57      slli t1, t1, 2                  # 13
58      add t1,t0,t1                    # 14
59
60      loop:
61                                      # t0 = a[0]   a[1]    a[2]    a[3]    a[4]
62          lw a0, 0(t0)                # 15    23      31      39      47
63          li a7 1                     # 16    24      32      40      48
64          ecall                       # 17    25      33      41      49
65
66          la a0, space                # 18    26      34      42      50
67          li a7 4                     # 19    27      35      43      51
68          ecall                       # 20    28      36      44      52
69
70          addi t0, t0, 4              # 21    29      37      45      53
71          bne t0,t1,loop              # 22    30      38      46      54
72
73          la a0, newline              # 55
74          li a7, 4                    # 56
75          ecall                       # 57
76
77          ret                         # 58
78
```

```
80  bubblesort:
81      mv a0,a7                        # 61
82      lw a1,N                         # 62
83
84      sort:
85      addi sp, sp, -20                # 63
86      sw ra, 16(sp)                   # 64
87      sw s3, 12(sp)                   # 65
88      sw s2, 8(sp)                    # 66
89      sw s1, 4(sp)                    # 67
90      sw s0, 0(sp)                    # 68
91
92      mv s2, a0                       # 69
93      mv s3, a1                       # 70
94
95      mv s0, zero                     # 71
96      for1tst:
97                                  # i = s0 = 0    1   1   2   3   4   5
98          slt t0, s0, s3              # 72    79      108 121 134 147
99
100     # t0 = 0 if s0 ≥ s3 (i ≥ n), t0 = 1    1   1   1   1   1   0
101         beq t0, zero, exit1         # 73    80      109 122 135 148
102
103         addi s1, s0, -1             # 74    81      110 123 136
104         for2tst:
105                                 # j = s1 = -1   0   -1  1   1   1
106             slti t0, s1, 0          # 75    82  104 111 124 137
107
108     # t0=1 if s1 < 0 (j < 0),     t0 = 1    0   1   0   0   0
109             bne t0, zero, exit2     # 76    83  105 112 125 138
110
111             slli t1, s1, 2          #       84      113 126 139
112             add t2, s2, t1          #       85      114 127 140
113             lw t3, 0(t2)            #       86      115 128 141
114             lw t4, 4(t2)            #       87      116 129 142
```

```
116   # t0=0 if t4 ≥ t3(v[j] <= v[j+1], 右大左小 regular)
117                       #             t0 =      1       0   0   0
118               slt t0, t4, t3          #       88      117 130 143
119               beq t0, zero, exit2     #       89      118 131 144
120
121               mv a7, s2               #       90
122               mv a0, s1               #       91
123               jal swap                #       92
124
125               addi s1, s1, -1  #j--   #       102
126
127               j for2tst               #       103
128           exit2:
129               addi s0, s0, 1          # 77        106 119 132 145
130           j for1tst                   # 78        107 120 133 146
131       exit1:
132       lw s0, 0(sp)                    #                           149
133       lw s1, 4(sp)                    #                           150
134       lw s2, 8(sp)                    #                           151
135       lw s3,12(sp)                    #                           152
136       lw ra,16(sp)                    #                           153
137       addi sp,sp, 20                  #                           154
138       ret                             #                           155
139
140   swap:
141
142       mv a2, a7                       #       93
143       mv a1, a0                       #       94
144
145       slli t1, a1, 2                  #       95
146       add t1, a2, t1                  #       96
147
148       lw t0, 0(t1)                    #       97
149       lw t2, 4(t1)                    #       98
150
151       sw t2, 0(t1)                    #       99
152       sw t0, 4(t1)                    #       100
153
154       ret                             #       101
```

Q2: What is the maximum number of variable be pushed into the stack at the same time when your code execute?

- We don’t need to answer this question. — by TA

Experience

I think the biggest problem is I am not familiar with risc-v, even things that different register have different ability, what is system service, and how to write instructions.....

Even though I had focused on class, I still need to search everything by myself.

Also, Ripes seems to be unpopular, when I write something wrong and google it, I find nothing. Ripes Version is a problem too, at the time I didn’t know what is system service, it is realy frustrated that even I just copied TAs’ example and it didn’t work.