西安电子科技大学
XIDIAN UNIVERSITY

第三章 运算方法与运算器

主讲：张骏鹏（博士，副教授）

西安电子科技大学

人工智能学院

1. 定点数运算
2. 浮点数运算
3. 运算器的基本结构

3.1 定点数运算

3.1.2 乘法运算

在一些简单的计算机中,乘法运算可以用软件来实现。利用计算机中设置的加法、移 位等指令,编写一段程序完成两数相乘。若 CPU 硬件结构简单,则这种做法实现乘法所用的时间较长,速度很慢。

另一种情况是在 ALU 等硬件的基础上,适当增加一些硬件构成乘法器。这种乘法器 的硬件要复杂一些,但速度比较快。速度最快的是全部由硬件实现的阵列乘法器,其硬件 更加复杂。可见,可以用硬件来换取速度。

1.原码乘法运算

1)原码一位乘法规则 假定被乘数 X、乘数Y 和乘积Z 为用原码表示的纯小数(下面的讨论同样适用于纯整数),分别为

$$[X]_{原}=x_0.x_{-1}\ x_{-2}\cdots x_{-(n-1)}$$

$$[Y]_{原}=y_0.y_{-1}\ y_{-2}\cdots y_{-(n-1)}$$

$$[Z]_{原}=z_0.z_{-1}\ z_{-2}\cdots z_{-(2n-1)}$$

其中，x_0、y_0、z_0是它们的符号位。特别提醒：小数点在编码中是隐含的，且小数点的位置是默认的，此处显示小数点仅为提示作用。本章其他处若小数编码中出现小数点，其作用也是如此。

原码一位乘法规则如下:

(1)乘积的符号为被乘数的符号位与乘数的符号位相异或。

(2)乘积的数值为被乘数的数值与乘数的数值之积,即

$$|Z|=|X\times Y|=|X|\times|Y| \tag{3.18}$$

(3)乘积的原码为

$$[Z]_{原}=[X\times Y]_{原}=(x_0\oplus y_0)(|X|\times|Y|) \tag{3.19}$$

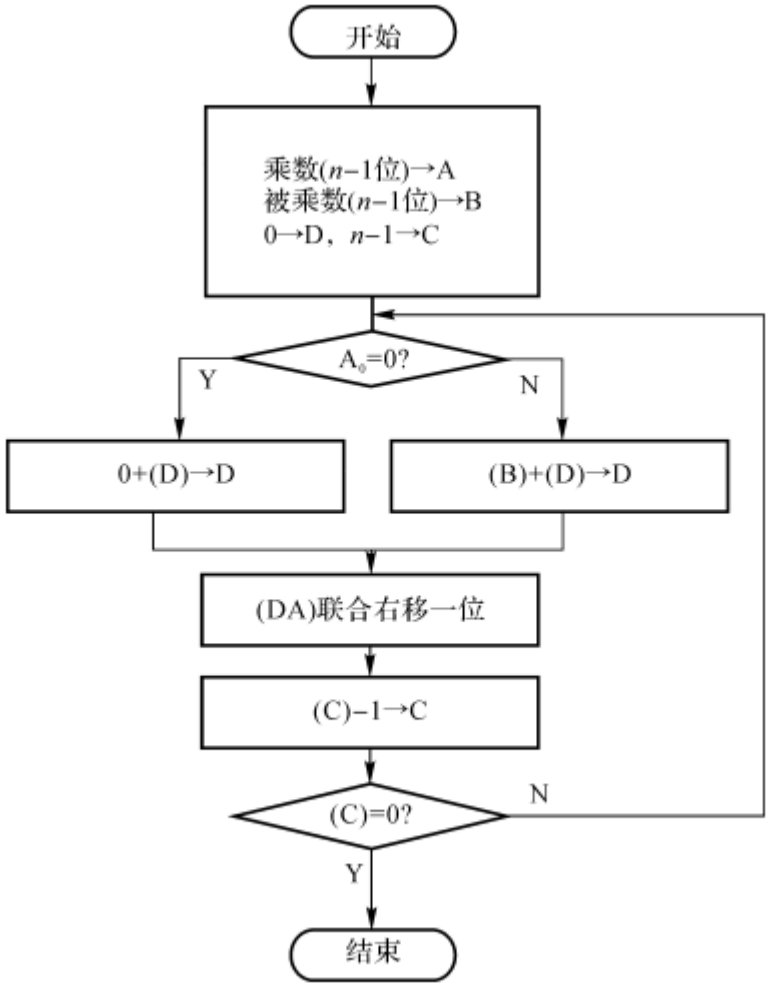


图3.10 数值乘法算法流程

	D				A				A_0	操作
0	0	0	0	0	1	0	1	1		A_0=1,+X
+0	1	1	0	1						
0	1	1	0	1						
0	0	1	1	0	1	1	0	1		右移一位
+0	1	1	0	1						A_0=1,+X
1	0	0	1	1						
0	1	0	0	1	1	1	1	0		右移一位
0	0	0	0	0						A_0=0,+0
0	1	0	0	1						
0	0	1	0	0	1	1	1	1		右移一位
+0	1	1	0	1						A_0=1,+X
1	0	0	0	1						
0	1	0	0	0	1	1	1	1		→右移一位

拼接符号后积为：$[XY]_{原}$=1.10001111。

图3.11 例3.11的乘法过程

4)原码一位乘法器的框图

根据以上对原码一位乘法的描述,可以设计出采用原码一位乘法的乘法器,如图3.12 所示。

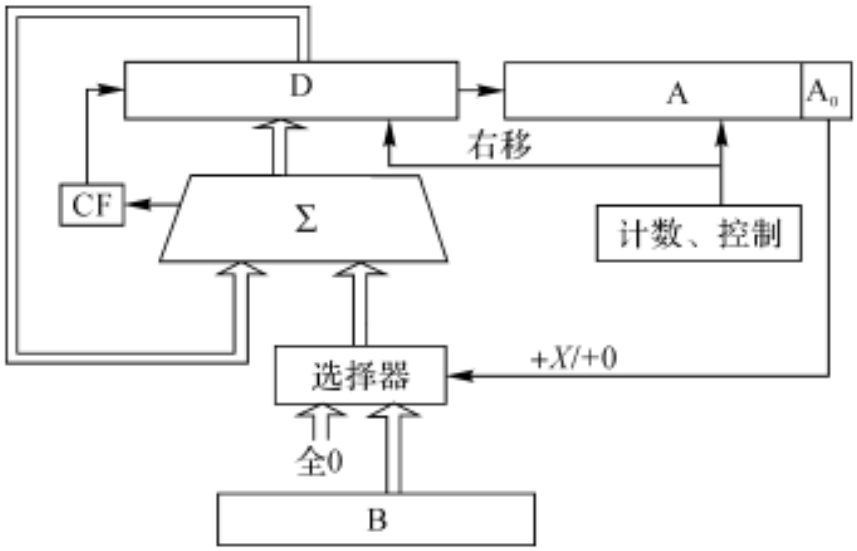


图3.12 原码一位乘法器的框图

2.补码乘法运算

计算机中经常采用补码表示数据,这时用原码进行乘法运算很不方便,因此,较多计算机采取补码进行乘法运算。一种经典的补码乘法算法为布斯法,它是补码一位乘法中的 一种,是由布斯(Booth)夫妇提出的。

$$[X]_{补}=x_0.\,x_{-1}x_{-2}\cdots x_{-(n-1)}$$
$$[Y]_{补}=y_0.\,y_{-1}y_{-2}\cdots y_{-(n-1)}$$

$$[X\times Y]_{补}=[X]_{补}\times[(y_{-1}-y_0)\times 2^0+(y_{-2}-y_{-1})\times 2^{-1}+\cdots+(y_{-(n-1)}-y_{-(n-2)})\times 2^{-(n-2)}+(0-y_{-(n-1)})\times 2^{-(n-1)}] \quad (3.20)$$

表 3.1　以乘数相邻两位为依据的布斯算法运算过程中的操作

y_i	y_{i-1}	$y_{i-1}-y_i$	操　　作
0	0	0	+0，右移一位
0	1	1	$+[X]_{补}$，右移一位
1	0	−1	$+[-X]_{补}$，右移一位
1	1	0	+0，右移一位

根据以上分析,可将布斯算法描述如下:

(1)乘数与被乘数均用补码表示,连同符号位一起参加运算。

(2)乘数最低位后增加一个附加位(用 A-1表示),设定初始值为0。

(3)从附加位开始,依据表3.1所示的操作完成式(3.20)的运算。 实现布斯算法的流程如图3.13所示。

例3.12 已知二进制数 X=0.1010,Y=-0.1101。利用布斯算法求[XY]补 。

解 (1)将两数用补码表示为

$$[X]_{补}=00.1010,\ [Y]_{补}=11.0011,\ [-X]_{补}=11.0110$$

(2)图3.14给出了布斯算法求解过程。由图3.14可知,$[X\cdot Y]_{补}$ =1.011111100。

符号	D	A	A_{-1}	操作
0 0	0 0 0 0	1 0 0 1 1	0	$+[-X]_{补}$
1 1	0 1 1 0			
1 1	0 1 1 0			
1 1	1 0 1 1	0 1 0 0 1	1	右移一位
0 0	0 0 0 0			+0
1 1	1 0 1 1			
1 1	1 1 0 1	1 0 1 0 0	1	右移一位
0 0	1 0 1 0			$+[X]_{补}$
0 0	0 1 1 1			
0 0	0 0 1 1	1 1 0 1 0	0	右移一位
0 0	0 0 0 0			+0
0 0	0 0 1 1			
0 0	0 0 0 1	1 1 1 0 1	0	右移一位
1 1	0 1 1 0			$+[-X]_{补}$
1 1	0 1 1 1			
1 1	1 0 1 1	1 1 1 1 0		右移一位

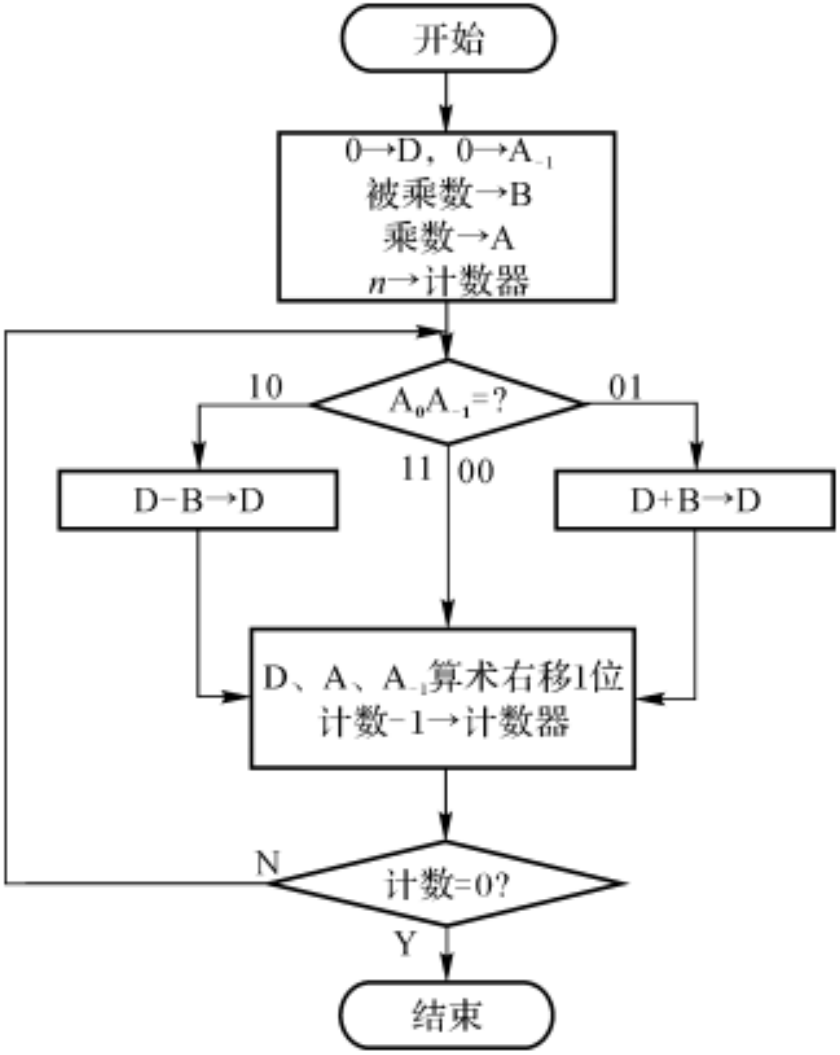


图3.13 布斯算法流程图

2)无符号数阵列乘法器

利用手算算式的结构及乘加单元电路可以方便地实现无符号数阵列乘法器,其结构如 图3.17所示。

在图3.17中,每一个小框即为一个基本乘加单元,这些基本单元按照类似于手算算式 的结构进行连接,能够完成手算算式中的乘加功能,最终获得两数的乘积。

利用无符号数阵列乘法器完成原码的数值相乘,再加入一个完成符号运算的异或门, 就构成了原码阵列乘法器。

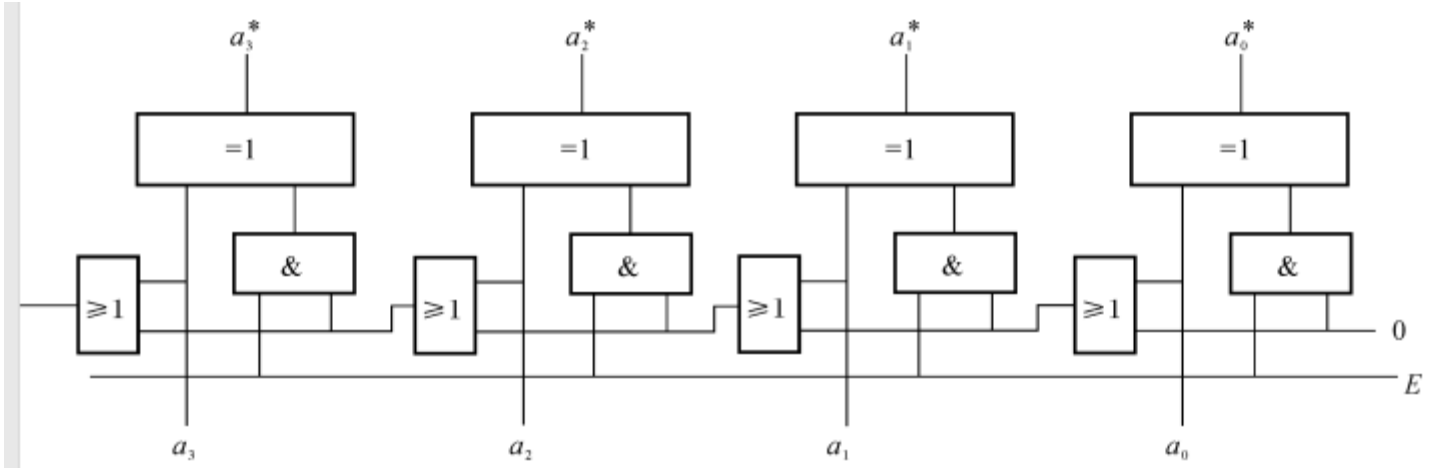
设二进制数 $X=X_3X_2X_1X_0$和$Y=Y_3Y_2Y_1Y_0$,计算$Z=XY$,列式如下:

$$
\begin{array}{ccccccccc}
 & & & & X_3 & X_2 & X_1 & X_0 & \\
 & & & \times & Y_3 & Y_2 & Y_1 & Y_0 & \\
\hline
 & & & & X_3Y_0 & X_2Y_0 & X_1Y_0 & X_0Y_0 & \\
 & & & X_3Y_1 & X_2Y_1 & X_1Y_1 & X_0Y_1 & & \\
 & & X_3Y_2 & X_2Y_2 & X_1Y_2 & X_0Y_2 & & & \\
 & X_3Y_3 & X_2Y_3 & X_1Y_3 & X_0Y_3 & & & & \\
\hline
Z_6 & Z_5 & Z_4 & Z_3 & Z_2 & Z_1 & Z_0 &
\end{array}
$$

3)补码阵列乘法器

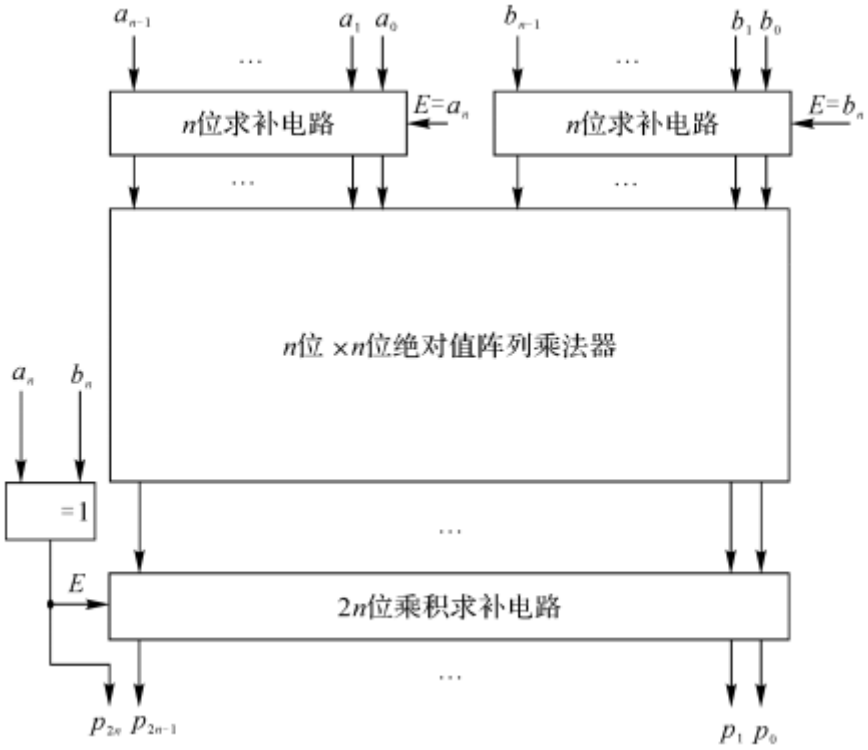
在无符号数阵列乘法器的基础上,很容易实现补码阵列乘法器。其基本思路是先求被乘数与乘数的绝对值(无符号数),然后进行无符号数阵列乘法,最后根据被乘数与乘数的符号决定最终乘积的符号。

为了实现补码阵列乘法器,先给出一个简单的求补电路,如图3.18所示。从图3.18中 可见,当控制端E=0时,输出与输入相同;当E=1时,可实现求补。只要将有符号数的 符号位加到控制端E 上,即可求得该符号数的绝对值。


a_3^*
a_2^*
a_1^*
a_0^*
=1
=1
=1
=1
&
&
&
&
≥1
≥1
≥1
≥1
0
E
a_3
a_2
a_1
a_0

4)适于流水线工作的阵列乘法器

图3.17所示阵列乘法器的最大缺点是:每一步部分积的计算都是用串行进位加法器来 实现的,因此即使采用硬件电路,其运算速度仍然很慢,令乘法器的使用者无法接受。为了提高阵列乘法器的速度,设计者做了大量的研究,其中包括设计了适于流水线工作的阵列 乘法器,具体内容详见7.2.2节。


a_{n-1}
a_1 a_0
b_{n-1}
b_1 b_0
n位求补电路
$E=a_n$
n位求补电路
$E=b_n$
n位 ×n位绝对值阵列乘法器
a_n
b_n
=1
E
2n位乘积求补电路
p_{2n} p_{2n-1}
p_1 p_0

3.1.3 除法运算

定点除法运算同样可用原码或补码实现。在实现除法的过程中,应注意除数不能为0, 而且还要保证相除所得的商是可以表示的。

1.原码除法运算

1)原码除法规则 原码除法运算规则如下:

(1)除数≠0。对于定点纯小数,|被除数|<|除数|;对于定点纯整数,|被除数|≥|除数|。

(2)与原码乘法类似,原码除法的商符和商值也是分别处理的。商符等于被除数的符 号与除数的符号相异或。商值等于被除数的数值除以除数的数值。

(3)将商符与商值拼接在一起即可得到商的原码。

例3.13 设二进制数 X=+0.1011,Y=+0.1101,求 X÷Y。

解 被除数 X 和除数Y 均为正数,则商的符号也为正。两数数值部分的除法手算过程 如图3.20所示。

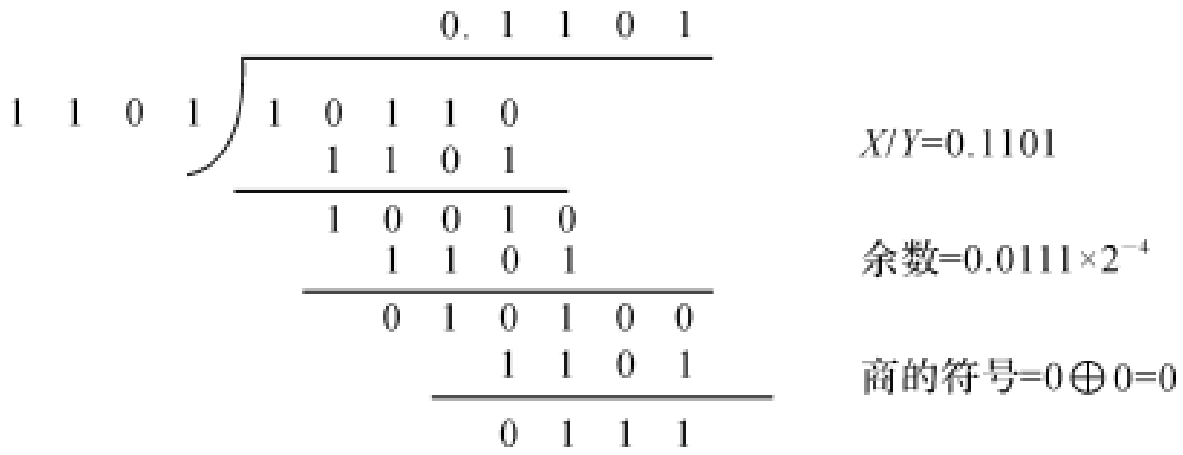


图3.20 例3.13的除法手算过程

2)恢复余数法

利用恢复余数法实现原码除法遵从上述除法规则,数值和符号单独处理。

图3.21为原码恢复余数算法流程。对于定点纯小数的数值部分,计算过程如下:

(1)被除数左移一位,减除数,若够减,上商为 1,若不够减,上商为 0,同时加除 数———恢复余数。

(2)余数左移一位,减除数,若够减,上商为1,若不够减,上商为0,同时加除数——— 恢复余数。重复此过程,直到除尽或精度达到要求为止。

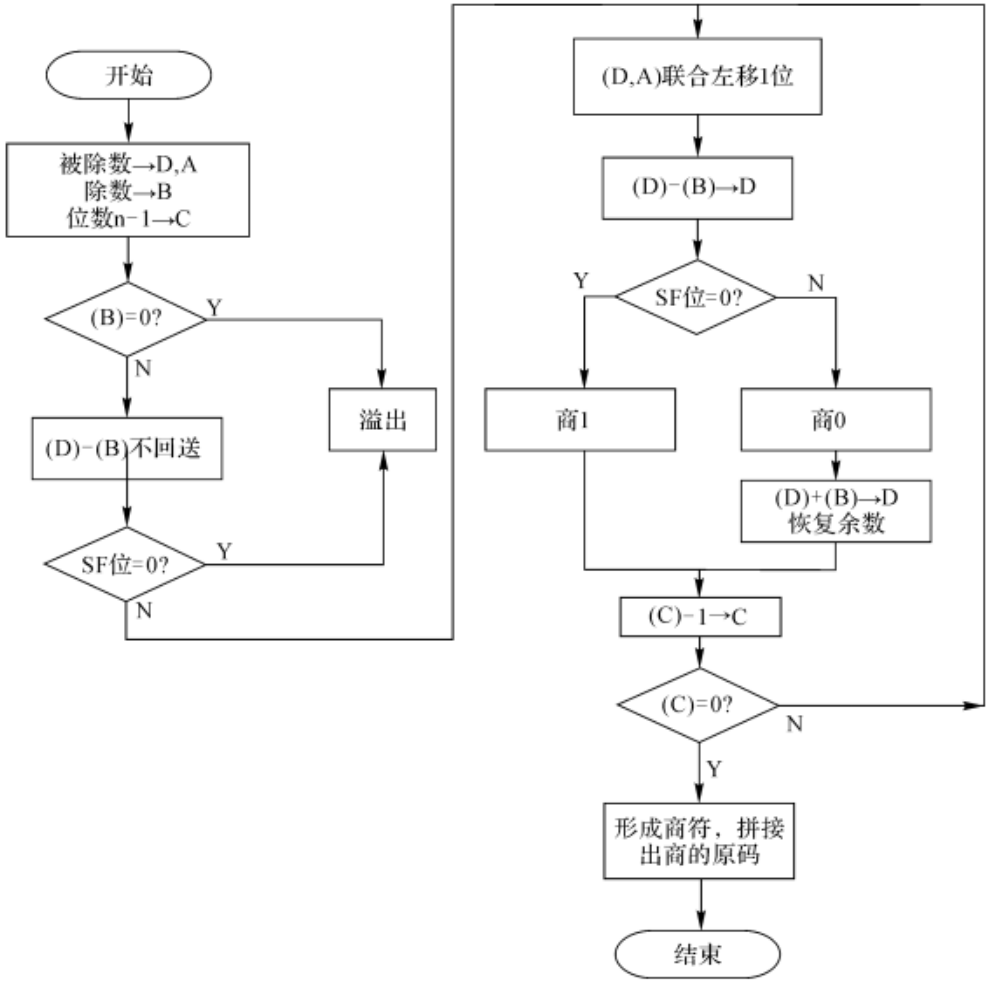


图3.21 原码恢复余数算法流程图

例3.14 若二进制被除数X=-0.10001011,除数Y=0.1110,试利用原码恢复余数法 求 X÷Y 的商及余数。

解 本例满足|X|<|Y|,且|Y|≠0。 对 X 和Y 编码,得

$$[X]_{原}=1.10001011$$
$$[Y]_{原}=0.1110$$

商符=1⊕0=1。数值除法过程如图3.22所示。

被除数(余数)										操作
符号		D				A				
0	0	1	0	0	0	1	0	1	1	
0	1	0	0	0	1	0	1	1	**0**	左移一位
1	1	0	0	1	0					−|Y|
0	0	0	0	1	1	0	1	1	**1**	够减，商为1
0	0	0	1	1	0	1	1	**1**	0	左移一位
1	1	0	0	1	0					−|Y|
1	1	1	0	0	0	1	1	**1**	**0**	不够减，商为0
0	0	1	1	1	0					+|Y|
0	0	0	1	1	0	1	1	**1**	**0**	恢复余数
0	0	1	1	0	1	1	**1**	**0**	0	左移一位
1	1	0	0	1	0					−|Y|
1	1	1	1	1	1	1	**1**	**0**	**0**	不够减，商为0
0	0	1	1	1	0					+|Y|
0	0	1	1	0	1	1	**1**	**0**	**0**	恢复余数
0	1	1	0	1	1	**1**	**0**	**0**	0	左移一位
1	1	0	0	1	0					−|Y|
0	0	1	1	0	1	**1**	**0**	**0**	**1**	够减，商为1

图3.22 例3.14恢复余数法实现数值除法的过程

3）加减交替法

（1）加减交替法。为了说明加减交替法的原理，下面回顾一下恢复余数法。假定第 i 次余数减除数（用 B 表示）得到当前余数 R_i，当 $R_i<0$ 时，应恢复余数，即 R_i+B。然后左移一位，即 $2(R_i+B)$。接下来进行下一次（第 $i+1$ 次）余数减除数，即 $R_{i+1}=2(R_i+B)-B=2R_i+B$。

也就是说，若第 i 次余数减除数得到当前余数 $R_i<0$，则不再需要立即加除数来恢复余数，而是将其左移一位，变为 $2R_i$，到第 $i+1$ 次余数运算时加除数，即 $R_{i+1}=2R_i+B$。因此，加减交替算法可描述如下：

① 若余数 $R\geqslant 0$，则商上 1，余数左移一位，减除数；

② 若余数 $R<0$，则商上 0，余数左移一位，加除数。

例 3.15 若二进制数 $X=-0.10001011$，$Y=0.1110$，试利用原码加减交替法求 $X\div Y$ 的商及余数。

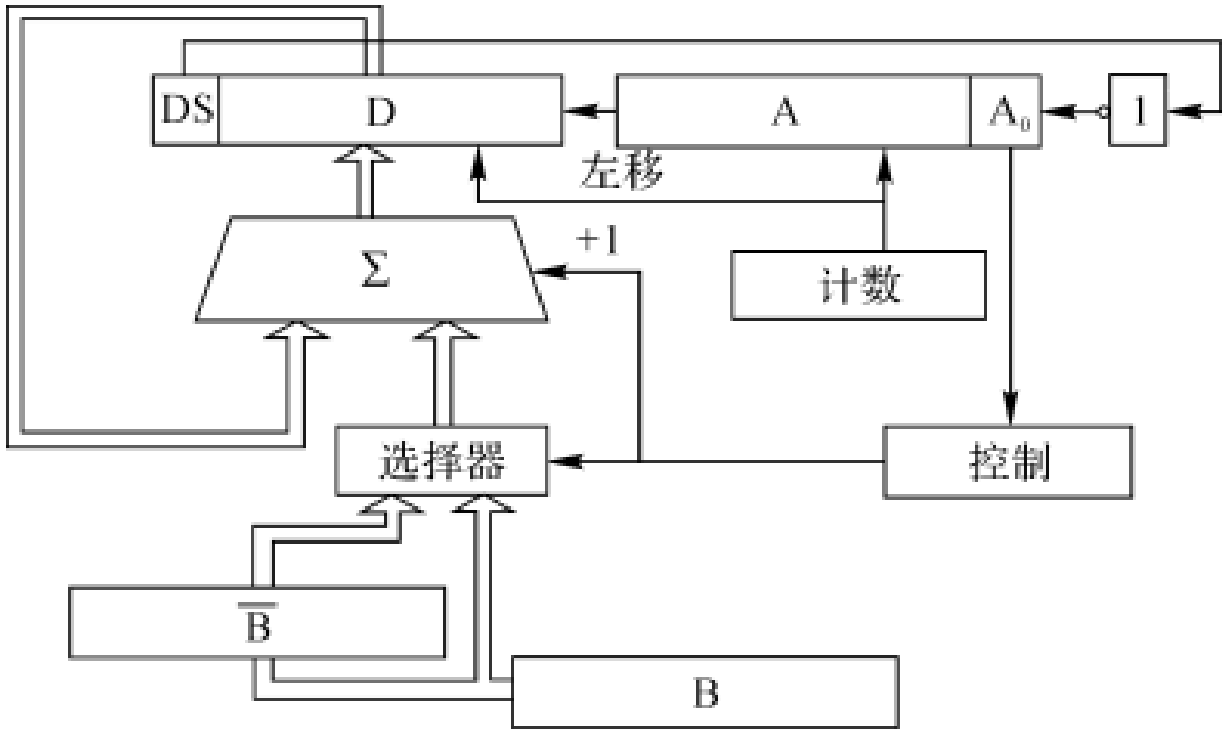
解 $[X]_{原}=1.10001011$，$[Y]_{原}=0.1110$，商符 $=1\oplus 0=1$。数值除法过程如图 3.23 所示。

被除数(余数)										操作
符号		D				A				
0	0	1	0	0	0	1	0	1	1	
0	1	0	0	0	1	0	1	1	0	左移一位
1	1	0	0	1	0					$-\|Y\|$
0	0	0	0	1	1	0	1	1	**1**	$R\geqslant 0$，商为1
0	0	0	1	1	0	1	1	**1**	0	左移一位
1	1	0	0	1	0					$-\|Y\|$
1	1	1	0	0	0	1	1	**1**	**0**	$R<0$，商为0
1	1	0	0	0	1	1	**1**	**0**	0	左移一位
0	0	1	1	1	0					$+\|Y\|$
1	1	1	1	1	1	1	**1**	**0**	**0**	$R<0$，商为0
1	1	1	1	1	1	**1**	**0**	**0**	0	左移一位
0	0	1	1	1	0					$+\|Y\|$
0	0	1	1	0	1	**1**	**0**	**0**	1	$R\geqslant 1$，商为1

图3.23 例3.15加减交替法实现除法的过程

(2)加减交替除法器硬件电路。

原码加减交替法作除法时符号与数值运算是分别进行的。图3.24给出了数值部分(无 符号数)除法的硬件框图。



2.补码除法运算

与乘法运算的情况类似,有时也需要实现补码除法。

1)补码除法规则

假设进行定点纯小数的补码除法运算,其先决条件是除数≠0且|被除数|<|除数|。 补码除法运算相对要复杂一些,其运算规则如下:

(1)如果被除数与除数同号,则被除数减除数;如果被除数与除数异号,则被除数加除 数。运算结果均称为余数。

(2)若余数与除数同号,则上商为1,余数左移一位,然后用余数减除数得新余数;若 余数与除数异号,则上商为0,余数左移一位,然后用余数加除数得新余数。

(3)重复(2),直至除尽或达到精度要求为止。

(4)修正商。在除不尽时,通常将商的最低位恒置1进行修正来保证精度。

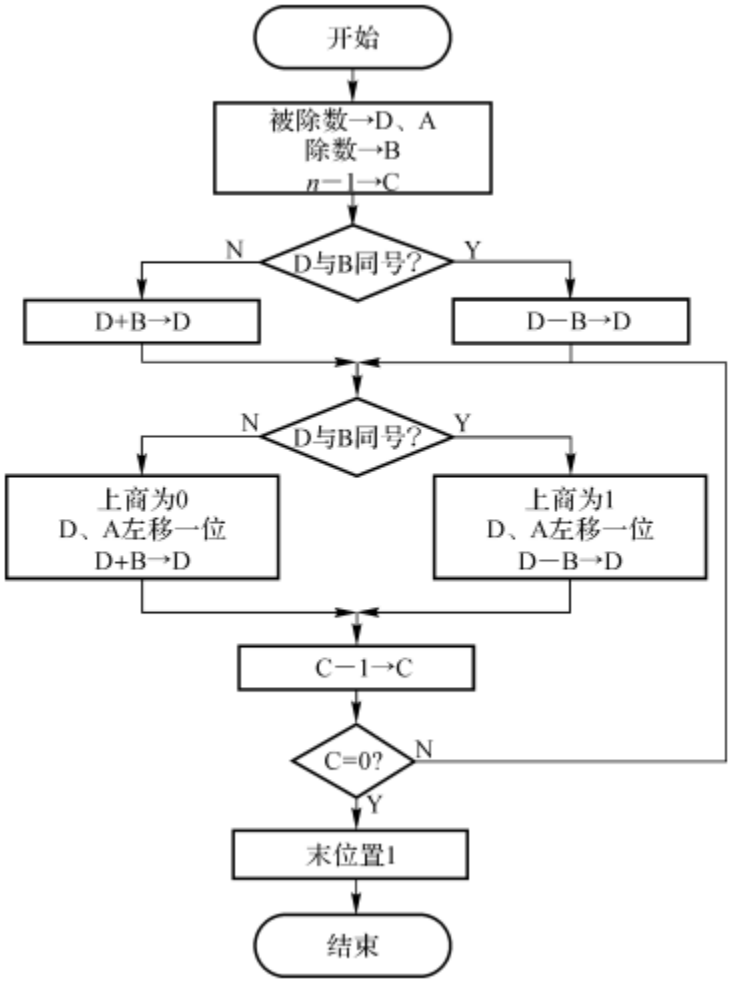


图3.25 补码除法算法流程框图

3.阵列除法器

前面所提到的除法器都是在加法器的基础上通过多次加减来实现除法的,其运算速度 必然受到限制。为了提高速度,可以利用专用硬件来完成除法运算。

1)补码进位及阵列基本单元

(1)补码运算的进位。 在无符号数进行减法运算时,是用被减数加上负减数的补码来实现的。而补码运算的 进位会出现如下情况:当被减数小而减数大时,没有进位□ 或者说没有借位);当被减数大 而减数小时,反而有进位□ 或者说有借位)。

例 3.16 分析十进制无符号数 65－32 和 32－65 采用补码运算时的进位情况。

解 无符号数 65 和 32 用 8 位二进制数表示，分别为

$$65=(01000001)_2$$

$$32=(00100000)_2$$

在进行减法运算时，可用加负数补码的方法来实现，它们的负数补码为

$$[-65]_{补}=10111111$$

$$[-32]_{补}=11100000$$

例 3.16 分析十进制无符号数 65－32 和 32－65 采用补码运算时的进位情况。

解 无符号数 65 和 32 用 8 位二进制数表示，分别为

$$65=(01000001)_2$$

$$32=(00100000)_2$$

在进行减法运算时，可用加负数补码的方法来实现，它们的负数补码为

$$[-65]_{补}=10111111$$

$$[-32]_{补}=11100000$$

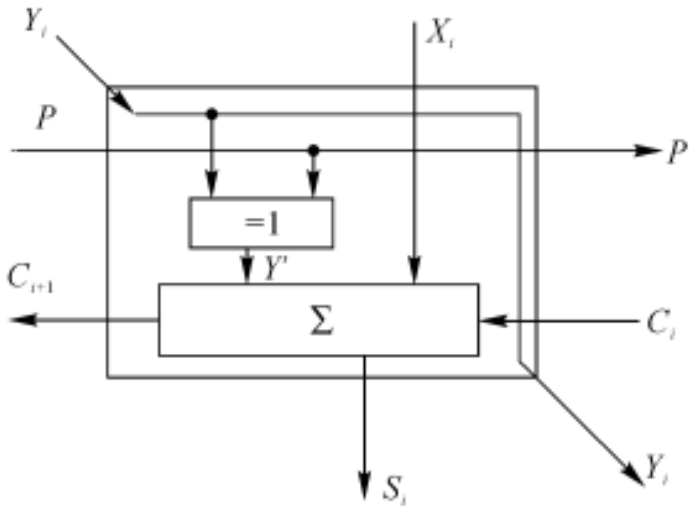
65－32 及 32－65 的运算过程如下：

```
    0 1 0 0 0 0 0 1          0 0 1 0 0 0 0 0
  + 1 1 1 0 0 0 0 0        + 1 0 1 1 1 1 1 1
  -----------------        -----------------
  [1]0 0 1 0 0 0 0 1       [0]1 1 0 1 1 1 1 1
    ↗                        ↗
   进位                     进位
```

从上述运算可见，够减时进位为 1(其真值运算没有借位)，不够减时进位为 0(其真值运算有借位)。

(2) 可控加减单元 CAS。

可控加减单元 CAS 如图 3.26 所示，它主要由两部分硬件组成。一是异或电路，在外部信号 P 的控制下，当 $P=0$ 时，其输出 $Y'=Y_i$；当 $P=1$ 时，其输出 $Y'=\overline{Y_i}$（将输入 Y_i 取反）。在下述阵列除法器中，Y 作为除数，若将其取反再加 1，即可对其求补。二是一位全加器，实现输入 Y'、X_i 及低级进位 C_i 的全加。



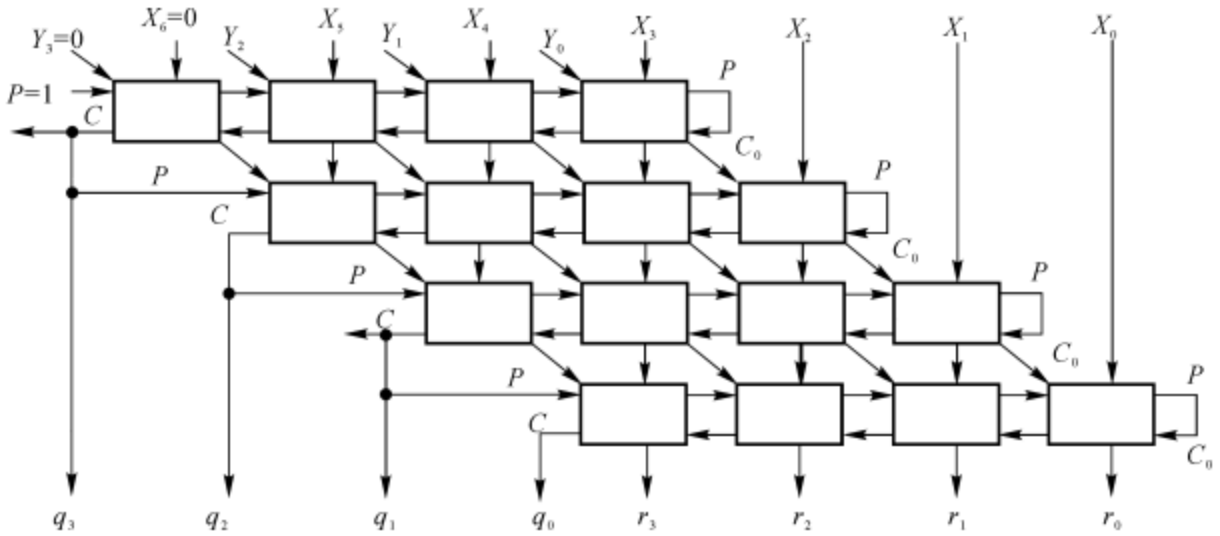
2）无符号数阵列除法器

无符号数阵列除法器如图 3.27 所示。被除数加在除法器的 $X_6 \sim X_0$ 端，并使最高位 $X_6=0$ 以保证结果正确；除数加在 $Y_3 \sim Y_0$ 端，且使 $Y_3=0$。

阵列第一行加载 $P=1$，保证将除数取反，$P=1$ 又加在第一行最后一个 CAS 的进位输入端 C_0，从而实现了对除数的求补，因此，阵列的第一行实现了被除数减除数。若够减，则进位 C 为 1，商 q_3 应为 1，即 $q_3=C=1$；若不够减，则进位 C 为 0，商 q_3 应为 0，故 $q_3=C=0$。

阵列除第一行外其他各行是除数右移一位(相当于余数左移一位)，再根据上一行运算余数的正负性来做如下操作：若上行相减结果够减，进位 C 为 1(上商为 1)，则使本行 $P=1$，做减法，余数减除数；若不够减，进位 C 为 0(上商为 0)，则使本行 $P=0$，做加法，余数加除数。加或减产生的进位 C 即为商 q。如此各行依次运算，便完成了原码数值(无符号数)加减交替除法运算。

利用图 3.27 所示的阵列除法器，得到的商为 $q_3q_2q_1q_0$，余数为 $r_3r_2r_1r_0$。注意，若是纯小数运算，应满足 $|X|<|Y|$。


Y_3=0
X_6=0
Y_2
X_5
Y_1
X_4
Y_0
X_3
X_2
X_1
X_0
P=1
C
P
C_0
q_3
q_2
q_1
q_0
r_3
r_2
r_1
r_0

3.2 逻辑与移位运算

3.2.1 逻辑运算

基本的逻辑运算包括与、或、非、异或等运算。计算机以“1”和“0”分别表示逻辑数据的 真和假两个状态。此时n 个0和1的数字组合不是算术数字,而是没有符号位的逻辑数据。 逻辑运算按位进行操作,各位之间互不影响,运算结果没有进位、借位、溢出等问题。

1.基本逻辑运算

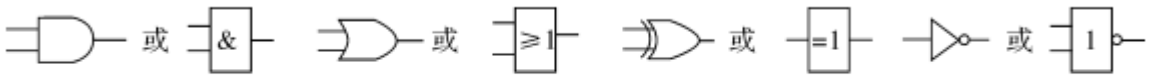
计算机中采用的基本逻辑运算列于表3.2中。

表 3.2　基本逻辑运算

X_i	Y_i	$X_i \cdot Y_i$	$X_i + Y_i$	$X_i \oplus Y_i$	$\overline{X_i}$
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

2.逻辑运算部件

实现逻辑与、或、异或、非运算的部件非常简单,分别为与门、或门、异或门和反向器, 如图3.28所示。将多个同类门集合在一起,就可以构成对n 位逻辑数据的与门、或门、异 或门和反向器。利用这些基本的门电路,可以实现复杂的组合逻辑,如图3.1中的一位全加 器、图3.18中的求补电路等。

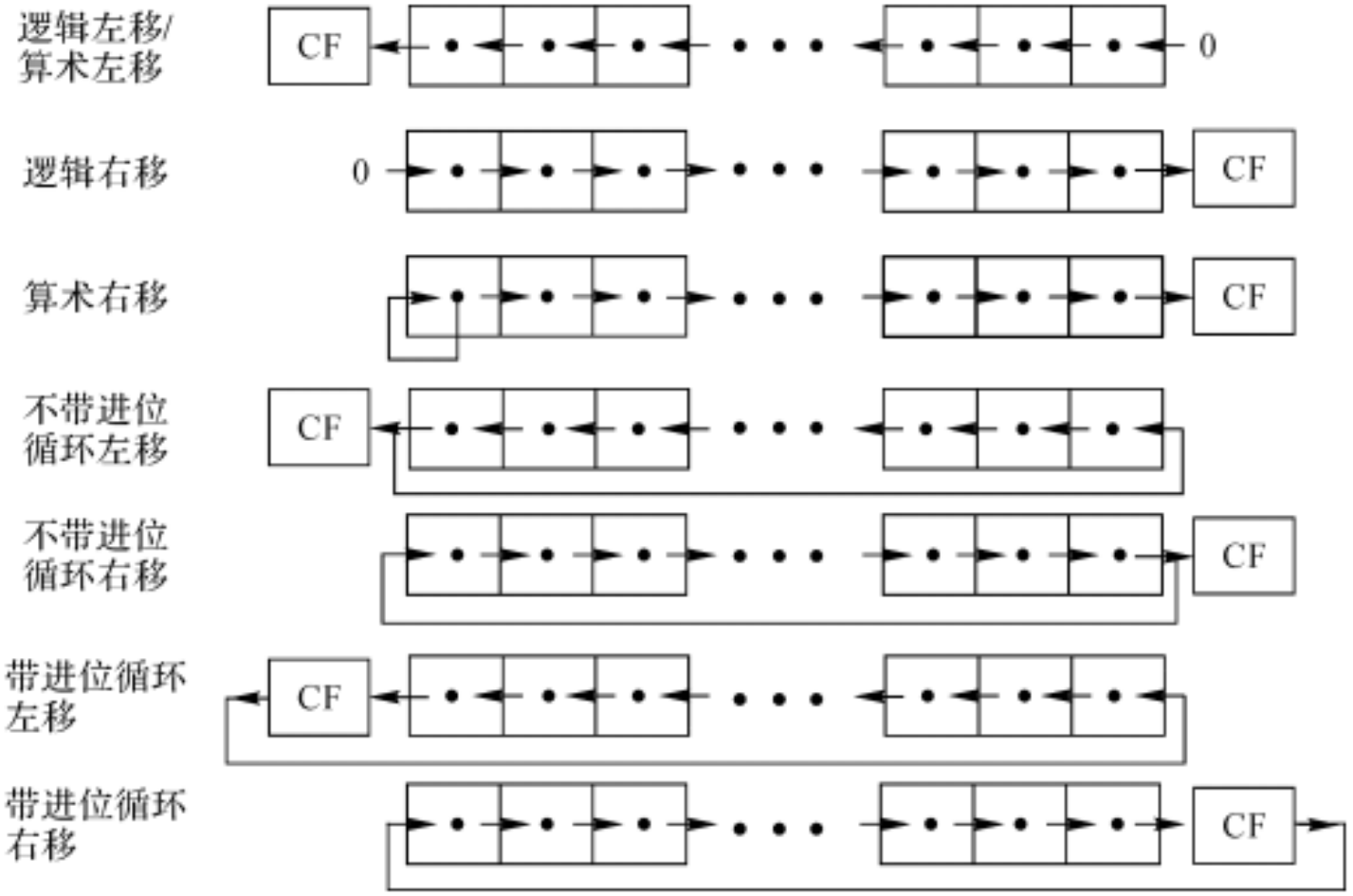


(a) 与门　(b) 或门　(c) 异或门　(d) 反向器

3. 2. 2 移位运算

对于无限长度二进制数,左移或者右移n 位相当于该数乘以或者除以2n 。由于计算机 的机器数字长是固定的,因此当机器数左移或右移n 位时,必然会使数据的低位或者高位 出现n 个空位。这些空位填写“0”还是“1”,取决于机器数采用的是无符号数还是有符号数。

在计算机中,常见的移位运算包括逻辑移位、算术移位、不带进位循环移位和带进位 循环移位,具体运算过程如图3.29所示。


逻辑左移/
算术左移
CF
0
逻辑右移
0
CF
算术右移
CF
不带进位
循环左移
CF
不带进位
循环右移
CF
带进位循环
左移
CF
带进位循环
右移
CF

3.3 浮点数运算

3.3.1 浮点加减运算

设两个规格化浮点数 $X=Mx \times 2^{Ex}$ 和$Y=My \times 2^{Ey}$,实现 $X \pm Y$ 运算的规则如下所述。

1.对阶

一般情况下,两浮点数的阶码不会相同。也就是说,两数的小数点没有对齐。和我们所 熟悉的十进制小数加减一样,在进行浮点数加减运算前需将小数点对齐,这称为对阶。只有当两数的阶码相同时才能进行尾数的加减运算。

对阶的原则是小阶对大阶,也就是将小阶码浮点数的阶码变成大阶码浮点数的阶码。 具体做法是:小阶码每增加1,该浮点数的尾数右移一位,直到小阶码增大到与大阶码相 同。这样在对阶时丢失的是尾数的低位,造成的误差很小。若是大阶对小阶,将丢失尾数的 高位,从而导致错误的结果。

2.尾数加(减)运算

对阶之后,尾数即可进行加(减)运算。实际运算中只需做加法即可,因为减法可以用 加法来实现。

3.规格化

尾数加减运算后,其结果有可能是一个非规格化数。如果结果的真值 M 不满足1/2≤ |M|<1,则该结果是非规格化数,需要进行规格化。规格化有两种情况:

(1)左规。如果尾数运算采用双符号补码,其结果为11.1xx…x或00.0xx…x,则需要 进行左规,即需将尾数左移。尾数每左移一位,阶码减1,直到使尾数成为规格化数为止

(2)右规。如果尾数运算采用双符号补码,其结果为10.xx…x或01.xx…x,则表示尾 数出现溢出,此时需要进行右规,即需将尾数右移一位,阶码加1。在浮点数加减运算中, 右规最多为1次。

4.舍入处理

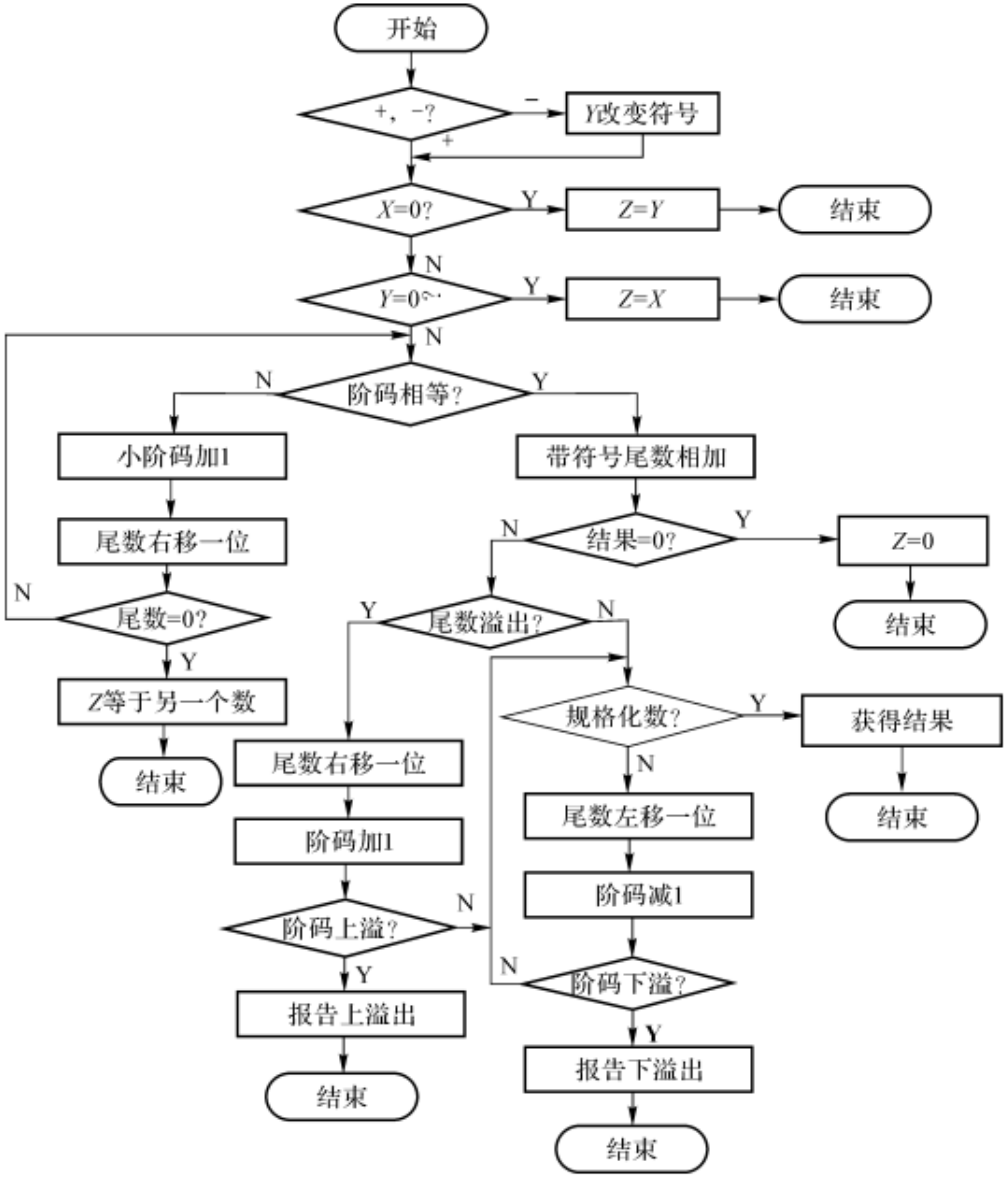
在对阶及规格化时需要将尾数右移,右移将丢掉尾数的最低位,这就出现了舍入的问 题。在进行舍入时,通常可采用以下方法。

(1)截断(尾)法:此法最简单,就是直接将右移出去的尾数低位丢弃。

(2)末位恒置1法:无论尾数右移丢弃的是0还是1,此法将保证要保留的尾数最低位 永远为1。

(3)0舍1入法:当尾数右移丢弃的是1时,要保留的尾数最低位加1;当尾数右移丢 弃的是0时,要保留的尾数最低位不变。这种方法误差较小。但遇到补码01.111…11这种 需右规的尾数时,采用此法会再次使尾数溢出,这种情况可采用截尾法。

假定两浮点数 X、Y 相加(或相减)的结果为浮点数Z,浮点数加(减)法流程如图3.31 所示。


开始
+，−?
−
Y改变符号
+
X=0?
Y
Z=Y
结束
N
Y=0?
Y
Z=X
结束
N
阶码相等?
N
Y
小阶码加1
尾数右移一位
尾数=0?
N
Y
Z等于另一个数
结束
带符号尾数相加
结果=0?
N
Y
Z=0
结束
尾数溢出?
Y
N
尾数右移一位
阶码加1
阶码上溢?
N
Y
报告上溢出
结束
规格化数?
Y
N
获得结果
结束
尾数左移一位
阶码减1
阶码下溢?
N
Y
报告下溢出
结束

例 3.25 若两浮点数为 $X=0.110101\times 2^{-010}$ 和 $Y=-0.101010\times 2^{-001}$，求两数之和及差。

解 设两浮点数阶码为 4 位，用补码表示，尾数用 8 位，均用双符号位补码表示，则两数可表示为

$$[X]_{浮}=$$

$$[Y]_{浮}=$$

例 3.25 若两浮点数为 $X=0.110101\times2^{-010}$ 和 $Y=-0.101010\times2^{-001}$，求两数之和及差。

解 设两浮点数阶码为 4 位，用补码表示，尾数用 8 位，均用双符号位补码表示，则两数可表示为

$$[X]_{浮}=1110\ 00.110101$$

$$[Y]_{浮}=1111\ 11.010110$$

(1) 对阶。阶差为

$$[\Delta E]_{补}=[E_X]_{补}+[-E_Y]_{补}=1110+0001=1111$$

即 X 的阶码比 Y 的阶码小。因此，X 尾数右移 1 位，使两者阶码相同，此时采用 0 舍 1 入，

结果为

$$[X]'_{浮}=1111\ 00.011011$$

THE END !

THANKS