Lab3

`Venus` 是一个`RISC-v` 模拟器。here

Exercise 1: Familiarizing yourself with Venus

把`ex1.s` 复制到`venus` ， 自己玩玩吧。

由于我本人已经有了`x86` 的基础，我们就简单地过下`RISC-V`吧。

```
add x5,x6,x7 #x5 = x6+x7
sub x5,x6,x7 #x5 = x6-x7
addi x5,x6,20 #x5 = x6+20
ld x5, 40(x6) #x5 = memory[x6+40], load doubleword
sd x5,40(x6)  # memory[x6+40] = x5 store doubleword
lw #load word
lwu #load unsigned word
sw  #store word
lh  # load halfword
lhu # load unsigned halfword
sh
lb
lbu
sb
lr.d
sc.d
lui
and
or
xor
andi
ori
xori
sll x5,x6,x7 # x5 = x6<<x7
srl x5,x6,x7 # x5 = x6>>x7
sra
slli x5,x6,3 # x5 = x6<<3
srli x5,x6,3 # x5 = x6>>3
```

Exercise 2: Translating from C to RISC-V

看我注释就行了。

`ex2.c`

```
int source[] = {3, 1, 4, 1, 5, 9, 0};
int dest[10];

int fun(int x) {
    return -x * (x + 1);
}

int main() {
    int k;
    int sum = 0;
    for (k = 0; source[k] != 0; k++) {
        dest[k] = fun(source[k]);
        sum += dest[k];
    }
    return sum;
}
```

`ex2.s`

```
.data
source:
    .word   3
    .word   1
    .word   4
    .word   1
    .word   5
    .word   9
    .word   0
dest:
    .word   0
    .word   0
    .word   0
    .word   0
    .word   0
    .word   0
    .word   0
    .word   0
    .word   0
    .word   0

.text
main:
    addi t0, x0, 0 #k
    addi s0, x0, 0 #sum
    la s1, source # source
    la s2, dest  # dest
loop:
    slli s3, t0, 2 # k*4
    add t1, s1, s3 # t1 = s1+k*4(索引)
    lw t2, 0(t1)  # t2  = source[k]
    beq t2, x0, exit #if source[k]==0,跳转exit
    add a0, x0, t2  #
    addi sp, sp, -8
    sw t0, 0(sp)
    sw t2, 4(sp)
    jal square
    lw t0, 0(sp)
    lw t2, 4(sp)
    addi sp, sp, 8
    add t2, x0, a0
    add t3, s2, s3
    sw t2, 0(t3)
    add s0, s0, t2
    addi t0, t0, 1
    jal x0, loop
square:
    add t0, a0, x0 #t0 = source[k]
    add t1, a0, x0 #t1 = source[k]
    addi t0, t0, 1 #source[k]+1
    addi t2, x0, -1 #t2 = -1
    mul t1, t1, t2 # -source[k]
    mul a0, t0, t1 # (source[k]+1)*(-source[k])
    jr ra
exit:
    add a0, x0, s0
    add a1, x0, x0
    ecall # Terminate ecall
```

Exercise 3: Factorial

本练习需要我们用汇编写一个求阶乘的函数

我们先写一个`c`语言吧，然后尝试将其翻译为汇编：

```
int facrotial(x){
    if (x<1){
        return 1;
    }
    else{
        return x*facrotial(x-1);
    }
}
```

`汇编`

```
.globl factorial

.data
n: .word 8

.text
main:
    la t0, n
    lw x10, 0(t0)
    jal x1, factorial

    addi x11, x10, 0
    addi x10, x0, 1
    ecall # Print Result

    addi x11, x0, '\n'
    addi x10, x0, 11
    ecall # Print newline

    addi x10, x0, 10
    ecall # Exit

factorial:
    addi sp,sp,-16
    sw x1, 8(sp)
    sw x10, 0(sp)
    addi x5, x10, -1 #x5 = n-1
    bge x5,x0,L1
    addi x10,x0,1
    addi sp,sp,16
    jr x1

L1:
    addi x10, x10, -1
    jal x1, factorial
    addi x6,x10,0
    lw x10,0(sp)
    lw x1,8(sp)
    addi sp,sp,16
    mul x10,x10,x6
    jr x1
```

建议写循环，递归比较难写对。

Exercise 4: RISC-V function calling with `map`

本任务要求我们实现一个函数`map`，其大概长这样

```
void map(struct node *head, int (*f)(int))
{
    if(!head) { return; }
    head->value = f(head->value);
    map(head->next,f);
}

```

第一个参数为链表的头节点，第二个参数为一个函数指针

```
.globl map

.text
main:
    jal ra, create_default_list
    add s0, a0, x0  # a0 = s0 is head of node list

    #print the list
    add a0, s0, x0
    jal ra, print_list

    # print a newline
    jal ra, print_newline

    # load your args
    add a0, s0, x0  # load the address of the first node into a0

    # load the address of the function in question into a1 (check out la on the green sheet)
    la a1, square

    # issue the call to map
    jal ra, map

    # print the list
    add a0, s0, x0
    jal ra, print_list

    # print another newline
    jal ra, print_newline

    addi a0, x0, 10
    ecall #Terminate the program

map:
    # Prologue: Make space on the stack and back-up registers
    addi sp, sp, -12
    sw ra, 0(sp)
    sw s0, 4(sp)
    sw s1, 8(sp)

    beq a0, x0, done    # If we were given a null pointer (address 0), we're done.

    add s0, a0, x0  # Save address of this node in s0
    add s1, a1, x0  # Save address of function in s1

    # Remember that each node is 8 bytes long: 4 for the value followed by 4 for the pointer to next.
    # What does this tell you about how you access the value and how you access the pointer to next?

    # load the value of the current node into a0
    # THINK: why a0?
    lw a0, 0(s0)

    # Call the function in question on that value. DO NOT use a label (be prepared to answer why).
    # What function? Recall the parameters of "map"
    jalr ra, a1, 0

    # store the returned value back into the node
    # Where can you assume the returned value is?
    sw a0, 0(s0)

    # Load the address of the next node into a0
    # The Address of the next node is an attribute of the current node.
    # Think about how structs are organized in memory.
    lw a0, 4(s0)

    # Put the address of the function back into a1 to prepare for the recursion
    # THINK: why a1? What about a0?
    la, a1, square

    # recurse
    jal, ra, map

done:
    # Epilogue: Restore register values and free space from the stack
    lw  ra, 0(sp)
    lw  s0, 4(sp)
    lw  s1, 8(sp)
    addi sp, sp, 12

    jr ra # Return to caller

square:
    mul a0 ,a0, a0
    jr ra

create_default_list:
    addi sp, sp, -12
    sw  ra, 0(sp)
    sw  s0, 4(sp)
    sw  s1, 8(sp)
    li  s0, 0       # pointer to the last node we handled
    li  s1, 0       # number of nodes handled
loop:   #do...
    li  a0, 8
    jal ra, malloc      # get memory for the next node
    sw  s1, 0(a0)   # node->value = i
    sw  s0, 4(a0)   # node->next = last
    add s0, a0, x0  # last = node
    addi    s1, s1, 1   # i++
    addi t0, x0, 10
    bne s1, t0, loop    # ... while i!= 10
    lw  ra, 0(sp)
    lw  s0, 4(sp)
    lw  s1, 8(sp)
    addi sp, sp, 12
    jr ra

print_list:
    bne a0, x0, printMeAndRecurse
    jr ra       # nothing to print
printMeAndRecurse:
    add t0, a0, x0  # t0 gets current node address
    lw  a1, 0(t0)   # a1 gets value in current node
    addi a0, x0, 1      # prepare for print integer ecall
    ecall
    addi    a1, x0, ' '     # a0 gets address of string containing space
    addi    a0, x0, 11      # prepare for print string syscall
    ecall
    lw  a0, 4(t0)   # a0 gets address of next node
    jal x0, print_list  # recurse. We don't have to use jal because we already have where we want to return to in ra

print_newline:
    addi    a1, x0, '\n' # Load in ascii code for newline
    addi    a0, x0, 11
    ecall
    jr  ra

malloc:
    addi    a1, a0, 0
    addi    a0, x0 9
    ecall
    jr  ra
```

- `j label`: 跳到label
- `jal dst label`:
- `jalr dst src imm`
- `jr src`