西安电子科技大学
XIDIAN UNIVERSITY

第五章 运算方法与运算器

主讲：张骏鹏（博士，副教授）

西安电子科技大学

人工智能学院

运算方法与运算器

- 定点数运算
  - 加减运算
  - 乘法运算
  - 除法运算
- 浮点数运算
  - 浮点加减运算
  - 浮点乘除法运算
- 运算器的基本结构

定点数运算——加减运算

有符号定点数的编码可以用原码、反码、补码、移码等形式表示。原则上讲，有符号数的加减运算可以用任何一种编码来实现，但实际中用得最多、最普遍的是补码。

回顾

补码：

设模为M，一个数 X 补码的一般定义为：

$$[X]_{补} = M + X \ (\text{mode M})$$

- 若 $X > 0$，则模 M 作为超出部分被舍去，$[X]_{补} = X$，因而正数的补码就是其本身；
- 若 $X < 0$，则 $[X]_{补}$ 就是以 M 为模的补数。

回顾

- 定点整数的补码定义：

$$[X]_{补}=\begin{cases}X=[X]_{原} & 0\le X\le 2^{n-1}\text{-}1\\ 2^n+X=2^n-|X| & -2^{n-1}\le X<0\end{cases}\quad \text{MOD } 2^n$$

定点数运算——加减运算

1.补码加减法

补码加法的运算法则为

$$[X+Y]_{补}=[X]_{补}+[Y]_{补}$$

由式(3.1)可以看到，两数和的补码就等于两数补码之和。利用补码求两数之和十分方便。

例3.1 有两个定点整数63和35,利用补码加法求63+35。

例3.1 有两个定点整数63和35,利用补码加法求63+35。

解 根据题意,用8位二进制补码表示63和35为

$$[63]_{补}=00111111$$

$$[35]_{补}=00100011$$

则

$$[63+35]_{补}=01100010$$

例3.2 有两个定点整数-63和-35,利用补码加法求-63+(-35)。

例3.2 有两个定点整数-63和-35,利用补码加法求-63+(-35)。

解 根据题意,用8位二进制补码表示-63和-35为

则

$$[-63]_{补}=11000001$$
$$[-35]_{补}=11011101$$
$$[-63+(-35)]_{补}=10011110$$

定点数运算——加减运算

在数值的补码表示法中，

- 对一个正数求补：对该数包括符号位在内的各位取反再加1，即可得到该数的负数;
- 若对该负数再求补,则又可得到原来的正数。

也就是说$[[X]_{补}]_{求补}=[-X]_{补}$,$[[-X]_{补}]_{求补}=[X]_{补}$。据此可得补码减法的运算法则为

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

例 3.3 有两个定点整数 63 和 35，利用补码减法求 63－35。

例 3.3 有两个定点整数 63 和 35，利用补码减法求 63−35。

解 根据题意，用 8 位二进制补码表示 63 和 35 为

$$[63]_{补}=00111111$$

$$[35]_{补}=00100011$$

而$[63-35]_{补}=[63]_{补}+[-35]_{补}$，同时，$[-35]_{补}=[[35]_{补}]_{求补}=11011101$，从而求出：

$$\begin{array}{r} 00111111 \\ +11011101 \\ \hline 100011100 \end{array}$$

得到$[63-35]_{补}=00011100$。请注意，计算机中要求运算器的原始数据和运算结果应具有相同的数据位数，所以本例中结果仅为 8 位，在相加过程中产生的进位 1 会作为状态信息(即进位标志 CF)保留在状态标志寄存器中。

2.溢出及判断

1)溢出的概念 我们首先通过下面的例子来了解什么是溢出。

例3.4 有两个定点整数63和85,利用补码加法求63+85。

解 用8位二进制补码表示63和85为

$$[63]_{补}=00111111$$

$$[85]_{补}=01010101$$

$$\begin{array}{r} 00111111 \\ +01010101 \\ \hline 10010100 \end{array}$$

我们把运算结果超出规定的数值范围而造成错误的现象称为溢出。若运算结果大于规定 的数值范围的上限,则称为上溢出;若运算结果小于规定的数值范围的下限,则称为下溢出。

一旦确定了运算字长和数据表示方法,数据表示的范围也就随之确定。只要运算结果超出所能表示的数据范围,就会发生溢出。发生溢出时,运算结果一定是错误的,所以必须采取措施防止溢出发生。

例 3.6 设二进制负整数 $X=-1111000$，$Y=-10010$。若用 8 位二进制补码表示，则 $[X]_{补}=10001000$，$[Y]_{补}=11101110$，求$[X+Y]_{补}$。

$$\begin{array}{r} 10001000 \\ +\ 11101110 \\ \hline 01110110 \end{array}$$

2)溢出的判定

(1) **双符号位(变形码)判决法**：采用两位表示符号，

- 即**00表示正号**，**11表示负号**，
- 一旦发生**溢出**，则两个符号位就一定不一致，通过判别两个符号位是否一致便可以判定是否发生了溢出。

若运算结果两符号分别用S_2、S_1表示,则溢出标志 OF 的逻辑表示式为

$$OF = S_2 \oplus S_1$$

当 OF=0时，判别溢出未发生；当 OF=1时,判别溢出发生。

例 3.7 设二进制正整数 $X=+1000001$，$Y=+1000011$。若用双符号 8 位二进制补码表示，则$[X]_{补}=001000001$，$[Y]_{补}=001000011$，求$[X+Y]_{补}$。

解 计算$[X]_{补}+[Y]_{补}$为

$$\begin{array}{r} 001000001 \\ +001000011 \\ \hline 010000100 \end{array}$$

由于结果的两个符号位 $S_2=0$ 和 $S_1=1$ 不一致，使 $OF=S_2 \oplus S_1=1$，因此发生溢出，运算结果不正确。

(2) **进位判决法**。若C_{n-1}表示最高数值位产生的进位，C_n表示符号位产生的进位(即进位标志 CF)，则溢出标志 OF的逻辑表示式为

$$OF = C_n \oplus C_{n-1}$$

在例 3.7 的运算中，$C_{n-1}=1$，$C_n=0$，故 $C_n \oplus C_{n-1}=1$，同样判定运算结果有溢出。

(3)根据运算结果的符号位和进位标志判别。该方法适用于两个同号数求和或异号数 求差时判别溢出。溢出标志OF的逻辑表达式为

$$OF = SF \oplus CF \tag{3.5}$$

其中,SF和 CF分别是运算结果的符号标志和进位标志。

(4)根据运算前后的符号位进行判别。若用 X_s、Y_s、Z_s 分别表示两个原始数据及运算 结果的符号位,则溢出标志 OF的逻辑表达式为

$$OF = X_s \cdot Y_s \cdot \overline{Z}_s + \overline{X}_s \cdot \overline{Y}_s \cdot Z_s \qquad (3.6)$$

该式表示,两个正数相加的结果为负数,溢出发生;或者两个负数相加的结果为正数,同样 溢出发生。

3.一位全加器的实现

设一位全加器的输入分别为 X_i和Y_i,低一位对该位的进位为C_i,全加器的结果(和) 为Z_i,向高一位的进位为C_{i+1},则实现一位全加器的逻辑表达式为

$$Z_i = X_i \oplus Y_i \oplus C_i \tag{3.7}$$

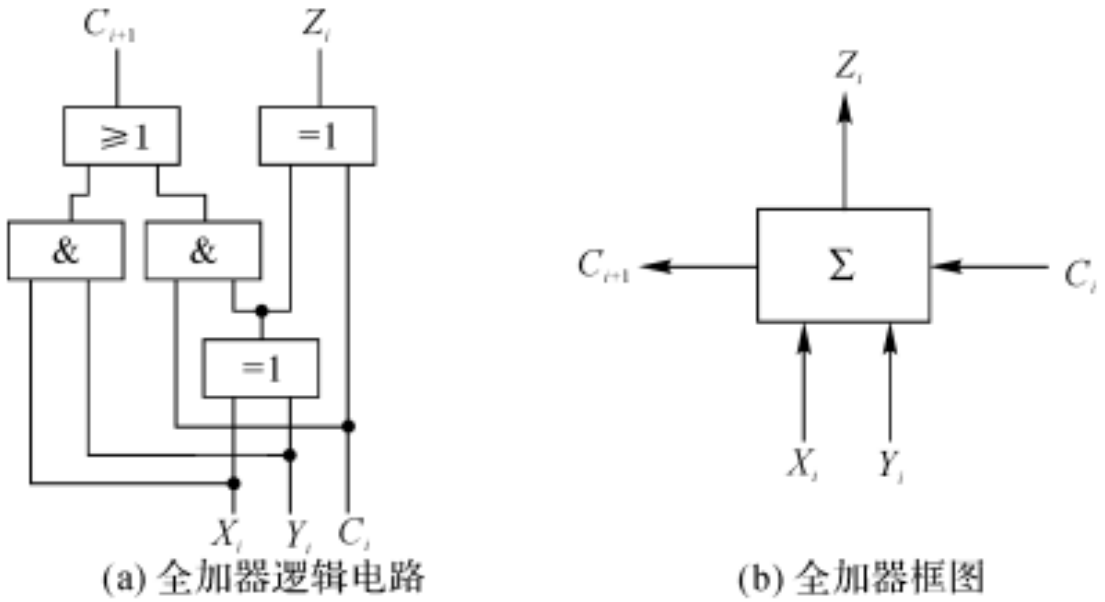
$$C_{i+1} = (X_i \cdot Y_i) + (X_i + Y_i) \cdot C_i = (X_i \cdot Y_i) + (X_i \oplus Y_i) \cdot C_i \tag{3.8}$$

若令 $G_i = X_i \cdot Y_i$，$P_i = X_i + Y_i$，则式(3.8)可写为

$$C_{i+1} = G_i + P_i \cdot C_i \tag{3.9}$$

其中,G_i为进位产生函数,P_i为进位传递函数。

图3.1(a)和(b)为实现上述逻辑的一位全加器逻辑电路及框图表示。



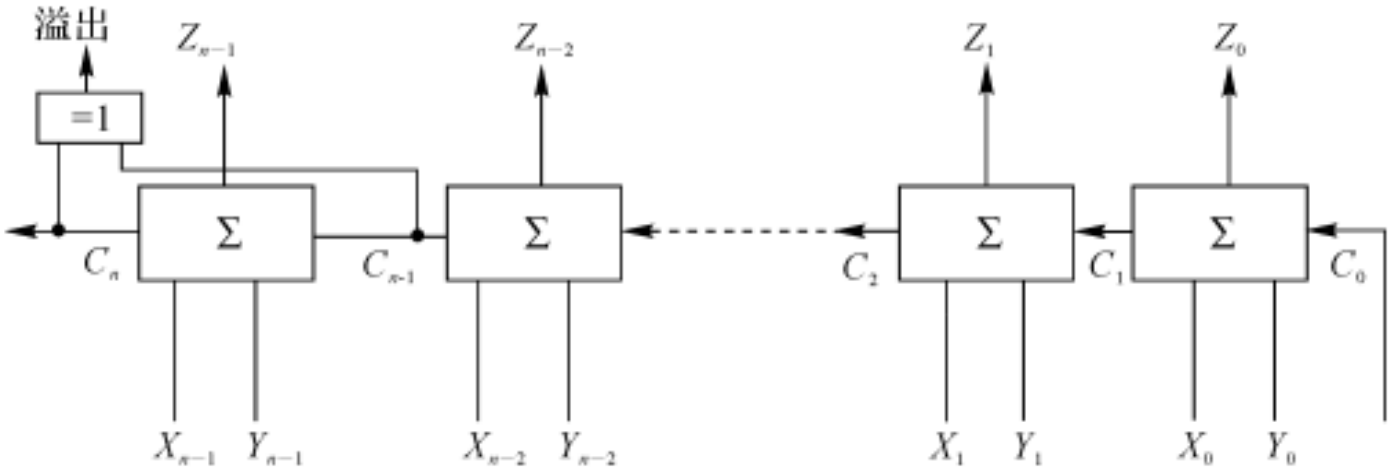
(a) 全加器逻辑电路　　(b) 全加器框图

图3.1 一位全加器逻辑电路及其框图

4.n 位加减器的实现

1)加法器

(1)行波进位加法器。将n 个一位全加器串接在一起,便可以构成n 位二进制数加法器。



从图3.2中可以看出:

① 加法器的进位逐位产生。

② 加法器的和逐位生成。

③ 图3.2中利用异或门实现了式(3.4)的溢出判别逻辑,该异或门的输入是C_{n-1}和C_n。

(2)并(先)行进位加法器(CLA)。行波进位加法器结构简单,实现成本低。但其致命的缺点在于,随着加法器位数的增加,串行生成的进位会造成加法速度大为降低。一种有效的改进方法是同时生成所有低位向高位的进位。

$$Z_i = X_i \oplus Y_i \oplus C_i$$

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

若令 $G_i = X_i \cdot Y_i$，$P_i = X_i + Y_i$，则式(3.8)可写为

$$C_{i+1} = G_i + P_i \cdot C_i$$

其中,四个进位生成逻辑表示式为

$$C_{i+1}=G_i+P_iC_i \tag{3.10}$$

$$C_{i+2}=G_{i+1}+P_{i+1}C_{i+1}=G_{i+1}+P_{i+1}G_i+P_{i+1}P_iC_i \tag{3.11}$$

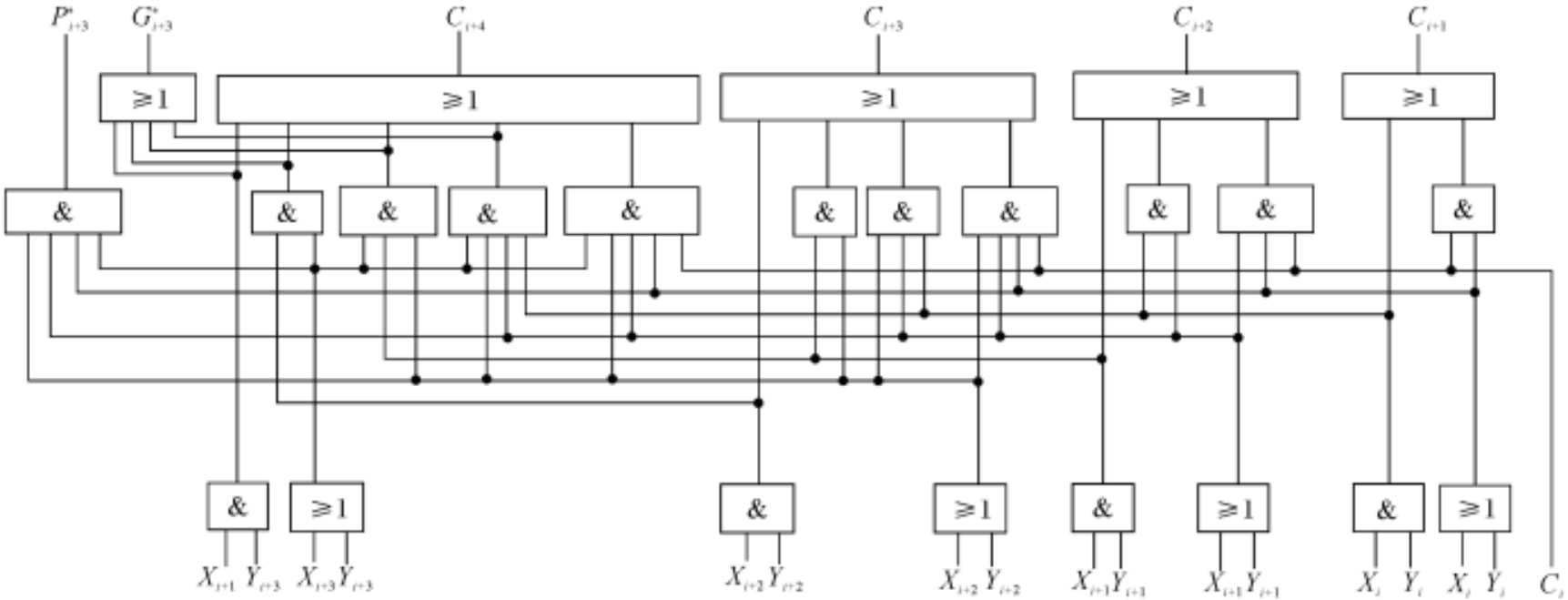
$$C_{i+3}=G_{i+2}+P_{i+2}C_{i+2}=G_{i+2}+P_{i+2}G_{i+1}+P_{i+2}P_{i+1}G_i+P_{i+2}P_{i+1}P_iC_i \tag{3.12}$$

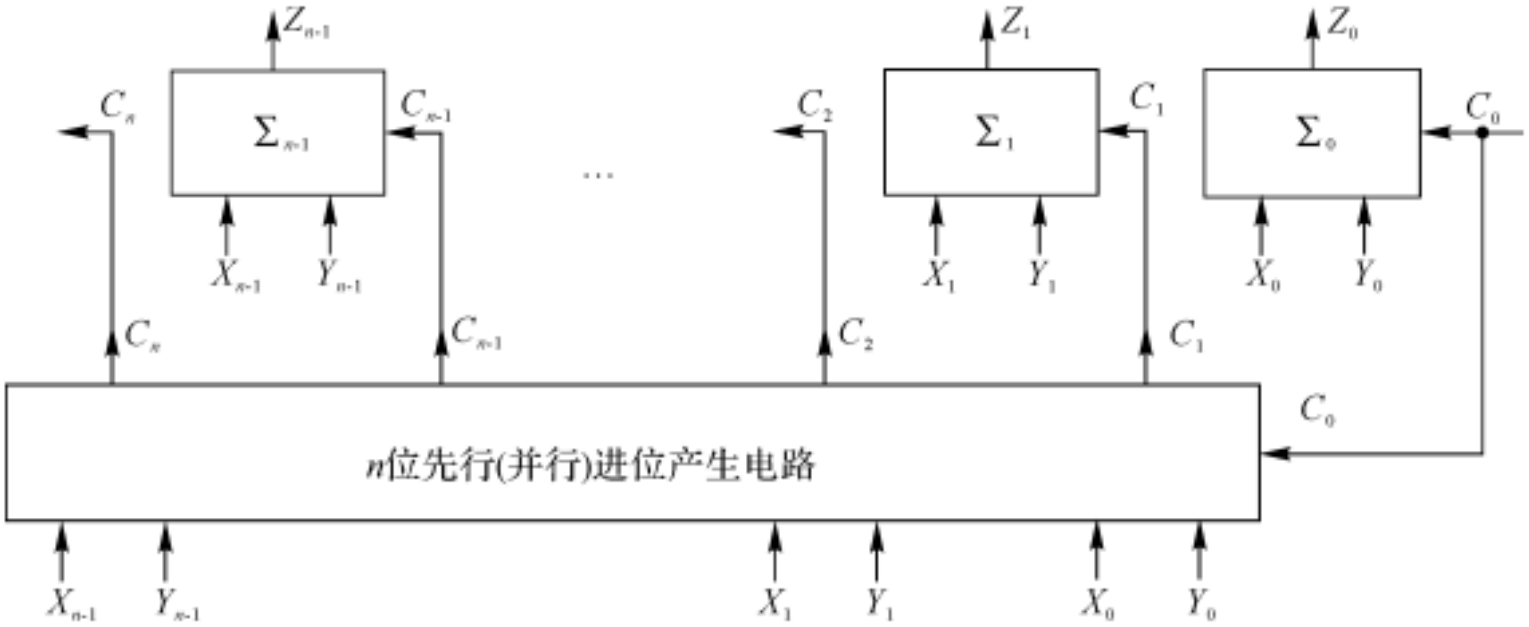
$$\begin{aligned}C_{i+4}&=G_{i+3}+P_{i+3}C_{i+3}\\&=G_{i+3}+P_{i+3}G_{i+2}+P_{i+3}P_{i+2}G_{i+1}+P_{i+3}P_{i+2}P_{i+1}G_i+P_{i+3}P_{i+2}P_{i+1}P_iC_i\\&=G^*_{i+3}+P^*_{i+3}C_i\end{aligned} \tag{3.13}$$

其中：

$$G^*_{i+3}=G_{i+3}+P_{i+3}G_{i+2}+P_{i+3}P_{i+2}G_{i+1}+P_{i+3}P_{i+2}P_{i+1}G_i$$

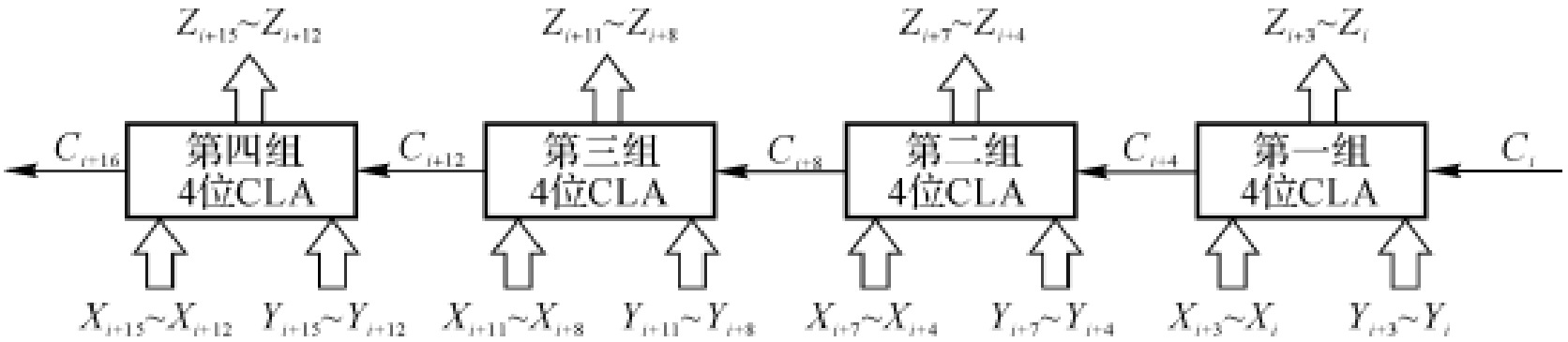
$$P^*_{i+3}=P_{i+3}P_{i+2}P_{i+1}P_i$$


P^*_{i+3}
G^*_{i+3}
C_{i+4}
C_{i+3}
C_{i+2}
C_{i+1}
≥1
&
X_{i+1} Y_{i+3}
X_{i+3} Y_{i+3}
X_{i+2} Y_{i+2}
X_{i+2} Y_{i+2}
X_{i+1} Y_{i+1}
X_{i+1} Y_{i+1}
X_i Y_i
X_i Y_i
C_i


Z_{n-1}
Z_1
Z_0
C_n
Σ_{n-1}
C_{n-1}
C_2
Σ_1
C_1
Σ_0
C_0
X_{n-1}
Y_{n-1}
X_1
Y_1
X_0
Y_0
C_n
C_{n-1}
C_2
C_1
C_0
n位先行(并行)进位产生电路
X_{n-1}
Y_{n-1}
X_1
Y_1
X_0
Y_0

(3)组内并行组间串行进位加法器。组内并行组间串行进位又称为单级先行进位。组 间进位是串行的,即每个组的进位输入是相邻低位组的进位输出,而每个组的进位输出是相邻高位组的进位输入。串行进位链的总延迟时间与分组数目成正比。

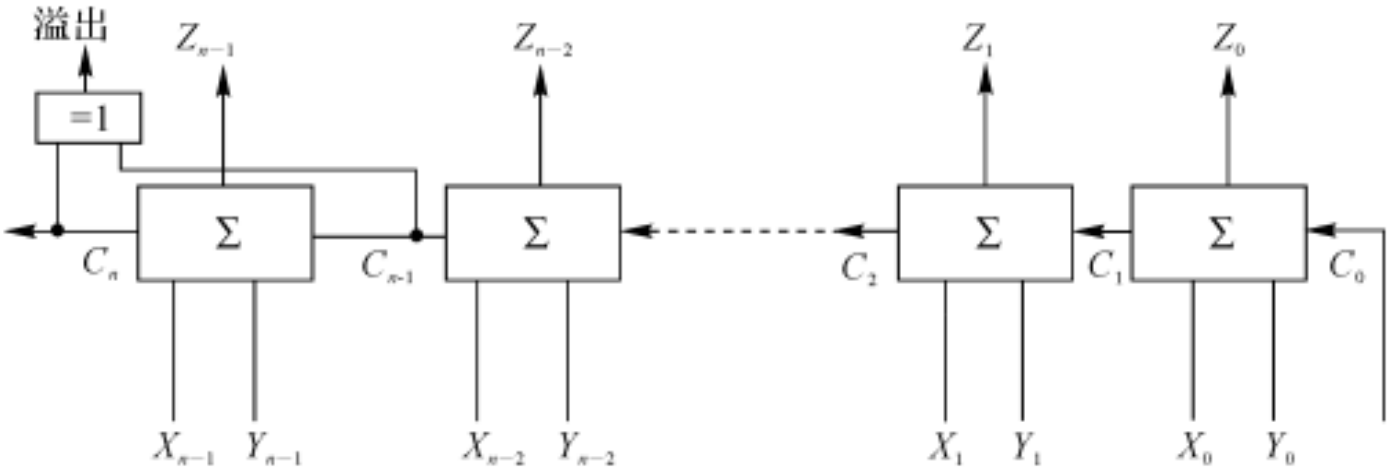
以16位加法器为例,将其分为 4 组,每组 4 位。各组内采用 4 位并行进位加法器 (CLA),组间采用串行进位方式,这样就构成了组内并行组间串行进位加法器,如图3.5所 示。若4位 CLA 的延时为 $\Delta t'(=\Delta t+\tau)$,则该16位并行加法器的计算时间就是$4\Delta t'$。若 n 位加法器分为m 组,则加法器的计算时间就是$m\Delta t'$。

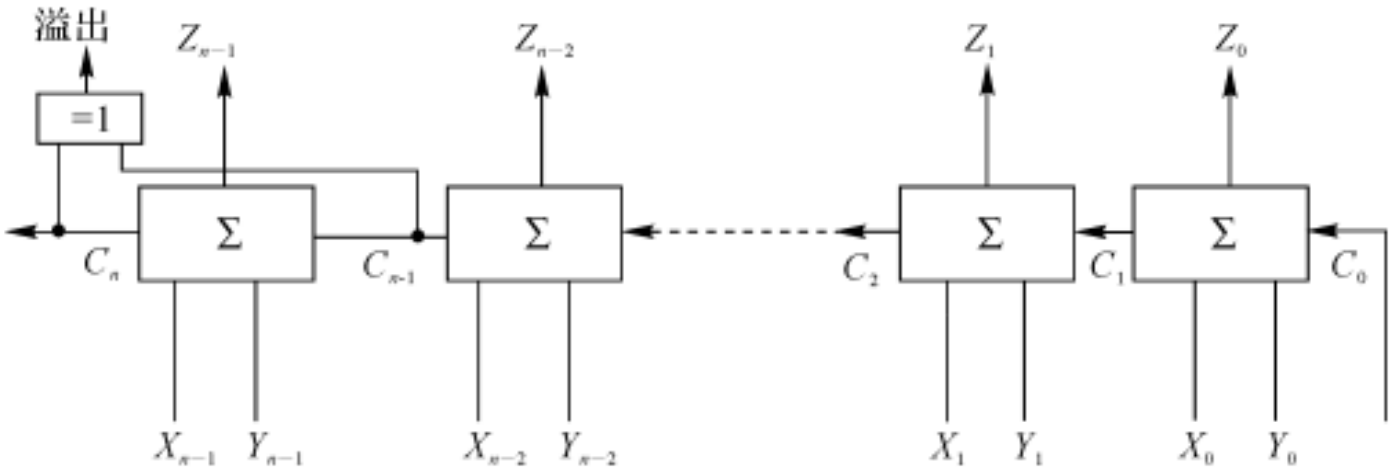

Z_{i+15}~Z_{i+12}
Z_{i+11}~Z_{i+8}
Z_{i+7}~Z_{i+4}
Z_{i+3}~Z_i
C_{i+16}
第四组
4位CLA
C_{i+12}
第三组
4位CLA
C_{i+8}
第二组
4位CLA
C_{i+4}
第一组
4位CLA
C_i
X_{i+15}~X_{i+12}
Y_{i+15}~Y_{i+12}
X_{i+11}~X_{i+8}
Y_{i+11}~Y_{i+8}
X_{i+7}~X_{i+4}
Y_{i+7}~Y_{i+4}
X_{i+3}~X_i
Y_{i+3}~Y_i

2)加法/减法器

$$[X+Y]_{补}=[X]_{补}+[Y]_{补}$$

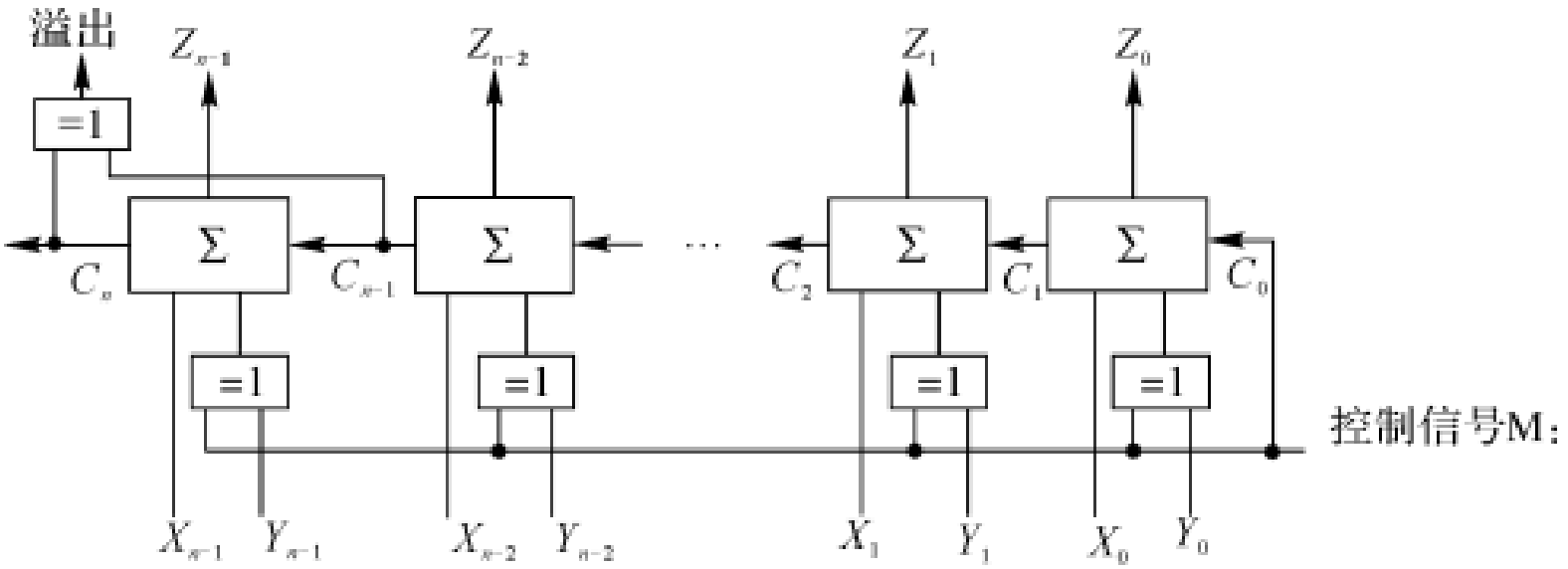
$$[X-Y]_{补}=[X]_{补}+[-Y]_{补}=[X]_{补}+[[Y]_{补}]_{求补}$$



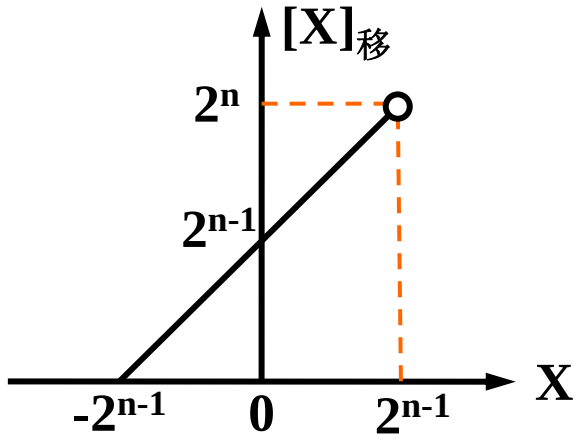

溢出
=1
Z_{n-1}
Z_{n-2}
Z_1
Z_0
Σ
Σ
Σ
Σ
C_n
C_{n-1}
C_2
C_1
C_0
X_{n-1}
Y_{n-1}
X_{n-2}
Y_{n-2}
X_1
Y_1
X_0
Y_0

在图3.7中,利用异或门和控制信号 M 可实现减法运算。

- 当 M =0时，异或门输出Y，实现加法 X+Y 的功能；
- 当 M =1时，异或门输出Y，Y与最低进位C_0=M =1相加，实现减数求补(求补)，然后与 X 做加法，从而实现减法 X-Y 的功能。



6.移码加减法



1)运算规则 由于移码多用在浮点数的阶码中，因此这里仅就定点整数移码的加减运算加以说明。 定点整数移码的加减运算规则如下:

(1) 两运算数据应为移码编码。

(2) 对两移码求和/差。

(3) 对结果进行修正———将结果的符号取反，即得到正确结果。

根据该规则，将前述的n 位加/减法运算器结果输出端的最高位(即符号位)加一个反相器，即可构成移码加/减法运算器。

例 3.9 用 8 位移码表示十进制数 57 和−35，并且用移码运算求两数和与差的移码。

解 十进制数 57 和−35 的移码编码为

$$[57]_{移}=10111001$$

$$[-35]_{移}=01011101$$

两者之和：

例 3.9 用 8 位移码表示十进制数 57 和−35，并且用移码运算求两数和与差的移码。

解 十进制数 57 和−35 的移码编码为

$$[57]_{移}=10111001$$

$$[-35]_{移}=01011101$$

两者之和：

$$[57]_{移}+[-35]_{移}=10111001+01011101=00010110$$

将结果符号位取反，得到

$$[57+(-35)]_{移}=10010110$$

两者之差：

$$[57]_{移}-[-35]_{移}=[57]_{移}+[[-35]_{移}]_{求补}$$

$$=10111001+10100011=01011100$$

将结果符号位取反，得到

$$[57-(-35)]_{移}=11011100$$

THE END !

THANKS