第五章　运算方法与运算器（三）

秦磊华 计算机学院

本章主要内容

基于补码数据表示研究运算方法和设计运算器（简）

5.4 定点乘法运算

5.5 定点除法运算

5.4 定点乘法运算

1. 移位操作

算术左移		原数		算术右移
1 1101 1100	←相当于乘2	1110 1110	相当于除2→	1111 0111 0

逻辑左移		原数		逻辑右移
1 1101 1100	←	1110 1110	→	0111 0111 0

2. 二进制乘法的手工过程

```
        1 0 1 0
×       1 0 1 1
---------------
        1 0 1 0      (0)
      1 0 1 0        (1)
    0 0 0 0          (2)
+ 1 0 1 0            (3)
---------------
  1 1 0 1 1 1 0
```

- ♦ 乘法可由加法实现 (!)
- ♦ 每次加数左移位数不同
- ♦ 每次加数的值要么为0，要么为X，取决于对应的乘数Yn
- ♦ 需要长度为2n的积寄存器

2. 二进制乘法的手工过程

```
        1 0 1 0
×       1 0 1 1
---------------
        1 0 1 0
      1 0 1 0
    0 0 0 0
+ 1 0 1 0
---------------
  1 1 0 1 1 1 0
```

1) 乘法可由加法实现 (!)

- 可直接由前面基于FA设计的运算器来实现吗？
- 如何解决？

循环累加

2. 二进制乘法的手工过程

```
        1 0 1 0
×       1 0 1 1
---------------
        1 0 1 0  ← (0)
      1 0 1 0    ← (1)
    0 0 0 0      ← (2)
+ 1 0 1 0        ← (3)
---------------
  1 1 0 1 1 1 0
```

2)每次加数左移位数不同

- 为什么每次加数要左移且位数不同？
- 如何解决？

运算完成后部分积右移1位，

实现累积右移效果

2. 二进制乘法的手工过程

```
          1 0 1 0
×         1 0 1 1
          -------
          1 0 1 0
        - - - - - -
      0 1 0 1 0
    +  1 0 1 0
       - - - - - -
        1 1 1 1
      0 1 1 1 1 0
    + 0 0 0 0
      - - - - - -
      0 1 1 1 1 0
      0 0 1 1 1 1 0
    + 1 0 1 0
      -------------
      1 1 0 1 1 1 0
      0 1 1 0 1 1 1 0
```

```
          1 0 1 0
×         1 0 1 1
    -------------
          1 0 1 0
        1 0 1 0
      0 0 0 0
+   1 0 1 0
    -------------
    1 1 0 1 1 1 0
```

5.4 定点乘法运算

2. 二进制乘法的手工过程

```
        1 0 1 0
×       1 0 1 1
---------------
        1 0 1 0
      → 0 1 0 1 | 0
      + 1 0 1 0
      - - - - - - - -
        1 1 1 1
      → 0 1 1 1 | 1 0
      + 0 0 0 0
      - - - - - - - -
        0 1 1 1 | 1 0
      → 0 0 1 1 | 1 1 0
      + 1 0 1 0
      ---------------
        1 1 0 1 | 1 1 0
      → 0 1 1 0 | 1 1 1 0
```

3) 需要长度为2n的积寄存器

- ♦ 长度为2n的积是逐步形成的
- ♦长度为2n的积由2部分构成

⇩

如何构成?

3. 原码一位乘法算法

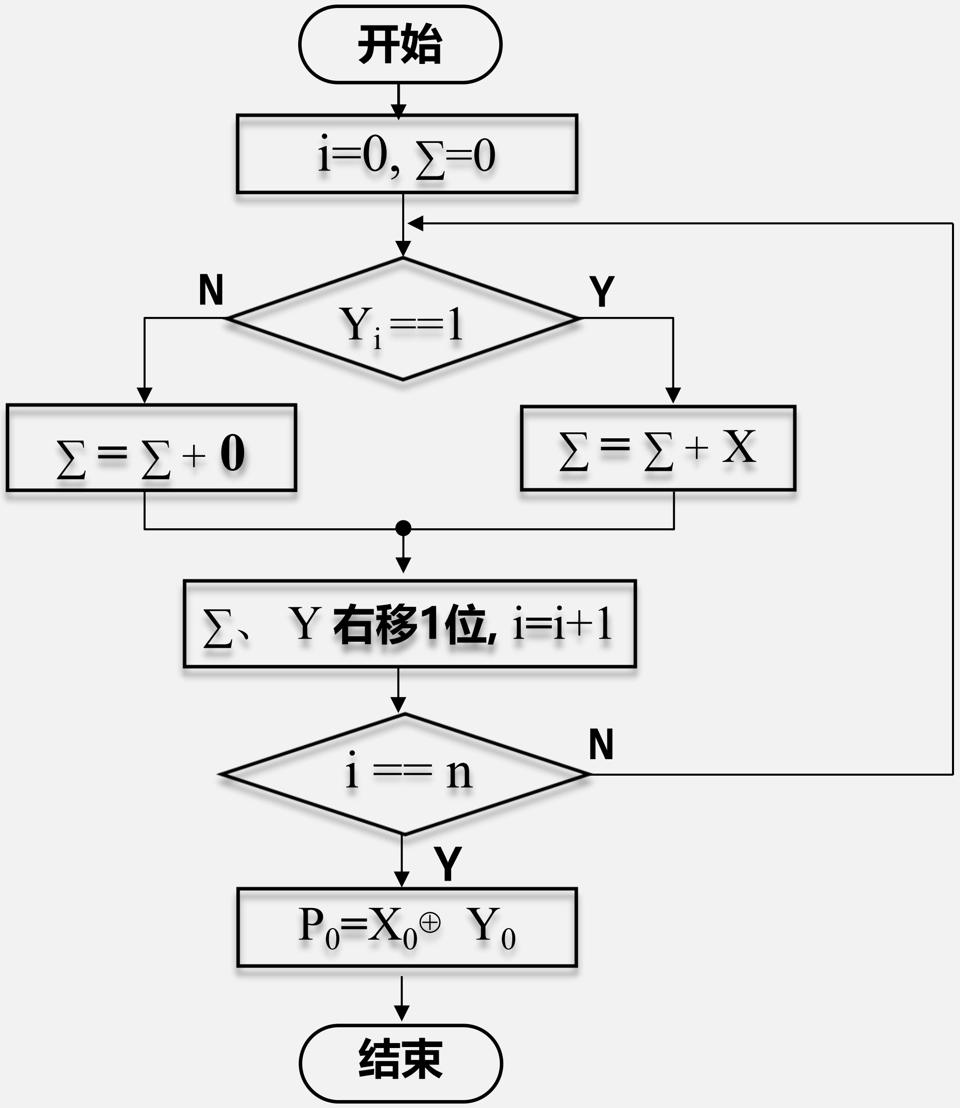
(1) 符号单独运算：直接异或

(2) 绝对值相乘

仅需考虑数值部分的计算

(3) n次加法，n 次右移

(4) 部分积右移包含进位位



3. 原码一位乘法算法

例1 X=+ 1101, Y=-11

求$[X]_原 \times [Y]_原$

解： $[X]_原 = 01101$

$[Y]_原 = 10011$

(1) 符号单独运算：直接异或

(2) 绝对值相乘

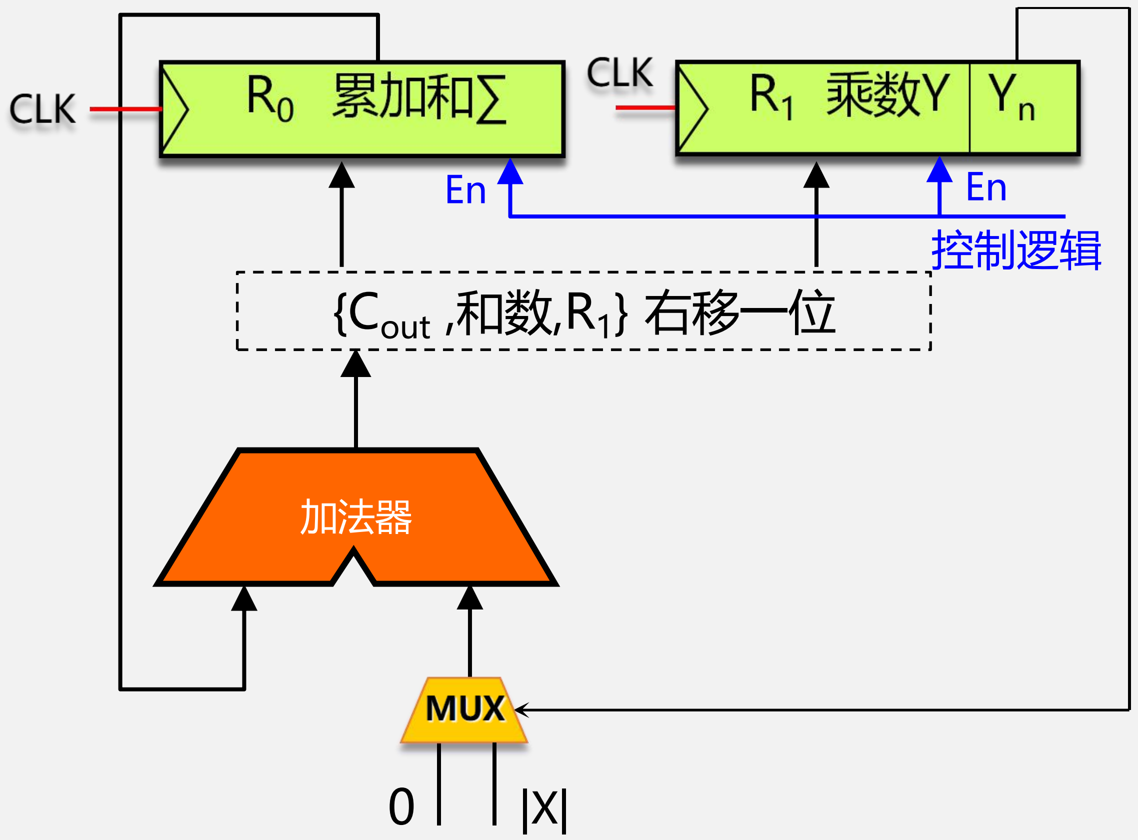
仅需考虑数值部分的计算

∑=0	000000		乘数判断位Y_n 0001<u>1</u>
	+001101		
	001101		
→	000110	1(?)	1000<u>1</u>
	+001101		
	010011	1	
→	001001	11	1100<u>0</u>
	+000000		
	001001	11	
→	000100	111	1110<u>0</u>
	+000000		
	000100	111	
→	000010	0111	0111<u>0</u>

$[X]_原 \times [Y]_原 = 1\ 0010\ 0111$

华中科技大学 计算机科学与技术学院 School of Computer Science & Technology, HUST

4. 原码一位乘法硬件实现



- R_0存放部分积高n位，初值为0
- R_1存放Y、∑其它位（如何载入Y初值）
- 运算结果右移后送寄存器输入！
- 时钟到来，R_0，R_1锁存新值
- 状态机控制使能信号停机
- 停机后乘积存放在R_0，R_1中

$$\{\sum, Y\} = \{\sum + Y_n|X|, Y\} / 2$$ 逻辑右移

5.补码booth一位乘法

1) 被乘数X符号任意，乘数Y为正

$[X]_补=X_0.X_1X_2\cdots X_n \quad [Y]_补=0.Y_1Y_2\cdots Y_n$

$[X]_补\times[Y]_补=(2+X)\times Y=(2^{n+1}+X)\times Y$ （MOD 2）

$= 2^{n+1}Y+XY$

$= 2\times 2^n\times 0.Y_1Y_2\cdots Y_n+XY$

$= 2\,(Y_1Y_2\cdots Y_n)+XY$

$= 2+XY$ （MOD 2）

$= [XY]_补 = [X]_补\times Y$ (1)

5.补码booth一位乘法

2) 被乘数[X]符号任意，乘数[Y]为负数

$[X]_{补}=X_0.X_1X_2\cdots X_n$ $[Y]_{补}=1.Y_1Y_2\cdots Y_n$

$[Y]_{补}=2+Y$ ⇒ $Y=[Y]_{补}-2$

$Y=[Y]_{补}-2=1+0.Y_1Y_2\cdots Y_n-2 \quad =0.Y_1Y_2\cdots Y_n-1$

$[X\times Y]_{补}=[X\times(0.Y_1Y_2\cdots Y_n-1)]_{补}$

$=[X\times 0.Y_1Y_2\cdots Y_n-X]_{补}$

$=[X\times 0.Y_1Y_2\cdots Y_n]_{补}-[X]_{补}$

$=[X]_{补}\times 0.Y_1Y_2\cdots Y_n-[X]_{补}$

$=[X]_{补}\times 0.Y_1Y_2\cdots Y_n-Y_0[X]_{补}$

当 Y为正数时，$Y_0=0$，因此，该式也包含了1)

华中科技大学 计算机科学与技术学院 School of Computer Science & Technology, HUST

5.补码booth一位乘法

$$[X\times Y]_{补}=[X]_{补}\times 0.Y_1Y_2\cdots Y_n - Y_0[X]_{补}$$

$$=[X]_{补}\times(-Y_0+0.Y_1Y_2\cdots Y_n)$$

$$=[X]_{补}\times(-Y_0+Y_12^{-1}+Y_22^{-2}+\cdots Y_n\,2^{-n})$$

$$=[X]_{补}\times[-Y_0+(Y_1-Y_12^{-1})+(Y_22^{-1}-Y_22^{-2})+\cdots(Y_n2^{-n+1}-Y_n2^{-n})]$$

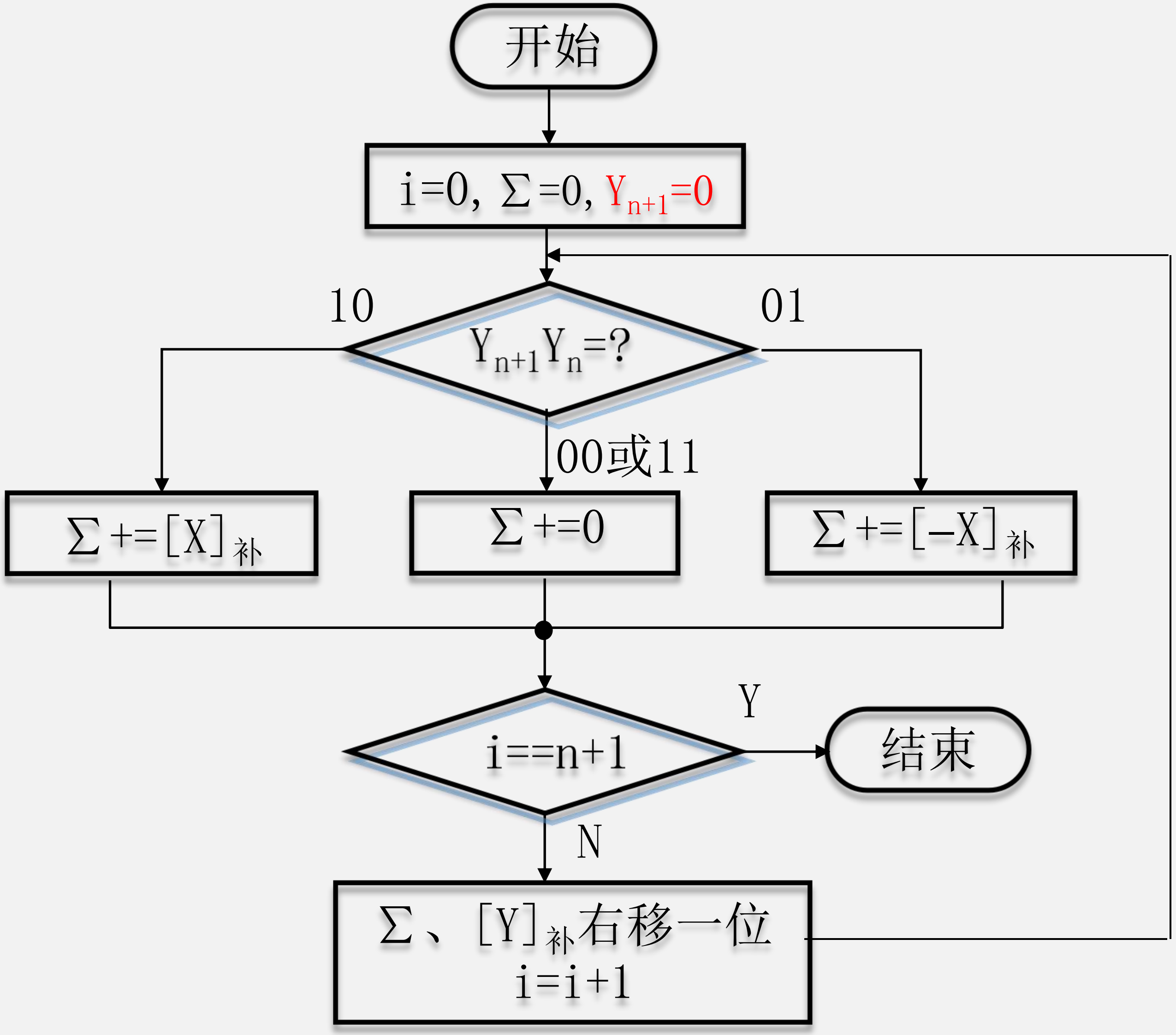
$$=[X]_{补}\times[(Y_1-Y_0)+(Y_2-Y_1)2^{-1}+(Y_3-Y_2)2^{-2}+\cdots(-Y_n)2^{-n}]$$

$$=[X]_{补}\times[(Y_1-Y_0)+(Y_2-Y_1)2^{-1}+(Y_3-Y_2)2^{-2}+\cdots(0-Y_n)2^{-n}]$$

$Y_{n+1}=0$

5.补码booth一位乘法

- 加法次数:n+1次
- 算术右移次数为n次
- 符号位参与运算



5.补码booth一位乘法

例2 X=+1101，Y=-11

求$[X]_{补} \times [Y]_{补}$

解： $[X]_{补}$=01101, $[-X]_{补}$=10011

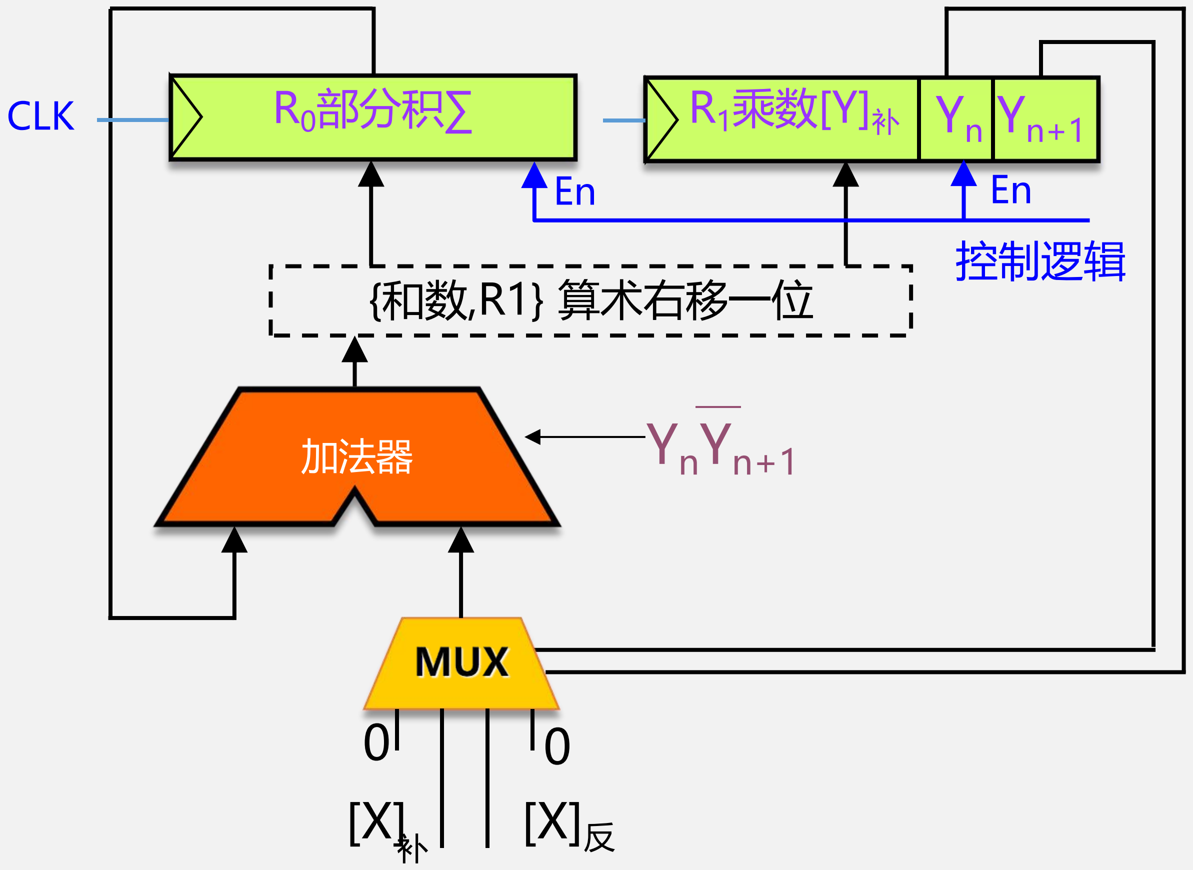
$[Y]_{补}$=11101

∑=0	00 0000		乘数判断位Y_nY_{n+1}
	+11 0011		1110**10**
	11 0011		
	→11 1001	1	1 111**01**
	+00 1101		
	00 0110	1	
	→00 0011	01	01 11**10**
	+11 0011		
	11 0110	01	
	→11 1011	001	001 1**11**
	+00 0000		
	11 1011	001	
	→11 1101	1001	1001 **11**
	+00 0000		
	11 1101	1001	

$[X]_{补} \times [Y]_{补}$ = 1 110 11001

华中科技大学 计算机科学与技术学院 School of Computer Science & Technology, HUST

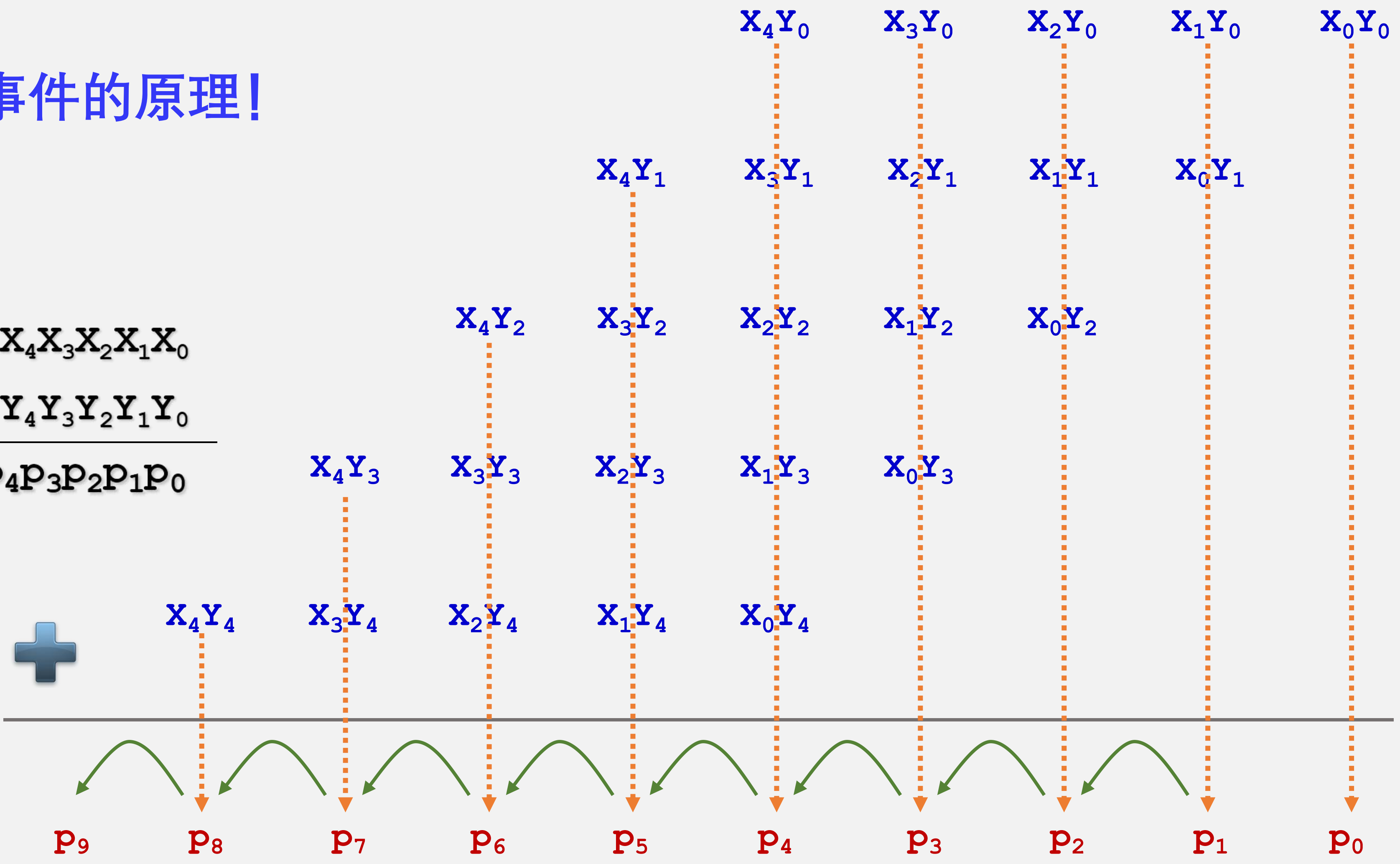
6.补码booth一位乘法硬件实现



7.阵列乘法器

加快经常性事件的原理！

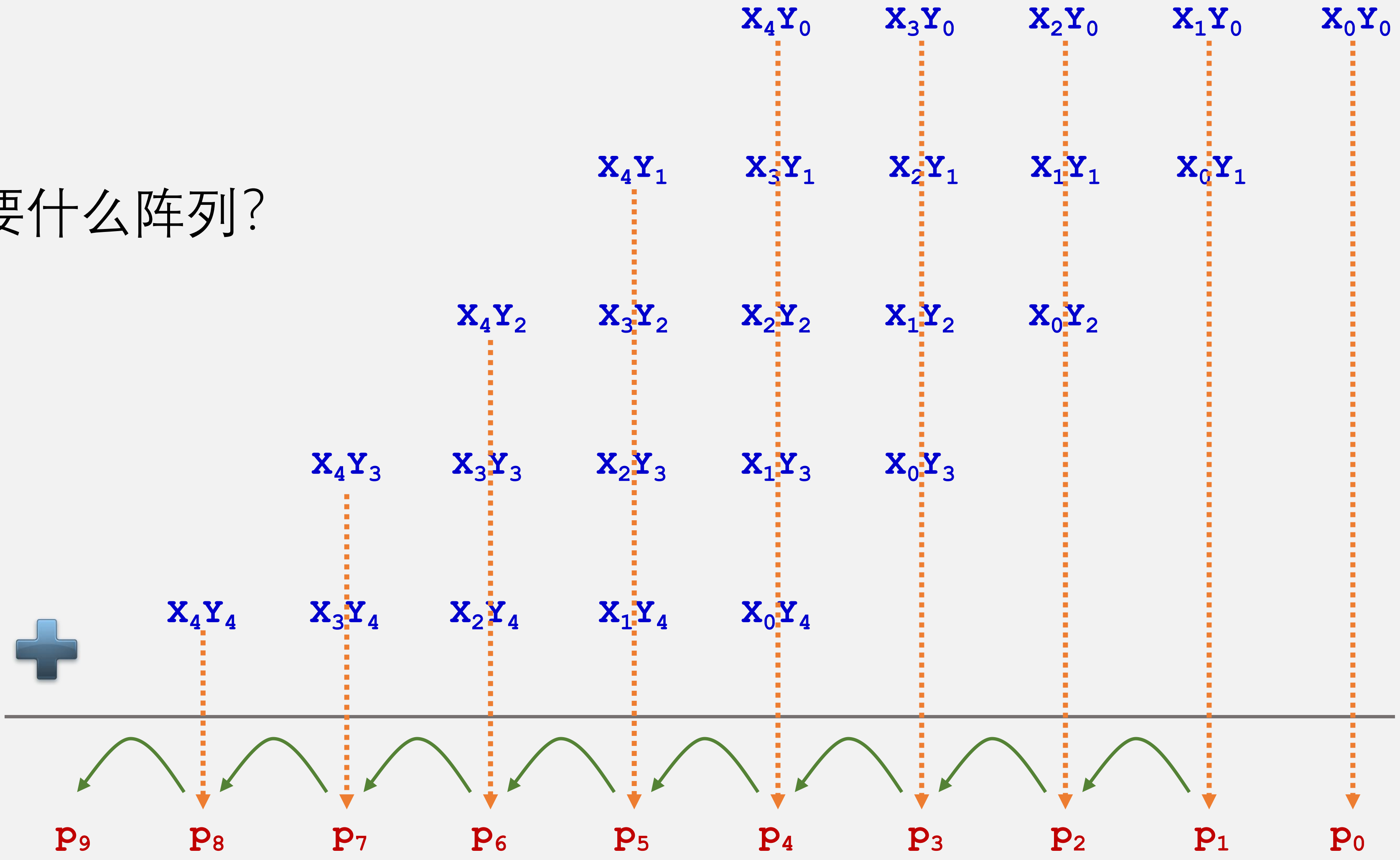
$$
\begin{array}{rr}
 & X_4X_3X_2X_1X_0 \\
\times & Y_4Y_3Y_2Y_1Y_0 \\
\hline
 & P_9P_8P_7P_6P_5P_4P_3P_2P_1P_0
\end{array}
$$



华中科技大学 计算机科学与技术学院 School of Computer Science & Technology, HUST

7. 阵列乘法器

- 阵列?
- 实现右边乘法需要什么阵列?



7. 阵列乘法器

1) 与门阵列

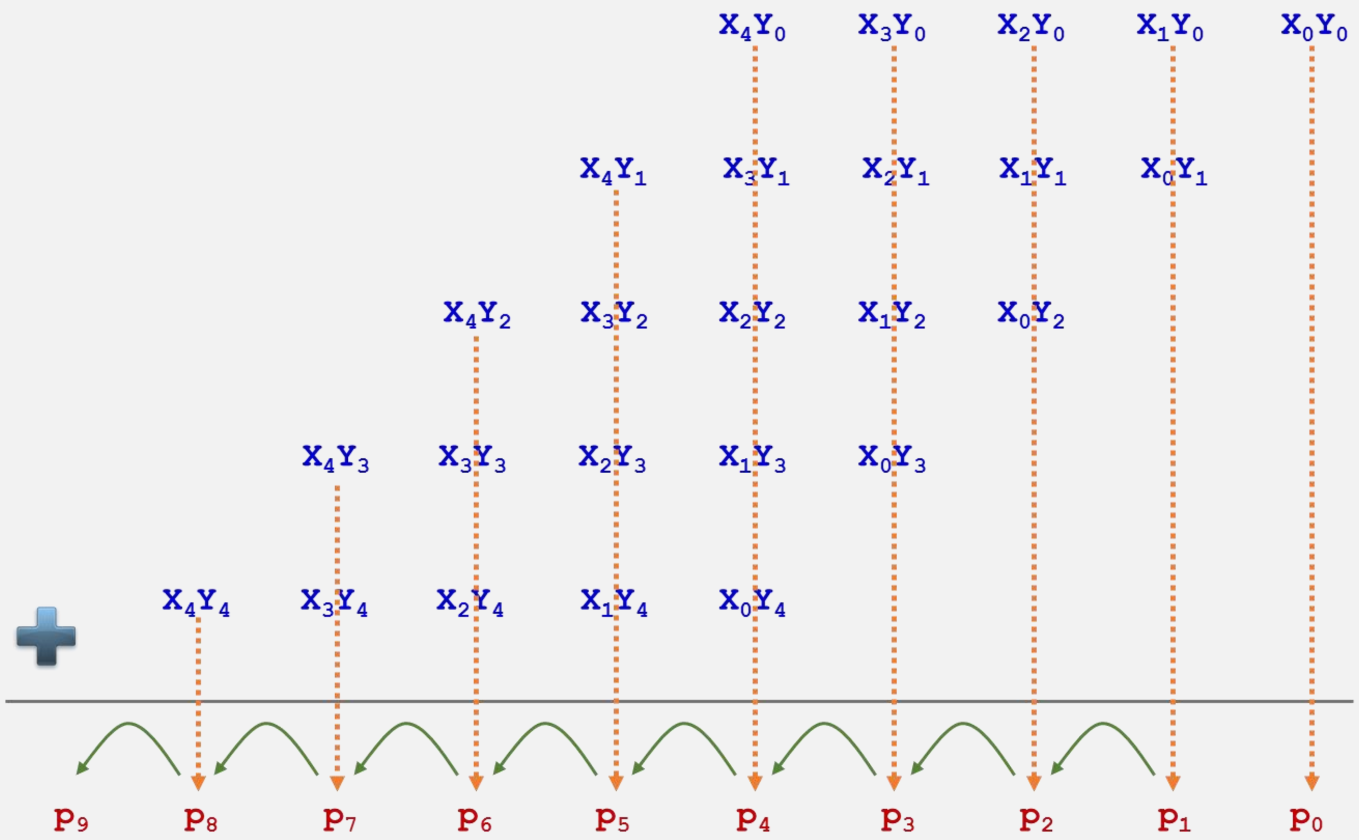
R=X*Y

$1 \times 1 = 1 \quad = 1 \cdot 1$

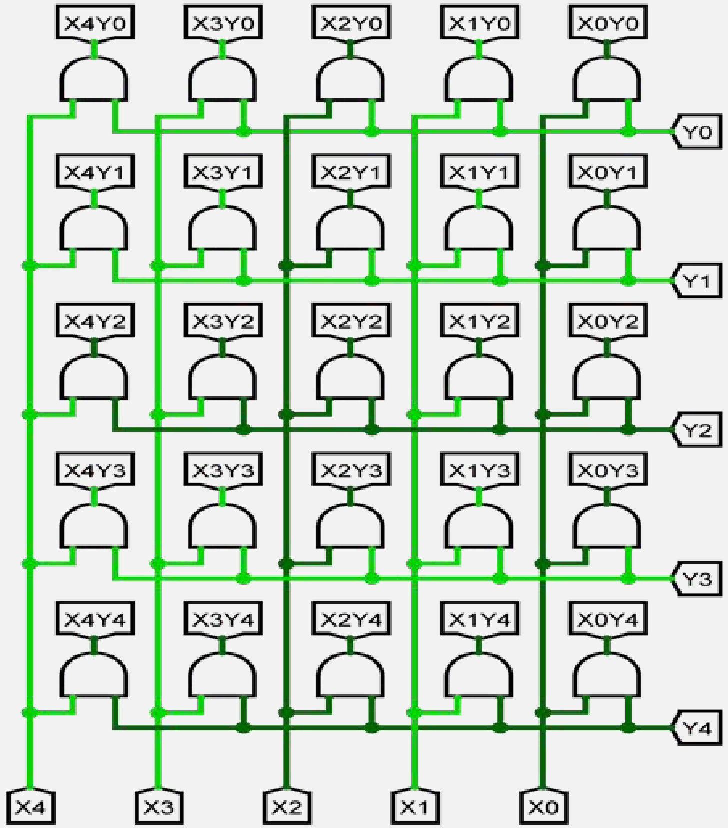
$1 \times 0 = 0 \quad = 1 \cdot 0$

$0 \times 1 = 0 \quad = 0 \cdot 1$

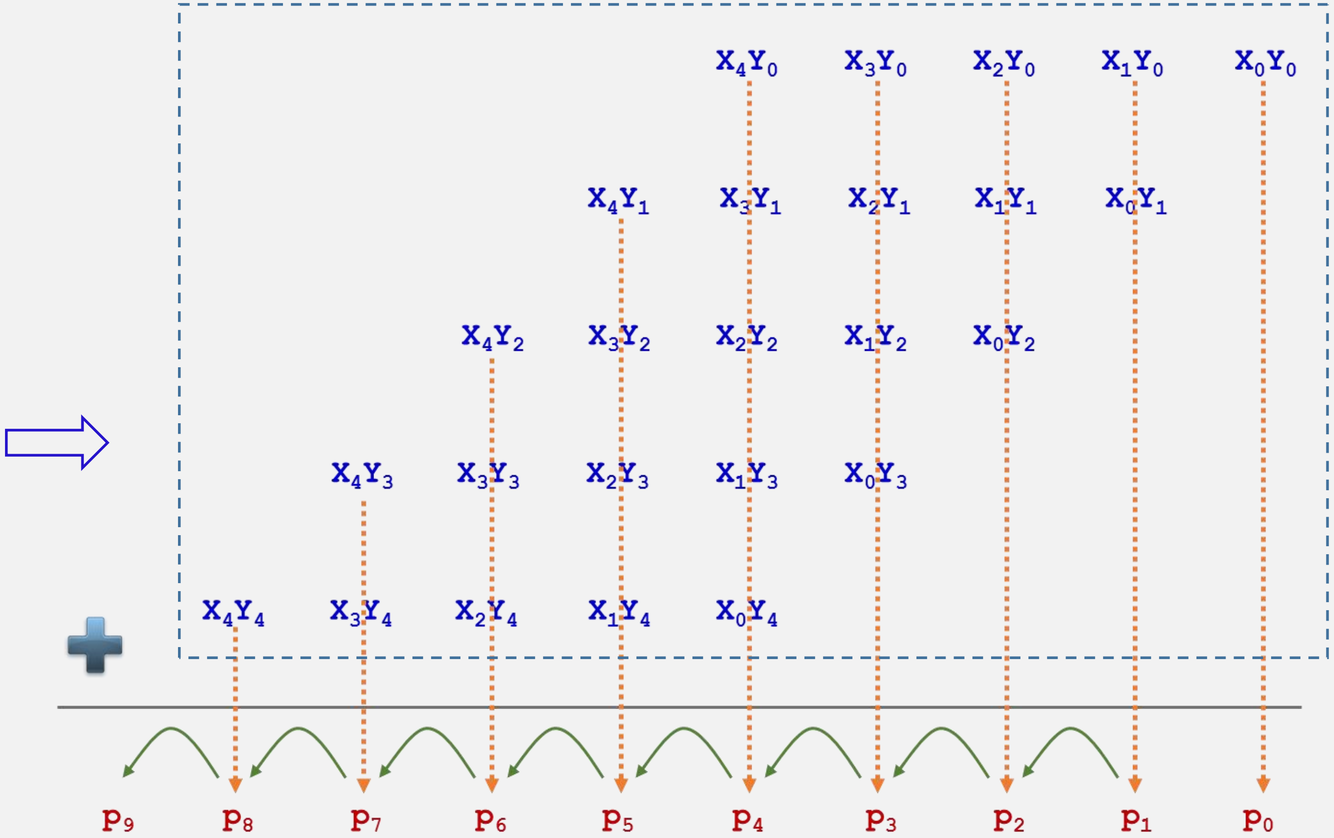
$0 \times 0 = 0 \quad = 0 \cdot 0$



7. 阵列乘法器



X4Y0 X3Y0 X2Y0 X1Y0 X0Y0
Y0
X4Y1 X3Y1 X2Y1 X1Y1 X0Y1
Y1
X4Y2 X3Y2 X2Y2 X1Y2 X0Y2
Y2
X4Y3 X3Y3 X2Y3 X1Y3 X0Y3
Y3
X4Y4 X3Y4 X2Y4 X1Y4 X0Y4
Y4
X4 X3 X2 X1 X0

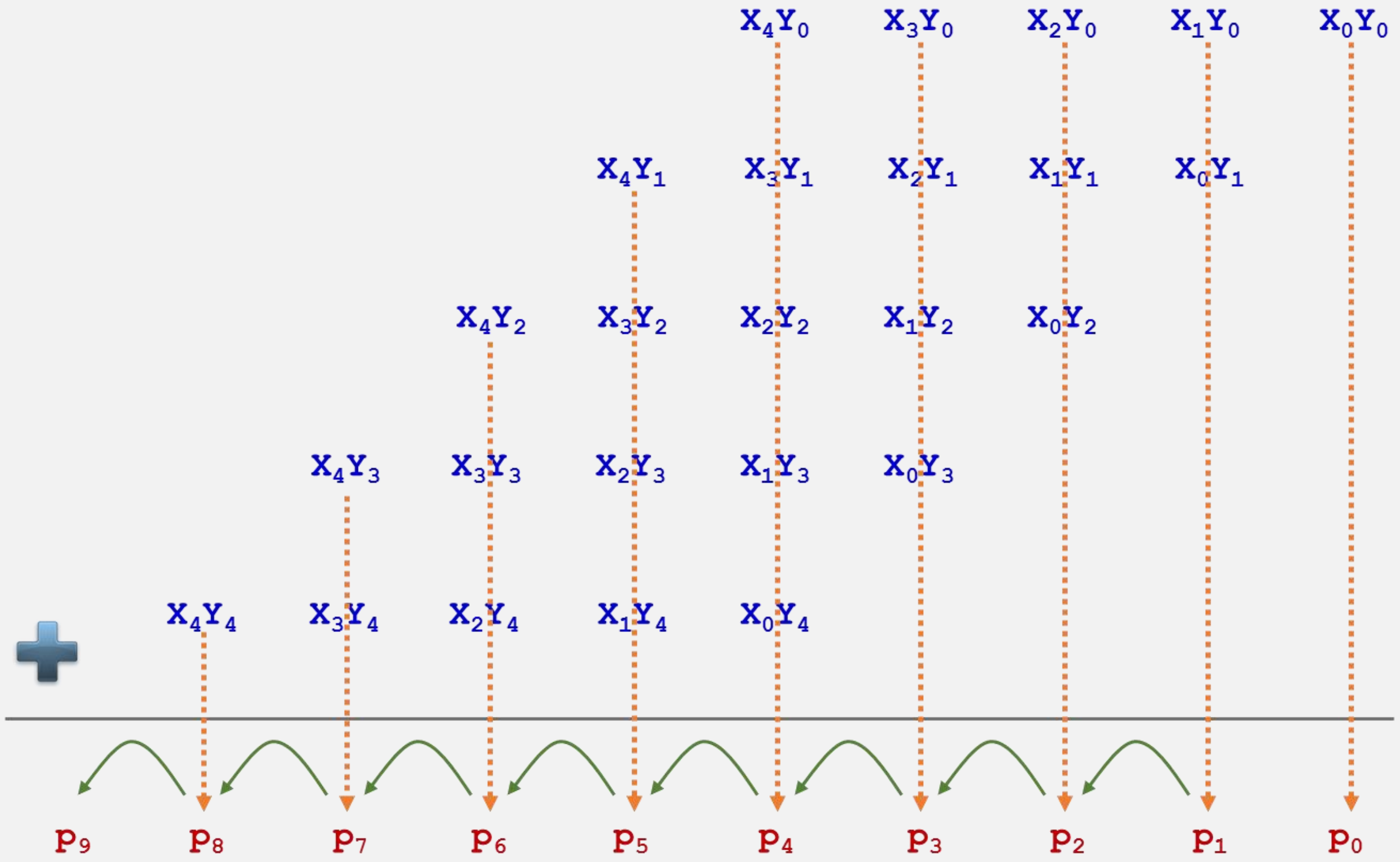


X_4Y_0 X_3Y_0 X_2Y_0 X_1Y_0 X_0Y_0
X_4Y_1 X_3Y_1 X_2Y_1 X_1Y_1 X_0Y_1
X_4Y_2 X_3Y_2 X_2Y_2 X_1Y_2 X_0Y_2
X_4Y_3 X_3Y_3 X_2Y_3 X_1Y_3 X_0Y_3
X_4Y_4 X_3Y_4 X_2Y_4 X_1Y_4 X_0Y_4
P_9 P_8 P_7 P_6 P_5 P_4 P_3 P_2 P_1 P_0

7. 阵列乘法器

1) 与门阵列

2) 还需要什么阵列?

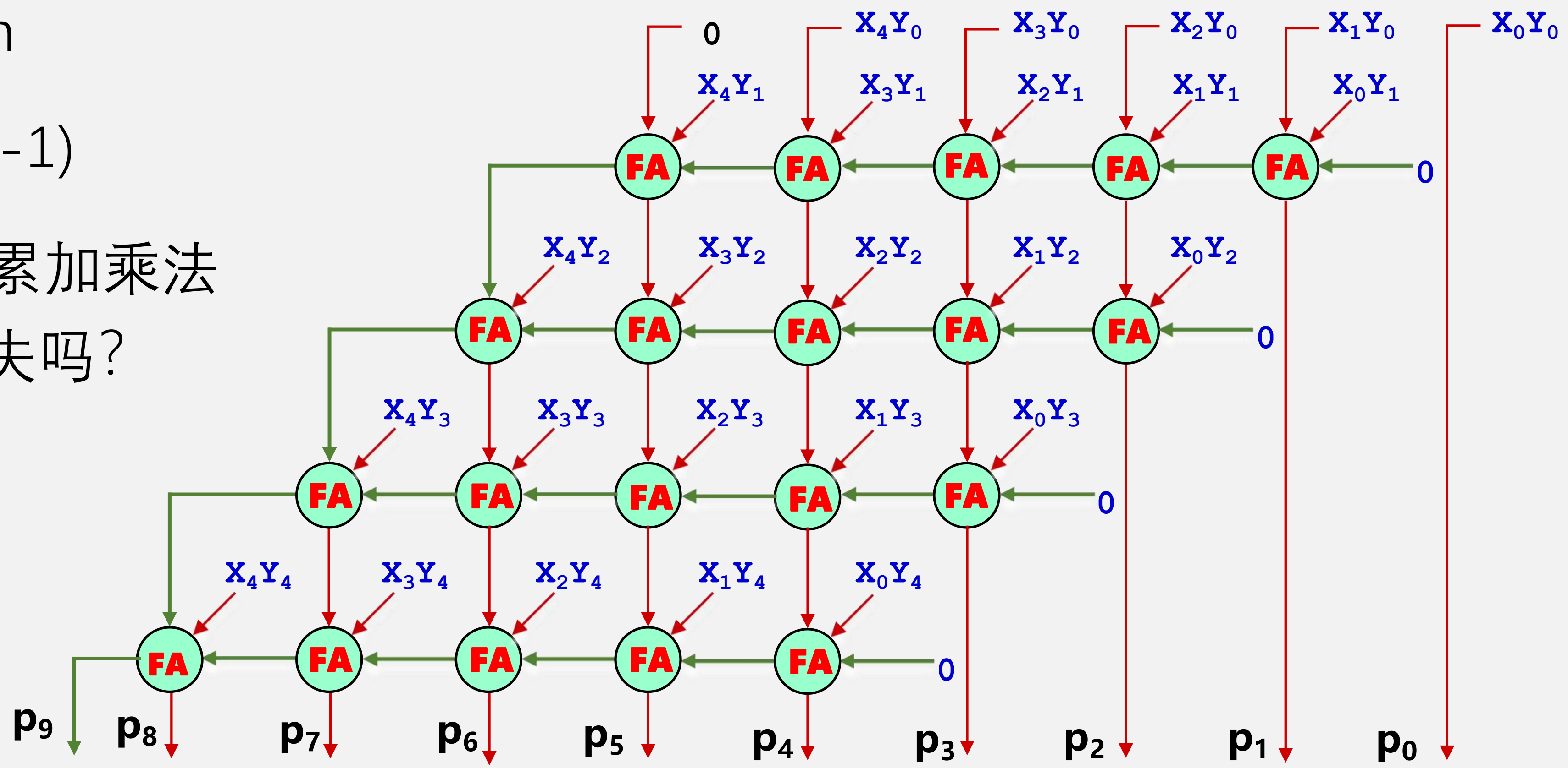


7. 阵列乘法器

1)与门阵列:n*n

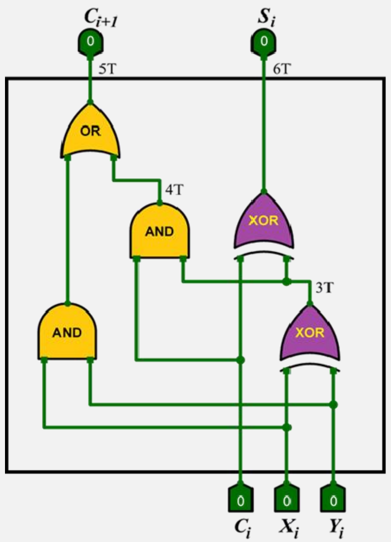
2) FA阵列:n*(n-1)

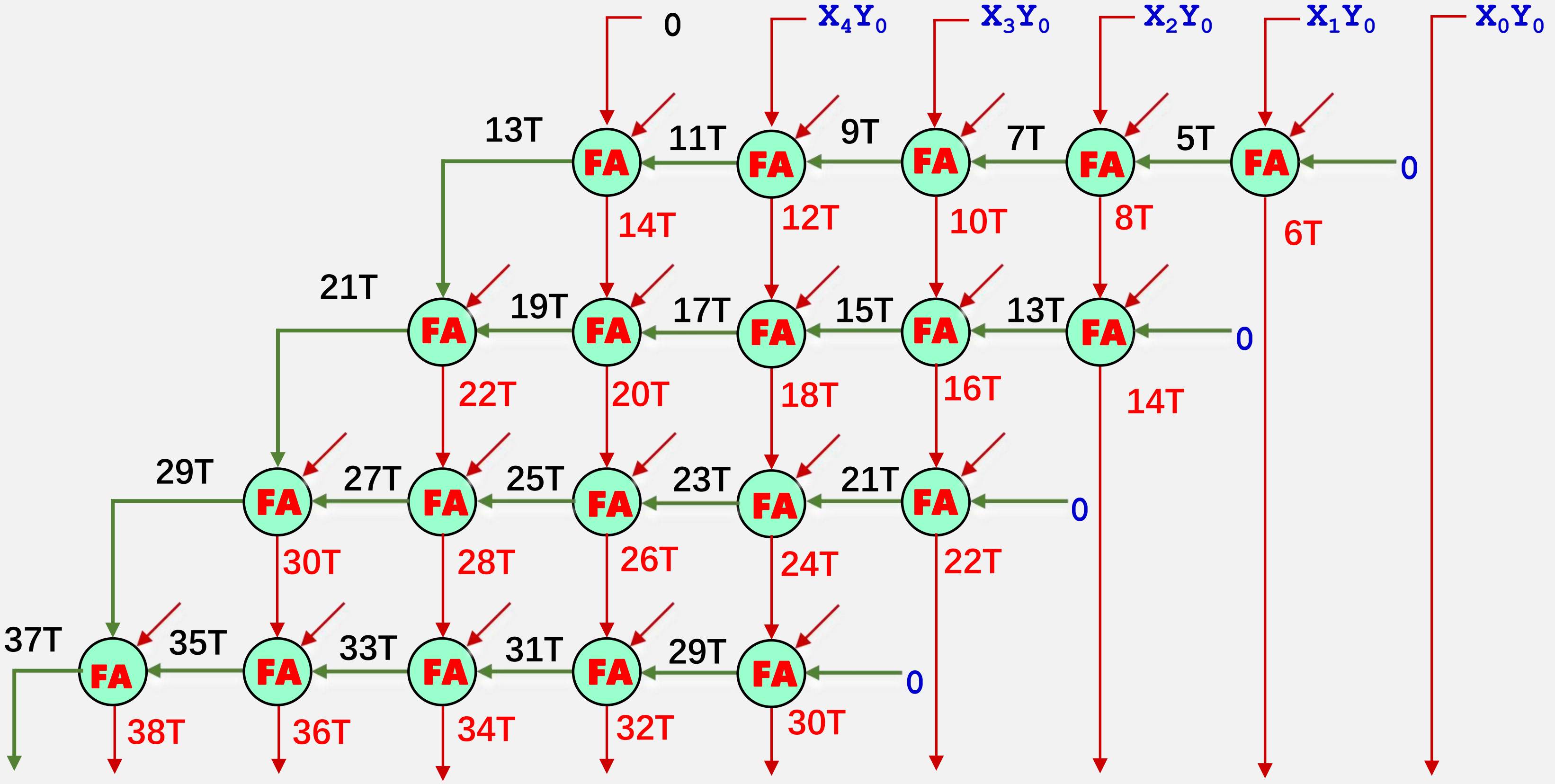
3) 能缓解循环累加乘法带来的性能损失吗?



7. 阵列乘法器

◆总延时为多少?



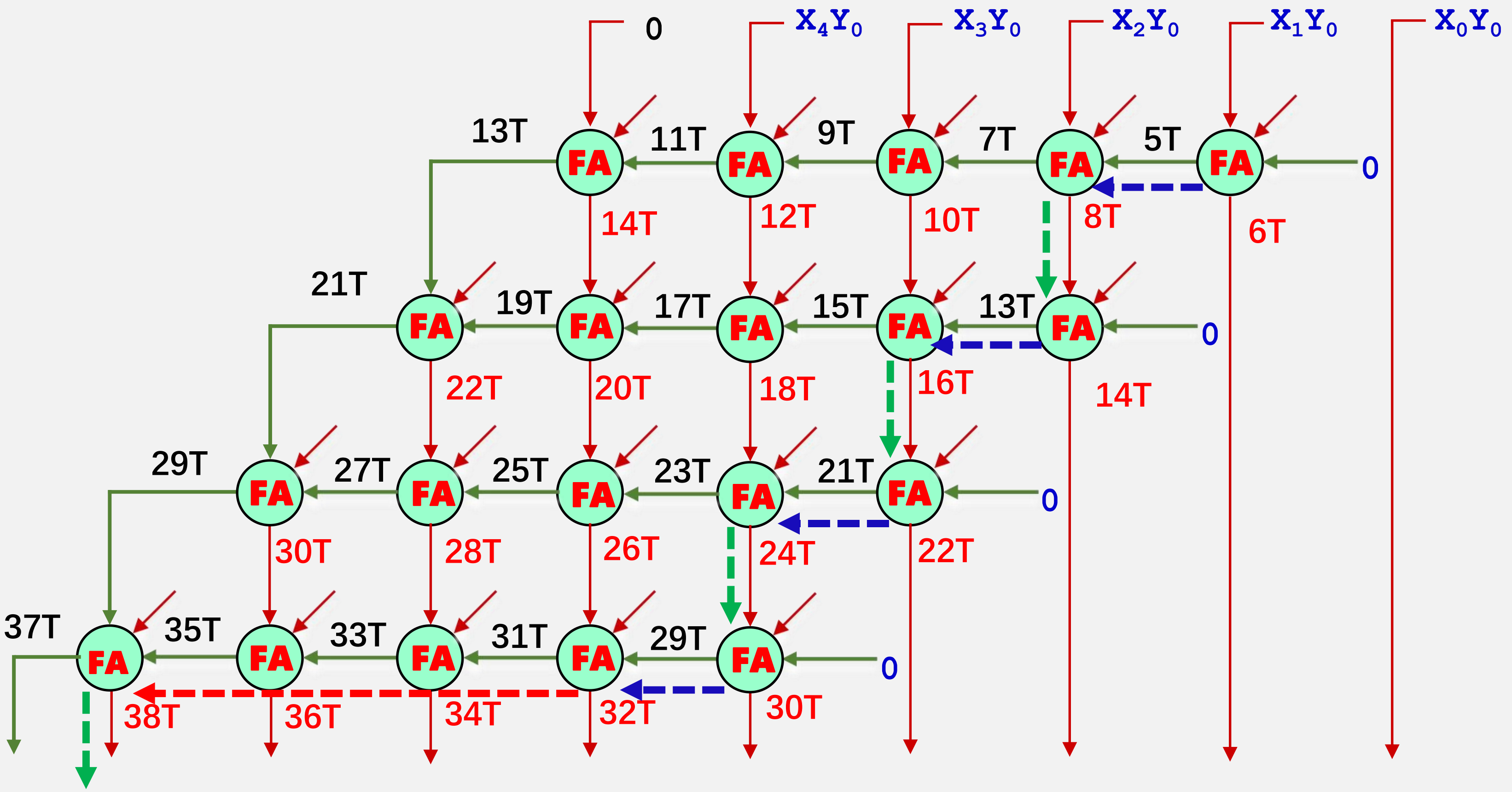


华中科技大学 计算机科学与技术学院 School of Computer Science & Technology, HUST

7. 阵列乘法器

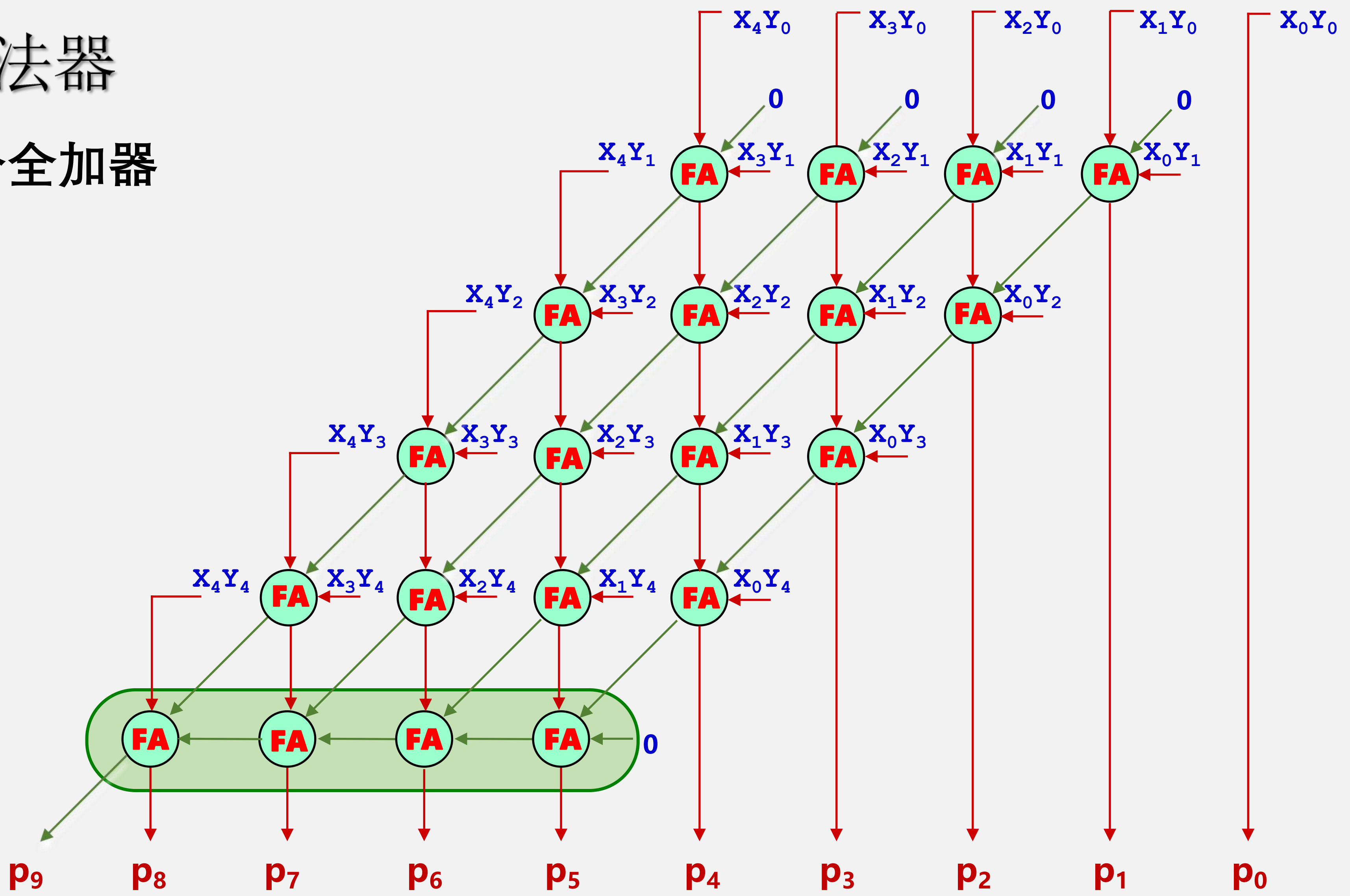
◆总延时为多少?

◆还能优化吗?

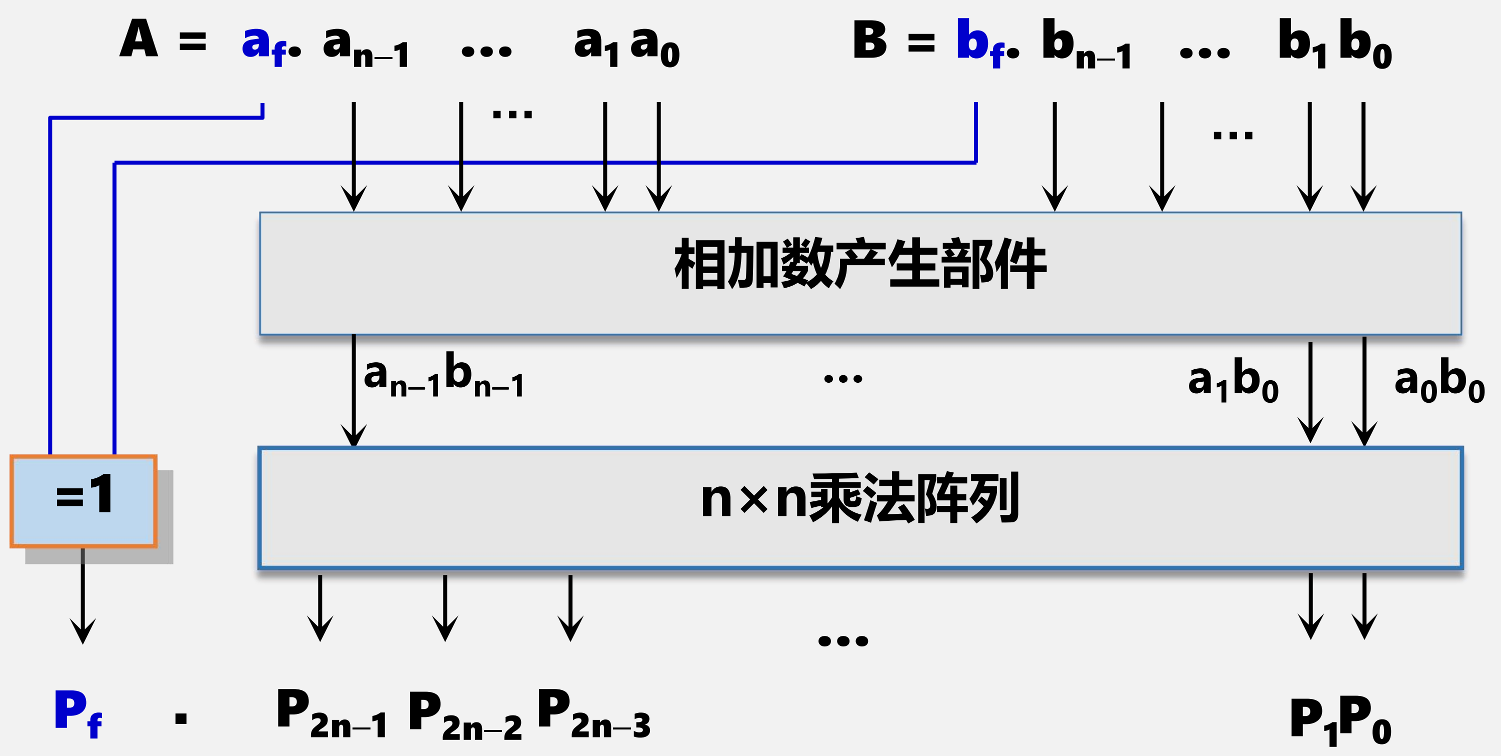


7. 阵列乘法器

♦n*(n-1)个全加器

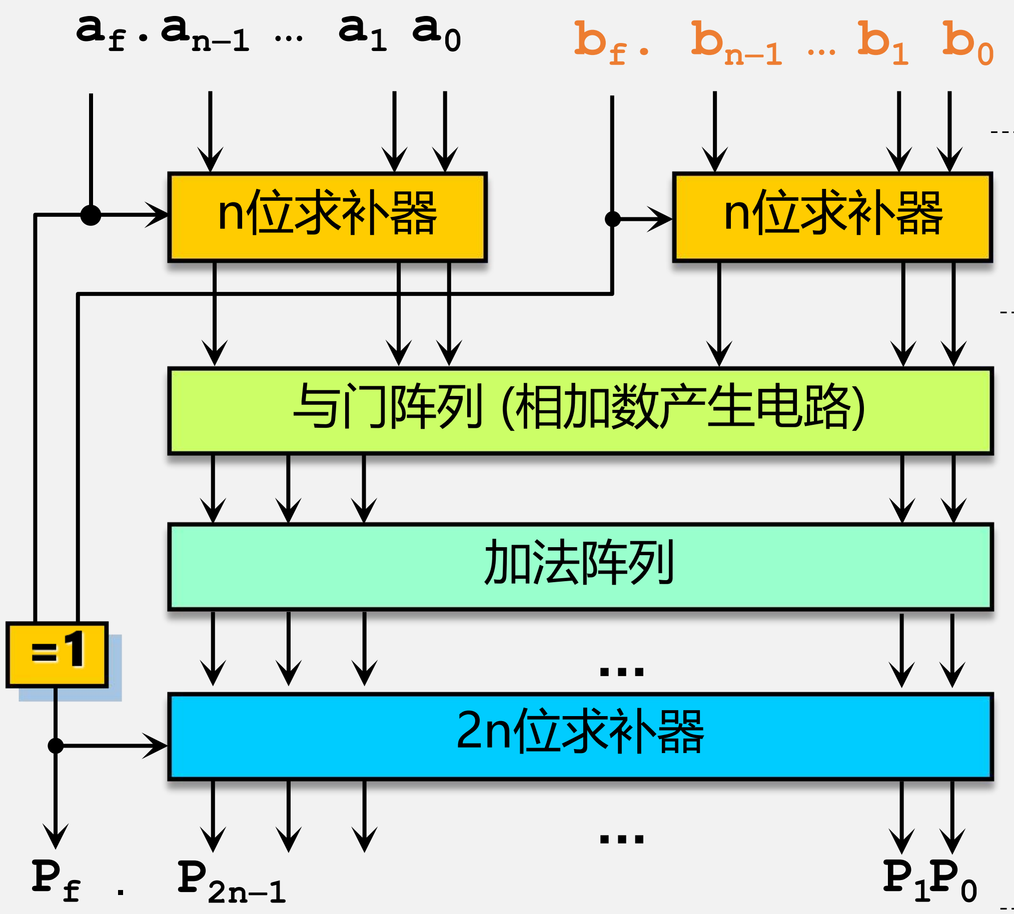


7. 阵列乘法器



(原码阵列乘法器)

7. 阵列乘法器



(补码阵列乘法器)

X=+15 y= -13

$[X]_补$=0 1111 $[Y]_补$=1 0011

X' = 1111 Y'= 1101

```
      1 1 1 1
    0 0 0 0
  1 1 1 1
1 1 1 1
-----------------
1 1 0 0 0 0 1 1
```

$[x\bullet y]_补$=1 00111101

8. 高级语言中的整数乘法及溢出判断

高级语言中两个n位整数相乘得到的结果通常也是一个n位整数，结果只取2n位乘积中的低n位。

```
int mul(int x, int y)
{
    int z=x*y;
    return z;
}
```

如何判断Z是否溢出？

https://www.csdn.net/tags/MtTaMg4sNTE4MjM4LWJsb2cO0O0O.html

8. 高级语言中的整数乘法及溢出判断

https://blog.csdn.net/weixin_31092009/article/details/117121295

```
/* Determine whether arguments can be multiplied without overflow.
*/
int tmul_ok(int x, int y)
{undefined
#if 0
int p = x * y;
return !x || p/x==y;
#endif
return umul_ok(x, y); /* 直接调用 */
}
```

9. 汇编语言中的整数乘法及溢出判断

- ♦硬件不判溢出，仅保留2n位乘积，供软件使用
- ♦程序不判断溢出，编译器也不生成用于溢出处理的代码，会发生整数溢出问题。
- ♦乘法指令的操作数长度为n,而乘积长度为2n：

IA-32:

16位乘积结果存放在AX (8*8)

32位乘积结果存放在DX-AX (16*16)

64位乘积或EDX-EAX (32*32)

MIPS: 32位带符号整数相乘，64位乘积置于两个32位内部寄存器Hi和Lo中，可根据Hi寄存器中的每一位是否等于Lo寄存器中的第一位来进行溢出判断.

5.4 定点乘法运算

https://www.cnblogs.com/os2os/p/6565252.html

1996年6月4日，阿丽亚娜5型运载火箭（Ariane 5）在法国库鲁的欧洲运载火箭发射场发射，37秒后火箭解体并爆炸。火箭的开发费用大约70亿美元，火箭本体及运载的设备价值约5亿美元。两周后的调查报告指出，事故原因来自将火箭的水平速度的64位浮点数转换成16位整数时的溢出。

使用强类型语言时，当数值类型不相同时，转换是必须的。从位数多的数据类型向少的转换有危险，但并不是一定出问题。

阿丽亚娜5型的转换，在4型时就在使用，但4型没有出问题。因为在决定做浮点数到整数的转换前，开发人员做过仔细调查，可以确定浮点数转换后的大小不会超过16位整数的范围，也就是说不会有溢出。但是，在开发5型时，这部分功能被直接拿来使用，而没有意识到5型火箭的水平速度产生的数值要比4型大很多，16位整数类型不能放下转换后的数值，这导致了灾难性的事故。

10. 高级语言中的变量与常量乘法

♦按照前述运算方法，整数乘法运算比移位和加法等运算所用时间长很多(!)

♦编译器在处理变量与常数相乘,如20*X时，往往以移位、加法和减法的组合运算来代替乘法运算。

x*20转换为(x<<4)+(x<<2)

一次乘法转换成了一次移4位、一次移2位和1次加法

x*15转换为(x<<4) - x

♦移位加减组合运算和直接相乘结果一样（包括溢出）

♦是否优化取决于组合运算周期数是否小于乘法开销

5.5 定点除法运算

1. 手工除法

```
              0.1101  ← 商
0.1011 ) 0.1001001
         0.01011
         0.001110
         0.001011
         0.00001110
         0.00001011
余数     0.00000011
```

- ♦除法可由减法实现
- ♦ Y每次右移次数不等
- ♦要判断两数是否够减（？）

- 不够减，商上0,
- 够减，商上1，求余数
- 除数左移一位
- 继续按规则上商直至所需位数

2. 计算机中除法对手工除法的改进

1)对手工算法的改进

- 判断两数的大小用减法实现，若差>0 ,则够除;反之不够除;
- 将手工每次右移除数,改为左移被除数;
- 最后的余数需要右移(2)中左移的次数.

3. 原码恢复余数法除法

◆**法则:**

设$[X]_{原}=X_fX_1X_2X_3\cdots Xn$, $[Y]_{原}=Y_fY_1Y_2Y_3\cdots Yn$, 用原码一位除法求

$Q=Q_0Q_1Q_2Q_3\cdots Qn = X/Y$

则: $Q_f = X_f \oplus Y_f$

$[Q]_{原}=(X_f \oplus Y_f)+(0X_1X_2X_3\cdots Xn/0Y_1Y_2Y_3\cdots Yn)$

◆最后的余数需要右移(2)中左移的次数.

3. 原码恢复余数法除法

试商通过X – Y 进行

- 试商结果 > 0时，商上 1 ；
- 试商的结果 < 0 时？？，

  恢复余数!

  即执行+Y，显然，此时不能上商

5.5 定点除法运算

3. 原码恢复余数法除法

例已知 X=1001，Y= -1011

用原码一位除法求 X ÷ Y

解：$[X]_{原}$= 01001

$[Y]_{原}$= 11011

$[|X|]_{补}$ = 01001

$[|Y|]_{补}$ = 01011

$[-|Y|]_{补}$ = 10101

Q=11101　R= 00001$*2^{-4}$

0⊕1 =1

被除数/余数R	上商位Q_n（? ）	说明
001001 [-Y] + 110101		X-Y试商
111110 + 001011	0	R<0,商上0, +Y恢复余数
001001 ← 010010 [-Y] + 110101		R=2R，-Y试商
000111 ← 001110 [-Y] + 110101	01	R>0,Q_n=1 R=2R，-Y试商
000011 ← 000110 [-Y] + 110101	011	R>0,Q_n=1 R=2R,-Y试商
111011 + 001011	0110	R<0,Q_n=0, +Y 恢复余数
000110 ← 001100 [-Y] + 110101		
← 000001	01101	R>0,Q_n=1

3. 原码恢复余数法除法

- ♦需要进行恢复余数的操作
- ♦恢复余数的操作次数不确定，影响除法速度和控制。
- ♦实际应用通常采用不恢复余数除法/加减交替法。

4. 原码加减交替法除法

- 设某次除法运算余数为$R_i>0$，将R_i左移一位，然后减除数试商：

  $2R_i-Y$

- 若结果小于0，则商上0，并恢复余数：

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

- 再左移并试商：

  $2*2R_i-Y=4R_i-Y$ ……… (1)

- 若$2R_i-Y<0$，商上0，不恢复余数，而是直接左移一位并加Y:

  $2*(2R_i-Y)+Y=4R_i-Y$ ……… (2)

华中科技大学 计算机科学与技术学院 School of Computer Science & Technology, HUST

4. 原码加减交替法除法

例已知 X=1001，Y= -1011

用原码一位除法求 X ÷ Y

解： $[X]_原$= 01001

$[Y]_原$= 11011

$[|X|]_补$ = 01001

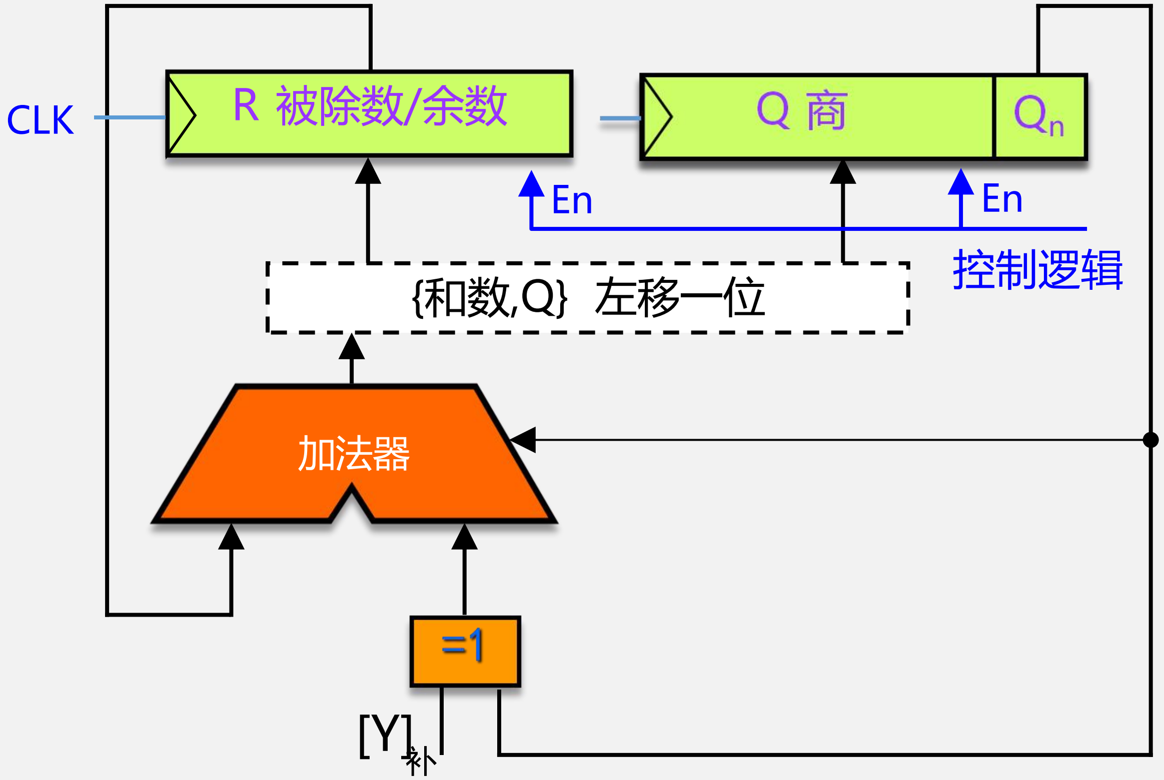
$[|Y|]_补$ = 01011

$[-|Y|]_补$ = 10101

Q=11101　R= $0.0001*2^{-4}$

	被除数/余数R	上商位Q_n（? ）	说明
	001001		X-Y试商
[-Y]	+ 110101		
	111110	0	R<0,商上0,
	← 111100		R=2R+Y
[Y]	+ 001011		
	000111	01	R>0,Q_n=1
	← 001110		R=2R-Y
[-Y]	+ 110101		
	000011	011	R>0,Q_n=1
	← 000110		R=2R-Y
[-Y]	+ 110101		
	111011	0110	R<0,商上0,
	← 110110		R=2R+Y
[Y]	+ 001011		
	← 000001	01101	R>0,Q_n=1

4. 原码加减交替法除法(电路)



5. 补码加减交替法除法

1) 符号位参加运算

2)试商方法不同于原码一位除法

♦回顾原码一位除法的试商-----减法实现

♦若采用原码试商方法存在的问题 (为什么不能直接减来试商)

3)补码一位除法的试商及运算方法

♦被除数与除数同号，被除数减除数；反之加除数,该步不上商;

♦余数与除数同号，商上1，余数左移一位，下次减除数;
反之，商上0，余数左移一位，下次加除数

♦重复上一步，包括符号位在内共执行n + 1次，且最后一步只移商

5. 补码加减交替法除法

例: 已知 x = - 0.1001 y = + 0.1101 用补码一位除法求 x / y

解: $[x]_{补}= 1.0111$ $[y]_{补}=0.1101$ $[-y]_{补}=1.0011$

	被除数/余数	商	说明
	11.0111		被除数与除数异号 被除数加除数
+ $[y]_{补}$	00.1101		
	00.0100		余数与除数同号,商上1 ,左移减除数
←	00.1000	1	
+ $[-y]_{补}$	11.0011		
	11.1011		余数与除数异号, 商上0, 左移, 加除数
←	11.0110	1.0	
+ $[y]_{补}$	00.1101		
	00.0011		余数与除数同号,商上1 ,左移 余数减除数

5. 补码加减交替法除法

例: 已知 x = - 0.1001　y = + 0.1101 用补码一位除法求 x / y

解: $[x]_{补}$ = 1.0111　$[y]_{补}$=0.1101　$[-y]_{补}$=1.0011

	被除数/余数	商	说明
	00.0011		余数与除数同号,商上1 ,左移 余数减除数
←	00.0110	1.01	
+ $[-y]_{补}$	11.0011		
	11.1001		余数与除数异号, 商上0 , 左移,加除数
←	11.0010	1.010	
+ $[y]_{补}$	00.1101		
	11.1111		余数与除数异号, 商上0, 移商
←		1.0100	

x /y = - 0.1100　余数= - 0.00000001

5. 补码加减交替法除法

4)商的校正

(1)商需要校正的原因

补码一位除法公式是在商的末位恒置“1”的条件下推导的,商为负数时得到的是反码

(2)商校正方法

♦能除尽时, 若除数 > 0,不校正; 除数 < 0 , 商加 2^{-n} 校正

♦不能除尽时， 若商 > 0 ,不校正 ;商 < 0 , 加 2^{-n} 校正

上例中,不能除尽,且 商< 0 ,故需要校正

校正后的商为 1.0100 + 0.0001 = 1.0101

即 $x / y = -0.1011$

5. 补码加减交替法除法

5)余数的校正

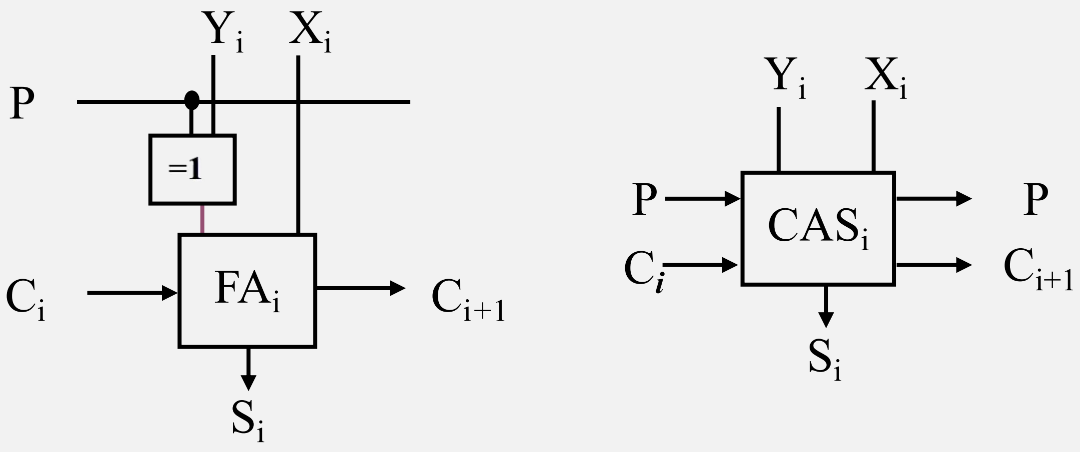
(1)余数需要校正的原因

不恢复余数法是先比较，后上商

(2)余数校正方法

- ♦若商 > 0, 余数与被除数异号时，余数加除数进行校正；
- ♦若商 < 0, 余数与被除数异号时，余数减除数进行校正。

6. 阵列除法器(电路)



(a) 电路

(b) 符号表示

可控制加/减法(CAS)单元

◆ P=0时(全加器)

$$S_i = X_i \oplus (P \oplus Y_i) \oplus C_i$$

$$C_{i+1} = X_i Y_i + (X_i \oplus (P \oplus Y_i))\, C_i$$

$$S_i = X_i \oplus Y_i \oplus C_i$$

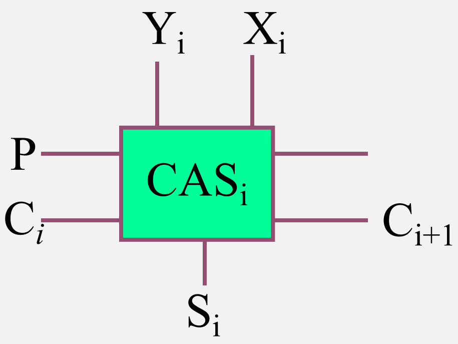
$$C_{i+1} = X_i Y_i + (X_i \oplus Y_i)\, C_i$$

◆ P=1时(全减 – 还要注意C_i)

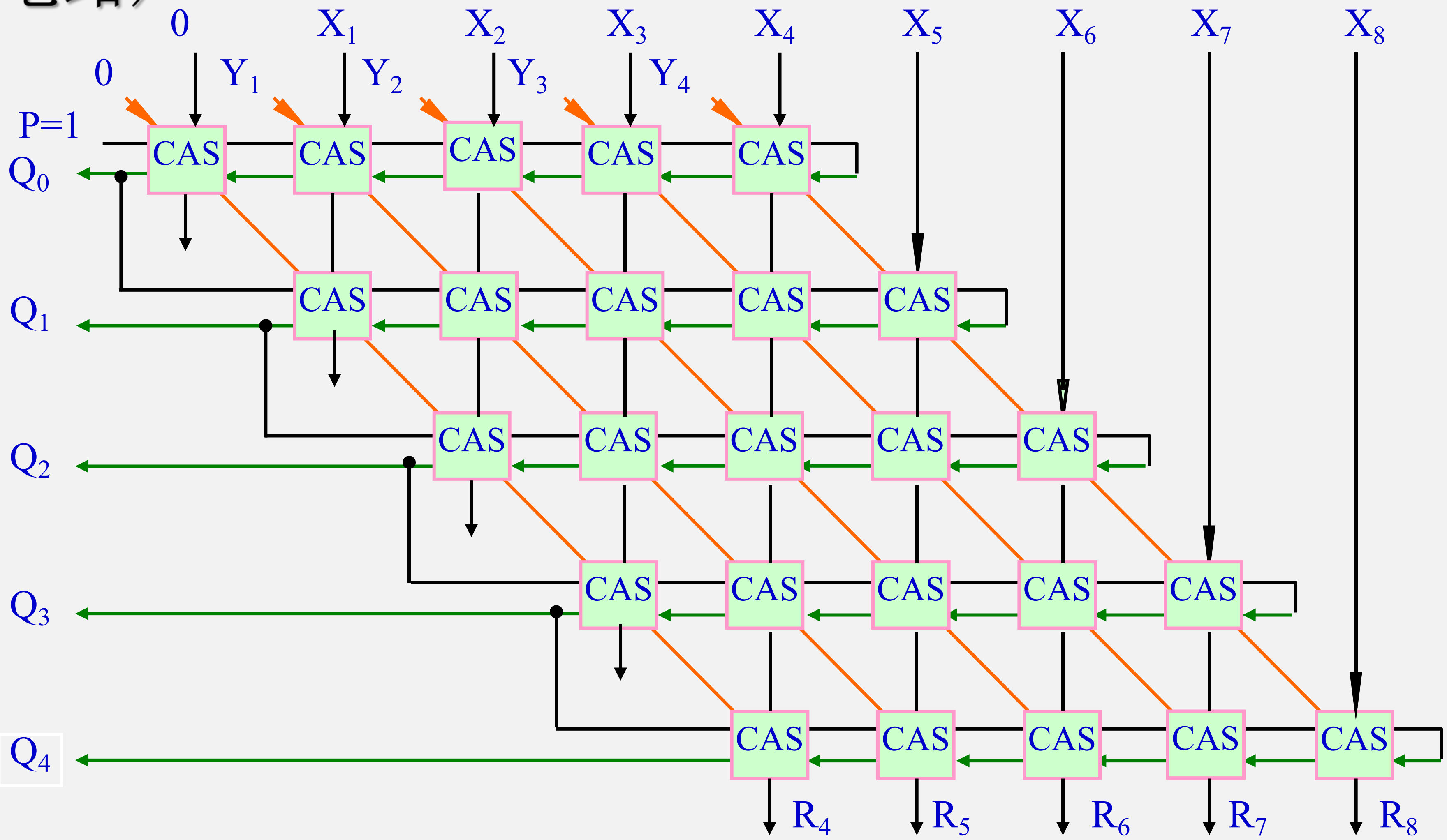
$$S_i = X_i \oplus \overline{Y}_i \oplus C_i$$

$$C_{i+1} = X_i Y_i + (X_i \oplus \overline{Y}_i)\, C_i$$

6. 阵列除法器(电路)



$C_f = Q_n$



华中科技大学 计算机科学与技术学院 School of Computer Science & Technology, HUST

6. 阵列除法器（电路）

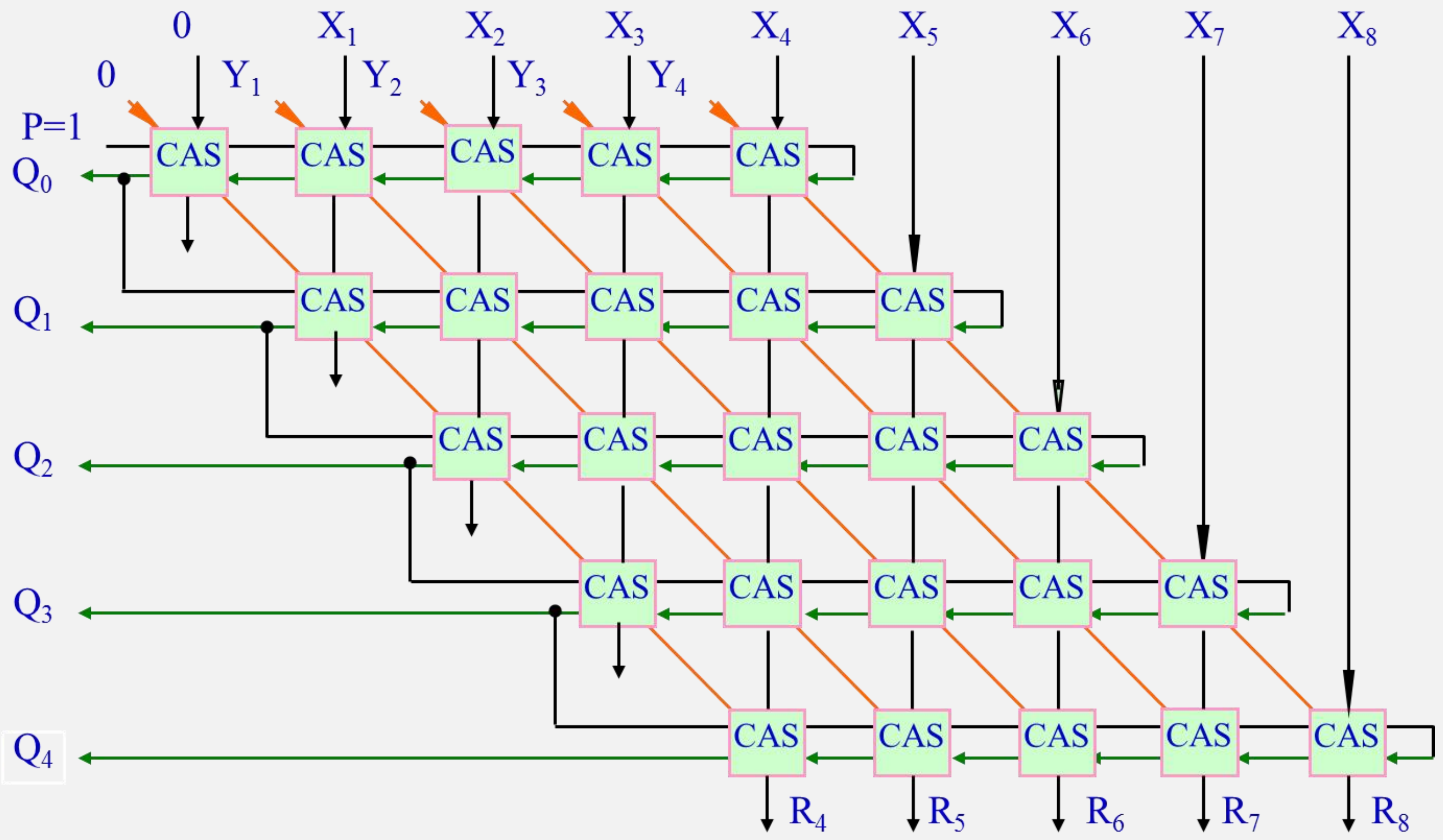
	被除数/余数	商	说明
	11.0111		被除数与除数异号 被除数加除数
+ $[y]_{补}$	00.1101		
	00.0100		余数与除数同号,商上1 ,左移减除数
←	00.1000	1	
+ $[-y]_{补}$	11.0011		
	11.1011		余数与除数异号, 商上0, 左移, 加除数
←	11.0110	1.0	
+ $[y]_{补}$	00.1101		
	00.0011		余数与除数同号,商上1 ,左移 余数减除数

6. 阵列除法器（电路）

♦n*n个CAS单元

♦(n*n) ×9T?

♦如何优化?



第三部分完