Computer Organization Lab 1 Report

109550121 温柏萱

Bubble Sort

```
.data
n: .word 4
arr:  .word 5, 3, 6, 7
arr_size:  .byte 16
str1: .string "Array "
str2: .string "Sorted "
_space: .string " "
newline: .string "\n"
.text
main:

    la a0, str1
    li a7, 4
    ecall
    jal ra, printArray

    jal ra, bubble_sort

    la a0, str2
    li a7, 4
    ecall

    jal ra, printArray
    li a7, 10
    ecall
bubble_sort:
    addi sp, sp, -16
    sw ra, 0(sp)
```

```
    la a0, arr
    lw a5, n
    mv t1, zero
    mv t2, a5
    # t1 = i, t2 = j
    outerloop:
        beq t1, a5, endLoop # i < n
        addi t2, t1, -1 # j = i - 1
        addi t1, t1, 1 # i++

        innerloop:
            blt t2, zero, outerloop # j >= 0

            slli t3, t2, 2
            add t4, a0, t3

            addi t2, t2, 1 # j++

            slli t5, t2, 2
            add t6, a0, t5

            addi t2, t2, -1 # j--
            # load a2 = arr[i], a3 = arr[i + 1]
            lw a2, 0(t4)
            lw a3, 0(t6)

            addi t2, t2, -1 # j--

            bge a3, a2, innerloop
            # swap
            sw a3, 0(t4)
            sw a2, 0(t6)

            jal s0, innerloop
    endLoop:
        lw ra, 0(sp)
        addi sp, sp, 16
        jr ra
printArray:
```

```
    lw t6, arr_size
    lb t3, arr_size
    la t1, arr
    addi sp, sp, -16
    sw ra, 0(sp)
  # let t6 be the counter of times and t0 be the printing index
    for:
        lw a0, 0(t1)
        li a7, 1
        ecall
        addi t2, t2, 4

        la a0, _space
        li a7, 4
        ecall
        addi t1, t1, 4
        addi t3, t3, -4
        bne t3, zero, for
  EndLoop:
        la a0, newline
        li a7, 4
        ecall

    lw ra, 0(sp)
    addi sp, sp, 16
    jr ra
```

1. How many instructions are actually executed?

208 instructions were executed.

MAIN	BUBBLE_SORT	PRINT_ARRAY
11	95	56

In the bubble_sort function, to be convenient, I use i and j to represent the registers t_1 and t_2 respectively. I first let $i = 0$ and $j = n$, the termainate condition of the entire function is when $i = n$. Since $5 > 3$, we swap the two, and thus it will cost 15 instructions. Since the data afterwards were all in ascending order, thus it was just loading the data and scanning over the array without swapping. The entire process ran for 6 times, 13 instructions every time, thus $6 \times 13 + 15 = 95$.

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

1 variable was pushed into the stack for the return address of the `printArray` function and was used again for the return address for the `bubble_sort` function.

3. Experience

This is the part that I struggled the most, since I had touble forming a loop. Knowing that a loop will keep going on and on until the terminate condition is satisified is completely different from actually hand-carving every operation. Thus just to print the array was a grand and long way, I even cired because I felt helpless and unable to learn the language fast enough as I am supposed to. Sometimes, I just don't understand why a line of code was there and what it was for. E.g. I know that lw $a_0,\ n$ is to load the variable n to register a_0, but why do I need to load before using it? Or what is the difference between `mv` and `lw` in this case. Although these questions may seem obvious to the TAs or some of my classmates, they are "not" anything near straightforward to me. And my friends thought I wasn't trying but simply whining, but truthfully, I witheld too little knowledge to complete the homework. They somehow had some knowledge about assembly code or computer organization before, that's why they couldn't understand the feeling of being drowned by helplessness and despair, feeling that I was so useless nor imagine my circumstance. I even thought about giving up, or just quit. But I think when TAs are giving us a task like this with a brand new language, it would be better and a lot more helpful to give more explanation of the sample code and why it was written so. Because I knew I wasn't the only one feeling useless and dizzy about assembly language. Some might say we learn from doing, but learning doesn't have to be painful. A lot of times during the homework, I wasn't even sure what I was doing, and I wasn't alone on feeling this. And I am afraid that the result of this might not be what the TAs and the professor wanted. So a little more detailed explanation on what we are learning will definitly do a great help. x

Greatest Common Divisor

```
.data
n1: .word 4
n2: .word 8
 str1: .string "GCD value of "
 str2: .string " and "
 str3: .string " is "

 .text
 main:
    lw a1, n1
    lw a2, n2
    jal ra, gcd
# Print the result to console
    lw a1, n1
    lw a2, n2
    jal ra, printResult
# Exit program
    li a7, 10
    ecall
gcd:
    addi sp, sp, -12
    sw ra, 8(sp)
    sw a1, 4(sp)
    sw a2, 0(sp)

    beq a2, zero, ngcd
    rem a3, a1, a2
    mv a1, a2
    mv a2, a3
    jal gcd

    lw a2, 0(sp)
    lw a1, 4(sp)
    lw ra, 8(sp)
    addi sp, sp, 12
    jr ra
```

```
ngcd:
    mv a3, a1
    addi sp, sp, 12
    jr ra
printResult:
    mv t0, a1
    mv t1, a2
    mv t2, a3

    la a0, str1
    li a7, 4
    ecall

    mv a0, t0
    li a7, 1
    ecall

    la a0, str2
    li a7, 4
    ecall

    mv a0, t1
    li a7, 1
    ecall

    la a0, str3
    li a7, 4
    ecall

    mv a0, t2
    li a7, 1
    ecall
```

1. How many instructions are actually executed?

The gcd function recursed for 3 times, the first time with the parameters $(4, 8)$, executing 9 instructions, the second time with the parameters $(8, 4)$, executing 8 instructions, and the last time with $(4, 0)$, executing 6 instructions. The return function of gcd took 5 instructions.

MAIN	GCD	PRINT
8	37	18

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

9 variables are pushed into the stack.

Suppose the parameters passed to gcd is a and b, then the stack has 3 layers, for the current answer, a, and b. The stack was called 3 times, thus $3 \times 3 = 9$.

3. Experience

Since I did this part after the Fibonacci Sequence, thus I think I was a tiny bit more familiar with the "feeling" of assembly code. And since my speed for learning a new programming language is incredibly slow, there were still loads of things that I had to consult my friends. Thus this is still a wobbling journey.

When I first executed on Ripes, it wouldn't show a single thing, then I realized it was the problem with the Mac m1 chip. All the applications designed especially for the x86 Mac cannot work without Rosetta, thus I then installed Rosetta.

What was the most interesting is that it wasn't entirely the chip's fault, cause I wrote the code wrong thus it was absolutely normal to not generate the assembly code on Ripes.

Then I thought about how to terminate the recursion, my friends told me that instead of jumping to a fucntion that does the recursion, I may as well build a fuction for termination. Thus I did, and I think it resulted in a better-looking function.

Fibonacci Sequence

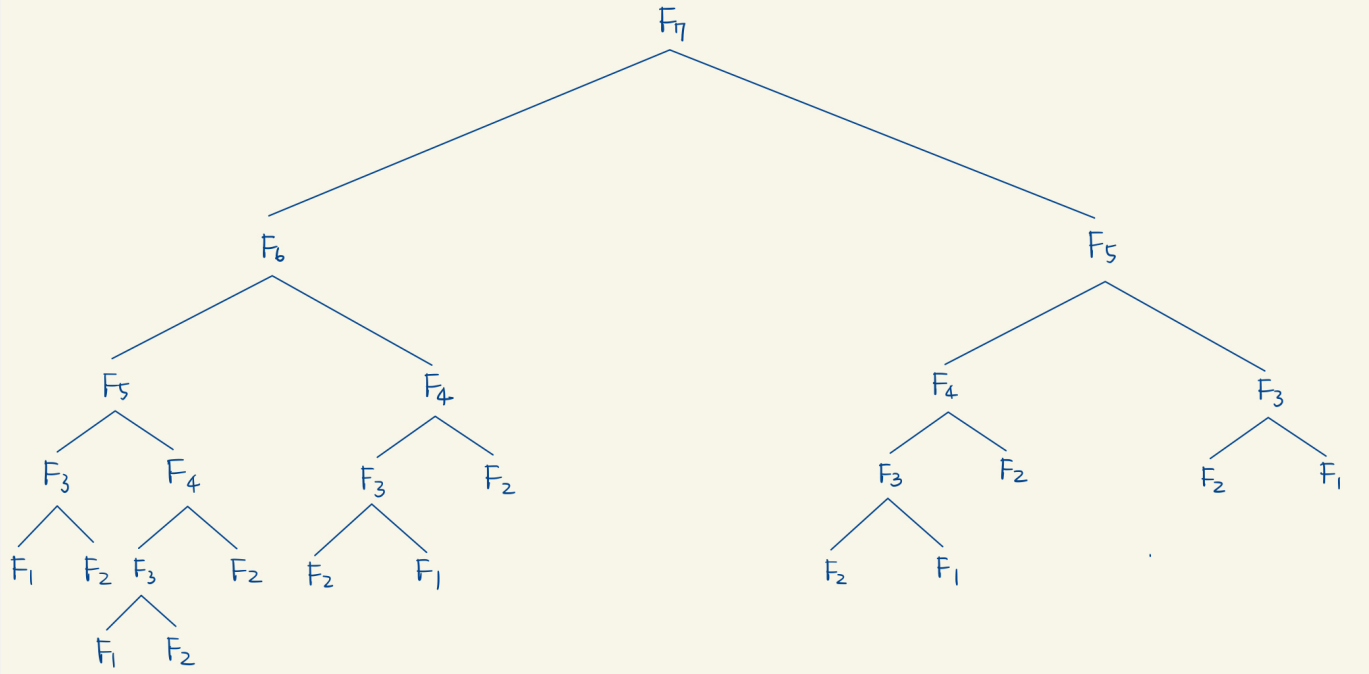
```
.data
argument: .word 7 # Number to find the factorial value of
str1: .string " the number in the Fibonacci sequence is "

.text
main:
        lw      a0, argument
        lw      a1, argument
        jal     ra, Fibonacci

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

        # Exit program
        li      a7, 10
        ecall
Fibonacci:
  mv t2, a1
  addi t2, t2, -2
  bge t2, zero, nFibonacci

  mv a0, a1
  jr ra
nFibonacci:
  addi sp, sp, -12
  sw ra, 4(sp)
  sw a1, 0(sp)

  addi a1, a1, -1
  jal Fibonacci

  lw a1, 0(sp)
  sw a0, 8(sp)

  addi a1, a1, -2
```

```
  jal Fibonacci

  lw t0, 8(sp)
  add a0, a0, t0

  lw ra, 4(sp)
  addi sp, sp, 12
  jr ra
printResult:
  mv t0, a0
  mv t1, a1

  mv a0, t0
  li a7, 1
  ecall

  la a0, str1
  li a7, 4
  ecall

  mv a0, t1
  li a7, 1
  ecall
  ret
```

1. How many instructions are actually executed?

465

The number of recursions in the Fibonacci function is illustrated below.



MAIN	FIBONACCI	PRINT
8	445	12

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

18 variables are pushed into the stack.

For each layer, the stack contains current value, F_{n-1}, and F_{n-2}. Since it was called 6 times, thus $6 \times 3 = 18$.

3. Experience

This is the first assembly code I've ever written in my life, thus I had trouble merely understanding the factorial code itself. I used to thought writing C code is difficult and mechanical, but now I've found an ever more machenical programming language, that is assembly language. This is totally resonable and logical, but I'm just whining for the lack of ability of self. Thankfully for the friends I have who helped me a lot and explained patiently how every line of the assebmly code worked so I could struggle but still finish the task.

At first, I kept forgetting what `jr` is for or how `ra` must be stored so the function could find its way back, but after examining some examples, I finally realized what it was about.

Alison Wen