中山大学数据科学与计算机学院本科生实验报告

（2017 学年春季学期）

课程名称：**计算机组成原理实验**　　任课教师：郭雪梅　　助教：李声涛、王绍菊

年级&班级	1518	专业(方向)	软件工程(移动信息工程)
学号	15352408	姓名	张镓伟
电话	13531810182	Email	709075442@qq.com
开始日期	4.7	完成日期	4.14

一、实验题目

1.MIPS 程序编写练习

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
.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
.data
n1:  .word    14
n2:  .word    27
```

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)。

2. 编写两个版本的 first1pos (starting from first1pos.s) 函数, 在$a0 中给定一个数, 而在$v0 中返回$a0 字中最左边的非零位的位置. 如果$a0 的值是 0, 在$v0 中存 -1 . 在查找此位置的过程中,允许你修改$a0 值.位置从 0 (最右位)到 31 (符号位). 其中一种解应该重复移位$a0, 每次移位后,检查符号位. 另一种方法是初始时使用 0x80000000 作为掩码,并不断右移该掩码来检查$a0 的每一位.

```
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:     # your code goes here

    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
```

二、实验目的

1. 熟悉并掌握一些简单的汇编指令
2. 练习函数调用，知道哪些东西要入栈，熟练各种逻辑操作
2. 提高将课堂所学知识进行实际应用的能力。

三、实验内容

1. 实验原理

1. 函数参数在在函数调用前会被放进栈中。如果这有一个函数，那么一个空的占位符(4 bytes)会在函数调用前被放进栈中，用来保存函数的返回值。
   每个函数应该有如下开头：

```
sw        $fp, ($sp) # push old frame pointer (dynamic link)
move      $fp, $sp     #frame pointer now points to the top of stack
subu      $sp, $sp, 500      # allocate say 500 bytes in the stack
                                    #      (for frame size = 500)
sw        $ra, -4($fp)       # save return address in frame
sw        $v0, -8($fp)       # save static link in frame
```

最简单情况下，叶子子程序不需要保存返回地址，需要入栈保护$t0，$t1 的情况：

```
subu    $sp, $sp, 8      # allocate say 8 bytes in the stack
sw      $t0, 0($sp)              # push x
sw      $t1,4 ($sp)              # push c
lw      $t0, 0($sp)              # pop x
lw          $t1,4 ($sp)              # push c
addu        $sp, $sp, 8            # pop stack
```

2. 实验步骤

1.swap：

1.首先考虑要先分配多少 bytes 给栈。这题中函数没有返回结果，只有两个参数和一个局部变量 temp，所以我们只要分配 12bytes 给栈。
2.我规定 4($sp)存储变量 n1 的地址，8$(sp)存储变量 s2 的地址，12$(sp)存储变量 temp 的值。
3.流程：分配 12bytes 给栈->将 n1,n2 的地址分别存到对应的栈的位置->取出 n1 的值->temp=*n1 并存到栈中相应位置->取出 n2 的值 2 取出 n1 的地址->将 n2 的值写入 n1 的地址中->取出 temp 的值和 n2 的地址->将 temp 的值写入 n2 的地址->结束

代码见附件

2. first1pos：

下面先说明第一个版本的做法(即重复移位$a0, 每次移位后,检查符号位)：

1. 首先思考如何设计算法。要找出$a0 从左起第一个不为 0 的位置，我们可以写成循环，初始化答案为 31，从最高位开始，每次左移一位，答案就减一，这样我们每次只要检测最高位。把每次左移之后的数和 0x8000 相与，若结果不为 0 说明找到了，退出循环。若答案减到了-1，说明$a0=0，退出循环。
2. 算法设计好了我们可以思考应该先分配多少内存给栈。这里我们要保存的东西有函数的返回值(最终存储在$v0)、掩码、$a0、循环终止指标常数-1，调用函数前的地址$ra 共 5 个，所以要分配 20 bytes。
3. 规定$ra 的值存储在 20($sp),原数存储在 16($sp)，终止条件存储在 12($sp)，掩码存储在 8($sp)，函数返回值($v0 的值)存储在 4($sp)。
4. 答案先初始化 31，每次循环，先分别 load 原数和掩码，相与，若结果为 0 则退出循环。load 答案的值，答案值减一，save 答案值在栈中相应位置。load 终止条件，若答案的值=终止条件则退出循环。load 原数，将原数左移一位，save 左移后的原数到栈中相应位置。调到循环开头继续循环。
2. 退出循环时要 load 答案 和 函数调用前的地址。

下面先说明第二个版本的做法(即初始时使用 0x80000000 作为掩码,并不断右移该掩码来检查$a0 的每一位)：

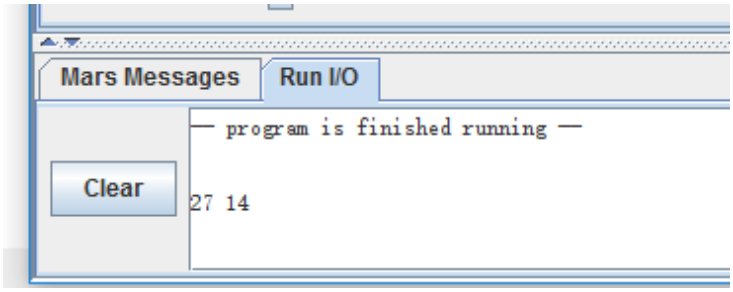
这个版本的做法其实与第一个版本类似，只需要将第一个版本的第 4 步中的“load 原数，将原数左移一位，save 左移后的原数到栈中相应位置”改为“load 掩码，将掩码右移一位，save 右移后的掩码到栈中相应位置”即可。

两个版本的代码详见附件

四、实验结果

1.swap：

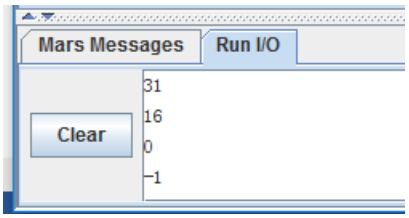
将 n1=14，将 n2=27 互换后输出 n1，n2



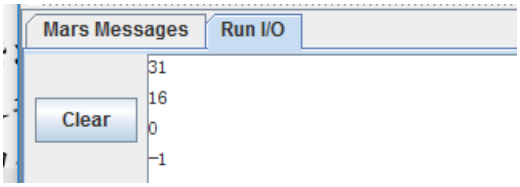
2. first1pos:

分别对$a0=0x80000000, 0x00010000, 1 和 0 四种情况执行一次函数

第一个版本：



第二个版本：



五、实验感想

通过这次实验，我对 MIPS 汇编指令有了更深刻的理解。这次主要学习了函数调用过程中的栈分配。栈用来传递函数参数，存储返回值信息，保存寄存器以供恢复调用前处理机状态。每次调用一个函数，都要为该次调用的函数实例分配栈空间。为单个函数分配的那部分栈空间就叫做 stack frame，也就是说，stack frame 这个说法主要是为了描述函数调用关系的。

上网查阅资料得知 Stack frame 组织方式的重要性和作用体现在两个方面：

第一，它使调用者和被调用者达成某种约定。这个约定定义了函数调用时函数参数的传递方式，函数返回值的返回方式，寄存器如何在调用者和被调用者之间进行共享；

第二，它定义了被调用者如何使用它自己的 stack frame 来完成局部变量的存储和使用。

实话说，第一次看到这次的练习题目的时候我是有些不知道如何下手的，因为之前看书的时候也不太理解这一部分。直到我上网查阅到上述资料后，并与同学相互讨论才初步掌握了怎么去用$fp，$sp 给函数的相关信息分配空间等。总的来说这次学到了不少知识，对之前在理论课上很多不太了解的只是都有了更好的了解。

附录：

1.swap 函数部分：

```
swap:   move    $fp, $sp    #FRAME  POINTER NOW POINTS TO THE TOP OF STACK
    addiu $sp,$sp,-12 # ALLOCATE 12 BYTES IN THE STACK
    sw $a0,4($sp)     #将n1的地址存储在4($sp)
    sw $a1,8($sp)     #将n2的地址存储在8($sp)
    lw $a0,0($a0)     #$a0= n1的值
    sw $a0,12($sp)    #temp=n1
    lw $a0,0($a1)     #$a0=n2的值
    lw $a1,4($sp)     #$a1= n1的地址
    sw $a0,0($a1)     # *n1=*n2
    lw $a0,12($sp)     #$a0=temp
    lw $a1,8($sp)     #$a1= n2的地址
    sw $a0,0($a1)        # *n2=temp
    addiu $sp,$sp,12
        jr $31
```

2. first1pos 函数部分：

第一个版本：

```
17  firstlpos:  # your code goes here
18          move    $fp,  $sp
19          addiu   $sp,$sp,-20
20          sw      $a0,16($sp)
21          sw      $ra,20($sp)
22
23          addiu   $a0,$0,31   #初始化v0=31
24          sw      $a0,4($sp)
25
26          addiu   $a0,$0,0x80000000 #掩码
27          sw      $a0,8($sp)
28
29          addiu   $a0,$0,-1    #终止条件
30          sw      $a0,12($sp)
31
32          loop:
33              lw $a0,16($sp)   #load 原数
34              lw $a1,8($sp)    #load 掩码
35              and $a1,$a0,$a1 #原数 与上 掩码
36              bne $a1,$0,return    #and了之后不为0说明当前位为1
37
38              lw $a1,4($sp)       #load v0
39              addiu $a1,$a1,-1     #v0-=1
40              sw $a1,4($sp)       # save v0
```

```
41
42              lw $a0,12($sp)      #load 终止条件
43              beq $a1,$a0,return  #v0到-1就退出循环,说明$a0=0
44
45              lw $a0,16($sp)      #load 原数
46              sll $a0,$a0,1       #将原数左移一位
47              sw $a0,16($sp)      # save 左移一位后的原数
48              beq $a0,$a0,loop    # 无条件继续循环，因为终止循环不在这里判断
49          return:
50              lw  $v0,4($sp)      #load v0
51              lw  $ra,20($sp)     #load 调用函数前的地址
52              addi $sp,$sp,20
53              jr  $ra
```

第二个版本：

```
17  firstlpos:  # your code goes here
18          move    $fp, $sp
19          addiu   $sp,$sp,-20
20          sw      $a0,16($sp)
21          sw      $ra,20($sp)
22
23          addiu   $a0,$0,31   #初始化v0=31
24          sw      $a0,4($sp)
25
26          addiu   $a0,$0,0x80000000 #掩码
27          sw      $a0,8($sp)
28
29          addiu   $a0,$0,-1   #终止条件
30          sw      $a0,12($sp)
31
32          loop:
33              lw $a0,16($sp)  #load 原数
34              lw $a1,8($sp)   #load 掩码
35              and $a1,$a0,$a1 #原数 与上 掩码
36              bne $a1,$0,return   #and了之后不为0说明当前位为1
37
38              lw $a1,4($sp)       #load v0
39              addiu $a1,$a1,-1    #v0-=1
40              sw $a1,4($sp)       # save v0
```

```
41
42              lw $a0,12($sp)      #load 终止条件
43              beq $a1,$a0,return  #v0到-1就退出循环,说明$a0=0
44
45              lw $a0,8($sp)       #load 掩码
46              srl $a0,$a0,1       #将掩码右移一位
47              sw $a0,8($sp)       # save 右移一位后的掩码
48              beq $a0,$a0,loop    # 无条件继续循环，因为终止循环不在这里判断
49          return:
50              lw  $v0,4($sp)       #load v0
51              lw  $ra,20($sp)     #load 调用函数前的地址
52              addi $sp,$sp,20
53              jr  $ra
```