



《计算机组成原理与接口技术实验》 实验报告

学院名称：数据科学与计算机学院

学生姓名：罗剑杰

学号：15331229

专业（班级）：15 软件工程三（6）班

合作者：无

时间：2017 年 3 月 24 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!”。

简要实验原理：通过字符串的读入，正向和反向迭代遍历字符串，流程控制来实现实验的目的，详细设计在第四部分

三. 实验器材

PC机一台，PCSpim 模拟器软件一套。。

四. 实验分析与设计

- a) 使用自顶向下的设计方法，先考虑宏观任务，总体的宏观流程图：



本次实验的全局变量寄存器的使用：

\$s0: 存第二个字符串的首地址

\$s1: 存第一个字符串的首地址

\$s2: 在统计第一个字符串的信息时用来存每一次迭代需要检查的字符

\$s3: 存第一个字符串除去了不合法的空格后的长度（不包括'\n'）

\$s4: 存第一个字符串原始的长度（不包括'\n'）

\$s5: 存第一个字符串的空格的个数

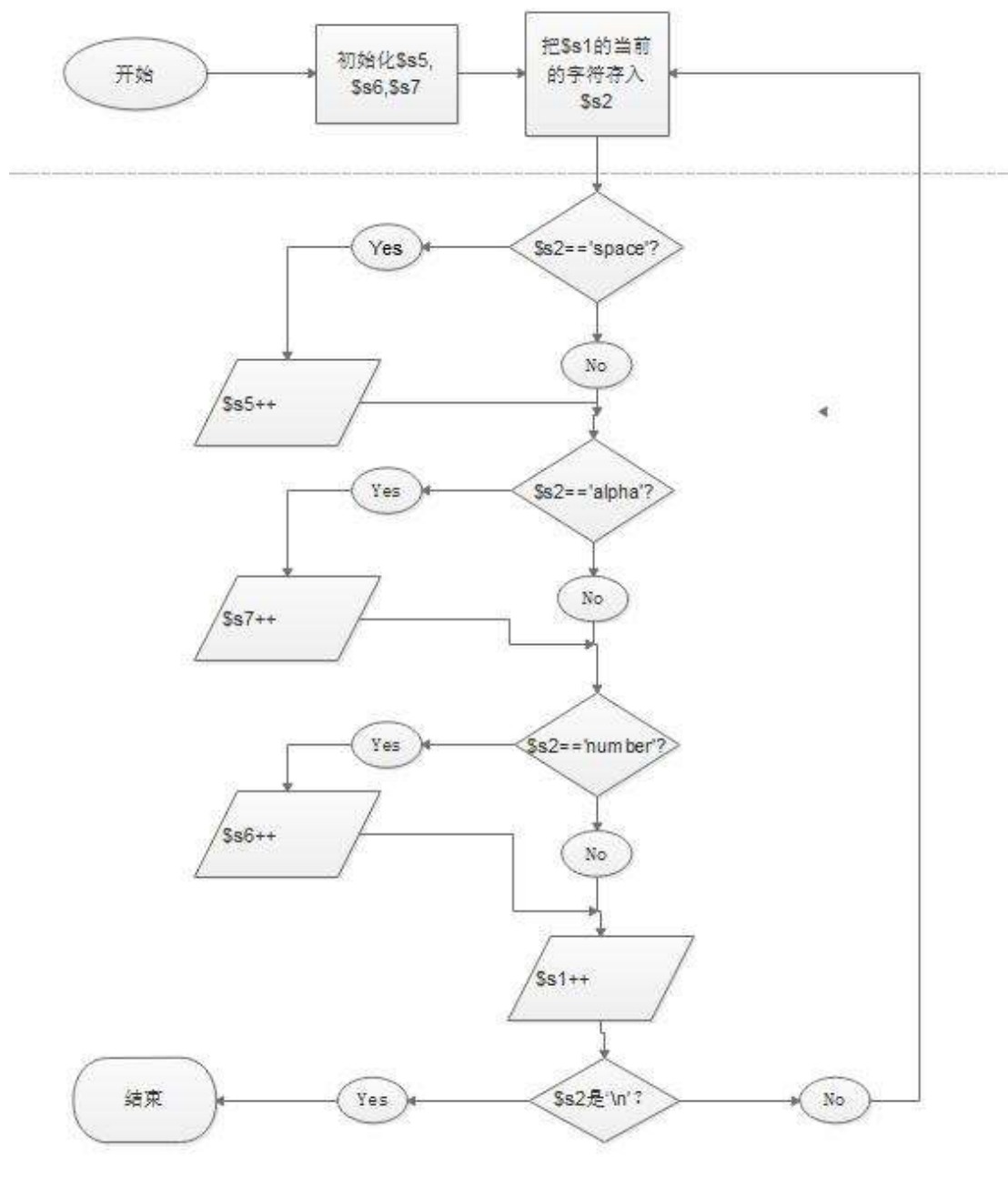
\$s6: 存第一个字符串的数字的个数

\$s7: 存第一个字符串的字母的个数

b) 1.读入字符串的操作

i. 申请好内存（在本次实验中固定字符串长度为60）使用调用号为8的调用就好

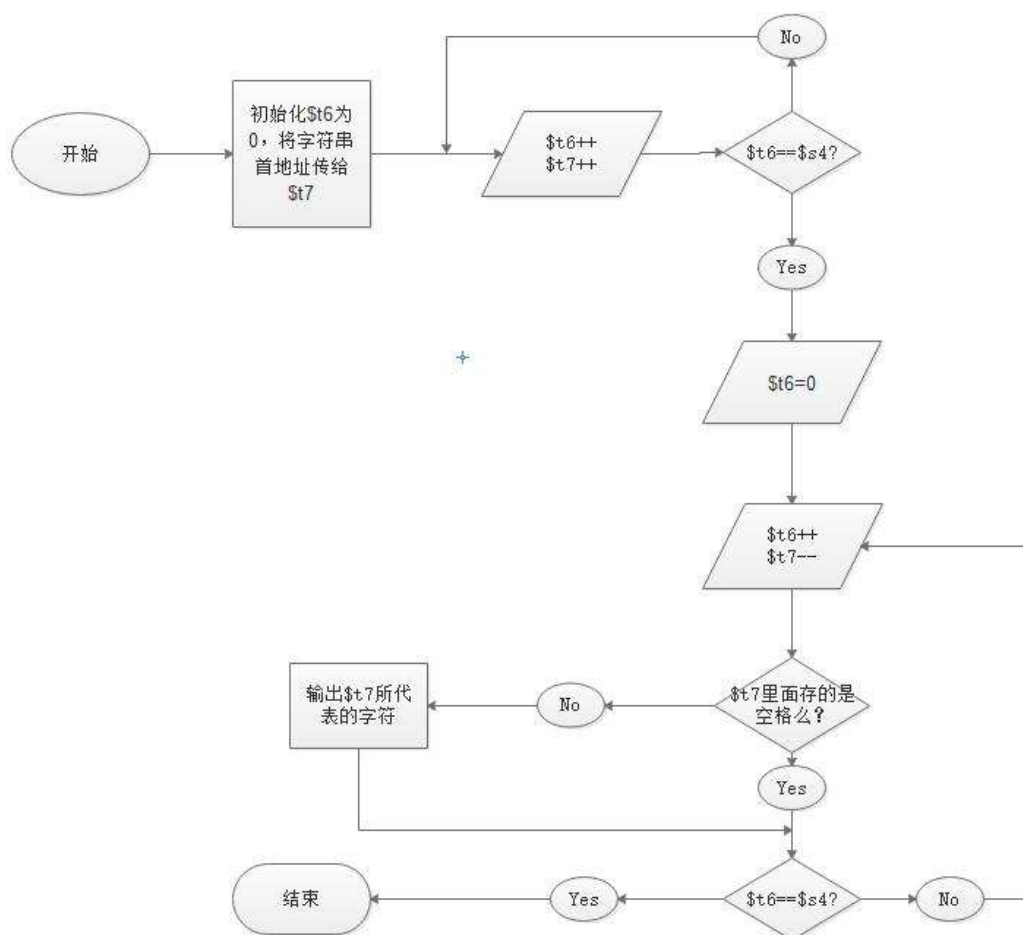
c) 2.统计字符串字符个数的相关信息



思路：分模块来进行比较，虽然这样子设计会有出现“判断了字符是空格后还是会继续判断字符是不是字母还有判断字符是不是数字”的情况，即可能出现无用的比较，但是这样写的好处是代码风格非常良好，而且分模块易于调试，流程看上去非常直观，所以选择这种方式。

d) 3. 反向迭代显示字符串的实现

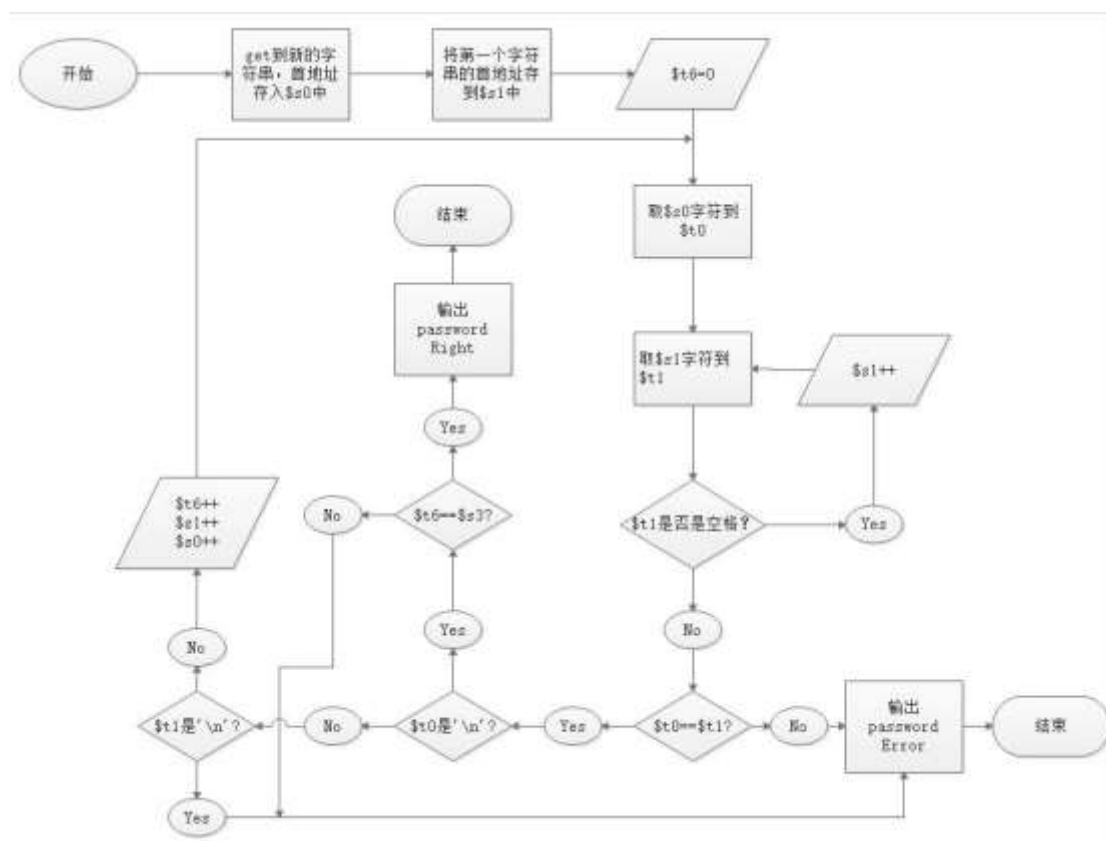
主要思路：通过第二步统计好的\$s5,\$s6,\$s7的内容，可以得到想要的各种相应长度，\$s4,\$s3的结果可以计算出来，通过\$s4的内容用循环把一个临时的寄存器\$t7移到字符串的末尾，然后把一个临时寄存器\$t6作为临时计数器来进行反向迭代。当碰到非空格的字符就通过系统调用11号来输出这个字符串



e) 4. 输入新的字符串并且比较是否相等

思路：主动遍历第二个字符串，当遍历到'\n'的时候退出循环。通过\$s4和\$t6，控制遍历第一个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，如果有什么不同就是不一样了。

如果当第二个字符串遍历到'\n'的时候还没有出现错误，那么第二个字符串要么和第一个字符串完全匹配，要么是第一个字符串的子串，和第一个字符串的前面一部分完全匹配。此时\$t6计数器恰好是第二个字符串的长度（不包括'\n'），将\$t6和\$s3比较一下就可以知道是不是子字符串。



五、实验心得

1. 四选一的题目，乍一看第二题好像相对简单，但是觉得做字符串分析可能可以锻炼更多，所以选了第四题，事实证明的确可以学到了很多

2. 代码重构了4遍，第一次直接写，第二次乱改东拼西凑实现了一些基本功能，在前两次的基础上和查阅了网上的资料后，逐渐悟到了MIPS的分模块编程的感觉，重构了一遍，思路结构比较清晰，易读懂，同时加上了各种注释。第四次是通过调试修复一些小的bug，进一步优化代码结构，最终完成作业。

3. 遇到的问题1：如何通过ASCII码来判断是否是字母或者是数字呢？在mips里面同时实现大于和小于的方法一下子并不是十分容易想出来，这是我的第一次尝试，想用nor来取非来模拟大于等于的这个条件：

```

isAlpha:
    sltiu $t0, $s2, 123
    sltiu $t1, $s2, 97
    nor $t1, $t1, $t1
    and $t0, $t0, $t1

    sltiu $t2, $s2, 107
    sltiu $t3, $s2, 81
    nor $t3, $t3, $t3
    and $t2, $t2, $t3

    or $t0, $t0, $t2

    bne $t0, $zero, countAlpha
    jr $ra

```

但是实际上事与愿违，因为计算机上的取非是在每一个位置上的对应位取非，因此00000000取非并不是00000001。从调试器中可以看出来：

R0 (r0) = 00000000	R8 (t0) = 00000001	R16 (s0) = 00000000	R24 (t8) = 00000000
R1 (e0) = 10010000	R9 (t1) = ffffffff	R17 (s1) = 10010000	R25 (t9) = 00000000
R2 (v0) = 00000004	R10 (t2) = 00000001	R18 (s2) = 00000068	R26 (k0) = 00000000
R3 (v1) = 00000000	R11 (t3) = ffffffff	R19 (s3) = 00000000	R27 (k1) = 00000000
R4 (e0) = 10010000	R12 (t4) = 00000000	R20 (s4) = 00000000	R28 (gp) = 10008000
R5 (e1) = 0000003c	R13 (t5) = 00000000	R21 (s5) = 00000000	R29 (sp) = 7ffff74c

[0x004000c0]	0x2e4e006b	sltiu \$t0, \$s2, 123	; 68: sltiu \$t2, \$s2, 107
[0x004000c4]	0x2e4b0051	sltiu \$t1, \$s2, 97	; 69: sltiu \$t3, \$s2, 81
[0x004000c8]	0x016b5827	nor \$t1, \$t1, \$t1	; 70: nor \$t3, \$t3, \$t3
[0x004000cc]	0x014b5024	and \$t0, \$t0, \$t1	; 71: and \$t2, \$t2, \$t3
[0x004000d0]	0x010e4025	or \$t0, \$t0, \$t2	; 73: or \$t0, \$t0, \$t2
[0x004000d4]	0x15000002	bne \$t0, \$zero, countAlpha	; 75: bne \$t0, \$zero, countAlpha

因此只能换一种方法，我在stackoverflow上提出了问题，然后得到了好心人的帮助，使用xor（异或）来实现‘num1 <= \$t1 <= num2’ 的语意，的确是非常简易强大的。

```

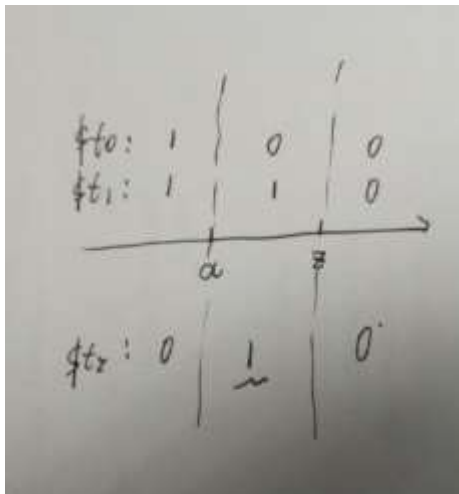
isAlpha:
    # within a-z range
    sltiu $t0, $s2, 0x61 # Lt 'a'?
    sltiu $t1, $s2, 0x7B # Lt 'z' + 1?
    xor $t2, $t0, $t1 # in range a-z ?

```

（此处提供自己的 Stack Overflow 上的提问的网址以表示真实性：

<http://stackoverflow.com/questions/42958229/how-to-implement-not-operation-in-mips>）

看一下图形化的解释：



4.遇到的问题2: jal 的使用非常有用，它可以把整个代码模块化成不同的函数，方便阅读和调试，但是有时候它会自动转到end的地址，通过调试需要在某些jal之前把\$ra给初始化成0，可以暂时解决这个问题，但是实际的机制从调试器上有点难以看出来。

5. 体会：代码风格（开发效率）和机器执行效率的权衡。

还是前面的统计字符数据的例子，我一开始使用的是顺序的结构，也就是说，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。按理来说，这样写可以避免无用的判断，提高运行的效率。但是实际操作上来看，因为mips每条指令的简单性导致代码的行数很多，这样写到最后回使自己绕晕，而且每一个步骤的中间都有可能会出现寄存器迭代然后jump回循环开始的代码，非常冗余而且易错，调试也看得很烦。我在第三次的重构里面，直接把循环里面用三个jal来分级模块处理，只在循环里面处理寄存器的迭代，模块里面专注于范围的比较，这么写下来非常清晰，虽然可能会出现无用比较，但是个人觉得对于汇编这种底层语言，这种速度的提升大概可以忽略，所以舍弃了一点机器执行效率，提高了代码的可理解性，也就是降低了后期的代码维护成本，**我觉得这一点是本次实验中最重要领悟**

6. 体会：写到了第三遍才意识到mips是解释性语言，逐行解释运行，想要跳转完全靠j, jal, jr。所以的话if else语句的实现就和平常的有一点不太一样。当beq语句没有跳转是，它下面的就是else的执行内容。通过缩进和命名代码段来实现这些语意

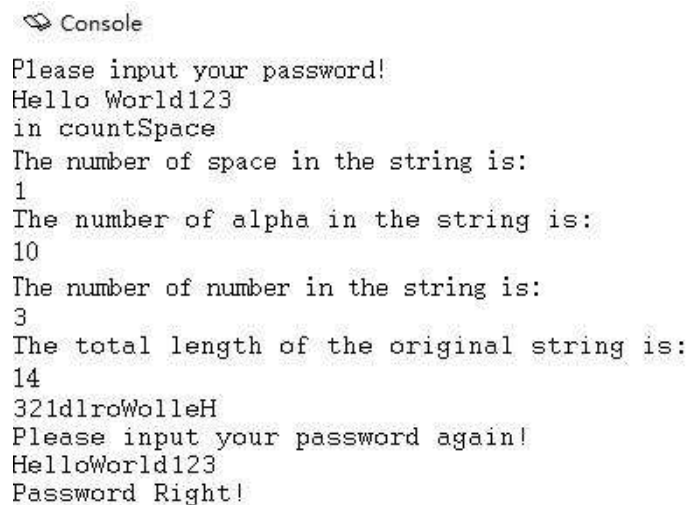
7. 体会：缩进在mips中并不会像python那样产生重大影响，但是结构性的体验在mips这种冗长的语言里面对于**提高代码可读性**来说是非常有必要的。

8. 体会：注释非常重要，否则回看代码的时候不知道什么寄存器存的是什么了。更别说别

人去读懂代码了。所以，我们在编写代码前应该对于一些全局变量的寄存器赋予一些特定的含义，在整个程序的过程中不要轻易改变其中代表的信息，这个提前的决定会给你的编程带来很大的方便。

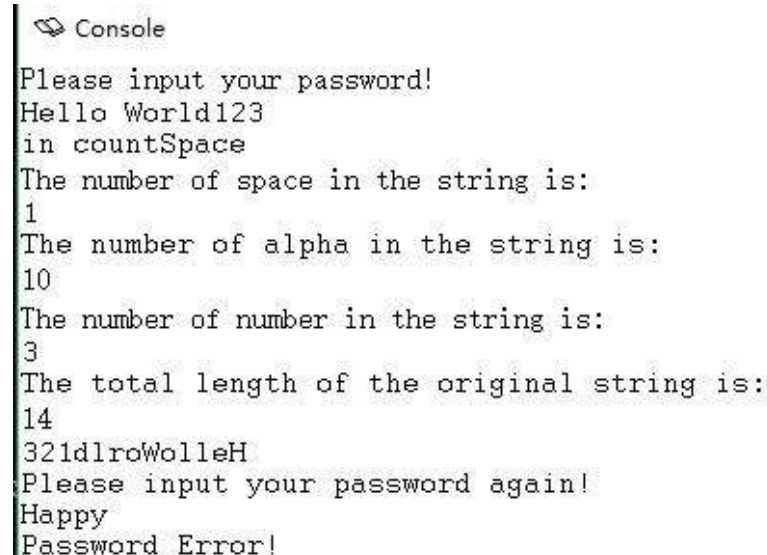
9. 收获总结：在实验的后半段逐渐get到了mips编程的一些语法和编码风格。也学会了使用模拟器和相应的调试器（虽然不得不说调试的功能还是不太人性化）。达到了最初的实验目的

实验结果截图：



```

Console
Please input your password!
Hello World123
in countSpace
The number of space in the string is:
1
The number of alpha in the string is:
10
The number of number in the string is:
3
The total length of the original string is:
14
321dlroWolleH
Please input your password again!
HelloWorld123
Password Right!
  
```



```

Console
Please input your password!
Hello World123
in countSpace
The number of space in the string is:
1
The number of alpha in the string is:
10
The number of number in the string is:
3
The total length of the original string is:
14
321dlroWolleH
Please input your password again!
Happy
Password Error!
  
```


【程序代码】

```
.data
    # string buffer
    buffer: .space 60
    newstring: .space 60

    # Tips messages
    inputmessage: .asciiiz "Please input your password!\n"
    repeatmessage: .asciiiz "Please input your password again!\n"
    cpright: .asciiiz "Password Right!\n"
    cpwrong: .asciiiz "Password Error!\n"

    # report messages
    spaceReport: .asciiiz "The number of space in the string is:\n"
    alphaReport: .asciiiz "The number of alpha in the string is:\n"
    numberReport: .asciiiz "The number of number in the string is:\n"
    lenReport: .asciiiz "The total length of the original string is:\n"
    newline: .asciiiz "\n"

.text
.globl main

main:
    li $v0 4
    la $a0 inputmessage
    syscall

    li $v0 8
    la $a0 buffer
    li $a1 60
    syscall

    # initialize the counter
    # $s5 records the number of space
    # $s6 records the number of numbers
    # $s7 records the number of alpha
    # $s1 contains the original string
    la $s1, buffer
    move $s5, $zero
    move $s6, $zero
    move $s7, $zero
    jal loop
```

\$s2 contains the current judge char, '\n':0xa is the end of the string

loop:

lb \$s2, 0(\$s1)

jal isSpace

jal isAlpha

jal isNumber

addi \$s1, \$s1, 1

bne \$s2, 0xa, loop

move \$ra, \$zero # when there is no such a phase, when it comes to line59, the \$ra will come to end

jal printStatics

jal backwardprint

li \$v0, 4

la \$a0, newline

syscall

j comparepasswd

backwardprint:

use \$t7 as the temp backford pointer of the string

use \$t6 as the counter

move \$t6, \$zero

la \$t7, buffer

aloop:

addi \$t7, \$t7, 1

addi \$t6, \$t6, 1

bne \$t6, \$s4, aloop

move \$t6, \$zero

backloop:

addi \$t7, \$t7, -1

addi \$t6, \$t6, 1

lb \$t5, 0(\$t7)

beq \$t5, 0x20, backloop

```
# print the char
li $v0, 11
la $a0, 0($t5)
syscall
```

```
bne $t6, $s4, backloop
jr $ra
```

```
isSpace:
    beq $s2, 0x20, countSpace
    jr $ra
```

```
countSpace:
    li $v0 4
    la $a0, inCountSpace
    syscall
    addi $s5, $s5, 1
    jr $ra
```

```
isAlpha:
    # within a-z range
    sltiu $t0, $s2, 0x61    # lt 'a'?
    sltiu $t1, $s2, 0x7B    # lt 'z' + 1?
    xor $t2, $t0, $t1      # in range a-z ?

    # within A-Z range
    sltiu $t3, $s2, 0x41    # lt 'A'?
    sltiu $t4, $s2, 0x5C    # lt 'Z' + 1?
    xor $t5, $t3, $t4      # in range A-Z?
```

```
or $t6, $t2, $t5    # judge if it is an alpha
add $s7, $s7, $t6
```

```
jr $ra
```

```
isNumber:
    sltiu $t0, $s2, 0x30    # lt '0'?
    sltiu $t1, $s2, 0x3A    # lt '9' + 1?
    xor $t2, $t0, $t1      # in range 0-9?
```

```
add $s6, $s6, $t2
```

```
jr $ra
```

printStatics:

```
li $v0, 4
la $a0, spaceReport
syscall
li $v0, 1
la $a0, 0($s5)
syscall
```

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

```
li $v0, 4
la $a0, alphaReport
syscall
li $v0, 1
la $a0, 0($s7)
syscall
```

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

```
li $v0, 4
la $a0, numberReport
syscall
li $v0, 1
la $a0, 0($s6)
syscall
```

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

```
# calculate the total length of the original string and store in $t0
# $s4 store the total length of the original string
move $s4, $zero
add $s4, $s5, $s6
add $s4, $s4, $s7
# print the total length
li $v0, 4
la $a0, lenReport
syscall
```

```
li    $v0, 1
la    $a0, 0($s4)
syscall
```

```
li $v0, 4
la $a0, newline
syscall
jr $ra
```

```
end:
li $v0, 10
syscall
```

```
comparepasswd:
li $v0, 4
la $a0, repeatmessage
syscall
```

```
li $v0, 8
la $a0, newstring
la $a1, 60
syscall
```

```
# $s0 has the newstring
# $s1 has the oldstring
la $s0, newstring
la $s1, buffer
```

```
move $t6, $zero
# $s3 contains the length of oldstring without spaces
add $s3, $s6, $s7
```

```
cloop:
lb $t0, 0($s0)
```

```
oldstringGetNoSpaceLoop:
lb $t1, 0($s1)
bne $t1, 0x20, continue
addi $s1, $s1, 1
j oldstringGetNoSpaceLoop
```

```
continue:
bne $t0, $t1, cmpwrong
beq $t0, 0xa, isSubstring
# if newstring not to the end yet, oldstring to, then it is wrong
```

```
beq $t1, 0xa, cmpwrong
```

```
addi $s0, $s0, 1
```

```
addi $s1, $s1, 1
```

```
addi $t6, $t6, 1
```

```
j cloop
```

```
isSubstring:
```

```
# use the true length of two strings to judge whether it is a substring
```

```
beq $s3, $t6, cmpright
```

```
j cmpwrong
```

```
cmpwrong:
```

```
li $v0, 4
```

```
la $a0, cpwrong
```

```
syscall
```

```
j end
```

```
cmpright:
```

```
li $v0, 4
```

```
la $a0, cpright
```

```
syscall
```

```
j end
```