

计算机组成原理

2017年修订

西南交通大学信息科学与技术学院
唐慧佳 hjtang@home.swjtu.edu.cn



第4章 数值的机器运算

- § 4.1 基本算术运算的实现
- § 4.2 定点加减运算
- § 4.3 带符号数的移位和舍入操作
- § 4.4 定点乘法运算
- § 4.5 定点除法运算
- § 4.6 规格化浮点运算
- § 4.7 十进制整数的加法运算
- § 4.8 逻辑运算与实现
- § 4.9 运算器的基本组成结构



第4章 数值的机器运算

本章要点:

1. 与门、与或门、三态门、寄存器等器件的典型应用方法，以及它们的组合应用方法；
2. 加法器的先行进位原理；定点数的加减乘除运算方法(其中补码数运算最重要)，并能把数据表示、运算方法(算法)和电路实现联系起来；
3. 微操作的概念及描述方法；
4. 逻辑运算及其实现方法、浮点数的运算方法；
5. 定点运算器的典型结构.

第4章 数值的机器运算

本章内容包括两个方面：

算术、逻辑运算的运算原理和规则 -- 运算方法(算法)
运算算法的硬件实现 -- 运算电路

本章的教学目标：

- (1) 掌握运算器内部的工作原理和基本结构；
- (2) 提高硬件方面的专业素质；
- (3) 提升计算思维能力
 即如何把问题转化为可计算的算法。

第4章 数值的机器运算

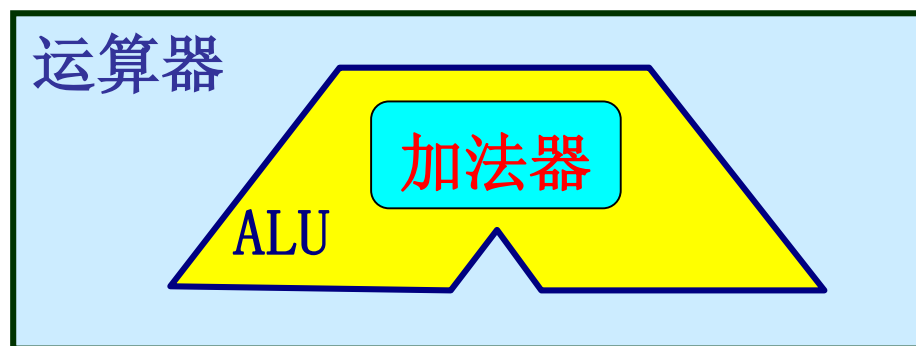
几个名词及其相互关系：

运算方法： 算术运算和逻辑运算的运算规则。

运算器： 运算算法的硬件电路实现。

ALU： 运算器的核心部件

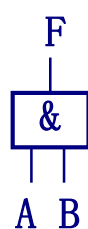
加法器： ALU中最基本的部件



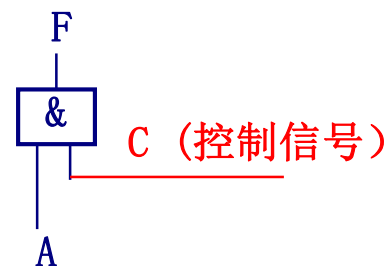
§ 4.0 预备知识

1. 门电路

与门

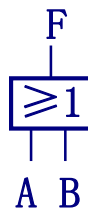


$$F = A \cdot B$$



应用例子

或门

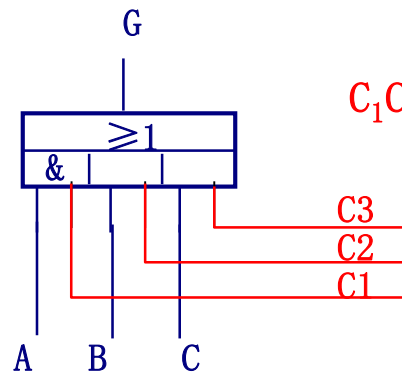
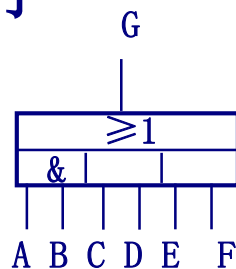


$$F = A + B$$

§ 4.0 预备知识

1. 门电路

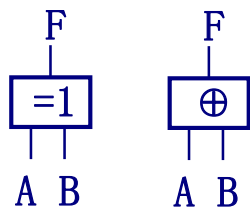
与或门



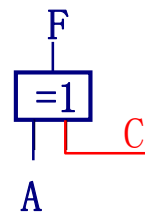
$C_1C_2C_3=100$ 时, $G = A$

应用例子

异或



$F = A \oplus B$
(半加)



$C=0$ 时, $F = A$

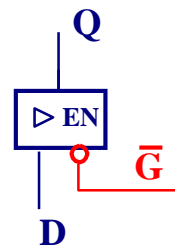
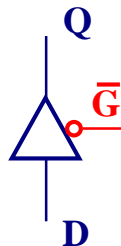
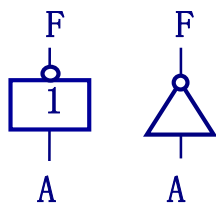
$C=1$ 时, $F = \bar{A}$

应用例子

§ 4.0 预备知识

1. 门电路

非门、三态门等



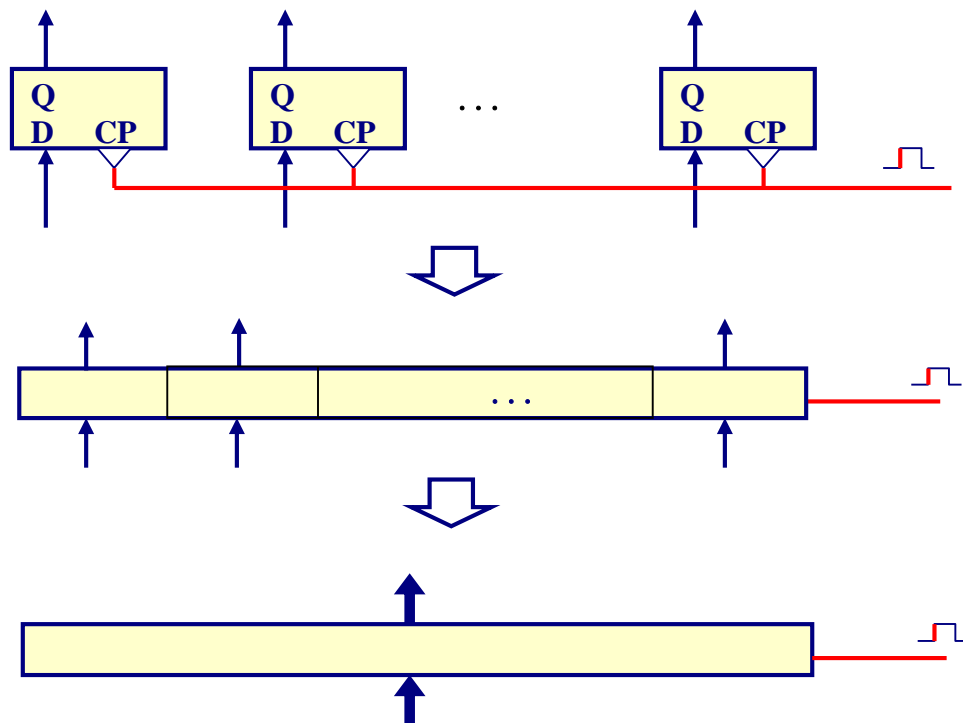
$\bar{G}=0$ 时, $Q = D$;
 $\bar{G}=1$ 时, Q 为高阻态
。

§ 4.0 预备知识

2. 寄存器

具有记忆（存储）功能。

典型结构：由多个触发器组成，每个触发器对应 1 位。





计算机各个部件的内部，主要由两种类型的电路组成：

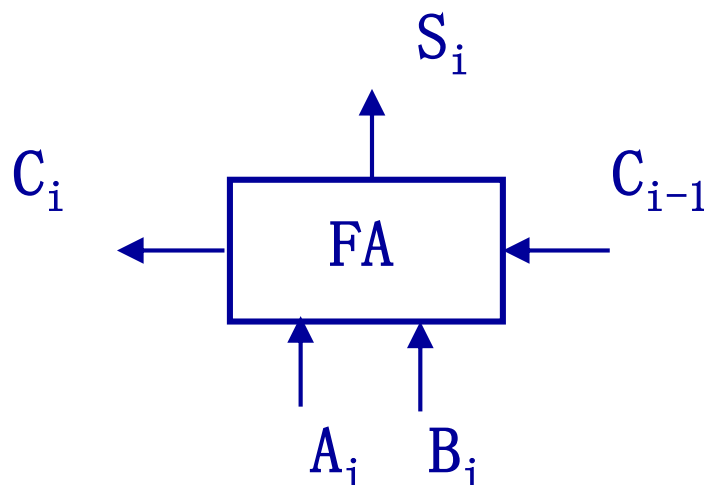
- 1) 没有记忆功能的组合逻辑电路，由门电路组成；
- 2) 带记忆功能的器件，最典型的就是寄存器，其它记忆元件也可以看作是寄存器。

§ 4.1 基本算术运算的实现

加法运算是最基本的算术运算，减、乘、除运算最终都可以归结为加法运算。

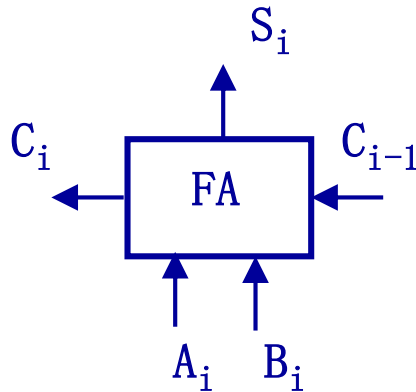
4.1.1 加法器

1. 全加器 (FA)



4.1.1 加法器

1. 全加器 (FA)



A:	0	0	1	0
B:	1	0	1	1
S:	1	1	0	1

输入量: A_i 、 B_i 、 C_{i-1} (低位传来的进位);

输出量: S_i (本位和)、 C_i (向高位的进位)。

$$S_i = A_i \oplus B_i \oplus C_{i-1}$$

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

2. 串行加法器和并行加法器

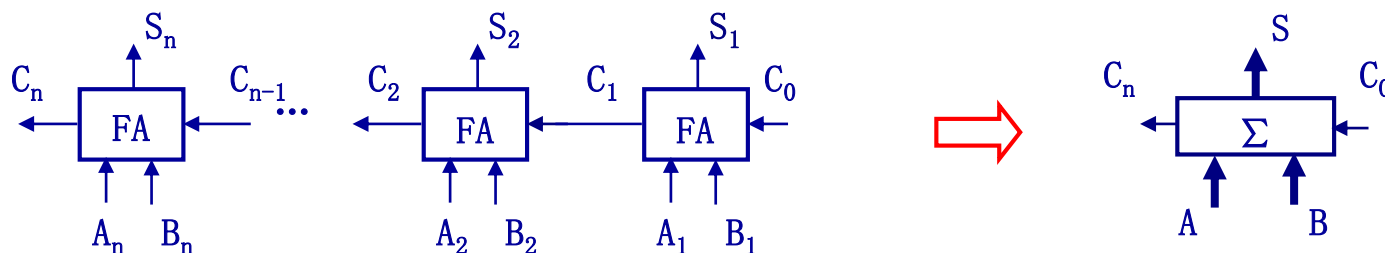
1) 串行加法器用一个全加器进行 n 位字长的加法运算。

特点:器件少，但运算速度慢，因为运算数据需逐位串行送入加法器进行运算。

2) 并行加法器由 n 个全加器组成(n 为字长)，数据的各位同时运算。

4.1.2 进位的产生和传递

最简单的并行加法器是串行进位（行波进位）加法器：



虽然操作数的各位是同时提供的，但高位运算需要使用低位运算后所产生的进位。

$$C_1 = G_1 + P_1 C_0 \quad (G_i = A_i B_i \text{ 称为本地进位})$$

$$C_2 = G_2 + P_2 C_1 \quad (P_i = A_i \oplus B_i \text{ 称为传递进位})$$

$$C_3 = G_3 + P_3 C_2$$

$$C_4 = G_4 + P_4 C_3$$

.....

A:	0	0	1	0
B:	1	0	1	1
S:	1	1	0	1

C_0

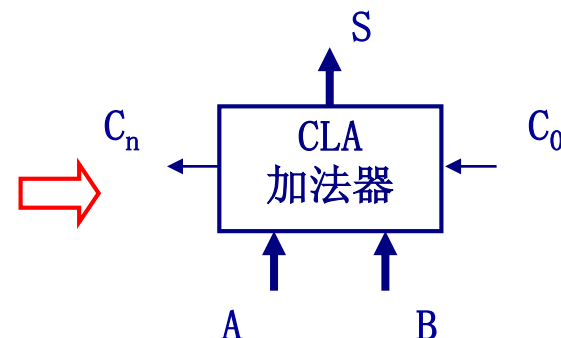
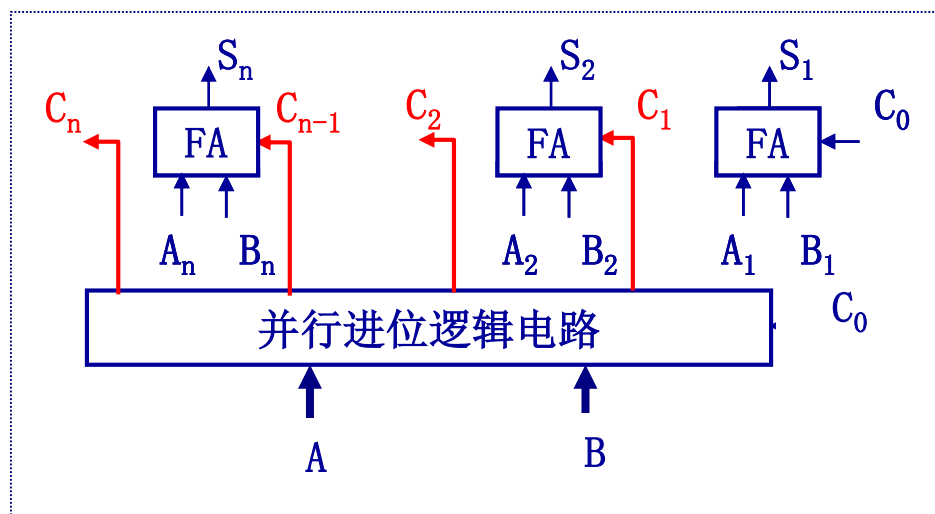
由此可见，提高并行加法器速度的关键是尽量加快进位产生和传递的速度。

4.1.3 并行加法器的快速进位

1. 并行进位方式

又叫先行进位，解决加法器中进位的传递速度问题。

基本思路：让各位的进位输入与低位的进位产生无关，仅与两个参加操作的数有关，以提高加法器的运算速度。



串行进位逻辑:

$$C_1 = G_1 + P_1 C_0 \quad (\text{其中: } G_i = A_i B_i \text{ 称为本地进位})$$

$$C_2 = G_2 + P_2 C_1 \quad (\quad P_i = A_i \oplus B_i \text{ 称为传递进位})$$

$$C_3 = G_3 + P_3 C_2$$

$$C_4 = G_4 + P_4 C_3$$

.....

可改写为:

$$C_1 = G_1 + P_1 C_0$$

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

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

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

.....

—— 并行进位 (先行进位CLA—Carry Look Ahead)

并行进位逻辑:

$$C_1 = G_1 + P_1 C_0$$

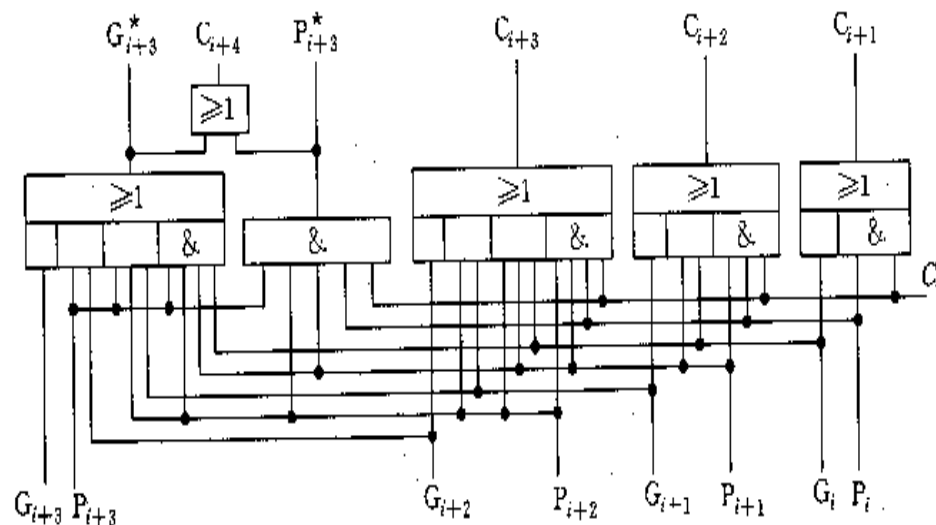
$$C_2 = G_2 + P_2 G_1 + P_2 P_1 C_0$$

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

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

.....

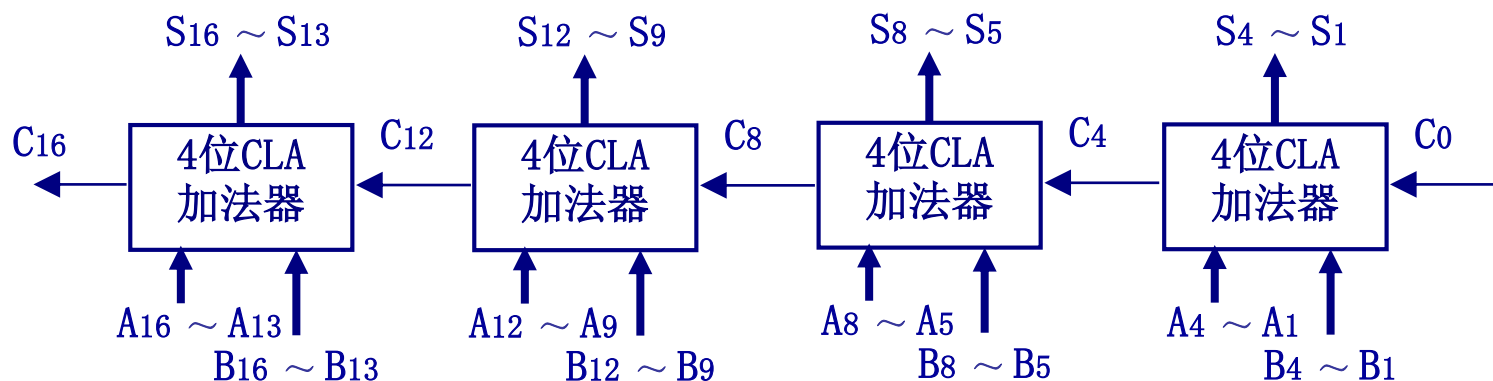
并行进位逻辑电路
可以用与或门实现，其
每个进位输出 C_i 仅由 G_i 、 P_i 及最低进位输入 C_0 决定，可以同时产生。



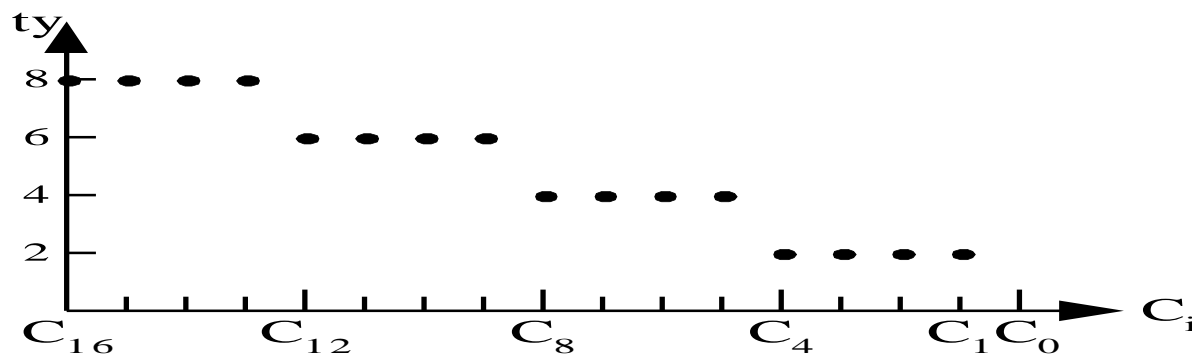
2. 分组并行进位方式

(1) 单级先行进位方式（组内并行、组间串行）

以16位加法器为例：



先行进位时间图



2. 分组并行进位方式

(2) 多级先行进位方式（组内并行、组间并行）

把单级先行进位中的式子变成：

$$\begin{aligned}
 C_4 &= G_4 + P_4 G_3 + P_4 P_3 G_2 + P_4 P_3 P_2 G_1 + P_4 P_3 P_2 P_1 C_0 = G_1^* + P_1^* C_0 \\
 C_8 &= G_8 + P_8 G_7 + P_8 P_7 G_6 + P_8 P_7 P_6 G_5 + P_8 P_7 P_6 P_5 C_4 = G_2^* + P_2^* C_4 \\
 &\dots\dots
 \end{aligned}$$

再改写成：

$$C_4 = G_1^* + P_1^* C_0$$

$$C_8 = G_2^* + P_2^* G_1^* + P_2^* P_1^* C_0$$

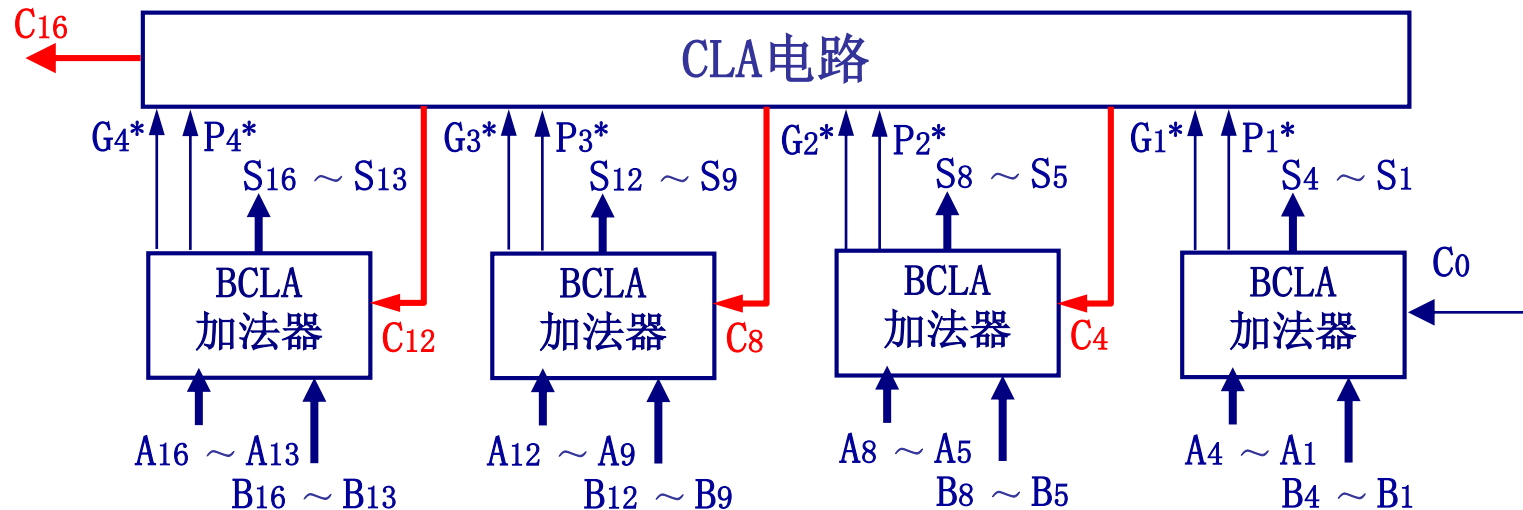
$$C_{12} = G_3^* + P_3^* G_2^* + P_3^* P_2^* G_1^* + P_3^* P_2^* P_1^* C_0$$

$$C_{16} = G_4^* + P_4^* G_3^* + P_4^* P_3^* G_2^* + P_4^* P_3^* P_2^* G_1^* + P_4^* P_3^* P_2^* P_1^* C_0$$

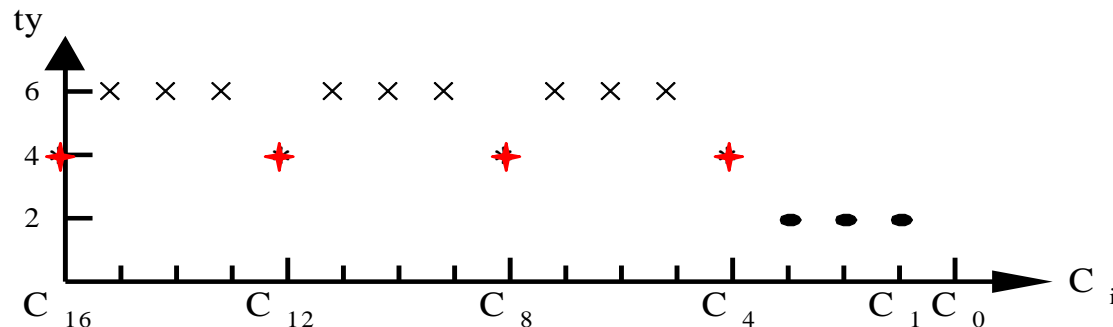
—— 与上述 CLA 的表达式相同，只是变量名称不同而已。

G_i^* 称为组进位产生函数， P_i^* 称为组进位传递函数。

成组先行进位电路BCLA，是CLA电路的修改，增加了 G_i^* 和 P_i^* 逻辑输出，去掉了其中最高位并行进位的输出。



两级先行进位时间图



? 思考:

1. 在全加器里进位输出表达式 $C_i = A_i B_i + (A_i \oplus B_i) C_{i-1}$ 为什么可以简化为 $C_i = A_i B_i + (A_i + B_i) C_{i-1}$?
2. 先行进位主要是解决什么问题, 采用什么设计思路?

§ 4.2 定点加减运算

定点数的加减运算包括原码、补码和反码3种带符号数的加减运算，其中补码加减运算实现起来最方便。

4.2.1 原码加减运算

运算规则：

- (1) 符号位单独处理，用绝对值(即尾数)参加运算；
- (2) 原码同号相加或异号相减时，尾数作加法运算，得和的原码尾数(需要判溢出)，最后结果取被加(减)数的数符；
- (3) 原码异号相加或同号相减时，尾数作减法运算，不需判溢出，减的结果为负时应把结果变补才是原码的尾数，结果的原码按是否够减决定结果数符。

注：减法运算 $A-B$ 可转换为加法运算 $A+[B]_{\text{变补}}$ 。

4.2.2 补码加减运算

1. 运算方法

符号位参加运算。

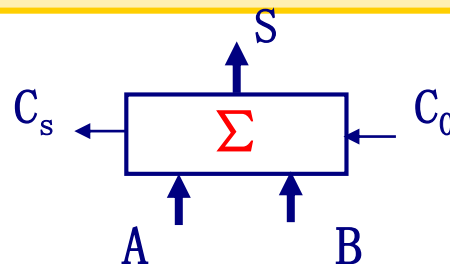
补码运算的两个重要公式：

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

$$[X-Y]_{\text{补}} = [X]_{\text{补}} + [-Y]_{\text{补}} \pmod{M} \quad \text{----- 减法可转化为加法运算!}$$

$$\boxed{} = \boxed{[X]_{\text{补}}} + \boxed{[\overline{Y}]_{\text{补}}} + \boxed{\text{“末位1”}} \pmod{M}$$

$$S = A + B + C_0$$



2. 运算溢出判断

1) 根据运算前后数的符号位判断

设：操作数 $A = A_s, A_1 A_2 \dots A_n$

操作数 $B = B_s, B_1 B_2 \dots B_n$

其和为： $S = S_s, S_1 S_2 \dots S_n$

则：溢出条件 $V_f = \underbrace{A_s \cdot B_s \cdot \overline{S_s}} + \underbrace{\overline{A_s} \cdot \overline{B_s} \cdot S_s}$

两个负数相加，结果却为正数

两个正数相加，结果却为负数

2. 运算溢出判断

2) 采用进位位判断

设： C_s 为符号位产生的进位，

C_1 为最高数值位产生的进位

则：溢出条件 $V_f = \overline{C_s} \cdot C_1 + C_s \cdot \overline{C_1} = C_s \oplus C_1$

3) 运算时补码采用双符号位（变形补码）

$$\text{溢出条件 } V_f = \overline{S_{s1}} \cdot S_{s2} + S_{s1} \cdot \overline{S_{s2}} = S_{s1} \oplus S_{s2}$$

$S_{s1}S_{s2}=01$ ，结果正溢

$S_{s1}S_{s2}=10$ ，结果负溢

左边的符号位 S_{s1} 叫做真符。

当结果的双符号位 $S_{s1}S_{s2}$ 为00或11时，值用补码能够表示。

例：① $A=0.1011$, $B=-0.1110$, 求 $[A+B]_{\text{补}}$ 。

$$\begin{array}{r}
 [A]_{\text{补}} \quad 00.1011 \\
 +) [B]_{\text{补}} \quad 11.0010 \\
 \hline
 \quad \quad 11.1101 \\
 \therefore [A+B]_{\text{补}} = 1.1101
 \end{array}$$

② $A=0.1011$, $B=-0.0010$, 求 $[A-B]_{\text{补}}$ 。

$$\begin{array}{r}
 [A]_{\text{补}} \quad 00.1011 \\
 +) [-B]_{\text{补}} \quad 00.0010 \\
 \hline
 \quad \quad 00.1101 \\
 \therefore [A-B]_{\text{补}} = 0.1101
 \end{array}$$

③ $A=-0.1101, B=-0.1010$, 求 $[A+B]_{\text{补}}$ 。

$$\begin{array}{r}
 [A]_{\text{补}} \quad 11.0011 \\
 +) [B]_{\text{补}} \quad 11.0110 \\
 \hline
 10.1001
 \end{array}$$

$\therefore [A+B]_{\text{补}}$ 负溢出!

④ $A=0.1101, B=-0.1010$, 求 $[A-B]_{\text{补}}$ 。

$$\begin{array}{r}
 [A]_{\text{补}} \quad 00.1101 \\
 +) [-B]_{\text{补}} \quad 00.1010 \\
 \hline
 01.0111
 \end{array}$$

$\therefore [A-B]_{\text{补}}$ 正溢出!

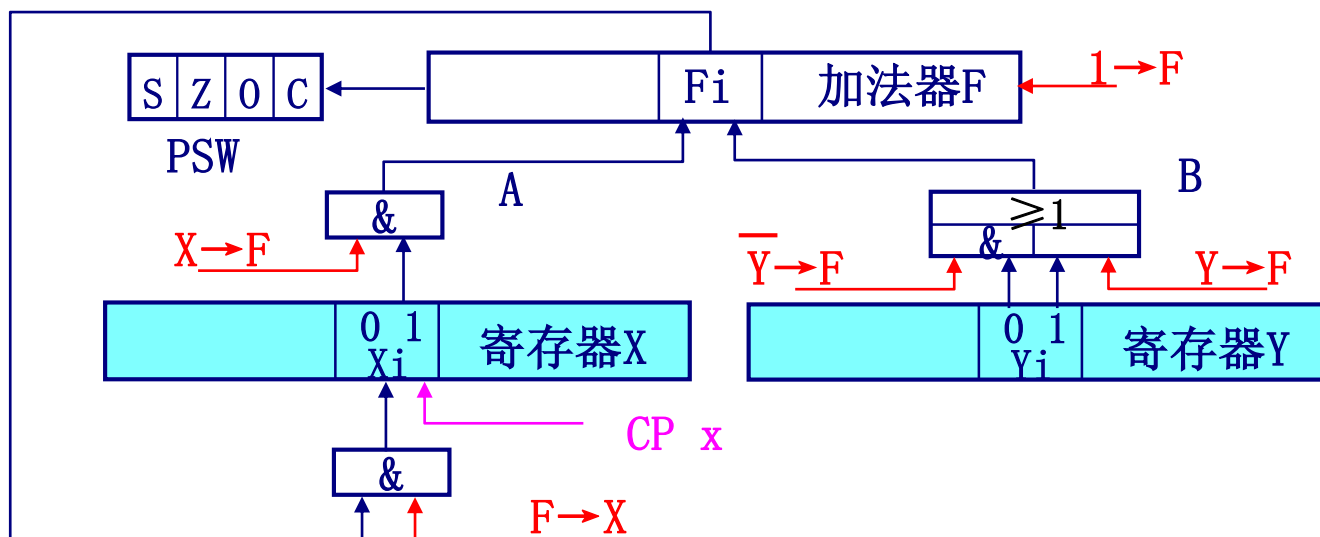
习题：P135 2, 4, 5,

6(用恒舍法, 并求 $[2Y]_{\text{补}}$), 7, 8, 10(2)(4), 11

习题：P120 2, 4, 5,

6(用恒舍法, 并求 $[2Y]_{\text{补}}$), 7, 8, 10(2)(4), 12

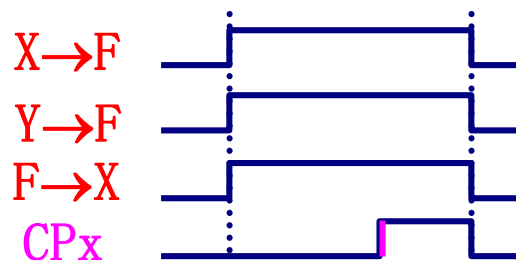
4.2.3 补码定点加减运算的实现



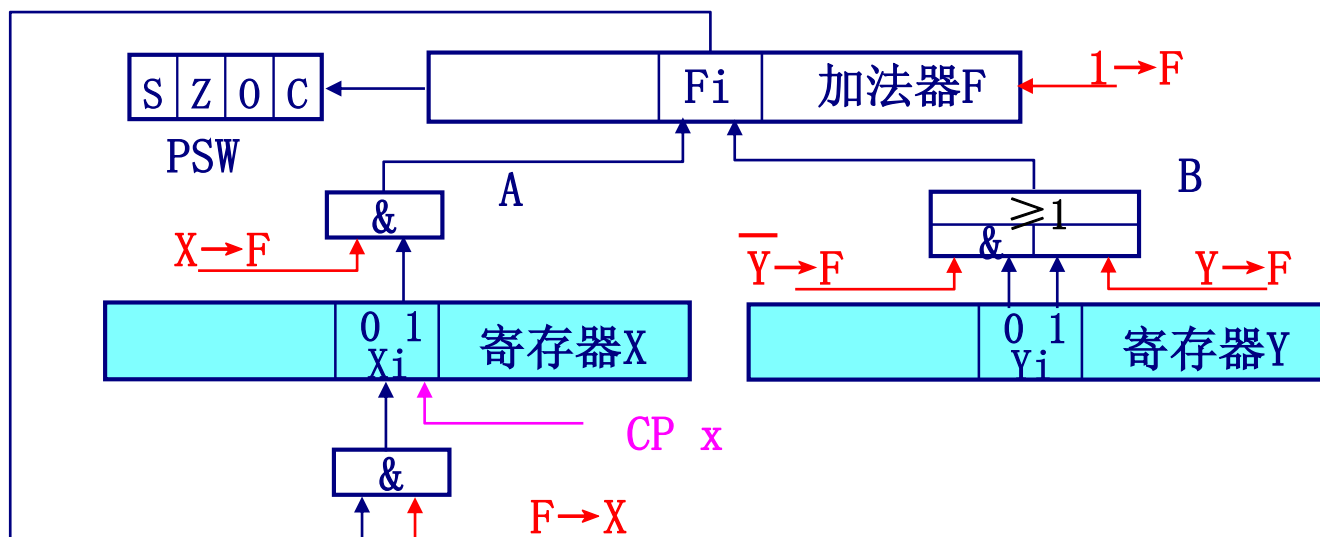
加法运算 (即 $X \leftarrow X + Y$)，应给该运算器提供如下控制信号：

$X \rightarrow F$; $Y \rightarrow F$; $F \rightarrow X$; CP_x (其它控制信号为低电平)

控制信号的波形为：



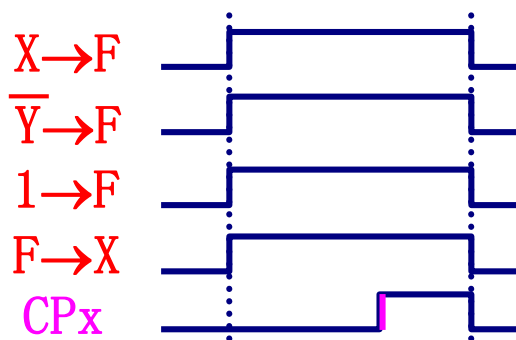
4.2.3 补码定点加减运算的实现



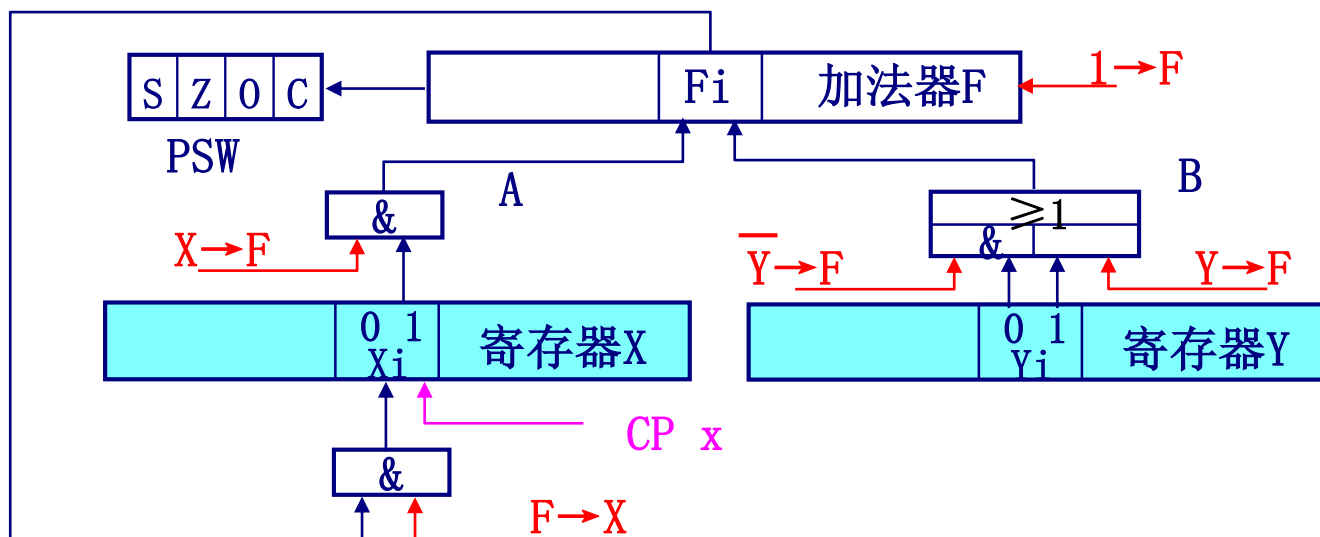
减法运算 (即 $X \leftarrow X - Y$)，应给该运算器提供如下控制信号：

$X \rightarrow F$; $\bar{Y} \rightarrow F$; $F \rightarrow X$; $1 \rightarrow F$; CP_x

控制信号的波形为：



4.2.3 补码定点加减运算的实现

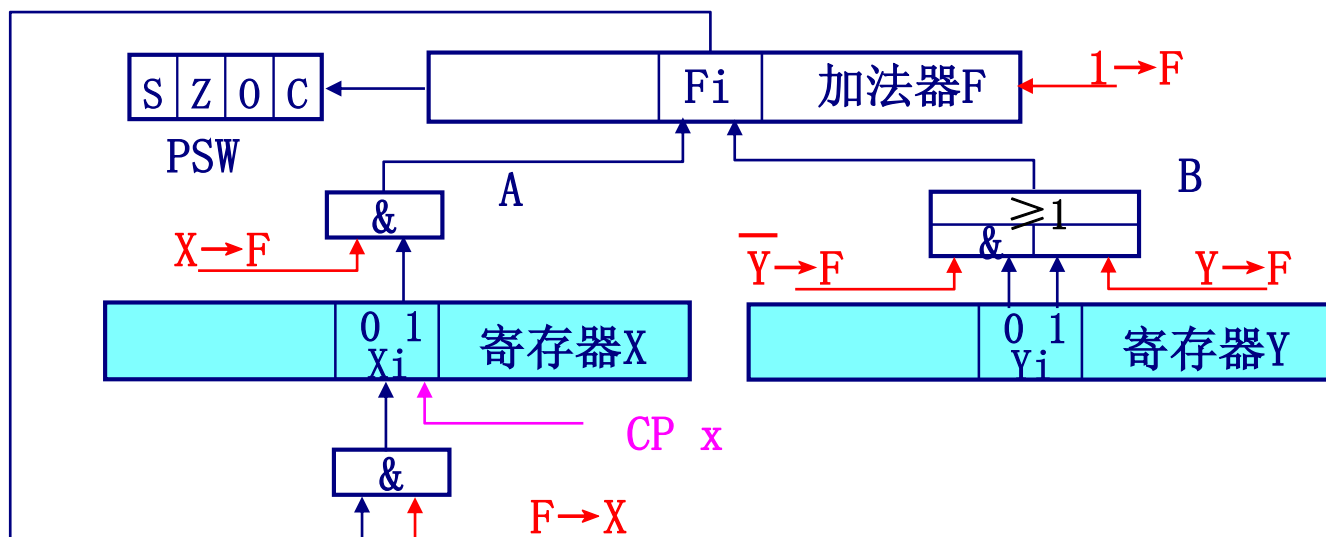



减法运算 (即 $X \leftarrow X - Y$)，应给该运算器提供如下控制信号：

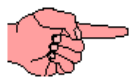
$X \rightarrow F$; $\bar{Y} \rightarrow F$; $F \rightarrow X$; $1 \rightarrow F$; CP_x

注：控制信号“ $1 \rightarrow F$ ”为加法器的最低位的进位输入。
上述的加法或减法运算都是一步完成的。

4.2.3 补码定点加减运算的实现



 计算机硬件所实现的所有的功能，都是通过把它分解成一步一步的基本操作来实现的，这些基本操作称为微操作。每个微操作都是寄存器到寄存器的传送。



硬件电路的微操作，可用寄存器传送语言来描述。

例如：

$R0 \leftarrow R1$ （或 $R1 \rightarrow R0$ ）

表示寄存器R1的数据送给寄存器R0

$AR \leftarrow DB \leftarrow PC$

表示寄存器PC的数据经过数据总线DB送给寄存器AR

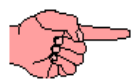
$AR \leftarrow PC$

表示寄存器PC的数据送给寄存器AR

$X \leftarrow X + Y$

表示寄存器X和寄存器Y相加后结果送给寄存器X

寄存器传送语言，与指令系统功能描述的写法有些差异！



注意：

微操作 “ $AR \leftarrow DB \leftarrow PC$ ”如果分成如下两步：

(1) $DB \leftarrow PC$

(2) $AR \leftarrow DB$

——错了！

这是因为DB没有记忆功能，上述两步各自都不能构成微操作！

课外扩展阅读：寄存器传送语言（英文材料 Chapter 4）

§ 4.3 带符号数的移位和舍入操作

带符号数的移位指算术移位。

算术左移1位即乘以2操作，算术右移1位即除以2操作，移位的规则与码制有关。

4.3.1 带符号数的移位操作

1. 原码的移位规则

符号位均不变，空出位一律以“0”补入。

例：移位前 $1X_1X_2 \dots X_{n-1}X_n$

移位后 $1X_2X_3 \dots X_n \underline{0}$ （若 X_1 为1则溢出）

右移后 $1\underline{0}X_1 \dots X_{n-2}X_{n-1}$ （ X_n 丢弃）

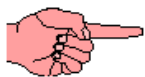
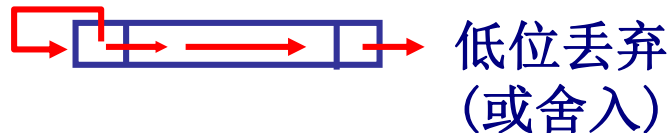
2. 补码的移位规则

左移：符号位不变，所有位左移，末位补入“0”

如果所有位左移后符号位变了，则溢出！



右移：符号位不变，连同符号位右移



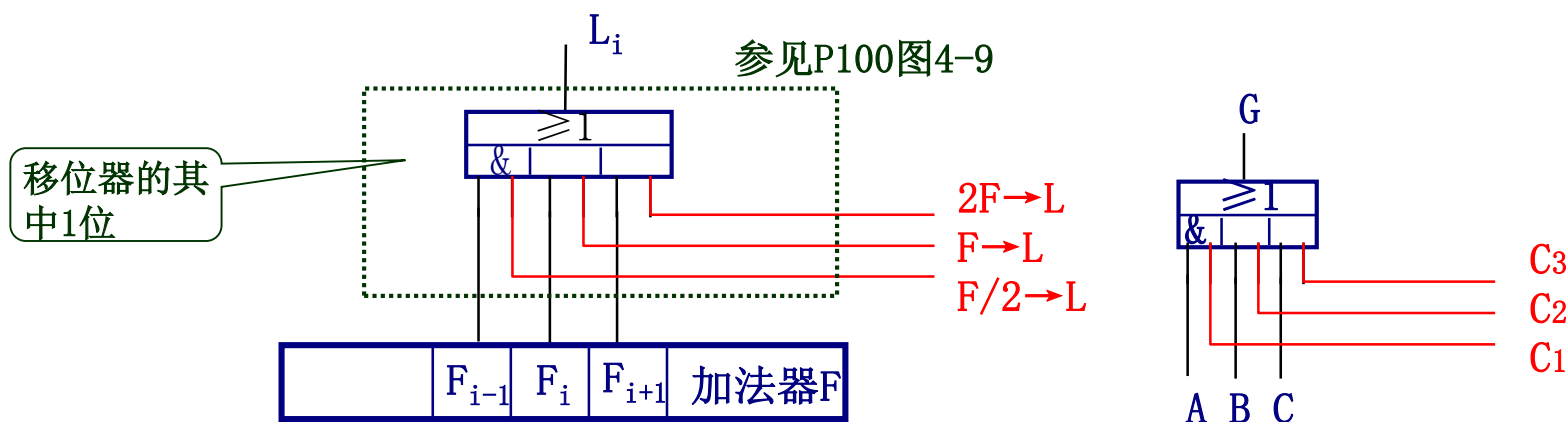
注意： 机器数移位后总的位数不变！

3. 移位功能的实现

(1) 由移位寄存器来实现

(2) 用移位器来实现

移位器可以由多路选择器构成，常接在加法器的输出端，可以实现直传（不移位）、左斜一位送（左移一位）和右斜一位送（右移一位）的功能。



注意：多路选择器是没有记忆功能的！

移位操作除了算术移位外，还有逻辑移位和循环移位。

逻辑移位：逻辑左移



逻辑右移



循环移位：小循环(左移为例)



大循环(左移为例)



4.3.2 带符号数的舍入操作

算术右移时常见的舍入方法有：

(1) 恒舍法（切断）

末尾多余部分的位一律舍去。

(2) 恒置1法（冯·诺依曼舍入法）

不论末尾舍去的是什么，都把保留部分的最低位置1。

(3) 下舍上入法

（0舍1入）

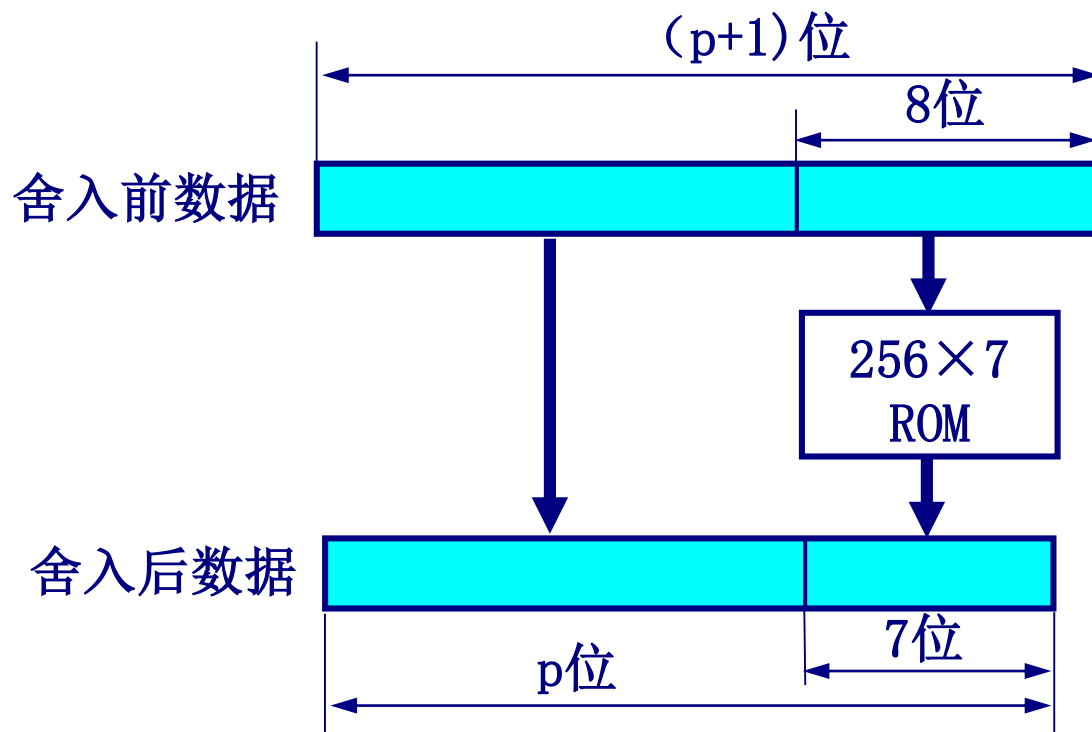
(4) 查表舍入法（ROM舍入法）

各种舍入方法的误差统计数值不同。

4.3.2 带符号数的舍入操作

查表舍入的一般方法：

当 K 位数据的高 $K-1$ 位为全“1”时按恒舍法填入 $K-1$ 位全“1”，否则其余单元都按下舍上入法来填其内容。



思考：计算机硬件中的微操作有些什么特点？

习题：P136 2, 4, 5, 6(用恒舍法, 并求出 $[2Y]_{\text{补}}$), 7
8, 10(2)(4), 11

习题：P120 2, 4, 5, 6(用恒舍法, 并求出 $[2Y]_{\text{补}}$), 7
8, 10(2)(4), 12



§ 4.4 定点乘法运算

乘除的实现途径:

- 1) 软件实现 (低档机中只提供加、减、移位等指令)
- 2) 在加减运算器基础上增加少量电路实现(有乘除指令)
- 3) 设置专用的高速阵列乘除运算器。

4.4.1 原码一位乘法

被乘数、乘数用原码表示，所求的积也用原码表示。

处理方法：符号位单独处理

$$P_S = X_S \oplus Y_S$$

绝对值相乘得积的尾数 $|P| = |X| \times |Y|$

手算例子

$$0.1101 \times 0.1011 = ?$$

$$\begin{array}{r}
 0.1101 \\
 \times 0.1011 \\
 \hline
 1101 \\
 1101 \\
 0000 \\
 1101 \\
 \hline
 0.10001111
 \end{array}$$

多个数相加：

- 1) 可将 k 位乘转换为“k 次累加与右移”
- 2) 用阵列结构的乘法器

4.4.1 原码一位乘法

手算例子

$$0.1101 \times 0.1011 = ?$$

$$\begin{array}{r}
 0.1101 \\
 \times 0.1011 \\
 \hline
 1101 \\
 1101 \\
 0000 \\
 1101 \\
 \hline
 0.1001111
 \end{array}$$

0000

积的累加和初值为0



4.4.1 原码一位乘法

手算例子

$$0.1101 \times 0.1011 = ?$$

$$\begin{array}{r}
 0.1101 \\
 \times 0.1011 \\
 \hline
 1101 \\
 0000 \\
 1101 \\
 \hline
 0.1001111
 \end{array}$$



4.4.1 原码一位乘法

手算例子

$$0.1101 \times 0.1011 = ?$$

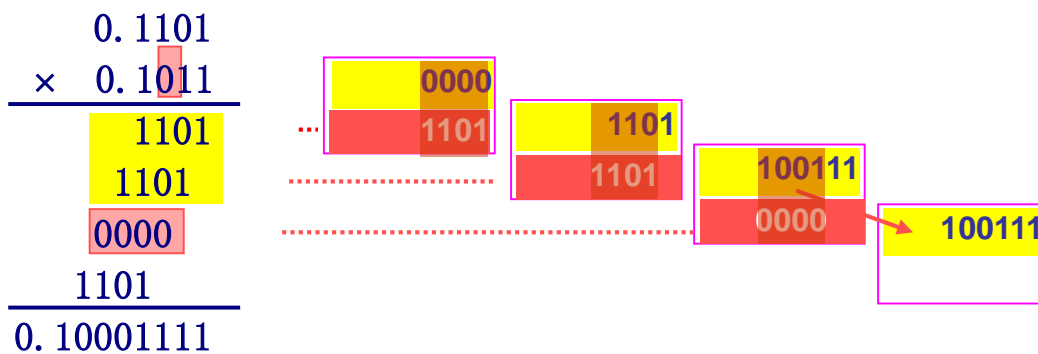
0.1101	
× 0.1011	
1101	
1101	
0000	
1101	
0.10001111	

... 0000 1101 1101 100111

4.4.1 原码一位乘法

手算例子

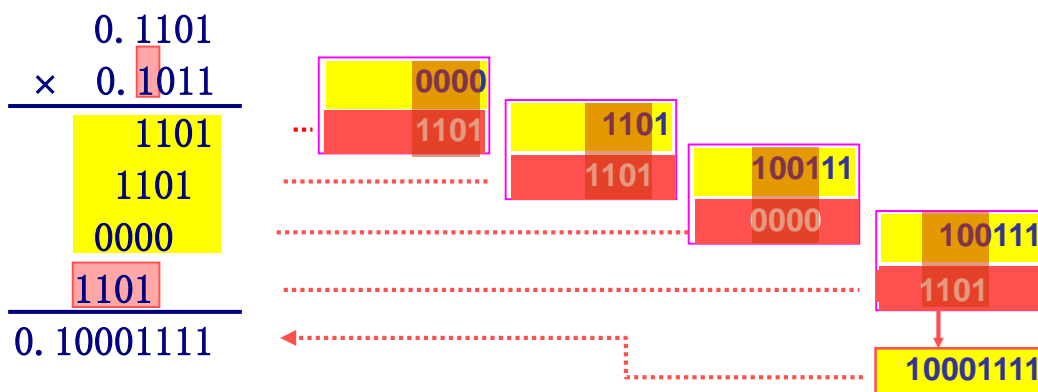
$$0.1101 \times 0.1011 = ?$$



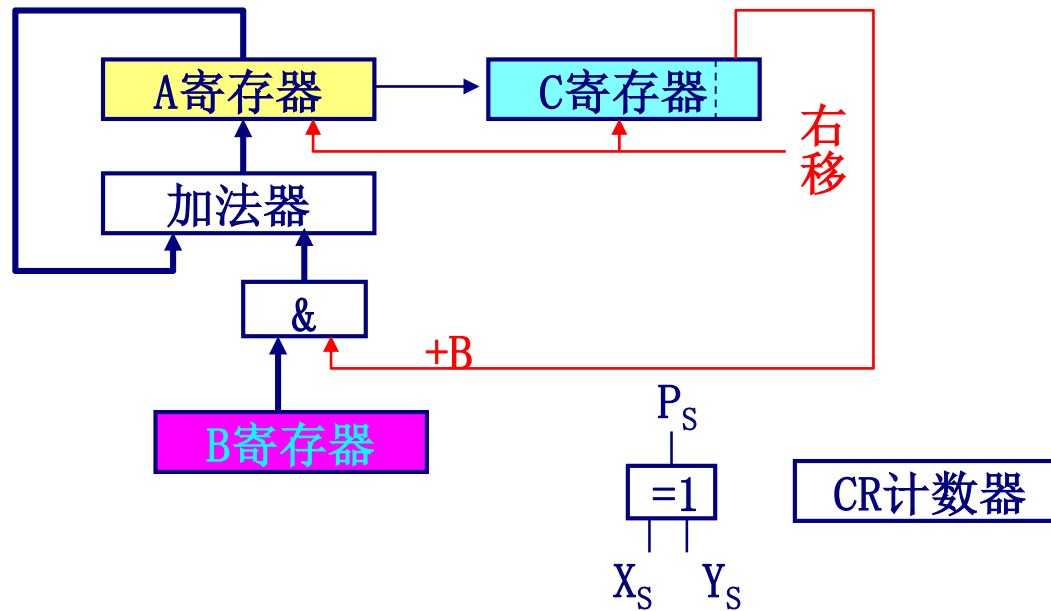
4.4.1 原码一位乘法

手算例子

$$0.1101 \times 0.1011 = ?$$



原码一位乘法框图：（P104图4-12）

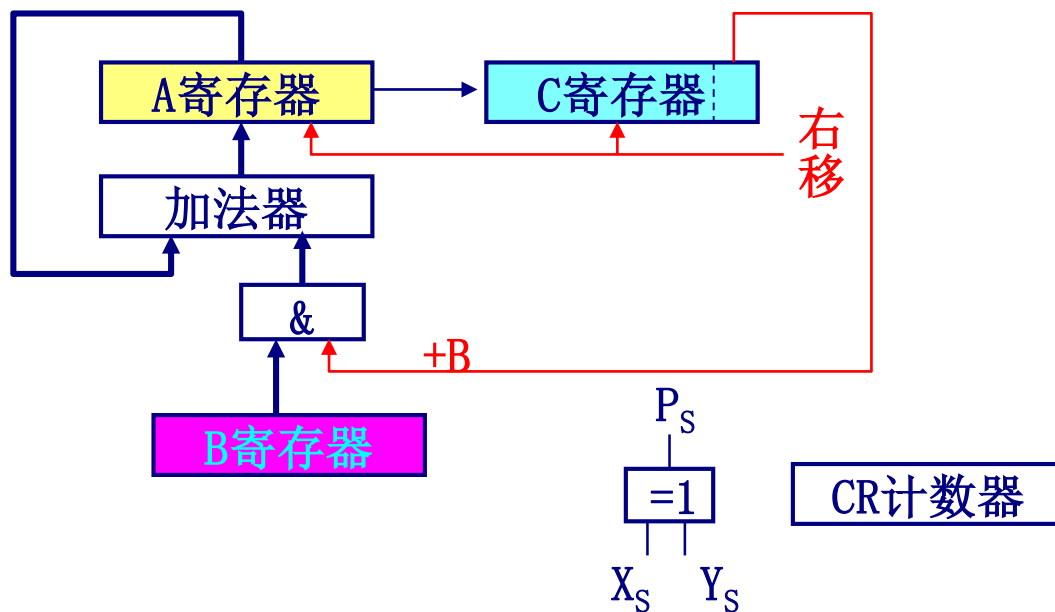


B寄存器 —— 被乘数

C寄存器 —— 乘数。运算结束后乘数不再保留，改为存放乘积的低位部分

A寄存器 —— 初值为 0，存放部分积或最后乘积的高位部分

原码一位乘法框图：（P104图4-12）



加法器：n+2位的

与门: $n+2$ 个, (控制是加被乘数还是加0)

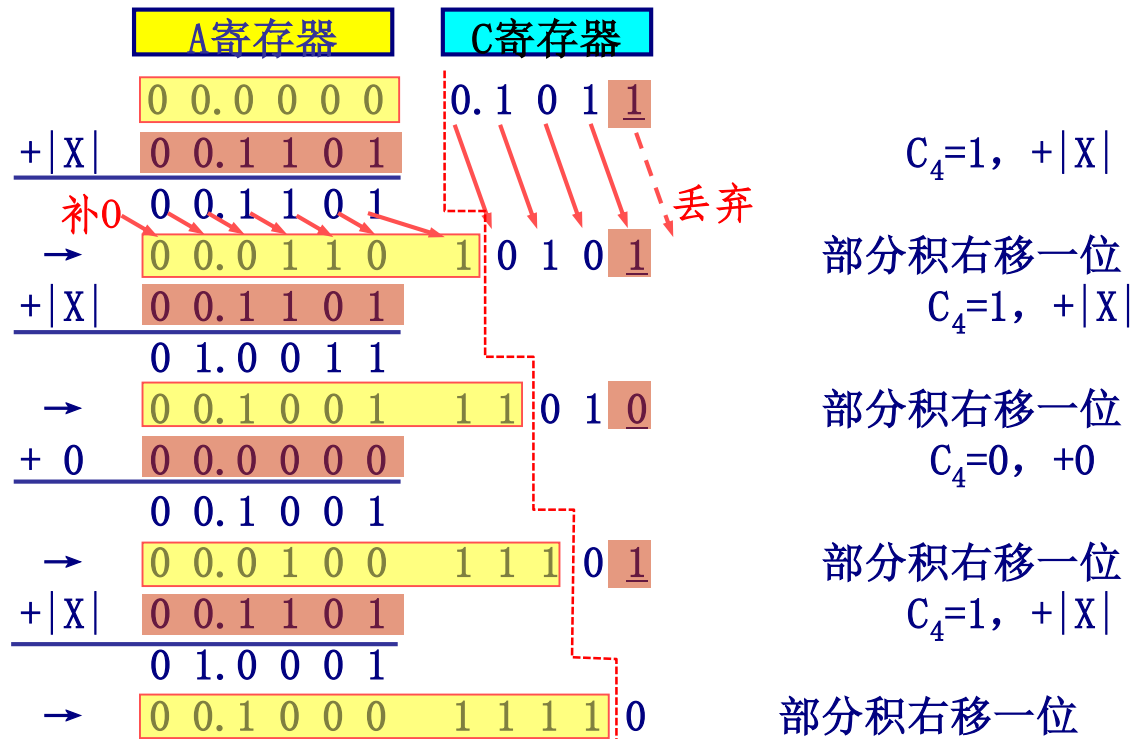
异或门：1个，处理符号位

计算机的计算过程:

例: 已知: $X=0.1101$, $Y=-0.1011$, 求: $X \times Y$ 。

$|X|=00.1101 \rightarrow B$, $|Y|= .1011 \rightarrow C$, $0 \rightarrow A$

$$\begin{array}{r}
 0.1101 \\
 \times 0.1011 \\
 \hline
 1101 \\
 1101 \\
 0000 \\
 1101 \\
 \hline
 0.10001111
 \end{array}$$



$C_4=1, +|X|$

部分积右移一位

$C_4=1, +|X|$

部分积右移一位

$C_4=0, +0$

部分积右移一位

$C_4=1, +|X|$

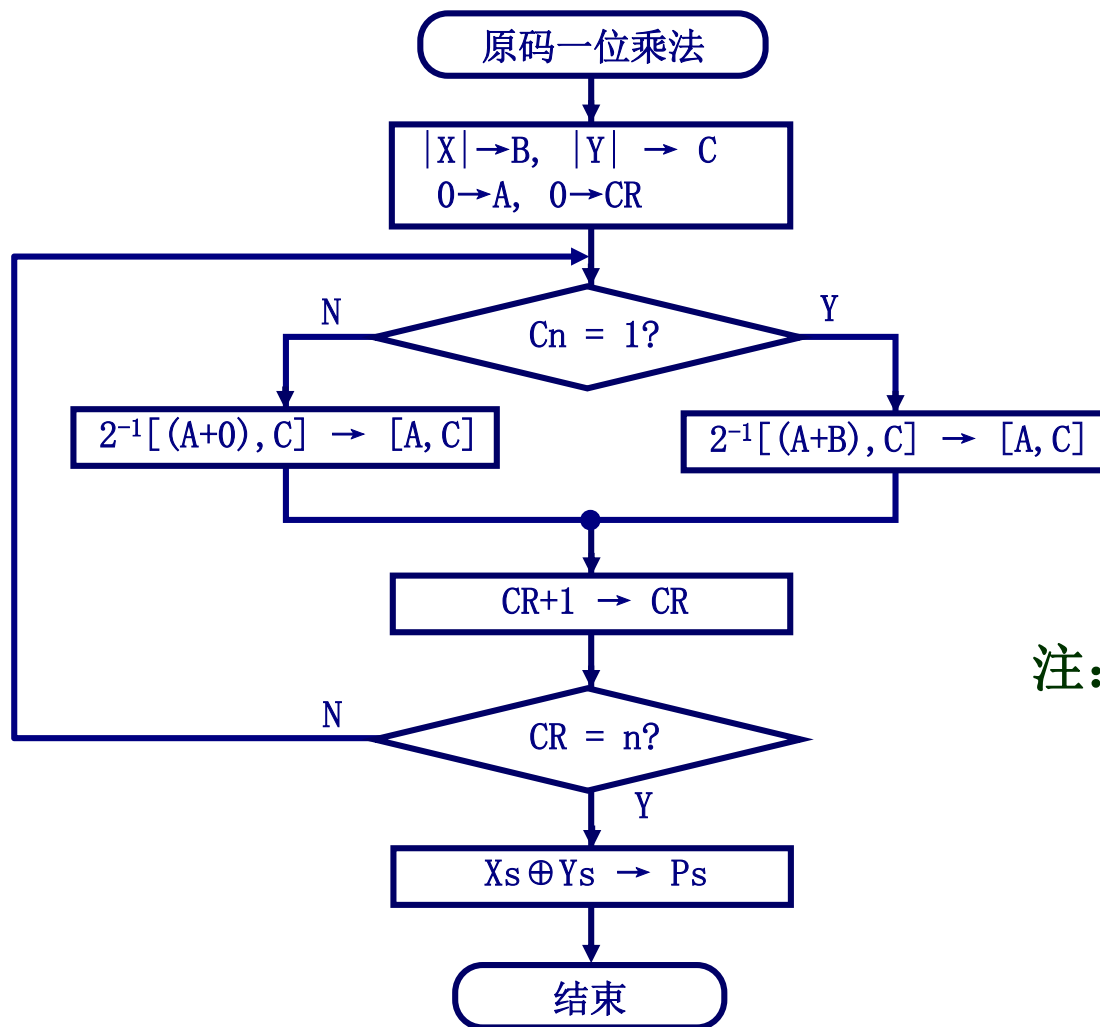
部分积右移一位

$$\therefore P_s = X_s \oplus Y_s = 0 \oplus 1 = 1$$

$$\therefore X \times Y = -0.10001111$$

B寄存器

原码一位乘法流程图 (P104图4-11)



注:

B -- 被乘数

C -- 乘数

[A, C] -- 部分积

4.4.2 补码一位乘法

设 被乘数 $[X]_{\text{补}} = X_s \cdot X_1 X_2 \dots X_n$

乘数 $[Y]_{\text{补}} = Y_s \cdot Y_1 Y_2 \dots Y_n$

则 $[X \times Y]_{\text{补}} = [X]_{\text{补}} \times (0.Y_1 Y_2 \dots Y_n) + [-X]_{\text{补}} \times Y_s$ (证明略)

1. 校正法

校正法是将 $[X]_{\text{补}}$ 和 $[Y]_{\text{补}}$ 按原码规则运算，所得结果再根据乘数的符号再加以校正，从而得到正确的 $[X \times Y]_{\text{补}}$ 。

补码乘法校正法在乘数为负数需要多一步校正，控制起来要复杂一些。

$$[X \times Y]_{\text{补}} = [X]_{\text{补}} \times (0.Y_1 Y_2 \dots Y_n) + [-X]_{\text{补}} \times Y_s$$

2. 比较法——Booth乘法

$$\begin{aligned}
[X \times Y]_{\text{补}} &= [X]_{\text{补}} \times (0.Y_1 Y_2 \dots Y_n) + [-X]_{\text{补}} \times Y_s \\
&= [X]_{\text{补}} \times (Y_1 2^{-1} + Y_2 2^{-2} + \dots + Y_n 2^{-n}) + [-X]_{\text{补}} \times Y_s \\
&= [X]_{\text{补}} \times [-Y_s + Y_1 (2^0 - 2^{-1}) + Y_2 (2^{-1} - 2^{-2}) + \dots + Y_n (2^{-(n-1)} - 2^{-n}) + 0] \\
&= [X]_{\text{补}} \times [(Y_1 - Y_s) 2^0 + (Y_2 - Y_1) 2^{-1} + \dots + (Y_{n+1} - Y_n) 2^{-n}] \quad | Y_{n+1} = 0
\end{aligned}$$

$$2^{-1} = 2^0 - 2^{-1}$$



2. 比较法——Booth乘法

$$\begin{aligned}
[X \times Y]_{\text{补}} &= [X]_{\text{补}} \times (0.Y_1 Y_2 \dots Y_n) + [-X]_{\text{补}} \times Y_s \\
&= [X]_{\text{补}} \times (Y_1 2^{-1} + Y_2 2^{-2} + \dots + Y_n 2^{-n}) + [-X]_{\text{补}} \times Y_s \\
&= [X]_{\text{补}} \times [-Y_s + Y_1 (2^0 - 2^{-1}) + Y_2 (2^{-1} - 2^{-2}) + \dots + Y_n (2^{-(n-1)} - 2^{-n}) + 0] \\
&= [X]_{\text{补}} \times [(Y_1 - Y_s) 2^0 + (Y_2 - Y_1) 2^{-1} + \dots + (Y_{n+1} - Y_n) 2^{-n}] \quad | Y_{n+1} = 0 \\
&= [X]_{\text{补}} (Y_1 - Y_s) + [X]_{\text{补}} (Y_2 - Y_1) 2^{-1} + \dots + [X]_{\text{补}} (Y_{n+1} - Y_n) 2^{-n} \\
&= [X]_{\text{补}} (Y_1 - Y_s) + 2^{-1} ([X]_{\text{补}} (Y_2 - Y_1) + 2^{-1} (\dots + 2^{-1} ([X]_{\text{补}} (Y_{n+1} - Y_n) + 0) \dots))
\end{aligned}$$

$$2^{-1} = 2^0 - 2^{-1}$$

注: $S = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 \quad (t = 2^{-1})$

需7次乘4次加!

而 $S = a_0 + t(a_1 + t(a_2 + t(a_3 + t(a_4 + 0))))$

只需4次乘5次加! 而且运算很有规律(加乘的循环)



2. 比较法——Booth乘法

可用递推公式计算：

$$[Z_0]_{\text{补}} = 0$$

$$[Z_1]_{\text{补}} = 2^{-1} \{ [Z_0]_{\text{补}} + (Y_{n+1} - Y_n) [X]_{\text{补}} \}$$

$$[Z_2]_{\text{补}} = 2^{-1} \{ [Z_1]_{\text{补}} + (Y_n - Y_{n-1}) [X]_{\text{补}} \}$$

.....

$$[Z_n]_{\text{补}} = 2^{-1} \{ [Z_{n-1}]_{\text{补}} + (Y_2 - Y_1) [X]_{\text{补}} \}$$

$$\therefore [X \times Y]_{\text{补}} = [Z_n]_{\text{补}} + (Y_1 - Y_s) [X]_{\text{补}}$$

-- 初始部分积

-- 第1次累加并右移之后的部分积

-- 第2次累加并右移之后的部分积

-- 第n次累加并右移之后的部分积

-- 最后1次累加，但不移位！

2. 比较法——Booth乘法

可用递推公式计算：

$$[Z_0]_{\text{补}} = 0$$

-- 初始部分积

$$[Z_1]_{\text{补}} = 2^{-1} \{ [Z_0]_{\text{补}} + (Y_{n+1} - Y_n) [X]_{\text{补}} \}$$

-- 第1次累加并右移之后的部分积

$$[Z_2]_{\text{补}} = 2^{-1} \{ [Z_1]_{\text{补}} + (Y_n - Y_{n-1}) [X]_{\text{补}} \}$$

-- 第2次累加并右移之后的部分积

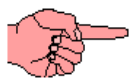
.....

$$[Z_n]_{\text{补}} = 2^{-1} \{ [Z_{n-1}]_{\text{补}} + (Y_2 - Y_1) [X]_{\text{补}} \}$$

-- 第n次累加并右移之后的部分积

$$\therefore [X \times Y]_{\text{补}} = [Z_n]_{\text{补}} + (Y_1 - Y_s) [X]_{\text{补}}$$

-- 最后1次累加，但不移位！



由此可见，Booth乘法可以把符号位和数值位同等对待，一起参加运算。运算共需做n+1次累加(用双符号位运算)，n次移位，第n+1次不移位。

Booth乘法运算规则:

判断位	Y_n	Y_{n+1}	操作
	0	0	原部分积+0, 再右移一位
	0	1	原部分积+[X] _补 , 再右移一位
	1	0	原部分积加[-X] _补 , 再右移一位
	1	1	原部分积+0, 再右移一位

例：已知 $X=-0.1101$ ， $Y=0.1011$ ；求 $X \times Y$ 。（P106 例9）

$[X]_{\text{补}}=11.0011 \rightarrow \text{B寄存器}$ ， $[Y]_{\text{补}}=0.1011 \rightarrow \text{C寄存器}$ ， $0 \rightarrow \text{A寄存器(累加器)}$

$[-X]_{\text{补}}=00.1101$

Booth乘法示例

	A寄存器	C寄存器
	0 0 0 0 0 0	0 1 0 1 1 0
$+ [-X]_{\text{补}}$	0 0 1 1 0 1	
\rightarrow	0 0 1 1 0 1	1 0 1 0 1 1
$+0$	0 0 0 0 0 0	
\rightarrow	0 0 0 1 1 0	0 1 0 1 0 1
$+ [X]_{\text{补}}$	1 1 0 0 1 1	
\rightarrow	1 1 0 1 1 0	0 0 1 0 1 0
$+ [-X]_{\text{补}}$	0 0 1 1 0 1	
\rightarrow	0 0 1 0 0 0	0 0 0 1 0 1
$+ [X]_{\text{补}}$	1 1 0 0 1 1	
	1 1 0 1 1 1	

说明

$C_4C_5=10$ ， $+ [-X]_{\text{补}}$

部分积右移一位

$C_4C_5=11$ ， $+0$

部分积右移一位

$C_4C_5=01$ ， $+ [X]_{\text{补}}$

部分积右移一位

$C_4C_5=10$ ， $+ [-X]_{\text{补}}$

部分积右移一位

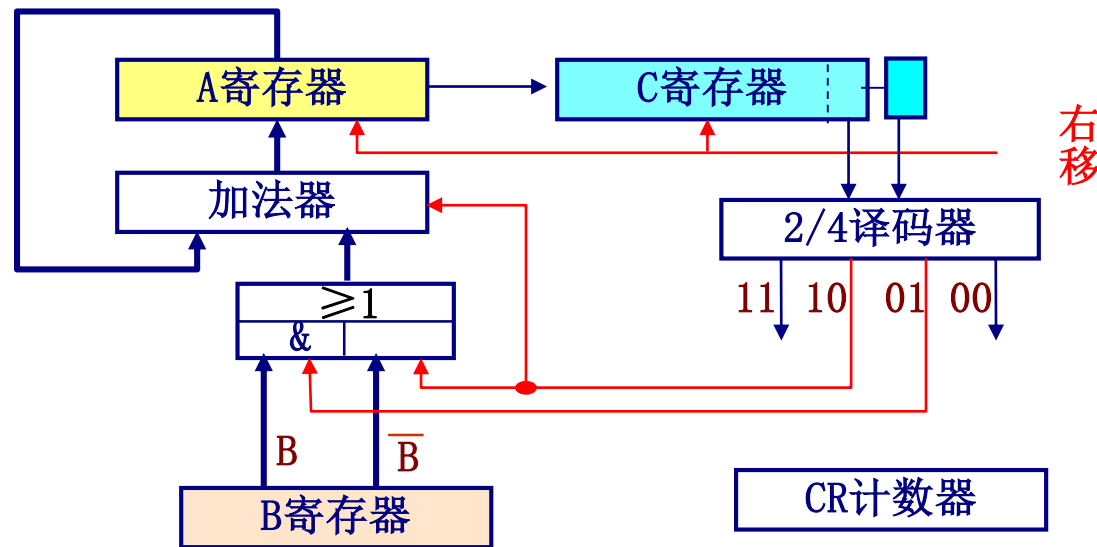
$C_4C_5=01$ ， $+ [X]_{\text{补}}$

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

$\therefore X \times Y = -0.10001111$



3. Booth乘法运算的实现



思考: P136 9

习题: P136 2, 4, 5, 6(用恒舍法, 并求 $[2Y]_{\text{补}}$), 7,
8, 10(2) (4), 11

习题: P120 2, 4, 5, 6(用恒舍法, 并求 $[2Y]_{\text{补}}$), 7,
8, 10(2) (4), 12

4.4.3 补码两位乘法

每次处理乘数中的两位，从而使乘法的速度提高了一倍。
可理解为将Booth乘法的每两步累加移位合并为一次来做。

$$\text{Booth中: } [Z_i]_{\text{补}} = 2^{-1}([Z_{i-1}]_{\text{补}} + (Y_{k+1} - Y_k)[X]_{\text{补}})$$

$$[Z_{i+1}]_{\text{补}} = 2^{-1}([Z_i]_{\text{补}} + (Y_k - Y_{k-1})[X]_{\text{补}})$$

⇓ 合并后

$$\text{两位乘: } [Z_{i+1}]_{\text{补}} = 2^{-2}([Z_{i-1}]_{\text{补}} + (Y_{k+1} + Y_k - 2Y_{k-1})[X]_{\text{补}})$$

右移2位

判断位

被乘数和部分积取3个符号位；

乘数的数值位n为偶数时取2个符号位，共需作 $(n/2)+1$ 次累加， $n/2$ 次移位（最后一次不移位）；当n为奇数时，乘数只需1个符号位，共需 $(n+1)/2$ 次累加和移位，但最后一次仅移一位。

4.4.4 阵列乘法器

由高速乘法模块组成，以提高乘法运算的速度。

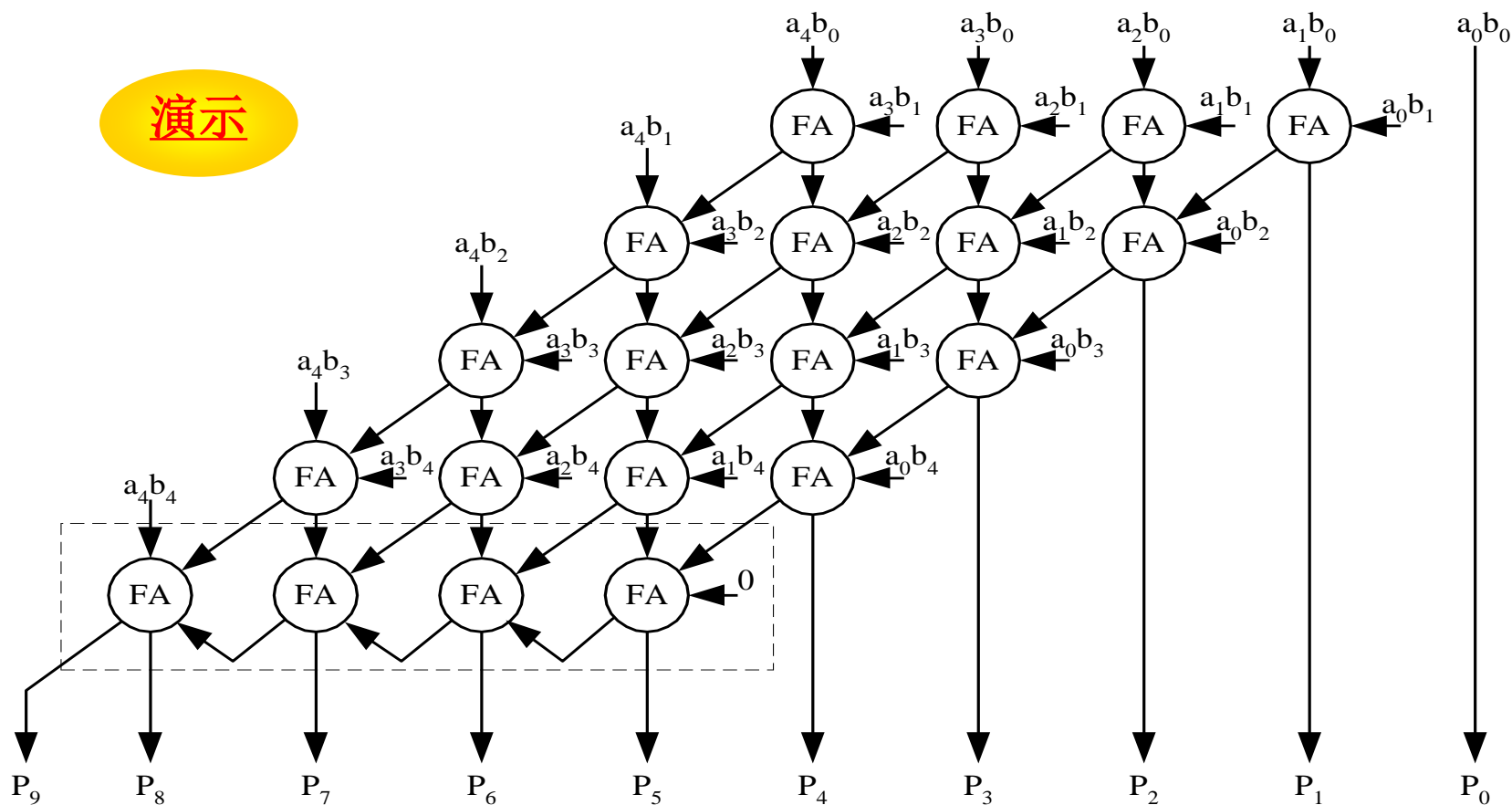
1. 不带符号的阵列乘法器

例如： $m=n=5$ ，

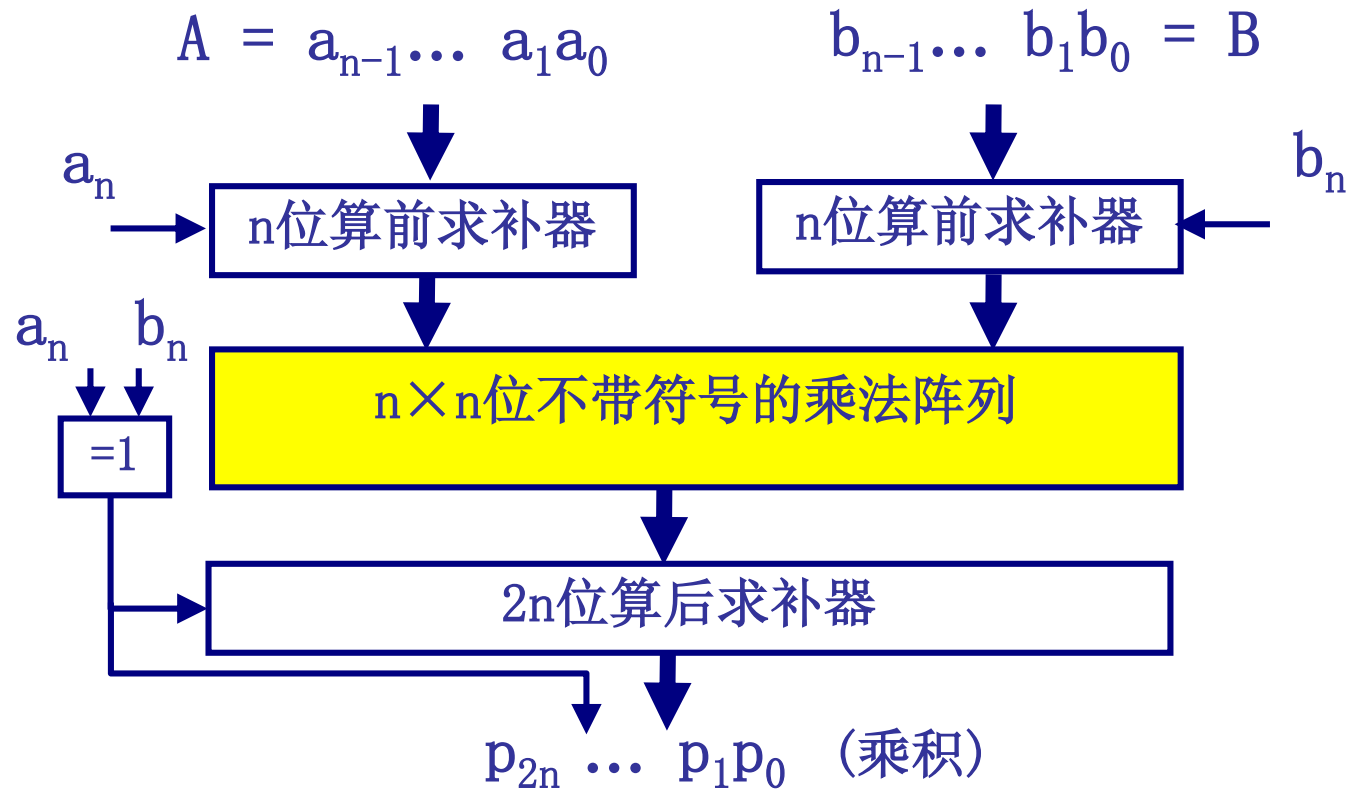
					a_4	a_3	a_2	a_1	a_0	$=A$
				\times	b_4	b_3	b_2	b_1	b_0	$=B$
					a_4b_0	a_3b_0	a_2b_0	a_1b_0	a_0b_0	
				a_4b_1	a_3b_1	a_2b_1	a_1b_1	a_0b_1		
		a_4b_2	a_3b_2	a_2b_2	a_1b_2	a_0b_2				
	a_4b_3	a_3b_3	a_2b_3	a_1b_3	a_0b_3					
$+$	a_4b_4	a_3b_4	a_2b_4	a_1b_4	a_0b_4					
P_9	P_8	P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0	$=$

1. 不带符号的阵列乘法器

演示



2. 带符号的阵列乘法器（间接法）



? 思考： 求补器的电路设计？

§ 4.5 定点除法运算

引例: $X=0.1011$, $Y=0.1101$

用手算求商C和余数R

	0.1 1 0 1		
0.1 1 0 1	0.1 0 1 1 0 0 0 0		$R_0 = X$
-)	0.0 1 1 0 1	0.0 1 0 0 1 0	$-2^{-1} Y$
	0.0 1 0 0 1 0	1 1 0 1	$-2^{-2} Y$
-)	1 0 1 0 0	1 1 0 1	$-2^{-4} Y$
-)	0.0 0 0 0 0 1 1 1		$R_4 \text{ (余数)}$

结果: $C = 0.1101$, $R = R_4 = 0.000001111$

问题:

① 机器如何判断够减?

i. 逻辑比较器

ii. 先减1次! 余数 ≥ 0 则够减, 否则不够减。

§ 4.5 定点除法运算

引例: $X=0.1011$, $Y=0.1101$

用手算求商C和余数R

	0.1 1 0 1	
0.1 1 0 1	<div style="display: inline-block; text-align: right; padding-right: 10px;">0.1 0 1 1 0 0 0 0</div> <div style="display: inline-block; text-align: right; padding-right: 10px;">0.0 1 1 0 1</div> <div style="display: inline-block; text-align: right; padding-right: 10px;">0.0 1 0 0 1 0</div> <div style="display: inline-block; text-align: right; padding-right: 10px;">1 0 1 0 0</div> <div style="display: inline-block; text-align: right; padding-right: 10px;">1 1 0 1</div> <div style="display: inline-block; text-align: right; padding-right: 10px;">0.0 0 0 0 0 1 1 1</div>	$R_0 = X$ $-2^{-1} Y$ $-2^{-2} Y$ $-2^{-4} Y$ $R_4 \text{ (余数)}$
-)		
-)		
-)		
-)		

结果: $C = 0.1101$, $R = R_4 = 0.000001111$

问题:

② 商符如何确定?

原码: 单独处理 —— 模2加 (异或)

补码: 带符号运算。负值上商为反码。

§ 4.5 定点除法运算

引例: $X=0.1011$, $Y=0.1101$

用手算求商C和余数R

	0.1 1 0 1		
0.1 1 0 1	0.1 0 1 1 0 0 0 0		$R_0 = X$
-)	0.0 1 1 0 1	0.0 1 0 0 1 0	$-2^{-1} Y$
	0.0 1 0 0 1 0	1 1 0 1	$-2^{-2} Y$
-)	1 0 1 0 0	1 1 0 1	$-2^{-4} Y$
-)	0.0 0 0 0 0 1 1 1		$R_4 \text{ (余数)}$

结果: $C = 0.1101$, $R = R_4 = 0.000001111$

问题:

③ 如何进行计算?

- i. 可将n位除转化成若干次“减法—移位”
- ii. 采用阵列除法模块可实现快速除法。

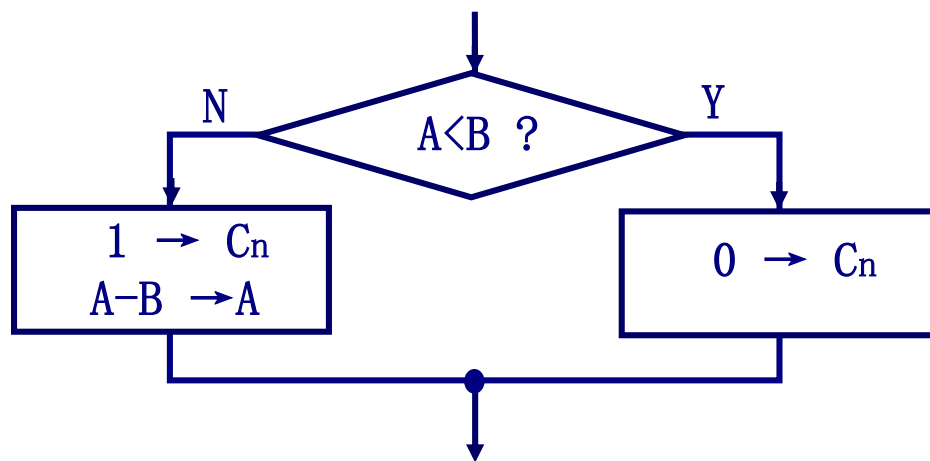
4.5.1 原码除法运算

1. 原码比较法和恢复余数法

(1) 比较法

用比较线路实现比较，计算过程类似于手工运算。

(增加了硬件的代价)

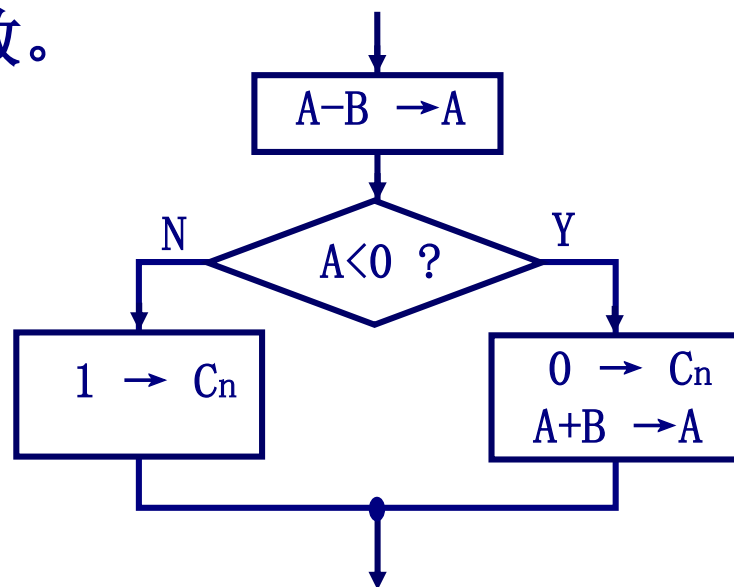


4.5.1 原码除法运算

1. 原码比较法和恢复余数法

(2) 恢复余数法

先做减法试探是否够减：若部分余数为非负表示够减，该位商上“1” 否则表示不够减，该位商上“0”， 并要恢复余数。



缺点：

运算的实际操作次数不固定。

恢复余数降低了除法的执行速度。

4.5.1 原码除法运算

1. 原码比较法和恢复余数法

(2) 恢复余数法

先做减法试探是否够减：若部分余数为非负表示够减，该位商上“1”否则表示不够减，该位商上“0”，并要恢复余数。求第 i 次求商操为：

$$\textcircled{1} \quad r_i \leftarrow 2r_{i-1} - Y; \quad (\text{试减})$$

$$\textcircled{2} \quad \text{若够减} (r_i \geq 0), \text{ 商} 1。$$

否则 $(r_i < 0)$ ，商0、恢复余数

$$r_i \leftarrow r_i + Y \quad (r_i \text{恢复为原来的} 2r_{i-1})$$

2. 原码不恢复余数法（原码加减交替法）

求新余数 r_{i+1} 时，把恢复余数、左移和相减简化为左移相加

$$2(r_i + Y) - Y \rightarrow 2r_i + Y$$

即，若 $r_i > 0$ ，商1，下次作 $2r_i - Y$ 求新余数和商

若 $r_i < 0$ ，商0，下次作 $2r_i + Y$ 求新余数和商

使运算的次数固定。通式表示：

$$r_{i+1} = 2r_i + (1 - 2Q_i)Y$$

? 思考：

1. 从原码的恢复余数法变到加减交替法，其数学推导并不复杂，但电路的实现变得简单了，运算速度也快了，你从中得到了什么启发？

原码加减交替除法示例：

已知： $X = -0.10101$ ， $Y = 0.11110$ ，求： $X \div Y$ 。（P113 例12）

$$|X| = 00.10101 \rightarrow A, \quad |Y| = 00.11110 \rightarrow B, \quad 0 \rightarrow C$$

$$[|Y|]_{\text{变补}} = 11.00010$$



原码加减交替除法示例:

	A寄存器	C寄存器	说明
	0 0.1 0 1 0 1	0.0 0 0 0 0	
$+ [Y]_{\text{变补}}$	1 1.0 0 0 1 0		$- Y $
	1 1.1 0 1 1 1	0.0 0 0 0 0	余数为负, 商 0
←	1 1.0 1 1 1 0		左移一位
$+ Y $	0 0.1 1 1 1 0		$+ Y $
	0 0.0 1 1 0 0	0.0 0 0 0 1	余数为正, 商 1
←	0 0.1 1 0 0 0		左移一位
$+ [Y]_{\text{变补}}$	1 1.0 0 0 1 0		$- Y $
	1 1.1 1 0 1 0	0.0 0 0 1 0	余数为负, 商 0
←	1 1.1 0 1 0 0		左移一位
$+ Y $	0 0.1 1 1 1 0		$+ Y $
	0 0.1 0 0 1 0	0.0 0 1 0 1	余数为正, 商 1
←	0 1.0 0 1 0 0		左移一位
$+ [Y]_{\text{变补}}$	1 1.0 0 0 1 0		$- Y $
	0 0.0 0 1 1 0	0.0 1 0 1 1	余数为正, 商 1
←	0 0.0 1 1 0 0		左移一位
$+ [Y]_{\text{变补}}$	1 1.0 0 0 1 0		$- Y $
	1 1.0 1 1 1 0	0.1 0 1 1 0	余数为负, 商 0
$+ Y $	0 0.1 1 1 1 0		$+ Y $
	0 0.0 1 1 0 0		

商符: $Q_s = X_s \oplus Y_s = 1 \oplus 0 = 1$

∴ 商 = -0.10110

余数 = -0.01100×2^{-5} (余数与被除数同号)



原码加减交替除法示例:

A寄存器		C寄存器	说明
	0 0.1 0 1 0 1	0.0 0 0 0 0	
$+ [Y]_{\text{变补}}$	1 1.0 0 0 1 0		$- Y $
	1 1.1 0 1 1 1	0.0 0 0 0 0	余数为负, 商 0
←	1 1.0 1 1 1 0		左移一位
$+ Y $	0 0.1 1 1 1 0		$+ Y $
	0 0.0 1 1 0 0	0.0 0 0 0 1	余数为正, 商 1
←	0 0.1 1 0 0 0		左移一位
$+ [Y]_{\text{变补}}$	1 1.0 0 0 1 0		$- Y $
	1 1.1 1 0 1 0	0.0 0 0 1 0	余数为负, 商 0
←	1 1.1 0 1 0 0		左移一位
$+ Y $	0 0.1 1 1 1 0		$+ Y $
	0 0.1 0 0 1 0	0.0 0 1 0 1	余数为正, 商 1
←	0 1.0 0 1 0 0		左移一位
$+ [Y]_{\text{变补}}$	1 1.0 0 0 1 0		$- Y $
	0 0.0 0 1 1 0	0.0 1 0 1 1	余数为正, 商 1
←	0 0.0 1 1 0 0		左移一位
$+ [Y]_{\text{变补}}$	1 1.0 0 0 1 0		$- Y $
	1 1.0 1 1 1 0	0.1 0 1 1 0	余数为负, 商 0
$+ Y $	0 0.1 1 1 1 0		$+ Y $
	0 0.0 1 1 0 0		
商符: $Q_s = X_s \oplus Y_s = 1 \oplus 0 = 1$			

即, $X/Y = -0.10110 + (-0.01100 \times 2^{-5})/0.11110$

或写成 $X/Y = -(0.10110 + (0.01100 \times 2^{-5})/0.11110)$ (余数项的分子和分母都取正)



4.5.2 补码加减交替除法运算

1. 求商及新余数的运算规则

$[r_{i+1}]_{\text{补}}$ 与 $[Y]_{\text{补}}$ 同号：上商1，下一步作左移、相减；
异号：上商0，下一步作左移、相加。

末位商恒置“1”

此法操作简单，易于实现。运算的最大误差为 2^{-n}

2. 补码加减交替除法示例

例：已知 $X=0.1000$, $Y=-0.1010$, 求 $X \div Y$ 。 (P116 例13)

$$[X]_{\text{补}} = 00.1000 \rightarrow A, \quad [Y]_{\text{补}} = 11.0110 \rightarrow B, \quad 0 \rightarrow C$$

$$[-Y]_{\text{补}} = 00.1010$$

2. 补码加减交替除法示例

	A	C	说明
	0 0.1 0 0 0	0.0 0 0 0	
$+[Y]_{\text{补}}$	1 1.0 1 1 0		$[X]_{\text{补}}$ 与 $[Y]_{\text{补}}$ 异号, $+[Y]_{\text{补}}$
	1 1.1 1 1 0	0.0 0 0 1	$[r_i]_{\text{补}}$ 与 $[Y]_{\text{补}}$ 同号, 商 1
←	1 1.1 1 0 0		左移一位
$+[-Y]_{\text{补}}$	0 0.1 0 1 0		$+[Y]_{\text{补}}$
	0 0.0 1 1 0	0.0 0 1 0	$[r_i]_{\text{补}}$ 与 $[Y]_{\text{补}}$ 异号, 商 0
←	0 0.1 1 0 0		左移一位
$+[Y]_{\text{补}}$	1 1.0 1 1 0		$+[Y]_{\text{补}}$
	0 0.0 0 1 0	0.0 1 0 0	$[r_i]_{\text{补}}$ 与 $[Y]_{\text{补}}$ 异号, 商 0
←	0 0.0 1 0 0		左移一位
$+[Y]_{\text{补}}$	1 1.0 1 1 0		$+[Y]_{\text{补}}$
	1 1.1 0 1 0	0.1 0 0 1	$[r_i]_{\text{补}}$ 与 $[Y]_{\text{补}}$ 同号, 商 1
←	1 1.0 1 0 0		左移一位
$+[-Y]_{\text{补}}$	0 0.1 0 1 0		$+[Y]_{\text{补}}$
	1 1.1 1 1 0	1.0 0 1 1	末位恒置 1

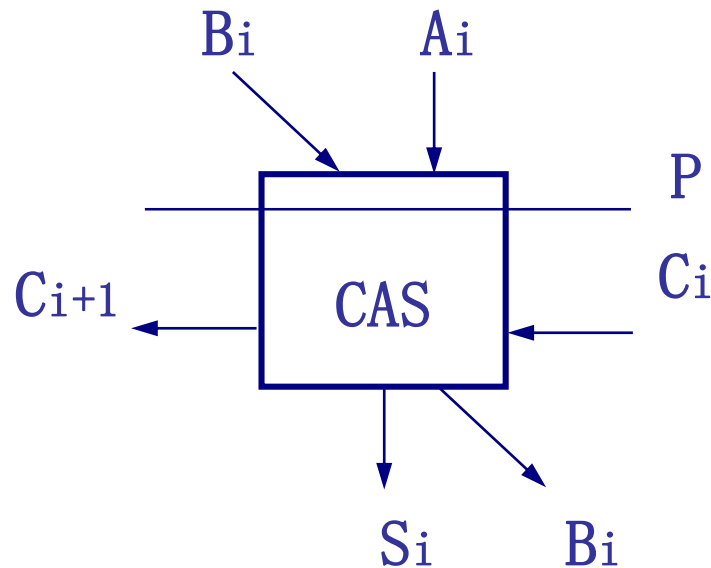
加减共 $n+1$ 次

$[\text{商}]_{\text{补}} = 1.0011$



4.5.3 阵列除法器

1. 可控的加法/减法单元CAS单元

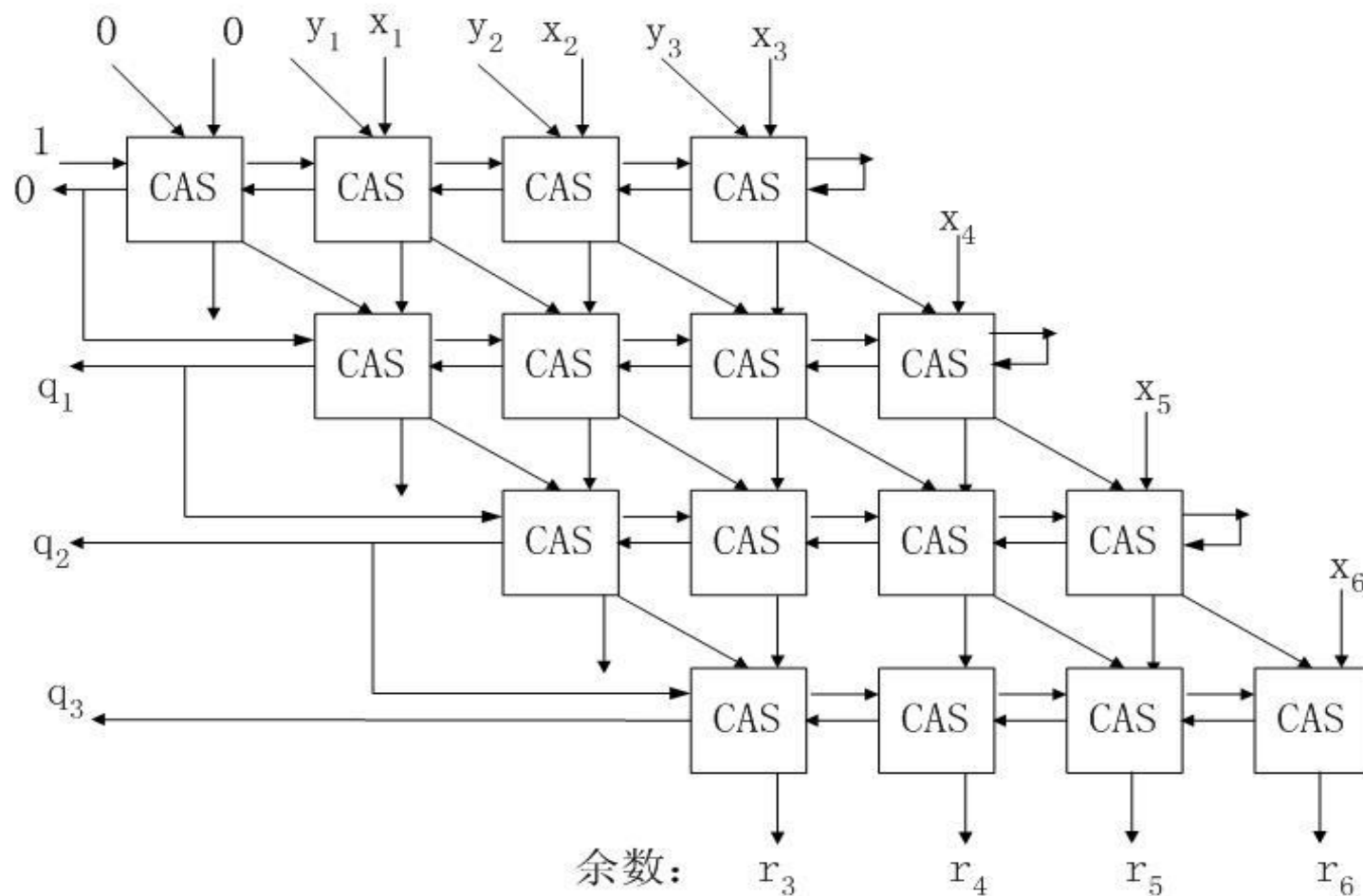


$P=0$, 作加法运算

$P=1$, 作减法运算

4.5.3 阵列除法器

2. 阵列除法原理图



习题：P136 2, 4, 5, 6 (用恒舍法, 并求 $[2Y]_{\text{补}}$),
7, 8, 10(2) (4), 11

习题：P120 2, 4, 5, 6 (用恒舍法, 并求 $[2Y]_{\text{补}}$), 7,
8, 10(2) (4), 12

课堂练习：已知 $X = -0.1000$, $Y = 0.1101$, 用原码、补码加减交替除法计算 $[X/Y]_{\text{原}}$ 、 $[X/Y]_{\text{补}}$, 要求写出规范的运算过程。

§ 4.6 规格化浮点运算

引例：十进制数运算

$$\begin{aligned} & 3.35 \times 10^4 + 9.78 \times 10^5 \\ &= 0.335 \times 10^5 + 9.78 \times 10^5 \\ &= 10.115 \times 10^5 \\ &\approx 1.01 \times 10^5 \end{aligned}$$

设两个非0的规格化浮点数分别为：

$$A = M_A \times 2^{E_A} \quad \text{记为 } (M_A, E_A)$$

$$B = M_B \times 2^{E_B} \quad \text{记为 } (M_B, E_B)$$

阶码为定点整数，尾数为定点小数

∴ 浮点运算可归结为尾数部分和阶码部分的定点运算。

4.6.1 浮点加减运算

规格化浮点数A、B加减运算通式为：

$$\begin{aligned} A \pm B &= (M_A, E_A) \pm (M_B, E_B) \\ &= \begin{cases} (M_A \pm M_B \times 2^{-(E_A - E_B)}, E_A) & E_A > E_B \\ (M_A \times 2^{-(E_B - E_A)} \pm M_B, E_B) & E_A < E_B \end{cases} \end{aligned}$$

1. 浮点数加减运算步骤

(1) 对阶

使两个数的阶码相等（即小数点位置对齐）

①求阶差

$$\Delta E = E_B - E_A$$

②小阶向大阶对阶

- 1) $\Delta E = 0$ 不需再对阶
- 2) $\Delta E > 0$ 则 $E_A \leftarrow E_A + \Delta E$, M_A 算术右移 ΔE 位
- 3) $\Delta E < 0$ 则 $E_B \leftarrow E_B + \Delta E$, M_B 算术右移 ΔE 位

1. 浮点数加减运算步骤

(2) 尾数求和(差)

$$M_A \pm M_B \rightarrow M_C$$

(3) 尾数结果规格化

以2为基数的二进制补码尾数规格化形式为：

$$00.1X...X$$

$$\text{或 } 11.0X...X$$

例如： $00.010 \times 2^{010} \rightarrow 00.100 \times 2^{001}$ （向左规格化）

$10.110 \times 2^{010} \rightarrow 11.011 \times 2^{11}$ （向右规格化）

1. 浮点数加减运算步骤

(4) 舍入

最简单的舍入方法是恒舍法，即无条件的丢掉正常尾数最低位之后的全部数值。

(5) 溢出判断

阶码采用双符号位来判溢，当规格化操作后，阶码的两个符号位不同，则溢出。

以补码表示的阶码为例：

01, 00...0 — 上溢！机器需停止运算，做溢出中断处理。

10, 00...0 — 下溢！机器不做溢出处理，而是按机器零处理。

2. 浮点数加减运算举例

有两浮点数为 $A = 0.101110 \times 2^{-01}$,

$B = - (0.101011) \times 2^{-10}$, 求 $A+B$ 。

数的格式：阶码4位，用移码（偏置值为 2^3 ）表示；

尾数8位，用补码表示，包含一位符号位，即

阶码： 尾 数

$[A]_{\text{浮}} = 0111; 0.1011100$

$[B]_{\text{浮}} = 0110; 1.0101010$

(1) 对阶

$[B]_{\text{浮}}' = 0111; 1.1010101$

(2) 尾数求和

$$\begin{array}{r}
 00.1011100 \\
 +11.1010101 \\
 \hline
 00.0110001
 \end{array}$$

(3) 尾数结果规格化

$$[A+B]_{\text{浮}} = 0110; 0.1100010$$

即 $A+B = (0.110001) \times 2^{-10}$ ，未发生溢出

4.6.2 浮点乘除运算

乘法：阶码相加，尾数相乘

$$X \times Y = (M_X \times M_Y) 2^{E_X + E_Y}$$



注意：

1) 当阶码用移码表示时，

$$[E_A \pm E_B]_{\text{移}} = [E_A]_{\text{移}} \pm [E_B]_{\text{移}} + 2^n \quad (\text{即符号位需修正})$$

2) 结果要规格化。左规时调整阶码后如果发生阶码下溢，则做机器零处理。

4.6.2 浮点乘除运算

除法：阶码相减，尾数相除

$$X/Y = (M_X/M_Y) 2^{E_X-E_Y}$$

具体步骤：预置（判0）

尾数调整（使 $|M_X| < |M_Y|$ ）

求阶差及尾数除法

规格化，判溢出

4.6.3 浮点运算器的实现

主要组成：

(1) 阶码运算部件

完成阶码加、减，以及控制对阶时小阶的尾数右移次数和规格化时对阶码的调整；

(2) 尾数运算部件

用来完成尾数的四则运算以及判断尾数是否已规格化；

(3) 溢出判断电路等

现代计算机可把浮点运算部件做成任选件，或称为**协处理器**。它只能协助主处理器工作，不能单独工作。

§ 4.7 十进制整数的加法运算

一些通用计算机中设有十进制数据表示，可以直接对十进制整数进行算术运算。

实现方法：

- 1) 十进制运算器
- 2) 在二进制运算部件上增加少量设备与通路
- 3) 二进制运算指令+十进制调整指令(微机上用得更多)

下面以一位十进制加法运算为例。

§ 4.7 十进制整数的加法运算

1. 余3码十进制加法器

余3码：用四位二进制数表示一位十进制数，每一个十进制位的值比二进制码多3。

例：3—0110, 5—1000, 9—1100

$$\begin{array}{r}
 3+5=8 \\
 0110 \text{ 余}3 \\
 +) 1000 \text{ 余}3 \\
 \hline
 1110 \text{ 余}6
 \end{array}$$

无进位，结果
应减去3修正！

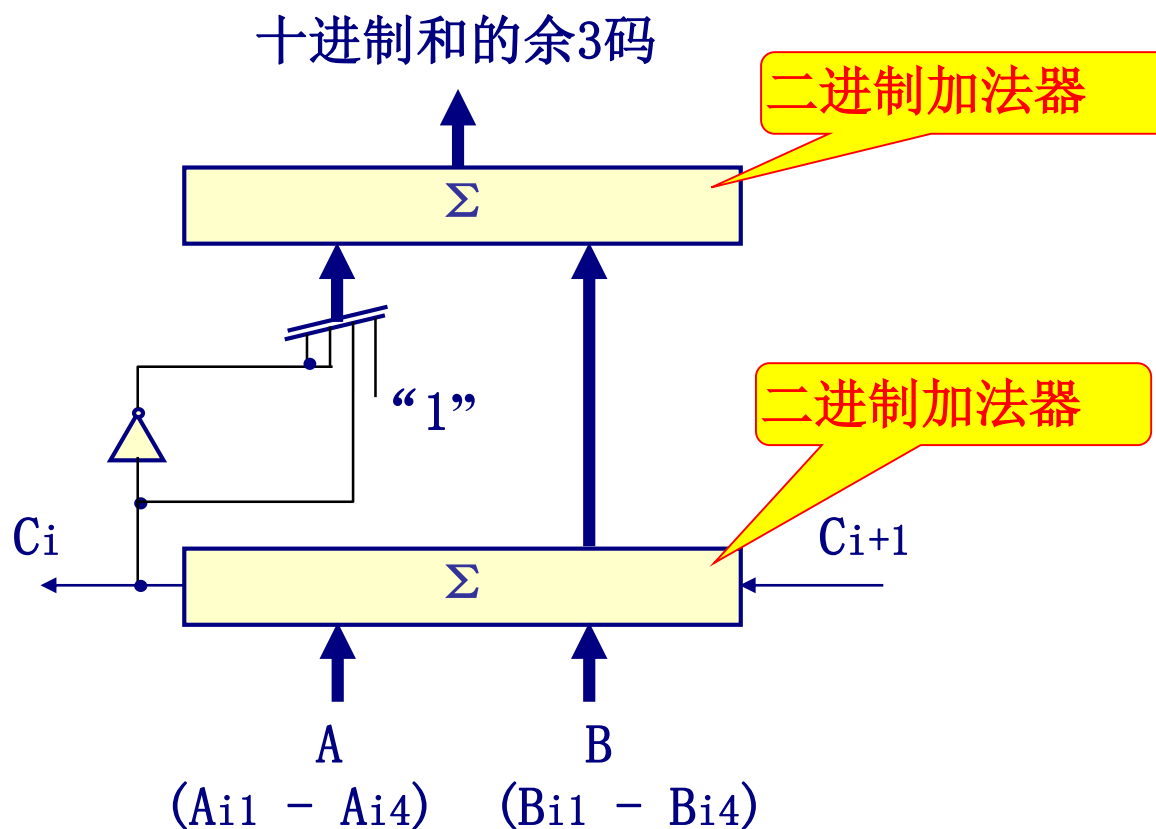
$$\begin{array}{r}
 3+9=12 \\
 0110 \text{ 余}3 \\
 +) 1100 \text{ 余}3 \\
 \hline
 10010 \text{ 余}0
 \end{array}$$

16

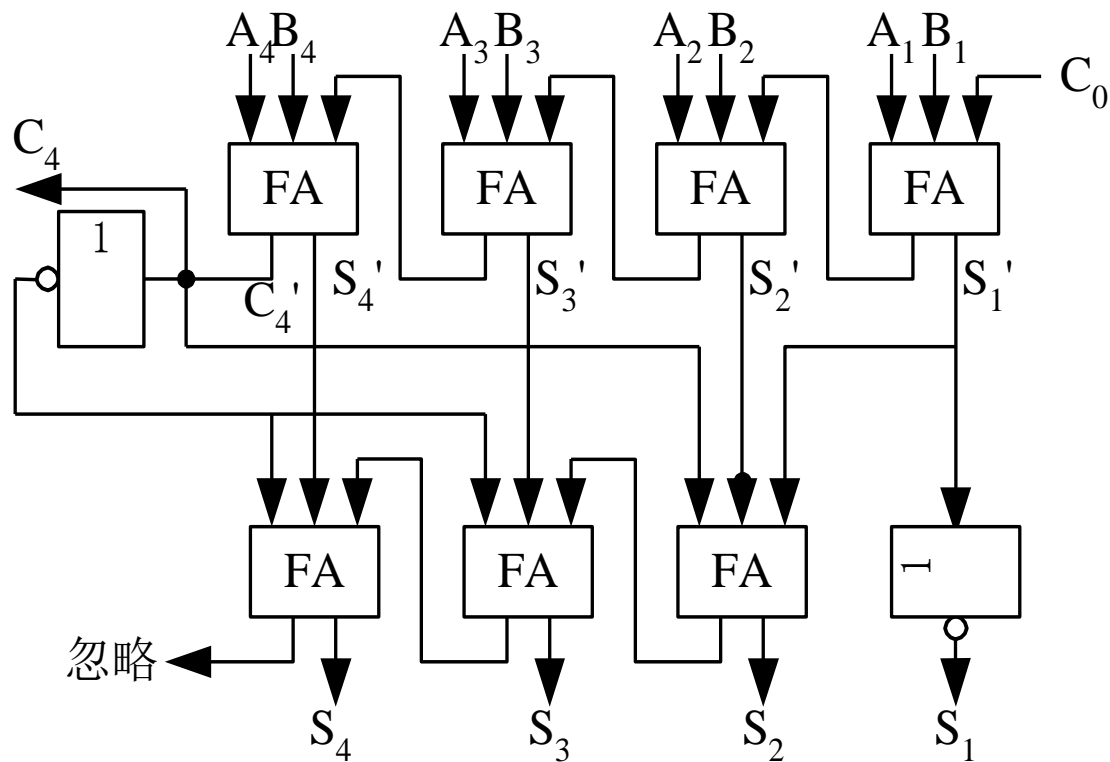
有进位，结果
应加上3修正！

$$\begin{array}{rcl}
 \therefore C=1 \text{ 时} & +3 & 0 \ 0 \ 1 \ 1 \\
 C=0 \text{ 时} & -3 & 1 \ 1 \ 0 \ 1 \\
 & & \uparrow \ \uparrow \ \uparrow \ \uparrow \\
 & & \overline{C} \ \overline{C} \ C \ \text{“1”}
 \end{array}$$

一位余3码加法器原理：



一位余3码加法器：



2. 8421码十进制加法器

8421码:

用四位二进制数表示一位十进制数的另一种方法。

0 — 0000

1 — 0001

.....

9 — 1001

例

① $3+5=8$

$$\begin{array}{r}
 0011 \\
 +) 0101 \\
 \hline
 1000 \text{ -- 结果正确!}
 \end{array}$$

② $3+9=12$

$$\begin{array}{r}
 0011 \\
 +) 1001 \\
 \hline
 1100 \text{ -- 结果超过9, 需校正!} \\
 +) 0110 \text{ -- +6校正} \\
 \hline
 1,0010
 \end{array}$$

③ $8+9=17$

$$\begin{array}{r}
 1000 \\
 +) 1001 \\
 \hline
 1,0001 \text{ -- 有进位, 需+6校正} \\
 +) 0110 \text{ -- +6校正 (为什么?)} \\
 \hline
 1,0111
 \end{array}$$

∴ 8421码加法包括 和校正 和 进位校正:

相加结果 ≤ 9 时, 不需校正;

相加结果 >9 时或有进位时, 和加6校正, 并产生进位。

大于9的 数必须修 正	S_{i4}'	S_{i3}'	S_{i2}'	S_{i1}'	对应的十进制数
	1	0	1	0	10
	1	0	1	1	11
	1	1	0	0	12
	1	1	0	1	13
	1	1	1	0	14
	1	1	1	1	15

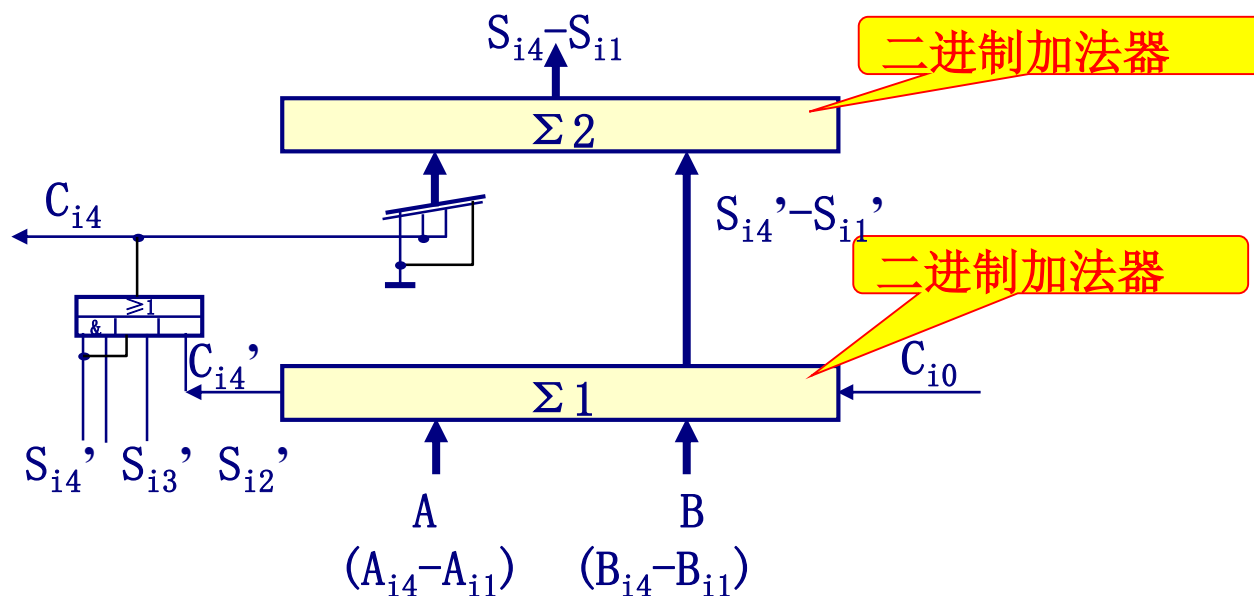
判断二进制加法结果是否大于9的逻辑表达式为：

$$C_{i4} = C_{i4}' + S_{i4}' S_{i3}' + S_{i4}' S_{i2}'$$

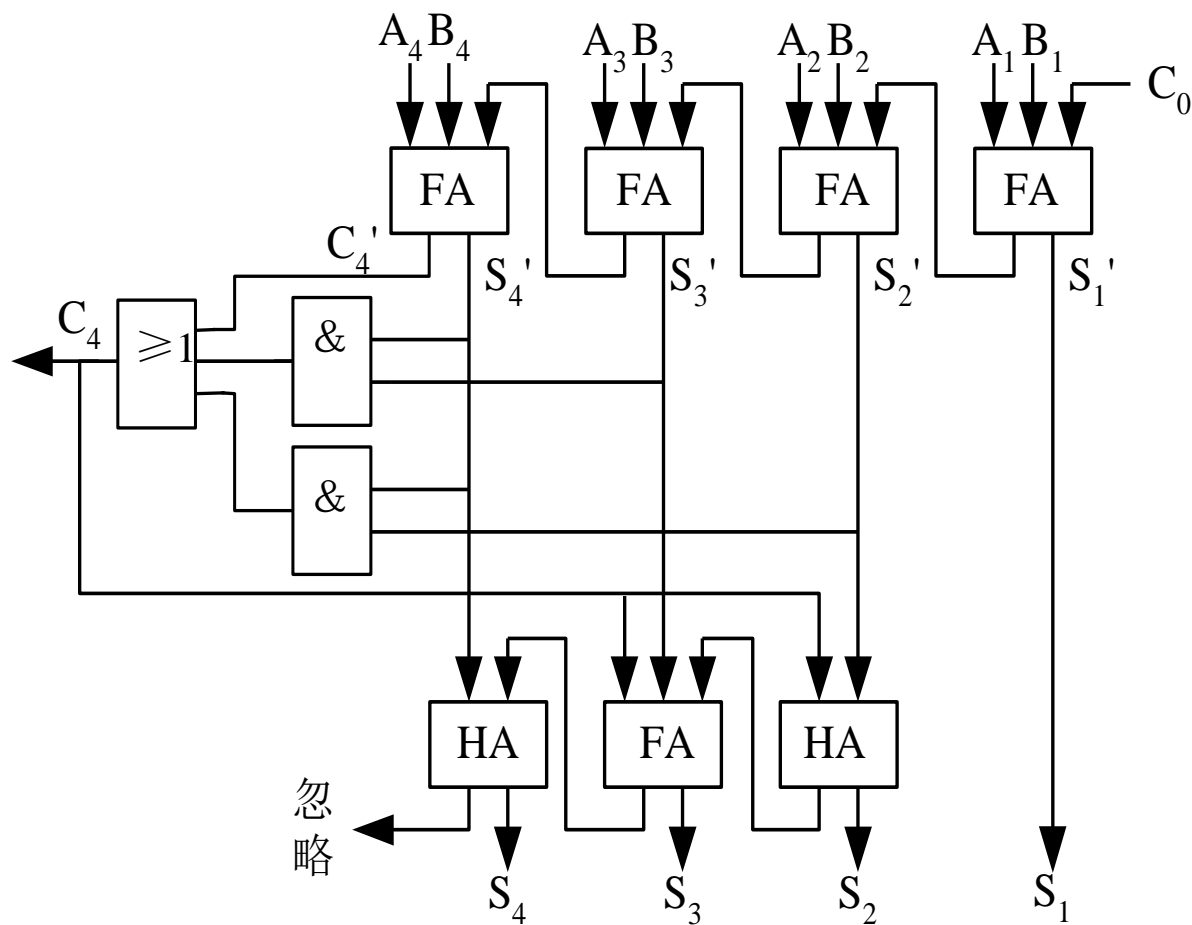
判断二进制加法结果是否大于9的逻辑表达式为：

$$C_{i4} = C_{i4}' + S_{i4}' S_{i3}' + S_{i4}' S_{i2}'$$

一位8421码加法器原理：



一位8421码加法器:



§ 4.8 逻辑运算与实现

逻辑运算特点：位与位之间没有进位或借位的关系。
可用与门、或门、异或门、非门等实现。

	1011	X
\wedge	1101	Y
<hr/>		
	1001	F

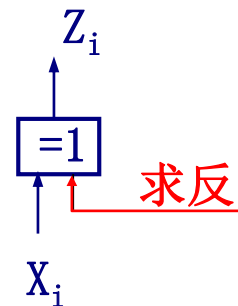
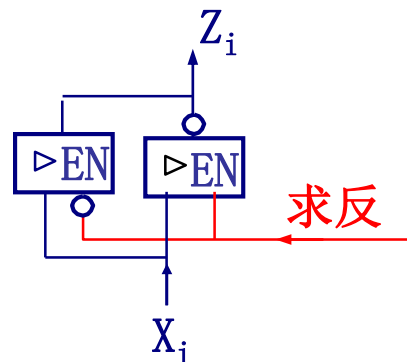
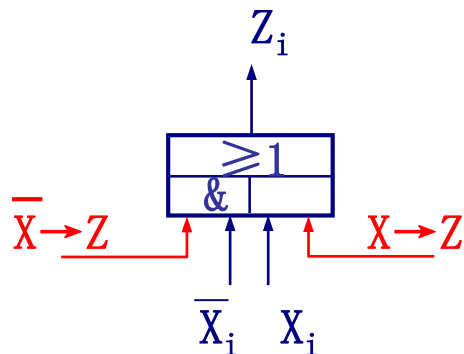
1. 逻辑非（求反）

对各位按位取反。

例：输入： $X=X_0X_1\dots X_n$,

输出： $Z=Z_0Z_1\dots Z_n$,

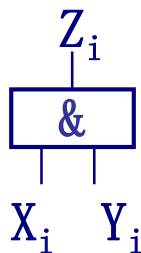
则： $Z_i = \overline{X_i}$ ($i=0, 1, \dots, n$)



2. 逻辑乘（按位与）

$$Z_i = X_i \wedge Y_i \quad (i=0, 1, \dots, n)$$

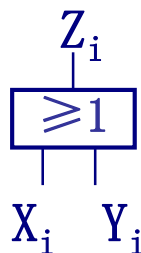
可用与门实现



3. 逻辑加（按位或）

$$Z_i = X_i \vee Y_i \quad (i=0, 1, \dots, n)$$

可用或门实现，也可通过逻辑乘和逻辑非实现。



4. 逻辑异或

$$Z_i = X_i \oplus Y_i \quad (i=0, 1, \dots, n)$$

异或又称半加、不带进位加、模2加。

§ 4.9 运算器的基本组成结构

运算器在控制器的控制下，不仅可以完成数据信息的算逻运算，还可以作为数据信息的传送通路。

§ 4.9 运算器的基本组成结构

4.9.1 运算器结构

1. 运算器的基本结构

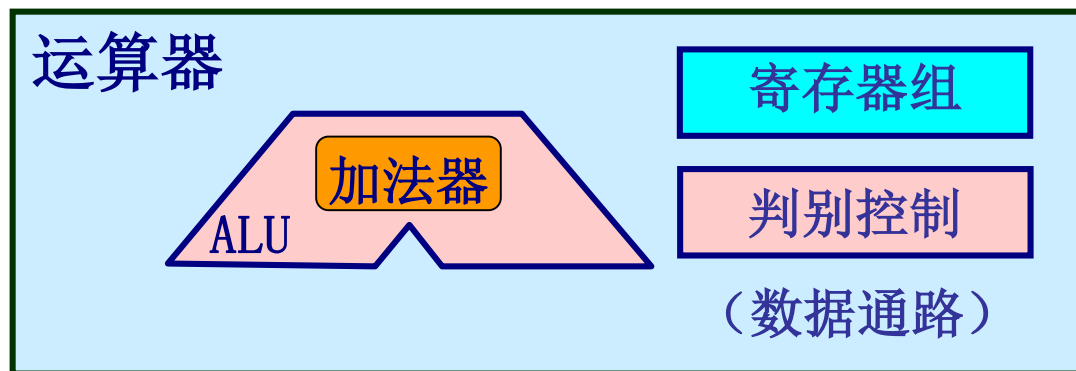
基本的运算器包含---

ALU：实现基本算术、逻辑运算功能(核心部件)

寄存器组：提供操作数与暂存结果

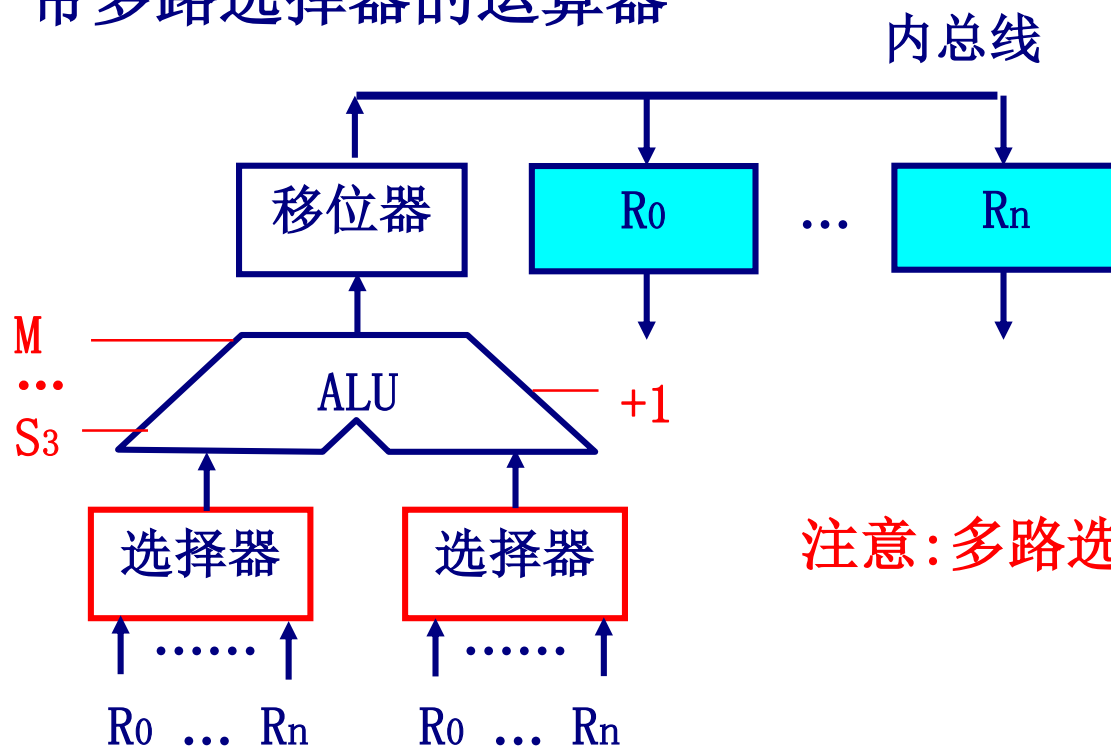
判别逻辑和控制电路

数据传输通路



常见的基本结构有两类：

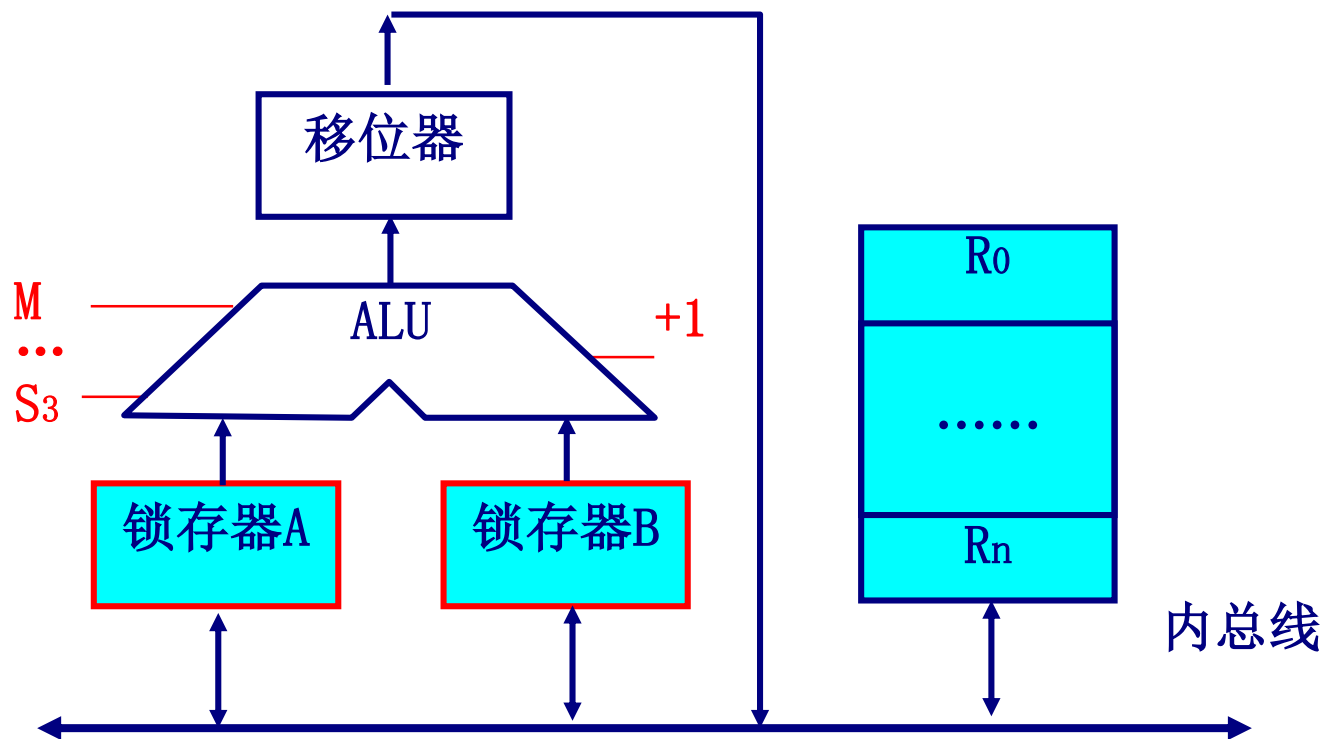
(1) 带多路选择器的运算器



注意：多路选择器无记忆功能！

常见的基本结构有两类：

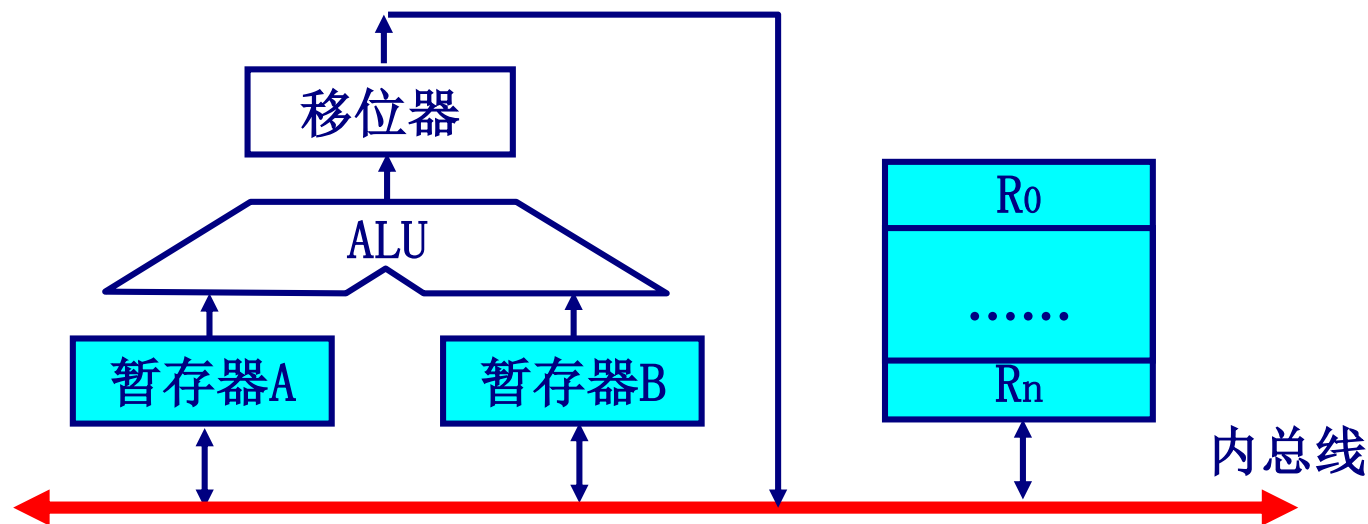
(2) 带输入锁存器的运算器



2. 运算器的内部总线结构

运算器内的各功能模块之间的连接也广泛采用总线结构。

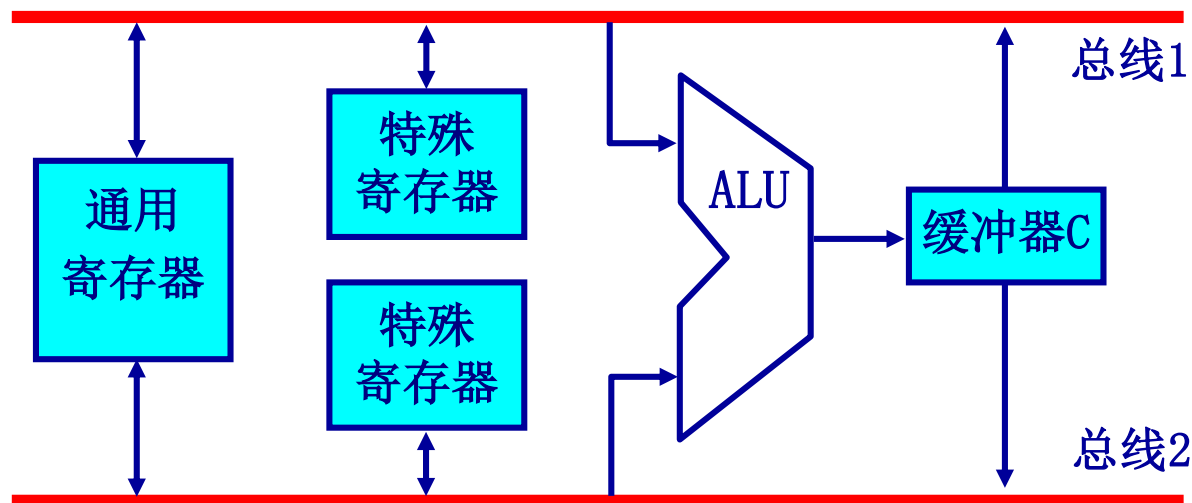
(1) 单总线结构运算器（如图4-25）



例如，实现 $R0+R1 \rightarrow R2$ 运算需三步：

- ① $R0 \rightarrow A$
- ② $R1 \rightarrow B$
- ③ $A+B \rightarrow R2$

(2) 双总线结构运算器

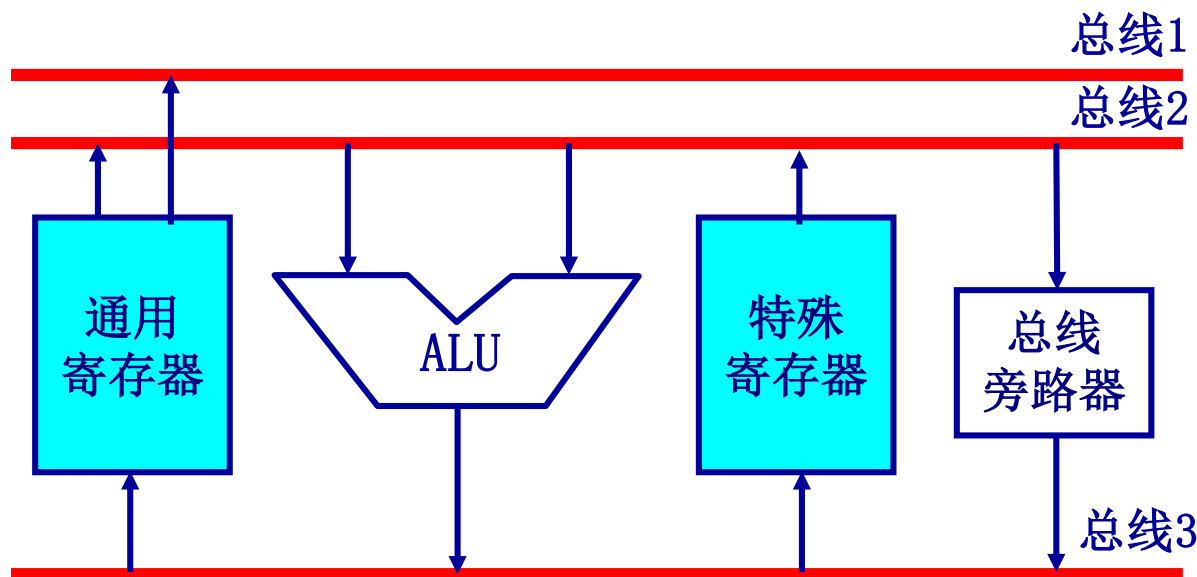


例如，实现 $R0+R1 \rightarrow R2$ 运算需两步：

- ① $R0+R1 \rightarrow C$
- ② $C \rightarrow R2$

通用寄存器组必须是双端口器件，可同时提供两个操作数。

(3) 三总线结构运算器



例如，实现 $R0+R1 \rightarrow R2$ 运算只需一步。

通用寄存器组必须是双端口器件，可同时提供两个操作数。

如果要实现 $R0 \rightarrow R1$ ，可通过总线旁路器把数据送出。

4.9.2 ALU举例

1. ALU电路1

逻辑运算：用与门、或门和异或门实现

算术运算：加法器为核心

$$F_i = X_i \oplus Y_i \oplus C_i$$

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

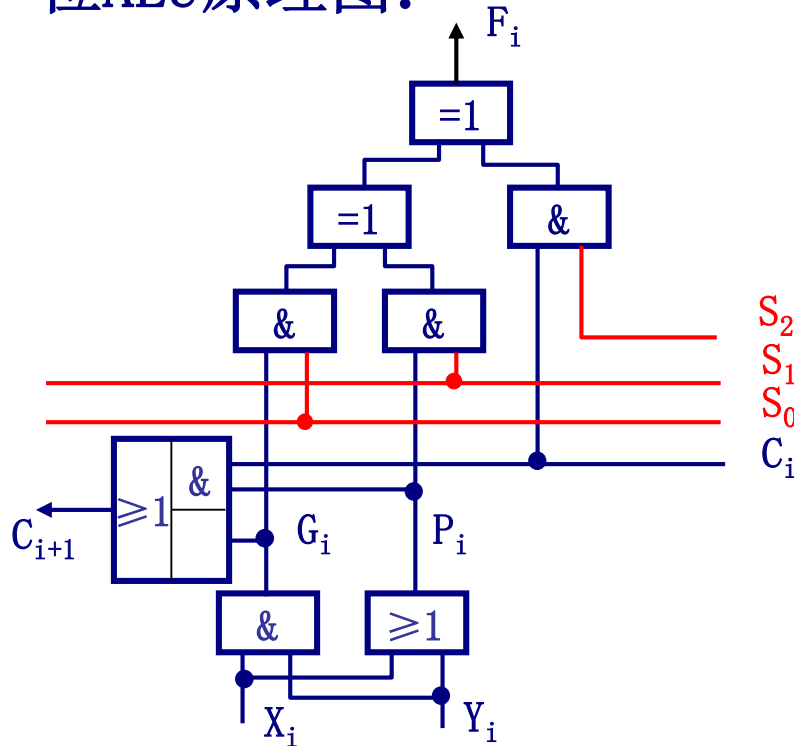
	1011	X
\wedge	1101	Y
	<hr/> 1001	F

	1011	X
+	0001	Y
	<hr/> 1100	F

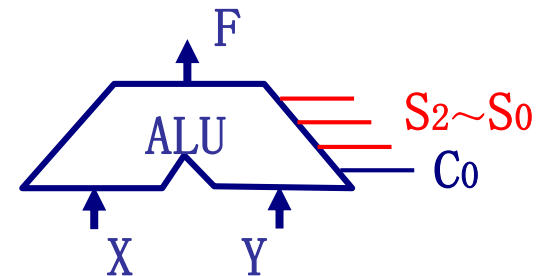
4.9.2 ALU举例

1. ALU电路1

一位ALU原理图：



S_2	S_1	S_0	F_i
0	0	0	0
0	0	0	$X_i \wedge Y_i$
0	0	0	$X_i \vee Y_i$
0	0	0	$X_i \oplus Y_i$
0	0	0	$X_i + Y_i$

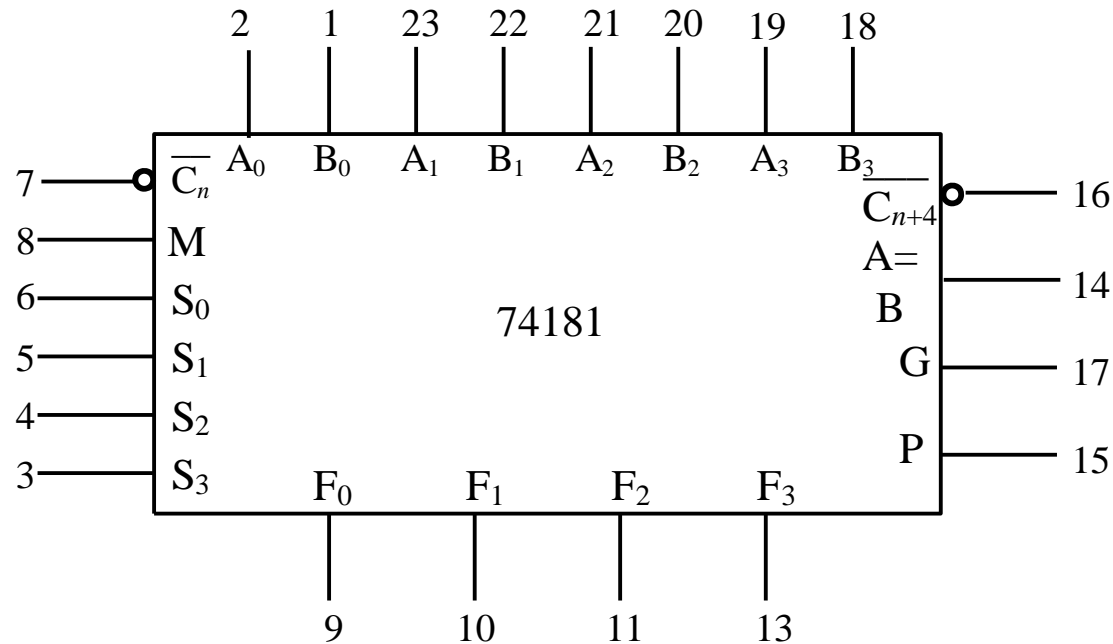


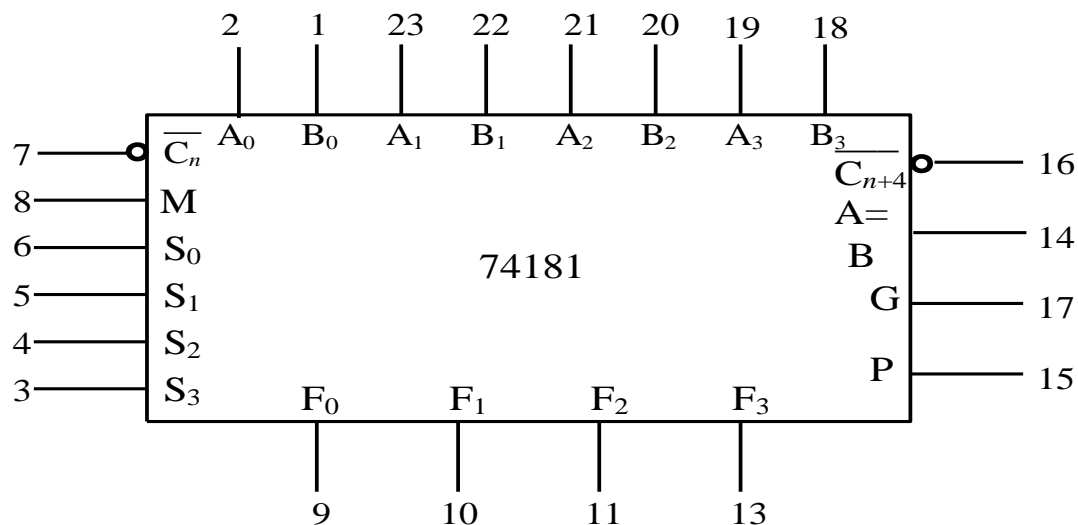
S_2 用于控制位与位之间的进位联系

2. 4位ALU芯片74181

74181能执行16种算术运算（先行进位）和16种逻辑运算。

其运算功能见P132表4-14。





以正逻辑的功能为例：

输入数据A、B为正逻辑，末位进位输入为负逻辑；
输出F为正逻辑。

若要进行“ $F \leftarrow A \text{ 加 } B$ ”运算，则应给其提供的控制信号为：

$$S_3 S_2 S_1 S_0 = 1001, M=0, \overline{C_n}=1$$

可以用74181级连实现多位ALU，或用74181和74182构成多级先行进位的ALU。

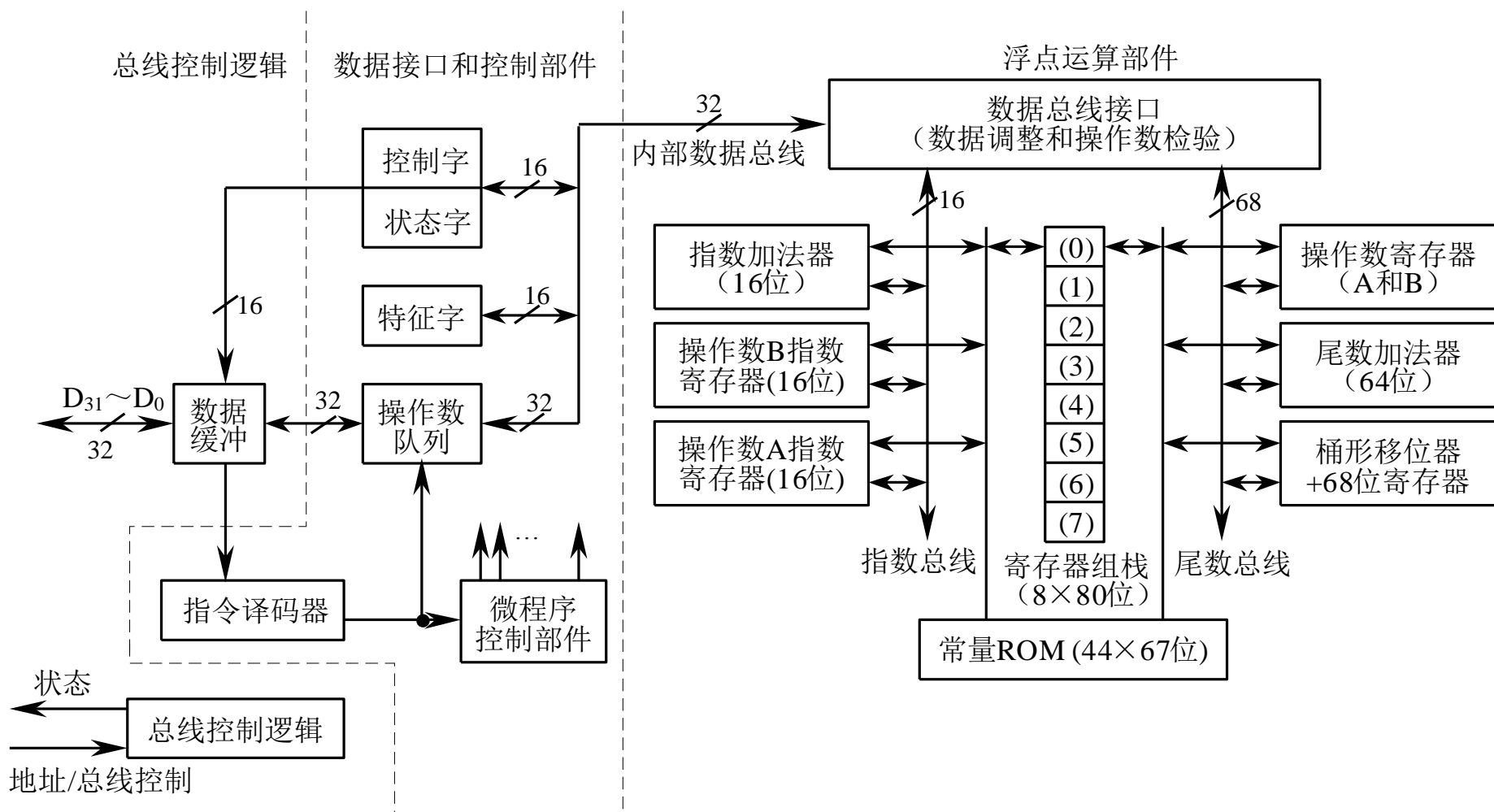
4.9.3 浮点运算器举例（课外阅读）

1. 80X87的数据格式

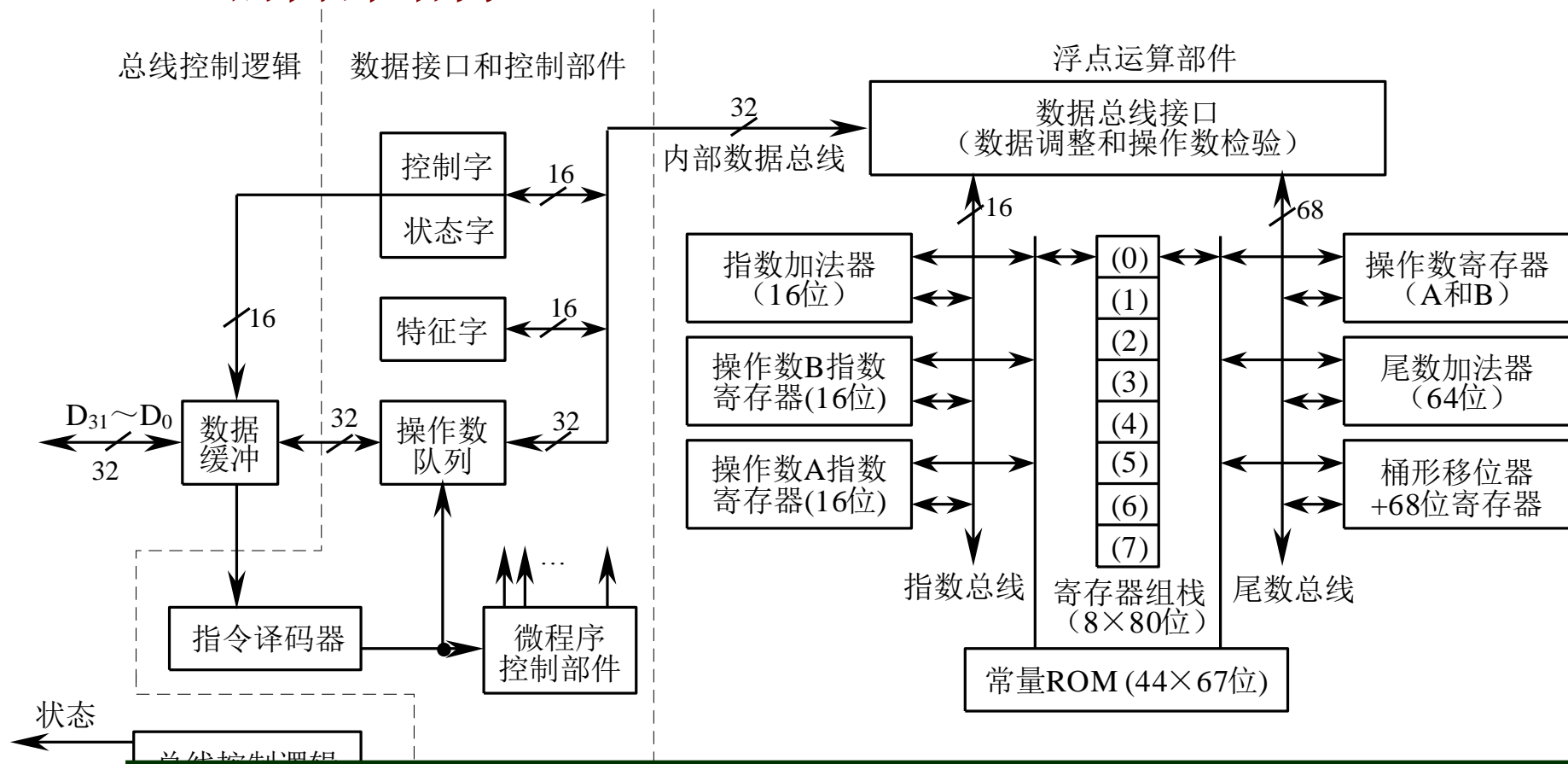
2. 80X87的内部结构

（见教材P134）

80X87的内部结构

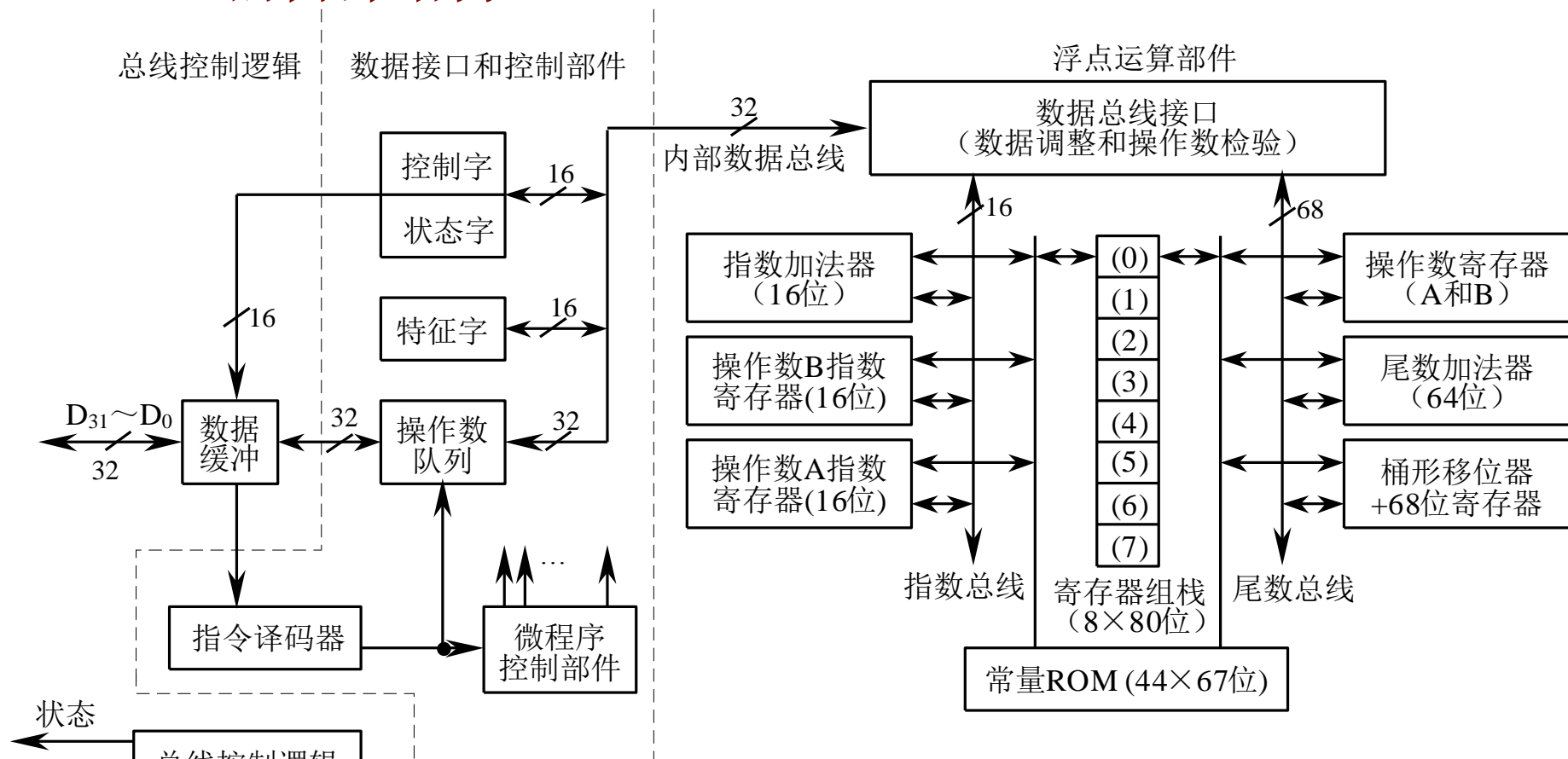


80X87的内部结构



80X87从主存取数或向主存写数时，均用80位的临时浮点数与其他数据类型执行自动转换。在80X87中的全部数据都以80位临时浮点数的形式表示。

80X87的内部结构



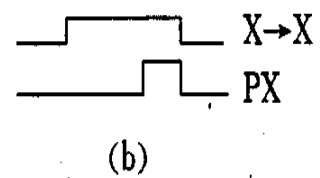
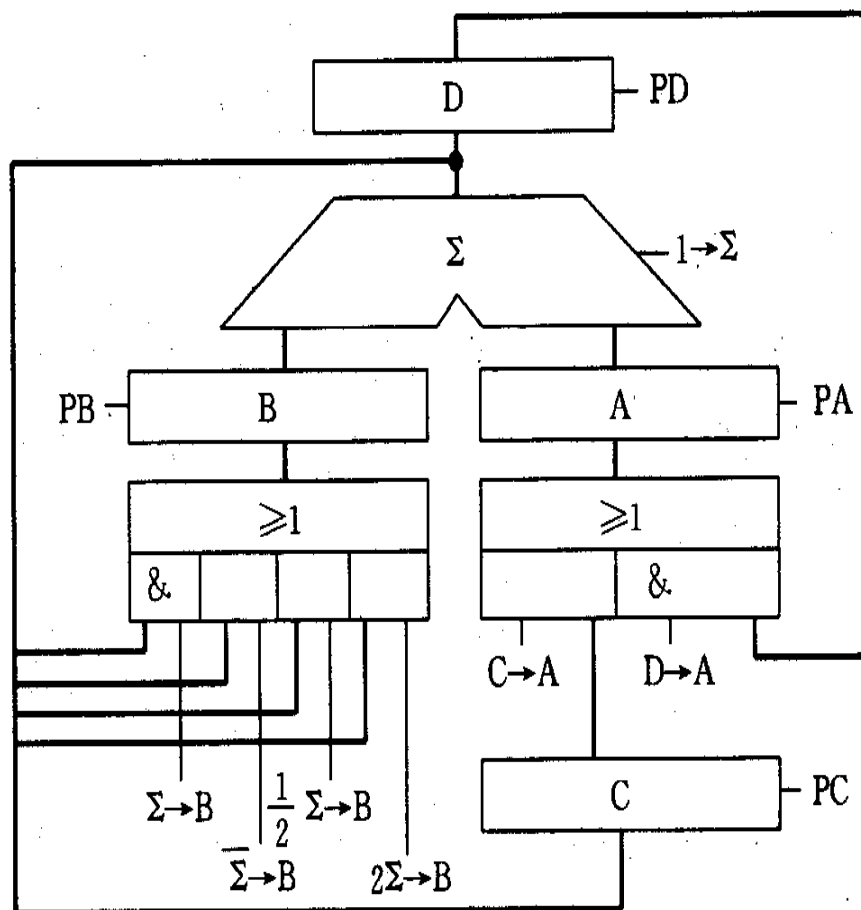
80X87与主微处理器协同工作，微处理器执行所有的常规指令，而80X87只执行专门的算术协处理器指令，称为换码（ESC）指令。微处理器和协处理器可以同时或并行执行各自的指令。

补充4-1.

下图为某模型机数据加工通路。其中 Σ 为并行加法器， $I \rightarrow \Sigma$ 为最低位的进位。**A, B, C, D**为四个寄存器，**PA, PB, PC, PD**分别为四个寄存器的数据接收信号，且均为脉冲信号。图中其余的**X \rightarrow X**控制信号均为电平信号，电平信号与脉冲信号的时间关系如下图所示**(b)**所示。试拟出在该图上实现下列运算所需的微操作。

- (1) $D - C \rightarrow D$
- (2) $D/2 + C \rightarrow D$
- (3) $D - 1 \rightarrow D$
- (4) $2C + I \rightarrow C$

补充4-1.

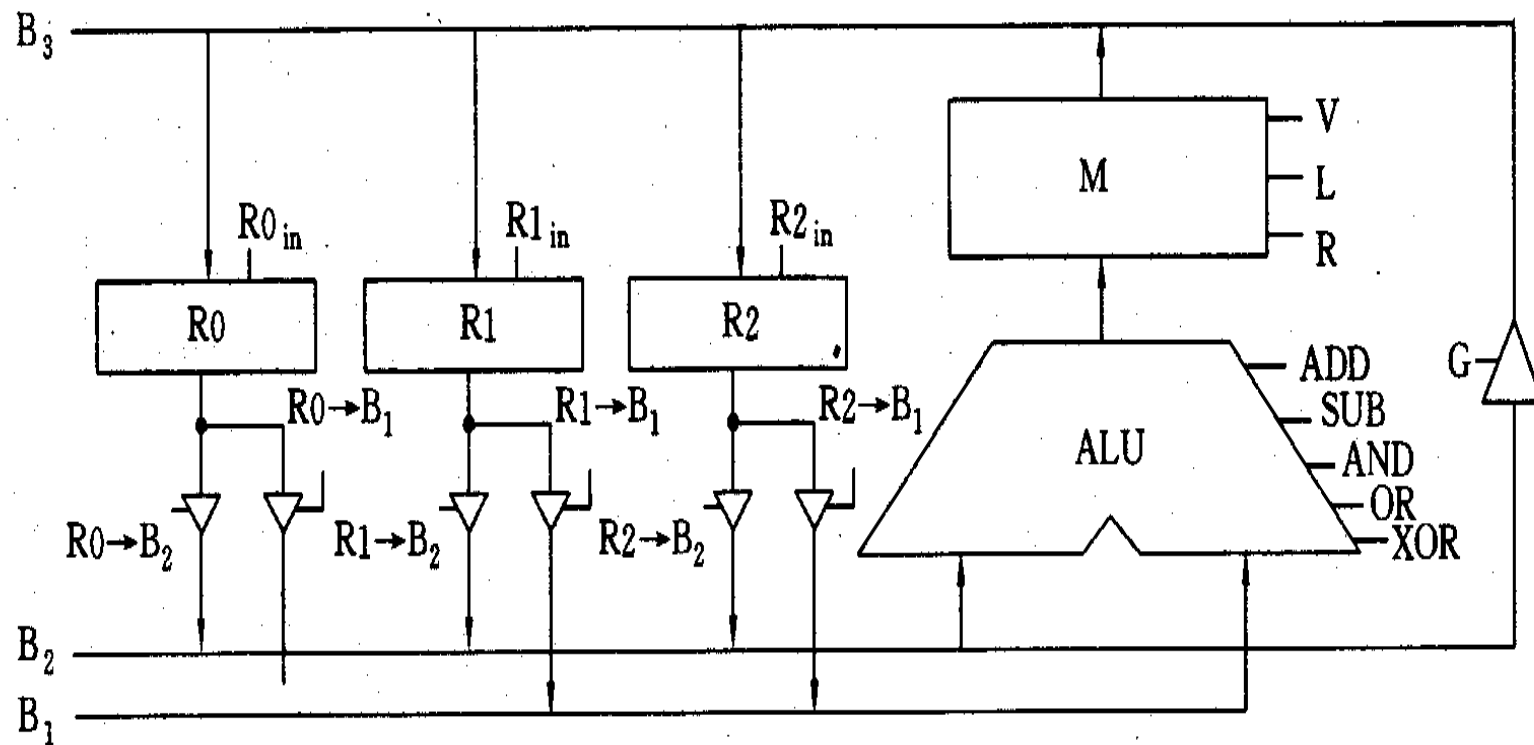


补充4-2.

某机运算器为三总线结构（如下图），三总线分别称为**B1**，**B2**，**B3**，连接**B2**，**B3**的控制信号为**G**。算逻部件**ALU**可进行**ADD**，**SUB**，**AND**，**OR**，**XOR**五种运算，输出多路器可进行直送(**V**)，左移一位(**L**)，右移一位(**R**)三种操作。三个通用寄存器**R0**，**R1**，**R2**都有输入和输出信号。

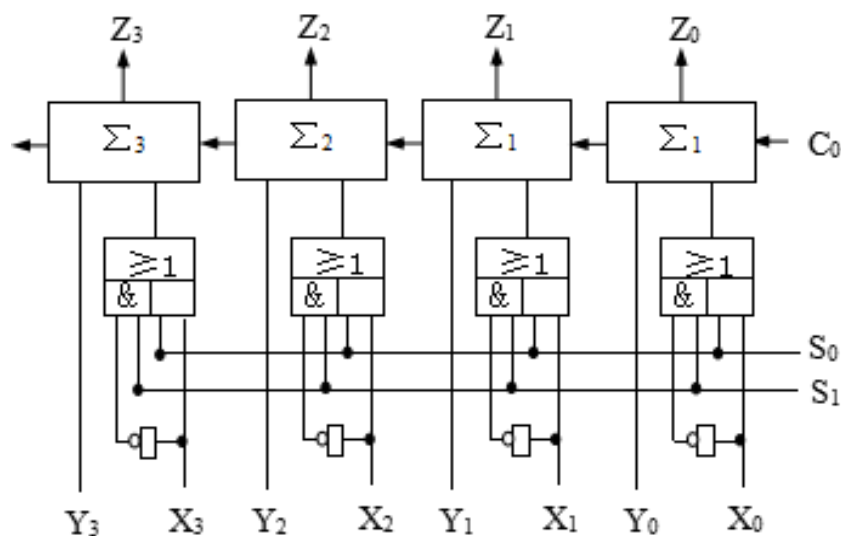
试写出实现下列功能所需的微操作： $1/2[R1-R2] \rightarrow R1$

补充4-2.



例题：

一个简单的运算器如下图所示， $x_3 \sim x_0$ ， $y_3 \sim y_0$ 为输入数据（补码表示的整数）， $z_3 \sim z_0$ 为输出数据， c_0 为最低进位，试分析该运算器在 s_1 ， s_0 ， c_0 在以下不同组合下各完成什么运算？



$S_1 S_0 C_0$	运 算
0 0 0	$Z =$
0 0 1	$Z =$
0 1 0	$Z =$
1 0 1	$Z =$
1 1 0	$Z =$

[解]

$S_1 S_0 C_0$	运 算
0 0 0	$Z=Y_3 Y_2 Y_1 Y_0=Y$
0 0 1	$Z=Y_3 Y_2 Y_1 Y_0+1=Y+1$
0 1 0	$Z=Y_3 Y_2 Y_1 Y_0+X_3 X_2 X_1 X_0=Y+X$
1 0 1	$Z=Y_3 Y_2 Y_1 Y_0-X_3 X_2 X_1 X_0=Y-X$
1 1 0	$Z=Y_3 Y_2 Y_1 Y_0-1=Y-1$

一、单选题

1. 运算器由许多部件组成，但核心部分是____。
A. 数据总线 B. 算术逻辑运算单元
C. 多路开关 D. 累加寄存器
2. 把 n 个全加器串接起来，就可进行两个 n 位数的相加，这种加法器称为____。
A. 串行进位的串行加法器 B. 并行进位的并行加法器
C. 串行进位的并行加法器 D. 并行进位的串行加法器
3. 当采用变形补码（双符号位）运算时，发生负溢的特征是双符号位为____。
A. 00 B. 01 C. 10 D. 11
4. 在定点机中执行算术运算时会产生溢出，其原因是____。
A. 主存容量不够 B. 操作数过大
C. 操作数地址过大 D. 运算结果无法表示
5. 若浮点数的阶码和尾数都用补码表示，判断运算结果是否规格化数的方法是____。
A. 阶符与数符相同 B. 数符与尾数最高有效数位相同
C. 阶符与数符相异 D. 数符与尾数最高有效位相异



一、单选题

6. 下溢指的是_____。
- A. 运算结果的绝对值小于机器所能表示的最小绝对值
 - B. 运算的结果小于机器所能表示的最小负数
 - C. 运算的结果小于机器所能表示的最小正数
 - D. 运算结果的最低有效位产生的错误
7. 若一台计算机的字长为4个字节，则表明该机器_____。
- A. 能处理的数值最大为4位十进制数
 - B. 能处理的数值最多由4位二进制数组成
 - C. 在CPU中能够作为一个整体加以处理的二进制代码为32位
 - D. 在CPU中运算的结果最大为2的32次方

二、填空题

1. 为判断溢出，可采用双符号位补码，此时正数的符号用_____表示，负数的符号用_____表示。
2. 一个基数为2的浮点数，当其补码尾数右移_____bit时，为使其值不变，阶码应该加1。
3. 一个基数为16的浮点数，当其补码尾数右移_____bit时，为使其值不变，阶码应该加1。
4. 行波进位的缺点是_____。
5. 正数补码算术移位时，符号位不变，空位补_____。负数补码算术左移时，符号位不变，低位补_____。负数补码算术右移时，符号位不变，高位补_____，低位_____。

参考答案:

一、单选题: **BACDD AC**

二、填空题

1. 00, 11

2. 1

3. 4

4. 运算速度慢

5. 0, 0, 1, 舍去

本章要点:

1. 理解和掌握与门、与或门、三态门、寄存器等器件的典型应用方法，以及它们的组合应用方法；
2. 加法器的先行进位原理；定点数的加减乘除运算方法(其中补码数运算最重要)，并能把数据表示、运算方法(算法)和电路实现联系起来；
3. 学会用寄存器传送语言描述硬件的微操作；
4. 理解逻辑运算及其实现方法、浮点数的运算方法；
5. 了解定点运算器的典型结构。