第二，如果$[x]_{补}$与$[y]_{补}$异号，商为负，则“够减”时上商“0”，“不够减”时上商“1”（按反码规则上商）。

结合比较规则与上商规则，便可得商值的确定方法，如表 6.23 所示。

表 6.23　商值的确定

$[x]_{补}$与$[y]_{补}$	商	$[R]_{补}$与$[y]_{补}$	商值
同号	正	同号，表示“够减”	1
		异号，表示“不够减”	0
异号	负	异号，表示“够减”	0
		同号，表示“不够减”	1

进一步简化，商值可直接由表 6.24 确定。

表 6.24　简化的商值确定

$[R]_{补}$与$[y]_{补}$	商值
同号	1
异号	0

② 在补码除法中，商符是在求商的过程中自动形成的。

在小数定点除法中，被除数的绝对值必须小于除数的绝对值，否则商大于 1 而溢出。因此，当$[x]_{补}$与$[y]_{补}$同号时，$[x]_{补}-[y]_{补}$所得的余数$[R_0]_{补}$必与$[y]_{补}$异号，上商“0”，恰好与商的符号（正）一致；当$[x]_{补}$与$[y]_{补}$异号时，$[x]_{补}+[y]_{补}$所得的余数$[R_0]_{补}$必与$[y]_{补}$同号，上商“1”，这也与商的符号（负）一致。可见，商符是在求商值过程中自动形成的。

此外，商的符号还可用来判断商是否溢出。例如，当$[x]_{补}$与$[y]_{补}$同号时，若$[R_0]_{补}$与$[y]_{补}$同号，上商“1”，即溢出。当$[x]_{补}$与$[y]_{补}$异号时，若$[R_0]_{补}$与$[y]_{补}$异号，上商“0”，即溢出。

当然，对于小数补码运算，商等于“-1”应该是允许的，但这需要特殊处理，为简化问题，这里不予考虑。

③ 新余数$[R_{i+1}]_{补}$的获得方法与原码加减交替法极相似，其算法规则如下：

当$[R_i]_{补}$与$[y]_{补}$同号时，商上“1”，新余数

$$[R_{i+1}]_{补}=2[R_i]_{补}-[y]_{补}=2[R_i]_{补}+[-y]_{补}$$

当$[R_i]_{补}$与$[y]_{补}$异号时，商上“0”，新余数

$$[R_{i+1}]_{补}=2[R_i]_{补}+[y]_{补}$$

将此算法列于表 6.25 中。

表 6.25　新余数的算法

$[R_i]_{补}$与$[y]_{补}$	商	新余数$[R_{i+1}]_{补}$
同号	1	$[R_{i+1}]_{补}=2[R_i]_{补}+[-y]_{补}$
异号	0	$[R_{i+1}]_{补}=2[R_i]_{补}+[y]_{补}$

如果对商的精度没有特殊要求，一般可采用“末位恒置 1”法，这种方法操作简单，易于实现，而且最大误差仅为 2^{-n}。

例 6.26　已知 $x=0.1001$，$y=0.1101$，求$\left[\frac{x}{y}\right]_{补}$。

解：由 $x=0.1001, y=0.1101$

得　$[x]_{补}=0.1001, [y]_{补}=0.1101, [-y]_{补}=1.0011$

其运算过程如表 6.26 所示。

表 6.26　例 6.26 的运算过程

被除数(余数)	商	说　明
0.1001	0.0000	
+1.0011		$[x]_{补}$与$[y]_{补}$同号，+$[-y]_{补}$
1.1100	0	$[R]_{补}$与$[y]_{补}$异号，上商“0”
1.1000	0	←1 位
+0.1101		+$[y]_{补}$
0.0101	01	$[R]_{补}$与$[y]_{补}$同号，上商“1”
0.1010	01	←1 位
+1.0011		+$[-y]_{补}$
1.1101	010	$[R]_{补}$与$[y]_{补}$异号，上商“0”
1.1010	010	←1 位
+0.1101		+$[y]_{补}$
0.0111	0101	$[R]_{补}$与$[y]_{补}$同号，上商“1”
0.1110	0101$\underline{1}$	←1 位，末位商恒置“1”

所以　$\left[\frac{x}{y}\right]_{补}=0.1011$

例 6.27　已知 $x=-0.1001$，$y=+0.1101$，求$\left[\frac{x}{y}\right]_{补}$。

解：由 $x=-0.1001, y=+0.1101$

得　$[x]_{补}=1.0111, [y]_{补}=0.1101, [-y]_{补}=1.0011$

其运算过程如表 6.27 所示。

表 6.27　例 6.27 的运算过程

被除数(余数)	商	说　明
1.0111	0.0000	
+0.1101		$[x]_{补}$与$[y]_{补}$异号,+$[y]_{补}$
0.0100	1	$[R]_{补}$与$[y]_{补}$同号,上商“1”
0.1000	1	←1 位
+1.0011		+$[-y]_{补}$
1.1011	10	$[R]_{补}$与$[y]_{补}$异号,上商“0”
1.0110	10	←1 位
+0.1101		+$[y]_{补}$
0.0011	101	$[R]_{补}$与$[y]_{补}$同号,上商“1”
0.0110	101	←1 位
+1.0011		+$[-y]_{补}$
1.1001	1010	$[R]_{补}$与$[y]_{补}$异号,上商“0”
1.0010	10101	←1 位,末位商恒置“1”

所以　$\left[\frac{x}{y}\right]_{补}=1.0101$

可见,n 位小数补码除法共上商 n+1 次(末位恒置“1”),第一次商可用来判断是否溢出。共移位 n 次,并用移位次数判断除法是否结束。

(2) 补码加减交替法所需的硬件配置

补码加减交替法所需的硬件配置基本上与图 6.11 相似,只是图 6.11 中的 S 触发器可以省掉,因为补码除法的商符在运算中自动形成。此外,在寄存器中存放的均为补码。

例 6.28　设 X、Y、Z 均为 n +1 位寄存器(n 为最低位),机器数采用 1 位符号位。若除法开始时操作数已放在合适的位置,试分别描述原码和补码除法商符的形成过程。

解:设 X、Y、Z 均为 n+1 位寄存器,除法开始时被除数在 X 中,除数在 Y 中,S 为触发器,存放商符,Z 寄存器存放商。原码除法的商符由两操作数(原码)的符号位进行异或运算获得,记作 $X_0 \oplus Y_0 \rightarrow S$。

补码除法的商符由第 1 次上商获得,共分两步。

第一步,若两操作数符号相同,则被除数减去除数(加上“求补”以后的除数),结果送 X 寄存器;若两操作数符号不同,则被除数加上除数,结果送 X 寄存器,记作

$$\overline{X_0 \oplus Y_0} \cdot (X+\overline{Y}+1)+(X_0 \oplus Y_0) \cdot (X+Y) \rightarrow X$$

第二步,根据结果的符号和除数的符号确定商值。若结果的符号 X_0 与除数的符号 Y_0 同号,则上商“1”,送至 Z_n 保存;若结果的符号 X_0 与除数的符号 Y_0 异号,则上商“0”,送至 Z_n 保存,记作

$$\overline{X_0 \oplus Y_0} \rightarrow Z_n$$

如果机器数采用补码，实现乘法和除法均用补码运算，那么，为了与补码乘法取得相同的寄存器位数，表 6.26 和表 6.27 中的被除数(余数)可取双符号位，整个运算过程与取 1 位符号位完全相同(见例 6.34 下的表 6.31)。

(3) 补码加减交替法的控制流程

补码加减交替法的控制流程如图 6.13 所示。

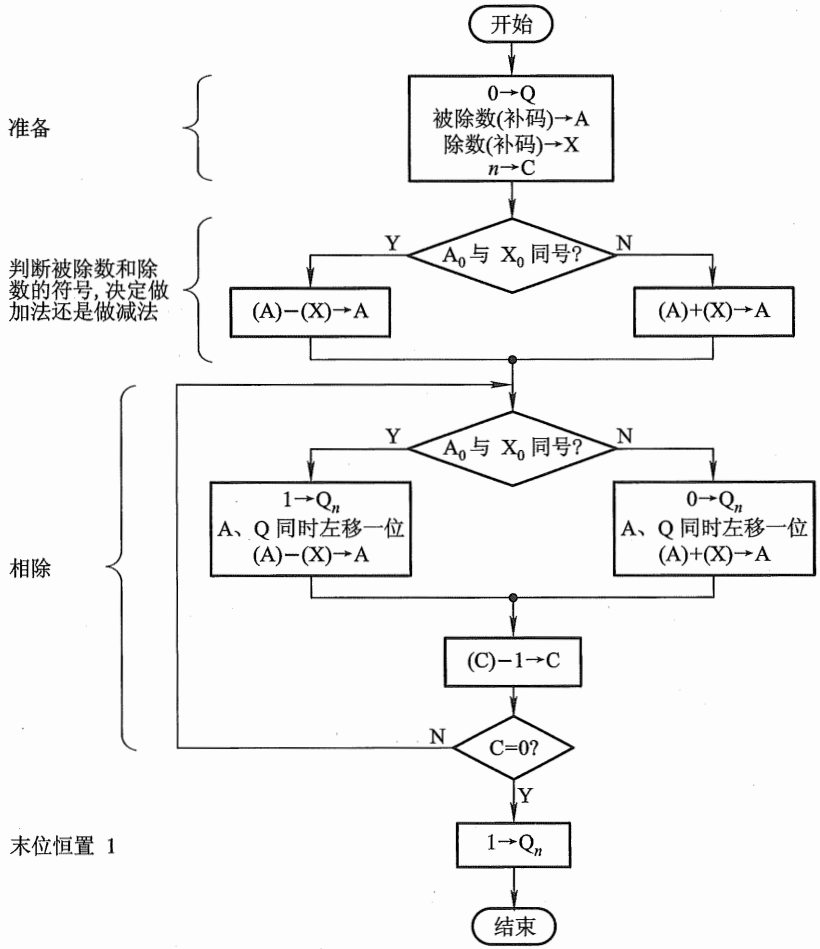

图 6.13　补码加减交替法控制流程

除法开始前，Q 寄存器被清零，准备接收商，被除数的补码在 A 中，除数的补码在 X 中，计数器 C 中存放除数的位数 n。除法开始后，首先根据两操作数的符号确定是做加法还是减法，加(或减)操作后，即上第一次商(商符)，然后 A 和 Q 同时左移一位，再根据商值的状态决定加或减除数，这样重复 n 次后，再上一次末位商“1”(恒置“1”法)，即得运算结果。

补充说明几点：

① 图中未画出补码除法溢出判断的内容。

② 按流程图所示，多做一次加（或减）法，其实在末位恒置“1”前，只需移位而不必做加（或减）法。

③ 与原码除法一样，图中均未指出对 0 进行检测。实际上在除法运算前，先检测被除数和除数是否为 0。若被除数为 0，结果即为 0；若除数为 0，结果为无穷大。这两种情况都无须继续做除法运算。

④ 为了节省时间，上商和移位操作可以同时进行。

以上介绍了计算机定点四则运算方法，根据这些运算规则，可以设计乘法器和除法器。有些机器的乘、除法可用编程来实现。分析上述运算方法对理解机器内部的操作过程和编制乘、除法运算的标准程序都是很有用的。

6.4　浮点四则运算

从 6.2 节浮点数的讨论可知，机器中任何一个浮点数都可写成

$$x = S_x \cdot r^{j_x}$$

的形式。其中，S_x 为浮点数的尾数，一般为绝对值小于 1 的规格化数（补码表示时允许为 -1），机器中可用原码或补码表示；j_x 为浮点数的阶码，一般为整数，机器中大多用补码或移码表示；r 为浮点数的基数，常用 2、4、8 或 16 表示。以下以基数为 2 进行讨论。

6.4.1　浮点加减运算

设两个浮点数

$$x = S_x \cdot r^{j_x}$$

$$y = S_y \cdot r^{j_y}$$

由于浮点数尾数的小数点均固定在第一数值位前，所以尾数的加减运算规则与定点数的完全相同。但由于其阶码的大小又直接反映尾数有效值小数点的实际位置，因此当两浮点数阶码不等时，因两尾数小数点的实际位置不一样，尾数部分无法直接进行加减运算。为此，浮点数加减运算必须按以下几步进行。

① 对阶，使两数的小数点位置对齐。

② 尾数求和，将对阶后的两尾数按定点加减运算规则求和（差）。

③ 规格化，为增加有效数字的位数，提高运算精度，必须将求和（差）后的尾数规格化。

④ 舍入，为提高精度，要考虑尾数右移时丢失的数值位。

⑤ 溢出判断，即判断结果是否溢出。

1. 对阶

对阶的目的是使两操作数的小数点位置对齐，即使两数的阶码相等。为此，首先要求出阶差，再按小阶向大阶看齐的原则，使阶小的尾数向右移位，每右移一位，阶码加 1，直到两数的阶码相等为止。右移的次数正好等于阶差。尾数右移时可能会发生数码丢失，影响精度。

例如，两浮点数 $x=0.1101\times2^{01}$，$y=(-0.1010)\times2^{11}$，求 $x+y$。

首先写出 x,y 在计算机中的补码表示。

$[x]_{补}=00,01;\ 00.1101$，$[y]_{补}=00,11;\ 11.0110$

在进行加法前，必须先对阶，故先求阶差：

$$[\triangle_j]_{补}=[j_x]_{补}-[j_y]_{补}=00,01+11,01=11,10$$

即 $\triangle_j=-2$，表示 x 的阶码比 y 的阶码小，再按小阶向大阶看齐的原则，将 x 的尾数右移两位，其阶码加 2，

得

$$[x]'_{补}=00,11;\ 00.0011$$

此时，$\triangle_j=0$，表示对阶完毕。

2. 尾数求和

将对阶后的两个尾数按定点加(减)运算规则进行运算。

如上例中的两数对阶后得

$$[x]'_{补}=00,11;\ 00.0011$$

$$[y]_{补}=00,11;\ 11.0110$$

则 $[S_x+S_y]_{补}$ 为

$$\begin{array}{rl} 0\ 0.0\ 0\ 1\ 1 & [S_x]'_{补} \\ +1\ 1.0\ 1\ 1\ 0 & [S_y]_{补} \\ \hline 1\ 1.1\ 0\ 0\ 1 & [S_x+S_y]'_{补} \end{array}$$

即

$$[x+y]_{补}=00,11;\ 11.1001$$

3. 规格化

由 6.2.2 节可知，当基值 $r=2$ 时，尾数 S 的规格化形式为

$$\frac{1}{2}\leqslant|S|<1 \tag{6.19}$$

如果采用双符号位的补码，则

当 $S>0$ 时，其补码规格化形式为

$$[S]_{补}=00.1\times\times\cdots\times \tag{6.20}$$

当 $S<0$ 时，其补码规格化形式为

$$[S]_{补}=11.0\times\times\cdots\times \tag{6.21}$$

可见，当尾数的最高数值位与符号位不同时，即为规格化形式，但对 $S<0$ 时，有两种情况需特殊处理。

① $S=-\frac{1}{2}$，则$[S]_{补}=11.100\cdots0$。此时对于真值$-\frac{1}{2}$而言，它满足式(6.19)，对于补码($[S]_{补}$)而言，它不满足于式(6.21)。为了便于硬件判断，特规定$-\frac{1}{2}$不是规格化的数(对补码而言)。

② $S=-1$，则$[S]_{补}=11.00\cdots0$，因小数补码允许表示-1，故-1视为规格化的数。

当尾数求和(差)结果不满足式(6.20)或式(6.21)时，则需规格化。规格化又分左规和右规两种。

(1) 左规

当尾数出现 00.0××…×或 11.1××…×时，需左规。左规时尾数左移一位，阶码减 1，直到符合式(6.20)或式(6.21)为止。

如上例求和结果为

$$[x+y]_{补}=00,11;\ 11.1001$$

尾数的第一数值位与符号位相同，需左规，即将其左移一位，同时阶码减 1，得

$$[x+y]_{补}=00,10;\ 11.0010$$

则

$$x+y=(-0.1110)\times2^{10}$$

(2) 右规

当尾数出现 01.××…×或 10.××…×时，表示尾数溢出，这在定点加减运算中是不允许的，但在浮点运算中这不算溢出，可通过右规处理。右规时尾数右移一位，阶码加 1。

例 6.29　已知两浮点数 $x=0.1101\times2^{10}$，$y=0.1011\times2^{01}$，求 $x+y$。

解：x、y 在机器中以补码表示为

$$[x]_{补}=00,10;\ 00.1101$$

$$[y]_{补}=00,01;\ 00.1011$$

① 对阶：

$$[\triangle_j]_{补}=[j_x]_{补}-[j_y]_{补}$$
$$=00,10+11,11=00,01$$

即 $\triangle_j=1$，表示 y 的阶码比 x 的阶码小 1，因此将 y 的尾数向右移一位，阶码相应加 1，即

$$[y]'_{补}=00,10;\ 00.0101$$

这时$[y]'_{补}$的阶码与$[x]_{补}$的阶码相等，阶差为 0，表示对阶完毕。

② 求和：

$$\begin{array}{rl} 0\ 0.1\ 1\ 0\ 1 & [S_x]_{补} \\ +0\ 0.0\ 1\ 0\ 1 & [S_y]'_{补} \\ \hline 0\ 1.0\ 0\ 1\ 0 & [S_x+S_y]'_{补} \end{array}$$

即

$$[x+y]_{补}=00,10;\ 01.0010$$

③ 右规：

运算结果两符号位不等，表示尾数之和绝对值大于 1，需右规，即将尾数之和向右移一位，阶码加 1，故得

$$[x+y]_{补}=00,11;\ 00.1001$$

则

$$x+y=0.1001\times 2^{11}$$

4. 舍入

在对阶和右规的过程中，可能会将尾数的低位丢失，引起误差，影响精度。为此可用舍入法来提高尾数的精度。常用的舍入方法有以下两种。

(1) “0 舍 1 入”法

“0 舍 1 入”法类似于十进制数运算中的“四舍五入”法，即在尾数右移时，被移去的最高数值位为 0，则舍去；被移去的最高数值位为 1，则在尾数的末位加 1。这样做可能使尾数又溢出，此时需再做一次右规。

(2) “恒置 1”法

尾数右移时，不论丢掉的最高数值位是“1”或“0”，都使右移后的尾数末位恒置“1”。这种方法同样有使尾数变大和变小的两种可能。

综上所述，浮点加减运算经过对阶、尾数求和、规格化和舍入等步骤。与定点加减运算相比，显然要复杂得多。

例 6.30　设 $x=2^{-101}\times(-0.101000)$，$y=2^{-100}\times(+0.111011)$，并假设阶符取 2 位，阶码的数值部分取 3 位，数符取 2 位，尾数的数值部分取 6 位，求 $x-y$。

解：由　　$x=2^{-101}\times(-0.101000)$，　$y=2^{-100}\times(+0.111011)$

得　　$[x]_{补}=11,011;\ 11.011000$，　$[y]_{补}=11,100;\ 00.111011$

① 对阶：

$$[\triangle_j]_{补}=[j_x]_{补}-[j_y]_{补}=11,011+00,100=11,111$$

即 $\triangle_j=-1$，则 x 的尾数向右移一位，阶码相应加 1，即

$$[x]'_{补}=11,100;\ 11.101100$$

② 求和：

$$\begin{aligned}[S_x]'_{补}-[S_y]'_{补}&=[S_x]'_{补}+[-S_y]_{补}\\&=11.101100+11.000101\\&=10.110001\end{aligned}$$

即　　$[x-y]_{补}=11,100;\ 10.110001$

尾数符号位出现“10”，需右规。

③ 规格化：

右规后得　$[x-y]_{补}=11,101;\ 11.011000\ \boxed{1}$

④ 舍入处理：

采用“0 舍 1 入”法，其尾数右规时末位丢 1，则有

$$\begin{array}{r} 1\ 1.0\ 1\ 1\ 0\ 0\ 0 \\ +\qquad\qquad\quad 1 \\ \hline 1\ 1.0\ 1\ 1\ 0\ 0\ 1 \end{array}$$

所以　　$[x-y]_{补}=11,101;\ 11.011001$

5. 溢出判断

与定点加减法一样，浮点加减运算最后一步也需判断溢出。在浮点规格化中已指出，当尾数之和(差)出现 01.××…×或10.××…×时，并不表示溢出，只有将此数右规后，再根据阶码来判断浮点运算结果是否溢出。

若机器数为补码，尾数为规格化形式，并假设阶符取 2 位，阶码的数值部分取 7 位，数符取 2 位，尾数的数值部分取 n 位，则它们能表示的补码在数轴上的表示范围如图 6.14 所示。

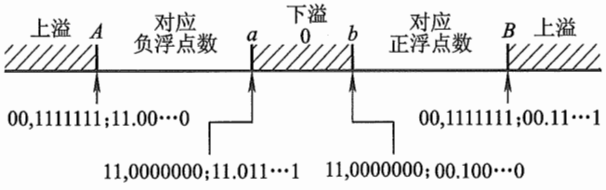

图 6.14　补码在数轴上的表示

图中 A、B、a、b 的坐标均为补码表示，分别对应最小负数、最大正数、最大负数和最小正数。它们所对应的真值如下：

A 最小负数　　$2^{+127}\times(-1)$

B 最大正数　　$2^{+127}\times(1-2^{-n})$

a 最大负数　　$2^{-128}\times(-2^{-1}-2^{-n})$

b 最小正数　　$2^{-128}\times 2^{-1}$

注意，由于图 6.14 所示的 A、B、a、b 均为补码规格化的形式，故其对应的真值与图 6.2 所示的结果有所不同。

在图 6.14 中 a、b 之间的阴影部分对应的阶码小于−128，这种情况称为浮点数的下溢。下溢时，浮点数值趋于零，故机器不做溢出处理，仅把它作为机器零。

在图 6.14 中 A、B 两侧的阴影部分对应的阶码大于+127，这种情况称为浮点数的上溢。此刻，浮点数真正溢出，机器需停止运算，做溢出中断处理。一般说浮点溢出，均是指上溢。

可见，浮点机的溢出与否可由阶码的符号决定，即

阶码$[j]_{补}=01$，××…×为上溢。

阶码$[j]_{补}=10$，××…×为下溢，按机器零处理。

当阶符为“01”时，需做溢出处理。

例 6.30 经舍入处理后得$[x-y]_{补}=11,101;\ 11.011001$，阶符为“11”，不溢出，故最终结果为

$$x-y=2^{-011}\times(-0.100111)$$

例 6.31　设机器数字长 16 位，阶码 5 位（含 1 位阶符），基值为 2，尾数 11 位（含 1 位数符）。对于两个阶码相等的数按补码浮点加法完成后，由于规格化操作可能出现的最大误差的绝对值是多少？

解：两个阶码相等的数按补码浮点加法完成后，仅当尾数溢出需右规时会引起误差。右规时尾数右移一位，阶码加 1，可能出现的最大误差是末尾丢 1，例如：

结果为　　00,1110;01.××××××××××1

右规后得　　00,1111;00.1××××××××××1

考虑到最大阶码是 15，最后得最大误差的绝对值为$(10000)_{\text{补}}=2^4$。

当计算机中阶码用移码表示时，移码运算规则参见浮点乘除运算。

最后可得浮点加减运算的流程。

例 6.32　要求用最少的位数设计一个浮点数格式，必须满足下列要求。

（1）十进制数的范围：负数 -10^{38} ~ -10^{-38}；正数 $+10^{-38}$ ~ 10^{38}。

（2）精度：7 位十进制数据。

解：（1）由　$2^{10}>10^3$

可得　　$(2^{10})^{12}>(10^3)^{12}$，即 $2^{120}>10^{36}$

又因为　　$2^7>10^2$

所以　　$2^7\times2^{120}>10^2\times10^{36}$，即 $2^{127}>10^{38}$

同理　　$2^{-127}<10^{-38}$

故阶码取 8 位（含 1 位阶符），当其用补码表示时，对应的数值范围为 −128 ~ +127。

（2）因为 $10^7\approx2^{23}$，故尾数的数值部分可取 23 位。加上数符，最终浮点数取 32 位，其中阶码 8 位（含 1 位阶符），尾数 24 位（含 1 位数符）。

6. 浮点加减运算流程

图 6.15 为浮点补码加减运算的流程图。

6.4.2　浮点乘除法运算

两个浮点数相乘，乘积的阶码应为相乘两数的阶码之和，乘积的尾数应为相乘两数的尾数之积。两个浮点数相除，商的阶码为被除数的阶码减去除数的阶码，尾数为被除数的尾数除以除数的尾数所得的商，可用下式描述。

设两浮点数

$$x=S_x\cdot r^{j_x}$$

$$y=S_y\cdot r^{j_y}$$

则

$$x\cdot y=(S_x\cdot S_y)\times r^{j_x+j_y}$$

$$\frac{x}{y}=\frac{S_x}{S_y}\cdot r^{j_x-j_y}$$

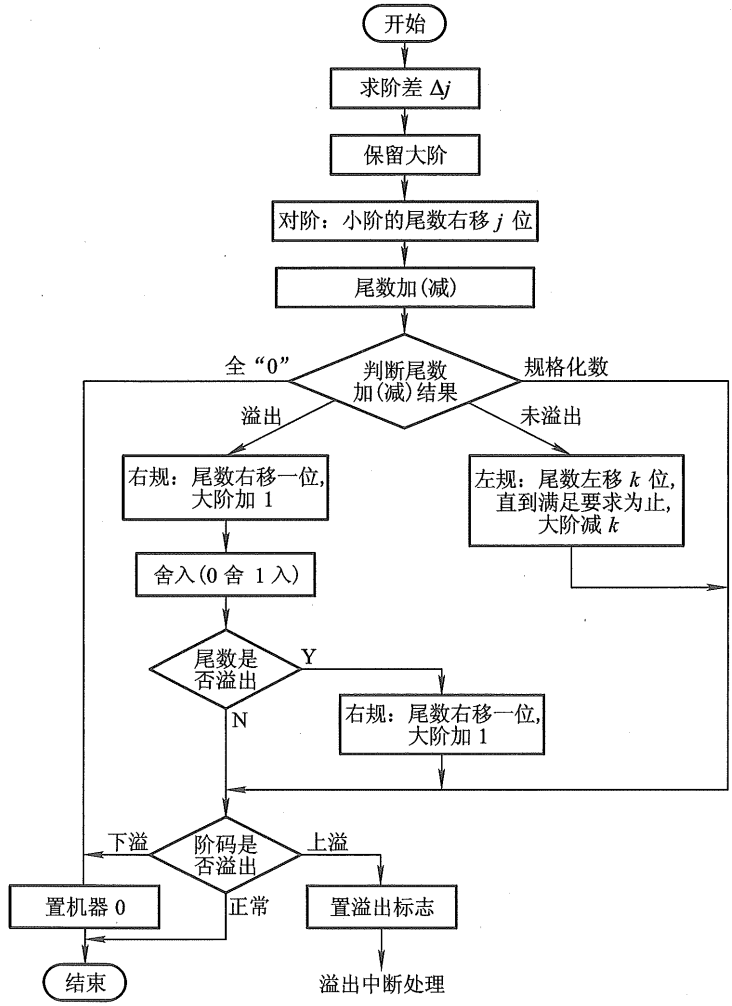

图 6.15　浮点补码加减运算流程

在运算中也要考虑规格化和舍入问题。

1. 阶码运算

若阶码用补码运算，乘积的阶码为$[j_x]_{补}+[j_y]_{补}$，商的阶码为$[j_x]_{补}-[j_y]_{补}$。两个同号的阶码相加或异号的阶码相减可能产生溢出，此时应做溢出判断。

若阶码用移码运算，则

因为　　$[j_x]_{移}=2^n+j_x \quad -2^n\leqslant j_x<2^n$　（n 为整数的位数）

　　$[j_y]_{移}=2^n+j_y \quad -2^n\leqslant j_y<2^n$　（n 为整数的位数）

所以　　$[j_x]_{移}+[j_y]_{移}=2^n+j_x+2^n+j_y$

$$=2^n+[2^n+(j_x+j_y)]$$
$$=2^n+[j_x+j_y]_{移}$$

可见，直接用移码求阶码和时，最高位多加了一个 2^n，要得到移码形式的结果，必须减去 2^n。

由于同一个真值的移码和补码数值部分完全相同，而符号位正好相反，即

$$[j_y]_{补}=2^{n+1}+j_y \quad (\mathrm{mod}\ 2^{n+1})$$

因此，求阶码和可用下式完成

$$[j_x]_{移}+[j_y]_{补}=2^n+j_x+2^{n+1}+j_y$$
$$=2^{n+1}+[2^n+(j_x+j_y)]$$
$$=[j_x+j_y]_{移}(\mathrm{mod}\ 2^{n+1})$$

则直接可得移码形式。

同理，当做除法运算时，商的阶码可用下式完成

$$[j_x]_{移}+[-j_y]_{补}=[j_x-j_y]_{移}$$

可见进行移码加减运算时，只需将移码表示的加数或减数的符号位取反（即变为补码），然后进行运算，就可得阶和（或阶差）的移码。

阶码采用移码表示后又如何判断溢出呢？如果在原有移码符号位的前面（即高位）再增加1位符号位，并规定该位恒用“0”表示，便能方便地进行溢出判断。溢出的条件是运算结果移码的最高符号位为1。此时若低位符号位为0，表示上溢；低位符号位为1，表示下溢。如果运算结果移码的最高符号位为0，即表明没有溢出。此时若低位符号位为1，表明结果为正；低位符号位为0，表示结果为负。

例如，若阶码取3位（不含符号位），则对应的真值范围是-8～+7。

当 $j_x=+101, j_y=+100$ 时，则有

$[j_x]_{移}=01,101$；$[j_y]_{补}=00,100$

故　$[j_x+j_y]_{移}=[j_x]_{移}+[j_y]_{补}=01,101+00,100=10,001$　　结果上溢

$[j_x-j_y]_{移}=[j_x]_{移}+[-j_y]_{补}=01,101+11,100=01,001$　　结果为+1

当　$j_x=-101, j_y=-100$ 时，则有

$[j_x]_{移}=00,011$，$[j_y]_{补}=11,100$

故　$[j_x+j_y]_{移}=[j_x]_{移}+[j_y]_{补}=00,011+11,100=11,111$　　结果下溢

$[j_x-j_y]_{移}=[j_x]_{移}+[-j_y]_{补}=00,011+00,100=00,111$　　结果为-1

2. 尾数运算

（1）浮点乘法尾数运算

两个浮点数的尾数相乘，可按下列步骤进行。

① 检测两个尾数中是否有一个为0，若有一个为0，乘积必为0，不再做其他操作；如果两尾数均不为0，则可进行乘法运算。

② 两个浮点数的尾数相乘可以采用定点小数的任何一种乘法运算来完成。相乘结果可能

要进行左规，左规时调整阶码后如果发生阶下溢，则作机器零处理；如果发生阶上溢，则作溢出处理。此外，尾数相乘会得到一个双倍字长的结果，若限定只取 1 倍字长，则乘积的若干低位将会丢失。如何处理丢失的各位值，通常有两种方法。

其一，无条件地丢掉正常尾数最低位之后的全部数值，这种方法称为截断处理，处理简单，但影响精度。

其二，按浮点加减运算讨论的两种舍入原则进行舍入处理。对于原码，采用 0 舍 1 入法时，不论其值是正数或负数，“舍”使数的绝对值变小，“入”使数的绝对值变大。对于补码，采用 0 舍 1 入法时，若丢失的位不是全 0，对正数来说，“舍”“入”的结果与原码分析正好相同；对负数来说，“舍”“入”的结果与原码分析正好相反，即“舍”使绝对值变大，“入”使绝对值变小。为了使原码、补码舍入处理后的结果相同，对负数的补码可采用如下规则进行舍入处理。

① 当丢失的各位均为 0 时，不必舍入。

② 当丢失的各位数中的最高位为 0 时，且以下各位不全为 0，或丢失的各位数中的最高位为 1，且以下各位均为 0 时，则舍去被丢失的各位。

③ 当丢失的各位数中的最高位为 1，且以下各位又不全为 0 时，则在保留尾数的最末位加 1 修正。

例如，对下列 4 个补码进行只保留小数点后 4 位有效数字的舍入操作，如表 6.28 所示。

表 6.28　补码舍入操作实例

$[x]_{补}$ 舍入前	舍入后	对应真值 x
1.01110000	1.0111(不舍不入)	-0.1001
1.01111000	1.0111(舍)	-0.1001
1.01110101	1.0111(舍)	-0.1001
1.01111100	1.1000(入)	-0.1000

如果将上述 4 个补码变成原码后再舍入，其结果列于表 6.29 中。

表 6.29　原码舍入操作实例

$[x]_{原}$ 舍入前	舍入后	对应真值 x
1.10010000	1.1001(不舍不入)	-0.1001
1.10001000	1.1001(入)	-0.1001
1.10001011	1.1001(入)	-0.1001
1.10000100	1.1000(舍)	-0.1000

比较表 6.28 和表 6.29 可见，按照上述的约定对负数的补码进行舍入处理，与对其原码进行舍入处理后的真值是一样的。

下面举例说明浮点乘法运算的全过程。

设机器数阶码取 3 位(不含阶符),尾数取 7 位(不含数符),要求阶码用移码运算,尾数用补码运算,最后结果保留 1 倍字长。

例 6.33　已知 $x=2^{-101}\times0.0110011$,$y=2^{011}\times(-0.1110010)$,求 $x\cdot y$。

解:由 $x=2^{-101}\times0.0110011$,$y=2^{011}\times(-0.1110010)$

得

$$[x]_{补}=11,011;\ 00.0110011$$

$$[y]_{补}=00,011;\ 11.0001110$$

① 阶码运算:

$$[j_x]_{移}=00,011,[j_y]_{补}=00,011$$

$$[j_x+j_y]_{移}=[j_x]_{移}+[j_y]_{补}$$

$$=00,011+00,011$$

$$=00,110\ 对应真值-2$$

② 尾数相乘(采用 Booth 算法):

其过程如表 6.30 所示。

表 6.30　例 6.33 尾数相乘过程

部分积	乘　数	y_{n+1}	说　明
00.0000000	1.0001110	0	
00.0000000	0100011 1	0	→1 位
+ 11.1001101			$+[-S_x]_{补}$
11.1001101	0		
11.1100110	1010001 1	1	→1 位
11.1110011	0101000 1	1	→1 位
11.1111001	1010100 0	1	→1 位
+ 00.0110011			$+[S_x]_{补}$
00.0101100	1·010		
00.0010110	0101010 0	0	→1 位
00.0001011	0010101 0	0	→1 位
00.0000101	1001010 1	0	→1 位
+ 11.1001101			$+[-S_x]_{补}$
11.1010010	1001010		

③ 规格化:

尾数相乘结果为$[S_x \cdot S_y]_{补}=11.10100101001010$,需左规,即

$$[x \cdot y]_{补}=11,110;\ 11.10100101001010$$

左规后　　$[x \cdot y]_{补}=11,101;\ 11.01001010010100$

④ 舍入处理:

尾数为负,按负数补码的舍入规则,取 1 倍字长,丢失的 7 位为 0010100,应“舍”,故最终结果为

$$[x \cdot y]_{补}=11,101;\ 11.0100101$$

$$x \cdot y=2^{-011}\times(-0.1011011)$$

(2) 浮点除法尾数运算

两个浮点数的尾数相除,可按下列步骤进行。

① 检测被除数是否为 0,若为 0,则商为 0;再检测除数是否为 0,若为 0,则商为无穷大,另作处理。若两数均不为 0,则可进行除法运算。

② 两浮点数尾数相除同样可采取定点小数的任何一种除法运算来完成。对已规格化的尾数,为了防止除法结果溢出,可先比较被除数和除数的绝对值,如果被除数的绝对值大于除数的绝对值,则先将被除数右移一位,其阶码加 1,再作尾数相除。此时所得结果必然是规格化的定点小数。

例 6.34　按补码浮点运算步骤,计算$\left[2^5\times\left(+\frac{9}{16}\right)\right]\div\left[2^3\times\left(-\frac{13}{16}\right)\right]$。

解:令　$x=\left[2^5\times\left(+\frac{9}{16}\right)\right]=2^{101}\times(0.1001)$

$$y=\left[2^3\times\left(-\frac{13}{16}\right)\right]=2^{011}\times(-0.1101)$$

所以　$[x]_{补}=00,101;\ 00.1001$

$[y]_{补}=00,011;\ 11.0011$,$[-S_y]_{补}=00.1101$

① 阶码相减:

$$[j_x]_{补}-[j_y]_{补}=00,101-00,011=00,101+11,101=00,010$$

② 尾数相除(采用补码除法):

其过程如表 6.31 所示。表中被除数(余数)采用双符号位,与采用一位符号位结果一致。

表 6.31　例 6.34 尾数相除过程

被除数(余数)	商	说　明
00.1001	0.0000	
+ 11.0011		$[S_x]_{补}$与$[S_y]_{补}$异号,+$[S_y]_{补}$
11.1100	1	$[R]_{补}$与$[S_y]_{补}$同号,上商“1”
11.1000	1	←1 位
+ 00.1101		+$[-S_y]_{补}$