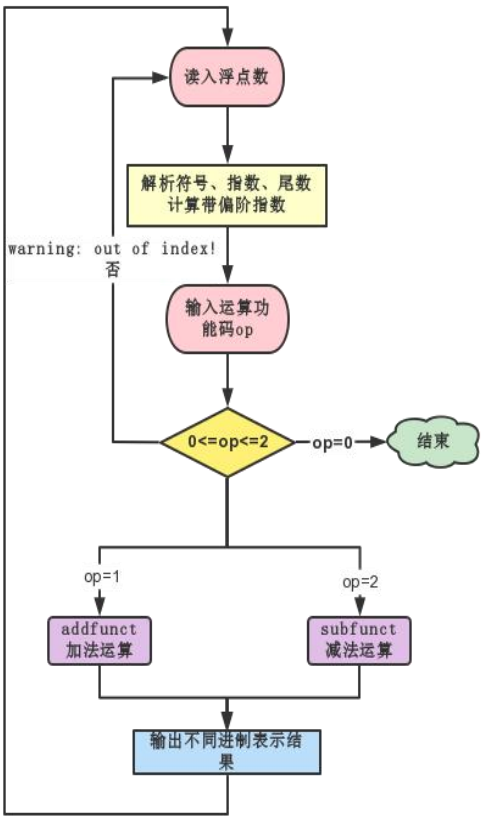
MIPS SOC 设计报告

一、设计简介

本设计实现了在**没有浮点表示**和计算硬件的条件下，用软件方法**实现 IEEE 754 单精度浮点数的表示**及浮点数**加减法运算**的功能（即只利用整数运算指令来编写，不使用浮点指令），并且提供人机交互方式供用户选择相应的运算功能。本设计接受十进制实数形式的输入，在内存中以 IEEE 754 单精度方式表示，支持以二进制和十六进制的方式显示输出。

二、设计方案（30%）

（一）总体设计思路



项目程序如流程图所示，首先通过系统调用接收键盘输入的两个浮点数并解析浮点数的符号、指数、尾数并计算带偏阶的指数；其次，接收输入的运算码（0 代表退出，1 代表加法，2 代表减法）；然后根据运算功能码跳转到对应的运算功能函数进行运算，最后输出不同进制表示的结果。

其中输入输出函数、表达式解析、加减运算的设计思路将在后续模块展开介绍，此处不加赘述。主函数汇编指令如下：

```
.text           #代码段
    main:   #开始执行
            #将num1、num2的首地址从内存写入寄存器
            la      $s5,    num1                    #将num1的首地址保存到$s5寄存器
            la      $s6,    num2                    #将num2的首地址保存到$s6寄存器
            #跳转到输入函数，接收浮点数num1、num2并解析其符号、指数、尾数和带偏阶指数
            jal Input_funct                         #jal先将当前PC放入$ra再跳转
            #输入计算功能（0退出、1加法、2减法）
            la      $a0,    Tips3
            li      $v0,    4
            syscall                                 #打印"Please choose one function: 0 for exit, 1 for add, 2 for sub: \0"
            li      $v0,    5
            syscall                                 #调用系统$v0=5读取输入的整数值并存入$v0
            #此时$v0存储计算功能码，分别比较0、1、2用以跳转至相应函数，若不在该区间则出现异常
            li      $t0,    1
            beq     $v0,    $t0,    add_funct               #加法
            li      $t0,    2
            beq     $v0,    $t0,    sub_funct               #减法
            li      $t0,    0
            beq     $v0,    $t0,    exit_funct      #退出
            bne     $v0,    $t0,    default_funct   #输入的不是0-4
```

异常处理方式：

1. 若输入浮点数数据类型不是浮点数，或运算码类型不是整数，系统异常可以自动跳出并结束程序（见三.(二).错误 1）；

2. 若运算码类型不在 0~2 范围内，会进入 default 函数，给出提示"Sorry, your function number is out of index! Please input right function number between 0 and 2. "并重新接收输入；

3. 若计算结果发生溢出，跳转至相应模块并输出警告，详见（四）.2 输出程序设计

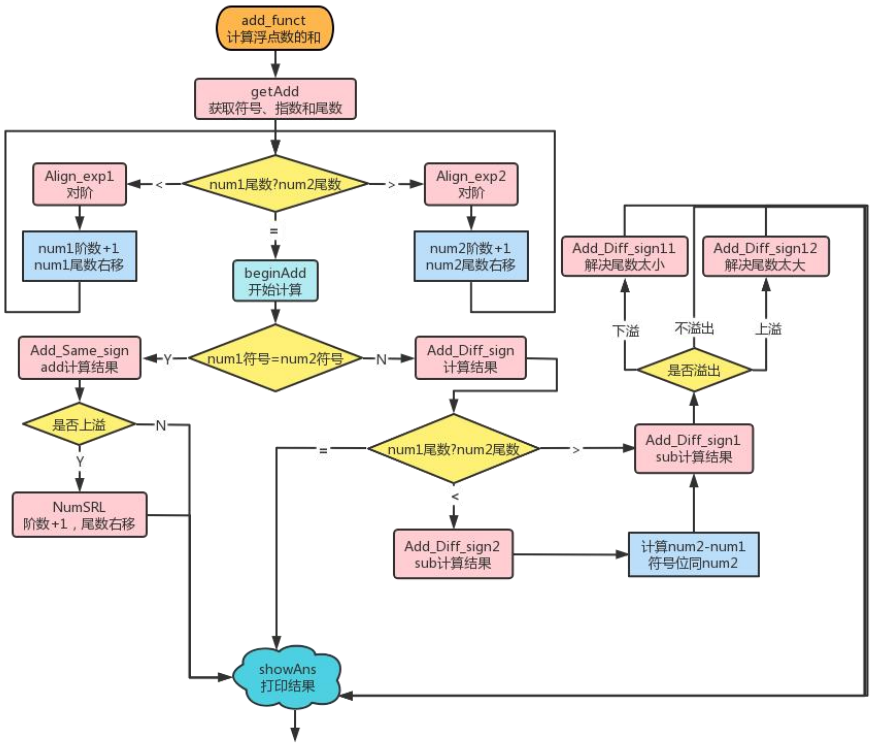
（二）加法模块设计

浮点数加法运算前需要先**对阶**，保证两个浮点数阶数（即指数）相同才可以运算。若两个加数阶数不同，则需要将阶数小的向阶数大的对齐，因为**将阶数大的向阶数小的对齐会降低阶数大的浮点数的精度**。对阶完成后则可以用整数运算指令运算。为了判定运算，需要先判断 num1、num2 **符号是否相同**：若符号相同则直接相加，结果需要**判断上溢**后才可以输出；若符号不同，则用**尾数大的减尾数小的**，运算结果的符号与尾数大的相同，需要**判断上溢和下溢**。加法主模块汇编指令如下：

```
#加法流程：取num1、num2的符号位、阶、尾数->补全尾数的整数位->对阶->执行加法运算->输出
add_funct:
        jal     getAdd
        jal     binary
        jal     hex
        j       main                            #本次执行完毕，跳回主函数开头
```

```
getAdd:
        #取num1和num2的符号位
        lw      $s0,    4($s5)          #$s0是num1的符号位，$s1是num2的符号位
        lw      $s1,    4($s6)
        #取num1和num2的阶
        lw      $s2,    8($s5)          #$s2是num1的阶，$s3是num2的阶
        lw      $s3,    8($s6)
        #取num1和num2的尾数
        lw      $s4,    12($s5)         #$s4是num1的尾数，$s5是num2的尾数
        lw      $s5,    12($s6)
        #补全尾数的整数位1
        ori     $s4,    $s4,    0x00800000      #将整数位1补全
        ori     $s5,    $s5,    0x00800000
        #对阶
        sub     $t0,    $s2,    $s3             #比较num1和num2的阶数（指数）大小
        bltz    $t0,    Align_exp1              #$t0小于0，则表明num1的阶小于num2，需将num1右移对阶
        bgtz    $t0,    Align_exp2              #$t0大于0，需将num2右移对阶
        beqz    $t0,    beginAdd                #两个数阶相同，则直接相加
```

上述流程可用如下流程图表示：



接下来依次介绍**对阶**模块、**符号判断**模块、**同号相加与上溢判断**模块以及**异号相加与上、下溢判断**模块。

1. 对阶模块

若两个加数阶数不同，进入相加前需要先对阶。将阶数小的向阶数大的对齐，过程采用递归，阶数小的右移 1 位后判断与另一个浮点数阶数大小来决定回到 Align_exp 函数开头或开始相加。对应汇编指令如下：

```
#对阶：若两个加数阶数不同，进入相加前需要先对阶，将阶数小的向阶数大的对齐，因为反之会降低阶数大的数的精度
#对阶过程采用递归，阶数小的右移1位再判断回到Align_exp函数开头或开始相加
Align_exp1:     #num1的阶小于num2的阶，num1阶数+1，尾数右移
        addi    $s2,    $s2,    1               #num1阶数+1
        srl     $s4,    $s4,    1               #num1尾数右移
        sub     $t0,    $s2,    $s3             #循环判断
        bltz    $t0,    Align_exp1              #branch if less than zero
        beqz    $t0,    beginAdd                #branch if equal zero跳到相加
Align_exp2:     #num1的阶大于num2的阶，num2阶数+1，尾数右移
        addi    $s3,    $s3,    1
        srl     $s5,    $s5,    1
        sub     $t0,    $s2,    $s3
        bgtz    $t0,    Align_exp2
        beqz    $t0,    beginAdd
#此时num1、num2阶数相同，判断符号后才能相加
```

2. 符号判断模块

将 num1、num2 的符号按位异或，若结果为 0 则同号，否则异号。跳转至各自函数进行运算即可。汇编指令如下：

```
beginAdd:
        xor     $t1,    $s0,    $s1             #按位异或判断num1、num2符号是否相同（相同则$t1存32'b0，不同存32'b1）
        beq     $t1,    $zero,  Add_Same_sign   #num1、num2符号相同，则直接加(Add_Same_sign)
        j       Add_Diff_sign                   #num1、num2符号不同，跳转到(Add_Diff_sign)
```

3. 同号相加与上溢判断模块

同号相加可以直接使用整数运算指令 add 进行尾数相加，但得到的结果不能直接输出。因为相加的结果有可能上溢出，需要调整阶数。对于两个无符号 23bit 二进制数（num1、num2 的尾数）相加，如果结果的第 24 位为 1，则发生了上溢，需要右移尾数、阶数+1。汇编指令代码如下：

```
#num1、num2符号相同相加
Add_Same_sign:
        add     $t2,    $s4,    $s5             #尾数相加后的结果即为输出的尾数，但需要先判断是否溢出
        sge     $t3,    $t2,    0x01000000      #set if greater or equal 判断上溢
        #因为两个无符号23bit二进制数相加，如果结果的第24位为1，则发生了上溢，需要右移尾数、阶数+1
        bgtz    $t3,    NumSRL                  #上溢则尾数右移
        j       showAns                         #无溢出就跳转到结果输出部分
#num1、num2符号相同相加后上溢出，需要尾数右移，阶数+1
NumSRL:
        srl     $t2,    $t2,    1               #尾数右移
        addi    $s2,    $s2,    1               #阶数+1
        j       showAns                         #此时阶数、尾数正确，可以输出
```

4. 异号相加与上、下溢判断模块

异号相加需要先判断 num1、num2 尾数大小，用尾数大的减尾数小的，运算结果的符号与尾数大的相同，使用整数运算指令 sub。此处若 num1、num2 尾数大小相同可以直接输出结果：

```
#num1、num2符号不同相加
Add_Diff_sign:
        sub     $t2,    $s4,    $s5             #符号不同的数相加相当于先相减再加符号，但可能出现尾数过大（上溢）或过小（下溢）的情况
        bgtz    $t2,    Add_Diff_sign1          #如果num1的尾数比num2大，则跳转至Add_Diff_sign1（结果与num1同号）
        bltz    $t2,    Add_Diff_sign2          #如果num1的尾数比num2小，则跳转至Add_Diff_sign2（结果与num2同号）
        j       show0                           #如果它们的绝对值相等，则结果为0，可以跳转到特殊结果输出
```

尾数大的减尾数小的后可能出现尾数过大（上溢）或过小（下溢）的情况，需要分别处理。若结果第 24 位为 1，则发生上溢；若结果没发生上溢且第 23 位为 0，则发生下溢。处理完溢出情况即可输出：

```
#num1、num2符号不同相加，num1尾数比num2大，输出前需要先判断是否上溢或下溢
Add_Diff_sign1:
        blt     $t2,    0x00800000,     Add_Diff_sign11         #尾数太小，则需左移，将其规格化
        bge     $t2,    0x01000000,     Add_Diff_sign12         #如果尾数没有过小，那么就需要判断上溢
        j       showAns                                         #既不上溢也不下溢的结果过可以直接输出
#num1、num2符号不同相加，num1尾数比num2大，结果尾数太小
Add_Diff_sign11:
        sll     $t2,    $t2,    1                               #左移扩大尾数
        subi    $s2,    $s2,    1                               #阶数-1
        blt     $t2,    0x00800000,     Add_Diff_sign11         #循环扩大尾数
        j       showAns
#num1、num2符号不同相加，num1尾数比num2大，结果尾数太大
Add_Diff_sign12:
        srl     $t2,    $t2,    1                               #左移缩小尾数
        addi    $s2,    $s2,    1                               #阶数+1
        bge     $t2,    0x01000000,     Add_Diff_sign12
        j       showAns
```

对于 num1 尾数小于 num2 的情况，存在一个**模块复用**，跳转即可：

```
#num1、num2符号不同相加，num1尾数比num2小
Add_Diff_sign2:
        sub     $t2,    $s5,    $s4                     #将$t2中数化为正
        xori    $s0     $s0     0x00000001              #结果与num2同号
        j       Add_Diff_sign1                          #模块复用
```

此处注意需要将 num2 的符号位存入原本存放 num1 符号位的$s0，结合 num1>num2 的 num1 符号存在$s0 中，相当于$s0 中存放了结果的符号位，便于后续输出。

（三）减法模块设计

减法模块不必重新设计，因为 num1-num2 等于 num1+(-num2)，所以只需要将 num2 符号取反后复用 add_funct 模块即可。

```
#减法可以复用加法模块（将num2符号取反即可）
sub_funct:
        lw      $t1,    4($s6)                          #num2的符号位存入$t1
        xori    $t1,    $t1,    1                       #将num2符号位按位异或（取反）
        sw      $t1,    4($s6)
        jal     getAdd
        jal     binary
        jal     hex
        j main
```

（四）输入输出程序设计

利用伪指令 la、li 配合系统调用 syscall 实现输入和输出：

la $a0 XXX：将待输出的数值或字符串传入$a0

li $v0 XXX：输入或输出，1 表示输出整数，4 表示输出字符串，5 表示接收整数

syscall：系统调用

示例如下：

```
.data           #数据段
        funcint:        .asciiz "Please choose the algorithm function:\n"       #声明字符串
        funcflt:        .asciiz "\nPlease input a float:\n"
.text           #代码段
        main:
        #伪指令la、li
        #la $a0 XXX表示将待输出的数值或字符串传入$a0
        #li $v0 XXX表示输入或输出，1表示输出整数，2表示输出浮点数，4表示输出字符串，5表示接收整数，6表示接收浮点数
                #打印字符串
                la      $a0, funcint    #将待输出的字符串in放入$a0寄存器
                li      $v0, 4          #系统调用$v0=4时输出字符串（$a0中的值）
                syscall                 #系统调用
                #获取键盘输入的数字 int类型
                li      $v0, 5          #系统调用$v0=5时将标准输入读入$v0
                syscall                 #系统调用
                sw      $v0, 0($sp)     #将$v0中数据存入内存0($sp)
                #打印整数
                lw      $a0, 0($sp)     #将内存0($sp)中数据写入$a0寄存器
                li      $v0, 1          #系统调用$v0=1时输出整数（$a0中的值）
                syscall

                #打印字符串
                la      $a0, funcflt
                li      $v0, 4
                syscall
                #获取键盘输入的数字 float类型
                li      $v0, 6
                syscall
                s.s     $f0, 0($sp)
                #打印浮点数
                l.s     $f12, 0($sp)
                li      $v0, 2
                syscall
```

控制台输入输出：

```
Please choose the algorithm function:
3
3
Please input a float:
3.14159
3.14159
```

根据上述操作，设计本实验的输入输出如下。

1. 输入模块

```
#将输入浮点数存入相应寄存器并截取符号、指数、尾数
Input_funct:
        #打印"Please input the first float:\0"
        la $a0,Tips1
        li $v0,4
        syscall
        #系统调用读取输入的浮点数，存入$f0
        li $v0,6
        syscall
        #将$f0中的数据存入$s1并放入内存
        mfc1 $s1,$f0
        sw $s1,0($s5)
        #打印"Please input the second float:\0"
        la $a0,Tips2
        li $v0,4
        syscall
        #系统调用读取输入的浮点数，存入$f0
        li $v0,6
        syscall
        #将$f0中的数据存入$s2并放入内存
        mfc1 $s2,$f0
        sw $s2,0($s6)
```

2. 输出模块

输出结果前需要对分布在不同寄存器中的结果的符号、指数、尾数进行整合，采取的办法是将符号位左移 31 位，指数左移 23 位，尾数不动，因为三个 31 bit 数的其他位均为 0，因此三个数相加即可得到 IEEE754 表示的结果。

```
#将结果还原回31位数据
sll     $s0,    $s0,    31          #前面的处理已经将结果的符号位存入$s0，直接左移至最高位
sll     $s2,    $s2,    23          #将指数位移动至相应位置
sll     $t2,    $t2,    9           #$t2中存放了输出的尾数，为防止尾数23位，采用先左移再右移的方式只留下0~22位的数值
srl     $t2,    $t2,    9
add     $s2,    $s2,    $t2         #符号位+指数+尾数=结果
add     $s0,    $s0,    $s2
mtc1    $s0,    $f12
#输出
li      $v0,    2
syscall
li      $v0,    4
la      $a0,    NewLine
syscall
jr $ra
```

注意，此处输出前需要对结果进行判断，若偏阶指数超出 0~255 的范围，则计算结果溢出，需要给出异常提示：

```
#判断是否下溢
#单精度浮点数只有8位指数位（含有偏阶）
#若小于0则原指数小于-128，即结果下溢；
#若大于255则原指数大于127，即结果上溢出（精度原因，无法通过偏移解决）
blt     $s3,    0,      downOverflow    #下溢，跳转到downOverflow打印"Down Overflow Excption!"
#判断是上溢
bgt     $s2,    255,    upOverflow      #上溢，跳转到upOverflow打印"Up Overflow Excption!"
```

```
#最终结果下溢
downOverflow:
        la      $a0,    Overflow2
        li      $v0,    4
        syscall                         #打印"Down Overflow Excption!"
        jr $ra
#最终结果上溢
upOverflow:
        la      $a0,    Overflow1
        li      $v0,    4
        syscall                         #打印"Up Overflow Excption!"
        jr $ra
```

并且由于 IEE754 标准下的位数限制，尾数若超出 23 位不应认为是结果异常，只是一次精度缺失。为了将尾数限定在 23 位，将其先左移 9 位再右移 9 位即可。

（五）表达式解析函数设计

根据 IEEE754 单精度浮点数表示法，用 32 位二进制数表示一个浮点数，其中第 31 位为符号位，23~30 位为指数位，0~22 位为尾数位。因此解析表达式时，用特定的 32 位二进制数与浮点数按位与即可解析出其对应的符号位、指数和尾数。不同段对应特定按位与的数如下：

数据段	二进制按位与特征数	十六进制按位与特征数
符号位	32’b1000_0000_0000_0000_0000_0000_0000_0000	0x8000_0000
指数	32’b0111_1111_1000_0000_0000_0000_0000_0000	0x7f80_0000
尾数	32’b0000_0000_0111_1111_1111_1111_1111_1111	0x007f_ffff

再对指数减去偏阶 127 即可得到浮点数的带偏阶指数。上述过程汇编指令如下：

```
#将num1符号位存入4($s5)
andi  $t1,$s1,0x80000000        #0x80000000为16进制数，二进制为32'b1000_...._0000，和$s1中的num1按位与得到num1符号位（31位）
srl   $t1,$t1,31                #右移31位对齐
sw    $t1,4($s5)
#将num2符号位存入4($s6)
andi  $t1,$s2,0x80000000
srl   $t1,$t1,31
sw    $t1,4($s6)
#将num1指数存入8($s5)
andi  $t1,$s1,0x7f800000        #二进制为32'b0111_1111_1000_..._0000，和$s1中的num1按位与得到num1指数（23~30位）
srl   $t2,$t1,23                #右对齐
sw    $t2,8($s5)
#将num2指数存入8($s6)
andi  $t1,$s2,0x7f800000
srl   $t3,$t1,23
sw    $t3,8($s6)
#将num1尾数存入12($s5)
andi   $t1,$s1,0x007fffff       #二进制为32'b0000_..._0111_1111_..._1111，和$s1中的num1按位与得到num1尾数（0~22位）
sw     $t1,12($s5)
#将num2尾数存入12($s6)
andi  $t1,$s2,0x007fffff
sw    $t1,12($s6)
```

```
#将num1带偏阶指数存入16($s5)
addi $t4,$0,0x0000007f          #偏阶127
sub  $t1,$t2,$t4                #指数-偏阶得到带偏阶指数
sw   $t1,16($s5)
#将num2带偏阶指数存入16($s6)
sub  $t1,$t3,$t4
sw   $t1,16($s6)
```

三、实验过程（40%）

（一）设计工作日志

2022.3.20 10:00~12:00 复习 MIPS 指令集、IEEE754 浮点数表示法

2022.3.20 14:30~18:00 安装 JDK 和 Mars 仿真器

2022.3.24 15:00~17:30 （上次安装的 Mars 仿真器与 win10 系统不兼容）调试 Mars 及其打开方式

2022.3.26 09:30~12:00 学习 Mars 仿真器使用方法

2022.3.27 14:30~17:00 学习汇编语言基本语法，代码段与数据段等分布

2022.4.5 8:30~12:30 学习 MIPS 指令实现输入输出

2022.4.5 15:00~19:00 实现项目的输入输出模块并进行相关调试

（4 月初开始计组实验，因调试 Latex 的系统字体缺失问题，项目设计进度停滞了一段时间..... 后又因 Vivado 安装不当 C 盘爆满，只能重装电脑，直至四月底才恢复项目设计）

2022.4.30 18:30~21:30 学习 MIPS 解析表达式并实现该模块

2022.5.3 9:00~17:00 学习并实现浮点数加法模块，分模块书写调用

2022.5.4 8:30~12:30 调试昨天的加法模块，解决了部分问题

2022.5.7 10:00~12:00 继续调试加法模块，不同符号的浮点数加法仍然计算错误

2022.5.14 11:00~17:00 调试加法模块，加法模块基本完成

2022.5.15 20:00~22:00 依据加法模块的经验，自行编写减法模块

2022.5.21 9:00~12:00 发现减法模块与加法模块思路基本相同，发现可以复用加法模块，删除了原先编写的减法模块，稍加处理后复用加法

2022.5.22 9:00~12:00 减法模块在正数减负数的情况下计算错误，后调试成功

2022.5.26 14:00~18:00 更改优化输出模块，根据功能实现转二进制、十六进制的输出

2022.5.27 8:30~10:00 进行样例测试与异常测试并记录实验结果

2022.5.27 17:00~22:00 撰写实验报告

（二）主要的错误记录

1、错误 1

（1）**问题描述**：一开始设计完的输入模块不接受输入，一次性继续执行完，与预期相悖。

```
Please input the first float:
Please input the second float:
Your function number is out of index! Please input right function number:
Exit
Up Overflow Exception
Down Overflow Exception
The binary result is:
The hexadecimal result is: The decimal result is:
```

（2）**错误原因**：忘记在输出字符串后添加输入模块

（3）**解决办法**：加上输入模块即可

```
#打印"Please input the first float:\0"
la $a0,Tips1
li $v0,4
syscall
#系统调用读取输入的浮点数，存入$f0
li $v0,6
syscall
```

2、错误 2

（1）**问题描述**：对于非规范输入程序无法识别，会抛出异常

```
Please input the first float:3.622
Please input the second float:5.899
Please choose one function: 0 for exit, 1 for add, 2 for sub: 0-

Assemble: operation completed successfully.

Go: running MIPS_calculation.asm

Error in D:\KSoftware\Mars\workspace\MIPS_calculation.asm line 42: Runtime exception at 0x00400028: invalid integer input (syscall 5)

Go: execution terminated with errors.
```

（2）**错误原因**：输入不规范

（3）**解决办法**：在输入前加入提示输入格式，只接受规范化输入

3、错误 3

（1）**问题描述**：设计过程用到的寄存器数量太多，0~31 号寄存器数量有限，不够实现。

（2）**解决办法**：临时寄存器中的数据用完即可被覆盖，为后续指令提供空间。同时若保存寄存器内数据长时间不用或跳转至其他函数部分执行，可将保存寄存器内的值存入内存，等到需要时候再取回保存寄存器即可。

4、错误 4

（1）**问题描述**：执行完函数指令后无法跳回主函数继续执行，后续指令混乱。

（2）**解决办法**：跳转函数指令用 jar 跳转，将跳转前指令地址存入$ra 指令寄存器，函数末尾加上跳转指令 jr $ra，函数执行完毕后即可回到跳转前指令地址继续执行。

```
#跳转回调用函数前的PC（保存在指令寄存器$ra中）
jr $ra
```

5、错误 5

（1）**问题描述**：设计函数 Align_exp 进行对阶时，准备采用 C++中 while 循环思想进行汇编，后与其他同学交流后发现可以通过递归调用的思想，汇编过程极其简单，比 while 循环过程简单得多。

（2）**解决办法**：通过递归调用实现，具体代码如下：

```
#对阶过程采用递归，阶数小的右移1位再判断回到Align_exp函数开头或开始相加
Align_exp1:     #num1的阶小于num2的阶，num1阶数+1，尾数右移
        addi    $s2,    $s2,    1               #num1阶数+1
        srl     $s4,    $s4,    1               #num1尾数右移
        sub     $t0,    $s2,    $s3             #循环判断
        bltz    $t0,    Align_exp1              #branch if less than zero
        beqz    $t0,    beginAdd                #branch if equal zero跳到相加
Align_exp2:     #num1的阶大于num2的阶，num2阶数+1，尾数右移
        addi    $s3,    $s3,    1
        srl     $s5,    $s5,    1
        sub     $t0,    $s2,    $s3
        bgtz    $t0,    Align_exp2
        beqz    $t0,    beginAdd
```

6、错误 6

（1）**问题描述**：执行完一次计算后直接结束，无法实现多组输入的运算。

（2）**解决办法**：在 add_funct 函数末尾加上 j main 指令，回到主函数开头便可以重新接收浮点数输入。

7、错误 7

（1）**问题描述**：由于计算中存在多处相似模块可以复用，若全部重写则工程量巨大，因此不必重写，稍加处理复用即可。

（2）**解决办法**：对相似模块处理后跳转回已有模块即可。例如两个正数相减可以复用正数+负数的模块，负数-正数可以复用正数-负数的模块。

8、错误 8

（1）**问题描述**：一开始运算时没有将 num1 和 num2 的尾数中整数 1 补上，导致运算总是与预期相悖。

（2）**解决办法**：在解码时将 num1 和 num2 的尾数中整数 1 补全，便于后续计算。

9、错误 9

（1）**问题描述**：一开始运算时正数减负数结果为正数加负数的结果，其他组合的计算结果均正确。

```
Please input the first float:3.14
Please input the second float:6.28
Please choose one function: 0 for exit, 1 for add, 2 for sub: 2
The decimal result of calculation is:-3.1400003
The binary result of calculation is:11000000010010001111010111000100
The hexadecimal result of calculation is:C048F5C4
```

（2）**解决办法**：在将 num2 符号位取反时误写成了按位或，于是正数减负数的情况被解析成正数加负数，因此计算出错。改为按位异或即可。

```
lw      $t1,    4($s6)                  #num2的符号位存入$t1
xori    $t1,    $t1,    1               #将num2符号位按位异或（取反）
```

10、错误 10

（1）**问题描述**：一开始在数据段声明字符串时没有注意顺序，误将字符串在浮点数前声明，运行后出现地址不对齐而导致的错误。

```
line 69: Runtime exception at 0x00400064: store address not aligned on word boundary 0x10010033

Go: execution terminated with errors.
```

（2）**解决办法**：.ascii 不会在字符串后加上'\0'，而.asciiz 会在字符串加'\0'。两者均以字节为单位存储数据，.asciiz 之后分配的空间首地址有可能无法字对齐。因此将.ascii 和.asciiz 声明在最后面。

```
.data
#数据声明，声明代码中使用的变量名：
        function:       .asciiz "This is a project used for calculation of floats.\n"   #字符串
        num1:   .space 20       #20个字节空间$s5
```

四、设计结果

（一）设计交付物说明

项目源代码：MIPS_calculation.asm

实验报告：实验报告.pdf

（二）设计演示结果

下给出多组样例测试代码功能的完整性：

样例 1：**常规正数+正数**

```
Please input the first float:3.15698
Please input the second float:2.34569
Please choose one function: 0 for exit, 1 for add, 2 for sub: 1
The decimal result of calculation is:5.50267
The binary result of calculation is:01000000101100000001010111011111
The hexadecimal result of calculation is:40B015DF
```

样例 2：**常规正数+负数**

```
Please input the first float:5.689
Please input the second float:-1.234
Please choose one function: 0 for exit, 1 for add, 2 for sub: 1
The decimal result of calculation is:4.4550004
The binary result of calculation is:01000000100011101000111101011101
The hexadecimal result of calculation is:408E8F5D
```

样例 3：**常规负数+负数**

```
Please input the first float:-1.356
Please input the second float:-3.569
Please choose one function: 0 for exit, 1 for add, 2 for sub: 1
The decimal result of calculation is:-4.9249997
The binary result of calculation is:11000000100111011001100110011001
The hexadecimal result of calculation is:C09D9999
```

样例 4：**常规正数-正数**

```
Please input the first float:4.231
Please input the second float:3.698
Please choose one function: 0 for exit, 1 for add, 2 for sub: 2
The decimal result of calculation is:0.533
The binary result of calculation is:00111111000010000111001010110000
The hexadecimal result of calculation is:3F0872B0
```

样例 5：**常规正数-负数**

```
Please input the first float:9.659
Please input the second float:-3.568
Please choose one function: 0 for exit, 1 for add, 2 for sub: 2
The decimal result of calculation is:13.227
The binary result of calculation is:01000001010100111010000111001011
The hexadecimal result of calculation is:4153A1CB
```

样例 6：**常规负数-正数**

```
Please input the first float:-3.692
Please input the second float:-7.288
Please choose one function: 0 for exit, 1 for add, 2 for sub: 2
The decimal result of calculation is:3.5960002
The binary result of calculation is:01000000011001100010010011011110
The hexadecimal result of calculation is:406624DE
```

样例 7：**常规负数-负数**

```
Please input the first float:-6.3449
Please input the second float:-2.384
Please choose one function: 0 for exit, 1 for add, 2 for sub: 2
The decimal result of calculation is:-3.9609003
The binary result of calculation is:11000000011111010111111101100100
The hexadecimal result of calculation is:C07D7F64
```

样例 8：**不同指数的数相加**

```
Please input the first float:32.961
Please input the second float:1.544
Please choose one function: 0 for exit, 1 for add, 2 for sub: 1
The decimal result of calculation is:34.504997
The binary result of calculation is:01000010000010100000010100011110
The hexadecimal result of calculation is:420A051E
```

样例 9：**不同指数的数相减**

```
Please input the first float:365.512
Please input the second float:7.541
Please choose one function: 0 for exit, 1 for add, 2 for sub: 2
The decimal result of calculation is:357.971
The binary result of calculation is:01000011101100101111110001001010
The hexadecimal result of calculation is:43B2FC4A
```

样例 10：**不同指数与符号的数运算**

```
Please input the first float:305.674
Please input the second float:-0.59
Please choose one function: 0 for exit, 1 for add, 2 for sub: 1
The decimal result of calculation is:305.084
The binary result of calculation is:01000011100110001000101011000001
The hexadecimal result of calculation is:43988AC1
```

样例 12：**指数输入**

```
Please input the first float:3.4E30
Please input the second float:3.4E30
Please choose one function: 0 for exit, 1 for add, 2 for sub: 1
The decimal result of calculation is:6.8E30
The binary result of calculation is:01110010101010111010011111111000
The hexadecimal result of calculation is:72ABA7F8
```

样例 11：**溢出**

```
Please input the first float:99999999999999999999999999999999999999999999
Please input the second float:999999999999999999999999999999999999999999999
Please choose one function: 0 for exit, 1 for add, 2 for sub: 1
The decimal result of calculation is:Up Overflow Exception!
The binary result of calculation is:00000000000000000000000000000000
The hexadecimal result of calculation is:00000000
```

样例 12：**操作码错误**

```
Please input the first float:6.1234
Please input the second float:3.6669
Please choose one function: 0 for exit, 1 for add, 2 for sub: 3
Sorry, your function number is out of index! Please input right function number between 0 and 2.
```

根据上述多种样例，程序均能计算出正确结果，与预期相符。

五、总结（可选）

本实验实现了在没有浮点表示和计算硬件条件下，用软件方法实现 IEEE 754 单精度浮点数的表示及运算功能，并提供了人机交互方式供用户选择相应的功能。计算出相应结果后，以十进制、二进制和十六进制的方式显示输出。

经过该实验的学习与设计，我较为深入地了解了 IEEE754 要求下浮点数的存储及其在 MIPS 指令中和底层硬件中的计算步骤与方法，对硬件的底层运算和浮点数计算方法的了解具有较深的影响。

六、参考文献

[1] MIPS 汇编语言学习笔记 19：获取用户输入的整数
https://gaozhiyuan.net/assembly/mips-assembly-getting-users-input-integers.html
[2] MIPS 汇编语言学习笔记 20：获取用户输入的单精度浮点数
https://gaozhiyuan.net/assembly/mips-assembly-getting-users-input-floats.html
[3] MIPS 小总结
https://blog.csdn.net/qq_45551930/article/details/109641713
[4] IEEE-754 标准与浮点数运算
https://blog.csdn.net/m0_37972557/article/details/84594879
[5] FPGA 的 IEEE754 协议浮点数加减法运算原理及过程分析
https://blog.csdn.net/qq_33239106/article/details/111593264
[6]【计算机基础】详解 IEEE754 浮点数规格化表示（小数点左边隐含一位 1）
https://blog.csdn.net/liu_jiachen/article/details/100138857

七、附录

MIPS_calculation.asm 代码

```
#主函数流程：
    #读入浮点数和运算码 -> 解析符号、指数、尾数以及带偏阶指数 -> 根据运算功能码跳转到运算功能 -> 输出不同进制表示的结果
.data           #数据段（存放于内存中）
#数据声明，声明代码中使用的变量名：
    num1:       .space 20       #0($s5)：num1；4($s5)：符号位；8($s5)：指数；12($s5)：尾数；16($s5)：偏阶
    num2:       .space 20       #0($s6)：num2；4($s6)：符号位；8($s6)：指数；12($s6)：尾数；16($s6)：偏阶
    result:     .space 16       #0($s7)：二进制小数结果；4($s7)：指数；8($s7)：符号位；12($s7)：IEEE754 结果；

    Tips1:      .asciiz "Please input the first float:\0"
    Tips2:      .asciiz "Please input the second float:\0"
```

```
    Tips3:          .asciiz "Please choose one function: 0 for exit, 1 for add, 2 for sub: \0"
    Tips4:          .asciiz "Sorry, your function number is out of index! Please input right function number between 0 and 2. \n"
    Tips5:          .asciiz "Exit\0"
    Overflow1:      .asciiz "Up Overflow Excption!\n"
    Overflow2:      .asciiz "Down Overflow Excption!\n"
    Precision:.asciiz "Precision loss!\n"
    Ansb:           .asciiz     "The binary result of calculation is:\0"
    Ansh:           .asciiz"The hexadecimal result of calculation is:\0"
    Ansd:           .asciiz     "The decimal result of calculation is:\0"
    NewLine:        .asciiz "\n"

.text       #代码段
    main:     #开始执行
        #将 num1、num2 的首地址从内存写入寄存器
        la    $s5,       num1          #将 num1 的首地址保存到$s5 寄存器
        la    $s6, num2                #将 num2 的首地址保存到$s6 寄存器
        #跳转到输入函数，接收浮点数 num1、num2 并解析其符号、指数、尾数和带偏阶指数
        jal Input_funct                #jal 先将当前 PC 放入$ra 再跳转
        #输入计算功能（0 退出、1 加法、2 减法）
        la    $a0, Tips3
        li    $v0, 4
        syscall                        #打印"Please choose one function: 0 for exit, 1 for add, 2 for sub: \0"
        li    $v0, 5
        syscall                        #调用系统$v0=5 读取输入的整数值并存入$v0
        #此时$v0 存储计算功能码，分别比较 0、1、2 用以跳转至相应函数，若不在该区间则出现异常
        li    $t0, 1
        beq  $v0, $t0, add_funct      #加法
        li    $t0, 2
        beq  $v0, $t0, sub_funct      #减法
        li    $t0, 0
        beq  $v0, $t0, exit_funct     #退出
        bne  $v0, $t0, default_funct  #输入的不是 0-4

#将输入浮点数存入相应寄存器并截取符号、指数、尾数和带偏阶指数
Input_funct:
    #打印"Please input the first float:\0"
    la $a0,Tips1
    li $v0,4
    syscall
    #系统调用读取输入的浮点数，存入$f0
    li $v0,6
    syscall
    #将$f0 中的数据存入$s1 并放入内存
    mfc1 $s1,$f0
    sw $s1,0($s5)
```

```
        #打印"Please input the second float:\0"
    la $a0,Tips2
    li $v0,4
    syscall
    #系统调用读取输入的浮点数，存入$f0
    li $v0,6
    syscall
    #将$f0 中的数据存入$s2 并放入内存
    mfc1 $s2,$f0
    sw $s2,0($s6)
    #将 num1 符号位存入 4($s5)
    andi $t1,$s1,0x80000000          #0x80000000 为 16 进制数，二进制为 32'b1000_...._0000，
和$s1 中的 num1 按位与得到 num1 符号位（31 位）
    srl   $t1,$t1,31                 #右移 31 位对齐
    sw     $t1,4($s5)
    #将 num2 符号位存入 4($s6)
    andi $t1,$s2,0x80000000
    srl   $t1,$t1,31
    sw     $t1,4($s6)
    #将 num1 指数存入 8($s5)
    andi $t1,$s1,0x7f800000          #二进制为 32'b0111_1111_1000_..._0000，和$s1 中的
num1 按位与得到 num1 指数（23~30 位）
    srl   $t2,$t1,23                 #右对齐
    sw     $t2,8($s5)
    #将 num2 指数存入 8($s6)
    andi $t1,$s2,0x7f800000
    srl   $t3,$t1,23
    sw     $t3,8($s6)
    #将 num1 尾数存入 12($s5)
    andi    $t1,$s1,0x007fffff       #二进制为 32'b0000_..._0111_1111_..._1111，和$s1 中
的 num1 按位与得到 num1 尾数（0~22 位）
    sw      $t1,12($s5)
    #将 num2 尾数存入 12($s6)
    andi $t1,$s2,0x007fffff
    sw     $t1,12($s6)
    #将 num1 带偏阶指数存入 16($s5)
    addi $t4,$0,0x0000007f           #偏阶 127
    sub   $t1,$t2,$t4                #指数-偏阶得到带偏阶指数
    sw     $t1,16($s5)
    #将 num2 带偏阶指数存入 16($s6)
    sub   $t1,$t3,$t4
    sw     $t1,16($s6)
    #跳转回调用函数前的 PC（保存在指令寄存器$ra 中）
    jr $ra

#加法流程：取 num1、num2 的符号位、阶、尾数->补全尾数的整数位->对阶->执行加法运
算->输出
#add_funct->getAdd->(Align_exp)->beginAdd->Add_Same_sign/Add_Diff_sign->NumSRL->showA
ns
```

```
add_funct:
    jal   getAdd
    jal   binary
    jal   hex
    j     main                    #本次执行完毕，跳回主函数开头
getAdd:
    #取 num1 和 num2 的符号位
    lw    $s0, 4($s5)             #$s0 是 num1 的符号位，$s1 是 num2 的符号位
    lw    $s1, 4($s6)
    #取 num1 和 num2 的阶
    lw    $s2, 8($s5)             #$s2 是 num1 的阶，$s3 是 num2 的阶
    lw    $s3, 8($s6)
    #取 num1 和 num2 的尾数
    lw    $s4, 12($s5)            #$s4 是 num1 的尾数，$s5 是 num2 的尾数
    lw    $s5, 12($s6)
    #补全尾数的整数位 1
    ori   $s4, $s4, 0x00800000    #将整数位 1 补全
    ori   $s5, $s5, 0x00800000
    #对阶
    sub   $t0, $s2, $s3           #比较 num1 和 num2 的阶数（指数）大小
    bltz  $t0, Align_exp1         #$t0 小于 0，则表明 num1 的阶小于 num2，需将 num1 右移对阶
    bgtz $t0, Align_exp2          #$t0 大于 0，需将 num2 右移对阶
    beqz$t0, beginAdd             #两个数阶相同，则直接相加
#对阶：若两个加数阶数不同，进入相加前需要先对阶。将阶数小的向阶数大的对齐，因为反之会降低阶数大的数的精度
#对阶过程采用递归，阶数小的右移 1 位再判断回到 Align_exp 函数开头或开始相加
Align_exp1:    #num1 的阶小于 num2 的阶，num1 阶数+1，尾数右移
    addi $s2, $s2, 1              #num1 阶数+1
    srl   $s4, $s4, 1             #num1 尾数右移
    sub   $t0, $s2, $s3           #循环判断
    bltz  $t0, Align_exp1         #branch if less than zero
    beqz$t0, beginAdd             #branch if equal zero 跳到相加
Align_exp2:    #num1 的阶大于 num2 的阶，num2 阶数+1，尾数右移
    addi $s3, $s3, 1
    srl   $s5, $s5, 1
    sub   $t0, $s2, $s3
    bgtz $t0, Align_exp2
    beqz$t0, beginAdd
#此时 num1、num2 阶数相同，判断符号后才能相加
beginAdd:
    xor   $t1, $s0, $s1           #按位异或判断 num1、num2 符号是否相同（相同则$t1 存 32'b0，不同存 32'b1）
    beq  $t1, $zero,    Add_Same_sign    #num1、num2 符号相同，则直接加(Add_Same_sign)
    j     Add_Diff_sign                  #num1、num2 符号不同，跳转到(Add_Diff_sign)
#num1、num2 符号相同相加
Add_Same_sign:
    add  $t2, $s4, $s5            #尾数相加后的结果即为输出的尾数，但需要先判断是否溢出
```

```
    sge  $t3, $t2, 0x01000000   #set if greater or equal 判断上溢
    #因为两个无符号 23bit 二进制数相加，如果结果的第 24 位为 1，则发生了上溢，需要右移尾数、阶数+1
    bgtz $t3, NumSRL            #上溢则尾数右移
    j    showAns                #无溢出就跳转到结果输出部分
#num1、num2 符号相同相加后上溢出，需要尾数右移，阶数+1
NumSRL:
    srl  $t2, $t2, 1            #尾数右移
    addi $s2, $s2, 1            #阶数+1
    j    showAns                #此时阶数、尾数正确，可以输出
#num1、num2 符号不同相加
Add_Diff_sign:
    sub  $t2, $s4, $s5          #符号不同的数相加相当于先相减再加符号，但可能出现尾数过大（上溢）或过小（下溢）的情况
    bgtz $t2, Add_Diff_sign1    #如果 num1 的尾数比 num2 大，则跳转至 Add_Diff_sign1（结果与 num1 同号）
    bltz $t2, Add_Diff_sign2    #如果 num1 的尾数比 num2 小，则跳转至 Add_Diff_sign2（结果与 num2 同号）
    j    show0                  #如果它们的绝对值相等，则结果为 0，可以跳转到特殊结果输出
#num1、num2 符号不同相加，num1 尾数比 num2 大，输出前需要先判断是否上溢或下溢
Add_Diff_sign1:
    blt  $t2, 0x00800000, Add_Diff_sign11     #尾数太小，则需左移，将其规格化
    bge  $t2, 0x01000000, Add_Diff_sign12     #如果尾数没有过小，那么就需要判断上溢
    j    showAns                #既不上溢也不下溢的结果过可以直接输出
#num1、num2 符号不同相加，num1 尾数比 num2 大，结果尾数太小
Add_Diff_sign11:
    sll  $t2, $t2, 1            #左移扩大尾数
    subi $s2, $s2, 1            #阶数-1
    blt  $t2, 0x00800000, Add_Diff_sign11     #循环扩大尾数
    j    showAns
#num1、num2 符号不同相加，num1 尾数比 num2 大，结果尾数太大
Add_Diff_sign12:
    srl  $t2, $t2, 1            #左移缩小尾数
    addi $s2, $s2, 1            #阶数+1
    bge  $t2, 0x01000000, Add_Diff_sign12
    j    showAns
#num1、num2 符号不同相加，num1 尾数比 num2 小
Add_Diff_sign2:
    sub  $t2, $s5, $s4          #将$t2 中数化为正
    xori    $s0     $s0     0x00000001          #结果与 num2 同号
    j    Add_Diff_sign1         #模块复用

#减法可以复用加法模块（将 num2 符号取反即可）
sub_funct:
    lw   $t1, 4($s6)            #num2 的符号位存入$t1
    xori    $t1, $t1, 1         #将 num2 符号位按位异或（取反）
    sw   $t1, 4($s6)
```

```
    jal    getAdd
    jal    binary
    jal    hex
    j main
#op 输入 0 时退出
exit_funct:
    la    $a0,       Tips5
    li    $v0,       4
    syscall
    li    $v0, 10                    #结束程序
    syscall
#op 不符合规范时回到 main 开头重新输入
default_funct:
    la $a0,Tips4
    li $v0,4
    syscall
    j main

#打印不同进制的结果
showAns:
    #打印十进制结果
    li    $v0, 4
    la    $a0, Ansd
    syscall
    #判断是否下溢
    #单精度浮点数只有 8 位指数位（含有偏阶）
    #若小于 0 则原指数小于-128，即结果下溢；
    #若大于 255 则原指数大于 127，即结果上溢出（精度原因，无法通过偏移解决）
    blt   $s3, 0,    downOverflow      #下溢，跳转到 downOverflow 打印"Down Overflow
Excption!"
    #判断是上溢
    bgt   $s2, 255, upOverflow         #上溢，跳转到 upOverflow 打印"Up Overflow Excption!"
    #将结果还原回 31 位数据
    sll    $s0, $s0, 31                #前面的处理已经将结果的符号位存入$s0，直接左移至
最高位
    sll    $s2, $s2, 23                #将指数位移动至相应位置
    sll    $t2, $t2, 9                 #$t2 中存放了输出的尾数，为防止尾数 23 位，采用先
左移再右移的方式只留下 0~22 位的数值
    srl    $t2, $t2, 9
    add  $s2, $s2, $t2                 #符号位+指数+尾数=结果
    add  $s0, $s0, $s2
    mtc1       $s0, $f12
    #输出
    li    $v0, 2
    syscall
    li    $v0, 4
    la    $a0, NewLine
    syscall
    jr $ra
```

```
#最终结果下溢
downOverflow:
    la    $a0, Overflow2
    li    $v0, 4
    syscall                          #打印"Down Overflow Excption!"
    jr $ra
#最终结果上溢
upOverflow:
    la    $a0, Overflow1
    li    $v0, 4
    syscall                          #打印"Up Overflow Excption!"
    jr $ra

# 转化成二进制
binary:
    li    $v0, 4
    la    $a0, Ansb
    syscall                          #打印"The binary result of calculation is:"
    addu      $t5, $s0, $0           #$s0 中存放的 IEEE754 标准的计算结果
    add  $t6, $t5, $0
    addi $t7, $0,  32
    addi $t8, $t0, 0x80000000        #判断结果指数的正负
    addi $t9, $0,  0
binary_transfer:                     #执行完 binary 顺序执行 binary_transfer
    #$t6:IEEE754 标准的计算结果   $t7:32'b0000_..._0100_0000   $t8:结果的指数正负
    subi $t7, $t7, 1
    and  $t9, $t6, $t8
    srl  $t8, $t8, 1
    srlv $t9, $t9, $t7
    add  $a0, $t9, $0
    li    $v0, 1
    syscall
    beq  $t7, $t0, back
    j     binary_transfer
#转化成十六进制（用 4 位二进制转 1 位十六进制即可）
hex:
    li    $v0, 4
    la    $a0, Ansh
    syscall
    addi $t7, $0,  8
    add  $t6, $t5, $0
    add  $t9, $t5, $0
hex_transfer:
    beq  $t7, $0,  back
    subi $t7, $t7, 1
    srl  $t9, $t6, 28
    sll  $t6, $t6, 4
    bgt  $t9, 9,   getAscii
    li    $v0, 1
```

```
        addi $a0, $t9, 0
        syscall
        j       hex_transfer
#转变为 ascii 码
getAscii:
        addi $t9, $t9, 55
        li      $v0, 11
        add  $a0, $t9, $0
        syscall
        j       hex_transfer
#计算结果为 0 的输出
show0:
        mtc1        $zero,      $f12
        li      $v0, 2
        syscall
        jr $ra
#转化为指定进制输出后回到调用函数前的指令
back:
        la          $a0, NewLine
        li          $v0, 4
        syscall
        jr $ra
```