**实验一：MIPS程序设计**

实验时间： 2024年02月28日

实验人员： 杨润东

指导老师： 刘森

1. **实验目的**

熟悉MIPS汇编语言和QtSpim Simulator的使用.

(Note: code of each helper function is attached to the appendix)

1. **实验过程**

对实验中的每一项任务，应该包括设计过程；包括输入过程，输出过程，以及部分核心代码（不要贴入全部代码！），代码应具有较强的可读性（良好的代码风格和适当的注释）。

1. 调试给定的程序p1.asm, p2.asm, p3.asm并记录结果.

PC(program counter) 在运行结束后均与运行前不同.

1. p1.asm

Registers whose value is changed are:

$v0, $t2, $t3, $ra

the reason is that:

ori operation changed the value of $t2 to 0x28;

ori operation changed the value of $t3 to 0x11;

After ‘add’ operation, $t3 = $t3+$t2 which is 0x39;

ori won’t change the value of $0.

ra is changed because the OS called main function without resetting it.

1. p2.asm

We can use stack to store ‘words’ by manipulating stack pointer($sp). A word is 4 bytes in 32-bit MIPS,

so we can use ‘sw’(store word) by “sw $t2, 0($sp)”, “sw $t3, 4($sp)”, i.e. the address to write to increase by 4. To access these values stored in stack, we can call “lw”.

To terminate the program, set $v0 to be 0xa and use ‘syscall’.

1. p3.asm

in “A: .word 1,2,3,5,7” we defined a array.

The values contained by registers are:

$t0: the address of h

$t1, the address of A

$t2, the 0th word in array h which is 0x28.

$t3, the 12th word in array A which is integer 0x13.(before line 42)

② 改写p1.asm，使用MIPS汇编指令和QtSPIM模拟器，接收两个整数，计算结果后输出.

First use the function ‘sumpr’ to address one loop of inputs and outputs:

.data

# string literals

pmpt1: .asciiz "Please enter 1st number: "

pmpt2: .asciiz "Please enter end number: "

pmpt31: .asciiz "The result of "

pmpt32: .asciiz " & "

pmpt33: .asciiz " is: "

pmpt4: .asciiz "Do you want to try another(0-continue/1-exit): "

empty: .asciiz ""

dashn: .asciiz "\n"

###############

# read two integers and print the sum.

###############

sumpr:

    # store original value

    addi $sp, $sp, -12

    sw $s0, 0($sp)

    sw $s1, 4($sp)

    sw $ra, 8($sp)

    # get the left operand

    addi $a0, $0, 4

    la $a1, pmpt1

    jal print

    jal readint

    add $s0, $a0, $0

    # get the second operand

    addi $a0, $0, 4

    la $a1, pmpt2

    jal print

    jal readint

    add $s1, $a0, $0

    addi $a0, $0, 4

    la $a1, pmpt31

    jal print

    addi $a0, $0, 1

    add $a1, $s0, $0

    jal print

    addi $a0, $0, 4

    la $a1, pmpt32

    jal print

    addi $a0, $0, 1

    add $a1, $s1, $0

    jal print

    addi $a0, $0, 4

    la $a1, pmpt33

    jal print

    add $a1, $s0, $s1

    addi $a0, $0, 1

    jal println

    # reset $s0, $s1.

    lw $s0, 0($sp)

    lw $s1, 4($sp)

    lw $ra, 8($sp)

    addi $sp, $sp, 12

    # return

    jr $ra

Then use branch control to make it possible to run in a loop, consider the ‘run’ function below:

###############

# run sumpr many times

###############

run:

    # use $s0 as counter.

    addi $sp, $sp, -8

    sw $s0, 0($sp)

    sw $ra, 4($sp)

  loop:

    jal sumpr

    addi $a0, $0, 4

    la $a1, pmpt4

    jal print

    jal readint

    add $s0, $a0, $0

    bne $s0, $0, endloop

    j loop

  endloop:

    # restore $s0, $ra.

    lw $s0, 0($sp)

    lw $ra, 4($sp)

    addi $sp, $sp, 8

    # return.

    jr $ra

It will not stop unless we enter ‘1’ at prompt ‘pmpt4’.

Then use a main function to complete the task.

main:

    # call function 'run'

    jal run

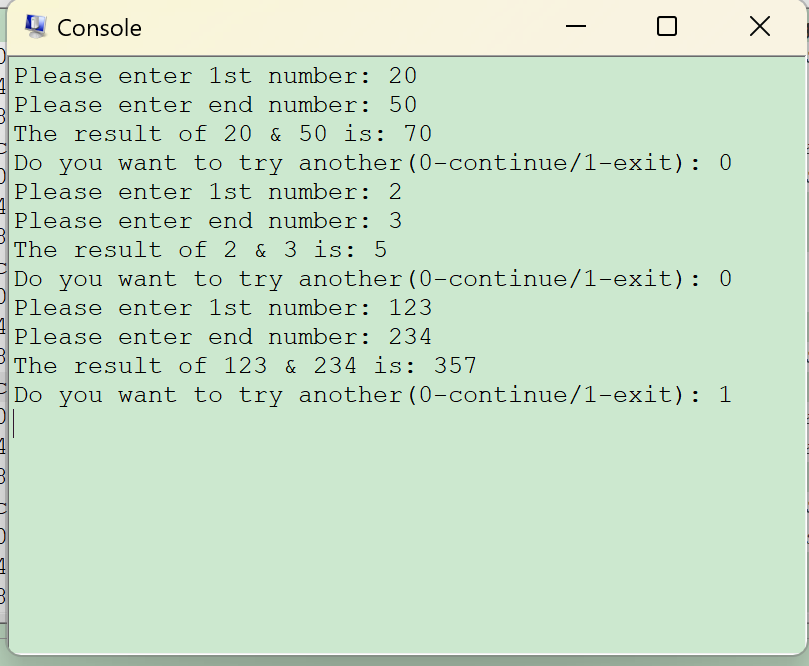
    # terminate

    addi $v0, $0, 10

    syscall

A typical output looks like this:

check all s registers and $sp, their values are not changed. Thus the program is correct.

 ③ 把C代码翻译成MIPS代码.

First translate the ‘sumn’ function as follows:

###############

# sumn

# @param $a0: the array

# @param $a1: size of the array

# @returns sums of the array($a0)

###############

sumn:

    # $s0: the array

    # $s1: the loop variable idx.

    # $s2: the sum.

    addi $sp, $sp, -12

    sw $s0, 0($sp)

    sw $s1, 4($sp)

    sw $s2, 8($sp)

    add $s0, $a0, $0

    add $s1, $0, $0

    add $s2, $0, $0

  lp:

    add $t0, $s1, $0

    sll $t0, $t0, 2

    add $t0, $s0, $t0

    lw $t1, 0($t0)

    add $s2, $t1, $s2

    addi $s1, $s1, 1

    bne $a1, $s1, lp

  endLp:

    # store the result in $a0.

    add $a0, $s2, $0

    # reset stack pointer and s register

    lw $s0, 0($sp)

    lw $s1, 4($sp)

    lw $s2, 8($sp)

    addi $sp, $sp, 12

    # return

    jr $ra

Then translate the ‘main’ function as follows:

main:

    # store the arrs to $s0.

    # size in $t0.

    # sum in $s1.

    addi $sp, $sp, -8

    sw $s0, 0($sp)

    sw $s1, 4($sp)

    # construct the array(in stack)

    # arr = {9, 7, 15, 19, 20, 30, 11, 18}

    addi $sp, $sp, -32

    add $s0, $sp, $0

    # 9

    addi $t0, $0, 9

    sw $t0, 0($s0)

    # 7

    addi $t0, $0, 7

    sw $t0, 4($sp)

    # 15

    addi $t0, $0, 15

    sw $t0, 8($sp)

    # 19

    addi $t0, $0, 19

    sw $t0, 12($sp)

    # 20

    addi $t0, $0, 20

    sw $t0, 16($sp)

    # 30

    addi $t0, $0, 30

    sw $t0, 20($sp)

    # 11

    addi $t0, $0, 11

    sw $t0, 24($sp)

    # 18

    addi $t0, $0, 18

    sw $t0, 28($sp)

    # N = 8

    addi $t0, $0, 8

    addi $sp, $sp, -4

    sw $t0, 0($sp)

    add $a0, $s0, $0

    add $a1, $t0, $0

    jal sumn

    add $s1, $a0, $0 # result in $s1.

    lw $t0, 0($sp)

    addi $sp, $sp, 4

    # print the result.

    addi $sp, $sp, -4

    sw $t0, 0($sp)

    addi $a0, $0, 4

    la $a1, pmpt

    jal print

    lw $t0, 0($sp)

    addi $sp, $sp, 4

    addi $sp, $sp, -4

    sw $t0, 0($sp)

    addi $a0, $0, 1

    add $a1, $s1, $0

    jal print

    lw $t0, 0($sp)

    addi $sp, $sp, 4

    addi $sp, $sp, 32  # restore stack.

    lw $s0, 0($sp)

    lw $s1, 4($sp)

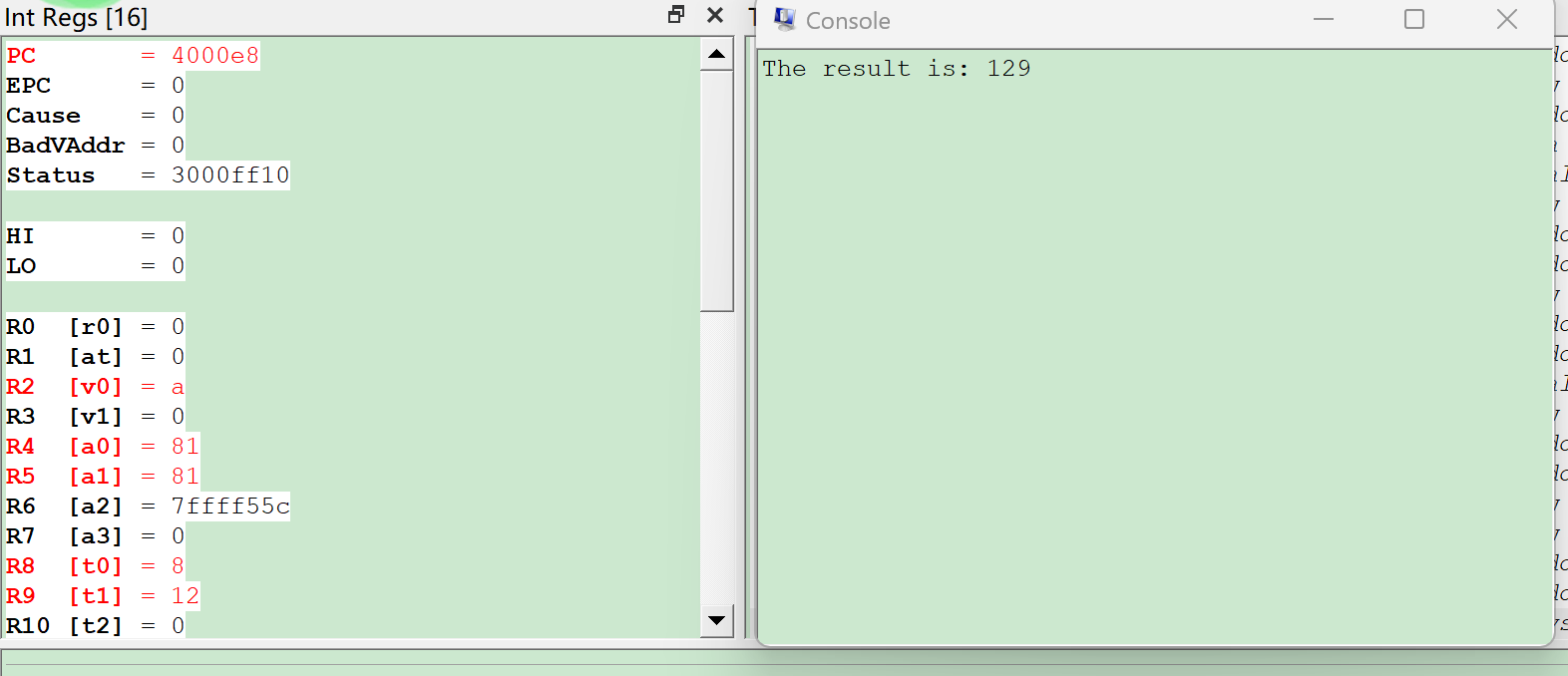
    addi $sp, $sp, 8

    # terminate the program

    addi $v0, $0, 10

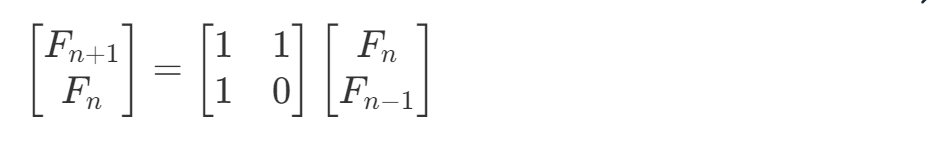
    syscall

A typical result should look like this: (S registers and $sp are not changed)



④ 汇编代码优化

Suppose is the nth element of Fibonacci series, then it can be computed by the following formula:



having . Then can be computed in ). Compared with original version whose run time is , this indeed has a better performance.

The ‘fib’ function looks like this(Helper function MatMul and MatSquare is added to appendix):

###################

# @brief calculate Fn(the nth element in Fibonacci.

# @param $a0: n.

###################

fib:

    # $s0: store A.

    # $s1: store x.

    # $s2: the result

    # $s3: the n.

    # ra

    addi $sp, $sp, -20

    sw $s0, 0($sp)

    sw $s1, 4($sp)

    sw $ra, 8($sp)

    sw $s2, 12($sp)

    sw $s3, 16($sp)

    addi $s3, $a0, -2  # $s3 = n - 2

    ble $s3, $0, end

    addi $t0, $0, 1  # one

    addi $sp, $sp, -16

    sw $t0, 0($sp)

    sw $t0, 4($sp)

    sw $t0, 8($sp)

    sw $0, 12($sp)

    add $s0, $sp, $0

    addi $sp, $sp, -8

    sw $t0, 0($sp)

    sw $t0, 4($sp)

    add $s1, $sp, $0

  loop:

    addi $t0, $0, 1

    and $t0, $s3, $t0

    srl $s3, $s3, 1

    beq $t0, $0, next

    add $a0, $s0, $0

    add $a1, $s1, $0

    jal MatMul

  next:

    add $a0, $s0, $0

    jal MatSquare

    bne $s3, $0, loop

  endloop:

    lw $a0, 0($s1)

    addi $sp, $sp, 24

    lw $s0, 0($sp)

    lw $s1, 4($sp)

    lw $ra, 8($sp)

    lw $s2, 12($sp)

    lw $s3, 16($sp)

    addi $sp, $sp, 20

    # return

    jr $ra

  end:

    # if n <= 2, return 1 immediately.

    lw $s0, 0($sp)

    lw $s1, 4($sp)

    lw $ra, 8($sp)

    lw $s2, 12($sp)

    lw $s3, 16($sp)

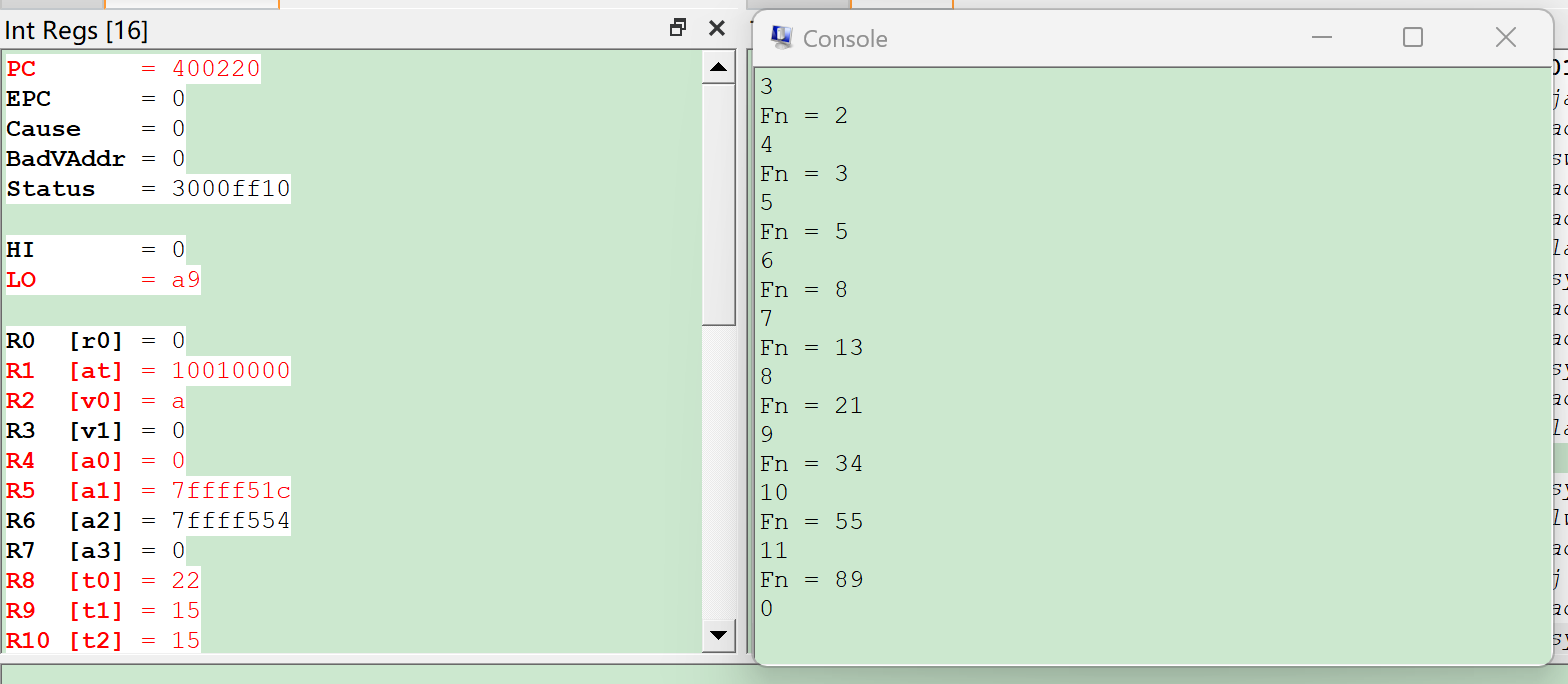
    addi $sp, $sp, 20

    addi $a0, $0, 1

    jr $ra

The main function takes an integer n once a time and output . If input is 0, terminate.

A typical output of this program:



1. **实验结论**

填写实验结论（可选，可以不填写）。

1. **实验感想**

对寄存器$0进行写操作不改变$0的值; 掌握了MIPS Assembly和QtSpim的使用;

**\*请大家不要改动字号、行间距等格式**

**Appendix(of all helper functions used in my implementation):**

###############

# print 'something' to output stream terminate current line.

# Usage:

# $a0 = 1: print integer(at $a1).

# $a0 = 4: print given string(at $a1)

###############

println:

    add $v0, $a0, $0

    add $a0, $a1, $0

    syscall

    # print '\n'.

    addi $v0, $0, 4

    la $a0, dashn    # load address

    syscall

    jr $ra

###############

# print 'something' to output stream.

# Usage:

# $a0 = 1: print integer(at $a1).

# $a0 = 4: print given string(at $a1)

###############

print:

    add $v0, $a0, $0

    add $a0, $a1, $0

    syscall

    jr $ra

###############

# read a integer and set to $a0.

###############

readint:

    addi $v0, $0, 5

    syscall

    add $a0, $v0, $0

    jr $ra

###################

# @brief calculate Ax, A is a (2,2) matrix, x is a 2-dim vector.

# @param $a0: matrix A.

# @param $a1: x[out].

# This is in-place, no return value.

###################

MatMul:

    addi $sp, $sp, -8

    sw $s0, 0($sp)

    sw $s1, 4($sp)

    add $s0, $0, $0

    add $s1, $0, $0

    # do $s0.

    lw $t0, 0($a1)  # x[0]

    lw $t1, 4($a1)  # x[1]

    lw $t2, 0($a0)  # A[0][0]

    lw $t3, 4($a0)  # A[0][1]

    mul $t2, $t2, $t0  # A[0][0] \* x[0]

    add $s0, $s0, $t2

    mul $t3, $t3, $t1  # A[0][1] \* x[1]

    add $s0, $s0, $t3

    # do $s1.

    lw $t2, 8($a0)   # A[1][0]

    lw $t3, 12($a0)  # A[1][1]

    mul $t2, $t2, $t0  # A[1][0] \* x[0]

    add $s1, $s1, $t2

    mul $t3, $t3, $t1  # A[1][1] \* x[1]

    add $s1, $s1, $t3

    # in-place malipulate x.

    sw $s0, 0($a1)

    sw $s1, 4($a1)

    # reset s and sp.

    lw $s0, 0($sp)

    lw $s1, 4($sp)

    addi $sp, $sp, 8

    jr $ra

###################

# @brief calculate A^2 in place.

# @param $a0: the (2, 2) matrix A.

# @returns none

###################

MatSquare:

    addi $sp, $sp, -16

    sw $s0, 0($sp)

    sw $s1, 4($sp)

    sw $s2, 8($sp)

    sw $s3, 12($sp)

    # initialize $s0-$s3.

    add $s0, $0, $0

    add $s1, $0, $0

    add $s2, $0, $0

    add $s3, $0, $0

    # load the matrix.

    lw $t0, 0($a0)

    lw $t1, 4($a0)

    lw $t2, 8($a0)

    lw $t3, 12($a0)  # don't touch these values later.

    # calculate new A[0][0]

    mul $t4, $t0, $t0

    mul $t5, $t1, $t2

    add $t4, $t4, $t5

    sw $t4, 0($a0)  # done

    # calculate new A[0][1]

    mul $t4, $t0, $t1

    mul $t5, $t1, $t3

    add $t4, $t4, $t5

    sw $t4, 4($a0)  # done

    # calculate new A[1][0]

    mul $t4, $t2, $t0

    mul $t5, $t3, $t2

    add $t4, $t4, $t5

    sw $t4, 8($a0)  # done

    # calculate new A[1][1]

    mul $t4, $t2, $t1

    mul $t5, $t3, $t3

    add $t4, $t4, $t5

    sw $t4, 12($a0)  # done

    lw $s0, 0($sp)

    lw $s1, 4($sp)

    lw $s2, 8($sp)

    lw $s3, 12($sp)

    addi $sp, $sp, 16

    # return

    jr $ra