故　$[x \cdot y]_{补} = 1.11001001$

由表6.19可见，与补码一位乘相比（参见表6.16和表6.17），补码两位乘的部分积多取1位符号位（共3位），乘数也多取1位符号位（共2位），这是由于乘数每次右移2位，且用3位判断，故采用双符号位更便于硬件实现。可见，当乘数数值位为偶数时，乘数取2位符号位，共需作 $n/2$ 次移位，最多作 $n/2+1$ 次加法，最后一步不移位；当 n 为奇数时，可补0变为偶数位，以简化逻辑操作。也可对乘数取1位符号位，此时共进行 $n/2+1$ 次加法运算和 $n/2+1$ 次移位（最后一步移一位）。

对于整数补码乘法，其过程与小数补码乘法完全相同。为了区别于小数乘法，在书写上将符号位和数值位中间的“.”改为“,”即可。

6.3.4　除法运算

1. 分析笔算除法

以小数为例，设 $x=-0.1011$，$y=0.1101$，求 x/y。

笔算除法时，商的符号心算而得：负正得负。其数值部分的运算如下面的竖式所示。

$$
\begin{array}{r}
0.1101 \\
0.1101\overline{)\,0.10110} \\
\underline{0.01101} \qquad 2^{-1}\cdot y \\
0.010010 \\
\underline{0.001101} \qquad 2^{-2}\cdot y \\
0.00010100 \\
\underline{0.00001101} \qquad 2^{-4}\cdot y \\
0.00000111
\end{array}
$$

所以　　商 $x/y=-0.1101$，余数 $=0.00000111$

其特点可归纳如下：

① 每次上商都是由心算来比较余数（被除数）和除数的大小，确定商为“1”还是“0”。

② 每做一次减法，总是保持余数不动，低位补0，再减去右移后的除数。

③ 上商的位置不固定。

④ 商符单独处理。

如果将上述规则完全照搬到计算机内，实现起来有一定困难，主要问题如下：

① 机器不能“心算”上商，必须通过比较被除数（或余数）和除数绝对值的大小来确定商值，即 $|x|-|y|$，若差为正（够减）上商1，差为负（不够减）上商0。

② 按照每次减法总是保持余数不动低位补0，再减去右移后的除数这一规则，则要求加法器的位数必须为除数的两倍。仔细分析发现，右移除数可以用左移余数的方法代替，其运算结果是一样的，但对线路结构更有利。不过此刻所得到的余数不是真正的余数，只有将它乘上 2^{-n} 才是

真正的余数。

③ 笔算求商时是从高位向低位逐位求的，而要求机器把每位商直接写到寄存器的不同位置也是不可取的。计算机可将每一位商直接写到寄存器的最低位，并把原来的部分商左移一位，这样更有利于硬件实现。

综上所述便可得原码除法运算规则。

2. 原码除法

原码除法和原码乘法一样，符号位是单独处理的，下面以小数为例。

设

$$[x]_{原}=x_0.x_1x_2\cdots x_n$$

$$[y]_{原}=y_0.y_1y_2\cdots y_n$$

则

$$\left[\frac{x}{y}\right]_{原}=(x_0\oplus y_0).\frac{0.x_1x_2\cdots x_n}{0.y_1y_2\cdots y_n}$$

式中，$0.x_1x_2\cdots x_n$ 为 x 的绝对值，记作 x^*；$0.y_1y_2\cdots y_n$ 为 y 的绝对值，记作 y^*。

即商符由两数符号位进行异或运算求得，商值由两数绝对值相除（x^*/y^*）求得。

小数定点除法对被除数和除数有一定的约束，即必须满足下列条件：

$$0<|\text{被除数}|\leqslant|\text{除数}|$$

实现除法运算时，还应避免除数为 0 或被除数为 0。前者结果为无限大，不能用机器的有限位数表示；后者结果总是 0，这个除法操作没有意义，浪费了机器时间。商的位数一般与操作数的位数相同。

原码除法中由于对余数的处理不同，又可分为恢复余数法和不恢复余数法（加减交替法）两种。

（1）恢复余数法

恢复余数法的特点是：当余数为负时，需加上除数，将其恢复成原来的余数。

由上所述，商值的确定是通过比较被除数和除数的绝对值大小，即 x^*-y^* 实现的，而计算机内只设加法器，故需将 x^*-y^* 操作变为 $[x^*]_{补}+[-y^*]_{补}$ 的操作。

例 6.24　已知 $x=-0.1011$，$y=-0.1101$，求 $\left[\frac{x}{y}\right]_{原}$。

解：由 $x=-0.1011$，$y=-0.1101$

得　$[x]_{原}=1.1011$，$x^*=0.1011$

$[y]_{原}=1.1101$，$y^*=0.1101$，$[-y^*]_{补}=1.0011$

表 6.20 列出了例 6.24 商值的求解过程。

表 6.20　例 6.24 恢复余数法的求解过程

被除数（余数）	商	说　明
0.1011	0.0000	
+ 1.0011		$+[-y^*]_{补}$（减去除数）

续表

被除数(余数)	商	说　明
1.1110 + 0.1101	0	余数为负,上商“0” 恢复余数+$[y^*]_{补}$
0.1011 1.0110 + 1.0011	0	被恢复的被除数 ←1 位 +$[-y^*]_{补}$(减去除数)
0.1001 1.0010 + 1.0011	01 01	余数为正,上商“1” ←1 位 +$[-y^*]_{补}$(减去除数)
0.0101 0.1010 + 1.0011	011 011	余数为正,上商“1” ←1 位 +$[-y^*]_{补}$(减去除数)
1.1101 + 0.1101	0110	余数为负,上商“0” 恢复余数+$[y^*]_{补}$
0.1010 1.0100 + 1.0011	0110	被恢复的余数 ←1 位 +$[-y^*]_{补}$(减去除数)
0.0111	01101	余数为正,上商“1”

故　商值为 0.1101

商的符号位为

$$x_0 \oplus y_0 = 1 \oplus 1 = 0$$

所以

$$\left[\frac{x}{y}\right]_{原} = 0.1101$$

由此例可见,共左移(逻辑左移)4 次,上商 5 次,第一次上的商在商的整数位上,这对小数除法而言,可用它作溢出判断。即当该位为“1”时,表示此除法溢出,不能进行,应由程序进行处理;当该位为“0”时,说明除法合法,可以进行。

在恢复余数法中,每当余数为负时,都需恢复余数,这就延长了机器除法的时间,操作也很不规则,对线路结构不利。加减交替法可克服这些缺点。

(2) 加减交替法

加减交替法又称不恢复余数法,可以认为它是恢复余数法的一种改进算法。

分析原码恢复余数法得知：

当余数 $R_i>0$ 时，可上商“1”，再对 R_i 左移一位后减除数，即 $2R_i-y^*$。

当余数 $R_i<0$ 时，可上商“0”，然后先做 R_i+y^*，即完成恢复余数的运算，再做 $2(R_i+y^*)-y^*$，即 $2R_i+y^*$。

可见，原码恢复余数法可归纳如下：

当 $R_i>0$，商上“1”，做 $2R_i-y^*$ 的运算。

当 $R_i<0$，商上“0”，做 $2R_i+y^*$ 的运算。

这里已经看不出余数的恢复问题了，而只是做加 y^* 或减 y^*，因此，一般将其称为加减交替法或不恢复余数法。

例 6.25　已知 $x=-0.1011, y=0.1101$，求 $\left[\frac{x}{y}\right]_{原}$。

解： 由 $x=-0.1011, y=0.1101$

得　　$[x]_{原}=1.1011, x^*=0.1011$

$[y]_{原}=0.1101, y^*=0.1101, [-y^*]_{补}=1.0011$

表 6.21 列出了此例商值的求解过程。

表 6.21　例 6.25 加减交替法的求解过程

被除数(余数)	商	说　明
0.1011 +1.0011	0.0000	 +$[-y^*]_{补}$(减除数)
1.1110 1.1100 +0.1101	0 0	余数为负，上商“0” ←1 位 +$[y^*]_{补}$(加除数)
0.1001 1.0010 +1.0011	01 01	余数为正，上商“1” ←1 位 +$[-y^*]_{补}$(减除数)
0.0101 0.1010 +1.0011	011 011	余数为正，上商“1” ←1 位 +$[-y^*]_{补}$(减除数)
1.1101 1.1010 +0.1101	0110 0110	余数为负，上商“0” ←1 位 +$[y^*]_{补}$(加除数)
0.0111	01101	余数为正，上商“1”

商的符号位为

$$x_0 \oplus y_0 = 1 \oplus 0 = 1$$

所以

$$\left[\frac{x}{y}\right]_{原} = 1.1101$$

分析此例可见，n 位小数的除法共上商 $n+1$ 次（第一次商用来判断是否溢出），左移（逻辑左移）n 次，可用移位次数判断除法是否结束。倘若比例因子选择恰当，除法结果不溢出，则第一次商肯定是 0。如果省去这位商，只需上商 n 次即可，此时除法运算一开始应将被除数左移一位减去除数，然后再根据余数上商。读者可以自己练习。

需要说明一点，表 6.21 中操作数也可采用双符号位，此时移位操作可按算术左移处理，最高符号位是真正的符号，次高位符号位在移位时可被第一数值位占用。

（3）原码加减交替法所需的硬件配置

图 6.11 是实现原码加减交替法运算的基本硬件配置框图。

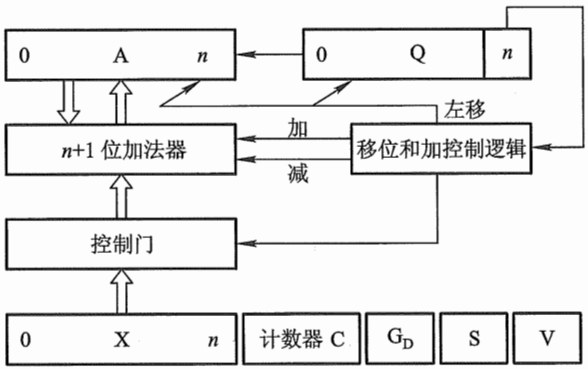

图 6.11　原码加减交替法运算的基本硬件配置

图中 A、X、Q 均为 $n+1$ 位寄存器，其中 A 存放被除数的原码，X 存放除数的原码。移位和加控制逻辑受 Q 的末位 Q_n 控制（$Q_n=1$ 做减法，$Q_n=0$ 做加法），计数器 C 用于控制逐位相除的次数 n，G_D 为除法标记，V 为溢出标记，S 为商符。

（4）原码加减交替法控制流程

图 6.12 为原码加减交替法控制流程图。

除法开始前，Q 寄存器被清零，准备接收商，被除数的原码放在 A 中，除数的原码放在 X 中，计数器 C 中存放除数的位数 n。除法开始后，首先通过异或运算求出商符，并存于 S。接着将被除数和除数变为绝对值，然后开始用第一次上商判断是否溢出。若溢出，则置溢出标记 V 为 1，停止运算，进行中断处理，重新选择比例因子；若无溢出，则先上商，接着 A、Q 同时左移一位，然后再根据上一次商值的状态，决定是加还是减除数，这样重复 n 次后，再上最后一次商（共上商 $n+1$次），即得运算结果。

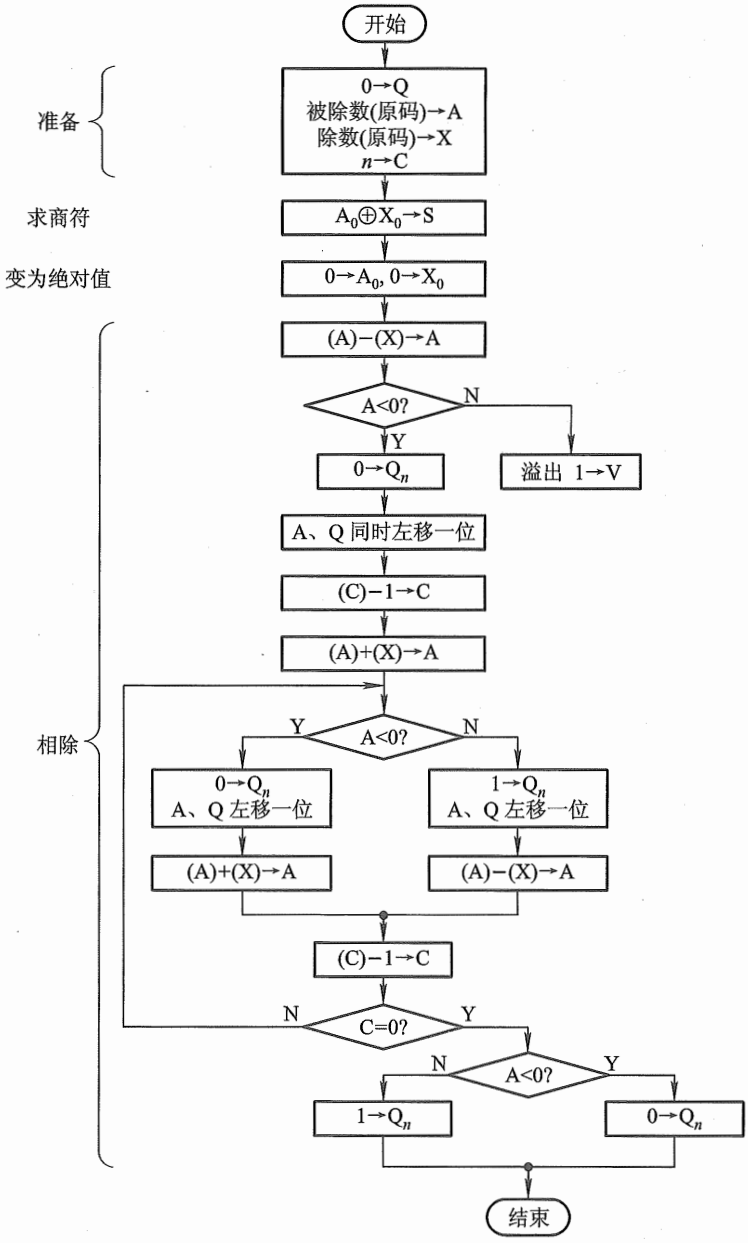

图 6.12　原码加减交替法控制流程图

对于整数除法，要求满足以下条件：

$$0 < |\text{除数}| \leqslant |\text{被除数}|$$

因为这样才能得到整数商。通常在做整数除法前，先要对这个条件进行判断，若不满足上述

条件,机器发出出错信号,程序要重新设定比例因子。

上述讨论的小数除法完全适用于整数除法,只是整数除法的被除数位数可以是除数的两倍,且要求被除数的高 n 位要比除数(n 位)小,否则即为溢出。如果被除数和除数的位数都是单字长,则要在被除数前面加上一个字的0,从而扩展成双倍字长再进行运算。

为了提高除法速度,可采用阵列除法器,有关内容参见附录6B。

3. 补码除法

与补码乘法类似,也可以用补码完成除法操作。补码除法也分恢复余数法和加减交替法,后者用得较多,在此只讨论加减交替法。

(1) 补码加减交替法运算规则

补码除法的符号位和数值部分是一起参加运算的,因此在算法上不像原码除法那样直观,主要需要解决3个问题:① 如何确定商值;② 如何形成商符;③ 如何获得新的余数。

① 欲确定商值,必须先比较被除数和除数的大小,然后才能求得商值。

- 比较被除数(余数)和除数的大小

补码除法的操作数均为补码,其符号又是任意的,因此要比较被除数 $[x]_{补}$ 和除数 $[y]_{补}$ 的大小就不能简单地用 $[x]_{补}$ 减去 $[y]_{补}$。实质上比较 $[x]_{补}$ 和 $[y]_{补}$ 的大小就是比较它们所对应的绝对值的大小。同样在求商的过程中,比较余数 $[R_i]_{补}$ 与除数 $[y]_{补}$ 的大小,也是比较它们所对应的绝对值的大小。这种比较的算法可归纳为以下两点。

第一,当被除数与除数同号时,做减法,若得到的余数与除数同号,表示“够减”,否则表示“不够减”。

第二,当被除数与除数异号时,做加法,若得到的余数与除数异号,表示“够减”,否则表示“不够减”。

此算法如表6.22所示。

表6.22 比较算法表

比较 $[x]_{补}$ 与 $[y]_{补}$ 的符号	求余数	比较 $[R_i]_{补}$ 与 $[y]_{补}$ 的符号
同号	$[x]_{补}-[y]_{补}$	同号,表示“够减”
异号	$[x]_{补}+[y]_{补}$	异号,表示“够减”

- 商值的确定

补码除法的商也是用补码表示的,如果约定商的末位用“恒置1”的舍入规则,那么除末位商外,其余各位的商值对正商和负商而言,上商规则是不同的。因为在负商的情况下,除末位商以外,其余任何一位的商与真值都正好相反。因此,上商的算法可归纳为以下两点。

第一,如果 $[x]_{补}$ 与 $[y]_{补}$ 同号,商为正,则“够减”时上商“1”,“不够减”时上商“0”(按原码规则上商)。

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。

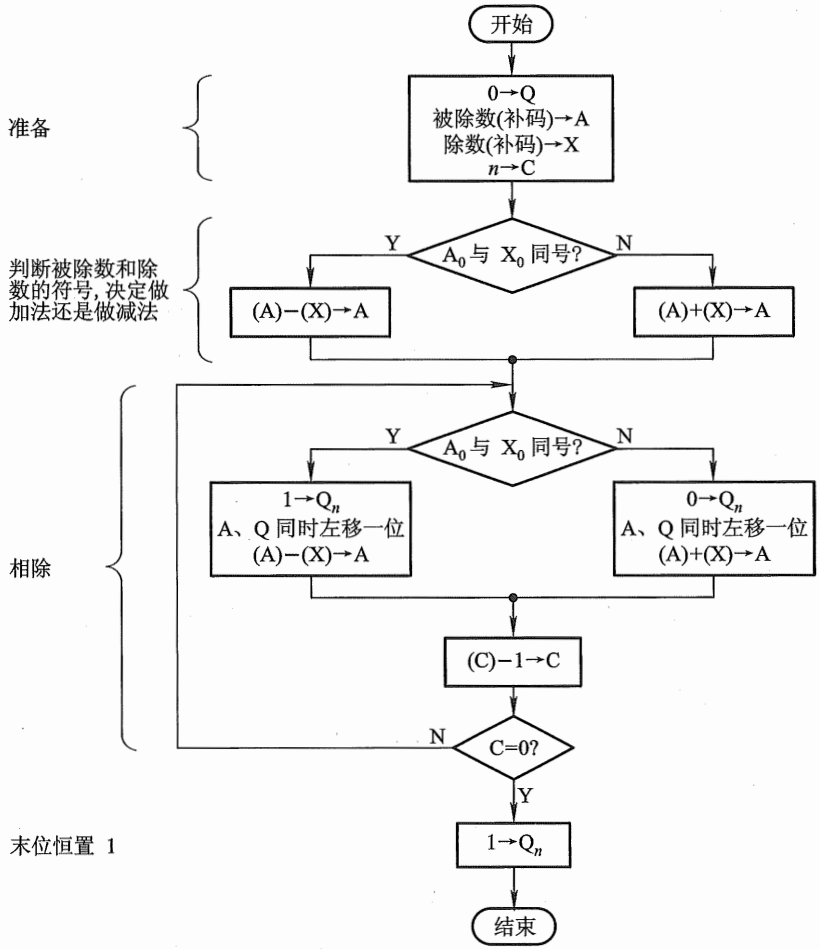

图 6.13　补码加减交替法控制流程

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。