汇编程序设计实验

实验简介

学生信息

杨雅儒，无73，2017011071 。

实验目的

- 了解MIPS处理器的硬件结构，学会用底层思维实现指令需求
- 学会如何调试汇编程序

实验任务

实现三个汇编程序——冒泡排序、快速排序以及归并排序。

实验环境及注意事项

在本机Linux上进行测试，使用vim编写，MARS运行。**另外麻烦助教注意一下，由于本机Linux上的mips不支持相对路径，于是使用了一个绝对路径 `\home\yyr\Work\asm\a.in` 。提交的代码以本地能够正常执行为准，故希望助教测试时首先更改代码首部的文件名字符串。**

另外注意本次实验中使用的测试程序如view和check为Linux下编译，且check.cpp调用了system()函数，同系统相关，不可直接在Windows下使用。

测试

三份代码全部在本地测试超过50组大数据(包括n=1000的极限数据)，以及30组小数据(包括n=1的极限数据)，基本可以保证没有问题。

程序内部的调试代码

首先是在三个程序内部写了调试代码，会在命令行输出排序前和排序后的数组。其中冒泡排序和快速排序用的函数为printArrToScreen，输出数组；归并排序用的函数为printListToScreen，用于遍历并输出链表中的数据。两个函数都会以空格作为分隔输出，且末尾带有换行。

view.cpp

view.cpp，用于显示二进制文件（while不停读取4bytes直到文件结尾），用法为：

```
./view a.in
```

check.cpp

对拍程序，用C++实现，可以自动地不停测试数据，用法：

```
./check QuickSort.s
```

实验流程及算法思路

冒泡排序

总体即按照课件上的C代码实现，使用`Arr: .space 4050`创建了足够大的数组，书写了主函数、sort函数、chswap函数以及调试用的printArrToScreen函数。

主函数

主函数做的工作就是从文件中读入数组，调用printArrToScreen函数，然后调用sort函数进行排序，再次调用printArrToScreen函数之后再将数组写入到文件，最后结束程序（需要注意文件读写后需要关闭文件）。

sort函数

参数：$a0传入数组基址，$a1传入n。

内容：sort函数做的工作就是利用冒泡排序的算法，对整个Arr数组进行从小到大排序。冒泡一共进行n轮，每轮从右向左冒泡，其中利用chswap函数进行判断并确定是否交换数组的两相邻位置。

ckswap函数

参数：$a0传入表示可能交换$a0和$a0+4地址的值 (不满足顺序时交换)。

内容：判断内存中$a0地址的值是否大于$a0+4地址的值，若是则进行交换。

printArrToScreen函数

参数：$a0传入数组基址，$a1传入n表示数组长度。

内容：将整个数组以空格间隔的形式输出到屏幕，并且最后输出换行符。

快速排序

总体结构上同冒泡排序类似，也是使用`Arr: .space 4050`创建了足够大的数组，书写主函数、sort函数以及调试用的printArrToScreen函数，只是将其中sort函数更换为了快速排序的逻辑。

主函数

主函数做的工作就是从文件中读入数组，调用printArrToScreen函数，然后调用sort函数进行排序，再次调用printArrToScreen函数之后再将数组写入到文件，最后结束程序（需要注意文件读写后需要关闭文件）。

sort函数

参数：$a0传入数组基址，$a1传入left, $a2传入right。

内容：利用快速排序，将数组的[left,right]区间内的数进行排序。快排的思路是每次选出位于中间的数，将比它小的数都放到它左边，比它大的数都放到它右边，然后再递归调用即可。

printArrToScreen函数

同前。

归并排序

归并排序与前面两个排序算法差异较大，主要是使用链表实现。首先生成整体链表，每次等分成两部分，对左右部分链表分别用sort函数递归，之后便成为了有序链表，再使用merge函数进行合并即可。

内容包括主函数、sort函数、merge函数、printListToScreen函数。

主函数

主函数做的工作即通过不停读入，建立初始的链表，在输出调试信息后调用sort函数获得排序之后的链表，再次

输出调试信息后写回文件并结束程序。

sort函数

参数：传入$a0表示链表首指针，传回$v0表示排序后的首指针。

内容：首先通过小步大步同时走的算法，找到链表中间位置，然后从这里断开链表，在分别递归左右链表之后，通过merge函数将两个有序链表进行合并。

merge函数

参数：$a0传入左链表首地址l_head，$a1传入右链表首地址r_head，$v0传出合并后链表首地址。

内容：通过在左链表中不断找到右链表的各部分应当插入的位置，进行插入实现。

printListToScreen函数

参数：$a0传入链表首地址。

内容：通过遍历该链表，将其中的数据以空格分隔的形式输出到屏幕，并且最后加上换行符。

问题总结与思考

调试问题

- **调试时回转到上一步可能会出问题**，例如文件打开时，回转到syscall之前再向下运行则会出现打开失败的情况。
- 在Linux系统下，文件打开时 "./a.in" 似乎不行，改成绝对路径即可运行；而在Windows系统下，测试中使用"a.in"可以。
- 开数组的时候，在.data下使用类似 `Arr: .space 4050`，但要注意**数组要开在其它东西前面否则会出错**，具体原因暂不清楚。
- 注意jal时一定要记得保存ra，而不只是t0~t9等。

思考

本次实验中出现了一个小问题——快排比冒泡排序慢。

经过思考以及先验知识，我认为问题应当出在lw和sw上，我所写的快排代码中，因为需要递归，故大量含有lw和sw，而这两个指令是非常慢的。

但是那么为什么用C++写快排的时候就不会这么慢呢？ 我认为这是因为编译器在此做了一些优化，避免了lw和sw的频繁使用。

附：代码

view.cpp

```C++
#include
#include
#include
```

include

using namespace std;

FILE *file ; int main(int argc, char* argv[]){ if(argc<2) { cout << "Please specify the file name.\n" ; return 0 ; } file = fopen(argv[1],"rb") ; if(file==NULL){ cout << "An error occured while reading the file.\n" ; return 0 ; } int t; while(fread(&t,4,1,file)){ cout << t << ' ' ; } cout << '\n' ; fclose(file) ; return 0 ; } ```

check.cpp

```C++

# include

# include

# include

# include

# include

using namespace std;

int rd(int l,int r){ return (unsigned)rand()*rand()%(r-l+1)+l ; }

FILE *input,* output ; char inputName[]="a.in" , outputName[]="a.out" ; int n , A[1005] ;

int writeInt(int a, FILE* file){ return fwrite(&a, sizeof(int), 1, file) ; }

int readInt(int &a, FILE* file){ return fread(&a, sizeof(int), 1, file) ; }

int main(int argc, char *argv[]){ srand(time(NULL)+(long long)new int) ; if(argc<2){ cout << "Please specify the file name.\n" ; return 0 ; } for(int i=1;i<=1000000;++i){ cout << "#" << i << ":" << '\n' ; //生成数据 input = fopen(inputName, "wb") ; int n=rd(5,1000) ; writeInt(n, input) ; for(int i=1;i<=n;++i){ A[i]=rd(-10000,10000) ; writeInt(A[i],input) ; } fclose(input) ;

```
    //执行并检查
    system("[-e ./a.out] && rm a.out") ;
    char inst[105] ;
    sprintf(inst,"java -jar Mars4_5.jar %s > log",argv[1]) ;
    system(inst) ;
    output = fopen(outputName,"rb") ;
    if(output==NULL){
        cout << "No output file generated.\n" ;
        return 0 ;
    }
    int a ;
    sort(A+1,A+n+1) ;
    for(int i=1;i<=n;++i){
        if(!readInt(a,output) || a!=A[i]){
            cout << "Wrong answer!\n" ;
```
```

```
            return 0 ;
        }
    }
    if(readInt(a,output)){
        cout << "Longer than std.\n" ;
        return 0 ;
    }
    fclose(output) ;
    cout << "Accepted!!!\n" ;
    cout << '\n' ;
}
return 0 ;
```

} ```

冒泡排序

``` .data Arr: .space 4050 space: .asciiz " " line: .asciiz "\n" infile: .asciiz "/home/yyr/Work/asm/a.in" outfile: .asciiz "/home/yyr/Work/asm/a.out"

.text .global main

main:
li $v0, 13 la $a0, infile li $a1, 0 #读取 li $a2, 0 #模式，设定为0即可 syscall addu $a0, $0, $v0 li $v0, 14 la $a1, Arr li $a2, 4 #读取四个字节，为n syscall la $a1, Arr lw $s1, 0($a1) #s1=n li $v0, 14 la $a1, Arr sll $a2, $s1, 2 syscall #读取数组 li $v0, 16
syscall #关闭文件

```
la $a0, Arr
move $a1, $s1
jal   printArrToScreen #调试

li $s7, 1
beq $s7, $s1, skipSort #特殊处理n=1的情况
la $a0, Arr
move $a1, $s1
jal   sort   #排序
```

skipSort:

```
la $a0, Arr
move $a1, $s1
jal   printArrToScreen #调试

li $v0, 13
la $a0, outfile
li $a1, 1   #写入
li $a2, 0   #模式，设定为0即可
syscall
addu $a0, $0, $v0
li $v0, 15
la $a1, Arr
sll $a2, $s1, 2
syscall
li $v0, 16
syscall   #关闭文件
```
```

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

sort: # a0传入数组基址，a1传入n addu $t0, $0, $a0 # t0为基址 addu $t1, $0, $a1 # t1=n li $t2, 0 # t2=0 作为i loopi: sll $t3, $t1, 2 # t3 初始赋为倒数第二个位置的地址 addu $t3, $t3, $t0 subi $t3, $t3, 8 loopj: addu $a0, $0, $t3 # a0=t3 (当前地址) addu $s0, $0, $ra # 暂存ra jal ckswap # 调用ckswap进行判断 addu $ra, $0, $s0 # 还原ra subiu $t3, $t3, 4 bleu $t0, $t3, loopj # t0(基址)<=t3(当前地址)时跳转到loopj

```
addiu $t2, $t2, 1  # t2=t2+1
blt   $t2, $t1, loopi     # t2(i)<t1(n) 时跳转到loopi
jr    $ra
```

ckswap: # a0传入表示可能交换a0和a0+4地址的值 (不满足顺序时交换) lw $t8, 0($a0) lw $t9, 4($a0) ble $t8, $t9, exitSwap #t8<=t9时跳转到exitSwap(即不交换) sw $t8, 4($a0) sw $t9, 0($a0) exitSwap: jr $ra

printArrToScreen: # a0传入数组基址，a1传入n addu $t6, $0, $a0 # t6存基址 addu $t7, $0, $a1 # t7存n li $t8, 0 # t8(i)=0 loop: sll $t9, $t8, 2 addu $t9, $t9, $t6 # t9赋值为目标地址 li $v0, 1 lw $a0, 0($t9) syscall # 打印数字 li $v0, 4 la $a0, space syscall # 打印空格 addiu $t8, $t8, 1 # i=i+1 blt $t8, $a1, loop # t8(i)<a1(n)时跳转到loop li $v0, 4 la $a0, line # 打印换行 syscall jr $ra ```

快速排序

``` .data Arr: .space 4050 space: .asciiz " " line: .asciiz "\n" infile: .asciiz "/home/yyr/Work/asm/a.in" outfile: .asciiz "/home/yyr/Work/asm/a.out"

```
.text
.global main

main:
li $v0, 13
la $a0, infile
li $a1, 0   #读取
li $a2, 0   #模式，设定为0即可
syscall
addu $a0, $0, $v0
li $v0, 14
la $a1, Arr
li $a2, 4   #读取四个字节，为n
syscall
la $a1, Arr
lw $s1, 0($a1) #s1=n
li $v0, 14
la $a1, Arr
sll $a2, $s1, 2
syscall   #读取数组
li $v0, 16
syscall   #关闭文件

la $a0, Arr
move $a1, $s1
jal   printArrToScreen #调试
```
```

```
la    $a0, Arr
li    $a1, 0
subi  $a2, $s1, 1
jal   sort          #排序

la    $a0, Arr
move  $a1, $s1
jal   printArrToScreen  #调试

li    $v0, 13
la    $a0, outfile
li    $a1, 1        #写入
li    $a2, 0        #模式，设定为0即可
syscall
addu  $a0, $0, $v0
li    $v0, 15
la    $a1, Arr
sll   $a2, $s1, 2
syscall
li    $v0, 16
syscall             #关闭文件

li    $v0, 10
syscall             # exit


sort: # a0传入数组基址，a1传入left, a2传入right
move  $t0, $a0      #t0存数组基址
move  $t1, $a1      #t1存left
move  $t2, $a2      #t2存right
move  $t3, $t1      #t3(i)=t1(left)
```

move $t4, $t2 #t4(j)=t2(right) add $t5, $t3, $t4 srl $t5, $t5, 1
sll $t5, $t5, 2 add $t5, $t5, $t0 lw $t5, 0($t5) #t5(mid)=Arr[(i+j)/2] sortLoop: #while(i<=j)
startLoopi: sll $t6, $t3, 2 add $t6, $t6, $t0 lw $t6, 0($t6) #t6=Arr[i] bge $t6, $t5, endLoopi addiu $t3, $t3, 1 #t3(i)=t3(i)+1 b startLoopi endLoopi: #while(arr[i]mid) bgt $t3, $t4, endIf1 #t3(i)>t4(j)时跳出 sll $t6, $t3, 2 add $t6, $t6, $t0 lw $t8, 0($t6) #t8=Arr[i] sll $t7, $t4, 2 add $t7, $t7, $t0 lw $t9, 0($t7) #t9=Arr[j] sw $t8, 0($t7) #Arr[j]=t8 sw $t9, 0($t6) #Arr[i]=t9 addiu $t3, $t3, 1 #t3(i)=t3(i)+1 subi $t4, $t4, 1 #t4(j)=t4(j)-1 endIf1: bge $t1, $t4, endIf2 #t1(left)>=t4(j)时跳出 subi $sp, $sp, 28 sw $t0, 0($sp) sw $t1, 4($sp) sw $t2, 8($sp) sw $t3, 12($sp) sw $t4, 16($sp) sw $t5, 20($sp) sw $ra, 24($sp) move $a0, $t0 move $a1, $t1 move $a2, $t4 jal sort #sort(Arr,left,j) lw $ra, 24($sp) lw $t5, 20($sp) lw $t4, 16($sp) lw $t3, 12($sp) lw $t2, 8($sp) lw $t1, 4($sp) lw $t0, 0($sp) addi $sp, $sp, 28 endIf2: bge $t3, $t2, endIf3 #t3(i)>=t2(right)时跳出 subi $sp, $sp, 28 sw $t0, 0($sp) sw $t1, 4($sp) sw $t2, 8($sp) sw $t3, 12($sp) sw $t4, 16($sp) sw $t5, 20($sp) sw $ra, 24($sp) move $a0, $t0 move $a1, $t3 move $a2, $t2 jal sort #sort(Arr,i,right) lw $ra, 24($sp) lw $t5, 20($sp) lw $t4, 16($sp) lw $t3, 12($sp) lw $t2, 8($sp) lw $t1, 4($sp) lw $t0, 0($sp) addi $sp, $sp, 28 endIf3: jr $ra

```
printArrToScreen:         # a0传入数组基址，a1传入n
addu  $t6, $0, $a0     # t6存基址
addu  $t7, $0, $a1     # t7存n
li    $t8, 0          # t8(i)=0
printLoop:
sll   $t9, $t8, 2
addu  $t9, $t9, $t6   # t9赋值为目标地址
li    $v0, 1
```

lw $a0, 0($t9) syscall # 打印数字 li $v0, 4 la $a0, space syscall # 打印空格 addiu $t8, $t8, 1 # i=i+1 blt $t8, $a1, printLoop # t8(i)<a1(n)时跳转到printLoop li $v0, 4 la $a0, line # 打印换行 syscall jr $ra
```

## 归并排序

``` .data num: .space 4 space: .asciiz " " line: .asciiz "\n" infile: .asciiz "/home/yyr/Work/asm/a.in" outfile: .asciiz "/home/yyr/Work/asm/a.out"

.text .global main

main:
li $v0, 13 la $a0, infile li $a1, 0 #读取 li $a2, 0 #模式，设定为0即可 syscall move $s2, $v0 #s2=fd move $a0, $s2 li $v0, 14 la $a1, num li $a2, 4 #读取四个字节，为n syscall lw $s1, 0($a1) #s1=n

```
li    $v0, 9
li    $a0, 8
syscall            #新建一个结点
sw    $0, 4($v0)   #next指针初始为0
move  $s3, $v0     #s3存放首指针
move  $s4, $s3     #s4存放当前指针

li    $v0, 14
move  $a0, $s2
la    $a1, num
li    $a2, 4
syscall
lw    $s7, 0($a1)  #s7临时存储读入的一个数
sw    $s7, 0($s3)  #放入首指针的数据中
li    $s5, 2       #s5存放循环变量i
```

inputLoop: bgt $s5, $s1, endInputLoop #s5(i)>s1(n)时退出循环 li $v0, 9 li $a0, 8 syscall #新建一个结点 sw $0, 4($v0) #next指针初始为0 sw $v0, 4($s4) #上一个位置的next指向当前结点地址 move $s4, $v0 #s4=v0，指向当前位置

```
li    $v0, 14
move  $a0, $s2
la    $a1, num
li    $a2, 4
syscall
lw    $s7, 0($a1)  #s7临时存储读入的一个数
sw    $s7, 0($s4)  #放入当前结点的数据中

addiu $s5, $s5, 1
b     inputLoop
```

endInputLoop: move $a0, $s2 #a0=s2(fd) li $v0, 16
syscall #关闭文件

```
move  $a0, $s3
jal   printListToScreen  #调试

move  $a0, $s3
jal   sort
```

```
move  $s3, $v0

move  $a0, $s3
jal   printListToScreen  #调试

li    $v0, 13
la    $a0, outfile
li    $a1, 1        #写入
li    $a2, 0        #模式，设定为0即可
syscall
move  $s2, $v0      #s2=fd
move  $s4, $s3      #s4指向当前位置
```

outputLoop: li $v0, 15 move $a0, $s2 move $a1, $s4 li $a2, 4 syscall #输出当前结点数据 lw $s4, 4($s4) #s4=s4->next bnez $s4, outputLoop

```
li    $v0, 16
syscall            #关闭文件

li    $v0, 10
syscall            # exit
```

merge: # a0传入左链表首地址l_head，a1传入右链表首地址r_head，v0传出合并后链表首地址 move $t8, $a0 move $t9, $a1 li $v0, 9 li $a0, 8 syscall #新建一个虚拟结点head sw $t8, 4($v0) #next指针初始为l_head move $t0, $v0 #t0作为p_left move $t1, $t9 #t1作为p_right move $t2, $v0 #t2作为head mergeLoop1: mergeLoop2: lw $t9, 4($t0) #t9=p_left->next beqz $t9, endMergeLoop2 lw $t9, 0($t9) #t9=t9->val lw $t8, 0($t1) #t8=p_right->val bgt $t9, $t8, endMergeLoop2 lw $t0, 4($t0) #p_left=p_left->next b mergeLoop2 endMergeLoop2: lw $t9, 4($t0) #t9=p_left->next bnez $t9, endMergeIf1 sw $t1, 4($t0) #p_left->next=p_right b endMergeLoop1 #break endMergeIf1: move $t3, $t1 #t3作为p_right_temp mergeLoop3: lw $t9, 4($t3) #t9=p_right_temp->next beqz $t9, endMergeLoop3 lw $t9, 0($t9) #t9=t9->val lw $t8, 4($t0) #t8=p_left->next lw $t8, 0($t8) #t8=t8->val bgt $t9, $t8, endMergeLoop3 lw $t3, 4($t3) #p_right_temp=p_right_temp->next b mergeLoop3 endMergeLoop3: lw $t4, 4($t3) #t4作为temp_right_pointer_next lw $t9, 4($t0) #t9=p_left->next sw $t9, 4($t3) #p_right_temp->next=p_left->next sw $t1, 4($t0) #p_left->next=p_right move $t0, $t3 #p_left=p_right_temp move $t1, $t4 #p_right=temp_right_pointer_next beqz $t1, endMergeLoop1 # if(p_right==NULL) break; b mergeLoop1 endMergeLoop1: lw $v0, 4($t2) #return head->next jr $ra

sort: #传入a0表示首指针head，传回v0表示排序后的首指针 move $t0, $a0 #t0作为head lw $t9, 4($t0) #t9=head->next bnez $t9, endSortIf1 move $v0, $a0 jr $ra #return head; endSortIf1: move $t1, $t0 #t1作为stride_1_pointer move $t2, $t0 #t2作为stride_2_pointer sortLoop1: lw $t9, 4($t2) #t9=stride_2_pointer->next beqz $t9, endSortLoop1 move $t2, $t9 #stride_2_pointer=t9 lw $t9, 4($t2) #t9=stride_2_pointer->next beqz $t9, endSortLoop1 move $t2, $t9 #stride_2_pointer=t9 lw $t1, 4($t1) #stride_1_pointer=stride_1_pointer->next b sortLoop1 endSortLoop1: lw $t2, 4($t1) #stride_2_pointer=stride_1_pointer->next sw $0, 4($t1) #stride_1_pointer->next=NULL

```
move  $a0, $t0
subi  $sp, $sp, 16
sw    $t0, 0($sp)
sw    $t1, 4($sp)
sw    $t2, 8($sp)
sw    $ra, 12($sp)
jal   sort
lw    $ra, 12($sp)
```

```
lw    $t2, 8($sp)
lw    $t1, 4($sp)
lw    $t0, 0($sp)
addiu $sp, $sp, 16
move  $t3, $v0     #t3作为l_head=msort(head);

move  $a0, $t2
subi  $sp, $sp, 20
sw    $t0, 0($sp)
sw    $t1, 4($sp)
sw    $t2, 8($sp)
sw    $t3, 12($sp)
sw    $ra, 16($sp)
jal   sort
lw    $ra, 16($sp)
lw    $t3, 12($sp)
lw    $t2, 8($sp)
lw    $t1, 4($sp)
lw    $t0, 0($sp)
addiu $sp, $sp, 20
move  $t4, $v0     #t4作为r_head=msort(stride_2_pointer);

move  $a0, $t3
move  $a1, $t4
subi  $sp, $sp, 4
sw    $ra, 0($sp)
jal   merge
lw    $ra, 0($sp)
addiu $sp, $sp, 4
jr    $ra          #return merge(l_head,r_head);
```

printListToScreen: # a0传入链表首地址 addu $t6, $0, $a0 # t6存当前地址 printLoop: li $v0, 1 lw $a0, 0($t6) syscall # 打印数字 li $v0, 4 la $a0, space syscall # 打印空格 lw $t6, 4($t6) bnez $t6, printLoop # 下一个非空的时候继续循环 li $v0, 4 la $a0, line # 打印换行 syscall jr $ra ```