《计算机组成原理》项目报告

年级、专业、班级	计算机科学与技术（卓越）1 班	姓名	韩昊辰
实验题目	项目 IEEE754 单精度浮点数运算		
实验时间	2023/6/2	实验地点	重庆大学图书馆
实验成绩		实验性质	☐验证性 ☑设计性 ☐综合性

教师评价：

☑算法/实验过程正确；　☑源程序/实验内容提交　☑程序结构/实验步骤合理；

☑实验结果正确；　☑语法、语义正确；　☑报告规范；

其他：

评价教师签名：

一、实验目的

（1）深入掌握二进制数的表示方法以及不同进制数的转换
（2）掌握二进制不同编码的表示方法。
（3）掌握 IEEE 754 中单精度浮点数的表示和计算。

二、实验项目内容

假设没有浮点表示和计算的硬件，用软件方法采用仿真方式实现 IEEE 754 单精度浮点数的表示及运算功能，具体要求如下：

(1) 程序需要提供人机交互方式（字符界面）供用户选择相应的功能；

(2) 可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；

(3) 可实现浮点数的加减(或者乘除)运算；

(4) 使用 MIPS 汇编指令，但是不能直接使用浮点指令，只能利用整数运算指令来编写软件完成。

报告创建时间：2020.6.28

三、实验过程或算法（源程序）

1）程序整体流程分析

主函数流程如下：
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

读入数据时需要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。

运算支持加减法功能，其中加法流程为：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；减法则可以复用加法模块，只需要将 num2 符号取反即可

打印结果需要判断结果是否溢出并作相应处理。

2）定义变量

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"
```

3）代码段编写

主函数：

```
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
```

读取输入数据模块：

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
```

加法模块：

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
```

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，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
```

减法模块：

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
```

打印结果模块：

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
```

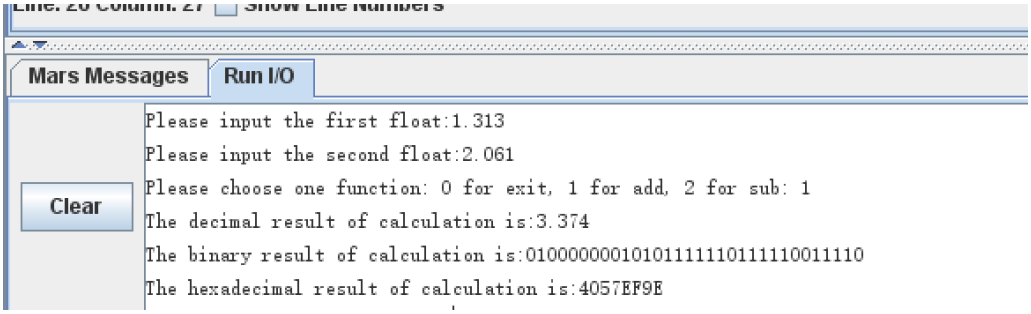
```
        syscall
        jr $ra
#最终结果下溢
downOverflow:
        la    $a0, Overflow2
        li    $v0, 4
        syscall                         #打印"Down Overflow Excption!"
        jr $ra
#最终结果上溢
upOverflow:
        la    $a0, Overflow1
        li    $v0, 4
        syscall                        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

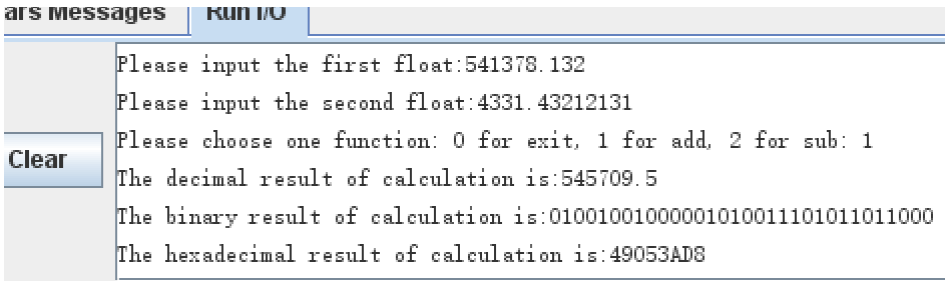
```
        sll    $t6,  $t6,  4
        bgt    $t9,  9,    getAscii
        li     $v0, 1
        addi $a0, $t9,  0
        syscall
        j      hex_transfer
#转变为 ascii 码
getAscii:
        addi $t9,  $t9,  55
        li     $v0, 11
        add  $a0, $t9,  $0
        syscall
        j      hex_transfer
#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
```

四、实验结果及分析和（或）源程序调试过程

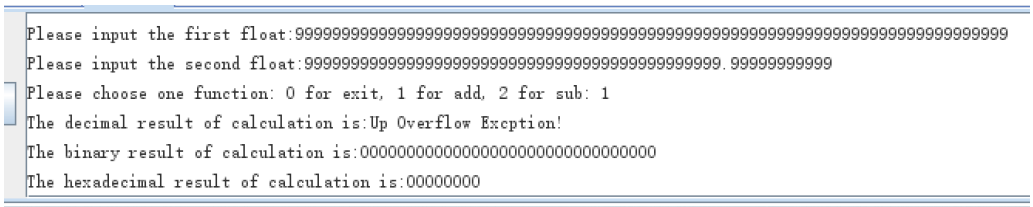
1）加法



运算实现 2.061+1.313，并将结果以二进制，十进制和十六进制输出

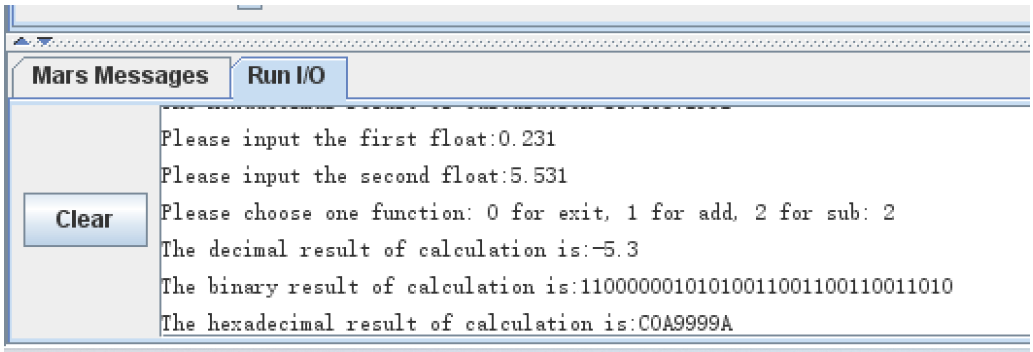


对较大数的加法检验



上溢检验

2）减法



运算实现 0.231-5.531，并将结果以二进制，十进制和十六进制输出