软件设计文档

设计简介

该软件实现了在没有浮点表示和计算硬件情况下，使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。软件实现了以下功能：

1. 提供了人机交互方式供用户选择相应功能。
2. 可以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. 可以进行浮点数的加减乘除基本运算。

需求分析

1. 软件要能够提供给用户GUI或字符界面的UI，使用户可以选择功能，进行浮点数转换或是浮点数计算。
2. 软件要能够接受十进制输入，在内存中以IEEE 754单精度方式表示后，以二进制和十六进制的方式显示输出。
3. 软件要能够根据用户输入的不同运算符进行基本运算。
4. 软件要使用MIPS等汇编指令实现，并且不可用浮点指令。

设计阶段

根据需求，可以设计不同的模块与之对应：

1. 实现UI时，可以使用字符界面，输入字符串提示用户输入选择的数字，同时使用变量接收用户输入，根据具体输入进行跳转。
2. 当用户进行浮点数转换时，提醒用户输入一个十进制浮点数并接收，在软件内部转换成二进制和十六进制表示后再输出给用户即可。
3. 当用户进行浮点数计算时，提醒用户输入一个表达式（例如1.25+1.25），输入后在软件内部进行计算与转换成二进制、十六进制后输出给用户即可。
4. 当用户进行浮点数计算时，要能够根据用户输入的不同运算符进行不同计算，因此需要实现不同符号对应的不同模块。

开发阶段

软件要求

因为要使用软件仿真，在此选用MARS（An IDE for MIPS Assembly Language Programming）。软件体积很小，已与设计文档一同打包。

1. UI界面开发

UI选择字符界面，因此需要先定义好提示给用户的字符串，方便后续调用。

```
.data
    input:              .space 100

    Conb:               .asciiz "==>The binary result is:\0"            # 转换输出
```

```
    Conh:               .asciiz "==>The hexadecimal result is:\0"       # 转换输出
    Ansb:               .asciiz "==>The binary result of calculation is:\0"
    Ansh:               .asciiz "==>The hexadecimal result of calculation is:\0"
    MsgInputExpressionWrong:    .asciiz "syntax error!\n"
    MsgWelcome:         .asciiz "==Please select a function==\n 1. Conversion\n
2. Calculation\n 3. Exit\n Enter a number(1-3):\0"
    MsgConversion:      .asciiz "Please enter the floating point number to be
converted:\n"
    MsgCalculation:     .asciiz "Please enter an expression, e.g. 1+1:\n"
    MsgReInput:         .asciiz "Wrong input, please try again!"
    NewLine:            .asciiz "\n"
    output:             .asciiz "syntax error!\n"
```

然后定义主函数根据用户输入进行跳转

```
.text

main:   #开始执行
    #欢迎界面(1.转换, 2.计算, 3.退出)
    la      $a0,    MsgWelcome
    li      $v0,    4
    syscall
    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1、2、3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,    BeginConversion     #跳转到转换
    li      $t0,    2
    beq     $v0,    $t0,    BeginCalculation    #跳转到计算
    li      $t0,    3
    beq     $v0,    $t0,    SystemExit          #退出
    bne     $v0,    $t0,    ErrorInput      #错误输入
```

2. 浮点数转换开发

当用户选择"1"（浮点数转换时），跳转到浮点数转换界面，此时需要给用户提示，提示用户输入一个浮点数。

```
BeginConversion:
    #打印提示
    la      $a0,    MsgConversion
    li      $v0,    4
    syscall
    #请求输入
    j       ReceiveInput
```

提示后跳转到接收输入的模块

```
    #读取第一个操作数
    la      $a0,    input
    addi    $sp,    $sp,    -8
    sw      $t0,    0($sp)
    sw      $t1,    4($sp)
    jal     StrToFloat
```

```
    lw      $t0,    0($sp)
    lw      $t1,    4($sp)
    addi    $s0,    $v0,    0
    addi    $sp,    $sp,    +8

    #不运算，直接展示为二进制
    bne     $t1,    10,     OpNotDisplay    #10是'\n'
    addi    $a0,    $s0,    0
    jal     FloatToBin
    jal     ConPrint
    jal     hex
    j       main
    OpNotDisplay:
```

接收输入后跳转到转换模块`FloatToBin`、二进制输出模块`ConPrint`、十六进制输出模块`hex`，输出后重新跳转回main，等待用户下一次进行输入选择。

3. 浮点数运算开发

当用户选择"2"（浮点数运算时），跳转到浮点数运算界面，此时需要给用户提示，提示用户输入一个表达式。

```
BeginCalculation:
    #打印提示
    la      $a0,    MsgCalculation
    li      $v0,    4
    syscall
    #请求输入
    j       ReceiveInput
```

提示后跳转到接收输入的模块

```
ReceiveInput:
    #输入
    jal     Scan

    #查找操作符
    la      $t0,    input
    FindOperationLoop:
    addi    $t0,    $t0,    +1
    lbu     $t1,    ($t0)
    li      $t2,    47      #48是'0'
    slt     $t2,    $t2,    $t1
    li      $t3,    58      #57是'9'
    slt     $t3,    $t1,    $t3
    li      $t4,    46      #46是'.'
    seq     $t4,    $t4,    $t1
    and     $t2,    $t2,    $t3
    or      $t2,    $t2,    $t4
    bnez    $t2,    FindOperationLoop
    FindOperationEnd:

    #读取第一个操作数
    la      $a0,    input
```

```
addi    $sp,    $sp,    -8
sw      $t0,    0($sp)
sw      $t1,    4($sp)
jal     StrToFloat
lw      $t0,    0($sp)
lw      $t1,    4($sp)
addi    $s0,    $v0,    0
addi    $sp,    $sp,    +8

#不运算，直接展示为二进制
bne     $t1,    10, OpNotDisplay    #10是'\n'
addi    $a0,    $s0,    0
jal     FloatToBin
jal     ConPrint
jal     hex
j       main
OpNotDisplay:

#获取第二个操作数
addi    $a0,    $t0,    +1
addi    $sp,    $sp,    -8
sw      $t1,    0($sp)
sw      $s0,    4($sp)
jal     StrToFloat
lw      $t1,    0($sp)
lw      $s0,    4($sp)
addi    $sp,    $sp,    +8
addi    $s1,    $v0,    0

#相加
bne     $t1,    43, OpNotAdd    #43是'+'
addi    $a0,    $s0,    0
addi    $a1,    $s1,    0
jal     FloatAdd
addi    $a0,    $v0,    0
addi    $s0,    $v0,    0
jal     FloatToBin
jal     CalPrint
jal     hex
j       main
OpNotAdd:

#相减
bne     $t1,    45, OpNotSub    #45是'-'
addi    $a0,    $s0,    0
addi    $a1,    $s1,    0
jal     FloatSub
addi    $a0,    $v0,    0
addi    $s0,    $v0,    0
jal     FloatToBin
jal     CalPrint
jal     hex
j       main
OpNotSub:

#相乘
```

```
    bne     $t1,    42, OpNotMul     #42是'*'
    addi    $a0,    $s0,    0
    addi    $a1,    $s1,    0
    jal     FloatMul
    addi    $a0,$v0,0
    addi    $s0,$v0,0
    jal     FloatToBin
    jal     CalPrint
    jal     hex
    j       main
    OpNotMul:

    #相除
    bne     $t1,    47, OpNotDiv     #47是'/'
    addi    $a0,    $s0,    0
    addi    $a1,    $s1,    0
    jal     FloatDiv
    addi    $a0,    $v0,    0
    addi    $s0,    $v0,    0
    jal     FloatToBin
    jal     CalPrint
    jal     hex
    j       main
    OpNotDiv:
    j       InputExpressionWrong
```

根据表达式中的运算符进行不同运算模块的跳转

3.1 加法模块开发

加法的一般流程是：

1. 拆开两个数，获取符号、指数、有效数
2. 对阶
3. 尾数相加
4. 规格化

根据该流程分别实现开发。

```
FloatAdd:
    #浮点数相加，v0 = a0 + a1
    #分别取符号位，指数位，尾数位
    lui     $t0,    0x8000      #0x8000是1000 0000 0000 0000
    and     $s0,    $a0,    $t0
    and     $s1,    $a1,    $t0
    lui     $t0,    0x7f80      #0x7f80是0111 1111 1000 0000
    and     $s2,    $a0,    $t0
    srl     $s2,    $s2,    23
    and     $s3,    $a1,    $t0
    srl     $s3,    $s3,    23
    lui     $t0,    0x007f      #0x007f是0000 0000 0111 1111
    ori     $t0,    $t0,    0xffff
    and     $s4,    $a0,    $t0
    and     $s5,    $a1,    $t0
    lui     $t0,    0x0080      #若浮点数非0,则补回第23位的1
```

```
beqz    $s2,    a0IsZero
or      $s4,    $s4,    $t0
a0IsZero:
beqz    $s3,    a1IsZero
or      $s5,    $s5,    $t0
a1IsZero:

#对阶
a0SiftRightLoop:
slt     $t0,    $s2,    $s3
beqz    $t0,    a0SiftRightEnd
srl     $s4,    $s4,    1
addi    $s2,    $s2,    +1
j       a0SiftRightLoop
a0SiftRightEnd:
a1SiftRightLoop:
slt     $t0,    $s3,    $s2
beqz    $t0,    a1SiftRightEnd
srl     $s5,    $s5,    1
addi    $s3,    $s3,    +1
j       a1SiftRightLoop
a1SiftRightEnd:

#尾数相加
beqz    $s0,    a0NotNeg
sub     $s4,    $zero,  $s4
a0NotNeg:
beqz    $s1,    a1NotNeg
sub     $s5,    $zero,  $s5
a1NotNeg:
add     $t0,    $s4,    $s5
bnez    $t0,    ResultNotZero
li      $v0,    0
jr      $ra
ResultNotZero:
srl     $t1,    $t0,    31
li      $v0,    0
beqz    $t1,    ResultNotNeg
lui     $v0,    0x8000      #0x8000是1000 0000 0000 0000
sub     $t0,    $zero,  $t0
ResultNotNeg:

#规格化
addi    $sp,    $sp,    -12
sw      $s2,    0($sp)
sw      $v0,    4($sp)
sw      $ra,    8($sp)
li      $a0,    0
addi    $a1,    $t0,    0
jal     Normalize
lw      $t0,    0($sp)
lw      $t1,    4($sp)
lw      $ra,    8($sp)
addi    $sp,    $sp,+12
add     $v1,    $v1,    $t0
sll     $v1,    $v1,    23  #23是指数位的起始位
```

```
    or      $v0,    $v0,    $t1
    or      $v0,    $v0,    $v1
    jr      $ra
```

3.2 减法模块开发

减法在加法的基础上开发：

1. 对第一个数的符号位取反
2. 调用加法模块

```
FloatSub:
    #浮点数相减，v0 = a0 - a1
    #对a1的符号位取反
    lui     $t0,    0x8000
    xor     $a1,    $a1,    $t0

    #a0加a1
    addi    $sp,    $sp,    -4
    sw      $ra,    0($sp)
    jal     FloatAdd
    lw      $ra,    0($sp)
    addi    $sp,    $sp,    +4
    jr      $ra
```

3.3 乘法模块开发

乘法的一般流程是：

1. 拆开两个数，获取符号、指数、有效数
2. 符号异或
3. 指数相加并减去一个偏阶
4. 尾数相乘并规格化

根据该流程分别实现开发。

```
FloatMul:
    #浮点数相乘，v0 = a0 * a1
    #分别取符号位，指数位，尾数位
    lui     $t0,    0x8000      #0x8000是1000 0000 0000 0000
    and     $s0,    $a0,    $t0
    and     $s1,    $a1,    $t0
    lui     $t0,    0x7f80      #0x7f80是0111 1111 1000 0000
    and     $s2,    $a0,    $t0
    srl     $s2,    $s2,    23
    and     $s3,    $a1,    $t0
    srl     $s3,    $s3,    23
    lui     $t0,    0x007f       #0x007f是0000 0000 0111 1111
    ori     $t0,    $t0,    0xffff
    and     $s4,    $a0,    $t0
    and     $s5,    $a1,    $t0
    lui     $t0,    0x0080      #若有浮点数为0,则直接返回0，否则补回第23位的1
    bnez    $s2,    MultiplierNotZero
    li      $v0,    0
```

```
    jr      $ra
    MultiplierNotZero:
    bnez    $s3,    MultiplicandNotZero
    li      $v0,    0
    jr      $ra
    MultiplicandNotZero:
    or      $s4,    $s4,    $t0
    or      $s5,    $s5,    $t0

    #符号异或
    xor     $v0,    $s0,    $s1

    #指数相加并减去一个偏阶
    add     $t0,    $s2,    $s3
    addi    $t0,    $t0,    -127

    #尾数相乘并规格化
    mult    $s4,    $s5
    mfhi    $a0
    mflo    $a1
    addi    $sp,    $sp,    -12
    sw      $v0,    0($sp)
    sw      $t0,    4($sp)
    sw      $ra,    8($sp)
    jal     Normalize
    lw      $t0,    0($sp)
    lw      $t1,    4($sp)
    lw      $ra,    8($sp)
    add     $t1,    $t1,    $v1
    addi    $t1,    $t1,    -23     #积的浮点位置在第56位和第55位之间，规格化默认浮点位置在第23位和第22位之间，因此这里指数减去23
    sll     $t1,    $t1,    23  #23是指数位的起始位
    or      $v0,    $v0,    $t0
    or      $v0,    $v0,    $t1
    jr      $ra
```

3.4 除法模块开发

除法的一般流程是：

1. 拆开两个数，获取符号、指数、有效数
2. 符号异或
3. 指数相减并加上一个偏阶
4. 尾数相除并规格化

根据该流程分别实现开发。

```
FloatDiv:
    #浮点数相除，v0 = a0 / a1
    #分别取符号位，指数位，尾数位
    lui     $t0,    0x8000      #0x8000是1000 0000 0000 0000
    and     $s0,    $a0,    $t0
    and     $s1,    $a1,    $t0
    lui     $t0,    0x7f80      #0x7f80是0111 1111 1000 0000
```

```
    and     $s2,    $a0,    $t0
    srl     $s2,    $s2,    23
    and     $s3,    $a1,    $t0
    srl     $s3,    $s3,    23
    lui     $t0,    0x007f      #0x007f是0000 0000 0111 1111
    ori     $t0,    $t0,    0xffff
    and     $s4,    $a0,    $t0
    and     $s5,    $a1,    $t0
    lui     $t0,    0x0080      #若a1为0则返回NaN，若a0为0则返回0，若都不为0则补回第23位
的1
    bnez    $s3,    DivisorNotZero
    lui     $v0,    0xffff      #指数位全1，尾数位不全为0，即NaN
    jr      $ra
    DivisorNotZero:
    bnez    $s2,    DividentNotZero
    li      $v0,    0
    jr      $ra
    DividentNotZero:
    or      $s4,    $s4,    $t0
    or      $s5,    $s5,    $t0

    #符号异或
    xor     $v0,    $s0,    $s1

    #指数相减并加回一个偏阶
    sub     $t0,    $s2,    $s3
    addi    $t0,    $t0,    +127

    #尾数相除
    li      $t1,    0       #商
    lui     $t2,    0x8000      #0x8000是1000 0000 0000 0000
    TryQuotientLoop:
    beqz    $t2,    TryQuotientEnd
    sle     $t3,    $s5,    $s4
    beqz    $t3,    QuotientNotOne
    sub     $s4,    $s4,    $s5
    or      $t1,    $t1,    $t2
    QuotientNotOne:
    sll     $s4,    $s4,    1
    srl     $t2,    $t2,    1
    j       TryQuotientLoop
    TryQuotientEnd:

    #规格化
    li      $a0,    0
    addi    $a1,    $t1,    0
    addi    $sp,    $sp,    -12
    sw      $v0,    0($sp)
    sw      $t0,    4($sp)
    sw      $ra,    8($sp)
    jal     ormalize
    lw      $t0,    0($sp)
    lw      $t1,    4($sp)
    lw      $ra,    8($sp)
    addi    $sp,    $sp,    +12
    add     $t1,    $t1,    $v1
```

```
    addi    $t1,    $t1,    -8  #商的浮点位置在第31位和30位之间，规格化默认浮点位置在第23
位和22位之间，因此这里指数位需要减去8
    sll     $t1,    $t1,    23  #23是指数位的起始位
    or      $v0,    $v0,    $t0
    or      $v0,    $v0,    $t1
    jr      $ra
```

测试阶段

运行程序进行测试。

UI测试

当运行软件时会显示：

```
==Please select a function==
1. Conversion
2. Calculation
3. Exit
Enter a number(1-3):
```

UI显示功能正常。

浮点数转换测试

提示用户输入1-3数字进行功能选择，此时选择1，会进入浮点数转换界面：

```
Enter a number(1-3):1
Please enter the floating point number to be converted:
```

会提示用户输入一个浮点数进行转换，此时可以输入浮点数进行测试：

```
3.1415926
==>The binary result is:0100 0000 0100 1001 0000 1111 1101 1010
==>The hexadecimal result of calculation is:40490FDA
```

使用在线转换网站验证结果：

十进制(3.1415926)的单精度浮点数值：**40490FDA**,(01000000010010010000111111011010)

请输入数值： 3.1415926 长度(1~25)

转换类型： ◉ 十进制转单精度浮点数 ○ 单精度浮点数转十进制 ○ STM明渠流量计MODBUS协议返回包

开始转换

发现转换结果一致，功能正常。

浮点数计算测试

浮点数转换结束后跳转回选择界面：

```
3.1415926
==>The binary result is:0100 0000 0100 1001 0000 1111 1101 1010
==>The hexadecimal result of calculation is:40490FDA
==Please select a function==
 1. Conversion
 2. Calculation
 3. Exit
Enter a number(1-3):
```

此时选择2进入计算界面：

```
==Please select a function==
 1. Conversion
 2. Calculation
 3. Exit
 Enter a number(1-3):2
Please enter an expression, e.g. 1+1:
```

加法功能测试

提示用户输入一个表达式，此时输入`3.1415+3.1415`测试加法功能：

```
3.1415+3.1415
==>The binary result of calculation is:0100 0000 1100 1001 0000 1110 0101 0110
==>The hexadecimal result of calculation is:40C90E56
```

使用在线网站验证结果，`3.1415+3.1415=6.283`：

十进制(6.283)的单精度浮点数值：**40C90E56**,(01000000110010010000111001010110)

请输入数值： 6.283 长度(1~25)

转换类型： ◉ 十进制转单精度浮点数 ○ 单精度浮点数转十进制 ○ STM明渠流量计MODBUS协议返回包

开始转换

发现结果一致，加法功能正常。

减法功能测试

再次选择计算功能，此时输入`3.1415-3`测试减法功能：

```
3.1415-3
==>The binary result of calculation is:0011 1110 0001 0000 1110 0101 0110 0000
==>The hexadecimal result of calculation is:3E10E560
```

使用在线网站验证结果，`3.1415-3=0.1415`：

十进制(0.1415)的单精度浮点数值：**3E10E560**,(00111110000100001110010101100000)

请输入数值： 0.1415 长度(1~25)

转换类型： ◉ 十进制转单精度浮点数 ○ 单精度浮点数转十进制 ○ STM明渠流量计MODBUS协议返回包

开始转换

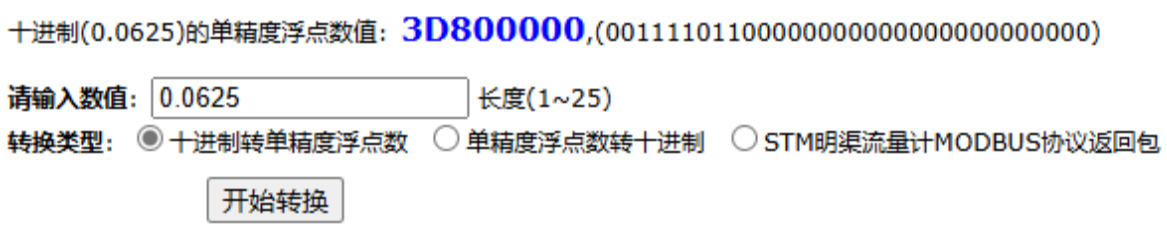
发现结果一致，减法功能正常。

乘法功能测试

再次选择计算功能，此时输入`0.25*0.25`测试乘法功能：

```
0.25*0.25
==>The binary result of calculation is:0011 1101 1000 0000 0000 0000 0000 0000
==>The hexadecimal result of calculation is:3D800000
```

使用在线网站验证结果，`0.25*0.25=0.0625`：



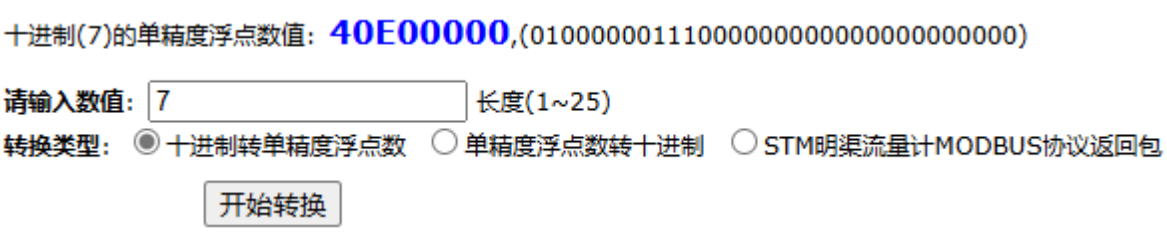
发现结果一致，乘法功能正常。

除法功能测试

再次选择计算功能，此时输入`3.5/0.5`测试除法功能：

```
3.5/0.5
==>The binary result of calculation is:0100 0000 1110 0000 0000 0000 0000 0000
==>The hexadecimal result of calculation is:40E00000
```

使用在线网站验证结果，`3.5/0.5=7`：



发现结果一致，除法功能正常。

测试完毕，功能正常，需求已全部实现。

参考

MARS（MIPS汇编程序和运行时模拟器） mars mips-CSDN博客

软件模拟实现IEEE-754单精度浮点数运算 - 『编程语言区』 - 吾爱破解 - LCG - LSG |安卓破解|病毒分析|www.52pojie.cn

IEEE 754单精度浮点数转十进制 / 十进制转单精度浮点数 / 在线转换 (styb.cn)

计算机组成原理：浮点数的加、减、乘、除运算（含实例完整运算） 浮点数运算-CSDN博客

IEEE754浮点数及其运算 非规格化数怎么计算 ieee754-CSDN博客

IEEE754 浮点数：简读+案例=秒懂 ieee754浮点数的计算-CSDN博客

《初学计算机组成原理之MIPS指令集及汇编》 mips汇编-CSDN博客

MIPS 指令集(共31条） mips指令 中大于等于号-CSDN博客

附录

本软件的所有代码

```
.data
    input:              .space 100

    Conb:               .asciiz "==>The binary result is:\0"            # 转换输出
    Conh:               .asciiz "==>The hexadecimal result is:\0"       # 转换输出
    Ansb:               .asciiz "==>The binary result of calculation is:\0"
    Ansh:               .asciiz "==>The hexadecimal result of calculation is:\0"
    MsgInputExpressionWrong:    .asciiz "syntax error!\n"
    MsgWelcome:         .asciiz "==Please select a function==\n 1. Conversion\n
2. Calculation\n 3. Exit\n Enter a number(1-3):\0"
    MsgConversion:      .asciiz "Please enter the floating point number to be
converted:\n"
    MsgCalculation:     .asciiz "Please enter an expression, e.g. 1+1:\n"
    MsgReInput:         .asciiz "Wrong input, please try again!"
    NewLine:            .asciiz "\n"
    output:             .asciiz "syntax error!\n"


.text

main:   #开始执行
    #欢迎界面(1.转换, 2.计算, 3.退出)
    la      $a0,    MsgWelcome
    li      $v0,    4
    syscall
    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1、2、3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,    BeginConversion     #跳转到转换
    li      $t0,    2
    beq     $v0,    $t0,    BeginCalculation    #跳转到计算
    li      $t0,    3
    beq     $v0,    $t0,    SystemExit          #退出
    bne     $v0,    $t0,    ErrorInput      #错误输入

BeginConversion:
    #打印提示
    la      $a0,    MsgConversion
    li      $v0,    4
    syscall
    #请求输入
    j       ReceiveInput

BeginCalculation:
    #打印提示
    la      $a0,    MsgCalculation
    li      $v0,    4
    syscall
    #请求输入
    j       ReceiveInput
```

```
SystemExit:
    #直接退出
    li      $v0,    10
    syscall

ReceiveInput:
    #输入
    jal     Scan

    #查找操作符
    la      $t0,    input
    FindOperationLoop:
    addi    $t0,    $t0,    +1
    lbu     $t1,    ($t0)
    li      $t2,    47      #48是'0'
    slt     $t2,    $t2,    $t1
    li      $t3,    58      #57是'9'
    slt     $t3,    $t1,    $t3
    li      $t4,    46      #46是'.'
    seq     $t4,    $t4,    $t1
    and     $t2,    $t2,    $t3
    or      $t2,    $t2,    $t4
    bnez    $t2,    FindOperationLoop
    FindOperationEnd:

    #读取第一个操作数
    la      $a0,    input
    addi    $sp,    $sp,    -8
    sw      $t0,    0($sp)
    sw      $t1,    4($sp)
    jal     StrToFloat
    lw      $t0,    0($sp)
    lw      $t1,    4($sp)
    addi    $s0,    $v0,    0
    addi    $sp,    $sp,    +8

    #不运算，直接展示为二进制
    bne     $t1,    10, OpNotDisplay    #10是'\n'
    addi    $a0,    $s0,    0
    jal     FloatToBin
    jal     ConPrint
    jal     hex
    j       main
    OpNotDisplay:

    #获取第二个操作数
    addi    $a0,    $t0,    +1
    addi    $sp,    $sp,    -8
    sw      $t1,    0($sp)
    sw      $s0,    4($sp)
    jal     StrToFloat
    lw      $t1,    0($sp)
    lw      $s0,    4($sp)
    addi    $sp,    $sp,    +8
```

```
    addi    $s1,    $v0,    0

    #相加
    bne     $t1,    43, OpNotAdd    #43是'+'
    addi    $a0,    $s0,    0
    addi    $a1,    $s1,    0
    jal     FloatAdd
    addi    $a0,    $v0,    0
    addi    $s0,    $v0,    0
    jal     FloatToBin
    jal     CalPrint
    jal     hex
    j       main
    OpNotAdd:

    #相减
    bne     $t1,    45, OpNotSub    #45是'-'
    addi    $a0,    $s0,    0
    addi    $a1,    $s1,    0
    jal     FloatSub
    addi    $a0,    $v0,    0
    addi    $s0,    $v0,    0
    jal     FloatToBin
    jal     CalPrint
    jal     hex
    j       main
    OpNotSub:

    #相乘
    bne     $t1,    42, OpNotMul     #42是'*'
    addi    $a0,    $s0,    0
    addi    $a1,    $s1,    0
    jal     FloatMul
    addi    $a0,$v0,0
    addi    $s0,$v0,0
    jal     FloatToBin
    jal     CalPrint
    jal     hex
    j       main
    OpNotMul:

    #相除
    bne     $t1,    47, OpNotDiv     #47是'/'
    addi    $a0,    $s0,    0
    addi    $a1,    $s1,    0
    jal     FloatDiv
    addi    $a0,    $v0,    0
    addi    $s0,    $v0,    0
    jal     FloatToBin
    jal     CalPrint
    jal     hex
    j       main
    OpNotDiv:
    j       InputExpressionWrong

InputExpressionWrong:
```

```
    la      $a0,    MsgInputExpressionWrong
    li      $v0,    4
    syscall
    j       SystemExit

ErrorInput:
    la      $a0,    MsgReInput
    li      $v0,    4
    syscall
    j       main

Scan:
    #输入字符串，输入的字符串保存在input中
    la      $a0,    input
    li      $a1,    100
    li      $v0,    8
    syscall
    jr      $ra

ConPrint:
    la      $a0,    Conb
    li      $v0,    4
    syscall
    la      $a0,    output
    li      $v0,    4
    syscall
    jr      $ra


CalPrint:
    #输出计算结果output
    la      $a0,    Ansb
    li      $v0,    4
    syscall
    la      $a0,    output
    li      $v0,    4
    syscall
    jr      $ra

StrToFloat:
    #将字符串转化成浮点数，输入变量a0是字符串,返回值v0是浮点数
    #判断符号
    li      $v0,    0
    lbu     $t0,    ($a0)
    bne     $t0,    43, IsMinus     #43是'+'
    lui     $v0,    0
    addi    $a0,    $a0,    +1
    IsMinus:
    bne     $t0,    45, IsMinusEnd  #45是'-'
    lui     $v0,    0x8000
    addi    $a0,    $a0,    +1
    IsMinusEnd:

    #找浮点字符在字符串的位置
    addi    $t0,    $a0,    0
    FindFloatPointLoop:
```

```
    lbu     $t1,    ($t0)
    li      $t2,    47          #47是'0'
    slt     $t2,    $t2,    $t1
    li      $t3,    58          #58是'9'
    slt     $t3,    $t1,    $t3
    and     $t2,    $t2,    $t3
    addi    $t0,    $t0,    +1
    bnez    $t2,    FindFloatPointLoop

    #将整数部分转化为二进制数
    addi    $sp,    $sp,    -16
    sw      $t0,    0($sp)
    sw      $t1,    4($sp)
    sw      $v0,    8($sp)
    sw      $ra,    12($sp)
    jal     StrToInt
    addi    $s0,    $v0,    0
    lw      $t0,    0($sp)
    lw      $t1,    4($sp)
    lw      $v0,    8($sp)
    lw      $ra,    12($sp)
    addi    $sp,    $sp,    16

    #将小数部分转化成定点小数
    li      $s1,    0
    bne     $t1,    46, StrNoFraction   #46是'.'
    addi    $sp,    $sp,    -12
    sw      $s0,    0($sp)
    sw      $v0,    4($sp)
    sw      $ra,    8($sp)
    addi    $a0,    $t0,    0
    jal     StrToInt
    addi    $a0,    $v0,    0
    addi    $a1,    $v1,    0
    jal     IntToSignificant
    addi    $s1,    $v0,    0
    lw      $s0,    0($sp)
    lw      $v0,    4($sp)
    lw      $ra,    8($sp)
    addi    $sp,    $sp,    12
    StrNoFraction:

    #判断是否为0
    bnez    $s0,    StrNotZero
    bnez    $s1,    StrNotZero
    li      $v0,    0
    jr      $ra
    StrNotZero:

    #规格化尾数并确定阶码
    addi    $sp,    $sp,    -8
    sw      $v0,    0($sp)
    sw      $ra,    4($sp)
    addi    $a0,    $s0,    0
    addi    $a1,    $s1,    0
    jal     Normalize
```

```
    lw      $t0,    0($sp)
    lw      $ra,    4($sp)
    addi    $sp,    $sp,    +8
    or      $v0,    $v0,    $t0
    addi    $v1,    $v1,    118     #127是偏阶，因为定点小数的浮点在31位，浮点数的浮点在
22位，所以这里阶码还要减回9
    sll     $v1,    $v1,    23  #23是指数位的起始位
    or      $v0,    $v0,    $v1
    jr      $ra

StrToInt:
    #将字符串转化成整数，输入变量a0是字符串，返回值v0是转化成的整数,v1是字符串的长度(包括前导0)
    li      $v0,    0
    li      $v1,    0
    CharToInt:
    lbu     $t1,    ($a0)
    li      $t2,    47      #48是'0'
    slt     $t3,    $t2,    $t1
    li      $t2,    58      #57是'9'
    slt     $t4,    $t1,    $t2
    and     $t3,    $t3,    $t4
    beqz    $t3,    CharToIntEnd
    addi    $t1,    $t1,    -48     #48是'0'
    mul     $v0,    $v0,    10
    add     $v0,    $v0,    $t1
    addi    $v1,    $v1,    +1
    addi    $a0,    $a0,    +1
    j       CharToInt
    CharToIntEnd:
    jr      $ra

IntToSignificant:
    #将整数转化成定点小数，a0是整数，a1是该整数十进制的长度(包括前导0)，v0是转化后的定点小数
    #计算10的a1次方
    li      $s0,    1
    MultTenLoop:
    beqz    $a1,    MultTenEnd
    mul     $s0,    $s0,    10
    addi    $a1,    $a1,    -1
    j       MultTenLoop
    MultTenEnd:

    #乘二取整法
    li      $v0,    0
    lui     $t0,    0x8000
    SubtractDivisorLoop:
    sll     $a0,    $a0,    1   #左移一位即乘二
    blt     $a0,    $s0,    NotSet1
    sub     $a0,    $a0,    $s0
    or      $v0,    $v0,    $t0
    NotSet1:
    srl     $t0,    $t0,    1
    bnez    $t0,    SubtractDivisorLoop
    jr      $ra

Normalize:
```

```
    #将浮点数规格化表示，a0是尾数的高字部分，a1是尾数的低字部分，v0是规格化后的尾数，v1是指数的变化值
    #最高位在第23位左边则右移
    li      $v1,    0
    lui     $t0,    0x0100      #0x0100是0000 0001 0000 0000，即只有第24位为1
    ShiftRightLoop:
    sne     $t1,    $a0,    0
    sleu    $t2,    $t0,    $a1
    or      $t1,    $t1,    $t2
    beqz    $t1,    ShiftRightEnd
    sll     $t2,    $a0,    31  #取高半字的第0位，补到低半字的第31位
    srl     $a0,    $a0,    1
    srl     $a1,    $a1,    1
    or      $a1,    $a1,    $t2
    addi    $v1,    $v1,    +1  #指数加一
    j       ShiftRightLoop
    ShiftRightEnd:

    #最高位在第23位右边则左移
    lui     $t0,    0x0080      #0x0080是0000 0000 1000 0000，即只有第23位为1
    ShiftLeftLoop:
    sltu    $t1,    $a1,    $t0
    beqz    $t1,    ShiftLeftEnd
    sll     $a1,    $a1,    1
    addi    $v1,    $v1,    -1
    j   ShiftLeftLoop
    ShiftLeftEnd:

    #省略尾数的第23位
    addi    $v0,    $a1,0
    not     $t0,    $t0         #t0取反后即为0xff7f ffff，只有第23为为0
    and     $v0,    $v0,    $t0
    jr      $ra


FloatToBin:
    #将浮点数转化成二进制字符串，a0是浮点数，转化后的字符串写入output中
    lui     $t0,    0x8000
    la      $t1,    output
    li      $t2,    8
    ReadHalfByteLoop:       #每读4bit加一个空格
    beqz    $t2,    ReadHalfByteEnd
    li      $t3,    4
    ReadBitLoop:
    beqz    $t3,    ReadBitEnd
    and     $t4,    $t0,    $a0
    srl     $t0,    $t0,    1
    beqz    $t4,    BitNotSet
    li      $t5,    49      #49是'1'
    j       BitNotSetEnd
    BitNotSet:
    li      $t5,    48          #48是'0'
    BitNotSetEnd:
    sb      $t5,    ($t1)
    addi    $t1,    $t1,    +1
    addi    $t3,    $t3,    -1
```

```
    j       ReadBitLoop
    ReadBitEnd:
    li      $t5,    32      #32是空格
    sb      $t5,    ($t1)
    addi    $t1,    $t1,    +1
    addi    $t2,    $t2,    -1
    j       ReadHalfByteLoop
    ReadHalfByteEnd:
    li      $t5,    10      #10是回车
    sb      $t5,    ($t1)
    jr      $ra

FloatAdd:
    #浮点数相加，v0 = a0 + a1
    #分别取符号位，指数位，尾数位
    lui     $t0,    0x8000      #0x8000是1000 0000 0000 0000
    and     $s0,    $a0,    $t0
    and     $s1,    $a1,    $t0
    lui     $t0,    0x7f80      #0x7f80是0111 1111 1000 0000
    and     $s2,    $a0,    $t0
    srl     $s2,    $s2,    23
    and     $s3,    $a1,    $t0
    srl     $s3,    $s3,    23
    lui     $t0,    0x007f      #0x007f是0000 0000 0111 1111
    ori     $t0,    $t0,    0xffff
    and     $s4,    $a0,    $t0
    and     $s5,    $a1,    $t0
    lui     $t0,    0x0080      #若浮点数非0,则补回第23位的1
    beqz    $s2,    a0IsZero
    or      $s4,    $s4,    $t0
    a0IsZero:
    beqz    $s3,    a1IsZero
    or      $s5,    $s5,    $t0
    a1IsZero:

    #对阶
    a0SiftRightLoop:
    slt     $t0,    $s2,    $s3
    beqz    $t0,    a0SiftRightEnd
    srl     $s4,    $s4,    1
    addi    $s2,    $s2,    +1
    j   a0SiftRightLoop
    a0SiftRightEnd:
    a1SiftRightLoop:
    slt     $t0,    $s3,    $s2
    beqz    $t0,    a1SiftRightEnd
    srl     $s5,    $s5,    1
    addi    $s3,    $s3,    +1
    j       a1SiftRightLoop
    a1SiftRightEnd:

    #尾数相加
    beqz    $s0,    a0NotNeg
    sub     $s4,    $zero,  $s4
    a0NotNeg:
    beqz    $s1,    a1NotNeg
```

```
    sub     $s5,    $zero,  $s5
    a1NotNeg:
    add     $t0,    $s4,    $s5
    bnez    $t0,    ResultNotZero
    li      $v0,    0
    jr      $ra
    ResultNotZero:
    srl     $t1,    $t0,    31
    li      $v0,    0
    beqz    $t1,    ResultNotNeg
    lui     $v0,    0x8000      #0x8000是1000 0000 0000 0000
    sub     $t0,    $zero,  $t0
    ResultNotNeg:

    #规格化
    addi    $sp,    $sp,    -12
    sw      $s2,    0($sp)
    sw      $v0,    4($sp)
    sw      $ra,    8($sp)
    li      $a0,    0
    addi    $a1,    $t0,    0
    jal     Normalize
    lw      $t0,    0($sp)
    lw      $t1,    4($sp)
    lw      $ra,    8($sp)
    addi    $sp,    $sp,+12
    add     $v1,    $v1,    $t0
    sll     $v1,    $v1,    23  #23是指数位的起始位
    or      $v0,    $v0,    $t1
    or      $v0,    $v0,    $v1
    jr      $ra

FloatSub:
    #浮点数相减，v0 = a0 - a1
    #对a1的符号位取反
    lui     $t0,    0x8000
    xor     $a1,    $a1,    $t0

    #a0加a1
    addi    $sp,    $sp,    -4
    sw      $ra,    0($sp)
    jal     FloatAdd
    lw      $ra,    0($sp)
    addi    $sp,    $sp,    +4
    jr      $ra

FloatMul:
    #浮点数相乘，v0 = a0 * a1
    #分别取符号位，指数位，尾数位
    lui     $t0,    0x8000      #0x8000是1000 0000 0000 0000
    and     $s0,    $a0,    $t0
    and     $s1,    $a1,    $t0
    lui     $t0,    0x7f80      #0x7f80是0111 1111 1000 0000
    and     $s2,    $a0,    $t0
    srl     $s2,    $s2,    23
    and     $s3,    $a1,    $t0
```

```
    srl     $s3,    $s3,    23
    lui     $t0,    0x007f       #0x007f是0000 0000 0111 1111
    ori     $t0,    $t0,    0xffff
    and     $s4,    $a0,    $t0
    and     $s5,    $a1,    $t0
    lui     $t0,    0x0080      #若有浮点数为0,则直接返回0，否则补回第23位的1
    bnez    $s2,    MultiplierNotZero
    li      $v0,    0
    jr      $ra
    MultiplierNotZero:
    bnez    $s3,    MultiplicandNotZero
    li      $v0,    0
    jr      $ra
    MultiplicandNotZero:
    or      $s4,    $s4,    $t0
    or      $s5,    $s5,    $t0

    #符号异或
    xor     $v0,    $s0,    $s1

    #指数相加并减去一个偏阶
    add     $t0,    $s2,    $s3
    addi    $t0,    $t0,    -127

    #尾数相乘并规格化
    mult    $s4,    $s5
    mfhi    $a0
    mflo    $a1
    addi    $sp,    $sp,    -12
    sw      $v0,    0($sp)
    sw      $t0,    4($sp)
    sw      $ra,    8($sp)
    jal     Normalize
    lw      $t0,    0($sp)
    lw      $t1,    4($sp)
    lw      $ra,    8($sp)
    add     $t1,    $t1,    $v1
    addi    $t1,    $t1,    -23     #积的浮点位置在第56位和第55位之间，规格化默认浮点位置在第23位和第22位之间，因此这里指数减去23
    sll     $t1,    $t1,    23  #23是指数位的起始位
    or      $v0,    $v0,    $t0
    or      $v0,    $v0,    $t1
    jr      $ra

FloatDiv:
    #浮点数相除，v0 = a0 / a1
    #分别取符号位，指数位，尾数位
    lui     $t0,    0x8000      #0x8000是1000 0000 0000 0000
    and     $s0,    $a0,    $t0
    and     $s1,    $a1,    $t0
    lui     $t0,    0x7f80      #0x7f80是0111 1111 1000 0000
    and     $s2,    $a0,    $t0
    srl     $s2,    $s2,    23
    and     $s3,    $a1,    $t0
    srl     $s3,    $s3,    23
    lui     $t0,    0x007f      #0x007f是0000 0000 0111 1111
```

```
    ori     $t0,    $t0,    0xffff
    and     $s4,    $a0,    $t0
    and     $s5,    $a1,    $t0
    lui     $t0,    0x0080      #若a1为0则返回NaN，若a0为0则返回0，若都不为0则补回第23位
的1
    bnez    $s3,    DivisorNotZero
    lui     $v0,    0xffff      #指数位全1，尾数位不全为0，即NaN
    jr      $ra
    DivisorNotZero:
    bnez    $s2,    DividentNotZero
    li      $v0,    0
    jr      $ra
    DividentNotZero:
    or      $s4,    $s4,    $t0
    or      $s5,    $s5,    $t0

    #符号异或
    xor     $v0,    $s0,    $s1

    #指数相减并加回一个偏阶
    sub     $t0,    $s2,    $s3
    addi    $t0,    $t0,    +127

    #尾数相除
    li      $t1,    0       #商
    lui     $t2,    0x8000      #0x8000是1000 0000 0000 0000
    TryQuotientLoop:
    beqz    $t2,    TryQuotientEnd
    sle     $t3,    $s5,    $s4
    beqz    $t3,    QuotientNotOne
    sub     $s4,    $s4,    $s5
    or      $t1,    $t1,    $t2
    QuotientNotOne:
    sll     $s4,    $s4,    1
    srl     $t2,    $t2,    1
    j       TryQuotientLoop
    TryQuotientEnd:

    #规格化
    li      $a0,    0
    addi    $a1,    $t1,    0
    addi    $sp,    $sp,    -12
    sw      $v0,    0($sp)
    sw      $t0,    4($sp)
    sw      $ra,    8($sp)
    jal     Normalize
    lw      $t0,    0($sp)
    lw      $t1,    4($sp)
    lw      $ra,    8($sp)
    addi    $sp,    $sp,    +12
    add     $t1,    $t1,    $v1
    addi    $t1,    $t1,    -8  #商的浮点位置在第31位和30位之间，规格化默认浮点位置在第23
位和22位之间，因此这里指数位需要减去8
    sll     $t1,    $t1,    23  #23是指数位的起始位
    or      $v0,    $v0,    $t0
    or      $v0,    $v0,    $t1
```

```
    jr      $ra

hex:
    #转化成十六进制（用4位二进制转1位十六进制即可）
    addu    $t5,    $s0,    $0      #$s0中存放的IEEE754标准的计算结果
    li      $v0,    4
    la      $a0,    Ansh
    syscall
    addi    $t7,    $0, 8
    add     $t6,    $t5,    $0
    add     $t9,    $t5,    $0
    j       HexTransfer

HexTransfer:
    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       HexTransfer

#转变为ascii码
getAscii:
    addi    $t9,    $t9,    55
    li      $v0,    11
    add     $a0,    $t9,    $0
    syscall
    j       HexTransfer

#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
```