Computer Organization lab 1

Information

- author : 0712238, Yan-Tong Lin, 林彥彤
- usage : Computer Organization Lab 1 for tf cheng's class
- due date : 2020/4/2
- updated date : 2020/3/26, 2020/4/2
- version control : https://github.com/EazyReal/Computer-Organization-2020 (https://github.com/EazyReal/Computer-Organization-2020)
- online version of this MD file
  - Computer Organization lab 1 (/ZLYhJj6BRgSIQz9daM9OZA)
- reference:
  - function call (https://zhu45.org/posts/2017/Jul/30/understanding-how-function-call-works/)
  - 2020 計算機組織 (/8LSdgoryQQy1Fnq214bC4A)

Description

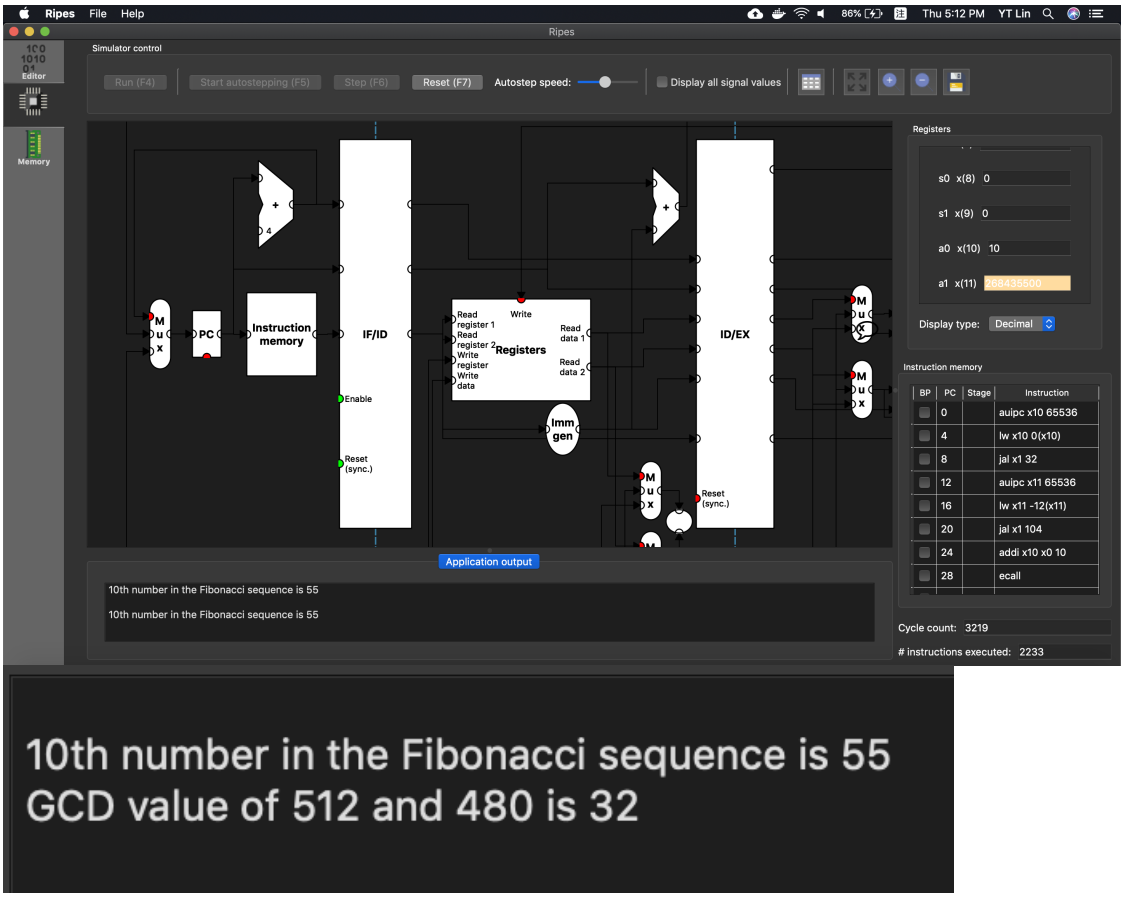
- translate C code into RISC-V Assembly
- the following is practiced:
  - function call
    - param passing, return value
    - when to use memory or register
  - array manipulation
    - how to manipulate array in assembly
    - pointer concept in assembly
  - register allocation
    - shold be clear about constant, mutable, pointer
- editted in VSCode
- executed with Ripes-continuous-mac-x86_64
  - both Windows and Mac environment are tested

Problems Encoutered

- gcd when use swap function without modifying return address
  - I did not understand the meaning of ra and jal quite well
  - after checking definition of ra and jal, the problem was resolved.

Fibbonaci

Execution Result



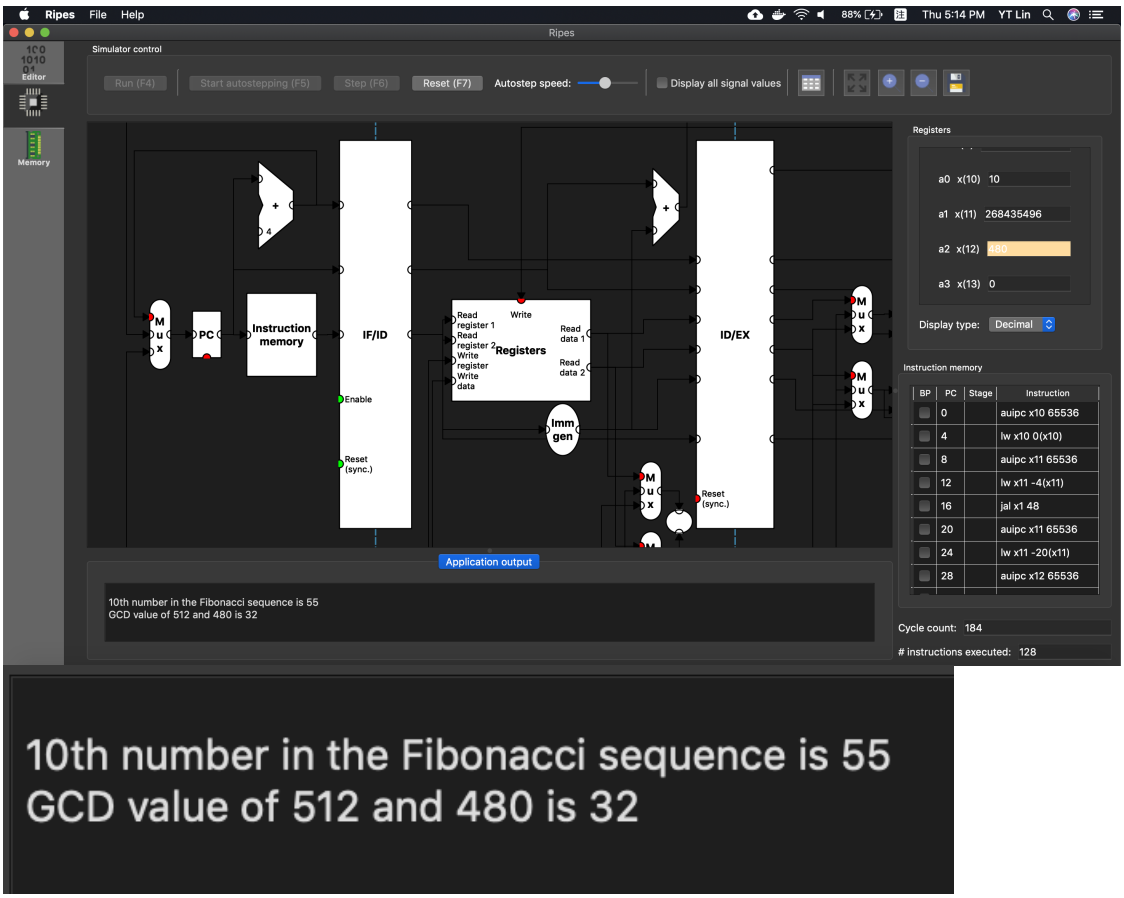
Code

```
# author : Yan-Tong Lin
# all right reserves
# no plagiarism

# usage : Fibonacci(arg), print result

.data
#the arg-th Fibonacci is to be found
arg: .word 10
str: .string "th number in the Fibonacci sequence is "
endl: .string "\n"

.text
main:
        #call Fibonacci(arg), param in a0
        lw       a0, arg
        jal      ra, Fibonacci_base
        #return values of Fibonacci_base/Fibonacci_recursive are both a0

        # Print the result to console
        lw       a1, arg
        jal      ra, printResult

        # Exit program
        li       a0, 10
        ecall

Fibonacci_base:
        addi     sp, sp, -32
        sw       ra, 16(sp)
        sw       a0, 0(sp)
        #a0 should be saved cause f is called twice dependent to a0 in one
        addi     t0, a0, -2
        #if t0 = ao-2 >= 0, go recursive else is base case
        bge      t0, zero, Fibonacci_recursive

        #mv       a0, a0
        #return arg is arg is 0 or 1
        #no need to change a0

        #end functioncall
        addi     sp, sp, 32
        jalr     x0, x1, 0


Fibonacci_recursive:
        #call Fibonacci(a0-1) and save to sp+8
        addi     a0, a0, -1
        jal      ra, Fibonacci_base
        sw       a0, 8(sp)
        #save result, save in register could be overwrite by recursibe call

        #call Fibonacci(a0-1), return val is now in a0
        lw       a0, 0(sp)
        addi     a0, a0 ,-2
        jal      ra, Fibonacci_base

        #add together
        lw       t1, 8(sp)
        add      a0, a0, t1

        #end functioncall
        #mv      a0, a0 #return value is a0 already
        lw       ra, 16(sp)
        addi     sp, sp, 32
        ret


```

```
# expects:
# a1: the ao-th Fibonacci number
# a0: Result
printResult:
        mv      t0, a1
        mv      t1, a0

        mv      a1, t0
        li      a0, 1
        ecall

        la      a1, str
        li      a0, 4
        ecall

        mv      a1, t1
        li      a0, 1
        ecall

        la      a1, endl
        li      a0, 4
        ecall

        ret

```

GCD

Execution Result



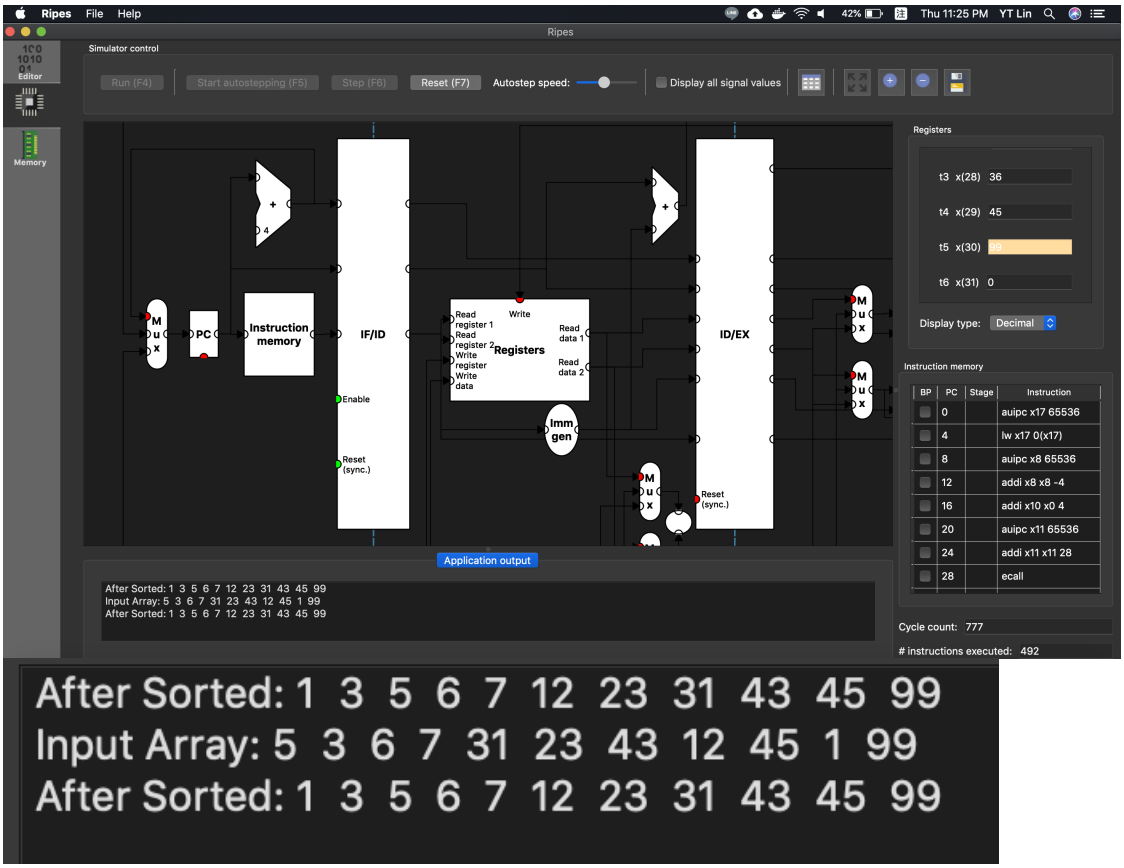
Code

```
# author : Yan-Tong Lin
# all right reserves
# no plagiarism

# usage : gcd(arg1, arg2), print result

.data

#define N1 512
#define N2 480
arg1: .word 512 # Number to find the factorial value of
arg2: .word 480
#define str1 "GCD value of "
#define str2 " and "
#define str3 " is "
#define endl "\n"
str1: .string "GCD value of "
str2: .string " and "
str3: .string " is "
endl: .string "\n"

.text
main:
        lw       a0, arg1   # Load argument from static data
        lw       a1, arg2
        jal      ra, gcd_base
        # Jump-and-link to the 'gcd_base' label

        # Print the result to console
        # mv       a0, a0
        lw       a1, arg1
        lw       a2, arg2
        jal      ra, printResult

        # Exit Main program
        li       a0, 10
        ecall

gcd_base:
        addi     sp, sp, -32
        sw       ra, 16(sp) #save return address
        #sw       a0, 0(sp) #save args
        #sw       a1, 8(sp)
        addi     t0, a1, -1
        bge      t0, zero, gcd_recursive #branch if n - 1 >= 0

        #here n(a1) <= 0
        #lw       a0, 0(sp)  #return value(a0) = parameter(a0), no need to
        #lw       ra, 16(sp) #base case can return without loading back ra
        addi     sp, sp, 32
        jalr     x0, ra, 0

gcd_recursive:
        # now is in else
        # calc module => change params => recurse
        # no need swap

        # loop: = module computation
        loop:
                blt     a0, a1, done # a1>=a0 break
                sub     a0, a0, a1   # a0-=a1
                j       loop
        done:
                #now a0 = r(remainder of m%n)
                #now a1 = n(n)
                jal      ra, swap
                #now a0 = n
                #now a1 = r
```

```
                jal      ra, gcd_base

                #return directly (now a0 should be result)
                lw       ra, 16(sp)
                addi     sp, sp, 32
                ret

### need to use jal(to change ra) or ret to main(bug)
swap:
        mv t0, a0
        mv a0, a1
        mv a1, t0
        ret #was bug without jal, return to main, debug by step-by-step ex


# expects:
# a0: gcd(arg1, arg2)
# a1: arg1
# a2: arg2
printResult:
        mv       t0, a0
        mv       t1, a1
        mv       t2, a2

        # a1 = param, a0 = print type
        la       a1, str1
        li       a0, 4
        ecall

        mv       a1, t1
        li       a0, 1
        ecall

        la       a1, str2
        li       a0, 4
        ecall

        mv       a1, t2
        li       a0, 1
        ecall

        la       a1, str3
        li       a0, 4
        ecall

        mv       a1, t0
        li       a0, 1
        ecall

        la       a1, endl
        li       a0, 4
        ecall

        ret
```

BubbleSort

Execution Result


Ripes File Help
Simulator control
Run (F4)
Start autostepping (F5)
Step (F6)
Reset (F7)
Autostep speed:
Display all signal values
Editor
Memory
Instruction memory
PC
IF/ID
Registers
Imm gen
ID/EX
Registers
t3 x(28) 36
t4 x(29) 45
t5 x(30) 99
t6 x(31) 0
Display type: Decimal
Instruction memory
BP PC Stage Instruction
0 auipc x17 65536
4 lw x17 0(x17)
8 auipc x8 65536
12 addi x8 x8 -4
16 addi x10 x0 4
20 auipc x11 65536
24 addi x11 x11 28
28 ecall
Application output
After Sorted: 1 3 5 6 7 12 23 31 43 45 99
Input Array: 5 3 6 7 31 23 43 12 45 1 99
After Sorted: 1 3 5 6 7 12 23 31 43 45 99
Cycle count: 777
instructions executed: 492

After Sorted: 1 3 5 6 7 12 23 31 43 45 99
Input Array: 5 3 6 7 31 23 43 12 45 1 99
After Sorted: 1 3 5 6 7 12 23 31 43 45 99

Code

```
# author : Yan-Tong Lin
# all right reserves
# no plagiarism

# usage :
# arr = the array that want to sort
# N, size of arr


.data
N:     .word 11
arr:   .word 5,3,6,7,31,23,43,12,45,1,99
msg1:  .string "Input Array: "
msg2:  .string "After Sorted: "
space: .string "  "
endl:  .string "\n"

# a7 = N(constant in this program), s1 = i
# s0 = arr start pointer(const)
# s1 = arr pointer
# t1 = i
# t2 = j
# t3 = delta with t2(t2 << 2)
# t4 = arr[j]
# t5 = arr[j+1]

.text
main:
       # a7 = N, constant in this program
       lw      a7, N
       la      s0, arr

       # print msg1
       li      a0, 4
       la      a1, msg1
       ecall

       # load array to s0
       # i = 0
       # print array with loop
       la      s1, arr
       li      t1, 0
       jal     ra, print_loop

       # print endl
       li      a0, 4
       la      a1, endl
       ecall

       # now use t1 as i, init with 0
       # about to start sort
       # t1 = i, s0 = arr start, constant when sorting
       li      t1, 0
       la      s0, arr

#Bubble Sort Starts Here

fori:
       # if i == N, break
       beq     t1, a7, end_of_sort
       # j = i - 1
       addi    t2, t1, -1

forj:
       #if j < 0, break
       blt    t2, zero, to_nxt_i

       # t3 = j << 2
```

```
        # 4 byte per word, t3 is the actual distance from arr start(t4)
        slli    t3, t2, 2
        # t4 is now arr[j]
        add     s1, s0, t3

        # load arr[j] and arr[j+1]
        lw      t4, 0(s1)
        lw      t5, 4(s1)

        # if arr[j] <= arr[j+1](i.e., is sorted before)
        # break
        ble     t4, t5 , to_nxt_i

        # else swap arr[j] and arr[j+1](by direct saving), continue
        sw      t4, 4(s1)
        sw      t5, 0(s1)

        # j--, continue
        addi    t2, t2, -1
        j       forj

#end of the inner loop
to_nxt_i:
        #i++
        addi    t1, t1, 1
        j       fori


#End of the outer loop
end_of_sort:
        # print msg2
        li      a0, 4
        la      a1, msg2
        ecall

        # i = 0
        # print sorted array with loop
        li      t1, 0
        la      s1, arr
        jal     ra, print_loop

        # endl
        li      a0, 4
        la      a1, endl
        ecall

        #Exit program
        li      a0, 10
        ecall


# expectations:
# s1 is the array pointer now
# t1 is i
print_loop:
        lw      a1, 0(s1)
        li      a0, 1
        ecall

        # print space
        la      a1, space
        li      a0, 4
        ecall

        # i(t1)++
        # arr ptr(s1) +=4
        addi    s1, s1, 4
        addi    t1 ,t1, 1

```

```
        blt     t1, a7, print_loop
        ret
```

Development ScreenShots