汇编大作业报告

2016011058 漆耘含

一、 快速排序

1. 代码思路

快速排序的核心是寻找一个参考值，将其放在正确的位子上，再递归处理其左列和右序列，递归的边界长度是 1

代码的思路和大作业中给出的代码思路差不多，先找出位于中间的数，以此为基准，找出左边比中间数大的，找出右边比中间数小的，然后两两交换，如果还没有到中间的数，则继续寻找左边比中间数大的和右边比中间数小的，进行交换，之后重复上述操作。

在走完一趟之后，进行递归处理，把左序列和右序列进行递归。因为使用的是汇编语言，需要自己进行压栈操作，因此每一次递归之前，需要进行压栈操作，把 left，right 和当前的 ra 的值压栈，在执行完一次左边序列和右序列的递归之后，进行出栈操作，并且把 ra 的值保存为出栈的 ra 的值，返回上一级递归，一直重复下去，直到返回主函数。

2. 代码展示

```
##qsort##
.data
    in_buff: .space 512
    output_buff: .space 512
    input_file: .asciiz "E:/a.in"
    out_file: .asciiz"E:/a.out"

############################
.text
    la $a0,input_file   #input_file 是一个字符串
    li $a1,0        #0 为写入
    li $a2,0        #只读
```

```
        li $v0,13       #13, 打开文件
        syscall

        move $a0,$v0      #把文件描述符载入到$a0 中
        la $a1,in_buff        #in_buff 为数据缓存区
        li $a2,4      #读取四个字节
        li $v0,14     #14, 读取文件
        syscall

        lw $t1,in_buff        #读取第一个数
        mul $t2,$t1,4     #把第一个数乘 4，然后把剩下的数全部读
出来
        addi,$a1,$a1,4        #把 in_buff 的地址+4
        move $a2,$t2      #把要读的字节数保存在 a2 中
        li $v0,14     #14，读取文件
        syscall

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

    ######################################################
    #下面构建链表
    ######################################################
        addi $a0,$zero,8  #申请八比特的空间
        addi $v0,$zero,9  #9，申请空间编号
        syscall
        move $s0,$v0      #记录头结点指针
        move $t2,$s0      #记录当前节点指针
        move $t0,$zero        #循环变量赋初值为 0
```

```
    la $a1,in_buff          #把 in_buff 的地址保存到 a1 中
    addi $a1,$a1,4          #a1 的地址加四
    addi $t1,$t1,-1         #t1 保存的是一共有多少个数（从 0 开始）

new:
    addi $a0,$zero,8  #申请八比特的空间
    addi $v0,$zero,9
    syscall
    mul $t3,$t0,4       #循环变量乘 4
    add $a3,$a1,$t3         #把地址调整到要保存的那个数的地址
    lw $t4,0($a3)       #把要保存的数读出来
    sw $t4,0($v0)       #把它保存到结点的前四字里面
    sw $zero,4($v0)         #把结点的后四字保存为 0，代表链表尾
    sw $v0,4($t2)       #把新节点加到原有链表上
    move $t2,$v0        #把链表指针移到新加的结点
    addi $t0,$t0,1          #循环变量+1
    ble $t0,$t1,new         #t1 里面保存的是数的个数
    lw $t2,4($s0)       #t2 重新指向头结点的后继

    ############################################################
    #t2 指向的是头结点的后继
    #t1 保存的是一共的元素个数（从 0 开始）
    ############################################################
    li,$t5,0        #t5 为 left
    move $t6,$t1        #t6 为 right
    addi $sp,$sp,-12  #压栈
    sw $t5,4($sp)       #把 left 压栈
```

```
    sw $t6,0($sp)      #把 right 压栈
    jal quickSort1         #转到 quickSort1 函数
    j write            #在 quickSort 函数结束之后，进行保存文
件函数

quickSort1:
    sw $ra,8($sp)      #把调用该函数的$ra 压栈
quickSort:
    lw $t5,4($sp)
    lw $t6,0($sp)
    move $a1,$t5       #a1 相当于 i，从 left 开始
    move $a2,$t6       #a1 相当于 j，从 right 开
    add $t3,$a1,$a2
    srl $t3,$t3,1      #t3 为 mid 的标号
    move $v1,$t3
    jal search     #搜寻中间值 mid
    lw $s1,0($t7)      #s1 保存中间值

L1:
    sub $s2,$a2,$a1        #如果 i<=j，则继续
    bltz $s2,continue
L2:
    move $t3,$a1       #寻找第 i 个数
    jal search
    lw $s3,0($t7)
    sub $s4,$s1,$s3        #比较它和 mid 的大小关系
    addi $a1,$a1,1         #i++
    bgtz $s4,L2        #如果第 i 个数小于 mid，则继续循环
    addi $a1,$a1,-1
```

```
L3:
    move $t3,$a2      #寻找第 j 项
    jal search
    lw $s5,0($t7)
    sub $s4,$s1,$s5       #判断与 mid 的大小关系
    addi $a2,$a2,-1       #如果第 j 个数大于 mid，则继续循环
    bltz $s4,L3
    addi $a2,$a2,1

    sub $s4,$a2,$a1       #如果 i<=j，则交换两个数，并在链表
中更新
    bltz $s4,L1

    move $t3,$a1
    jal search    #search 函数是找到第 i 个元素

    lw $s3,0($t7)
    move $t3,$a2
    jal search    #search 函数是找到第 j 个元素

    lw $s5,0($t7)     #下面交换两个数
    move $s6,$s3
    move $s3,$s5
    move $s5,$s6
    move $t3,$a1      #把这两个数保存回去
    jal search
    sw $s3,0($t7)
    move $t3,$a2
```

```
    jal search
    sw $s5,0($t7)
    addi $a1,$a1,1
    addi $a2,$a2,-1
    lw $t2,4($s0)
    j L1            #进入 L1 的循环

continue:           #继续递归操作
    sub $s4,$a2,$t5
    bgtz $s4,left     #对左边进行递归
    sub $s4,$t6,$a1
    bgtz $s4,right        #对右边进行递归
    lw $ra,8($sp)
    addi $sp,$sp,12       #在递归完成后，出栈操作
    jr $ra

left:
    addi $sp,$sp,-12  #在递归之前，进行压栈
    sw $ra,8($sp)
    sw $t5,4($sp)
    sw $a2,0($sp)
    j quickSort

right:
    addi $sp,$sp,-12  #在递归之前，进行压栈
    sw $ra,8($sp)
    sw $a1,4($sp)
    sw $t6,0($sp)
    j quickSort
```

```
search:                 #搜寻第 t3 个数
    li $t4,0
    lw $t7,4($s0)
Loop1: beq $t4,$t3,return       #如果相同就返回
    lw $a0,4($t7)
    beq $a0,$zero,return
    lw $t7,4($t7)     #转到下一个结点
    addi $t4,$t4,1
    j Loop1

return:
    jr $ra          #在整个 quickSort 函数结束之后，返回主函数
write:
    li $t0,0
    la $s1,output_buff
    move $s0,$t2
save:               #把排序了的保存在 out_buff 中
    mul $t2,$t0,4
    add $t3,$s1,$t2
    lw $t7,0($s0)
    sw $t7,0($t3)
    lw $s0,4($s0)
    addi $t0,$t0,1
    sub $s2,$t0,$t1
    bgtz $s2,out
    j save

out:                #输送到二进制文件中去
```

```
la $a0,out_file
li $a1,1
li $a2,0
li $v0,13
syscall

addi $t2,$t2,4
move $a0,$v0
la $a1,output_buff
move $a2,$t2
li $v0,15
syscall

li $v0,16
syscall
```

3. 问题处理

a. 因为是第一次使用汇编语言编程，对于 syscall 的操作还不是很熟悉，在进行文件读取的时候，我把二进制文件中的内容都读出来了，并且也保存在 in_buff 里面了，我用一个循环输出的时候，程序一直报错，说 lw 没有读取到一个字的内存，刚开始非常奇怪，后来经过调试发现，是因为在数据段（.data）的时候，我开的内存区是写在文件名之后，内存没有对齐，因此在 lw 读取的时候，会出错。

b. 我对与 j，jal，jr 这三者的区别不是很清楚，这就导致了在刚开始编写的时候，出了很多错误，后来找了本书来看看，搞清楚了这三者的区别。

c. 在压栈的时候，出了很多 bug，刚开始，对于递归的过程只有一个大概模糊的了解，在第一次写完之后发现出错了，后来画了一张图来理解递归的过程，慢慢调试，最终才写出来了。

二、 归并排序

1. 代码思路

链表的归并排序的核心是将链表分成前后两段，分别排序后对两个有序链表归并，递归边界是当前链表长度为 1

代码的思路和大作业中的代码思路差不多，先是调用 msort 函数，找到其中点，然后把它划分成左右两个链表，然后通过递归把它不断分解成左右两个子链表，递归的边界是传过去的子链表长度为 1，当进行完一次左右递归之后，会调用 merge 函数，将将个链表进行合并，最开始是长度 1 合并到长度 2，之后再从长度 2 合并到长度 4，一直到结束。在递归的时候，我用了两个栈，第一个栈是进行函数递归的，在每一次递归开始之前进行压栈，在返回的时候出栈；第二个栈是用来保存 merge 函数返回的地址（赋给 left 和 right 的），每返回一次就压栈保存下来，在后面要用的时候再出栈。

2. 代码展示

```
##msort##
.data
    in_buff: .space 512
    out_buff: .space 512
    record :.space 1024
    input_file: .asciiz"E:/a.in"
    output_file:.asciiz"E:/a.out"

.text
    la $k1,record
    la $a0,input_file #input_file 是一个字符串
    li $a1,0
    li $a2,0
    li $v0,13
```

```
        syscall

        move $a0,$v0
        la $a1,in_buff
        li $a2,4
        li $v0,14
        syscall

        lw $t1,in_buff        #读取第一个数
        mul $t2,$t1,4     #把第一个数乘 4，然后把剩下的数全部读
出来
        addi,$a1,$a1,4        #把 in_buff 的地址+4
        move $a2,$t2      #把要读的字节数保存在 a2 中
        li $v0,14     #14，读取文件
        syscall

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

    ####################################################
    #下面构建链表
    ####################################################

        addi $a0,$zero,8  #申请八比特的空间
        addi $v0,$zero,9  #9，申请空间编号
        syscall
        move $s0,$v0      #记录头结点指针
        move $t2,$s0      #记录当前节点指针
        move $t0,$zero        #循环变量
```

```
    la $a1,in_buff
    addi $a1,$a1,4

    addi $t1,$t1,-1

new:
    addi $a0,$zero,8  #申请八比特的空间
    addi $v0,$zero,9
    syscall
    mul $t3,$t0,4     #循环变量乘 4
    add $a3,$a1,$t3
    lw $t4,0($a3)     #把新节点接入到原来的链表上
    sw $t4,0($v0)
    sw $zero,4($v0)
    sw $v0,4($t2)
    move $t2,$v0
    addi $t0,$t0,1
    ble $t0,$t1,new       #t1 里面保存的是数的个数

    lw $t2,4($s0)     #t2 重新指向头结点的后继

    move $v1,$t2

#############################################################
#t2 指向的是头结点的后继
#t1 保存的是一共的元素个数（从 0 开始）
#############################################################
#以下两行相当于主函数
```

```
    jal c1
    j finish

c1:
    addi $sp,$sp,-8       #进行压栈操作，把 head 和当前的 ra 保存在栈里面
    sw $t2,4($sp)
    sw $ra,0($sp)
    move $t4,$t2
    jal msort


#a3 是 msort 返回的 head
msort:              #归并主函数
    lw $t3,4($t4)
    bne $t3,$zero,c_msort    #如果为空，那么返回
    move $a3,$t4      #如果为空，那么进行出栈操作
    lw $ra,0($sp)
    addi $sp,$sp,8
    jr $ra

c_msort:
    move $t5,$t4      #$t5=strike_2_pointer
    move $t6,$t4      #$t6=stride_1_pointer

loop1:              #找中点
    lw $t3,4($t5)
    beq $t3,$zero,c2_msort
    lw $t5,4($t5)
```

```
    lw $t3,4($t5)
    beq $t3,$zero,c2_msort
    lw $t5,4($t5)
    lw $t6,4($t6)
    j loop1

c2_msort:          #分为两个链表，并且把右边的链表头压栈
    lw $t5,4($t6)
    sw $zero,4($t6)
    addi $sp,$sp,-4
    sw $t5,0($sp)

#s1 是 l_head,s2 是 r_head
    jal l_head
    move $s1,$a3       #把返回的结果保存在函数调用栈里面，
（本程序中有两个栈，此栈是自己开的——record）
    addi $k1,$k1,4
    sw $s1,0($k1)

    jal r_right
    move $s2,$a3       #把返回的结果保存在一个栈里面
    addi $k1,$k1,4
    sw $s2,0($k1)

    j merge

c3_msort:
    lw $ra,0($sp)      #返回，出栈
    addi $sp,$sp,8
```

```
    jr $ra

l_head:
    addi $sp,$sp,-8       #左链表压栈操作
    sw $t4,4($sp)
    sw $ra,0($sp)
    jal msort

r_right:
    lw $t4,0($sp)     #右链表压栈操作
    addi $sp,$sp,4
    addi $sp,$sp,-8
    sw $t4,4($sp)
    sw $ra,0($sp)
    jal msort


#s1 是 l_head,s2 是 r_head,s3 是 head
#s4 是 p_left,s5 是 p_right
merge:
    lw $s1,-4($k1)
    lw $s2,0($k1)
    addi $k1,$k1,-8

    addi $a0,$zero,8 #申请八比特的空间
    addi $v0,$zero,9
    syscall
    move $s3,$v0
    sw $s1,4($s3)
```

```
    move $s4,$s3
    move $s5,$s2

loop2:
    loop3:              #寻找左链中的插入位子
        lw $t7,4($s4)
        beq $t7,$zero,c_loop2
        lw $t9,0($t7)
        lw $v1,0($s5)
        sub $v0,$t9,$v1
        bgtz $v0,c_loop2
        lw $s4,4($s4)
        j loop3
    c_loop2:            #如果达到左链尾部，右链直接接上
        lw $t7,4($s4)
        bne $t7,$zero,c_merge
        sw $s5,4($s4)
        j c3_merge

#s6 是 p_right_temp
c_merge:
    move $s6,$s5
loop4:                  #寻找右链待插入片段
    lw $t7,4($s6)
    beq $t7,$zero,c2_merge
    lw $t9,0($t7)
    lw $t8,4($s4)
    lw $v1,0($t8)
    sub $v0,$t9,$v1
```

```
    bgtz $v0,c2_merge
    lw $s6,4($s6)
    j loop4

#s7 是 temp_right_pointer_next
c2_merge:              #完成插入操作
    lw $s7,4($s6)
    lw $k0,4($s4)
    sw $k0,4($s6)
    sw $s5,4($s4)
    move $s4,$s6
    move $s5,$s7
    beq $s5,$zero,c3_merge
    j loop2

c3_merge:
    lw $a3,4($s3)
    j c3_msort



finish:
    la $s1,out_buff
    li $t0,0
save:
    mul $t2,$t0,4
    add $t3,$s1,$t2
    lw $t7,0($a3)
    sw $t7,0($t3)
    lw $a3,4($a3)
```

```
        addi $t0,$t0,1
        sub $s2,$t0,$t1
        bgtz $s2,out
        j save

    out:                    #保存在二进制文件中
        la $a0,output_file
        li $a1,1
        li $a2,0
        li $v0,13
        syscall

        addi $t2,$t2,4
        move $a0,$v0
        la $a1,out_buff
        move $a2,$t2
        li $v0,15
        syscall

        li $v0,16
    syscall
```

3. 问题处理

1. 在进行链表处理的时候出现了一些问题，在进行递归的时候，我不太清楚到底什么该入栈，这个问题困扰了我很久，后来慢慢看代码，理清楚了。我的解决方案是使用两个栈，一个是用来保存函数调用的$sp，另一个是我自己创的一个栈，用来保存merge返回的指针。刚开始我用的一个栈，全压在一起，发现行不通，后来放成两个栈，运行就正常了。

三、 冒泡排序

1. 代码思路

本程序的思路就是很正常的冒泡排序，代码和课件上的差不多，我将它完善了一下

2. 代码展示

```
##冒泡排序##
.data
    input_buff:.space 512

.text
    la $t5,input_buff     #这是我自己给的一组数，没有经过文件读写操作
    li $t6,2          #保存的数为2 7 0 9
    sw $t6,0($t5)
    li $t6,7
    sw $t6,4($t5)
    li $t6,0
    sw $t6,8($t5)
    li $t6,9
    sw $t6,12($t5)

    la $a0,input_buff
    li $a1,4
    jal sort
    j finish

sort:
    addi $sp,$sp,-20      #保存原来的值，在完成之后恢复
```

之前的值

```
        sw $ra,16($sp)
        sw $s3,12($sp)
        sw $s2,8($sp)
        sw $s1,4($sp)
        sw $s0,0($sp)

        move $s0,$zero          #给循环变量赋值，s0 为 i
    loopbody1:
        bge $s0,$a1,exit1    #如果 s0>a1，则跳转出循环

        addi $s1,$s0,-1         #s0 为 j,在内存循环开始之
前，赋初值 i-1
    loopbody2:
        blt $s1,$zero,exit2     #如果 j<0 则跳转
        sll $t1,$s1,2
        add $t2,$a0,$t1
        lw $t3,0($t2)           #读取两个元素值
        lw $t4,4($t2)
        ble $t3,$t4,exit2    #进行比较

        move $s2,$a0
        move $s3,$a1
        move $a0,$s2
        move $a1,$s1
        jal swap            #调用交换函数

        addi $s1,$s1,-1
        j loopbody2
```

```
exit2:
    addi $s0,$s0,1
    j loopbody1
exit1:
    lw $ra,16($sp)              #恢复在函数调用之前的这些值
    lw $s3,12($sp)
    lw $s2,8($sp)
    lw $s1,4($sp)
    lw $s0,0($sp)
    addi $sp,$sp,20
    jr $ra              #返回到主函数

swap:                   #交换函数
    sll $t1,$a1,2
    add $t1,$a0,$t1
    lw $t0,0($t1)
    lw $t2,4($t1)
    sw $t2,0($t1)
    sw $t0,4($t1)
    jr $ra

finish:                     #进行结果输出
    la $t5,input_buff
    lw $a0,0($t5)
    li $v0,1
    syscall
    lw $a0,4($t5)
    li $v0,1
    syscall
```

```
        lw $a0,8($t5)
        li $v0,1
        syscall
        lw $a0,12($t5)
        li $v0,1
syscall
```