

东南大学计算机学院

计算机系统组成

主讲教师： 徐造林

第4章 运算器

- 运算器是计算机中的执行部件，它可以对二进制数据进行各种**算术和逻辑运算**；
- 运算器也是计算机内部**数据信息的重要通路**。
- 本讲重点介绍运算器的核心部件——**算术逻辑运算单元ALU**的构成与工作原理，以及数据在运算器中基本**运算方法**。

4.1 运算器的基本组成与功能

- ▲ **运算器**是实现对数值数据的**算术运算**和逻辑数据的**逻辑操作**。
- ▲ 包括算术逻辑运算单元**ALU**，保存操作数、结果、出错信息和状态信息的各种**寄存器**以及**多路转换器**、**数据总线**、**控制线路**等逻辑组件。

1. 算术逻辑运算单元ALU

- ▲ 运算器中完成数据算术与逻辑运算的部件称之为算术与逻辑运算单元(ALU)；ALU是**运算器的核心**。
- ▲ ALU通常表示为**两个输入**端口，**一个输出**端口和**多个功能控制**信号端的一个逻辑符号。

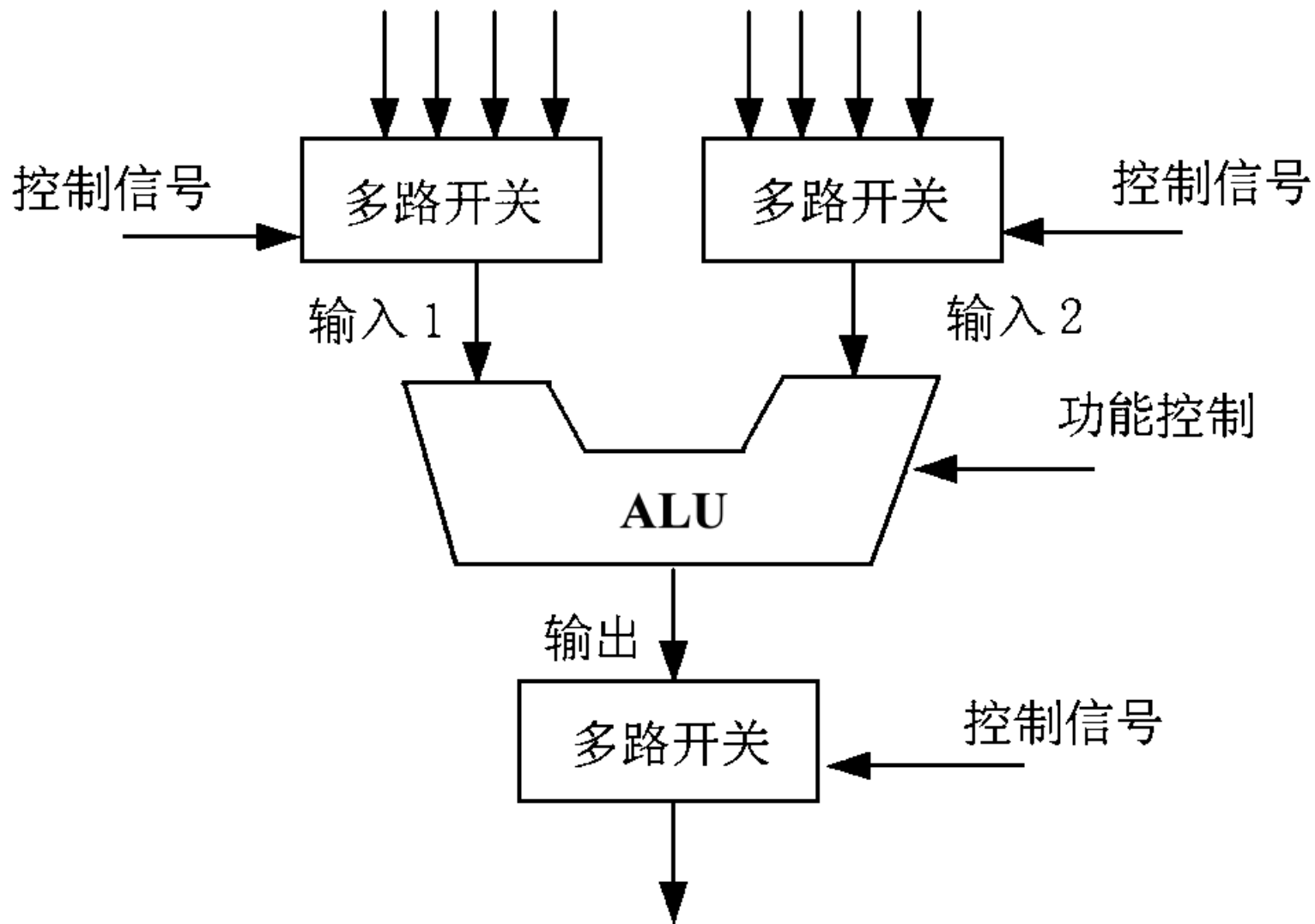


图4.1 ALU的逻辑符号表示与多路开关

2. 通用寄存器组

- ▲ **通用寄存器**组用于暂时存放参加运算的数据和某些中间结果。
 - 通用寄存器的数量越多，对提高运算器性能和程序执行速度越有利；
 - 通用寄存器组是**对用户开放**的，用户可以通过指令去使用这些寄存器。
- ▲ 在运算器中用来提供一个操作数并存放运算结果的通用寄存器称为**累加器**。
 - 如：ADD **A**, R_j

3. 专用寄存器

- ▲ **专用寄存器**用于记录指令执行过程中的**重要状态**标记，及运算前后数据的**暂存缓冲**等。
- **循环计数器**，它对程序员是透明的；
- **程序状态字(PSW)**，它存放着指令执行结果的某些状态；它对程序员是开放的；
- **堆栈指针SP(Stack Pointer)**，它指示了堆栈的使用情况；对程序员也是开放的。

- 8086/8088 标志寄存器

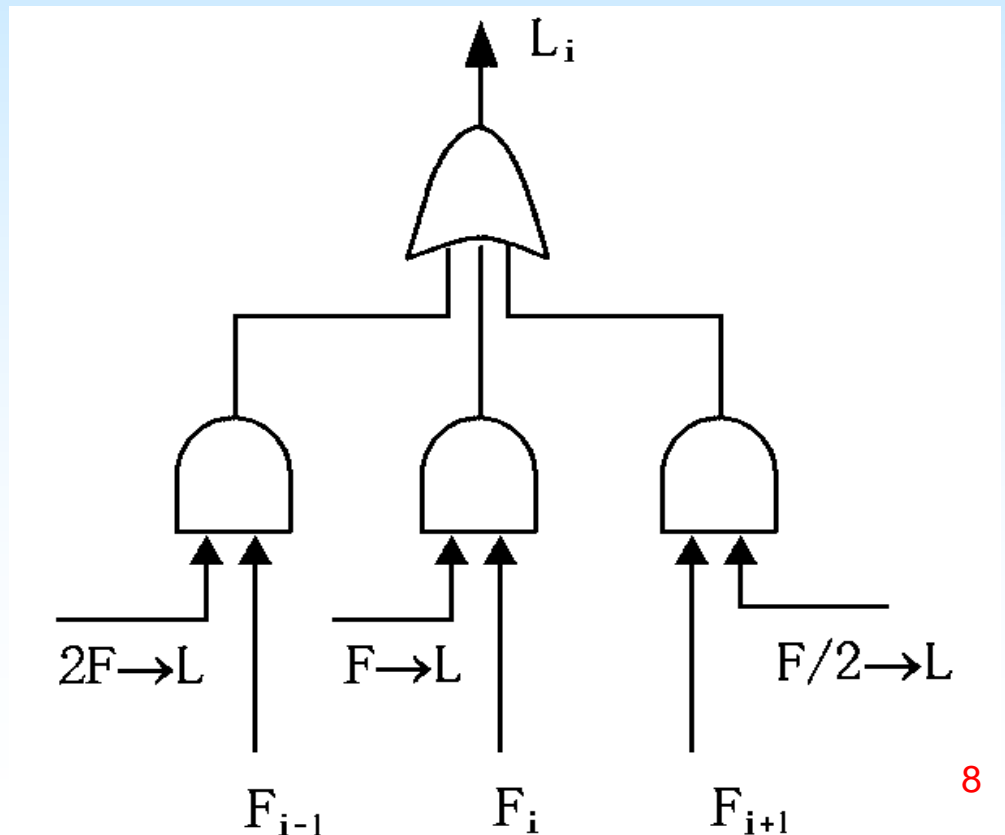
15				11	10	9	8	7	6		4		2		0
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

8086/8088 标志寄存器FLAG

- 8086/8088 段寄存器及IP

4. 控制逻辑

- ▲ **控制逻辑**产生控制信号来控制运算器执行多种运算功能。
- 在ALU的输出端设置**移位线路**来实现左移、右移和直送。
- 移位线路是一个多路选择器。
- 右图：实现移位功能的多路选择器



5. 运算器的组成

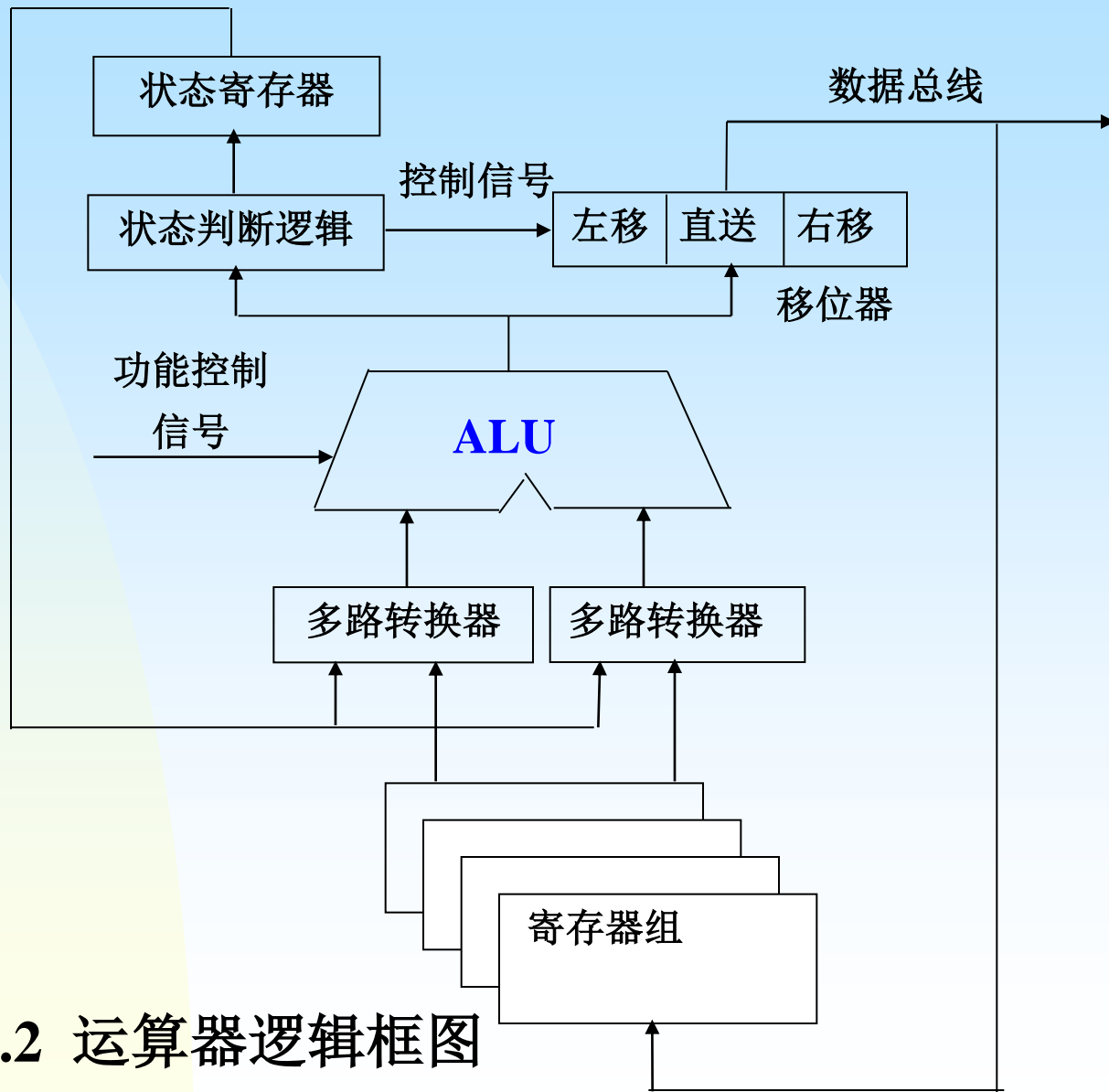


图4.2 运算器逻辑框图

6. 中央处理器CPU

▲ 运算器和控制部件合在一起成为中央处理器CPU

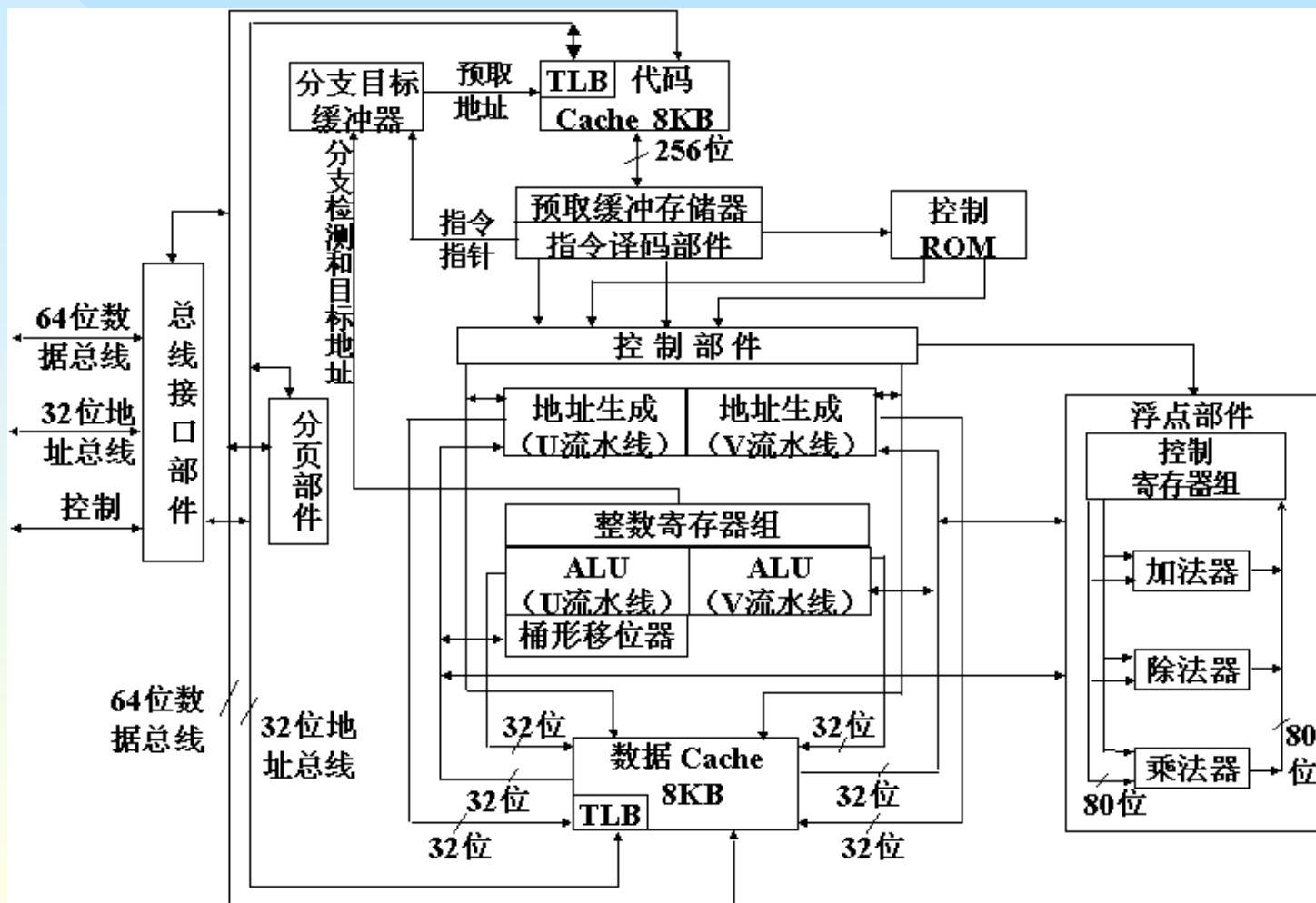


图4.3 奔腾CPU结构框图

4.2 加法器及定点加减法运算

4.2.1 加法器的实现

- ◆ 运算器中各种运算都是分解成加法运算进行的，**加法器**是计算机中最基本的运算单元。

1. 二进制加法单元

1) 半加

- ▲ 两个一位二进制数相加(**不考虑低位的进位**)，称为**半加**。实现半加操作的电路，称为**半加器**。

- 表4.1 半加运算真值表

X_i	Y_i	S_i	C_i
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- S_i 和 C_i 的逻辑表达式:

$$S_i = \bar{X}_i Y_i + X_i \bar{Y}_i$$

$$C_i = X_i Y_i$$

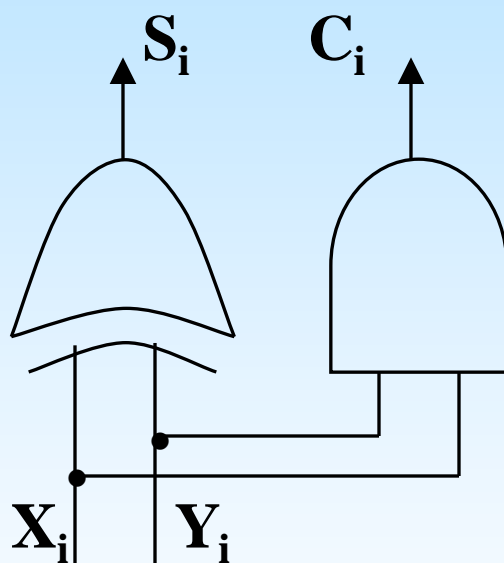
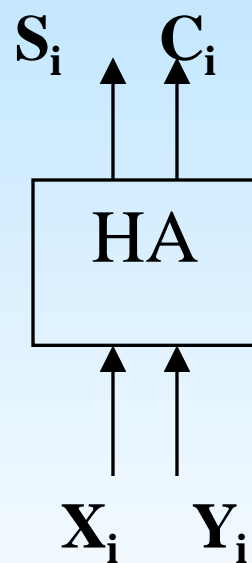


图4.4 (a) 逻辑图



(b) 符号表示

2) 全加

- ▲ 考虑低位进位的加法运算就是全加运算，实现全加运算的电路称为全加器。

- 表 2.9 全加运算的真值表

X_i	Y_i	C_{i-1}	S_i	C_i
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

- 根据真值表，可写出 F_i 和 C_i 的逻辑表达式：

$$S_i = \bar{X}_i Y_i \bar{C}_{i-1} + X_i \bar{Y}_i \bar{C}_{i-1} + \bar{X}_i \bar{Y}_i C_{i-1} + X_i Y_i C_{i-1} = X_i \oplus Y_i \oplus C_{i-1}$$

$$C_i = X_i Y_i \bar{C}_{i-1} + \bar{X}_i Y_i C_{i-1} + X_i \bar{Y}_i C_{i-1} + X_i Y_i C_{i-1} = (X_i \oplus Y_i) C_{i-1} + X_i Y_i$$

▲ 全加器逻辑图和全加器的符号表示

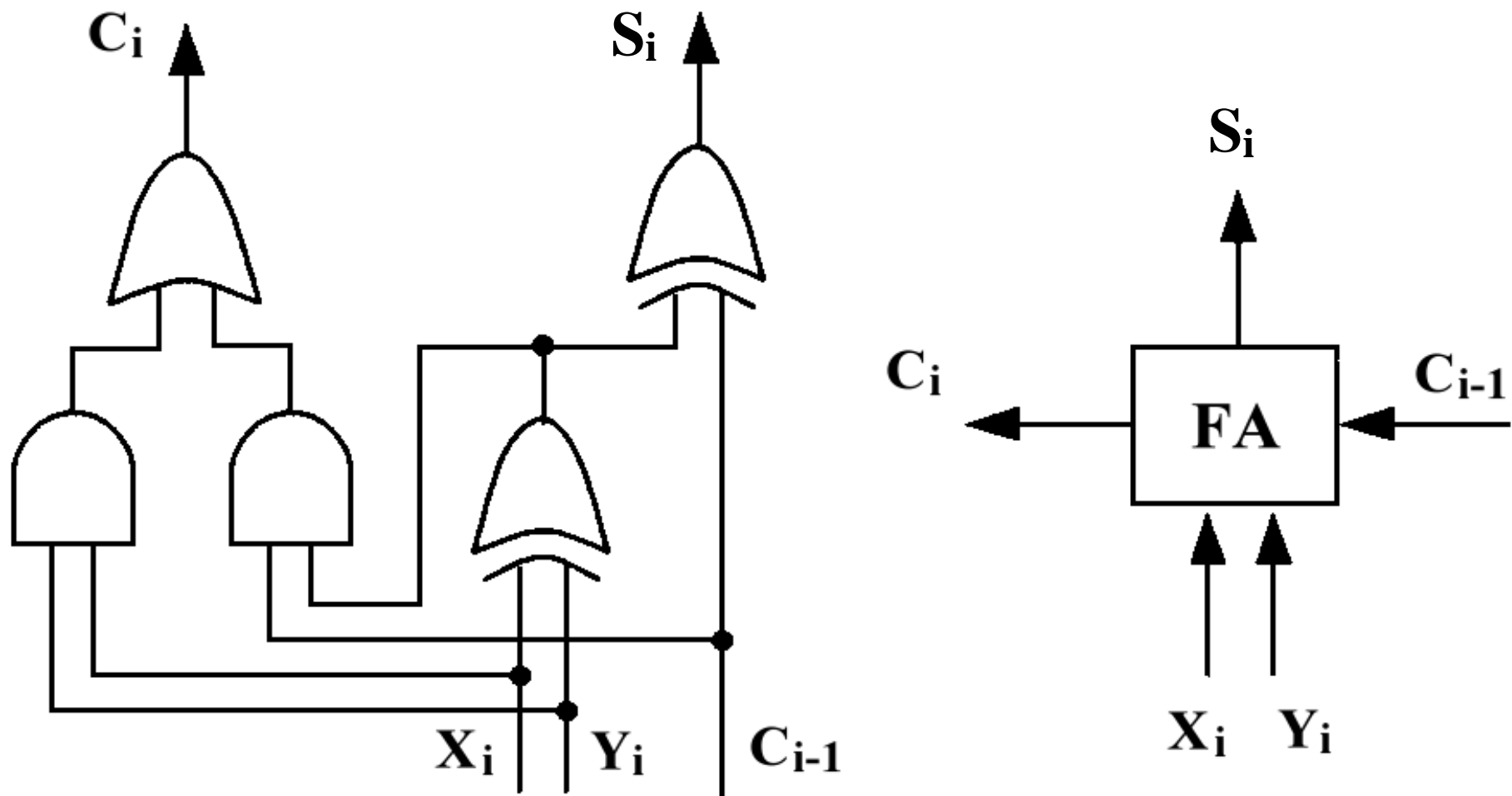


图 4.5 全加器的逻辑图和符号表示

2. 串行进位与并行进位

◆ n 个全加器相连可得 n 位的加法器

1) 串行进位或行波进位加法器

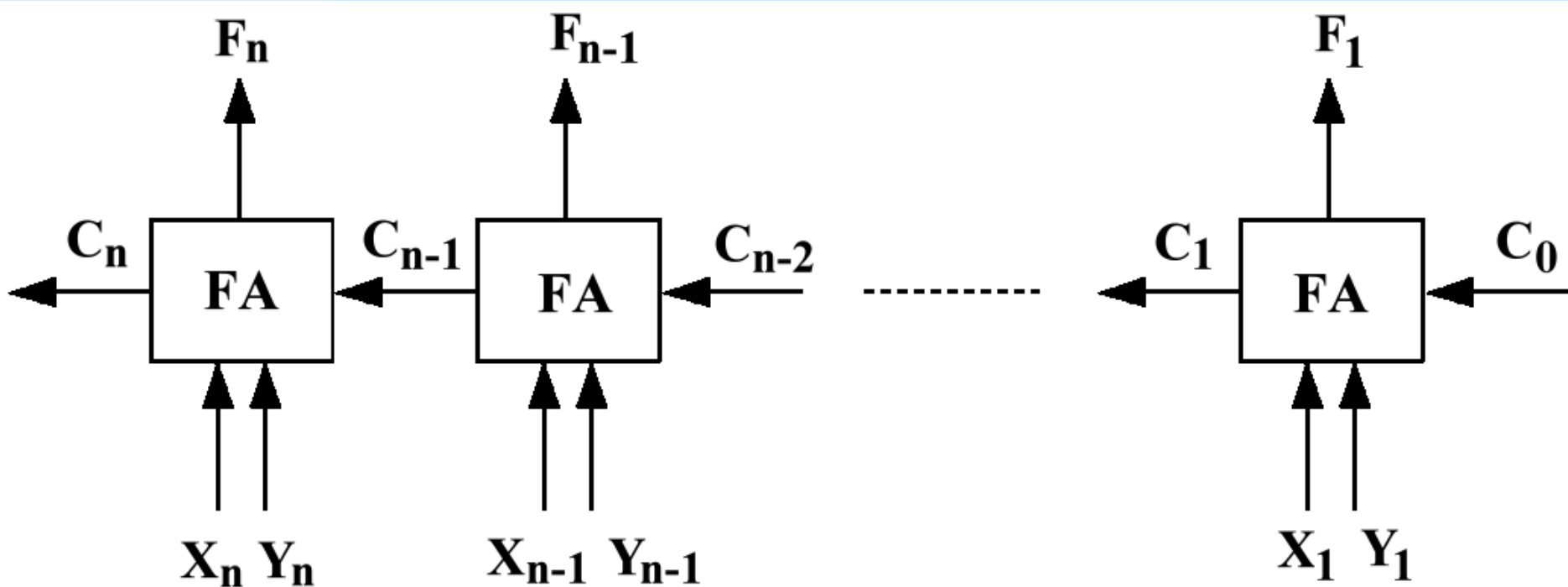


图4.6 n 位加法器

2) 先行进位或并行进位加法器

- 预先形成各位进位，将进位信号同时送到各位全加器的进位输入端。

▲ 就4位加法器，讨论其进位 C_1 、 C_2 、 C_3 和 C_4 的产生条件：

① 下述条件中任一条满足就可生成 $C_1=1$ ：

1) X_1 、 Y_1 均为“1”；

2) X_1 、 Y_1 任一个为“1”，且进位 C_0 为“1”。

- 可得 C_1 的表达式为：

$$C_1 = X_1 Y_1 + (X_1 + Y_1) C_0$$

② 下述条件中任一条满足，可生成 $C_2=1$ 。

1) X_2 、 Y_2 均为“1”；

2) X_2 、 Y_2 任一个为“1”，且进位 C_1 为“1”。

• 可得 C_2 的表达式为：

$$\begin{aligned} C_2 &= X_2 Y_2 + (X_2 + Y_2) C_1 \\ &= X_2 Y_2 + (X_2 + Y_2) X_1 Y_1 + (X_2 + Y_2) (X_1 + Y_1) C_0 \end{aligned}$$

③ 同理，可得 C_3 的表达式为：

$$\begin{aligned} C_3 &= X_3 Y_3 + (X_3 + Y_3) C_2 \\ &= X_3 Y_3 + (X_3 + Y_3) [X_2 Y_2 + (X_2 + Y_2) X_1 Y_1 + \\ &\quad (X_2 + Y_2) (X_1 + Y_1) C_0] \\ &= X_3 Y_3 + (X_3 + Y_3) X_2 Y_2 + (X_3 + Y_3) (X_2 + Y_2) X_1 Y_1 + \\ &\quad (X_3 + Y_3) (X_2 + Y_2) (X_1 + Y_1) C_0 \end{aligned}$$

④ 同理,可得 C_4 的表达式为:

$$\begin{aligned}C_4 &= X_4 Y_4 + (X_4 + Y_4) C_3 \\&= X_4 Y_4 + (X_4 + Y_4) [X_3 Y_3 + (X_3 + Y_3) X_2 Y_2 + \\&\quad (X_3 + Y_3) (X_2 + Y_2) X_1 Y_1 + (X_3 + Y_3) (X_2 + Y_2) (X_1 + Y_1) C_0] \\&= X_4 Y_4 + (X_4 + Y_4) X_3 Y_3 + (X_4 + Y_4) (X_3 + Y_3) X_2 Y_2 + \\&\quad (X_4 + Y_4) (X_3 + Y_3) (X_2 + Y_2) X_1 Y_1 + \\&\quad (X_4 + Y_4) (X_3 + Y_3) (X_2 + Y_2) (X_1 + Y_1) C_0\end{aligned}$$

▲ 定义两个辅助函数:

$$P_i = X_i + Y_i$$

$$G_i = X_i Y_i$$

- P_i 表示进位传递函数, 当 X_i 、 Y_i 中有一个为“1”时, 若有低位进位输入, 则本位向高位传送进位。

- G_i 表示进位产生函数，当 X_i 、 Y_i 均为“1”时，本位一定向高位产生进位输出。

▲ 将 P_i 、 G_i 代入前面的 $C_1 \sim C_4$ 式，可得：

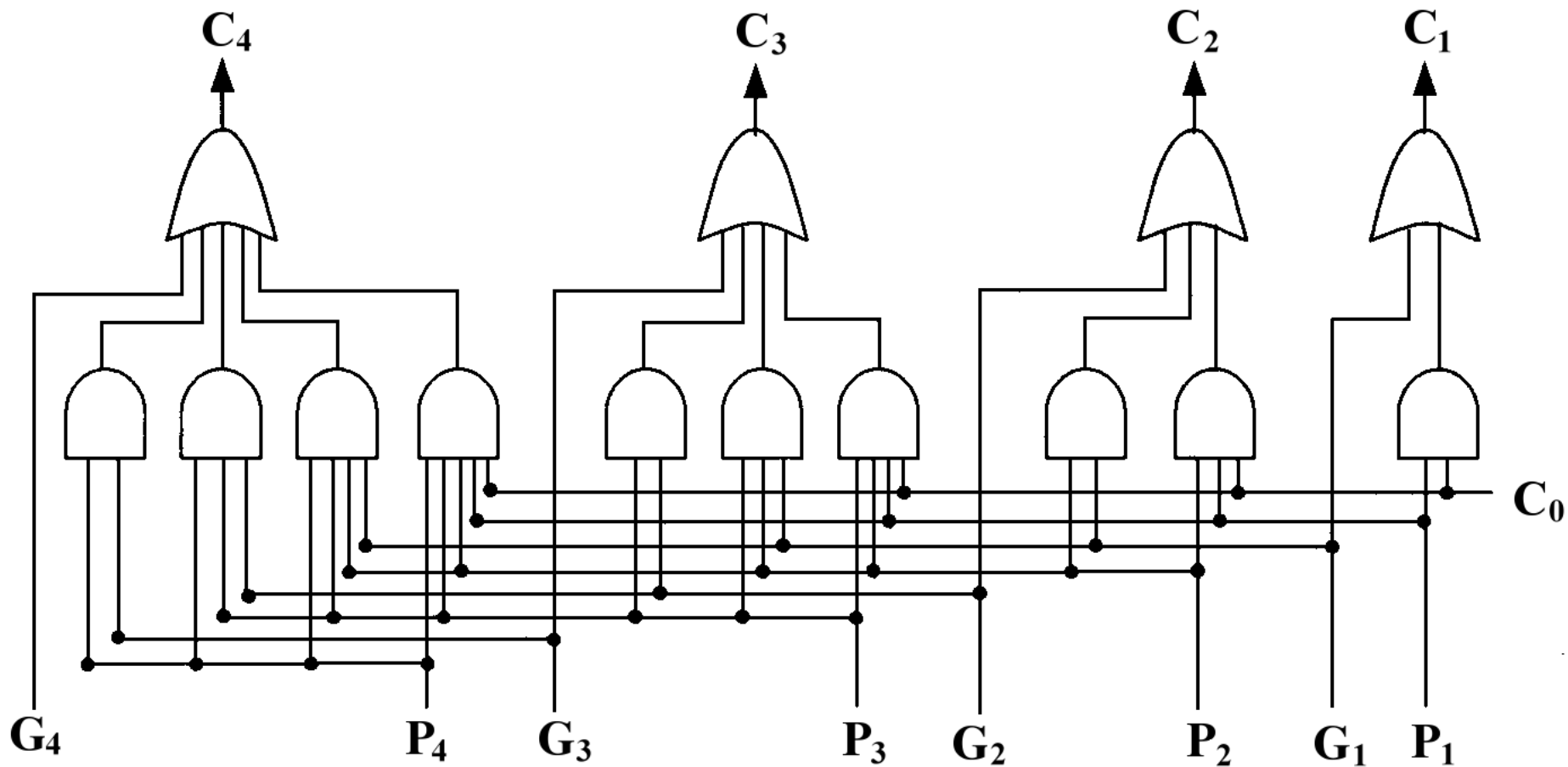
$$C_1 = G_1 + P_1 C_0$$

$$C_2 = G_2 + P_2 G_1 + P_2 P_1 C_0$$

$$C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_0$$

$$C_4 = G_4 + P_4 G_3 + P_4 P_3 G_2 + P_4 P_3 P_2 G_1 + P_4 P_3 P_2 P_1 C_0$$

- 先行进位产生电路



- 其中 $P_i = X_i + Y_i$

$$G_i = X_i Y_i$$

4位先行进位加法器

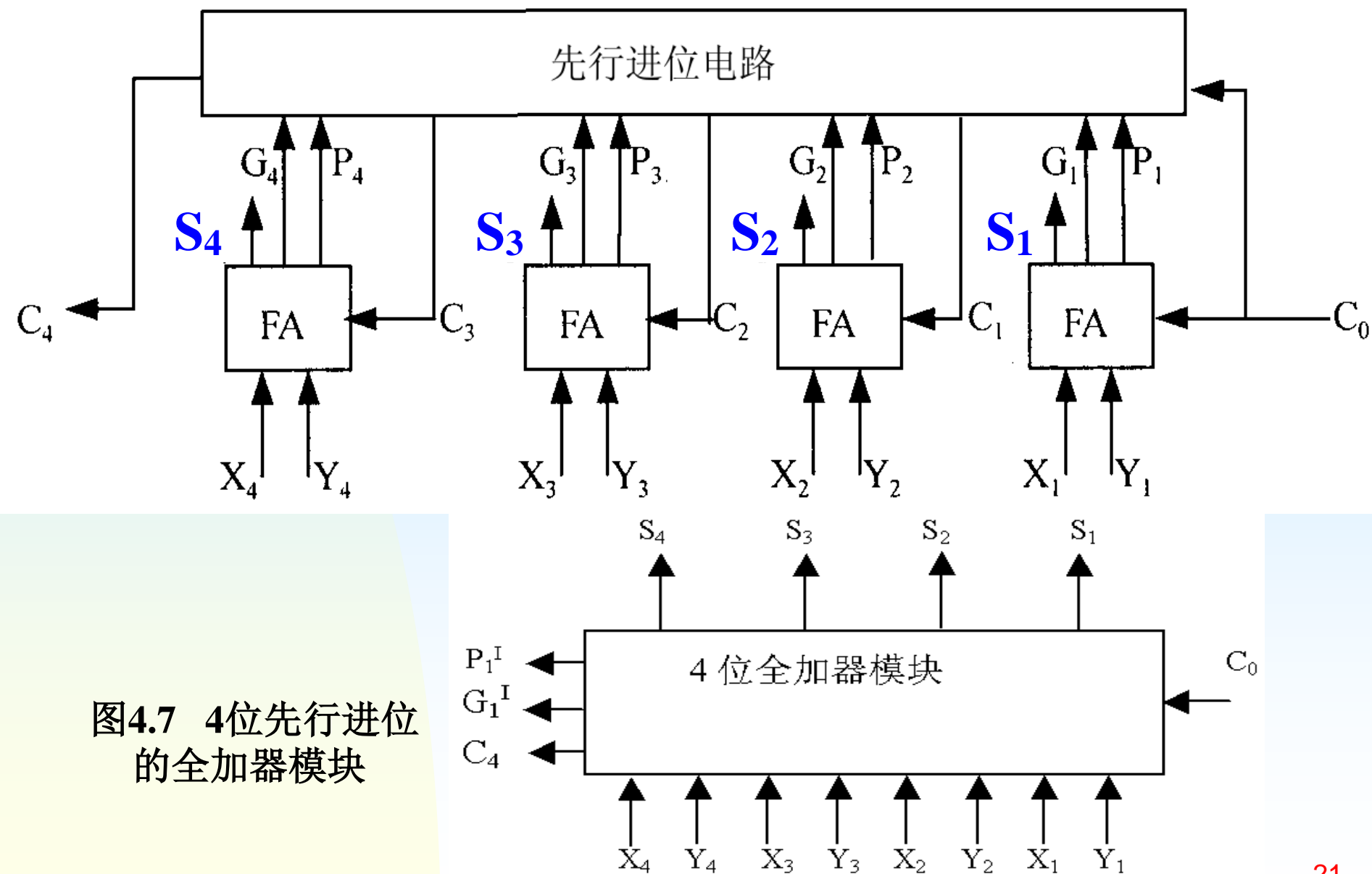
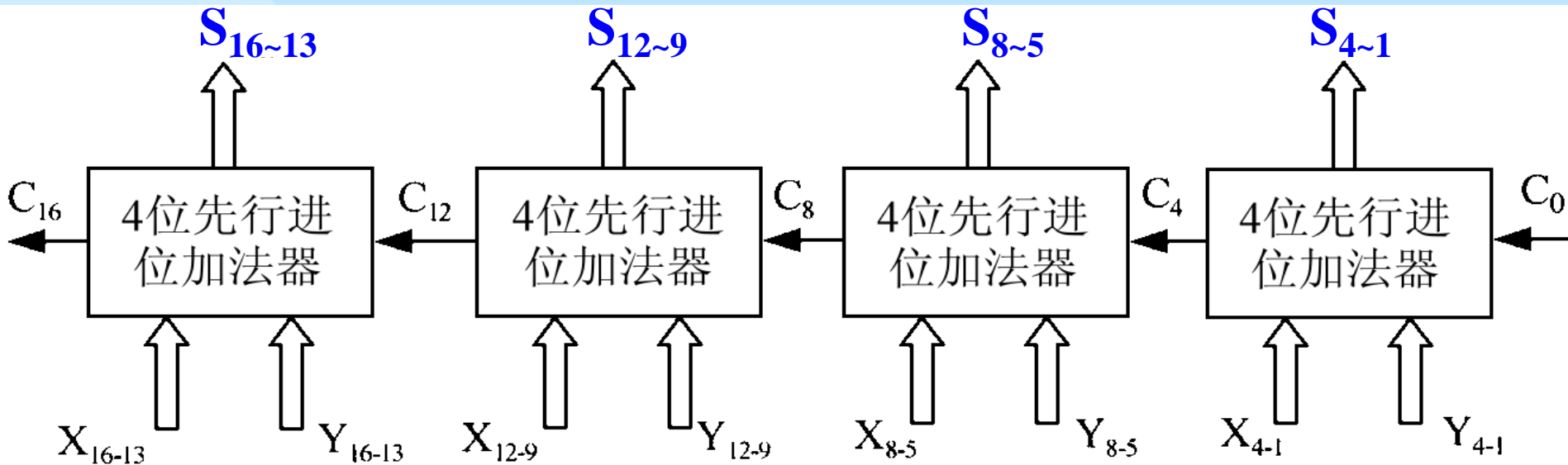


图4.7 4位先行进位的全加器模块

▲ 组内并行，组间串行进位的16位全加器



- 在各加法单元之间，进位信号是串行传送的，而在加法单元内，进位信号是并行传送的。

▲ 组内并行，组间并行进位的16位全加器

- $C_m = G_m + P_m C_0$
- C_m 表示4位加法器的进位输出， P_m 表示4位加法器的进位传递输出， G_m 表示4位加法器的进位产生输出。
- P_m 和 G_m 分别为：

$$P_m = P_4 P_3 P_2 P_1$$

$$G_m = G_4 + P_4 G_3 + P_4 P_3 G_2 + P_4 P_3 P_2 G_1$$

- 应用于四个4位先行进位加法器，则有：

$$C_{m1} = G_{m1} + P_{m1} C_0$$

$$\begin{aligned} C_{m2} &= G_{m2} + P_{m2} C_{m1} \\ &= G_{m2} + P_{m2} G_{m1} + P_{m2} P_{m1} C_0 \end{aligned}$$

$$\begin{aligned} C_{m3} &= G_{m3} + P_{m3} C_{m2} \\ &= G_{m3} + P_{m3} G_{m2} + P_{m3} P_{m2} G_{m1} + P_{m3} P_{m2} P_{m1} C_0 \end{aligned}$$

$$\begin{aligned} C_{m4} &= G_{m4} + P_{m4} C_{m3} \\ &= G_{m4} + P_{m4} G_{m3} + P_{m4} P_{m3} G_{m2} + \\ &\quad P_{m4} P_{m3} P_{m2} G_{m1} + P_{m4} P_{m3} P_{m2} P_{m1} C_0 \end{aligned}$$

3. ALU部件

- ◆ ALU是一种能进行多种算术运算与逻辑运算的组合逻辑电路，它的基本逻辑结构是先行进位加法器。
- ◆ 74181型4位ALU中规模集成电路工作原理
 - 能对两个4位二进制代码 $A_3A_2A_1A_0$ 和 $B_3B_2B_1B_0$ 进行16种算术运算（当M为低电位时）和16种逻辑运算（当M为高电位时），产生结果 $F_3F_2F_1F_0$ 。
 - 16种运算操作由 $S_3S_2S_1S_0$ 四位控制选择
 - C_n 是ALU的最低位进位输入，低电平有效，即 $C_n=L$ 表示有进位输入；
 - C_{n+4} 是ALU进位输出信号。

▲ 表 4.2 74181型4位ALU在正逻辑下的功能表

S_3	S_2	S_1	S_0	M=H 逻辑运算	M=L 算术运算	
					$\overline{C}_n=1$	$\overline{C}_n=0$
L	L	L	L	\overline{A}	A	A+1
L	L	L	H	$\overline{A+B}$	A+B	(A+B)加 1
L	L	H	L	$\overline{A} \bullet B$	$A + \overline{B}$	(A+ \overline{B})加 1
L	L	H	H	“0”	减 1	“0”
L	H	L	L	$\overline{A} \bullet \overline{B}$	A 加(A $\bullet \overline{B}$)	A 加(A $\bullet \overline{B}$) 加 1
L	H	L	H	\overline{B}	(A $\bullet \overline{B}$)加(A+B)	(A $\bullet \overline{B}$)加(A+B)加 1
L	H	H	L	$A \oplus B$	A 减 B 减 1	A 减 B
L	H	H	H	$A \bullet \overline{B}$	(A $\bullet \overline{B}$)减 1	A $\bullet \overline{B}$
H	L	L	L	$\overline{A+B}$	A 加(A $\bullet B$)	A 加(A $\bullet B$) 加 1
H	L	L	H	$\overline{A \oplus B}$	A 加 B	A 加 B 加 1
H	L	H	L	B	(A $\bullet B$)加(A+ \overline{B})	(A $\bullet B$)加(A+B) 加 1
H	L	H	H	$A \bullet B$	(A $\bullet B$)减 1	A $\bullet B$
H	H	L	L	“1”	A 加 A	A 加 A 加 1
H	H	L	H	$A + \overline{B}$	A 加(A+B)	A 加(A+B) 加 1
H	H	H	L	A+B	A 加(A+ \overline{B})	A 加(A+ \overline{B}) 加 1
H	H	H	H	A	A 减 1	A

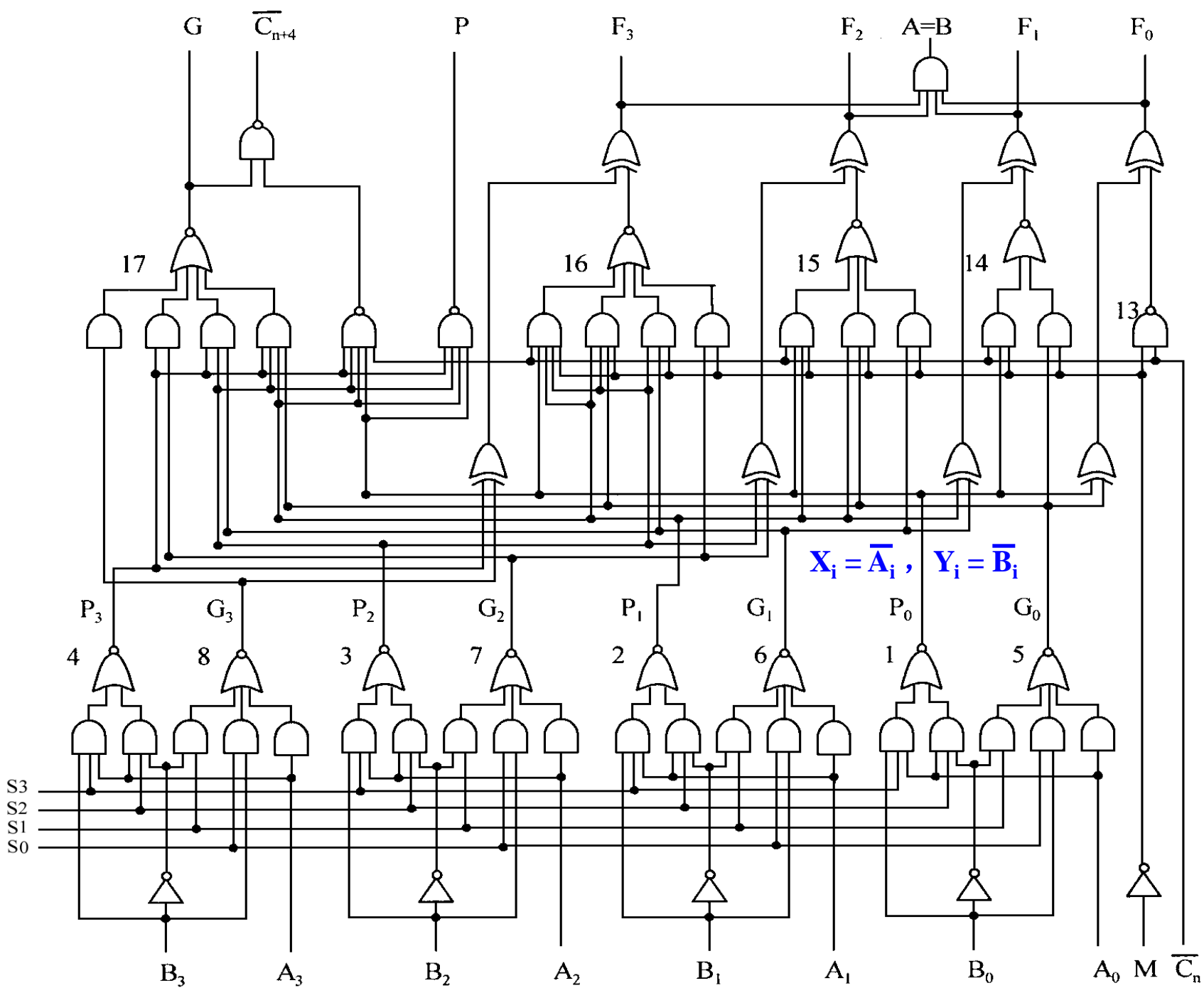


图 4.9 (b) 74181 型 ALU 逻辑图

1) ALU实现加法操作的原理

▲ 当 $S_3S_2S_1S_0=HLLH$ ， $M=L$ 时，ALU实现对 $A_3A_2A_1A_0$ 和 $B_3B_2B_1B_0$ 两个4位二进制代码在进位输入 C_n 参与下的加法运算；

• 即： $F_i = A_i \oplus B_i \oplus C_{n+i}$ ($i=3, 2, 1, 0$)

• 设 $X_i = \overline{A_i}$ ， $Y_i = \overline{B_i}$ ；可推导 X_i 、 Y_i 和 A_i 、 B_i 的关系：

$$X_i + Y_i = \overline{A_i B_i}$$

$$X_i Y_i = \overline{A_i + B_i}$$

$$X_i \oplus Y_i = A_i \oplus B_i$$

- 由图4.10可知：

$$P_i = \overline{A_i \bar{B}_i S_2 + A_i B_i S_3} = X_i + Y_i$$

$$G_i = \overline{A_i + B_i S_0 + \bar{B}_i S_1} = X_i Y_i$$

- ALU就可以改画成以 X_i 、 Y_i 为输入的结构较简单的单元；
- 可证明 $G_i \oplus P_i = X_i \oplus Y_i$ 。
- 图3.10 是一个由先行进位加法器组成的单元，电路输出：
- $F_i = X_i \oplus Y_i \oplus C_{n+i} = A_i \oplus B_i \oplus C_{n+i}$ 。

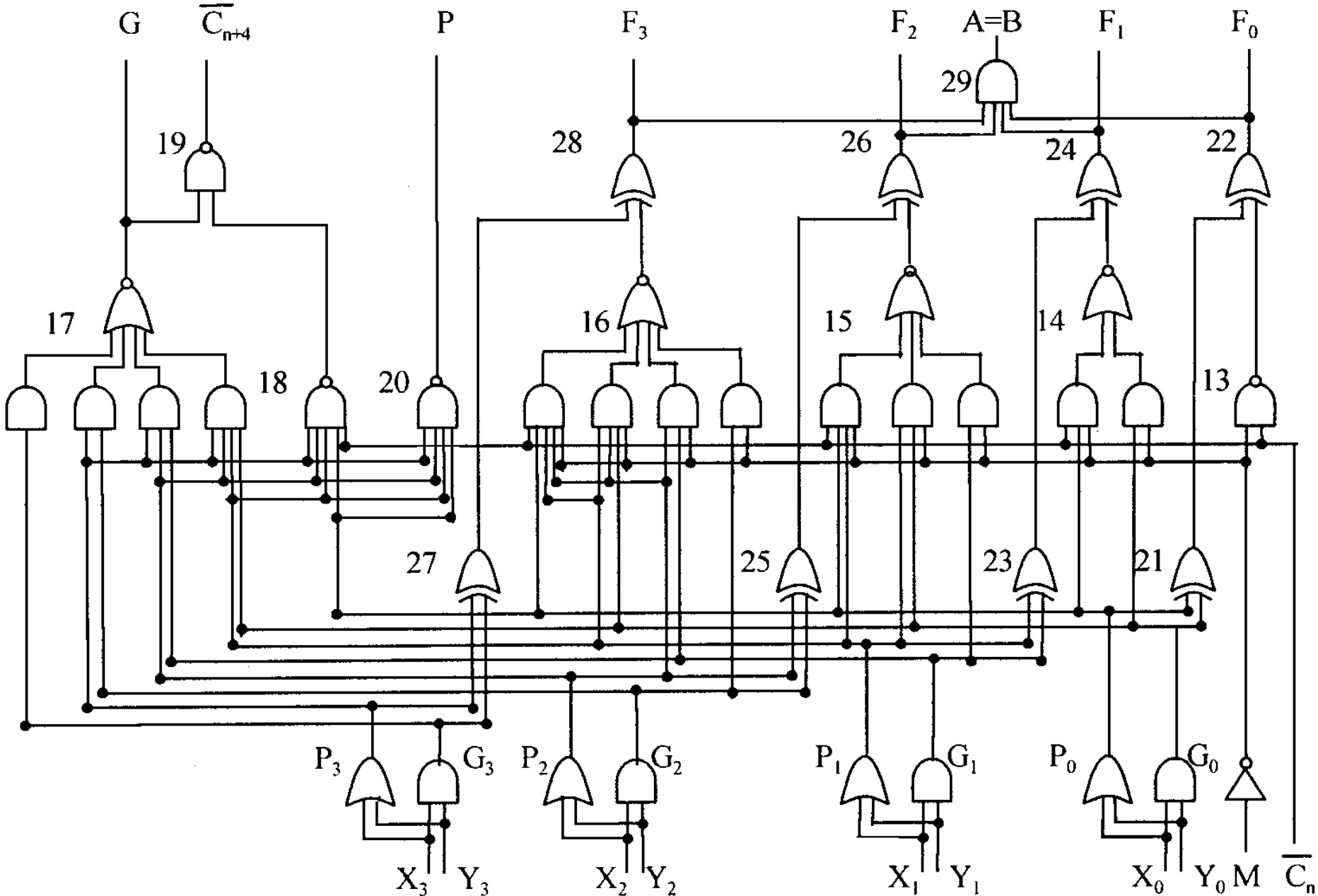


图4.10 74181 作加法运算时的简化逻辑图

2) ALU单元实现逻辑运算

- ▲ 当M=H时，由图4.10可知，进位门13~16均被封锁， $F_i = P_i \oplus G_i$ ，位间不发生关系，电路执行逻辑运算。
- $S_3S_2S_1S_0 = \text{HLLH}$ 时， $F_i = P_i \oplus G_i = \overline{A_i} \oplus B_i$ ，对输入数据 $A_3A_2A_1A_0$ 和 $B_3B_2B_1B_0$ 执行逻辑“**同或**” (异或非)操作。
- $S_3S_2S_1S_0 = \text{HHHH}$ 时， $F_i = P_i \oplus G_i = A_i$ ，即 $F=A$ ，此时，电路执行“**传送A**”的操作。
- ▲ 按以上方法，可全面分析、理解74181的逻辑图和真值表。

▲ 左图是 74181ALU在正逻辑下的图形符号

▲ 下图 是用4片74181组成的16位ALU，芯片内用先行进位方法，但片间为串行进位。

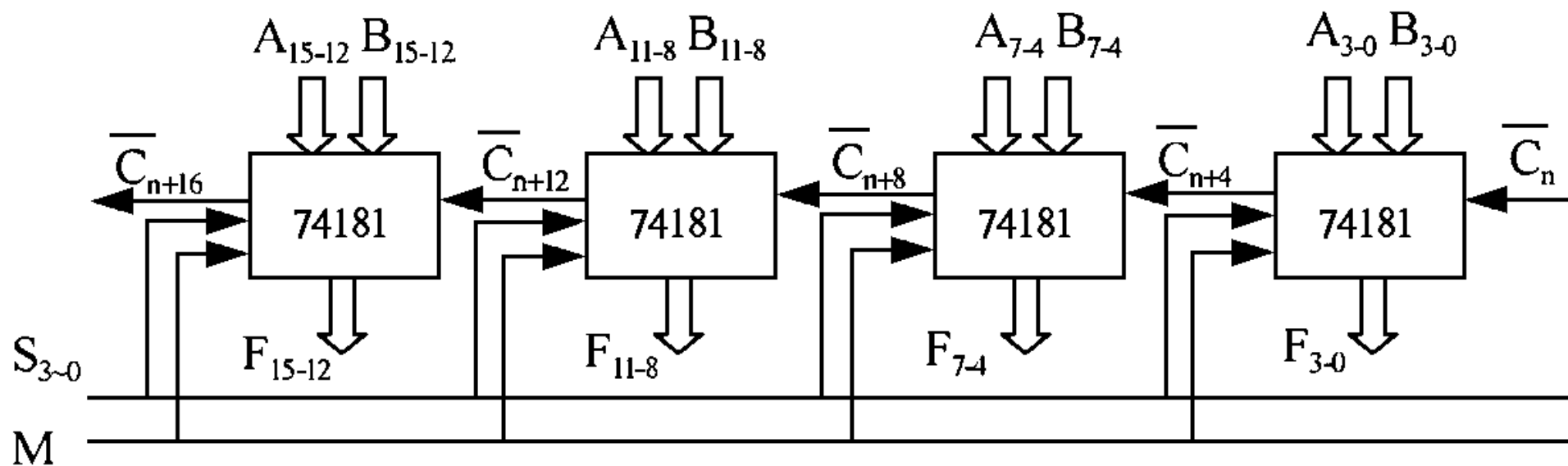
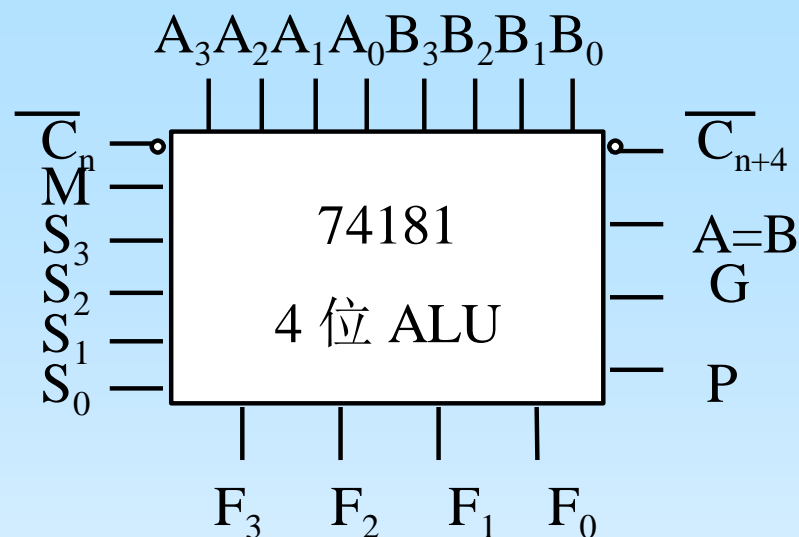


图4.11 用四片74181构成的16位ALU

- ▲ 在图4.10的74181的逻辑图中，用先行进位方法产生的进位输出 C_{n+4} 和图中P、G的输出信号用表达式：

- 考虑算术运算， $M=L$

$$P = \overline{P_3 P_2 P_1 P_0}$$

$$G = \overline{G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0}$$

$$\overline{C_{n+4}} = \overline{G \overline{P_3 P_2 P_1 P_0} \overline{C_n}} = \overline{G} + \overline{P} \overline{C_n} = \overline{G P + G C_n}$$

- \overline{P} 称为片间进位传递函数， \overline{G} 称为片间进位产生函数
- ▲ 根据74181提供的G、P信号，很容易实现芯片之间的先行进位。

▲ 在图4.12 中，高三片74181的片间进位输入可以表示为如下表达式：

$$\begin{aligned}\bar{C}_{n+4} &= \bar{G}_{(0)} + \bar{P}_{(0)} \bar{C}_n \\ &= \overline{G_{(0)} P_{(0)} + G_{(0)} C_n}\end{aligned}$$

$$\begin{aligned}\bar{C}_{n+8} &= \bar{G}_{(1)} + \bar{P}_{(1)} \bar{C}_{n+4} = \bar{G}_{(1)} + \bar{P}_{(1)} (\bar{G}_{(0)} + \bar{P}_{(0)} \bar{C}_n) \\ &= \overline{G_{(1)} P_{(1)} + G_{(1)} G_{(0)} P_{(0)} + G_{(1)} G_{(0)} C_n}\end{aligned}$$

$$\begin{aligned}\bar{C}_{n+12} &= \bar{G}_{(2)} + \bar{P}_{(2)} \bar{C}_{n+8} \\ &= \bar{G}_{(2)} + \bar{P}_{(2)} \overline{G_{(1)} P_{(1)} + G_{(1)} G_{(0)} P_{(0)} + G_{(1)} G_{(0)} C_n} \\ &= \overline{G_{(2)} P_{(2)} + G_{(2)} G_{(1)} P_{(1)} + G_{(2)} G_{(1)} G_{(0)} P_{(0)} + G_{(2)} G_{(1)} G_{(0)} C_n}\end{aligned}$$

$$P = P_{(3)} P_{(2)} P_{(1)} P_{(0)}$$

$$G = G_{(3)} P_{(3)} + G_{(3)} G_{(2)} P_{(2)} + G_{(3)} G_{(2)} G_{(1)} P_{(1)} + G_{(3)} G_{(2)} G_{(1)} G_{(0)}$$

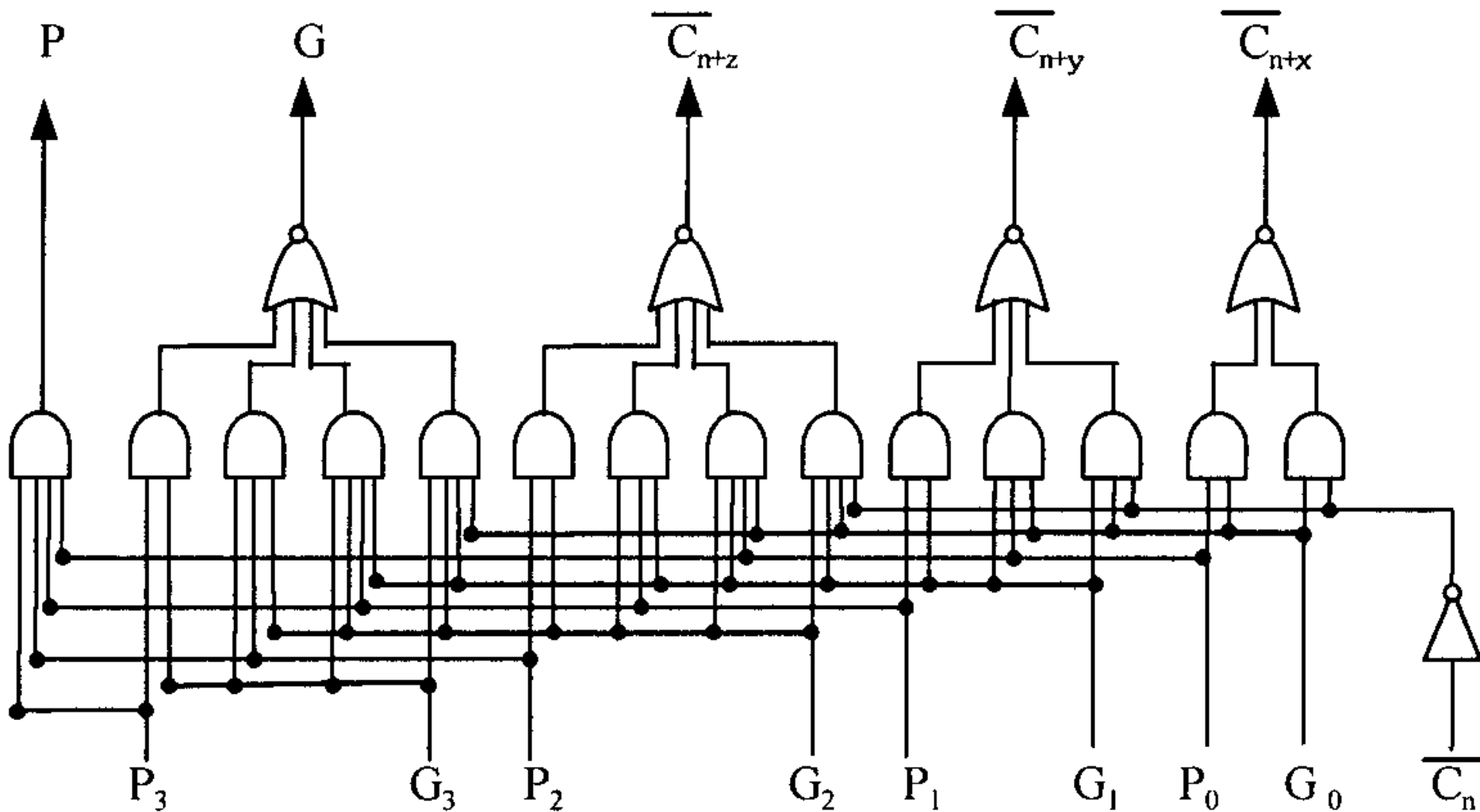


图4.12 片间先行进位产生电路(74182)

- ▲ 用四片74181和一片74182芯片组成的16位快速ALU；利用16片74181和5片74182芯片可以组成64位快速ALU。

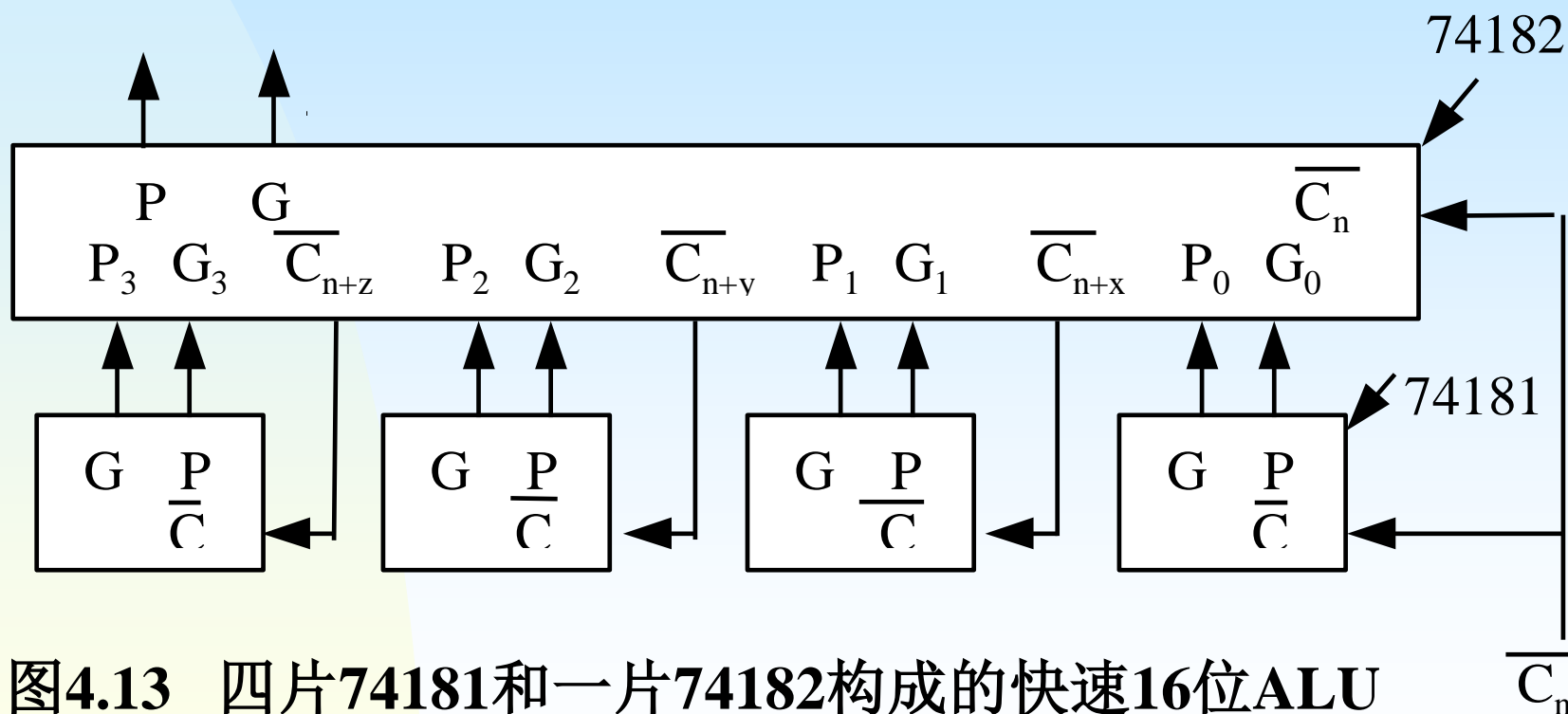


图4.13 四片74181和一片74182构成的快速16位ALU

4.2.2 补码定点加、减法

- ◆ 计算机的一个重要特点是它只能用**有限的数码位数**来表示操作数和操作结果。
 - 定点加、减法运算只有在**遵守模运算规则**的限制条件下其结果才是正确的，否则就会出现结果“溢出”。
 - 制定用来表示正、负数的各种码制；通过**数据编码**来简化数据的运算。

◆ 补码的加、减法运算公式：

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

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

- 在补码制方法下，无论X、Y是正数还是负数，加、减法运算统一采用加法来处理；
- $[X]_{\text{补}}$ 和 $[Y]_{\text{补}}$ 的符号位和数值位一起参与求和运算，加、减运算结果的符号位也在求和运算中直接得出。

◆ 例1: 已知 $[X]_{\text{补}}=01001$, $[Y]_{\text{补}}=11100$;
求 $[X+Y]_{\text{补}}$, $[X-Y]_{\text{补}}$ 。

• 解: $[-Y]_{\text{补}}=00100$

$$\begin{aligned} [X+Y]_{\text{补}} &= ([X]_{\text{补}} + [Y]_{\text{补}}) \quad \text{MOD } 2^5 \\ &= (01001 + 11100) \quad \text{MOD } 2^5 \\ &= 00101 \end{aligned}$$

$$\begin{aligned} [X-Y]_{\text{补}} &= ([X]_{\text{补}} + [-Y]_{\text{补}}) \quad \text{MOD } 2^5 \\ &= (01001 + 00100) \quad \text{MOD } 2^5 \\ &= 01101 \end{aligned}$$

◆ 实现补码加、减法运算的逻辑电路

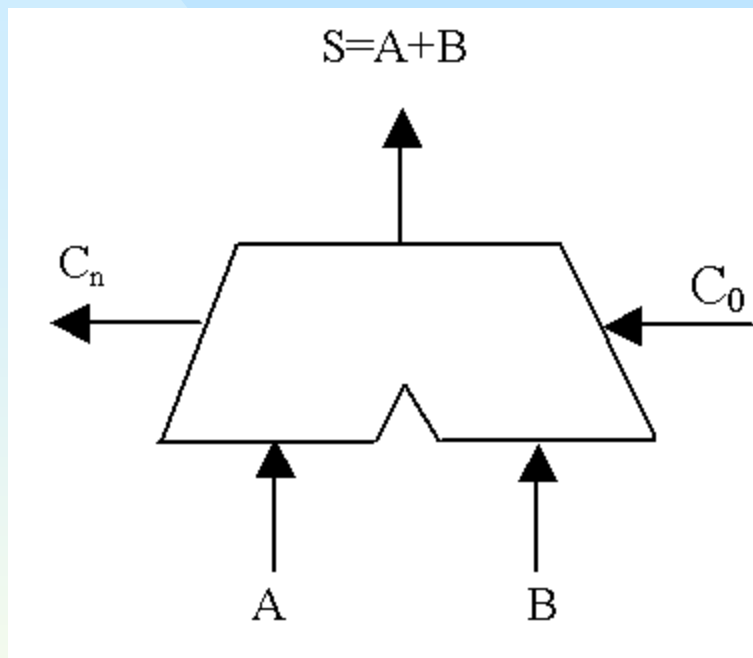


图4.14 加法器的逻辑符号

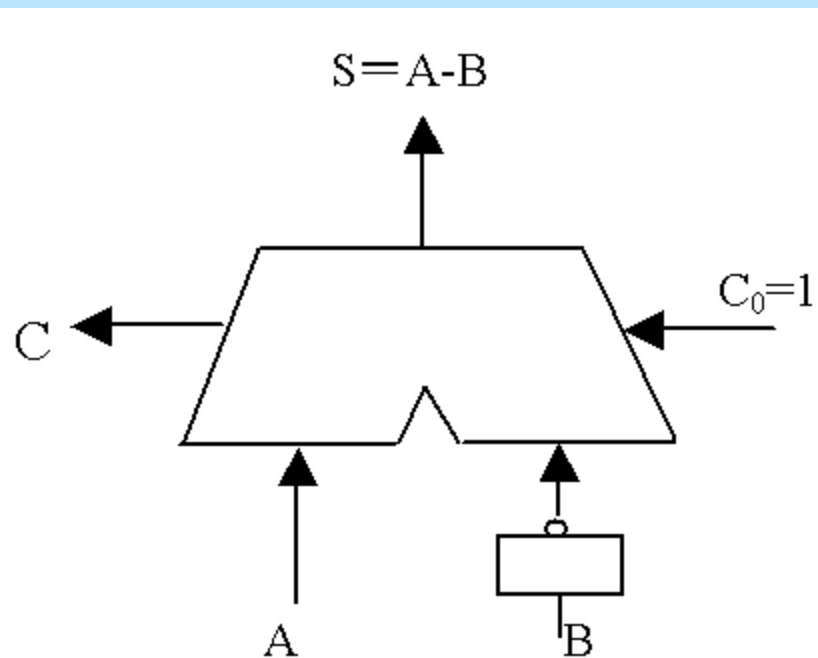


图4.15 减法器的逻辑符号

$$B_i^* = B_i \oplus C_0$$

$$C_0 = \begin{cases} 0, & \text{加 (ADD) 操作} \\ 1, & \text{减 (SUB) 操作} \end{cases}$$

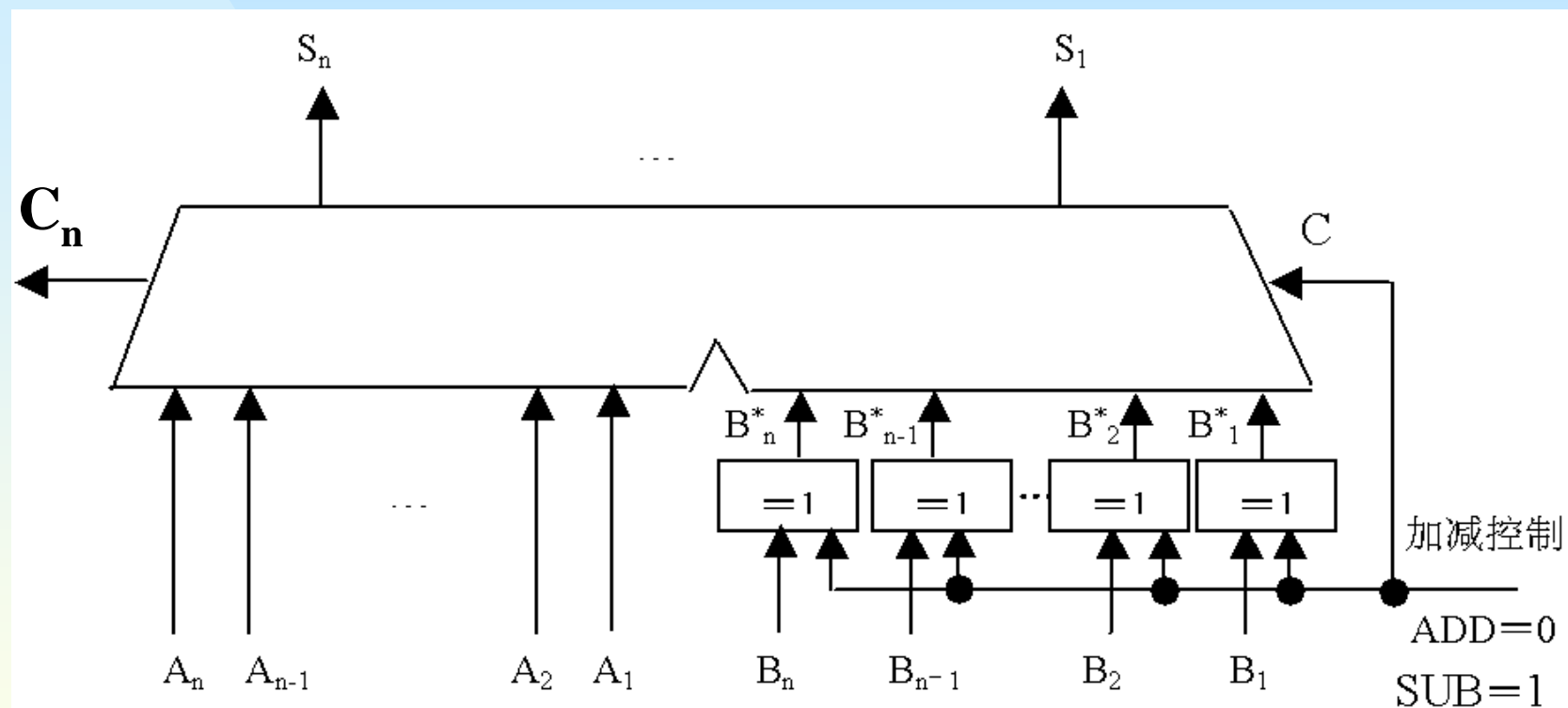


图4.16 n位补码加法器

4.2.3 数据溢出及检测

- ◆ 当算术运算的结果超出了数码位数允许的数据范围时，就产生溢出。
- 对于n位的二进制码表示的补码整数，它可表示的数据范围为 $-2^{n-1} \sim 2^{n-1}-1$ 。
- 若结果超过了允许表示的最大正数时，产生的溢出称为上溢；
- 若结果超过了允许表示的最小负数时，产生的溢出称为下溢。
- 在运算器中应设有溢出判别线路和溢出标志位。

- ◆ 例2: 已知 $[X]_{\text{补}}=01010$, $[Y]_{\text{补}}=01010$
 $[X+Y]_{\text{补}}=(01010+01010) \text{ MOD } 2^5 = 10100$ 溢出
 - $[10]_{10} + [10]_{10} = [20]_{10} > 15$ 产生上溢
- ◆ 例3: 已知 $[X]_{\text{补}}=10010$, $[Y]_{\text{补}}=00100$
 $[X-Y]_{\text{补}}=(10010+11100) \text{ MOD } 2^5 = 01110$ 溢出
 - $[-14]_{10} - [4]_{10} = [-18]_{10} < -16$ 产生下溢

◆ 溢出常用的判别方法:

① 根据数据表示范围判断

- 带符号数相加时，符号位产生的进位有自动丢弃和指示溢出两种可能性。
- 不同符号的数相加不可能产生溢出。
- 当两个操作数符号相同而和数的符号位与操作数符号也相同时不可能产生溢出。
- 只有当两个操作数符号相同而和数的符号位与操作数符号不同时才产生溢出。

$$\begin{aligned} \text{OVR} &= A_n B_n \overline{S_n} + \overline{A_n} \overline{B_n} S_n \\ &= (A_n \oplus S_n) (B_n \oplus S_n) \end{aligned}$$

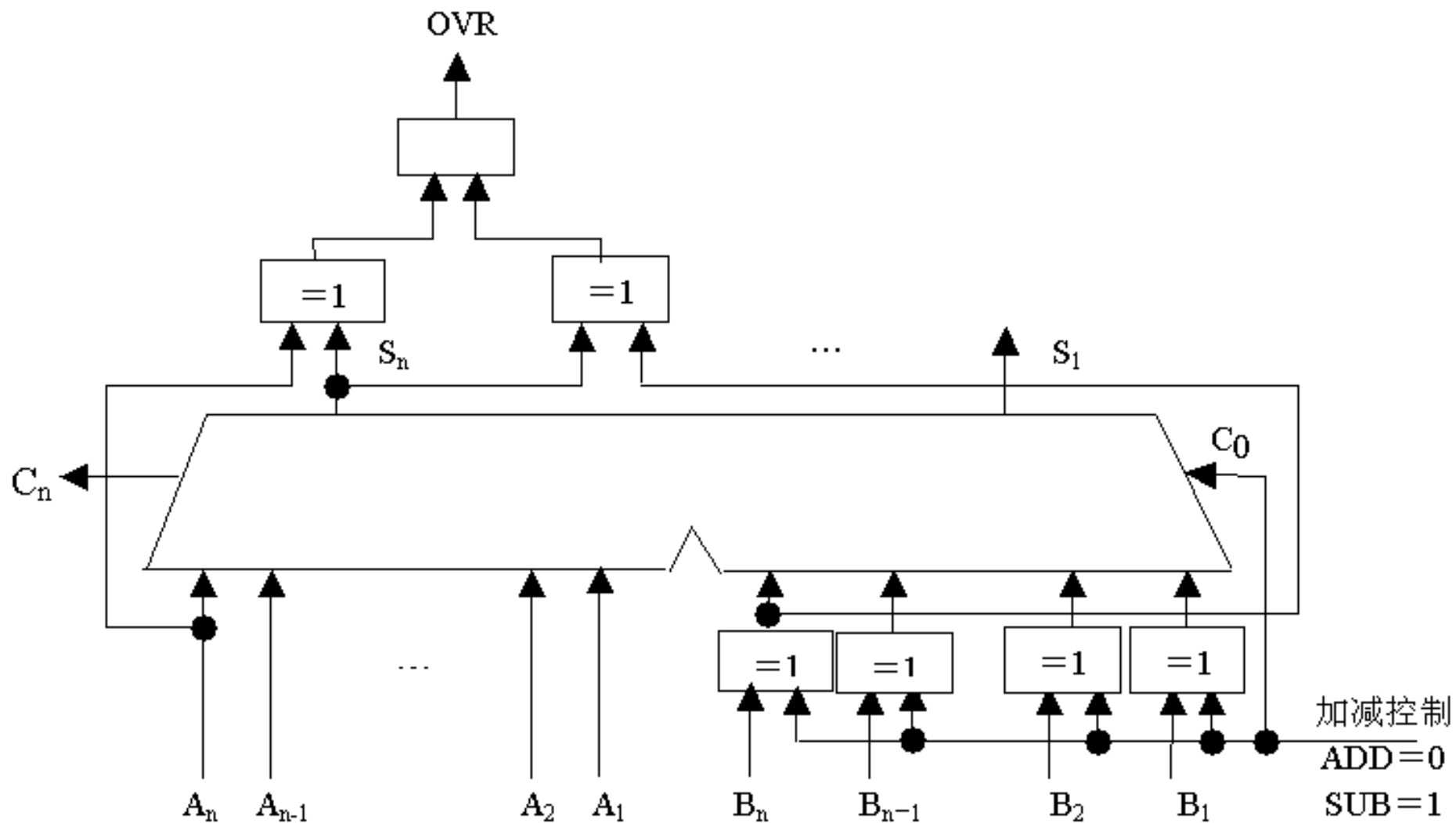


图4.17 带溢出检测电路的补码加法器

② 两个补码数实现加、减运算时，若最高数值位向符号位的进位值 C_{n-1} 与符号位产生的进位输出值 C_n 不相同，表明加减运算产生了溢出OVR；

• 可以表示为：

$$\text{OVR} = C_{n-1} \oplus C_n$$

OVR=1表示结果有溢出，OVR=0表示结果正确。

- 在例1中，求 $[X+Y]_{\text{补}}$ 时： $\text{OVR} = C_{n-1} \oplus C_n = 1 \oplus 1 = 0$ ，结果正确。
- 在例2中，求 $[X+Y]_{\text{补}}$ 时： $\text{OVR} = C_{n-1} \oplus C_n = 1 \oplus 0 = 1$ ，结果溢出。
- 在例3中，求 $[X-Y]_{\text{补}}$ 时： $\text{OVR} = C_{n-1} \oplus C_n = 0 \oplus 1 = 1$ ，结果溢出。

③ 常用**双符号位**方法来判别加、减法运算是否有溢出。

$$\text{OVR} = \bar{S}_{n+1}S_n + S_{n+1}\bar{S}_n = \mathbf{S_{n+1} \oplus S_n}$$

- 两个正数双符号位的运算为**00+00=00**时，**结果不溢出**；
 - 两个正数双符号位的运算为**00+00+1=01**时，**结果上溢**；
 - 两个负数的双符号位运算为 **(11+11+1) MOD 4 =11**时，**结果不溢出**；
 - 两个负数的双符号位的运算为 **(11+11) MOD 4=10**时，**结果下溢**。
- ▲ 采用模4补码运算，其运算结果的**两个符号位不一致**时，**产生溢出**。

4.3 定点乘法运算及实现

◆ 乘法运算的实现方法

- 用软件实现乘法运算;
- 在加法器的基础上增加一些硬件实现乘法运算;
- 设置专用硬件乘法器实现乘法运算。

◆ 常规的乘法运算方法

- 笔---纸乘法方法;
 - 原码乘法;
 - 带符号位运算的补码乘法;
- ## ◆ 用组合逻辑线路构成的阵列乘法器。

4.3.1 原码乘法

1. 原码一位乘

- ◆ 用原码实现乘法运算时，符号位与数值位是分开计算的；
- 设： $[X]_{\text{原}} = x_n x_{n-1} \dots x_1 x_0$ ， $[Y]_{\text{原}} = y_n y_{n-1} \dots y_1 y_0$
(其中 x_n 、 y_n 分别为它们的符号位)
- 若 $[X \times Y]_{\text{原}} = z_{2n} z_{2n-1} \dots z_1 z_0$ (z_{2n} 为结果之符号位)
则 $z_{2n} = x_n \oplus y_n$
- $z_{2n-1} \dots z_1 z_0 = (x_{n-1} \dots x_1 x_0) \times (y_{n-1} \dots y_1 y_0)$ 类似两个无符号数相乘。

◆ 笔---纸乘法方法

▲ 例1. $X=1011, Y=1101$,
 $X \times Y$ 的笔---纸乘法过程:

1011	被乘数 $X=x_3x_2x_1x_0=1011$
$\times 1101$	乘 数 $Y=y_3y_2y_1y_0=1101$
<hr/>	

1011	$X \times y_0 \times 2^0$
------	---------------------------

0000	$X \times y_1 \times 2^1$
------	---------------------------

1011	$X \times y_2 \times 2^2$
------	---------------------------

$+ 1011$	$X \times y_3 \times 2^3$
----------	---------------------------

<hr/>	
10001111	

$$X * Y = \sum_{i=0}^3 (X * Y_i * 2^i) = 10001111$$

• 因此 $X \times Y = 10001111$

◆ 就笔---纸乘法方法，为提高效率而采取的改进措施

- ① 每将乘数 Y 的一位乘以被乘数得 $X \times y_i$ 后，就将该结果与前面所得的结果累加，得到部分积 P_i ；
- ② 将部分积 P_i 右移一位与 $X \times y_i$ 相加；加法运算始终对部分积中的高 n 位进行；
- ③ 对乘数中“1”的位执行加法和右移运算，对“0”的位只执行右移运算，而不执行加法运算；

▲ 上述乘法运算可以归结为循环地计算
下列算式：

- 设 $P_0=0$

$$P_1 = 2^{-1} (P_0 + X \times y_0)$$

$$P_2 = 2^{-1} (P_1 + X \times y_1)$$

$$P_{i+1} = 2^{-1} (P_i + X \times y_i) \quad (i=0,1,2,3, \dots, n-1)$$

.....

$$P_n = 2^{-1} (P_{n-1} + X \times y_{n-1})$$

- 显然, $X \times Y = P_n$

▲ 对于两个 n 位无符号数乘法的一种可行的算法：

- 1) 置计数器为 n ;
- 2) 清除 $2n$ 位部分积寄存器;
- 3) 检查乘数最右位（初始时为最低位），若为“1”，加被乘数到部分积高 n 位中;
- 4) 将部分积右移一位;
- 5) 将乘数右移一位;
- 6) 计数器减1，结果不为0，则从3)开始重新执行；若结果为0，则从部分积寄存器读出乘积。

▲ 实现这种方法的二个定点数乘法的逻辑电路框图

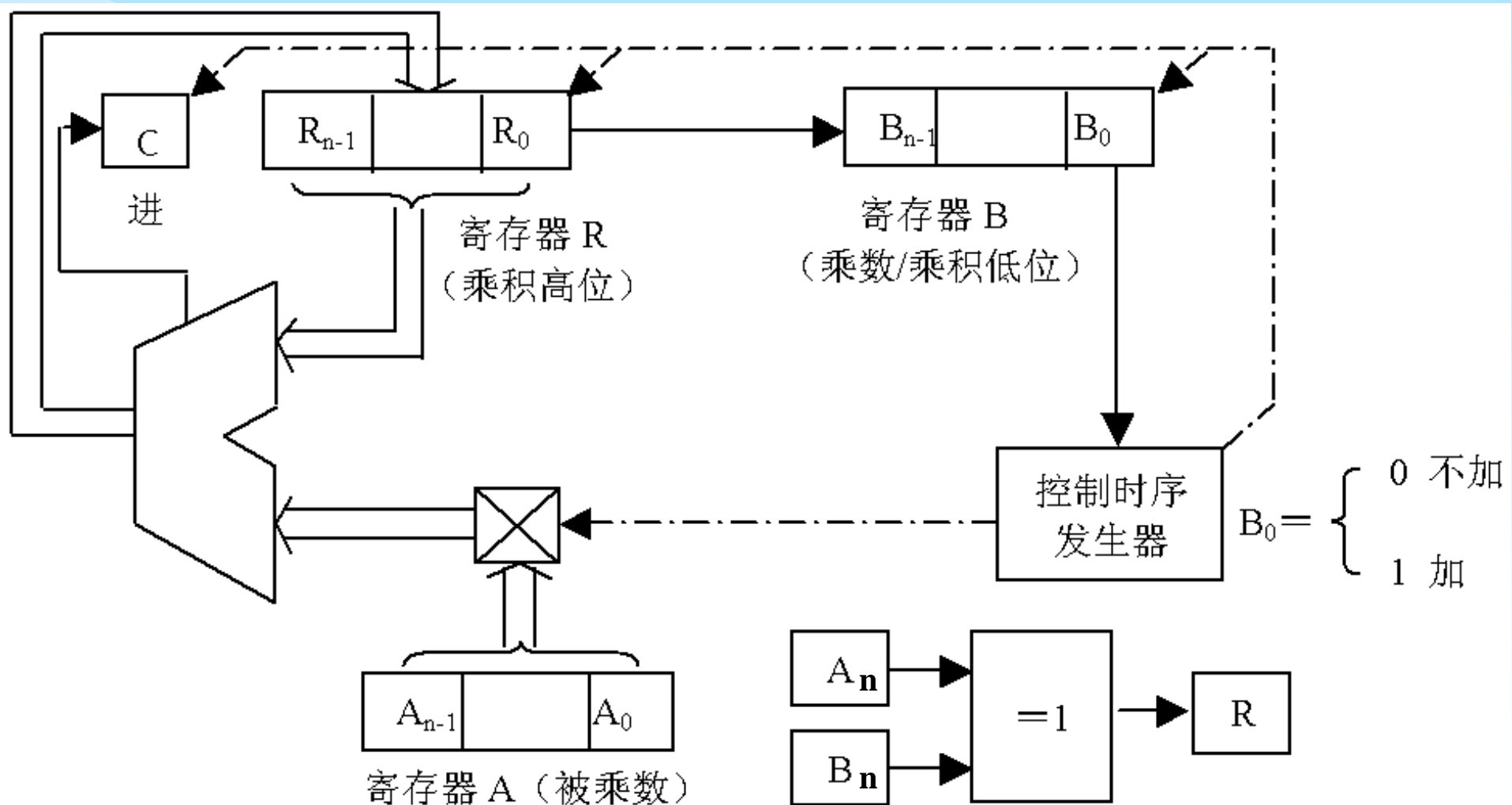


图4.18 原码一位乘原理图

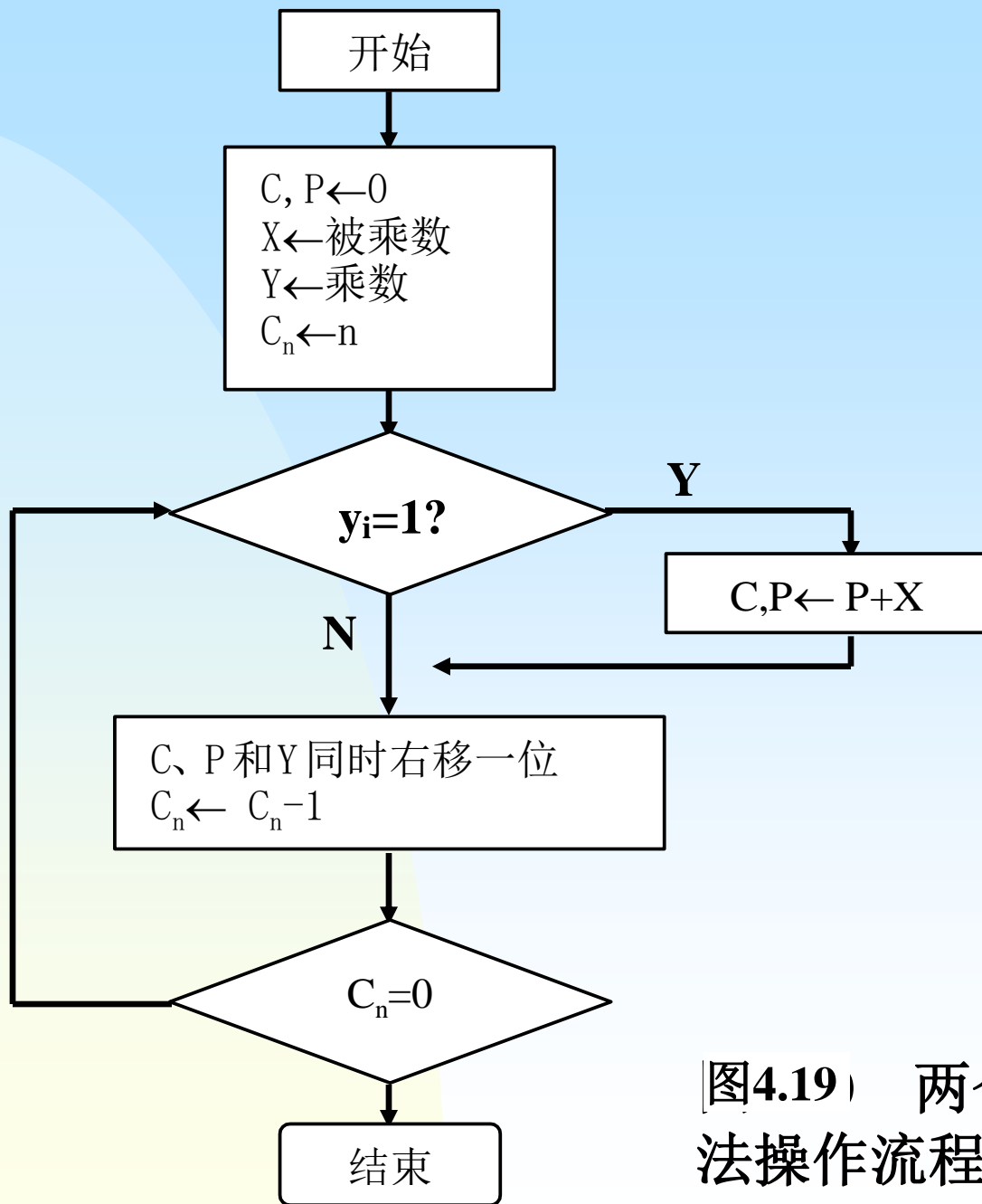


图4.19) 两个定点小数的乘法操作流程

▲ 例1. 已知 $[X]_{\text{原}} = 01101$, $[Y]_{\text{原}} = 01011$,

• 若 $[X \times Y]_{\text{原}} = z_8 z_7 \dots z_0$

则 $z_8 = 0 \oplus 0 = 0$

• $z_7 \dots z_0 = 1101 \times 1011$ 的计算采用上述乘法流程, 实现的具体过程如下:

C	P	Y	说明
0	0000	1011	开始, 设 $P_0 = 0$
<hr/>			
	+ 1101		$y_0 = 1$, + X
0	<hr/> 1101		
			C, P 和 Y 同时右移一位
0	0110	1 101	得 P_1
<hr/>			

0	0110	1	101	得P ₁
	+ 1101			y ₁ =1, + X
1	0011			C, P 和Y同时右移一位
0	1001	11	10	得P ₂
				y ₂ =0, 不作加法
				C, P 和Y同时右移一位
0	0100	111	1	得P ₃
	+ 1101			y ₃ =1, + X
1	0001			C, P 和Y同时右移一位
0	1000	1111		得P ₄

• $z_7 \dots z_0 = 10001111$

• $[X \times Y]_{\text{原}} = z_8 z_7 \dots z_0 = 0 \ 10001111 = 0 \ 010001111$

2. 原码二位乘

◆ 原码二位乘法的思想

- 为提高乘法的速度，一步求出**两位的部分积**。
- 采用原码二位乘法，只需增加少量的逻辑线路，就可以将乘法的速度提高一倍。

◆ 在乘法中，乘数的每两位有四种可能的组合：

$$y_{i+1}y_i = \mathbf{00} \text{ — } P_{i+1} = 2^{-2} (P_i + \mathbf{0} * X)$$

$$y_{i+1}y_i = \mathbf{01} \text{ — } P_{i+1} = 2^{-2} (P_i + \mathbf{1} * X)$$

$$y_{i+1}y_i = \mathbf{10} \text{ — } P_{i+1} = 2^{-2} (P_i + \mathbf{2} * X)$$

$$y_{i+1}y_i = \mathbf{11} \text{ — } P_{i+1} = 2^{-2} (P_i + \mathbf{3} * X)$$

- 实现+3X有两种方法:

- ① 分+X再+2X来进行, 此法速度较低;
- ② 以4X-X来代替3X运算, 在本次运算中只执行-X, 而+4X则归并到下一拍执行;

$$P_{i+1}=2^{-2}(P_i+3X)=2^{-2}(P_i-X+4X)=2^{-2}(P_i-X)+X。$$

◆ 用 y_{i+1} 、 y_i 和Cowe三位来控制乘法操作

- 触发器Cowe用来记录是否欠下+X, 若是, 则1→Cowe。

◆ 原码两位乘法运算规则

表 4.3 原码两位乘法运算规则

y_i	y_{i-1}	Cowe	操 作		迭代公式
0	0	0		$0 \rightarrow \text{Cowe}$	$2^{-2}(P_i)$
0	0	1	$+X$	$0 \rightarrow \text{Cowe}$	$2^{-2}(P_i + X)$
0	1	0	$+X$	$0 \rightarrow \text{Cowe}$	$2^{-2}(P_i + X)$
0	1	1	$+2X$	$0 \rightarrow \text{Cowe}$	$2^{-2}(P_i + 2X)$
1	0	0	$+2X$	$0 \rightarrow \text{Cowe}$	$2^{-2}(P_i + 2X)$
1	0	1	$-X$	$1 \rightarrow \text{Cowe}$	$2^{-2}(P_i - X)$
1	1	0	$-X$	$1 \rightarrow \text{Cowe}$	$2^{-2}(P_i - X)$
1	1	1		$1 \rightarrow \text{Cowe}$	$2^{-2}(P_i)$

◆ 对于原码两位乘算法的说明：

1) 乘积符号与其绝对值分别处理，即

$$|Z|=|X| \times |Y| = (x_{n-1} \cdots x_0) \times (y_{n-1} \cdots y_0)$$

$$z_n = x_n \oplus y_n$$

2) 部分积与被乘数采用三个符号位；

3) 在实际运算中 $-X$ 操作以 $+[-X]_{\text{补}}$ 完成；

4) 每次按表3.3判断当前两位乘数及上一次“欠帐”情况，以决定做什么操作。

◆ 原码两位乘法运算过程举例

例1: 已知 $[X]_{\text{原}} = 0111001$, $[Y]_{\text{原}} = 0100111$,

$[|X|]_{\text{补}} = 0111001$, $[-|X|]_{\text{补}} = 1000111$;

• 若 $[X \times Y]_{\text{原}} = z_{12}z_{11} \dots z_0$

则 $z_{12} = 0 \oplus 0 = 0$

• $z_{11} \dots z_0 = 111001 * 100111$ 具体过程:

P	Y	T	说明
000 000000	1001 11	0	开始, $P_0=0$, $T=0$
<hr/>			
+111 000111			$y_1y_0T=110$, $-X$, $T=1$
<hr/>			
111 000111			P和Y同时右移2位

111 000111					P和Y同时右移2位
111 110001	11	1001	1		得P ₁
<hr/>					
+ 001 110010					y ₃ y ₂ T=011 , +2X ,T=0
001 100011					P和Y同时右移2位
000 011000	1111	10	0		得P ₂
<hr/>					
+ 001 110010					y ₅ y ₄ T=100 , +2X ,T=0
010 001010					P和Y同时右移2位
000 100010	101111	0			得P ₃
<hr/>					

• $z_{11} \dots z_0 = 100010101111$

• 因此 $[X \times Y]_{\text{原}} = 0\ 100010101111 = 0\ 0100010101111$

4.3.2 补码乘法

1. 补码一位乘

◆ 考查两个补码乘法运算的例子

例1: 已知 $X = 0.1011$, $Y = 0.0001$

$$[X]_{\text{补}} = 0.1011, [Y]_{\text{补}} = 0.0001$$

$$[X \times Y]_{\text{补}} = 0.00001011$$

$$[X]_{\text{补}} \times [Y]_{\text{补}} = 0.00001011$$

- 显然, $[X \times Y]_{\text{补}} = [X]_{\text{补}} \times [Y]_{\text{补}}$

例2: 已知 $X=0.1011$, $Y= - 0.0001$

$$[X]_{\text{补}} = 0.1011, [Y]_{\text{补}} = 1.1111$$

$$[X \times Y]_{\text{补}} = 1.1110101$$

$$[X]_{\text{补}} \times [Y]_{\text{补}} = 1.01010101$$

- 显然, $[X \times Y]_{\text{补}} \neq [X]_{\text{补}} \times [Y]_{\text{补}}$

▲ 对两个正数来说, 它们补码的乘积等于它们乘积的补码。若乘数是负数时, 这种情况就不成立了。

◆ Booth（布斯）乘法

▲ A.D.Booth算法思想：

- 相乘二数用补码表示，它们的符号位与数值位一起参与乘法运算过程，得出用补码表示的乘法结果。

▲ Booth算法推导：

- 已知 $[X]_{\text{补}} = x_n x_{n-1} \cdots x_0$ ， $[Y]_{\text{补}} = y_n y_{n-1} \cdots y_0$ ；
- 根据补码定义（定点整数）：

$$[Y]_{\text{补}} = 2^{n+1} + Y = 2^{n+1} y_n + Y \quad y_n = \begin{cases} 0, & \text{当 } Y \geq 0 \\ 1, & \text{当 } Y < 0 \end{cases}$$

- 可得出其真值:

$$Y = [Y]_{\text{补}} - 2^{n+1}y_n$$

$$\begin{aligned} X \times Y &= X \times \{[Y]_{\text{补}} - 2^{n+1}y_n\} \\ &= X\{y_n 2^n + y_{n-1} 2^{n-1} + \cdots + y_1 2^1 + y_0 2^0 - 2^{n+1}y_n\} \\ &= X\{-y_n 2^n + y_{n-1} 2^{n-1} + \cdots + y_1 2^1 + y_0 2^0\} \\ &= 2^n (y_{n-1} - y_n) X + 2^{n-1} (y_{n-2} - y_{n-1}) X + \cdots \\ &\quad + 2^1 (y_0 - y_1) X + 2^0 (0 - y_0) X \end{aligned}$$

- y_{-1} 为增设的一个附加位, 初值为0;

$$\begin{aligned} [X \times Y]_{\text{补}} &= [2^n (y_{n-1} - y_n) X + 2^{n-1} (y_{n-2} - y_{n-1}) X + \cdots \\ &\quad + 2^1 (y_0 - y_1) X + 2^0 (y_{-1} - y_0) X]_{\text{补}} \end{aligned}$$

$$\text{令 } [X \times Y]_{\text{补}} = [X \times Y]'_{\text{补}} \times 2^n$$

$$[X \times Y]'_{\text{补}} = [(y_{n-1} - y_n)X + 2^{-1}(y_{n-2} - y_{n-1})X + \dots + 2^{-(n-1)}(y_0 - y_1)X + 2^{-n}(y_{-1} - y_0)X]_{\text{补}}$$

• 得到如下递推公式

令 $[P_0]_{\text{补}} = 0$, 有:

$$[P_1]_{\text{补}} = [2^{-1}(P_0 + (y_{-1} - y_0) \times X)]_{\text{补}}$$

$$[P_2]_{\text{补}} = [2^{-1}(P_1 + (y_0 - y_1) \times X)]_{\text{补}}$$

$$\vdots$$

$$[P_i]_{\text{补}} = [2^{-1}(P_{i-1} + (y_{i-2} - y_{i-1}) \times X)]_{\text{补}} \quad (i=1 \sim n)$$

$$\vdots$$

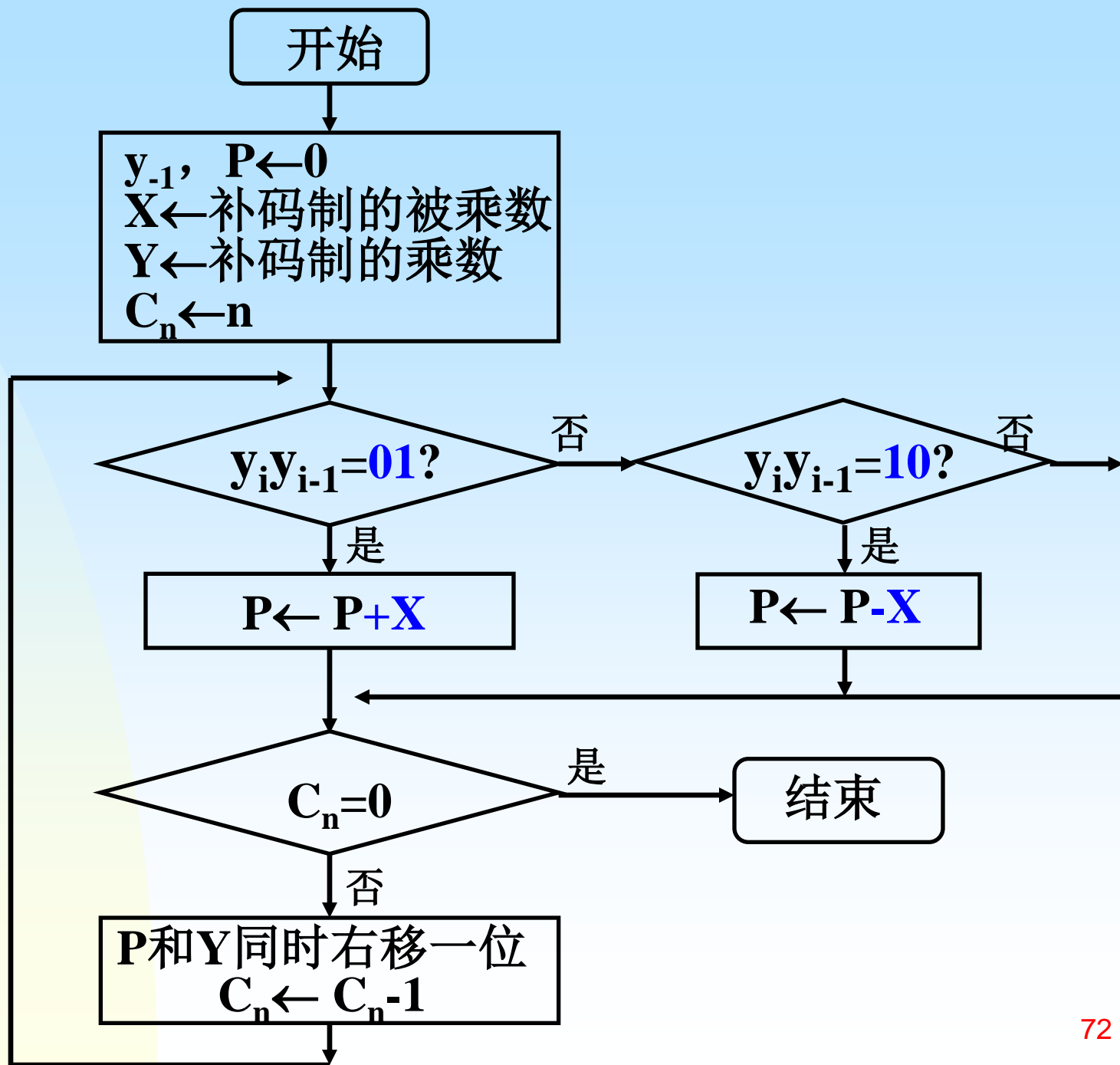
$$[P_n]_{\text{补}} = [2^{-1}(P_{n-1} + (y_{n-2} - y_{n-1}) \times X)]_{\text{补}}$$

$$[X * Y]'_{\text{补}} = [P_{n+1}]_{\text{补}} = [P_n + (y_{n-1} - y_n) \times X]_{\text{补}}$$

- $(y_{i-1}-y_i)X$ 实际上并不做乘法，只要比较相邻两位乘数以决定 $+X$ 、 $-X$ 或 $+0$ 。
- 在计算机中，对于定点整数，只要认定小数点在乘积之末，相当于将小数点右移 n 位。
- 对乘数的连续两位 y_i 和 y_{i-1} 进行判断

若 $y_i y_{i-1} = 01$,	则 $[P_{i+1}]_{\text{补}} = [2^{-1} (P_i + X)]_{\text{补}}$
若 $y_i y_{i-1} = 10$,	则 $[P_{i+1}]_{\text{补}} = [2^{-1} (P_i - X)]_{\text{补}}$
若 $y_i y_{i-1} = 00$ 或 11 ,	则 $[P_{i+1}]_{\text{补}} = [2^{-1} P_i]_{\text{补}}$
- 一个补码数据的右移是连同符号位右移，且最高位补充符号位的值。

图 4.20 布斯乘法运算流程图



▲ 例3: 已知 $[X]_{\text{补}}=01101$, $[Y]_{\text{补}}=10110$,
 $[-X]_{\text{补}}=10011$ 。

- 用布斯乘法计算 $[X \times Y]_{\text{补}}$ 的过程如下

P	Y	y_{-1}	说明
00 0000	10110 0	0	开始, 设 $y_{-1}=0$, $[P_0]_{\text{补}}=0$
00 0000	0 1011 0	0	$y_0 y_{-1} = 00$, P、Y同时右移一位 得 $[P_1]_{\text{补}}$
+11 0011			$y_1 y_0 = 10$, $+[-X]_{\text{补}}$
11 0011			P、Y同时右移一位
11 1001	10 101 1	1	得 $[P_2]_{\text{补}}$
11 1100	110 10 1	1	$y_2 y_1 = 11$, P、Y同时右移一位 得 $[P_3]_{\text{补}}$

11 1100	110	10 1	得 $[P_3]_{\text{补}}$
+00 1101			$y_3y_2=01$, $+ [X]_{\text{补}}$
00 1001			P、Y同时右移一位
00 0100	1110	1 0	得 $[P_4]_{\text{补}}$
+11 0011			$y_4y_3=10$, $+ [-X]_{\text{补}}$
11 0111	1110	1	最后一次不右移

- 因此, $[X \times Y]_{\text{补}} = 101111110 = 1\ 101111110$
- ▲ 布斯乘法的算法过程为 $n+1$ 次的“判断—加减—右移”的循环, 判断的次数为 $n+1$ 次, 右移的次数为 n 次。
- ▲ 在布斯乘法中, 遇到连续的“1”或连续的“0”时, 是跳过加法运算, 直接实现右移操作的, 运算效率高。

2. 补码二位乘

◆ 补码二位乘法可以用布斯乘法过程来推导

- $[P_{i+1}]_{\text{补}} = 2^{-1} \{ [P_i]_{\text{补}} + (y_{i-1} - y_i) \times [X]_{\text{补}} \}$

- $[P_{i+2}]_{\text{补}} = 2^{-1} \{ [P_{i+1}]_{\text{补}} + (y_i - y_{i+1}) \times [X]_{\text{补}} \}$

- $[P_{i+2}]_{\text{补}} = 2^{-2} \{ [P_i]_{\text{补}} + (y_{i-1} + y_i - 2y_{i+1}) \times [X]_{\text{补}} \}$

▲ 根据乘数的两位代码 y_{i+1} 和 y_i 以及右邻位 y_{i-1} 的值的组合作为判断依据，跳过 $[P_{i+1}]_{\text{补}}$ 步，即从 $[P_i]_{\text{补}}$ 直接求得 $[P_{i+2}]_{\text{补}}$ 。

表 4.4 乘数3位代码组合构成的判断规则

乘数代码对		右邻位	加减判断规则	操 作
y_{i+1}	y_i	y_{i-1}	$y_{i-1}+y_i-2y_{i+1}$	$[P_{i+2}]_{\text{补}}$
0	0	0	0	$2^{-2}[P_i]_{\text{补}}$
0	0	1	$+ [X]_{\text{补}}$	$2^{-2}\{[P_i]_{\text{补}} + [X]_{\text{补}}\}$
0	1	0	$+ [X]_{\text{补}}$	$2^{-2}\{[P_i]_{\text{补}} + [X]_{\text{补}}\}$
0	1	1	$+ 2[X]_{\text{补}}$	$2^{-2}\{[P_i]_{\text{补}} + 2[X]_{\text{补}}\}$
1	0	0	$+ 2[-X]_{\text{补}}$	$2^{-2}\{[P_i]_{\text{补}} + 2[-X]_{\text{补}}\}$
1	0	1	$+ [-X]_{\text{补}}$	$2^{-2}\{[P_i]_{\text{补}} + [-X]_{\text{补}}\}$
1	1	0	$+ [-X]_{\text{补}}$	$2^{-2}\{[P_i]_{\text{补}} + [-X]_{\text{补}}\}$
1	1	1	0	$2^{-2}[P_i]_{\text{补}}$

◆ 设乘数 $[Y]_{\text{补}} = y_n y_{n-1} \dots y_1 y_0$ ，导出补码二位乘法中的计算量。

(1) 当 n 为偶数时，乘法运算过程中的总循环次数为 $n/2+1$ ；最后一次不右移。

(2) 当 n 为奇数时，为使方法统一，对定点整数可在数值位最高位前补一位（正数补“0”负数补“1”），定点小数可在末位补一位“0”，凑成偶数。

◆ 例4：已知 $[X]_{\text{补}} = 00011$ ， $[Y]_{\text{补}} = 11010$ ； $[-X]_{\text{补}} = 11101$ 。用补码二位乘法计算 $[X \times Y]_{\text{补}}$ 的过程。

P	Y	y ₋₁	说明
000 0000	11010	0	开始, 设y ₋₁ =0, [P ₀] _补 =0
+111 1010	111 1010		y ₁ y ₀ y ₋₁ = 100 , +2[-X] _补
111 1010			P和Y同时右移二位
111 1110	10	110 1	得[P ₁] _补
+111 1101	111 1011		y ₃ y ₂ y ₁ = 101 , +[-X] _补
111 1011			P和Y同时右移二位
111 1110	1110	1 1	得[P ₂] _补
			y ₄ y ₄ y ₃ = 111 , 不做加法
			最后一次不右移

- $[X \times Y]_{\text{补}} = 111101110 = \text{1111101110}$

4.3.3 阵列乘法器

- ◆ 实现上述执行过程的**阵列结构**的乘法器，称为阵列乘法器(Array Multiplier)
- 设两个四位二进制数相乘 $X \times Y$ ，其中 $X = x_3x_2x_1x_0$ ， $Y = y_3y_2y_1y_0$ ； X 和 Y 都是**无符号定点数**。

				x_3	x_2	x_1	x_0
			\times	y_3	y_2	y_1	y_0
				$x_3 y_0$	$x_2 y_0$	$x_1 y_0$	$x_0 y_0$
			+	$x_3 y_1$	$x_2 y_1$	$x_1 y_1$	$x_0 y_1$
				$x_3 y_2$	$x_2 y_2$	$x_1 y_2$	$x_0 y_2$
				$x_3 y_3$	$x_2 y_3$	$x_1 y_3$	$x_0 y_3$
乘积 Z:	z_6	z_5	z_4	z_3	z_2	z_1	z_0

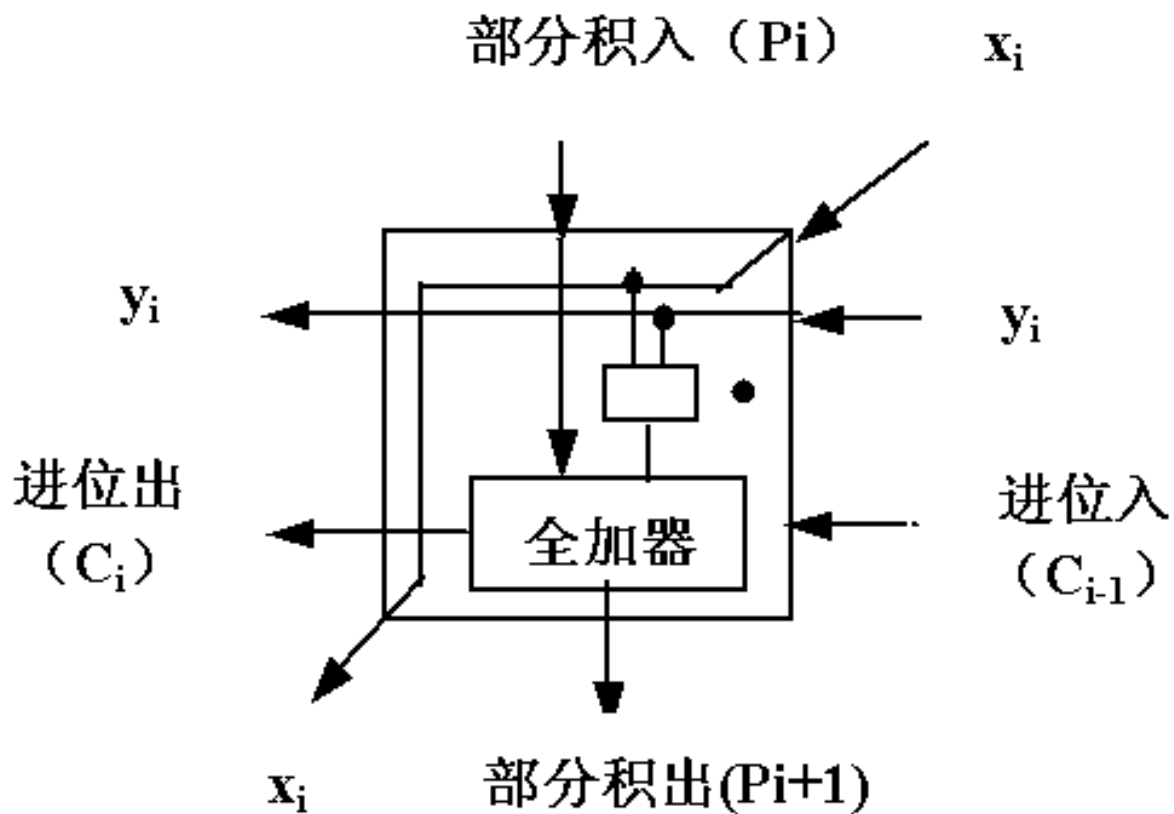


图4.21 阵列乘法器单元电路

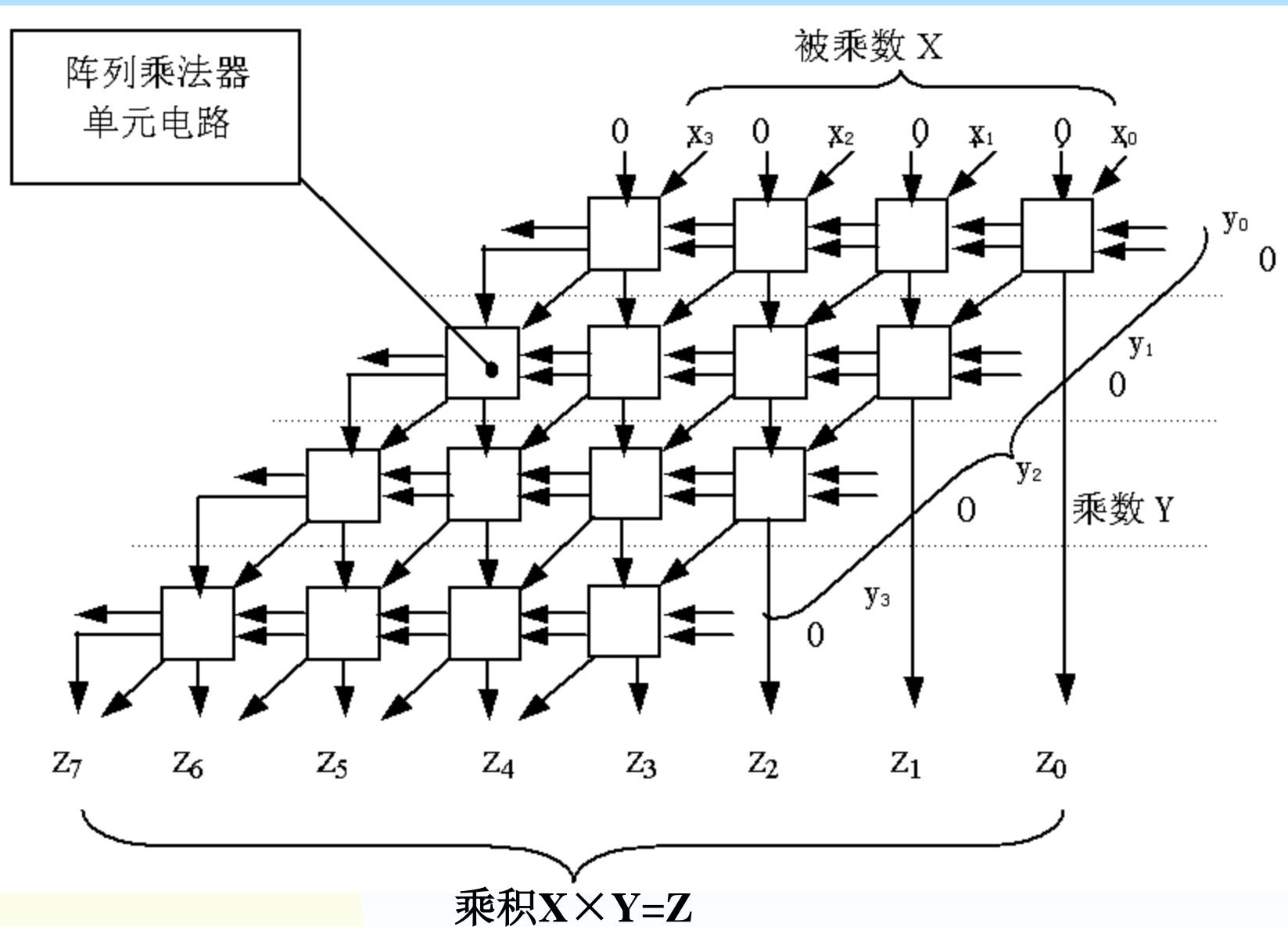


图4.22 直接实现定点数绝对值相乘的阵列乘法器

▲ 存储进位加法器（Carry Save Adder），简称CSA

$$S_i = \bar{X}_i \bar{Y}_i Z_i + \bar{X}_i Y_i \bar{Z}_i + X_i \bar{Y}_i \bar{Z}_i + X_i Y_i Z_i$$

$$C_i = X_i Y_i + Y_i Z_i + X_i Z_i$$

- 将进位信息暂时保留，留待下一级加法处理。

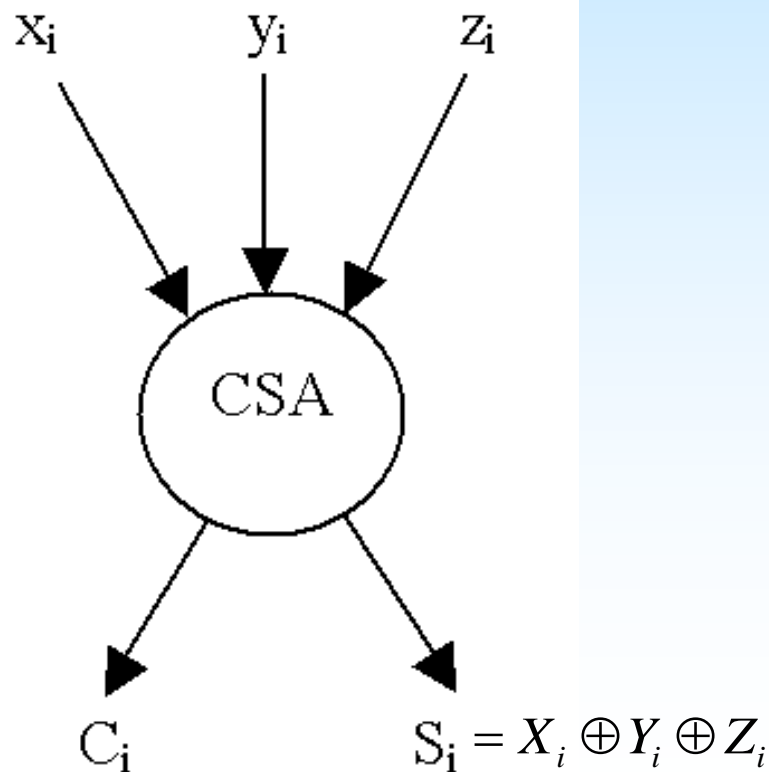


图4.23 伪加器单元

▲ 为解决进位问题，采用下图所示的阵列乘法器

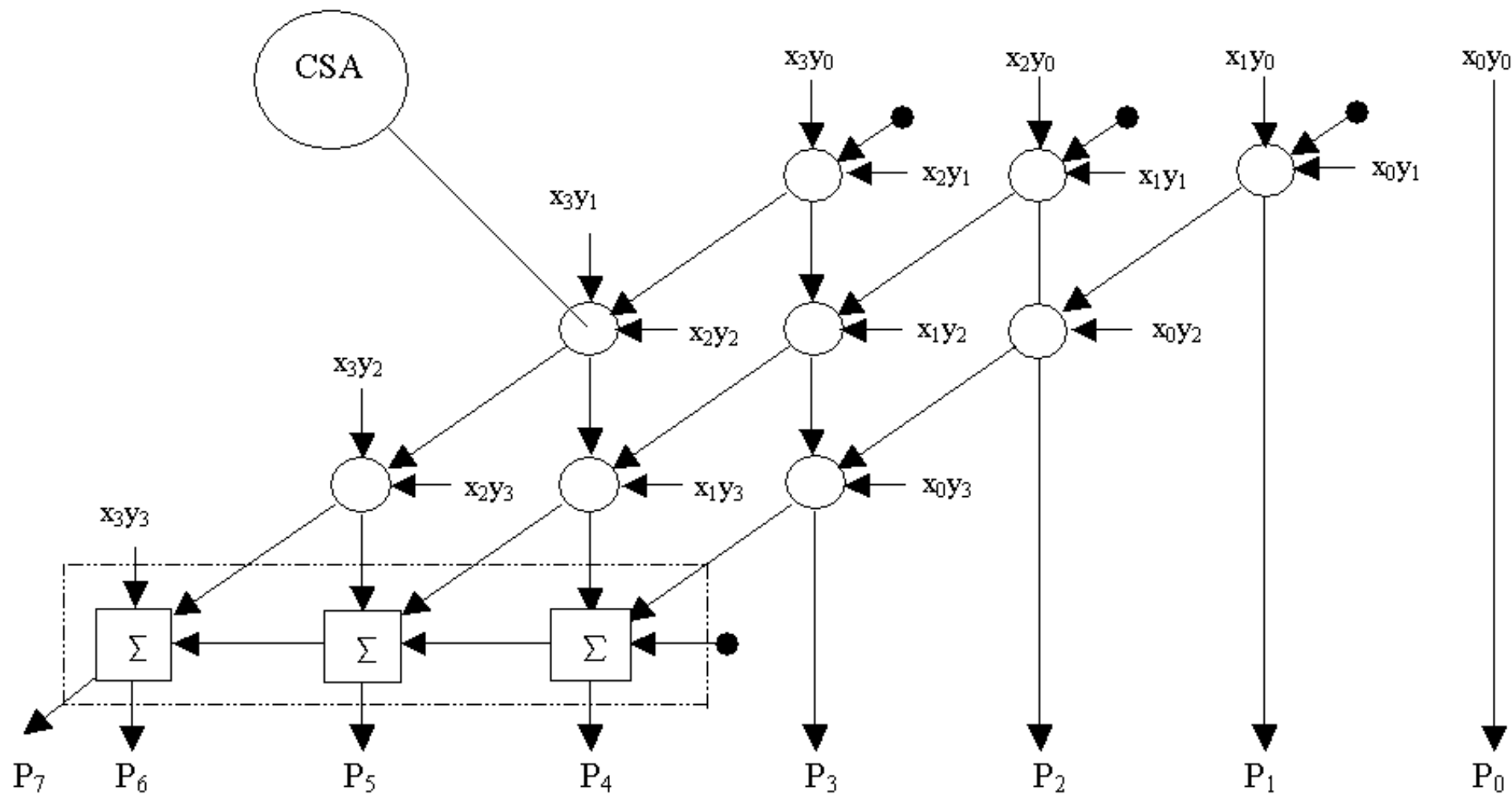


图4.24 4×4 阵列乘法器原理图

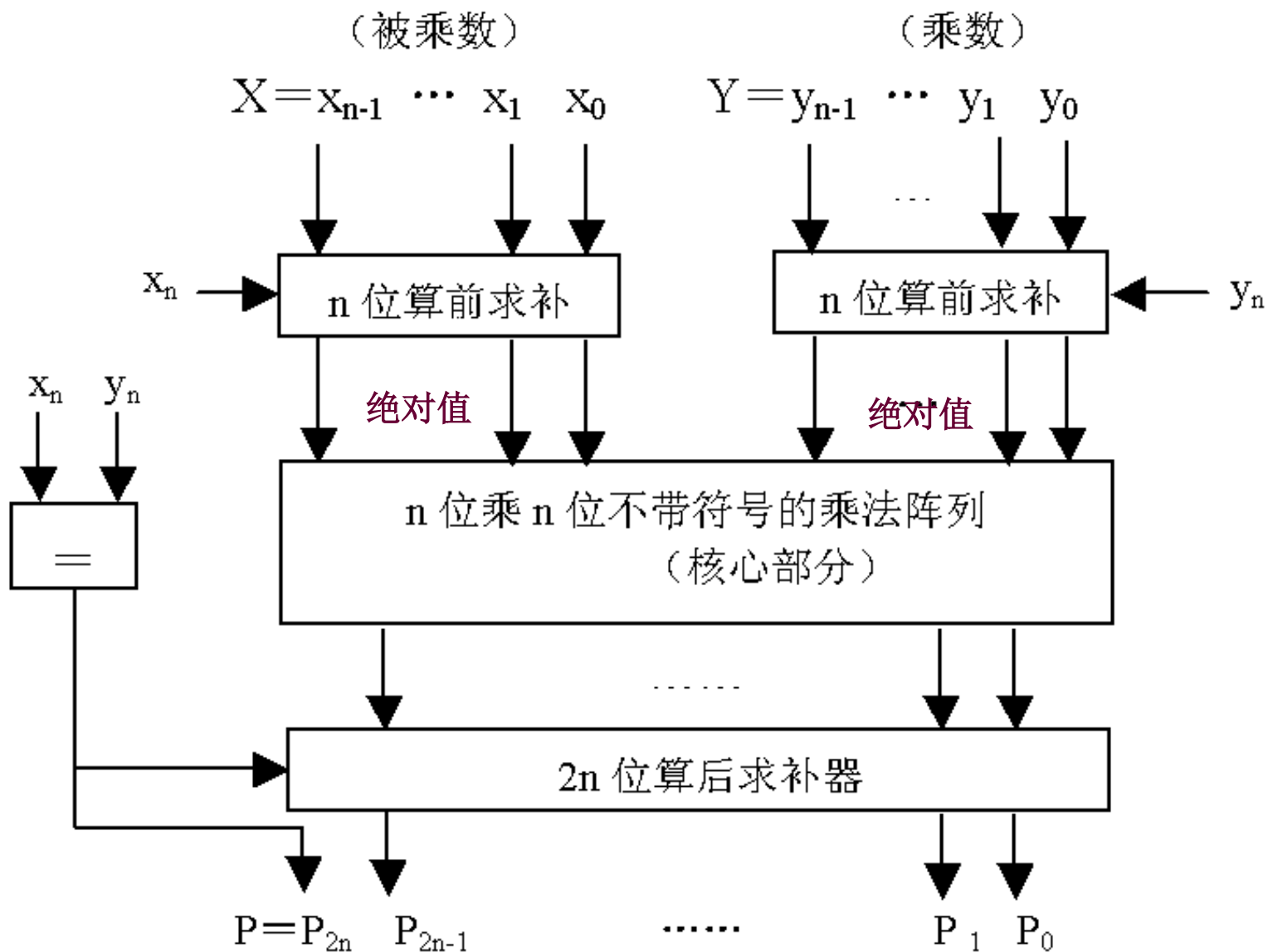


图4.25 带求补级的阵列乘法器框图

◆ 阵列乘法器的特点

- 组织结构规则性强，标准化程度高；适合用超大规模集成电路实现，且能获得很高的运算速度。
- 集成电路的价格不断下降，阵列乘法器在某些数字系统中(如在信号及数据处理系统中)。
- Booth算法的乘法运算也可以用阵列乘法器的方法实现，但要求的单元更复杂。

4.4 定点除法运算及实现

- ◆ 在定点运算中，除法的算法主要有比较法、恢复余数法和
不恢复余数法三种。
- ◆ 定点整数，定点小数除法的差异：
 - ①在
小数除法中，被除数和除数应满足如下条件：
 $0 < |\text{被除数}| < |\text{除数}|$ 。
 - ②在
整数除法中，被除数和除数应满足如下条件：
 $0 < |\text{除数}| \leq |\text{被除数}|$ 。
 - ③定点整数除法允许
双字长的被除数除以单字长的除数，得到单字长的商和单字长的余数。

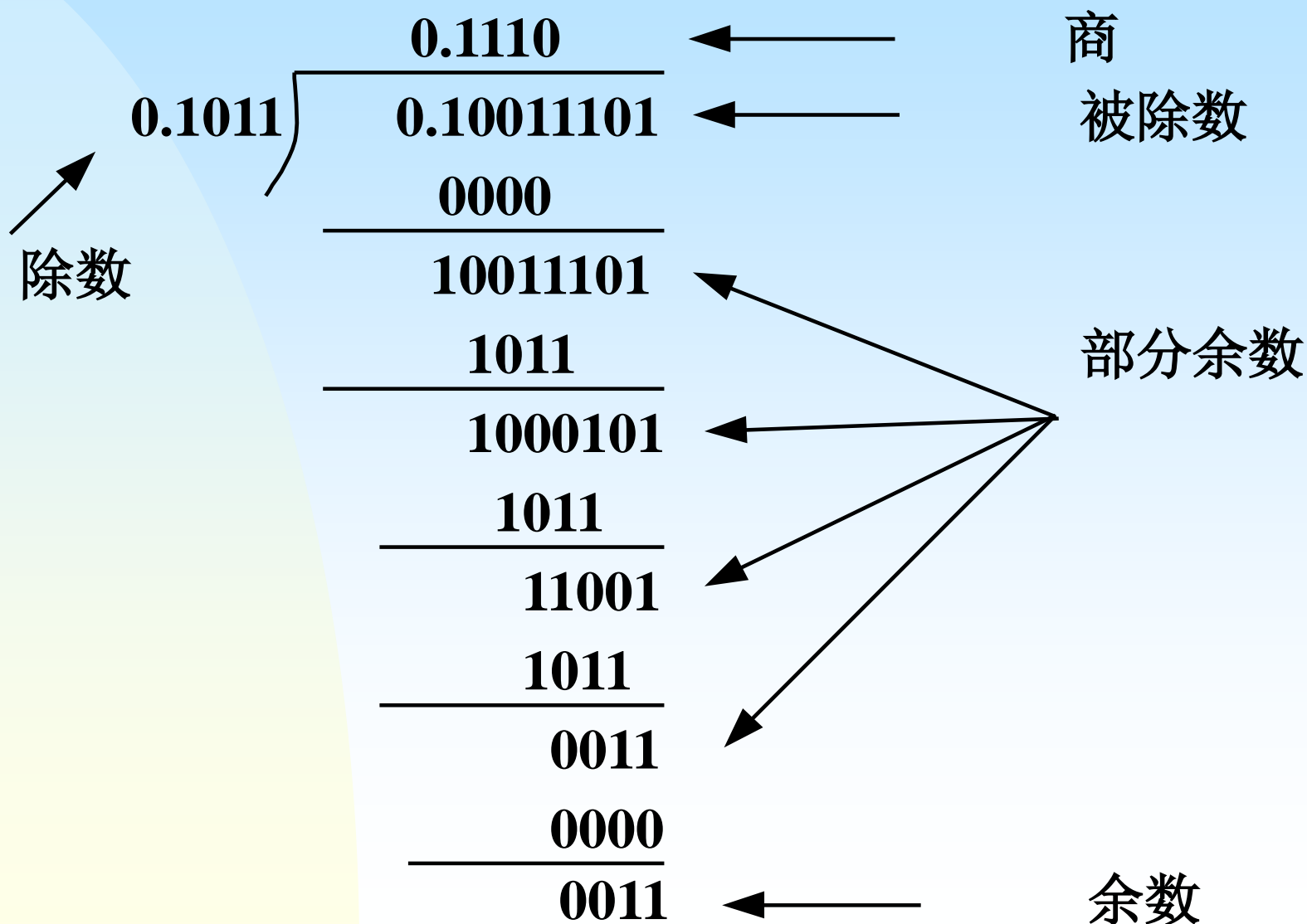
4.4.1 原码除法运算

1. 比较法（笔-纸方法）的除法步骤：

- ① 被除数与除数比较，决定上商；若被除数小，上商0；否则上商1；得到部分余数。
- ② 将除数右移，再与上一步部分余数比较，决定上商，并且求得新的部分余数。
- ③ 重复执行第②步，直到求得的商的位数足够为止。

例1: 已知两正数 $X=0.10011101$,

$Y=0.1011$



- X/Y 的商为0.1110，余数为 0.0011×2^{-4}

▲ 原码一位除法规律

- 原码一位除法运算与原码一位乘法运算一样，要区分符号位和数值位两部分。
- 商的符号为相除两数符号的“异或”值，商的数值为两数的绝对值之商。
- 余数的符号应与被除数同号；所得的商和余数是以原码表示。

- ◆ 计算机中实现二个正数除法时，有几点不同：
- ① 比较运算用减法来实现，由结果的正负来判断两数的大小；结果为正，上商1；结果为负，上商0。
 - ② 减法运算时，加法器中两数的对齐是用部分余数左移实现，并与除数比较；左移出界的部分余数的高位都是0，对运算不会产生任何影响。
 - ③ 每一次上商都是把商写入商值寄存器的最低位，然后部分余数和商一起左移，腾空商寄存器的最低位以备上新的商。
 - ④ 部分余数随商一起左移了 n 位；所以正确的余数应为 $2^{-n} R_n$ 。

- ◆ 采用部分余数减去除数的方法比较两者的大小，当减法结果为负，即上商0时，破坏了部分余数。可采取两种措施。

2. 恢复余数的除法

- ▲ 两个正的定点小数 X 和 Y ， $X=0.x_1x_2\ldots x_n$ ， $Y=0.y_1y_2\ldots y_n$ ，求解 X/Y 的商和余数的方法：

第1步： $R_1 = X - Y$

- 若 $R_1 < 0$ ，则上商 $q_0 = 0$ ，同时恢复余数： $R_1 = R_1 + Y$ 。
- 若 $R_1 \geq 0$ ，则上商 $q_0 = 1$ 。

- q_0 位不是符号位，而是两定点小数相除时的整数部分； $q_0=1$ 时，当作溢出处理。

第2步：若已求得第 i 次的部分余数为 R_i ，则第 $i+1$ 次的部分余数为： $R_{i+1} = 2R_i - Y$

- 若 $R_{i+1} < 0$ ，上商 $q_i = 0$ ，同时恢复余数： $R_{i+1} = R_{i+1} + Y$ 。
- 若 $R_{i+1} \geq 0$ ，则上商 $q_i = 1$ 。

第3步：不断循环执行第2步，直到求得所需位数的商为止。

例1： 已知 $[X]_{\text{原}}=0.1011$ ， $[Y]_{\text{原}}=1.1101$ ；
求 $[X/Y]_{\text{原}}$ 。

- 计算分为符号位和数值位两部分
 - $[X/Y]_{\text{原}}$ 商的符号位： $0 \oplus 1 = 1$
 - $[X/Y]_{\text{原}}$ 商的数值位计算采用恢复余数法。运算中的减法操作用补码加法实现。
 - $[-|Y|]_{\text{补}} = 10011$ 。
- ▲ 分别标识为X和Y运算过程：

部分余数	商	说明
00 1011	0000 □	开始 $R_0=X$
+11 0011		$R_1=X-Y$
11 1110	0000 0	$R_1<0$, 则 $q_0=0$
+00 1101		恢复余数: $R_1=R_1+Y$
00 1011		得 R_1
01 0110	000 0	$2R_1$ (部分余数和商同时左移)
+11 0011		$-Y$
00 1001	000 01	$R_2>0$, 则 $q_1=1$
01 0010	00 01	$2R_2$ (左移)
+11 0011		$-Y$
00 0101	00 011	$R_3>0$, 则 $q_2=1$

00 0101	00 011	$R_3 > 0$, 则 $q_2 = 1$
00 1010	0 011	$2R_3$ (左移)
+11 0011		$-Y$
11 1101	0 0110	$R_4 < 0$, 则 $q_3 = 0$
+00 1101		恢复余数: $R_4 = R_4 + Y$
00 1010	0 0110	得 R_4
01 0100	0110	$2R_4$ (左移)
+11 0011		$-Y$
00 0111	01101	$R_5 > 0$, 则 $q_4 = 1$

- 可见商的数值位为1101
- $[X/Y]_{\text{原}} = 1.1101$, 余数为 $0.0111 * 2^{-4}$ 。

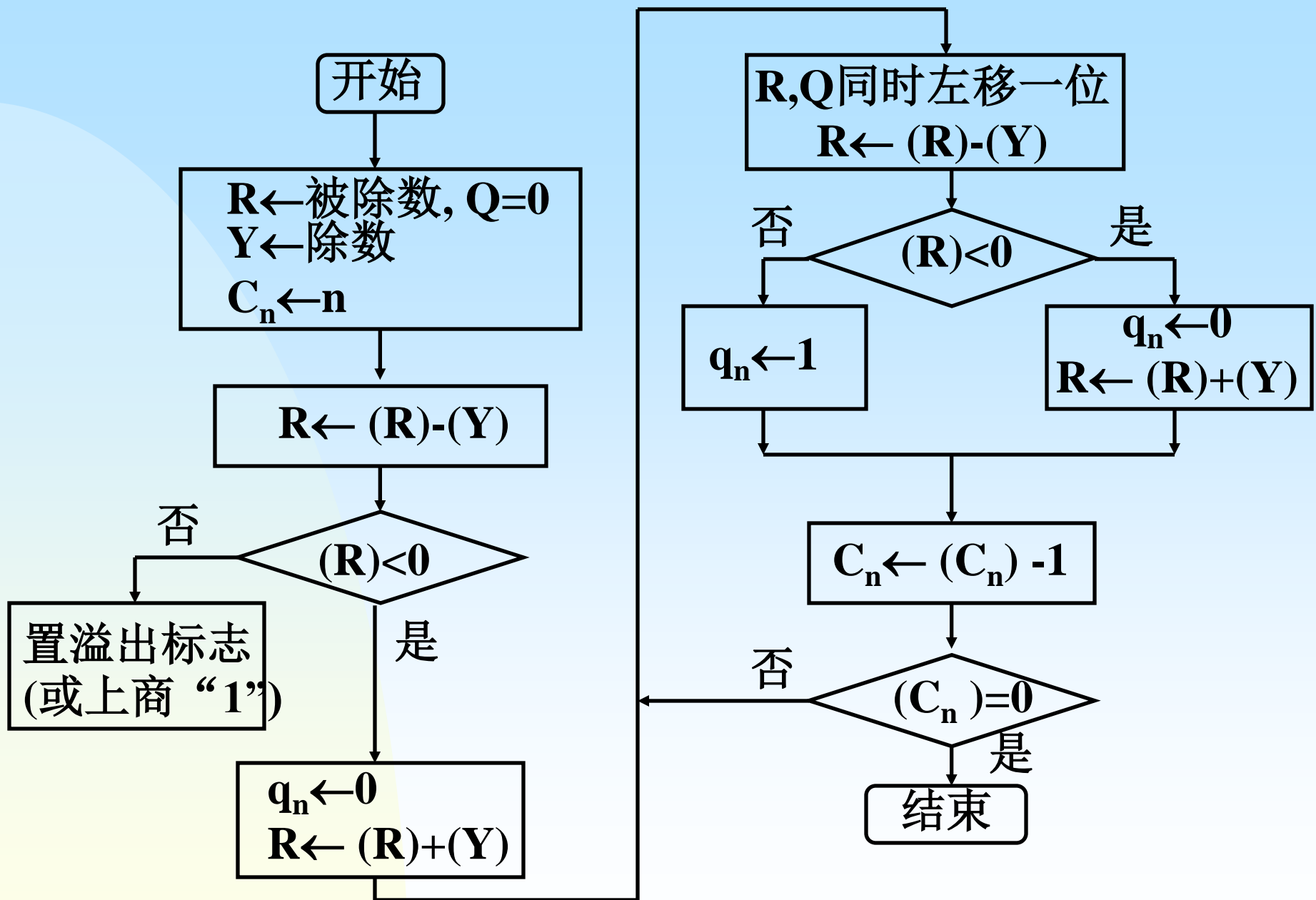


图4.26 恢复余数除法的运算流程图

3. 不恢复余数的除法（加减交替法）

- ▲ 当第 i 次的部分余数为负时，跳过恢复余数的步骤，直接求第 $i+1$ 次的部分余数。
- ▲ 对两个正的定点小数 X 和 Y 采用不恢复余数除法的基本步骤：

第1步： $R_1 = X - Y$

- 若 $R_1 < 0$ ，则上商 $q_0 = 0$ ；
- 若 $R_1 \geq 0$ ，则上商 $q_0 = 1$ ；
- q_0 代表两定点小数相除时的整数部分，当 $q_0 = 1$ 时，将当作溢出处理。

第2步：若已求得部分余数 R_i ，则第 $i+1$ 次的部分余数为：

- 若 $R_i < 0$ ，上商 $q_{i-1} = 0$ ， $R_{i+1} = 2R_i + Y$ ，上一步中多减去的 Y 在这一步中弥补回来(+ $2Y$)；
- 若 $R_i \geq 0$ ，上商 $q_{i-1} = 1$ ， $R_{i+1} = 2R_i - Y$ ，保持原有的除法过程；

第3步：不断循环执行第2步，直到求得所需位数的商为止。

- 结束时，若余数为负值，要执行恢复余数的操作 $R_n = R_n + Y$ 。

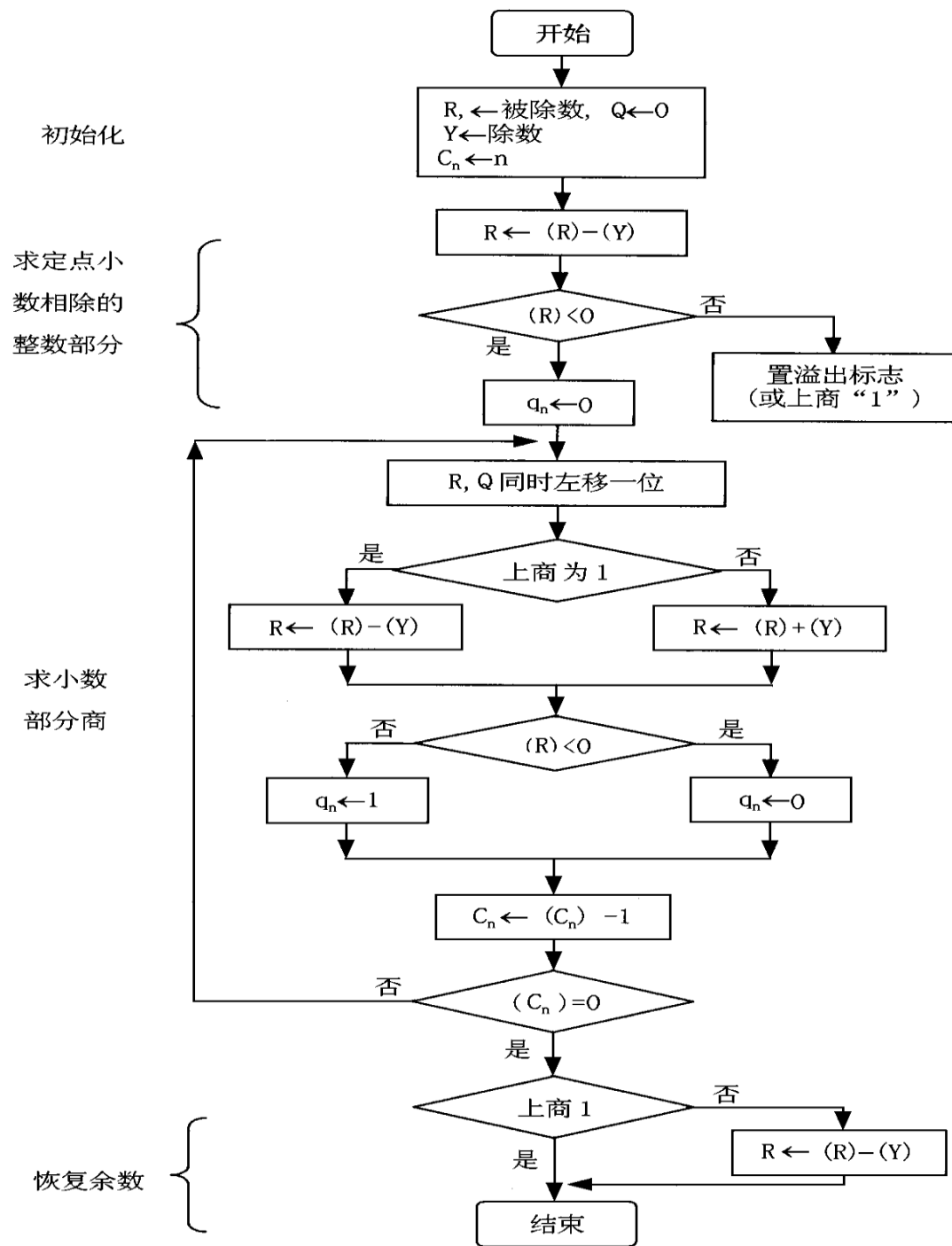


图4.27 实现两正定点数相除的不恢复余数除法运算流程

例1：已知 $[X]_{\text{原}} = 0.1011$ ，
 $[Y]_{\text{原}} = 1.1101$ ；计算 $[X/Y]_{\text{原}}$ 。

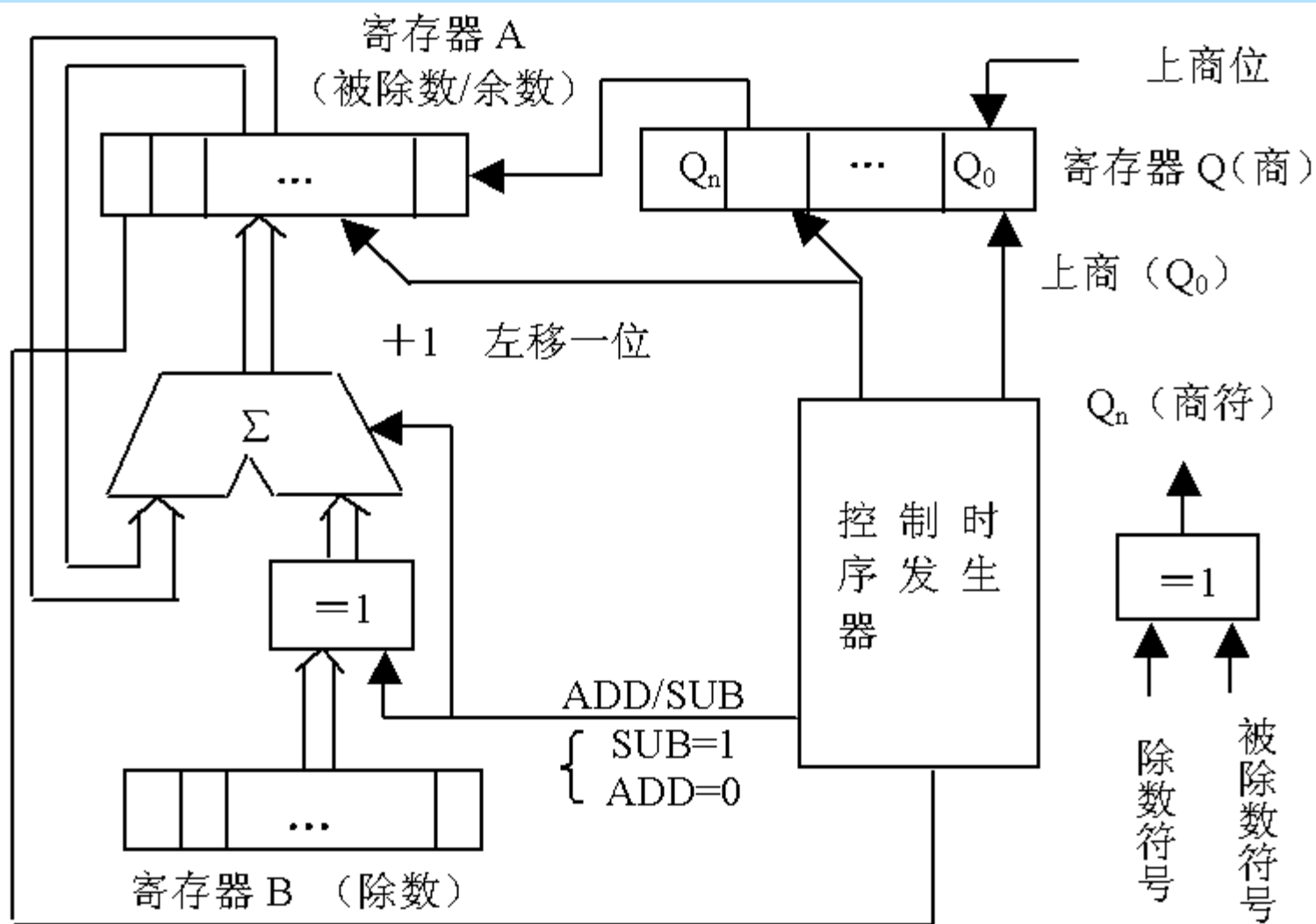
- 计算分为符号位和数值位两部分
- $[X/Y]_{\text{原}}$ 商的符号位： $0 \oplus 1 = 1$
- $[X/Y]_{\text{原}}$ 商的数值位的计算采用不恢复余数的除法
- 参加运算的数据是 $[|X|]_{\text{补}}$ 和 $[|Y|]_{\text{补}}$ 两数，分别标识为 X 和 Y； $[-|Y|]_{\text{补}} = 10011$ 。
- 运算过程如下：

部分余数	商	说明
00 1011	0000	□ 开始 $R_0=X$
+11 0011		$R_1=X-Y$
11 1110	0000 0	$R_1<0$, 则 $q_0=0$
11 1100	000 0	$2R_1$ (部分余数和商同时左移)
+00 1101		+Y
00 1001	000 01	$R_2>0$, 则 $q_1=1$
01 0010	00 01	$2R_2$ (左移)
+11 0011		-Y
00 0101	00 011	$R_3>0$, 则 $q_2=1$

00 0101	00 011	$R_3 > 0$, 则 $q_2 = 1$
00 1010	0 011	$2R_3$ (左移)
+11 0011		$-Y$
11 1101	0 0110	$R_4 < 0$, 则 $q_3 = 0$
11 1010	0 110	$2R_4$ (左移)
+00 1101		$+Y$
00 0111	0 1101	$R_5 > 0$, 则 $q_4 = 1$

- 商的数值位为1101
- $[X/Y]_{\text{原}} = 1.1101$; R_5 与X同号, 余数为 0.0111×2^{-4} 。
- n位数的不恢复余数除法需要n次加法运算; 对恢复余数除法来说, 平均需要 $3n/2$ 次加法运算。

4. 原码除法器的组成



符号位反馈信息

4.4.2 补码除法运算

- (1) 在补码除法中，符号位与数值位同等参与整个除法运算过程，商的符号位在除法运算中产生。
- (2) 部分余数和除数都用带符号位的补码表示时，部分余数与除数是否够除，不能用两数直接相减的方法来判断。

表 4.5 两补码数是否够减的判别方法

部分余数[R] _补	除数[Y] _补	[R] _补 -[Y] _补		[R] _补 + [Y] _补	
		0	1	0	1
0	0	够减 (商1)	不够减(商0)	够减(商0) 不够减(商1)	不够减(商1) 够减(商0)
0	1				
1	0	不够减(商0)	够减 (商1)	不够减(商1) 够减(商0)	够减(商0)
1	1				

商为负数补码

- 表中的0或1表示相应数的符号值；
- 补码除法的结果是连同符号位在內的补码形式的商。

- ◆ 被除数为 $[X]_{\text{补}}$ ，除数为 $[Y]_{\text{补}}$ ，则 $[X/Y]_{\text{补}}$ 的补码除法运算规则：

第1步：若 $[X]_{\text{补}}$ 与 $[Y]_{\text{补}}$ 同号：

$$R_1 = [X]_{\text{补}} - [Y]_{\text{补}}$$

- 若 $[X]_{\text{补}}$ 与 $[Y]_{\text{补}}$ 异号： $R_1 = [X]_{\text{补}} + [Y]_{\text{补}}$
- 若 R_1 与 $[Y]_{\text{补}}$ 同号：上商 $q_0 = 1$
- 若 R_1 与 $[Y]_{\text{补}}$ 异号：上商 $q_0 = 0$
- 在这里需做除法溢出判断（定点小数）：
 - 若 $[X]_{\text{补}}$ 与 $[Y]_{\text{补}}$ 同号且上商 $q_0 = 1$ ，则溢出。
 - 若 $[X]_{\text{补}}$ 与 $[Y]_{\text{补}}$ 异号且上商 $q_0 = 0$ ，则溢出。

第2步：若已求出部分余数 R_i ，

- R_i 与 $[Y]_{\text{补}}$ 同号： $R_{i+1} = 2R_i - [Y]_{\text{补}}$
- R_i 与 $[Y]_{\text{补}}$ 异号： $R_{i+1} = 2R_i + [Y]_{\text{补}}$
- 同样，若 R_{i+1} 与 $[Y]_{\text{补}}$ 同号：上商 $q_i = 1$
- 若 R_{i+1} 与 $[Y]_{\text{补}}$ 异号：上商 $q_i = 0$

第3步：重复执行第2步，直到求得足够位数的商为止。
商为负数时，商的末位加1。

例3：已知 $[X]_{\text{补}}=1.0111$, $[Y]_{\text{补}}= 0.1101$ 。

▲ $[X/Y]_{\text{补}}$ 的计算过程如下：

部分余数	商	说明
11 0111	0000	□ 开始 $R_0=[X]$
+00 1101		$R_1=[X]+[Y]$
00 0100	0000 1	R_1 与 $[Y]$ 同号，则 $q_0=1$
00 1000	000 1	$2R_1$ （部分余数和商同时左移）
+11 0011		$+[-Y]$
11 1011	000 10	R_2 与 $[Y]$ 异号，则 $q_1=0$
11 0110	00 10	$2R_2$
+00 1101		$+[Y]$
00 0011	00 101	R_3 与 $[Y]$ 同号，则 $q_2=1$

00 0011	00 101	R_3 与[Y]同号, 则 $q_2=1$
00 0110	0 101	$2R_3$
+11 0011		$+[-Y]$
11 1001	0 1010	R_4 与[Y]异号, 则 $q_3=0$
11 0010	1010	$2R_4$
+00 1101		$+[Y]$
11 1111	10100	R_5 与[Y]异号, 则 $q_4=0$
	+ 1	商为负数, 末位加1
	10101	

- $[X/Y]_{\text{补}} = \mathbf{1.0101}$; 余数的补码为 $\mathbf{1.1111} \times 2^{-4}$ 。
- $[X/Y]_{\text{真}} = -0.1011$; 余数的真值为 -0.0001×2^{-4} 。

4.4.3 阵列除法器

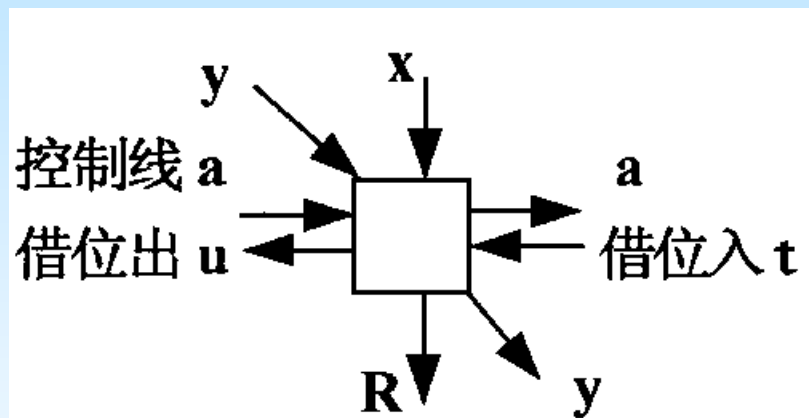
◆ 用组合逻辑线路来完成除法运算

▲ 图2.32 是一个组合逻辑单元；包含借位入和借位出的全减器。

- **X**: 被减数二进制代码
- **Y**: 减数二进制代码
- **t**: 低位传送来的借位入信号
- **u**: 本单元送往高位的借位出信号； $u = \overline{X}Y + \overline{X}t + Yt$

- **a**: 控制信号；这个控制信号一方面决定本单元**R**的生成值，另一方面也传送给低位。

- **R**: 本单元的结果，即本位的差； $R = X \oplus \overline{a} (Y \oplus t)$ ；
当 $a=0$ 时， $R = X - (Y + t)$ ；当 $a=1$ 时， $R = X$ 。



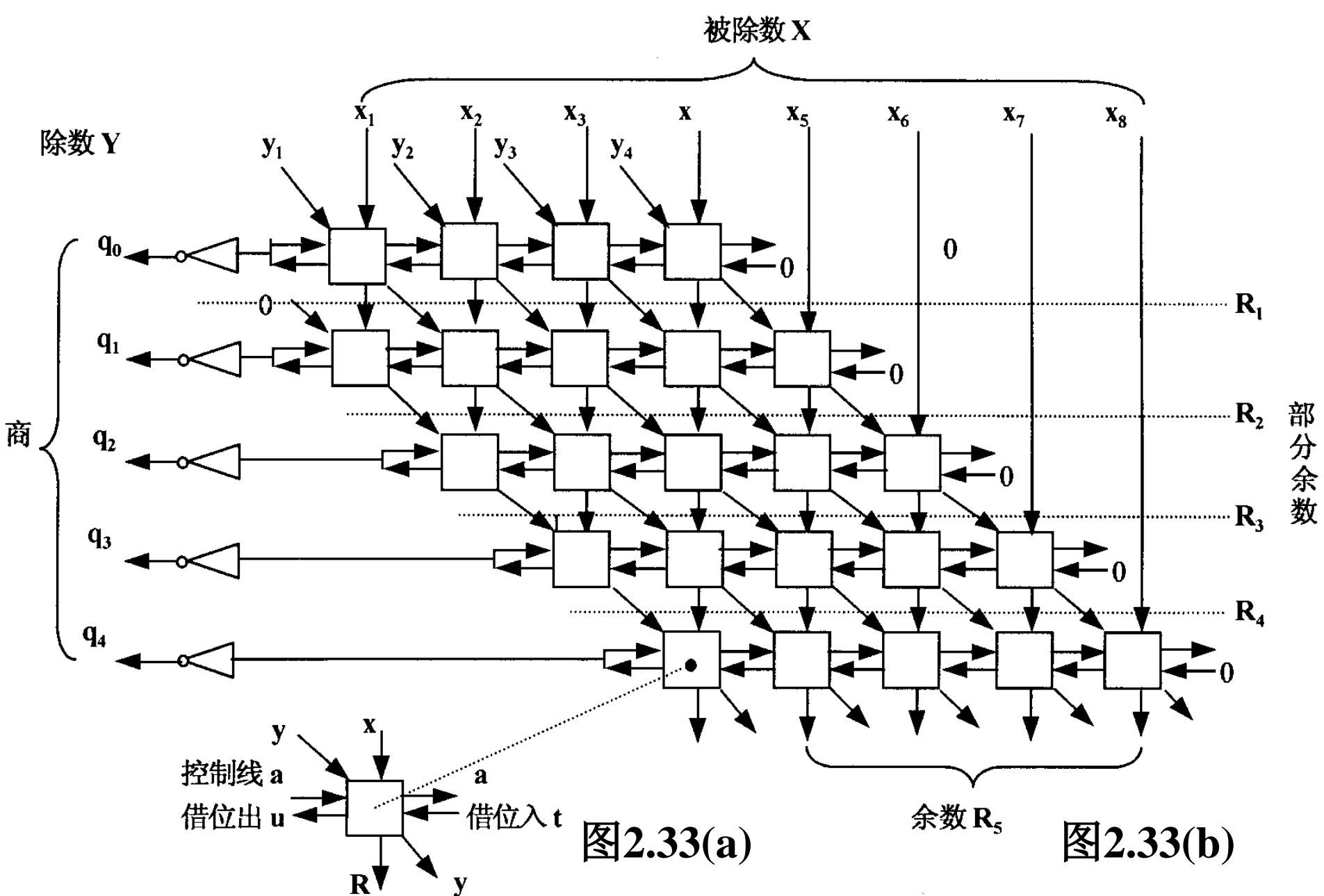
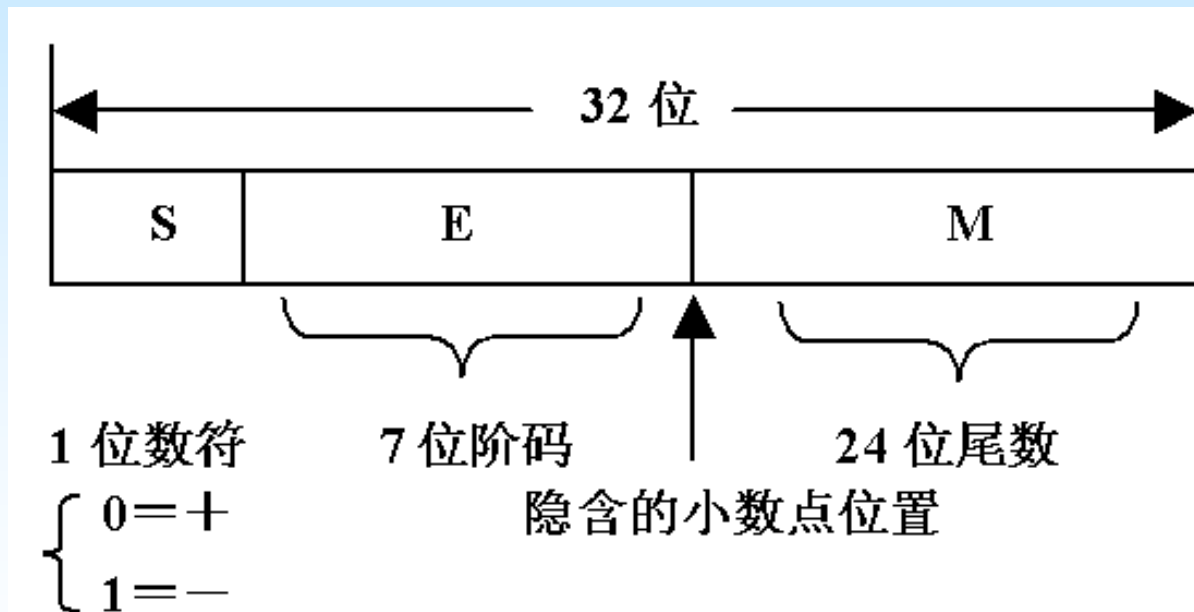


图4.28 直接实现定点数绝对值相除的阵列除法器

4.5 浮点运算

- ◆ 浮点数比定点数表示范围宽，有效精度高。

- ▲ 浮点数常见格式



- ▲ 规格化浮点数

4.5.1 浮点加、减法

◆ 浮点数X和Y加减法运算的规则

▲ 两个二进制浮点数X和Y可以表示为：

- $X=M_x \times 2^{E_x}$, $Y=M_y \times 2^{E_y}$; M_x , E_x 包含数符和阶符;
- $X \pm Y$ 的结果表示为 $M_z \times 2^{E_z}$ 。
- 浮点数X和Y是按规格化数存放，它们运算后的结果也是规格化的。

▲ 考查十进制数的加法运算

- $123 \times 10^2 + 456 \times 10^3 = 12.3 \times 10^3 + 456 \times 10^3$
- $= (12.3 + 456) \times 10^3 = 468.3 \times 10^3$

▲ 推论浮点数X和Y加减法运算的规则为：

- $X + Y = (M_x \times 2^{E_x - E_y} + M_y) \times 2^{E_y}$,
不失一般性，设 $E_x \leq E_y$
- $X - Y = (M_x \times 2^{E_x - E_y} - M_y) \times 2^{E_y}$,

◆ 计算机中实现X和Y加、减法运算的步骤为：

第1步：对阶

- ▲ 对阶使得原数中较大的阶码成为两数的公共阶码；
 - 小阶码的尾数按两阶码的差值决定右移的数量。
- ▲ 两阶码的差值表示为： $\Delta E = E_x - E_y$
 - 若 $\Delta E \leq 0$ ，则 $E_z \leftarrow E_y$ ， $E_x \leftarrow E_y$ ， $M_x \leftarrow M_x \times 2^{E_x - E_y}$
 - 若 $\Delta E > 0$ ，则 $E_z \leftarrow E_x$ ， $E_y \leftarrow E_x$ ， $M_y \leftarrow M_y \times 2^{E_y - E_x}$

▲ 小阶码的尾数右移规则：

- ① 原码形式的尾数右移时，符号位不参加移位，数值位右移，空出位补0。
 - 补码形式的尾数右移时，符号位与数值位一起右移，空出位填补符号位的值。
- ② 尾数的右移，使得尾数中原来 $|\Delta E|$ 位有效位移出。
 - 移出的这些位应保留，并且参加后续运算；这对运算结果的精确度有一定影响。
 - 保留的多余的位数（q位）称为警戒位。

第2步：尾数加减

▲ 对尾数进行加、减运算

- $M_z \leftarrow M_x \pm M_y$

第3步：尾数规格化

▲ 设浮点数的尾数用补码表示，且加、减运算时采用双符号位，则规格化形式的尾数应是如下形式：

- 尾数为正数时： **00**1xx...x
- 尾数为负数时： **11**0xx...x

▲ 尾数违反规格化的情况有以下两种可能：

① 尾数加、减法运算中产生溢出

- 正溢出时，符号位为01；
- 负溢出时，符号位为10；
- 规格化采取的方法是：

尾数右移一位，阶码加1；这种规格化称为右规。

表示为： $M_z \leftarrow M_z \times 2^{-1}$ ， $E_z \leftarrow E_z + 1$ 。

② 尾数的绝对值小于二进制的0.1。补码形式的尾数表现为最高数值位与符号位同值。

- 尾数为正数时： $\underbrace{00}_{\text{符号位}} \underbrace{00\cdots 01x\cdots x}_{\text{数值位}}^{\text{K个0}}$

- 尾数为负数时： $\underbrace{11}_{\text{符号位}} \underbrace{11\cdots 10x\cdots x}_{\text{数值位}}^{\text{K个1}}$

- 采取规格化的方法：

符号位不动，数值位逐次左移，阶码逐次减1，直到满足规格化形式的尾数。

- 这种规格化称为左规。

表示为： $M_z \leftarrow M_z \times 2^k$ ， $E_z \leftarrow E_z - k$

第4步：尾数的舍入处理

▲ 对结果尾数进行舍入处理方法

① 0舍1入法

- 警戒位（**q位**）中的**最高位为1**时，就在尾数**末尾加1**；
- 警戒位（**q位**）中的最高位为0时，**舍去所有的警戒位**；
- 这种方法的最大误差为 **$\pm 2^{-(n+1)}$** ，**n**为有效尾数位数。

② 恒置1法

- 不论警戒位为何值，尾数的有效最低位恒置1。
- 恒置1法产生的最大误差为 **$\pm 2^{-n}$** ，**n**为有效尾数位数。

③ 恒舍法

- 无论警戒位的值是多少，都舍去。
 - 尾数的结果就取其有效的n位的值。
 - 称为趋向零舍入(**Round toward zero**)。
- ▲ 上述几种舍入方法对**原码形式**的尾数进行舍入处理，舍入的效果与真值舍入的效果是一致的。
- 对于**补码形式的负的尾数**来说，所进行的舍入处理将与真值的舍入效果可能不一致。

- 对负数的补码来说，执行0舍处理使得原值变大，1入处理反而使得原值变小。
- ▲ 对于0舍1入法，负数补码的舍入规则为：负数补码警戒位中的最高位为1且其余各位不全为0时，在尾数末尾加1，其余情况舍去尾数。
- ▲ 舍入处理后需检测溢出情况；若溢出，需再次规格化（右规）。

第5步：阶码溢出判断


- 若阶码 E_z 正溢出，表示浮点数已超出允许表示数范围的上限，置溢出标志。
- 若阶码 E_z 负溢出，运算结果趋于零，置结果为机器零。
- 浮点数加、减法运算正常结束，浮点数加、减法运算结果为 $M_z \times 2^{E_z}$ 。

例2: 已知 $X = 0.11011011 \times 2^{010}$,
 $Y = -0.10101100 \times 2^{100}$;

- 用补码来表示浮点数的尾数和阶码

- $[X]_{\text{浮}} = 0 \ 0 \ 010 \ 11011011$

- $[Y]_{\text{浮}} = 1 \ 0 \ 100 \ 01010100$



 数符 阶符 阶码 尾数

▲ $[X+Y]_{\text{浮}} = M_z \times 2^{E_z}$, 执行 $[X+Y]_{\text{浮}}$ 的过程如下:

① 对阶

- $\Delta E = E_x - E_y = 00 \ 010 + 11 \ 100 = 11 \ 110$

- 即 $\Delta E = -2$, M_x 右移两位, $M_x = 0 \ 00110110 \ 11$

- $E_b = E_y = 0 \ 100$

警戒位₁₂₄

② 尾数加法

- $M_b = M_x + M_y$

$$\begin{array}{r} \textcolor{red}{00} \text{ } 00110110 \text{ } \underline{11} \\ + \textcolor{red}{11} \text{ } 01010100 \\ \hline \textcolor{red}{11} \text{ } 10001010 \text{ } \underline{11} \end{array} \quad \begin{array}{l} \swarrow \\ \searrow \end{array} \text{警戒位}$$

- 因此 $M_b = \textcolor{red}{11} \textcolor{brown}{10001010} \textcolor{blue}{11}$

③ 尾数规格化

- 尾数没有溢出，但符号位与最高数值位有 $K=1$ 位相同，需左规：

- M_b 左移 $K=1$ 位： $M_b = \textcolor{red}{11} \text{ } 00010101 \textcolor{blue}{1}$

- E_b 减1： $E_b = 00 \text{ } 011$

④ 舍入处理

- 采用恒舍法，执行舍操作。
- 得： $M_b = 11\ 00010101$

⑤ 阶码溢出判断

- 阶码无溢出， $X+Y$ 正常结束，得：
- $[X+Y]_{\text{浮}} = 1\ 0\ 011\ 00010101,$
- 即 $X+Y = -0.11101011 \times 2^{011}$

4.5.2 浮点乘、除法运算

- ◆ 对浮点数的乘、除法运算来说，免去对阶这一步。
 - 对结果数据规格化、舍入处理和阶码判断与前述相同。
- ◆ 对两个规格化的浮点数 $X=M_x \times 2^{E_x}$ ， $Y=M_y \times 2^{E_y}$ ，实现乘、除法运算的规则如下：
 - $X \times Y = (M_x \times 2^{E_x}) \times (M_y \times 2^{E_y}) = (M_x \times M_y) \times 2^{E_x + E_y}$
 - $X/Y = (M_x \times 2^{E_x}) / (M_y \times 2^{E_y}) = (M_x / M_y) \times 2^{E_x - E_y}$

1. 浮点数乘法的运算步骤

(1) 两浮点数相乘

- $M_z = M_x \times M_y$ $E_z = E_x + E_y$

(2) 尾数规格化

- M_x 和 M_y 都是绝对值大于或等于0.1的二进制小数，两数乘积 $M_x \times M_y$ 的绝对值是大于或等于0.01的二进制小数。
- 不可能溢出，不需要右规。对于左规来说， M_z 最多左移一位，阶码 E_z 减1。

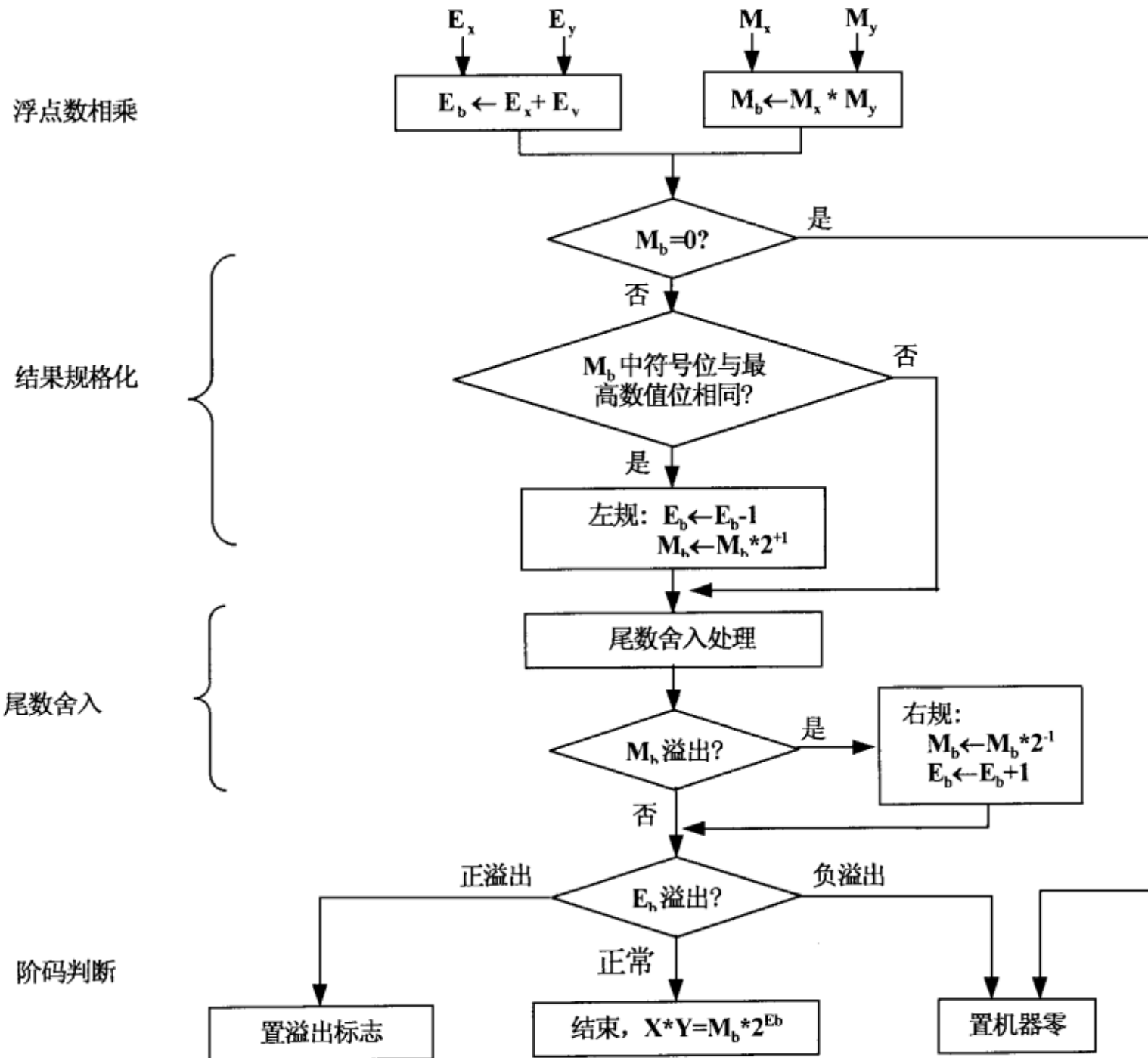
(3) 尾数舍入处理

- $M_x \times M_y$ 产生双字长乘积，如要求得到单字长结果，低位乘积当作警戒位进行结果舍入处理。
- 若要求结果保留双字长乘积，不需要舍入处理。

(4) 阶码溢出判断

- 对 E_z 的溢出判断完全相同于浮点数加、减法的相应操作。

图 4.29 浮点数乘法运算流程图



3. 浮点数除法的运算步骤

(1) 除数是否为0，若 $M_y = 0$ ，出错报告。

(2) 两浮点数相除

- 可以表示为： $M_z = M_x / M_y$ ， $E_z = E_x - E_y$

(3) 尾数规格化

- 两数相除 M_x / M_y 的绝对值是大于或等于0.1且小于10的二进制小数。所以，对 M_z 不需要左规操作。
- 若溢出，执行右规： M_z 右移一位，阶码 E_z 加1。

(4) 尾数舍入处理

(5) 阶码溢出判断

- (4)、(5) 两步操作相同于浮点数加、减法的相应操作。

除数为0检测

浮点数相除

结果规格化

尾数舍入

阶码判断

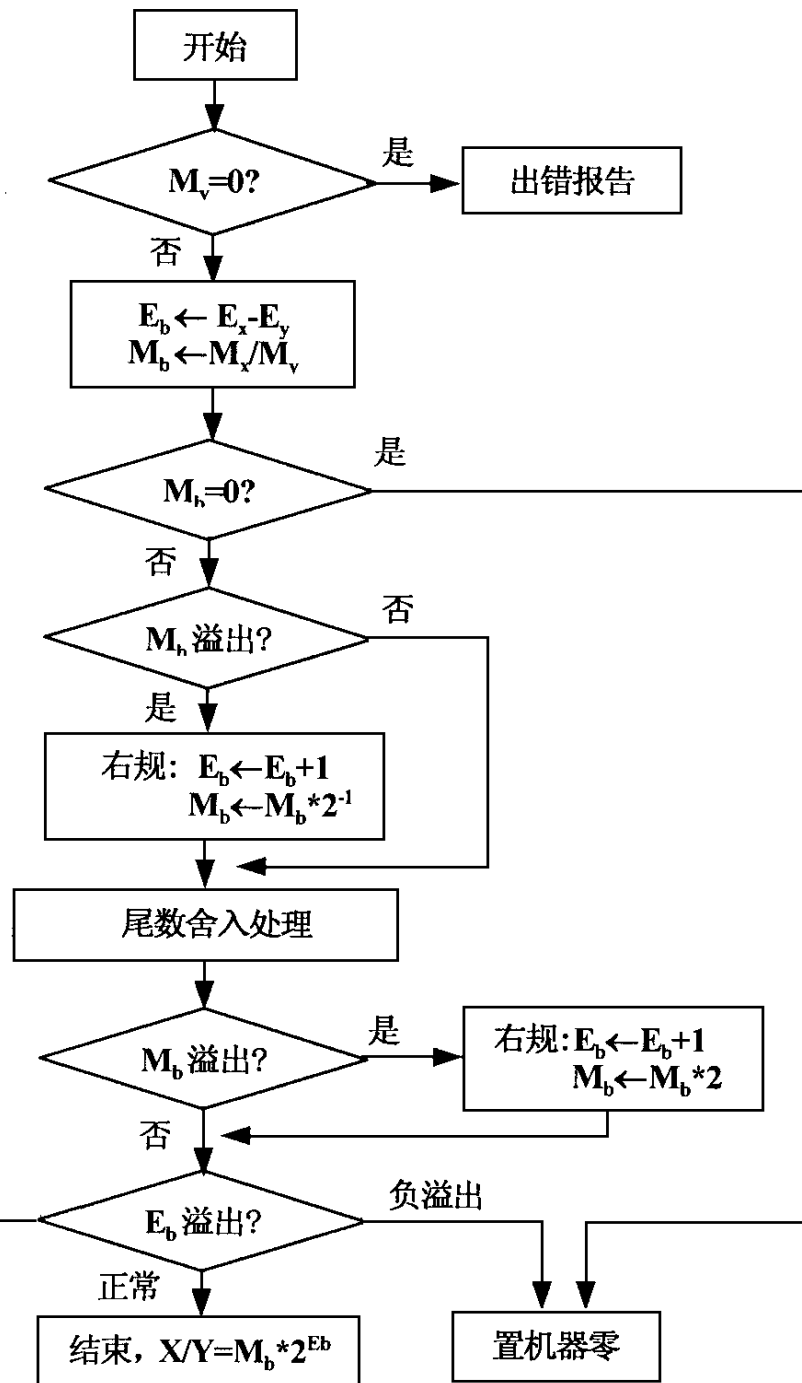


图 4.30

浮点数除法运算流程图

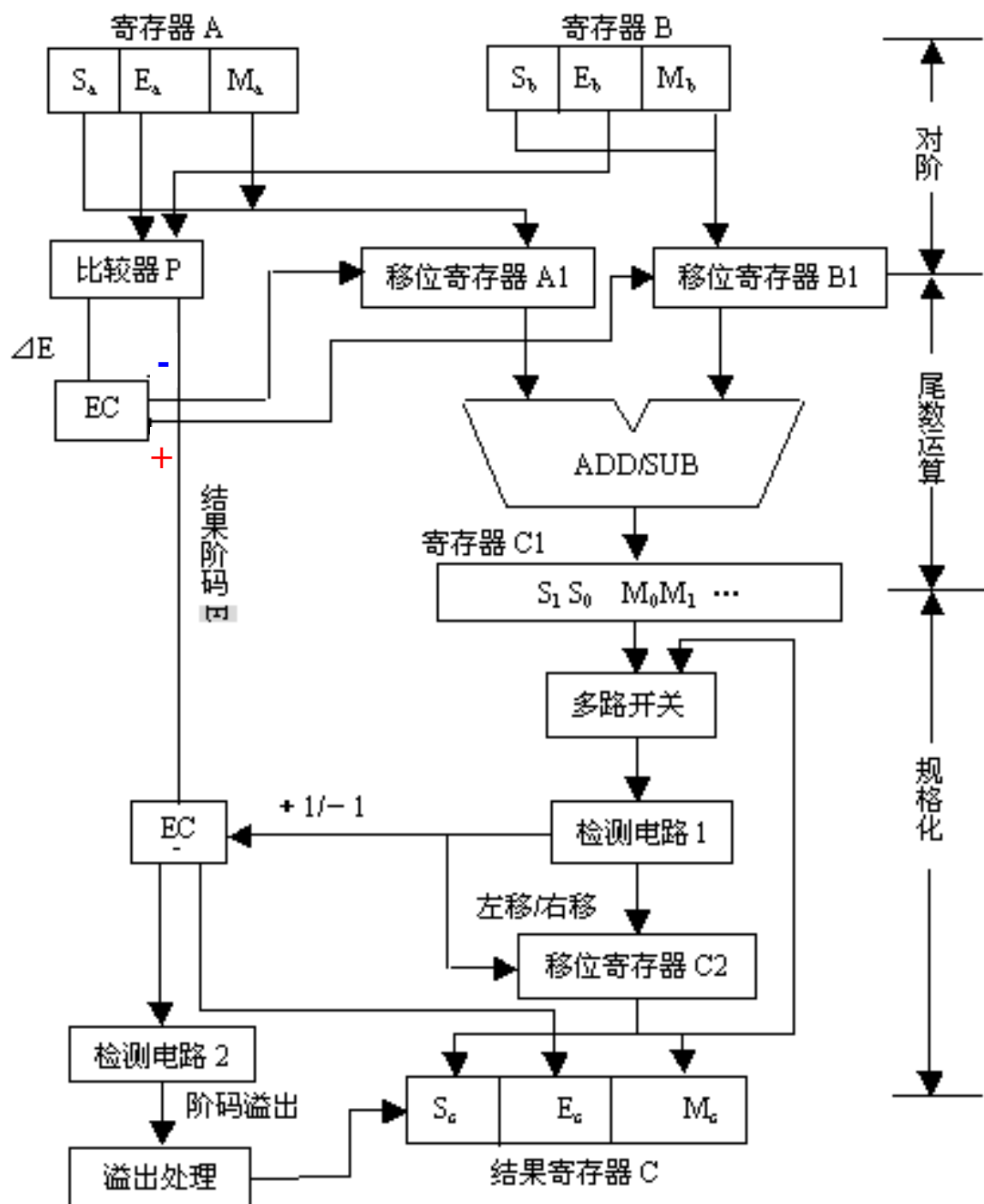
4.5.3 浮点运算部件

- ◆ 实现浮点四则运算的方案：1) 硬件方法构成专用的浮点运算部件；2) 软件方法。

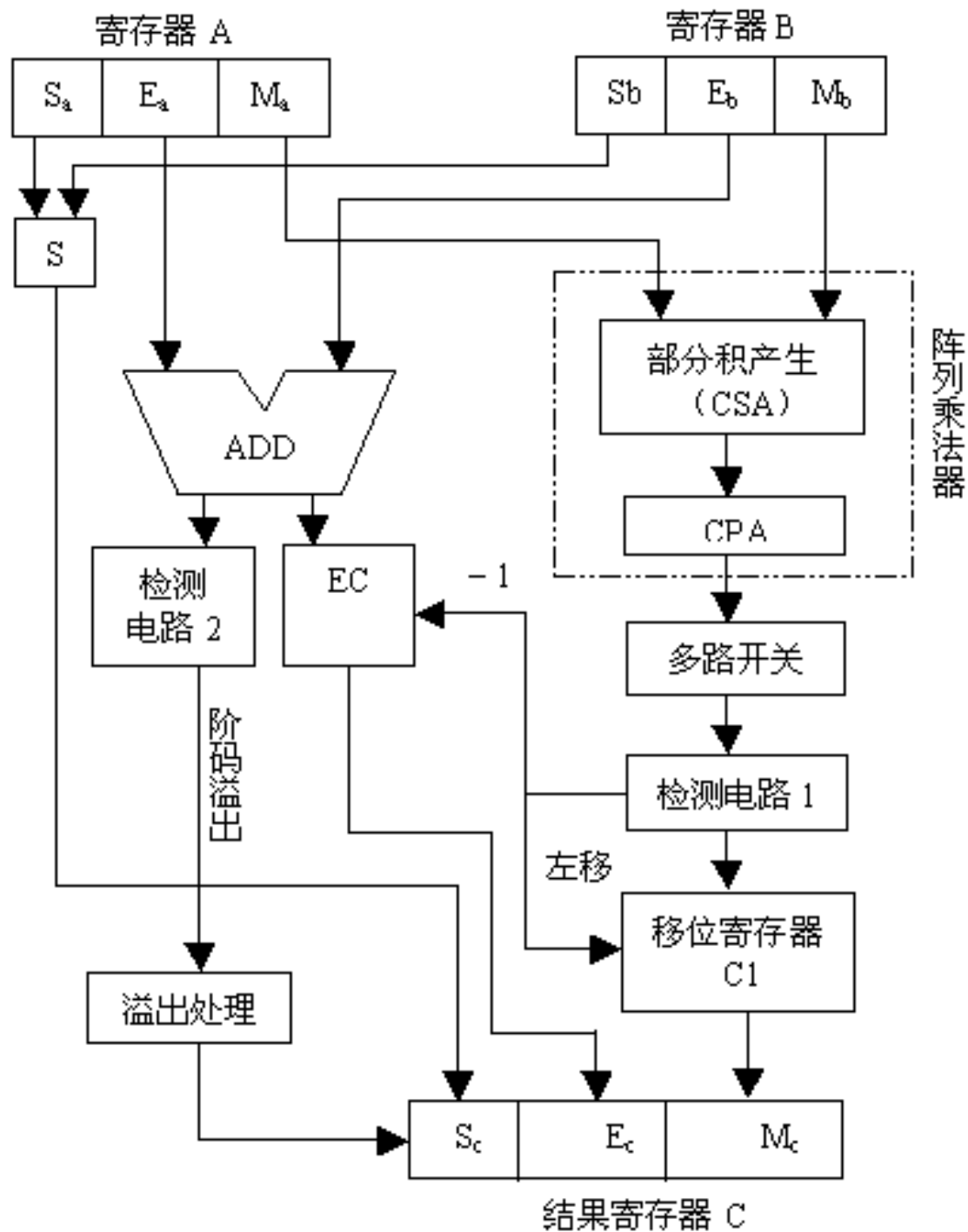
1. 浮点加、减运算部件

- 浮点加、减运算需要对阶码和尾数分别予以不同的处理。

图4.31 浮点加减运算部件逻辑框图



2. 浮点乘法 运算部件



4.6 十进制数的加、减法运算

- 在二进制加、减法运算的基础上，通过适当的校正来实现将二进制的“和”改变成所要求的十进制格式。

1. 十进制的加法运算

◆ BCD码加法结果修正的规则：

- (1) 两个BCD数码相加之和等于或小于1001，即十进制的9，不需要修正。
- (2) 两个BCD数码相加之和大于或等于1010且小于或等于1111， $(10\sim15)_{10}$ 之间，需要在本位加6修正。修正的结果是向高位产生进位。

(3) 两个BCD数码相加之和**大于1111**，加法的过程已经向高位产生了进位，对本位也要进行加6修正。

例1: $(15)_{10} + (21)_{10} = (36)_{10}$

• BCD码加法运算:

0001 0101	15
+ 0010 0001	+21
<hr/>	<hr/>
0011 0110	36

- 每个十进制位的BCD码和均小于9，因此，对计算结果无需修正

例2: $(15)_{10} + (26)_{10} = (41)_{10}$

• BCD码加法:

	0001 0101	15	
	+ 0010 0110	+ 26	进位
	<hr/>	<hr/>	
	0011 1011	41	
修正	+ 0110		
	<hr/>		
	0100 0001		

例3: $(18)_{10} + (18)_{10} = (36)_{10}$

BCD码加法:

0001 1000	18
+ 0001 1000	+ 18
<hr/>	<hr/>
0011 0000	36
修正 + 0110	
<hr/>	
0011 0110	

进位

- ▲ 处理BCD码的十进制加法器只需要在二进制加法器上添加适当的校正逻辑就可以了

◆ 实现 X_i 和 Y_i 相加的BCD码加法器

- BCD码分别是 $x_{i8}x_{i4}x_{i2}x_{i1}$ 和 $y_{i8}y_{i4}y_{i2}y_{i1}$ 得到4位二进制和 $z_{i8}^*z_{i4}^*z_{i2}^*z_{i1}^*$ 和进位输出 C_{i+1}^* 。
- 校正逻辑为：
$$C_{i+1} = \underbrace{C_{i+1}^*}_{\text{和大于1111}} + \underbrace{z_{i8}^* z_{i4}^* + z_{i8}^* z_{i2}^*}_{\text{和大于1001}}$$
- 十进制加法器用进位信号 C_{i+1} 产生校正因子0或6。

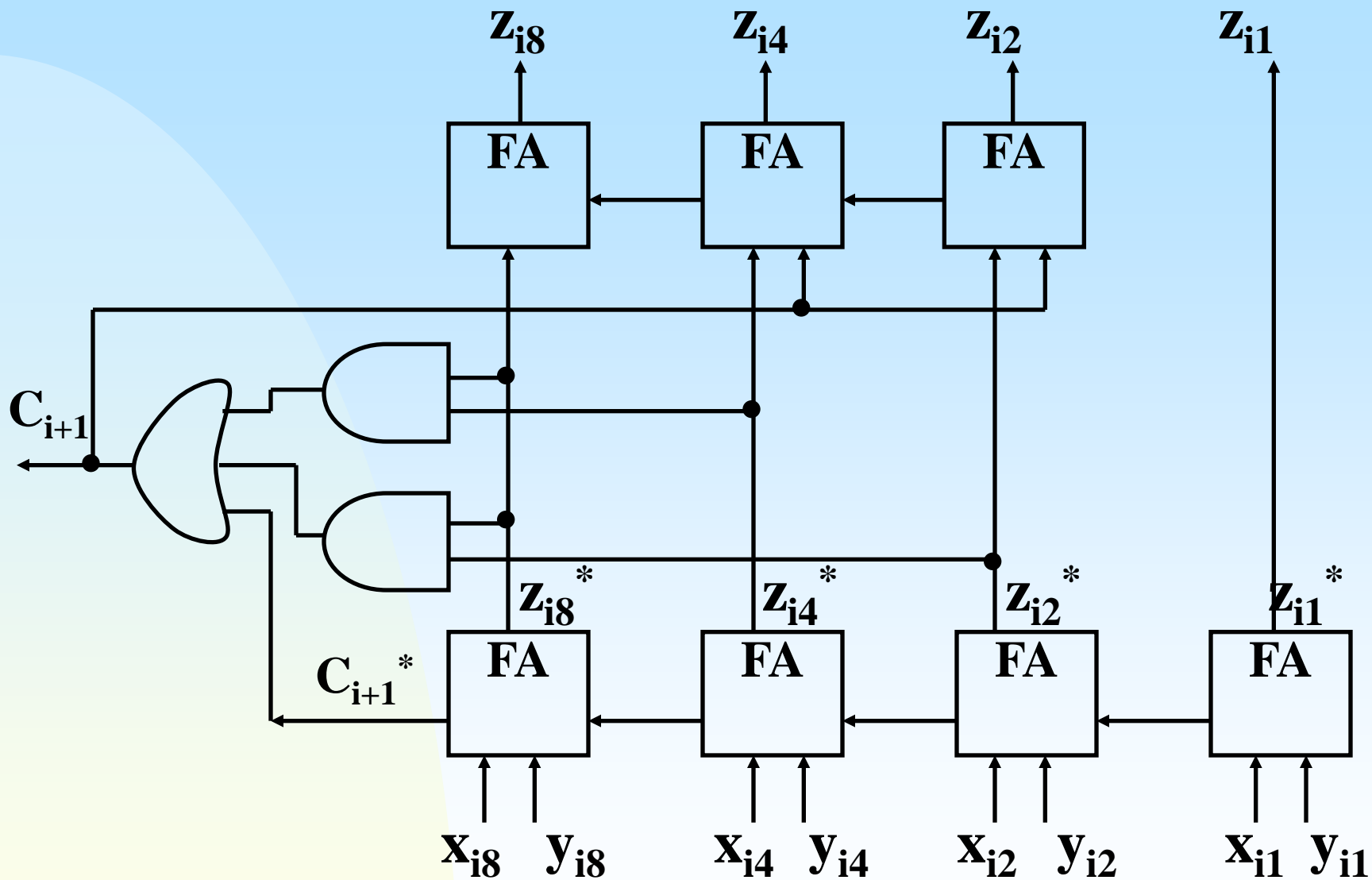


图4.32 处理一位十进制数字的BCD码加法器

- ◆ 用n个一位十进制数字的BCD码加法器可以构成一个n位十进制数的串行进位加法器

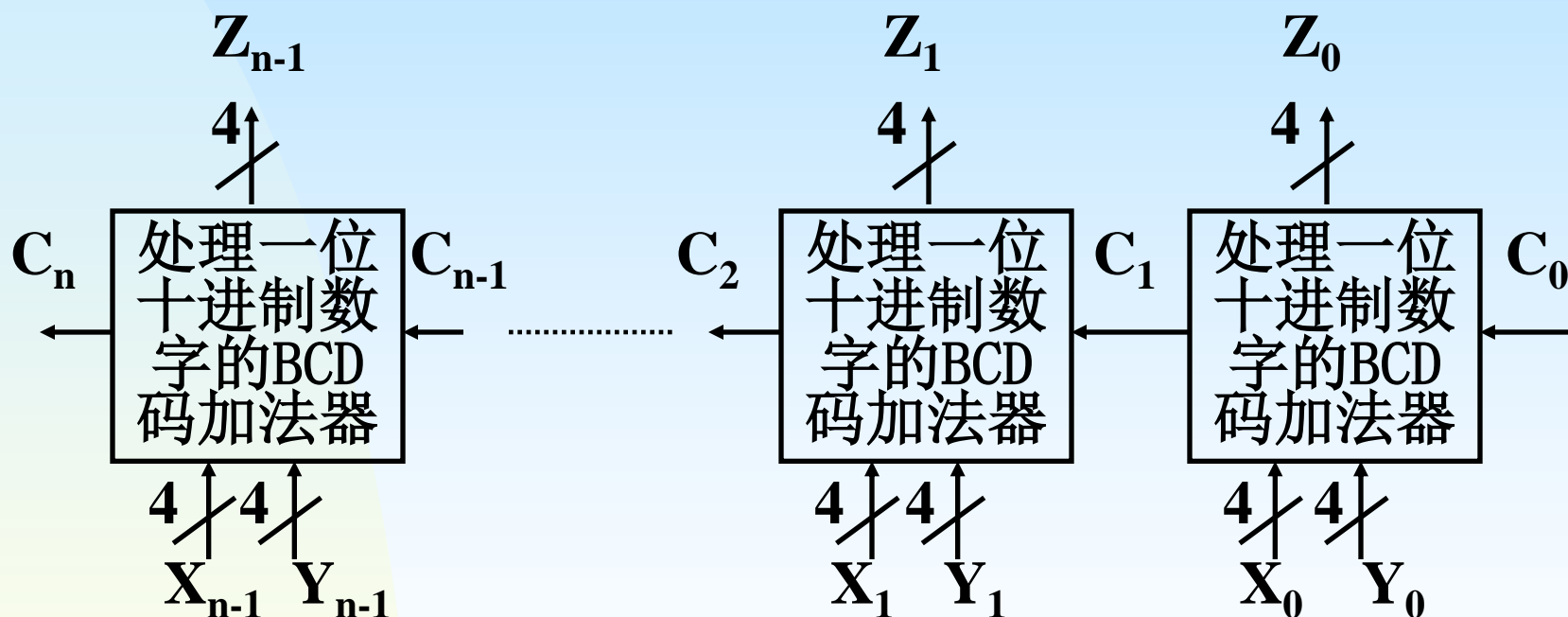


图4.33 n位十进制数的BCD码串行进位加法器

2. 十进制的减法运算

- ◆ 两个BCD码的十进制位的减法运算，常采用先**取减数的模9补码**，再与被减数相加的方法实现。
- ◆ BCD编码的十进制数字的模9补码的产生方法。
 - (1) 先将四位BCD码Y按位取反，再加上二进制1010（十进制10），结果的最高进位位丢弃；得 $9-Y$ 。

例如：BCD码0011（十进制3）

模9补码计算方法为：

- 先对0011按位取反得1100，再将1100加上1010且丢弃最高进位位得0110（十进制6）。
- 显然0011的模9补码是为0110。

(2) 先将四位BCD码加上0110（十进制6），再将每位二进制位按位取反。

例如：BCD码0011（十进制3）的模9补码的计算方法：

- 先计算 $0011 + 0110 = 1001$ ；
- 再对1001按位取反得0110。

- ◆ 实现BCD码的十进制数字Y的模9补码的组合电路
- ▲ 设Y的BCD码为 $y_8y_4y_2y_1$ ，Y的模9补码为 $b_8b_4b_2b_1$ ； $y_8y_4y_2y_1$ 和 $b_8b_4b_2b_1$ 之间的真值表关系：

表 4.6 十进制数字 Y 模 9 补码的真值表

y_8	y_4	y_2	y_1	b_8	b_4	b_2	b_1
0	0	0	0	1	0	0	1
0	0	0	1	1	0	0	0
0	0	1	0	0	1	1	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	0	0
0	1	1	0	0	0	1	1
0	1	1	1	0	0	1	0
1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0

- 根据真值表得出：

$$\begin{cases} b_1 = \bar{y}_1 \\ b_2 = y_2 \\ b_4 = y_4 \bar{y}_2 + \bar{y}_4 y_2 \\ b_8 = \bar{y}_8 \bar{y}_4 \bar{y}_2 \end{cases}$$
- 上式中加入变量M，并改写为如下式子：

$$\begin{cases} b_1 = y_1 \bar{M} + \bar{y}_1 M \\ b_2 = y_2 \\ b_4 = y_4 \bar{M} + (\bar{y}_4 y_2 + y_4 \bar{y}_2) M \\ b_8 = y_8 \bar{M} + \bar{y}_8 \bar{y}_4 \bar{y}_2 M \end{cases}$$
- 当M=0时， $b_8 b_4 b_2 b_1$ 表示Y本身；
 当M=1时， $b_8 b_4 b_2 b_1$ 表示Y的模9补码。

$$(57-34)_{10} = 0101\ 0111 + [0011\ 0100]_{\text{模}9\text{补}} + C_i$$

$$= 0101\ 0111 + 0110\ 0101 + 1$$

0101 0111	X的BCD码
0110 0101	[Y] _{模9补}
+ 1	C _i 低位进位
<hr/>	
1101	
+ 0110	BCD码修正
<hr/>	
1 0011	
<hr/>	
1100	
+ 0110	BCD码修正
<hr/>	
1 0010 0011	

▲ 实现上式的组合电路的逻辑符号

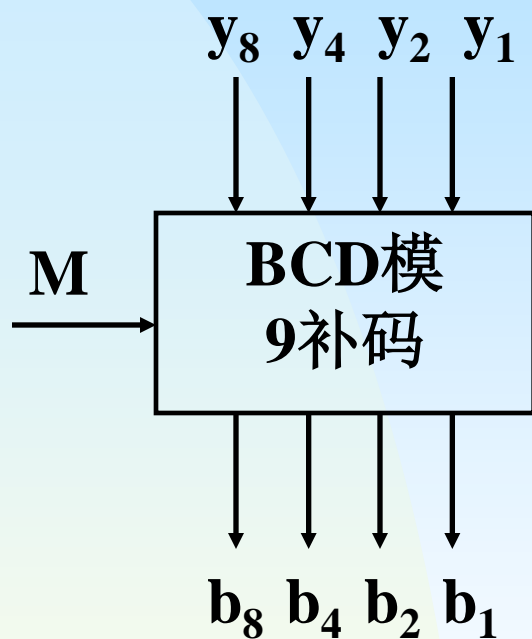


图4.34 实现运算组合电路符号表示

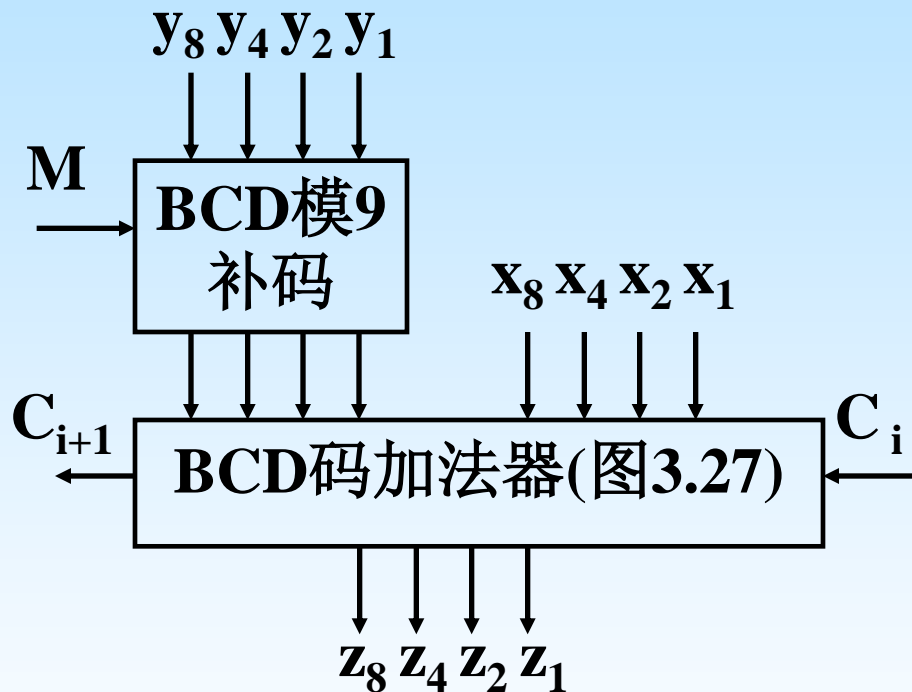


图4.35 两个BCD码的加减法运算线路

作业三： P80—**21**(**第一行**), 22, **24(2、3)**, 25;

作业四： P80~P81—26(1), **28(1)**, **33(1)** ;