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。

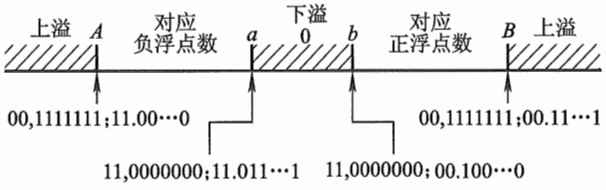

图 6.14　补码在数轴上的表示

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

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

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