計組Lab1 report 0816080

GCD

- gcd.s

```
.data
arg1: .word 4
arg2: .word 8
str1: .string "GCD value of "
str2: .string " and "
str3: .string " is "

.text
main:
        lw      a0, arg1       #load from data,store arg1 in a0
        lw      a1, arg2       #arg2 in a1
        jal     ra, gcd       # Jump-and-link to the 'gcd' label

        # Print the result to console

        mv      a2, a0
        lw      a0, arg1
        lw      a1, arg2
        jal     ra, printResult

        # Exit program
        li      a0, 10
        ecall

gcd:
        addi    sp, sp, -16    #call stack, reserve four 4-bytes-register space in stack
        sw      ra, 0(sp)      #store return address
        #sw      a0, 0(sp)
        beq     a1, zero ,RT

        rem     t0, a0,a1      #store temporarily, t0=a0%a1
        mv      a0,a1
        mv      a1,t0

        jal     ra,gcd

        lw      ra,0(sp)
        addi    sp,sp,16
        ret

        RT:
        addi    sp, sp, 16
        jalr    ra
```

```
# expects:
# a1 a0: Value which gcd number was computed from
# a2: result
printResult:
        mv      t0, a0
        mv      t1, a1
        mv      t2, a2


        la      a1, str1
        li      a0, 4
        ecall

        mv      a1, t0
        li      a0, 1
        ecall

        la      a1, str2
        li      a0, 4
        ecall

        mv      a1, t1
        li      a0, 1
        ecall

        la      a1, str3
        li      a0, 4
        ecall

        mv      a1, t2
        li      a0, 1
        ecall


        ret
```

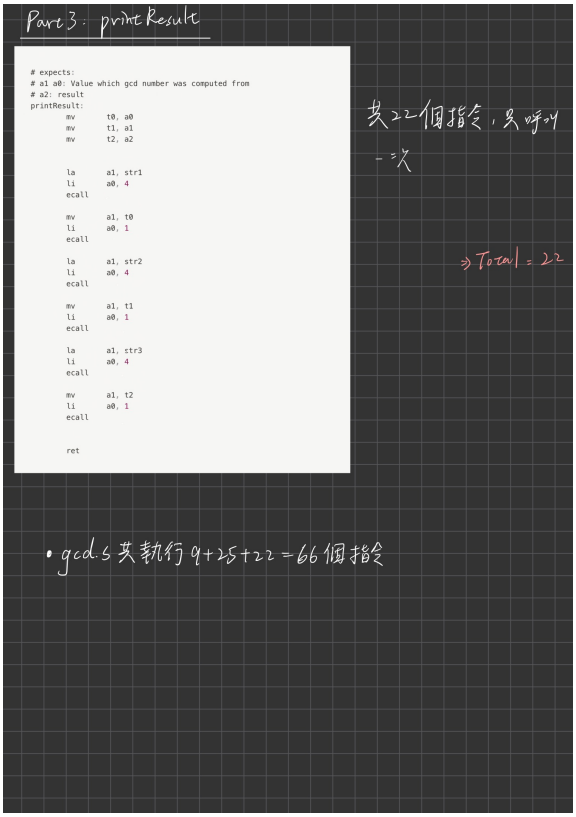
Q1 for GCD

How many instructions are actually executed? You have to explain clearly how you calculate your instructions. There is no specific answer.

Ans

66 instructions.

WHY



Q2 for GCD

What is the maximum number of variable be pushed into the stack at the same time when your code execute? There is only one correct answer.

Ans

3 variables.

不需要另外存變數，每次遞迴只需存return address，總共3次，所以是 3個

Fibonacci

- finbonacci.s

```
#int Fibonacci(int n) {
#   if(n==1)
#       return 1;
#   else if(n==0)
#       return 0;
#   else {
#       return(Fibonacci(n - 1) + Fibonacci(n - 2));
```

```
#   }
#}

.data
argument: .word 7 # Number to find the factorial value of
str1: .string " the number in the Fibonacci sequence is "


.text

#addi   ->變數和常數相加
#sw     ->register的東西 store 到memory
#lw     ->memory的東西 load 到register
main:
        lw      a0, argument   # Load argument from static data
        jal     ra, Fibonacci       # Jump-and-link to the 'Fibonacci' label

        # Print the result to console
        mv      a1, a0
        lw      a0, argument
        jal     ra, printResult

        # Exit program
        li      a0, 10
        ecall

Fibonacci:
        addi    sp, sp, -16    #call stack ,four 4-bytes register
        sw      ra, 8(sp)      #ra-> return address(function執行完要跳回下一行)
        sw      a0, 0(sp)      #a0->放function參數 save n
        beq     a0, zero,L1    #if(n == 0) return 0  ->jump to L1
        addi    t0, a0,-1
        beq     t0,zero,L2     #if(n==1) return 1    ->jump to L2

        #Fibonacci
        addi    a0, a0, -1     #set a0=(n-1)
        jal     ra, Fibonacci
        addi    sp, sp, -8
        sw      a0, 0(sp)

        lw      a0, 8(sp)       #set a0=n
        addi    a0, a0,-2       #set a0=(n-2)
        jal     ra, Fibonacci


        lw      t0, 0(sp)
        add     a0, a0, t0
        lw      ra, 16(sp)
        addi    sp, sp, 24

        ret

L1:
        addi    a0,zero,0
        addi    sp, sp, 16
        ret
L2:
        addi    a0,zero,1
```

```
        addi    sp, sp, 16
        ret

printResult:
        mv      t0, a0
        mv      t1, a1      #result

        mv      a1,t0
        li      a0,1
        ecall

        la      a1, str1
        li      a0, 4       #Print出a0的地址的字串
        ecall


        mv      a1, t1
        li      a0, 1       #Print a0數值
        ecall

        ret
```

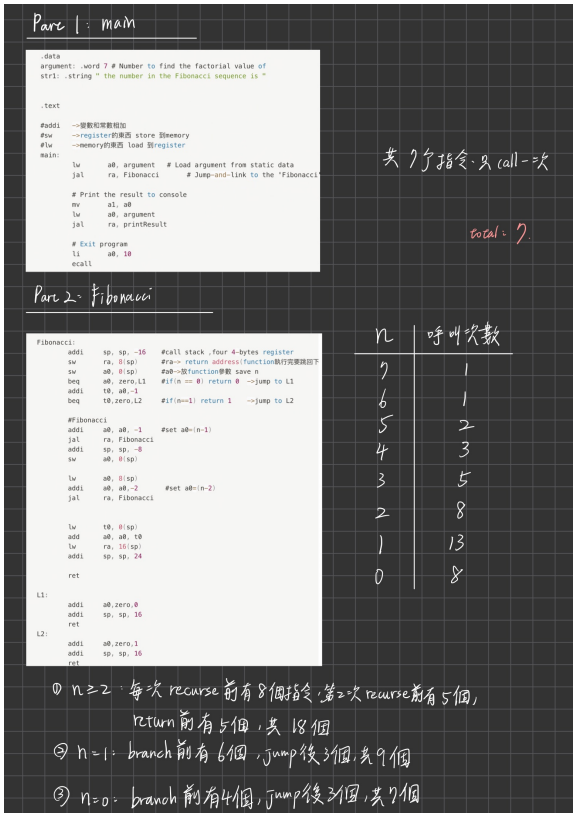
Q1 for Fibonacci

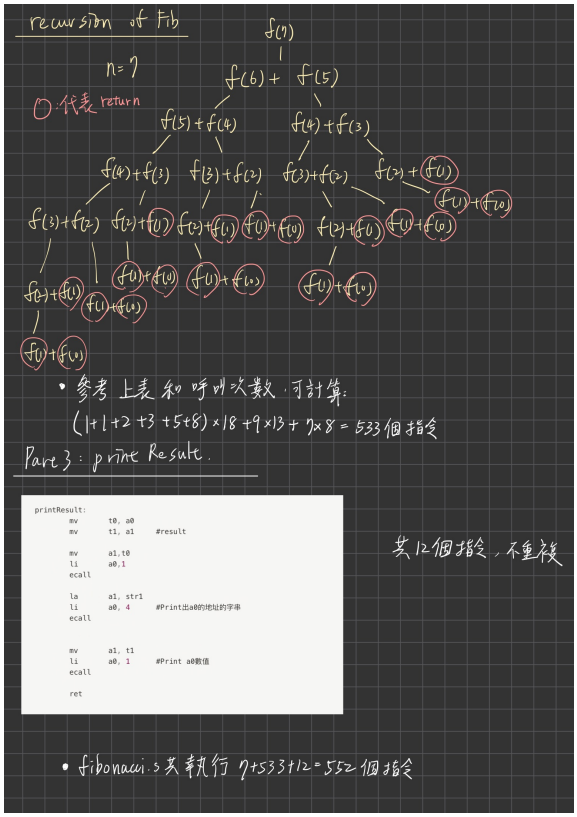
How many instructions are actually executed? You have to explain clearly how you calculate your instructions. There is no specific answer.

Ans

552 instructions.

WHY





Q2 for Fibonacci

What is the maximum number of variable be pushed into the stack at the same time when your code execute? There is only one correct answer.

Ans

15 variables.

每層遞迴都在stack裡存兩個變數，return address和傳入的參數，共七層遞迴(參考上面的`recursion`過程)，而在最後一層`f(1)+f(0)`的遞迴中還會先存`f(1)`的值再算`f(0)`，所以最多有15個變數

Bubble Sort

- bubble_sort.s

```
# This example shows an implementation of the mathematical
# factorial function (! function).

.data
n: .word 10 # Number to find the bubblesort value of
arr: .word 5,3,6,7,31,23,43,12,45,1
```

```
str1: .string "Array: "
str2: .string "Sorted: "
str3: .string " "
str4: .string "\n"

.text
main:
        la      a1, str1   # print the initial array
        li      a0, 4
        ecall

        la      a1,str4
        li      a0,4
        ecall

        jal     ra,printArray

        jal     ra,bubblesort

        la      a1,str2
        li      a0,4
        ecall

        jal     ra,printArray

        # Exit program
        li       a0, 10
        ecall

# t0:i
# t1:j
# t2:n
# t3:arr array
bubblesort:
        addi    sp, sp, -8    #call stack
        sw      ra, 0(sp)

        addi    t0, zero, 0   #i=0
        lw      t2,n          #n=10

outer_loop:
        #if i>=n, exit process
        bge     t0,t2,outer_end
        addi    t1,t0,-1      #j=i-1
inner_loop:
        #if j<0, exit inner loop
        blt     t1,zero,inner_end
        la      t3,arr     #load address
        #(arr+4)=arr[1] ,every integer is 4 byte
        slli    t6,t1,2     #shift left immediate-> t6=t1+(2*2)
        #j++ (4 byte= 1)
        add     t3,t3,t6    #arr +4
        lw      t4,0(t3)    #load arr[j]
        lw      t5,4(t3)    #load arr[j+1]

        bge     t5,t4,inner_end     #if arr[j+1]>=arr[j],no swap
        la      a0,arr
        mv      a1,t1       #save index in a1(i)
```

```
        jal     ra,swap
        addi    t1,t1,-1
        j       inner_loop
inner_end:
        # i = i + 1
      addi t0,t0,1
      j outer_loop
outer_end:
      # function complete
      lw ra,0(sp)
      addi sp,sp,8
      ret


swap:
        addi    sp,sp,-24
        #t0 t1 t2 will be uesd, have to save
        sw      t0,0(sp)
        sw      t1,8(sp)
        sw      t2,16(sp)
        #before call this function, it's already saved
        #index i in a1,arr address in a0
        #shift a1 to get i++
        slli    a1,a1,2
        add     t1,a0,a1
        #t1 now is array,swap ([i],[i+1])
        lw      t0,0(t1)
        lw      t2,4(t1)
        sw      t2,0(t1)
        sw      t0,4(t1)

        #load back
        lw t0,0(sp)
      lw t1,8(sp)
      lw t2,16(sp)
      addi sp,sp,24
      ret

# t0:i   t1:j    t2:arr
printArray:
        li t0,0
        lw t1,n

printArray_for:
        bge t0,t1,printArray_for_End

        la t2,arr
        slli t4,t0,2
        add t2,t2,t4

        lb a1,0(t2)
        li a0,1
        ecall

        la a1,str3
        li a0,4
        ecall
```

```
        # i++
        addi t0,t0,1
        j printArray_for

printArray_for_End:
        la a1,str4
        li a0,4
        ecall
        ret
```

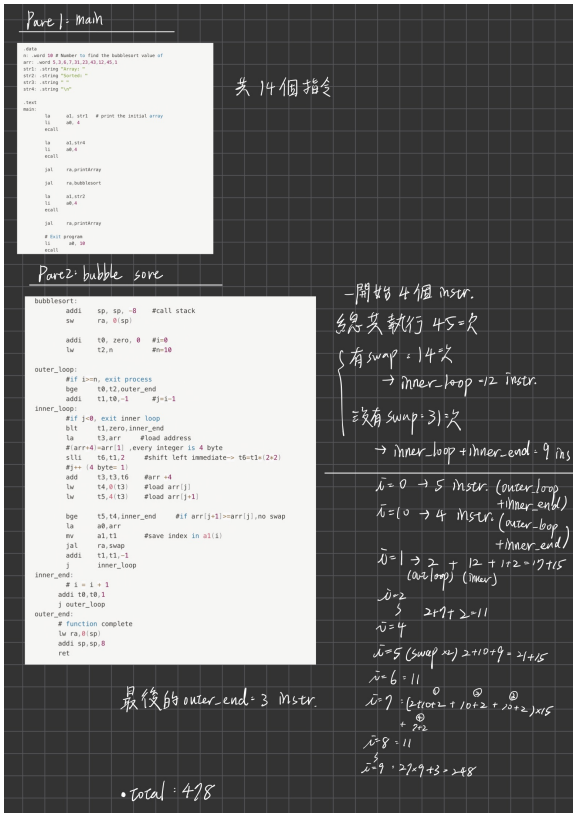
Q1 for Bubble Sort

How many instructions are actually executed? You have to explain clearly how you calculate your instructions. There is no specific answer.

Ans

746 instructions.

WHY



Q2 for Bubble Sort

What is the maximum number of variable be pushed into the stack at the same time when your code execute? There is only one correct answer.

Ans

4 variables.

`bubble sort`呼叫時會存一次address在stack裡，在有呼叫`swap`時會存三個變數，總共最多會存4個變數

Experience

我一開始想先用gcc跟線上轉譯器轉成assembly code，後來發現都比助教給的範例code還複雜，也不知道怎麼修改成ripes可編譯的code，於是決定自己翻譯。

我首先對照factorial.s和cpp檔讀懂組語大略的寫法後開始先寫finbonacci，但我一開始就遇到了瓶頸，我不知道從f(n-1)遞迴出來後怎麼存值而不改動到傳進f(n-2)的參數。後來也因為沒有善用stack的記憶體而導致無限迴圈。改了很久不知道怎麼辦就先去寫gcd，發現gcd遞迴比較少而且不用存參數，比較快寫出來。在gcd時我學到sw跟lw具體存取記憶體和參數的功用，後來回到fibonacci才知道可以先把n存起來，就可以在同個function做兩次遞迴。至於bubble sort是參考助教給的提示。

寫報告回答問題時我把遞迴過程寫出來我更明白遞迴的運作還有stack裡記憶體分配的方式。以前學遞迴的時候沒有這麼仔細的想過，現在感覺比較能徹底明白了。

整體來說，雖然翻譯組語花了很多時間，卻更知道store load branch的運作方式，如果不是自己寫的話我不會知道原來平常在寫C code 時會用到這麼多stack和暫存器，還有if else之類的都要分開寫，也清楚知道遞迴的邏輯了。