《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
学 生 姓 名 ：	郝裕玮
学　　号 ：	18329015
专业（班级）：	19 计算机类一（1）班
时　　间 ：	2020　年　10　月　28　日

成 绩：

实验一 ： MIPS汇编语言程序设计

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QEMU模拟器和GNU/Linux下x86-MIPS交叉编译环境的使用。

二. 实验内容

【排序】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。

例如对于输入

4 1 3 1 6 5 17 9 8 6

其输出为：

1 1 3 4 5 6 6 8 9 17

三. 实验器材

1. MARS模拟器
2. Java环境

四. 实验分析与设计

1. 实验分析

本实验需要我们对输入的10个无规律整数进行排序，并对其进行升序输出。考虑到数据量不大，所以我们选用冒泡排序来解决该问题。

冒泡排序的原理为：

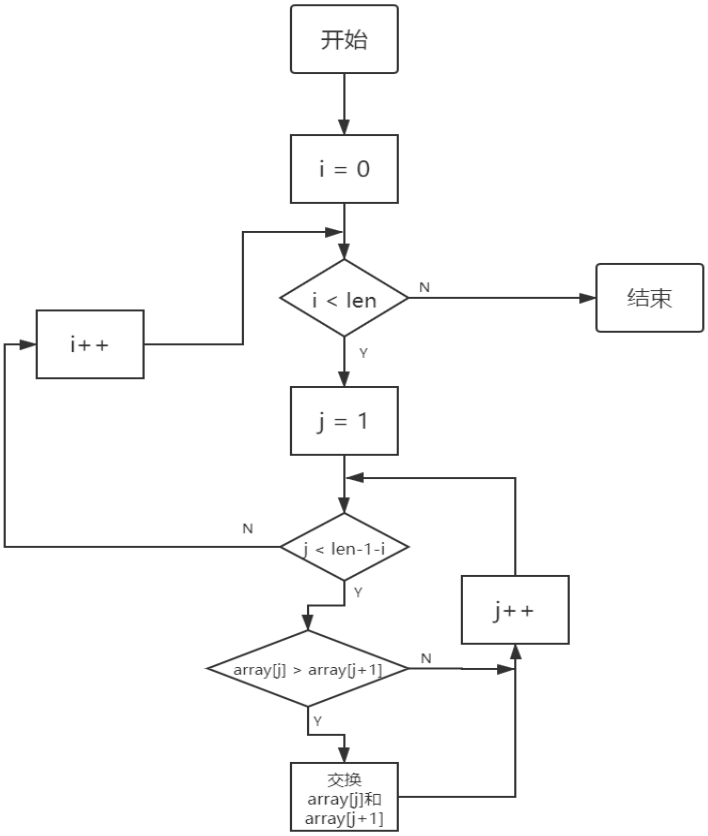
①比较相邻的元素。如果第一个比第二个大，就交换他们两个。

②对每一对相邻元素做同样的工作，从开始第一对到结尾的最后一对。在这一点，最后的元素应该会是最大的数。

③针对所有的元素重复以上的步骤，除了最后一个。

④持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。

所以我们可画出如下流程图：



所以，结合原理和流程图，我们可写出本题所对应的C语言代码：

```
#include <iostream>
using namespace std;
int main(){
    int i=0,j=0,k=0,temp=0,len;
    cin>>len;
    int array[len];
    for(i=0;i<10;i++){
        cin>>array[i];
    }
    for(i=0;i<len;i++){/* 外循环为排序趟数，len 个数进行 len-1 趟 */
        for(j=0;j<len-1-i;j++){/* 内循环为每趟比较的次数，第 i 趟比较 len-i
次 */
            if(array[j]>array[j+1]){
                temp=array[j];
                array[j]=array[j+1];
                array[j+1]=temp;
            }
        }
    }
```

```
    for(i=0;i<len;i++){
        cout<<array[i]<<" ";
    }
    cout<<endl;
    return 0;
}
```

对该代码进行转换，即可得到【程序代码】中的汇编代码。

同时该代码也针对实验要求进行了进一步的优化,即此程序可以对任意数量的数字进行排序（内存分配了400个字节，所以最多可对100个数字进行排序）。而并非只能对10个数字进行排序

对于汇编代码的具体解释与说明我已全部放在代码中（每一行代码都有具体注释），所以不再在此部分讲解代码，如给助教和老师带来不便还请谅解，谢谢！

五. 实验心得

1.实验中所碰到的问题及如何解决：

（1）对于 qemu 的调试使用不熟练，以至于实验前期在代码编译运行和 debug 上浪费了很多时间。之后学习了如何在 MARS 模拟器上进行调试，解决了这一问题。

（2）可能是由于个人能力有限，又或者是学习还不到位，我最终无法在 qemu 中手动实现 print_int 函数（虽然个人感觉自己写的是对的）。最终在咨询两位助教后确认了本次实验在其他模拟器实现也可以（不局限于 qemu）。所以最后我在 MARS 模拟器上完成了本次实验。因为 MARS 模拟器比 qemu 模拟器的限制更少，可以直接调用 read_int,print_int 等函数。所以这也直接帮我解决了我在 qemu 中遇到的最困难的一部分（算法部分并无太多实现困难）。

（3）在算法实现部分，我一开始使用的是内外层循环判断条件为i < len , j < len. 也成功运行并得到了正确的结果。但之后我想对代码进行优化，即把内层循环判断条件改为j < len-1-i;这样可以减少代码的复杂度。但一开始我是这么改的：

```
innerLoop:
  sub $t7, $t7, $t1  # 内部循环终止条件：j<len-1-i
  beq $t2, $t7, nextLoop  # 若j==len-1-i，则终结内层循环，进入nextLoop,准备重新进入外层循环
  lw $t3, 0($t5)  # 将t5中的地址所指向的主存单元(即array[j])存入t3
  lw $t4, 4($t5)  # 将t5地址+4所得的地址所指向的主存单元（即array[j+1]）存入t4
```

即在innerLoop的第一行加上sub $t7, $t7, $t1，但发现排序结果并不正常。经过debug我才意识到这里的len-1-i中的len-1应为定值，而我用t7这一不断变化的寄存器是无法存储len-1的，所以我将中间的t7换成了t8，让t8来存储定值len-1，最后发现成

功运行。

```
innerLoop:
  sub $t7, $t8, $t1  # 内部循环终止条件：j<len-1-i
  beq $t2, $t7, nextLoop  # 若j==len-1-i，则终结内层循环，进入nextLoop,准备重新进入外层循环
  lw $t3, 0($t5)  # 将t5中的地址所指向的主存单元(即array[j])存入t3
  lw $t4, 4($t5)  # 将t5地址+4所得的地址所指向的主存单元（即array[j+1]）存入t4
```

2. 实验收获

通过本次实验，我初步掌握了汇编语言和高级语言之间的联系，懂得了如何将高级语言（如C语言）的代码转换为汇编语言，同时我对各种寄存器的工作原理以及应用场景也有了初步的认识（$v0,$v1；$a0—$a3；$t0—$t9），并且懂得了如何通过系统调用来实现特定的输入输出。明白了内存在汇编语言中的重要性：给每个寄存器分配多少内存，该寄存器的作用域等等（内存何时销毁）。

在写汇编语言的过程中，我感受到了和高级语言的很多差别：高级语言更加简洁明了，易于理解，但内存方面的具体实现对我们来说是黑箱，是完全封闭的。汇编语言则从底层开始写起，让我明白了这样一个简单的冒泡排序算法是如何一步步通过分配内存和各种操作来实现的。

最后，这次实验中我对于qemu的使用还存在一些疑问，希望我能在后续的学习中逐步解决这些疑问。

【程序代码】

```
.data
  array: .space  400  # 每个整数给予4个字节的空间,400个字节则最多可对100个数字进行排序
  seperate: .asciiz " " # 输出时每个数字之间需要输出空格隔开
  UI: .asciiz "Please enter the numbers of elements you want to sort:\n" # 输入提示

.text
.globl main
main:
  la $a0, UI # 打印UI中的提示输入字符串
  li $v0, 4  # 调用print_string，打印字符串
  syscall
  li $v0, 5  # 调用read_int,读取输入的整数
  syscall
  move $t6,$v0  # 将数字个数存储到t6中
  la $t0, array  # 将数组首地址存储到t0中
```

```
  move $t1, $zero  # 循环变量 i 初始化为 0 并存储到 t1 中
  move $t2, $zero  # 循环变量 j 初始化为 0 并存储到 t1 中
  move $t5, $t0  # 将数组的头指针存储到 t5 中
  subi $t7, $t6, 1  # t7 的值为数组的长度减 1（因为数组的最后一个元素为
array[t6-1]）
  # 为称呼简便，以下的数组长度均简称为 len

inputNumber:
  li $v0, 5  # 读取每个数组元素
  syscall
  sw $v0, 0($t0)  # 将 v0 存入 t0 中的地址指向的主存单元中。
  addi $t0, $t0, 4  # 指向下一个连续的 4 字节地址用于存储新的输入整数
  addi $t1, $t1, 1  # i++
  blt $t1, $t6, inputNumber # 若 i<数组长度
  move $t1, $zero  # 若不满足 blt（即 i==len），则 i=0，让 i 能继续被用于 outerLoop
循环（在进入 outerLoop 之前完成 i 的初始化）
  subi $t8, $t6, 1 # 用于内部循环的终止条件（即 j<len-1-i）(t8==len-1)

outerLoop:
  la $t5, array # t5=array[0]
  move $t2, $zero  # j=0
  blt $t1, $t6, innerLoop  # 若 i<len，则进入 innerLoop
  beq $t1, $t6, init  # 若 i==len，则进入 init

innerLoop:
  sub $t7, $t8, $t1  # 内部循环终止条件：j<len-1-i
  beq $t2, $t7, nextLoop  # 若 j==len-1-i，则终结内层循环，进入 nextLoop,准
备重新进入外层循环
  lw $t3, 0($t5)  # 将 t5 中的地址所指向的主存单元(即 array[j])存入 t3
  lw $t4, 4($t5)  # 将 t5 地址+4 所得的地址所指向的主存单元（即 array[j+1]）存
入 t4
  bgt $t3 $t4 swap  # 若 t4 中的值小于 t3（即 array[j]>array[j+1]）将 t4 和 t3
进行交换（即将 array[j]和 array[j+1]进行交换）
  addi $t5, $t5, 4  # 若不满足 bgt（即 array[j]<=array[j+1]）,则数组的头指针
偏移 4 个字节（即指向位置+1）
  addi $t2, $t2, 1 # j++
  j innerLoop  # 再次进入 inner 循环

nextLoop:
  addi $t1, $t1, 1  # i++
  j outerLoop  # 内部循环（innerLoop）结束，重新回到外部循环（outerLoop）

swap:
  sw $t3, 4($t5)  # 将 t3 存入 t5 中的地址+4（array[j+1]）所指向的主存单元中。
```

```
  sw $t4, 0($t5)  # 将 t4 存入 t5 中的地址（array[j]）所指向的主存单元中。
  addi $t5, $t5, 4  # 数组的头指针偏移 4 个字节（即指向位置+1）
  addi $t2, $t2, 1 # j++
  j innerLoop  #再次进入 inner 循环

init:
  la $t0, array  # t0=array[0]
  addi $t1, $zero, 0  # i=0，使得 i 能继续用于 printResult 循环（在进入
printResult 之前完成 i 的初始化）

printResult:
  # 按顺序打印数组元素
  lw $a0, 0($t0)  # 将 t0 中的地址（初始值为数组头地址 array[0]）所指向的主存
单元存入 a0
  li $v0, 1  # 调用 print_int，打印整数
  syscall
  #打印空格隔开数组元素
  la $a0, seperate  #将 seperate 中的字符串存入 a0 中
  li $v0, 4  # 调用 print_string，打印字符串
  syscall
  addi $t0, $t0, 4  # t0 地址偏移 4 个字节（即 t0 指向的数组元素位置+1）
  addi $t1, $t1, 1  # i++
  blt $t1, $t6, printResult  #若 i 仍处于数组下标范围内（0—(数组长度-1)），则
再次重新进入 printResult 进行循环

  li $v0, 10  # 调用 exit，程序结束
  syscall
```