5.5 浮点数的表示与运算

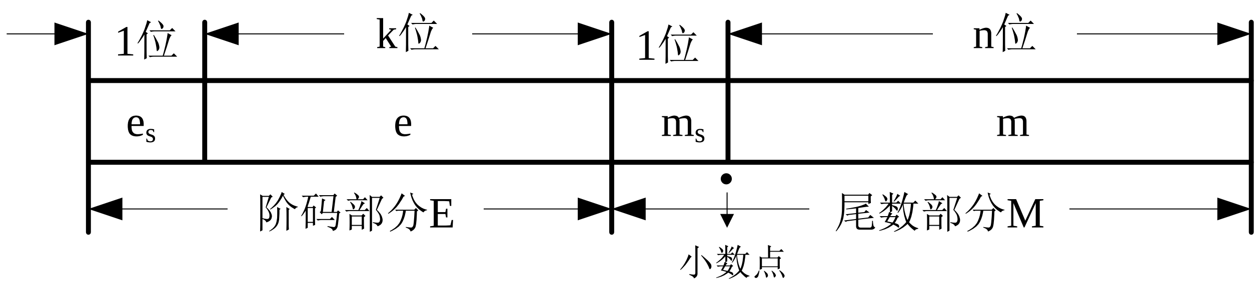
5.5.1 浮点数的基本格式

- 浮点表示法将数表示成“数值×指数”形式，即书写成：

  $N=M\times R^{E}$

  其中，M称作尾数，用小数表示；E称作阶码，用整数表示；R称作阶的基数或阶的底（其值固定，可取2，4，8，16等）。

- 浮点数的一般格式



- 浮点机器码书写形式

  E_s，$E_k\ldots E_2E_1$；$M_s\,.\,M_1M_2M_3\ldots M_{n-1}M_n$

- 浮点数阶码的位数决定浮点数的表示范围；

  尾数的位数决定浮点数的有效精度。

5.5.2 浮点阶的移码表示

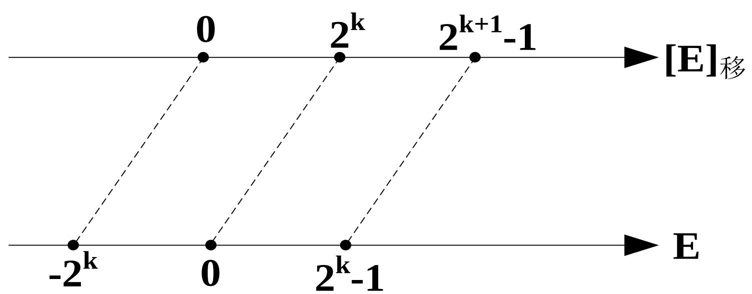
□ 移码的定义

一位符号位和 k 位数值位组成的移码, 其定义为:

$$[E]_{移} = 2^k + E \qquad -2^k \le E < 2^k$$

□ 式中，2^k称为移码的偏置常数或位移量。

□ **整数的真值和移码在数轴上的映射关系**



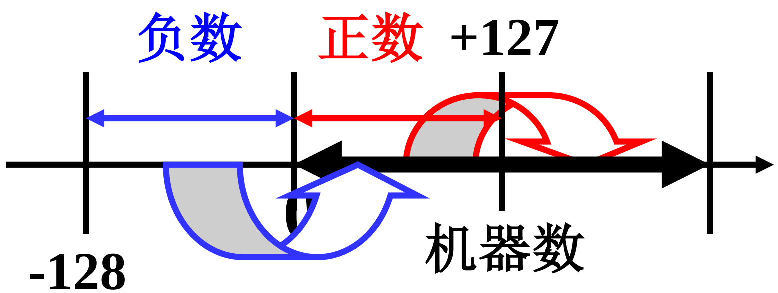
□ **移码表示的几何意义：将k位数的真值在数轴上向右平移了2^k个位置。这给两个数的比较操作带来了极大的方便。**

移码表示举例

例如：1位符号位和 7 位数值位组成的移码, 其定义为：

$$[E]_{移} = 2^7 + E \qquad -2^7 \leq E < 2^7$$

编码集合：00000000 ~ 11111111



8 位移码表示的机器数为：

数的真值在数轴上向右平移了 128 个位置。

通常移码只用于表示整数，且只用于表示浮点数的阶码部分。

移码和补码的关系：符号位取值相反，数值部分取值相同。

与补码类似，移码零的表示唯一（1000 0000）。

移码与其他机器码的比较

例：设机器数字长为8位（其中一位符号位），对于整数，当其分别代表无符号数、原码、反码、补码和移码时，对应的真值范围各为多少?

解：8位二进制代码的所有组合与无符号数、原码、反码、补码和移码所代表的真值的对应关系如下页表。

8位机器整数代表不同码制时的真值范围

二进制代码	无符号数对应的真值	原码对应的真值	反码对应的真值	补码对应的真值	移码对应的真值
0000 0000	0	+0	+0	0	−128
0000 0001	1	+1	+1	+1	−127
0000 0010	2	+2	+2	+2	−126
……	……	……	……	……	……
0111 1110	126	+126	+126	+126	−2
0111 1111	127	+127	+127	+127	−1
1000 0000	128	−0	−127	−128	0
1000 0001	129	−1	−126	−127	+1
1000 0010	130	−2	−125	−126	+2
……	……	……	……	……	……
1111 1101	253	−125	−2	−3	+125
1111 1110	254	−126	−1	−2	+126
1111 1111	255	−127	−0	−1	+127

5.5.3 浮点表示法

- 规格化的浮点数

  浮点数表示形式并不唯一。为能充分利用尾数有效位数，常规定尾数最高位必须是一个有效数值。满足规定的称为规格化浮点数，不满足规定的则称为非规格化浮点数。

- 规格化浮点尾数的绝对值范围应满足：

$$\frac{1}{R} \le |M| < 1 \ (\le 1，补码)$$ 若R=2，则 $$\frac{1}{2} \le |M| < 1 \ (\le 1, 补码)$$

- 当R=2时规格化浮点数尾数形式为

  $[M]_{原}$ = 0.1xx…x 或1.1xx…x，判断条件：$M_{MSB}=1$

  $[M]_{补}$ = 0.1xx…x 或1.0xx…x，判断条件：$M_s \oplus M_{MSB}=1$

浮点数的规格化处理

- 若浮点数超出规格化表示范围，需**同时调整数尾数和阶码的大小**，转换到规格化数。这一操作叫作对浮点数的**规格化处理**。具体又分**左规**和**右规**两种。
  - **右规**：尾数值溢出到数符位中。这在定点运算中属溢出出错，但浮点运算中可通过将**尾数右移一位，阶码加1**的操作，将溢出的尾数调整回正常表示范围中来，简称为**右规**。**右规只需进行一次**。
  - **左规**：若尾数绝对值小于规格化尾数值，可通过尾数左移调整回规格化范围中来，简称为**左规**。可能需要左移多位才能达到规格化要求。**左规时尾数每左移一位，阶码减1，直到符合规格化数条件为止。**

浮点机器零的表示

- 浮点运算时0的表现
  - 真值为0（真0）；
  - 尾数为0，阶码不为0且未达最小值（最负阶）；
  - 阶码已达最小值，尾数仍为非规格化数形式;
  - 阶码已小于最小值（下溢），尾数仍不为0。
- 除了真0，不论出现哪种情况，都说明数的绝对值已经小到无法用规格化浮点数表示了，在采用规格化浮点数的机器中只能作0处理。称为“机器0”。
- 为了保证浮点表示时0的形式上的一致性，通常规定机器零的标准格式为：尾数为0，阶码为最小值（最负阶）。
- 为便于判零操作实现，希望机器零的代码为一串零形式，许多计算机规定浮点数阶码用移码，尾数用补码或原码表示。

规格化浮点数的表示范围

- 当基为2，阶码用移码、尾数用原码表示时（阶移尾原格式），规格化浮点数的取值范围为：

最小负数： $-2^{2^k-1} \times (1-2^{-n})$

最大负数： $-2^{-2^k} \times 2^{-1}$

负数可表示范围

机 器 零： 0

最小正数： $2^{-2^k} \times 2^{-1}$

最大正数： $2^{2^k-1} \times (1-2^{-n})$

正数可表示范围

规格化浮点数的表示范围（续）

- 采用不同的码制来表示浮点数时，由于各码制的定义域不一样，会导致表示范围不同。例如
- 当基为2，采用阶移尾补格式表示时，规格化浮点数的取值范围则变为：

最小负数： -2^{2^k-1}

最大负数： $-2^{-2^k} \times (2^{-1}+2^{-n})$

负数可表示范围

机 器 零： 0

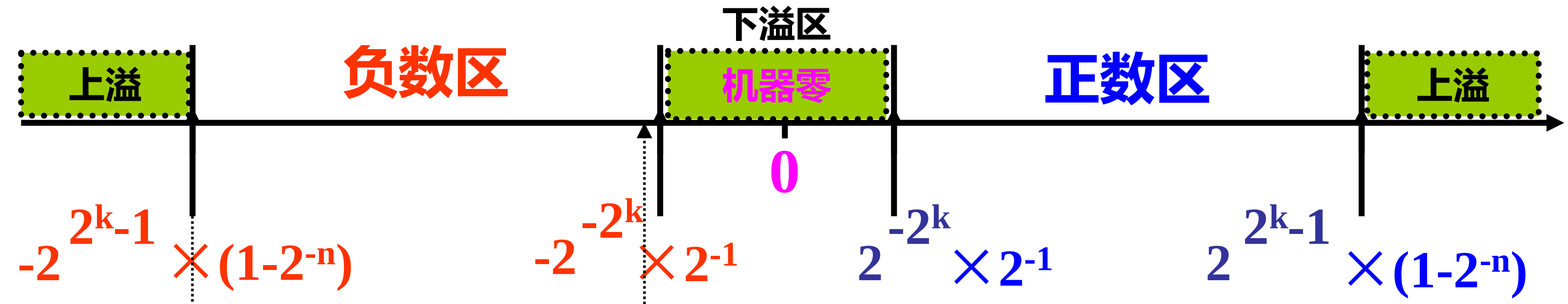
最小正数： $2^{-2^k} \times 2^{-1}$

最大正数： $2^{2^k-1} \times (1-2^{-n})$

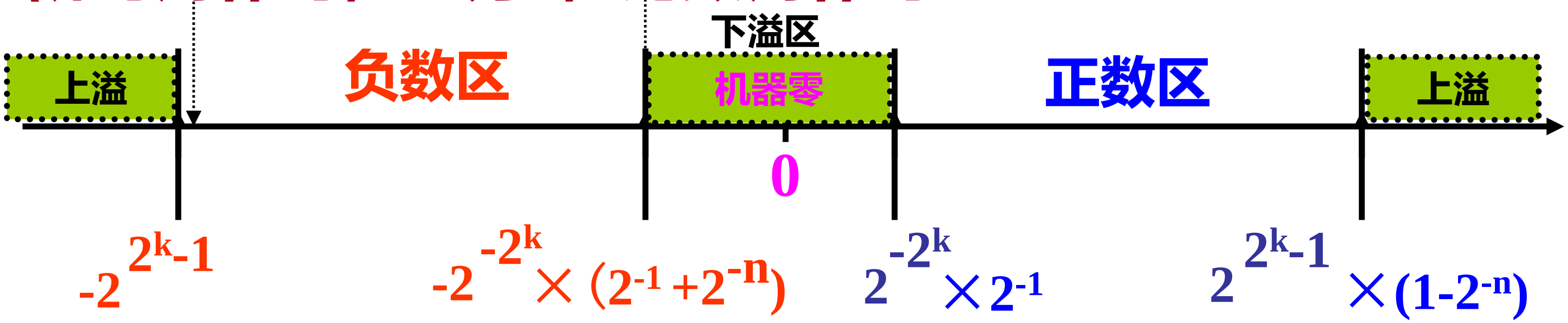
正数可表示范围

规格化浮点数范围的数轴表示

阶码用移码，基为2；尾数用原码



阶码用补码，基为2；尾数用补码



- 浮点数表示范围由**正数域、机器0和负数域三部分**组成。
- 真0是数轴上一个点，**机器0对应数轴上0点附近的一段区间**

规格化浮点数的溢出

- 浮点数溢出是阶码超出其表示范围，又分阶码上溢和阶码下溢两种。
  - 运算结果大于浮点数可表示的最大阶时，称为浮点数上溢。分为正上溢和负上溢。上溢属于溢出出错。需置溢出标志，并转溢出中断程序处理。
  - 运算结果小于最小阶（最负阶）称为浮点数下溢。也分为正下溢和负下溢。下溢时不作为出错，仅作为机器零。
- 浮点数溢出判断方法与阶码采用的码制有关，如果用补码或移码表示阶码，则设置双符号位判断溢出十分方便。

规格化浮点数的隐藏位表示

- 尾数采用原码规格化表示时，其绝对值的最高数值位必定为“1”。为了提高表数精度，可以采用“隐藏位”技术。
  - 把浮点数存入内存前，通过尾数的左移强行把该位去掉，同时约定该位数总是在小数点之后，居于尾数的最高位。
  - 在取出带有隐藏位的浮点数到运算器执行运算时，必须先恢复隐藏位，运算器的位数也要随之增加1位。

◆为避免二意性本教材约定：除非特别说明，书中所有与浮点格式相关的讨论、例题、习题等均不考虑隐藏位。
◆隐藏位方案在不同机器中实现时细节上会有所不同。

浮点基数的选择

❐ 基数对规格化尾数的影响

- R=4，规格化尾数绝对值≥1/4，原码规格化尾数的最高两位不全为0，补码规格化尾数中最高两位至少有一位与符号位不同。阶码+1或-1，尾数要右移或左移两位；
- R=8，规格化尾数绝对值≥1/8，原码规格化尾数的最高三位不全为0，补码规格化尾数中最高三位至少有一位与符号位不同。阶码+1或-1，尾数要右移或左移三位；
- R=16，规格化尾数绝对值≥1/16，原码规格化尾数的最高四位不全为0，补码规格化尾数中最高四位至少有一位与符号位不同。阶码+1或-1，尾数要右移或左移四位。
- 例如：尾数M=0.0001X…X，对于R=2的浮点数来说，这是一个非规格化数；对于R=16的浮点数来说，这已是一个规格化数，无需再进行规格化操作。

浮点基数的选择（续）

❐ 基数对表示范围的影响

基数R大，能使浮点数的表示范围明显增大。

例如，32位浮点数如果阶码为7位（含一位阶符），尾数25位（含一位数符），采用阶移尾补形式，则

- R=2，浮点数表示范围为$-1\times2^{63}\sim(1-2^{-24})\times2^{63}$；
- R=8，浮点数表示范围为$-1\times8^{63}\sim(1-2^{-24})\times8^{63}$；
- R=16，浮点数表示范围为$-1\times16^{63}\sim(1-2^{-24})\times16^{63}$；

由上例可以看出随着基数的变大，浮点数的表示范围有沿数轴向正负两边显著扩展的趋势。

浮点基数的选择（续）

- **基数对表示精度的影响**

  随着R增大，浮点数的**粒度随之变粗**，数轴上各点的**排列更稀疏**，表示精度随之下降。仍以32位浮点数为例来看变化情况。

  - R=2，规格化最小正数为$2^{-1}\times2^{-64}$，尾数最高位为有效数值；
  - R=8，规格化最小正数为$8^{-1}\times8^{-64}$，尾数的最高两位为0；
  - R=16，规格化最小正数为$16^{-1}\times16^{-64}$，尾数最高三位为0；

  分析：R越大，数据在数轴上的分布密度越稀，精度受影响越大。

- **结论：基数增大对浮点数表示范围有着正面的影响，但对表示精度却带来负面的影响。**

定点表示法与浮点表示法的比较

(1) 浮点数比定点数表示范围大；

(2) 浮点数有效精度比同字长定点数低。浮点表示法实质是牺牲精度为代价来扩大数据表示范围；

(3) 溢出判断方法不同。浮点运算以阶码上溢为溢出标志、定点数则以数值是否超出范围破坏了符号位为溢出标志；

(4) 浮点数由阶码和尾数两部分组成。比定点数结构复杂，运算步骤增多，运算速度下降，运算线路复杂；

(5) 由于表示范围大，浮点数运算时能有效减少运算结果产生溢出的次数，减轻编程时选择比例因子的工作量。

浮点数在表示范围、溢出处理和编程的方便性上优于定点数，而在表示精度、运算速度及运算的复杂性等方面又不如定点数。

浮点表示法对性能结构的影响

按照对浮点功能的设置方式，通常可以将计算机分为：

(1) 定点机：只能表示和处理定点数，指令系统不支持任何浮点功能。结构简单，成本低廉。

(2) 定点机+浮点选件：浮点协处理器是一种专门用来对计算机中的浮点数进行处理的LSI芯片，其内部主要包括浮点运算部件和浮点处理部件，可在定点机原有指令系统的基础上执行一套附加的浮点指令，完成对定点机浮点功能的扩展。 协处理器无法单独工作，只能和主CPU协同运行。浮点协处理器选件使浮点运算的速度大大提高。

(3) 浮点机：具有浮点运算指令和浮点运算器，实数处理速度和处理能力都十分强大。

浮点表示法举例

- 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 $-10^{38} \sim -10^{-38}$；

  正数 $+10^{-38} \sim +10^{38}$。

  （2）精度：7位十进制数据。

- 解：

  （1）浮点数表示范围的设计主要是确定阶码的位数。

  采用试探法： 由$2^{10} > 10^3$

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

  又因 $2^7 > 10^2$

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

  同理可得 $2^{-127} < 10^{-38}$

浮点表示法举例（续）

- 由上分析可知：阶码取8位（含1位阶符），且用补码或移码表示时，对应的浮点数取值范围为 $2^{-128} \sim 2^{+127}$，可以满足题意要求。

（2）浮点数精度的设计主要是确定尾数的位数。

由于$10^7 \approx 2^{23}$，故尾数的数值部分可取23位。加上1位数符位，可达24位。

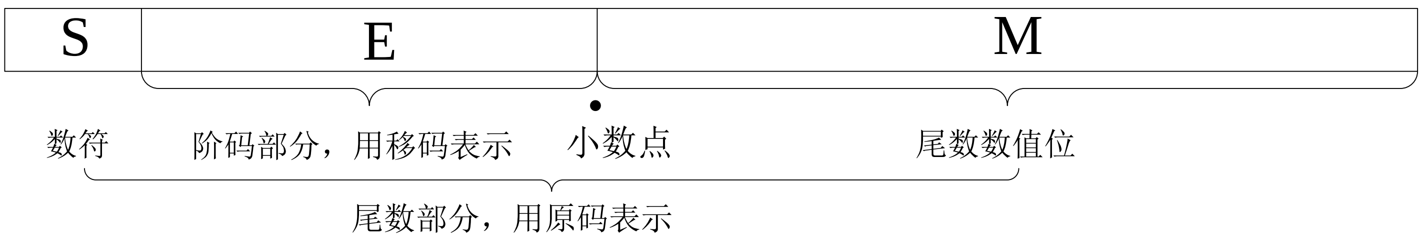
- 最终该浮点数至少取32位，其中阶码8位（含1位阶符），尾数24位（含1位数符），基数R=2。

5.5.4 IEEE754浮点标准

- **IEEE浮点格式**

  由IEEE（电气和电子工程师协会）于1985年提出，为不同计算机系统之间浮点数表示格式的统一标准。

- **IEEE754标准的浮点数格式**



- **为了便于讨论，我们把IEEE 标准浮点数格式书写为**

  $\mathbf{S, E_sE_k\ldots\ldots E_2E_1;\ .M_1M_2\ldots\ldots M_n}$

IEEE754标准浮点数种类

类型	数符S	阶码E	隐藏位	尾数值M	总位数	偏置常数	
						十六进制	十进制
短实数	1	8	有	23	32	7FH	+127
长实数	1	11	有	52	64	3FFH	+1023
临时实数	1	15	无	64	80	3FFFH	+16383

32位IEEE754标准浮点数

S(31)	E(30-23)	M(22-0)

754标准格式规定：

- 由数符S、阶码E、尾数M三部分组成，指数以2为底、尾数以2为基数；
- 符号位S占1位，安排在最高位，S=0表示正数，S=1表示负数；
- 阶码E占8位，移码表示，偏移量（偏置常数）为+127；
- 尾数M占低23位，用原码表示，小数点在尾数域的最前面；
- 尾数域表示的值是1.M。由于最高有效位总是1，可将1隐藏在小数点左边不予存储，尾数实际24位。则一个32位浮点数的实际真值为：

$$X=(-1)^{s}\times(1.M)\times 2^{E-127} \quad (E:1\sim254)$$

- 当阶码E=0…0且尾数M=0…0时，表示的真值X=±0；
- 当阶码E=1…1且尾数M=0…0时，表示的真值X=±∞；
- 当阶码E=0…0且尾数M不为0时，若隐藏位=0，表示非规格化数。
- 当阶码E=1…1且尾数M不为0时，用来表示非数（即不是一个数）。

IEEE754标准浮点数的舍入

- IEEE754标准给出四种可供选择的舍入处理方法

(1) 就近舍入：运算结果可被舍入成最接近的可表示数，类似于“0舍1入”法。这是标准默认的舍入方式。

(2) 朝0舍入：就是简单的截断法，丢掉多余位后不作任何进一步的处理。其舍入后果总是使结果的绝对值变小（单向负误差），朝向数轴原点趋近；

(3) 朝+∞舍入：结果朝正无穷大方向向上舍入。若尾数为正数，只要需舍去的多余位不全为0，做最低有效位加1操作；若尾数为负数，则简单的截尾；

(4) 朝-∞舍入：结果朝负无穷大方向向下舍入。处理方法与朝+∞舍入相反，正数做简单的截尾；对负数而言，只要多余位不全为0，做最低有效位加1操作。

IEEE754标准浮点数的主要特点

(1) 符号位放在浮点数格式的最高位，便于判0和判断符号操作的实现；

(2) 机器零为一串0的形式（E和M均为零），便于判0；

(3) 两浮点数进行大小的比较时，可不必区分阶码位和数值位，视同两定点数比较一样对待；

(4) 设置隐藏位可提高尾数的表示精度，且隐藏的数值为1，而不是通常的2^{-1}；

(5) 移码采用2^k-1而不是通常的2^k作为偏置常数，通过对0或全1极端阶码值的充分利用，增加了表示的多样性。

IEEE 754标准应用举例

1、将$(100.25)_{10}$转换成短浮点数格式。

解：(1) 把十进制数转换成二进制数

$$(100.25)_{10} = (110\ 0100.01)_2$$

(2) 规格化二进制数

$$110\ 0100.01 = 1.1001\ 0001 \times 2^{110}$$

(3) 计算出阶码的移码（偏置值+阶码真值）

111 1111(127H) + 110 = 1000 0101

(4) 以短浮点数格式存储该数

该数的符号位 = 0，阶码 = 1000 0101

尾数 = .100 1000 1000 0000 0000 0000 （23位）

所以，(100.25)10的短浮点数代码为：

0，1000 0101；.100 1000 1000 0000 0000 0000

十六进制值为：42C8 8000H

IEEE 754标准应用举例（续）

2、将短浮点数 C1C9 0000H 转换成十进制数。

解：(1) 把十六进制数转换成二进制形式，并分离出符号位、阶码和尾数

因为，C1C9 0000H = 1 1000 0011 100 1001 0000 0000 0000 0000B

所以， 符号位 = 1 (1位)

阶码 = 1000 0011 (8位)

尾数 = .100 1001 0000 0000 0000 0000 (23位)

(2) 计算出阶码的真值（即移码 - 偏置值）

1000 0011 -111 1111 = 100

(3) 以规格化二进制数形式写出此数：$1.1001\ 001 \times 2^{100}$

(4) 写成非规格化二进制数形式：1 1001.001

(5) 转换成十进制数，并加上符号位

$(-1\ 1001.001)_2 = (-25.125)_{10}$

5.5.5 规格化浮点加减运算

规格化浮点运算指参加运算的操作数为规格化浮点数，运算完成后结果也是规格化浮点数。

- **浮点运算特点**
  - **小数点需要对齐**
  - **要考虑阶码、尾数两部分操作**
  - **运算结果受规格化表示范围限制**

- **浮点数加减运算**

  对阶、尾数加减、规格化、舍入和结果判溢出五步。

规格化浮点加减运算基本规则

- 设有两个浮点数X和Y，分别表示为

$$X = M_x \times R^{E_x} \qquad Y = M_y \times R^{E_y}$$

- 若求：Z=X±Y=?
- 则浮点加减运算的基本规则可用通式进行描述（*R*=2）

$$Z = X \pm Y = \begin{cases} (M_x \pm M_y \times 2^{-(Ex-Ey)}) \times 2^{Ex}, & E_x \geq E_y \\ (M_x \times 2^{-(Ey-Ex)} \pm M_y) \times 2^{Ey}, & E_x < E_y \end{cases}$$

其中：$2^{-(Ex-Ey)}$和$2^{-(Ey-Ex)}$称为移位因子，反映出对阶时尾数右移的位数。对阶的规则为“小阶向大阶看齐”。

规格化浮点加减运算步骤

（0）检测0 。判断操作数是否为0。

（1） 对阶。两数小数点对齐，即“小阶向大阶看齐”。

（2）尾数运算。两尾数按定点加减运算规则求和（差）。

（3）结果规格化。若运算后尾数不再是规格化形式，对其进行规格化处理。

（4）舍入。为尽量避免精度损失，要对尾数右移时移出的保护位进行舍入。

（5）溢出判断。运算过程中对可能发生溢出出错的操作及时判溢出。

规格化浮点加减运算方法

❐ 0操作数检测

运算前先分别对两操作数X、Y进行判0操作

若$X=0$，则令$X+Y=Y$；$X-Y=-Y$；运算结束；

若$Y=0$，则令$X+Y= X-Y=X$；运算结束。

❐ 对阶

阶码的比较通过“求阶差”操作完成 。

$\Delta E=E_x-E_y$

- 若$\Delta E=0$，$E_x=E_y$，尾数可直接加减；
- 若$\Delta E\neq 0$，$E_x\neq E_y$，需要对阶。
  - ✧若$\Delta E>0$，$E_x>E_y$，E_y+1，$M_y\rightarrow 1$，$\Delta E-1$
    重复到$\Delta E=0$为止。此时$E_z=E_x$
  - ✧若$\Delta E<0$，$E_x<E_y$，E_x+1，$M_x\rightarrow 1$，$\Delta E+1$
    重复到$\Delta E=0$为止。此时$E_z=E_y$

规格化浮点加减运算方法（续）

❐ 尾数相加减

- ❍ 按定点小数加减运算规则进行

  $M_z = M_x \pm M_y$

- ❍ 当尾数用补码表示时有：

$$[M_z]_补 = [M_x \pm M_y]_补 = \begin{cases} [M_x]_补 + [M_y]_补 & \text{求和} \\ [M_x]_补 + [-M_y]_补 & \text{求差} \end{cases}$$

❐ 结果规格化

浮点加减运算结果可能发生两种非规格化情况：

- ❍ **尾数溢出，右规；**
- ❍ **尾数非0，但不满足规格化条件。左规。随着尾数逐位左移，可以将保护位逐位补入尾数的最低位。**

规格化浮点加减运算方法（续）

- **舍入**

  规格化处理后，可能还剩余有若干保护位需要舍去。常用的舍入方法可根据具体情况选用。在此约定本节的舍入处理均采用“0舍1入”法。

- **溢出判断**

  以阶码是否溢出作为判断条件。阶码上溢时作为溢出出错，转溢出处理；阶码下溢时表示为机器零即可。

  浮点加减运算有可能导致阶码上溢的操作是右规和舍入。

  - **由于右规过程中尾数右移一位时阶码+1，而如果阶码已为最大值，再加1就会向上超出其表示范围。**
  - **在采用“0舍1入”法进行舍入时，如果尾数末位加1后溢出，会引起右规，……接下来就和右规时上溢的情形一样了。**

> 溢出判断操作不是等浮点加减运算结束时再做，而是在运算过程中需要的步骤立即进行。

规格化浮点数加减运算举例

- 已知：$X = 2^{010} \times 0.11011011$，
  $Y = 2^{100} \times (-0.10101100)$
  采用阶补尾补格式表示，
  求：$Z_1=X+Y=$？ $Z_2=X-Y=$?

- 解：首先，写出X、Y的浮点机器数表示：
  阶码用 5 位补码（含2位符号位） 尾数用 10 位补码（含2位符号位）
  $[X]_{阶补尾补} = 00, 010; \ 00.11011011$
  $[Y]_{阶补尾补} = 00, 100; \ 11.01010100$

规格化浮点数加减运算举例（续）

$[X]_{阶补尾补} = 00, 010; 00.11011011$

$[Y]_{阶补尾补} = 00, 100; 11.01010100$

（1）对阶：

$[\Delta E]_{补} = [E_X]_{补} + [-E_Y]_{补} = 00, 010 + 11, 100 = 11, 110$

$\Delta E = -2 < 0$，即 $E_X < E_Y$

因此，$[M_X]_{补} = 00.00110110(11)$（即右移2位）

$[E_x]_{补} = 00, 100$

$[X]_{阶补尾补} = 00, 100;\ 00.00110110(11)$

（2）尾数求和差

加：

$$\begin{array}{r} 00.\ 00110110(11) \\ +\ 11.\ 01010100 \\ \hline 11.\ 10001010(11) \end{array}$$

减：

$$\begin{array}{r} 00.\ 00110110(11) \\ +\ 00.\ 10101100 \\ \hline 00.\ 11100010(11) \end{array}$$

规格化浮点数加减运算举例（续）

$[M_{z1}]_{补}=11.10001010(11)$；$[M_{z2}]_{补}=00.11100010(11)$

（3）结果规格化

本例和应左规一次操作，故得

和：$[Z_1]_{阶补尾补}=00,011$；$11.00010101(1)$

差：$[Z_2]_{阶补尾补}=00,100$；$00.11100010(11)$

（4）舍入

本例采用 0 舍 1 入方案得

$[Z_1]_{阶补尾补}=00,011$；11.00010101，舍

$[Z_2]_{阶补尾补}=00,100$；00.11100011，入

（5）判溢出

本例中，和、差的阶码均不溢出。

规格化浮点数加减运算举例（续）

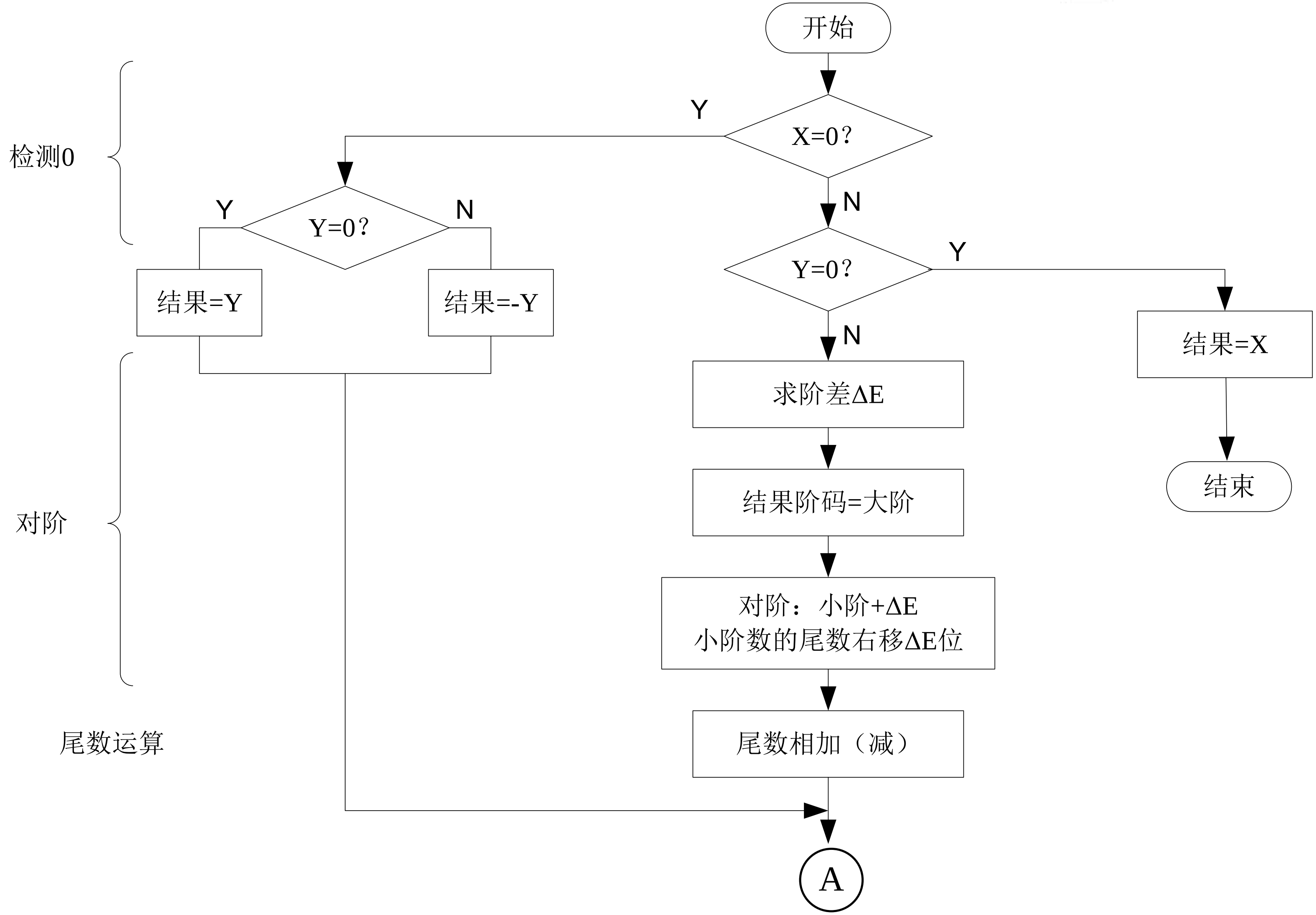
$[Z_1]_{阶补尾补}$= 00, 011； 11. 00010101

$[Z_2]_{浮阶补尾}$= 00, 100； 00. 11100011

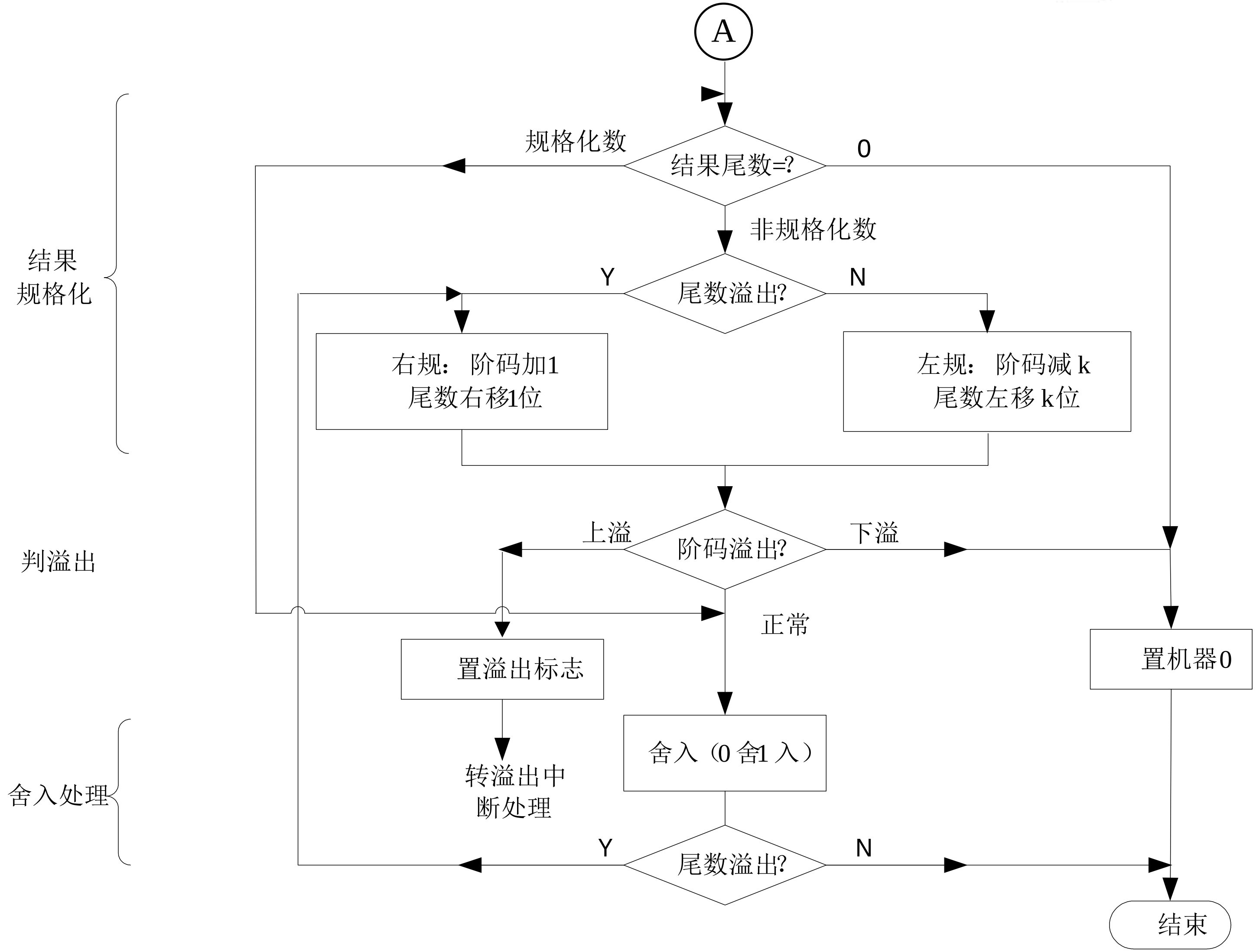
最后得：$Z_1 = 2^{011} \times (-0.11101011)$

$Z_2 = 2^{100} \times (0.11100011)$

浮点加减运算操作流程


开始
X=0？
Y
N
Y=0？
Y
N
结果=Y
结果=-Y
Y=0？
Y
N
结果=X
结束
求阶差ΔE
结果阶码=大阶
对阶：小阶+ΔE
小阶数的尾数右移ΔE位
尾数相加（减）
A
检测0
对阶
尾数运算

浮点加减运算操作流程（续）



5.5.6 规格化浮点乘除运算

❐ 基本规则

设两浮点数

$$X = M_x \times R^{E_x} \qquad Y = M_y \times R^{E_y}$$

若求：**$Z_p = X \times Y = ?$，$Z_q = X \div Y = ?$**

则浮点乘除运算的基本规则可描述为

$$Z_p = X \times Y = (M_x \times M_y) \times R^{E_x + E_y}$$

$$Z_q = X \div Y = (M_x \div M_y) \times R^{E_x - E_y}$$

即：阶码相加（减），尾数相乘（除）

规格化浮点数乘除运算步骤

(0) 检测0：

乘：X或Y=0，则 X×Y =0，不再运算；

除：X=0，则 X÷Y =0；Y=0，转非法除数处理。

(1) 阶码相加、减：

乘：E_X+E_Y；

除：$E_X- E_Y$（可能溢出）

(2) 尾数相乘、除：

乘：$M_X\times M_Y$；

除：$M_X \div M_Y$

(3) 规格化处理（左规或右规，可能溢出）；

(4) 舍入操作（可能带来又一次右规，可能溢出）；

(5) 判断结果的正确性，即检查阶码是否溢出。

- 其中，(3)、(4)、(5)三步操作方法同浮点加减运算。

移码加减运算

- **阶码相加减**

  若阶码采用补码，按补码加减法规则进行即可；

  若阶码采用移码，需按移码加减法规则进行。

- **算法推导**

$$[E_x]_{移} + [E_y]_{移} = 2^k + E_x + 2^k + E_y = [E_x + E_y]_{移} + 2^k$$

$$[E_x]_{移} - [E_y]_{移} = 2^k + E_x - 2^k - E_y = [E_x - E_y]_{移} - 2^k$$

即结果需要 $+2^k$ 或 -2^k 修正（符号位变反）。

- **改进算法**

$$[E_Z]_{移} = [E_x + E_y]_{移} = [E_x]_{移} + [E_y]_{补} \quad \mathrm{Mod}\ 2^{k+1}$$

$$[E_Z]_{移} = [E_x - E_y]_{移} = [E_x]_{移} + [-E_y]_{补} \quad \mathrm{Mod}\ 2^{k+1}$$

$$([E_x]_{移} + [E_y]_{补} = 2^k + E_x + 2^{k+1} + E_y = [E_x + E_y]_{移} \quad \mathrm{Mod}\ 2^{k+1}$$

$$[E_x]_{移} + [-E_y]_{补} = 2^k + E_x + 2^{k+1} - E_y = [E_x - E_y]_{移} \quad \mathrm{Mod}\ 2^{k+1})$$

- **注：运算前把移码转换为补码时，只要将其符号位变反。**

移码运算溢出判断

- 当阶码用移码表示时，为了方便判溢出，可采用双符号位表示，且最高位为0。即

  正数：01；负数：00

- 运算后若结果最高符号位为0时，正确；

  若结果最高符号位为1时，溢出。

  11：下溢，按机器0处理；

  10：上溢，溢出出错。

- 注意：移码用双符号位参加运算时，对应的补码也要以双符号位变形补码形式参加运算。

规格化浮点乘法运算举例

- 已知： $X=2^{010} \times 0.1011$， $Y=2^{100} \times (-0.1101)$

  求： $Z = X \times Y = ?$

- 解：写出X、Y的浮点机器数表示：

  阶码用 5 位移码（含2位符号位） 尾数用 5 位补码（含1位符号位）

  $[X]_{阶移尾补} = 01, 010; \ 0.1011$

  $[Y]_{阶移尾补} = 01, 100; \ 1.0011$

  (1) 阶码相加

  $[E_Z]_{移} = [E_x]_{移} + [E_y]_{补} = 01, 010 + 00, 100 = 01, 110$

  无溢出

规格化浮点乘法运算举例（续）

$[X]_{阶移尾补} = 01, 010;\ \ 0.1011$

$[Y]_{阶移尾补} = 01, 100;\ \ 1.0011$

（2）尾数相乘

$[M_z]_补 = [M_x]_补 \times [M_y]_补$

$= 000.1011 \times 11.00110$

$= 111.01110001$

（3）结果规格化

方法与加减法相同。

本例不需要规格化。

	部分积	乘数 $My_{n-1}My_nMy_{n+1}$	
	0 0 0. 0 0 0 0	1 1.0 0 1 1 0	
+	1 1 1. 0 1 0 1		$+[-M_x]_补$
	1 1 1. 0 1 0 1		
$\xrightarrow{2}$	1 1 1. 1 1 0 1	0 1 1 1.0 0 1	
+	0 0 0. 1 0 1 1		$+[M_x]_补$
	0 0 0. 1 0 0 0		
$\xrightarrow{2}$	0 0 0. 0 0 1 0	0 0 0 1 1 1. 0	
+	1 1 1. 0 1 0 1		$+[-M_x]_补$
	1 1 1. 0 1 1 1	0 0 0 1 0 0 0	
			清0

规格化浮点乘法运算举例（续）

$[E_Z]_{移} = 01, 110$

$[M_Z]_{补} = 111.01110001$

（4）舍入处理

方法与加减法相同。

本例采用0舍1入法，需舍。得

$[M_Z]_{补} = 1.0111$

（5）检查溢出

$[E_Z]_{移} = 01, 110$

移码最高符号位为0，本例不溢出。最后得

$[Z]_{阶移尾补} = 01, 110;\ 1.0111$

即 $Z = X \times Y = 2^{110} \times (-0.1001)$

规格化浮点除法运算举例（续）

- 已知：$X = 2^{-010} \times 0.1011$，$Y = 2^{100} \times (-0.1101)$

  求：$Q = X \div Y = ?$

- **解：阶码用5位移码　尾数用5位原码**

  （含2位符号位）（含1位符号位）

  则X、Y的机器数形式为

  $[X]_{阶移尾原} = 00, 110;\ 0.1011$

  $[Y]_{阶移尾原} = 01, 100;\ 1.1101$

（1）阶码相减

$[E_q]_{移} = [E_x]_{移} + [-E_y]_{补} = 00, 110 + 11, 100 = 00, 010$

无溢出

规格化浮点除法运算举例（续）

$[M_x]_{原} = 0.1011$

$[M_y]_{原} = 1.1101$

（2）尾数相除

$Q_s = (X_s \oplus Y_s)$

$= 0 \oplus 1 = 1$

$M_q^* = M_x^* \div M_y^*$

$= 0.1101$

则

$[M_q]_{原} = 1.1101$

注：也可以多求几位商，以供舍入。

被除数（余数）	商	说　明
0. 1 0 1 1	0. 0 0 0 0	
+ 1. 0 0 1 1		$+[-Y^*]_{补}$ (减除数)
1. 1 0 1 1		余数< 0, 商上0
← 1. 0 1 1 0	0. 0 0 0 0	左移一位
+ 0. 1 1 0 1		$+Y^*$
0. 1 0 0 1		余数> 0, 商上1
← 1. 0 0 1 0	0. 0 0 0 1	左移一位
+ 1. 0 0 1 1		$+[-Y^*]_{补}$
0. 0 1 0 1		余数> 0, 商上1
← 0. 1 0 1 0	0. 0 0 1 1	左移一位
+ 1. 0 0 1 1		$+[-Y^*]_{补}$
1. 1 1 0 1		余数< 0, 商上0
← 1. 1 0 1 0	0. 0 1 1 0	左移一位
+ 0. 1 1 0 1		$+Y^*$
0. 0 1 1 1		余数> 0, 商上1
	← 0. 1 1 0 1	商左移一位

规格化浮点除法运算举例（续）

$[Q]_{阶移尾原} = 00，010；\quad 1.1101$

（3）结果规格化：已是规格化数；

（4）舍入处理：不必舍入；

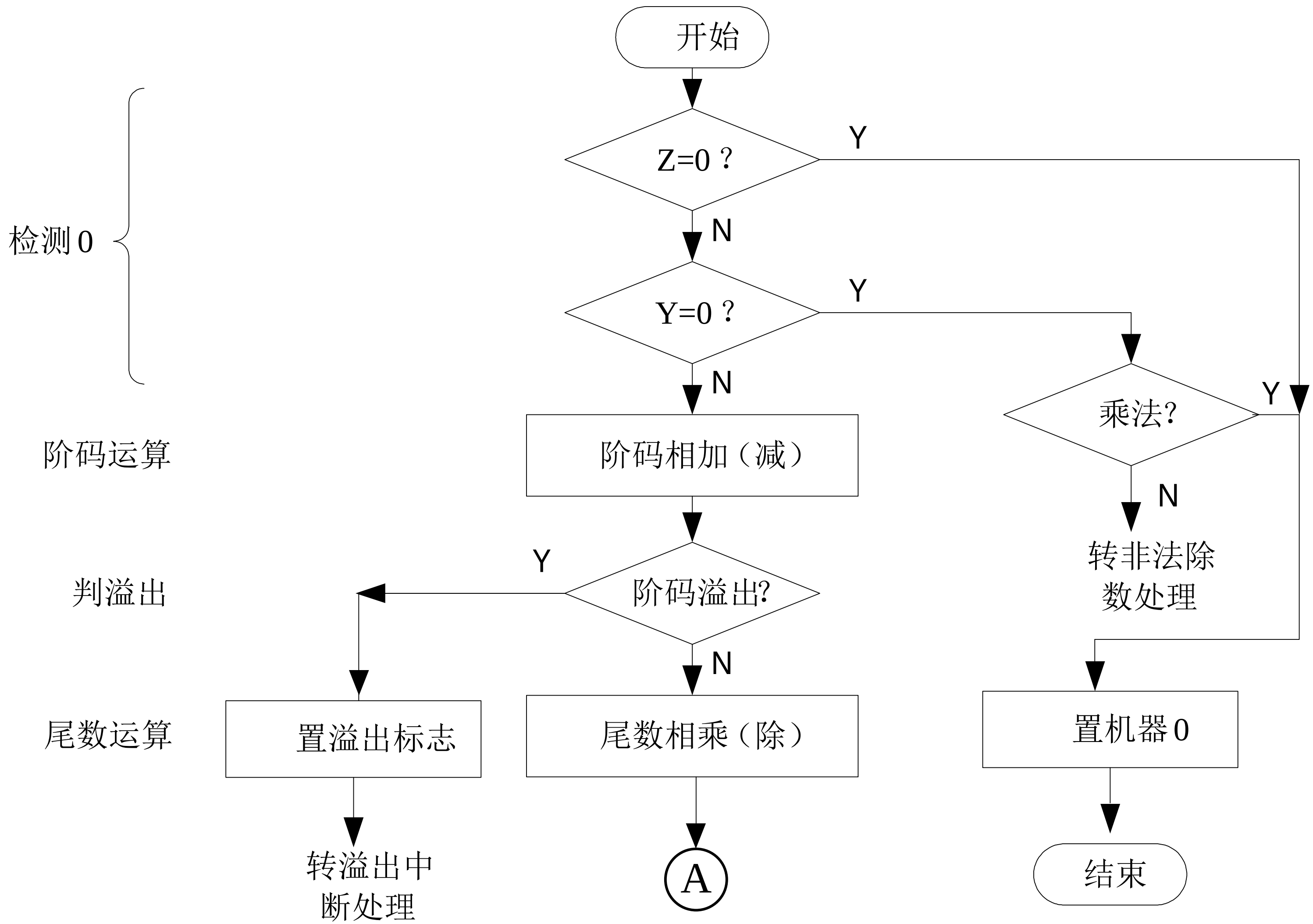
（5）检查溢出：不溢出。

则最终的商

$[Q]_{阶移尾原} = 00, 010；\ 1.\ 1101$

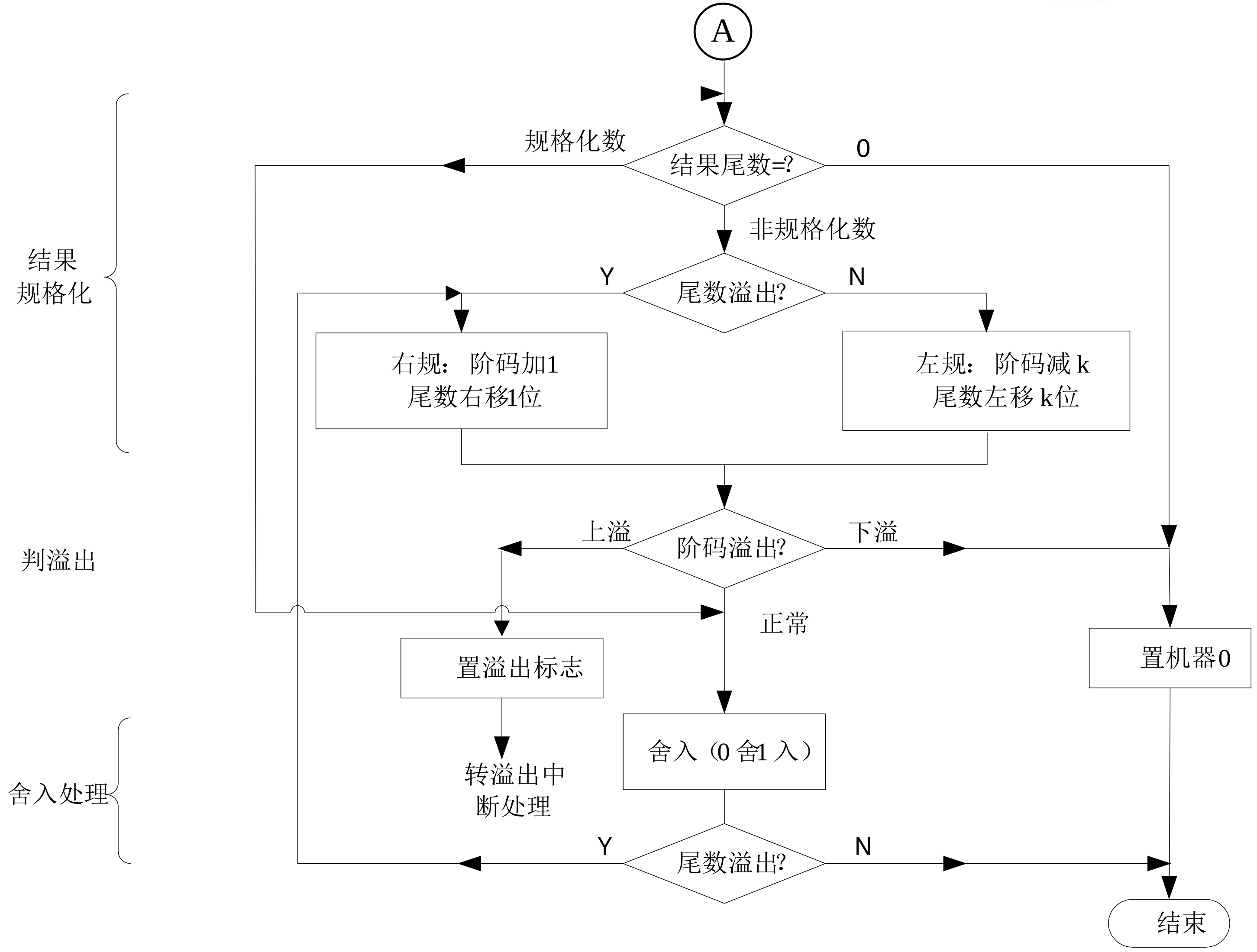
即 $X \div Y = 2^{-110} \times (-0.1101)$

浮点乘除运算操作流程



开始
Z=0 ?
Y
N
Y=0 ?
Y
N
检测 0
阶码运算
阶码相加（减）
乘法?
Y
N
转非法除数处理
判溢出
阶码溢出?
Y
N
尾数运算
置溢出标志
尾数相乘（除）
置机器 0
转溢出中断处理
A
结束

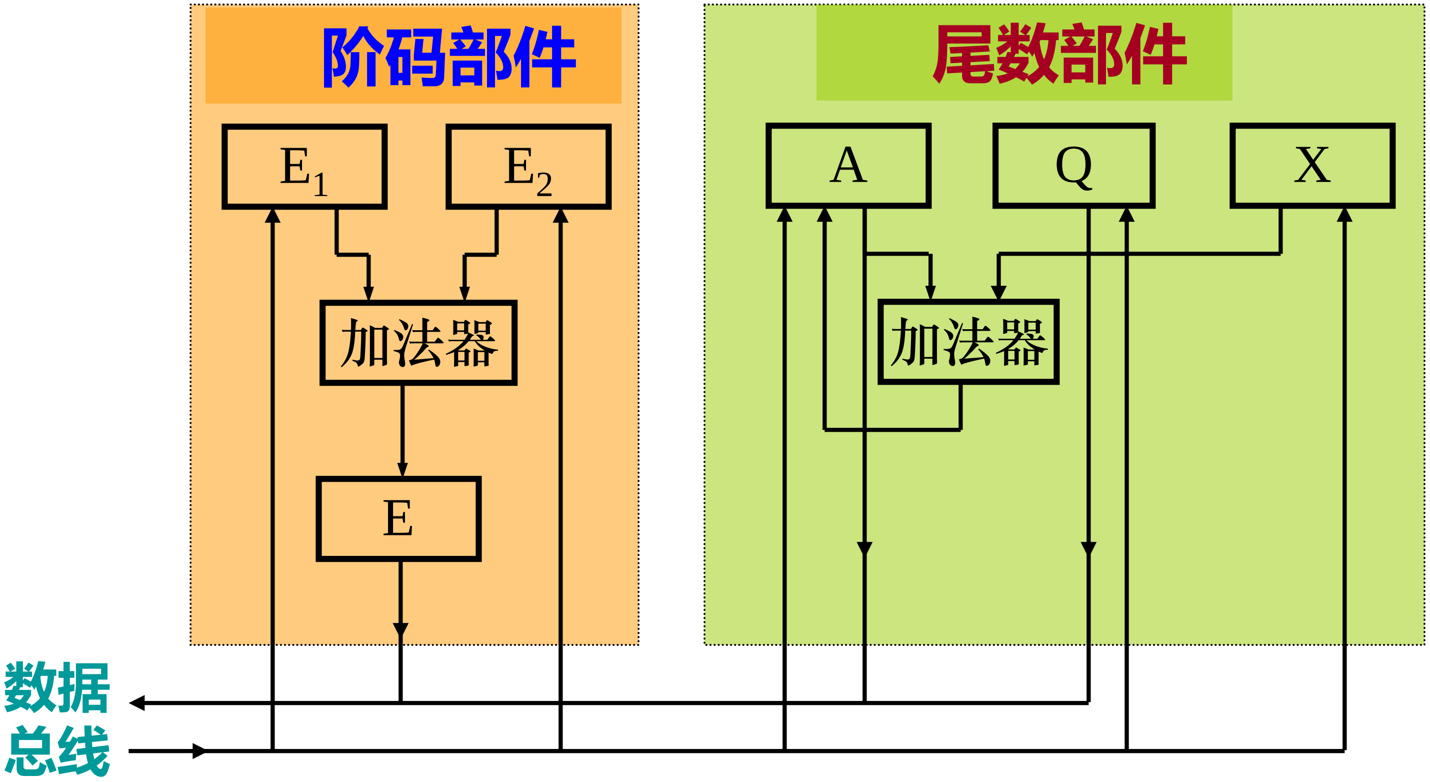
浮点加减运算操作流程（续）



6.5.7 浮点运算的实现

浮点运算器的基本配置

浮点运算器可看成由处理阶码的阶码部件（加/减）和处理尾数的尾数部件（四则运算）两个定点运算器松散组成。



本章内容总结

应掌握的主要内容： 数的机内表示　运算方法　先行进位技术

1. 数据在计算机内的表示

① 二进制数的原、反、补、移码表示

② 浮点数的机内表示

③ 各类机内数据的表示范围

2. 数值数据算术运算的实现算法

① 定点数的算术运算

- **补码加、减法运算**
- **原码一位乘法**
- **补码一位乘法、二位乘法**
- **原码加减交替除法**

本章内容总结

② 浮点数的算术运算 {
- 规格化浮点数加、减法运算
- 规格化浮点数乘、除法运算

③ 移位运算：算术移位、逻辑移位；舍入方法

3. 逻辑运算

4. 先行进位技术

① 行波进位加法器

② 先行进位加法器

③ 成组先行-级联进位

④ 多级先行进位

⑤ 进位延迟时间计算