实验 1：MIPS 程序设计

实验时间：2025/2/19 - 2025/2/22
实验人员：李维杰，23307140052
指导老师：王雪平

1 实验目的

实验目的：

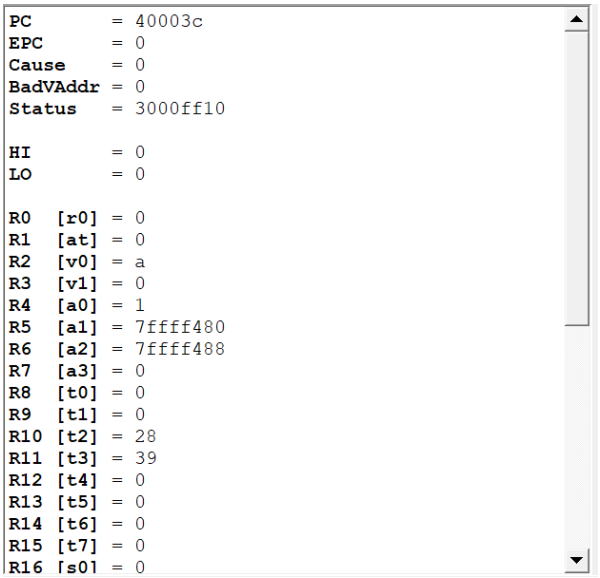
- 熟悉 QtSPIM 模拟器；
- 熟悉编译器、汇编程序和链接器；
- 熟悉 MIPS 体系结构的计算，包括
    a. MIPS 的数据表示；
    b. 熟悉 MIPS 指令格式和寻址方式；
    c. 熟悉 MIPS 汇编语言；
    d. 熟悉 MIPS 的各种机器代码表示，包括
        1) 选择结构；
        2) 循环结构；
        3) 过程调用：调用与返回、栈、调用约定等；
        4) 系统调用。

其中加粗的部分要特别注意掌握。

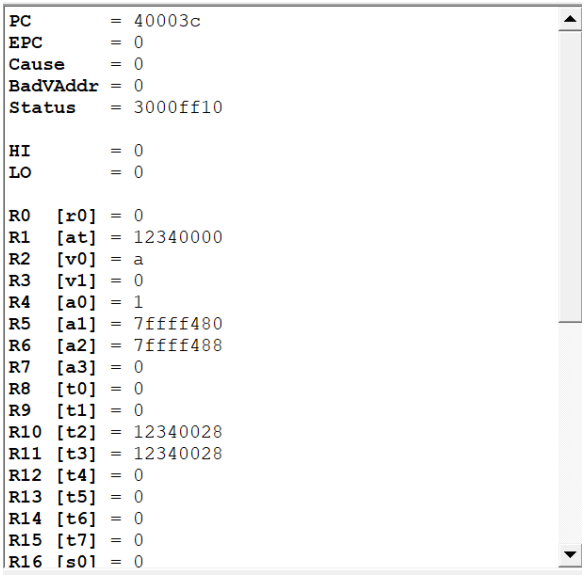
2 实验过程

2.1 调试

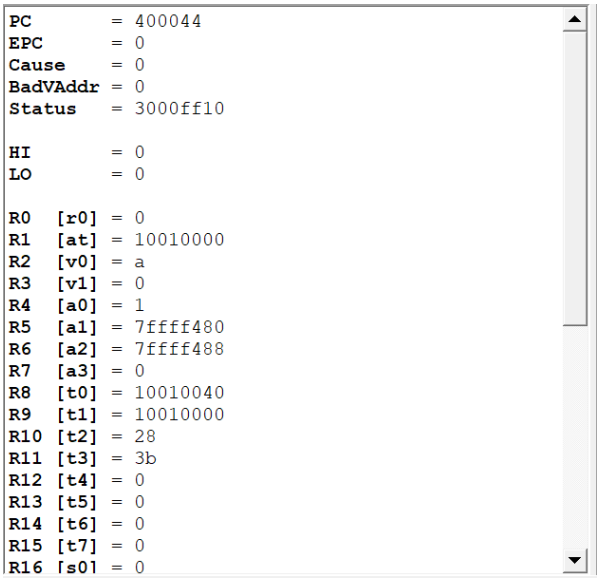
P1.asm 运行结果：



P2.asm 运行结果：



P3.asm 运行结果：



2.2 改写程序

翻译代码（源码见 p1-new.asm）

```
            .data

msg1:    .asciiz "Please enter 1st number: "
msg2:   .asciiz  "Please enter 2nd number: "
msg3:   .asciiz  "The result of "
msg4:   .asciiz  " & "
msg5:   .asciiz  " is: "
msg6:    .asciiz  "\nDo you want to try another(0—continue/1—
exit): "

        .text
```

```
        .globl main

main:
loop:
    li  $v0, 4
    la  $a0, msg1
    syscall

    li  $v0, 5
    syscall
    move    $t1, $v0

    li  $v0, 4
    la  $a0, msg2
    syscall

    li  $v0, 5
    syscall
    move    $t2, $v0

    add $t0, $t1, $t2

    li  $v0, 4
    la  $a0, msg3
    syscall

    li  $v0, 1
    move    $a0, $t1
    syscall

    li  $v0, 4
    la  $a0, msg4
    syscall

    li  $v0, 1
    move    $a0, $t2
    syscall

    li  $v0, 4
    la  $a0, msg5
    syscall

    li  $v0, 1
    move    $a0, $t0
```

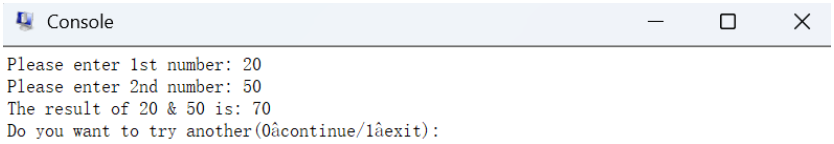
```
    syscall

    li  $v0, 4
    la  $a0, msg6
    syscall

    li  $v0, 5
    syscall
    move    $t3, $v0

    bne $t3, $0, done
    j   loop
done:
    li  $v0, 10
    syscall
```

运行结果如图：



2.3 把 C 代码翻译成 MIPS 代码

翻译代码（源码见 p4.asm）

```
     .data

arrs:   .word 9, 7, 15, 19, 20, 30, 11, 18  # 数组 arrs
N:      .word 8                             # 数组长度 N
msg:    .asciiz "The result is: "           # 输出的字符串

        .text
        .globl main

sumn:
        # 参数: $a0 -> arr, $a1 -> n
        # 返回值: $v0 -> sum

        move    $t0, $0         # sum = 0
        move    $t1, $0         # idx = 0

loop:
        slt     $t2, $t1, $a1
```

```
        beq     $t2, 0, done
       sll     $t3, $t1, 2 # 计算 arr[idx] 的地址偏移 (idx * 4)
        add     $t3, $t3, $a0     # arr + (idx * 4)
        lw      $t4, 0($t3)
        add     $t0, $t0, $t4
        addi    $t1, $t1, 1
        j       loop

done:
        move    $v1, $t0
        jr      $ra

main:
        la      $a0, arrs
        lw      $a1, N

        jal     sumn

        li      $v0, 4
        la      $a0, msg
        syscall

        li      $v0, 1
        move    $a0, $v1
        syscall

        li      $v0, 10
        syscall
```

2.4 优化代码

优化代码（源码见 fib-op.asm）

```
    ## Daniel J. Ellard -- 02/27/94
## fib-o.asm-- A program to compute Fibonacci numbers.
## An optimized version of fib-t.asm.
## main--
## Registers used:
## $v0 - syscall parameter and return value.
## $a0 - syscall parameter-- the string to print.

.text
main:
    subu $sp, $sp, 32       # Set up main’s stack frame:
```

```
    sw $ra, 28($sp)
    sw $fp, 24($sp)
    addu $fp, $sp, 32

    ## Get n from the user, put into $a0.
    li $v0, 5               # load syscall read_int into $v0.
    syscall                 # make the syscall.
    move $a0, $v0           # move the number read into $a0.
    jal fib                 # call fib.

    move $a0, $v0
     li $v0, 1                 # load syscall print_int into
$v0.
    syscall                 # make the syscall.

    la $a0, newline
    li $v0, 4
    syscall                 # make the syscall.

    li $v0, 10              # 10 is the exit syscall.
    syscall                 # do the syscall.

## fib-- (hacked-up caller-save method)
## Registers used:
## $a0 - initially n.
## $t0 - parameter n.
## $t1 - fib (n - 1).
## $t2 - fib (n - 2).

.text
fib:
    bgt $a0, 1, fib_recurse  # if n < 2, then just return a 1,
    li $v0, 1               # don't build a stack frame.
    jr $ra

# otherwise, set things up to handle
fib_recurse:                # the recursive case:
    li  $t0, 0
    li  $t3, 0
    li  $t4, 1
    li  $t5, 1
loop:
    slt $t1, $t0, $a0
    beq $t1, 0, done
```

```
    add $t5, $t3, $t4
    move $t3, $t4
    move $t4, $t5
    addi $t0, $t0, 1
    j   loop
done:
    move $v0, $t5
    jr $ra
## data for fib-o.asm:
.data
newline: .asciiz "\n"
```

说明：优化部分在于把递归改写为循环，使得求解的时间复杂度降为 O(n)。

3 实验结论

无

4 实验感想

学到了很多 MIPS 语言及应用，在学习过程中常与 C 语言作比较，对照着学可以轻松不少。