《计算机组成原理》实验报告

年级、专业、班级	2022 级计算机科学与技术(卓越)01 班	**姓名**	
实验题目	MIPS 汇编程序设计		
实验时间	2024 年 6 月 22 日	**实验地点**	
实验成绩	优秀/良好/中等	**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: 钟将

项目目标

(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, 掌握 IEEE 754 中单精度浮点数的表示和计算;

(3)能够应用 MIPS 汇编进行编程。

1 具体要求

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。
(5) 设计报告中给出程序的需求分析和关键算法的流程图，提交的代码注释比例 >40%，注释语义清晰。

2 实验设计

2.1 实验设计思路

2.1.1 设计目标

本设计实现了在没有浮点数表示和计算硬件的条件下，用软件方法实现 IEEE754 单精度浮点数的表示以及加减法运算的功能（即只使用整数运算指令来编写，而不是用浮点数汇编指令），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 IEEE754 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。

2.1.2 需求分析：

本程序首先需要用户使用键盘输入两个操作数，并指定运算符和输出格式。程序根据用户输入的两个操作数和运算符得到结果，并根据用户指定的输出格式将结果输出在屏幕上。

3 实验过程记录

3.1 实验设计

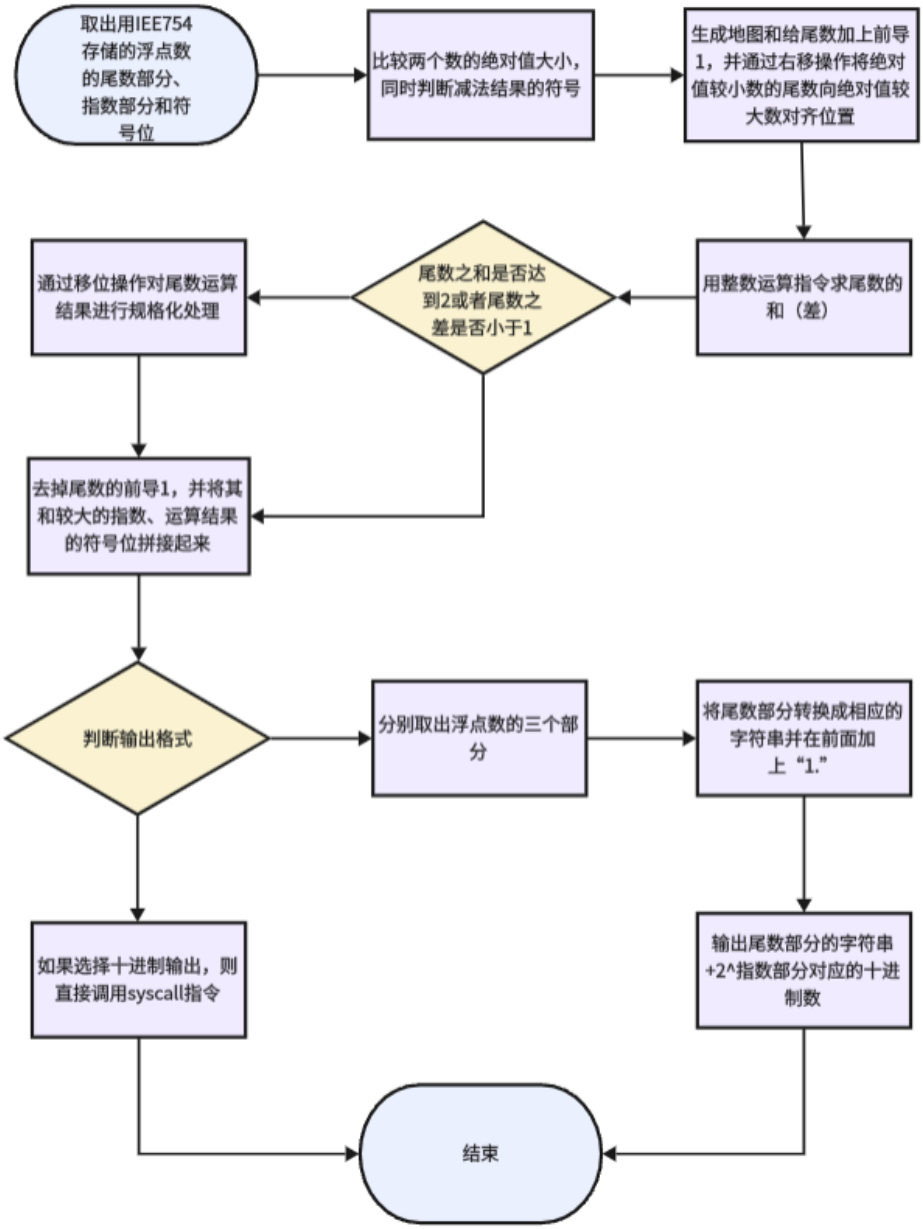

图 1: 算法流程图

3.2 问题以及解决方案

问题描述 1: 设计的过程中用到的寄存器数量太多,0 31 号寄存器数量有限,不够实现。

解决方案: 临时寄存器中的数据用完即可覆盖, 为后续的指令提供空间。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植入内存, 等到需要时候再取向保存寄存器即可。

问题描述 2: 在有嵌套函数的情况下,我们发现无法正确的跳转到上层主函数继续执行,从而导致后续指令混乱。

解决方案: 跳转指令使用 jar 跳转,但是惊喜嵌套的时候,需要将上层函数的返回地址 $ra 中的值通过栈指针 $sp 存入栈中,内嵌套函数结束后再将栈中的返回地址值传回 $ra 即可正确跳转回前指令地址继续进行。

问题描述 3: 执行完一次计算后直接结束,无法实现多组输入的输入。

解决方案: 在 add_funct 函数末尾加上 j main 指令,回到主函数开头即可重新接受浮点数输入。

问题描述 4: 运行时在 0x00400234 处出现了"address out of range" 错误。

解决方案: 这个错误通常发生在内存访问超出允许范围时。检查行 205 附近的内存访问指令。

4 实验结果与分析

4.1 实验结果图

```
Input the first float:1.11
Input the second float:2.22
Choose one function: 0 for exit, 1 for add, 2 for sub: 1
The decimal result of calculation is:3.33
The binary result of calculation is:01000000010101010001111010111000
The hexadecimal result of calculation is:40551EB8
```

图 2: 正数 + 正数

```
Input the first float:2.34
Input the second float:-2.11
Choose one function: 0 for exit, 1 for add, 2 for sub: 1
The decimal result of calculation is:0.23000002
The binary result of calculation is:00111110011010111000010100100000
The hexadecimal result of calculation is:3E6B8520
```

图 3: 正数 + 负数

```
Input the first float:1.23
Input the second float:1.11
Choose one function: 0 for exit, 1 for add, 2 for sub: 2
The decimal result of calculation is:0.120000005
The binary result of calculation is:00111101111110101111000010100100000
The hexadecimal result of calculation is:3DF5C290
```

图 4: 正数-正数

```
Input the first float:3.33
Input the second float:-2.34
Choose one function: 0 for exit, 1 for add, 2 for sub: 2
The decimal result of calculation is:5.6699996
The binary result of calculation is:01000000101101010111000010100011
The hexadecimal result of calculation is:40B570A3
```

图 5: 正数-负数

```
Input the first float:9999.3444
Input the second float:0.34
Choose one function: 0 for exit, 1 for add, 2 for sub: 1
The decimal result of calculation is:9999.685
The binary result of calculation is:01000110000111000011111010111101
The hexadecimal result of calculation is:461C3EBD
```

图 6: 不同指数的数相加

```
Input the first float:0.556
Input the second float:99.567
Choose one function: 0 for exit, 1 for add, 2 for sub: 2
The decimal result of calculation is:-99.011
The binary result of calculation is:11000010110001100000010110100010
The hexadecimal result of calculation is:C2C605A2
```

图 7: 不同指数的数相减

```
Input the first float:99999999999999999999999999999999999999999999999999999999999999999999
Input the second float:99999999999999999999999999999999999999999999999999999999999999999999
Choose one function: 0 for exit, 1 for add, 2 for sub: 1
The decimal result of calculation is:Up Overflow Excption!
The binary result of calculation is:00000000000000000000000000000000
The hexadecimal result of calculation is:00000000
```

图 8: 溢出

```
Input the first float:1
Input the second float:1
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.
```

图 9: 操作码错误

4.2 实验结果分析

实验结果显示,程序能够正确接收十进制实数输入,并准确转换为 IEEE 754 单精度浮点数格式。二进制和十六进制输出功能完善,浮点数加减运算也能正确执行。在 MIPS 汇编中,通过整数指令实现浮点运算的要求得到满足,证明了软件方法在硬件缺失情况下的可行性。

4.3 总结与体会

通过本项目,我深入理解了二进制数及不同进制数的转换,掌握了 IEEE 754 单精度浮点数的表示及运算方法,并在实践中熟悉了 MIPS 汇编语言。特别是在没有浮点硬件的情况下,通过整数指令模拟浮点运算,使我对计算机底层运算有了更深的认识。本次实验不仅增强了我的编程能力,也提高了我的问题解决能力和逻辑思维能力。

A MIPS 汇编代码

```
.data   # 数据段
# 数据声明, 声明代码中使用的变量名
  num1:    .space 20   # num1变量空间
  num2:    .space 20   # num2变量空间
  result:    .space 16   # result变量空间

  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: # 主函数
  # 将num1、num2的首地址加载到寄存器
  la    $s5,  num1      # 将num1的首地址加载到$s5
  la    $s6,  num2      # 将num2的首地址加载到$s6
  # 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       # 跳转到Input_funct
  # 输入计算功能（0退出、1加法、2减法）
  la  $a0,  Tips3
  li  $v0,  4
  syscall         # 打印提示信息
  li  $v0,  5
  syscall         # 读取输入的整数值并存入$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0-2

# 将输入浮点数存入相应寄存器并解析符号、指数、尾数和带偏阶指数
Input_funct:
  # 打印提示信息
  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)
  # 打印提示信息
  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)
  # 解析num1符号位
  andi $t1,$s1,0x80000000   # 提取符号位
  srl  $t1,$t1,31             # 右移对齐
  sw   $t1,4($s5)
  # 解析num2符号位
```

```
  andi $t1,$s2,0x80000000
  srl  $t1,$t1,31
  sw   $t1,4($s6)
  # 解析num1指数
  andi $t1,$s1,0x7f800000   # 提取指数位
  srl  $t2,$t1,23              # 右移对齐
  sw   $t2,8($s5)
  # 解析num2指数
  andi $t1,$s2,0x7f800000
  srl  $t3,$t1,23
  sw   $t3,8($s6)
  # 解析num1尾数
  andi  $t1,$s1,0x007fffff      # 提取尾数位
  sw    $t1,12($s5)
  # 解析num2尾数
  andi $t1,$s2,0x007fffff
  sw   $t1,12($s6)
  # 计算num1带偏阶指数
  addi $t4,$0,0x0000007f    # 偏阶127
  sub  $t1,$t2,$t4             # 指数减偏阶
  sw   $t1,16($s5)
  # 计算num2带偏阶指数
  sub  $t1,$t3,$t4
  sw   $t1,16($s6)
  # 跳转回调用函数前的PC
  jr $ra

# 加法流程: 取num1、num2的符号位、指数、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的指数
  lw  $s2,  8($s5)       # $s2是num1的指数, $s3是num2的指数
  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  # 补全整数位1
  ori $s5,  $s5,  0x00800000
  # 对阶
  sub $t0,  $s2,  $s3      # 比较num1和num2的指数大小
```

```
    bltz  $t0, Align_exp1  # num1指数小于num2, num1右移对阶
    bgtz  $t0, Align_exp2  # num1指数大于num2, num2右移对阶
    beqz  $t0, beginAdd    # 指数相同, 直接相加
# 对阶: 如果指数不同, 对齐指数, 较小的指数向较大的指数对齐
Align_exp1: # num1指数小于num2, num1指数+1, 尾数右移
    addi  $s2, $s2, 1   # num1指数+1
    srl $s4, $s4, 1   # num1尾数右移
    sub $t0, $s2, $s3     # 重新比较
    bltz  $t0, Align_exp1    # 循环对齐
    beqz  $t0, beginAdd    # 指数相同, 开始相加
Align_exp2: # num1指数大于num2, num2指数+1, 尾数右移
    addi  $s3, $s3, 1
    srl $s5, $s5, 1
    sub $t0, $s2, $s3
    bgtz  $t0, Align_exp2
    beqz  $t0, beginAdd
# 指数相同, 判断符号后相加
beginAdd:
    xor $t1, $s0, $s1     # 按位异或判断符号是否相同
    beq $t1, $zero, Add_Same_sign   # 符号相同, 直接相加
    j Add_Diff_sign       # 符号不同, 跳转减法
# 符号相同相加
Add_Same_sign:
    add $t2, $s4, $s5   # 尾数相加
    sge $t3, $t2, 0x01000000  # 判断上溢
    bgtz  $t3, NumSRL     # 上溢, 尾数右移
    j showAns       # 无溢出, 直接输出
# 符号相同相加后上溢, 尾数右移, 指数+1
NumSRL:
    srl $t2, $t2, 1   # 尾数右移
    addi  $s2, $s2, 1   # 指数+1
    j showAns       # 输出结果
# 符号不同相加
Add_Diff_sign:
    sub $t2, $s4, $s5   # 尾数相减
    bgtz  $t2, Add_Diff_sign1    # num1尾数大于num2
    bltz  $t2, Add_Diff_sign2    # num1尾数小于num2
    j show0       # 尾数相等, 结果为0
# 符号不同相加, num1尾数大于num2, 判断上溢或下溢
Add_Diff_sign1:
    blt $t2, 0x00800000, Add_Diff_sign11   # 尾数过小, 左移规格化
    bge $t2, 0x01000000, Add_Diff_sign12   # 尾数未过小, 判断上溢
    j showAns           # 无溢出, 直接输出
# 符号不同相加, num1尾数大于num2, 尾数过小
Add_Diff_sign11:
    sll $t2, $t2, 1       # 尾数左移
    subi  $s2, $s2, 1       # 指数-1
    blt $t2, 0x00800000, Add_Diff_sign11   # 循环扩大尾数
    j showAns
```

```
# 符号不同相加，num1尾数大于num2，尾数过大
Add_Diff_sign12:
  srl $t2, $t2, 1        # 尾数右移
  addi $s2, $s2, 1       # 指数+1
  bge $t2, 0x01000000, Add_Diff_sign12
  j showAns
# 符号不同相加，num1尾数小于num2
Add_Diff_sign2:
  sub $t2, $s5, $s4      # 取正数
  xori    $s0     $s0     0x00000001      # 结果与num2同号
  j Add_Diff_sign1       # 模块复用
# 减法，利用可重用的加法模块（反转num2的符号位）
sub_funct:
    lw    $t1,    4($s6)              # 将num2的符号位加载到$t1寄存器
    xori  $t1,    $t1,    1           # 反转num2的符号位（按位异或）
    sw    $t1,    4($s6)              # 将修改后的符号位存回到内存中
    jal   getAdd                      # 调用加法子程序
    jal   binary                      # 转换结果为二进制
    jal   hex                         # 转换结果为十六进制
    j     main                        # 返回主程序循环

# 当op输入为0时退出程序
exit_funct:
    la    $a0,    Tips5
    li    $v0,    4
    syscall
    li    $v0,    10                  # 结束程序
    syscall

# 当op输入不符合规范时返回到主程序重新输入
default_funct:
    la    $a0,    Tips4
    li    $v0,    4
    syscall
    j     main                        # 跳转回主程序

# 打印不同进制的结果
showAns:
    # 打印十进制结果
    li    $v0,    4
    la    $a0,    Ansd
    syscall

    # 检查是否下溢
    blt   $s3,    0,    downOverflow   # 如果下溢，跳转到downOverflow

    # 检查是否上溢
    bgt   $s2,    255,  upOverflow     # 如果上溢，跳转到upOverflow
```

```
    # 还原为31位数据
    sll   $s0,    $s0,  31            # 将符号位左移至最高位
    sll   $s2,    $s2,  23            # 将指数位移到相应位置
    sll   $t2,    $t2,  9             # 将尾数左移9位
    srl   $t2,    $t2,  9             # 只保留0~22位的尾数
    add   $s2,    $s2,  $t2           # 结合符号、指数和尾数
    add   $s0,    $s0,  $s2           # 合并为最终结果

    mtc1  $s0,    $f12                # 将结果移至浮点寄存器$f12

    # 打印IEEE754浮点数结果
    li    $v0,    2
    syscall

    li    $v0,    4
    la    $a0,    NewLine
    syscall
    jr    $ra                         # 返回调用函数

# 处理最终结果下溢情况
downOverflow:
    la    $a0,    Overflow2
    li    $v0,    4
    syscall
    jr    $ra                         # 返回调用函数

# 处理最终结果上溢情况
upOverflow:
    la    $a0,    Overflow1
    li    $v0,    4
    syscall
    jr    $ra                         # 返回调用函数

# 转换为二进制
binary:
    # 打印提示信息
    li    $v0,    4
    la    $a0,    Ansb
    syscall

    addu  $t5,    $s0,    $0          # 将IEEE754标准计算结果存入$t5
    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

    # 转换为二进制的循环
binary_transfer:
    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          # 返回到调用函数

    j     binary_transfer               # 继续二进制转换循环

# 转换为十六进制
hex:
    # 打印提示信息
    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          # 返回到调用函数
    subi  $t7,    $t7,    1
    srl   $t9,    $t6,    28
    sll   $t6,    $t6,    4
    bgt   $t9,    9,      getAscii      # 转换为ASCII码
    li    $v0,    1
    addi  $a0,    $t9,    0
    syscall
    j     hex_transfer                  # 继续十六进制转换循环

getAscii:
    addi  $t9,    $t9,    55
    li    $v0,    11
    add   $a0,    $t9,    $0
    syscall
    j     hex_transfer                  # 继续十六进制转换循环

# 如果结果为0，则显示结果
show0:
    mtc1  $zero,  $f12
    li    $v0,    2
    syscall
    jr    $ra                           # 返回调用函数

# 转换完成后返回调用函数
back:
    la    $a0,    NewLine
```

```
li      $v0,    4
syscall
jr      $ra                             # 返回调用函数
```