**中山大学数据科学与计算机学院本科生实验报告**

**（2019学年秋季学期）**

课程名称：**计算机组成原理实验**  任课教师：郭雪梅 助教：汪庭葳、刘洋旗

|  |  |  |  |
| --- | --- | --- | --- |
| 年级&班级 | **2018级1班** | 专业(方向) | **计算机类** |
| 学号 | **18308045** | 姓名 | **谷正阳** |
| 电话 | **13355426001** | Email | [**Guzy0324@163.com**](mailto:Guzy0324@163.com) |
| 开始日期 | **2019.10.16** | 完成日期 | **2019.10.16** |

**一、实验题目**

计算机结构与组成 实验3

1. **实验目的**

本次实验希望给大家更多关于函数调用练习的机会，特别是哪些东西需要入栈，另外还有逻辑操作的练习.

**三、实验内容**

**1.** 实验步骤

实验1.

文件swap.s 提供了一个调用你将编写的程序的代码模板。你可以在此程序上，添加你要写的swap代码，以方便测试。.

void swap (int \*px, int \*py) {

int temp;

temp = \*px;

\*px = \*py;

\*py = temp;

}

swap.s

.data

n1: .word 14

n2: .word 27

.text

main:

la $a0,n1

la $a1,n2

jal swap

li $v0,1 # print n1 and n2; should be 27 and 14

lw $a0,n1

syscall

li $v0,11

li $a0,' '

syscall

li $v0,1 #$a0:显示的整数值

lw $a0,n2

syscall

li $v0,11 #$a0:显示的字符

li $a0,'\n'

syscall

li $v0,10 # exit

syscall

swap: move $fp, $sp #FRAME POINTER NOW POINTS TO THE TOP OF STACK

addiu $sp,$sp,-16 # ALLOCATE 16 BYTES IN THE STACK

# your code goes here

…

addiu $sp,$sp,16

jr $31

编写汇编代码完成上述程序。由于所有C程序的局部变量都保存在栈中，因此变量temp也应保存在栈中 (未优化时的情况)。

换言之，不能使用$t0（或者其它寄存器）来对应temp。

提示: 一共需要使用6条lw/sw指令。

如果允许使用$t0来保存temp变量，实现程序优化，本题可能会简单很多，本练习的部分目的是考查临时变量的栈存储。

完成后给老师解释你的代码.

实验2. 熟悉MIPS汇编程序开发环境，学习使用MARs工具。知道如何查看内存空间分配

一.用汇编程序实现以下伪代码：要求使用移位指令实现乘除法运算。

Int main ()

{

Int K,Y;

Int Z[50];

Y=56;

For(k=0;k<50;K++) Z[k]=Y-16\*(k/4+210);

}

**2.** 实验原理

实验1：

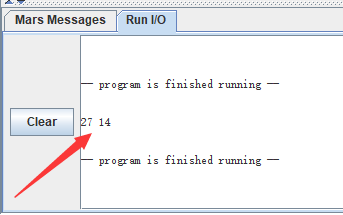


实验2：



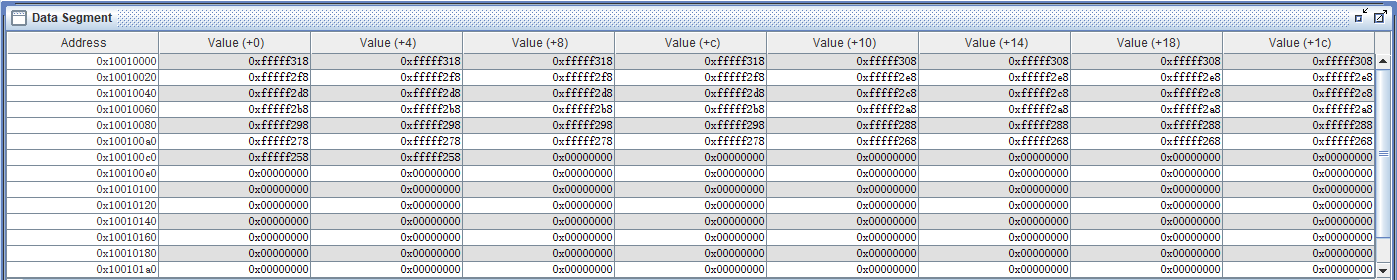
**四、实验结果**

实验1：



实验2：

内存占用情况映像即运行结果显示：

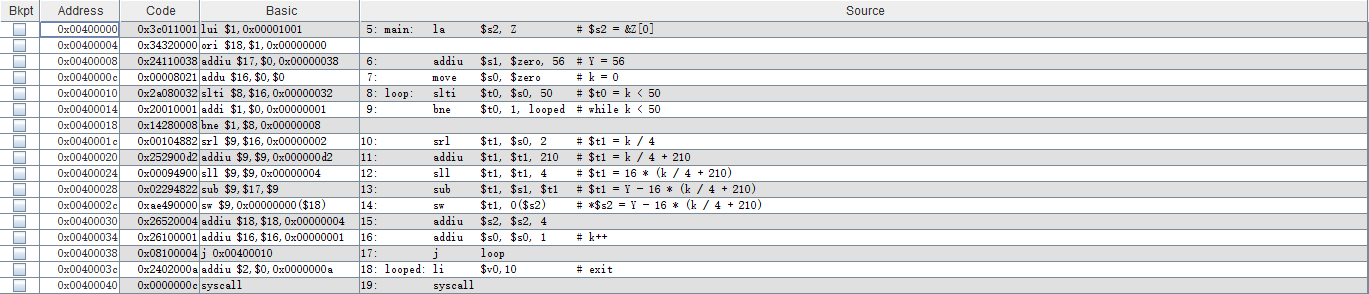


数据段内存映像

表格如下（数值都采用16进制）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 内存地址（16进制） | 变量名 | 值 | 内存地址（16进制） | 变量名 | 值 |
| 0x10010000 | **Z[0]** | 0xfffff318 | 0x10010064 | **Z[25]** | 0xfffff2b8 |
| 0x10010004 | **Z[1]** | 0xfffff318 | 0x10010068 | **Z[26]** | 0xfffff2b8 |
| 0x10010008 | **Z[2]** | 0xfffff318 | 0x1001006c | **Z[27]** | 0xfffff2b8 |
| 0x1001000c | **Z[3]** | 0xfffff318 | 0x10010070 | **Z[28]** | 0xfffff2a8 |
| 0x10010010 | **Z[4]** | 0xfffff308 | 0x10010074 | **Z[29]** | 0xfffff2a8 |
| 0x10010014 | **Z[5]** | 0xfffff308 | 0x10010078 | **Z[30]** | 0xfffff2a8 |
| 0x10010018 | **Z[6]** | 0xfffff308 | 0x1001007c | **Z[31]** | 0xfffff2a8 |
| 0x1001001c | **Z[7]** | 0xfffff308 | 0x10010080 | **Z[32]** | 0xfffff298 |
| 0x10010020 | **Z[8]** | 0xfffff2f8 | 0x10010084 | **Z[33]** | 0xfffff298 |
| 0x10010024 | **Z[9]** | 0xfffff2f8 | 0x10010088 | **Z[34]** | 0xfffff298 |
| 0x10010028 | **Z[10]** | 0xfffff2f8 | 0x1001008c | **Z[35]** | 0xfffff298 |
| 0x1001002c | **Z[11]** | 0xfffff2f8 | 0x10010090 | **Z[36]** | 0xfffff288 |
| 0x10010030 | **Z[12]** | 0xfffff2e8 | 0x10010094 | **Z[37]** | 0xfffff288 |
| 0x10010034 | **Z[13]** | 0xfffff2e8 | 0x10010098 | **Z[38]** | 0xfffff288 |
| 0x10010038 | **Z[14]** | 0xfffff2e8 | 0x1001009c | **Z[39]** | 0xfffff288 |
| 0x1001003c | **Z[15]** | 0xfffff2e8 | 0x100100a0 | **Z[40]** | 0xfffff278 |
| 0x10010040 | **Z[16]** | 0xfffff2d8 | 0x100100a4 | **Z[41]** | 0xfffff278 |
| 0x10010044 | **Z[17]** | 0xfffff2d8 | 0x100100a8 | **Z[42]** | 0xfffff278 |
| 0x10010048 | **Z[18]** | 0xfffff2d8 | 0x100100ac | **Z[43]** | 0xfffff278 |
| 0x1001004c | **Z[19]** | 0xfffff2d8 | 0x100100b0 | **Z[44]** | 0xfffff268 |
| 0x10010050 | **Z[20]** | 0xfffff2c8 | 0x100100b4 | **Z[45]** | 0xfffff268 |
| 0x10010054 | **Z[21]** | 0xfffff2c8 | 0x100100b8 | **Z[46]** | 0xfffff268 |
| 0x10010058 | **Z[22]** | 0xfffff2c8 | 0x100100bc | **Z[47]** | 0xfffff268 |
| 0x1001005c | **Z[23]** | 0xfffff2c8 | 0x100100c0 | **Z[48]** | 0xfffff258 |
| 0x10010060 | **Z[24]** | 0xfffff2b8 | 0x100100c4 | **Z[49]** | 0xfffff258 |

代码段内存映像



**五、实验感想**

更加明确了指针变量和一般的变量的不同，对addressing mode有了更深的理解。mips模拟c语言时将寄存器当作变量，将内存当作第一层解引用的指针（类似register direct addressing）。实际上这种做法是不好的，因为寄存器数量有限，c语言是将所有变量存在内存里，变量类似register direct addressing，指针类似memory indirect addressing。

**附录（流程图，注释过的代码）：**

实验1：

.data

n1: .word 14

n2: .word 27

.text

main:

la $a0,n1

la $a1,n2

jal swap

li $v0,1 # print n1 and n2; should be 27 and 14

lw $a0,n1

syscall

li $v0,11

li $a0,' '

syscall

li $v0,1 #$a0:显示的整数值

lw $a0,n2

syscall

li $v0,11 #$a0:显示的字符

li $a0,'\n'

syscall

li $v0,10 # exit

syscall

swap: move $fp, $sp #FRAME POINTER NOW POINTS TO THE TOP OF STACK

addiu $sp,$sp,-16 # ALLOCATE 16 BYTES IN THE STACK

# your code goes here

#lw $t0, 0($a0) #注释掉的是使用$t0，但$t0不对应temp的6条实现的代码（我认为更符合题意

#sw $t0, 0($sp)

#lw $t0, 0($a1)

#sw $t0, 0($a0)

#lw $t0, 0($sp)

#sw $t0, 0($a1)

sw $a0, 0($sp) #未注释掉的是不使用$t0的11条实现的代码

sw $a1, 4($sp)

lw $a0, 0($a0)

sw $a0, 8($sp)

lw $a0, 0($sp)

lw $a1, 0($a1)

sw $a1, 0($a0)

lw $a1, 4($sp)

lw $a0, 8($sp)

sw $a0, 0($a1)

lw $a0, 0($sp)

addiu $sp, $sp, 16

jr $31

实验2：

代码1：

.data

Z: .space 200

.text

main: la $s2, Z # $s2 = &Z[0]

addiu $s1, $zero, 56 # Y = 56

move $s0, $zero # k = 0

loop: slti $t0, $s0, 50 # $t0 = k < 50

bne $t0, 1, looped # while k < 50

srl $t1, $s0, 2 # $t1 = k / 4

addiu $t1, $t1, 210 # $t1 = k / 4 + 210

sll $t1, $t1, 4 # $t1 = 16 \* (k / 4 + 210)

sub $t1, $s1, $t1 # $t1 = Y - 16 \* (k / 4 + 210)

sw $t1, 0($s2) # \*$s2 = Y - 16 \* (k / 4 + 210)

addiu $s2, $s2, 4

addiu $s0, $s0, 1 # k++

j loop

looped: li $v0,10 # exit

syscall

代码2：

.data

Z: .space 200

.text

main: la $s2, Z # $s2 = &Z[0]

addiu $s1, $zero, 56 # Y = 56

move $s0, $zero # k = 0

loop: srl $t1, $s0, 2 # $t1 = k / 4

addiu $t1, $t1, 210 # $t1 = k / 4 + 210

sll $t1, $t1, 4 # $t1 = 16 \* (k / 4 + 210)

sub $t1, $s1, $t1 # $t1 = Y - 16 \* (k / 4 + 210)

sw $t1, 0($s2) # \*$s2 = Y - 16 \* (k / 4 + 210)

addiu $s2, $s2, 4

addiu $s0, $s0, 1 # k++

slti $t0, $s0, 50 # $t0 = k < 50

bne $t0, 1, looped # while k < 50

j loop

looped: li $v0,10 # exit

syscall

练习3：

.data

.align 2

Str: .asciiz "The sum of square from 1 to 100 is "

.text

main: addiu $t0, $zero, 1 #initialize

mulu $t1, $t0, $t0

addu $t2, $zero, $t1

loop: addiu $t0, $t0, 1

mulu $t1, $t0, $t0 #calculate the square of $t0

addu $t2, $t2, $t1 #add the suare to the sum

bne $t0, 100, loop

print: la $a0, Str #load the Str

li $v0, 4 #System call code 4 for printing a string

syscall

li $v0, 1 #System call code 1 for print integer

move $a0, $t2

syscall

exit: li $v0, 10

syscall

练习4：

version1：

main:

lui $a0,0x8000

jal first1pos

jal printv0

lui $a0,0x0001

jal first1pos

jal printv0

li $a0,1

jal first1pos

jal printv0

add $a0,$0,$0

jal first1pos

jal printv0

li $v0,10

syscall

first1pos: addi $sp, $sp, -12

sw $a0, 0($sp)

sw $t0, 4($sp)

sw $t1, 8($sp)

addi $t0, $zero, 31 #index = 31

loop: beq $t0, -1, looped

slt $t1, $a0, $zero

beq $t1, 1, looped #while (index != -1 && a0 >= 0)

sll $a0, $a0, 1 #a0<<

addi $t0, $t0, -1 #index--

j loop

looped: move $v0, $t0

lw $a0, 0($sp)

lw $t0, 4($sp)

lw $t1, 8($sp)

addi $sp, $sp, 12

jr $ra #return index

printv0:

addi $sp,$sp,-4

sw $ra,0($sp)

add $a0,$v0,$0

li $v0,1

syscall

li $v0,11

li $a0,'\n'

syscall

lw $ra,0($sp)

addi $sp,$sp,4

jr $ra

version2：

first1pos: addi $sp, $sp, -12

sw $a0, 0($sp)

sw $t0, 4($sp)

sw $t1, 8($sp)

addi $t0, $zero, 31 #index = 31

loop: beq $t0, -1, looped

andi $t1, $a0, 0x80000000

beq $t1, 0x80000000, looped #while (index != -1 && (a0 & 0x80000000) != 0x80000000)

sll $a0, $a0, 1 #a0<<

addi $t0, $t0, -1 #index--

j loop

looped: move $v0, $t0

lw $a0, 0($sp)

lw $t0, 4($sp)

lw $t1, 8($sp)

addi $sp, $sp, 12

jr $ra #return index