

Computer Organization lab 1

- Computer Organization lab 1
 - Information
 - Description
 - Problems Encoutered
 - Fibbonaci
 - Execution Result
 - Code
 - GCD
 - Execution Result
 - Code
 - BubbleSort
 - Execution Result
 - Code
 - Development ScreenShots

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](https://hackmd.io/ZLYhJj6BRgSIQz9daM9OZA)).
- reference:
 - function call (<https://zhu45.org/posts/2017/Jul/30/understanding-how-function-call-works/>).
 - 2020 計算機組織 ([/8LSdgoryQQy1Fng214bC4A](https://hackmd.io/8LSdgoryQQy1Fng214bC4A)).



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
- edited 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

The screenshot shows the Ripes simulator interface. The main window displays the processor architecture with stages IF/ID, ID/EX, and EX/WB. The registers panel shows the following values:

Register	Value
s0 x(8)	0
s1 x(9)	0
a0 x(10)	10
a1 x(11)	55

The instruction memory panel shows the following instructions:

BP	PC	Stage	Instruction
0			auipc x10 65536
4			lw x10 0(x10)
8			jal x1 32
12			auipc x11 65536
16			lw x11 -12(x11)
20			jal x1 104
24			addi x10 x0 10
28			ecall

The application output window displays the result: "10th number in the Fibonacci sequence is 55".

Code

```

1  # author : Yan-Tong Lin
2  # all right reserves
3  # no plagiarism
4
5  # usage : Fibonacci(arg), print result
6
7  .data
8  #the arg-th Fibonacci is to be found
9  arg: .word 10
10 str: .string "th number in the Fibonacci sequence is "
11 endl: .string "\n"
12
13 .text
14 main:
15     #call Fibonacci(arg), param in a0
16     lw      a0, arg
17     jal     ra, Fibonacci_base
18     #return values of Fibonacci_base/Fibonacci_recursive are both a0
19
20     # Print the result to console
21     lw      a1, arg
22     jal     ra, printResult
23
24     # Exit program
25     li      a0, 10
26     ecall
27
28 Fibonacci_base:
29     addi    sp, sp, -32
30     sw      ra, 16(sp)
31     sw      a0, 0(sp)
32     #a0 should be saved cause f is called twice dependent to a0 in one
33     addi    t0, a0, -2
34     #if t0 = a0-2 >= 0, go recursive else is base case
35     bge     t0, zero, Fibonacci_recursive
36
37     #mv      a0, a0
38     #return arg is arg is 0 or 1
39     #no need to change a0
40
41     #end functioncall
42     addi    sp, sp, 32
43     jalr    x0, x1, 0
44
45
46 Fibonacci_recursive:
47     #call Fibonacci(a0-1) and save to sp+8
48     addi    a0, a0, -1
49     jal     ra, Fibonacci_base
50     sw      a0, 8(sp)
51     #save result, save in register could be overwrite by recursive call
52
53     #call Fibonacci(a0-1), return val is now in a0
54     lw      a0, 0(sp)
55     addi    a0, a0, -2
56     jal     ra, Fibonacci_base
57
58     #add together
59     lw      t1, 8(sp)
60     add     a0, a0, t1
61
62     #end functioncall
63     #mv      a0, a0 #return value is a0 already
64     lw      ra, 16(sp)
65     addi    sp, sp, 32
66     ret
67
68

```

```

69 # expects:
70 # a1: the ao-th Fibonacci number
71 # a0: Result
72 printResult:
73     mv     t0, a1
74     mv     t1, a0
75
76     mv     a1, t0
77     li     a0, 1
78     ecall
79
80     la     a1, str
81     li     a0, 4
82     ecall
83
84     mv     a1, t1
85     li     a0, 1
86     ecall
87
88     la     a1, endl
89     li     a0, 4
90     ecall
91
92     ret
93

```

GCD

Execution Result

10th number in the Fibonacci sequence is 55
GCD value of 512 and 480 is 32

Code

```

1  # author : Yan-Tong Lin
2  # all right reserves
3  # no plagiarism
4
5  # usage : gcd(arg1, arg2), print result
6
7  .data
8
9  #define N1 512
10 #define N2 480
11 arg1: .word 512 # Number to find the factorial value of
12 arg2: .word 480
13 #define str1 "GCD value of "
14 #define str2 " and "
15 #define str3 " is "
16 #define endl "\n"
17 str1: .string "GCD value of "
18 str2: .string " and "
19 str3: .string " is "
20 endl: .string "\n"
21
22 .text
23 main:
24     lw      a0, arg1    # Load argument from static data
25     lw      a1, arg2
26     jal     ra, gcd_base
27     # Jump-and-link to the 'gcd_base' label
28
29     # Print the result to console
30     # mv     a0, a0
31     lw      a1, arg1
32     lw      a2, arg2
33     jal     ra, printResult
34
35     # Exit Main program
36     li      a0, 10
37     ecall
38
39 gcd_base:
40     addi    sp, sp, -32
41     sw      ra, 16(sp) #save return address
42     #sw     a0, 0(sp) #save args
43     #sw     a1, 8(sp)
44     addi    t0, a1, -1
45     bge     t0, zero, gcd_recursive #branch if n - 1 >= 0
46
47     #here n(a1) <= 0
48     #lw     a0, 0(sp) #return value(a0) = parameter(a0), no need to
49     #lw     ra, 16(sp) #base case can return without loading back ra
50     addi    sp, sp, 32
51     jalr    x0, ra, 0
52
53 gcd_recursive:
54     # now is in else
55     # calc module => change params => recurse
56     # no need swap
57
58     # loop: = module computation
59     loop:
60         blt     a0, a1, done # a1>=a0 break
61         sub     a0, a0, a1    # a0-=a1
62         j       loop
63     done:
64         #now a0 = r(remainder of m%n)
65         #now a1 = n(n)
66         jal     ra, swap
67         #now a0 = n
68         #now a1 = r

```

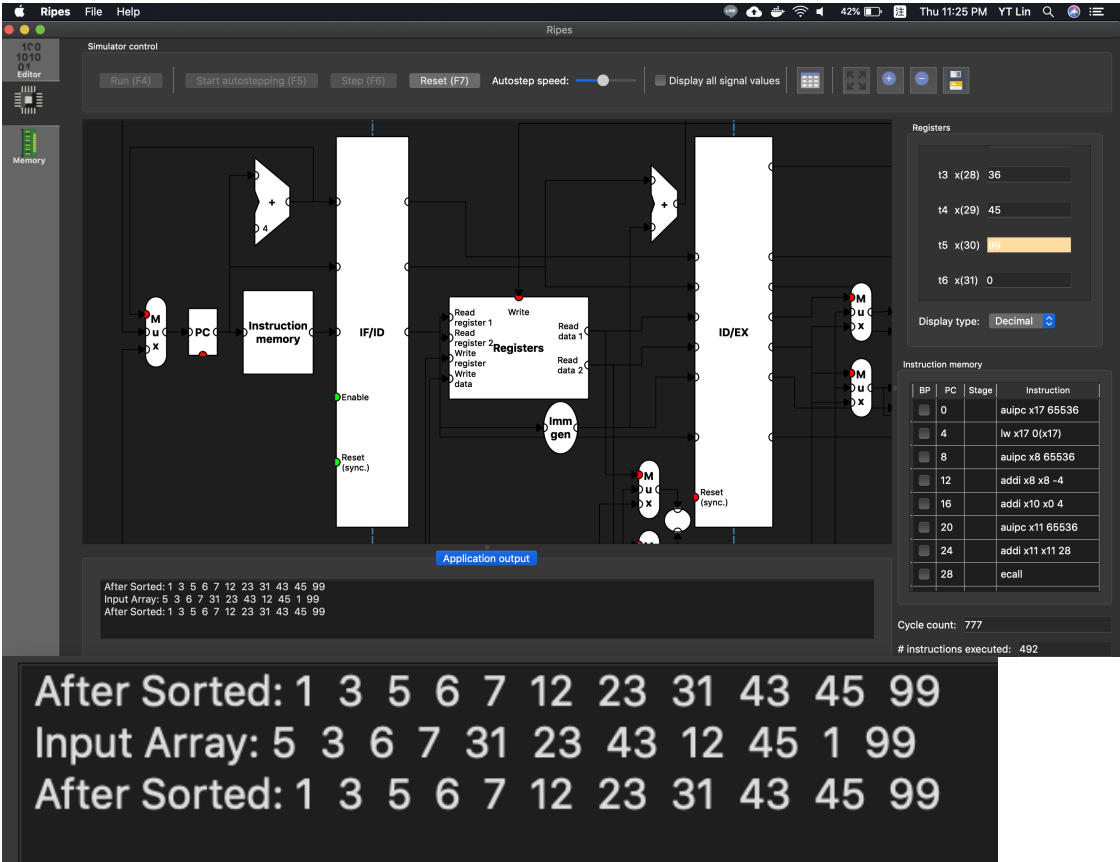
```

69         jal      ra, gcd_base
70
71         #return directly (now a0 should be result)
72         lw        ra, 16(sp)
73         addi      sp, sp, 32
74         ret
75
76     ### need to use jal(to change ra) or ret to main(bug)
77     swap:
78         mv t0, a0
79         mv a0, a1
80         mv a1, t0
81         ret #was bug without jal, return to main, debug by step-by-step ex
82
83
84     # expects:
85     # a0: gcd(arg1, arg2)
86     # a1: arg1
87     # a2: arg2
88     printResult:
89         mv        t0, a0
90         mv        t1, a1
91         mv        t2, a2
92
93         # a1 = param, a0 = print type
94         la        a1, str1
95         li        a0, 4
96         ecall
97
98         mv        a1, t1
99         li        a0, 1
100        ecall
101
102        la        a1, str2
103        li        a0, 4
104        ecall
105
106        mv        a1, t2
107        li        a0, 1
108        ecall
109
110        la        a1, str3
111        li        a0, 4
112        ecall
113
114        mv        a1, t0
115        li        a0, 1
116        ecall
117
118        la        a1, endl
119        li        a0, 4
120        ecall
121
122        ret

```

BubbleSort

Execution Result



Code

```

1  # author : Yan-Tong Lin
2  # all right reserves
3  # no plagiarism
4
5  # usage :
6  # arr = the array that want to sort
7  # N, size of arr
8
9
10 .data
11 N:      .word 11
12 arr:    .word 5,3,6,7,31,23,43,12,45,1,99
13 msg1:   .string "Input Array: "
14 msg2:   .string "After Sorted: "
15 space:  .string " "
16 endl:   .string "\n"
17
18 # a7 = N(constant in this program), s1 = i
19 # s0 = arr start pointer(const)
20 # s1 = arr pointer
21 # t1 = i
22 # t2 = j
23 # t3 = delta with t2(t2 << 2)
24 # t4 = arr[j]
25 # t5 = arr[j+1]
26
27 .text
28 main:
29     # a7 = N, constant in this program
30     lw      a7, N
31     la      s0, arr
32
33     # print msg1
34     li      a0, 4
35     la      a1, msg1
36     ecall
37
38     # load array to s0
39     # i = 0
40     # print array with loop
41     la      s1, arr
42     li      t1, 0
43     jal     ra, print_loop
44
45     # print endl
46     li      a0, 4
47     la      a1, endl
48     ecall
49
50     # now use t1 as i, init with 0
51     # about to start sort
52     # t1 = i, s0 = arr start, constant when sorting
53     li      t1, 0
54     la      s0, arr
55
56 #Bubble Sort Starts Here
57
58 fori:
59     # if i == N, break
60     beq     t1, a7, end_of_sort
61     # j = i - 1
62     addi    t2, t1, -1
63
64 forj:
65     #if j < 0, break
66     blt     t2, zero, to_nxt_i
67
68     # t3 = j << 2

```



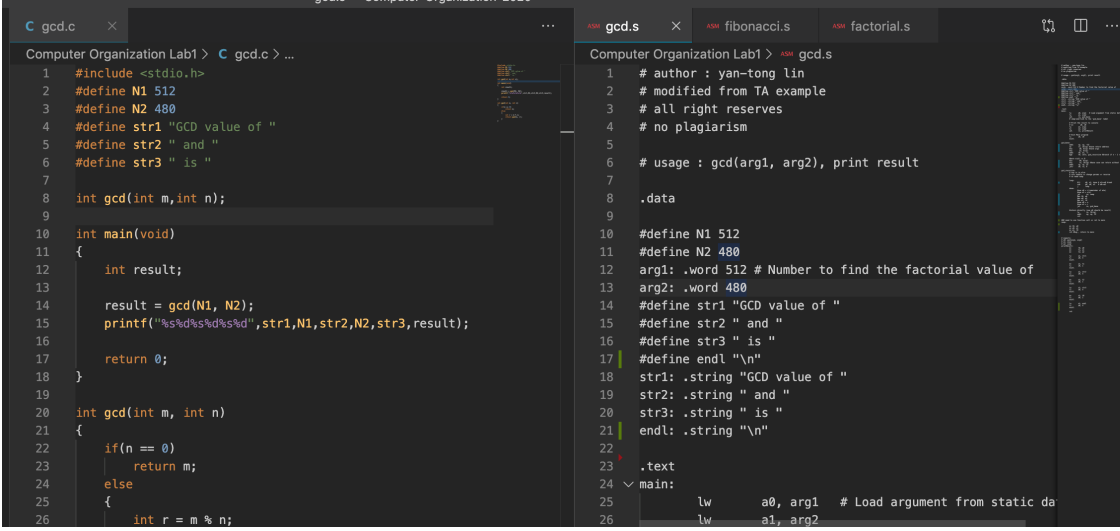
```

69         # 4 byte per word, t3 is the actual distance from arr start(t4)
70         slli    t3, t2, 2
71         # t4 is now arr[j]
72         add     s1, s0, t3
73
74         # load arr[j] and arr[j+1]
75         lw      t4, 0(s1)
76         lw      t5, 4(s1)
77
78         # if arr[j] <= arr[j+1](i.e., is sorted before)
79         # break
80         ble     t4, t5, to_nxt_i
81
82         # else swap arr[j] and arr[j+1](by direct saving), continue
83         sw      t4, 4(s1)
84         sw      t5, 0(s1)
85
86         # j--, continue
87         addi    t2, t2, -1
88         j       forj
89
90     #end of the inner loop
91     to_nxt_i:
92         #i++
93         addi    t1, t1, 1
94         j       fori
95
96
97     #End of the outer loop
98     end_of_sort:
99         # print msg2
100        li      a0, 4
101        la      a1, msg2
102        ecall
103
104        # i = 0
105        # print sorted array with loop
106        li      t1, 0
107        la      s1, arr
108        jal     ra, print_loop
109
110        # endl
111        li      a0, 4
112        la      a1, endl
113        ecall
114
115        #Exit program
116        li      a0, 10
117        ecall
118
119
120     # expectations:
121     # s1 is the array pointer now
122     # t1 is i
123     print_loop:
124         lw      a1, 0(s1)
125         li      a0, 1
126         ecall
127
128         # print space
129         la      a1, space
130         li      a0, 4
131         ecall
132
133         # i(t1)++
134         # arr ptr(s1) +=4
135         addi    s1, s1, 4
136         addi    t1, t1, 1
137

```

```
138     blt     t1, a7, print_loop
139     ret
```

Development ScreenShots



```
Computer Organization Lab1 > C gcd.c > ...
1  #include <stdio.h>
2  #define N1 512
3  #define N2 480
4  #define str1 "GCD value of "
5  #define str2 " and "
6  #define str3 " is "
7
8  int gcd(int m,int n);
9
10 int main(void)
11 {
12     int result;
13
14     result = gcd(N1, N2);
15     printf("%s%d%s%d%sd",str1,N1,str2,N2,str3,result);
16
17     return 0;
18 }
19
20 int gcd(int m, int n)
21 {
22     if(n == 0)
23         return m;
24     else
25     {
26         int r = m % n;
```

```
Computer Organization Lab1 > gcd.s
1  # author : yan-tong lin
2  # modified from TA example
3  # all right reserves
4  # no plagiarism
5
6  # usage : gcd(arg1, arg2), print result
7
8  .data
9
10 #define N1 512
11 #define N2 480
12 arg1: .word 512 # Number to find the factorial value of
13 arg2: .word 480
14 #define str1 "GCD value of "
15 #define str2 " and "
16 #define str3 " is "
17 #define endl "\n"
18 str1: .string "GCD value of "
19 str2: .string " and "
20 str3: .string " is "
21 endl: .string "\n"
22
23 .text
24 main:
25     lw     a0, arg1 # Load argument from static da
26     lw     a1, arg2
```