Report

GCD

(a)共跑了 40 個指令

細部指令：

1. Main: 共跑了 9 個指令
2. GCD：共跑了 9 個指令
3. Print：共跑了 22 個指令

```
.text
main:
    lw s0, n1          1
    lw s1, n2          2
    jal ra, GCD        3

    #print
    mv a1,s0           13
    lw s0,n1           14
    lw s1,n2           15
    jal ra,print       16

    #Exit program
    li a0, 10          39
    ecall              40

GCD:
    mv s2, s1              4    8
    rem s1, s0, s1         5    9
    mv s0, s2              6    10
    bnez s1, GCD           7    11
    jalr x0, x1, 0              12
```

```
print:
    mv t0, s0           17
    mv t1, s1           18
    mv t2, a1           19

    #print
    la a1, str1         20
    li a0, 4            21
    ecall               22

    mv a1, t0           23
    li a0, 1            24
    ecall               25

    la a1, str2         26
    li a0, 4            27
    ecall               28

    mv a1, t1           29
    li a0, 1            30
    ecall               31

    la a1, str3         32
    li a0, 4            33
    ecall               34

    mv a1, t2           35
    li a0, 1            36
    ecall               37

    ret                 38
```

GCD-(b)

因為沒有用到 stack，所以沒有變數被放進 stack 裡面

GCD-edi2：此方法有用到 stack

(a)共跑了 58 個指令

細部指令：

1. Main: 共跑了 8 個指令

2. GCD：共跑了 28 個指令

甲、Gcd:

i. (0)初始設定到判斷是否完成(addi sp, sp, -8 到 bne a6, zero, loop) 4*3 = 12

ii. (1)結束 gcd(addi sp, sp, 8 到 jalr x0, x1, 0) 2*1 = 2

乙、Loop

i. (2)取餘數，回去 gcd(mv a2, a6 到 jal ra, gcd) 4*2 = 8

ii. (3)lw 到 ret 3*2 = 6

丙、執行順序： 0 2 0 2 1 3 3

丁、執行次數：12+2+8+6 = 28

3. Print：共跑了 22 個指令

(b)最多有 6 個變數會放在 stack 裡面

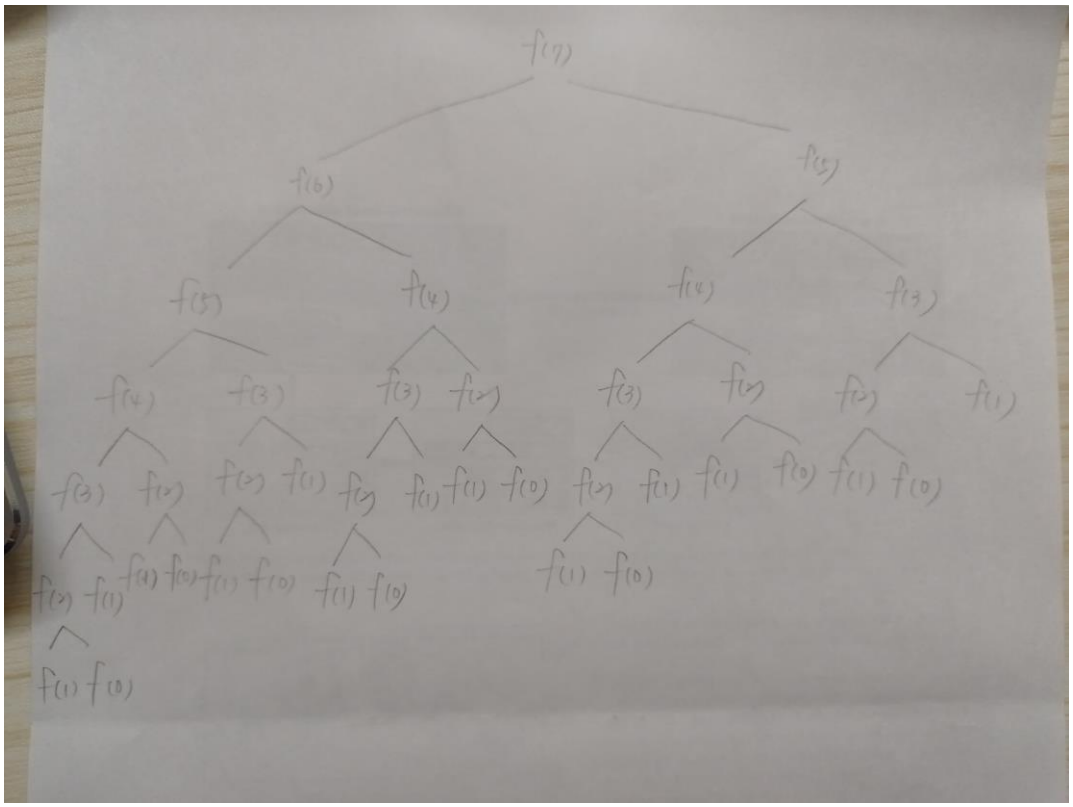
Fibonacci-(a)共 362 個指令，算的方法是透過宣告一個變數去計算，以下是細部指令：

1. Main：8 個指令
2. Print：12 個指令
3. Fibonacci 迴圈 342：

F(1)與 f(0)各執行兩條指令，判斷與 ret，剩下每個都會自己執行 15 條指令，因此遞迴結果 f(2) = 15+2+2 = 19，f(3) = 15+19+2 = 36，f(4) = 15+36+19 = 70，f(5) = 15+70+36 = 121，f(6) = 15+121+70 = 206，f(7) = 15+206+121 = 342

Fibonacci-(b)

最多會有 18 個變數存在 stack 中，因為每次開 3 個變數的空間，然後最深深度是 7，但是因為我的判斷式是寫在前面，所以最深的地方不會用到 stack，所以最多會有 18 個變數存在裡面



Bubble_sort-(a)

共 746 個指令，以程式計算得出

細部指令：

1. Main：14 個指令
2. Bubblesort：478 個指令

    甲、從 i = 0 開始， j = i – 1 ，進入 j 迴圈

    乙、拿到 j 與 j+1 的位置，並比較其值看是否進入 swap

    丙、Swap 有 15 個指令

    丁、不斷計算迴圈直到跳出 i 迴圈

3. printArray：每次呼叫共執行 127 個指令，因為呼叫兩次，共執行 254 個指令，另外，因為在 printArray 與 printArray_for 之間沒有中斷的指令因此會執行至 printArray_for，並在 printArray_for 裡呼叫 printArray_for_End

    甲、printArray：2 個指令

    乙、printArray_for：12 個指令跑十次，第十一次進去時只做一個比較然後跳到 printArray_for_End，共 121 個指令

    丙、printArray_for_End：4 個指令

Bubble_sort-(b)

一開始進入 sort 的部分便給變數一個空間，並等全部做完才釋放，另外，開了

3 個變數的空間給 swap 做交換，每次計算完 swap 之後 sp 就會回到原來的位置，因此最多會放 4 個變數在 stack 裡面

Experience：

1. 遇到的困難：

   甲、在 ripes 裡面如果有打錯的地方，右邊便會無法顯示 assembled code，需要慢慢將 bug 找出

   乙、一開始未注意到給定的記憶體位置，以為可以給變數，所以後來重新分配位置給變數

2. 心得：學到了有些參數的規定，例如要放 zero 或是 0，一開始有點不知道甚麼時候該放哪一種，另外，覺得關於記憶體配置對我來說還是十分具有挑戰，也希望在未來幾天向同學請益，另外 bubble_sort 的部分一開始也寫了一個自己的版本，發現記憶體無法像自己想的那樣讀取出來，希望能完善自己寫的版本。

3. 參考資料：https://reurl.cc/rakmar ，這是 risc-v 指令手冊