浮点数的加减法运算_ruidianbaihuo 的博客-CSDN 博客_浮点数相加

浮点数与定点数相比较有两个比较明显地特点：1、小数点位置不固定，但是在浮点数加减法运算的时候，小数点必须对齐；2、存储器中存储的不是浮点数的直接值，而存储的是符号，尾数，移码/阶码三种要素，所以不能直接相加减，实际上，针对浮点数的加减法运算，只需要对代表浮点数有效部分的尾数进行加减运算。
所以，浮点数的运算相对整数运算更复杂，有关浮点数的加减法运算包括以下5个基本步骤：对阶，尾数运算、规格化处理、舍入处理、益出处理。
—引自王达老师《深入理解计算机网络》一书

一、浮点数运算(总共包括5个步骤):

- 1、**对阶**
  对阶就是比较参与运算的浮点数的阶码大小，然后使他们的阶码(或移码)都一样，其最终目的是使参与运算的数的小数点位置对齐，以便得出正确的结果。
  既然需要使参与运算的浮点数的阶码一样，这时就要以某一个浮点数的阶码为标准了，通常以大阶码为准，即阶码小的数，要向阶码大的数对齐，也就是将小阶码的值调成与大阶码的值一样，这时，为了确保小阶码的数值与原来的保持一致，肯定需要对小阶码的尾数的小数点进行移位，就像十进制数中的$1.2345 * 10^3$，如果将指数3变成5，那么尾数1.2345的小数点位置不可能保持不变，要向左移两位，变成0.012345，所以，最终的形式为$0.012345 * 10^5$。
  因为是以大阶码为准，所以小阶码需要增大，那么尾数需要做相同倍数的减小，才能保持数值不变(即阶码小的尾数需要向右移对应的位数[小数点是向左移])，因为原则是，在尾数最高位前面加对应数量的0，然后原尾数最后的对应尾数直接丢弃，移了多少位就丢弃多少位(直接丢弃是因为存储尾数的存储器空间是固定的)。

  > 对阶中的尾数移位可以借助十进制的科学计数法来帮助理解，假设十进制数1234.5原来表示成$1.2345 * 10^3$，现在需要表示成10^6的格式，则对应的格式为$0.0012345 * 10^6$(在原来的小数点右边最高位前面移了三位，补加了两个0)，又假设小数点后面最多只能4位(浮点数中用来保存尾数的存储空间是固定的)，这样一来，最终的存储格式是$0.0012 * 10^6$，尾数中原来的"345"就被丢弃了，虽然这会在一定程度上导致与原值有区别，但还是在最大限度上保持了不变。

  示例：$X = 2^{0010} * 0.11000101$，$Y = 2^{0100} * 0.10101110$。对这两个浮点数进行加减法运算(尾数的符号位是体现在浮点数的符号位上，阶码和尾数均已采用补码形式表示)。
  X的阶码为0010，对应的十进制数位2，Y的阶码为0100，对应的十进制数为4，因此，X的阶码比Y的阶码小，此时应将X的阶码变成与Y的阶码一样，即都调成4，同时需要将X的尾数小数点左移2位(在小数点右边最高位前面加两个0，原来最低的2位被丢弃，这样才能使X的值与原来保持基本不变)。
  X原来的尾数为11000101，向右移两位(仍要保持原来的总位数不变，则要在前面补相应位数的0，原来最右边的对应两位丢弃)，则变成了00110001，这样一来，X的阶码也变成了0100，最终X在存储器中的格式是0 0100 00110001(原来为0 0010 11000101)。

- 2、**尾数运算**
  通过对阶将尾数的小数点对齐之后，就能将经过移位的尾数进行加减运算，这一步很简单，可以直接参照博文-"无符号二进制运算"。
  继续前面的示例：$X = 2^{0010} * 0.11000101$，$Y = 2^{0100} * 0.10101110$，现要求X+Y。
  X对阶后的尾数为00110001，Y的尾数不变，为10101110，两者相加：

```
1   X  00110001
2   Y  10101110
3 结果  11011111
```

- 3、**规格化处理**
  规格化处理主要是针对浮点数的尾数部分，规格化的尾数格式要求如下:
  尾数采用原码表示形式时：正数的规格化格式为：0.1XXX，负数的规格化形式为1.1XXX。
  尾数采用补码表示形式时，正数的规格化格式为：0.1XXX，负数的规格化形式为：1.0XXX。
  以上的最高位均代表符号位(0代表正数，1代表负数)，关键是看小数点后面的数的格式。

  > 对于双符号位(用两位来表示符号)的补码形式尾数，正数的规格化格式为00.1XXX，负数的规格化格式为：11.0XXX，以上最高两位代表符号位(00代表正数，11代表负数)，关键是看小数点后面数的格式。
  > 引入双符号位的设计目的就是为了能快速检测出运算结果是否有溢出，因为双符号位规定了"00代表正数，11代表负数"，如果最高两位不是这两种形式，而是"01"或者是"10"就能快速知道有溢出了，符号位为"01"的时候称为"上溢",即最高真值位相加后有进1，为"10"时，称为"下溢"，即最高真值位相减后有借1。
  > —引自王达老师《深入理解计算机网络》一书

> 对于双符号位(用两位来表示符号)的补码形式尾数，正数的规格化格式为00.1XXX，负数的规格化格式为：11.0XXX，以上最高两位代表符号位(00代表正数，11代表负数)，关键是看小数点后面数的格式。
> 引入双符号位的设计目的就是为了能快速检测出运算结果是否有溢出，因为双符号位规定了"00代表正数，11代表负数"，如果最高两位不是这两种形式，而是"01"或者是"10"就能快速知道有溢出了，符号位为"01"的时候称为"上溢",即最高真值位相加后有进1，为"10"时，称为"下溢"，即最高真值位相减后有借1。
> —引自王达老师《深入理解计算机网络》一书

本文内容仅涉及单符号位，有兴趣了解双符号位的，可以关注后续博文。
凡不符合以上格式要求的尾数均要进行规格化，对以上规格化格式进行总结可以得出：符号位与尾数最高位不一致才算是规格化，一致为非规格化。如1.0XXX，0.1XXX(最前面的为符号位)之类的尾数都是规格化的数，而1.1XXX，0.0XXX为非规格化的数，另外，如果尾数大余1，也是非规格化数。
对于非规格化的尾数需要进行相应的处理，处理方式又分为"左规"和"右规"，所谓"左规"就是尾数需要向左移位，每移1位阶码值减1，直到为规格化数为止，对应"右规"就是尾数要向右移位，每移1位阶码值减1，直到为规格化数为止。
采用"左规"还是"右规"的基本原则如下：
运算结果产生移益出(由于原来两尾数的最高有效位相加有进位或者相减有借位时形成的)时，必须进行"右规"，如双符号位的运算结果为10.XXX或01.XXX格式就是不符合规格化要求了，因为双符号位时符号位为"10"和"01"都是不正确的，右规时，最高有效位前补相应数量的0，此时，10.XXX格式右规后的格式为11.0XXX，而01.XXX格式右规后的格式为00.1XXX。
如果运算结果出现0.1XXX或者1.1XXX(即符号位与尾数最高有效位相同时)，必须进行左规，左规时最低有效位后补相应位数的0。
在上面的例子中，X与Y的尾数之和是11011111，又因为它是正数，所以可以表示成0.11011111，已经是规格化，不用再处理了。
如果X与Y相减，得出的尾数之差为负数，如1.1101111，这时就要进行规格化处理了，因为符号位1与尾数最高位1相同，这时需要进行左规，即向左移位，11011111要向左移两位才能使尾数的最高位与符号位不一致，移位后的尾数为01111100(最后两个0是补上去的)同时阶码要从原值0100相应减2，得到0010。

- 4、**舍入处理**
  在"对阶"和向左，向右规格化处理时，尾数要向左，向右移位，这样结果尾数与原先的尾数就会有一定的误差，因此要进行舍入处理，以尽可能减小这种误差。
  以下提供两种减小误差的方法：
  “0舍1入法”：
  类似十进制中的四舍五入法，即如果左规或右规时丢弃的是0，则舍去不计，反之要将尾数的末尾加1。
  如上面举例的X和Y的尾数之和为：1.11011111(最高位为符号位)，需要左规，得到1.01111100，由此可见左移时去掉的是前面的两位1，这种情况下就需要在尾数的最后一位加1，这样进行舍入处理的结果就是:1.01111101。
  “恒置1法”：
  只要有数位被左规或者右规丢弃掉，就要在尾数的末尾恒置1，但是这个方法精确度不高，因为从概率上来讲，丢弃0和1的概率各占50%。
  IEEE 754还有许多更复杂的舍入模式，有兴趣可以自行了解。

- 5、**溢出处理**

  如果采用双符号位补码形式，出现了"01"或者是"10"的形式就表示有溢出，但是，如果采用的是单符号位就不好判定了。这时需要根据对尾数进行规格化处理后的阶码是否超出了当前阶码所能表示的取值范围来判定了。
  上文已经提到了，右规后阶码需要加上对应的值，尾数每向右移一位就加1，最终可能导致阶码超出了当前类型的阶码所能表示的数值范围，这称为"上溢"。
  如(X+Y)补浮点数经过"对阶"和尾数相加后得到的尾数为10.101110011，显然它需要右规，结果为11.01110011，相当于尾数右移了一位，原来尾数的最高有效位"1"被丢弃了，同时阶码也需要相应加1，假设原先是用4位来保存阶码(算上阶码的符号位，所以，最大能表示的数是+7)，原阶码为0111，尾数右移一位后，阶码+1就变为8了，而这个是大于+7的，所以这是就需要进行溢出处理了。
  同时，尾数进行左规的时候，也有溢出的情况，因为左规后可能超出了用于存储阶码的存储位所能表示的最小值(称为"下溢")，如左规后得到的新阶码为-5，假设原先的阶码是用4为来存储的，现在要求尾数再左移3位，这样阶码也要减小3，得到-8，而-8也超出了4位阶码所能表示的最小值-7(阶码的最高位为符号位)。如果发现有溢出，计算机会根据以下原则进行处理：
  如果是上溢，立即停止运算，做中断处理。
  如果是下溢，整个尾数按0处理。

浮点数的加减法运算过程详解（面向小白的）_lisheng19870305 的专栏-CSDN博客

一. 浮点数在计算机内的表示

1.定点数：小数点固定在某个位置不动的数据。

有两种可能的表示方法（第一种为整数，第二种为带小数点的小数）：
第一种：直接举例： 1×××××××.（纯整数，小数点在最右边，即最低位右边）。
第二种：直接举例： 0. ×××××××（纯小数，小数点在符号位右边（此处0为符号位，是单符号位））。

2.浮点数：小数点位置可以浮动的数据。
常用下式表示：

N = M*RE

N为浮点数，M为尾数，E为阶码（个人认为是指数）,R为阶的基数（底），一般R为2或8或16。此式与十进制的科学计数法的计数方法类似，比如123500 = 1.235*105，10为底数，5为指数。

计算机内浮点数的表示常用下图：

Ms(符号位)	E（阶数）	M（尾数）
1位	n+1位（里边包括1位符号位）	m位

至于做题的时候好多符号位都是两位符号位而不是1位符号位，个人认为是便于发现数据有没有溢出（加减是否溢出的判定规则之一是运算完毕后结果的两个符号位不相同即为溢出）。所以做题时本人习惯即使给的是1位符号位，也要写成两位的。

还有一个要说的是1.01101表示的是 **-0.01101**，而不是 -1.01101。

浮点数加减运算之前要了解的东西就这么多了吧...开始说加减运算。

二. 浮点数的加减运算步骤

不上公式，不用字母表示，一点都不好理解。就用常规的数字表达式做例子说。

第一个例子： X = 00.1011×23 Y = 00.1001×24

浮点数的加减运算分为五步：

1.对阶： 对阶还是比较好理解的。把指数小的数（X）的指数（3）转化成和指数高的数（Y）的指数（4）相等，同时指数小的数（X）的尾数的符号位后边补两个数指数之差的绝对值个（1个）0。对于本例来说，就是把X变为：

X = 00.01011 ×24

此时有人会考虑，原数算上两位符号位是6位，转化之后成了7位，最低位的那个1要舍去啊。其实最好不要舍去不写，在它下面画个下划线标注一下这原本是要舍去的那个数即可（因为如果在这里随便舍掉数字不写，后边计算过程可能会出错）。
还有人分不清到底是写一位符号位还是两位符号位，其实写几位都不影响最后结果。个人喜欢用两位（就算题目里写一位也可以自己写成两位，按两位去做），原因说过了，不再赘述。

2.尾数相加减
按照例子来说，尾数相加减：
0 0 . 0 1 0 1 1
+
0 0 . 1 0 0 1 （**注意看是怎么对齐的**）
等于
0 0 . 1 1 1 0 1
这是相加，相减是把减数换成对应的补码再做相加运算即可。

3.规格化
规格化对初学者来说可能有点混乱，不过总结一下还是很明确的。
当出现以下两种情况时需要进行规格化。

①两个符号位不相同，右规：两个符号位不同，说明运算结果溢出。此时要进行右规，即把运算结果的尾数右移一位。需要右规的只有如下两种情况：**01××××和10××××。01×××右移一位的结果为001×××；10××××右移一位的结果为110×××。**最后将阶码（指数）+1。

②两个符号位相同，但是最高数值位与符号位相同，左规：两个符号位相同，说明没有溢出。此时要把尾数连续左移，直到最高数值位与符号位的数值不同为止。需要左规的有如下两种情况：**111×××和000×××。111×××左移一位的结果为11×××0；000×××左移一位的结果为00×××0。**最后将阶码（指数）减去移动的次数。

4.舍入

执行右规或者对阶时，有可能会在尾数低位上增加一些值，最后需要把它们移掉。（上面我说过，进行尾数加减时不要把对阶过程在尾数低位上增加的值去掉，不是不去掉，而是在舍入这一步去掉。）比如说，原来参与运算的两个数（加数和被加数）算上符号位一共有6个数，通过上边三个操作后运算结果变成了8个数，这时需要把第7和8位的数去掉。如果直接去掉，会使精度受影响，通常有下边两个方法：

①0舍1入法：

第一个例子：运算结果：X = 00.11010111，假设原本加数和被加数算上符号位一共有6个数，结果X是10个数，那么要去掉后四个数（0111）。由于0111首位是0（即**要去掉的数**的**最高位**为0），这种情况下，直接去掉这四个数就可以。该例最后结果为 X = 00.1101

第二个例子：运算结果 Y = 00.11001001，这时要去掉的数为1001四个数，由于这四个数的首位为1（即**要去掉的数**的**最高位**为1），这种情况下，直接去掉这四个数，再在去掉这四个数的新尾数的末尾加1。如果+1后又出现了溢出，继续进行右规操作。该例最后结果为 Y = 00.1101。

②置1法：

这个比较简单，去掉多余的尾数，然后保证去掉这四个数的新尾数的最后一位为1（即是1不用管，是0改成1）即可。比如 Z=00.11000111，置1法之后的结果为Z=00.11001。

5.检查阶码是否溢出：

阶码溢出在规格化和右移的过程中都有可能发生，若**阶码不溢出**，加减运算正常结束（即判断浮点数是否溢出，不需要判断尾数是否溢出，直接判断阶码是否溢出即可）。若阶码下溢，置运算结果为机器0（通常阶码和尾数全置0）。若上溢，置溢出标致。

好了。关于浮点数的加减我觉得就这些了，如果哪里有错误或者有什么不足，请指出，谢谢！