



# 《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：黄羽盼

学号：14331106

专业（班级）：14 软件工程三（6）班

合作者：蒋子涵

时间：2016 年 3 月 30 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

从内存中读取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

PC机一台，PCSpim 模拟器软件一套。

### 四. 实验分析与设计

说明：根据需要书写相关内容，如：

程序流程图、分析、设计、实验步骤和实验结果及分析等。

#### 1. 分析题目。

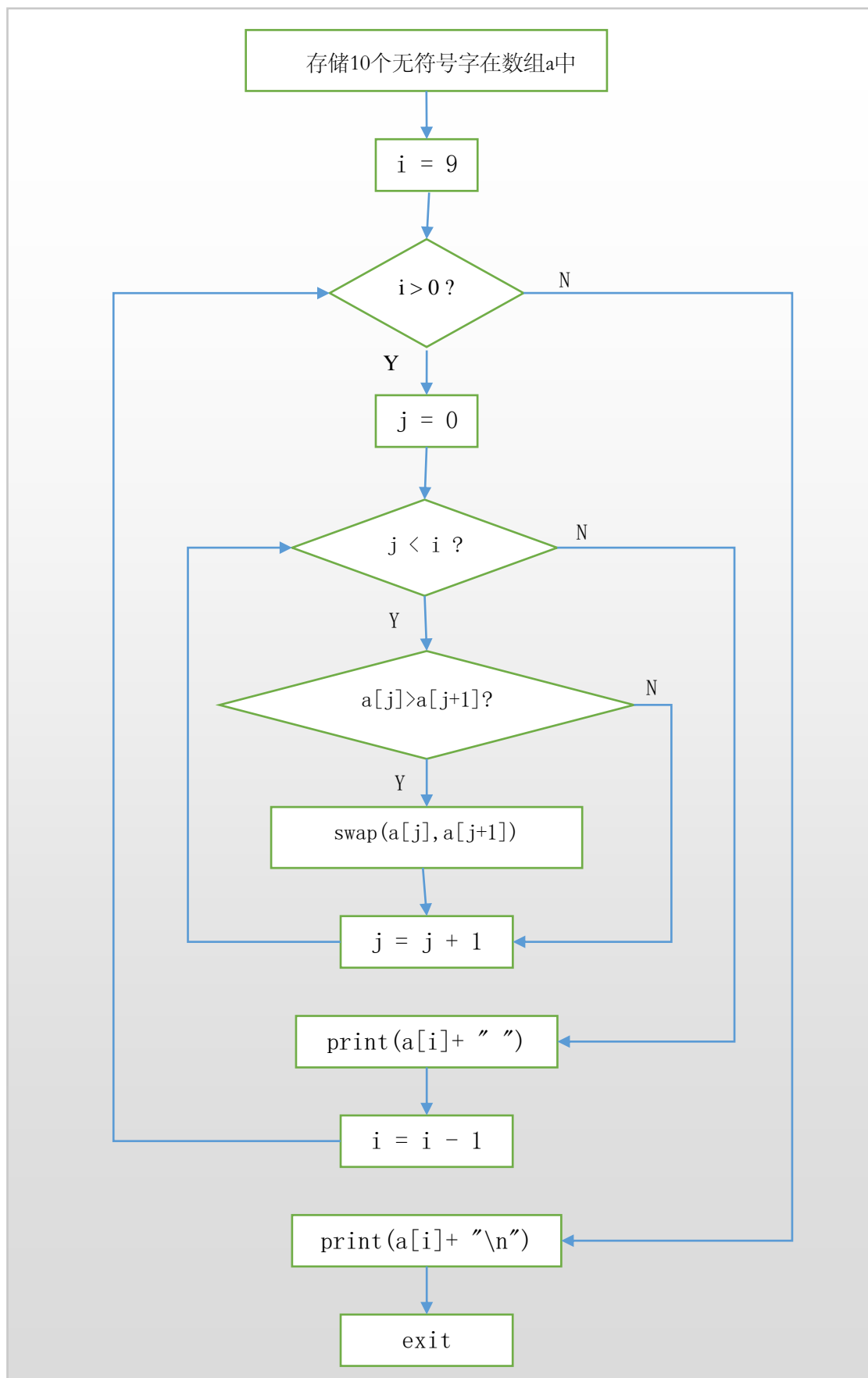
首先题目要求从内存中读取10个无符号字数，回顾‘load-t.asm’的例子，其中展示了从存储器中读取各种类型的数据装入寄存器的方法。针对这道题目，可以选用‘lhu’来将半字（不带符号）装入HALFWORD。并自定义10个无符号字数存入数组中。

其次题目要求这些数从大到小进行排序。我采用了基础排序算法：冒泡排序。需要用到二重循环。外层循环用i指向本轮循环找到的最大数。内层循环由j指示，每次从第一个数与后面未筛选出来的数进行比较，以将本轮循环最大的数冒泡上升。

最后题目要求排序结果在屏幕上显示出来。由于冒泡排序采用每轮循环找出一个相对最大数，因此每轮内层循环都输出，最后跳出外层循环时也输出即可按从大到小的顺序输出结果。

#### 2. 设计程序流程图。

根据第一步的分析，可以设计出如下程序流程图：



## 3. 编写C语言程序

由于C语言是我熟悉的编程语言，我采用先编写C语言程序再向汇编语言过渡的方法。代码如下所示：

```
#include<stdio.h>
int main() {
    //读取10个无符号字数
    unsigned int a[] = { 231, 4, 6, 7, 3, 3, 2, 654, 4, 17};
    //冒泡排序 （从大到小）
    int i, j;
    for (i = 9; i > 0; --i) {
        for (j = 0; j < i; ++j) {
            if (a[j] > a[j+1]){
                int temp = a[j];
                a[j] = a[j+1];
                a[j + 1] = temp;
            }
        }
        //输出结果
        printf("%d ", a[i]);
    }
    printf("%d\n", a[i]);
    return 0;
}
```

#### 4. 编写MIPS汇编语言程序

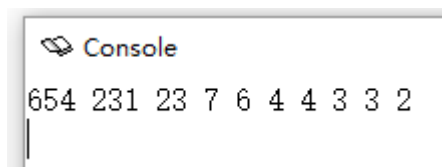
具体来说，分为以下几个步骤。

- 1) 学习书本和课件中的关于MIPS汇编语言程序设计的基本方法和PCSpim模拟器的使用的理论知识。
- 2) 学习三个实例的编写方法，并用PCSpim模拟器运行三个实例，利用单步运行、断点调试等方法观察每一个命令的用法及其作用。
- 3) 将程序分解成两层（循环）OuterLoop和InnerLoop，根据主要逻辑写出框架。并分出子模块，如交换两个数、打印某个数。主要利用jump和branch to跳转到自身或其他部分来实现循环或跳出循环以及控制某些部分内容的执行与否。
- 4) 对照MIPS汇编指令及SYSCALL系统功能调用表编写指令。也参照实例和课本上常用逻辑功能的写法。写的过程中在模拟器中运行以测试代码。出现过的问题包括语法错误、运行时错误和输出结果错误，都能通过查找资料及单步执行和调试的方法解决。

5) 得到正确输出结果后规范代码以形成良好的代码风格。具体有为代码添加注释、缩进、空格、分区换行、修改部分命名。这些在编写过程中也应当尽量注意。

## 5. 实验结果与分析

成功从内存中读取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。如下所示:

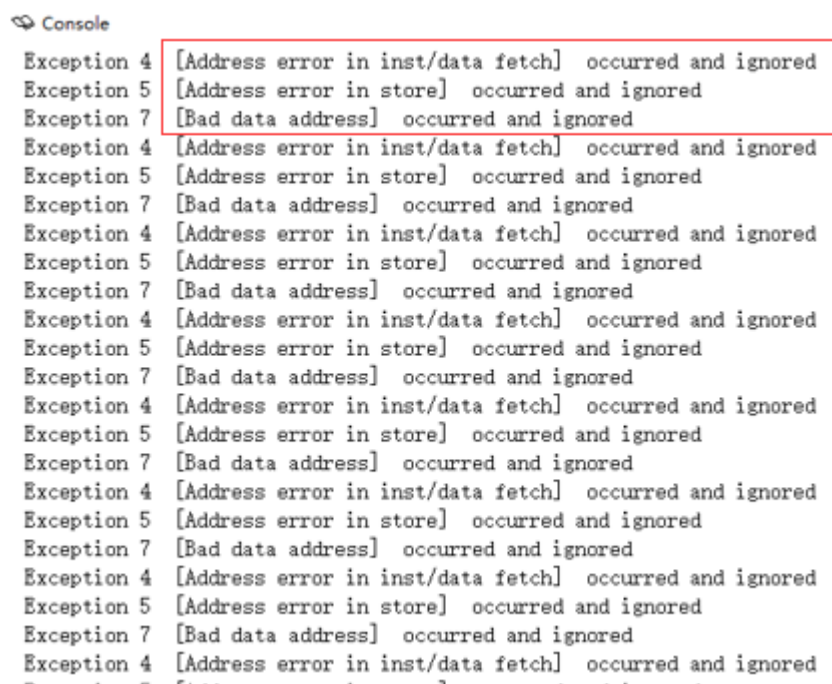


## 五. 实验心得

1. 实验过程中遇到的问题及解决的方法。

下面通过文字说明和截图的形式来记录问题及解决方法。由于图是在实验中遇到问题时及时截的图，后来对代码进行了一定修改，因此最终代码与之略有不同。

1) 首次编写完完整的代码运行时出现了如下错误提示:



```

Console
Exception 7 [Bad data address] occurred and ignored
Exception 4 [Address error in inst/data fetch] occurred and ignored
Exception 5 [Address error in store] occurred and ignored
Exception 7 [Bad data address] occurred and ignored
15 Exception 4 [Address error in inst/data fetch] occurred and ignored
Exception 5 [Address error in store] occurred and ignored
Exception 7 [Bad data address] occurred and ignored
Exception 4 [Address error in inst/data fetch] occurred and ignored
Exception 5 [Address error in store] occurred and ignored
Exception 7 [Bad data address] occurred and ignored
Exception 4 [Address error in inst/data fetch] occurred and ignored
Exception 5 [Address error in store] occurred and ignored
Exception 7 [Bad data address] occurred and ignored
15 Exception 4 [Address error in inst/data fetch] occurred and ignored
Exception 5 [Address error in store] occurred and ignored
Exception 7 [Bad data address] occurred and ignored
Exception 4 [Address error in inst/data fetch] occurred and ignored
Exception 5 [Address error in store] occurred and ignored
Exception 7 [Bad data address] occurred and ignored
15 Exception 4 [Address error in inst/data fetch] occurred and ignored
Exception 5 [Address error in store] occurred and ignored
Exception 7 [Bad data address] occurred and ignored
15 15 15

```

可见错误为[Bad data address] , [Address error in inst/data fetch] , [Address error in store], 并且几次输出了某个数15。

解决方法:

抛出的异常提示是地址错误。由于这是首次运行, 我选择首先通过肉眼检查一遍代码的方式。一行一行地看, 重新理清和审视自己的思路, 尤其要注意自己写的时候就不确定和有疑问的地方。看了一遍, 我发现在交换两个数时我是这样写的:

```

lhu $t4, 0($t3)           # t4=array[j]
lhu $t5, 4($t3)           # t5=array[j+1]
slt $t6, $t5, $t4         # if(a[j+1]<a[j]) t6=1; else t6=0
beq $t6, $zero, SkipSwap  # go to SkipSwap if a[j+1]>=a[j]
#Swap:
lhu $t7, 0($t4)           # t7(temp)=array[j];
sw $t5, 0($t4)            # array[j]=array[j+1]
sw $t7, 0($t5)            # array[j+1]=temp

```

此处我持疑问态度。为了一探究竟, 我单步运行程序(F10), 发现在下方方框所示这一步时进入了kernel并输出了Exception的错误信息。

[illegible]

```

[0x00000190] 000070207 sw $t0, 0($a)
[0x00000194] 0x401a6800 mfc0 $26, $13 : 95: mfc0 $k0 $13 # Cause register
[0x00000198] 0x001a2082 srl $4, $26, 2 : 96: srl $a0 $k0 2 # Extract ExCode Field
[0x0000019c] 0x3084001f andi $4, $4, 31 : 97: andi $a0 $a0 0x1f
[0x000001a0] 0x34020004 ori $2, $0, 4 : 101: li $v0 4 # syscall 4 (print_str)
[0x000001a4] 0x3c049000 lui $4, -28672 [__m1_] : 102: le $a0 __m1_
[0x000001a8] 0x0000000c syscall : 103: syscall
[0x000001ac] 0x34020001 ori $2, $0, 1 : 105: li $v0 1 # syscall 1 (print_int)

```

```

[0x800001a0]    0x34020004    ori $2, $0, 4           ; 101: li $v0 4           # syscall 4 (print_str)
[0x800001a4]    0x3c049000    lui $4, -28672 [_mi_]   ; 102: la $a0 _mi_
[0x800001a8]    0x0000000c    syscall                 ; 103: syscall
[0x800001ac]    0x34020001    ori $2, $0, 1           ; 105: li $v0 1           # syscall 1 (print_int)
[0x800001b0]    0x001a2002    srl $4, $26, 2          ; 106: srl $a0 $k0 2      # Extract ExcCode Field
[0x800001b4]    0x3084001f    andi $4, $4, 31         ; 107: and $a0 $a0 0x1f
[0x800001b8]    0x0000000c    syscall                 ; 108: syscall
[0x800001bc]    0x34020004    ori $2, $0, 4           ; 110: li $v0 4           # syscall 4 (print_str)

```

证实了确实是在这一步出错的，我进行了分析：交换时，由于前面曾从存储器读取array[j]，array[j+1]的数据到寄存器\$t4，\$t5中，我直接把\$t4，\$t5和array[j]，array[j+1]等同来作为赋值的对象，这是错误的。由于\$t3指向array[j]，应当修改如下：

```
#Swap:
lhu $t7, 0($t3)      # t7(temp)=array[j];
sw $t5, 0($t3)        # array[j]=array[j+1]
sw $t7, 4($t3)        # array[j+1]=temp
```

修改后不再出现刚才的错误信息。

我的认识:

出现错误提示时可以通过报错的具体内容查找资料,了解这一类错误的一般原因(本题是地址出错),再进而通过单步调试、断点调试的方法快速锁定出错的地方,再根据程序的具体情况分析错误原因,修改后测试和验证。此类错误有可能层层关联,需要细心跟踪代码,观察地址及数据变化。

- 2) 进行了上面第一步的修改后不再报错,但是输出全为4而非十个不全相等的数。

解决方法：

由于此次为结果错误,我采用单步执行跟踪观察各寄存器和数据变化的方法来找错。注意我存储的数组的十个十进制数为{ 231, 4, 6, 7, 3, 3, 2, 654, 4, 23}, 对应的十六进制数为 {0x000000e7, 0x00000004, 0x00000006, 0x00000007, 0x00000003, 0x00000003, 0x00000002, 0x0000028e, 0x00000004, 0x00000017}

单步执行时观察到内层循环第一轮j = 0时是正确的, 231和4进行了交换, t4=array[j], t5=array[j+1].第二轮j = 1时应当是t4=231, t5=6, 但控制台结果显示t4, t5并没有变化。如图:

```

R0 (r0) = 00000000 R8 (t0) = 00000009 R16 (s0) = 00000000 R24 (t8) = 00000000
R1 (at) = 10010000 R9 (t1) = 00000001 R17 (s1) = 00000000 R25 (t9) = 00000000
R2 (v0) = 00000000 R10 (t2) = 00000001 R18 (s2) = 00000000 R26 (k0) = 00000000
R3 (v1) = 00000000 R11 (t3) = 10010000 R19 (s3) = 00000000 R27 (k1) = 00000000
R4 (a0) = 00000000 R12 (t4) = 00000004 R20 (s4) = 00000000 R28 (gp) = 10008000
R5 (a1) = 7ffff004 R13 (t5) = 000000e7 R21 (s5) = 00000000 R29 (sp) = 7ffff0fc
R6 (a2) = 7ffff004 R14 (t6) = 00000001 R22 (s6) = 00000000 R30 (s8) = 00000000
R7 (a3) = 00000000 R15 (t7) = 000000e7 R23 (s7) = 00000000 R31 (ra) = 00400018

<
[0x00400034] 0x0128502a slt $t0, $9, $8 ; 16: slt $t2, $t1, $t0 #if(j<i) t2=1
[0x00400038] 0x1140000a beq $t0, $0, 40 [skip-0x00400038]; 17: beq $t2, $zero, skip #滑槽弧顶
[0x0040003c] 0x8d6c0000 lw $t2, 0($t1) ; 19: lw $t4, 0($t3) #t4=array[j]
[0x00400040] 0x8d6d0004 lw $t3, 4($t1) ; 20: lw $t5, 4($t3) #t5=array[j+1]
[0x00400044] 0x01ac702a slt $t4, $t3, $t2 ; 21: slt $t6, $t5, $t4 #if(a[j+1]<a[j]
[0x00400048] 0x11c00004 beq $t4, $0, 16 [skipExchange-0x00400048]; 22: beq $t6, $zero, skipEx
[0x0040004c] 0x8d6f0000 lw $t5, 0($t1) ; 24: lw $t7, 0($t3) #t7(temp)=array[
[0x00400050] 0xad6d0000 sw $t3, 0($t1) ; 25: sw $t5, 0($t3) #array[j]=array[

<
DATA
[0x10000000]...[0x10010000] 0x00000000 0x00000004 0x00000006 0x00000007
[0x10010000] 0x00000003 0x00000003 0x00000002 0x0000028e
[0x10010010] 0x00000004 0x00000017 0x00000020 0x00000000
[0x10010020]
[0x10010030]...[0x10040000] 0x00000000

```

这说明array[j]没有及时更新。看到代码相应部分:

```

    lhu $t4, 0($t3)           # t4=array[j]
    lhu $t5, 4($t3)           # t5=array[j+1]
    slt $t6, $t5, $t4         # if(a[j+1]<a[j]) t6=1; else t6=0
    beq $t6, $zero, SkipSwap   # go to SkipSwap if a[j+1]>=a[j]
    #Swap:
    lhu $t7, 0($t3)           # t7(temp)=array[j];
    sw $t5, 0($t3)            # array[j]=array[j+1]
    sw $t7, 4($t3)            # array[j+1]=temp

SkipSwap:
    addi $t1, $t1, 1          # ++j
    j InnerLoop               # jump to test of inner loop

```

可以看到\$t3贯穿了array[j]的赋值, 它应当始终和同步指向数组中第j位的地址。

但在上图中我只更新了j却没有更新\$t3.修改如下:

```

    addi $t1, $t1, 1          # ++j
    addi $t3, $t3, 4          # update t3
    j InnerLoop               # jump to test of inner loop

```



成功输出了从大到小排序好的结果。

我的认识：

此类问题属于结果不正确。一般来说是逻辑错误。同样可以通过调试单步观察来找出错误。这个过程和编写C语言程序有相似之处，但由于汇编语言涉及更多更底层的实现，需要更多的指令，容易遗忘某个寄存器的更新等，要格外注意。这样可以通过记忆课本实例、加强练习基本功能逻辑的实现来减少出错。

- 3) 进行了上面第二步的修改后输出了从大到小排序好的结果,但是输出了十一个数字而非十个，多输出了一个2。如下所示：

 Console

```
654 231 23 7 6 4 4 3 3 2 2|
```

解决方法：


很容易想到是重复输出了最后一位，找到相应代码，发现此处逻辑有问题，应当把对*i*==0的判断放在进入*i*循环的初始位置。

```
beq $t0, $zero, ExitOuterLoop # go to ExitOuterLoop if i==0
addiu $t0, $t0, -1             # i-=1
j OuterLoop                    # jump to test of outer loop
```

改正如下：

```
OuterLoop:
    beq $t0, $zero, ExitOuterLoop # go to ExitOuterLoop if i==0
```

得到了正确的输出：

 Console

```
654 231 23 7 6 4 4 3 3 2|
```

我的认识：

这一类错误也是输出结果错误。是由于逻辑顺序颠倒导致的。具体来说，是没能正确“翻译”C语言程序。由于C语言程序的几句话在MIPS汇编语言中要扩展成十几句话，这使得在具体实现时容易出现逻辑错误。同样需要细心和理解以及练习运用。

- 4) 输出结果正确但是输出最后一个数字后没有如预想般输出换行符。

解决方法：

找到相应代码：

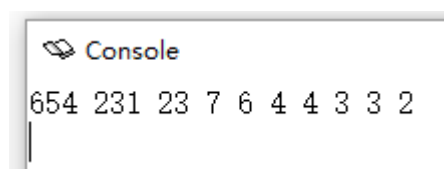
```
la $a0, lineBreak      # load address to $a0
li $v0, 10              # exit
syscall                 # system call
```

一开始我没有看出来哪里错了,进行了调试才意识到在将一个换行符load到\$a0

后我没有执行输出的系统调用命令。改正如下：

```
la $a0, lineBreak      # load address to $a0
li $v0, 4               # prints a character string
syscall                 # system call
li $v0, 10              # exit
syscall                 # system call
```

成功换行：



我的认识：

这是一个非常微小的错误,主要是由于粗心导致。但是如果减少此类简单错误发生的频率可以提高编程效率,应当注意。

## 2. 体会

- 1) 初学MIPS汇编语言,感觉很新鲜。在学C和C++时就感到原来程序设计是即使对一个简单的逻辑也将其层层剖析大卸八块,用最直白的语言告诉一根筋的电脑每一步该怎么做。如今学习汇编语言,感觉更加痛快,又深入了一层,将软硬件结合,将原来的一句话再进行分解,对应到寄存器、存储器等的操作。就像是一个先将书读薄,再将书读厚的探索过程。这让我逐步学习到计算机的组成原理是什么,计算机内部是怎么工作的。这种化繁为简的思想和原理体现在程序和工程设计甚至是生活中的方方面面。例如围棋高手AlphaGo就是一款围棋人工智能程序。这个程序利用“价值网络”去计算局面,用“策略网络”去选择下子。它的“思想”是由一步步最基本的命令组成的。
- 2) 初次接触PCSpim模拟器,感觉很精简。但是光看课件也不是很能理解各部分数据的含义。而在动手操作编写程序时才能真正学习理解。譬如模拟器界面中

的四个部分分别代表什么含义，在执行过程总会有什么变化，有什么作用。代码部分的内存地址、指令代码、指令和伪指令的区别和联系等等都自然会在编写程序时用到，尤其是需要调试时。我感觉这和C或C++语言调试大同小异，都是去观察指令执行到哪一步时哪些变量或数据发生了变化。只是汇编语言涉及到更底层的实现，需要深刻理解存储器中的数据是怎么样存储的，地址高地位的变化等。直到做完实验，回看课件，才发现课件中讲得非常详细和清楚，是做实验加深了我对这些知识的理解。比如实验过程中出现了输出全为同一个数的错误，就需要去找到相应时刻寄存器的变化和存储器中存储的值，并将十六进制ASCII码值转化成十进制再找到对应字符，从而判断出错的原因，这建立在理解存储结构和字符串、地址存储顺序（字符从左到右，地址由低到高）基础上。

- 3) 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，问题的主要原因是对MIPS汇编语言不是很熟悉。打完后翻到书本第二章后面发现有swap()和sort()函数的实例，其所用方法和指令以及指令还有寄存器等的分配都和我有所不同，书上的代码包括注释等都更加规范，可能也更加高效。我的代码可能简洁一些。我很期待日后进一步学习更好的编程方法，不停地改进。总的来说，我觉得这门课很有意思，可以想象也不简单，希望自己在保持学习兴趣的情况下投入它的怀抱。

### 【程序代码】

```
#####
#
#           从内存中读取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。
#
#####
.text                # 代码段 声明
.globl main          # globl指明程序的入口地址 main
main:                # 入口地址 main
    addi $t0, $zero, 9    # i=9

OuterLoop:
    beq $t0, $zero, ExitOuterLoop # go to ExitOuterLoop if i==0
    la $t3,array          # load array's base address to t3
```

```
    addi $t1, $zero, 0          # j=0

InnerLoop:
    slt $t2, $t1, $t0          # if(j<i) t2=1; else t2=0;
    beq $t2, $zero, ExitInnerLoop # go to ExitInnerLoop if j>=i

    lhu $t4, 0($t3)             # t4=array[j]
    lhu $t5, 4($t3)             # t5=array[j+1]
    slt $t6, $t5, $t4           # if(a[j+1]<a[j]) t6=1; else t6=0
    beq $t6, $zero, SkipSwap    # go to SkipSwap if a[j+1]>=a[j]
    #Swap:
    lhu $t7, 0($t3)             # t7(temp)=array[j];
    sw $t5, 0($t3)              # array[j]=array[j+1]
    sw $t7, 4($t3)              # array[j+1]=temp

SkipSwap:
    addi $t1, $t1, 1            # ++j
    addi $t3, $t3, 4            # update t3
    j InnerLoop                 # jump to test of inner loop

ExitInnerLoop:
    lhu $a0, 0($t3)             # load word to a0
    li $v0, 1                   # load immediate, prints $a0 to standard output
    syscall                     # system call

    la $a0, space               # load address to $a0
    li $v0, 4                   # prints a character string to standard output
    syscall

    addiu $t0, $t0, -1          # i-=1
    j OuterLoop                 # jump to test of outer loop

ExitOuterLoop:
    la $t3, array               # load array's base address to t3
    lhu $a0, 0($t3)             # load word to a0
    li $v0, 1                   # load immediate, prints $a0 to standard output
    syscall                     # system call

    la $a0, lineBreak           # load address to $a0
    li $v0, 4                   # prints a character string to standard output
    syscall                     # system call

    li $v0, 10                  # exit
    syscall                     # system call
```

```
.data                                # 数据段
array:                              # 变量名称
    .word 231, 4, 6, 7, 3, 3, 2, 654, 4, 23
space:                              # 字符串定义
    .ascii " "                      #以“空格”字符“0x00”作为终止符结束
lineBreak:
    .ascii "\n"
```