2.9 浮点运算

2.9.1 浮点数的表示

定点表示法的缺点：数的表示范围小，精度差（对于很小的数，前导 0 多）。

对于十进制数，人们常使用科学计数法（Scientific Notation）。

例：

18800000000 可表示成 1.88×10^{10}；

0.000000000188 可表示成 1.88×10^{-10}。

这种类型的表示方法称为“浮点数(floating point number)表示。

对于二进制数：

$$\pm M\times R^{\pm E}$$

这样的数存储在一个二进制字的三个字段中：

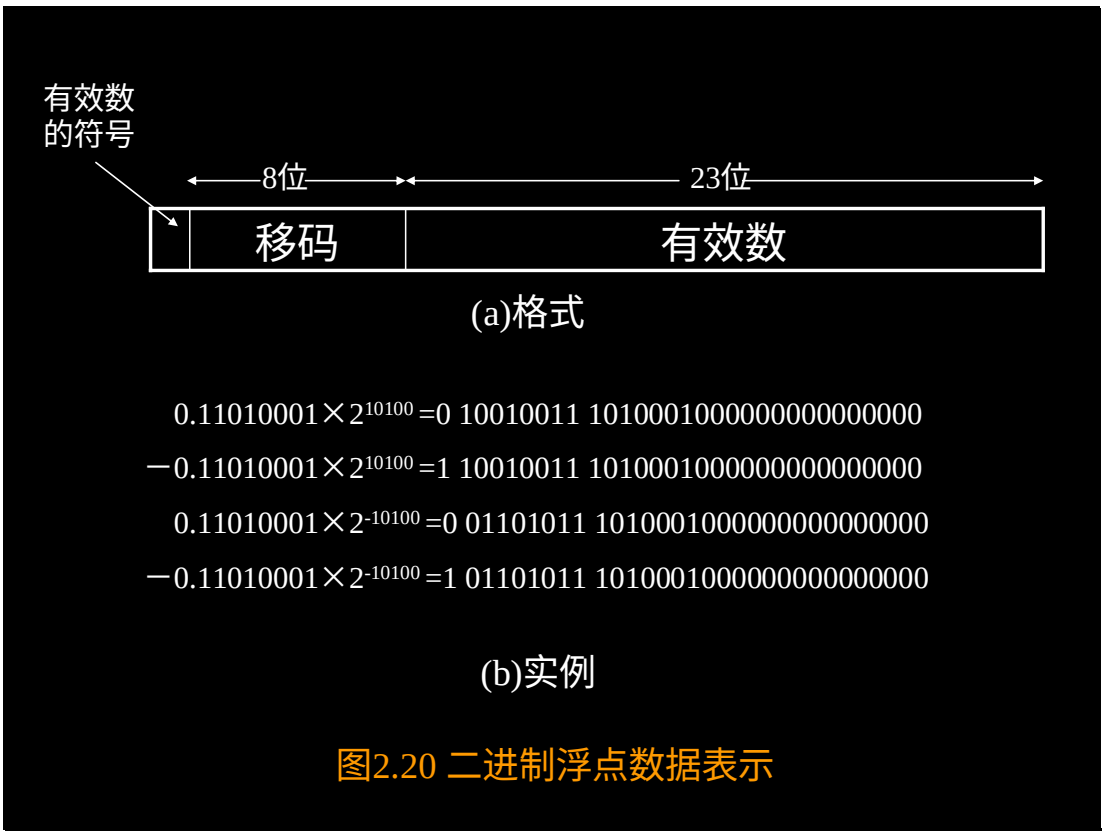
1.符号（Sign）：正或负

2.有效数 M（Significand）

3.指数 E（Exponent）

基值 R 是隐含的并且不需存储。

例：二进制数的一种表示方法



(a)格式

$0.11010001\times 2^{10100}=0\ 10010011\ 10100010000000000000$

$-0.11010001\times 2^{10100}=1\ 10010011\ 10100010000000000000$

$0.11010001\times 2^{-10100}=0\ 01101011\ 10100010000000000000$

$-0.11010001\times 2^{-10100}=1\ 01101011\ 10100010000000000000$

(b)实例

图2.20 二进制浮点数据表示

最左位保存数的符号（0=正，1=负）。指数值存于第 1 位到第 8 位，存放指数字的字段称为阶码字段（或指数字段）。字的最后一部分是有效数字段。

通常，浮点真值具有以下的形式：

$$(-1)^{s}\times m\times 2^{e} \qquad (2.63)$$

S：是符号（二进制）；

m：有效数字段代表的真值；

e：是阶码（指数）字段代表的真值。

2.9.2 移码的运算

1．移码和补码之间的关系式

$$[x]_{移}=[x]_{补}+B \quad (\mathrm{mod}\ M) \qquad (2.65)$$

证：

由移码定义：

$$[x]_{移}=x+B$$

由补码定义：

$$[x]_{补}=M+x \quad (\mathrm{mod}\ M)$$

$$[-x]_{补}=M-x \quad (\mathrm{mod}\ M)$$

于是：

$$[x]_{移}+[-x]_{补}=M+B \quad (\mathrm{mod}\ M)$$

通常$B>0$（如IEEE754标准），于是有

$$[x]_{移}+[-x]_{补}=B \quad (\mathrm{mod}\ M)$$

$$[x]_{移}=B-[-x]_{补} \quad (\mathrm{mod}\ M)$$

利用$-[-x]_{补}=[x]_{补}$ $(\mathrm{mod}\ M)$，所以有如下的移码与补码的关系式：

$$[x]_{移}=[x]_{补}+B \quad (\mathrm{mod}\ M)$$

证毕

2．真值和的移码用真值的移码表示

真值和的移码可以用真值移码的和表示成下面的公式。

$$[x+y]_{移}=[x]_{移}+[y]_{移}-B \quad (\mathrm{mod}\ M) \qquad (2.66)$$

$$[x+y]_{移}=[x]_{移}+[y]_{移}+[-B]_{补} \quad (\mathrm{mod}\ M) \qquad (2.67)$$

证：

根据（2.65）式有

$$[x]_{移}=[x]_{补}+B \quad (\mathrm{mod}\ M)$$

$$[y]_{移}=[y]_{补}+B \quad (\mathrm{mod}\ M)$$

于是

$$[x]_{移}+[y]_{移}=([x]_{补}+[y]_{补}+B)+B \quad (\mathrm{mod}\ M)$$

由于$[x+y]_{补}=[x]_{补}+[y]_{补}$ （mod M），于是有

$$[x]_{移}+[y]_{移}=([x+y]_{补}+B)+B \quad (\mathrm{mod}\ M)$$

另一方面，由（2.65）式直接得

$$[x+y]_{移}=[x+y]_{补}+B \quad (\mathrm{mod}\ M)$$

于是

$$[x]_{移}+[y]_{移}=[x+y]_{移}+B \quad (\mathrm{mod}\ M)$$

即

$$[x+y]_{移}=[x]_{移}+[y]_{移}-B \quad (\mathrm{mod}\ M)$$

利用补码定义$[-B]_{补}=M-B$（mod M），将上式右边加M得

$$[x+y]_{移}=[x]_{移}+[y]_{移}+[-B]_{补} \quad (\mathrm{mod}\ M) \qquad (2.67)$$

证毕

3．真值和的移码用真值的补码表示

$$[x+y]_{移}=[x]_{补}+[y]_{补}+[B]_{补} \quad (\mathrm{mod}\ M) \qquad (2.69)$$

证：

由（2.65）式以及补码加法公式有：

$$[x+y]_{移}=[x+y]_{补}+B \quad (\text{mod } M)$$

$$[x+y]_{移}=[x]_{补}+[y]_{补}+[B]_{补} \quad (\text{mod } M)$$

证毕

4．真值差的移码用真值的移码表示

$$[x-y]_{移}=[x]_{移}-[y]_{移}+[B]_{补} \quad (\text{mod } M) \qquad (2.68)$$

证明略。

5．真值差的移码用真值的补码表示

$$[x-y]_{移}=[x]_{补}+[-y]_{补}+[B]_{补} \quad (\text{mod } M) \qquad (2.70)$$

证明略。

阶码用移码表示的优点是：真值大的，其移码形式的机器数也大，便于比较浮点数的大小。

2.9.3 二进制浮点数表示的 IEEE 754 标准

最重要的浮点表示法定义在 IEEE 754 标准中。开发这个标准是为了便于程序从一类处理器移植到另一类处理器上的可移植性，这个标准获得广泛的认可，并已用于当代所有各类处理器和算术协处理器中。

IEEE 754 浮点数格式

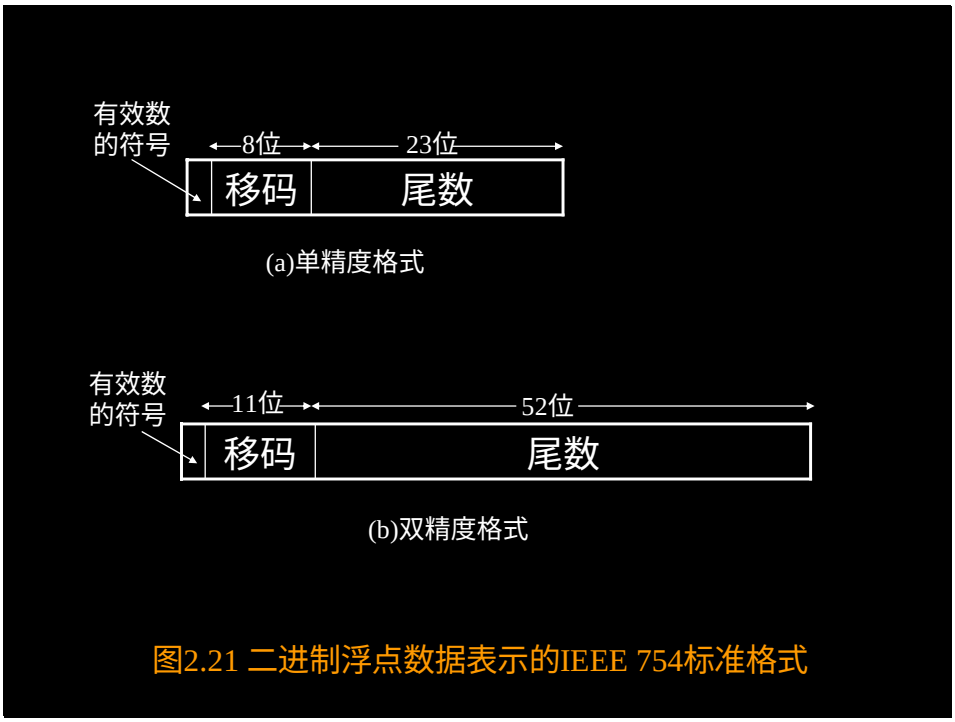


图2.21 二进制浮点数据表示的IEEE 754标准格式

IEEE 754 标准定义了 32 位的单精度和 64 位的双精度两种格式（图 2.21），它们的阶码字段分别是 8 位和 11 位，隐含的基值是 2。另外，标准还定义单精度和双精度的两种扩展格式。

下面以单精度数为例来介绍 IEEE 754 标准，其概念也适用于双精度浮点数。

为了简化浮点数的操作，一般需要对它们进行规格化（Normalized）。一个规格化的数具有如下形式：$\pm 1.bb\cdots b \times 2^{\pm E}$。这里的 b 是二进制数字 0 或 1。

在规格化的二进制浮点数中，小数点前

面的一位数总是 1，故可将这个 1 省略，不出现在 32 位浮点数格式中，我们称其为隐藏位(hidden bit)。在 IEEE754 标准中，为了在有效字段中能表示更多的数据，不存储这个 1。也就是说在单精度表示中，有效数实际上是 24 位（1 位隐含位+23 位小数位）。这种表示方法的特点是：符号总是存于字的最左边的位（第 1 位）；有效数的第 1 位总是 1，并且不需要存于有效数字段中。如果把有效数记为 M，则 $M=1+F$，F 是尾数。因此，在 IEEE754 标准中，有效数字段实际上存储的是尾数。以后这一字段也称为尾数字段。

IEEE754 浮点数由 3 部分组成：符号(sign)、阶码(exponent)和尾数。单精度的格式为 32 位，如图 2.21（a）所示。

约定：用大写字母表示 IEEE 754 浮点数字段中的二进制代码，而相应的小写字母表示对应的真值。

阶码：E 用移码表示，$E=e+B$，见 (2.64) 式。

尾数：有效数 M 的小数部分。

有效数记为 M，$M=1+F$；F 是尾数，是有效数的小数部分（F 的小数点应位于有效数字段最高位的前面，这与前面介绍过的定点小数是不同的），23 位长，

符号：用 S 代表一位符号位，0 表示正数，l 表示负数。

IEEE 754 浮点数的真值记为 v；IEEE 754 单精度浮点数标准采用偏移量为 $B=127$。

为了表示∞和一些特殊的数据，阶码 E（即移码）的最小值 0 和最大值 255（十进制）做特殊用途。

正常移码的最小值：$E=1$，对应的真值（十进制）为

$$e=E-B=1-127=-126。$$

正常移码的最大值： $E=254$（十进制），对应的真值（十进制）为

$$e=E-B=254-127=+127。$$

IEEE 754 单精度浮点数的阶码真值的取值范围为-126～+127（十进制）。

IEEE 754 单精度浮点数规格化(normalized)的数值 v 可以表示为：

$$v=(-1)^S\times 2^{E-127}\times(1+F) \qquad (2.71)$$

或：

$$v=(-1)^S\times 2^{E-127}\times M \qquad (2.71a)$$

数的分类有如下表示：

对于阶码（移码）值范围在 1~254（单精度，真值的范围是-126~+127）表示了一个规格化的非零浮点数。

除了以上的规格化数之外，IEEE 754 还规定了以下 4 种特殊情况：

（1）+0，-0：如果$E=0$，且$F=0$，则定义$v=(-1)^S\times 0=0$。此时的机器0为阶码和尾数均为0的形式，给硬件的判0带来很大方便。

（2）非规格化数(denormalized) DNRM：如果 $E=0$，但 $F\neq 0$，则定义 $v=$DNRM。全 0 阶码与非 0 尾数一起表示一个非规格化数。由于 0 的最高有效数值位不会是 1，阶码全 0（$E=0$）是一种保留值，使得这种情况下硬件不会在尾数字段的最高位加上一个 1。此时二进制小数点左边的隐藏位可以理解为是 0 并且指数真值是-126 或-1022。数的正负取决于它的符号位。

（3）+∞，-∞，正负无穷大：若 $E=255$，且 $F=0$，则定义 $v=(-1)^S\infty$，全 1 阶码与全 0 尾数一起表示正无穷大或负无穷大，取决于它的符号位。无穷大表示式有时是有用的。把（上）溢出看成一个错误条件而停止程序执行，还是把它看成是一个∞值带入程序并继续处理，这样的裁决权留给用户。

（4）NaN，非数(not a number)：若 $E=255$，且 $F\neq0$，则定义 $v=$NaN。全 1 阶码与非 0 尾数一起给出 NaN 值，它意味着不是一个数（Not a number）。非数（NaN）用来通知各种异常条件。

绝对值最小的规格化浮点数的浮点数形式是：

S 00000001 00000000000000000000000

即：

$$v=(-1)^S\times2^{-126}\times(1+0.00000000000000000000000)$$

绝对值最大的规格化浮点数的浮点数形是：

S 11111110 11111111111111111111111

即：

$$v=(-1)^S\times2^{127}\times1.11111111111111111111111$$

2.9.4 浮点算术运算

1. 浮点乘法运算

浮点乘法运算规则在所有的运算规则中是最简单的，因此我们首先讨论它。

设两个浮点数x和y（真值）：

$$\begin{cases} x = \pm 2^{e_x} m_x \\ y = \pm 2^{e_x} m_y \end{cases} \qquad (2.72)$$

则浮点乘法运算规则是：

$$xy = \pm 2^{e_x + e_y} \left(m_x + m_y \right) \qquad (2.73)$$

乘法法则：乘积有效数是相乘两数的有效数之积，乘积的指数是相乘两数的指数之和，符号是同号为正，异号为负。

1. 第一部分是使用普通的整数运算器进行有效数的乘法运算。因为 IEEE 754 浮点数的有效数是采用符号－绝对值（原码）形式存储的，乘法器只需要能处理无符号数的乘法运算就可以了。
2. 第二部分要计算新的指数。因为指数是

采用移码形式存储的，故可以采用移码的运算公式来计算新的指数。

浮点数的运算应该注意以下的问题：

1. 指数上溢（Exponent Overflow）：一个正指数超出了最大允许指数值。某些系统将其设计成+∞或-∞。
2. 指数下溢（Exponent Underflow）：一个负指数超出了最大（绝大值）允许指数值。这意味着那个数太小无法表示，一般可报告成 0。
3. . 有效数下溢（Significand Underflow）:处理有效数对齐时，从有效数右端有数字丢失。下面将要讨论，此时需要某种形式的舍入。
4. 有效数上溢（Significand Overflow）:同符号的两个有效数相加可能导致最高有效位的进位。这可通过重新对齐来修正，后面将说明。

舍入策略（Rounding Policy）。对有效数操作的结果通常存入更长的寄存器中。当结果回到浮点格式时，必须要排除掉多余的位。

1. 就近舍入（Round To Nearest）：结果被舍入成最近的可表示的数。
2. 朝+∞舍入（Round Toward +∞）：结果向正无穷大方向取入，即结果舍入到正无穷大方向的最近的可表示的数（向正（右）方向找到第一个可表示的数）。
3. 朝－∞舍入（RoundToward －∞）：结果向负无穷大方向取“入”，即结果舍入到负无穷大方向的最近的可表示的数（向负（左）方向找到第一个可表示的数）。
4. 朝 0 舍入（Round Toward 0）：结果朝 0 取舍。

就近舍入是 IEEE 754 标准的默认的策略。以单精度浮点格式为例，当有效位数超

过 24 位时，使用一个离它最近的可表示的浮点数。当有两个可表示的数离它同样近时，如果最低有效位为 0，则简单地舍弃多余的附加位；如果最低有 效位为 l，则加 1，使最低有效位变成 0。总的来讲，这种舍入策略可以形象地说成是“ 四舍六入 ”方法，即四肯定舍，六一定入。“ 五 ”要单独处理，这与平时入们常说的“ 四舍五入 ”是不同的。

例如，当 24 位有效数字后面的附加位为 10010 时，由于它超过了可表示数最低有效位的一半，所以向最低有效位加 l。即向上舍入到一个可表示的浮点数，使得结果的绝对值变大。

当附加位为 01111 时，由于它未超过可表示数最低有效位的一半，所以简单地把它舍弃掉，即向下舍入到一个可表示的浮点数，使得结果的绝对值变小。

有效数字后面的附加位是 10000 时，IEEE 754 采取的方法是强迫结果是偶数：若计算结果是严格位于两个可表示数的正中间，则结果的最低可表示位当前是 1，则值向上入，即向最低有效位加 1；若最低可

表示位当前是 0，则值向下舍，即简单地忽略附加位。

朝－∞舍入和朝－∞舍入策略是朝正或负的无穷大方向舍入。它们在实现一种称为区间算法（Interval Arithmetic）的技术中是有用的。

朝 0 舍入：简单截断，不管多余位。

优点：简单；

缺点：在计算中产生一致的向下偏差。这是比任何偏差都更为严重的偏差，因为这种偏差对任何产生非零多余位的运算都有影响。

例 2.31：用 IEEE754 单精度浮点数完成以下两个浮点数的乘法，$x=-1\times2^3$，$y=1\times2^4$。

解：

$$x=(-1)^1\times2^{130-127}\times(1+0)$$

$$y=(-1)^0\times2^{131-127}\times(1+0)$$

再与（2.71）式比较得：$S_x=1$，$S_Y=0$，$E_x=(130)_{10}=10000010$，$E_y=(131)_{10}=10000011$，$F_x=F_y=00000000000000000000000$，于是浮点数格式为：

X=1 10000010 00000000000000000000000

Y=0 10000011 00000000000000000000000

计算阶码。利用（2.73）和（2.67）式并注意到 $B=(127)_{10}=01111111$ 有：

$$E=E_x+E_y+10000001 \quad (\text{mod } 2^8)$$

即：

$$\begin{array}{r} 10000010 \\ 10000011 \\ +\ 10000001 \\ \hline 10000110 \end{array}$$

于是乘积的阶码的移码为：E=10000110。对应的真值（十进制）是 $e=E-127=134-127=7$。

求乘积的有效数。

$$M=(1+F_x)\times(1+F_y)$$

$$=1.00000000000000000000000。$$

于是，F=0.00000000000000000000000。这是已经规格化的尾数。如果不是，则需要进行规格化处理，然后还要进行舍入处理。

乘积的符号是负，即 S=1。

于是得到了浮点数的乘积的格式如下：

V=1 10000110 00000000000000000000000

浮点数乘法流程图

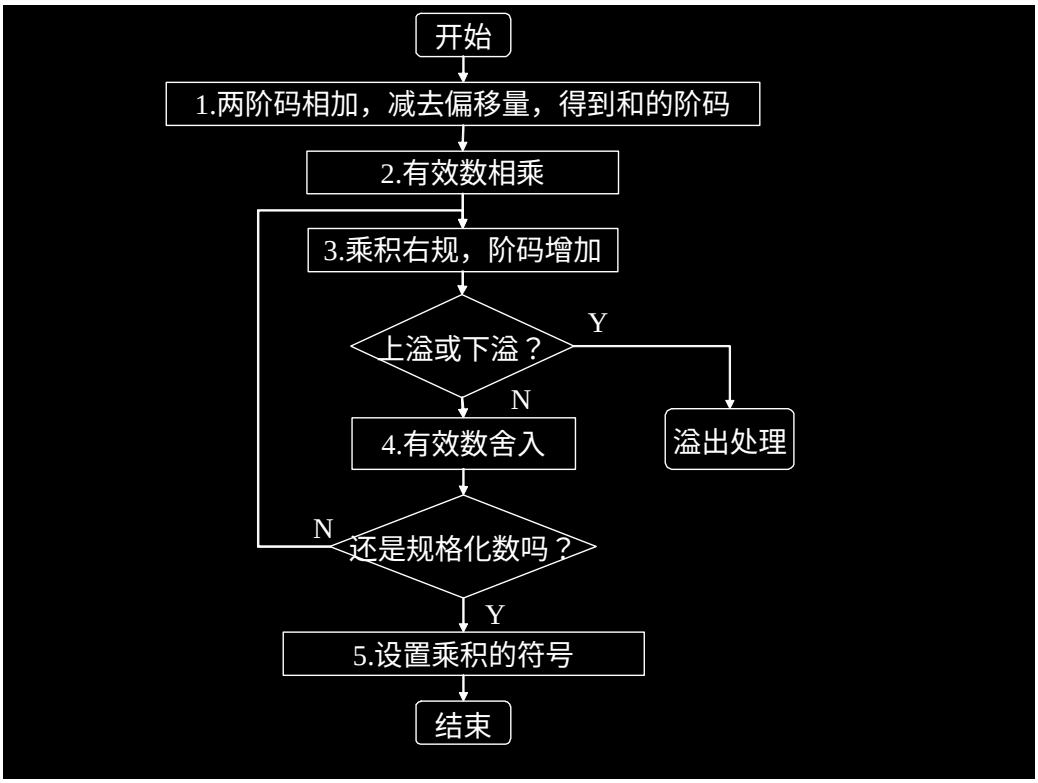


图 2.22

2. 浮点乘法硬件实现的逻辑电路

确定符号位：异或门输出的是结果的符号位（$S_c=S_a \oplus S_b$）。

阶码运算：加法器 ADD 完成的运算是 $E_c=E_a+E_b-127$。

有效数乘法运算：有效数乘法计算完成 $1.F_a \times 1.F_b$。**其整数部分只可能有 01，10 和 11 这 3 种情况**。

证明：

$$1.F_a \times 1.F_b=(1+F_a)\times(1+F_b)$$

$$=1+(F_a+F_b)+(F_a\times F_b) \qquad (2.74)$$

第一种情况：

1	1
(F_a+F_b)	0.bb…b
$(F_a\times F_b)$	+ 0.bb…b
$1+(F_a+F_b)+(F_a\times F_b)$	0 1.bb…b 最高数值位无进位
	1 0.bb…b 最高数值位有进位

第二种情况：

1	1
(F_a+F_b)	1.bb…b
$(F_a\times F_b)$	+ 0.bb…b
$1+(F_a+F_b)+(F_a\times F_b)$	1 0.bb…b 最高数值位无进位
	1 1.bb…b 最高数值位有进位

即整数部分只可能有 01，10 和 11 这 3 种情况。

硬件实现的逻辑电路

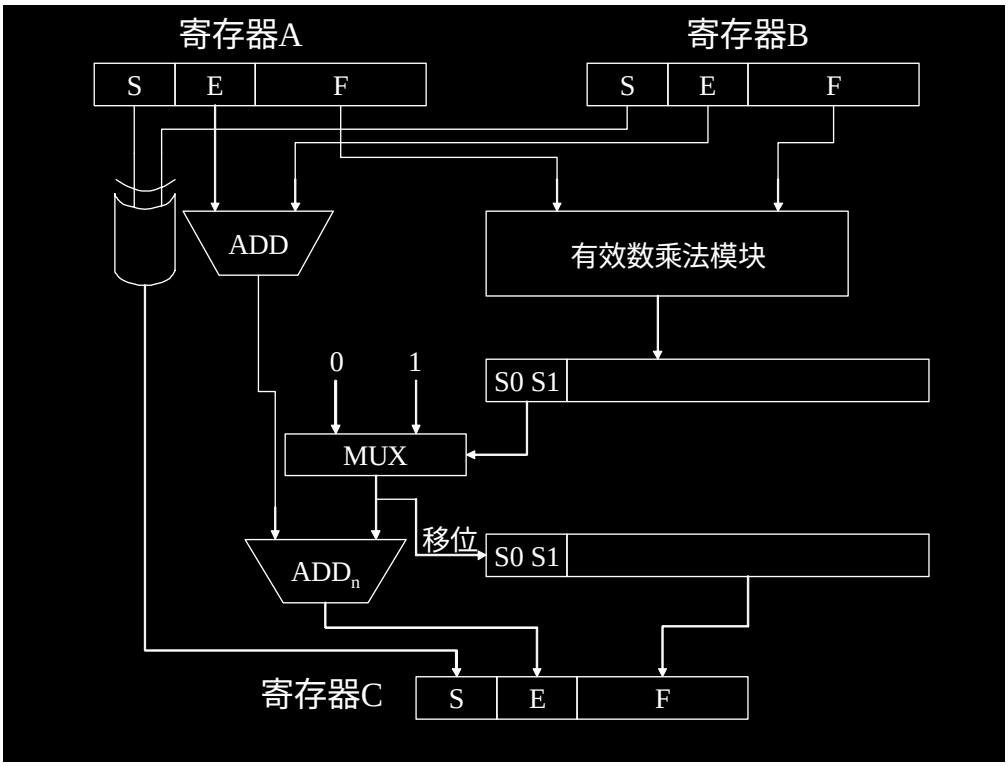


图 2.23

3. 浮点除法运算

设两个浮点数x和y，x和y分别为被除数(dividend)和除数(divisor)，$q=x/y$为商(quotient)，

$$\begin{cases} x = \pm 2^{e_x} m_x \\ y = \pm 2^{e_x} m_y \end{cases} \tag{2.75}$$

则浮点除法运算规则是：

$$xy = \pm 2^{e_x + e_y} \left(m_x + m_y \right) \tag{2.76}$$

除法法则：商的有效数是两相除的浮点数的有效数之商，商的阶码是两相除的浮点数的阶码之差。商的符号是同号为正，异号为负。

例 2.32：用 IEEE754 单精度浮点数完成以下两个浮点数的除法，$x=-1\times2^3$，$y=1\times2^4$，求 x/y。

解：

$x=(-1)^1\times2^{130-127}\times(1+0)$

$y=(-1)^0\times2^{131-127}\times(1+0)$

再与（2.71）式比较得：$S_x=1$，$S_Y=0$，$E_x=(130)_{10}=10000010$，$E_y=(131)_{10}=10000011$，$F_x=F_y=00000000000000000000000$，于是浮点数格式为：

X=1 10000010 00000000000000000000000

Y=0 10000011 00000000000000000000000

计算商的阶码。利用公式：$E=E_x-E_y+01111111\ (\mathrm{mod}\ 2^8)$ 得：

$$\begin{array}{r} 10000010 \\ 01111101 \\ +\quad 01111111 \\ \hline 01111110 \end{array}$$

于是商的阶码的移码为：E=01111110。对应的真值（十进制）是$e=E-127=126-127=-1$。

求商的有效数。可以采用前面介绍过的定点数的除法求得：

$Q=(1+F_x)/(1+F_y)=1.000000000000000000000$。

于是，F=0.00000000000000000000。这是已经规格化的尾数。如果不是，则需要进行规格化处理，然后还有进行舍如处理。

商的符号是负。即 S=1。

于是得到了浮点数的商的格式如下：

V=1 01111110 00000000000000000000

4. 浮点除法硬件实现的逻辑电路

符号确定：异或门输出的是结果的符号位（$S_c=S_a \oplus S_b$）。

阶码计算：加法器 ADD 完成的运算是 $E_c=E_a-E_b+127$（见（2.68）式），E_c 是计算结果的阶码。

有效数计算：求 $1.F_a \div 1.F_b$。由于尾数是 $1.F_a \div 1.F_b$ 的小数部分，商的整数部分只可能是 0 或 1。

证明：

最小商：1.000…00／1.111…11>0.100…00

最大商：1.111…11／1.000…00=1.111…11

而所有的商都介于最小商和最大商之间。

硬件实现的逻辑电路

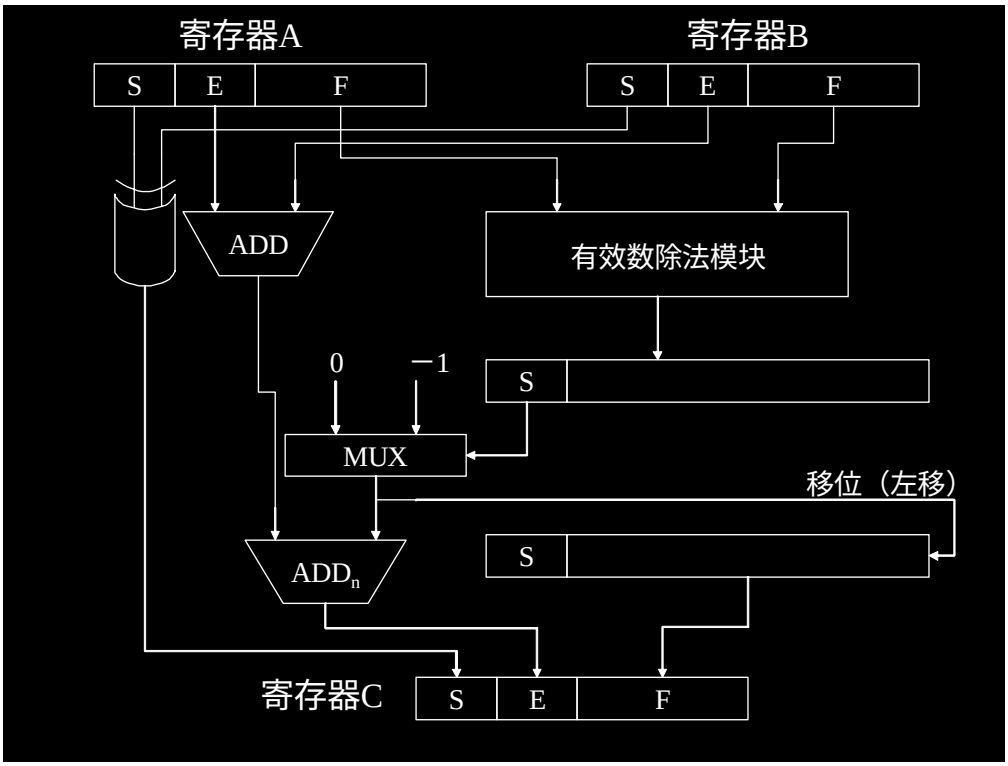


图 2.24

5. 浮点加减法运算

设两个浮点数x和y，

$$\begin{cases} x = \pm 2^{e_x} m_x \\ y = \pm 2^{e_x} m_y \end{cases}$$

则浮点除法运算规则是：

$$x \pm y = \left(m_x 2^{e_x - e_y} \pm m_y\right) 2e^y \quad (2.77)$$

几个概念：

对阶：若两数阶码不等，表示小数点位置没有对齐，此时必须使两数阶码相等，这个过程叫做对阶。

规格化：有效数的最高位是1。

舍入：为了减小误差而采取的措施。

1.对阶

要对阶，首先应求出两数阶码*Ex*和*Ey*之差，即：

$$\triangle E = Ex - Ey$$

当$\triangle E \neq 0$时，要通过有效数的移位以改变*Ex*或*Ey*，使之相等。

由于有效数左移引起最高有效位丢失，误差大，故通常有效数右移，此时阶码要增加。于是有如下的对阶原则。

对阶原则：小阶向大阶看齐。

2.有效数求和

有效数的求和可以按照前面介绍过的方法进行。

3.规格化

有效数求和时结果的整数部分可能大于 1，此时应采取向右规格化(简称右规)措施。向右规格化规则：尾数右移 1 位，阶码加 1。

4.舍入

在IEEE 754标准中，有几种舍入方法。

5.检查阶码是否溢出

浮点数溢出以其阶码溢出表现出来。

阶码下溢：置运算结果为非规格化数或浮点形式的机器0。

阶码上溢：置溢出标志。

总之，浮点加减法运算比较复杂一些。我们以一个例子来对浮点加减运算规则进行描述。

例 2.33：用 IEEE754 单精度浮点数完成以下两个浮点数的加法运算（就近舍入。），$x=1.0101\times2^3$，$y=1.1001\times2^4$，求 $z=x+y$。

解：真值：

$x=(-1)^0\times2^{130-127}\times(1+0.0101)$

$y=(-1)^0\times2^{131-127}\times(1+0.1001)$

再与（2.71）式比较得：$S_x=1$，$S_Y=0$，$E_x=(130)_{10}=10000010$，$E_y=(131)_{10}=10000011$，$F_x=0.01010000000000000000000$，$F_y=0.10010000000000000000000$，于是浮点数的 IEEE754 格式为：

X=0 10000010 01010000000000000000000

Y=0 10000011 10010000000000000000000

第一步，对阶

求阶差：利用（2.77）和（2.68）式并注意到 $B=(127)_{10}=01111111$ 有：

$\Delta E=E_x-E_y+01111111 \quad (\mathrm{mod}\ 2^8)$

即：

$$\begin{array}{r} 10000010 \\ 01111101 \\ +\quad 01111111 \\ \hline 01111110 \end{array}$$

阶差的移码为：$\Delta E=01111110$。对应的真值（十进制）是 $\Delta e=\Delta E-127=126-127=-1$。因此，$E_x<E_y$。

移位：按照小阶向大阶看齐的对阶原则，应当右移 X 的有效数 1 位（注意，是尾数连同隐藏位一起右移），同时 X 的阶码加 1。对阶后的 X 为(移出的尾数用括号扩起来)：

0 10000011 10101000000000000000000(0)

第二步，有效数求和（注意，是尾数连同隐藏位一起求和）。

$$
\begin{array}{l}
\quad 0.10101000000000000000000(0) \\
+\ 1.10010000000000000000000 \\
\hline
1\ 0.00111000000000000000000(0)
\end{array}
$$

第三步，规格化

求和后的整数部分大于 1，需要将有效数规格化。求和后的有效数右移 1 位，阶码加 1。由于 1 的移码是 10000000，利用(2.68)式并注意到$[-B]=[-127]_{补}=10000001$ 有：

$$
\begin{array}{r}
10000011 \\
10000000 \\
+\ 10000001 \\
\hline
10000100
\end{array}
$$

故和为：

Z=0 10000100 00111000000000000000000(00)

第四步，舍入处理

由于移出的位为 00，按照 IEEE 754 标准的就近舍入策略，00 应当舍去。

第五步，溢出判断

无溢出。真值是：$z=1.000111\times 2^5$

算法流程图

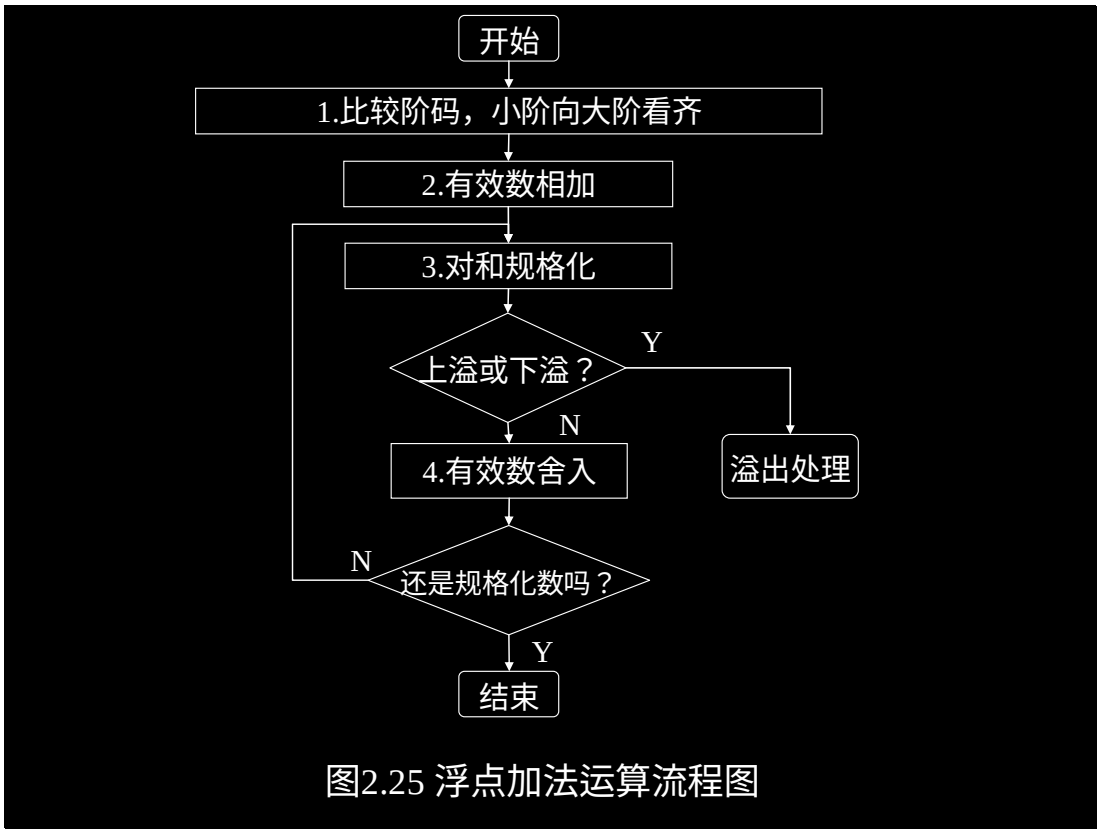


图2.25 浮点加法运算流程图

硬件实现的逻辑电路

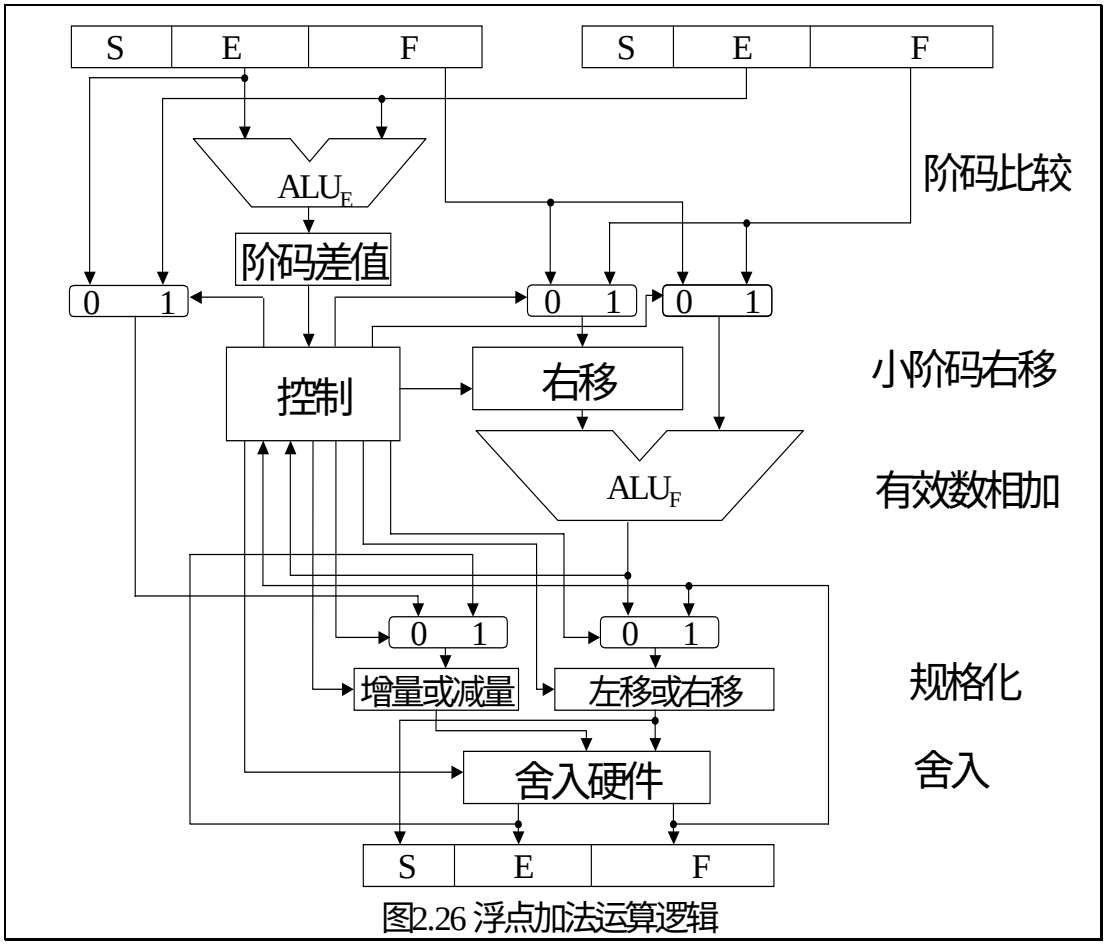


图2.26 浮点加法运算逻辑