中山大學
SUN YAT-SEN UNIVERSITY

《计算机组成原理实验》

实验报告

（实验一）

学 院 名 称 ：	数据科学与计算机学院
专业（班级）：	17 软件工程 2 班
学 生 姓 名 ：	张伟焜
学　　号 ：	17343155
时　　间 ：	2018　年　10　月　16　日

成 绩 ：

实验一 ： MIPS汇编语言程序设计实验

一. 实验目的

1. 初步认识和掌握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个无符号字数或从内存中读取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

电脑一台，PCSpim仿真器软件一套。

四. 实验过程与结果

设计思想与方法:

在程序设计课程中，对于数字的大小排序问题，常见的一种排序方法为冒泡法。为了提高实验效率，我的设计思想就是，先用C++来实现冒泡排序，再按照C++代码，将其“翻译”成MIPS汇编语言。

首先，冒泡排序C++代码如下:

```
#include<iostream>

using namespace std;

int main() {
    int num_array[10];
    int p, q;
    for (int i = 0; i < 10; i++) {
        cout << "Please input an integer: ";
        cin >> num_array[i];
```

```
    }
    cout << "Bubble sorting: ";
    for (p = 9; p > 0; p--) {
        for (q = 0; q < p; q++) {
            if (num_array[q] > num_array[q + 1]) {
                int temp = num_array[q];
                num_array[q] = num_array[q + 1];
                num_array[q + 1] = temp;
            }
        }
        cout << num_array[q] << " > ";
    }
    cout << num_array[p] << endl;
    return 0;
    }
```

注意到，实现冒泡排序需要两个for循环（外循环和内循环），同时还出现了if语句的跳转。此外，C++中还使用了数组来存放这10个整数。

在MIPS汇编语言中，没有for，if，等语句。但通过阅读MIPS资料发现，伪指令：bne, bgt, bgtz 等可以实现变量的比较及跳转，这对冒泡排序的实现是非常有必要的。我们可以用条件判断加跳转来实现for循环。

分析：

1.在汇编语言中，我们可已通过申请连续空间来模拟数组的实现。一个整数占四个字节，存放十个整数需40个字节，因此使用.space 40来申请40字节空间

2.在输入过程中，我们应使用变量来记录当前整数个数，因此令$t1=10，记录数组最大容量，初始化$t4=0，每读取一个整数，$t4加1，当$t4==$t1则输入完毕。

3.同理，$t2,$t3用来记录外层循环和内层循环的次数，当$t2==$t3，内层循环结束，当$t2==0，外层循环结束。

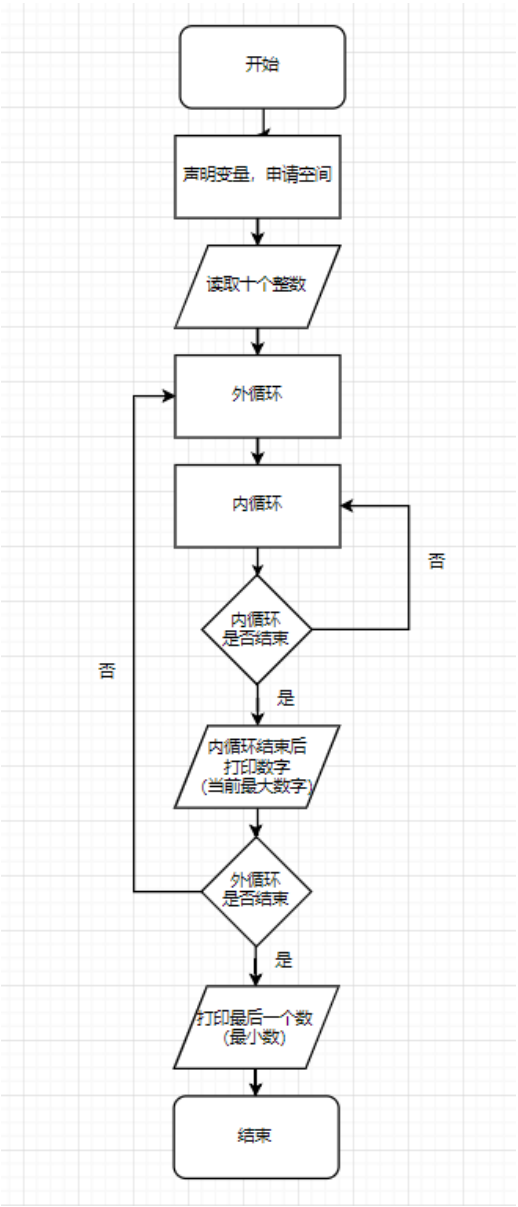
4.每次内循环，比较相邻两个数字的大小，若前面的数大，则两数交换。这样内循环结束时，最大的数字会排在最后，把此数打印出来。

5.每完成一次外循环，$t2-1，因为没完成一次外循环，就有一个数字确定了位置。

6.对于输入和输出，使用系统调用。读入整数$v0=5；打印整数$v0=1；打印字符串$v0=4，$v0=10，退出。

7.关于比较大小和跳转。使用伪指令bne, bgt, bgtz。

实验步骤流程图：



实验结果：

Console

```
Please input an integer: 12
Please input an integer: 54
Please input an integer: 87
Please input an integer: 65
Please input an integer: 23
Please input an integer: 1
Please input an integer: 58
Please input an integer: 0
Please input an integer: 999
Please input an integer: 245
Bubble sorting: 999 > 245 > 87 > 65 > 58 > 54 > 23 > 12 > 1 > 0
```

五. **实验心得**

遇到的问题:

1.调试输入部分时,刚开始无论按什么数字,都不会在屏幕上显示,起初以为是PCSpim的安装有问题，后来才发现在运行时，必须将输入法调为英文输入，才能输入数字。

2.在交换两个数字时，我以为要像C++一样引入临时变量来存放一个数，就把代码写的繁琐了，深入了解MIPS后，发现可以间接寻址，直接对$t9进行操作，只需控制偏移量，配合sw，将寄存器中的数字放在对应的内存地址中，很轻松的可以实现数字的交换。

3.最开始将$t2（外循环的目标书）初始值设为10，发现程序出错，经分析后，发现出现了越界现（num_array[i+1]）

4.将打印操作放入外循环后，发现只能打印九个数字，最小的那个数没有打印出来。经分析，是因为外循环只执行了9次，最后需要单独补充打印最后一个数。

思考及收获:

因为刚开始接触MIPS，对许多指令还不是很熟悉，后来我想到先写出C++代码，然后对其进行转换，这大大提高了实验效率。

通过C++和MIPS的转换,我发现MIPS的代码量明显大于C++。而且感觉相比较于C++，只看MIPS代码，难以看出程序的层次结构（可能是因为C++有“{}”）。

不像C++声明数组时只需说明要存放几个数，MIPS需要自己计算并申请内存空间（.space）。此外，MIPS中若想打印字符串，也需提前将字符串内容存放起来。

汇编语言中不能直接声明数组类型的变量，需要设计者自己计算所需空间大小，然后使用.space 申请内存空间。

对于代码优化的问题，可以先在C++中完成在转换成MIPS，比如实现打印操作时不需要再写一个循环，直接放在外层循环中，可以减少代码量。而一些MIPS指令也可以进行替代优化。比如:

原代码:

```
sub $t0,$t1,$t4   #判断是否读入了10个数
bgtz $t0,input
```

优化代码:

```
bne $t4,$t1,input #当t4与t1不相等，则继续输入
```

原代码：

```
sub   $t0,$t2,$t3
bgtz $t0,inloop
```

优化代码：

```
bne $t2,$t3,inloop#判断内层循环是否全部完成(是否$t2=t3)
```

当我最初接触MIPS的时候，感觉它的指令很多，很繁杂。但是，通过本次实验，我熟悉了基本的常用指令。同时，我发现按照英文原义记指令很高效。比如：bgez(be greater than zero), bne(be not equal)等等。

不足之处&未解决的问题：

每次只允许用户输入一个数字，没能实现连续输入多个数字。

【程序代码】

```
.data
  num_array:
    .space 40    #存放10个数字的数组，一个整数4个字节，10个整数40个字节
  str1:
    .asciiz "Please input an integer: "      #提示用户输入一个整数
  str2:
    .asciiz "Bubble sorting: "
  greater_than:
    .asciiz ">"
  space:
    .asciiz " "

.text
.globl main
```

```
main:
   addi $t1,$zero,10 #t1=10，记录数组的最大容量
   addi $t2,$zero,9   #t2=9，记录外层循环的目标数
   addi $t3,$zero,0   #t3=0，记录内层循环次数
   addi $t4,$zero,0   #t4=0，记录当前数组中元素个数
   la $t9, num_array #将数组的地址（num_array[0]赋给t9）

input:   #用户输入
   la $a0,str1          #输出str1
   li $v0,4
   syscall

   li $v0,5    #功能5，从键盘读入一个整数
   syscall

   sw $v0, 0($t9) #将v0转存到num_array[0]
   addi $t9,$t9,4#使$9指向数组下一位(相邻4个字节)
   addi $t4,$t4,1#$t4计数加一
   bne $t4,$t1,input #当t4与t1不相等，则继续输入

   la $a0,str2          #输出str2，准备开始Bubble Sorting
   li $v0,4
   syscall

outloop:
   la $t9,num_array  #load address 使指针重新指向数组头
   move $t3,$zero     #将内层循环的计数变量初始化为0
```

```
inloop:
   lw $t5,0($t9)  #取出a[i]
   lw $t6,4($t9)  #取出a[i+1]
   bgt $t6,$t5,judge_inloop   #如果a[i+1]小于a[i]则进行交换，如果a[i+1]大于a[i]则完成一次内层循环，跳转到judge_inloop
   sw $t6,0($t9)
   sw $t5,4($t9)

judge_inloop:
   addi $t3,$t3,1#t3++，表示inloop执行一次
   addi $t9,$t9,4#数组指针指向下一位
   bne $t2,$t3,inloop      #判断内层循环是否全部完成(是否$t2=t3)

print:   #如果内层循环全部完成，则数组的最后一位一定是最大的，此时打印
   lw $a0,0($t9)  #load word
   li $v0,1    #打印整数
   syscall
   la $a0,space   #打印空格
   li $v0,4
   syscall
   la $a0,greater_than    #打印大于号
   li $v0,4
   syscall
   la $a0,space   #打印空格
   li $v0,4
   syscall

judge_outloop:
   #outloop完成一次,num_array[$t2]已经为最大的数字,下次循环不需再进行比较
```

```
addiu $t2,$t2,-1
bgtz $t2,outloop    #t2>0 则外循环未完成 继续进行外循环


la $t9,num_array #打印最后一个数
lw $a0,0($t9)
li $v0,1
syscall

#退出
li $v0,10
syscall
```