COMPUTER STRUCTURE

PA1: Human compiler

Computer science & engineering

2017-18538 황선영

Problem 1: Greatest common divisor

```c
#include "gcd.h"

int gcd(const long lhs_, const long rhs_)
{
        long lhs = lhs_;
        long rhs = rhs_;

        while(lhs != rhs)
        {
                if(lhs > rhs)
                {
                        lhs = lhs - rhs;
                        continue;
                }
                rhs = rhs - lhs;
        }

        return lhs;
}
```

```
GCD:
        beq a0, a1, done
        bge a1, a0, b_bigger
        sub a0, a0, a1
        beq a0, a0, GCD
b_bigger:
        sub a1, a1, a0
        beq a1, a1, GCD
done:
        ret

addi sp, sp, -8

sw a0, 4(sp)
sw a1, 0(sp)
jal GCD
lw a0, 4(sp)
lw a1, 0(sp)
addi sp, sp, 8
```

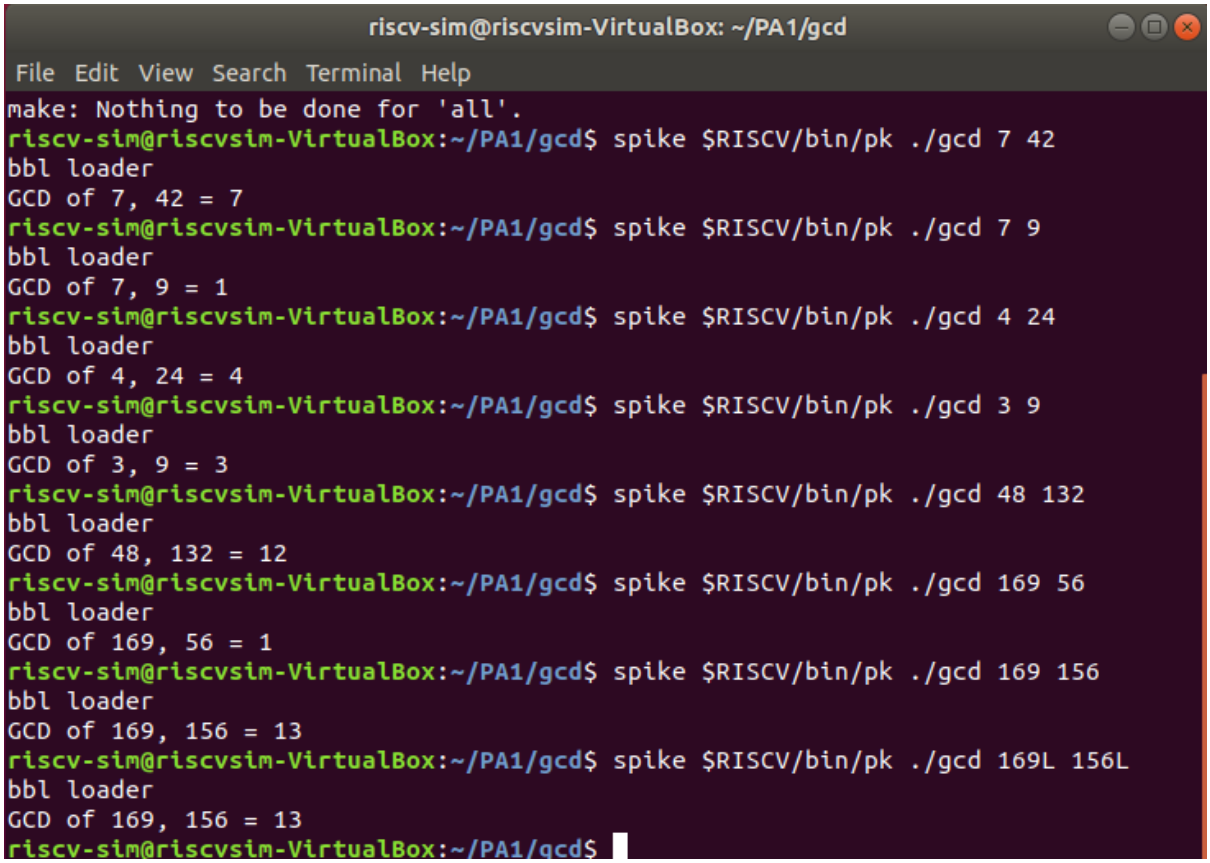
왼쪽은 최대공약수를 구하는 방법을 c로 나타낸 code이다. 오른쪽은 이 코드를 assembly로 나타낸 code이다. 먼저 알고리즘을 구현했다. a0, a1은 각각 lhs, rhs에 해당한다.

-GCD: while문에서 lhs와 rhs가 같으면 lhs을 return하므로 a0과 a1이 같으면 done으로 이동하도록 했다. if문에서 rhs가 lhs보다 크거나 같으면 if문을 빠져나가 b_bigger로 이동하도록 했다. 그 다음 if문 안에서 뺄셈을 수행하고 unconditional jump를 이용해 while loop를 구현했다.

-b_bigger: while문에서 rhs가 lhs보다 크거나 같을 때 뺄셈을 수행한 후 unconditional jump를 이용해 while loop를 돌도록 구현했다.

-done: lhs를 리턴하고 끝낸다.

-stack: 스택에 word 두개를 저장하기 위해 뺄셈으로 공간을 마련하고 a0, a1을 저장한 후 함수 gcd를 호출하고 스택에서 a0, a1를 pop했다.



이 assembly code를 실행하면 다음과 같은 결과가 나온다.

Problem 2: Fibonacci sequence

```
fibonacci:
        #------Your code starts here------
        #LHS: a0, RHS: a1

#type of count is unsigned
        addi sp, sp, -16
        sd a0, 8(sp)
        sd a1, 0(sp)

#assign opt[0]=1, opt[1]=1
        li t0, 1
        sd t0, 0(a0)


        li t1, 2
        blt a1, t1, fibonacci_ret # if

        sd t0, 8(a0)

        li t2, 2
        beq zero, zero, fibo

        fibo:
                blt a1, t2, exit
                mv t5, t2
                addi t5, t5, -2
```

```
        slli t6, t5, 3 # t6 = (i-2)*8
        add t6, t6, a0 # t6 = opt + (i-2)*8
        ld t3, 0(t6)
        ld t4, 8(t6)
        add s1, t3, t4
        sd s1, 16(t6)

        addi t2, t2, 1
        beq zero, zero, fibo

fibonacci_ret:
        jr ra

exit:
ld a1, 0(sp)
ld a0, 8(sp)
addi sp, sp, 16
```

먼저 스택에 배열의 첫번째 주소와 길이를 push한다. 그 후 opt[0]과 opt[1]에 1을 assign한다.

그 후 배열의 길이가 2보다 작으면 fobonacci_ret으로 보낸 후 return 한다. 그렇지 않으면 배열의 i번째 요소에 배열의 (i-1)번째 요소와 (i-2)번째 요소를 더하여 assign한다. I의 크기가 count-1이 되기전까지 반복한 후 동작을 마친다.

```
riscv-sim@riscvsim-VirtualBox:~/PA1$ cd fibonacci
riscv-sim@riscvsim-VirtualBox:~/PA1/fibonacci$ spike $RISCV/bin/pk ./fibonacci 5
bbl loader
1
1
2
3
5
z  0000000000000000 ra 0000000000013064 sp 000000007f7e9ae0 gp 000000000001da08
tp 0000000000000000 t0 0000000000000000 t1 000000007f7e9270 t2 0000219000030015
s0 000000000001e220 s1 000000000001d020 a0 000000000001d778 a1 000000000001d768
a2 0000000000000000 a3 0000000000000000 a4 000000000001e210 a5 0000000000000038
a6 000000000001e248 a7 0000000000000039 s2 0000000000000000 s3 0000000000000000
s4 0000000000000000 s5 0000000000000000 s6 0000000000000000 s7 0000000000000000
s8 0000000000000000 s9 0000000000000000 sA 0000000000000000 sB 0000000000000000
t3 0000000000000000 t4 0000000000000064 t5 0000000000000000 t6 0000000000000000
pc 000000000001312e va 0000000000000018 insn       ffffffff sr 8000000200046020
User store segfault @ 0x0000000000000018
riscv-sim@riscvsim-VirtualBox:~/PA1/fibonacci$ spike $RISCV/bin/pk ./fibonacci 6
bbl loader
1
1
2
3
5
8
```

결과는 다음과 같다. 결과는 잘 출력이 되나 count의 값이 홀수일 경우에만 결과를 출력한 후 user store segfault가 발생하는데 원인은 찾지 못하였다.

Problem 3: Maze Solving

```
        ld s1, 0(a0)
        mv s2, a1
        mv s3, a2
your_funct:
        addi sp, sp, -56
        sd ra, 48(sp)
        sd a0, 40(sp)
        sd a1, 32(sp)
        sd a2, 24(sp)
        sd a3, 16(sp)
        sd s4, 8(sp)
        sd s5, 0(sp)

        li t0, 20
        blt a2, t0, zero_checker

        li a0, -1
        jr ra

        zero_checker:
                bge a0, zero, zero_check

                zero_check:
                bge a1, zero, outof
                ld s5, 0(sp)
                ld s4, 8(sp)
                ld a3, 16(sp)
                ld a2, 24(sp)
                ld a1, 32(sp)
                ld a0, 40(sp)
                ld ra, 48(sp)
                addi sp, sp, 56

                li a0, -1
                jr ra
```

```
outof:
        blt a0, s2, Dead
        Dead:
        blt a1, s3, Dead_end

        ld s5, 0(sp)
        ld s4, 8(sp)
        ld a3, 16(sp)
        ld a2, 24(sp)
        ld a1, 32(sp)
        ld a0, 40(sp)
        ld ra, 48(sp)
        addi sp, sp, 56

        li a0, -1
        jr ra

Dead_end:
        mul t1, a1, s2
        add t1, t1, a0
        slli t1, t1, 3
        add t1, s1, t1
        ld t2, 0(t1)
        beq t2, zero, Success

        ld s5, 0(sp)
        ld s4, 8(sp)
        ld a3, 16(sp)
        ld a2, 24(sp)
        ld a1, 32(sp)
        ld a0, 40(sp)
        ld ra, 48(sp)
        addi sp, sp, 56

        li a0, -1
        jr ra
```

Traverse 함수의 일부이다. x_pos, y_pos, depth, prev_trav은 각각 a0, a1, a2, a3이고 g_maze, g_width, g_height는 각각 s1, s2, s3이다. Min과 result는 각각 s4, s5이다.

이 부분은 Dead_end까지의 코드를 어샘블리로 구현하였다.

```
Success:
        mv t3, s2
        addi t3, t3, -1
        bne a0, t3, tra
        next:
        mv t3, s3
        addi t3, t3, -1
        bne a1, t3, tra

        ld s5, 0(sp)
        ld s4, 8(sp)
        ld a3, 16(sp)
        ld a2, 24(sp)
        ld a1, 32(sp)
        ld a0, 40(sp)
        ld ra, 48(sp)
        addi sp, sp, 56

        mv a0, a2

        jr ra

tra:
        li s4, -1
        li t3, 3
        beq a3, t3, left

        addi sp, sp, -56
        sd ra, 48(sp)
        sd a0, 40(sp)
        sd a1, 32(sp)
        sd a2, 24(sp)
        sd a3, 16(sp)
        sd s4, 8(sp)
        sd s5, 0(sp)

        addi a1, a1, -1
        addi a2, a2, 1
        li a3, 0

        jal ra, your_funct
        ld s4, 0(a0) # min = your_funct
        ld s5, 0(sp)
        ld s4, 8(sp)
        ld a3, 16(sp)
        ld a2, 24(sp)
        ld a1, 32(sp)
        ld a0, 40(sp)
        ld ra, 48(sp)
        addi sp, sp, 56

left:
        li t3, 2
        beq a3, t3, right

        addi sp, sp, -56
        sd ra, 48(sp)
        sd a0, 40(sp)
        sd a1, 32(sp)
        sd a2, 24(sp)
        sd a3, 16(sp)
        sd s4, 8(sp)
        sd s5, 0(sp)

        addi a0, a0, -1
        addi a2, a2, 1
        li a3, 1

        jal ra, your_funct
        ld s5, 0(a0)     # result = your_funct

        blt s5, zero, right

        ld s5, 0(sp)
        ld s4, 8(sp)
        ld a3, 16(sp)
        ld a2, 24(sp)
        ld a1, 32(sp)
        ld a0, 40(sp)
        ld ra, 48(sp)
        addi sp, sp, 56

        bge s4, zero, left_mer
        mv s4, s5
        beq zero, zero, right

        left_mer:
                bge s5, s4, right
                mv s4, s5
```

maze.c 코드를 어샘블리로 구현하였다. 함수를 호출할 때는 모든 레지스터를 스택에 push한 후 함수 호출 외의 다른 동작들을 구현한 후 다시 모든 레지스터를 스택에서 pop했다. 앞으로 나오는 함수 호출도 같은 방식으로 구현했다.

```
right:
        li t3, 1
        beq a3, t3, down

        addi sp, sp, -56
        sd ra, 48(sp)
        sd a0, 40(sp)
        sd a1, 32(sp)
        sd a2, 24(sp)
        sd a3, 16(sp)
        sd s4, 8(sp)
        sd s5, 0(sp)

        addi a0, a0, 1
        addi a2, a2, 1
        li a3, 2

        jal ra, your_funct
        ld s5, 0(a0)     # result = your_funct
        blt s5, zero, down

        ld s5, 0(sp)
        ld s4, 8(sp)
        ld a3, 16(sp)
        ld a2, 24(sp)
        ld a1, 32(sp)
        ld a0, 40(sp)
        ld ra, 48(sp)
        addi sp, sp, 56

        bge s4, zero, right_mer
        mv s4, s5
        beq zero, zero, down

        right_mer:
                bge s5, s4, down
                mv s4, s5
```

```
down:
        li t3, 0
        beq a3, t3, return

        addi sp, sp, -56
        sd ra, 48(sp)
        sd a0, 40(sp)
        sd a1, 32(sp)
        sd a2, 24(sp)
        sd a3, 16(sp)
        sd s4, 8(sp)
        sd s5, 0(sp)

        addi a1, a1, 1
        addi a2, a2, 1
        li a3, 3

        jal ra, your_funct
        ld s5, 0(a0)     # result = your_funct
        blt s5, zero, return

        ld s5, 0(sp)
        ld s4, 8(sp)
        ld a3, 16(sp)
        ld a2, 24(sp)
        ld a1, 32(sp)
        ld a0, 40(sp)
        ld ra, 48(sp)
        addi sp, sp, 56

        bge s4, zero, down_mer
        mv s4, s5
        beq zero, zero, return

        down_mer:
                bge s5, s4, return
                mv s4, s5
```

```
return:
        mv a0, s4
        ld s5, 0(sp)
        ld s4, 8(sp)
        ld a3, 16(sp)
        ld a2, 24(sp)
        ld a1, 32(sp)
        ld a0, 40(sp)
        ld ra, 48(sp)
        addi sp, sp, 56

    #Ret
    jr      ra
    .size   your_funct, .-your_funct

    #------Your code ends here

    .align  1
    .globl  solve_maze
    .type   solve_maze, @function
solve_maze:
    #------Your code starts here------
    #maze: a0, width: a1, height: a2

    addi sp, sp, -32
    sd ra, 24(sp)
    sd a0, 16(sp)
    sd a1, 8(sp)
    sd a2, 0(sp)

    ld t1, 0(a0)
    mv t2, a1

#here load 0, 0, 0, T_RIGHT

    li a0, 0
    li a1, 0
    li a2, 0
    li a3, 2
    jal ra, your_funct

    mv a0, t1
    mv a1, t2

    ld a2, 0(sp)
    ld a1, 8(sp)
    ld a0, 16(sp)
    ld ra, 24(sp)
```

solve_maze 함수에서는 레지스터를 스택에 push한 다음 traverse(0,0,0,2)를 return하기 위하여 적당한 레지스터에 값을 load한 후 jar을 이용해 your_funct에 jump한 다음 레지스터를 스택에서 pop했다.

```
riscv-sim@riscvsim-VirtualBox:~/PA1/maze$ spike $RISCV/bin/pk ./maze input1.txt
bbl loader
Maze layout:
0111
0001
1000
The shortest path length is: 158592
riscv-sim@riscvsim-VirtualBox:~/PA1/maze$ spike $RISCV/bin/pk ./maze input2.txt
bbl loader
Maze layout:
001111111
001111111
100000001
110000111
111101111
111101000
111100010
111111110
111110000
The shortest path length is: 158592
```

배열은 잘 출력되나 정답이 나오지 않는다. 함수 호출이나 return address, return value를 잘 살펴보았으나 원인과 해결책을 찾지 못하였다.