# 计算机组成原理实验报告

姓名： 张佳晨　　　　　　学号：11713020

## 一、 实验目的

1. Learn about the implementation of iteration and recursion by MIPS.
2. Learn about the implementation of procedures.
3. Know how to using stack and heap.

## 二、 实验内容

1. Implemented the Fibolacci number by loop and recursive methods respectively, count the the number of instructions required by fib(5).
2. write and test a program that reads in a positive integer using the SPIM system calls. If the integer is not positive, the program should terminate with the message “Invalid Entry”; otherwise the program should print out the names of the digits of the integers, delimited by exactly one space. For example, if the user entered “728,” the output would be “Seven Two Eight.”
3. Write and test a MIPS assembly language program to compute and print the first 100 prime numbers. A number n is prime if no numbers except 1 and n divide it evenly. You should implement two routines:
    1. test_prime (n) Return 1 if n is prime and 0 if n is not prime.
    2. main () Iterate over the integers, testing if each is prime. Print the first 100 numbers that are prime.

## 三、 实验步骤（阐述代码思路或操作步骤）

### Problem 1.1

> compute fib($s0)
>
> fib(0) = 0; fib(1) = 1; and then using iteration to compute fib(x)

```
.text
.global main

main:
    li      $s0, 5
    li      $a0, 0
    li      $a1, 1
    jal     fib             # jump to fib and save position to $ra
    add     $a0, $v0, $zero
    li      $v0, 1
    syscall
    li      $v0, 10
    syscall
    fib:
        li      $t0, 0
        beq     $t0, $s0, fib_zero  # if $t0 == $s0 then target
        addi    $t0, $t0, 1         # $t0 = $t0 + 1
        beq     $t0, $s0, fib_one   # if $t0 == $s0 then target
        loop:
            addi    $t0, $t0, 1         # $t0 = $t0 + 1
            add     $v0, $a0, $a1       # $v0 = $a0+ $a1
            beq     $t0, $s0, end_loop
            add     $a0, $a1, $zero
            add     $a1, $v0, $zero
            j       loop                # jump to loop
        end_loop:
            jr      $ra                 # jump to $ra
        fib_zero:
        li      $v0, 0
        jr      $ra                 # jump to $ra
        fib_one:
        li      $v0, 1
        jr      $ra                 # jump to $ra
```

## Problem 1.2

> Using recursion to compute fib($a0)

```
# int fib(int n){
#   if (n == 0)
#       return 0;
#   else if (n == 1)
#        return 1;
#   else
#       x = fib(n-1);
#       y = fib(n-2);
#       return x + y;
# }

    .text
    .global main
    main:
    li      $a0, 5      # $a0 = 5
    jal     fib
    add     $a0, $v0, $zero
    li      $v0,    1
    syscall
    li      $v0,    10
    syscall

    fib:
        addi    $sp, $sp, -8        # malloc the space of 2 word    $sp = $sp + -8
        sw      $ra, 4($sp)         # save the return address onto the stack
        sw      $a0, 0($sp)         # save the argument n onto the stack

        slti    $t0, $a0, 2                # test for n < 2
        beq     $t0, $zero, LargeThan_1     # if n >= 2 then goto L1
        addi    $sp, $sp, 8
        # n <= 1
        slti    $t0, $a0, 1                # test for n < 1
        beq     $t0, $zero, Equal_1     # if n >= 1 then goto L1
        # n = 0
            addi    $v0, $v0, 0     # return 0
            jr      $ra                 # return to the callerl
        Equal_1:
            addi    $v0, $v0, 1     # return 1
            jr      $ra                 # return to the callerl
        LargeThan_1: # n >= 2
                    # malloc the space of 2 word    $sp = $sp + -8
            addi    $a0, $a0, -1            # $a0 = n - 1
            addi    $sp, $sp, -4
            sw      $a0, 0($sp)

            jal     fib                 # jump to fact and save position to $ra

            lw      $a0, 0($sp)         # $a0 = n - 1

            addi    $a0, $a0, -1            # $a0 = n - 1
            addi    $sp, $sp,  4
            jal     fib             # jump to fact and save position to $ra

            add    $v1, $v0, $v1     # $v1 = fib(n-2)
            lw      $a0, 0($sp)
            lw      $ra, 4($sp)             # restore the return address
            addi    $sp, $sp, 8
            jr      $ra                     # return to the caller
```

## Problem 2

> Using div to get the reminder and the quotation.
>
> Using recursion to print the Highest position of the number.
>
> Using branch to implement a structure like `switch` to print words.

```
 .macro end
    li $v0,10
    syscall
.end_macro
.macro print_string(%str)
    .data
    pstr: .asciiz %str
    .text
    la $a0,pstr
    li $v0,4
    syscall
```

```
.end_macro
    .text
    main:
        print_string("\n> Please input a positive integer and press ENTER:")
        li      $v0, 5              # $v0 = 5 syscall to read an integer and store it in
$v0
        syscall
        slt     $t0, $zero, $v0     # $t0 = $v0 > 0 ? 1 : 0
        beq     $t0, $zero, Invalid_Entry_End   # if $v0 <= 0 then goto Invalid_Entry_End
        addi    $a0, $v0, 0         # $a0 = $v0 + 0
        jal     print_digits                # jump to print_digits and save position to $ra
        end

        Invalid_Entry_End:
            print_string("\nInvalid Entry\nSystem Exits")
            end
    print_digits:
        addi    $sp, $sp, -16           # $sp = $sp + 16
        sw      $s0, 12($sp)        #
        sw      $s1, 8($sp)
        sw      $s2, 4($sp)
        sw      $s3, 0($sp)

        addi    $s0, $a0, 0             # $t0 = $a0 + 0
        addi    $s1, $zero, 10          # $t0 = $a0 + 0

        Loop:
            div     $s0, $s1            # Lo = $s0 / 10  Hi = $s0 % 10
            mflo    $s0
            mfhi    $a0
            addi    $sp, $sp, -8
            sw      $ra, 4($sp)         # save the return address onto the stack
            sw      $a0, 0($sp)         # save the argument n onto the stack
            bne     $s0, $zero, L1  # if $s0 != $zero then target
                addi    $sp, $sp, -4
                sw  $ra, 0($sp)
                jal     print_digit_name
                lw  $ra, 0($sp)
                addi    $sp, $sp, 12            # $sp = $sp + 8
                jr      $ra                 # return to the callerl
            L1:
                jal     Loop                # jump to Loop and save position to $ra
                lw      $a0, 0($sp)     #
                addi    $sp, $sp, 4
                jal     print_digit_name
                lw      $ra, ($sp)             # restore the return address
                addi    $sp, $sp, 4             # adjust stack pointer to pop 2 items
                jr      $ra                 # jump to $ra

        jr      $ra                 # jump to $ra

    print_digit_name:
        li      $t0, 0
        beq     $a0, $t0, Zero
        addi    $t0, $t0, 1         # $t0 = $t1 + 1
        beq     $a0, $t0, One
        addi    $t0, $t0, 1         # $t0 = $t1 + 1
        beq     $a0, $t0, Two
        addi    $t0, $t0, 1         # $t0 = $t1 + 1
        beq     $a0, $t0, Three
        addi    $t0, $t0, 1         # $t0 = $t1 + 1
        beq     $a0, $t0, Four
        addi    $t0, $t0, 1         # $t0 = $t1 + 1
        beq     $a0, $t0, Five
        addi    $t0, $t0, 1         # $t0 = $t1 + 1
        beq     $a0, $t0, Six
        addi    $t0, $t0, 1         # $t0 = $t1 + 1
        beq     $a0, $t0, Seven
        addi    $t0, $t0, 1         # $t0 = $t1 + 1
        beq     $a0, $t0, Eight
        addi    $t0, $t0, 1         # $t0 = $t1 + 1
        beq     $a0, $t0, Nine

        Nine:
        print_string("Nine ")
        jr      $ra                 # jump to $ra
        Eight:
        print_string("Eight ")
        jr      $ra                 # jump to $ra
        Seven:
        print_string("Seven ")
        jr      $ra                 # jump to $ra
        Six:
        print_string("Six ")
        jr      $ra                 # jump to $ra
        Five:
        print_string("Five ")
        jr      $ra                 # jump to $ra
        Four:
        print_string("Four ")
        jr      $ra                 # jump to $ra
        Three:
        print_string("Three ")
        jr      $ra                 # jump to $ra
        Two:
        print_string("Two ")
        jr      $ra                 # jump to $ra
        One:
        print_string("One ")
        jr      $ra                 # jump to $ra
        Zero:
        print_string("Zero ")
        jr      $ra                 # jump to $ra
```

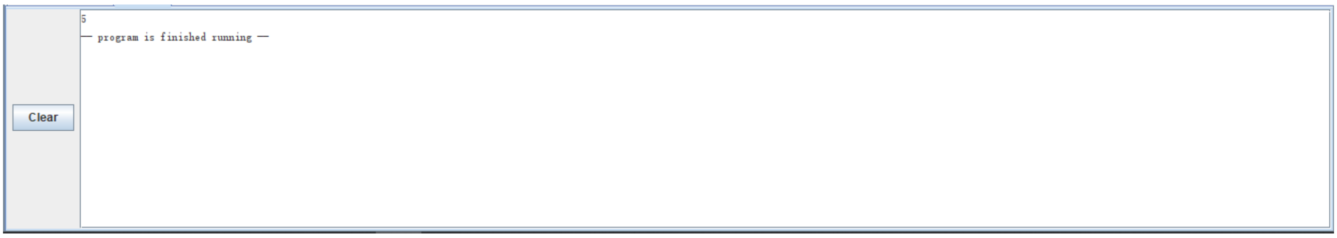
## Problem 3

> Using `main` to iterator over the integers and record the number of prime number it has printed.
>
> Using `test_prime` to check if a given number is prime. Return 1 if n is prime and 0 if n is not prime.

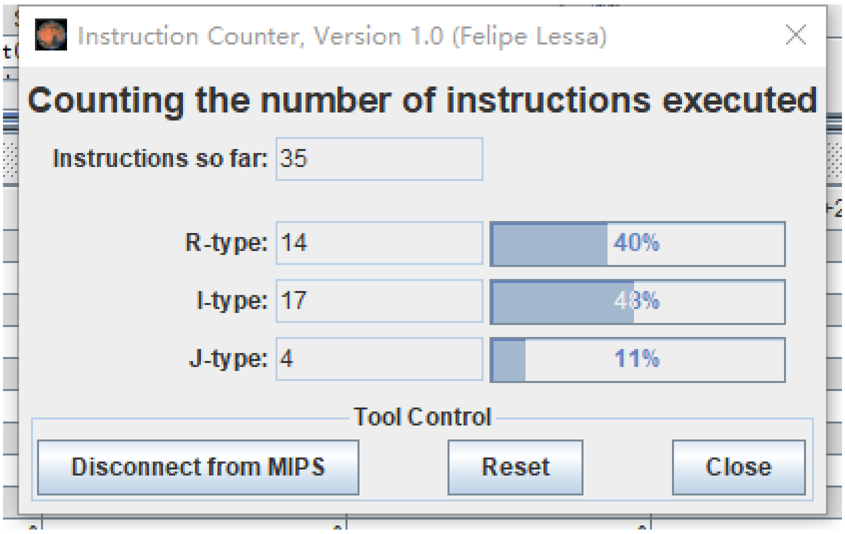
```
 .macro end
    li $v0,10
    syscall
.end_macro
.macro print_string(%str)
    .data
    pstr: .asciiz %str
    .text

    la $a0,pstr
    li $v0,4
    syscall

.end_macro
.text
.globl main

    main:
        move    $a0, $zero
        li      $s0, 100        # $s0 = 100
        li      $s1, 0

        Loop:
            addi    $a0, $a0, 1         # $a0 = $a1 + 1
            beq     $s1, $s0, EndLoop
            jal     test_prime
            beq     $v0, $zero, Loop
            addi    $s1, $s1, 1
            li      $v0, 1              # print $a0
            syscall
            move $t0, $a0
            print_string("\n")
            move $a0, $t0
            j   Loop
        EndLoop:
        end

    test_prime:
        addi    $t1, $zero, 1
        slt     $t0, $t1, $a0         # $t0 = $a0 > 1 ? 1 : 0
        beq     $t0, $zero, NotPrime    # if $a0 <= 1 then target
        addi    $t0, $zero, 1           # $t0 =$zero1 + 0
        Loop_test_prime:
            addi    $t0, $t0, 1
            beq     $t0, $a0, isPrime   # if $t0 == $a0 then isPrime
            div     $a0, $t0            # $a0 / $t0
            mfhi    $t1                 # $t3 = $a0 mod $t0
            beq     $t1, $zero, NotPrime
            j       Loop_test_prime
        isPrime:
        li      $v0, 1
        jr      $ra                 # jump to $ra
        NotPrime:
        li      $v0, 0
        jr      $ra                 # jump to $ra
```

# 四、 实验结果（截图并配以适当的文字说明）

## Problem 1.1



```
5
— program is finished running —
```


Instruction Counter, Version 1.0 (Felipe Lessa)
Counting the number of instructions executed
Tool Control
Disconnect from MIPS
Reset
Close

Instructions so far: 35

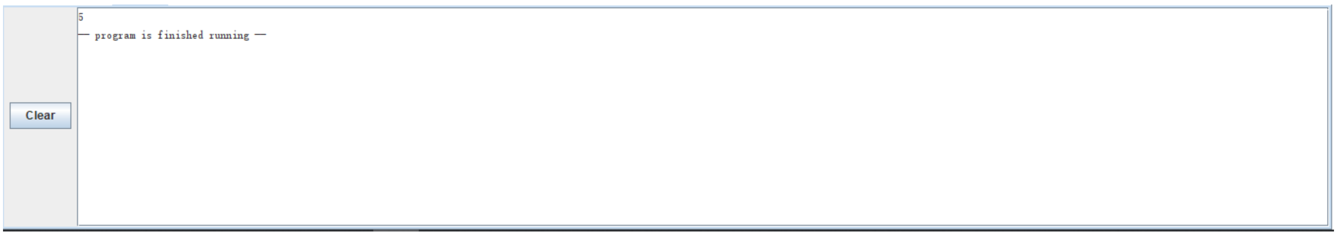
R-type: 14 40%

I-type: 17 48%

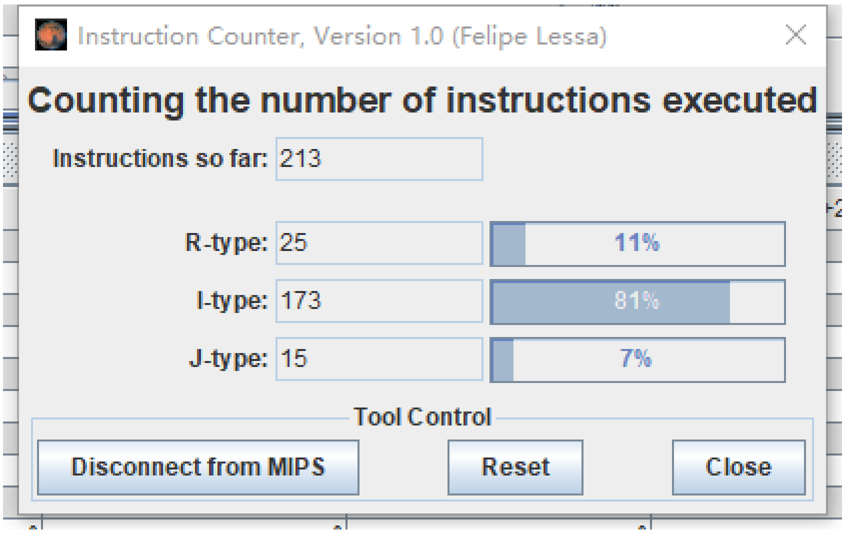
J-type: 4 11%

> Fib(5) = 5
>
> Using `MARS` > `Tools` > `Instruction Counter`, click `Connect to MIPS` and execute the code.
>
> The total instruction for computing `fib(5)` using iteration is 35.
>
> PS. The number of instructions executed is the number of basic instructions, some of which is are split from complex instructions I wrote.

## Problem 1.2



```
5
— program is finished running —
```



Counting the number of instructions executed

Instructions so far: 213

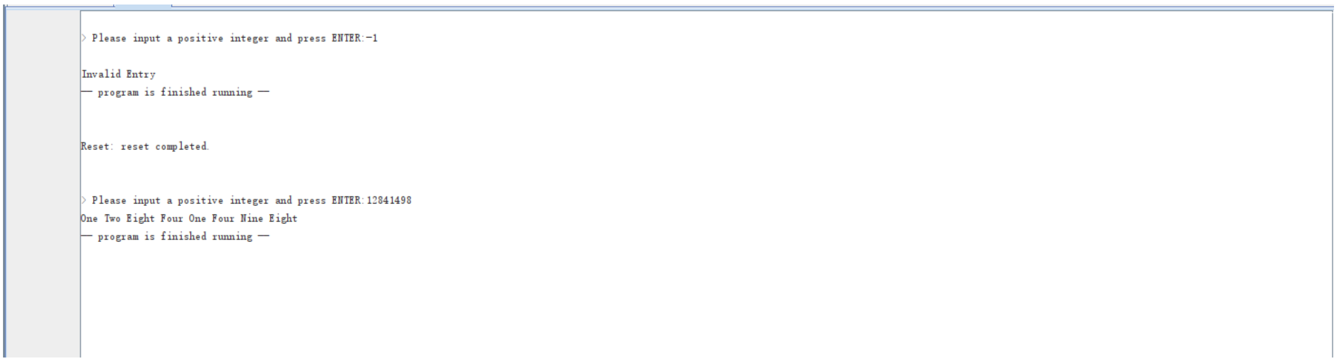
R-type: 25 11%

I-type: 173 81%

J-type: 15 7%

> Fib(5) = 5
>
> Using `MARS` > `Tools` > `Instruction Counter`, click `Connect to MIPS` and execute the code.
>
> The total instruction for computing `fib(5)` using recursion is 213.
>
> PS. The number of instructions executed is the number of basic instructions, some of which is are split from complex instructions I wrote.

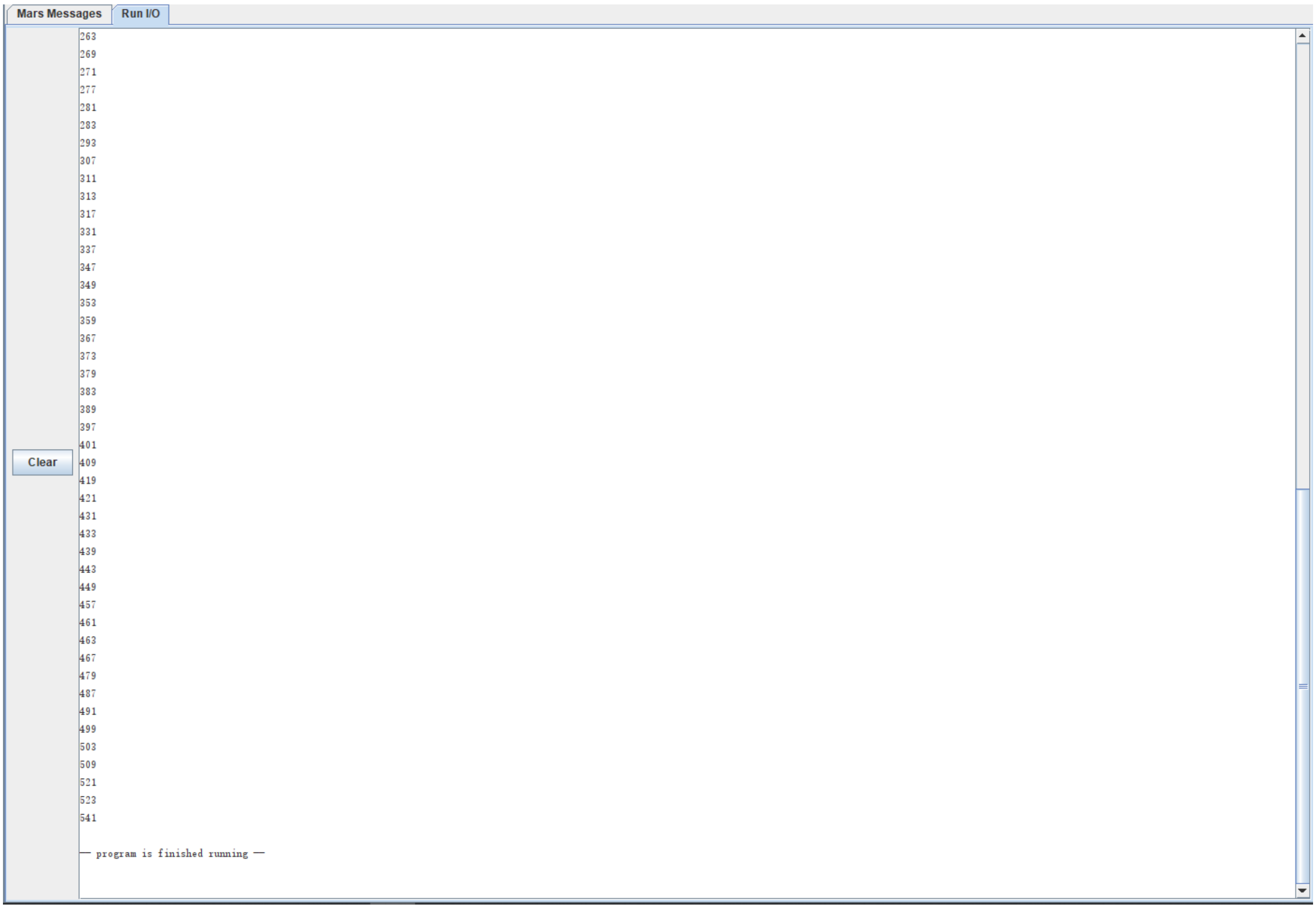
## Problem 2



```
> Please input a positive integer and press ENTER:-1

Invalid Entry
-- program is finished running --


Reset: reset completed.


> Please input a positive integer and press ENTER:12841498
One Two Eight Four One Four Nine Eight
-- program is finished running --
```

> When my input is `-1`, the program should terminate with the message “Invalid Entry”.
>
> When my input is `12841498`, the program print `One Two Eight Four One Four Nine Eight`.

## Problem 3



```
263
269
271
277
281
283
293
307
311
313
317
331
337
347
349
353
359
367
373
379
383
389
397
401
409
419
421
431
433
439
443
449
457
461
463
467
479
487
491
499
503
509
521
523
541

— program is finished running —
```

> The program outputs the first 100 prime numbers, in which the 100th prime number is 541.

## 五、 实验分析（遇到的问题以及解决方案）

> Not yet

## 六、 实验小结与体会

1. Recursion is much slower than iteration and it needs more memory addressing and more stack memory.
2. It will be better that using more iteration than recursion.