计算机结构与组成　　实验课 2

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.

阅读

书的第 2.5-2.7 节.

基本准备

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;
}
```

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.

```
1  .text
2  main:
3          la      $a0, n1
4          la      $a1, n2
5          jal     swap
6          li      $v0, 1    # print n1 and n2; should be 27 and 14
7          lw      $a0, n1
8          syscall
9          li      $v0, 11
10         li      $a0, ' '
11         syscall
12         li      $v0, 1
13         lw      $a0, n2
14         syscall
15         li      $v0, 11
16         li      $a0, '\n'
17         syscall
18         li      $v0, 10   # exit
19         syscall
20
```

```
21  swap:
22          ori $sp, $sp, 0x2ffc        #栈初始化
23          addiu $sp, $sp, -16         #在栈中压入四个字的空间
24          lw $24, ($4)                #将$a0数据(n1)放到$t8中
25          sw $24, 12($sp)             #将$t8数据存入栈中
26          lw $24, 0($5)               #将$a1数据(n2)放到$t8中
27          sw $24, 0($4)               #将$t8数据(n2)存到$a0中
28          lw $24, 12($sp)             #将栈中的数据(n1)放到$t8中
29          sw $24, ($5)                #将$t8数据(n1)存到$a1中
30  L_1:    addiu $sp, $sp, 16          #恢复栈指针
31          jr $31                      #函数返回
32
33          .data
34  n1:     .word   14
35  n2:     .word   27
36
```

```
27 14

— program is finished running —
```

练习 2

在代码 nchoosek.s 中加入前言(prologue)和后语(epilogue) ,使其能计算"n 选 k". 从 n 个不同的元素中一次取 k 个数的不同取法(组合数). (这也是杨辉三角形的第(n,k)个数.) 将运行的过程和结果给老师看.

```
 1  main:
 2          li      $a0, 4
 3          li      $a1, 0
 4          jal     nchoosek                # evaluate C(4,0); should be 1
 5          jal     printv0
 6          li      $a0, 4
 7          li      $a1, 1
 8          jal     nchoosek                # evaluate C(4,1); should be 4
 9          jal     printv0
10          li      $a0, 4
11          li      $a1, 2
12          jal     nchoosek                # evaluate C(4,2); should be 6
13          jal     printv0
14          li      $a0, 4
15          li      $a1, 3
16          jal     nchoosek                # evaluate C(4,3); should be 4
17          jal     printv0
18          li      $a0, 4
19          li      $a1, 4
20          jal     nchoosek                # evaluate C(4,4); should be 1
21          jal     printv0
```

```
22          li      $a0, 4
23          li      $a1, 5
24          jal     nchoosek                # evaluate C(4,5); should be 0
25          jal     printv0
26          li      $v0, 10
27          syscall
28
29
30  nchoosek:       # you fill in the prologue
31          addi    $sp, $sp, -16           # 在栈中压入四个字的空间
32          sw      $ra, 12($sp)            # 在栈中保存返回地址
33          sw      $s2, 8($sp)             # 保存$s2，用于保存C(n-1,k)
34          sw      $s1, 4($sp)             # 保存$s1，用于保存k
35          sw      $s0, 0($sp)             # 保存$s1，用于保存n-1
36          beq     $a1, $0, return1        # 如果k=0，跳转到return1
37          beq     $a0, $a1, return1       # 如果n=k，跳转到return1
38          beq     $a0, $0, return0        # 如果n=0，跳转到return0
39          blt     $a0, $a1, return0       # 如果n<k，跳转到return0
40
```

```
41          addi    $a0, $a0, -1            # $a0中的数据改为n - 1
42          move    $s0, $a0                # 将$a0中的数据(n-1)存入$s0
43          move    $s1, $a1                # 将$a1中的数据(k)存入$s1
44          jal     nchoosek                # $v0 = C(n-1,k)
45
46          move    $s2, $v0                # 将$v0中的数据(C(n-1,k))存入$s2
47          move    $a0, $s0                # 将$s0中的数据(n-1)存入$a0
48          addi    $a1, $s1, -1            # 将$s1中的数据(k)减1存入$a1
49          jal     nchoosek                # $v0 = C(n-1,k-1)
50
51          add     $v0, $v0, $s2           # C(n,k) = C(n-1,k) + C(n-1,k-1)
52          j       return                  # 跳转到return
53  return0:
54          move    $v0, $0                 # $v0 = 0
55          j       return                  # 跳转到return
56  return1:
57          addi    $v0, $0, 1              # $v0 = 1
58          j       return                  # 跳转到return
59
```

```
60  return:         # you fill in the epilog
61          lw      $s0, 0($sp)             # 恢复旧的$s0中数据
62          lw      $s1, 4($sp)             # 恢复旧的$s1中数据
63          lw      $s2, 8($sp)             # 恢复旧的$s2中数据
64          lw      $ra, 12($sp)            # 恢复旧的返回地址
65          addi    $sp, $sp, 16            # 恢复栈指针
66          jr      $ra                     # 函数返回
67
68  printv0:
69          addi    $sp, $sp, -4
70          sw      $ra, 0($sp)
71          move    $a0, $v0
72          li      $v0, 1
73          syscall
74          li      $a0, '\n'
75          li      $v0, 11
76          syscall
77          lw      $ra, 0($sp)
78          addi    $sp, $sp, 4
79          jr      $ra
```

```
1
4
6
4
1
0

-- program is finished running --
```

练习 3

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.

可以和你的同伴分别来做此工作, 其中一个同学做第一个版本,另一个同学做第二个版本. 工作完成后,向你的同伴解释你的代码, 然后把运行情况给老师检查.

第一种方法（左移$a0）

```
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
        beq     $a0, $0, false          # 如果$a0=0，跳转到false
        addi    $s0, $0, 31             # $s0赋值31
        addi    $s1, $0, 0x80000000     # $s1赋值0x80000000
loop:
        and     $t0, $a0, $s1           # 将$a0和$s1按位与的结果放到$t0
        bne     $t0, $0, then           # 如果$t0!=0，即此时$a0最高位不为0，跳转到then
        sll     $a0, $a0, 1             # 如果$t0=0，即此时$a0最高位依然是0，将$a0左移一位
        addi    $s0, $s0, -1            # $s0 = $s0 - 1 计算检测$a0的哪一位
        j       loop                    # 跳转到loop，继续查找
then:
        move    $v0, $s0                # $v0 = $s0，结果为$a0字中最左边的非零位的位置
        jr      $ra                     # 函数返回
false:
        addi    $v0, $0, -1             # $v0 = -1，未找到非零数
        jr      $ra                     # 函数返回

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
```

```
31
16
0
-1

-- program is finished running --
```

第二种方法（以 0x80000000 作掩码并不断右移）

```
1  main:
2          lui     $a0, 0x8000
3          jal     first1pos
4          jal     printv0
5          lui     $a0, 0x0001
6          jal     first1pos
7          jal     printv0
8          li      $a0, 1
9          jal     first1pos
10         jal     printv0
11         add     $a0, $0, $0
12         jal     first1pos
13         jal     printv0
14         li      $v0, 10
15         syscall
16
```

```
17 first1pos:     # your code goes here
18         beq    $a0, $0, false          # 如果$a0=0，跳转到false
19         addi   $s0, $0, 31             # $s0赋值31
20         addi   $s1, $0, 0x80000000     # $s1赋值0x80000000（掩码）
21 loop:
22         and    $t0, $a0, $s1           # 将$a0和$s1按位与的结果放到$t0
23         bne    $t0, $0, then           # 如果$t0!=0，即此时$a0检测的该位不为0，跳转到then
24         srl    $s1, $s1, 1             # 如果$t0=0，即此时$a0检测的该位依然是0，将$s1（掩码）右移一位
25         addi   $s0, $s0, -1            # $s0 = $s0 - 1 计算检测$a0的哪一位
26         j      loop                    # 跳转到loop，继续查找
27 then:
28         move   $v0, $s0                # $v0 = $s0，结果为$a0字中最左边的非零位的位置
29         jr     $ra                     # 函数返回
30 false:
31         addi   $v0, $0, -1             # $v0 = -1，未找到非零数
32         jr     $ra                     # 函数返回
33
```

```
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
```

```
31
16
0
-1

— program is finished running —
```