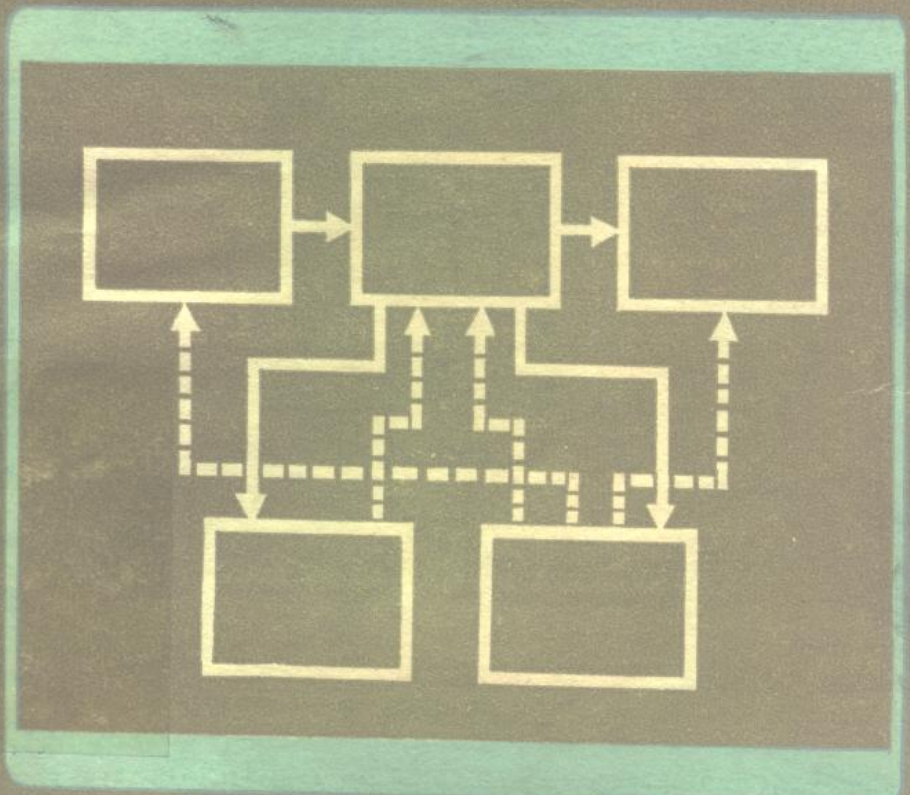
广播电视大学教材

数字电子计算机原理

下 册

李大友 编



高等教育出版社

广播电视大学教材

数字电子计算机原理

下 册

李大友 编

高等教育出版社

内 容 简 介

本书较全面地介绍了数字电子计算机硬件组成、工作原理及微型计算机系统组织；在软件方面，介绍了汇编语言及其程序设计的基本方法。各章均附有习题(包括思考题)。

全书分上下两册出版。上册包括计算机的一般描述、运算基础、逻辑基础、计算机结构、指令系统、汇编语言及其程序设计初步；下册包括运算方法、运算器、控制器、存贮器、外[illegible]设备、接口、信息传送方式和微型计算机系统组织。

本书为中央广播电视大学电子类专业教材，也可供高等工科院校有关专业师生及工程技术人员参考。

本书责任编辑是姚玉洁。

广播电视大学教材
数字电子计算机原理
下 册
李大友编
*
高等教育出版社出版
新华书店北京发行所发行
北京印刷一厂印装
*
开本 787×1092 1/32 印张 10.5 字数 240,000
1981 年 4 月第 1 版 1985 年 8 月第 5 次印刷
印数 259,261—271,110
书号 15010·0324 定价 1.70 元

目　录

第八章　运算方法和运算器

本章主要讨论在计算机中实现算术运算和逻辑运算的方法及数据信息在运算器中的传送过程。

由于加减法采用补码比较方便，乘除法采用原码比较方便，所以重点介绍补码加减法和原码乘除法。

§8.1　定点补码加减法运算

一、定点补码加法运算

第二章曾介绍过，为了便于判断溢出，在运算过程中通常采用变形码。如不加说明，下面提到的补码均为变形补码。

补码加法按和之补码等于补码之和的规则进行，即

$$[X+Y]_{补}=[X]_{补}+[Y]_{补} \pmod{2^{n+1}}$$

例 1

已知 $X=0\,1\,0\,1\,0\,0, Y=0\,0\,0\,1\,0\,1$，试进行补码加法运算。

$$\begin{array}{rl} & [X]_{补}=0\,0\quad 0\,1\,0\,1\,0\,0 \\ +) & [Y]_{补}=0\,0\quad 0\,0\,0\,1\,0\,1 \\ \hline & [X+Y]_{补}=0\,0\quad 0\,1\,1\,0\,0\,1 \end{array}$$

所以　　$X+Y=0\,1\,1\,0\,0\,1$

例 2

已知 $X=0\,1\,0\,0\,1\,1, Y=-0\,0\,0\,1\,1\,1$，试进行补码加法运算。

$$
\begin{array}{rl}
[X]_{补}= & 0\ 0\ 0\ 1\ 0\ 0\ 1\ 1 \\
+)\ [Y]_{补}= & 1\ 1\ 1\ 1\ 1\ 0\ 0\ 1 \\
\hline
[X+Y]_{补}= & \boxed{1}\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0
\end{array}
$$

丢失

所以 $X+Y=0\,0\,1\,1\,0\,0$

例 3

已知 $X=-0\,1\,1\,0\,0\,1$，$Y=-0\,0\,0\,1\,1\,0$，试进行补码加法运算。

$$
\begin{array}{rl}
[X]_{补}= & 1\ 1\ 1\ 0\ 0\ 1\ 1\ 1 \\
+)\ [Y]_{补}= & 1\ 1\ 1\ 1\ 1\ 0\ 1\ 0 \\
\hline
[X+Y]_{补}= & \boxed{1}\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 1
\end{array}
$$

丢失

所以 $X+Y=-0\,1\,1\,1\,1\,1$

从上面三个例子可见，用补码进行加法运算，只要将给定的真值用补码表示，就可以直接做加法运算。在运算过程中不必判断加数和被加数的正负，一律做加法。将运算结果转换为真值即可。

在计算机中，数据一律以机器数的形式出现（不是以真值的形式出现）。数据在存贮器中可以以原码的形式存贮，这时如进行补码运算，数据从存贮器中取出后，在参加运算之前由计算机自行转换为补码表示，然后参加运算；也可以以补码形式存贮，这时进行补码运算时，数据从存贮器取出后即可参加运算，就不必进行转换了。但是，由于加减法多进行补码运算，而乘除法又多进行原码运算，所以数据究竟以原码形式还是以补码形式存贮，可视设计要求而定。

二、定点补码减法运算

对于定点补码减法运算，由于存在：

$$[X-Y]_{补}=[X+(-Y)]_{补}$$

$$=[X]_{补}+[-Y]_{补} \pmod{2^{n+1}}$$

所以，在进行定点补码减法运算时，只将减数 Y 转换为 $[-Y]_{补}$，减法运算便转换成加法运算了。在§2.5中曾介绍过已知 $[Y]_{补}$ 求 $[-Y]_{补}$ 的方法，这时只要将 $[Y]_{补}$ 进行一次求补操作，即可得到 $[-Y]_{补}$（即对 $[Y]_{补}$ 连同符号位一起求反加1）。所以，补码减法运算，可归结为**对减数求补，然后做加法。**

例1

已知 $X=0\,1\,0\,1\,1\,0$，$Y=0\,1\,1\,0\,0\,1$，试用补码进行减法运算。

$$[X]_{补}=0\,0\quad 0\,1\,0\,1\,1\,0$$

$$[Y]_{补}=0\,0\quad 0\,1\,1\,0\,0\,1$$

$$[-Y]_{补}=1\,1\quad 1\,0\,0\,1\,1\,1$$

$$[X]_{补}=0\,0\quad 0\,1\,0\,1\,1\,0$$

$$+)\ [-Y]_{补}=1\,1\quad 1\,0\,0\,1\,1\,1$$

$$[X-Y]_{补}=1\,1\quad 1\,1\,1\,1\,0\,1$$

所以 $$X-Y=-0\,0\,0\,0\,1\,1$$

例2

已知 $X=-0\,1\,0\,1\,0\,1$，$Y=-0\,1\,1\,0\,1\,0$，试用补码进行减法运算。

$$[X]_{补}=1\,1\quad 1\,0\,1\,0\,1\,1$$

$$[Y]_{补}=1\,1\quad 1\,0\,0\,1\,1\,0$$

$$[-Y]_{补}=0\,0\quad 0\,1\,1\,0\,1\,0$$

$$[X]_{补}=\quad 1\ 1\quad 1\ 0\ 1\ 0\ 1\ 1$$

$$+)[-Y]_{补}=\quad 0\ 0\quad 0\ 1\ 1\ 0\ 1\ 0$$

$$[X-Y]_{补}=\boxed{1}\ 0\ 0\quad 0\ 0\ 0\ 1\ 0\ 1$$

↑ 丢失

所以　　$X-Y=+0\ 0\ 0\ 1\ 0\ 1$

§8.2　加法器及进位系统

加法器用来完成加法运算。它可分为**串行加法器、并行加法器**和**串并行加法器**。在串行加法器中只用一个一位全加器，数据逐位串行送入加法器进行运算；并行加法器由多位全加器组成，其位数多少取决于字长，数据的各位同时参加运算；串并行加法器是前两者的折衷方案。本节只介绍并行加法器。

一、全加器

在§3.8中，已经介绍了全加器的性质和设计方法。从§3.8我们得到全加器表达式为

$$\begin{cases} S_i=\bar{A}_i\bar{B}_iC_{i-1}+\bar{A}_iB_i\bar{C}_{i-1}+A_i\bar{B}_i\bar{C}_{i-1}+A_iB_iC_{i-1} & (8.2.1)\\ C_i=A_iB_i+A_iC_{i-1}+B_iC_{i-1} & (8.2.2) \end{cases}$$

根据表达式(8.2.1)和(8.2.2)可设计出不同形式的全加器逻辑线路(见§3.8)，但归结起来，全加器是一种具有三个输入端和两个输出端的逻辑网络，如图8.2.1所示。

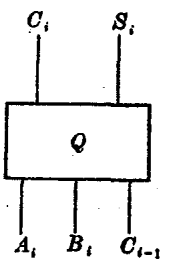


图8.2.1　全加器逻辑符号

其中：

A_i 和 B_i——分别为第 i 位的被加数和加数；

C_{i-1}——第 $i-1$ 位送第 i 位的进位；

S_i——第 i 位的全加和；

C_i——第 i 位送第 $i+1$ 位的进位。

二、进位链

每一位全加器都有一个从较低位送来的进位和向较高位的进位。将各位之间的进位线路连接起来构成的进位逻辑网络，称为**进位链**。

对式(8.2.2)稍加改变，得到

$$C_i = A_iB_i + (A_i + B_i)C_{i-1} \tag{8.2.3}$$

由表达式可知，进位 C_i 可分解为 A_iB_i 和 $(A_i+B_i)C_{i-1}$ 两部分。二者之一为 1，都可以产生进位 C_i。A_iB_i 仅取决于本位参加运算的两数 A_i 和 B_i，而与低位传来的进位 C_{i-1} 无关。因此，称 A_iB_i 为第 i 位产生的**本地进位**。第二项 $(A_i+B_i)C_{i-1}$ 形成进位要由 (A_i+B_i) 和 C_{i-1} 两者决定。若使低位送来的进位 C_{i-1} 能通过本位，则必须 $(A_i+B_i)=1$。每一位的 (A_i+B_i) 都起这样的连通作用。如果 $(A_i+B_i)=1$，就称进位链在第 i 位被**接通**，或者说，低位来的进位能通过本位；如果 $(A_i+B_i)=0$，则进位 $C_{i-1}, C_{i-2}, \cdots$ 对第 i 位，第 $i+1$ 位，…均无影响。这时，我们称进位链在第 i 位被**阻断**。因此，我们称 $(A_i+B_i)C_{i-1}$ 为第 i 位产生的**传送进位**，而 (A_i+B_i) 称为**传送条件**。

进位分为本地进位和传送进位，是进位的基本性质。

三、并行加法器及进位系统

1. 串行进位的并行加法器

图 8.2.2 给出了串行进位的并行加法器的逻辑图。

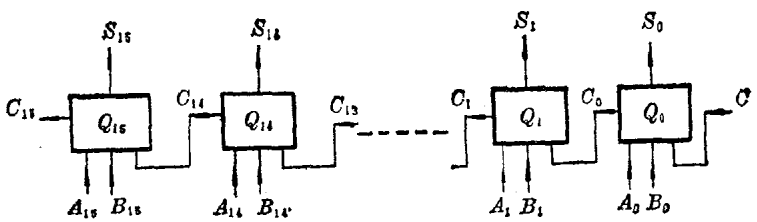


图 8.2.2　串行进位的并行加法器

所谓**并行加法器**，就是参加运算的两数可同时送到各位全加器的输入端，在加法器的输出端一次获得运算结果的加法器。

串行进位并行加法器的进位是串行逐级传递的。这种进位链，称为**串行进位链**。

串行进位链结构简单，容易理解，其缺点是严重地影响了加法器的运算速度。例如，有如下两数 A、B 相加，并且在最低位加 1，则

$$
\begin{array}{rcccccccccccccccc}
A= & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
B= & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\
C= & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\hline
+) & C_{16} & C_{14} & C_{13} & C_{12} & C_{11} & C_{10} & C_9 & C_8 & C_7 & C_6 & C_5 & C_4 & C_3 & C_2 & C_1 & C_0 \quad C \\
\hline
 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \quad 1
\end{array}
$$

第 0 位实际上是三个 1 相加，本位产生全加和为 1，产生进位 $C_0=1$；第 1 位 $A_1=0$，$B_1=1$，$C_0=1$，相加结果全加和为 0，进位 $C_1=1$；第 2 位，$A_2=0$，$B_2=1$，$C_1=1$，相加结果全加和为 0，进位 $C_2=1$；……。

从上式可知，最高位进位 C_{15} 的产生，在这种最坏的情况下，是最低位进位逐级传递的结果。因为进位的传递每经过一级门都要产生延迟，所以当加法器位数很多时，其延迟时间是很可观的，这就影响了加法器的运算速度。

为了进一步了解串行进位链在加法器中的连接，我们举一个十六位串行进位用**与或非**门组成的加法器实例。现将式(3.8.5)，(3.8.6)，(3.8.7)和(3.8.8)重写如下：

$$\begin{cases} \bar{S}_i=\overline{\bar{C}_iA_i+\bar{C}_iB_i+\bar{C}_iC_{i-1}+A_iB_iC_{i-1}} \\ \bar{C}_i=\overline{A_iB_i+A_iC_{i-1}+B_iC_{i-1}} \end{cases} \tag{8.2.4}\tag{8.2.5}$$

$$\begin{cases} S_i=\overline{C_i\bar{A}_i+C_i\bar{B}_i+C_i\bar{C}_{i-1}+\bar{A}_i\bar{B}_i\bar{C}_{i-1}} \\ C_i=\overline{\bar{A}_i\bar{B}_i+\bar{A}_i\bar{C}_{i-1}+\bar{B}_i\bar{C}_{i-1}} \end{cases} \tag{8.2.6}\tag{8.2.7}$$

在串行进位的并行加法器中，为了节省各位之间进位信号的传递时间，我们可以在各位之间交替使用原码输入和反码输入的两组公式，从而得到正负逻辑交替的并行加法器结构，如图 8.2.3 所示。

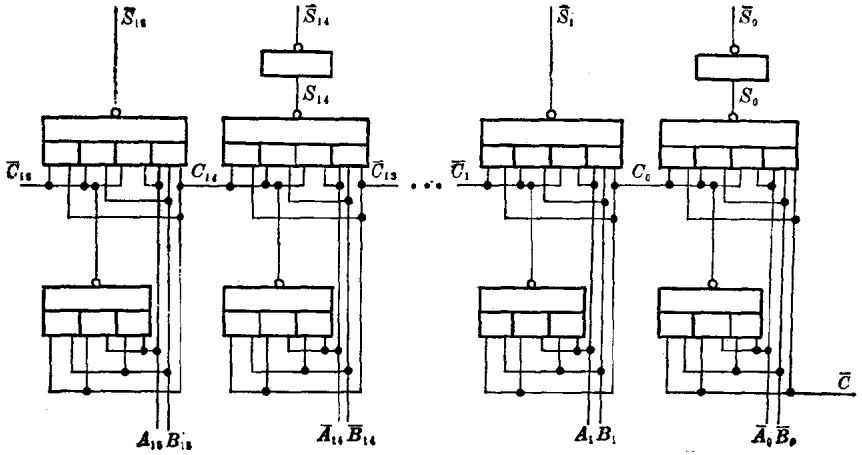


图 8.2.3 正负逻辑交替并行加法器结构

假定能同时提供 A、B 两数从低位到高位的各位的原变量和反变量以及送给最低位的进位 $\bar{C}$，而且各位的**半加和** $A_i \oplus B_i = 1$，我们分析一下串行进位加法器完成一次加法运算最多需要多少时间。

设**与或非**门平均传输延迟时间为 50ns。分析图 8.2.3 可知，从 $\bar{C}$ 的送入到 $\bar{C}_{15}$ 的产生需要 16 级**与或非**门的延迟，到产生全和 $\bar{S}_{15}$，又增加一级**与或非**门的延迟。所以，从送入 A、B 两数，到加法器产生运算结果，在最坏的情况下，需 17 级**与或非**门的延迟时间，$50\text{ns} \times 17 = 850\text{ ns}$。

很显然，串行进位的并行加法器结构简单，但进位传送速度慢，从而影响加法器速度的提高。

由于这种加法器在计算中进位的传送时间太长，不能满足运算速度的要求，所以这种加法器只在速度要求不高时采用。如速度要求高，则需要加以改进。目前提高进位速度的方法主要有两种：一是，采用高速器件以减少传送时间；二是，改进进位链的逻辑设计。下面主要讨论第二种方法。

2. 分组跳跃进位

串行进位链的进位时间太长。为了提高运算速度，必须减少进位传送时间。为此，可以采用并行进位的方法。因为

$$C_i = A_iB_i + (A_i + B_i)C_{i-1} \tag{8.2.8}$$

$$= A_iB_i + (A_i + B_i)[A_{i-1}B_{i-1} + (A_{i-1} + B_{i-1})C_{i-2}] \tag{8.2.9}$$

$$= A_iB_i + (A_i + B_i)\{A_{i-1}B_{i-1} + (A_{i-1} + B_{i-1})[A_{i-2}B_{i-2} + (A_{i-2} + B_{i-2})C_{i-3}]\} \tag{8.2.10}$$

可以看出：在式(8.2.8)中 C_i 的形成直接依赖它的较低一位进位 C_{i-1}；

在式(8.2.9)中，C_i 的形成直接依赖它的较低两位进位 C_{i-2}；

在式(8.2.10)中，C_i 的形成直接依赖它的较低三位进位 C_{i-3}。

如果我们将式(8.2.10)继续推导下去，可以使所有各位的进位都直接依赖最低位进位 C。也就是说，所有各位的进位可以直接从 C 并行产生。这种形式的进位链，称为**并行进位链**。

实际上，完全采用并行进位链是不现实的。因为式(8.2.10)再继续推导下去，逻辑表达式越来越长，从而使电路结构庞杂。因此，通常采用分组跳跃进位的方式。例如，可把十六位并行加法器分为四组，每组四位。在组内每位之间实现并行快速进位，组间也可以实行并行快速进位。下面我们说明组间快速进位的情况。

所谓**组间快速进位**，就是低位组向高位组有进位时，该进位信号由低位组内的最低位进位信号直接形成，而不需要经过组内四位间的逐位传递过程。

例如，十六位加法器分为四组，低位组向高位组的进位分别为：C_{15}、C_{11}、C_7 和 C_3。进位信号 C_3 直接由 C 形成；C_7 直接由 C_3 形成；以下类推。

$$C_3 = A_3B_3 + (A_3 + B_3)C_2$$

$$=A_3B_3+(A_3+B_3)A_2B_2+(A_3+B_3)(A_2+B_2)A_1B_1$$
$$+(A_3+B_3)(A_2+B_2)(A_1+B_1)A_0B_0$$
$$+(A_3+B_3)(A_2+B_2)(A_1+B_1)(A_0+B_0)C \qquad (8.2.11)$$

式(8.2.11)表明:

当 A_3、B_3 均为 1;

A_3、B_3 中有一个为 1,且 A_2、B_2 均为 1;

A_3、B_3 中有一个为 1,A_2、B_2 中有一个为 1,且 A_1、B_1 均为1;

A_3、B_3 中有一个为 1,A_2、B_2 中有一个为 1,A_1、B_1 中有一个为1,且 A_0、B_0 均为 1;

A_3、B_3 有一个为1,A_2、B_2有一个为1,A_1、B_1 有一个为 1,A_0、B_0 有一个为 1,且 C 为 1

时,C_3 都能直接产生,不需要通过 $C\to C_1\to C_2\to C_3$ 这样的传递过程,所以是快速进位。

变换式(8.2.11),得

$$C_3=A_3B_3$$
$$+(A_3+B_3)(A_2+B_2)A_2B_2$$
$$+(A_3+B_3)(A_2+B_2)A_1B_1$$
$$+(A_3+B_3)(A_2+B_2)(A_1+B_1)(A_0+B_0)A_0B_0$$
$$+(A_3+B_3)(A_2+B_2)(A_1+B_1)(A_0+B_0)C$$
$$=A_3B_3$$
$$+\overline{\bar{A}_3\bar{B}_3+\bar{A}_2\bar{B}_2}\cdot A_2B_2$$
$$+\overline{\bar{A}_3\bar{B}_3+\bar{A}_2\bar{B}_2}\cdot A_1B_1$$
$$+\overline{\bar{A}_3\bar{B}_3+\bar{A}_2\bar{B}_2}\cdot\overline{\bar{A}_1\bar{B}_1+\bar{A}_0\bar{B}_0}\cdot A_0B_0$$
$$+\overline{\bar{A}_3\bar{B}_3+\bar{A}_2\bar{B}_2}\cdot\overline{\bar{A}_1\bar{B}_1+\bar{A}_0\bar{B}_0}\cdot C \qquad (8.2.12)$$

若令

$$E_1=\overline{\bar{A}_3\bar{B}_3+\bar{A}_2\bar{B}_2}$$
$$E_0=\overline{\bar{A}_1\bar{B}_1+\bar{A}_0\bar{B}_0}$$

则 $$C_3=A_3B_3+E_1A_2B_2+E_1A_1B_1+E_1E_0A_0B_0+E_1E_0C \qquad (8.2.13)$$

对于 C_7、C_{11} 和 C_{15} 也可以导出与式(8.2.13)类似的逻辑表达式。如:

$$C_7=A_7B_7+E_3A_6B_6+E_3A_5B_5+E_3E_2A_4B_4+E_3E_2C_3 \quad (8.2.14)$$

其中

$$E_3=\overline{\bar{A}_7\bar{B}_7+\bar{A}_6\bar{B}_6}$$

$$E_2=\overline{\bar{A}_5\bar{B}_5+\bar{A}_4\bar{B}_4}$$

根据表达式(8.2.13)和(8.2.14)画出两组之间的进位 C_7 和 C_3 的逻辑图,如图 8.2.4 所示。

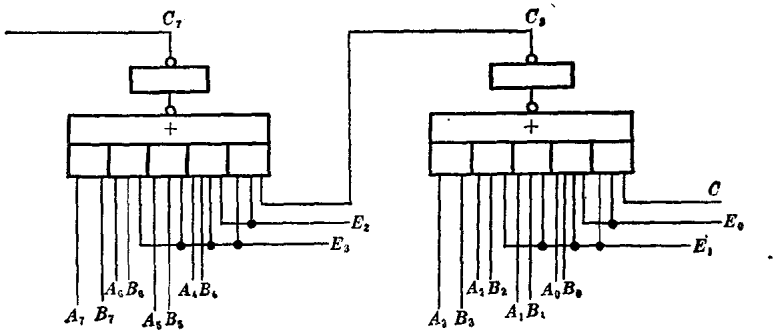


图 8.2.4 组间快速进位逻辑图

组成快速进位的方式和线路是很多的,我们就不一一叙述了。

§8.3 运算器中的数据传送

一、寄存器之间的数据传送

运算器中寄存器的作用是暂存被加工的数据及运算结果。

当指令系统中没有乘除法指令时,则寄存器至少要有两个,一个用来存放被加数,另一个用来存放加数。运算结果一般存放在被加数寄存器中,这样可以节省一个寄存器。

在有乘除法指令时,运算器中至少要有三个寄存器。在进行乘法时,分别用来存放被乘数、乘数和部分积;在进行除法时,则分别用来存放被除数、除数和商。

随着计算机技术的发展，目前小型机和微型机多采用多通用寄存器结构。多通用寄存器结构为运算和数据暂存提供了很多方便。通用寄存器的数目有 4 个、8 个、16 个或更多个。

不管运算器中寄存器多少，在运算过程中都有寄存器间数据传送的问题。

传送方法一般有三种：

1. 直接传送

两寄存器间的代码直接进行传送，依靠控制门的接收命令决定代码如何传送，如图 8.3.1 所示。

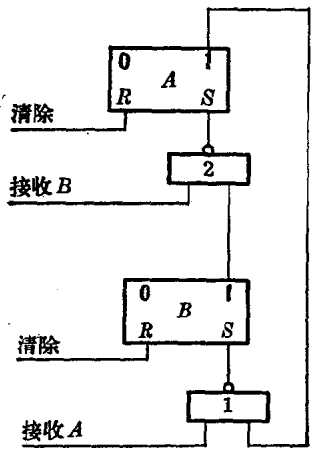


图 8.3.1 直接传送逻辑图

直接传送的特点是寄存器的每一位触发器都需要专用的控制门，传送速度快，但是，当寄存器的数目增多时，传送控制门的数量将急剧增加，因此这种方式很少采用。

2. 通过加法器 Q 实现数据传送

寄存器中的数据一般都要在加法器中进行运算，运算结果往往也要传送到各寄存器。因此在运算器中，加法器便可以作为数据传送的通道，利用这些通道也可以实现寄存器之间的数据传送。

图 8.3.2 给出通过加法器实现数据传送的逻辑图(一位)。

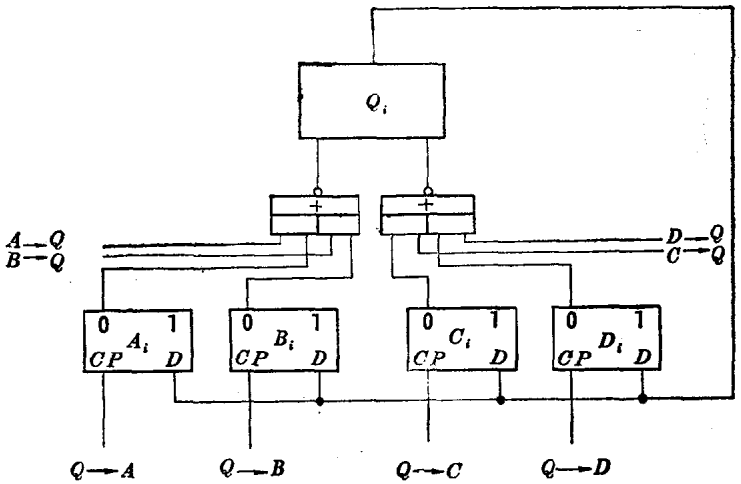


图 8.3.2 通过加法器 Q 实现数据传送

例如，要把 A 寄存器中的数据传送到 C 寄存器，只要发出信号 $A \rightarrow Q, Q \rightarrow C$ 即可。

同理，若把 D 寄存器中的数据传送到 A 寄存器，则发出信号 $D \rightarrow Q$ 和 $Q \rightarrow A$ 便可实现。

目前采用这种方式传送数据的机器较多。

3. 利用专用的代码总线实现寄存器之间的数据传送。

专用代码总线实际上就是用**与或非**门组成的多路开关，它可以用来实现寄存器之间的数据传送。图 8.3.3 给出了四个寄存器之间实现数据传送的逻辑图(一位)。

图 8.3.3 上面是一个四**与或非**门，下面是四个寄存器。利用这个线路可以实现各寄存器之间的数据传送。

例如，要把 A 寄存器中的内容传送到 C 寄存器，只要发出信号 $A \rightarrow ZX$ 和 $ZX \rightarrow C$ 即可实现。

利用这种方式传送代码，在同一时间里，只能由一个寄存器向