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

1. 定点数运算
2. 浮点数运算
3. 运算器的基本结构

5.1 定点数运算

5.1.2 乘法运算

在一些简单的计算机中，乘法运算可以用软件来实现。利用计算机中设置的加法、移位等指令，编写一段程序完成两数相乘。若 CPU 硬件结构简单，则这种做法实现乘法所用的时间较长，速度很慢。

另一种情况是在 ALU 等硬件的基础上，适当增加一些硬件构成乘法器。这种乘法器的硬件要复杂一些，但速度比较快。速度最快的是全部由硬件实现的阵列乘法器，其硬件更加复杂。可见，可以用硬件来换取速度。

1.原码乘法运算

1)原码一位乘法规则 假定被乘数 X、乘数Y 和乘积Z 为用原码表示的纯小数(下面的讨论同样适用于纯整数),分别为

$$[X]_{原}=x_0.x_{-1}\ x_{-2}\cdots x_{-(n-1)}$$

$$[Y]_{原}=y_0.y_{-1}\ y_{-2}\cdots y_{-(n-1)}$$

$$[Z]_{原}=z_0.z_{-1}\ z_{-2}\cdots z_{-(2n-1)}$$

其中，x_0、y_0、z_0是它们的符号位。特别提醒：小数点在编码中是隐含的，且小数点的位置是默认的，此处显示小数点仅为提示作用。本章其他处若小数编码中出现小数点，其作用也是如此。

原码一位乘法规则如下:

(1)乘积的符号为被乘数的符号位与乘数的符号位相异或。

(2)乘积的数值为被乘数的数值与乘数的数值之积,即

$$|Z|=|X\times Y|=|X|\times|Y| \tag{3.18}$$

(3)乘积的原码为

$$[Z]_{原}=[X\times Y]_{原}=(x_0\oplus y_0)(|X|\times|Y|) \tag{3.19}$$

2）原码一位乘法的实现思路

下面通过示例介绍手算乘法的计算过程。

例 3.10 若$[X]_{原}=0.1101$，$[Y]_{原}=1.1011$，求两者之积。

解 乘积的符号为 $z_0=0\oplus 1=1$。

数值之积的手算过程如下：

$$
\begin{array}{r}
1101 \\
\times\quad 1011 \\
\hline
1101 \\
1101 \\
0000 \\
1101 \\
\hline
.10001111
\end{array}
$$

将该计算结果小数点左边加上乘积的符号 1，即可获得乘积的原码为

$$[Z]_{原}=[X\times Y]_{原}=1.10001111$$

由本例可得数值计算的方法，即被乘数 X 与乘数的某一位 $2^i y_i$ 相乘得到部分和 $2^i y_i X$，所有部分和加在一起得到乘积的数值。

例3.11 $[X]_{原}$ =0.1101,$[Y]_{原}$ =1.1011,求两者之积。

解 (1)乘积的符号为z_0=0⊕1=1。

(2)利用原码一位乘法求两乘数的数值之积,其过程见图3.11。

(3)将乘积的符号与数值之积拼接在一起,得到最终的乘积,见图3.11下部。

	D				A				A_0	操作
0	0	0	0	0	1	0	1	1		A_0=1,+X
+0	1	1	0	1						
0	1	1	0	1						
0	0	1	1	0	1	1	0	1		右移一位
+0	1	1	0	1						A_0=1,+X
1	0	0	1	1						
0	1	0	0	1	1	1	1	0		右移一位
0	0	0	0	0						A_0=0,+0
0	1	0	0	1						
0	0	1	0	0	1	1	1	1		右移一位
+0	1	1	0	1						A_0=1,+X
1	0	0	0	1						
0	1	0	0	0	1	1	1	1		→右移一位

拼接符号后积为：$[XY]_{原}$=1.10001111。

图3.11 例3.11的乘法过程

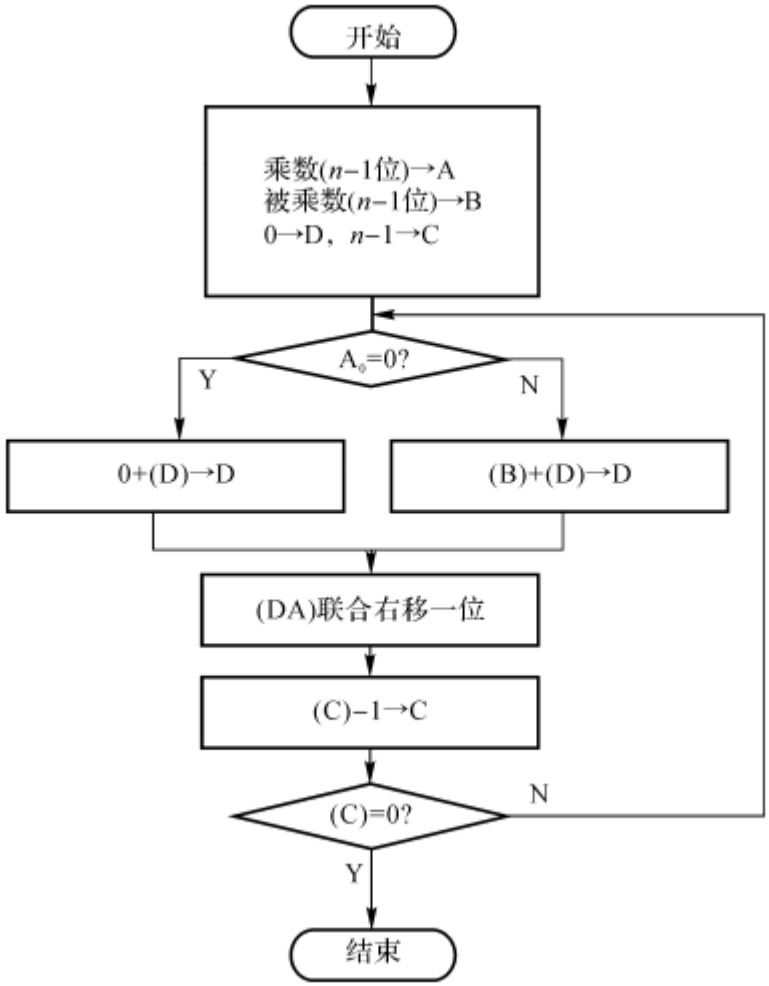


图3.10 数值乘法算法流程

4)原码一位乘法器的框图

根据以上对原码一位乘法的描述,可以设计出采用原码一位乘法的乘法器,如图3.12 所示。

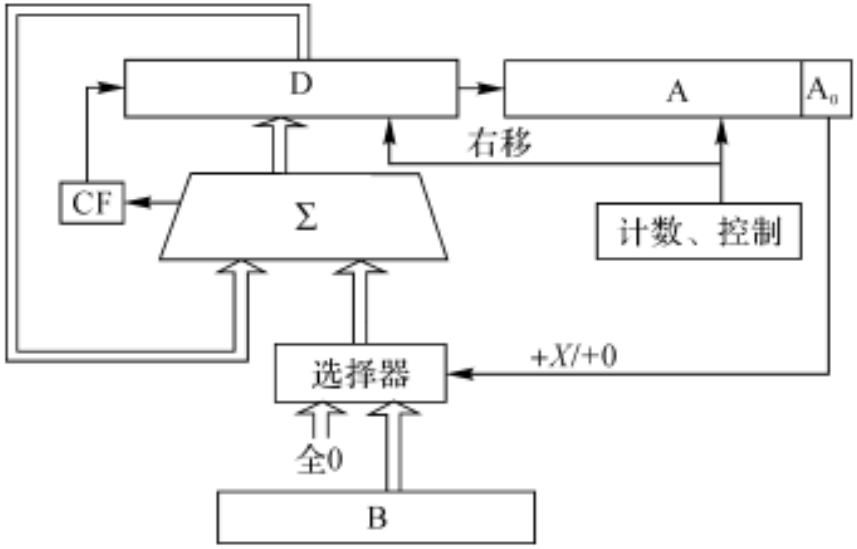


图3.12 原码一位乘法器的框图

2.补码乘法运算

计算机中经常采用补码表示数据，这时用原码进行乘法运算很不方便，因此，较多计算机采取补码进行乘法运算。一种经典的补码乘法算法为布斯法，它是补码一位乘法中的一种，是由布斯(Booth)夫妇提出的。

$$[X]_{补}=x_0.\ x_{-1}x_{-2}\cdots x_{-(n-1)}$$
$$[Y]_{补}=y_0.\ y_{-1}y_{-2}\cdots y_{-(n-1)}$$

$$[X\times Y]_{补}=[X]_{补}\times[(y_{-1}-y_0)\times 2^{0}+(y_{-2}-y_{-1})\times 2^{-1}+\cdots+(y_{-(n-1)}-y_{-(n-2)})\times 2^{-(n-2)}+(0-y_{-(n-1)})\times 2^{-(n-1)}] \quad (3.20)$$

表 3.1　以乘数相邻两位为依据的布斯算法运算过程中的操作

y_i	y_{i-1}	$y_{i-1}-y_i$	操　　作
0	0	0	+0，右移一位
0	1	1	$+[X]_{补}$，右移一位
1	0	−1	$+[-X]_{补}$，右移一位
1	1	0	+0，右移一位

根据以上分析,可将布斯算法描述如下:

(1)乘数与被乘数均用补码表示,连同符号位一起参加运算。

(2)乘数最低位后增加一个附加位(用 A-1表示),设定初始值为0。

(3)从附加位开始,依据表3.1所示的操作完成式(3.20)的运算。 实现布斯算法的流程如图3.13所示。

例3.12 已知二进制数 X=0.1010,Y=-0.1101。利用布斯算法求[XY]补 。

解 (1)将两数用补码表示为

$$[X]_{补}=00.1010,\ [Y]_{补}=11.0011,\ [-X]_{补}=11.0110$$

(2)图3.14给出了布斯算法求解过程。由图3.14可知,$[X\cdot Y]_{补}$ =1.011111100。

符号	D	A	A_{-1}	操作
0 0	0 0 0 0	1 0 0 1	1	0	
1 1	0 1 1 0				$+[-X]_{补}$
1 1	0 1 1 0				
1 1	1 0 1 1	0 1 0 0	1	1	右移一位
0 0	0 0 0 0				+0
1 1	1 0 1 1				
1 1	1 1 0 1	1 0 1 0	0	1	右移一位
0 0	1 0 1 0				$+[X]_{补}$
0 0	0 1 1 1				
0 0	0 0 1 1	1 1 0 1	0	0	右移一位
0 0	0 0 0 0				+0
0 0	0 0 1 1				
0 0	0 0 0 1	1 1 1 0	1	0	右移一位
1 1	0 1 1 0				$+[-X]_{补}$
1 1	0 1 1 1				
1 1	1 0 1 1	1 1 1 1	0		右移一位

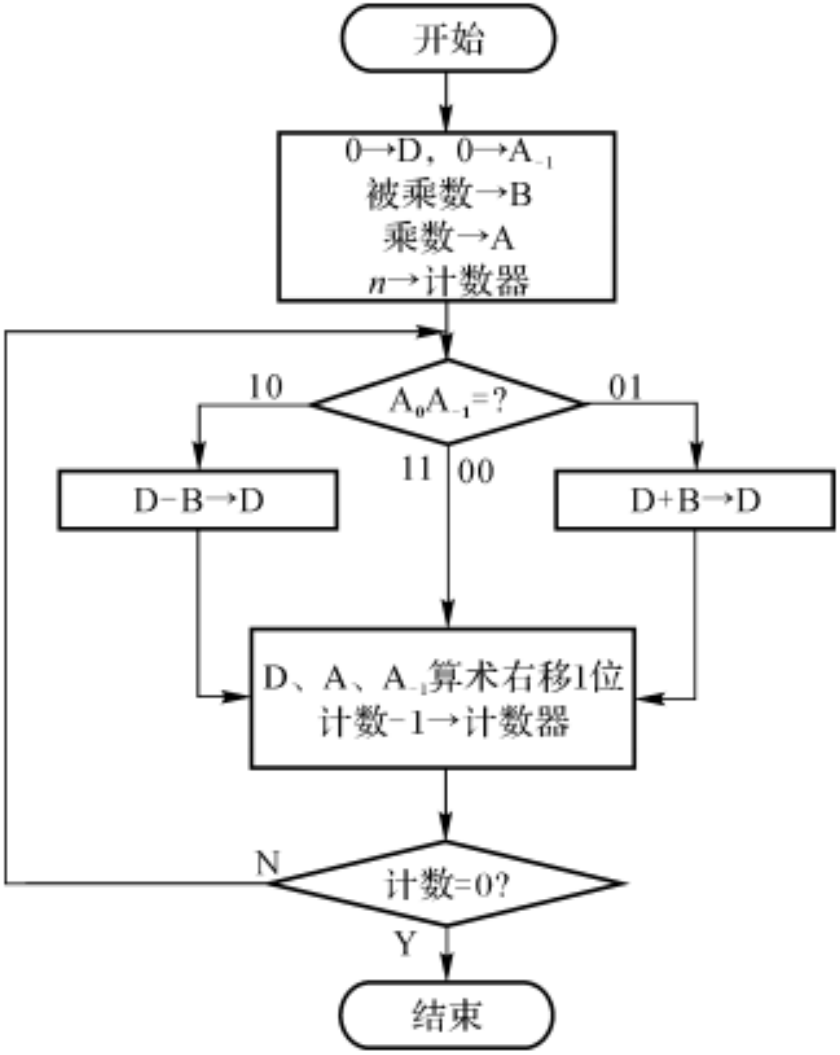


图3.13 布斯算法流程图

2)布斯算法乘法器的硬件框图

根据布斯算法的描述,可以设计出乘法器的硬件框图,如图3.15所示。

2)布斯算法乘法器的硬件框图

根据布斯算法的描述,可以设计出乘法器的硬件框图,如图3.15所示。

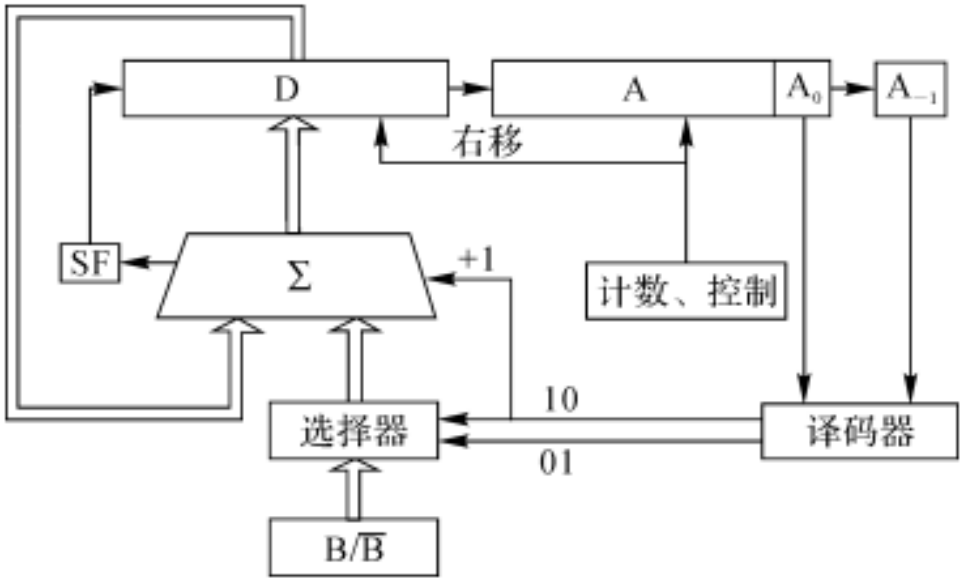


图3.15 布斯算法乘法器的硬件框图

3.阵列乘法器

1)手算及单元电路

在上述乘法运算中,是利用简单的硬件进行多次加法和多次移位来实现乘法的。显然, 这样难以获得高的运算速度。为了提高运算速度,可以采取类似人工手算的方法。

设二进制数 $X=X_3X_2X_1X_0$和$Y=Y_3Y_2Y_1Y_0$,计算$Z=XY$,列式如下:

$$
\begin{array}{ccccccc}
 & & & X_3 & X_2 & X_1 & X_0 \\
 & & \times & Y_3 & Y_2 & Y_1 & Y_0 \\
\hline
 & & & X_3Y_0 & X_2Y_0 & X_1Y_0 & X_0Y_0 \\
 & & X_3Y_1 & X_2Y_1 & X_1Y_1 & X_0Y_1 & \\
 & X_3Y_2 & X_2Y_2 & X_1Y_2 & X_0Y_2 & & \\
X_3Y_3 & X_2Y_3 & X_1Y_3 & X_0Y_3 & & & \\
\hline
Z_6 & Z_5 & Z_4 & Z_3 & Z_2 & Z_1 & Z_0
\end{array}
$$

从上式可以看到,X_iY_j是与运算,而Z_i是对相应列中各个与结果的求和。每一对相与 求和操作可以用图3.16所示的基本乘加单元电路来实现。

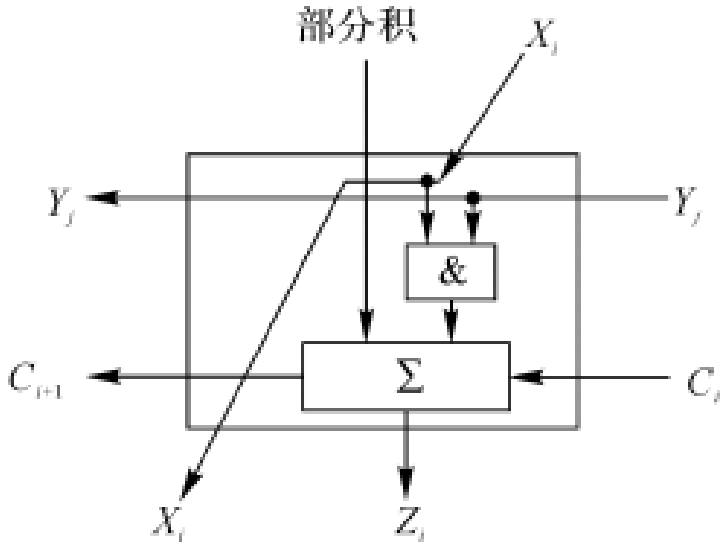


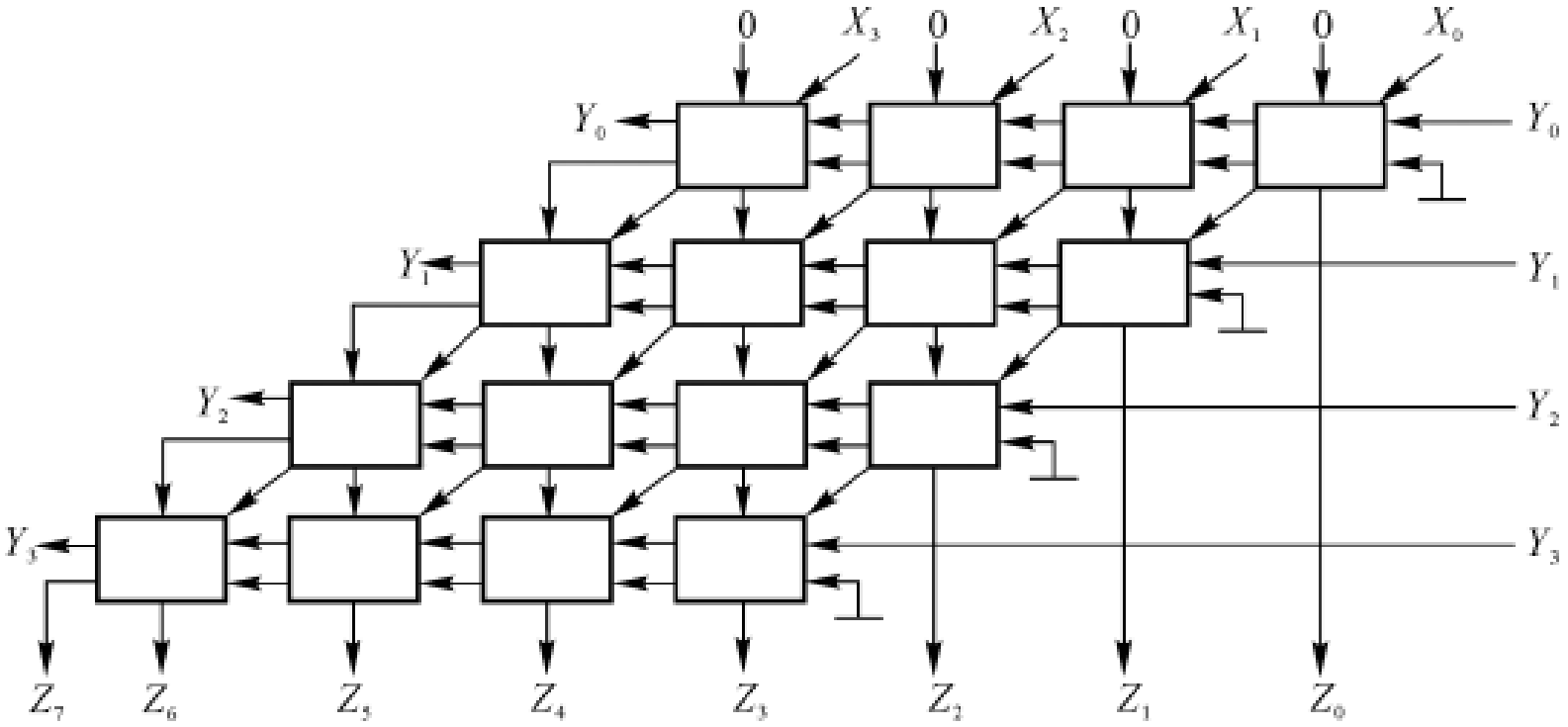
图3.16 基本乘加单元电路

2)无符号数阵列乘法器

利用手算算式的结构及乘加单元电路可以方便地实现无符号数阵列乘法器,其结构如 图3.17所示。

在图3.17中,每一个小框即为一个基本乘加单元,这些基本单元按照类似于手算算式 的结构进行连接,能够完成手算算式中的乘加功能,最终获得两数的乘积。

利用无符号数阵列乘法器完成原码的数值相乘,再加入一个完成符号运算的异或门, 就构成了原码阵列乘法器。


0
X_3
0
X_2
0
X_1
0
X_0
Y_0
Y_1
Y_2
Y_3
Y_0
Y_1
Y_2
Y_3
Z_7
Z_6
Z_5
Z_4
Z_3
Z_2
Z_1
Z_0

THE END !

THANKS