



# 《计算机组成原理与接口技术实验》 实验报告

(实验一)

学 院 名 称 : 数据科学与计算机学院

学 生 姓 名 : 陈亚楠

学 号 : 16340041

专业(班级) : 16 软件工程一(1) 班

时 间 : 2018 年 4 月 3 日

---

成绩：

---

## 实验一：MIPS汇编语言程序设计

---

### 一. 实验目的

1. 认识和掌握MIPS汇编语言程序设计的基本方法；
2. 熟悉 PCSpim 模拟器的使用。

### 二. 实验内容

编写一个程序：先从键盘输入一个字符串（有英文字母，可能也有数字），然后显示其中数字的个数、英文字母的个数和字符串的长度；字符串中不能有空格，若有将其删除，并将改变后的字符串按相反的顺序显示出来；输入第二个字符串，然后将输入的字符串与前面处理后的字符串比较是否相同，若相同，输出“Password Right!”，否则输出“Password Error!”。

### 三. 实验器材

电脑一台、PCSpim 模拟器软件一套。

### 四. 实验分析与设计

1. 关于本次实验的部分说明：

- (1) 本次实验中的所有字符串长度为50；
- (2) 存放结果的寄存器说明：

\$s0: 存放第一个输入的字符串的首地址；

\$s1: 存放原字符串中数字的个数；

\$s2: 存放原字符串中字母的个数；

\$s3: 存放原字符串的长度，包含空格；

\$s4: 存放字符串中空格的个数；

\$s5: 存放第二个输入的字符串的首地址；

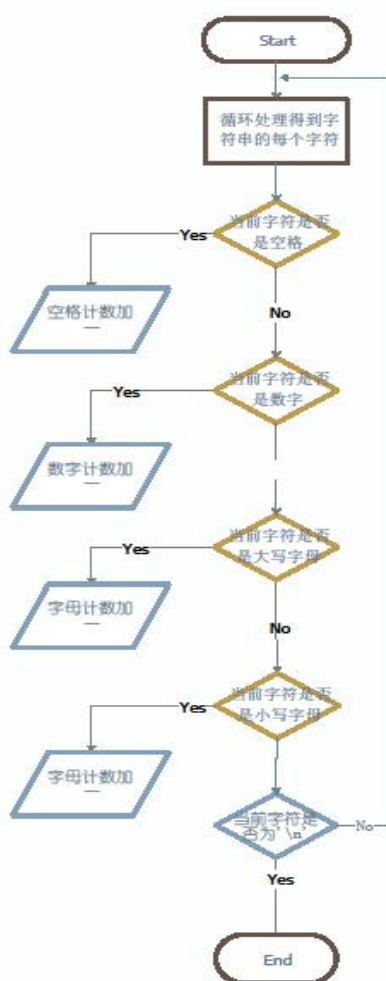
\$s6: 存放处理后的原字符串的长度，不包含空格。

2. 本次汇编语言程序设计实验整体采用自顶向下、逐步求精的模块化的设计思想，将整个实验的主要任务分为三部分：



### 3. 三个主要任务的实现：

#### (1) 统计输入字符串中的空格、数字、字母的数量：



本次实验中，将统计字母的数量问题转换为统计大写字母的数量与统计小写字母的数量两个问题，避免了分支指令的过多嵌套，是代码的可读性降低。

#### (2) 逆序输出字符串：

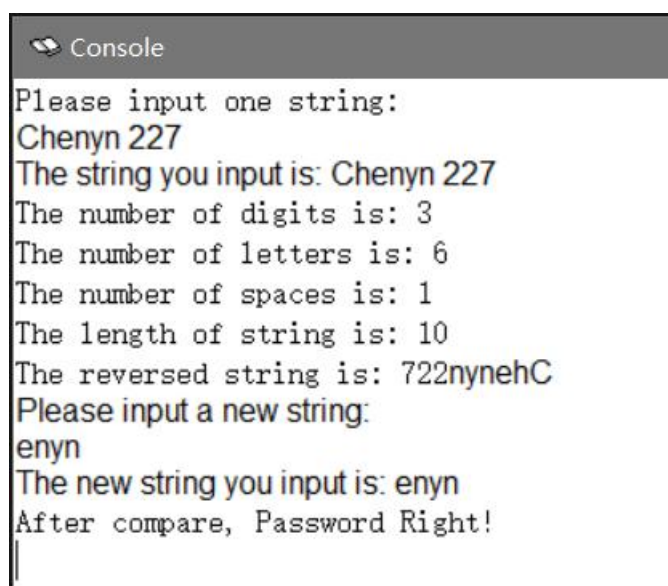
通过使用循环把一个临时寄存器作为指针指向字符串的末尾，同时需要一个临时寄存器

作为计数器进项正向与反向的计数。反向时，碰到非空格的字符输出这个字符，逐渐形成一个字符串。

(3) 比较原字符串与新输入的字符串是否相同：

遍历新输入的字符串，当遍历到'\n'的时候退出循环。控制遍历原字符串不会超出访问范围，里面嵌套循环找到下一个非空格的字符。将两个字符相比较，如果不同则这两个字符串不一样。如果当第二个字符串遍历到'\n'的时候未出现错误，则有两种情况：新输入的字符串与原字符串相同，或者新输入的字符串是原字符串的子串。

4. 实验结果：



```
Console
Please input one string:
Chenyn 227
The string you input is: Chenyn 227
The number of digits is: 3
The number of letters is: 6
The number of spaces is: 1
The length of string is: 10
The reversed string is: 722nynehC
Please input a new string:
enyn
The new string you input is: enyn
After compare, Password Right!
```

## 五. 实验心得

1. 代码重构：刚开始写代码的时候，在主函数里专门写了计算字符串长度的代码，后来发现，字符串的长度可以直接由数字的个数、字母的个数、空格的个数相加处理得到；

2. 问题解决：在通过ASCII码来判断一个字符是否是字母的时候，使用了过多的IF嵌套，致使程序运行到判断字符是否为字母的时候子程序无法返回到主函数：

```

# judge if less than or equal to 'z'
bleu $s7, 0x7a, ifLowerGreater
jr $ra

# judge if greater than or equal to 'a'
ifLowerGreater:
    bgeu $s7, 0x61, letterCount
    jal ifUpperLess
    jr $ra

# judge if less than or equal to 'Z'
ifUpperLess:
    bleu $s7, 0x5a, ifUpperGreater
    jr $ra

# judge if greater than or equal to 'A'
ifUpperGreater:
    bgeu $s7, 0x41, letterCount
    jr $ra

letterCount:
    addi $s2, $s2, 1
    jr $ra

```

```

# judge if lowerletter
isLowerLetter:
    la $a0, inLowerLetter
    li $v0, 4
    syscall

# judge if less than or equal to 'z'
bleu $s7, 0x7a, ifLowerGreater
jr $ra

# judge if greater than or equal to 'a'
ifLowerGreater:
    bgeu $s7, 0x61, letterCount
    jr $ra

# judge if upperletter
isUpperLetter:
    la $a0, inUpperLetter
    li $v0, 4
    syscall

# judge if less than or equal to 'Z'
ifUpperLess:
    bleu $s7, 0x5a, ifUpperGreater
    jr $ra

# judge if greater than or equal to 'A'
ifUpperGreater:
    bgeu $s7, 0x41, letterCount
    jr $ra

```

这是因为jr语句是使得当前子程序跳转到相对于自己的上一级的寄存器的内容为地址的指令，由于上面的程序嵌套了多个子程序，jr语句返回的情况也不都相同。因此，将该问题转换为统计大写字母的数量与统计小写字母的数量两个问题，避免了嵌套，代码可读性也提高了。

### 3. 个人体会：

（1）注释：MIPS汇编语言程序设计中，涉及到了很多的寄存器，因此在代码编写过程中及时注释当前使用的寄存器的作用是非常必要的；

（2）代码风格（最重要的体会）：在统计字符数据信息的问题中，一开始使用的是顺序的结构，即如果判断了当前字符是空格，就直接跳回循环开始，否则进入判断是否是字母的语句，同理数字的判断也是。尽管这样写可以避免很多无用的判断，提高运行的效率，但是就我个人实际而言，由于MIPS指令的简单性会导致代码的增加，也使得自己容易逻辑混乱。后来就直接把用三个jal语句来分级模块处理，提高了代码的可读性，降低了维护成本。

（3）设计方法：MIPS汇编语言程序设计与C语言程序设计都采用了自顶向下、逐步求精的模块化的方法，将大的问题转换为几个小的问题，层次清晰，效率更高。

### 【程序代码】

.data

# string inputed by user

```
# string length less than 50

string: .space 50

newString: .space 50


# prompt message

inputStringMessage: .asciiz "Please input one string:\n"
confirmInputMessage: .asciiz "The string you input is: "
digitNumberMessage: .asciiz "The number of digits is: "
letterNumberMessage: .asciiz "The number of letters is: "
stringLengthMessage: .asciiz "The length of string is: "
spaceNumberMessage: .asciiz "The number of spaces is: "
reverseStringMessage: .asciiz "The reversed string is: "
inputNewStringMessage: .asciiz "Please input a new string:\n"
confirmNewStringMessage: .asciiz "The new string you input is: "
compareMessage: .asciiz "After compare, "
compareSuccessMessage: .asciiz "Password Right!\n"
compareFailMessage: .asciiz "Password Error!\n"
newline: .asciiz "\n"


# debug message

inLoop: .asciiz "execute in loop\n"
inDigit: .asciiz "execute in judge digit\n"
inLowerLetter: .asciiz "execute in judge lowerletter\n"
inUpperLetter: .asciiz "execute in judge upperletter\n"
inSpace: .asciiz "execute in judge space\n"
inNewLine: .asciiz "execute in judge newline\n"
inReverse: .asciiz "execute in reverse loop\n"


.text

.globl main
```

```
main:

    # print input string message
    la $a0, inputStringMessage
    li $v0, 4
    syscall

    # read string
    la $a0, string
    li $v0, 8
    syscall

    # print confirm input message
    la $a0, confirmInputMessage
    li $v0, 4
    syscall

    # print input string
    la $a0, string
    li $v0, 4
    syscall

    # print a newline
    #la $a0, newline
    #li $v0, 4
    #syscall

    # init counter
    # $s0 store the original string
    # $s1 store the number of digit
    # $s2 store the number of letter
```

```
# $s3 store the length of original string
# $s4 store the number of space
la $s0, string
move $s1, $zero
move $s2, $zero
move $s3, $zero
move $s4, $zero

#get the number of digit, letter, space, length

# $s7 store the char of string in every loop
loop:
    # la $a0, inLoop
    # li $v0, 4
    # syscall

    lb $s7, 0($s0)
    # judge if space, digit, letter
    jal isSpace
    jal isDigit
    jal isLowerLetter
    jal isUpperLetter

    addi $s0, $s0, 1
    # judge if string over
    # la $a0, inNewLine
    # li $v0, 4
    # syscall
    bne $s7, 0x0a, loop
    move $ra, $zero
```



```
# length handle
add $t0, $s1, $s2
add $s3, $t0, $s4

jal printDatas

jal reverseString

la $a0, newline
li $v0, 4
syscall

#input a new string
la $a0, inputNewStringMessage
li $v0, 4
syscall

# read new string
la $a0, newString
li $v0, 8
syscall

# print confirm new input message
la $a0, confirmNewStringMessage
li $v0, 4
syscall

# print new string
la $a0, newString
```

```
li $v0, 4
syscall

# compare string and newString
j compareFuntion

# judge if space
isSpace:
    # la $a0, inSpace
    # li $v0, 4
    # syscall
    beq $s7, 0x20, spaceCount
    jr $ra

spaceCount:
    addi $s4, $s4, 1
    jr $ra

# judge if digit
isDigit:
    # la $a0, inDigit
    # li $v0, 4
    # syscall
    # judge if less than or equal to '9'
    bleu $s7, 0x39, ifGreater
    jr $ra

# judge if greater than or equal to '0'
ifGreater:
```

```
    bgeu $s7, 0x30, digitCount
    jr $ra
```

```
digitCount:
```

```
    addi $s1, $s1, 1
    jr $ra
```

```
# judge if lowerletter
```

```
isLowerLetter:
```

```
    # la $a0, inLowerLetter
    # li $v0, 4
    # syscall
```

```
# judge if less than or equal to 'z'
```

```
    bleu $s7, 0x7a, ifLowerGreater
    jr $ra
```

```
# judge if greater than or equal to 'a'
```

```
ifLowerGreater:
```

```
    bgeu $s7, 0x61, letterCount
    jr $ra
```

```
# judge if upperletter
```

```
isUpperLetter:
```

```
    # la $a0, inUpperLetter
    # li $v0, 4
    # syscall
```

```
# judge if less than or equal to 'Z'
```

```
ifUpperLess:
```

```
        bleu $s7, 0x5a, ifUpperGreater
        jr $ra

# judge if greater than or equal to 'A'
ifUpperGreater:
        bgeu $s7, 0x41, letterCount
        jr $ra

letterCount:
        addi $s2, $s2, 1
        jr $ra

#print the number of digit, letter, space, length message
printDatas:
        la $a0, digitNumberMessage
        li $v0, 4
        syscall
        li $v0, 1
        la $a0, 0($s1)
        syscall
        la $a0, newline
        li $v0, 4
        syscall

        la $a0, letterNumberMessage
        li $v0, 4
        syscall
        li $v0, 1
        la $a0, 0($s2)
        syscall
```

```
la $a0, newline
```

```
li $v0, 4
```

```
syscall
```

```
la $a0, spaceNumberMessage
```

```
li $v0, 4
```

```
syscall
```

```
li $v0, 1
```

```
la $a0, 0($s4)
```

```
syscall
```

```
la $a0, newline
```

```
li $v0, 4
```

```
syscall
```

```
la $a0, stringLengthMessage
```

```
li $v0, 4
```

```
syscall
```

```
li $v0, 1
```

```
la $a0, 0($s3)
```

```
syscall
```

```
la $a0, newline
```

```
li $v0, 4
```

```
syscall
```

```
jr $ra
```

```
reverseString:
```

```
# print prompt message
```

```
la $a0, reverseStringMessage
```

```
li $v0, 4
```

```
syscall
```

```
# $t1 counter
```

```
# $t2 as a pointer to the end of string gradually
```

```
move $t1, $zero
```

```
la $t2, string
```

```
# a loop move $t2 to the end of string
```

```
toEndLoop:
```

```
    addi $t1, $t1, 1
```

```
    addi $t2, $t2, 1
```

```
    bne $t1, $s3, toEndLoop
```

```
    move $t1, $zero
```

```
# reverse and printf char
```

```
reverseLoop:
```

```
    addi $t1, $t1, 1
```

```
    addi $t2, $t2, -1
```

```
    lb $t3, 0($t2)
```

```
    beq $t3, 0x20, reverseLoop
```

```
    #print the char
```

```
    li $v0, 11
```

```
    la $a0, 0($t3)
```

```
    syscall
```

```
    #judge string if over
```

```
    bne $t1, $s3, reverseLoop
```

```
    jr $ra
```

```
#compare string and newString
```

```
compareFuntion:
```

```
la $a0, compareMessage
li $v0, 4
syscall

# $s0 store the original string
# $s5 store the newstring
la $s0, string
la $s5, newstring

move $t4, $zero

# $s6 store the length of original string without space
add $s6, $s1, $s2

loopA:
    lb $t5, 0($s5)
    loopB:
        lb $t6, 0($s0)
        bne $t6, 0x20, continue
        addi $s0, $s0, 1
        j loopB

    continue:
        bne $t5, $t6, compareFail
        beq $t5, 0x0a, isSubstring
        beq $t6, 0x0a, compareFail
        addi $s0, $s0, 1
        addi $s5, $s5, 1
        addi $t4, $t4, 1
        j loopA
```

compareFail:

la \$a0, compareFailMessage

li \$v0, 4

syscall

j exitProgram

compareSuccess:

la \$a0, compareSuccessMessage

li \$v0, 4

syscall

j exitProgram

isSubstring:

beq \$s6, \$t4, compareSuccess

j compareFail

exitProgram:

li \$v0, 10

syscall