第五章　运算器

§5·1· 概述

运算器是执行算术运算（加、减、乘、除）和邏輯运算及其它操作的部件·

在整机中运算器可直接与存儲器进行信息代码交換，而执行具体操作，运算器有时也与外部设备发生信息交換·因此，运算器是计算机的中心部件·

由于运算器要执行很多计算及邏輯判断工作，因此，其运算速度是很重要的，速度愈快，则在单位时间內处理的內容愈多，則用途愈大，尤其在处理內容較多时和对象过程变化很快的条件下，均要求计算机的速度提高·在通用电子计算机中目前可达到每秒几百万次的运算速度·然而，在工业控制用计算机中則不需要这样高的速度，一般每秒几千次到几万次就可以满足要求了·

运算器分为并行、串行、串并行等运算方式·一般采用并行运算方式的为并行机，串行运算的为串行机，串并行运算的为串并行机·

§5·2· 半加器、全加器、減法器

为了对运算器較清楚的了解，这里叙述半加器、全加器、減法器·

1. 半加器：两个数码相加，不考虑进位称之为半加器·

设两个数，被加数为X，加数为y，和数为S，进位数为C，則可列出半加表如下：

表5·1　半加表

被加数	加数	和数	进位数
X	y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

由上表可列出布尔表达式如下：

$$\begin{cases} \overline{X}y + X\overline{y} = S \\ X \cdot y = C \end{cases}$$

根据布尔表达式得到半加器邏輯：

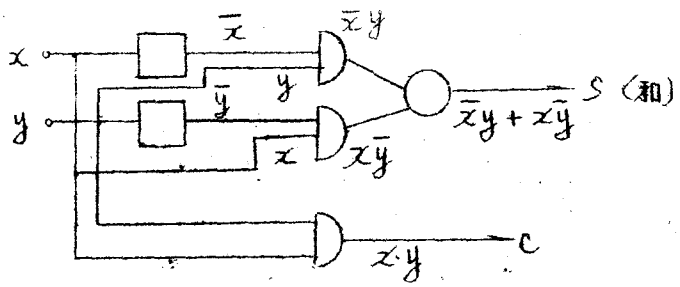

图 5 • 1 半加器邏輯图（a）

图 5 • 1 只是半加器的一种形式，实际半加器的形式很多，将上述半加器的布尔表达式变换形式即可得到不同的半加器，变换形式的目的是，适应设计机器邏輯的统一性，简单方便灵活。图 5 • 1 为“与或”邏輯。

将半加器布尔表示式变换如下：

(1)
$$\begin{cases} S = \overline{X}y + X\overline{y} = \overline{X} \cdot y + X\overline{y} + X\overline{X} + y\overline{y} = (X+y)(\overline{X}+\overline{y}) \\ C = Xy = \overline{\overline{X} + \overline{y}} \end{cases}$$

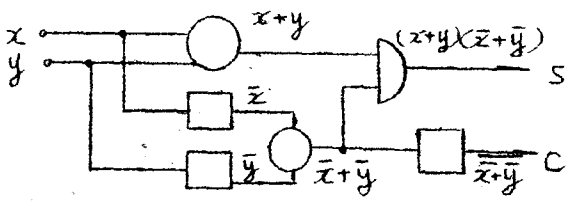

图 5 • 2 半加器邏輯图（b）

(2)
$$\begin{cases} S = (X+y)(\overline{X}+\overline{y}) = (X+y)(\overline{X \cdot y}) \\ C = X \cdot y \end{cases}$$

有时为了简化起见，用一符号  表示半加器。

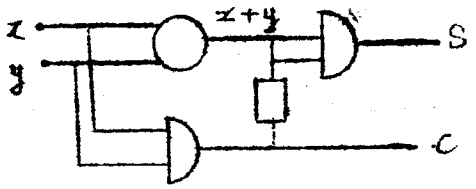

图5·3 半加器逻辑图（C）

2. 全加器：

在二进制（或十进制）两数相加过程中，两数相加的和数一定要考虑低位向高位有无进位，因而实际上是三个数相加，两数相加考虑进位的加法器称为全加器。

设：X_i —— 被加数第i位；

y_i —— 加数第i位；

C_{i-1} —— 第（i-1）位向第i位的进位；

C_i —— 第i位向第（i+1）位的进位；

S_i —— 第i位的和。

列出全加表：

图5·2 全加表

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

由全加表得到布尔表达式：

$$\begin{cases} S_i = \bar{X}_i y_i \bar{C}_{i-1} + X_i \bar{y}_i \bar{C}_{i-1} + \bar{X}_i \bar{y}_i C_{i-1} + X_i y_i C_{i-1} & \cdots\cdots(1) \\ C_i = X_i y_i \bar{C}_{i-1} + \bar{X}_i y_i C_{i-1} + X_i \bar{y}_i C_{i-1} + X_i y_i C_{i-1} & \cdots\cdots(2) \end{cases}$$

进位 C_i 可以这样来简化，由前面布尔表达式的几个公式知 $A+A=A$，将 C_i 表达式右方增加原有的两项，即：$X_i y_i C_{i-1} + X_i y_i C_{i-1}$

$$\begin{aligned} 则\ C_i &= (X_i y_i \bar{C}_{i-1} + X_i y_i C_{i-1}) + (\bar{X}_i y_i C_{i-1} + X_i y_i C_{i-1}) + \\ &\quad + (X_i \bar{y}_i C_{i-1} + X_i y_i C_{i-1}) \\ &= X_i y_i (\bar{C}_{i-1} + C_{i-1}) + y_i C_{i-1} (\bar{X}_i + X_i) + X_i C_{i-1} (\bar{y}_i + y_i) \\ &= X_i y_i + y_i C_{i-1} + X_i C_{i-1} \end{aligned}$$

$$则 \begin{cases} S_i = \bar{X}_i y_i \bar{C}_{i-1} + X_i \bar{y}_i \bar{C}_{i-1} + \bar{X}_i \bar{y}_i C_{i-1} + X_i y_i C_{i-1} & \cdots\cdots(3) \\ C_i = X_i y_i + y_i C_{i-1} + X_i C_{i-1} & \cdots\cdots(4) \end{cases}$$

3. 并行加法器：

上述(1)(2)式是基本的全加器表达式，在计算机中如何实现这表达式呢？下面举出一些并行加法器的实现方法。

在并行相加运算中，一般需要两个寄存器，一个寄存被加数及和数称为累加器，一个寄存加数称为加数寄存器。

由前述知，并行相加过程中，有这样两个动作，一是相加的两数“相加”，二是“进位”。对“相加”和“进位”这两个动作也有不同的处理方法，因此分成先相加后进位加法器，先进位后相加加法器，这两种加法器的都是用二个节拍来完成的，所以可通称为二拍式加法器。

除了二拍式加法器外，尚有把进位和相加这两个动作在一拍中实现，这就是单拍同步式加法器。

下面对上述几种型式分别予以叙述。

(1) 先相加后进位加法器：

由式(1)知：$S_i=\bar{X}_i y_i \bar{C}_{i-1}+X_i \bar{y}_i \bar{C}_{i-1}+\bar{X}_i \bar{y}_i C_{i-1}+X_i y_i C_{i-1}$

可化成　：$S_i=(X_i y_i+\bar{X}_i \bar{y}_i)C_{i-1}+(\bar{X}_i y_i+X_i \bar{y}_i)\bar{C}_{i-1}$

$$=(\overline{X_i \varphi y_i})C_{i-1}+(X_i \varphi y_i)\bar{C}_{i-1} \quad \cdots\cdots\cdots\cdots(5)$$

（其中：

$$X_i \varphi y_i=\bar{X}_i y_i+X_i \bar{y}_i$$

$$\overline{X_i \varphi y_i}=\overline{\bar{X}_i y_i+X_i \bar{y}_i}=\overline{\bar{X}_i y_i} \cdot \overline{X_i \bar{y}_i}$$

$$=(X_i+\bar{y}_i)(\bar{X}_i+y_i)$$

$$=X_i \bar{X}_i+X_i y_i+\bar{X}_i \bar{y}_i+y_i \bar{y}_i$$

$$=X_i y_i+\bar{X}_i \bar{y}_i)$$

由(2)式：$C_i=X_i y_i \bar{C}_{i-1}+\bar{X}_i y_i C_{i-1}+X_i \bar{y}_i C_{i-1}+X_i y_i C_{i-1}$

可化成　$C_i=X_i y_i(\bar{C}_{i-1}+C_{i-1})+(\bar{X}_i y_i+X_i \bar{y}_i)C_{i-1}$

$$=X_i y_i+(X_i \varphi y_i)C_{i-1}$$

$$=X_i y_i \cdot y_i+\bar{X}_i \bar{y}_i y_i+(X_i \varphi y_i)C_{i-1}$$

$$=(\overline{X_i \varphi y_i})y_i+(X_i \varphi y_i)C_{i-1} \quad \cdots\cdots\cdots\cdots(6)$$

依(5)(6)两式，我們可令第一拍（＋C，－C）求得（$X_i \varphi y_i$），第二拍（＋C'，－C'）求得进位C_i再由C_i与（$X_i \varphi y_i$）进行按位加而得到和数S_i。

这里＋C命令表示相加运算，－C命令表示相減运算。

这里＋C'命令表示相加运算，－C'命令表示相減运算。

依(5)(6)两式可得到一位加法器邏輯图如下：

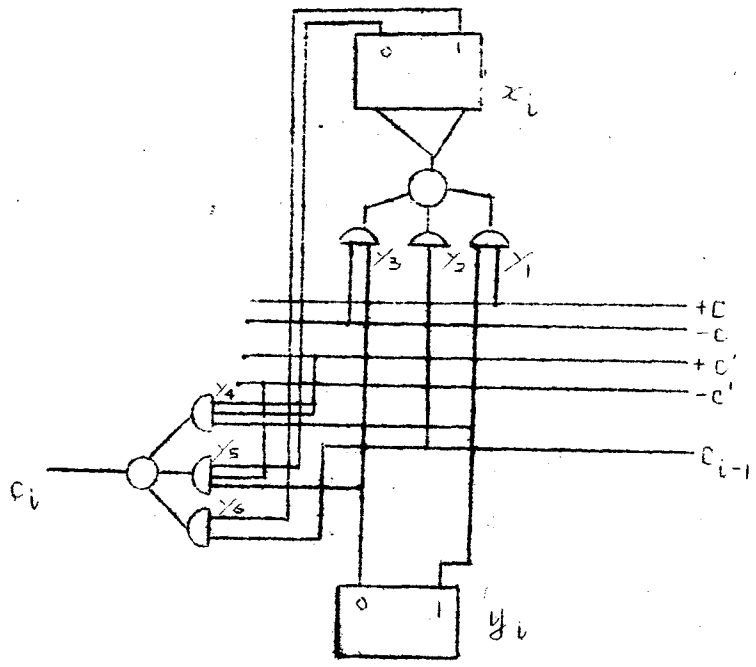

图 5 · 4　先加后进位一位加法器

如上图知，当相加时，第一拍 + C 完成 $X_i \varphi y_i$ ，假设初始状态 X_i 为 0，y_i 为 1，则 + C 命令到来时 y_1 与门开放，给 X_i 触发器计数得到部分和 $X_i \varphi y_i$ ，这时 X_i 状态为 1；

第二拍 + C ' 命令时，设 C_{i-1} 为 0，则 X_i 状态不变且与门 y_4 y_5 、y_6 均不通，则进位为 0，如果 $C_{i-1}=1$，则 X_i 状态又翻转为 0 得到和数 $S_i=0$，进位 $C_i=1$ 。

(2)　先进位，后相加加法器

由(4)式知：$C_i=X_i y_i+y_i C_{i-1}+X_i C_{i-1}$

$$=X_i y_i+(X_i+y_i)C_{i-1} \qquad \cdots\cdots\cdots\cdots(7)$$

由(7)式知，在 X_i、y_i、C_{i-1} 三者中有二个条件具备时即产生进位，这样我们就可以先得到进位信号 C_i ，然后根据进位信号和相应 y_i 的状态而得到和数 S_i 。

因为由(1)式知：$S_i=\bar{X}_i y_i \bar{C}_{i-1}+X_i \bar{y}_i \bar{C}_{i-1}+\bar{X}_i \bar{y}_i C_{i-1}+X_i y_i C_{i-1}$

式中被加数 X_i 事先已存放在累加器中（用B表示），如果累加器B中对应位 B_i 原来为“1”态，而根据(1)式和数 S_i 存于 B_i 中，第2，4两项满足 $S_i=1$，则表示 B_i 状态不变，因此2、4两项为多余项，可省略，这样(1)式可变成

$$S_i=\bar{X}_i y_i \bar{C}_{i-1}+\bar{X}_i \bar{y}_i C_{i-1} \qquad \cdots\cdots(8)$$

同理 $$\bar{S}_i=X_i \bar{y}_i C_{i-1}+X_i y_i \bar{C}_{i-1} \qquad \cdots\cdots(9)$$

又设B累加器可计数输入，也就是说如果 B_i 原来为“1”时，当和数 S_i 为“0”时，B_i 改变状态；当 B_i 原来为“0”时，和数为“1”时也改变 B_i 状态。这样就可根据(8)(9)二式进一步简化：

$$\begin{aligned}B_i&=S_i+\bar{S}_i=\bar{X}_i y_i \bar{C}_{i-1}+\bar{X}_i \bar{y}_i C_{i-1}+X_i \bar{y}_i C_{i-1}+X_i y_i \bar{C}_{i-1}\\&=\bar{X}_i(y_i \bar{C}_{i-1}+\bar{y}_i C_{i-1})+X_i(\bar{y}_i C_{i-1}+y_i \bar{C}_{i-1})\\&=(y_i \bar{C}_{i-1}+\bar{y}_i C_{i-1})+(\bar{X}_i+X_i)\\&=y_i \bar{C}_{i-1}+\bar{y}_i C_{i-1}\end{aligned}$$

$$\therefore\begin{cases}B_i=y_i \bar{C}_{i-1}+\bar{y}_i C_{i-1} & \cdots\cdots(10)\\ C_i=X_i y_i+X_i C_{i-1}+y_i C_{i-1} & \cdots\cdots(11)\\ \bar{C}_i=\bar{X}_i \bar{y}_i+\bar{X}_i \bar{C}_{i-1}+\bar{y}_i \bar{C}_{i-1} & \cdots\cdots(12)\end{cases}$$

由(5)(6)(7)式可得到并行相加邏輯：

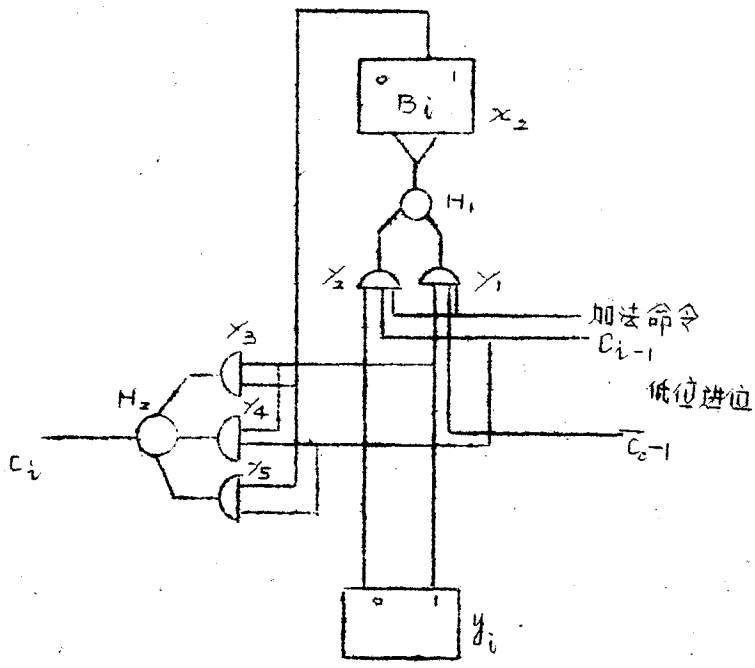

图 5 • 5　先进位后相加一位加法器邏輯

上图“与门” Y_1 实现 $y_i\bar{C}_{i-1}$，Y_2 实现 $\bar{y}_iC_{i-1}$ “或”门 H_1 得到 $y_i\bar{C}_{i-1}+\bar{y}_iC_{i-1}$，这输出信号去计数得到和数存于 B_i 中。

进位的实现是通过“与门” Y_3，Y_4，Y_5，“或门” H_2 组成的进位鏈实现的。当加命令来时，则完成单拍相加作用，因为进位信号已形成。

在实际的邏輯上要考虑负载的匹配和驱动问题，因此要在原来邏輯的基础上增加反相器，又考虑到同步工作方式（选通工作方式，前面第二章已叙述过）则同步式一位全加器邏輯图如下：

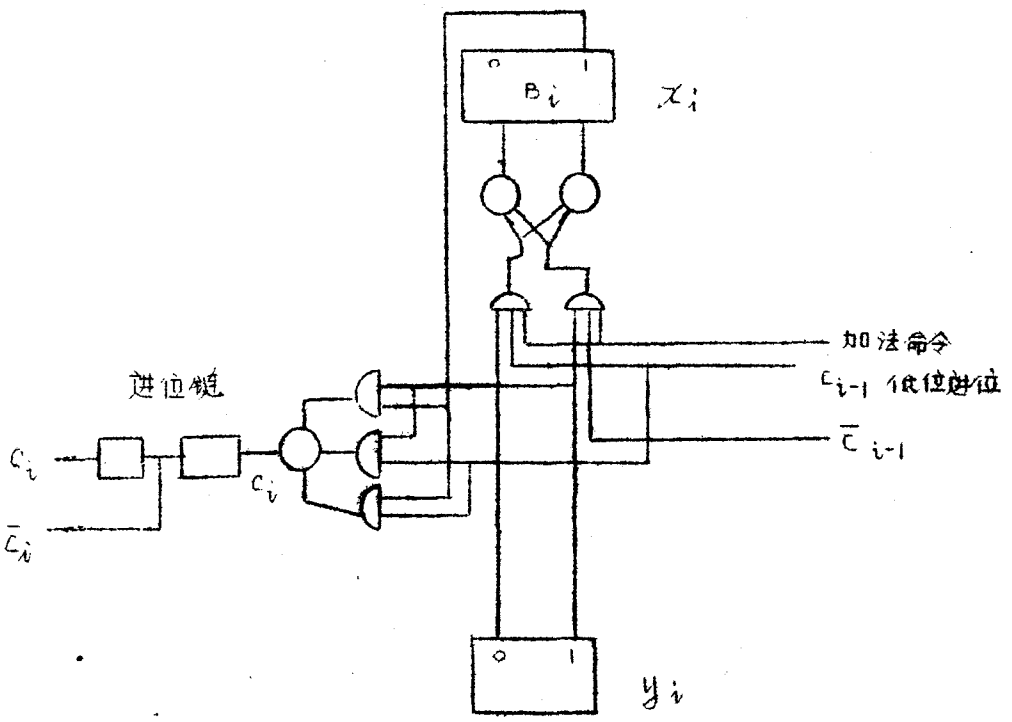

图 5 • 6　同步式一位全加器邏輯图

为使读者清楚起见，下面给出同步式一位全加器线路图：

在并行运算中，为了提高运算速度，尽量减小进位鏈的傳输延迟时间，在同步式加法器中，如果进位延迟时间，小于同步脉冲周期的$\frac{1}{2}$，则可完成单拍相加运算，这对于提高运算速度，特别减小乘、除法运算时间是大有好处的。从这一点来说，先进位后相加邏輯比先相加后进位邏輯优越，先相加后进位必需要二拍。

JDK — 331机系采用单拍同步式加法邏輯的。为了实现单拍加法运算操作，减小进位鏈延迟时间采用“正反邏輯交替进位鏈”。

由图 5 •　知，如果进位这样一级一级由低位向高位傳输，在最坏情况下，18位字长时傳输信号需经过18级“与或非非”线路，这延迟是较大的，如果采用交替进位法，如下图示：

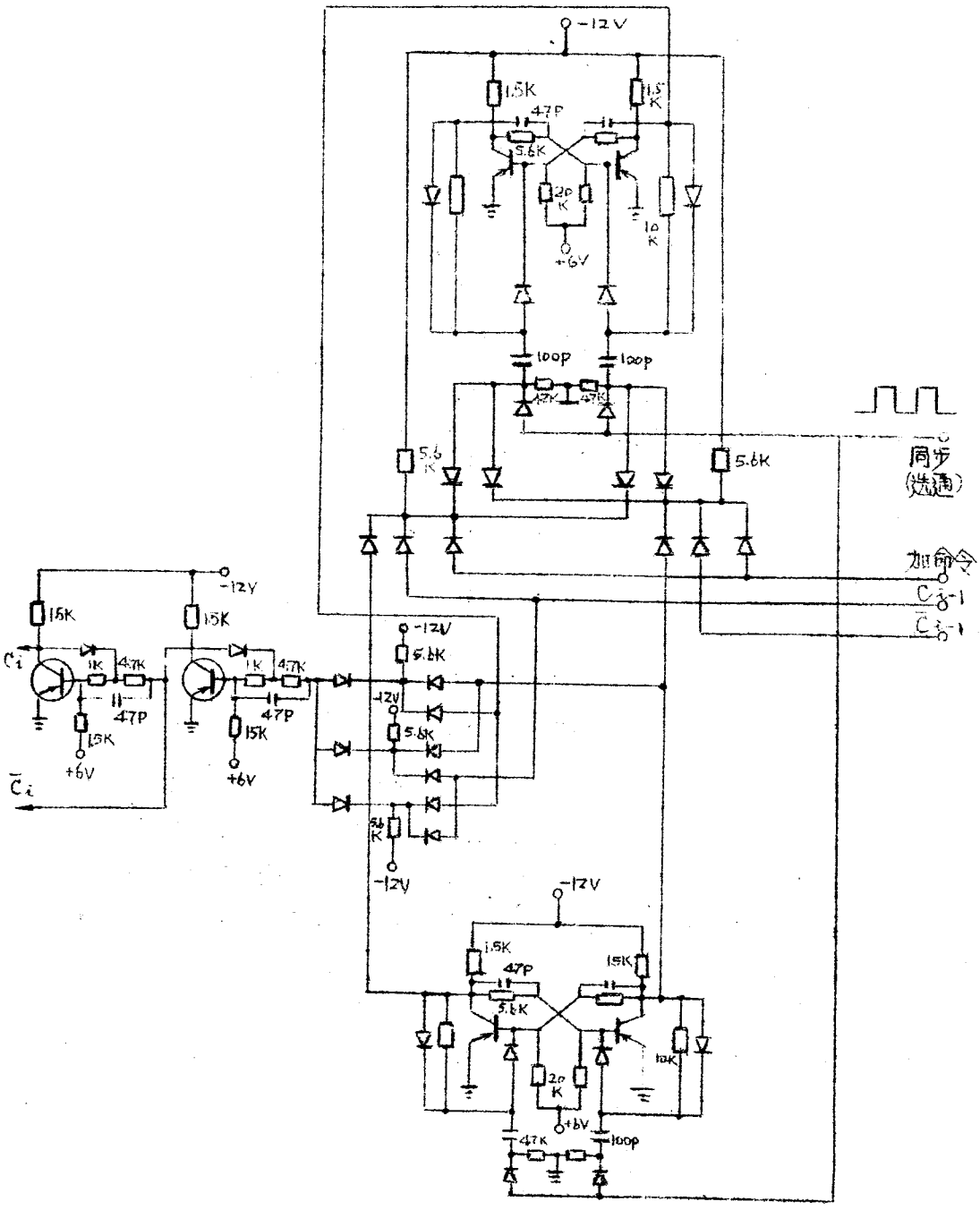

图 5 · 7　一位加法器线路

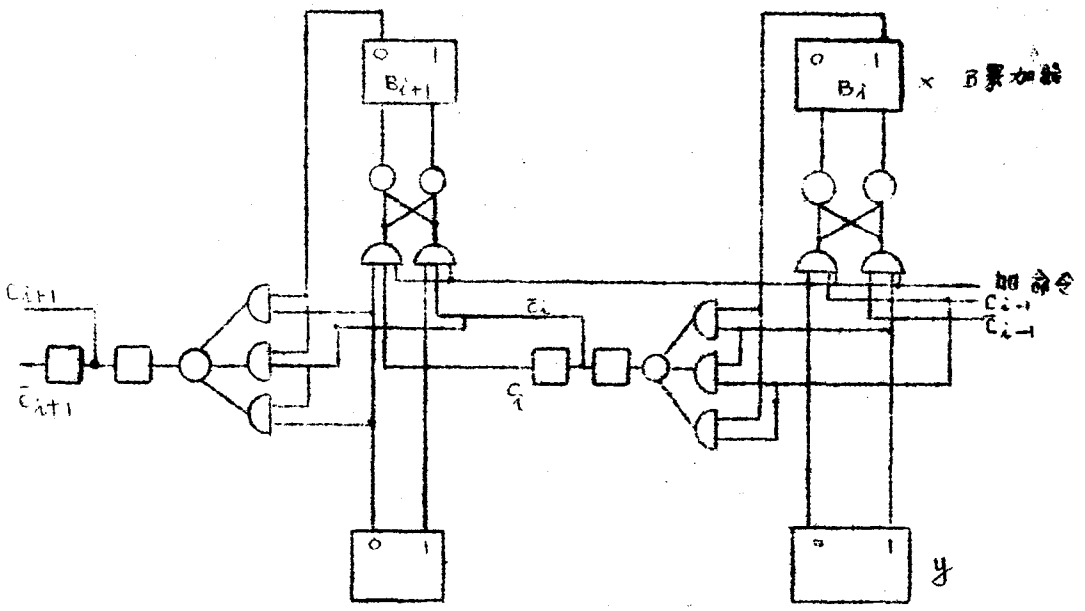

图 5 · 8　正反邏輯交替进位加法器原理图

正反邏輯交替进位原理是第一级用有进位式

$$C_i = X_i y_i + X_i C_{i-1} + y_i C_{i-1}$$　　形成进位，

第二级用无进位式 $\overline{C}_i = \overline{X}_i \overline{y}_i + \overline{X}_i \overline{C}_{i-1} + \overline{y}_i \overline{C}_{i-1}$ 形成进位，这样就构成了正反邏輯交替进位了，这样由低位向高位的进位傳输时间最坏情况通过 1 8 级“与或非”邏輯，因此減少了 1 8 级非门（反相器）的延迟时间。

§5 · 3 · 乘、除法及其它操作的实现

在实际的运算器中除了上述作加法这个最基本、最核心的操作之外，尚须作许多其它操作。

1. 乘法操作：

由第四章中知，乘法操作不过是一系列的加，右移而已，而加的

操作已叙述了，只是右移操作了。

另外，乘法操作，需要三个寄存器，一个存放被乘数，一个存放乘数，一个存放乘积。所以运算器一般由三个寄存器构成，分别叫做A寄存器，B累加器，C寄存器，在作乘法操作时，A寄存器存放乘数，B累加器存放乘积，C寄存器存放被乘数。根据乘法操作规则，只是乘数和部分积右移，因此，只需将A寄存器，B累加器右移即可。

右移的方法一般采用双与门移位法，逻辑图如下：

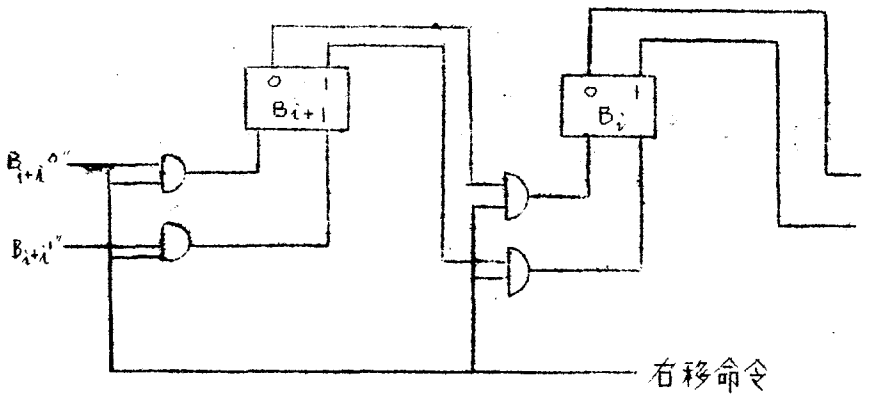

图5·9　右移逻辑

2. 除法操作：

由第四章知，除法操作用恢复余数法操作时是一系列的加或减，左移操作；用不恢复余数法操作时是一系列加或减操作，左移操作实现。二者判别方法不同而已。左移操作类同右移操作，采用双与门移位法。

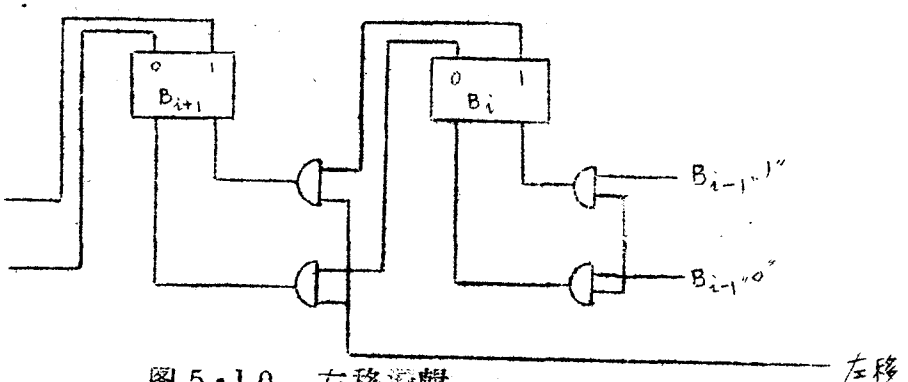

图5·10　左移逻辑

3. 邏輯乘操作：

在邏輯判断中，邏輯乘法用的较普遍，它完成相应位乘法作用，所以也叫按位乘邏輯符号为∧，邏輯表如下：

表5·2　　邏輯乘法

		B	
	∧	0	1
A	0	0	0
	1	0	1

A ∧ B

0 ∧ 0 = 0

0 ∧ 1 = 0

1 ∧ 0 = 0

1 ∧ 1 = 1

它广泛用于“抽数”，例如，A = 01110101 如果我們希望把A前4位保留，后四位去掉，则用邏輯乘B = 11110000 即可：

$$A \wedge B = C$$

$$\begin{array}{r} A = 0\,1\,1\,1\,0\,1\,0\,1 \\ \wedge\quad B = 1\,1\,1\,1\,0\,0\,0\,0 \\ \hline C = 0\,1\,1\,1\,0\,0\,0\,0 \end{array}$$

C前四位同A前四位。

在实际运用中，可抽出任何位。

在运算器中如何实现邏輯乘的操作呢？

如下图所示：

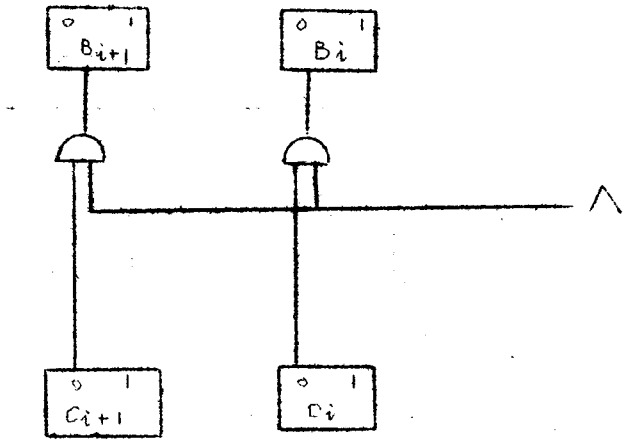

图5·11　邏輯乘邏輯

当A命令到来时，若$C_i=0$，则无条件的将B_i置0，当$C_i=1$则B_i保留原来状态，即达到了上述目的。

4. 按位加（也称不进位加）操作：

逻辑符号为φ

逻辑表

表5·3　　逻辑加表

A \ B	φ	0	1
0	0	0	1
1	1	1	0

A φ B
0 φ 0 = 0
0 φ 1 = 1
1 φ 0 = 1
1 φ 1 = 0

由表知逻辑式 $A\varphi B=A\bar{B}+\bar{A}B$

逻辑图：

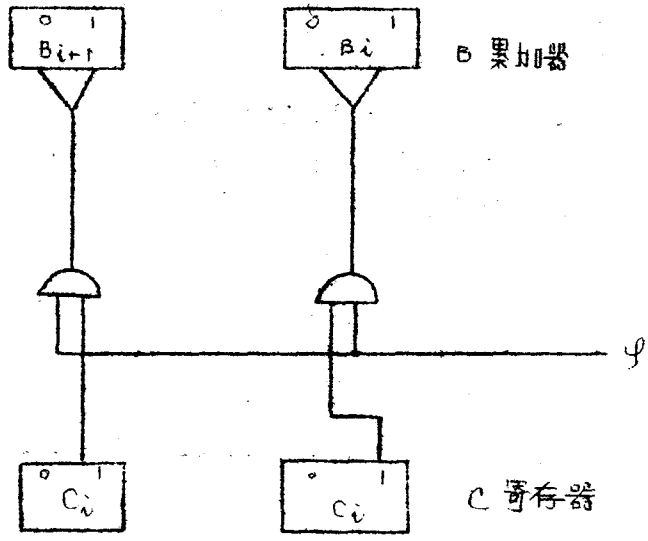

图5·12　按位加逻辑

由图知，当C_i为1时，φ命令到来时，B_i状态改变即是完成0变1，1变0（∵0φ1=1，1φ1=0），当$C_i=0$，则B_i状态不

变。

例：　A = 01110101　　B = 11110001

$$
\begin{array}{r}
A \varphi B = C \\
A = 0\,1\,1\,1\,0\,1\,0\,1 \\
\varphi B = 1\,1\,1\,1\,0\,0\,0\,1 \\
\hline
C = 1\,0\,0\,0\,0\,1\,0\,0
\end{array}
$$

5. 变反操作：

在运算过程中，尤其负数运算，或减法操作时往往需要变成补码（也即反码＋1）进行，因而变反操作是必要的。变反操作有下列两种方式：

(1) 反码发送：

将C寄存器的内容从0端输出，代表反码“1”输出。如图示，当－C反码发送时，0端有输出。

(2) 寄存器本身变反：

当变反命令到来时，C寄存器作计数，则0→1，1→0完成变反的目的。

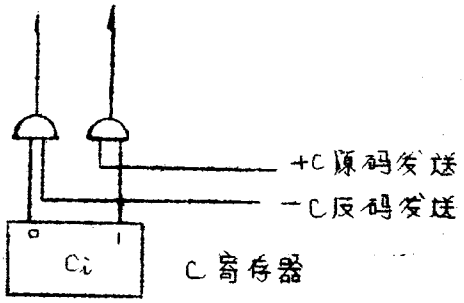

图5·13　原、反码发送邏輯

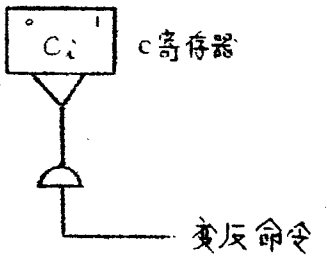

图5·14　变反操作

在补码运算中，需要的是补码，也即反码末位＋1，当变反后，尚须附加一附助位C_0，当变反时C_0置1，这样C_0在最末一位等效于最低位有进位“1”，因此两数相加时，等效反码末位＋1，从而完成补码运算目的。当加命令来时将C_0复位。

6. 代码傳送操作：

在运算过程中，往往需要代码在寄存器中互相傳送，这操作有下列几种形式：

(1) 先清除后傳送：如 B→A

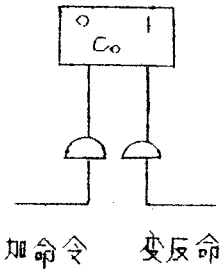

图 5·15 辅助位 C_0

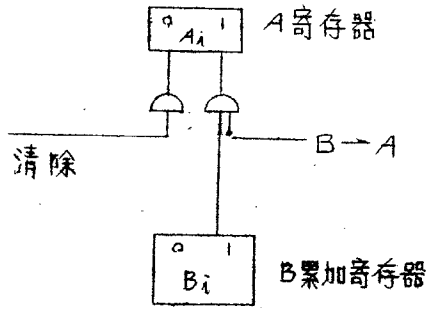

图 5·16 先清除后傳送邏輯

第一拍将 A_i 清除为“0”态，第二拍 B→A 命令将 B_i 內容傳送给 A_i，需二拍时间。

(2) 双端傳送：如 B ⇒A

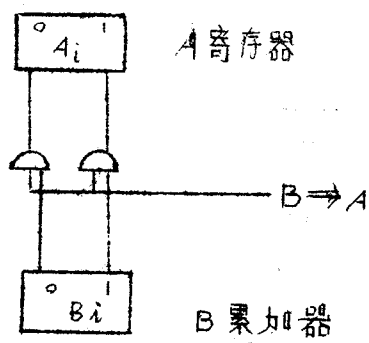

图 5·17 双端傳送邏輯

双端傳送则不管 A_i 状态如何，当 B ⇒A 命令到来时，B_i 內容无条件地傳送给 A_i，只一拍时间。