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

3.1.3 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3.14 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

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。

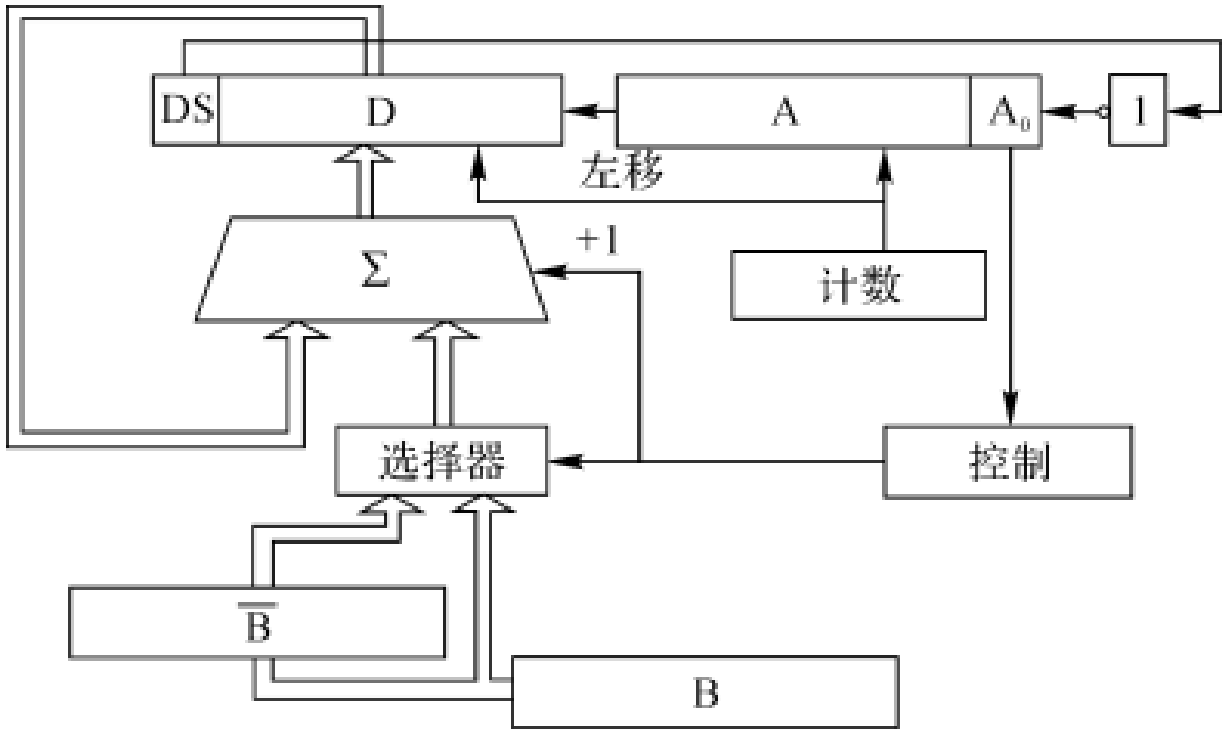
解 $[X]_{原}=1.10001011$，$[Y]_{原}=0.1110$，商符 $=1\oplus 0=1$。数值除法过程如图 3.23 所示。

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。

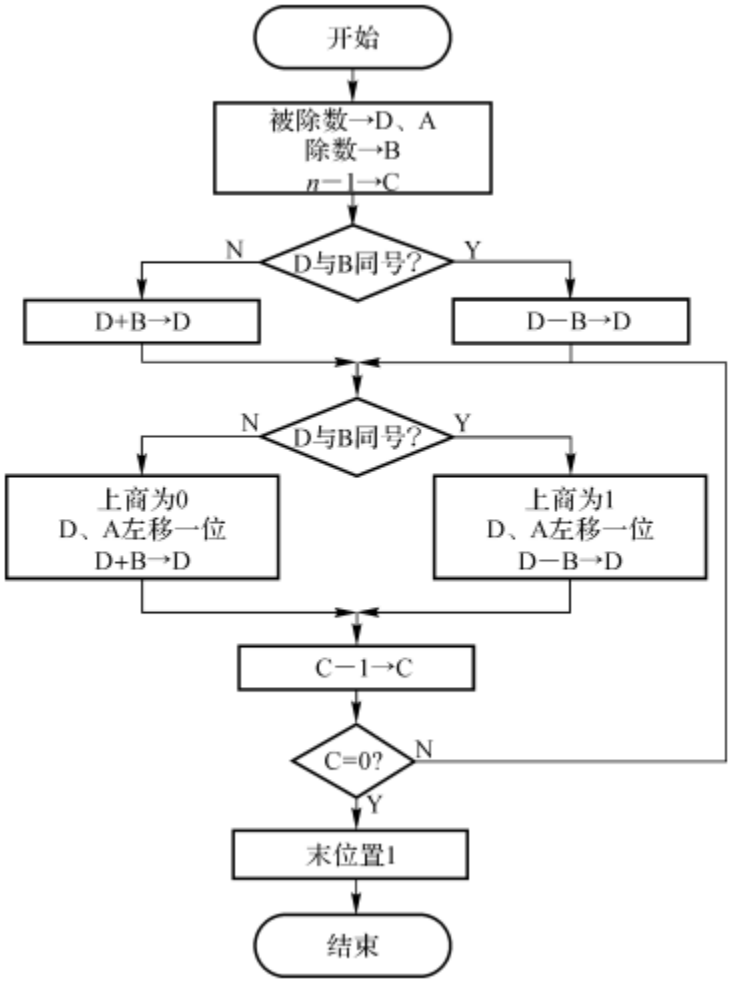


图3.25 补码除法算法流程框图

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

$$[-65]_{补}=10111111$$

$$[-32]_{补}=11100000$$

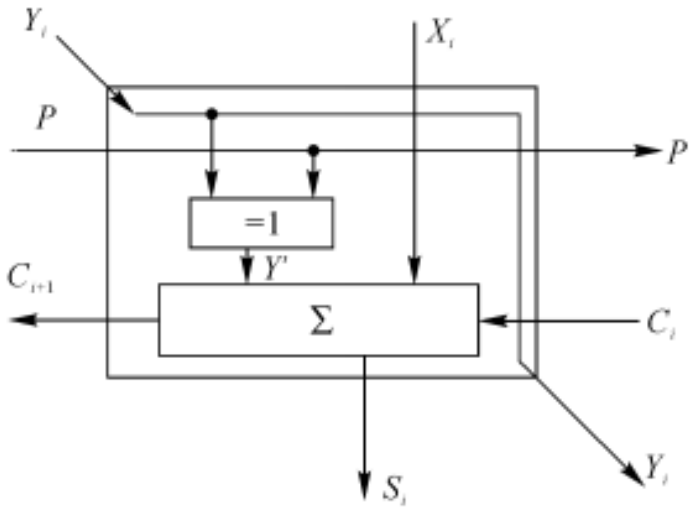
65－32 及 32－65 的运算过程如下：

```
    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。



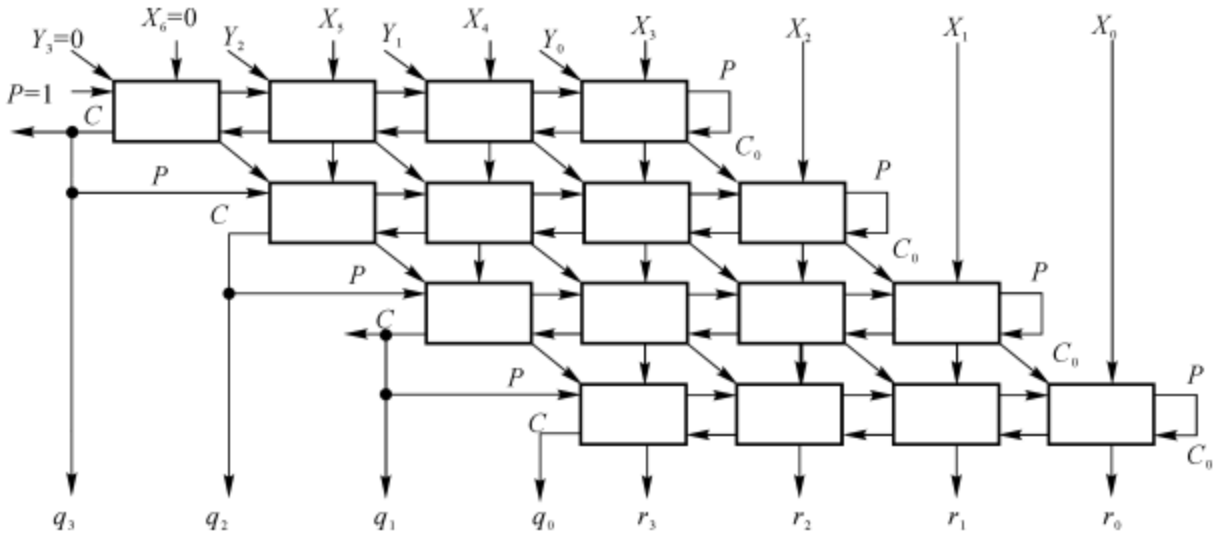
2）无符号数阵列除法器

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

3.3 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

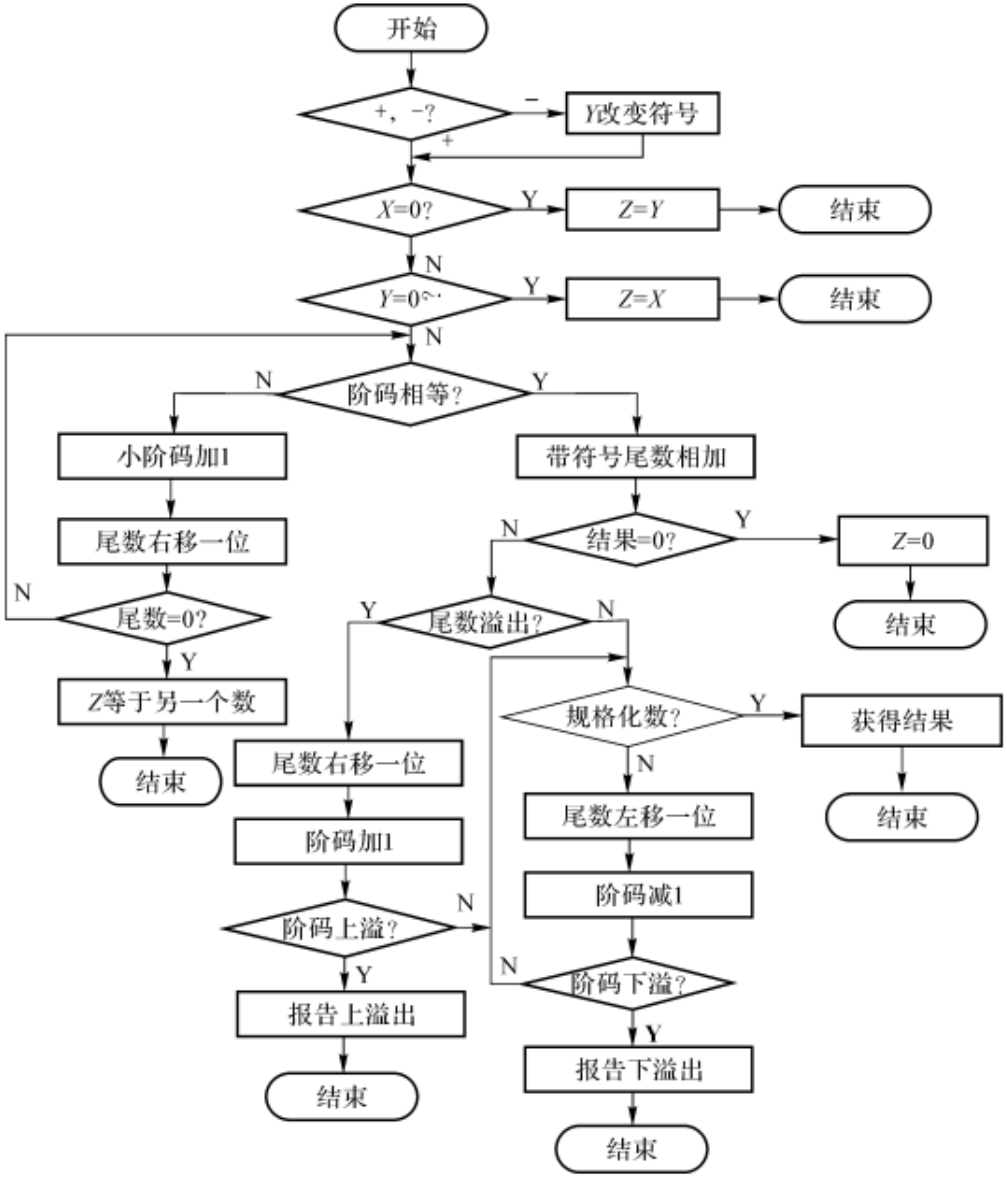
在对阶及规格化时需要将尾数右移,右移将丢掉尾数的最低位,这就出现了舍入的问 题。在进行舍入时,通常可采用以下方法。

(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
获得结果
结束
N
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

$$[X]'_{浮}=1111\ 00.011011$$

3.3.2 浮点乘除运算

1.浮点乘法运算

设两个规格化浮点数 $X=M_x \times 2^{Ex}$ 和$Y=My \times 2^{Ey}$,其乘积为

$$Z=(M_x \times M_y) \times 2^{(E_x+E_y)} \qquad (3.21)$$

该式表明,两浮点数乘积的阶码为两乘数阶码之和,乘积的尾数为两乘数尾数之积。由此 可得,浮点数乘法运算规则如下:

(1)参加乘法运算的两浮点数必须是规格化数,且不为0。只要有一个乘数为0,则乘 积必为0。

(2)求乘积的阶码,即$E_z=E_x+E_y$,并判断 E_z 是否溢出。当 $E_z>E_{max}$(最大阶码) 时,发生上溢出,即乘积已无法表示;当 $E_z<E_{min}$(最小阶码)时,发生下溢出,乘积可用0 表示。当发生溢出,尤其是上溢时,应重新定义浮点数或对两乘数作出限制。

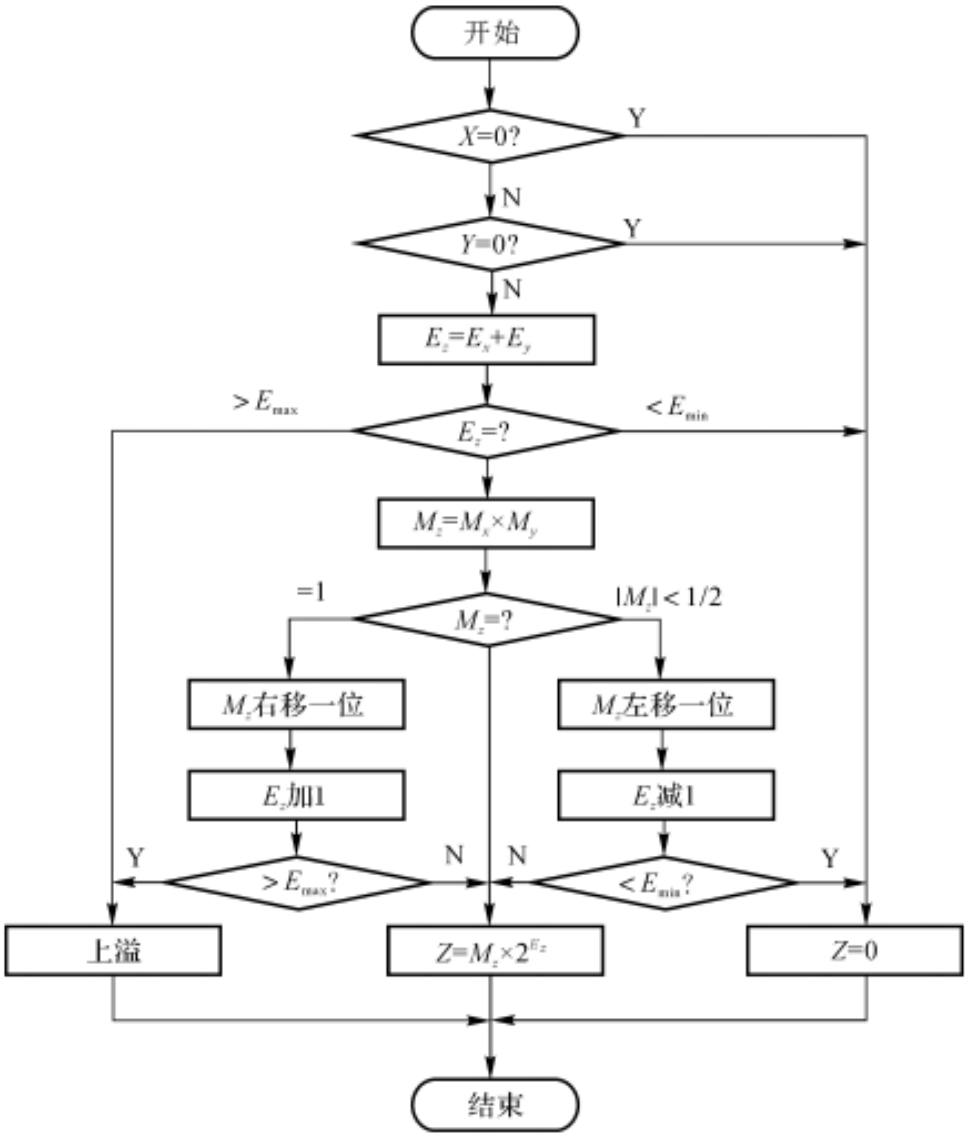
(3)两乘数的尾数相乘,可采用3.1.2节所介绍的方法进行。

(4)规格化乘积的尾数。假定尾数为n 位补码(其中含1位符号),其规格化正数范围 为$+1/2\sim+(1-2^{-(n-1)})$,规格化负数的范围为$-(1/2+2^{-(n-1)})\sim-1$。

(5)规格化中,右规时应注意采用某种舍入算法。

按照上述规则就可以获得两浮点数的乘积。

图3.32为浮点数乘法的流程图。


开始
X=0?
Y
N
Y=0?
Y
N
$E_z=E_x+E_y$
$>E_{max}$
E_z=?
$<E_{min}$
$M_z=M_x\times M_y$
=1
M_z=?
$|M_z|<1/2$
M_z右移一位
M_z左移一位
E_z加1
E_z减1
Y
$>E_{max}$?
N
N
$<E_{min}$?
Y
上溢
$Z=M_z\times 2^{E_z}$
Z=0
结束

2.浮点除法运算

设两个规格化浮点数 $X=M_x \times 2^{Ex}$ 和$Y=M_y \times 2^{Ey}$,其相除为

$$Z=(M_x \div M_y) \times 2^{E_x - E_y} \qquad (3.22)$$

该式表明,两浮点数相除之商的阶码为被除数的阶码减去除数的阶码,商的尾数为被除数 的尾数除以除数的尾数。由此可得,浮点数除法运算规则如下:

(1)参加除法运算的两浮点数必须是规格化数,除数不能为0。若被除数为0,则商必 为0。

(2)求商的阶码,即$E_z=E_x-E_y$,并判断Ez 是否溢出。 当$E_z>E_{max}$(最大阶码)时, 发生上溢出,即商已无法表示;当$E_z<E_{min}$(最小阶码)时,发生下溢出,商可用0表示。当 发生溢出,尤其是上溢时,应重新定义浮点数或对被除数、除数作出限制。

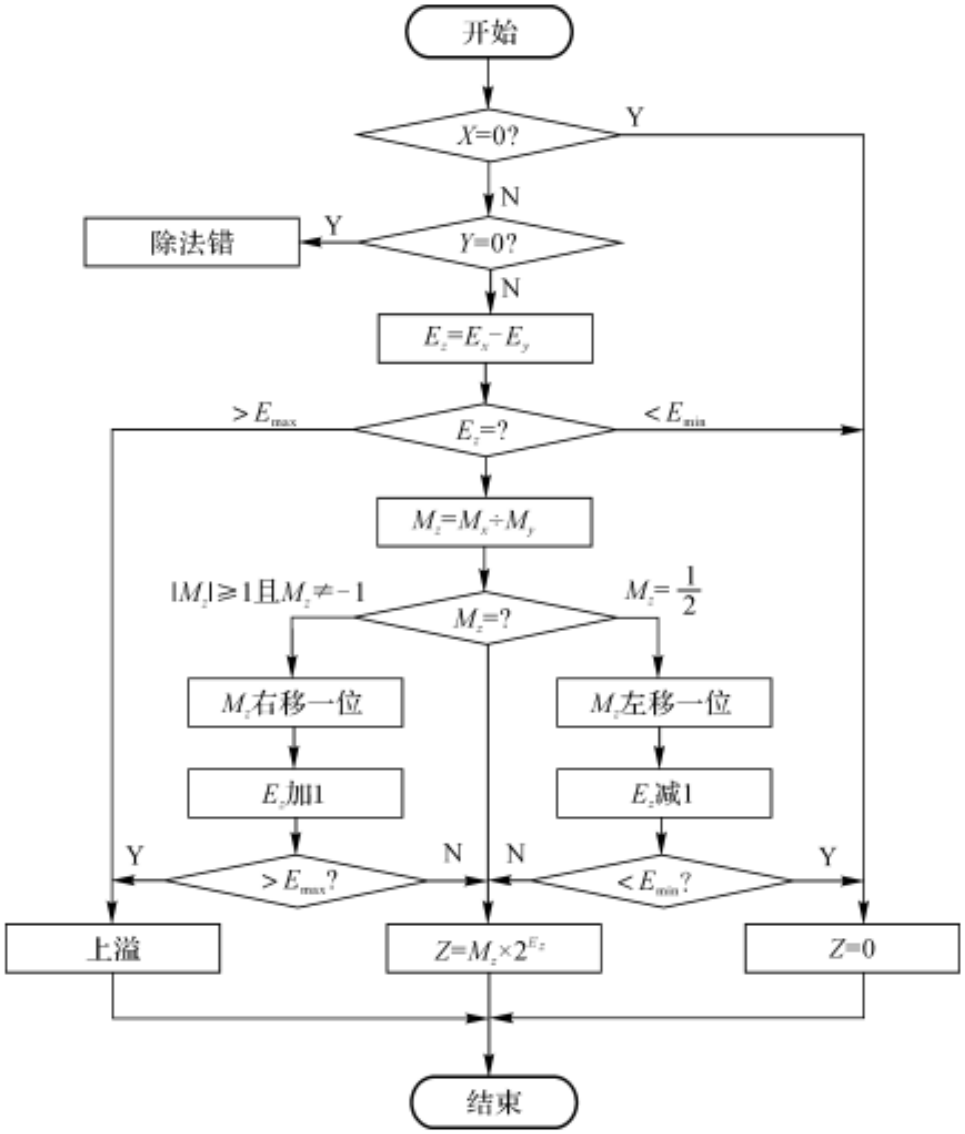
(3)被除数的尾数除以除数的尾数,由于进行的是浮点数运算,因此只要求两尾数是 规格化数,并不要求被除数的绝对值小于除数的绝对值。除法可采用3.1.3节所介绍的方法 进行。

(4)规格化商的尾数。根据上述规格化数的范围,两数相除时,可能出现1/2除以-1的情况,此时的商为-1/2,这不是规格化数(补码表示时),尾数需要左规,但只需要左规 1次,便可使商之尾数变为规格化数。

(5)规格化中,右规时应注意采用某种舍入算法。

按照上述规则就可以获得两浮点数相除的商。

图3.33为浮点数除法的流程图。


开始
X=0?
Y
N
除法错
Y=0?
Y
N
$E_z=E_x-E_y$
$>E_{max}$
E_z=?
$<E_{min}$
$M_z=M_x\div M_y$
$|M_z|\geq 1$且$M_z\neq -1$
M_z=?
$M_z=\frac{1}{2}$
M_z右移一位
M_z左移一位
E_z加1
E_z减1
Y
$>E_{max}$?
N
N
$<E_{min}$?
Y
上溢
$Z=M_z\times 2^{E_z}$
Z=0
结束

3.3.3 浮点数运算的实现方法

在计算机中实现浮点数运算可以采用不同的方法。

1.软件方法

在一些比较简单的微型机、单片机中,内部有算术逻辑单元,并设置有加减法指令,可 以依据浮点数运算流程编写程序来实现浮点数运算。这种方法速度慢,工作量大,应尽量 避免使用。

2.专用浮点处理器

专用浮点处理器是为没有浮点处理能力的处理器配置的。例如,在8086处理器构成的 微机中可以配置浮点协处理器8087,80286微机可配置80287,80386微机可配置80387 等,这样就可以提高计算机的浮点处理能力。今天的高档处理器早已把浮点协处理器集成 在处理器芯片内部,如从80486DX微机之后即是如此。

在设计计算机系统时,若采用的处理器不支持浮点运算,而系统又需要进行浮点运算, 则可以考虑在此系统中配置独立的浮点协处理器。

3.在处理器中设置浮点运算部件

如果有浮点运算的需求,则在设计处理器时,可以将浮点运算器放进处理器中。这样 的处理器在设计制造出来之后就能实现浮点运算的功能。

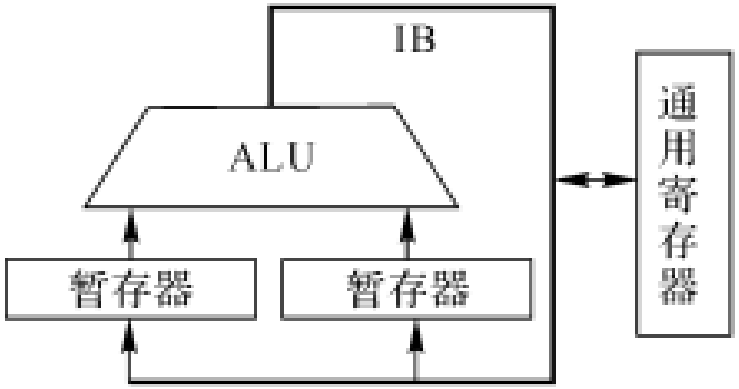
显然,后两种方法要付出硬件上的代价,但浮点运算的速度必然比软件实现要快。

浮点运算部件也可以采用流水线结构实现。

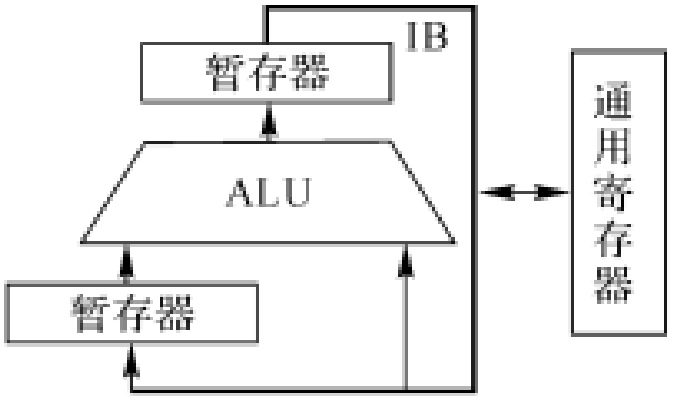
3.4 运算器基本结构

3.4.1 三种基本结构

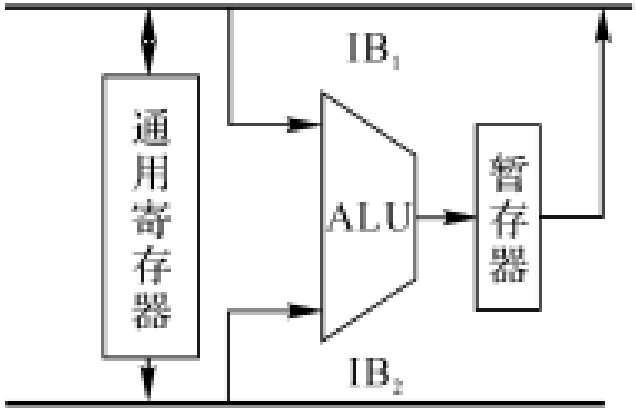
运算器内部大多数采用总线互连。根据运算器内部总线的连接方式,可将运算器的基 本结构分为单总线结构、双总线结构及三总线结构,如图3.34所示。



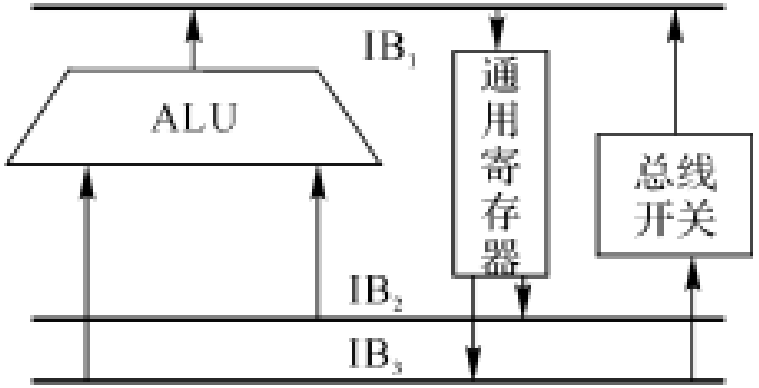
(a) 单总线结构1



(b) 单总线结构2



(c) 双总线结构

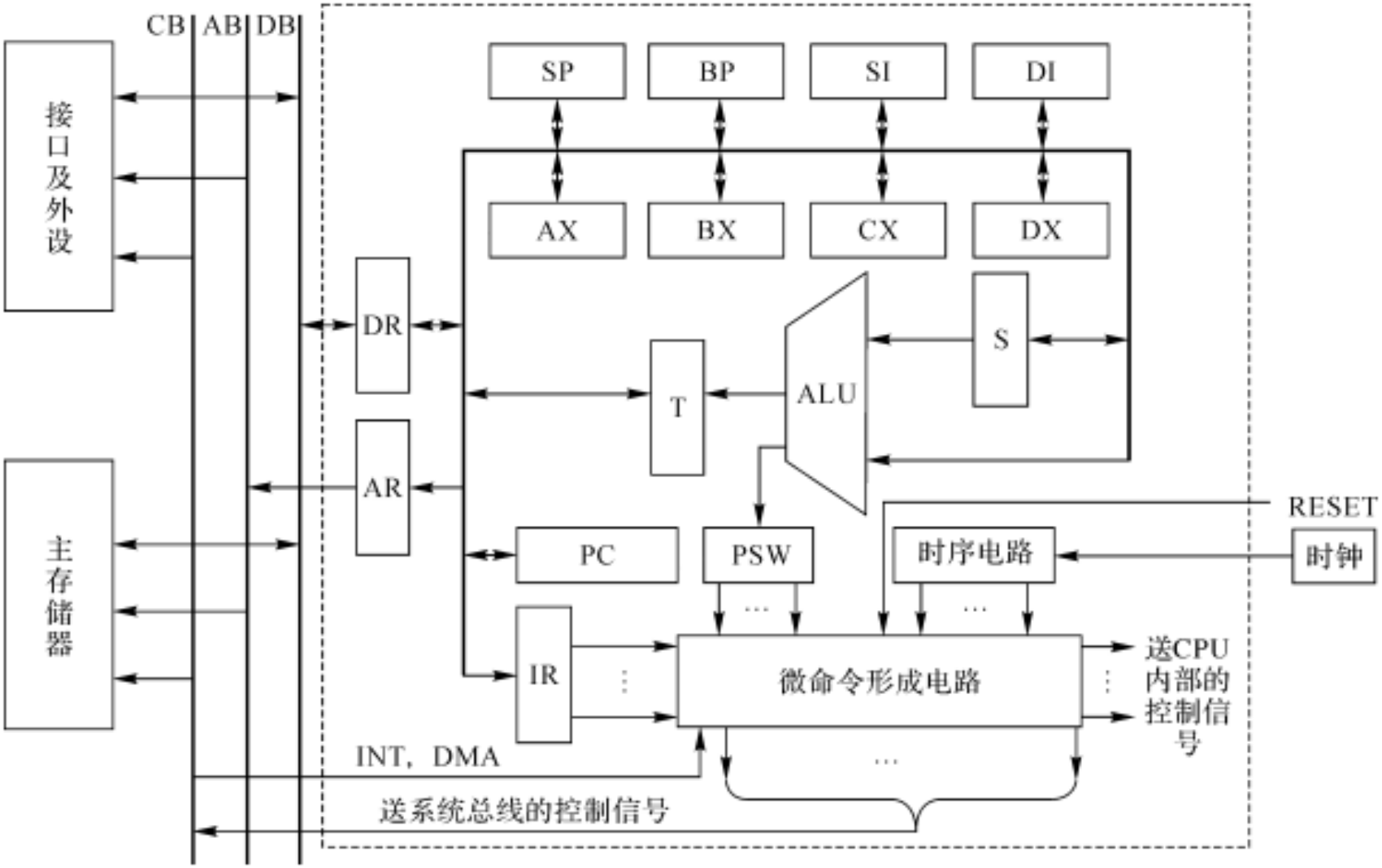


(d) 三总线结构

3.4.2 运算器实例

图3.35为由8086CPU 构成的微机系统简化框图,虚线框内为8086CPU,CPU 内部 及系统均采用单总线互连结构。

图3.35中的 ALU 与暂存器 S和 T、状态寄存器 PSW、通用寄存器(AX、BX、CX、 DX、SP、BP、SI、DI)构成了典型的单总线结构运算器。其中,ALU 完成算术逻辑运算以及算术/逻辑移位、循环移位等操作。


CB
AB
DB
接口及外设
主存储器
SP
BP
SI
DI
AX
BX
CX
DX
DR
AR
T
ALU
S
PC
PSW
时序电路
RESET
时钟
IR
微命令形成电路
送CPU内部的控制信号
INT，DMA
送系统总线的控制信号

THE END !

THANKS