



《计算机组成原理与接口技术实验》

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 : 陈明亮

学 号 : 16340023

专业(班级) : 16 软件工程一(1) 班

时 间 : 2018 年 04 月 07 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

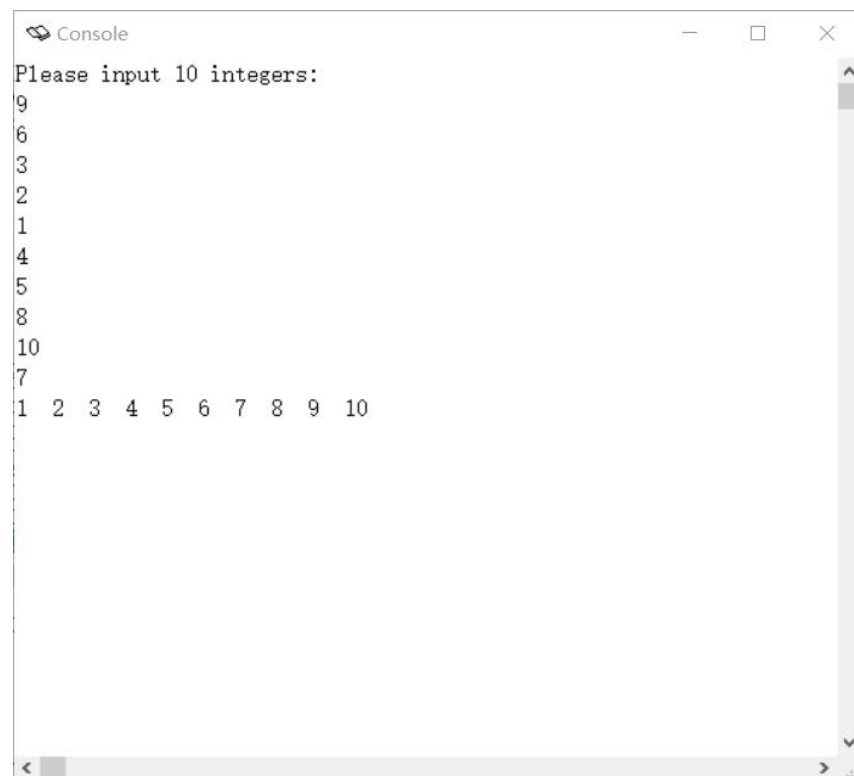
从键盘输入10个无符号数并从大到小进行排序，使用MIPS汇编语言编写快速排序算法，将10个无符号数的排序结果在屏幕上显示出来。

三. 实验器材

电脑一台、PCSpim模拟器软件一套。

四. 实验分析与设计

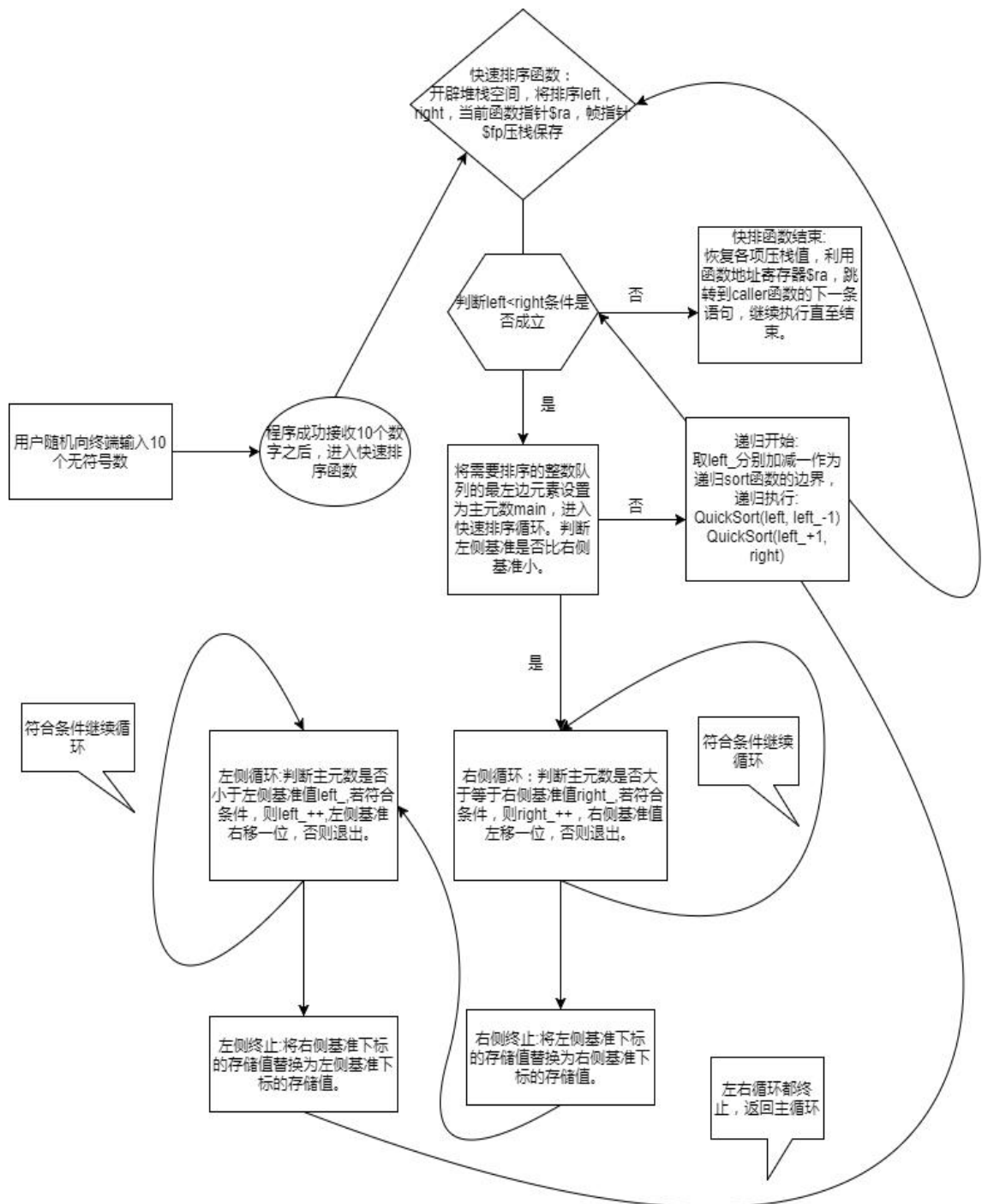
1. 程序执行结果图



```
Console
Please input 10 integers:
9
6
3
2
1
4
5
8
10
7
1 2 3 4 5 6 7 8 9 10
```

2. 程序流程图

本次排序使用了快速排序算法，主要的流程图见下方图片。总述：对于该十个整数，我们首先将排序边界重置为数组的首地址(left)和末地址(right)，利用MIPS循环条件的判断，执行QuickSort(left,right)，并建立主元数main(通常为left下标对应的数值)，以及左基准数left_和右基准数right_进行大小判断，将主元数移动到排序位置，左右递归进行，直至结束。



五. 实验心得

在本次实验的过程中，我在编写MIPS汇编指令方面遇到了许多难题，例如不知道系统调用输入输出指令，如何运用.data中的字符串来输出，以提示用户输入10个整数，以及各种指令具体的用处，还有循环条件的跳转等等问题。这些虽然说是初级问题，但对于我们这些MIPS语言新手来说也需要一定时间的摸索和联系。其次就是对于快速排序算法如何从C++语言转换到MIPS汇编语言，其中的算法核心倒是没变，但重要的地方在于对递归函数的参数值的压栈，保护其不被接下来的子程序改变，以及如何在递归函数结束之后，重新返回到caller函数的下一条指令（运用\$ra保存现场函数地址值）。期间遇到的一个大坑就是，在子程序结束返回原程序之后，PCSpim模拟器会对于各个寄存器的值都会重新刷新一次，所以任何你在MIPS递归函数中要保存的值，必须开辟堆栈空间存储，否则是无法真正地保存的。还有在使用PCSpim模拟器的时候，由于是刚开始使用，对于这些很多数字的模拟器未免有些头晕，并且执行完以后重新编译的功能，该模拟器似乎有些问题，需要每次都Reload一遍才能重新开始。但不论如何说，本次实验我都受益匪浅，希望在以后的实验中能够更加熟悉掌握MIPS汇编程序的书写，努力学习计算机组成原理。

【程序代码】

```
#####
# #
# 输入 10 个整数，进行快速排序并输出的 MIPS 汇编程序
# (输出时以空格区别各个整数)
# #
#####

# $s0 for int array store first address
# $s1 for every quick_sort's content of left pointer
# $t0 for array's first address, QuickSort(int left, int right), initial left
# $t1 for array's last address, QuickSort(int left, int right), initial right
# $t2 for quick_sort's temporary left pointer content
# $t3 for quick_sort's temporary right pointer content
# $t5 for increasing input count
# $t6 for array size 10
# $t7 for quick_sort's temporary $t0 store
```

```
# $t8 for quick_sort's temporary $t1 store
# $t9 for temporary result store

.text
.globl main

main:
    la $s0, array_space # Give $s0 the array space
    move $t0, $s0 # Initialize left parameter $t0, $t1
    move $t1, $s0
    addi $t5, $zero, 0 # Initialize array size and condition size
    addi $t6, $zero, 10
    li $v0, 4 # Output tip string
    la $a0, input_msg
    syscall

input_loop: # Loop for input integers
    li $v0, 5 # System call for integers input
    syscall
    sw $v0, 0($t1) # Store into array
    addi $t1, $t1, 4 # Array first address increasement
    addi $t5, $t5, 1
    bne $t5, $t6, input_loop # Loop condition
    addi $t1, $t1, -4 # Reset $t1
    addi $sp, $sp, -4 # Allocate stack 4 space
    sw $ra, 0($sp)
    jal quick_sort # Start quick sort
    lw $ra, 0($sp)
    addi $sp, $sp, 4 # Give back stack space
    j output_loop # Sort end, jump to output

quick_sort: # void QuickSort(int left, int right)
    addi $sp, $sp, -64 # Allocate stack space
    sw $ra, 28($sp) # Store the stack pointer of the last function
    sw $fp, 36($sp) # Store the base pointer of current registers' content
    sw $t0, 16($sp) # Push left parameter into stack, avoid modified by others
    sw $t1, 32($sp) # Push right parameter into stack, avoid modified by others
    addi $fp, $sp, 64 # Current base pointer up move 32 bits, to be seperated
                        # from other recursive functions with the same ename
    slt $t9, $t0, $t1 # $t9 to store the result of left < right
    bne $zero, $t9, less_than # Jump to another less function
    j sort_end

less_than: # Sort Less Loop, while(left < right)
```

```
    move $t7, $t0 # Temporary initialize
    move $t8, $t1
    lw $s1, 0($t0) # int main = store[left]

while_less_con:
    slt $t9, $t7, $t8 # if(left < right)
    bne $t9, $zero, while_less # Jump into while
    j con_end

while_less:
    while_less_right: # Right part loop, while(main > store[right])
        slt $t9, $t7, $t8 # Loop condition check1 if left < right
        beq $t9, $zero, right_end
        lw $t3, 0($t8)
        slt $t9, $t3, $s1 # Loop condition check2 if main >= store[right]
        bne $zero, $t9, right_end
        addi $t8, $t8, -4 # right--
        j while_less_right

    right_end:
        lw $t3, 0($t8) # store[left] = store[right]
        sw $t3, 0($t7)

    while_less_left: # Left part loop, while(main < store[left])
        slt $t9, $t7, $t8 # Loop condition check1 if left < right
        beq $t9, $zero, left_end
        lw $t2, 0($t7)
        slt $t9, $t2, $s1 # Loop condition check2 if main < store[left]
        beq $zero, $t9, left_end
        addi $t7, $t7, 4 # left++
        j while_less_left

    left_end:
        lw $t2, 0($t7) # store[right] = store[left]
        sw $t2, 0($t8)
        j while_less_con

con_end:
    sw $s1, 0($t7) # store[left] = main
    sw $t7, 8($sp) # Push left into stack
    lw $t0, 16($sp) # Pop $t0 from stack
    addi $t7, $t7, -4
    move $t1, $t7
    jal quick_sort # QuickSort($t0, left-1)
```

```
lw $t7, 8($sp) # Pop left from stack
lw $t1, 32($sp) # Pop $t1 from stack
addi $t7, $t7, 4
move $t0, $t7
jal quick_sort # QuickSort(left+1, $t1)

sort_end:
lw $ra, 28($sp) # Load stack push values
lw $t0, 16($sp)
lw $t1, 32($sp)
lw $fp, 36($sp)
addi $sp, $sp, 64 # Give back stack space
jr $ra # Jump to last recursive function address

output_loop:
li $v0, 1 # Output array[i]
lw $a0, 0($s0)
syscall

la $a0, sep_sign # Output space sign seperater
li $v0, 4
syscall

addiu $s0, $s0, 4 # Output condition check
addiu $t5, $t5, -1
bne $zero, $t5, output_loop

li $v0, 10 # End Program
syscall

.data
array_space: .space 40 # Allocate space for array

sep_sign: .asciiz " " # Space seperate

input_msg: .asciiz "Please input 10 integers: \n"
```