

浙江大学

《计算机系统原理》实验报告

作业名称 运算器设计

小组成员

产品经理

指导老师 楼学庆

一、使用手册

首先，打开 calculator.exe，进入运算器。

在进入运算器后首先应该选择运算的对象，是整数还是浮点数，选择整数按 1，选择浮点数按 2。

请选择运算对象 (0-整数 1-浮点数) : 0

选择好之后，程序会要求选择运算符，此时键入一个符合要求的运算符。

请输入运算符: +、-、*、/、%: %

输入运算符后，程序会要求输入两个符合条件的数，此时键入数字。

请输入两个整数: 17 -3

输入完成后，程序就会输出模拟运算的结果：

结果为: -2






二、程序设计

1、编程语言：C++

2、编译环境：Visual Studio 2019 (v142)

3、文件结构

该程序的主要函数和常量均声明在 integer.h 和 float.h 文件中，定义在 integer.cpp 和 float.cpp 文件中。而 calculator.cpp 则实现了运算器与用户之间的交互功能。若要在其它项目中调用整数或是浮点数运算的函数，引用 integer.h 和 float.h 这两个头文件即可。

 calculator.cpp	2020/6/30 23:01	C++ Source	3 KB
 float.cpp	2020/6/30 16:58	C++ Source	9 KB
 float.h	2020/6/30 16:58	C/C++ Header	1 KB
 integer.cpp	2020/6/30 22:55	C++ Source	3 KB
 integer.h	2020/6/30 22:55	C/C++ Header	1 KB

4、主要功能

a) 整数运算

程序的主要数据结构为 word，其代表着整数的二进制形式，并且以补码的形式储存。由于 unsigned int 所占空间为 4byte，因此 word 为 32 位整数类型，可表示从 -2^{31} 到 $2^{31}-1$ 的整数。其定义如下所示：

```
1. typedef unsigned int word;
```

程序主要实现的功能有字符串与整数之间的转换以及整数的加减乘除四则运算。其实现函数如下所示：

```
1. word atom(char* ch);    // Transform string into binary number
2. char* mtoa(word num);   // Transform binary number into string
3. word madd(word m, word n); // Addition
4. word msub(word m, word n); // Substraction
5. word mmul(word m, word n); // Multiplication
6. word mdiv(word m, word n); // Division
7. word mmod(word m, word n); // Modulus
```

b) 浮点数运算

程序的主要数据结构为 dwrd，其代表着浮点数的二进制形式，可以表示 0、规格化浮点数、无穷大数（INF）和非数（NaN），其不能表示非规格化浮点数，原非规格化浮点数的位置被指数为 2^{-127} 的规格化浮点数占据。其定义如下所示：

```
1. typedef unsigned int dwrd; // Binary number
```

程序主要实现的功能有字符串与浮点数之间的转换以及浮点数的加减乘除四则运算。其实现函数如下所示：

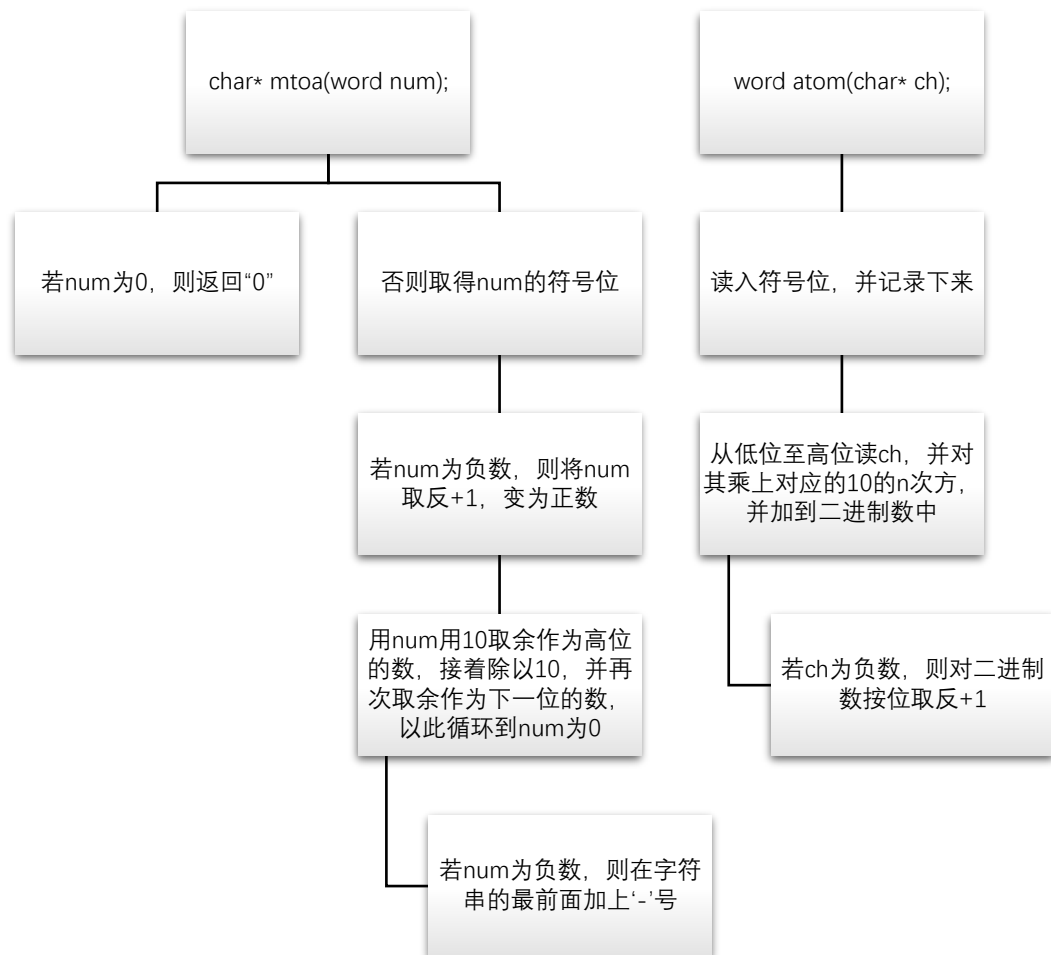
```
1. char* ftoa(dwrd a);      // Transform binary number into string
2. dwrd atof(char* s);      // Transform string into binary number
3. dwrd fadd(dwrd a, dwrd b); // Addition
4. dwrd fsub(dwrd a, dwrd b); // Subtraction
5. dwrd fmul(dwrd a, dwrd b); // Multiplication
6. dwrd fdiv(dwrd a, dwrd b); // Division
```

其中字符串与浮点数之间的转换对输入的数范围有较大限制，只能是在大约 $10^{\pm 7}$ 的范围之内。另外，浮点数转换为字符串的函数可以接受无穷大数（INF）和非数（NaN）的输

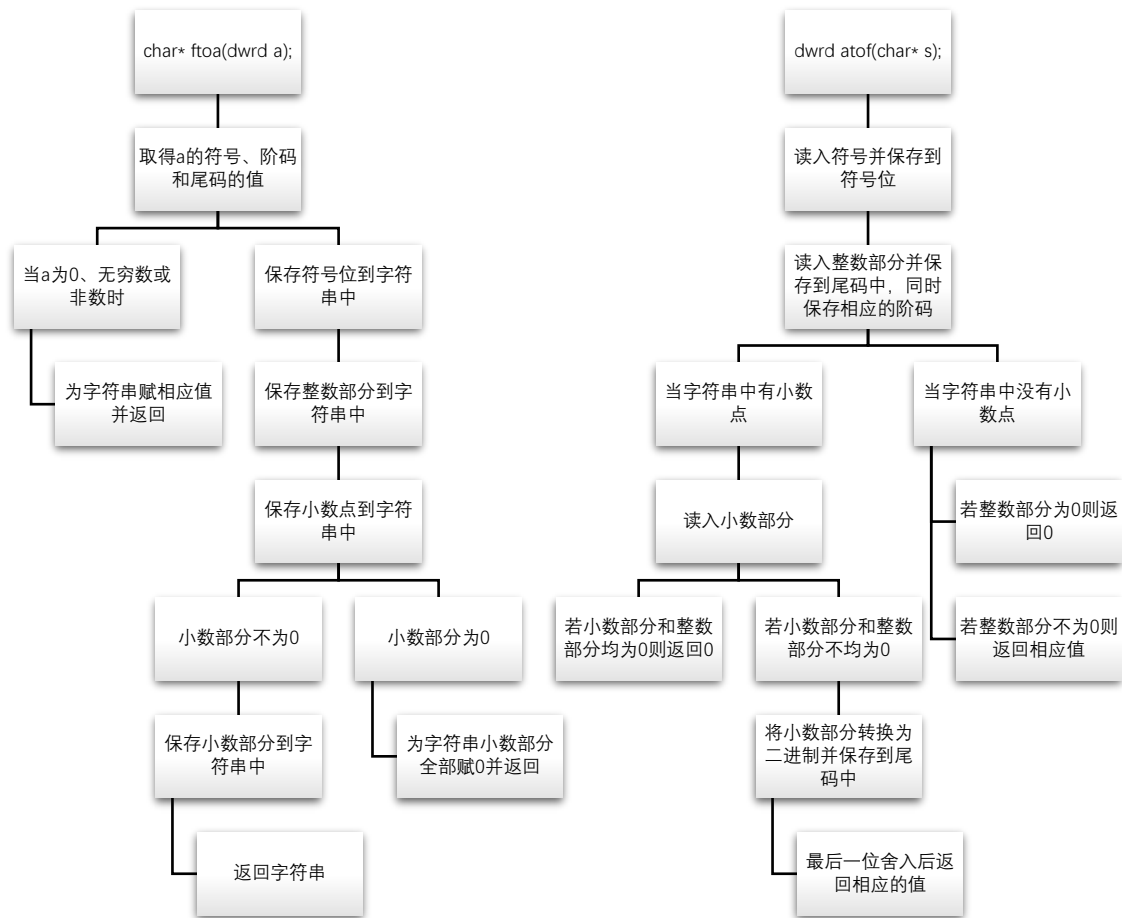
入，而且转换的字符串保留 18 位小数。而浮点数的加减乘除四则运算则可以接受 $2^{\pm 127}$ 范围内的数的输入，同时也能接受无穷大数（INF）和非数（NaN）的输入。

5、程序框图：

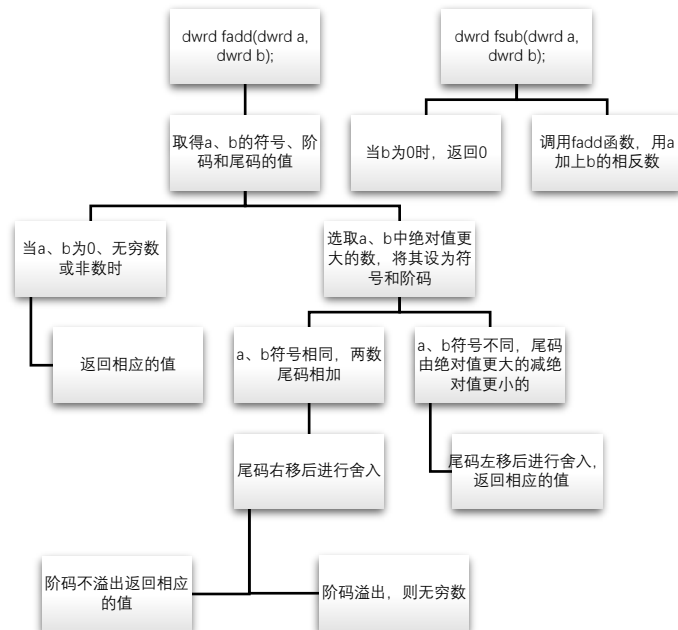
字符串与整数之间的转换：



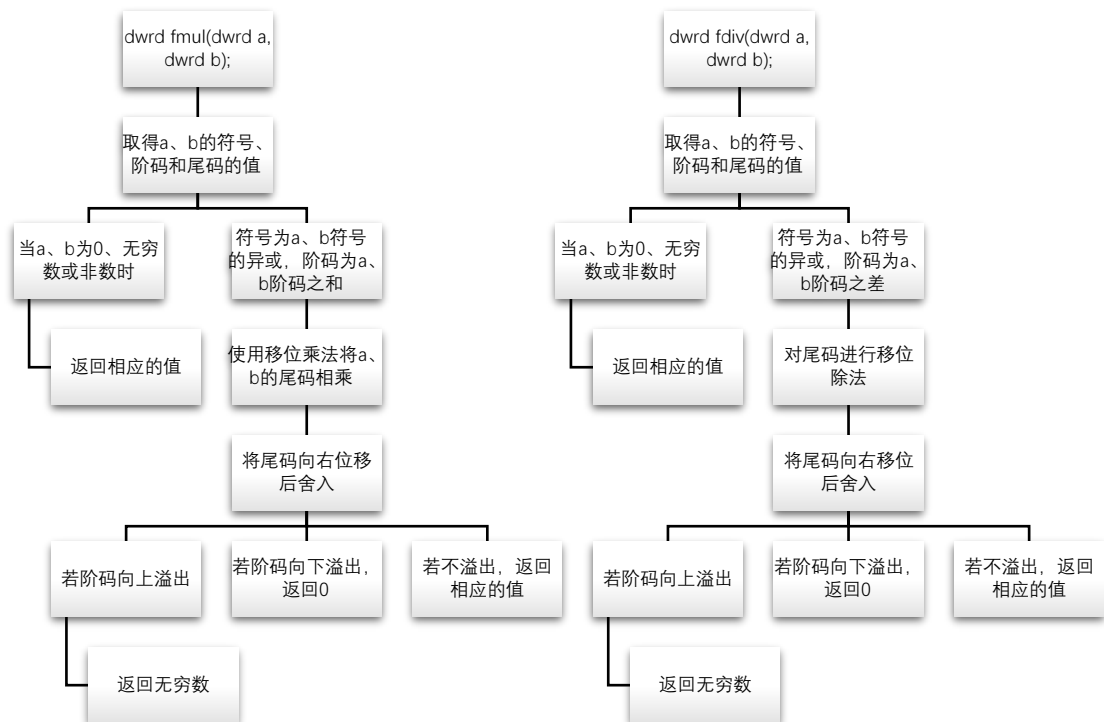
字符串与浮点数之间的转换：



浮点数的加减法：



浮点数的乘法:



三、实例分析

1、对浮点数函数进行单独的特殊测试

注：实例分析中红色加粗字体为输入，黄色荧光突出字体为测试结果。

测试样例 1 (大数、无穷数):

输入: 12345678901234567890123456789.123456789
1234567890123456789012345678.123456789

fadd 函数:

13580246243416241960736784384.00

fsub 函数:

11111110884774639981714997248.00

fmul 函数:

INF

fddiv 函数:

10.00

测试样例 2（小数、无穷数）：

[illegible][illegible]

fadd 函数:

[illegible]

fsub 函数:

0.000

fmul 函数:

INF

fddiv 函数:

1.000

从以上的两个测试样例可以看出，对浮点数进行加减乘除四则运算的四个函数对于大小数有很好的支持。另外，他们对于无穷数也有很好的支持。

2、对运算器进行综合测试

```
请选择运算对象(0-整数 1-浮点数): 0
请输入运算符: +、-、*、/、%: +
请输入两个整数: 22 -12
结果为: 10
```

```
请选择运算对象(0-整数 1-浮点数): 0
请输入运算符: +、-、*、/、%: %
请输入两个整数: 17 -3
结果为: -2
```

```
请选择运算对象(0-整数 1-浮点数): 1
请输入运算符: +、-、*、/: +
请输入两个浮点数: 12.1 23.12
结果为: 35.220001220703125000
```

```
请选择运算对象(0-整数 1-浮点数): 1
请输入运算符: +、-、*、/: *
请输入两个浮点数: 33839.1 22
结果为: 744460.25000000000000000000
```

四、 算法证明

1、 整数

a) 整数表示方法 (补码)

最高位为符号位，0 表示正数，1 表示负数，低位在正数时与原码低位相等，在负数时为原码低位取反加一。

$$X_{\text{补}} = (2^N + X) \bmod 2^N$$

b) 整数加法

加法

$$\begin{aligned} & \begin{cases} C_{i+1} = A_i B_i + B_i C_i + A_i C_i \\ S_i = A_i \oplus B_i \oplus C_i \end{cases} \\ & C_{i+1} = A_i B_i + (A_i + B_i) C_i \\ & P_i = A_i B_i \quad g_i = A_i + B_i \\ & C_{i+1} = P_i + g_i C_i \\ & \begin{cases} C_1 = P_0 + g_0 C_0 \\ C_2 = P_1 + g_1 C_1 = P_1 + g_1 P_0 + g_1 g_0 C_0 \\ C_3 = P_2 + g_2 C_2 = P_2 + g_2 P_1 + g_2 g_1 P_0 + g_2 g_1 g_0 C_0 \\ C_4 = P_3 + g_3 C_3 = P_3 + g_3 P_2 + g_3 g_2 P_1 + g_3 g_2 g_1 P_0 + g_3 g_2 g_1 g_0 C_0 \end{cases} \end{aligned}$$

$$\begin{aligned} S_1 &= A_1 \oplus B_1 \oplus C_1 & S_2 &= A_2 \oplus B_2 \oplus C_2 & S_3 &= A_3 \oplus B_3 \oplus C_3 & S_4 &= A_4 \oplus B_4 \oplus C_4 \\ C_5 &= P_4 + g_4 C_4 \\ & \vdots \end{aligned}$$

$$C_0 = 0$$

进位判断:

明显当最后一位进位时则进位
即 $C_{32} = 1$

溢出判断:

$C_{31} = 0 \quad C_{32} = 1 \Rightarrow$ 两个很小的负数相加导致溢出
 $C_{31} = 1 \quad C_{32} = 0 \Rightarrow$ 两个很大的正数相加导致溢出

算法分析:

该算法使得四个进位能够同时进行运算, 这也意味着和的四个位能够同时进行计算, 这就实现了组内并行, 组间串行的效果。在对 32 位的数进行运算时, 重复八次即可得到结果, 减少了运算时间。

c) 整数减法

减法

B 取相反数: $\begin{cases} C_0 = 1 \\ B_1 = \sim B_1 \quad B_2 = \sim B_2 \quad B_3 = \sim B_3 \dots \end{cases}$

其它算法推导过程与加法相同

进位判断: 与加法相反

溢出判断: 与加法相同

算法分析: 与加法相同

d) 整数乘法

乘法

$$\begin{aligned}
 X_{\text{补}} &= (2^N + X) \bmod 2^N \\
 aX_{\text{补}} &= (a2^N + aX) \bmod 2^N = (aX)_{\text{补}} \\
 (X * Y)_{\text{补}} &= X * Y_{\text{补}} \rightarrow g(X_{\text{补}}) * Y_{\text{补}} \\
 X_{\text{补}} &= a_{31}a_{30} \dots a_1a_0 \\
 X &= -a_{31}2^{31} + a_{30}2^{30} + \dots + a_12^1 + a_02^0 \\
 (X * Y)_{\text{补}} &= X * Y_{\text{补}} = (-a_{31}2^{31} + a_{30}2^{30} + \dots + a_12^1 + a_02^0) * Y_{\text{补}} \\
 &= [(-a_{31} + a_{30})2^{31} + (-a_{30} + a_{29})2^{30} + \dots + (-a_1 + a_0)2^1 + (a_0 + 0)2^0] * Y_{\text{补}}
 \end{aligned}$$

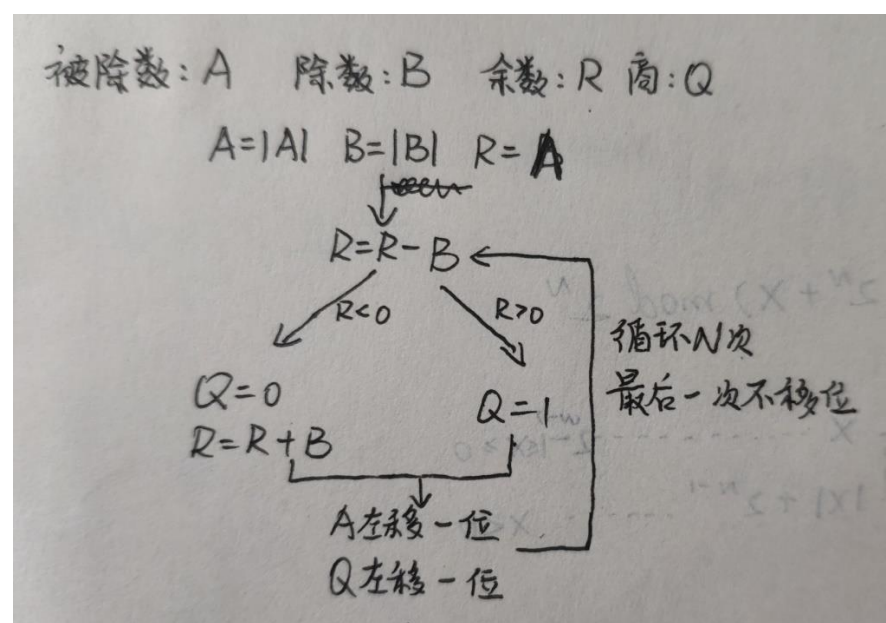
如果 $a_1 = a_0$, 部分积 $[Z_i]$ 加 0, 再右移一位
 如果 $a_1 a_0 = 01$, 部分积 $[Z_i]$ 加 $X_{\text{补}}$, 再右移一位
 如果 $a_1 a_0 = 10$, 部分积 $[Z_i]$ 加 $(-X)_{\text{补}}$, 再右移一位

算法分析:

该算法是一位 booth 算法, 大大减小了运算所需要的空间和时间

e) 整数除法

除法



A、B同号 Q不变。A、B异号 Q取反

算法分析：除法的算法较为复杂，没有较为简洁的方法

f) 整数取余

取余

A为正数 R不变。A为负数 R取反

其它与除法相同

算法分析：与除法相同

g) 整数大小比较

大小比较

设 $A = a_{31}a_{30}\dots a_1a_0$ $B = b_{31}b_{30}\dots b_1b_0$
① $a_{31} = 1$ $b_{31} = 0 \Rightarrow A < B$
② $a_{31} = 0$ $b_{31} = 1 \Rightarrow A > B$
③ $a_{31} = b_{31}$ $C = A - B = c_{31}c_{30}\dots c_1c_0$
 $c_{31} = 1 \Rightarrow A < B$
 $c_{31} = c_{30} = \dots = c_1 = c_0 = 0 \Rightarrow A = B$
否则 $\Rightarrow A > B$

h) 整数字位扩展

字位扩展

$$\therefore X_{补} = (2^N + X) \bmod 2^N$$

\therefore 字位扩展时在左边添加位，且与原最高位相同

2、浮点数

a) 浮点数结构

符号位 (1-bit) 阶码 (8-bit) 尾码 (23-bit)

符号位: 0 为正数, 1 为负数。

阶码: 以移码形式储存, 但是以 127 为基数, 存储的是 2 的指数。

尾码: 以原码形式储存, 但是没有符号位, 存储的是二进制系数的小数部分, 整数部分省略, 恒为 1。

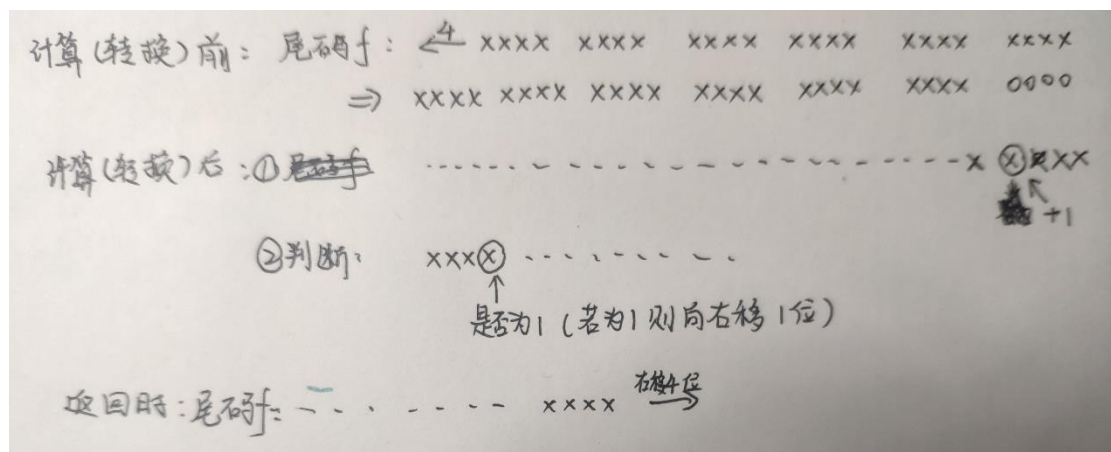
特殊数: 0: 阶码尾码均为 0

无穷数 (INF): 阶码为 255, 尾码为 0

非数 (NaN): 阶码为 255, 尾码不为 0

b) 舍入与溢出判断

舍入:



溢出判断: 判断阶码是否大于等于 255 或者小于 0

c) 二进制转换为字符串

① 获取二进制数的符号位 s 、阶码 e 、尾码 f ，并 ~~在尾码前加上 1~~

② 读取符号位

③ 读取整数：尾码 $\text{XXXX XXXX XXXX XXXX XXXX XXXX XXXX}$
 $\xrightarrow{e+1-127}$
 ↓
 整数部分

④ 储在小数点

⑤ 将小数部分移至第 28 位： $f: \text{XXXX XXXX XXXX XXXX XXXX XXXX XXXX}$
 $\Rightarrow \text{XXXX XXXX XXXX XXXX XXXX XXXX XXXX}$
 000 000 000

⑥ 设置基数为 10^n ，则最后小数点就会有 n 位。
~~设置基数为 $10^{10000000}$ ，即比小数高一位。~~

⑦ 开始循环。从小数第 28 位从左向右看，每移一次基数除以 2，若遇到 1，
 则在初始值为 0 的 decimal 变量上加上此时的基数。
 此过程实际上是以整数形式模拟 XX.10100... 的过程。
 $\downarrow \downarrow$
 $0.5 + 0.125 + \dots$

即 XX.10100...
 $\downarrow \downarrow$
 $3 \times 10^{-1} + 1.25 \times 10^{-1} + \dots$

⑧ 将小数部分储存，并返回

d) 字符串转换为二进制

① 读取符号位，保存到符号 s 中。

② 读取整数，保存到尾码中

③ 左移尾码 $\text{0000 0000 0000 0000 00 00 XXXX XXXX}$
 $\xleftarrow{\text{右移到最高位至 24 位，并记录数 } i}$
 整数

④ 阶码 $e = 23 - i + 127$

⑤ 读取小数部分，~~保存为整数形式~~，并设置哨兵。
 例： xx.37
 $\downarrow \quad \downarrow$
 $27 \quad 100$

⑥ 转换小数部分：
 例： $\text{xx.37} : 0.37 \times 2 = 0.74 < 1, \quad 0.74 \times 2 = 1.48 > 1$
 $\swarrow \quad \nwarrow$
 xx.01...
 $\swarrow \quad \nwarrow$
 $37 \times 2 = 74 < 100, \quad 74 \times 2 = 148$

⑦ 进行舍入并返回

e) 浮点数加法

- ① a、b 比较绝对值大小，将大的的符号和阶码保存
- ② 将较小一方的阶码和尾码调整至和较大方一样
- ③ 符号相同相加，尾码右移
- ④ 符号相异相减，尾码左移
- ⑤ 判断舍入与溢出并返回

f) 浮点数减法

用被减数加上减数的相反数

g) 浮点数乘法

- ① 两数符号相异则为负(1)，否则为正(0)
- ② ~~阶码 $e = ea + eb = ea_{\text{码}}$~~
 $e = ea + eb \quad e_{\text{码}} = e + 127 \Rightarrow e_{\text{码}} = ea_{\text{码}} + eb_{\text{码}} - 127$
- ③ 两数相乘，取前28位为尾码
- ④ 判断舍入与溢出并返回

h) 浮点数除法

- ① 两数符号相异为负(1)，否则为正(0)
- ② $e = ea - eb \quad e_{\text{码}} = e + 127 \Rightarrow e_{\text{码}} = ea_{\text{码}} - eb_{\text{码}} + 127$
- ③ 两数作移位除法，取28位
- ④ 判断舍入与溢出并返回