计算机组成原理

第三章 运算方法与运算器

3.1 定点数运算及溢出检测

1 定点数加法运算

$$[X]_{补}+[Y]_{补}=[X+Y]_{补} \mod 2^{n+1}$$

- 算法理解

例1 已知 X=+ 10010 Y= -10101 求X+Y

解: $[X]_{补}=010010$ $[Y]_{补}=101011$

$[X+Y]_{补}=[X]_{补}+[Y]_{补}=010010+101011$

$=111101$

所以: X+Y= - 00011

2 定点数减法运算

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

- 算法理解

例2 已知 $[Y]_{补}=10011$ 求 $[-Y]_{补}$

01101

解： $[Y]_{补}=10011$

$\therefore\ Y=-1101 \qquad -Y=1101$

$\therefore\ [-Y]_{补}=01101$ ✓

对比 $[Y]_{补}=10011$

可知 ： 通过右向左扫描$[Y]_{补}$，在遇到数字1及之前，直接输出遇到的数字，遇到1之后，取反输出，即可得到$[-Y]_{补}$，反之亦然！

2 定点减法运算

例3 已知 $X=+10101$ $Y=+10010$ 求X-Y

解: $[X]_{补}=010101$, $[Y]_{补}=010010$, $[-Y]_{补}=101110$

$[X-Y]_{补}=[X]_{补}+[-Y]_{补}=010101+101110$

$=1\ 000011$

所以: $X-Y=+000011$

3 数溢出的概念及其判断方法

1)溢出的概念

运算结果超出了某种数据类型的表示范围。

例4 已知 X=+ 10010 Y= +10101 求X+Y

解: $[X]_{补}=010010$ $[Y]_{补}=010101$

$[X+Y]_{补}=[X]_{补}+[Y]_{补}=010010+010101$

$=100111$

所以: X+Y= - 11001

两个正数之和为负数!

3 溢出的概念及其判断方法

例5 已知 X=- 10010 Y= -10101 求X+Y

解: $[X]_{补}=101110$ $[Y]_{补}=101011$

$[X+Y]_{补}=[X]_{补}+[Y]_{补}=101110+101011$

$=1\ 010001$

所以: X+Y= + 010001

两个负数之和为正数!

3 溢出的概念及其判断方法

2) 溢出的检测方法

- 溢出只可能发生在同符号数相加时，包括$[X]_{补}$与$[Y]_{补}$； $[X]_{补}$与$[-Y]$同号；

(1) 方法1：对操作数和运算结果的符号位进行检测

当结果的符号位与操作数的符号不相同时就表明发生了溢出

(设X0 ，Y0 为参加运算数的符号位， S0 为结果的符号位)

$$V = X_0Y_0\overline{S_0} + \overline{X_0}\,\overline{Y_0}S_0$$

当V=1时，运算结果溢出，根据该逻辑表达式，容易画出相应电路。

3 溢出的概念及其判断方法

(2)方法2：对最高数据位进位和符号进位进行检测

- 设运算时最高数据位产生的进位为C_1，符号位产生的进位为C_0，

溢出检测电路为： $V = C_0 \oplus C_1$

$$\begin{array}{r} 0.X_1 \\ +\ 0.Y_1 \\ \hline \end{array} \qquad\qquad \begin{array}{r} 1.X_1 \\ +\ 1.Y_1 \\ \hline \end{array}$$

此时：$C_0 = 0$,若$C_1 = 1$ 则改变了结果符号位，发生溢出。

此时：$C_0 = 1$,若$C_1 = 0$ 则改变了结果符号位，发生溢出。

3 溢出的概念及其判断方法

(3)方法3：用变型补码

$[X]_{补} = X_{f1}X_{f2}.X_1X_2X_3\ldots Xn \quad \bmod 2^{n+2}$

溢出的判断: $V = X_{f1} \oplus X_{f2}$

例6 已知 X=- 10010 Y= -10101 求X+Y

解: $[X]_{补}=1101110$ $[Y]_{补}=1101011$

$[X+Y]_{补}=[X]_{补}+[Y]_{补}=1101110+1101011$

$=1\ 10\ 10001$

$V=1\oplus 0=1$ 故发生溢出！

上述三种方法可基于逻辑表达式画出相应电路，

在后面的运算器部分，还将具体讲解

3 溢出的概念及其判断方法

(4) 溢出判断的软件方法

```
int tadd_ok(int x,int y) {
    int sum=x+y;
    int neg_over=x<0&&y<0&&sum>=0;
    int pos_over=x>=0&&y>=0&&sum<0;
    return !neg_over&&!pos_over; }
```

体会软/硬件功能的等效性和差异性！

体会软/硬协同的系统观！

4 无符号数运算的溢出判断

- 无符号数加法的溢出可用ALU的进位表示
- 无符号数减法的溢出也可用带加/减功能的ALU的进位取反后表示。

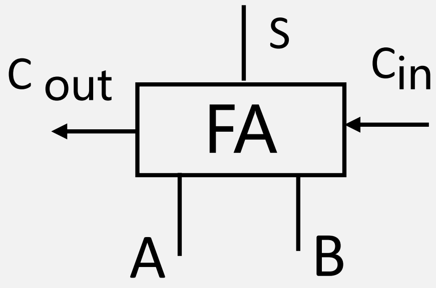
计算机组成原理

第三章 运算方法与运算器

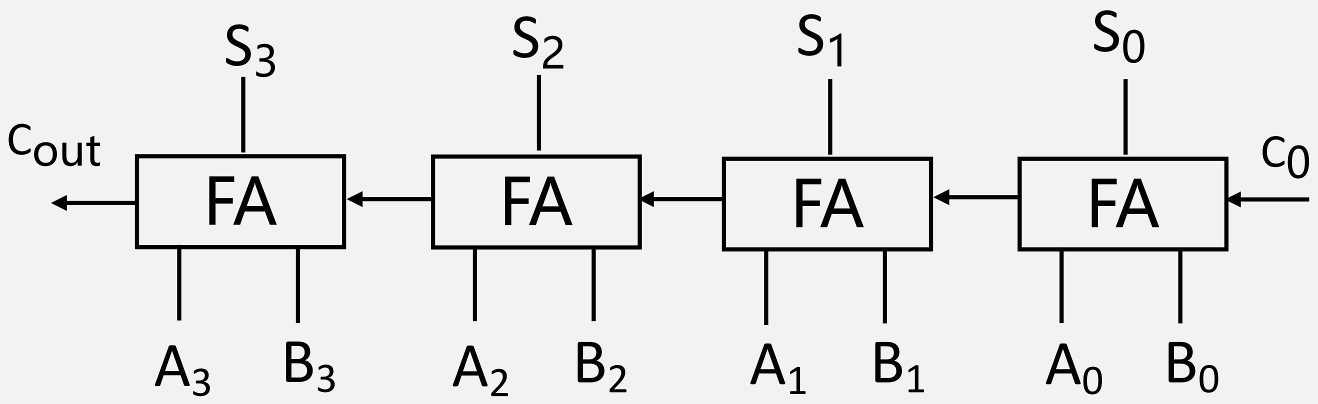
3.2 定点数补码加、减运算器设计

3.2 定点数补码加、减运算器设计

1 四位串行加法器的设计（基于一位全加器FA）



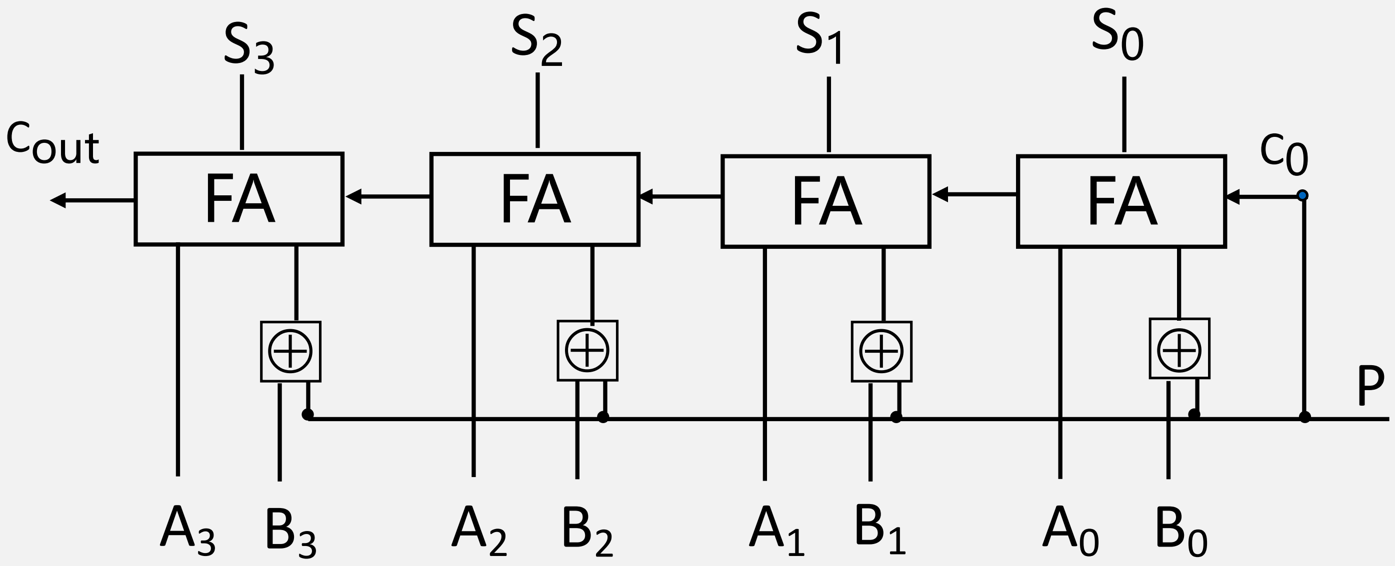
$$[X]_{\text{补}} + [Y]_{\text{补}} = [X]_{\text{补}} + [Y]_{\text{补}}$$



3.2 定点数补码加、减运算器设计

2 四位串行加/减法器设计

设计思路: $[X]_{补} - [Y]_{补} = [X]_{补} + [-Y]_{补}$



P=0 加法运算

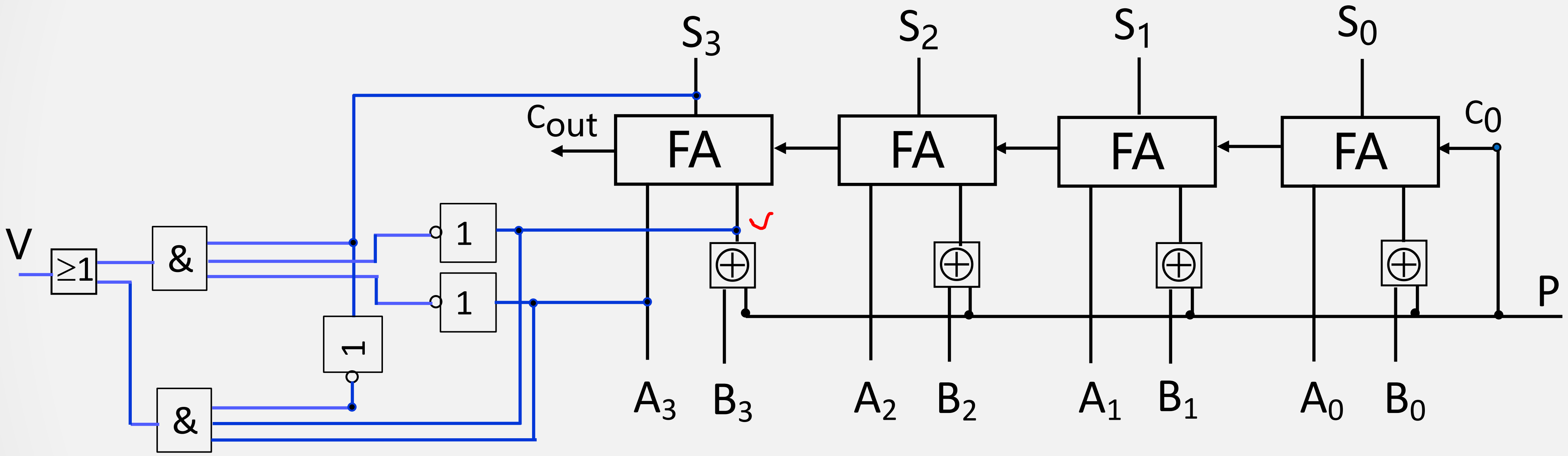
$$\begin{array}{r} 11011 \\ \oplus\ \underline{00000} \\ 11011 \end{array}$$

P=1 减法运算

$$\begin{array}{r} 11011 \\ \oplus\ \underline{11111} \\ 00100 \end{array}$$

3.2 定点数补码加、减运算器设计

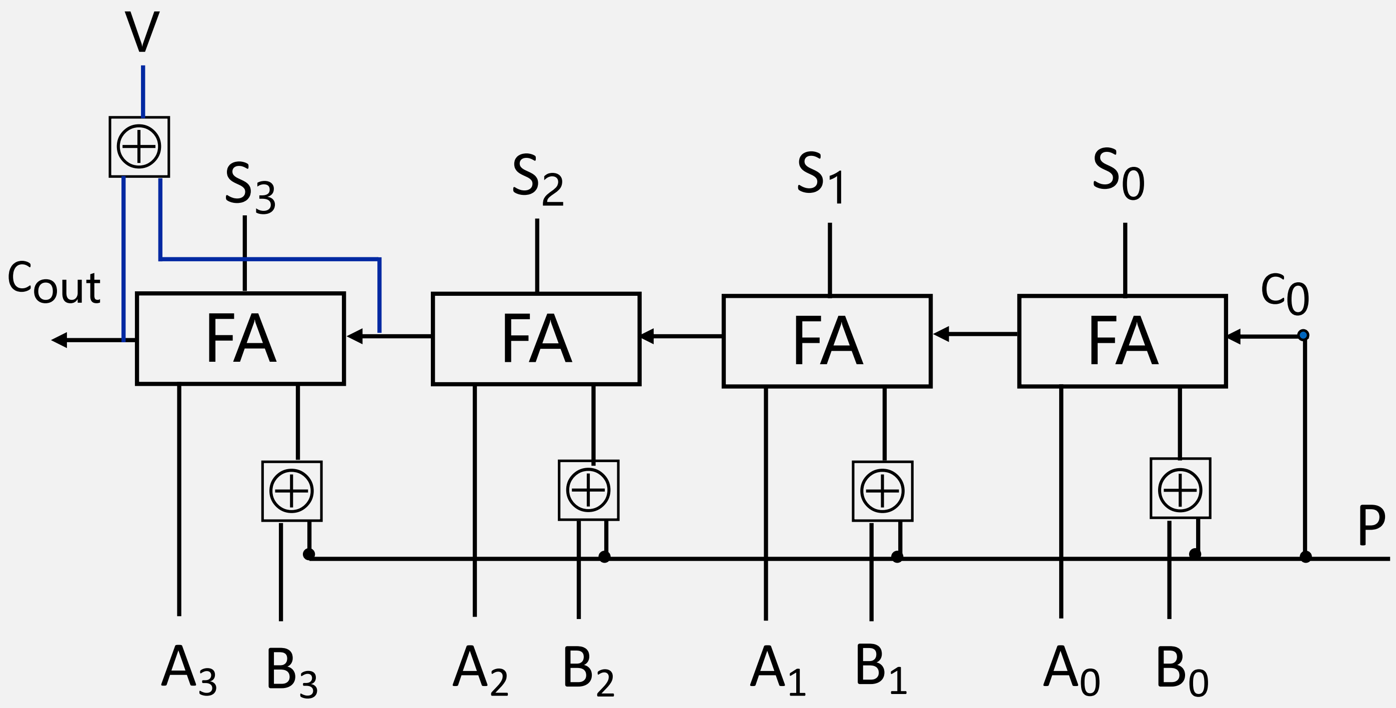
3 带溢出检测功能的加/减运算器



$$V = X_0Y_0\overline{S_0} + \overline{X_0}\,\overline{Y_0}S_0$$

3.2 定点数补码加、减运算器设计

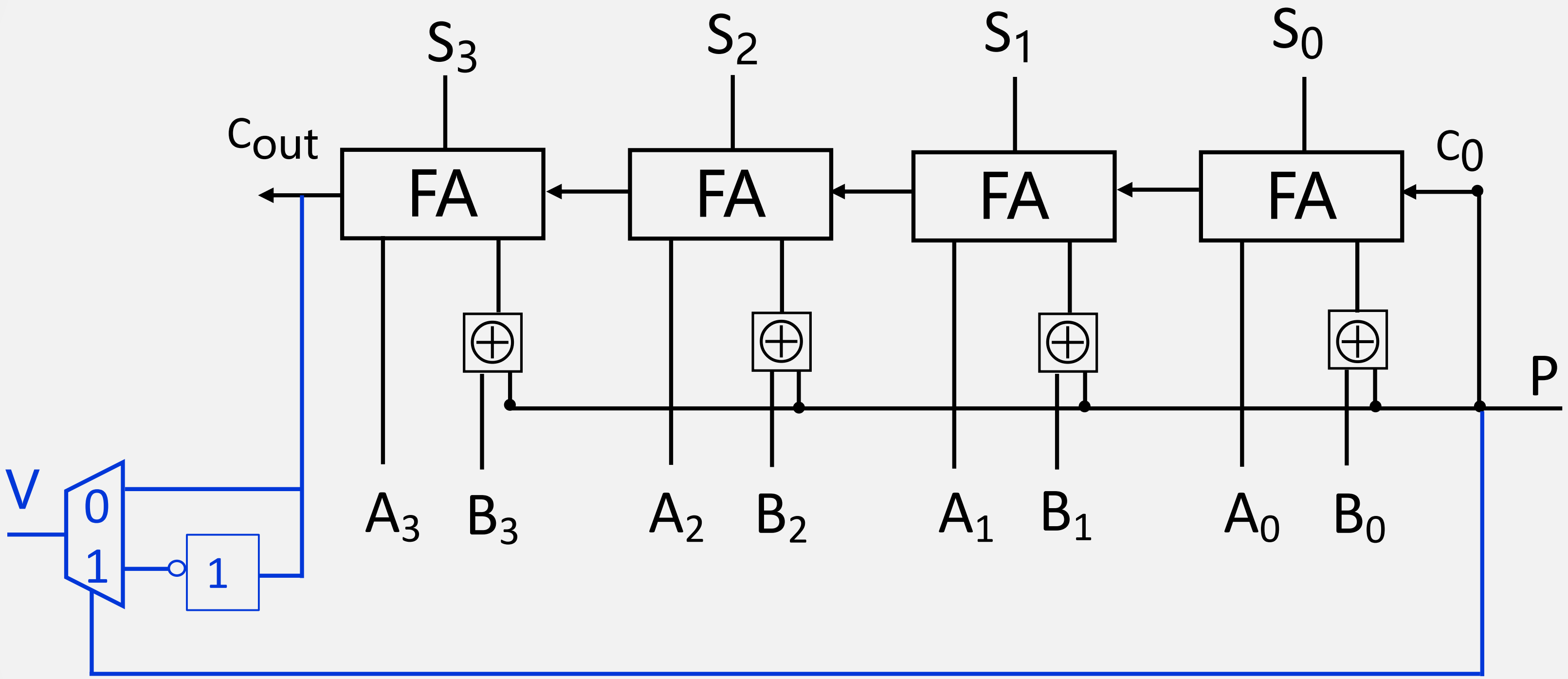
3 带溢出检测功能的加/减运算器



$$V = C_0 \oplus C_1$$

3.2 定点数补码加、减运算器设计

4 带无符号数溢出检测功能的加/减运算器

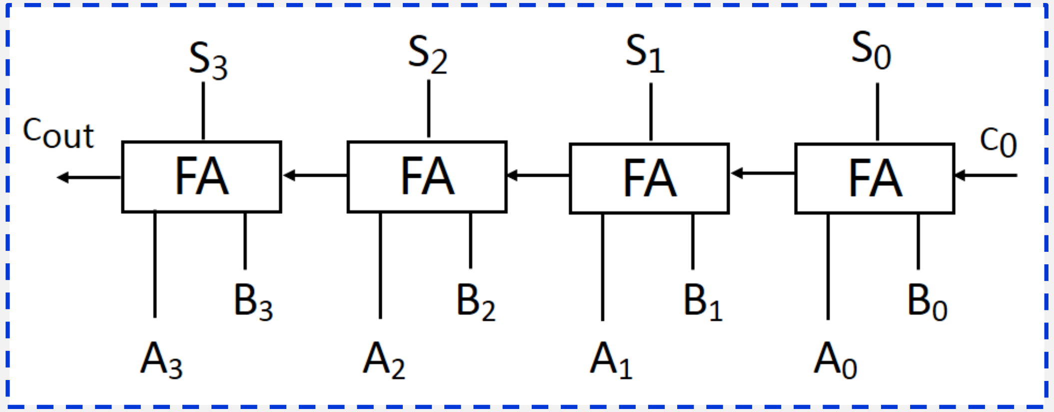


P=1 选择无符号数减法溢出（借位）

P=0时，选择无符号加法溢出（进位）

3.2 定点数补码加、减运算器设计

5 串行进位



$$C_{out}= A_iB_i+ (B_i+A_i) C_{in}$$

$$C_1= A_0B_0+ (B_0+A_0)C_0$$

$$C_2= A_1B_1+ (B_1+A_1)C_1$$

$$C_3= A_2B_2+ (B_2+A_2)C_2$$

$$C_4= A_3B_3+ (B_3+A_3)C_3$$

串行进位：运算速度慢！

3.2 定点数补码加、减运算器设计

6 并行进位 (先行进位)

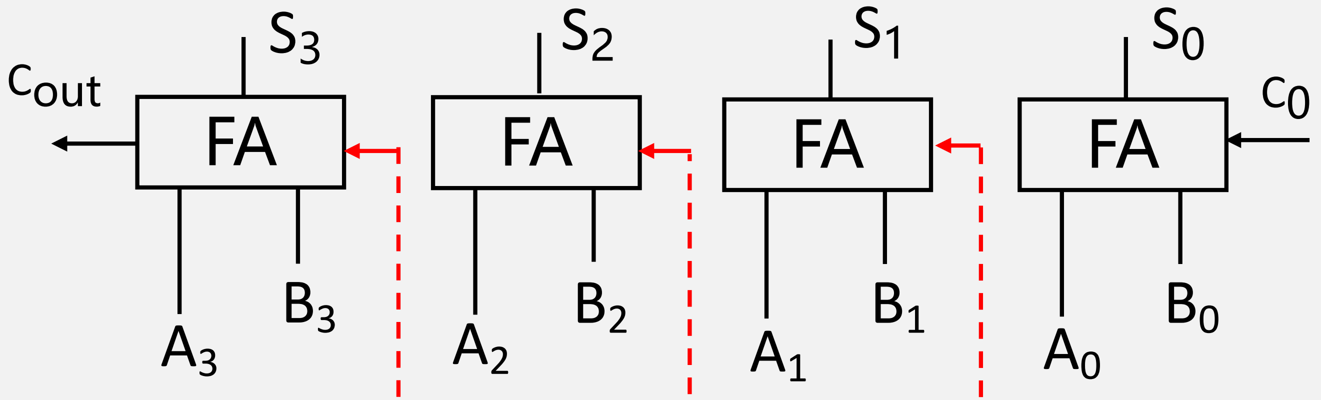
$$C_1 = A_0B_0 + (B_0+A_0)C_0$$

$$C_2 = A_1B_1 + (B_1+A_1)C_1 = A_1B_1 + (A_1+B_1)A_0B_0 + (A_1+B_1)(A_0+B_0)C_0$$

$$C_3 = A_2B_2 + (B_2+A_2)C_2 = A_2B_2+(A_2+B_2)(A_1B_1+(A_1+B_1)A_0B_0) + (A_2+B_2)(A_1+B_1)(A_0+B_0)C_0$$

$$C_4 = A_3B_3 + (B_3+A_3)C_3 = A_3B_3+(A_3+B_3)(A_2B_2+(A_2+B_2)(A_1B_1+(A_1+B_1)A_0B_0) + (A_3+B_3)(A_2+B_2)(A_1+B_1)(A_0+B_0)$$

5 4位并行进位运算器



$$C_1 = A_0B_0 + (B_0+A_0)C_0$$

$$C_2 = A_1B_1 + (A_1+B_1)A_0B_0 + (A_1+B_1)(A_0+B_0)C_0$$

$$C_3 = A_2B_2+(A_2+B_2)(A_1B_1+(A_1+B_1)A_0B_0) + (A_2+B_2)(A_1+B_1)(A_0+B_0)C_0$$

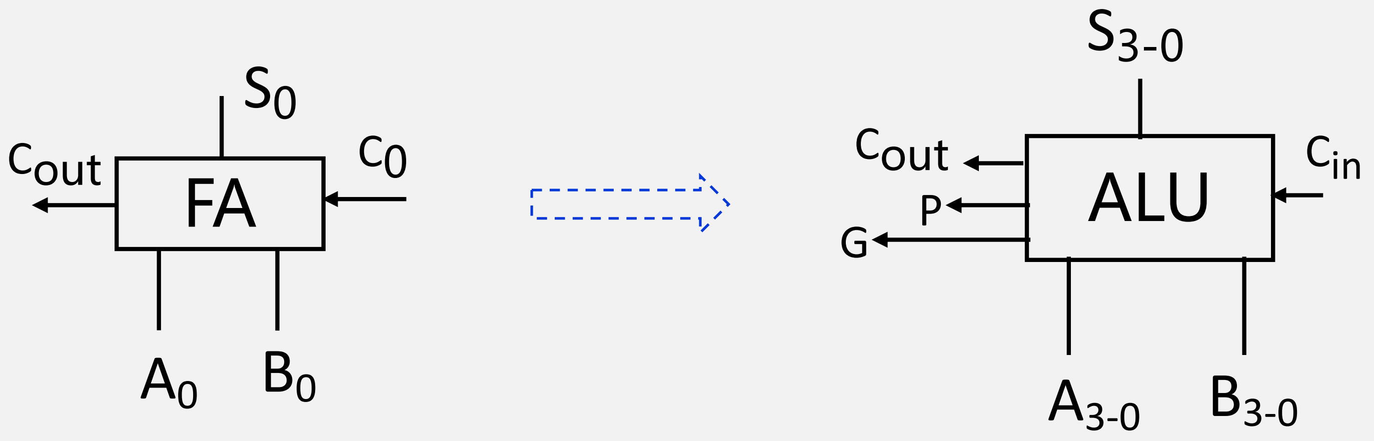
$$C_4=A_3B_3+(A_3+B_3)(A_2B_2+(A_2+B_2)(A_1B_1+(A_1+B_1)A_0B_0))+(A_3+B_3)(A_2+B_2)(A_1+B_1)(A_0+B_0)C_0$$

$G =A_3B_3+(A_3+B_3)(A_2B_2+(A_2+B_2)(A_1B_1+(A_1+B_1)A_0B_0))$：进位产生

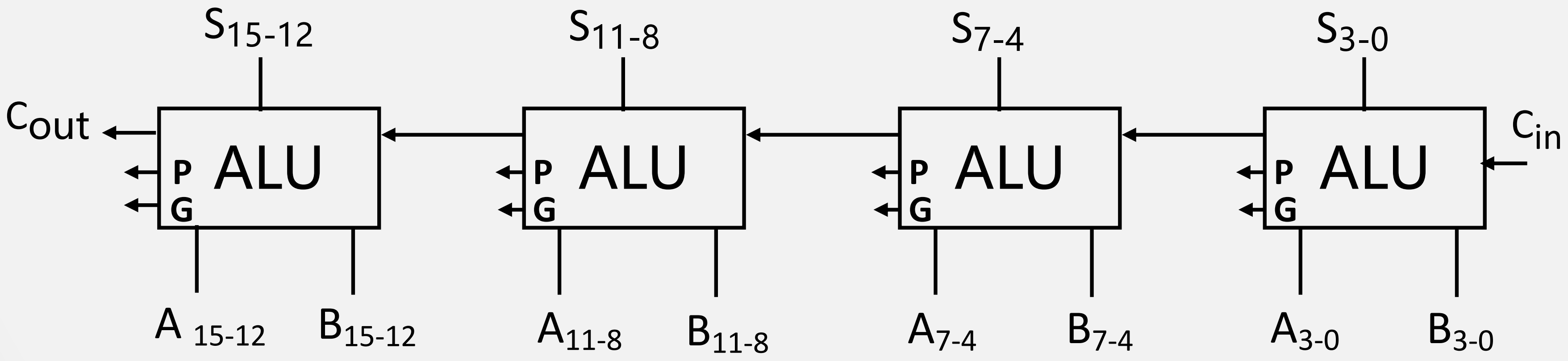
$P=(A_3+B_3)(A_2+B_2)(A_1+B_1)(A_0+B_0)$：进位传递函数

3.2 定点数补码加、减运算器设计

5 多位串行进位与并行进位运算器

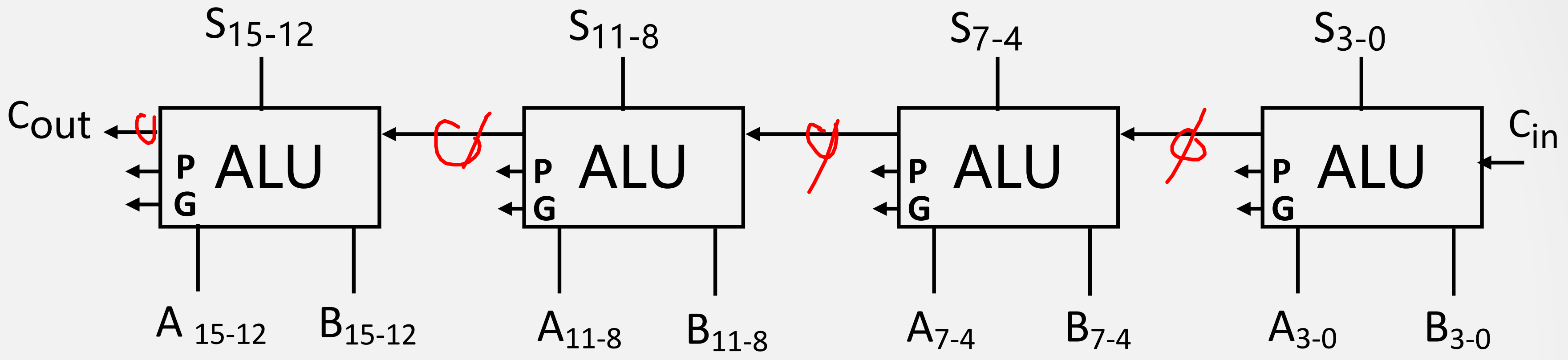


$$C_{out} = G + P \cdot C_0$$



5 多位串行进位与并行进位运算器

$$C_{out} = G + P \cdot C_0$$



$$C_4 = G_0 + P_0 \cdot C_0$$

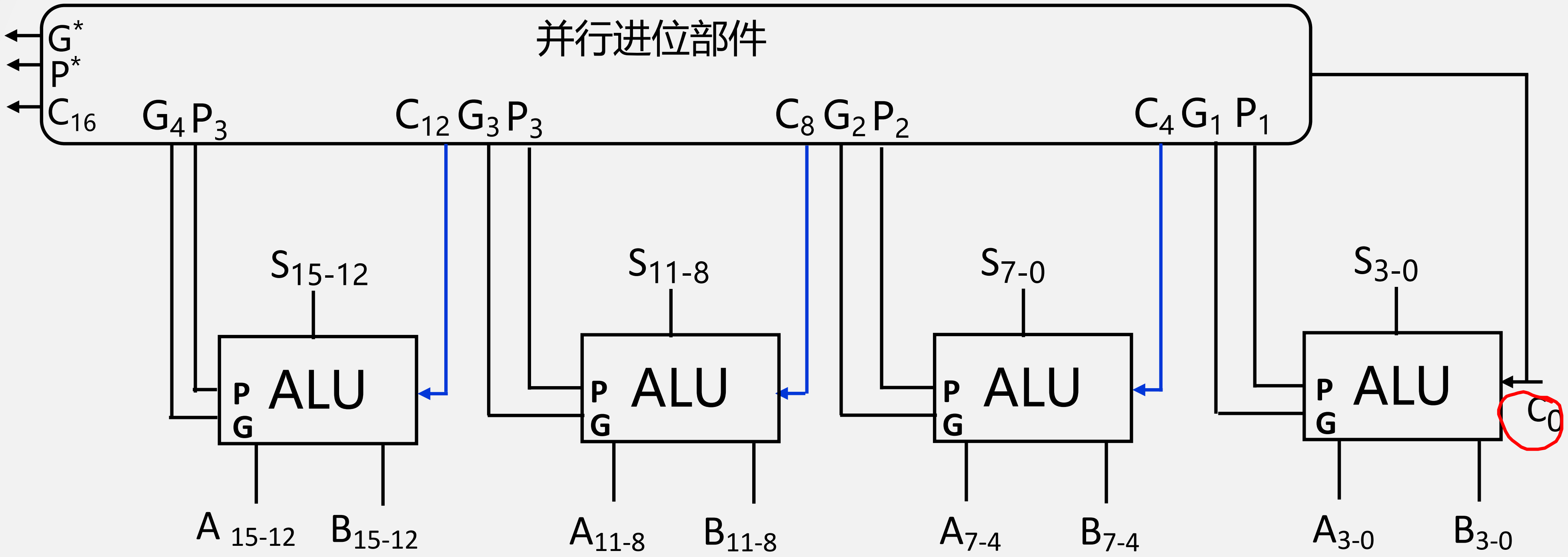
$$C_8 = G_1 + P_1 \cdot C_4 \quad = G_1 + P_1G_0 + P_1P_0 \cdot C_0$$

$$C_{12} = G_2 + P_2 C_8 \quad = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0 \cdot C_0$$

$$C_{16} = G_3 + P_3C_{12} \quad = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0 \cdot C_0$$

3.2 定点数补码加、减运算器设计

5 多位串行进位与并行进位运算器



$C_4 = G_1 + P_1G_0 + P_1P_0 \cdot C_0$

$C_8 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0 \cdot C_0$

$C_{12} = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0 \cdot C_0$

$C_{16} = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0 \cdot C_0$

计算机组成原理

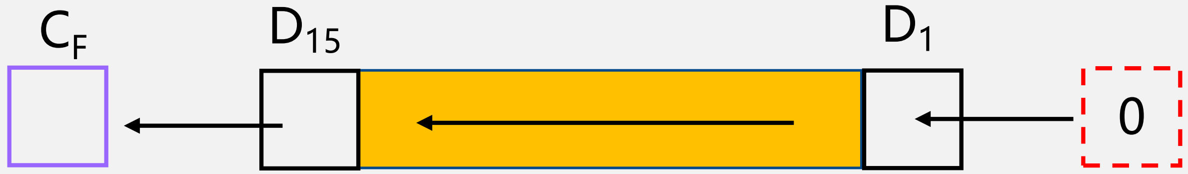
第三章 运算方法与运算器

3.3 原码一位乘法

3.3 原码一位乘法

1 移位操作及其意义

逻辑左移：

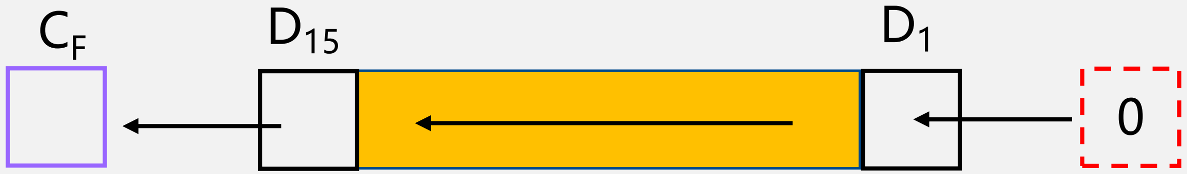


数据整体左移一位，最高位D_{15}被移出至C_F，最低位D_1补0

移位前 **0 1 1 0 1 1 1 0**

逻辑左移后 $\boxed{0}$ **1 1 0 1 1 1 0 0**

算术左移：



数据整体左移一位，最高位D_{15}被移至C_F ，最低位D_1补0

移位前 **0 1 1 0 1 1 1 0**

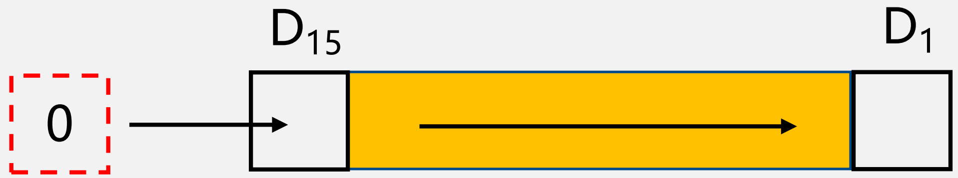
逻辑左移后 $\boxed{0}$ **1 1 0 1 1 1 0 0**

相当于乘2

3.3 原码一位乘法

1 移位操作及其意义

逻辑右移：



数据整体右移一位，最高位D_{15}补0，最低位D_1被移出

移位前　　　1 1 1 0 1 1 1 0

逻辑右移后　　0 1 1 1 0 1 1 1

算术右移：

数据整体右移一位，最高位D_{15}被复制填补D_{15} ，最低位D_1被移出

移位前　　　1 1 1 0 1 1 1 0

逻辑左移后　　1 1 1 1 0 1 1 1

相当于除2

3.3 原码一位乘法

2 二进制乘法的手工计算过程

$$
\begin{array}{r}
0.010 \\
\times \quad 0.101 \\
\hline
0010 \\
0000 \\
0010 \\
+\ 0000 \\
\hline
0001010
\end{array}
$$

a. 说明乘法可由加法实现

b. 存在的问题：

- 需要多输入的全加器（最多为n+1）；
- 需要长度为2n的积寄存器;
- 对应乘数的不同位，部分积左移次数不同，且乘法过程中总移位次数多。

3.3 原码一位乘法

2 二进制乘法的手工计算过程

```
       0.010
×      0.101
------------
        0010
       0000
      0010
+    0000
------------
     0001010
```

⇨

```
      0.010
×     0.101
-----------
       0010
→       001 0
+      0000
-----------
       0001 0
→       000 10
+      0010
-----------
       0010 10
→       001 010
+      0000
-----------
       0001010
```

如何解决上述问题（改进的方法）

- 需要多输入的全加器（最多为n+1）
  - 基于FA的循环累加0或被乘数
- 针对乘数不同位部分积左移次数不同的问题
  - 右移部分积！乘数寄存器
- 需要长度为2n的积寄存器
  - 从部分积和乘数寄存器取结果

3.3 原码一位乘法

3 原码一位乘法算法

- 符号位单独参加运算，数据位取绝对值参加运算。
- 运算法则：

设：$[X]_{原}=X_0.X_1X_2\ldots X_n$ $[Y]_{原}=Y_0.Y_1Y_2\ldots Y_n$

则：$P0 = X_0 \oplus Y_0$ $|P| = |X|\cdot|Y|$

- 运算过程采用改进的乘法运算方法。

3.3 原码一位乘法

3 原码一位乘法算法

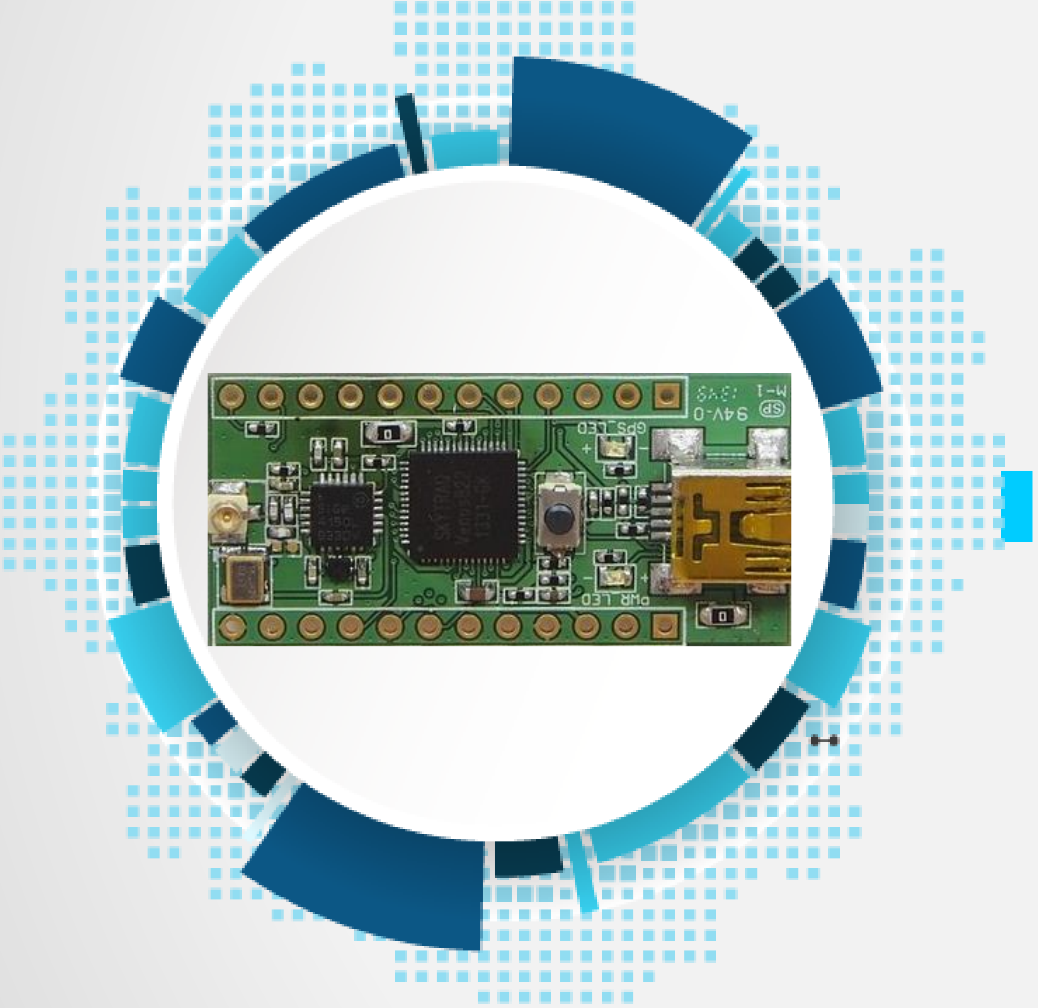
例1 已知 X = 0.110 Y= - 0.101 用原码一位乘法求X * Y

解: $[X]_{原}$ = 0.110 $[Y]_{原}$ = 1.101

部分积	|乘数| / 判断位	说明
00.000	Y_0.101	Y_3 =1 部分积 + |X|
+ 00.110		
00.110		每次运算结果右移1位
→ 00.011	0 Y_0.10	Y_3 =0 部分积 + 0
+ 00.000		
00.011		
→ 00.001	10 Y_0.1	Y_3 =1 部分积 + |X|
+ 00.110		
00.111		
→00.011	110 Y_0	

X*Y =(0⊕1).011110 = 1.011110

```
      0.0 1 0
×     0.1 0 1
      0 0 1 0
→       0 0 1  0
+     0 0 0 0
      0 0 0 1  0
→       0 0 0  1 0
+     0 0 1 0
      0 0 1 0  1 0
→       0 0 1  0 1 0
+     0 0 0 0
      0 0 0 1 0 1 0
```



计算机组成原理

第三章 运算方法与运算器

3.4 补码一位乘法

3.4 补码一位乘法

1 补码一位乘法的基本方法

设$[X]_补 = X_0X_1X_2X_3\ldots X_n$　$[Y]_补 = Y_0Y_1Y_2Y_3\ldots Y_n$

可证明：

$$[X\cdot Y]_补 = [X]_补\bullet(0.Y_1Y_2Y_3\ldots Yn) - Y_0\bullet [X]_补$$

进一步展开合并后可得：

$$[x\bullet y]_补=[x]_补\bullet \sum_{i=0}^{n}(y_{i+1} - y_i)2^{-i}$$ （符号位参加运算）

3.4 补码一位乘法

1 补码一位乘法的基本方法

$$[x \bullet y]_{补}=[x]_{补} \bullet \Sigma(y_{i+1}-y_i)2^{-i}$$ （符号位参加运算）

补码一位乘法的运算规则如下:

(1)如果$y_{n+1}=y_n$,部分积加0，部分积算术右移1位；

(2)如果$y_{n+1}y_n=10$,部分积加$[x]_{补}$，部分积算术右移1位；

(3)如果$y_{n+1}y_n=01$,部分积加$[-x]_{补}$，部分积算术右移1位.

重复进行n+1步，但最后一步不移位。

包括一位符号位，所得乘积为2n+1位，其中n为数据位位数.

3.4 补码一位乘法

1 补码一位乘法的基本方法

设$[X]_补 = X_0X_1X_2X_3…X_n$　$[Y]_补 = Y_0Y_1Y_2Y_3…Y_n$

$[x•y]_补=[x]_补• \Sigma(y_{i+1} - y_i)2^{-i}$　（符号位参加运算）

几个特殊问题的处理

(1) $i=n$时 ,$y_{n+1}=$ ？　　$y_{n+1}= 0$

(2) y_{n+1} 是哪个寄存器？　　在乘数寄存器Y后增加的一位

(3)算术右移的对象有哪些？　　部分积和乘数寄存器均右移

3.4 补码一位乘法

2 补码一位乘法的举例

例1 已知X= +1101 Y=+1011 用补码一位乘法求 X×Y

解： $[X]_{补}=01101$ $[Y]_{补}=01011$ $[-X]_{补}=10011$

	部分积	乘数	说明
	000000	010110	$Y_{n+1} < Y_n$ 部分积 $+[-X]_{补}$
+	110011		
	110011		
→	111001	101011	结果右移一位, $Y_{n+1} = Y_n$ 部分积 +0
+	000000		
	111001		
→	111100	110101	结果右移一位, $Y_{n+1} > Y_n$ 部分积 $+[X]_{补}$
+	001101		
	001001		

2 补码一位乘法的举例

部分积	乘数	说明
→ 000100	111010	将结果右移一位, $Y_{n+1} < Y_n$ 部分积 $+[-X]_{补}$
+ 110011		
110111		
→ 111011	111101	将结果右移一位, $Y_{n+1} > Y_n$ 部分积 $+[X]_{补}$
+ 001101		
001000		

$\therefore [X \cdot Y]_{补}=010001111$

$\therefore X \cdot Y=\ \ 010001111$

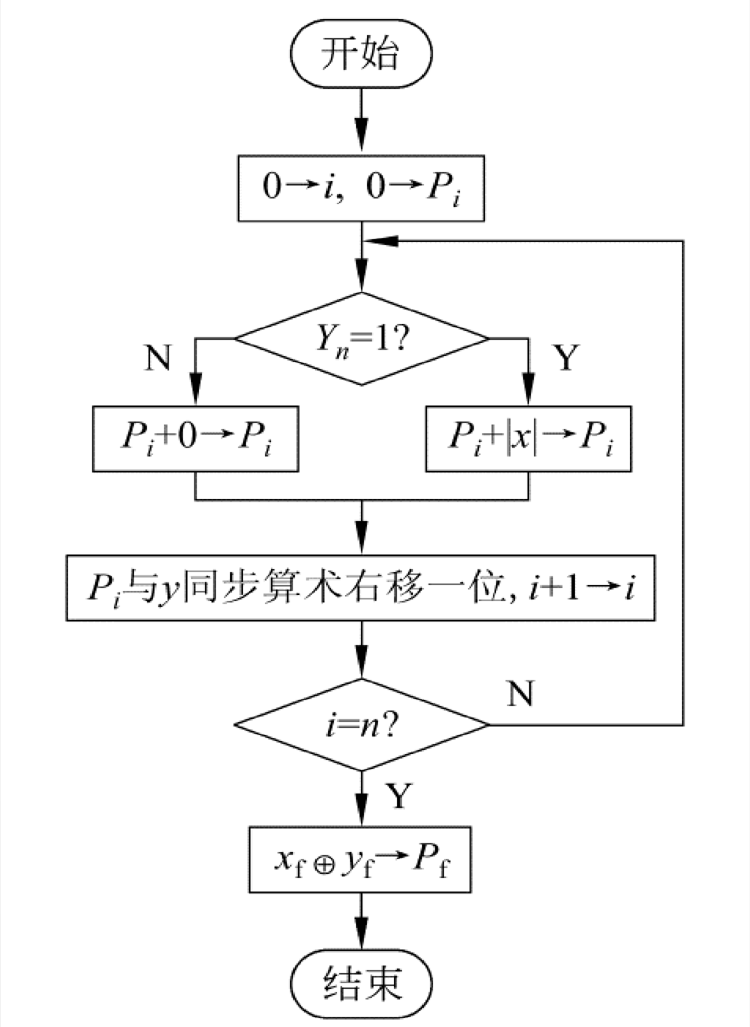
计算机组成原理

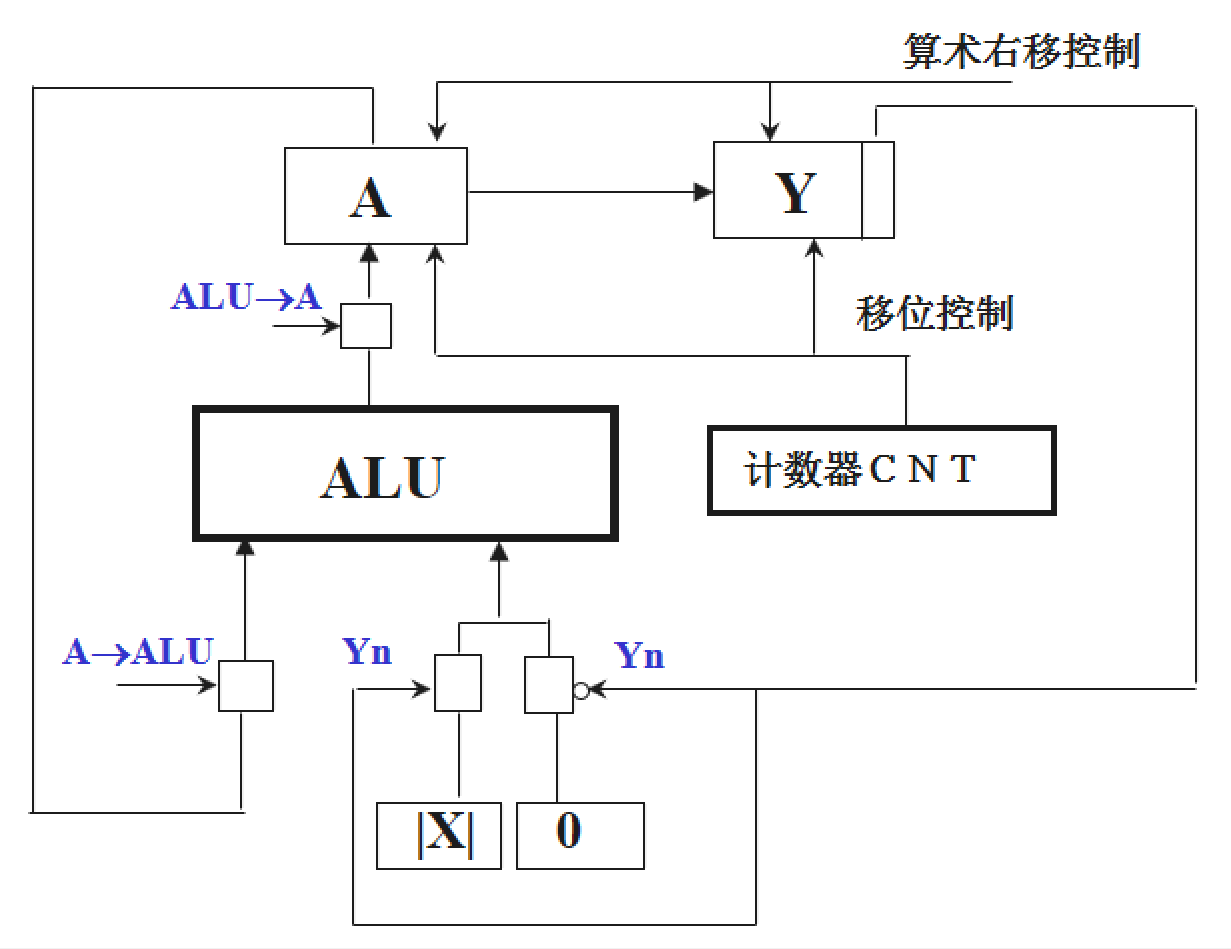
第三章 运算方法与运算器

3.5 乘法运算器设计

3.5 乘法运算器设计

1 原码一位乘法器设计





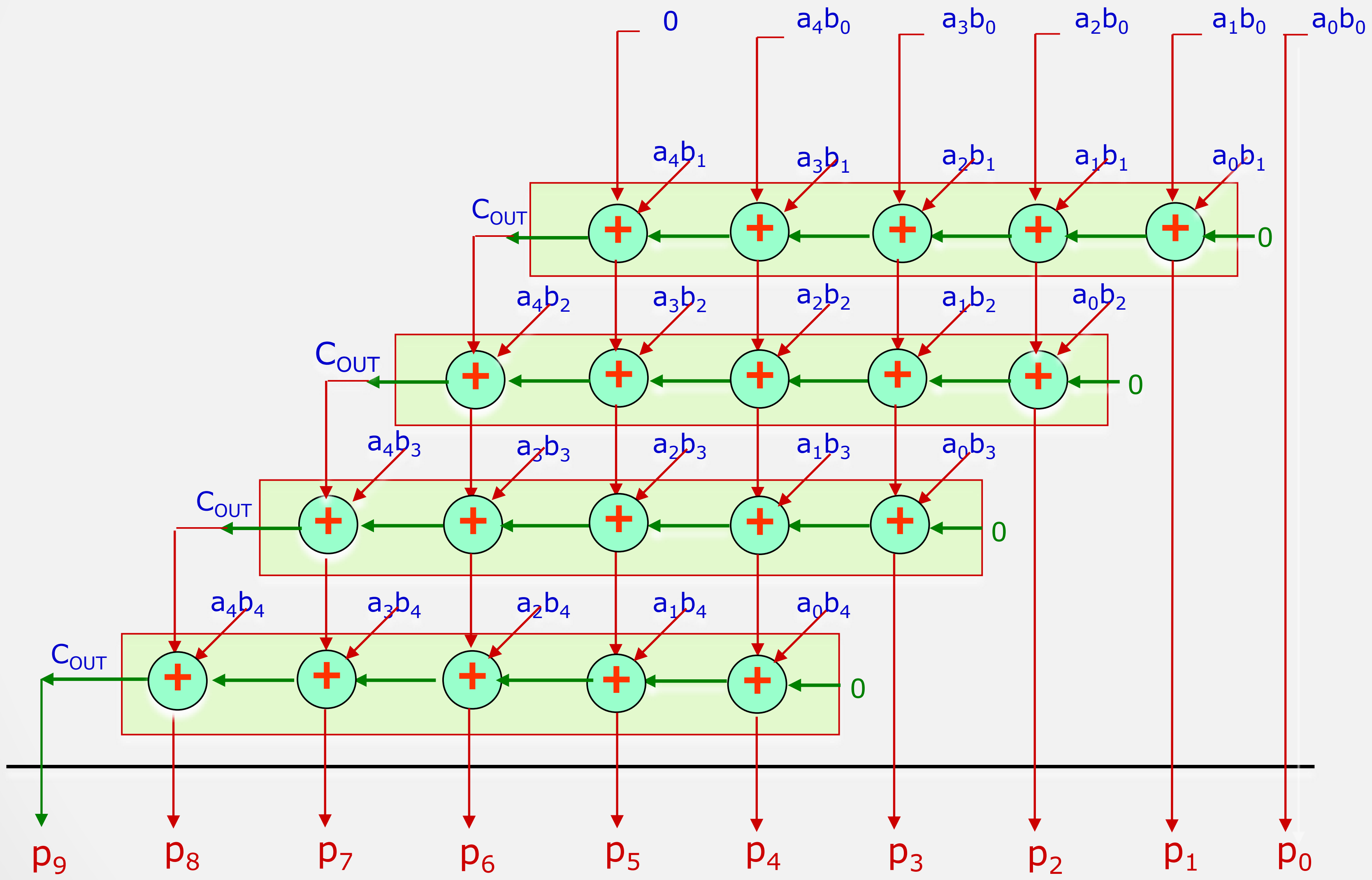
2 原码阵列乘法器设计

$$
\begin{array}{ccccccccc}
 & & & & a_{m-1} & a_{m-2} & \cdots & a_1 & a_0 & = A \\
 & & & & & b_{n-1} & \cdots & b_1 & b_0 & = B \\
\hline
 & & & & a_{m-1}b_0 & a_{m-2}b_0 & \cdots & a_1b_0 & a_0b_0 & \\
 & & & a_{m-1}b_1 & a_{m-2}b_1 & \cdots & a_1b_1 & a_0b_1 & & \\
 & & \cdots & \cdots & \cdots & \cdots & & & & \\
a_{m-1}b_1 & a_{m-2}b_1 & \cdots & a_1b_1 & a_0b_{n-1} & & & & & \\
\hline
P_{m+n-1} & p_{m+n-2} & \cdots & & & & & P_1 & P_0 & =P
\end{array}
$$

与运算 、 与项求和

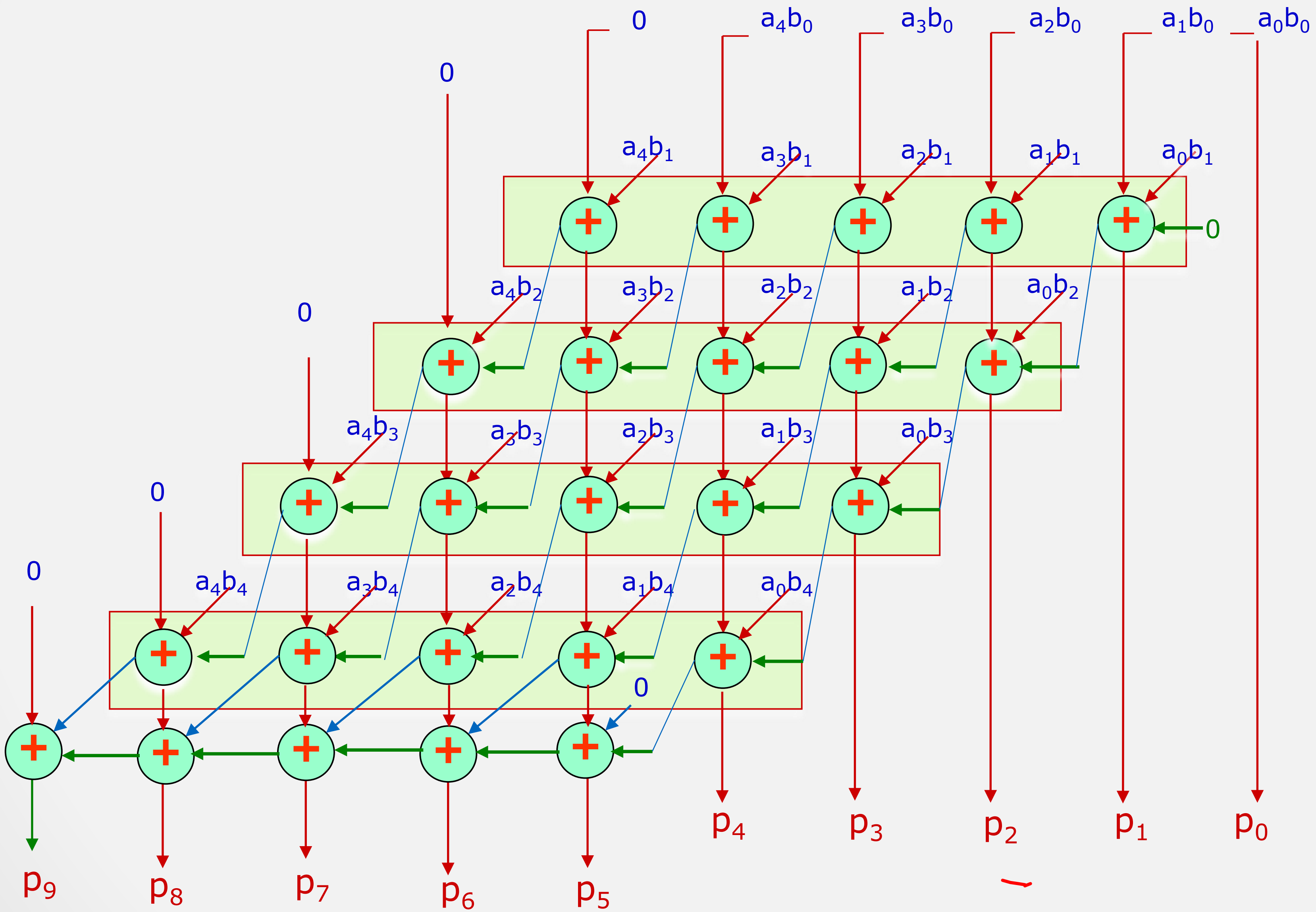
3.5 乘法运算器设计

2 原码阵列乘法器设计



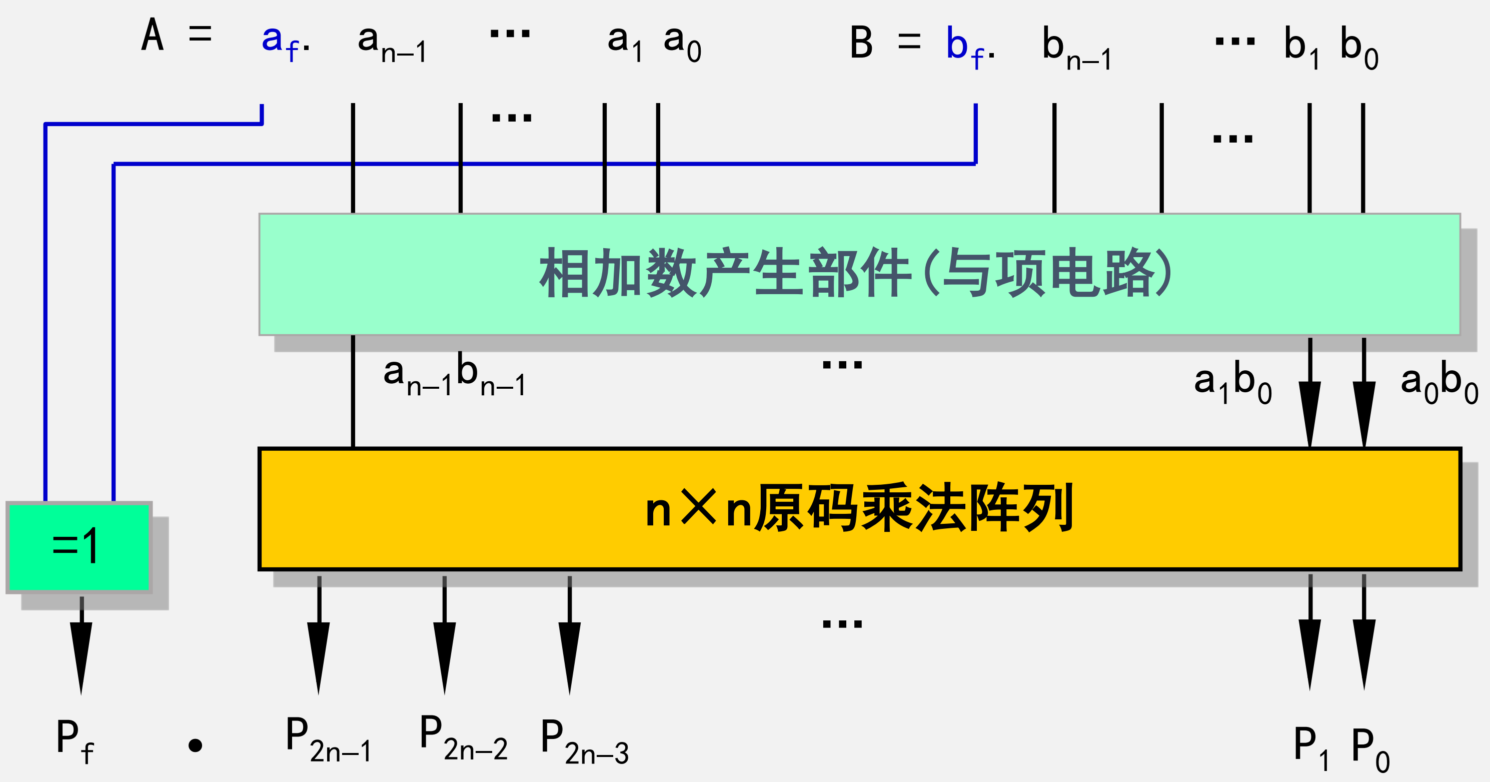
3.5 乘法运算器设计

2 原码阵列乘法器设计



3.5 乘法运算器设计

2 原码阵列乘法器设计



3 补码一位乘法器设计

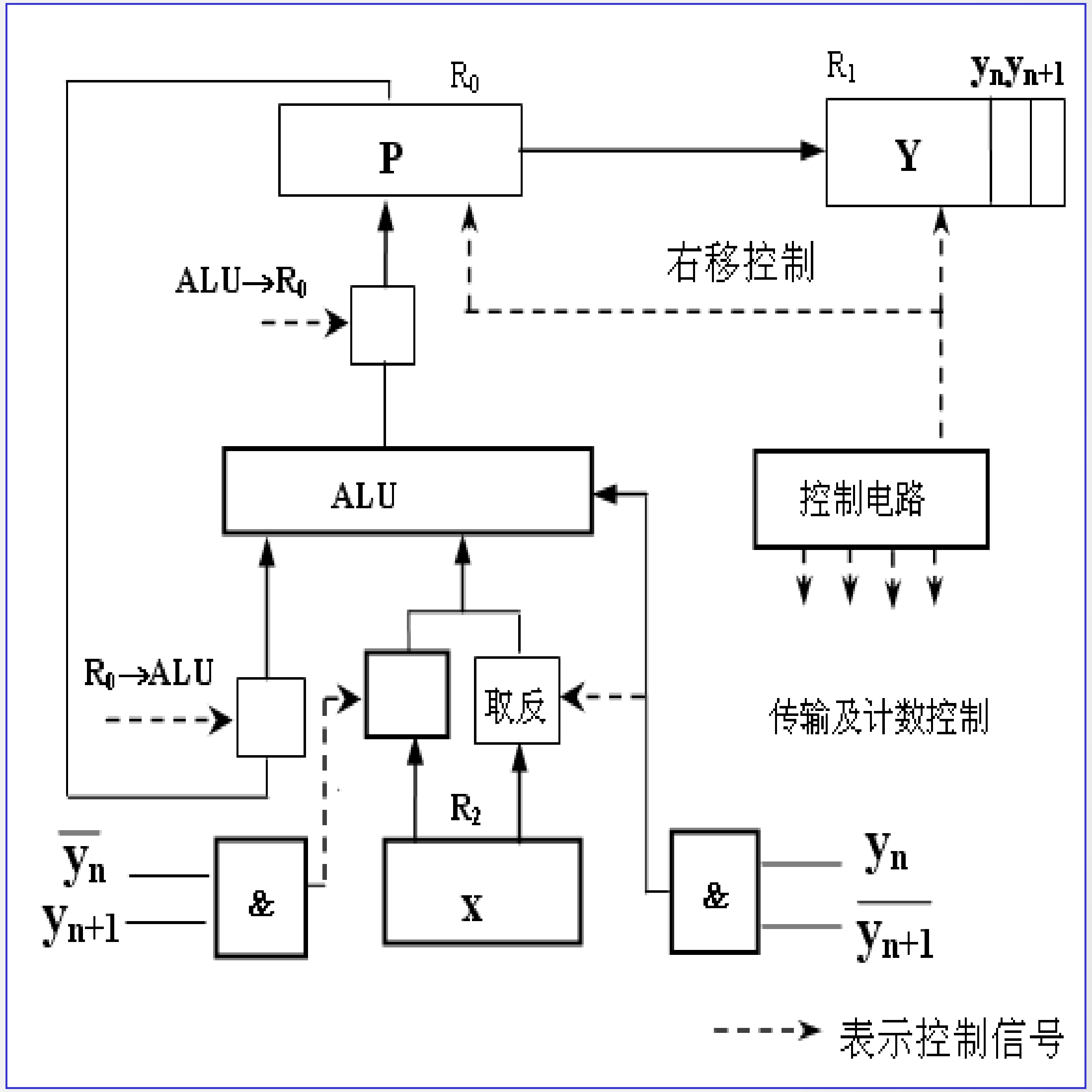
补码一位乘法的运算规则如下

(1)$y_{n+1}=y_n$,部分积加0，算术右移1位；

(2)$y_{n+1}y_n=10$,部分积加$[x]_补$，算术右移1位；

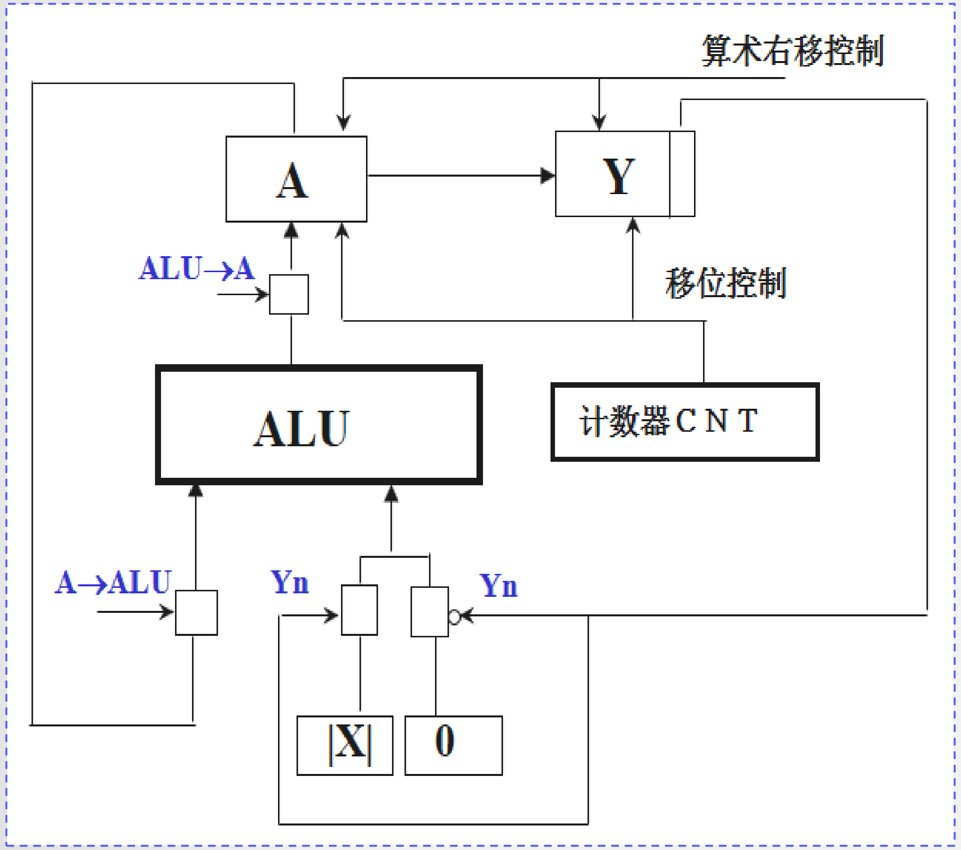
(3)$y_{n+1}y_n=01$,部分积加$[-x]_补$，算术右移1位.

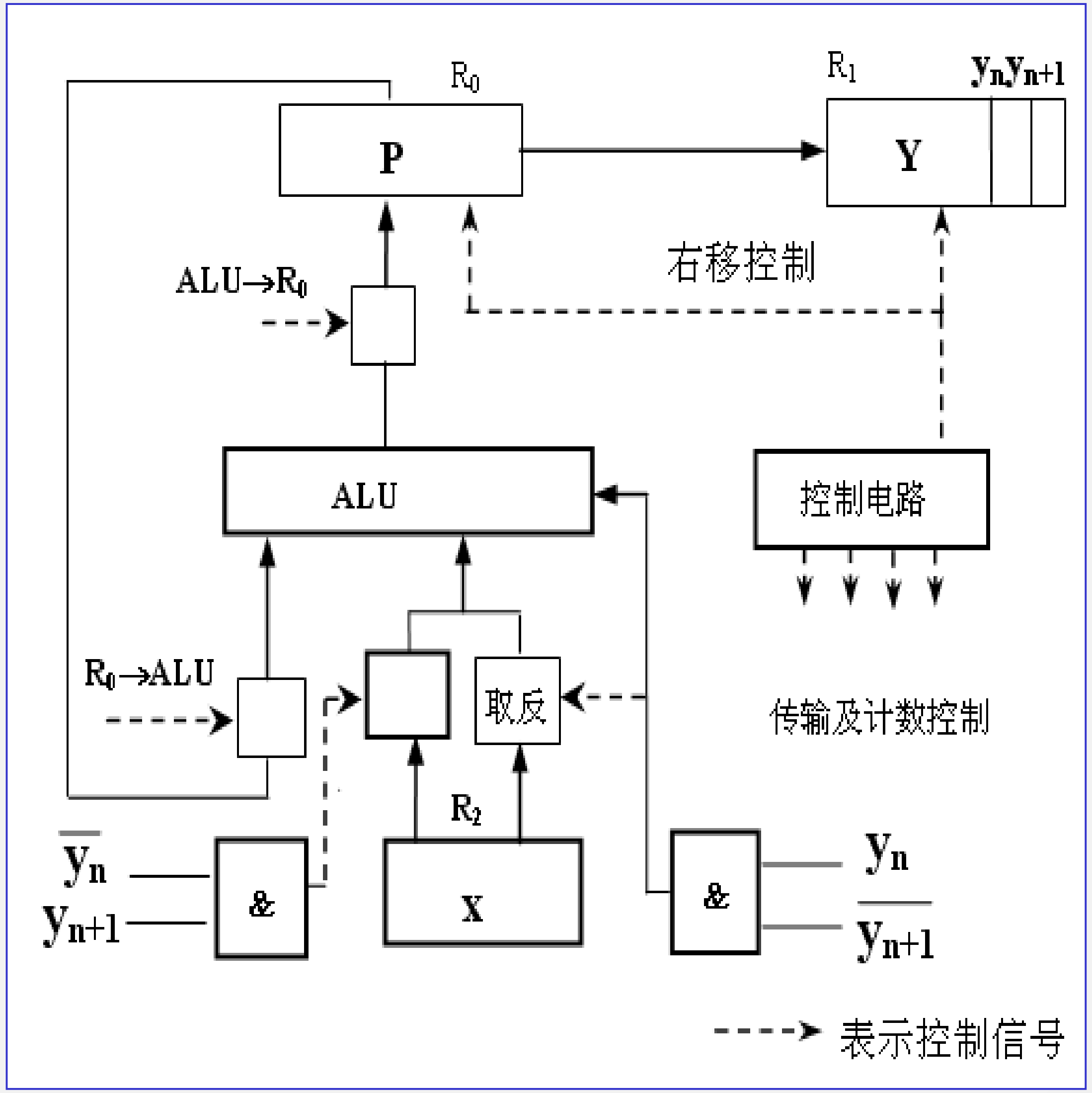
重复进行n＋1步，但最后一步不移位。



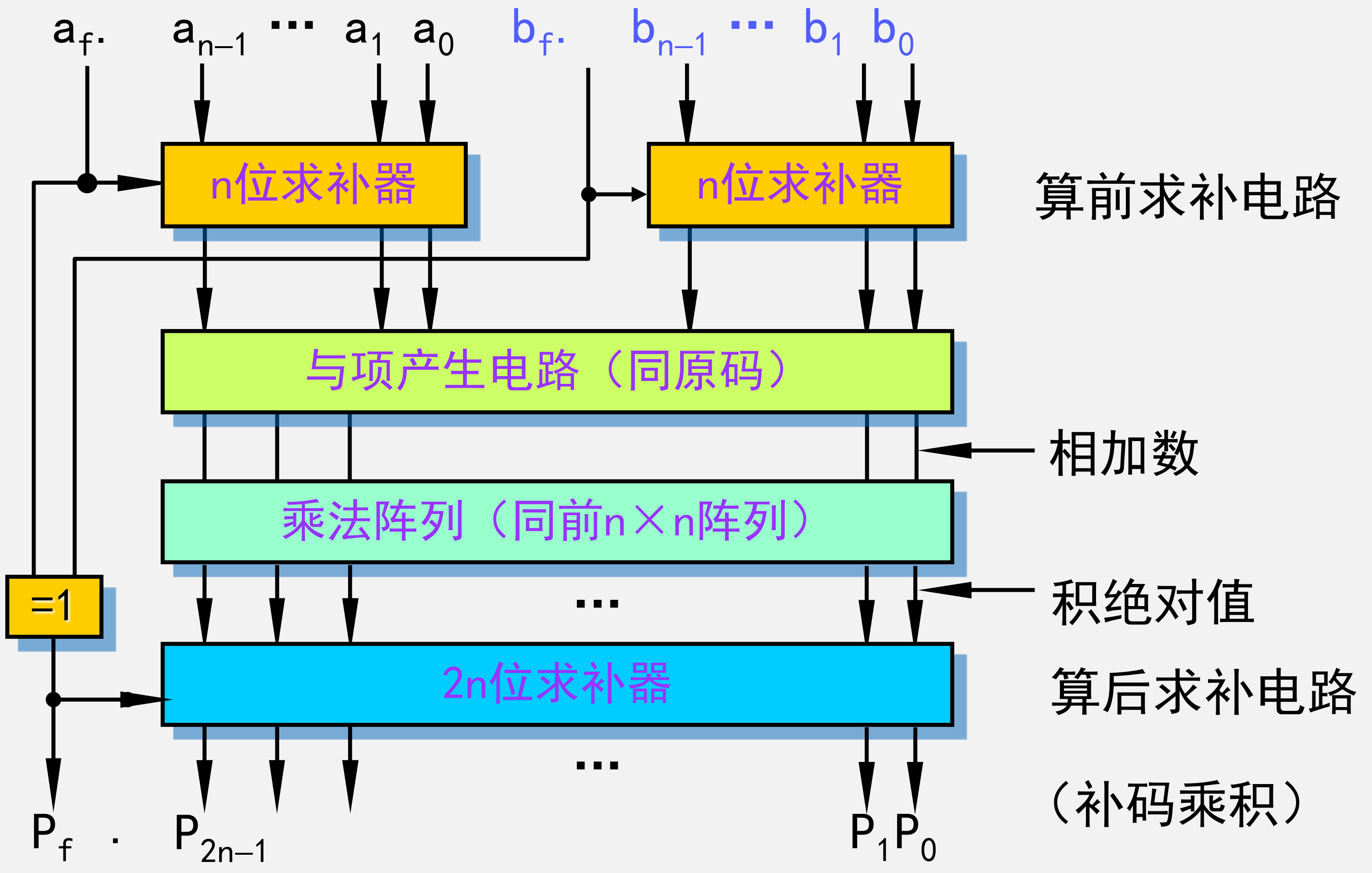
3.5 乘法运算器设计

3 补码一位乘法器设计





4 补码阵列乘法器设计



a_f.
a_{n-1} ··· a_1 a_0
b_f.
b_{n-1} ··· b_1 b_0
n位求补器
n位求补器
算前求补电路
与项产生电路（同原码）
相加数
乘法阵列（同前n×n阵列）
积绝对值
=1
2n位求补器
算后求补电路
（补码乘积）
P_f . P_{2n-1} ··· P_1P_0

计算机组成原理

第三章　运算方法与运算器

3.6 定点数除法

3.6 定点数除法

1 手工除法运算方法

商：0.1101	
0.1011) 0.10010	不够减，商上零，
- 0.01011	除数右移1位，够减，减除数，商上1
0.001110	
- 0.001011	除数右移2位，够减，减除数，商上1
0.0000110	
0.0001011	除数右移3位，不够减，不减除数，商上零
0.00001100	
- 0.00001011	除数右移4位，够减，减除数，商上1
0.00000001	

启示：除法可通过减法实现

问题：除数移位次数不固定且多

需要长度为2n位的余数寄存器

如何判断每步是否够减

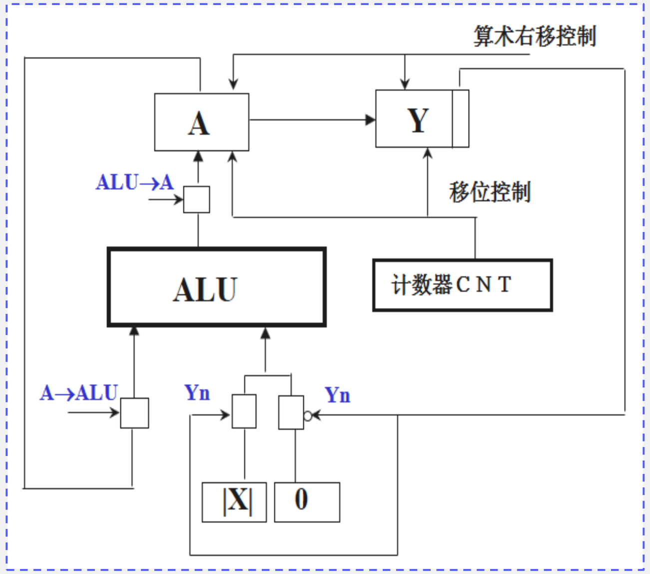
3.6 定点数除法

2 原码恢复余数除法

■ 如何判断是否够减

◆利用减法，通过余数符号判断

$$\begin{array}{r} 00.10010 \\ -\ 00.01011 \\ \hline 00.00111 \end{array} \qquad \begin{array}{r} 00.10010 \\ -00.11011 \\ \hline 11.10111 \\ +00.11011 \\ \hline 00.10010 \end{array}$$



■余数为正数时，够减　，商上1，将余数左移一位，再与除数做减法比较

■ 余数为负数时，不够减，商上0　，？

◆加除数恢复成原来的值　，将余数左移一位，再与除数做减法比较

■重复上述过程直到商达到所需要的位数为止。

2 原码恢复余数除法

已知 $X=0.1001$，$Y=-0.1011$，用原码一位除法求 X/Y

解：$[X]_{原}=0.1001$ $[|X|]_{补}=0.1001$

$[Y]_{原}=1.1011$ $[|Y|]_{补}=0.1011$ $[-|Y|]_{补}=1.0101$

2 原码除法运算方法

	被除数/余数	商	上商位	说明
	00.1001			减Y比较
$+[-Y]_{补}$	11.0101			
	11.1110		0	余数<0，商=0
+	00.1011			加Y恢复余数
	00.1001			左移一位
←	01.0010	0		减Y比较
$+[-Y]_{补}$	11.0101			
	00.0111		1	余数>0，商上1
←	00.1110	0.1		左移一位
$+[-Y]_{补}$	11.0101			减Y比较
	00.0011		1	余数>0，商上1
←	00.0110	0.11		左移一位
$+[-Y]_{补}$	11.0101			减Y比较
	11.1011		0	余数<0，商上0
+	00.1011			加Y恢复余数
	00.0110			左移一位
←	00.1100	0.110		减Y比较
$+[-Y]_{补}$	11.0101			
	00.0001	0.1101	1	余数>0，商上1，移商

最后结果：

商Q = （$X_0 \oplus Y_0$）.1101=1.1101

余数 R = 0.0001 * 2^{-4}

该方法存在的不足：

运算步数不确定

3 原码加/减交替除法运算方法（不恢复余数法）

- 设某次余数为R_i，将R_i左移一位减除数进行比较并上商，即：

  $2R_i-Y$

- 当上述结果小于0时，商上0，恢复余数，然后左移一位，减除数比较，即：

  $(2R_i-Y)+Y=2R_i$

  $2*2R_i-Y=4R_i-Y$

- 若当结果小于0时，商上0，不恢复余数而直接将余数左移一位，加Y:

  $2(2R_i-Y)+Y$

  $=2*2R_i-2Y+Y=4R_i-Y$

3 原码加/减交替除法运算方法（不恢复余数法）

已知 X = 0.1001，Y = - 0.1011，用原码一位除法求X/Y

	被除数/余数	商	上商位	说明
	00.1001			减Y比较
+[−Y]补	11.0101			
0	11.1110		0	余数 <0 商上零
←	11.1100	0		左移一位
+[Y]补	00.1011			加Y比较
1	00.0111		1	余数>0，商上1
←	00.1110	0.1		左移一位
+[−Y]补	11.0101			减Y比较
1	00.0011		1	余数>0，商上1
←	00.0110	0.11		左移一位
+[−Y]补	11.0101			减Y比较
0	11.1011		0	余数<0 商上零
←	11.0110	0.110		左移一位
+[Y]补	00.1011			加Y比较
1	00. 0001	0.1101	1	余数>0，商上1,移商

最后结果：

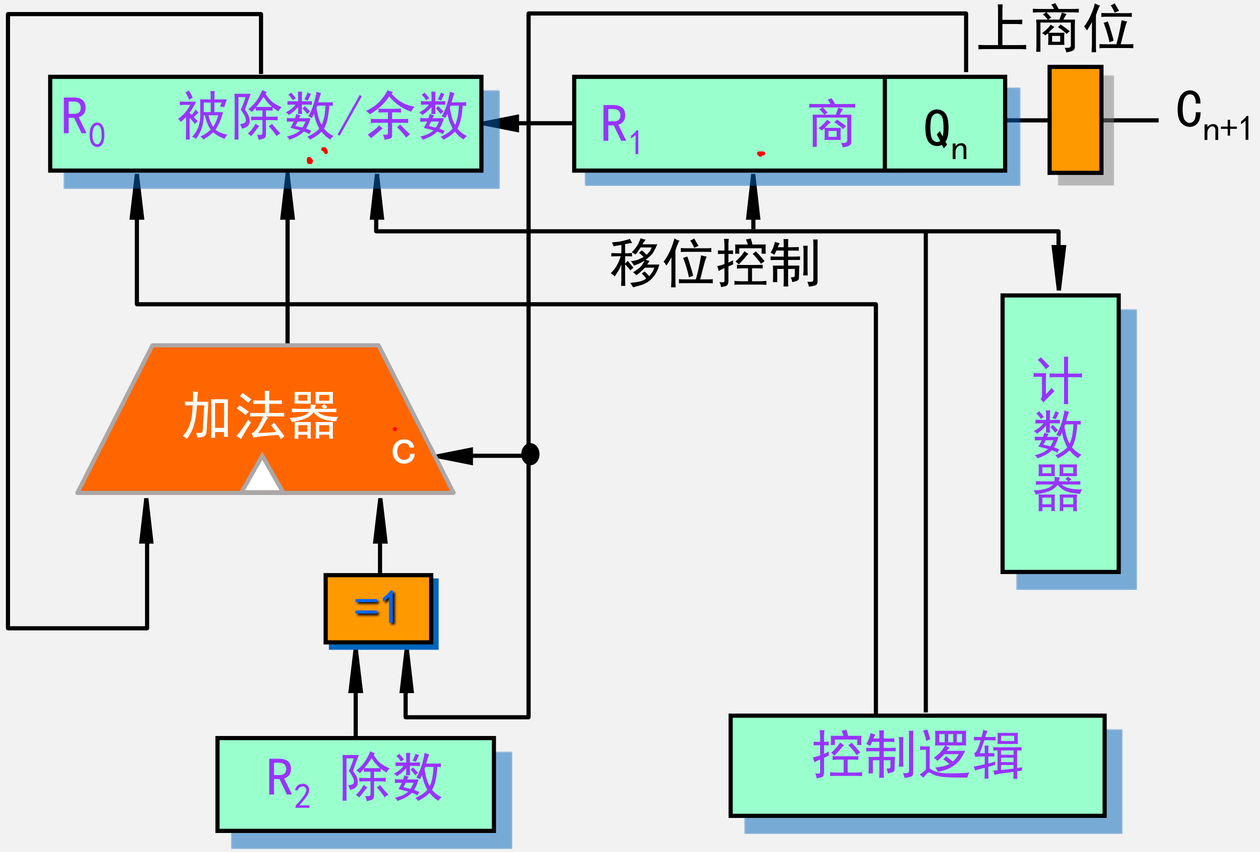
商Q = $X_0 \oplus Y_0$.1101 = 1.1101

余数 R = 0.0001 * 2^{-4}

	被除数/余数	商	上商位	说明
	00.1001			减Y比较
+[−Y]补	11.0101			
	11.1110		0	余数<0，商=0
+	00.1011			加Y恢复余数
	00.1001			左移一位
←	01.0010	0		减Y比较
+[−Y]补	11.0101			
	00.0111		1	余数>0，商上1
←	00.1110	0.1		左移一位
+[−Y]补	11.0101			减Y比较
	00.0011		1	余数>0，商上1
←	00.0110	0.11		左移一位
+[−Y]补	11.0101			减Y比较
	11.1011		0	余数<0，商上0
+	00.1011			加Y恢复余数
	00.0110			
←	00.1100	0.110		左移一位
+[−Y]补	11.0101			减Y比较
	00.0001	0.1101	1	余数>0，商上1，移商

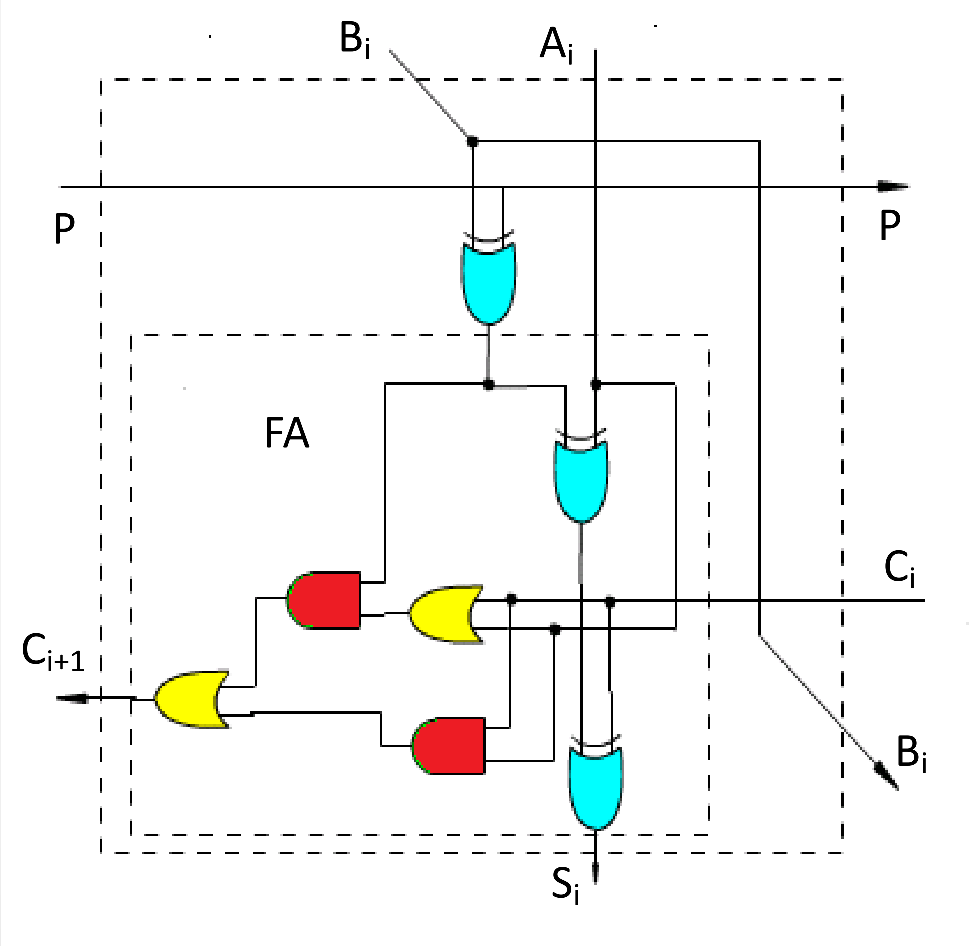
4 原码加/减交替除法实现逻辑

	被除数/余数	商	上商位	说明
	00.1001			减Y比较
+[-Y]补	11.0101			
	0 11.1110		0	余数<0 商上零
←	11.1100	0		左移一位
+[Y]补	00.1011			加Y比较
	1 00.0111		1	余数>0，商上1
←	00.1110	0.1		左移一位
+[-Y]补	11.0101			减Y比较
	1 00.0011		1	余数>0，商上1
←	00.0110	0.11		左移一位
+[-Y]补	11.0101			减Y比较
	0 11.1011		0	余数<0 商上零
←	11.0110	0.110		左移一位
+[Y]补	00.1011			加Y比较
	1 00. 0001	0.1101	1	余数>0，商上1,移商



5 阵列除法

1)可控制加/减法(CAS)单元



逻辑功能为：

$$S_i = A_i \oplus (B_i \oplus P) \oplus C_i$$
$$C_{i+1} = (A_i + C_i)(B_i \oplus P) + A_i C_i$$

P=0时实现加法功能

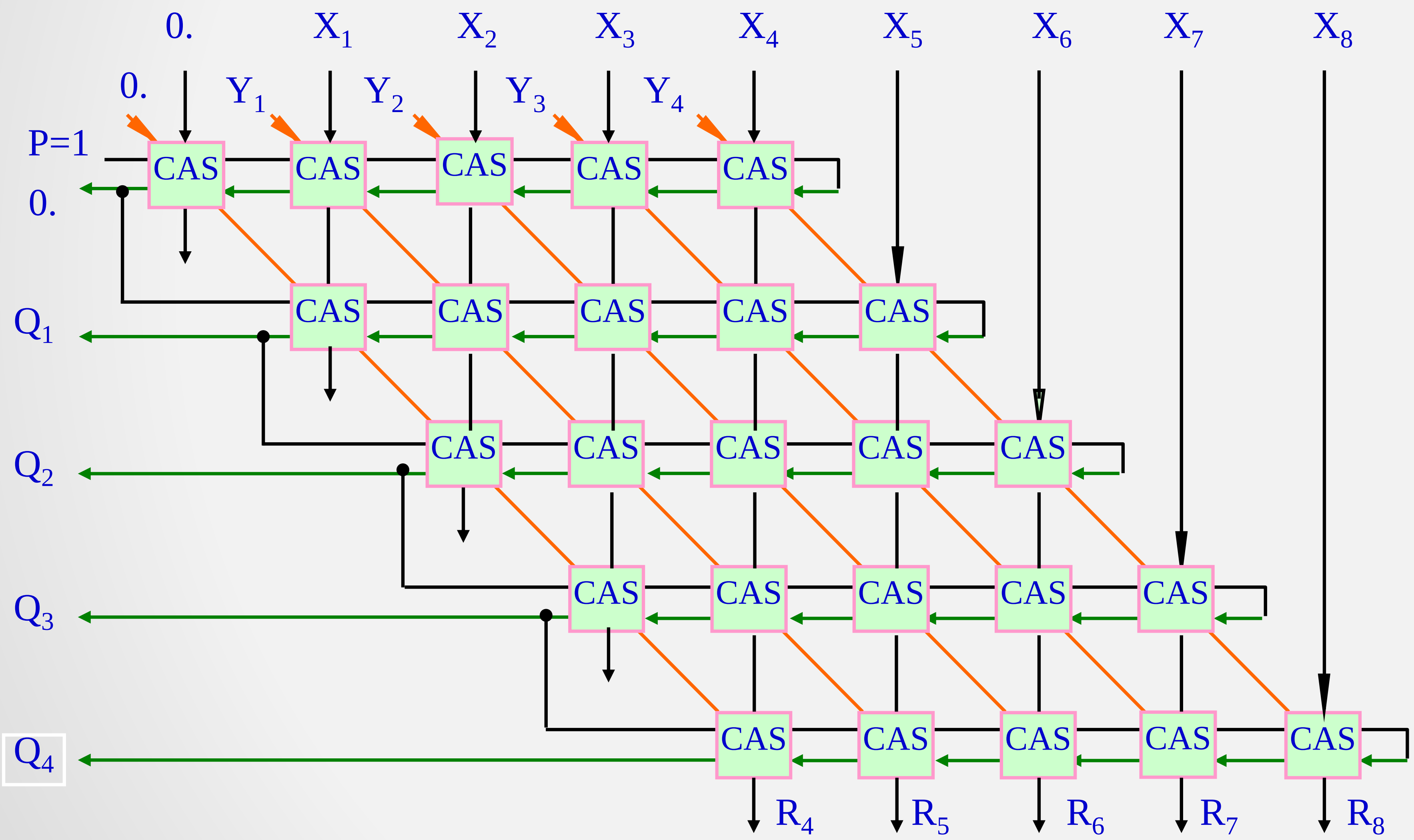
$$S_i = A_i \oplus B_i \oplus C_i$$
$$C_{i+1} = (A_i + C_i) B_i + A_i C_i$$

P=1时实现减法功能(全减)

$$S_i = A_i \oplus \overline{B_i} \oplus C_i$$
$$C_{i+1} = (A_i + C_i) \overline{B_i} + A_i C_i$$

5 阵列除法

2)基于CAS的阵列除法



- 注意连接、输入输出关系
- 使用原码不恢复余数法。第一步一定是减法，故P=1，以后各步做加还是减取决于前一步的商
- 最左边CAS的进位输出是商，且本位商决定下一步是执行加操作还是减操作
- 每执行完一步除法，就将除数右移一位(同手工除法)

计算机组成原理

第三章 运算方法与运算器

3.7 浮点数加减运算

1 规格化浮点数的概念

- 由于浮点数是将数据的表示范围与精确度分别表示的数据表示方法，若不对浮点数的表示作出明确规定，同一个浮点数的表示就不唯一，
- 规格化浮点数是指把一个浮点数按指定的格式进行转换，
- 由于浮点数是将数据的表示范围与精确度分别表示的数据表示方法，若不对浮点数的表示作出明确规定，同一个浮点数的表示就不唯一，
- 以浮点数一般格式为例，规格化浮点数的尾数形式为:

00.1Φ…Φ 或 11.0Φ…Φ。

2 浮点数规格化方法

- 当尾数结果为 00.0Φ…Φ 或 11.1ΦΦΦΦ，需要左规格化即将尾数向左移动，每移动一次，阶码减1，直到尾数形式为 00.1Φ…Φ 或 11.0Φ…Φ。
- 当尾数的结果为 01.Φ…Φ 或 10.ΦΦΦΦ， 表明尾数求和的结果 > 1,此时仅需要执行一次右移规格化， 阶码加 1 ,尾数形式即为00.1Φ…Φ 或 11.0Φ…Φ

3 浮点数加减运算方法及步骤

设 $x = 2^{Ex} \cdot Mx \quad y = 2^{Ey} \cdot My$

则： $x + y = (2^{Ex - Ey} \cdot Mx + My) \times 2^{Ey} \quad (Ey \geq Ex)$

1)对阶

- 求阶差；
- 右移阶码小的浮点数的尾数并同步增加其阶码，直至两数阶码相等。

2)尾数加/减

尾数加/减运算 （用对阶后的尾数）

3)结果规格化

3 浮点数加减运算方法及步骤

4)舍入

右移规格化时可能丢失一些低位的数值位, 为提高精度, 可采取舍入的方法:

- 0 舍 1 入：若右移出的是1则在最低位加1;
- 恒置 1： 只要数字位1被移掉,就将最后一位恒置成1。

5)溢出处理

浮点数的溢出标志：阶码溢出

- 阶码上溢 ： 阶码的符号位为 01
- 阶码下溢 ： 阶码的符号位为 10

4 浮点数加减运算举例

例 设 $x = 2^{010} \times 0.11011011$ $y = 2^{100} \times (-0.10101100)$ 求 x+y

解：先用补码形式表示x 和 y

$[X]_{补} = 00\ 010$，00.11011011

$[Y]_{补} = 00\ 100$，11.01010100

(1) 对阶

$[\Delta E]_{补} = [Ex]_{补} + [-Ey]_{补} = 00010 + 11100 = 11\ 110$

$\therefore \Delta E = -2$ x 的阶码 小于 y 的阶码

将x 的尾数向右移动2位，同时阶码加 2 ，对阶后的 x 为：

$[X]_{补} = 00\ 100$，00.0011011011

4 浮点数加减运算举例

例 设 $x = 2^{010} \times 0.11011011$ $y = 2^{100} \times (-0.10101100)$ 求 x+y

2)尾数运算

$$
\begin{array}{r}
00.00110110\ 11 \\
+\ \underline{11.01010100} \\
11.1000101011
\end{array}
$$

3) 尾数规格化处理

分析发现，只左移一次即可达到规格化要求。规格化后的结果为：

$[X+Y]_{补} = 00\ \ 011\ \ ,\ 11.000101011$

4) 舍入 （ 0 舍 1入 ）

在结果尾数的最低位加1，最后的结果为：

$[X+Y]_{补} = 00\ \ 011\ \ ,\ 11.00010110$ $X+Y = -0.11101010 \times 2^{011}$

4 浮点数加减运算举例

例2 浮点数加减运算过程一般包括对阶、尾数运算、规格化、舍入和判溢出等步骤。设浮点数的阶码和尾数均采用补码表示，且位数分别为5位和7位（均含2位符号位）。若有两个数$X=2^7\times29/32$，$Y=2^5\times5/8$，则用浮点加法计算X+Y的最终结果是:

A．00111 1100010 B. 00111 0100010

C．01000 0010001 D. 发生溢出

解题思路：

- $X= 2^{00111}\times 0.11101$； $Y = 2^{00101}\times 0.101$ ；对阶后大的阶码为00111
- 位数相加后的结果为：01.00010，
- 尾数需右移规格化，同时阶码加1后变成 01 000