

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：

助教：

年级&班级	19 计科超算	专业(方向)	
学号	18324034	姓名	林天皓
电话		Email	linth5@mail2.sysu.edu.cn
开始日期	2020.10.16	完成日期	2020.10.23

一、实验题目

实验 6 MIPS 汇编程序设计 mips 实验 3

二、实验目的

- 1.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。
- 2.了解 C 语言语句与汇编指令之间的关系。
- 3.掌握 MIPS 汇编程序设计，掌握 MARs 的调试技术。
- 4.了解 MIPS 汇编语言与机器语言之间的对应关系。
- 5.熟悉常见的 MIPS 汇编指令
- 6.掌握程序的内存映像。

三、实验内容与

1 实验原理

继续通过 MARS4.5 程序解析与执行 mips 汇编程序，在实验内容 1 中，需要熟练运用程序中的堆栈空间和存储与读取内存空间的 lw 和 sw 指令，通过合适的在堆栈中保存值与堆栈中的值恢复到临时的寄存器内，完成两个内存空间中的值 n1 和 n2 的交换过程。

在实验内容 2 中，需要了解在 c 语言中变量的声明，赋值，运算的操作如何转换为对应

的 mips 汇编代码，并且按照题目所给的表达式完成对数组内数据的计算，并将最终计算得到数组中每一位的数据的结果填写在实验结果所给的空格中。

2. 实验步骤

实验内容1:

该实验中，由题目要求，只能使用一个临时变量寄存器 t0，按照题目需求，我们至少需要储存两个数字，所以必须在程序的运行过程中通过使用堆栈内的空间中至少储存一个数字。

程序运行的流程如下：

1. 申请一个字的堆栈空间(由于堆栈仅有一个字节，以下所有的堆栈即为堆栈第 1 个位置)
2. 将 n1 储存到 t0 寄存器中
3. 将 t0 中的值储存到堆栈中
4. 将 n2 储存到 t0 寄存器中
5. 把 t0 寄存器中的值赋值到 a0 的地址中
6. 将堆栈中的值赋值到 t0 寄存器中
7. 把 t0 寄存器中的值赋值到 a1 的地址中

(具体的流程图在报告末尾的流程图中给出)

通过上述流程，即完成了把 a0 地址中值与 a1 地址中的值交换的目的。

完整程序代码如下：

```
.data
n1: .word 14
n2: .word 27
.text
main:
    la $a0,n1
    la $a1,n2
    jal swap
    li $v0,1
    lw $a0,n1
```

```
syscall
li $v0,11
li $a0,' '
syscall
li $v0,1
lw $a0,n2
syscall
li $v0,11
li $a0,'\n'
syscall
li $v0,10
syscall

swap:
move $fp,$sp
addiu $sp,$sp,-4

lw $t0 ,0($a0) # t0= n1
sw $t0 ,0($sp) # n1 入栈
lw $t0 ,0($a1) # t0 = n2
sw $t0 ,0($a0) # *a0 = n2
lw $t0 ,0($sp) # t0 = n1
sw $t0 ,0($a1) # *a1 = n2

addiu $sp,$sp,4
jr $ra
```

在运行 swap 中核心的六条代码的过程中，各个寄存器的值与储存空间中的值变化如下：

步 骤	\$t0	*sp	*a0 (n1)	*a1 (n2)
0	0	0	14	27
1	14	0	14	27
2	14	14	14	27
3	27	14	14	27
4	27	14	27	27
5	14	14	27	27
6	14	14	27	14

表 1-分步骤 swap 函数中内存与寄存器的值变化

实验内容2:

实现代码如下:

```
.data
K: .space 4
Y: .word 56
Z: .space 200

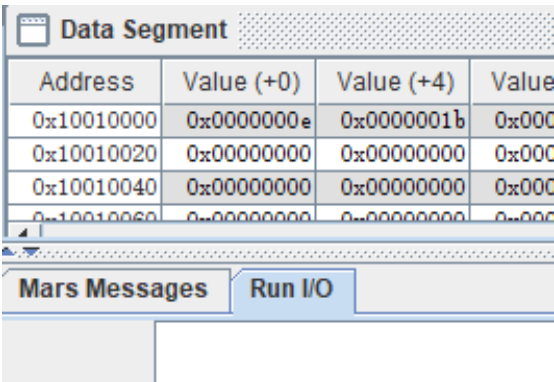
.text
main:
    la $t0,K
    lw $t0,0($t0)    #t0 = K
    la $t1,Z          #t1 为数组首地址
    la $t2,Y
    lw $t2,0($t2)    #t2 = Y
    addi $t3,$0,0     #t3 =0 用于循环变量
    addi $t5,$0,50    #t4 = 50 用于判断循环结束
loop:
    addi $t4,$t3,0     #t4=t3
    sra $t4,$t4,2      #t4=t4/4
    addi $t4,$t4,210   #t4=t4+210
    sll $t4,$t4,4      #t4=t4*16
    sub $t4,$t2,$t4    #t4=56-t2
    sw $t4,0($t1)      #计算结果储存回内存
    addi $t1,$t1,4     #数组指针 t1++
    addi $t3,$t3,1     #循环临时变量+1
    beq $t3,$t5,end   #判断是否跳出循环
    j loop
end:
    addi $v0,$0,10     #结束程序
    syscall
```

说明: 在该题目中, 首先需要完成对整型变量 K,Y,整形变量数组 Z 的声明, 在 mips 中通过 .space 来指定变量的长度, 通过这种方式可以声明指定长度的数组。通过 .word x 的方式声明一个整数变量并在该空间中储存一个 x 值。在 main 指令段中, 通过 la 指令将 K,Z,Y 存入寄存器中以供 cpu 调用。将用于循环的临时变量存入 t3 寄存器中。接下来使用算术右移和左移来实现原来 C 语言中的乘法和除法运算。将计算的结果储储存到对应的内存地址中去。

四、实验结果

实验内容1:

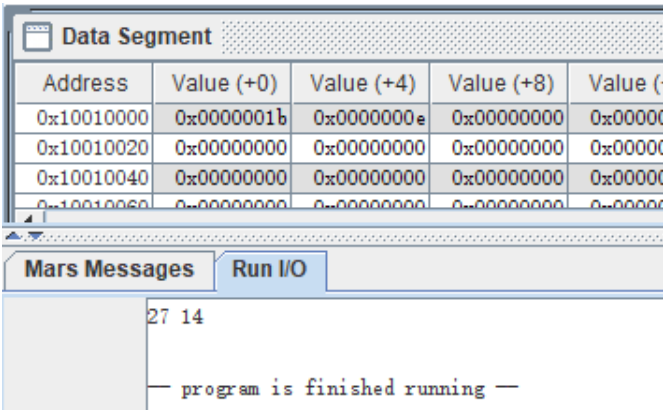
程序运行前:



Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)
0x10010000	0x0000000e	0x0000001b	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000

图 1-运行前 n1=14, n2=27

程序运行后:



Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)
0x10010000	0x0000001b	0x0000000e	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000

Mars Messages: 27 14
— program is finished running —

图 2-运行后 n1=14, n2=27

分析：经过运行程序之后，原版 a0 地址中值从 14 转换为了 27，a1 地址中的值从 27 转换为了 14，完成了交换，完成了实验要求。

实验内容2:

程序运行前:

Data Segment								
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000038	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Mars Messages

Run I/O

Assemble: assembling D:\programme\Computer-Organization-Experiment\ex6mips3\main.s
Assemble: operation completed successfully.

图 3-运行前内存中的值

程序运行后:

Data Segment								
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000038	0xfffff318	0xfffff318	0xfffff318	0xfffff318	0xfffff308	0xfffff308
0x10010020	0xfffff308	0xfffff308	0xfffff2f8	0xfffff2f8	0xfffff2f8	0xfffff2f8	0xfffff2e8	0xfffff2e8
0x10010040	0xfffff2e8	0xfffff2e8	0xfffff2d8	0xfffff2d8	0xfffff2d8	0xfffff2d8	0xfffff2c8	0xfffff2c8
0x10010060	0xfffff2c8	0xfffff2c8	0xfffff2b8	0xfffff2b8	0xfffff2b8	0xfffff2b8	0xfffff2a8	0xfffff2a8
0x10010080	0xfffff2a8	0xfffff2a8	0xfffff298	0xfffff298	0xfffff298	0xfffff298	0xfffff288	0xfffff288
0x100100a0	0xfffff288	0xfffff288	0xfffff278	0xfffff278	0xfffff278	0xfffff278	0xfffff268	0xfffff268
0x100100c0	0xfffff268	0xfffff268	0xfffff258	0xfffff258	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Mars Messages

Run I/O

— program is finished running —

图 4-运行后内存中的值

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
0x1001008	Z[0]	0xfffff318	0x100106c	Z[25]	0xfffff2b8
0x100100c	Z[1]	0xfffff318	0x1001070	Z[26]	0xfffff2b8
0x1001010	Z[2]	0xfffff318	0x1001074	Z[27]	0xfffff2b8
0x1001014	Z[3]	0xfffff318	0x1001078	Z[28]	0xfffff2a8
0x1001018	Z[4]	0xfffff308	0x100107c	Z[29]	0xfffff2a8
0x100101c	Z[5]	0xfffff308	0x1001080	Z[30]	0xfffff2a8
0x1001020	Z[6]	0xfffff308	0x1001084	Z[31]	0xfffff2a8
0x1001024	Z[7]	0xfffff308	0x1001088	Z[32]	0xfffff298
0x1001028	Z[8]	0xfffff2f8	0x100108c	Z[33]	0xfffff298
0x100102c	Z[9]	0xfffff2f8	0x1001090	Z[34]	0xfffff298
0x1001030	Z[10]	0xfffff2f8	0x1001094	Z[35]	0xfffff298
0x1001034	Z[11]	0xfffff2f8	0x1001098	Z[36]	0xfffff288
0x1001038	Z[12]	0xfffff2e8	0x100109c	Z[37]	0xfffff288
0x100103c	Z[13]	0xfffff2e8	0x10010a0	Z[38]	0xfffff288
0x1001040	Z[14]	0xfffff2e8	0x10010a4	Z[39]	0xfffff288
0x1001044	Z[15]	0xfffff2e8	0x10010a8	Z[40]	0xfffff278
0x1001048	Z[16]	0xfffff2d8	0x10010ac	Z[41]	0xfffff278
0x100104c	Z[17]	0xfffff2d8	0x10010b0	Z[42]	0xfffff278
0x1001050	Z[18]	0xfffff2d8	0x10010b4	Z[43]	0xfffff278
0x1001054	Z[19]	0xfffff2d8	0x10010b8	Z[44]	0xfffff268
0x1001058	Z[20]	0xfffff2c8	0x10010bc	Z[45]	0xfffff268
0x100105c	Z[21]	0xfffff2c8	0x10010c0	Z[46]	0xfffff268
0x1001060	Z[22]	0xfffff2c8	0x10010c4	Z[47]	0xfffff268
0x1001064	Z[23]	0xfffff2c8	0x10010c8	Z[48]	0xfffff258
0x1001068	Z[24]	0xfffff2b8	0x10010cc	Z[49]	0xfffff258

表 1-运行后内存 Z 中储存值

分析：0x10010004 地址储存 Y，0x10010008-0x100100cc 的地址储存数组 Z。程序运行后，经过与原本 C 语言程序计算的结果相比对结果无误，该汇编程序实现了对题目要求的 C 语言代码的转换。

综上，实验部分 1 与实验部分 2 均完成了实验任务

五、实验感想

本次实验是第三次使用 mips 汇编语言编写程序。本次实验内容 1 中通过使用堆栈来临时保存变量，实验内容 2 中，学习了如何把 C 语言中声明，赋值语句和计算语句逐条语句转换为对应的 mips 汇编代码。通过手动模拟编译器的行为，有了对编译器的初步理解。

除此之外，本人还尝试了使用编译器不同的优化等级来观察编译器所编译出来的 mips 汇编代码。

例如对于本次实验内容 2 中的部分：

```
void s(int a[]);
int main ()
{
    int k,Y;
    int Z[50];
    Y=56;
    for(k=0;k<50;k++) {
        Z[k]=Y-16*(k/4+210);
    }
    s(Z);
    return 0;
}
```

(其中调用一个 s 函数是为了防止编译器自动优化掉 main 中所运行的语句)

使用 mips gcc 5.4 编译器默认的编译选项下所编译出的 mips 汇编语句如下：(节选)

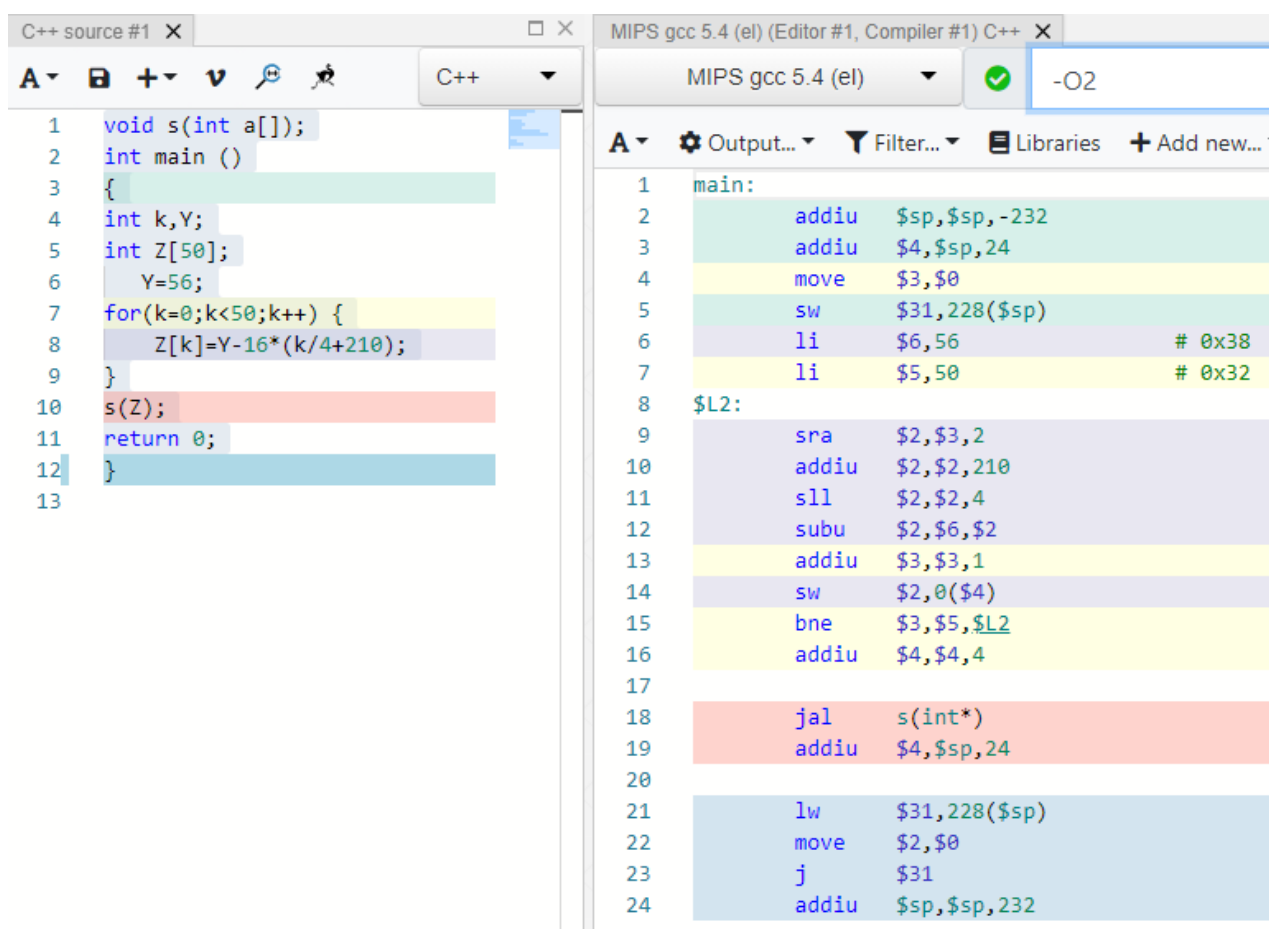

```
C++ source #1 X
1 void s(int a[]);
2 int main ()
3 {
4   int k,Y;
5   int Z[50];
6   Y=56;
7   for(k=0;k<50;k++) {
8     Z[k]=Y-16*(k/4+210);
9   }
10  s(Z);
11  return 0;
12 }
13

MIPS gcc 5.4 (el) (Editor #1, Compiler #1) C++ X
MIPS gcc 5.4 (el)
Compiler options...
Output... Filter... Libraries + Add new...
1 main:
2   addiu   $sp,$sp,-240
3   sw      $31,236($sp)
4   sw      $fp,232($sp)
5   move    $fp,$sp
6   li      $2,56          # 0x38
7   sw      $2,28($fp)
8   sw      $0,24($fp)
9 $L4:
10  lw      $2,24($fp)
11  nop
12  slt      $2,$2,50
13  beq      $2,$0,$L2
14  nop
15
16  lw      $2,24($fp)
17  nop
18  bgez     $2,$L3
19  nop
20
21  addiu    $2,$2,3
22 $L3:
23  sra      $2,$2,2
24  addiu    $2,$2,210
25  sll      $2,$2,4
26  lw      $3,28($fp)
27  nop
28  subu     $3,$3,$2
29  lw      $2,24($fp)
30  nop
31  sll      $2,$2,2
32  addiu    $4,$fp,24
33  addu     $2,$4,$2
34  sw      $3,8($2)
35  lw      $2,24($fp)
36  nop
37  addiu    $2,$2,1
38  sw      $2,24($fp)
```

图 5-默认编译选项生成的 mips 汇编代码(节选)

可见在默认编译选项的情况下，编译器会严格按照语句执行，例如其中的循环变量 K，每一次都从内存中读取到寄存器中，使用完毕后又将该变量存回内存中，十分繁琐。对于其中算式的计算，也是将每一步的结果都计算出来，存入堆栈中以便进行下一步计算，而没有通过变换顺序进行优化。其中的 lw 指令也强制使用了一个 nop 来等待内存准备数据，而没有合理安排 cpu 进行其他的工作。

但是当编译器打开-O2 优化选项时，发生很大的改变，生成的汇编代码如下：(完整)



```
C++ source #1 X
1 void s(int a[]);
2 int main ()
3 {
4 int k,Y;
5 int Z[50];
6 Y=56;
7 for(k=0;k<50;k++) {
8 Z[k]=Y-16*(k/4+210);
9 }
10 s(Z);
11 return 0;
12 }
13

MIPS gcc 5.4 (el) (Editor #1, Compiler #1) C++ X
MIPS gcc 5.4 (el) -O2
Output... Filter... Libraries + Add new...
1 main:
2 addiu $sp,$sp,-232
3 addiu $4,$sp,24
4 move $3,$0
5 sw $31,228($sp)
6 li $6,56 # 0x38
7 li $5,50 # 0x32
8 $L2:
9 sra $2,$3,2
10 addiu $2,$2,210
11 sll $2,$2,4
12 subu $2,$6,$2
13 addiu $3,$3,1
14 sw $2,0($4)
15 bne $3,$5,$L2
16 addiu $4,$4,4
17
18 jal s(int*)
19 addiu $4,$sp,24
20
21 lw $31,228($sp)
22 move $2,$0
23 j $31
24 addiu $sp,$sp,232
```

图 6-开启-O2 编译选项生成的 mips 汇编代码

这次，编译器直接将循环的临时变量再寄存器中进行操作，也没有使用后进行写回储存的操作，在 lw 指令之后也穿插了其他指令防止 cpu 闲置。这份汇编代码与我写的比较相近了，甚至在循环部分比我写的还要少一行。也十分的浅显易懂，可见编译器优化功能的强大实力。

附录（流程图，注释过的代码）：

实验内容1流程图

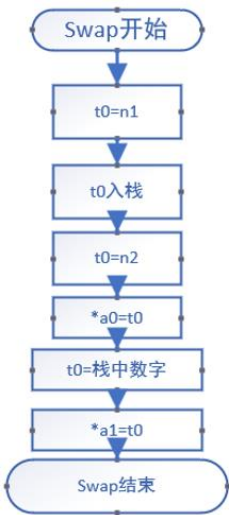


图 7-实验内容 1 中 swap 部分流程图

实验内容2流程图

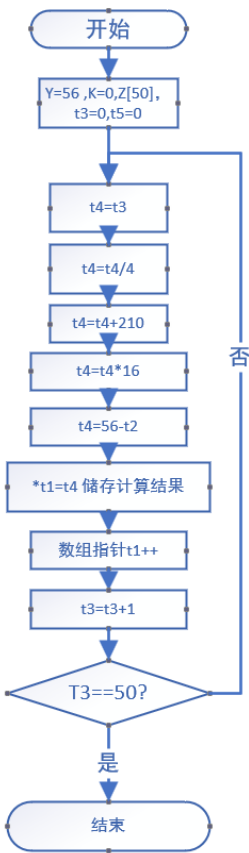


图 8-实验内容 2 中的程序流程图

本次实验涉及到的代码均在上文实验步骤内出现，此处不再重复。