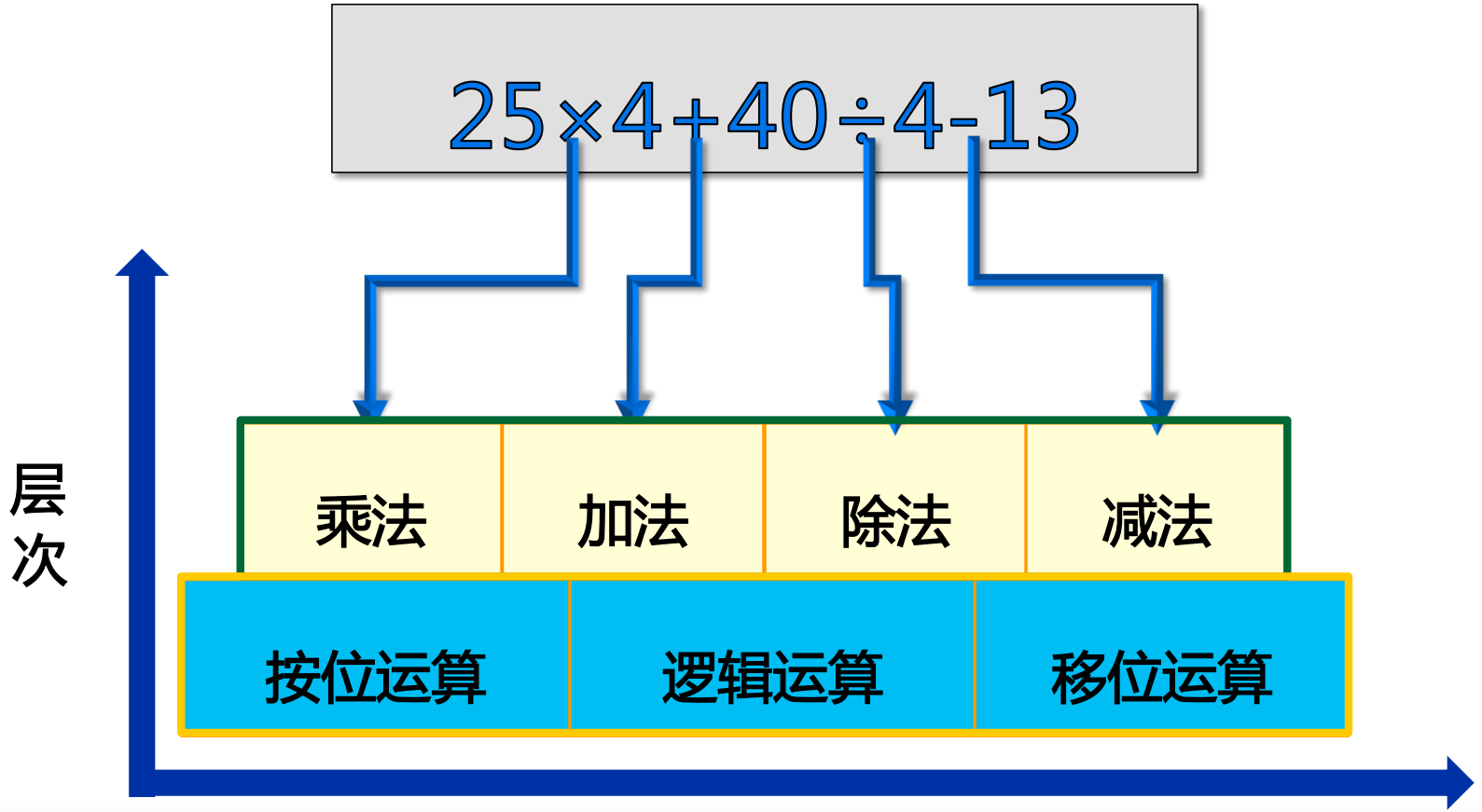
3.3 乘法运算

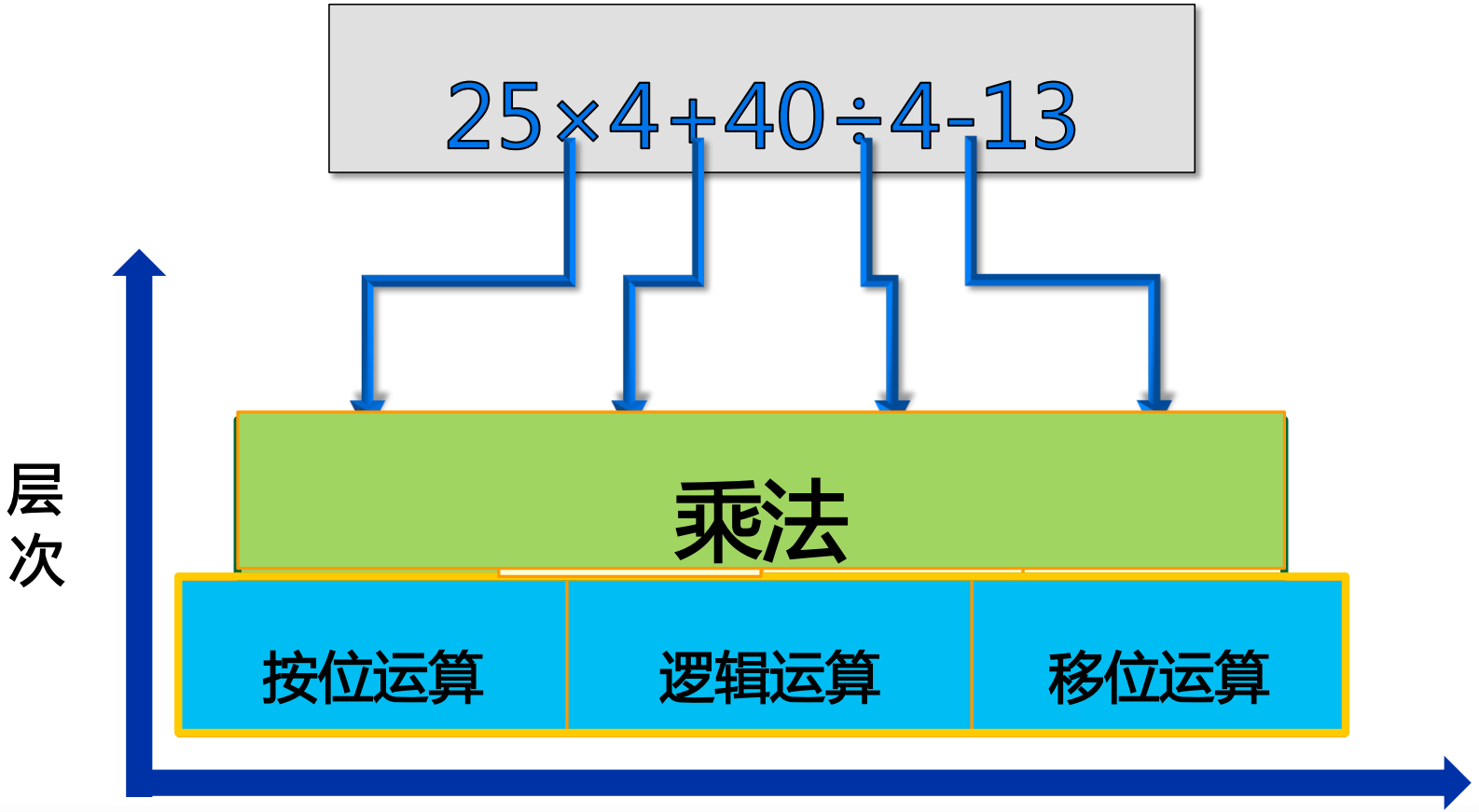
刘 芳 副教授

国防科学技术大学计算机学院

继续从上一个算式说起



继续从上一个算式说起



❑十进制乘法，例：

				2	4	5
	X			6	7	3
$M_0=AXB_3$				7	3	5
$M_1=AXB_2$		1	7	1	5	
$M_2=AXB_1$	1	4	7	0		
AXB	1	6	4	8	8	5

位积 $A \times B_i$

AXB = 164885

❑二进制乘法，例：

逻辑与

					1	1	0	1
			X		1	0	1	1
$M_0=A\&B_4$					1	1	0	1
$M_1=A\&B_3$				1	1	0	1	
$M_2=A\&B_2$			0	0	0	0		
$M_3=A\&B_1$		1	1	0	1			
AXB	1	0	0	0	1	1	1	1

位积 **A X B_i**

AXB = 10001111

3.3.1 二进制乘法

					1	1	0	1
			X		1	0	1	1
$M_0=A\&B_4$	0	0	0	0	1	1	0	1
$M_1=A\&B_3$	0	0	0	1	1	0	1	0
$M_2=A\&B_2$	0	0	0	0	0	0	0	0
$M_3=A\&B_1$	0	1	1	0	1	0	0	0
AXB	1	0	0	0	1	1	1	1

手工运算过程

位积 $A \times B_i$

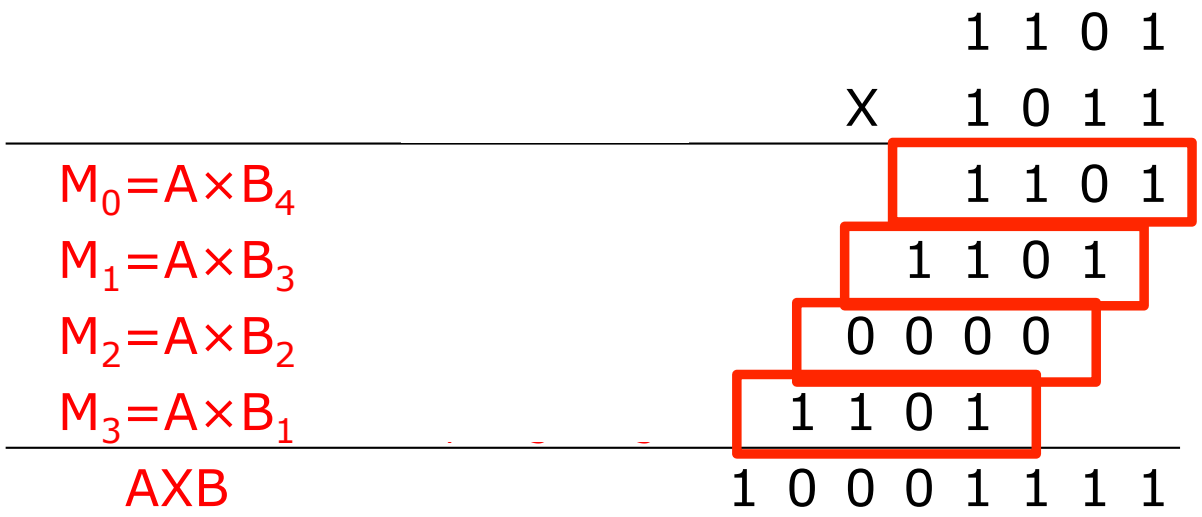
AXB = 10001111

计算机中怎么实现？

问题：1.加法器只有两个输入端，无法支持多路输入！

2.需要2n+1位加法器，不能有效利用全加器操作！

二进制乘法



改进方案1： 引入部分积P，改一次求和为累加求和

二进制乘法

能否等同于部分积右移1位？

只需要n+1位加法器

P_0

$M_0=A\times B_4$　　$P_1=P_0+M_0$

$M_1=A\times B_3$　　$P_2=P_1+M_1$

$M_2=A\times B_2$　　$P_3=P_2+M_2$

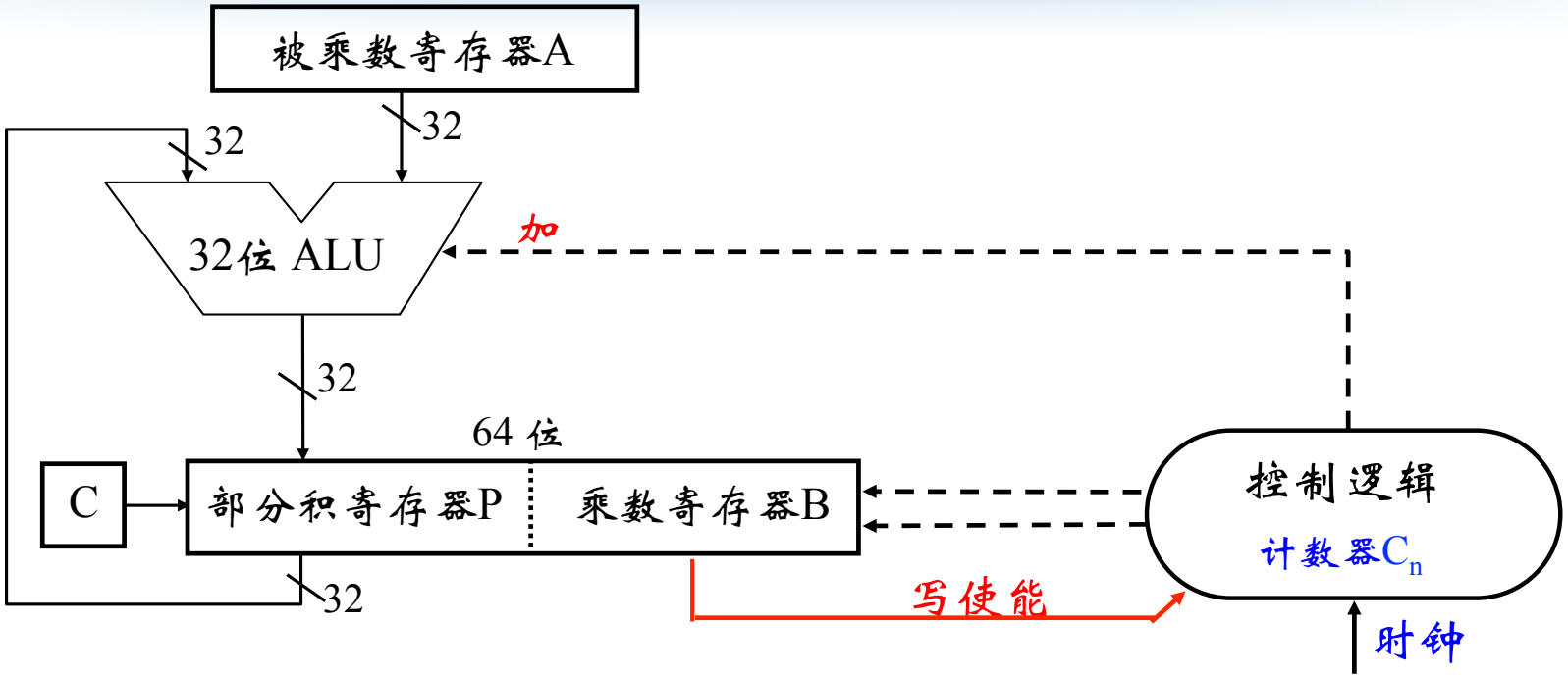
$M_3=A\times B_1$　　$P_4=P_3+M_3$

AXB

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

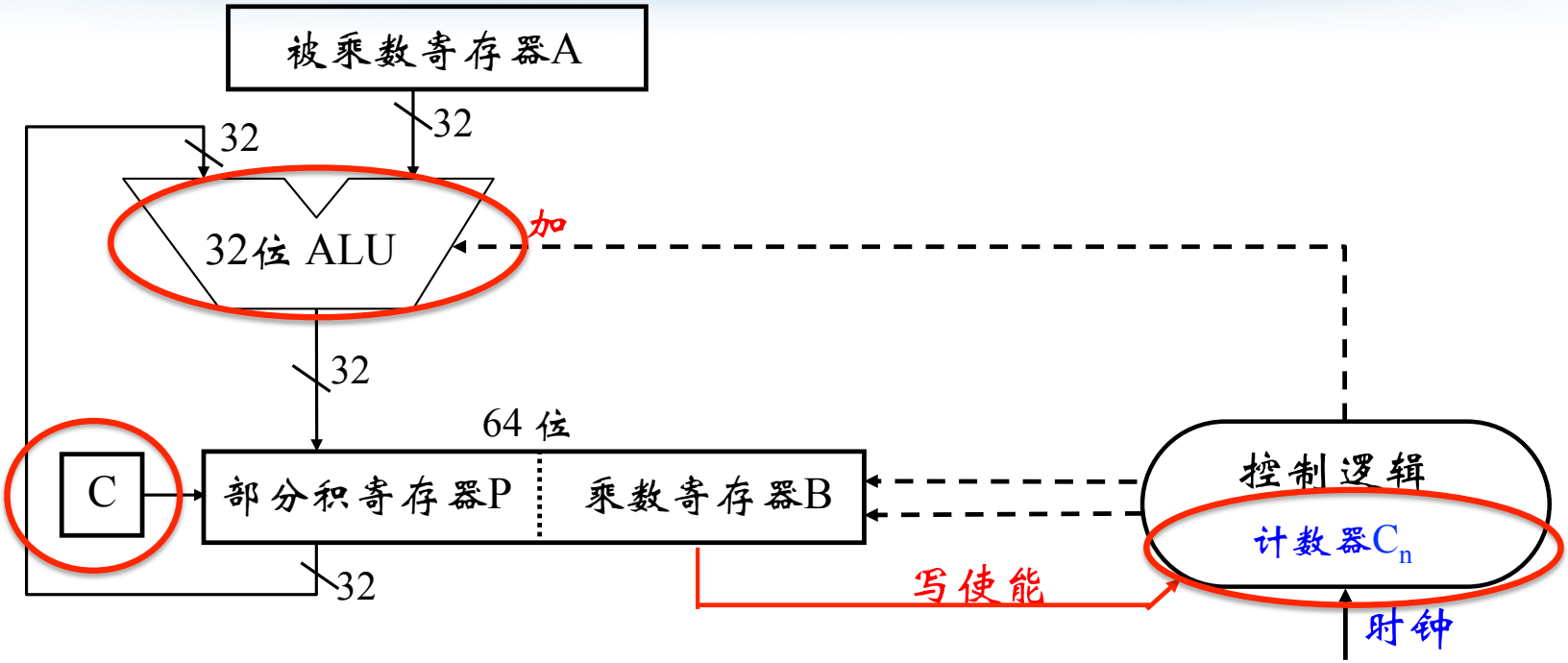
改进方案2：将部分积右移一位再求和，移出部分保存

二进制乘法



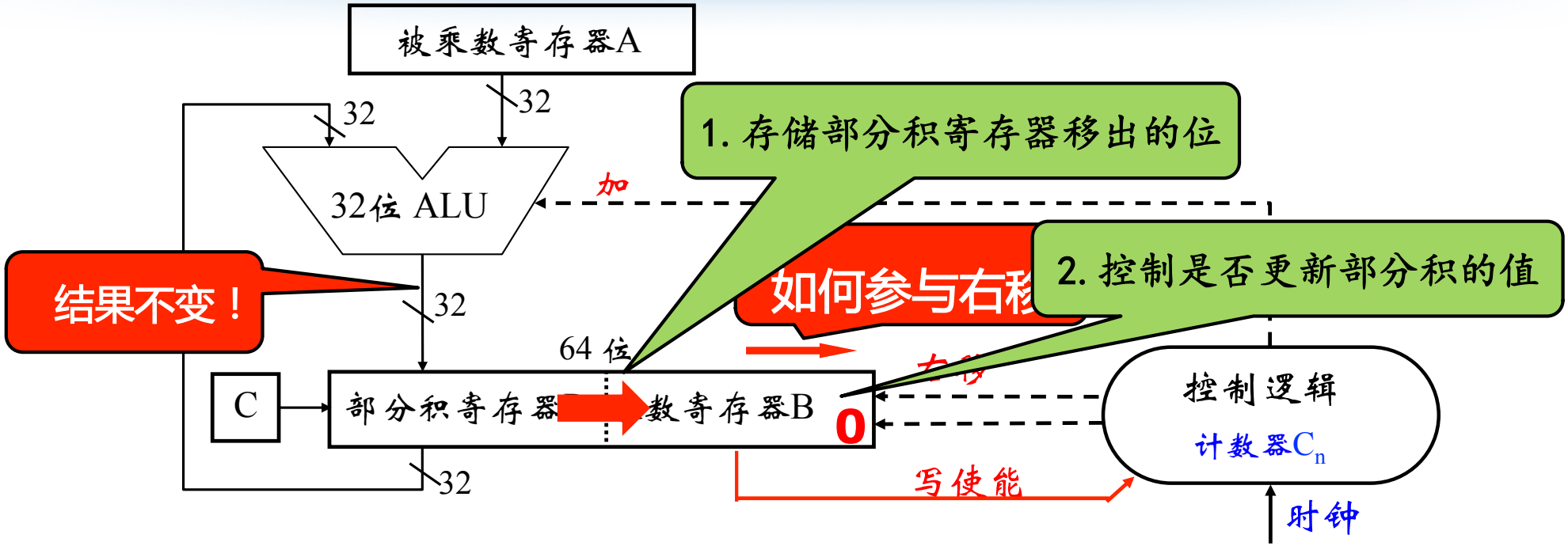
- 被乘数寄存器X：存放被乘数
- 部分积寄存器P：初始置$P_0=0$；结束时存放64位乘积的高32位
- 乘数寄存器B：初始置乘数；结束时存放64位乘积的低32位

二进制乘法



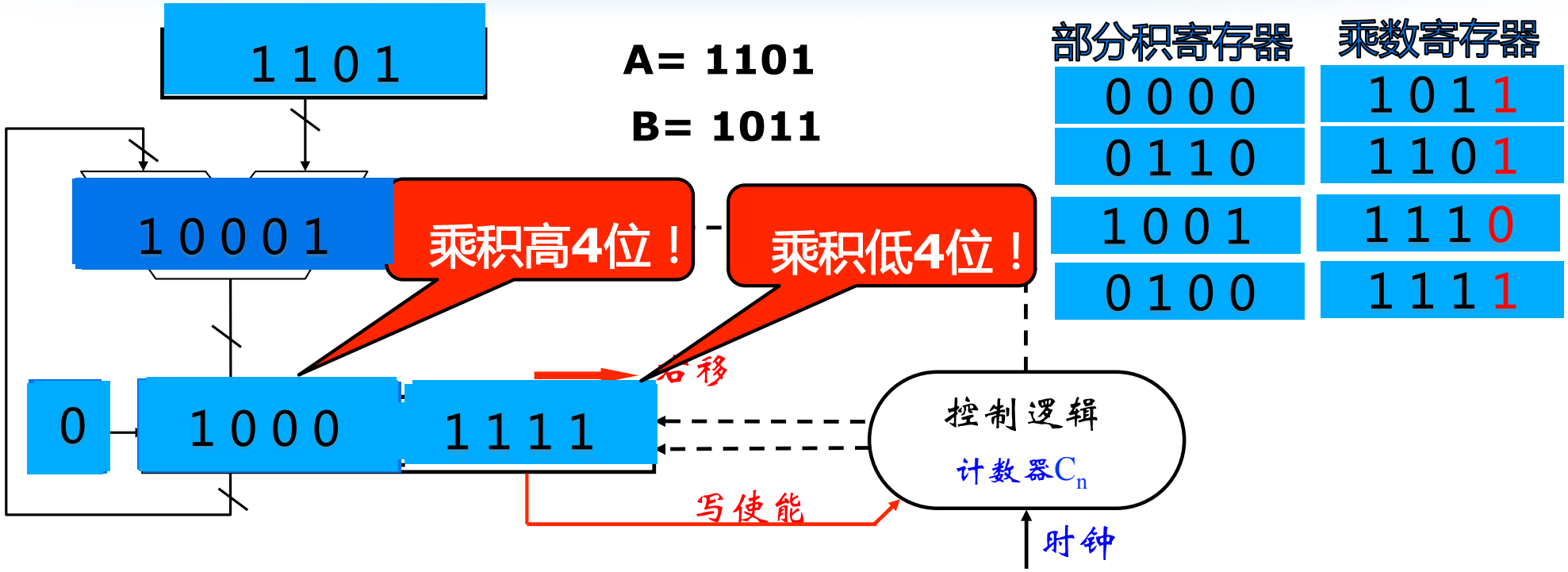
- 进位触发器C：保存加法器的进位信号
- 循环计数器C_n：循环次数(初值31)，每循环一次C_n-1，$C_n=0$结束
- ALU：乘法核心部件。在“加”逻辑控制下对P和A的内容进行运算，在“写使能”逻辑控制下运算结果被送回P，进位存放在C中

二进制乘法



每次循环都要对进位C、部分积寄存器P和乘数寄存器B
实现同步“右移”

二进制乘法



部分积寄存器	乘数寄存器
0000	1011
0110	1101
1001	1110
0100	1111

- 进位触发器C：保存加法器的进位信号
- 循环计数器C_n：循环次数(初值31)，每循环一次C_n-1，$C_n=0$结束
- ALU：乘法核心部件。在“加”逻辑控制下对P和B的内容进行运算，在“写使能”逻辑控制下运算结果被送回P，进位存放在C中

❑二进制乘法，例：

无符号数乘法

					1	1	0	1
			X		1	0	1	1
$M_1=A\&B_4$					1	1	0	1
$M_2=A\&B_3$				1	1	0	1	
$M_3=A\&B_2$			0	0	0	0		
$M_3=A\&B_1$		1	1	0	1			
AXB	1	0	0	0	1	1	1	1

位积 $A \times B_i$

AXB = 10001111

有符号怎么办？

原码一位乘法：符号位异或，数值部分绝对值相乘

原码一位乘法

定点小数

原码一位乘法举例

A= 0.1101，B= 0.1011

被乘数　　乘数

符号s	阶码e(整数)	尾数f(小数)

A*B=C=(0⊕0).10001111

部分积寄存器		乘数寄存器				
0．0000		1	0	1	1	
+) 0．1101						
0．1101						
0．0110		1	1	0	1	1（丢失）
+) 0．1101						
1．0011						
0．1001		1	1	1	0	1（丢失）
0．0100		1	1	1	1	0（丢失）
+) 0．1101						
1．0001						
0．1000		1	1	1	1	1（丢失）

原码一位乘法

机器运算算法

将A×B改写

$=A\times(0.1011)$

$=0.1A+0.00A+0.001A+0.0001A$

$=P_4$

分析：机器运算算法的特点

- 乘法的数值部分由多次累加和移位完成
- n次n+1位加法（此例中n=4）
- n次右移操作

$$P_0 \equiv 0$$

$$P_1=2^{-1}(B_1|A|+P_0)$$

$$P_2=2^{-1}(B_2|A|+P_1)$$

$$P_3=2^{-1}(B_3|A|+P_2)$$

$$P_4=2^{-1}(B_4|A|+P_3)$$

$$C=(A_0 \oplus B_0)\cdot P_4$$

原码一位乘法

❑原码一位乘法递推公式

$P_0 \equiv 0$

$P_1 = 2^{-1}(P_0 + B_n|A|)$

右移操作

$P_2 = 2^{-1}(P_1 + B_{n-1}|A|)$

…

$P_i = 2^{-1}(P_{i-1} + B_{n-i+1}|A|)$

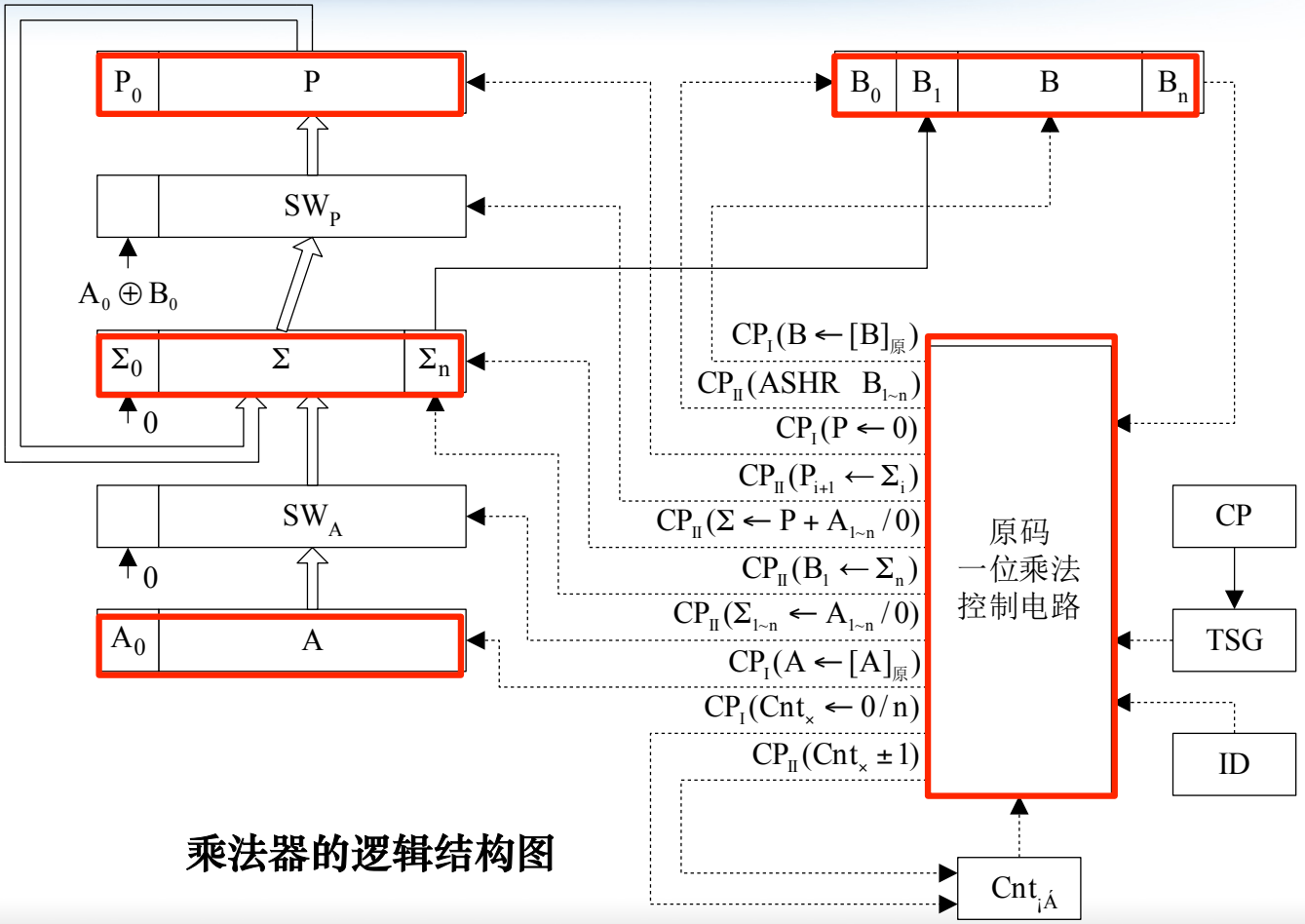
…

$P_n = 2^{-1}(P_{n-1} + B_1|A|) = |A| \times |B|$

$C = (A_0 \oplus B_0) \cdot P_n$

原码一位乘法

- 原码一位乘法器的组成
  - 被乘数寄存器
  - 乘数寄存器
  - 乘积寄存器
  - 加法器
  - 输入选择开关
  - 控制电路



乘法器的逻辑结构图

原码一位乘法

❑ 原码一位乘法过程

■ 初始化

$A \leftarrow A$，$B \leftarrow B$，$P \leftarrow 0$

$Cnt_{\times} \leftarrow n$，$T_{\times} \leftarrow 1$

■ 乘法：

$\Sigma \leftarrow 0.P_{1\sim n} + B_n \times 0.A_{1\sim n}$；

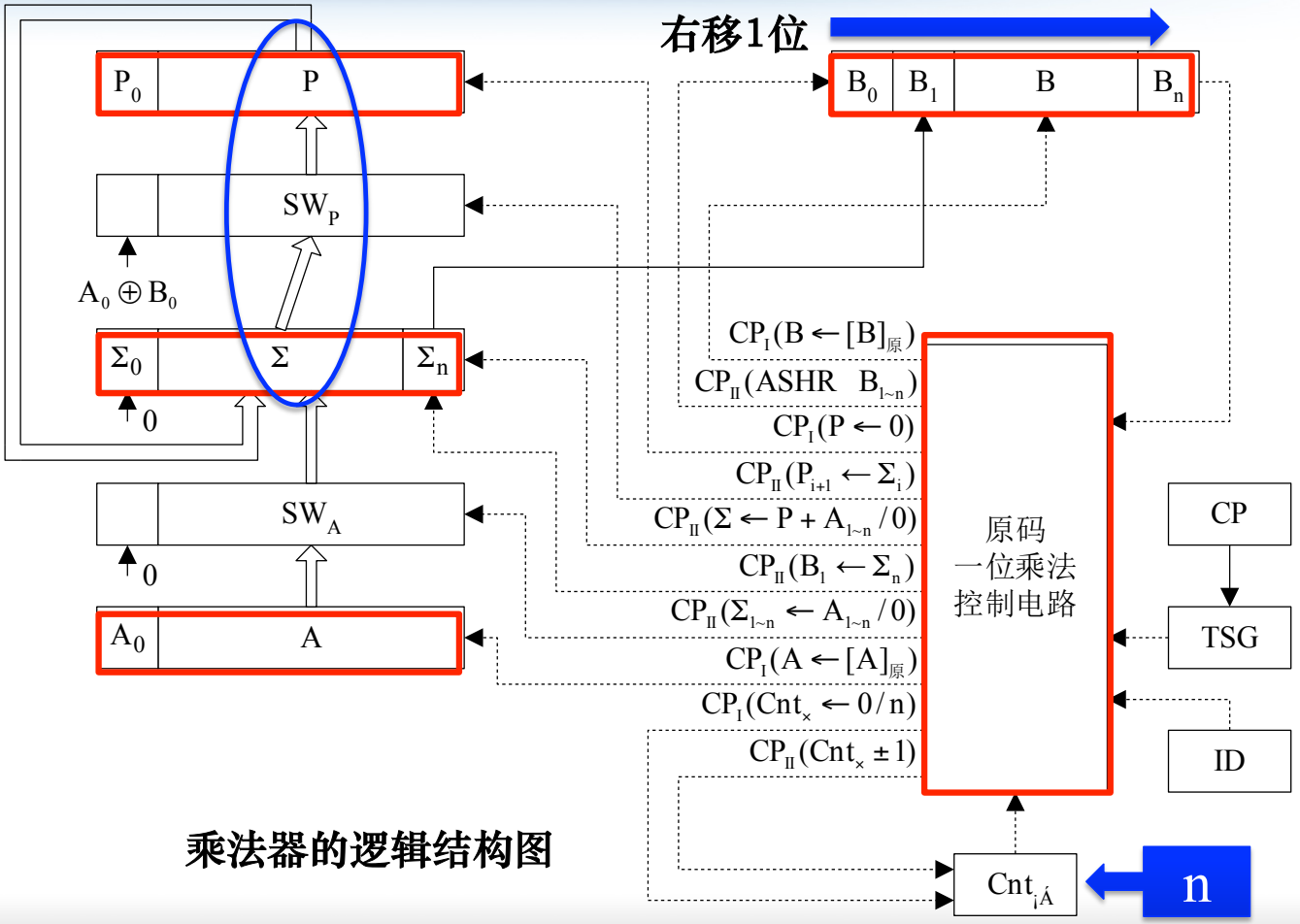
$P_{1\sim n} \leftarrow \Sigma_{0\sim n-1}$；

ASHR $B_{1\sim n}$，$B_1 \leftarrow \Sigma_n$；

$Cnt_{\times}$-1

■ 完成：

$Cnt_{\times}=0$，$T_{\times} \leftarrow 0$



乘法器的逻辑结构图

3.3.3 补码一位乘法

- 在原码一位乘法的基础上发展
- 比较法，由英国Booth夫妇首先提出，故又称为Booth乘法

例：C=A * B

被乘数 $[A]_{补}=A_{01}\ A_{02}.A_1A_2\cdots A_n$

乘数 $[B]_{补}=B_0.B_1B_2\cdots B_n$

乘积 $[C]_{补}=C_0.C_1C_2\cdots C_nC_{n+1}C_{n+2}\cdots C_{2n}$

乘数和被乘数的符号位参与运算

被乘数A采用模4补码(变形补码)，根据定义有：

$[A]_{补}=2^2+A\ (\mathrm{mod}\ 4)$ 且 $2^2+A=2^{n+2}+A\ (\mathrm{mod}\ 4)$

如果A>0，则$A_{01}\ A_{02}=00$；如果A<0，则$A_{01}\ A_{02}=11$

补码一位乘法

分两种情况讨论：

- 乘数非负（B≥0）
- 乘数为负（B<0）

补码一位乘法

- A符号任意，B为非负数
  - 当被乘数A ≥0时：$[A]_{补}=[A]_{原}$，$[B]_{补}=[B]_{原}$

    乘法过程与原码乘法相同，但符号位参加运算
  - 当被乘数A<0时：采用模4补码(变形补码)

    根据定义有：　$[A]_{补}=2^2+A=2^{n+2}+A \pmod 4$

    由于B≥0，$B_0=0$，故$[B]_{补}=B=0.B_1B_2\ldots B_n=\Sigma B_i2^{-i}$

    $[A]_{补}\cdot[B]_{补}=[A]_{补}\cdot B=(2^{n+2}+A)\cdot B$

    $=2^{n+2}\cdot\Sigma B_i2^{-i}+AB$

    $=2^2\cdot\Sigma B_i2^{n-i}+AB$　　(ΣB_i2^{n-i}为非负整数)

    $=2^2+A\cdot B \pmod 4$　　$=[A\cdot B]_{补}$

补码一位乘法

1、符号位按照补码规则参加运算
2、移位时按照补码规则移位

❑A符号任意，B为非负数

$[A\cdot B]_{原}=[A]_{原}\cdot[B]_{原}$	$[A\cdot B]_{补}=[A]_{补}\cdot[B]_{补}$
$P_0\equiv 0$	$[P_0]_{补}\equiv 0$
$P_1=2^{-1}(P_0+B_n\lvert A\rvert)$	$[P_1]_{补}=2^{-1}([P_0]_{补}+B_n\cdot[A]_{补})$
$P_2=2^{-1}(P_1+B_{n-1}\lvert A\rvert)$	$[P_2]_{补}=2^{-1}([P_1]_{补}+B_{n-1}\cdot[A]_{补})$
…	…
$P_i=2^{-1}(P_{i-1}+B_{n-i+1}\lvert A\rvert)$	$[P_i]_{补}=2^{-1}([P_{i-1}]_{补}+B_{n-i+1}\cdot[A]_{补})$
…	…
$P_n=2^{-1}(P_{n-1}+B_1\lvert A\rvert)=\lvert A\rvert\times\lvert B\rvert$	$[P_n]_{补}=2^{-1}([P_{n-1}]_{补}+B_1\cdot[A]_{补})$

补码一位乘法

分两种情况讨论：

- 乘数非负（B≥0）
- 乘数为负（B<0）

❑A符号任意，B为负数

$$[B]_{补}=B+2 \quad (\mathrm{mod}\ 2)$$

$$B=[B]_{补}-2=1.B_1B_2\ldots B_n-2=0.B_1B_2\ldots B_n-1$$

$$[A\cdot B]_{补}=[A\cdot(0.B_1B_2\ldots B_n)-A]_{补}$$

$$=[A\cdot(0.B_1B_2\ldots B_n)]_{补}+[-A]_{补}$$

$$=[A]_{补}\cdot(0.B_1B_2\ldots B_n)+[-A]_{补}$$

$$=[A]_{补}\cdot([B]_{补})_{尾}+[-A]_{补}$$

分两种情况讨论：

B为负数：$[A\cdot B]_{补}=[A]_{补}\cdot([B]_{补})_{尾}+[-A]_{补}$

B为非负数： $[A\cdot B]_{补}=[A]_{补}\cdot[B]_{补}$

两种情况可以综合吗？

$[A\cdot B]_{补}=[C]_{补}=[A]_{补}\cdot(0.B_1B_2\ldots B_n)+[-A]_{补}\cdot B_0$

B_i为B的补码的第i位的值

补码一位乘法

$$[A\cdot B]_{补}=[C]_{补}=[A]_{补}(0\cdot B_1B_2\ldots B_n)+[-A]_{补}\cdot B_0$$

$$[C]_{补}=[A]_{补}\cdot(B_1\cdot 2^{-1}+B_2\cdot 2^{-2}+\ldots B_n\cdot 2^{-n})-[A]_{补}\cdot B_0$$

$$=[A]_{补}\cdot[-B_0+(B_1-B_1\cdot 2^{-1})+(B_2\cdot 2^{-1}-B_2\cdot 2^{-2})+\ldots+(B_n\cdot 2^{-(n-1)}-B_n\cdot 2^{-n})]$$

$$=[A]_{补}\cdot[(B_1-B_0)\cdot 2^{0}+(B_2-B_1)\cdot 2^{-1}+\ldots+(B_{n+1}-B_n)\cdot 2^{-n}]$$

式中，$B_{n+1}\equiv 0$，将上式改写成

$$=[A]_{补}(B_1-B_0)+2^{-1}\{[A]_{补}(B_2-B_1)+2^{-1}\{[A]_{补}(B_3-B_2)+2^{-1}[$$

$$\ldots+2^{-1}\{[A]_{补}(B_{n+1}-B_n)+0\}\ \ldots\}\}$$

补码一位乘法

$$[C]_{补}=[A]_{补}(B_1-B_0)+2^{-1}\{[A]_{补}(B_2-B_1)+2^{-1}\{[A]_{补}(B_3-B_2)+2^{-1}[\cdots+2^{-1}\{[A]_{补}(B_{n+1}-B_n)+0\}\cdots\}\}$$

P_n

$[P_0]_{补}\equiv 0$；

$[P_1]_{补}=2^{-1}\{[P_0]_{补}+(B_{n+1}-B_n)[A]_{补}\}$；

$[P_2]_{补}=2^{-1}[\{[P_1]_{补}+(B_n-B_{n-1})[A]_{补}\}$；

…

$[P_i]_{补}=2^{-1}\{[P_{i-1}]_{补}+(B_{n-i+2}-B_{n-i+1})[A]_{补}\}$；

当前乘数寄存器的最后两位

…

$[P_n]_{补}=2^{-1}\{[P_{n-1}]_{补}+(B_2-B_1)[A]_{补}\}$

所以，$[C]_{补}=[P_{n+1}]_{补}=[P_n]_{补}+(B_1-B_0)[A]_{补}$

补码一位乘法

❑Booth乘法

比较法

判断位 (B_n,B_{n+1})	新部分积 $[P_{i+1}]_{补}=$	操作	说明
0 0	$2^{-1}[P_i]_{补}$	→1	右移一位
0 1	$2^{-1}\{[P_i]_{补}+[A]_{补}\}$	+,→1	加$[A]_{补}$后再右移一位
1 0	$2^{-1}\{[P_i]_{补}+[-A]_{补}\}$	-,→1	加$[-A]_{补}$后再右移一位
1 1	$2^{-1}[P_i]_{补}$	→1	右移一位

Booth乘法运算规则

- 被乘数A和部分积P均取两位符号位即变形码，乘数取一位符号位，符号位参与运算
- 乘数末尾增设附加位B_{n+1}，其初始值为0
- B_n和B_{n+1}构成各步运算的乘数判断位
- 按补码移位规则：部分积为正(第1符号位为0)，右移时有效位最高位补0；部分积为负，右移时有效位最高位补1
- 按Booth乘法表算法进行到第n+1步，但第n+1步的部分积不再移位

补码一位乘法

❑例1

被乘数：A= -0.011

乘数：B= 0.101

$[A]_{补} = 11.101$

$[-A]_{补} = 00.011$

$[B]_{补} = 0.101$

$[P_0]_{补} = 00.000$

$$[P_{n+1}]_{补} = [P_n]_{补} + (B_1 - B_0)[A]_{补}$$

		部分积	乘数	B_{n+1}
$[P_0]_{补}$		00.000	0.101	10
	$+[-A]_{补}$	00.011		
		00.011		
$[P_1]_{补}$	→1	00.001	1.010	01
	$+[A]_{补}$	11.101		
		11.110		
$[P_2]_{补}$	→1	11.111	0.101	10
	$+[-A]_{补}$	00.011		
		00.010		
$[P_3]_{补}$	→1	00.001	0.010	1
	$+[A]_{补}$	11.101		
		11.110		
$[P_4]_{补}$	→1	0	0.001	0

补码一位乘法

❑例1

被乘数：A= -0.011

乘数：B= 0.101

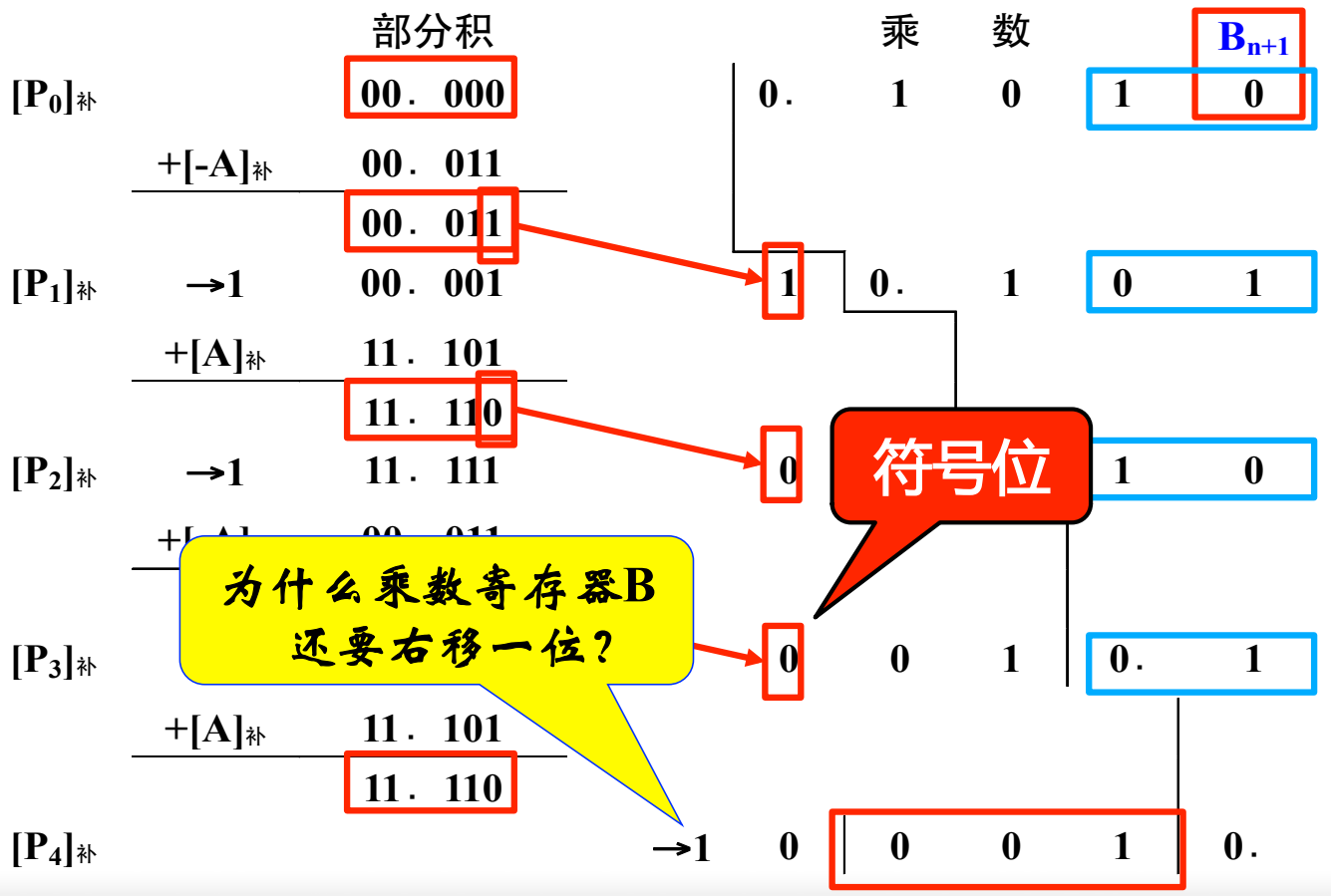
$[A]_{补}$ = 11.101

$[-A]_{补}$ = 00.011

$[B]_{补}$ = 0.101

$[P_0]_{补}$ = 00.000

$[A \times B]_{补}$=1.110001



补码一位乘法

❑例2

被乘数：A= -1

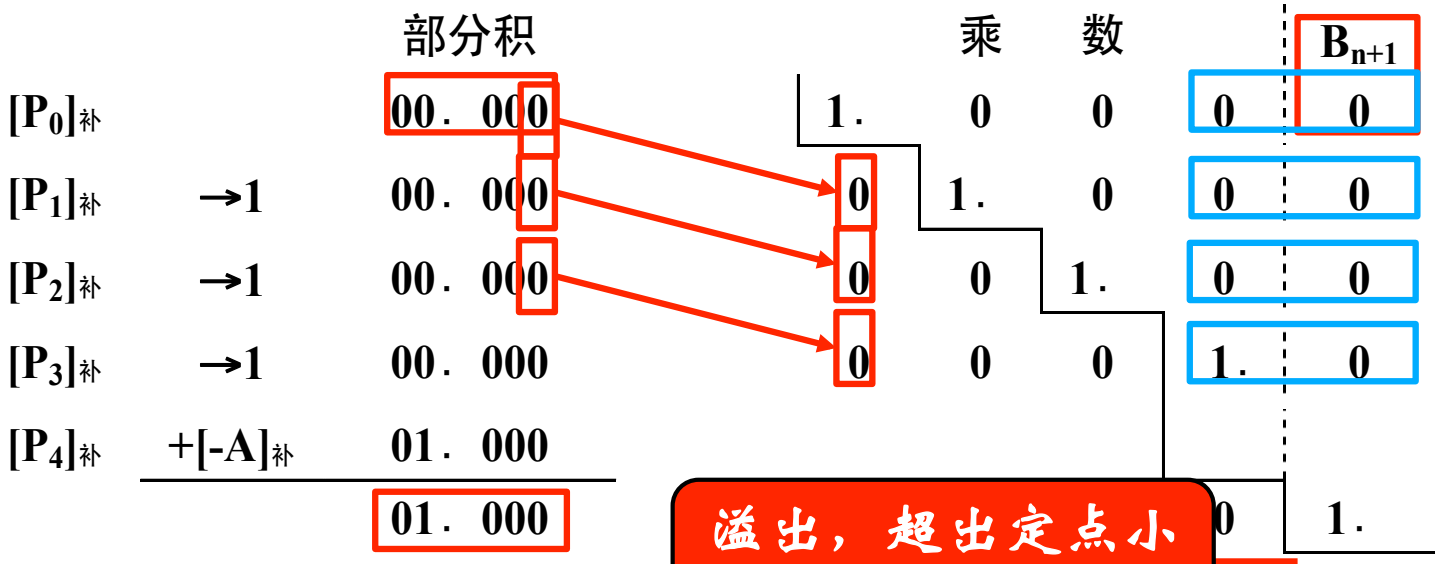
乘数：B= -1

$[A]_{补}$ =11.000

$[-A]_{补}$ =01.000

$[B]_{补}$ =1.000

$[P_0]_{补}$ = 00.000



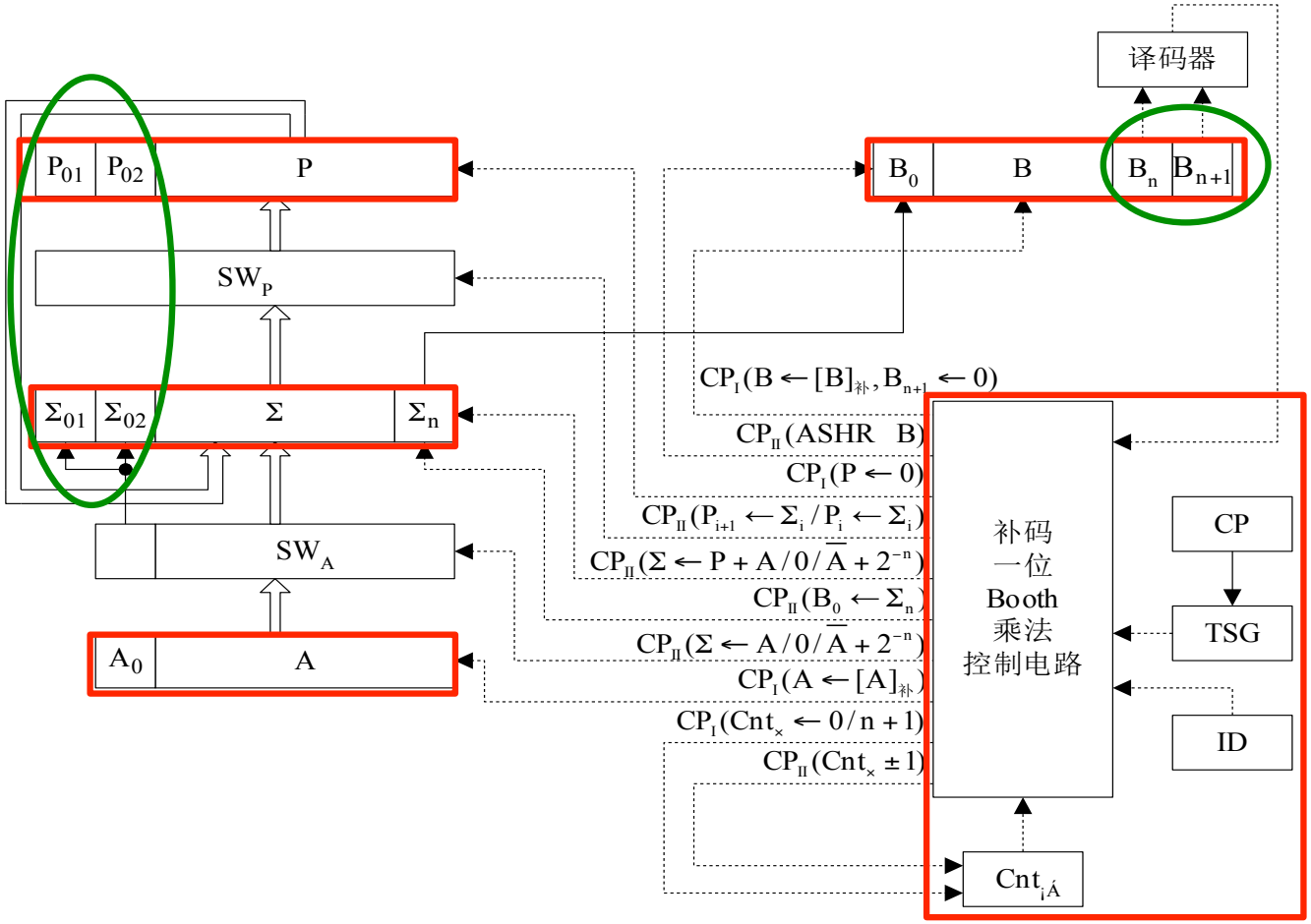
溢出，超出定点小数表示范围

$[A\times B]_{补}=[C]_{补}=01.000000=1$

定点小数补码乘法中唯一溢出情况！

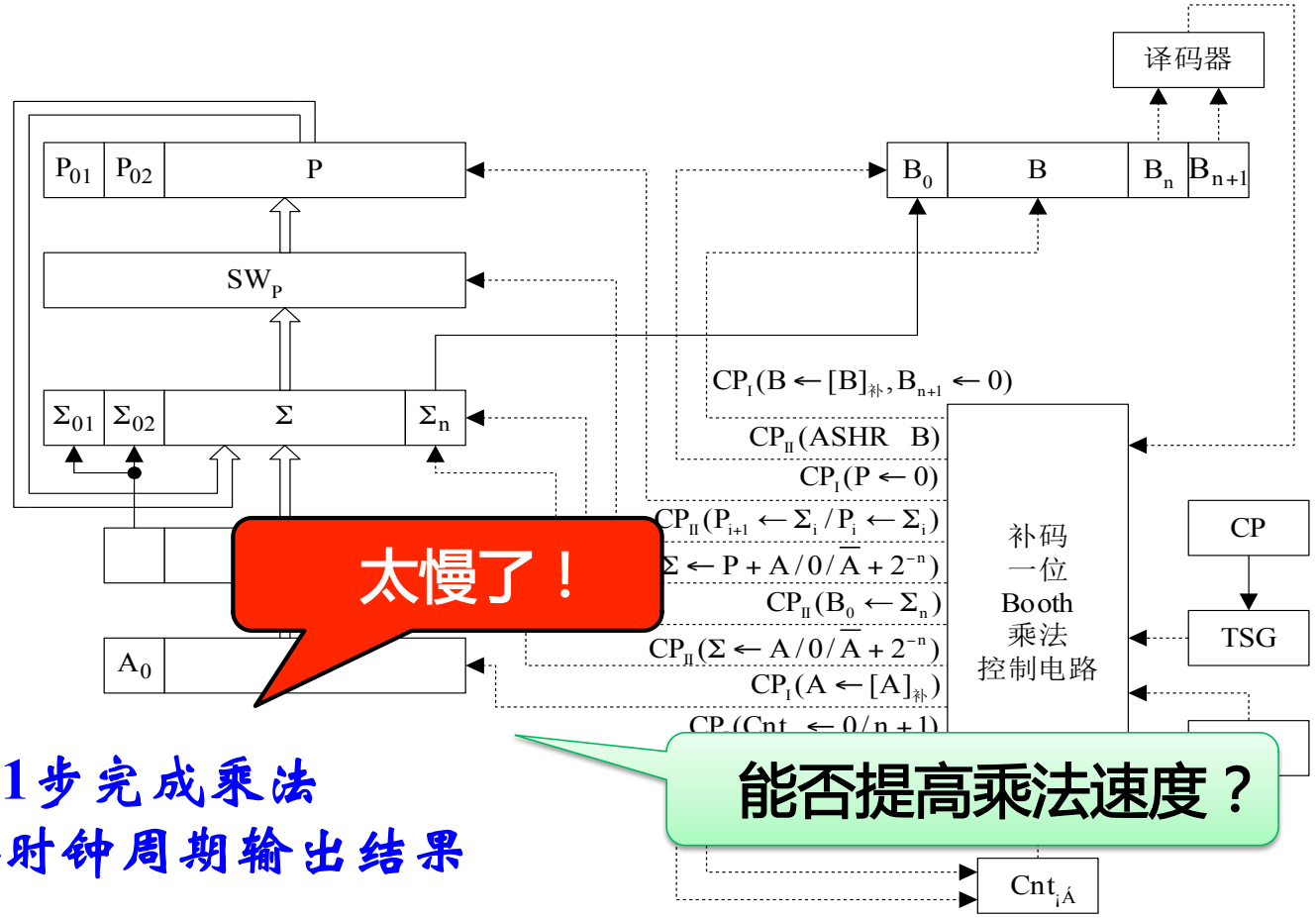
补码一位乘法器

- 补码一位乘法器的组成
  - 被乘数寄存器
  - 乘数寄存器
  - 乘积寄存器
  - 加法器
  - 输入选择开关
  - 控制电路



补码一位乘法器

- 补码一位乘法器的组成
  - 被乘数寄存器
  - 乘数寄存器
  - 乘积寄存器
  - 加法器
  - 输入选择开关
  - 控制电路



n+1步完成乘法

第n+2个时钟周期输出结果

3.3.4 快速乘法

❑提高乘法速度的方法

相乘

					1	1	0	1
			X		1	0	1	1
$M_1=AXB_4$					1	1	0	1
$M_2=AXB_3$				1	1	0	1	
$M_3=AXB_2$			0	0	0	0		
$M_3=AXB_1$		1	1	0	1			
AXB	1	0	0	0	1	1	1	1

相加

加速乘法的执行

每步处理多位乘数，实现快速乘法

例如：Booth两位乘法

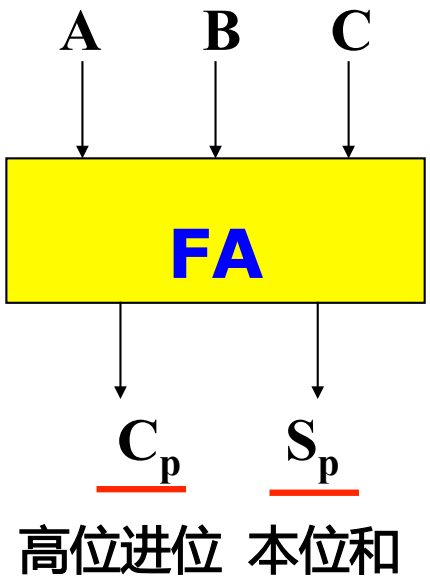
加速加法的执行

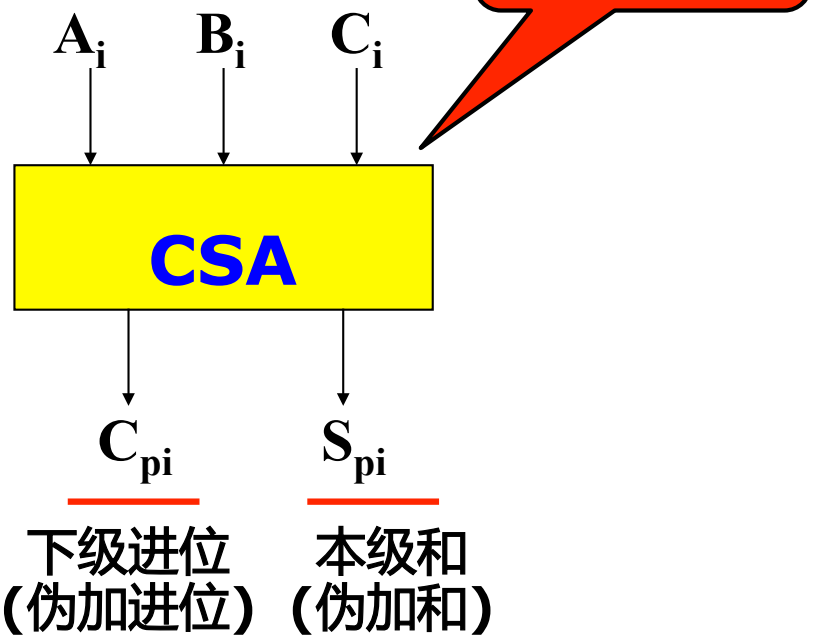
一步完成多个位积的相加，实现快速乘法

例如：使用伪加器构成柱形乘法器、利用实现多位乘的专用芯片构成阵列乘法器

❑加速加法的执行——柱形乘法器

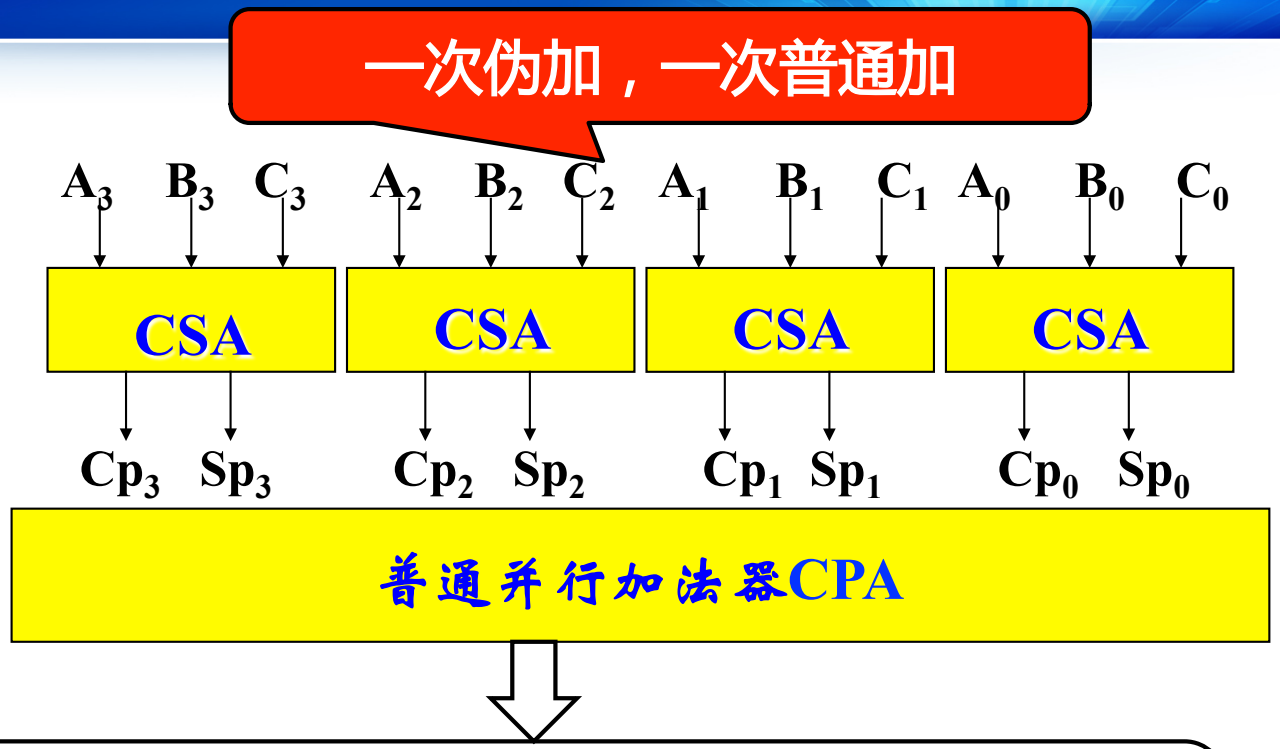
■一位全加器和存储进位加法器（伪加器）





快速乘法

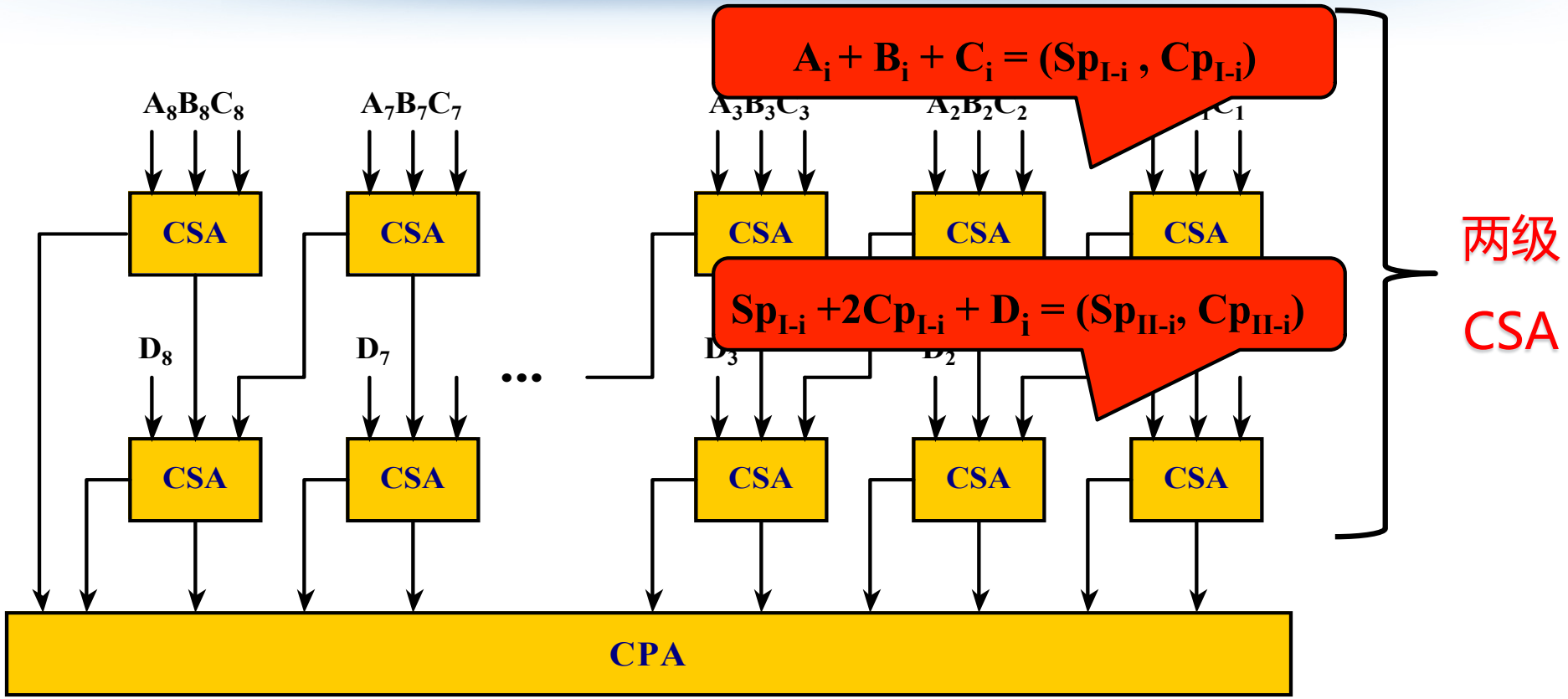
$$
\begin{array}{rl}
 & A_3A_2A_1A_0 \\
 & B_3B_2B_1B_0 \\
+) & C_3C_2C_1C_0 \\
\hline
 & S_3S_2S_1S_0
\end{array}
$$



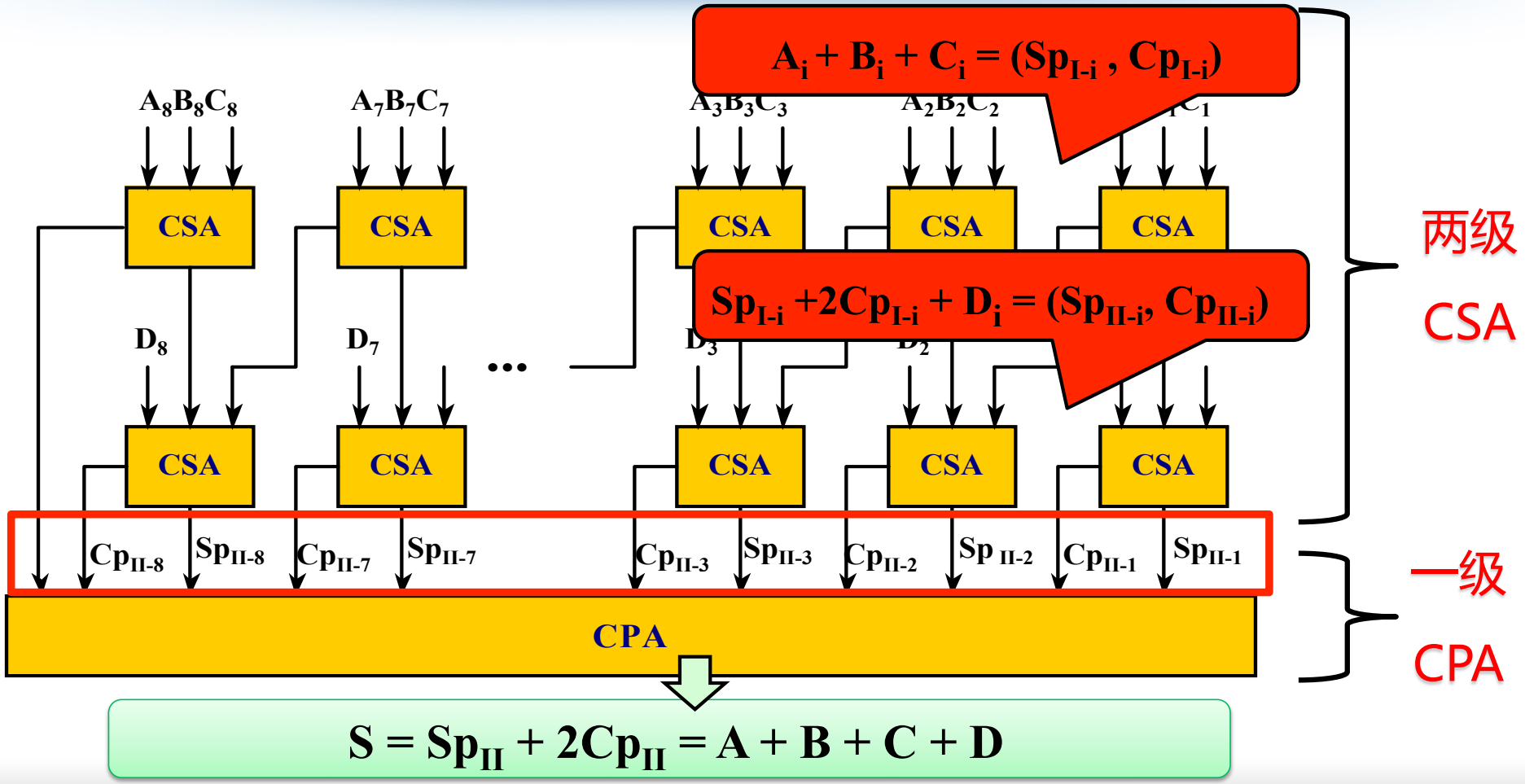
三个位积的和需将伪加和Sp_i与左移一位的伪加进位Cp_i相加求得：

$$S = Sp_3\,Sp_2Sp_1Sp_0 + 2 * Cp_3Cp_2Cp_1Cp_0$$

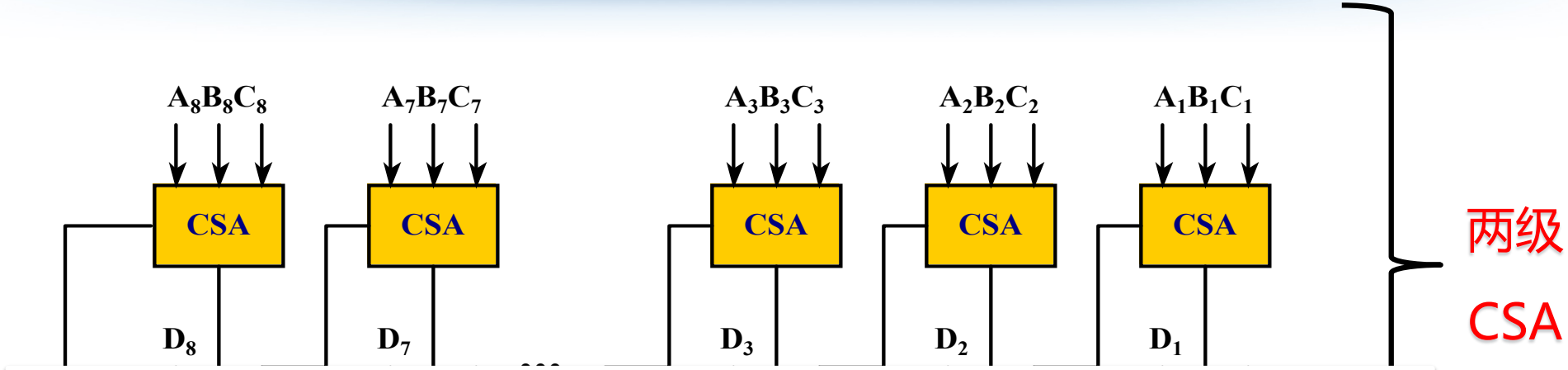
快速乘法


$A_i + B_i + C_i = (Sp_{I-i}, Cp_{I-i})$
$Sp_{I-i} + 2Cp_{I-i} + D_i = (Sp_{II-i}, Cp_{II-i})$
$A_8B_8C_8$
$A_7B_7C_7$
$A_3B_3C_3$
$A_2B_2C_2$
C_1
CSA
D_8
D_7
D_3
D_2
CPA
两级
CSA

快速乘法


$A_i + B_i + C_i = (Sp_{I-i}, Cp_{I-i})$
$Sp_{I-i} + 2Cp_{I-i} + D_i = (Sp_{II-i}, Cp_{II-i})$
$A_8B_8C_8$
$A_7B_7C_7$
$A_3B_3C_3$
$A_2B_2C_2$
C_1
CSA
D_8
D_7
D_3
D_2
Cp_{II-8} Sp_{II-8} Cp_{II-7} Sp_{II-7} Cp_{II-3} Sp_{II-3} Cp_{II-2} Sp_{II-2} Cp_{II-1} Sp_{II-1}
CPA
两级
CSA
一级
CPA
$S = Sp_{II} + 2Cp_{II} = A + B + C + D$

快速乘法



两级CSA

m个操作数加法网络：

m-2级存储进位加法器CSA的多级连接+1级CPA

一级CPA

加速加法的执行——阵列乘法器

✧利用若干全加器，完全由硬件直接计算乘法结果

加速加法的执行——阵列乘法器

手算过程

✧利用若干全加器，完全由硬件直接计算乘法结果

✧手算乘法过程(以4位无符号数为例)

$$
\begin{array}{cccccccc}
 & & & & A_3 & A_2 & A_1 & A_0 \\
\times & & & & B_3 & B_2 & B_1 & B_0 \\
\hline
 & & & & C_{30} & C_{20} & C_{10} & C_{00} \\
 & & & C_{31} & C_{21} & C_{11} & C_{01} & \\
 & & C_{32} & C_{22} & C_{12} & C_{02} & & \\
+ & C_{33} & C_{23} & C_{13} & C_{03} & & & \\
\hline
P_7 & P_6 & P_5 & P_4 & P_3 & P_2 & P_1 & P_0
\end{array}
$$

$C_{ij} = A_i B_j$

快速乘法

				1	1	0	1
×				1	0	1	1
				1	1	0	1
			1	1	0	1	
		0	0	0	0		
+	1	1	0	1			
1	0	0	0	1	1	1	1

乘法速度取决于逻辑门和加法器的传输延迟