



# 《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 : 杨元昊

学 号 : 16340274

专业(班级) : 16 软件工程四 (7) 班

时 间 : 2018 年 4 月 6 日

成绩：

---

## 实验一：

---

### 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

从键盘输入10个无符号字数并从大到小进行排序，排序结果在屏幕上显示出来。

### 三. 实验器材

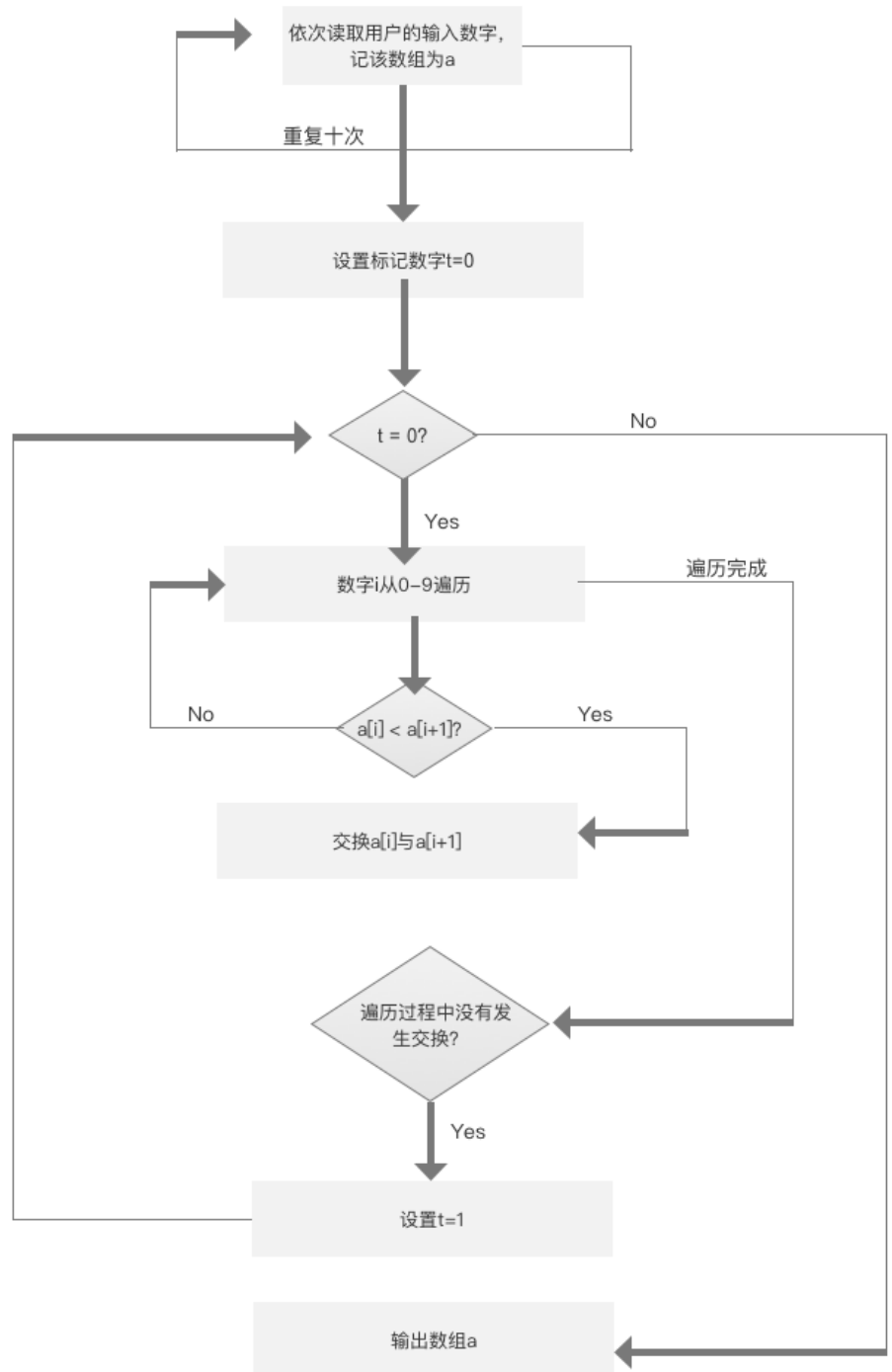
电脑一台、PCSpim模拟器软件一套。

### 四. 实验分析与设计

#### 1. 分析题目：

题目要求我们要从键盘输入十个无符号数字。在该网页中<http://logos.cs.uic.edu/366/notes/mips%20quick%20tutorial.htm>，作者展示了用系统接口来实现IO读写的功能。接着我们要把读取到的数字存入内存中，这样我们就需要利用sw命令来进行相关操作。在完成了上述步骤后，我们就要排序。本题中，我采用了改进版的冒泡排序，即数组中的数字两两排序，需要交换时交换，当遍历一遍数组都没有发生交换时，则排序完成。最后我们仍然利用IO相关知识来实现输出即可。

#### 2. 程序流程图



### 3. 编写MIPS汇编程序：

#### 1. 声明相关输出字符串与定义数组占据空间

```
.data
```

```
Array: .space 40
```

```
Prompt: .ascii "Please enter 10 numbers: \n"
```

```
Prompt2: .ascii "Enter any number: "
```

#### 2. 读入用户输入数字并储存在内存中

```

la      $a0, Prompt2 # gets a number in $v0.

li      $v0, 4

syscall

li      $v0, 5

syscall

sw      $v0, 0($s0) # saves $v0 at wherever $s0 points.

addi    $s0, $s0, 4 # move $s0 forward.

addi    $s2, $s2, 1 # increase the counter.

```

### 3. 实现排序功能

排序主要分为了两个部分，一个是outer loop，指明一个寄存器指向数组首地址。另一个是interloop，指定寄存器存储当前比较数组位数的地址，将他与相邻数字比较，不需要交换时则将当前比较的数组位数移一位，重新进入interloop环节。需要交换时，则进行交换，并同样进行上述操作。在interloop环节中，要判断是否没有任何交换，是则排序完毕。否则再进入outerloop环节。

outterLoop:

```
add $t1, $0, $0
```

```
la $s0, Array
```

innerLoop:

```
lw $t2, 0($s0)
```

```
lw $t3, 4($s0)
```

```
slt $t5, $t2, $t3
```

```
beq $t5, $0, continue
```

```
add $t1, $0, 1
```

```
sw $t2, 4($s0)
```

```
sw $t3, 0($s0)
```

continue:

```
addi $s0, $s0, 4
```

```
bne $s0, $t0, innerLoop
```

```
bne $t1, $0, outterLoop
```

#### 4. 实现输出排序后数组功能

用寄存器存一个值进行判断是否将十个数字都输出了，是则停止程序

```
lw $a0, ($s0) # load value into $a0.
```

```
li $v0, 1 # load 1 into $v0 to print an integer.
```

```
syscall # print value in $a0.
```

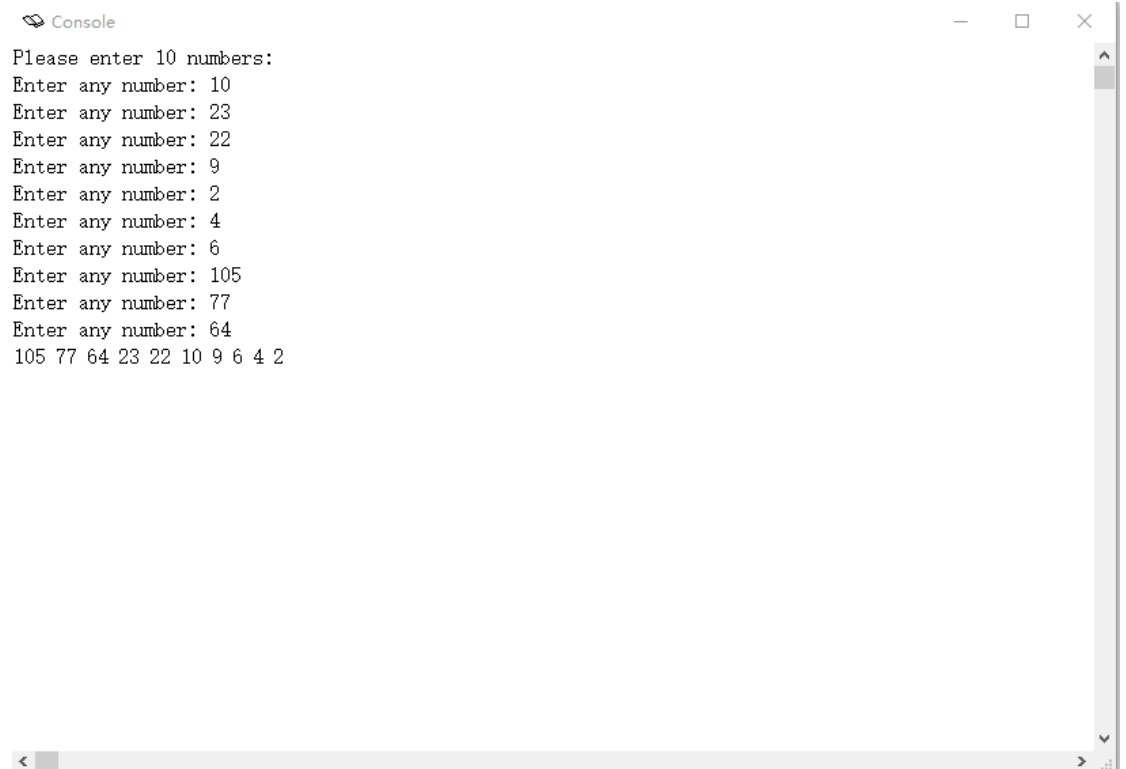
```
addi $s0, $s0, 4 # increment the Array pointer to print the next value.
```

```
addi $s2, $s2, 1 # increment the counter.
```

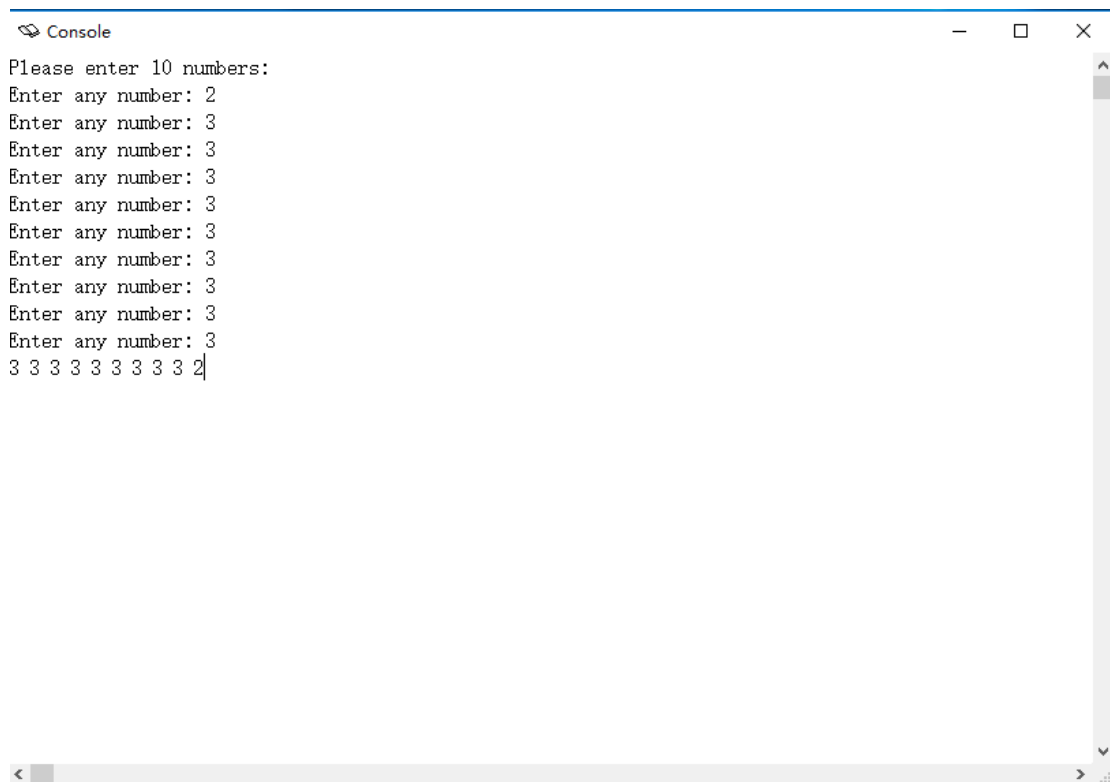
```
beq $s2, 9, EndPrint # if we have printed 10 numbers, then we have done
```

#### 5. 实验结果与分析

经过多次操作，发现给定输入，输出均符合预期



```
Console
Please enter 10 numbers:
Enter any number: 10
Enter any number: 23
Enter any number: 22
Enter any number: 9
Enter any number: 2
Enter any number: 4
Enter any number: 6
Enter any number: 105
Enter any number: 77
Enter any number: 64
105 77 64 23 22 10 9 6 4 2
```



```
Console
Please enter 10 numbers:
Enter any number: 2
Enter any number: 3
Enter any number: 3
Enter any number: 3
Enter any number: 3
Enter any number: 3
Enter any number: 3
Enter any number: 3
Enter any number: 3
Enter any number: 3
3 3 3 3 3 3 3 3 3 2
```

## 五. 实验心得

思考的问题:

要实现从键盘输入10个无符号字数并从大到小进行排序,排序结果在屏幕上显示出来这个功能,很重要的一个设计思路是按流程按功能实现。先将总的需求分解成多个小功能,然后依次实现多个小功能的函数。

此时面临的另一个问题就是如何才能正确的在汇编程序中做到函数调用呢?经过阅读老师提供的源代码,能够从中发现可以使用jal,来进行函数跳转,跳转的地址位置会被寄存器\$ra保存,也可以通过jr \$ra来实现跳转功能。除此之外,汇编程序的代码就会从上到下一直执行。

在正确地理解了函数跳转功能的实现后,对排序代码的编写也更得心应手了。在innerloop排序逻辑段内因为无论是否进行交换操作,都要进行相关值检查并判断下一步操作是在outerloop还是innerloop程序段中。于是我们可以将检查操作抽象成一个函数,并合理放置该函数的位置,需要交换时,交换后则执行此函数,不需要交换,那么就直接跳转到判断函数来进行下一步操作。

```

outterLoop:
    add $t1, $0, $0
    la $s0, Array
innerLoop:
    lw $t2, 0($s0)
    lw $t3, 4($s0)
    slt $t5, $t2, $t3
    beq $t5, $0, judge
    add $t1, $0, 1
    sw $t2, 4($s0)
    sw $t3, 0($s0)
judge:
    addi $s0, $s0, 4
    bne $s0, $t0, innerLoop
    bne $t1, $0, outterLoop

```

因为对汇编程序不熟悉，所以也很有必要先实现c语言的代码，再以此为依托来进行汇编代码的实现。

遇到的问题：

1. 对寄存器存储的是变量还是地址没有很好地区分。

在冒泡排序的内部循环中，我们需要将数组比较起始位置的数字与它相邻的数字放到寄存器中。这时，我们需要做的是读取地址所对应的数字的值，而非直接读取地址。在最初我对这两者间不同的指令没有很好区分，于是写成了

```

innerLoop:
    move $t2, $s0
    addi $s0, $s0, 4
    move $t3, $s0

```

这将导致输出数字非常奇怪的错误，经过调试，发现了t的值十分不正常

	General Reg
R8 (t0)	= 10010028 R
R9 (t1)	= 00020a00 R
R10 (t2)	= <del>000000267</del> ]
R11 (t3)	= <del>00000006</del> R
R12 (t4)	= 00000000 R
R13 (t5)	= 00000000 R

在反复阅读了课件中的相关代码和解释后，发现了这个错误，将指令由move更改为正确的lw。

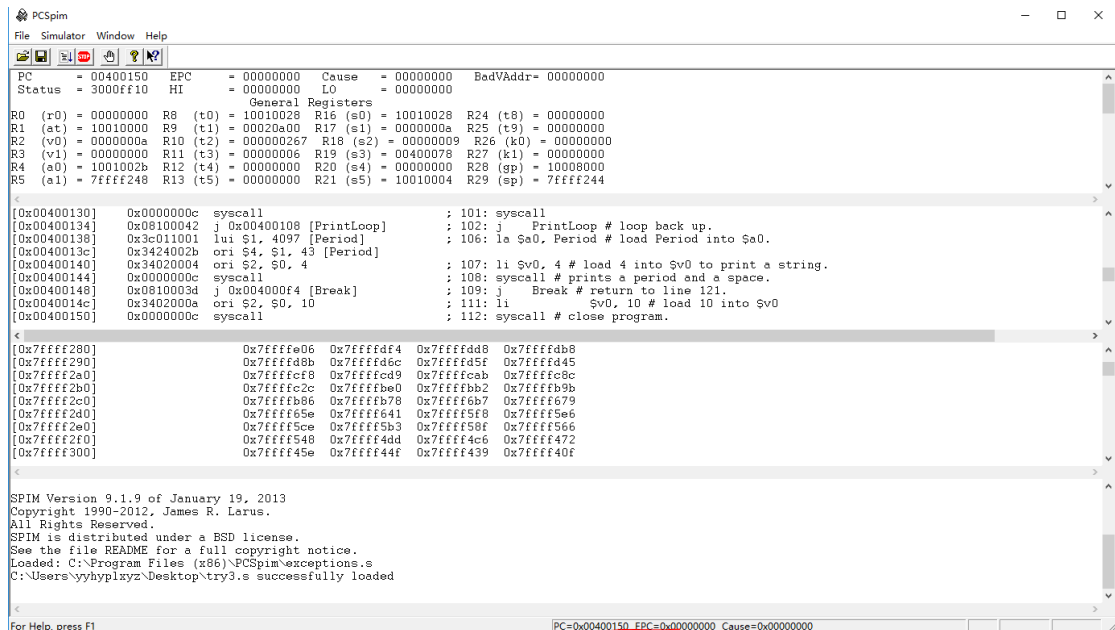
```

    lw $t2, 0($s0)
    lw $t3, 4($s0)

```

## 2. 程序一直不能正常退出

本以为mips汇编程序会在运行完代码块后自动退出程序的,但是出现了程序一直无法退出的后果,见下图。(右下角的数字会一直变化)



后来在网站上找到资料,知道了要将退出命令放置在寄存器中,再调用操作系统接口。就像是c语言的return 0一样。

```

exit: li    $v0, 10 # load 10 into $v0
      syscall # close program.

```

## 3. 对汇编语言下,函数的跳转不清楚

以为汇编语言程序的执行就像高级语言执行一样,也有很自然的栈的概念,从函数a跳转到函数b后,当b函数执行完了,会自动执行函数a中剩下的代码块。但实际上不会,需要我们自己用jal方法记录跳转的点,然后利用jr语句再跳转到该点上。

一开始代码是这么写的



```

main:
    la    $s0, Array # $s0 points to the first value of Array.
    la    $s5, Array
    jal    ReadNums
    la    $t0, Array
    add    $t0, $t0, 40
outterLoop:
    add    $t1, $0, $0
    la    $s0, Array
innerLoop:
    lw    $t2, 0($s0)
    lw    $t3, 4($s0)
    slt    $t5, $t2, $t3
    beq    $t5, $0, continue
    add    $t1, $0, 1
    sw    $t2, 4($s0)
    sw    $t3, 0($s0)
continue:
    addi   $s0, $s0, 4
    bne    $s0, $t0, innerLoop
    bne    $t1, $0, outterLoop

    addi   $s5, 4
    move    $s0, $s5
    li     $s2, 0 # resets our counter to be used with PrintNums.
    jal    PrintNums # jump and link PrintNums; $ra = line 38.

    j     exit # jump to exit.

ReadNums:
    la    $a0, Prompt # gets the amount of numbers we want saved into $1.
    li    $v0, 4
    syscall
    li    $s1, 10
    li    $s2, 0 # amount of numbers we've added so far.
    jal    GetNums

```

发现程序在进入Readnums函数后将会一直执行Readnums后的代码读取用户输入，输出数组等代码，而上面的排序部分的代码就不会再运行了。

```

[0x004000e8] 0x0000000c syscall ; 59: syscall
[0x004000ec] 0x34020005 ori $2, $0, 5 ; 60: li $v0, 5
[0x004000b0] 0x0000000c syscall ; 61: syscall
[0x004000b4] 0xae020000 sw $2, 0($16) ; 62: sw $v0, 0($s0) # saves $v0 at wherever $s0 points.
[0x004000b8] 0x22100004 addi $16, $16, 4 ; 63: addi $s0, $s0, 4 # move $s0 forward.
[0x004000bc] 0x22520001 addi $18, $18, 1 ; 64: addi $s2, $s2, 1 # increase the counter.
[0x004000c0] 0x22460004 addi $22, $22, 4 ; 65: addi $s6, $s6, 4 # increase the "last" pointer.
[0x004000c4] 0x12320002 beq $17, $18, 8 [Break-0x004000c4]; 66: beq $s1, $s2, Break # if our list is full, back to line 57.
[0x004000c8] 0x08100027 j 0x0040009c [GetNums] ; 67: j GetNums

[0x7ffff2b0] 0x7ffff2c 0x7ffffbe0 0x7ffffbb2 0x7ffffb9b
[0x7ffff2c0] 0x7ffffb86 0x7ffffb78 0x7ffff6b7 0x7ffff679
[0x7ffff2d0] 0x7ffff65e 0x7ffff641 0x7ffff5f8 0x7ffff5e6
[0x7ffff2e0] 0x7ffff6c9 0x7ffff5b3 0x7ffff58f 0x7ffff566
[0x7ffff2f0] 0x7ffff548 0x7ffff4dd 0x7ffff4c6 0x7ffff472
[0x7ffff300] 0x7ffff45e 0x7ffff44f 0x7ffff439 0x7ffff40f
[0x7ffff310] 0x7ffff3e6 0x7ffff3cb 0x7ffff3a1 0x7ffff30e
[0x7ffff320] 0x7ffff36f 0x7ffff35d 0x00000000 0x00000000
[0x7ffff330] 0x78280000 0x5c293638 0x70534350 0x705c6d69

```

于是经过考虑后，将代码段更改如下

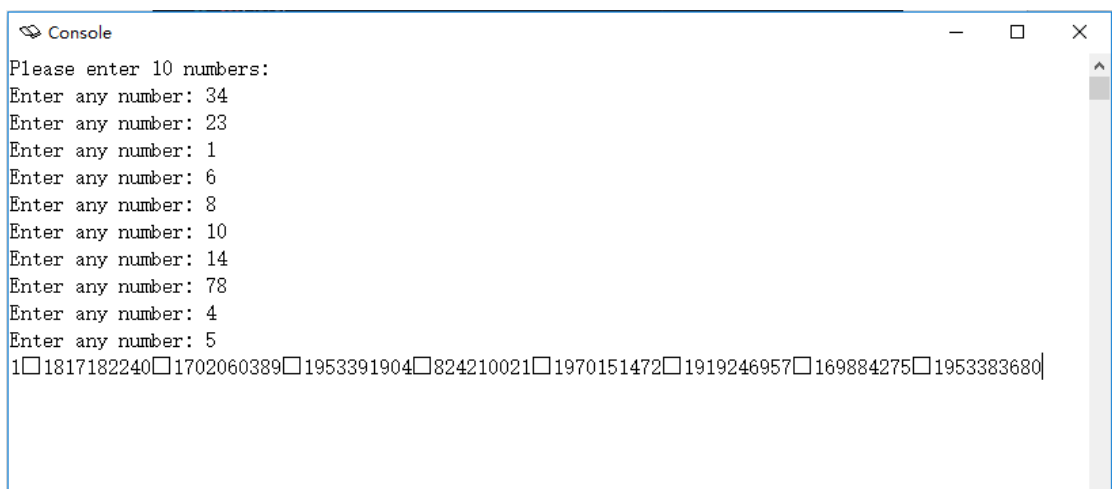
```

ReadNums:
    la    $a0, Prompt # gets the amount of numbers we want saved into s1.
    li    $v0, 4
    syscall
    li    $s1, 10
    li    $s2, 0 # amount of numbers we've added so far.
    move  $s3, $ra
    jal   GetNums
    move  $ra, $s3
    jr    $ra

```

这样就正确的实现了函数间的跳转了。

#### 4. 数组打印输出时起始的地址没有选对



```

Console
Please enter 10 numbers:
Enter any number: 34
Enter any number: 23
Enter any number: 1
Enter any number: 6
Enter any number: 8
Enter any number: 10
Enter any number: 14
Enter any number: 78
Enter any number: 4
Enter any number: 5
1□1817182240□1702060389□1953391904□824210021□1970151472□1919246957□169884275□1953383680

```

如上图所示，在最终打印时，输出的数组都是很奇怪的数字。这时候直观的想法就是我把数组的初始地址弄错了，因为在实际的操作过程中，有多次数组地址移位的操作，在这方面有差错是很正常的。经过肉眼观察，发现如下图部分

```

PrintLoop:
    lw    $a0, ($s0) # load value into $a0.
    li    $v0, 1
    syscall
    addi  $s0, $s0, 4
    addi  $s2, $s2, 1
    beq   $s2, 9, EndPrint # if we have printed 10 numbers, then we have done
    li    $v0, 4 # load 4 into $v0 to print a string

```

于是在输出打印时对寄存器中存储的数组首地址作调整

```

continue:
    addi $s0, $s0, 4
    bne $s0, $t0, innerLoop
    bne $t1, $0, outerLoop

    addi $s5, 4
    move $s0, $s5
    li $s2, 0
    jal PrintNums # jump and link PrintNums

    j exit # jump to exit.

```

接着输出结果正确。

#### 6. 分割符号没有正确显示

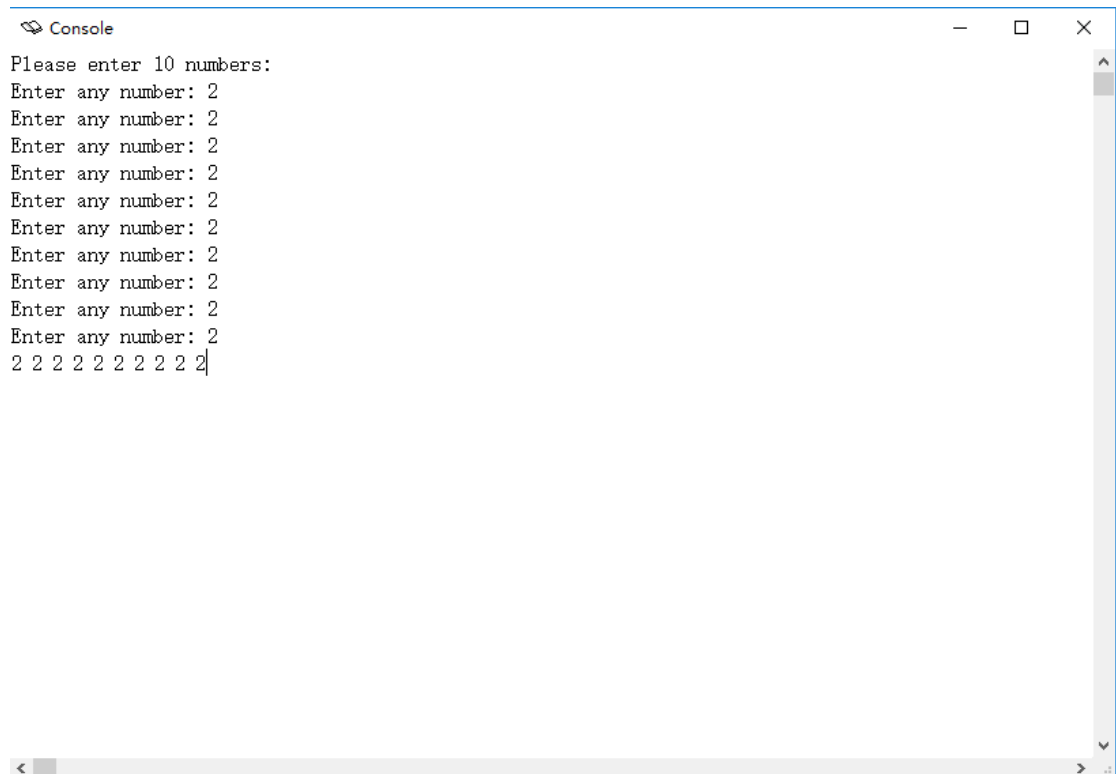
在实验中发现了一个问题就是输出中数字的间隔都是一个方块，而不是想要的空格。

后来经反复调试查看，发现是因为没有及时清空寄存器中的值，下次星空寄存器或者重启软甲即可实现输出空格的效果

```

Console
Please enter 10 numbers:
Enter any number: 32
Enter any number: 21
Enter any number: 6
Enter any number: 78
Enter any number: 55
Enter any number: 98
Enter any number: 23
Enter any number: 09
Enter any number: 45
Enter any number: 10
98□78□55□45□32□23□21□10□9|

```



```
Console
Please enter 10 numbers:
Enter any number: 2
Enter any number: 2
Enter any number: 2
Enter any number: 2
Enter any number: 2
Enter any number: 2
Enter any number: 2
Enter any number: 2
Enter any number: 2
Enter any number: 2
2 2 2 2 2 2 2 2 2 2
```

实验的思考与感悟：

1. Python的逻辑与C相比是高度抽象的,这样就省略了很多底层语言做的逻辑功能,让程序员可以更直接的解决问题。与汇编相比, c语言又是逻辑高度抽象的,它屏蔽了硬件中的许多操作过程和系统接口,将一个最简单的逻辑也要大卸八块,明确值在存储器和寄存器间的转移,这可以让我们更好地理解计算机的运行原理,但是编写汇编程序真的太麻烦了。

2、MIPS指令集初看上去很复杂,但是数据类的仅有load、store和move,当然按操作数的长短分许多lw、lh等等,但实际上就这三个。运算类的也仅仅完成基本功能,也根据操作数长短分了许多子指令。跳转类更少,要么无条件跳转,要么根据操作数跳转。这些指令确实属于最常用的80%的。只要掌握了这些最常用的指令就足够了

3、MIPS在很多方面跟c语言十分相像。的寻址方式很自然,就是寄存器加偏移寻址方式,这与c语言中的代码是一样的,也是实际编写中感觉最舒服的一部分。尤其是需要调试时。我感觉这和C或C++语言调试大同小异,都是设置断点,一步一步运行,再去看寄存器中值的变化,如果寄存寄存器中存的是地址,就去找对应的存储器中存储的值,比较麻烦的一点是存储器存储的值都是十六进制形式的,需要再做一次转换,很不直观。

4. 实验过程中遇到了一些问题，都能通过查阅资料及调试的方法解决，问题的主要原因是MIPS汇编程序的编写逻辑与高级语言的编写逻辑差异较大，指令更加繁琐，不像python或者c的语法，可以望文生义。在查阅资料的过程中也发现了指令代码，指令和伪指令的区别：汇编器为了方便使用，定义了许多伪指令，如li、ror等。最终会被扩展成多条实际指令。这样一来，好处就是能省力，但坏处就是对汇编器要求较高，而且对机器指令反汇编后难以还原为伪指令。而在最初对这部分内容没有概念时，给自己的理解增加了很多难度，因为总是会遇到很多各式各样奇怪的指令，主要是经过一个tutorial的教导，条分缕析讲解了重要指令的操作，

而对其他指令略去不讲，很好地达到了纲举目张的效果。网址为

<http://logos.cs.uic.edu/366/notes/mips%20quick%20tutorial.htm>

#### 【程序代码】

```
.data
Array: .space 40
Space: .asciiz ", "
Period: .asciiz ". "
Prompt: .asciiz "Please enter 10 numbers: \n"
Prompt2: .asciiz "Enter any number: "

.text

.globl main

main:
    la    $s0, Array # $s0 points to the first value of Array.
    la    $s5, Array
    jal    ReadNums # jump and link ReadNums;
```

```
        la    $t0, Array
        add   $t0, $t0, 40
outterLoop:
        add   $t1, $0, $0
        la    $s0, Array
innerLoop:
        lw    $t2, 0($s0)
        lw    $t3, 4($s0)
        slt   $t5, $t2, $t3
        beq   $t5, $0, continue
        add   $t1, $0, 1
        sw    $t2, 4($s0)
        sw    $t3, 0($s0)
continue:
        addi  $s0, $s0, 4
        bne   $s0, $t0, innerLoop
        bne   $t1, $0, outterLoop

        addi  $s5, 4
        move  $s0, $s5
        li    $s2, 0
        jal   PrintNums # jump and link PrintNums

        j     exit # jump to exit.

ReadNums:
        la    $a0, Prompt # gets the amount of numbers we want saved into s1.
        li    $v0, 4
```

```
syscall  
  
li      $s1, 10  
  
li      $s2, 0 # amount of numbers we've added so far.  
  
move    $s3, $ra  
  
jal     GetNums  
  
move    $ra, $s3  
  
jr      $ra
```

GetNums:

```
la      $a0, Prompt2 # gets a number in $v0.  
  
li      $v0, 4  
  
syscall  
  
li      $v0, 5  
  
syscall  
  
sw      $v0, 0($s0) # saves $v0 at wherever $s0 points.  
  
addi    $s0, $s0, 4 # move $s0 forward.  
  
addi    $s2, $s2, 1  
  
addi    $s6, $s6, 4  
  
beq     $s1, $s2, Break  
  
j GetNums
```

Break: #

```
jr      $ra
```

PrintNums:

```
move $s3, $ra  
  
jal PrintLoop # jump to PrintLoop;
```

```
move $ra, $s3
```

```
jr $ra
```

```
PrintLoop:
```

```
lw $a0, ($s0) # load value into $a0.
```

```
li $v0, 1
```

```
syscall
```

```
addi $s0, $s0, 4
```

```
addi $s2, $s2, 1
```

```
beq $s2, 10, EndPrint # if we have printed 10 numbers, then we have done
```

```
la $a0, Space # load Comma into $a0
```

```
li $v0, 4 # load 4 into $v0 to print a string
```

```
syscall
```

```
j PrintLoop # loop back up.
```

```
EndPrint:
```

```
    la $a0, Period # load Period into $a0.
```

```
    li $v0, 4 # load 4 into $v0 to print a string.
```

```
    syscall # prints a period and a space.
```

```
    j Break # return to line 121.
```

```
exit:  li      $v0, 10 # load 10 into $v0
```

```
        syscall # close program.
```