计算机组成与结构

第3章 运算方法与运算器

第3章 运算方法与运算器

3.1 定点数运算

3.1.1 加减运算

3.1.1 加减运算　1. 加减运算方法

- 补码加减法的依据：

$[X+Y]_{补}=[X]_{补}+[Y]_{补}$ ……①式

$[-X]_{补}=-[X]_{补}$ ……②式

$[X-Y]_{补}=[X]_{补}+[-Y]_{补}=[X]_{补}-[Y]_{补}$ ……③式

- 证明(以纯小数为例)：

必要条件：运算不发生溢出。

无论 $X\geqslant 0$ 还是 $X<0$，$[X]_{补}=2+X$ 均成立。

① $[X]_{补}+[Y]_{补}=2+X+2+Y=2+(2+(X+Y))$

$=2+[X+Y]_{补}=[X+Y]_{补}$，得证。

舍

② 根据①式，

$[X]_{补}+[-X]_{补}=[X-X]_{补}=[0]_{补}=0$，得证。

③ 根据①②式，得证。

3.1.1 加减运算　1. 加减运算方法

- 补码加减运算规则：
  - 参加运算的操作数用补码表示；
  - 补码的符号位与数值位同时进行加运算；
    - **加**：两数补码直接相加；
    - **减**：减数补码连同符号位一起按位取反，末位加1；再与被减数的补码相加。
  - 运算结果即为和/差的补码。

3.1.1 加减运算　1. 加减运算方法

【例】利用补码加法求：

63＋35＝？　　－63＋(－35)＝？　　63－35＝？

【解】

$[63]_{补}$ ＝00111111

$[-63]_{补}$＝11000001

$[35]_{补}$ ＝00100011

$[-35]_{补}$＝11011101

```
  00111111 …63
＋ 00100011 …35
----------------
  01100010 …98
```

```
  11000001 … －63
＋ 11011101 … －35
----------------
 1 10011110 … －98
```

```
  00111111 … 63
＋ 11011101 … －35
----------------
 1 00011100 … 28
```

3.1.1 加减运算　2. 溢出判断

- 当两个同符号的数相加（或者是相异符号数相减）时，运算结果可能发生溢出。

$$\begin{array}{rl} 00111111 & \cdots 63 \\ +01010101 & \cdots 85 \\ \hline 10010100 & \cdots -108 \end{array}$$

- 如何防止溢出？

  增加补码的二进制编码长度。

- 如何判断是否发生了溢出？
  - 双符号位判决法；
  - 进位判决法；
  - 根据运算结果的符号位和进位标志判别；
  - 根据运算前后的符号位进行判别。

3.1.1 加减运算　2. 溢出判断

1）双符号位判决法

若运算结果两符号分别用S_2S_1表示，则判别溢出的逻辑表示式为：$VF=S_2 \oplus S_1$

```
  00 1000001 … 65
+ 00 1000011 … 67
  ------------------
  01 0000100 … 溢出
```

$VF=S_2 \oplus S_1=1$，发生溢出。

双符号位：

- **00**：不溢出，结果为正；
- **11**：不溢出，结果为负；
- **10**：溢出，负溢；
- **01**：溢出，正溢。

2）进位判决法

若C_{n-1}为最高数值位向符号位的进位，C_n表示符号位向更高位的进位，则判别溢出的逻辑表示式为：

$$VF = C_{n-1} \oplus C_n$$

$$\begin{array}{r} 0\ xxxxxxx \\ +\ 1\ xxxxxxx \\ \hline x\ xxxxxxx \end{array}$$

例：

$$\begin{array}{r} 0\ 1000001 \cdots 65 \\ +\ 0\ 1000011 \cdots 67 \\ \hline 1\ 0000100 \cdots 溢出 \end{array}$$

0 1

$$\begin{array}{r} 0\ xxxxxxx \\ +\ 0\ xxxxxxx \\ \hline x\ xxxxxxx \end{array} \qquad \begin{array}{r} 1\ xxxxxxx \\ +\ 1\ xxxxxxx \\ \hline x\ xxxxxxx \end{array}$$

$VF = C_{n-1} \oplus C_n = 1$，发生溢出。

3）根据运算结果的符号位和进位标志判别

适用于两同号数求和或异号数求差时判别溢出。溢出的逻辑表达式为：

$$VF = SF \oplus CF$$

$$\begin{array}{r} 0\ xxxxxxx \\ +\ 0\ xxxxxxx \\ \hline c\ s\ xxxxxxx \end{array} \qquad \begin{array}{r} 1\ xxxxxxx \\ +\ 1\ xxxxxxx \\ \hline c\ s\ xxxxxxx \end{array}$$

CF SF　　CF SF

4）根据运算前后的符号位进行判断

若用Xs、Ys、Zs分别表示两个操作数及运算结果的符号位，当两同号数求和或异号数求差时，就有可能发生溢出。

溢出是否发生可根据运算前后的符号位进行判别，其逻辑表达式为：

$$VF = X_S \cdot Y_S \cdot \overline{Z_S} + \overline{X_S} \cdot \overline{Y_S} \cdot Z_S$$

4）根据运算前后的符号位进行判断

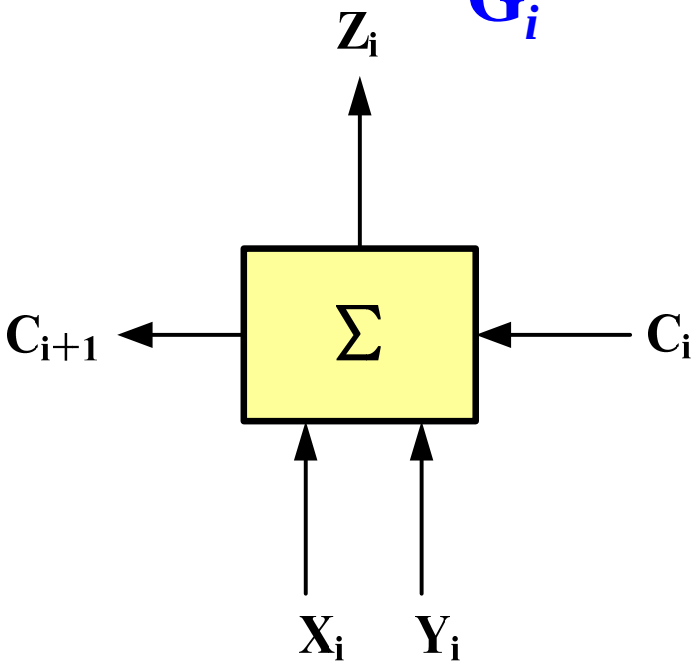
- 在CPU中，进行定点算术运算是否发生溢出，通常是由CPU中的硬件逻辑电路进行检测。
- 一旦溢出发生，则会
  - 在CPU中的标志寄存器中建立溢出标志；
  - 或者产生溢出中断。

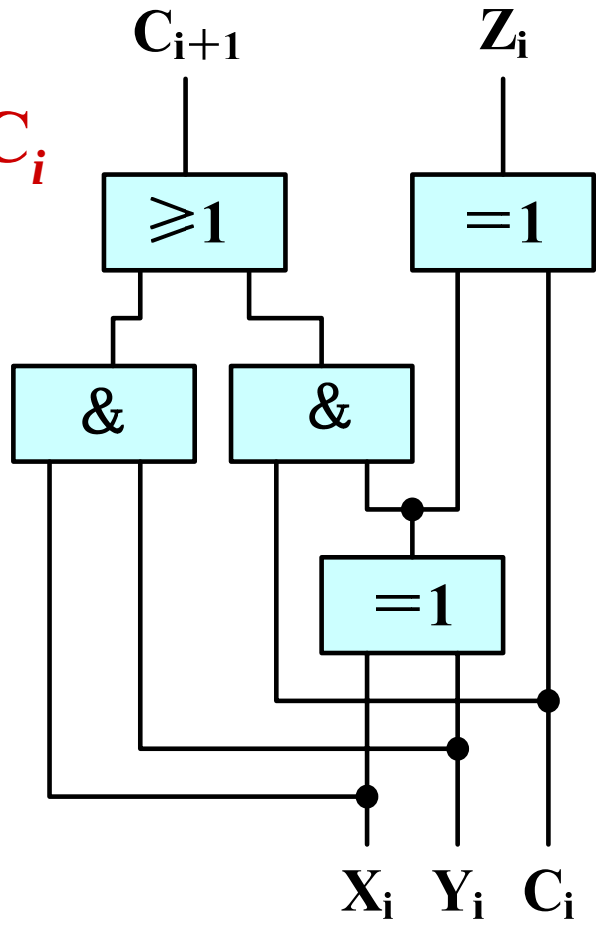
3.1.1 加减运算　3. 一位全加器的实现

设一位全加器的输入分别为 X_i 和 Y_i，低一位对该位的进位为 C_i。全加器的结果和向高一位的进位分别用 Z_i 和 C_{i+1} 表示。则一位全加器所实现的逻辑表达式：

$$Z_i = X_i \oplus Y_i \oplus C_i$$

$$C_{i+1} = \underbrace{(X_i \cdot Y_i)}_{G_i} + \underbrace{(X_i + Y_i)}_{P_i} \cdot C_i$$

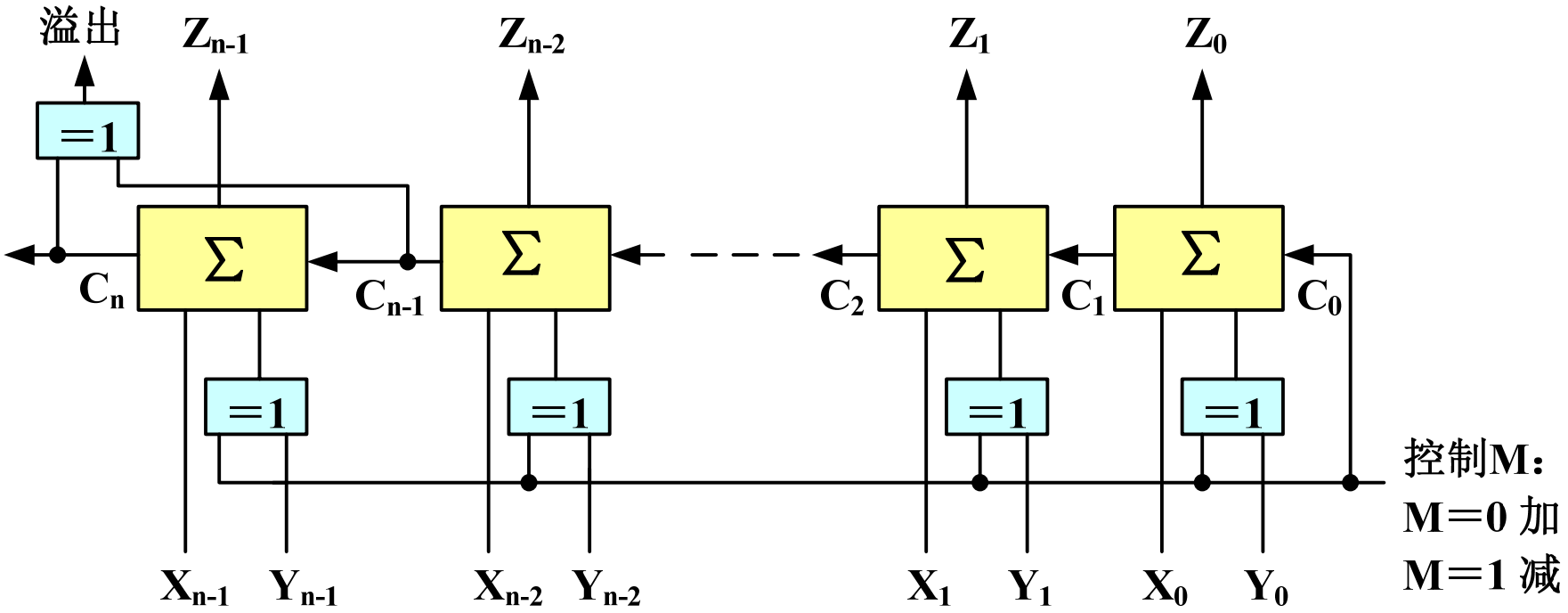




3.1.1 加减运算　4. *n*位加法器的实现

1）行波进位加法器

- 若一位全加器的进位延时为△t，则*n*位加法器的延时就是 $n \cdot \triangle t$。



行波进位的*n*位加法/减法器

2）先行进位加法器

四个进位的产生逻辑表达式：

$$C_{i+1}=G_i+P_iC_i$$

$$C_{i+2}=G_{i+1}+P_{i+1}C_{i+1}=G_{i+1}+P_{i+1}G_i+P_{i+1}P_iC_i$$

$$C_{i+3}=G_{i+2}+P_{i+2}C_{i+2}=G_{i+2}+P_{i+2}G_{i+1}+P_{i+2}P_{i+1}G_i+P_{i+2}P_{i+1}P_iC_i$$

$$C_{i+4}=G_{i+3}+P_{i+3}C_{i+3}$$

$$=\underline{G_{i+3}+P_{i+3}G_{i+2}+P_{i+3}P_{i+2}G_{i+1}+P_{i+3}P_{i+2}P_{i+1}G_i}+\underline{P_{i+3}P_{i+2}P_{i+1}P_i}C_i$$

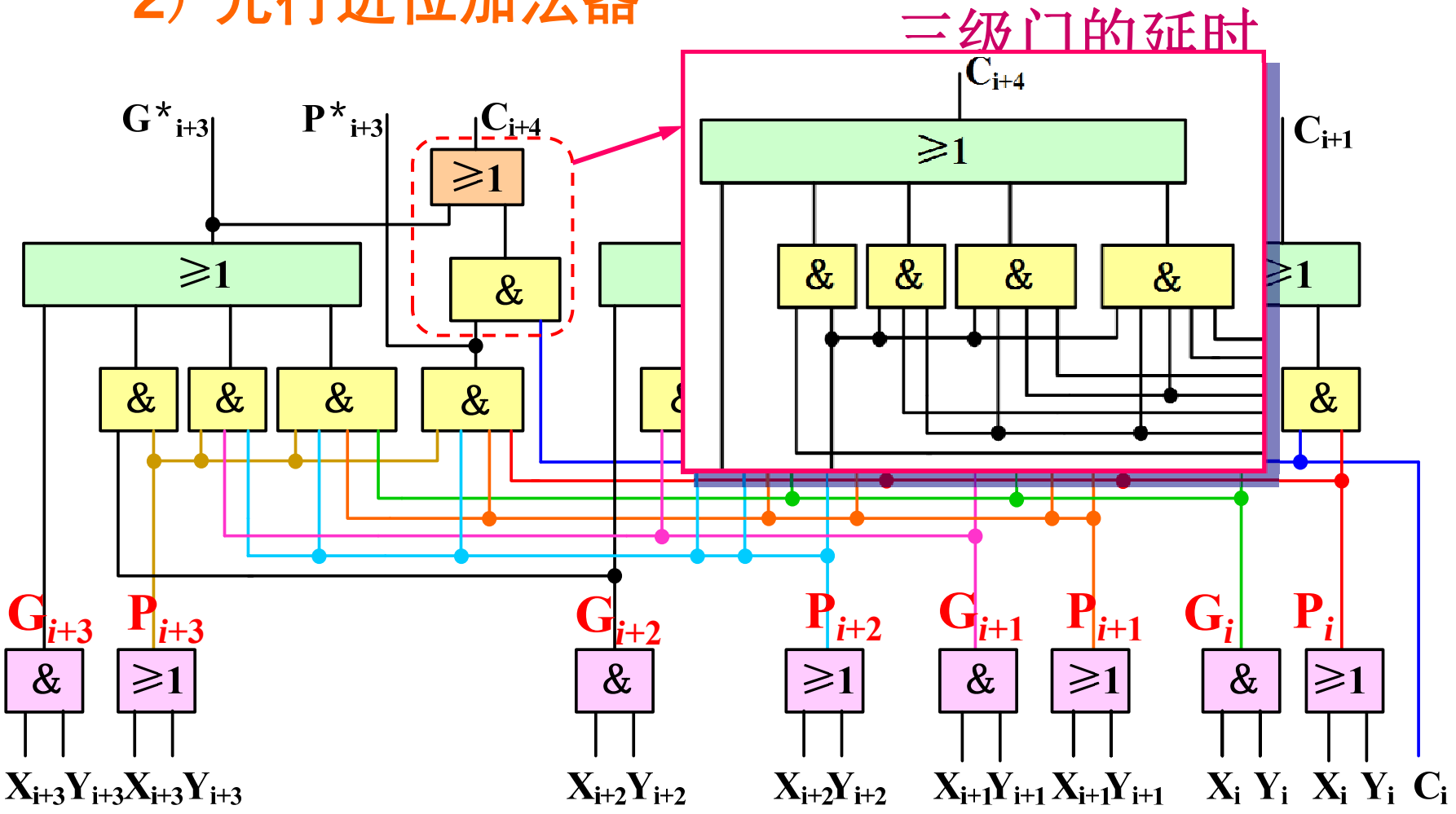
$$=G^*_{i+3}+P^*_{i+3}C_i$$

其中，$G^*_{i+3}=G_{i+3}+P_{i+3}G_{i+2}+P_{i+3}P_{i+2}G_{i+1}+P_{i+3}P_{i+2}P_{i+1}G_i$

$P^*_{i+3}=P_{i+3}P_{i+2}P_{i+1}P_i$

3.1.1 加减运算　4. n位加法器的实现

2）先行进位加法器

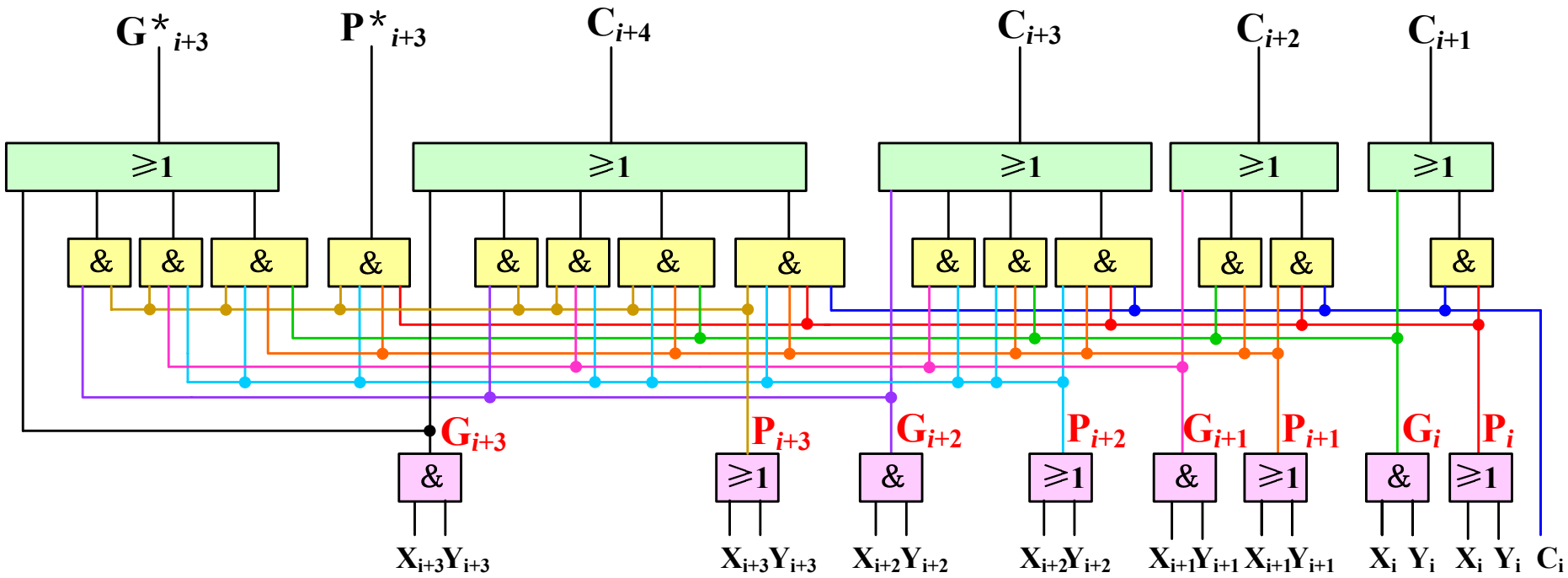


四位先行进位链电路

3.1.1 加减运算　4. *n*位加法器的实现

2）先行进位加法器

三级门的延时



四位先行进位链电路

3.1.1 加减运算　5. 8421 BCD 数加法器

1）定义

- 压缩 **BCD** 数
- 非压缩 **BCD** 数

2）加法运算

【例】计算压缩BCD数：

46＋32＝？　　46＋67＝？

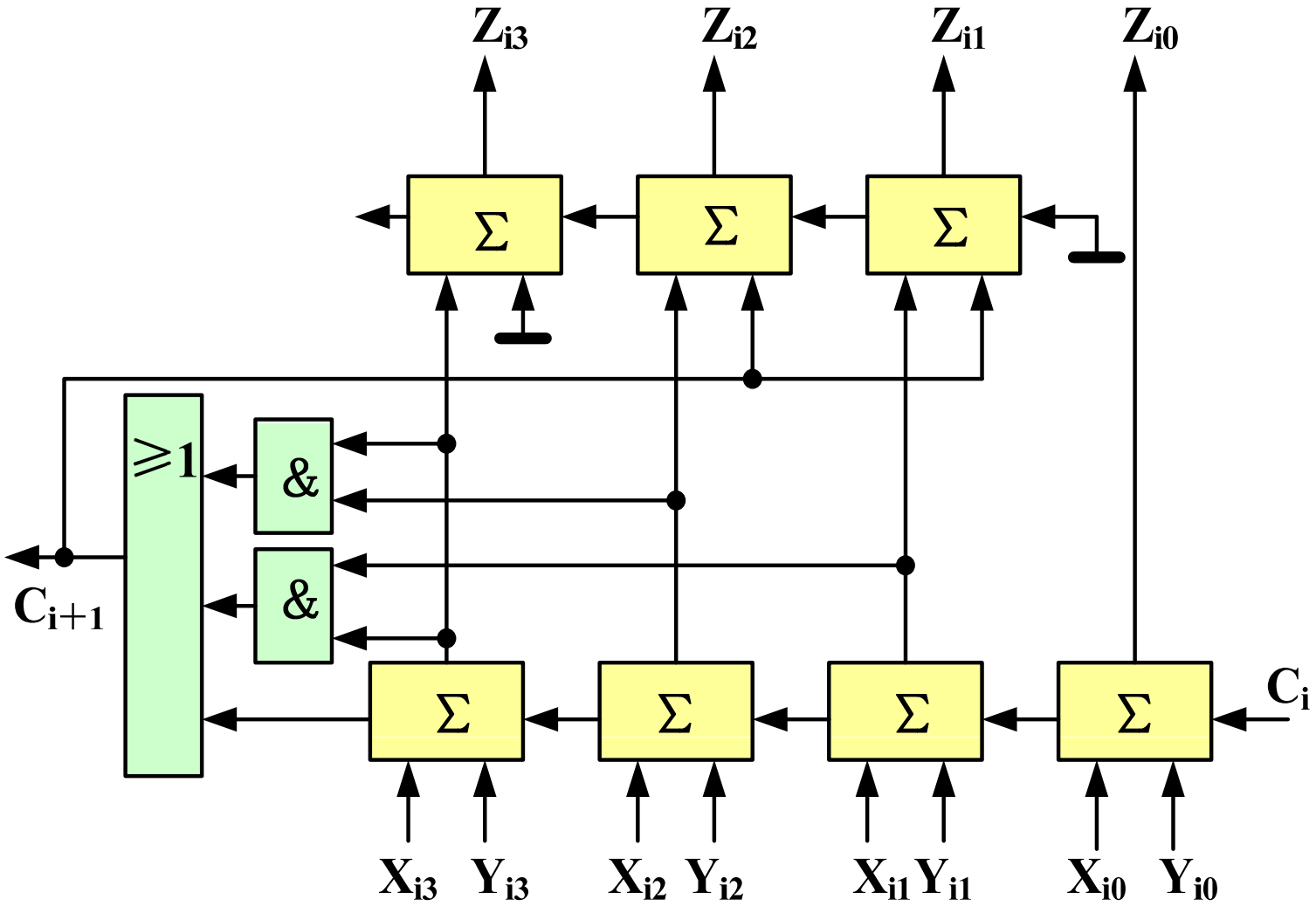
【解】

```
  0100 0110 … 46        0100 0110 … 46
+ 0011 0010 … 32      + 0110 0111 … 67
-------------------   -------------------
  0111 1000 … 78        1010 1101 … AD
                      + 0110 0110
                      -------------------
                      0001 0001 0011 … 113
```

BCD 加法的校正：

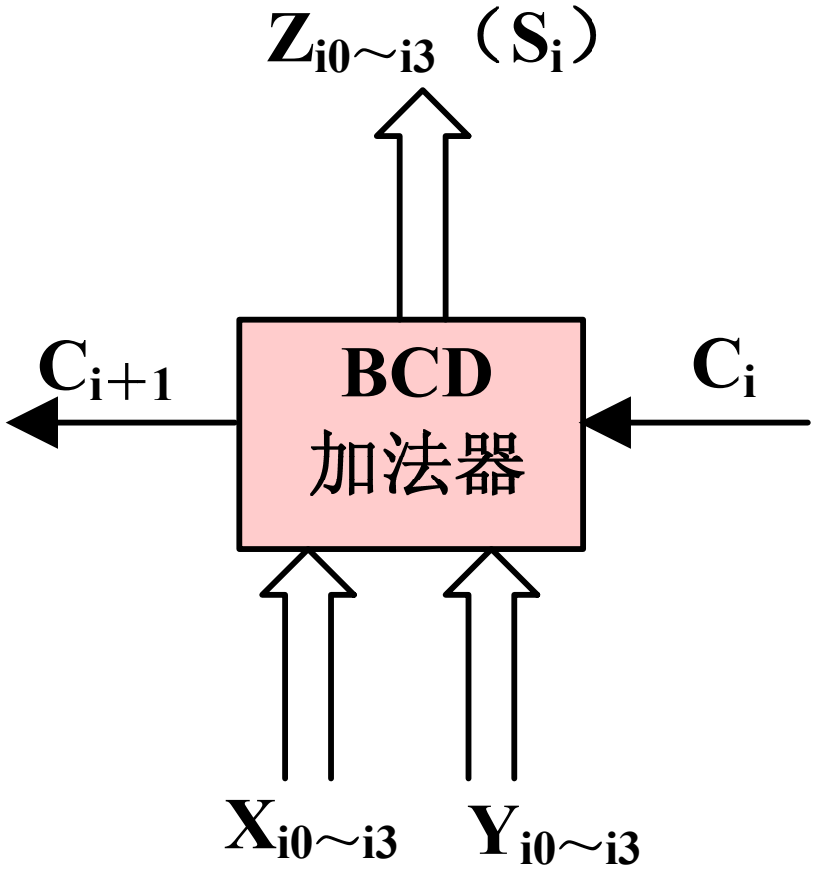
- 运算中某位BCD数(四位二进制数)相加的结果大于9或有向更高位的进位，则结果加6；
- 若不满足上述条件，则无需校正。

2）加法运算



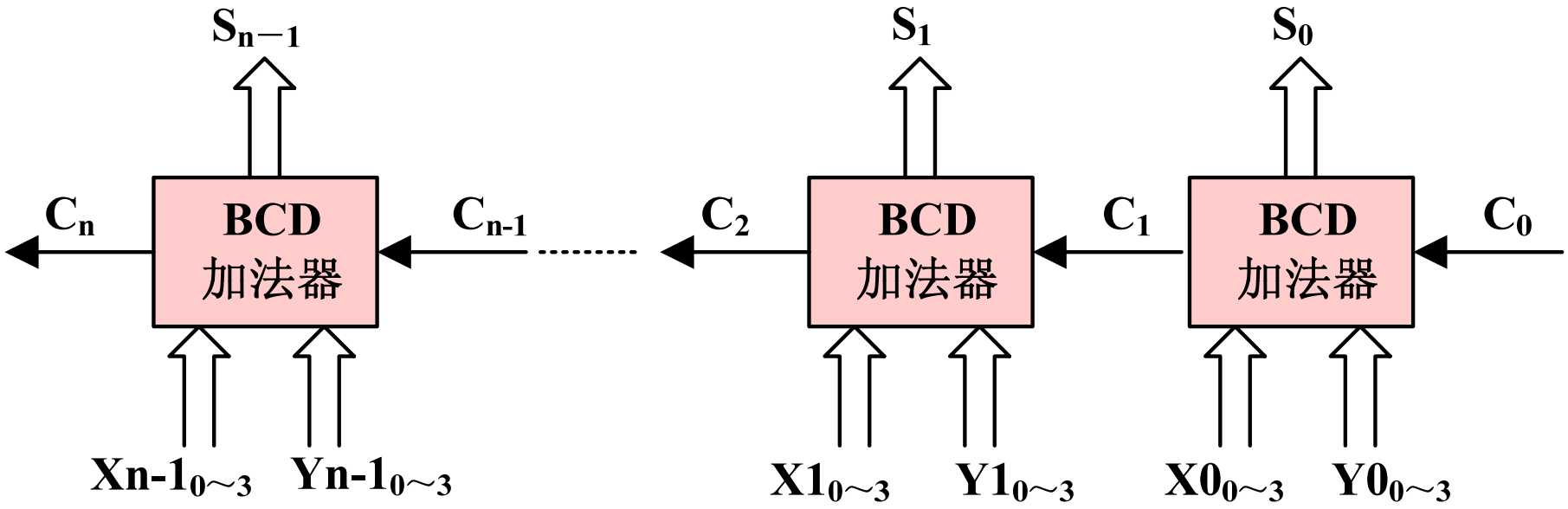
BCD加法器

2）加法运算



BCD加法器

2）加法运算



n位行波进位BCD加法器框图

3.1.1 加减运算　6. 移码加减运算

定点整数移码的加减运算的法则：

① 对两移码求和差时，首先对该两移码求和差；

② 对结果进行修正 —— 将结果的符号取反。

$[X]_{移}=2^{n-1}+X$

$[Y]_{移}=2^{n-1}+Y$

【证明】设X、Y为整数，机器字长n位

$$[X+Y]_{移}=2^{n-1}+X+Y$$

$$=2^{n-1}+([X]_{移}-2^{n-1})+([Y]_{移}-2^{n-1})$$

$$=[X]_{移}+[Y]_{移}-2^{n-1}$$

$$[X-Y]_{移}=[X]_{移}+[-Y]_{移}-2^{n-1}$$

$$[-Y]_{移}=2^{n-1}+(-Y)=2^{n-1}+(2^{n-1}-[Y]_{移})$$

$$=2^{n}-[Y]_{移}=\underbrace{((2^{n}-1)-[Y]_{移})}+1$$

$$=[[Y]_{移}]_{求补}$$

$[Y]_{移}$按位取反　末位加1

3.1.1 加减运算　6. 移码加减运算

【例】机器字长为8位，

$[X]_{移}=10111001$　　…… 57

$[Y]_{移}=01011101$　　…… −35

求：　$[X+Y]_{移}$，　$[X-Y]_{移}$。

【解】

$[X+Y]_{补}=[X]_{移}+[Y]_{移}=00010110$

因此，　　$[X+Y]_{移}=10010110$

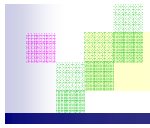
$[-Y]_{移}=[[Y]_{移}]_{求补}=10100011$

$[X-Y]_{补}=[X]_{移}+[-Y]_{移}=$

$$\begin{array}{r} 10111001 \\ +\ 10100011 \\ \hline 01011100 \end{array}$$

因此，

$[X-Y]_{移}=11011100$



3.1 定点数运算

3.1.2 乘法运算

3.1.2 乘法运算

- 原码乘法运算
  - 原码一位乘法
  - 原码二位乘法
- 补码乘法运算
  - 补码一位乘法：校正法，布斯(Booth)法
  - 补码二位乘法
- 阵列乘法器

用硬件换取速度

3.1.2 乘法运算　　1. 原码乘法运算

1）原码一位乘法的法则

假定被乘数X和乘数Y为用原码表示的纯小数，

$$[X]_{原}=X_0.X_{-1}X_{-2}\cdots X_{-(n-1)}$$

$$[Y]_{原}=Y_0.Y_{-1}Y_{-2}\cdots Y_{-(n-1)}$$

X_0、Y_0、Z_0为符号位

乘积为：$[Z]_{原}=Z_0.Z_{-1}Z_{-2}\cdots\cdots Z_{-(2n-1)}$

原码一位乘法的法则是：

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 乘积的绝对值为被乘数的绝对值与乘数的绝对值之积。即

$$[X]_{原}\times[Y]_{原}=(X_0\oplus Y_0)(|X|\times|Y|)$$

2）原码一位乘法的实现思路

【例】若$[X]_{原}=0.1101$，

$[Y]_{原}=1.1011$，

求两者之积。

【解】

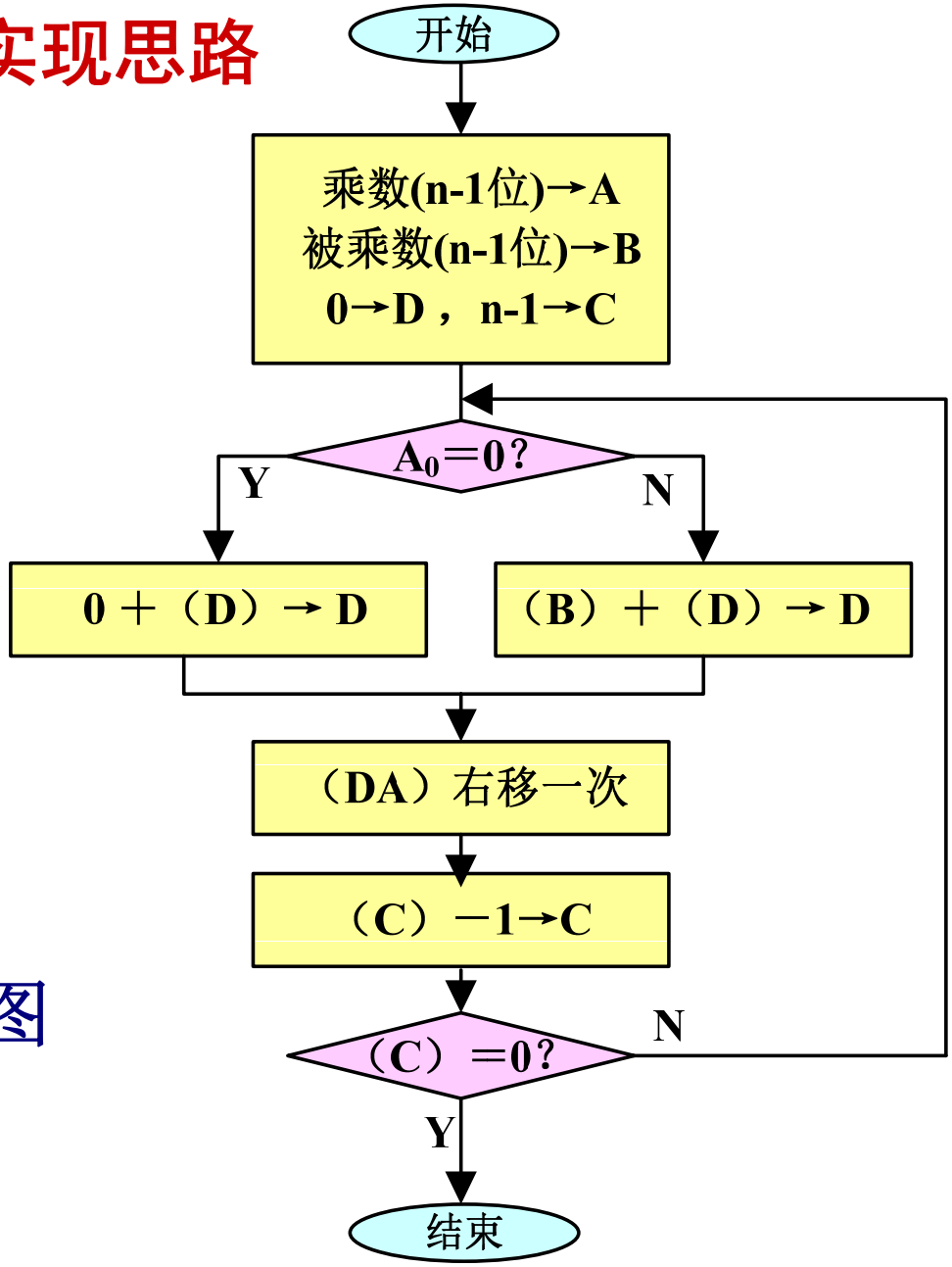
乘积的符号为：$0 \oplus 1=1$

```
      1101    B
    ×1011     A
------------
      1101
     1101
    0000
   1101
------------
1.10001111
```

	D	A	
CF 0	0000	1011	加
＋	1101	B	
0	1101	1011	
0	0110	1101	右移1位
＋	1101		加
1	0011	1101	右移1位
0	1001	1110	不加
0	0100	1111	右移1位
＋	1101		加
1	0001	1111	
0	1000	1111	右移1位

3.1.2 乘法运算　1. 原码乘法运算

2）原码一位乘法的实现思路



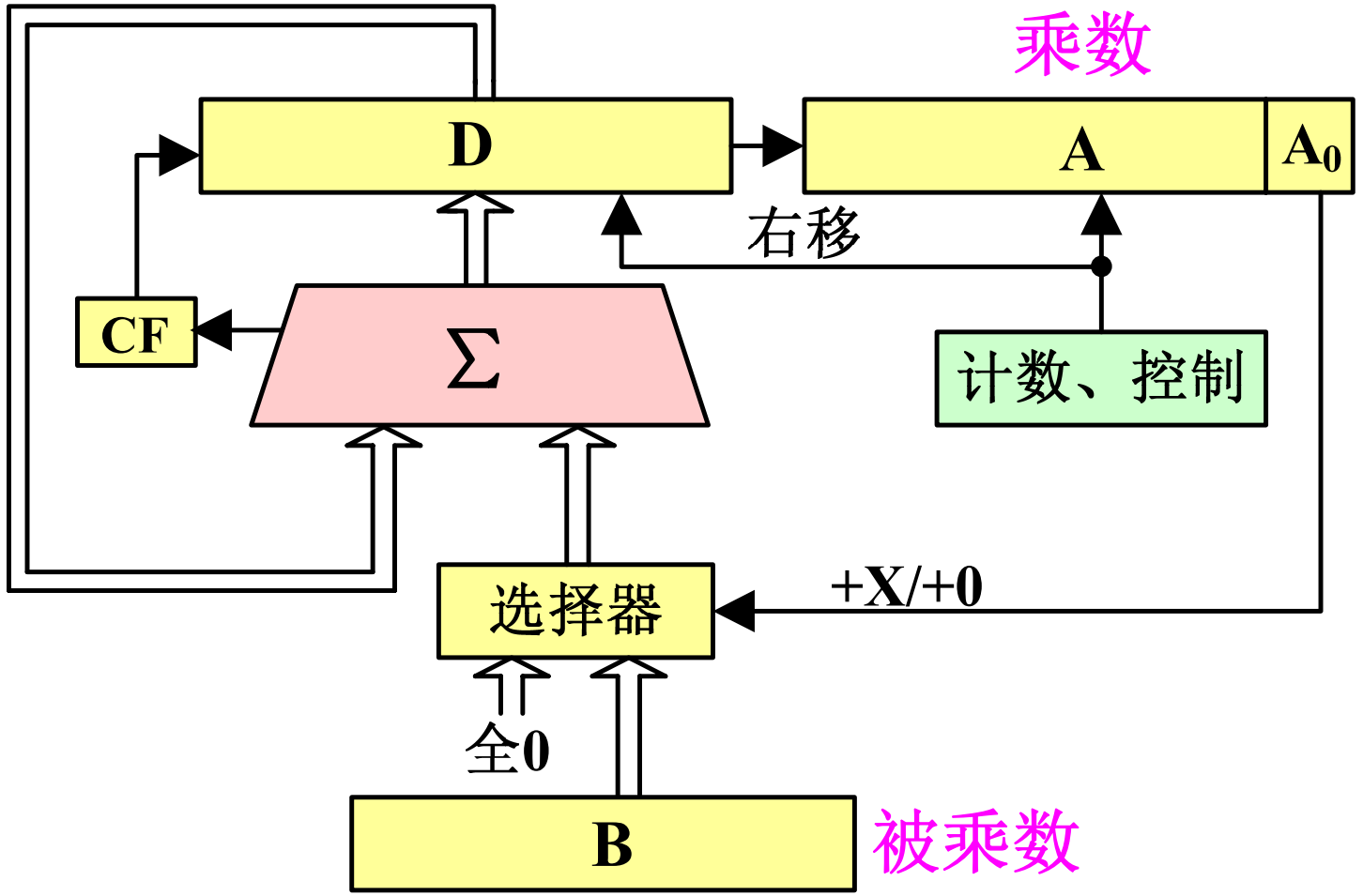
绝对值乘法思路框图

3）原码一位乘法的运算过程

	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		→右移一次
+0	1	1	0	1						A_0=1, +X
1	0	0	1	1	1	1	0	1		
0	1	0	0	1	1	1	1	0		→右移一次
0	0	0	0	0						A_0=0, +0
0	1	0	0	1	1	1	1	0		
0	0	1	0	0	1	1	1	1		→右移一次
+0	1	1	0	1						A_0=1, +X
1	0	0	0	1	1	1	1	1		
0	1	0	0	0	1	1	1	1		→右移一次

拼接符号，$[X]_{原} \cdot [Y]_{原} = 1.10001111$

4）原码一位乘法器框图



原码一位乘器框图

5）原码二位乘法

被乘数X和乘数Y为用原码表示的纯小数

$$[X]_{原}=X_0 \cdot X_{-1}X_{-2}\cdots X_{-(n-1)}$$

$$[Y]_{原}=Y_0 \cdot Y_{-1}Y_{-2}\cdots Y_{-(n-1)}$$

X_0、Y_0为符号位

两位乘数位有四种组合：

$Y_{i+1}Y_i=00$　对应＋0

$Y_{i+1}Y_i=01$　对应＋|X|

$Y_{i+1}Y_i=10$　对应＋2|X|

$Y_{i+1}Y_i=11$　对应＋3|X|

5）原码二位乘法

Y_{i+1}	Y_i	C	操　作
0	0	0	+0，右移2次，C＝0
0	0	1	+|X|，右移2次，C＝0
0	1	0	+|X|，右移2次，C＝0
0	1	1	+2|X|，右移2次，C＝0
1	0	0	+2|X|，右移2次，C＝0
1	0	1	－|X|，右移2次，C＝1
1	1	0	－|X|，右移2次，C＝1
1	1	1	+0，右移2次，C＝1

原码二位乘法的法则表

注意事项

- 原码二位乘跟原码一位乘一样，符号位和数值部分的运算是分开进行的，但它是用2位的乘数状态来决定新的部分积的形成。两位的乘数的状态由“00”-“11”，分别表示0-3，他们都是用部分积增加0-3倍被乘数后再右移两位。
- 由于3倍的被乘数无法通过被乘数左移来获得，所以可以通过4-1倍来实现，但是00-11只能表示0-3倍，所以要引入标志位C。通过乘数“11”表示的3倍，再加上C位置“1”来实现加1倍，可以实现4倍被乘数(加几倍被乘数都是加在部分积上的)。
- 4-1倍被乘数|x|中(|x|是绝对值，因为符号位分开运算)，有-|x|,所以要采用“$[-|x|]_{补}$”补码来实现，参与原码两位乘运算的操作数都是绝对值的补码(绝对值是正数，所以原码=补码。因为实际上是加3倍的|x|，所以结果为正，补码结果=原码结果)。
- 因为+2倍被乘数，也就是“$+[2|x|]_{补}$”时，使部分积的绝对值大于2，所以部分积需取3位符号位并以最高位符号位作真正符号位，才能保证正确运算。

5）原码二位乘法

【例】

设X＝＋0.100111，Y＝－0.100111，利用原码求积。

【解】

$[X]_{原}=0.100111$

$[Y]_{原}=1.100111$

$[-|X|]_{补}=1.011001$

3.1.2 乘法运算　1. 原码乘法运算

5）原码二位乘法

$[X]_{原}=0.100111$

$[-|X|]_{补}=1.011001$

$[Y]_{原}=1.100111$

原码二位乘法的运算过程

符号位	D	A	操作
0 0 0 1 1 1	0 0 0 0 0 0 0 1 1 0 0 1	1 0 0 1 1 1	C=0 −X
1 1 1 1 1 1 0 0 1	0 1 1 0 0 1 1 1 0 1 1 0 0 0 1 1 1 0	0 1 1 0 0 1	C=1 →右移二位 +2X
0 0 1 0 0 0 0 0 1	0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 1 0	0 0 0 1 1 0	C=0 →右移二位 +2X
0 0 1 0 0 0	0 1 1 1 1 1 0 1 0 1 1 1	1 1 0 0 0 1	C=0 →右移二位

乘积的符号为：$0\oplus1=1$，$[X\cdot Y]_{原}=1.010111110001$

3.1.2 乘法运算　　2. 补码乘法运算

补码乘法的运算规则

设被乘数X，乘数Y均为字长为n位的定点小数，且

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

（y_0为符号位的值，y_i为其他各位的值，2^i为各位的权）

1）当$Y\geqslant 0$，即$y_0=0$时，

$$Y=[Y]_{补}=0\cdot 2^0+y_{-1}\cdot 2^{-1}+\cdots\cdots+y_{-(n-2)}\cdot 2^{-(n-2)}+y_{-(n-1)}\cdot 2^{-(n-1)}$$

2）当$Y<0$，即$y_0=1$时，

$$[Y]_{补}=2+Y\ (\mathrm{mod}\ 2),$$

$$Y=-2+[Y]_{补}$$

$$=-2+y_0\cdot 2^0+y_{-1}\cdot 2^{-1}+\cdots+y_{-(n-2)}\cdot 2^{-(n-2)}+y_{-(n-1)}\cdot 2^{-(n-1)}$$

$$=-1\cdot 2^0+y_{-1}\cdot 2^{-1}+\cdots\cdots+y_{-(n-2)}\cdot 2^{-(n-2)}+y_{-(n-1)}\cdot 2^{-(n-1)}$$

$$\therefore\ Y=-y_0\cdot 2^0+y_{-1}\cdot 2^{-1}+\cdots\cdots+y_{-(n-2)}\cdot 2^{-(n-2)}+y_{-(n-1)}\cdot 2^{-(n-1)}$$

3.1.2 乘法运算　　2. 补码乘法运算

1）校正法

假定被乘数X和乘数Y为用补码表示的纯小数：

$[X]_{补}=X_0.X_{-1}X_{-2}\cdots X_{-(n-1)}$

$[Y]_{补}=Y_0.Y_{-1}Y_{-2}\cdots Y_{-(n-1)}$

X_0、Y_0为符号位

校正法补码一位乘法的算法公式：

$$[X\cdot Y]_{补}=[X]_{补}(-Y_0+0.Y_{-1}Y_{-2}\cdots Y_{-(n-1)})$$

$$=[X]_{补}(-Y_0 2^{0}+Y_{-1}2^{-1}+Y_{-2}2^{-2}+\cdots+Y_{-(n-1)}2^{-(n-1)})$$

注意事项

- 当乘数为正数时，不管被乘数符号如何，都可按原码乘法的规则进行运算(“加”和“移位”都按照补码规则进行)。
- 当乘数为负数时，补码乘法可以按照“先不考虑符号位，当作正数进行原码相乘，再在最后加上$[-x]_{补}$进行校正”的做法即可(x是被乘数，最后这步称为 校正)。
- 补码乘法中，乘积的符号位是在运算过程中自然形成的，与原码中的“符号位与数值部分分开计算”有着重要的区别。

3.1.2 乘法运算　2. 补码乘法运算

1）校正法

【例】已知

$X=-0.1101$

$Y=0.1011$

利用校正法补码一位乘法求积。

【解】

$[X]_{补}=11.0011$

$[Y]_{补}=00.1011$

$\therefore [X\cdot Y]_{补}=1.01110001$

符号	D	A	操 作
0 0	0 0 0 0	1 0 1 **1**	
＋ 1 1	0 0 1 1		＋$[X]_{补}$
1 1	0 0 1 1		
1 1	1 0 0 1	1 1 0 **1**	右移1位
＋ 1 1	0 0 1 1		＋$[X]_{补}$
1 0	1 1 0 0		
1 1	0 1 1 0	0 1 1 **0**	右移1位
＋ 0 0	0 0 0 0		＋0
1 1	0 1 1 0		
1 1	1 0 1 1	0 0 1 **1**	右移1位
＋ 1 1	0 0 1 1		＋$[X]_{补}$
1 0	1 1 1 0		
1 1	0 1 1 1	0 0 0 1	右移1位

3.1.2 乘法运算　2. 补码乘法运算

1）校正法

【例】已知

$X=-0.1101$

$Y=-0.1011$

利用校正法补码一位乘法求积。

【解】

$[X]_{补}=11.0011$

$[Y]_{补}=11.0101$

$\therefore [X\cdot Y]_{补}=0.10001111$

符号	D	A	操 作
0 0	0 0 0 0	0 1 0 **1**	
＋ 1 1	0 0 1 1		＋$[X]_{补}$
1 1	0 0 1 1		
1 1	1 0 0 1	1 0 1 **0**	右移1位
＋ 0 0	0 0 0 0		＋0
1 1	1 0 0 1		
1 1	1 1 0 0	1 1 0 **1**	右移1位
＋ 1 1	0 0 1 1		＋$[X]_{补}$
1 0	1 1 1 1		
1 1	0 1 1 1	1 1 1 **0**	右移1位
＋ 0 0	0 0 0 0		＋0
1 1	0 1 1 1		
1 1	1 0 1 1	1 1 1 1	右移1位
＋ 0 0	1 1 0 1		＋$[-X]_{补}$
0 0	1 0 0 0	1 1 1 1	

补码乘法的运算规则　2）布斯（Booth）法

$$Y=-y_0\cdot 2^0+y_{-1}\cdot 2^{-1}+\cdots\cdots+y_{-(n-2)}\cdot 2^{-(n-2)}+y_{-(n-1)}\cdot 2^{-(n-1)}$$

$$[X\cdot Y]_{补}$$

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

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

$$\because\ [-X]_{补}=-[X]_{补},\ \ y_{-1}\cdot 2^{-1}+\cdots+y_{-(n-2)}\cdot 2^{-(n-2)}+y_{-(n-1)}\cdot 2^{-(n-1)}\geqslant 0,$$

$$[X\cdot Y]_{补}$$

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

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

$$=[X]_{补}\cdot(-y_0\cdot 2^0+y_{-1}\cdot 2^0-y_{-1}\cdot 2^{-1}+y_{-2}\cdot 2^{-1}-y_{-2}\cdot 2^{-2}+\cdots$$

$$+y_{-(n-2)}\cdot 2^{-(n-3)}-y_{-(n-2)}\cdot 2^{-(n-2)}+y_{-(n-1)}\cdot 2^{-(n-2)}-y_{-(n-1)}\cdot 2^{-(n-1)}+0\cdot 2^{-(n-1)})$$

$$=[X]_{补}\cdot[(y_{-1}-y_0)\cdot 2^0+(y_{-2}-y_{-1})\cdot 2^{-1}+\cdots\cdots$$

$$+(y_{-(n-1)}-y_{-(n-2)})\cdot 2^{-(n-2)}+(0-y_{-(n-1)})\cdot 2^{-(n-1)}]$$

2）布斯（Booth）法

运算法则：

假定被乘数X和乘数Y为用补码表示的纯小数：

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

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

x_0、y_0为符号位

布斯法补码一位乘法的算法公式为：

$$[X\cdot Y]_{补}$$
$$=[X]_{补}[(y_{-1}-y_0)2^0+(y_{-2}-y_{-1})2^{-1}+(y_{-3}-y_{-2})2^{-2}$$
$$+\cdots+(y_{-(n-1)}-y_{-(n-2)})2^{-(n-2)}+(0-y_{-(n-1)})2^{-(n-1)}]$$

2）布斯(Booth)法

补码一位乘（Booth法）运算规律：

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

2）布斯(Booth)法

Booth 算法描述如下：

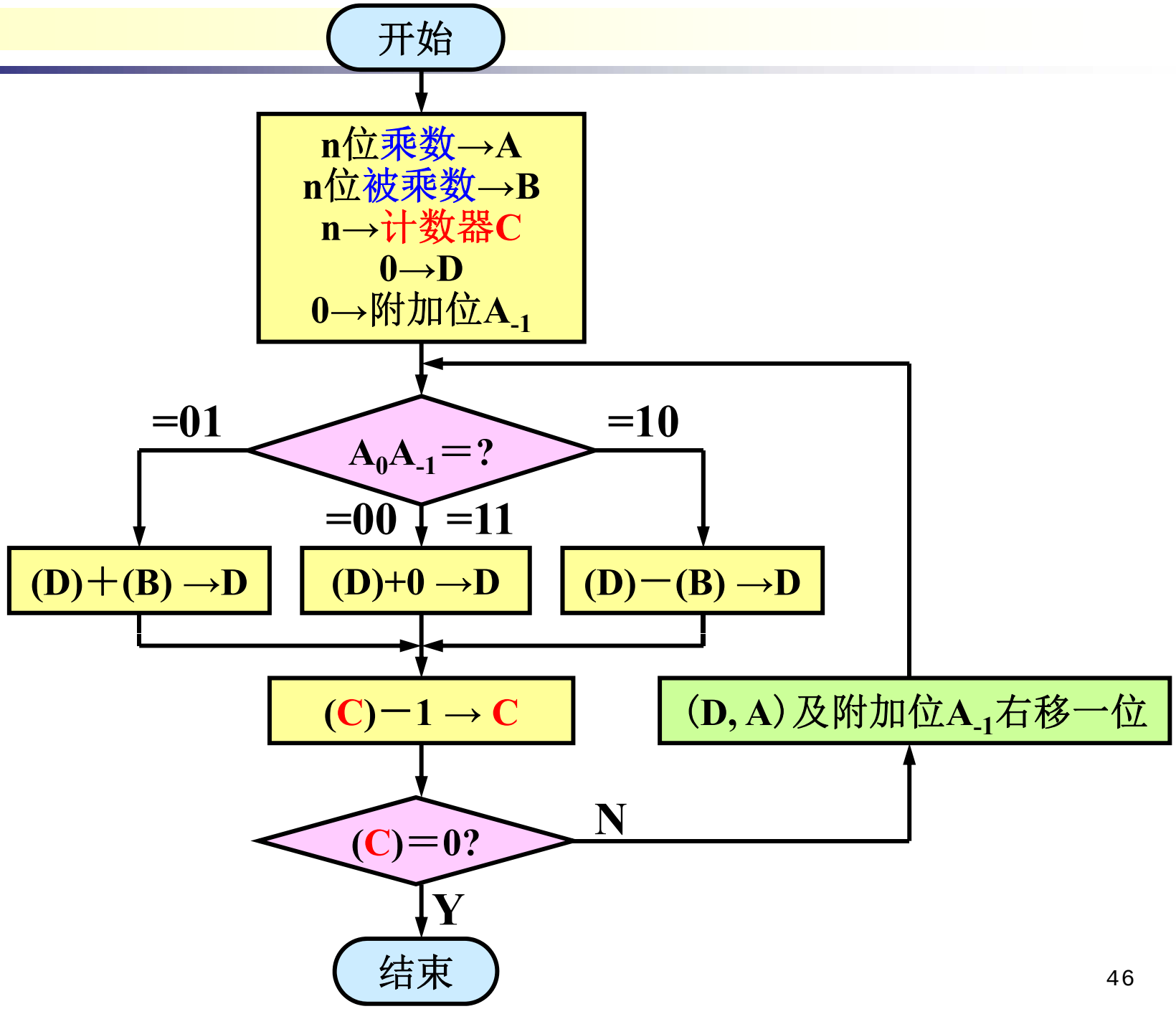
① 乘数与被乘数均用补码表示，连同符号位一起参加运算；运算结果(乘积)也是补码。

② 乘数最低位后增加一个附加位(可用A_{-1}表示)，初始设定为**0**。

③ 从附加位开始，按上表总共进行***n***次加操作、***n*-1**次右移操作(最后一次不右移)。

④ 右移按补码规则进行，即符号位复制。

补码一位乘(Booth法)的算法流程图



2）布斯（Booth）法

【例】

校正法1

校正法2

X＝0.1010

Y＝－0.1101

利用布斯法补码一位乘法求积。

【解】

$[X]_{补}$ ＝00.1010

$[-X]_{补}$ ＝11.0110

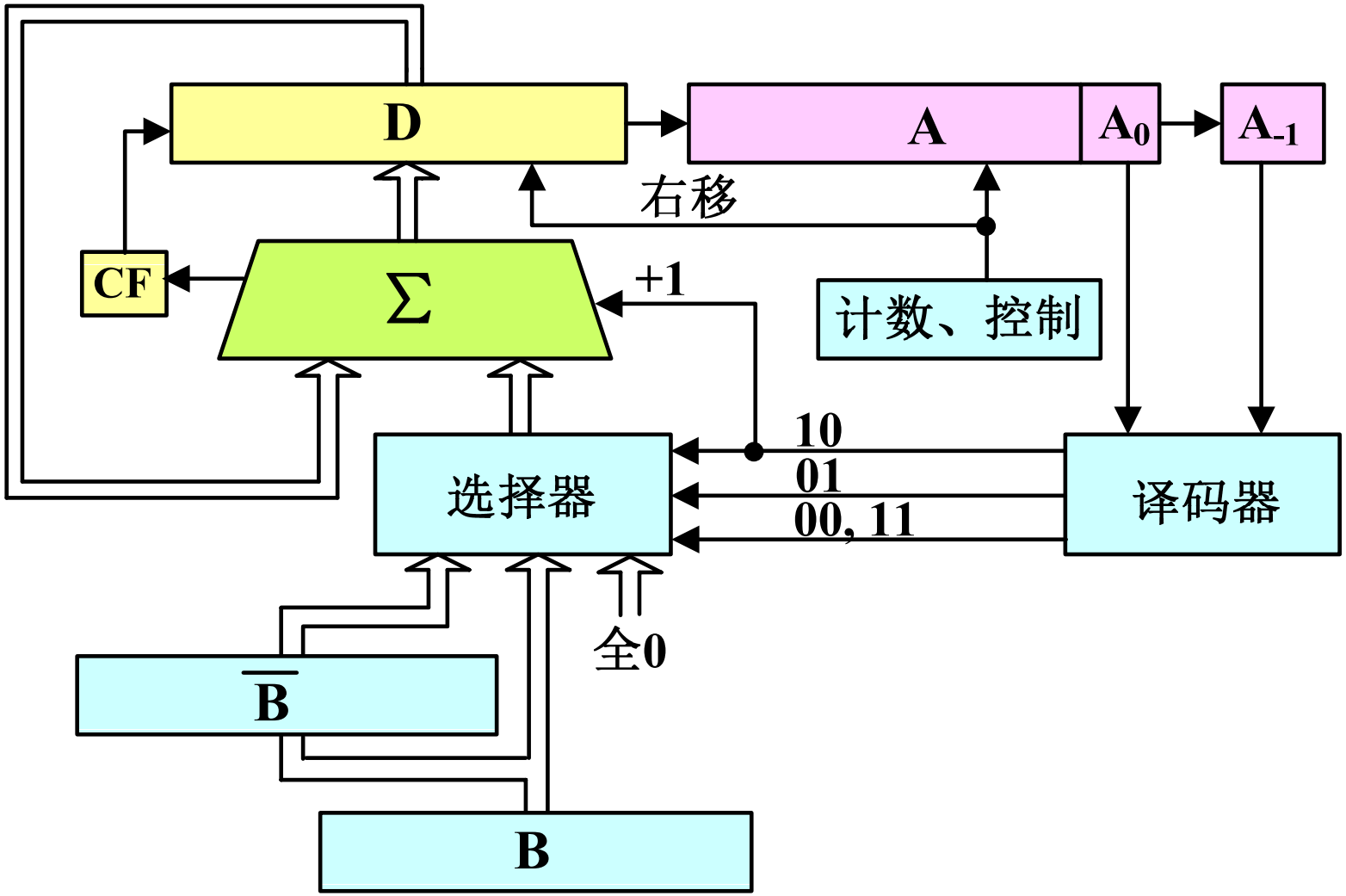
$[Y]_{补}$ ＝11.0011

∴ $[X \cdot Y]_{补}$

＝1.01111110

符号	D（部分积）	A	A_{-1}（附加位）	操作说明
0 0	0 0 0 0	1 0 0 1 1（乘数）	0	
1 1	0 1 1 0			＋$[-X]_{补}$
1 1	0 1 1 0			
1 1	1 0 1 1	0 1 0 0 1	1	右移1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1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1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1位
1 1	0 1 1 0			＋$[-X]_{补}$
1 1	0 1 1 1	1 1 1 0 1	0	不移位

2）布斯（Booth）法



补码一位乘法器（Booth算法）框图

3）补码二位乘法

Booth法改进，将Y_i与Y_{i-1}的状态比较和Y_{i-1}与Y_{i-2}的状态比较合在一起进行：

$$2(Y_{i-1}-Y_i)+(Y_{i-2}-Y_{i-1})=Y_{i-1}+Y_{i-2}-2Y_i$$

Y_i Y_{i-1} Y_{i-2}	$Y_{i-1}+Y_{i-2}-2Y_i$	操　作
0　0　0	0	+0，右移2位
0　0　1	1	$+[X]_{补}$，右移2位
0　1　0	1	$+[X]_{补}$，右移2位
0　1　1	2	$+2[X]_{补}$，右移2位
1　0　0	−2	$+2[-X]_{补}$，右移2位
1　0　1	−1	$+[-X]_{补}$，右移2位
1　1　0	−1	$+[-X]_{补}$，右移2位
1　1　1	0	+0，右移2位

3）补码二位乘法

补码二位乘法的法则：

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（可用A_{-1}），初始设定为0。

③ 从附加位开始，依据上表所示的操作规律，一次检测相邻3位决定具体的操作，并每次乘数右移2位。

④ 当乘数位数（包括符号位）为偶数n时，右移2位的次数为n/2次，最后一次只右移1位。

⑤ 当乘数位数（包括符号位）为奇数n时，可在乘数最后一位之后添加一个0，使乘数位数变为偶数n+1，右移次数为(n+1)/2，且最后一次只右移1位；此时，也可以将乘数增加一个符号位，使乘数位数变为偶数n+1，右移次数为[(n+1)/2－1]。

3）补码二位乘法

【例】已知

$X=-0.1101$

$Y=-0.1011$

试利用补码二位乘法求积。

【解】

$[Y]_{补}$ ＝ 11.0101

$[X]_{补}$ ＝111.0011

$2[X]_{补}$ ＝110.0110

$[-X]_{补}$ ＝000.1101

$2[-X]_{补}$ ＝001.1010

符号	D	A	A_{-1}	操作说明
0 0 0	0 0 0 0	1 1 0 1 0 1	0	
1 1 1	0 0 1 1			＋$[X]_{补}$
1 1 1	0 0 1 1			
1 1 1	1 1 0 0	1 1 1 1 0 1	0	右移2位
1 1 1	0 0 1 1			＋$[X]_{补}$
1 1 0	1 1 1 1			
1 1 1	1 0 1 1	1 1 1 1 1 1	0	右移2位
0 0 0	1 1 0 1			＋$[-X]_{补}$
0 0 0	1 0 0 0			
0 0 0	1 0 0 0	1 1 1 1 1 1	0	不右移

$\therefore [X\cdot Y]_{补}=0.10001111$

1）手算及单元电路

设$X=X_3X_2X_1X_0$，$Y=Y_3Y_2Y_1Y_0$，计算$X\cdot Y=?$

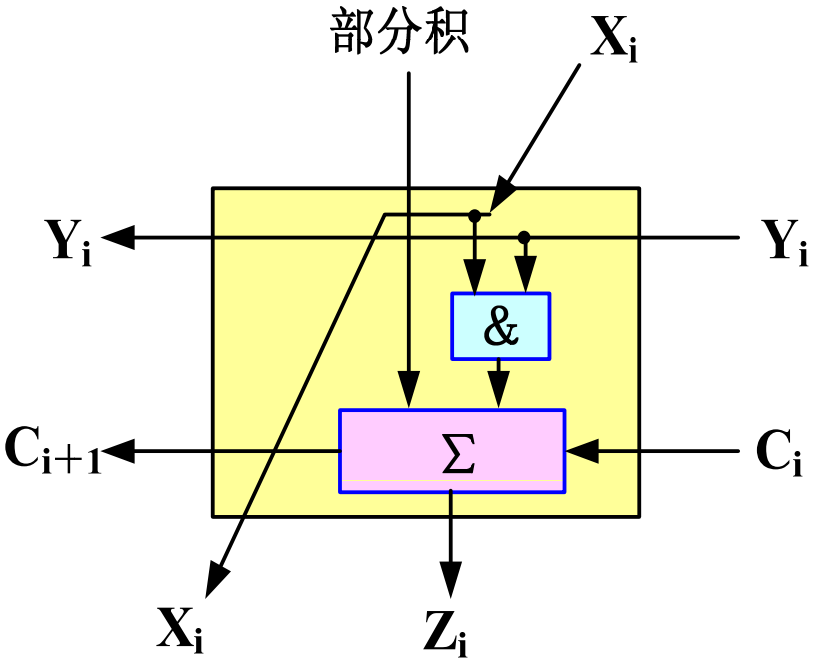
			X_3	X_2	X_1	X_0
		$\times$	Y_3	Y_2	Y_1	Y_0
			X_3Y_0	X_2Y_0	X_1Y_0	X_0Y_0
		X_3Y_1	X_2Y_1	X_1Y_1	X_0Y_1	
	X_3Y_2	X_2Y_2	X_1Y_2	X_0Y_2		
X_3Y_3	X_2Y_3	X_1Y_3	X_0Y_3			
Z_6	Z_5	Z_4	Z_3	Z_2	Z_1	Z_0

与运算

求和

1）手算及单元电路

设$X=X_3X_2X_1X_0$，$Y=Y_3Y_2Y_1Y_0$，计算$X\cdot Y=?$

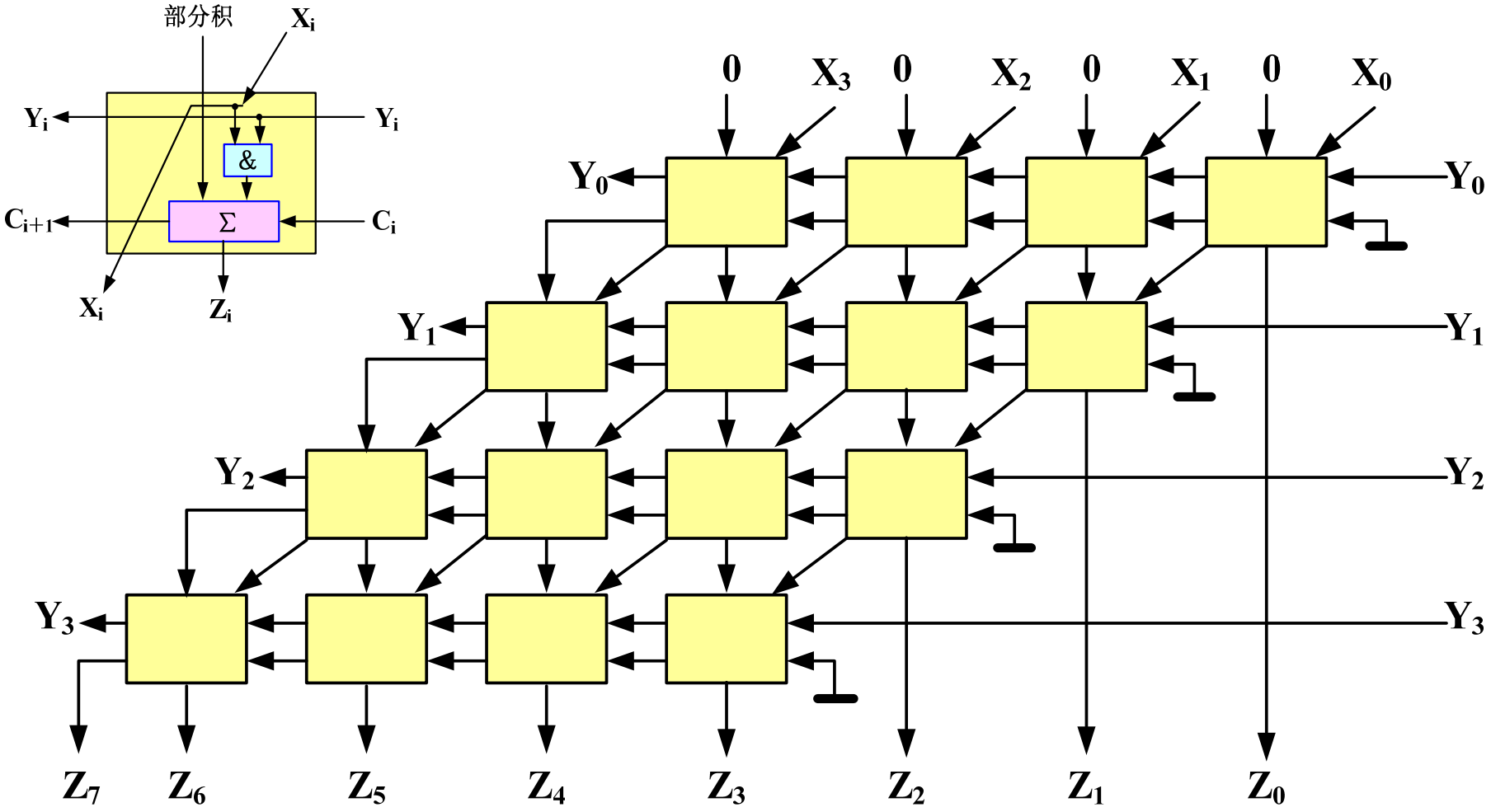


基本乘加单元框图

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

3.1.2 乘法运算　3. 阵列乘法器

2）绝对值(无符号数)阵列乘法器

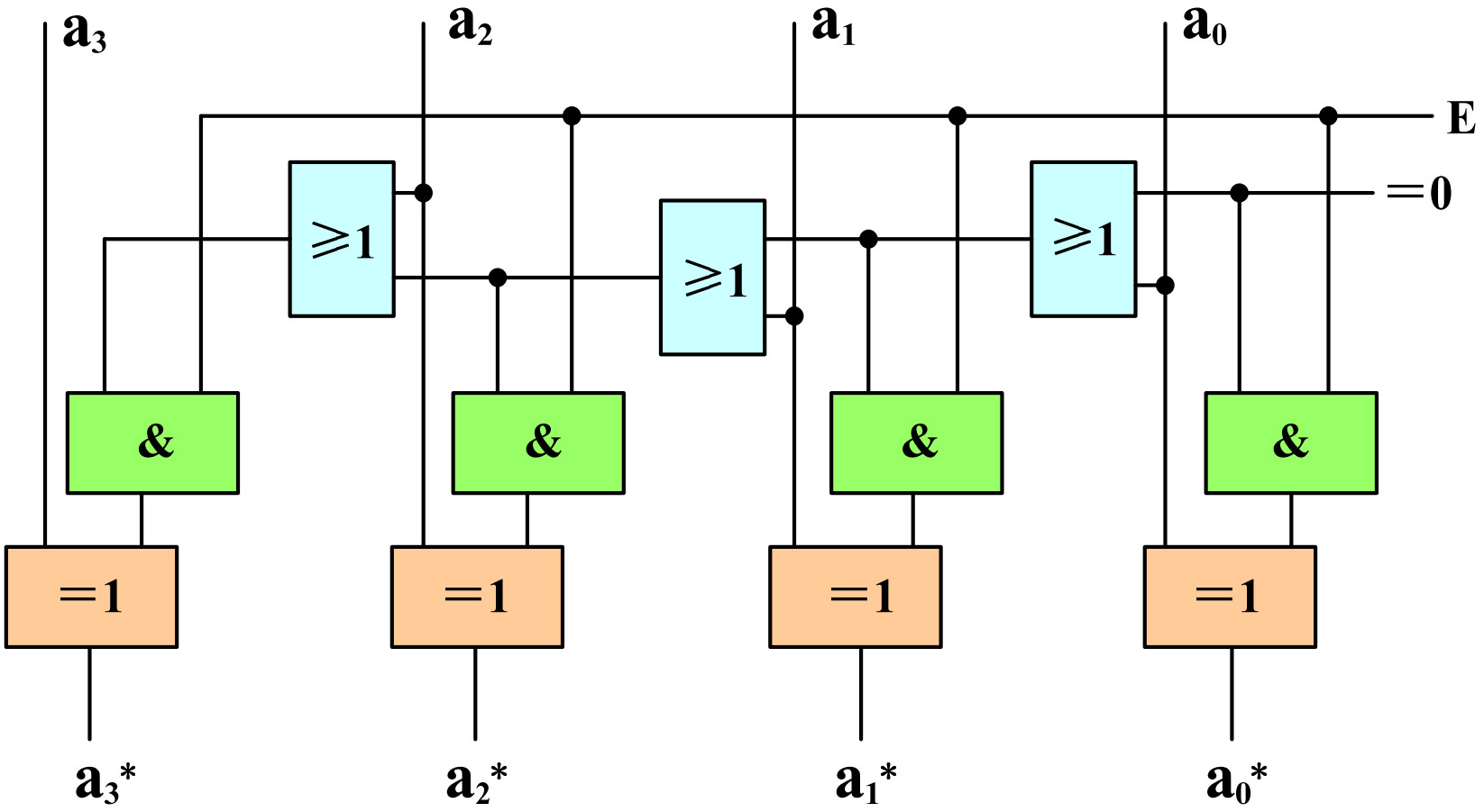


定点无符号数阵列乘法器

3）带符号数的阵列乘法器

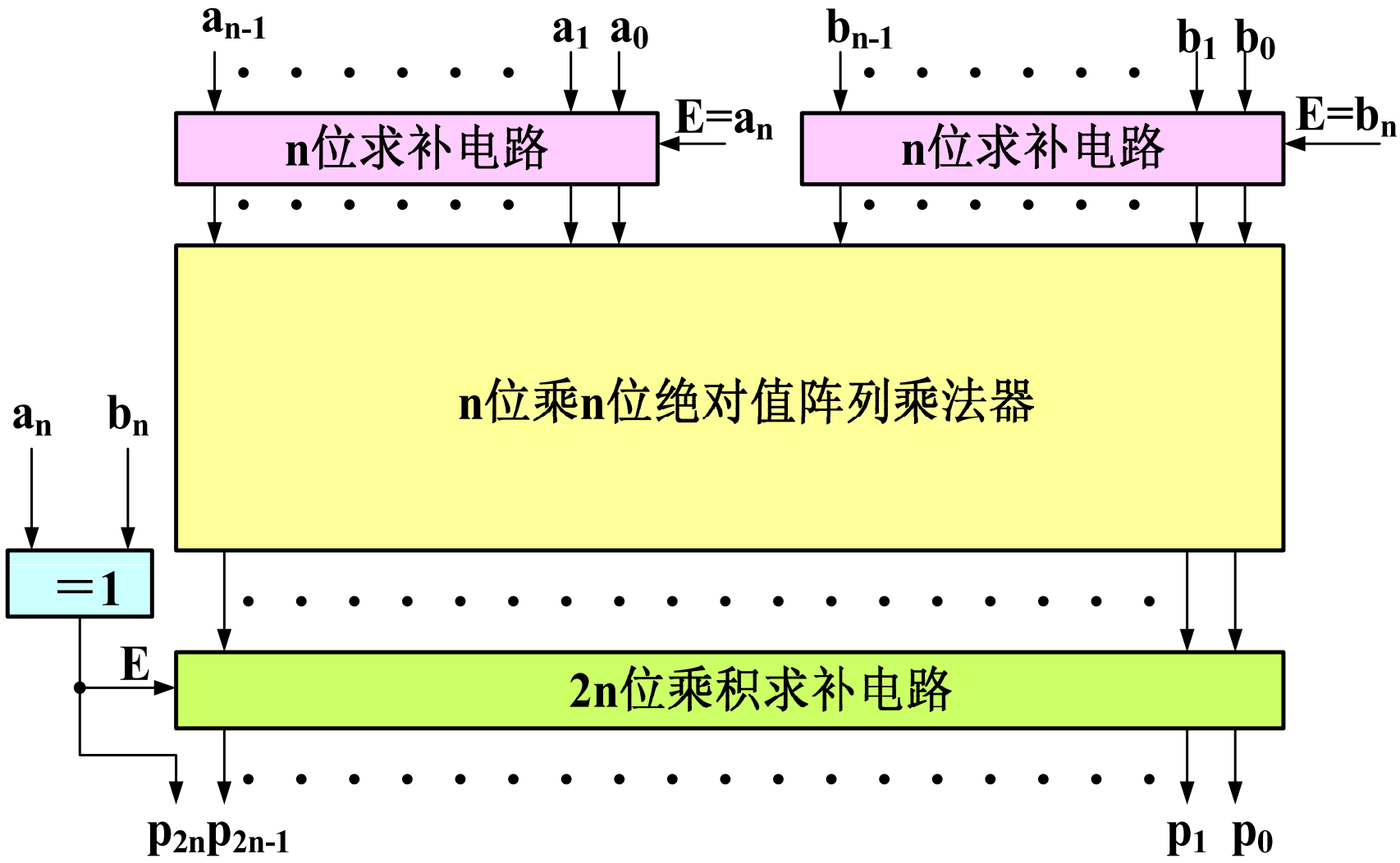
- 求被乘数与乘数的绝对值
- 进行绝对值乘法
- 根据被乘数与乘数的符号，决定最后乘积的符号。

3）带符号数的阵列乘法器



求补电路

3）带符号数的阵列乘法器



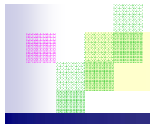
n+1位带符号数阵列乘法器框图

作业：

Page83：

13，17（1）（2），

20（1） Booth法

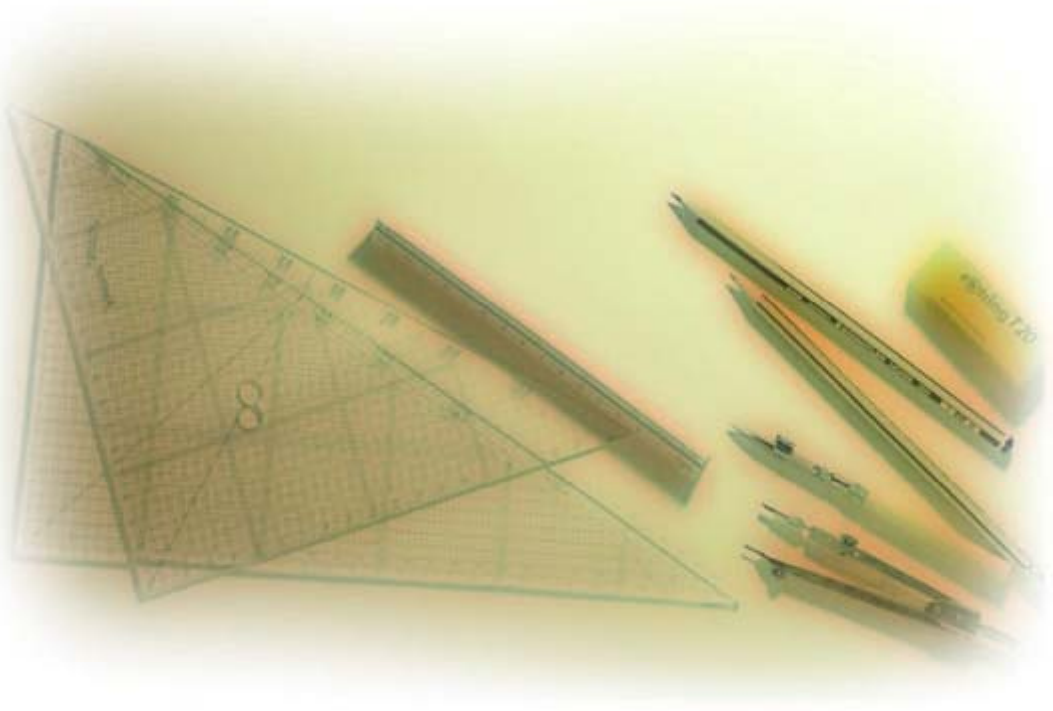


3.1 定点数运算

3.1.3 除法运算

3.1.3 除法运算

- 实现方式：
  - 原码
  - 补码
- 前提条件：
  - 除数不能为0
  - 商可以表示



3.1.3 除法运算　　1. 原码除法运算

1）原码除法的法则

① 前提条件：

- 除数≠0；
- 定点纯小数时，|被除数|＜|除数|；
- 定点纯整数时，|被除数|＞|除数|。

② 商的符号＝被除数的符号⊕除数的符号

③ |商|＝|被除数|÷|除数|

④ 将商的符号与商的值拼接在一起。

3.1.3 除法运算　　1. 原码除法运算

1）原码除法的法则

【例】设X＝0.1011，Y＝0.1101，求X÷Y＝？

【解】

$X \div Y = 0.1101$

余数$= 0.0111 \times 2^{-4}$

商的符号$= 0 \oplus 0 = 0$

```
                0.1101
          ┌──────────
.1101 ) .1011 0
          1101
          ────
          1001 0
           1101
           ────
           0101 00
             1101
             ────
          .0000 0111
```

- 被除数(余数)每次减去右移一次后的除数，决定上商。
- 实际构成除法器时：
  - 保持除数的位置不动，而每次余数左移一次。
  - 在CPU中，必须减过之后方能判断余数是否够减，当发现不够减时，在下面操作之前必须恢复余数。

2）恢复余数法

定点纯小数：

- 符号位单独处理。
- 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)。
- 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)。
- 重复上面的过程直到除尽或精度达到要求。
- 拼接商符得到商。

2）恢复余数法

【例】

被除数 X＝－0.10001011

除数 Y＝0.1110

利用原码恢复余数法求商及余数。

【解】

前提条件：|X|＜|Y|，|Y|≠0。

$[X]_{原}$＝1.10001011

$[Y]_{原}$＝0.1110

商符＝1⊕0＝1

绝对值除法过程：

3.1.3 除法运算　　1. 原码除法运算

$[X]_{原}=1.10001011$

$[Y]_{原}=0.1110$

$[-Y]_{补}=1.0010$

商符$=1\oplus0=1$

绝对值除法过程：

商$=1.1001$

余数

$=1.1101\times2^{-4}$

余数的符号与被除数一致。

符号	被除数(余数)	商	操作
0 0	1 0 0 0 1 0 1 1	0	
0 1	0 0 0 1 0 1 1 0		左移1位
1 1	0 0 1 0		−|Y|
0 0	0 0 1 1 0 1 1 0	1	够减，商为1
0 0	0 1 1 0 1 1 0 1		左移1位
1 1	0 0 1 0		−|Y|
1 1	1 0 0 0 1 1 0 1	0	不够减，商为0
0 0	1 1 1 0		+|Y|
0 0	0 1 1 0 1 1 0 1	0	恢复余数
0 0	1 1 0 1 1 0 1 0		左移1位
1 1	0 0 1 0		−|Y|
1 1	1 1 1 1 1 0 1 0	0	不够减，商为0
0 0	1 1 1 0		+|Y|
0 0	1 1 0 1 1 0 1 0	0	恢复余数
0 1	1 0 1 1 0 1 0 0		左移1位
1 1	余数	商	−|Y|
0 0	1 1 0 1 0 1 0 0	1	够减，商为1

2）恢复余数法

- 恢复余数法：不同的被除数和除数，其除的过程不规范，何时需恢复余数是不相同的，实现起来不便于控制。
- 加减交替法：

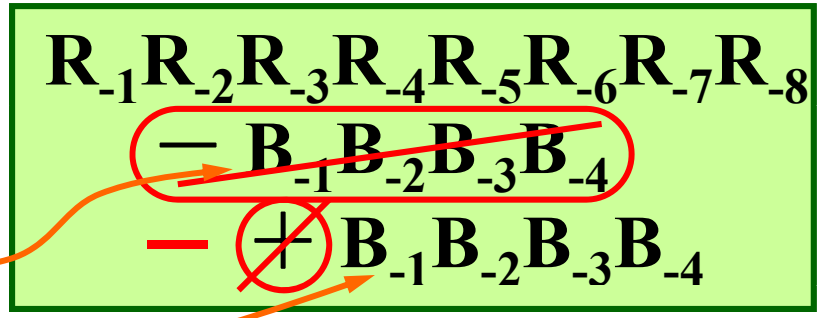
3）加减交替法

分析恢复余数法：

- 第 *i* 次余数减除数B，得余数R；
- 若R＜0，应：
  - 恢复余数，执行(R＋B)；
  - 左移一位，即2(R+B)；
  - 进行第 *i* +1 次余数减除数B操作，即：

$$2(R+B)-B = 2R+B$$

左移一位　恢复之后的余数　第 *i* +1 次减除数操作



3）加减交替法

加减交替法的运算法则：

- 若余数R≥0, 则商上1，余数左移一次，减除数；
- 若余数R＜0, 则商上0，余数左移一次，加除数。

3.1.3 除法运算　1. 原码除法运算

3）加减交替法

【例】

X＝－0.10001011

Y＝0.1110

利用原码加减交替法求商及余数。

【解】

$[X]_{原}$＝1.10001011

$[Y]_{原}$＝0.1110

$[-Y]_{补}$＝1.0010

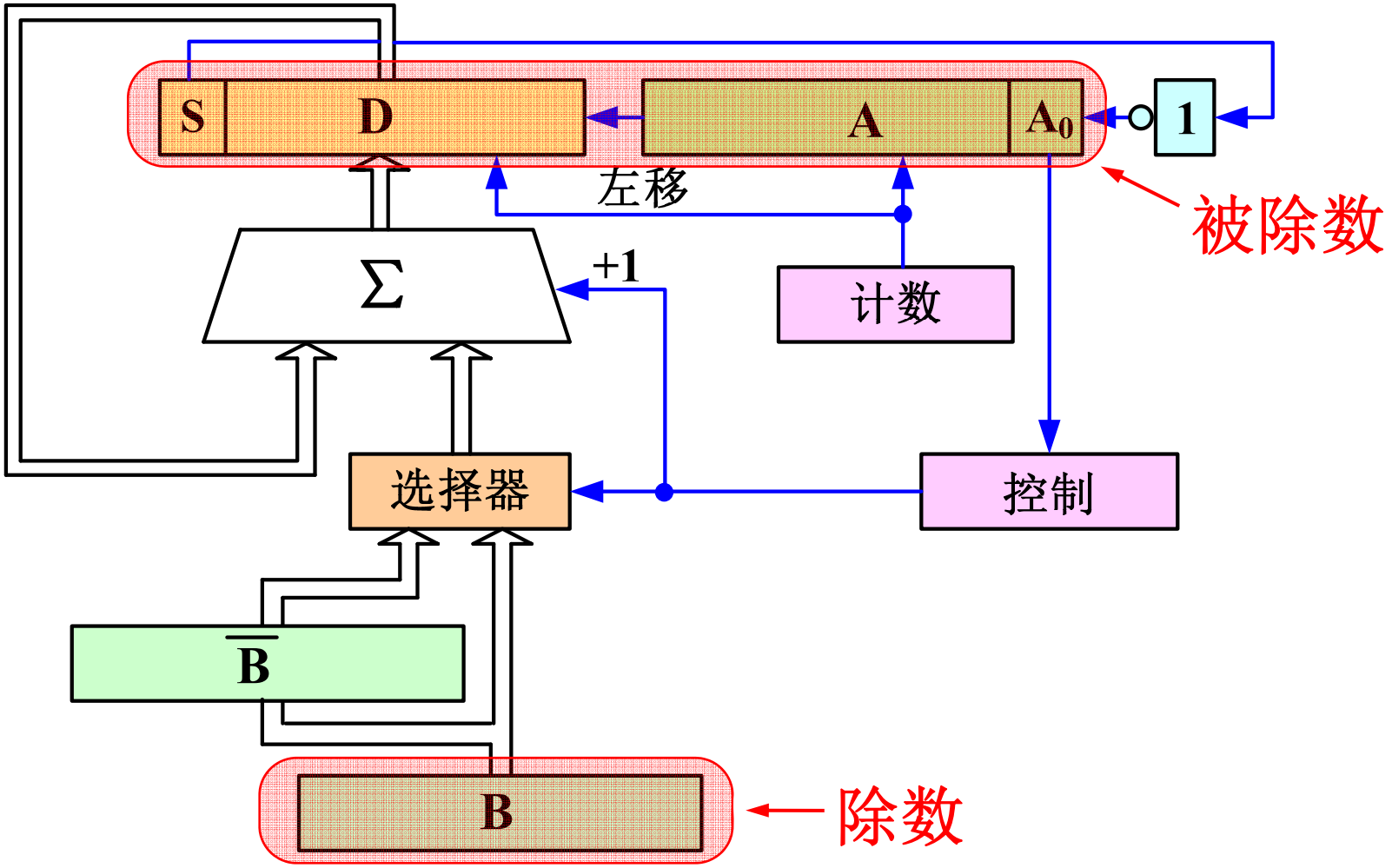
商符＝1⊕0＝1

$[X \div Y]_{原}$＝1.1001

余数＝1.1101×2^{-4}

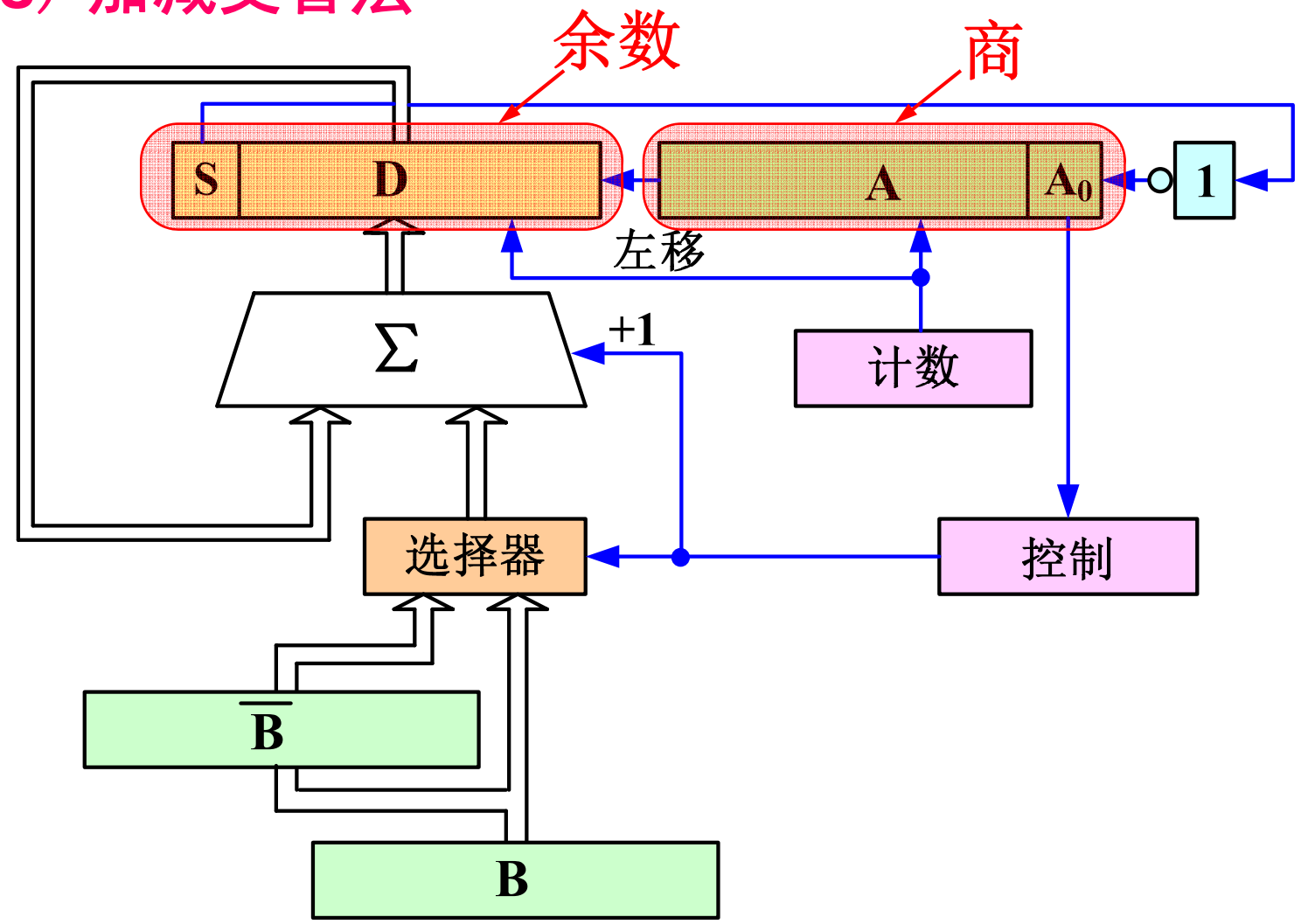
符号	被除数(余数)	商	操作
0 0	1 0 0 0 1 0 1 1	0	
0 1	0 0 0 1 0 1 1 0		左移1位
1 1	0 0 1 0		－|Y|
0 0	0 0 1 1 0 1 1 0	1	R≥0，商为1
0 0	0 1 1 0 1 1 0 1		左移1位
1 1	0 0 1 0		－|Y|
1 1	1 0 0 0 1 1 0 1	0	R＜0，商为0
1 1	0 0 0 1 1 0 1 0		左移1位
0 0	1 1 1 0		＋|Y|
1 1	1 1 1 1 1 0 1 0	0	R＜0，商为0
1 1	1 1 1 1 0 1 0 0		左移1位
0 0	1 1 1 0		＋|Y|
0 0	1 1 0 1 0 1 0 0	1	R≥0，商为1

3）加减交替法



加减交替法除法器框图

3）加减交替法



加减交替法除法器框图

1）补码除法法则

- 先决条件：
  - 定点纯小数
  - 除数≠0
  - |被除数|＜|除数|
- 补码除法的法则：

3.1.3 除法运算　　2. 补码除法运算

1）补码除法法则

- 补码除法的法则：

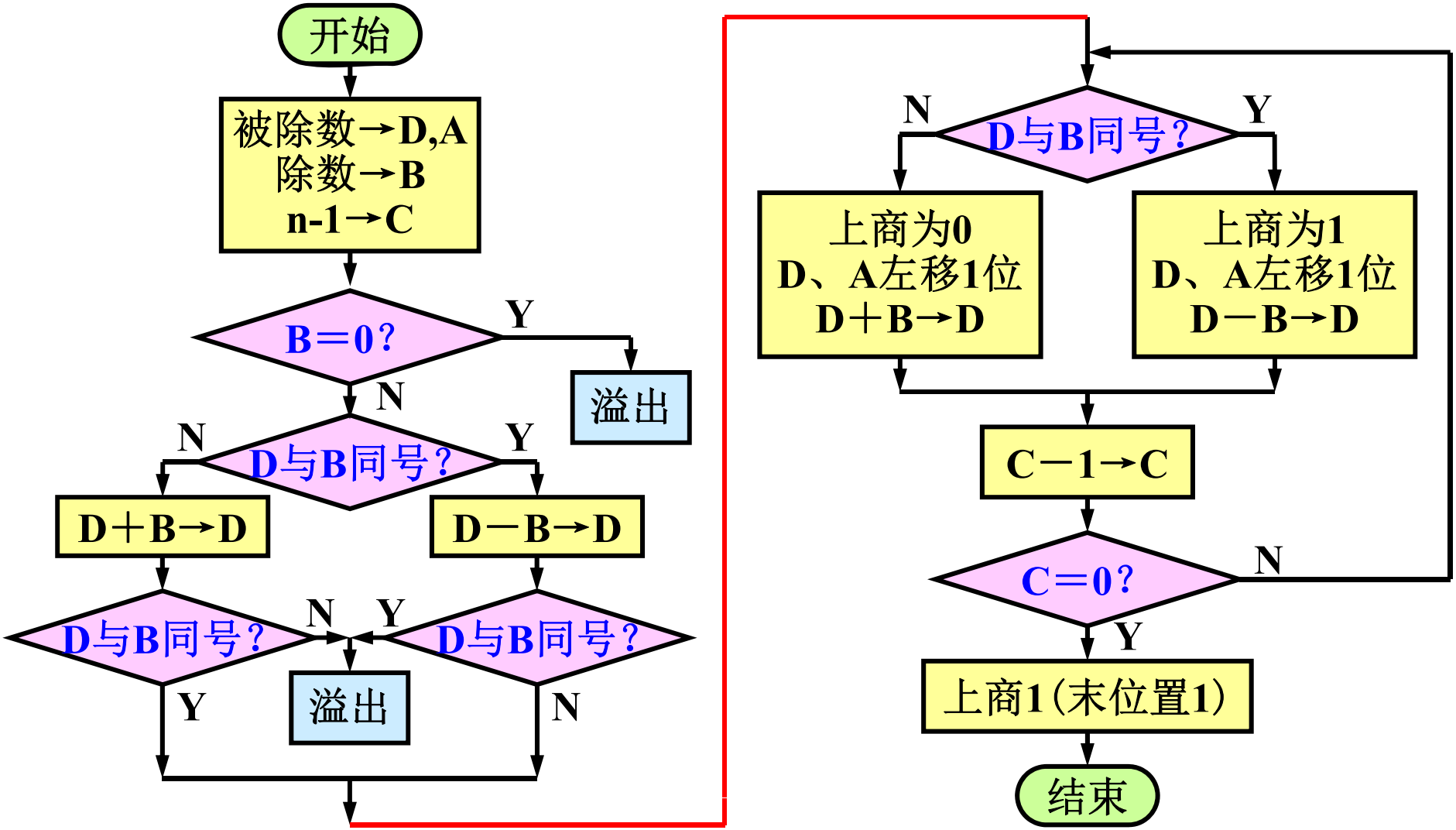
① 如果被除数与除数同号，被除数减除数；
如果被除数与除数异号，被除数加除数。
运算结果称为余数。

② 若余数与除数同号，上商为1，余数左移一位，
下次用余数减除数操作求商；
若余数与除数异号，上商为0，余数左移一位，
下次用余数加除数操作求商。

③ 重复②直至除尽或达到精度要求。

④ 商修正。在除不尽时，通常可用商的最低位恒置1进行修正来保证精度。

2）补码除法流程框图


开始
被除数→D,A
除数→B
n-1→C
B＝0?
Y
溢出
N
N
D与B同号?
Y
D＋B→D
D－B→D
D与B同号?
N
Y
D与B同号?
Y
溢出
N
N
D与B同号?
Y
上商为0
D、A左移1位
D＋B→D
上商为1
D、A左移1位
D－B→D
C－1→C
C＝0?
N
Y
上商1(末位置1)
结束

3）补码除法的过程

【例】

X＝－0.10001011

Y＝0.1110

利用补码除法求商及余数。

【解】

$[X]_{补}$＝1.01110101

$[Y]_{补}$＝0.1110

$[-Y]_{补}$＝1.0010

【解】

$[X]_{补}$ ＝1.01110101

$[Y]_{补}$ ＝0.1110

$[-Y]_{补}$ ＝1.0010

$[商]_{补}$ ＝1.01101

$[余数]_{补}$ ＝1.0011 ×2^{-4}

符号	被除数(余数)	商	操 作
1 1	0 1 1 1 0 1 0 1		X、Y异号
0 0	1 1 1 0		＋$[Y]_{补}$
0 0	0 1 0 1	1	R与Y同号，上商1
0 0	1 0 1 0 1 0 1 1		左移1位，下步减
1 1	0 0 1 0		＋$[-Y]_{补}$
1 1	1 1 0 0	0	R与Y异号，上商0
1 1	1 0 0 1 0 1 1 0		左移1位，下步加
0 0	1 1 1 0		＋$[Y]_{补}$
0 0	0 1 1 1	1	R与Y同号，上商1
0 0	1 1 1 0 1 1 0 1		左移1位，下步减
1 1	0 0 1 0		＋$[-Y]_{补}$
0 0	0 0 0 0	1	R与Y同号，上商1
0 0	0 0 0 1 1 0 1 1		左移1位，下步减
1 1	0 0 1 0		＋$[-Y]_{补}$
1 1	1	0	R与Y异号，上商0
1 1	0 0 1 1	1 0 1 1	

余数

商

【解】

$[X]_{补}$ ＝1.01110101

$[Y]_{补}$ ＝0.1110

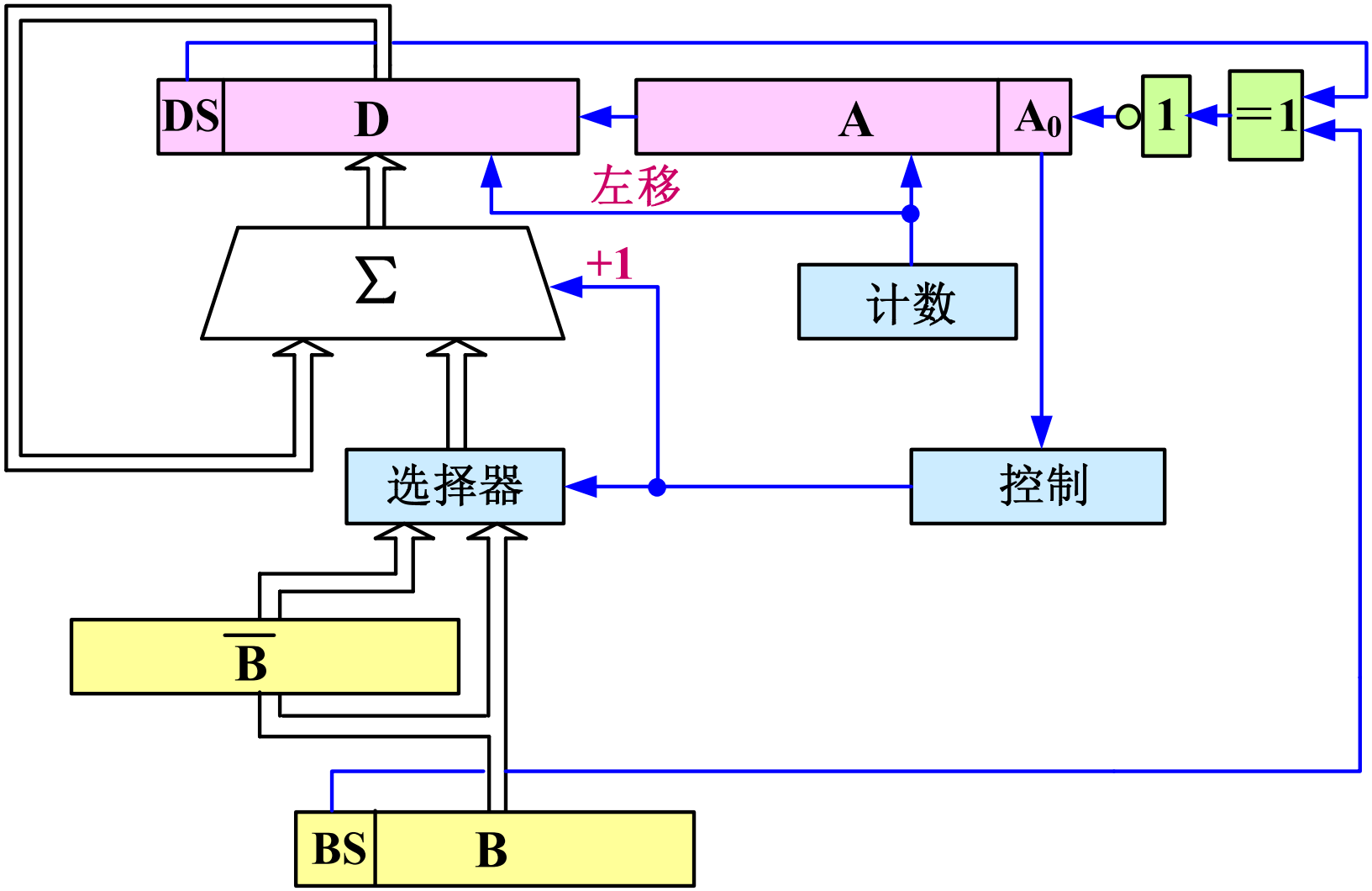
$[-Y]_{补}$ ＝1.0010

$[商]_{补}$ ＝1.0111

符号	被除数(余数)	商	操 作
1 1	0 1 1 1 0 1 0 1		X、Y异号
0 0	1 1 1 0		＋$[Y]_{补}$
0 0	0 1 0 1	1	R与Y同号，上商1
0 0	1 0 1 0 1 0 1 1		左移1位，下步减
1 1	0 0 1 0		＋$[-Y]_{补}$
1 1	1 1 0 0	0	R与Y异号，上商0
1 1	1 0 0 1 0 1 1 0		左移1位，下步加
0 0	1 1 1 0		＋$[Y]_{补}$
0 0	0 1 1 1	1	R与Y同号，上商1
0 0	1 1 1 0 1 1 0 1		左移1位，下步减
1 1	0 0 1 0		＋$[-Y]_{补}$
0 0	0 0 0 0	1	R与Y同号，上商1
0 0	0 0 0 1 1 0 1 1		左移1位
		1 0 1 1	1	末位恒置1

余数　　商

4）补码除法器框图



补码除法器框图

3.1.3 除法运算　　3. 阵列除法器

1）基本概念：补码运算的进位

- 在做无符号数减法时，用
  被减数＋$[减数]_{求补}$ 来实现。
  - 若被减数＜减数(不够减)，没有进位(借位)；
  - 若被减数＞减数(够减)，有进位(借位)。

【例】计算65－32、32－65。

【解】

$65 = 01000001_2$

$[-65]_{补} = 10111111_2$

$32 = 00100000_2$

$[-32]_{补} = 11100000_2$

```
  01000001
+ 11100000
----------
 100100001
```

有进位

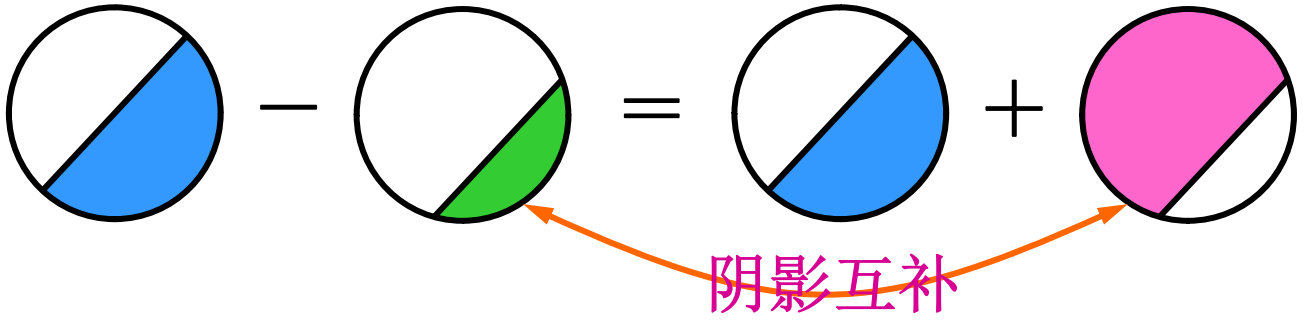
```
  00100000
+ 10111111
----------
  11011111
```

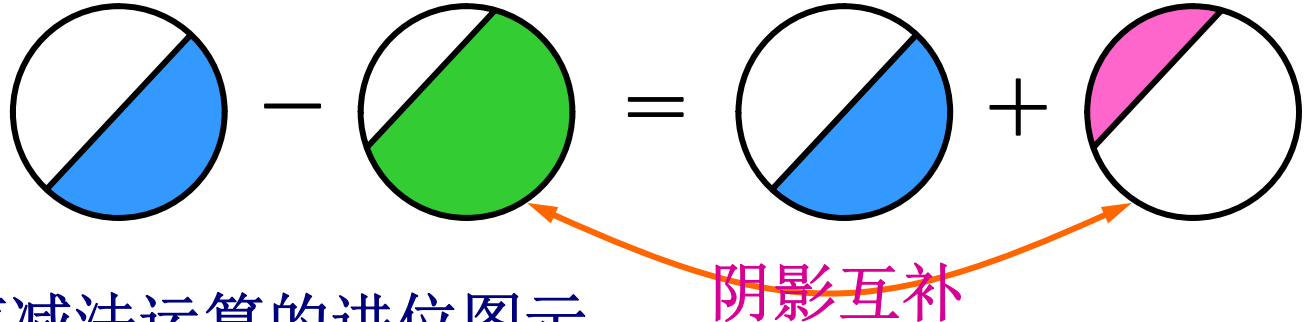
无进位

3.1.3 除法运算　　3. 阵列除法器

1）基本概念：补码运算的进位

- 在做无符号数减法时，用
  被减数＋[减数]$_{求补}$ 来实现。
  - 若被减数＜减数(不够减)，没有进位(借位)；
  - 若被减数＞减数(够减)，有进位(借位)。

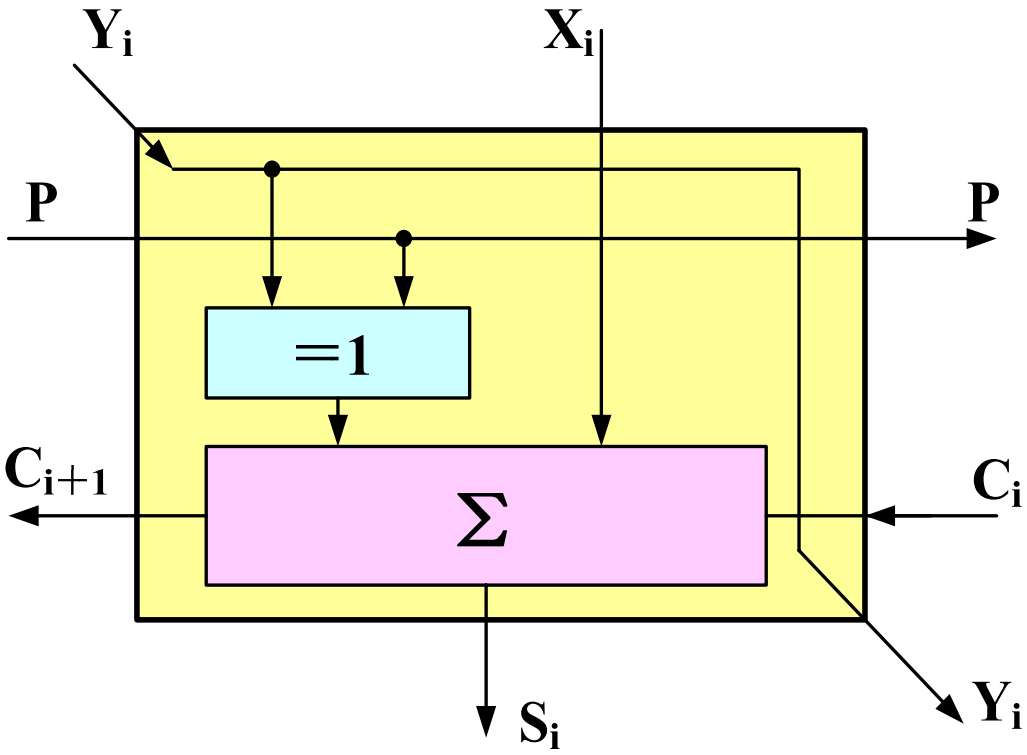




有模减法运算的进位图示

1）基本概念：可控加减单元CAS

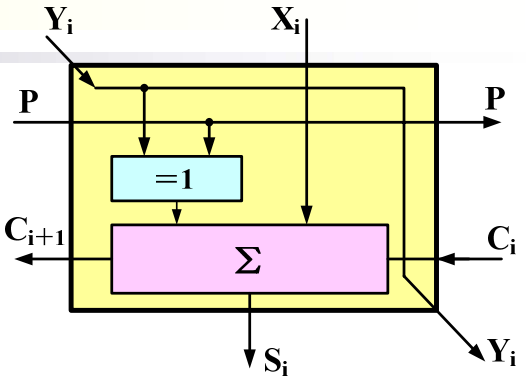
- 异或电路
- 全加器



3.1.3 除法运算　3. 阵列除法器

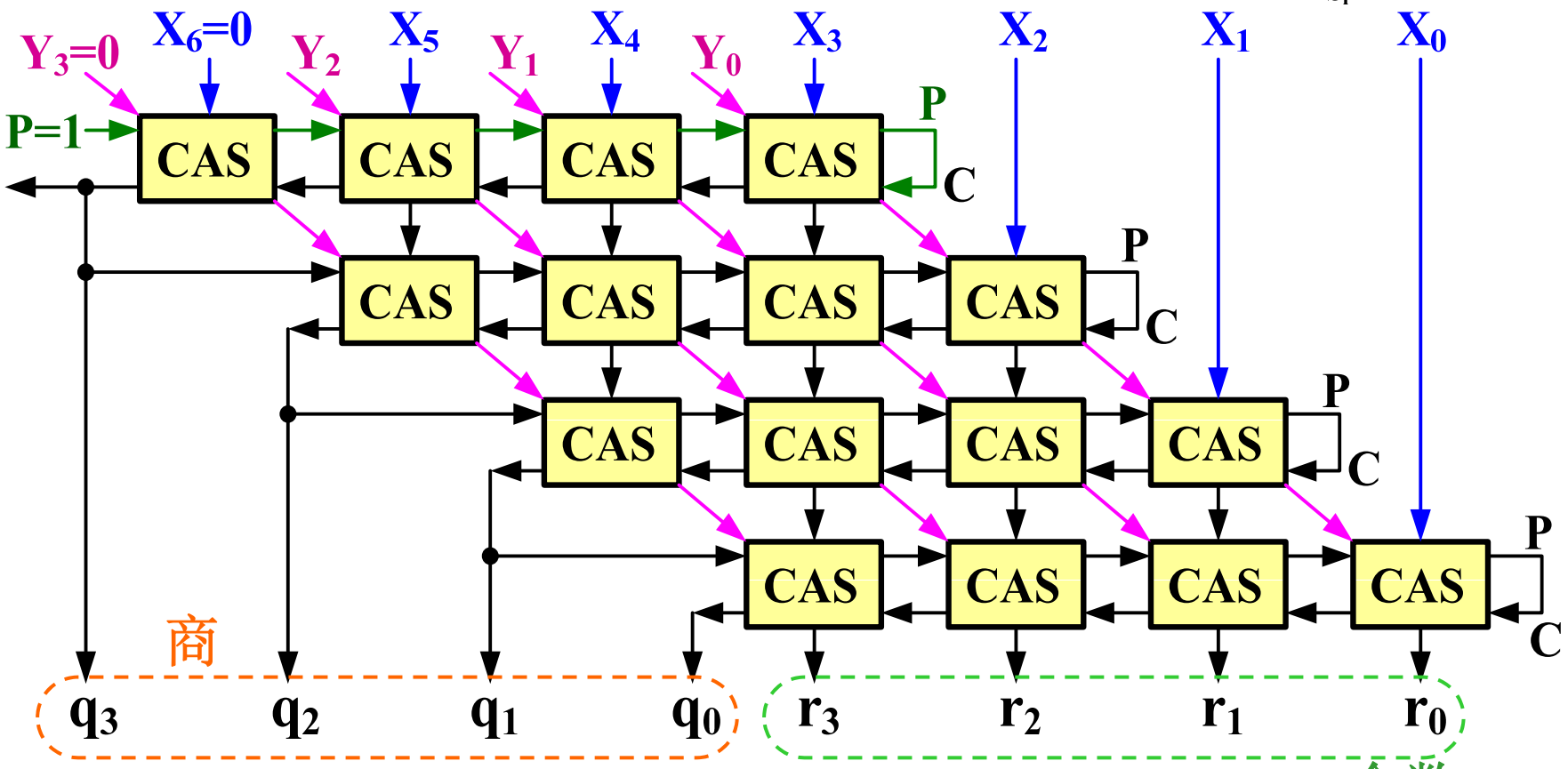
2）无符号数阵列除法器

可控加减单元CAS：



被除数：$X_6X_5X_4X_3X_2X_1X_0$

除数：$Y_3Y_2Y_1Y_0$



由可控加减单元CAS构成的阵列除法器

作业：

Page84：

22（1） 原码加减交替法