Z 80

微型电脑基础知识

株洲电子研究所
香港金山公司

前　言

微型电脑近年来在我国得到了迅速的发展和推广应用，越来越受到各个领域的重视，成为新技术领域里的一个热门。

为了配合我们研制的CMC—80微型电脑的研制、生产和推广使用，便于广大的初学者熟悉微型电脑、Z80系列微处理器器件和CMC—80微型电脑，我们编印了这本教材，作为《CMC—80微型电脑使用手册》的补充，提供给用户和感兴趣的读者作为参考。

本书第一章概要介绍了数字电路的有关知识。第二章介绍了两种为教学而设计的模型计算机，它们是近代计算机的简化模型，但可使读者大致了解微型电脑总体结构的全貌以及其中各主要部件的工作原理。第三章至第七章简要介绍了Z80系列中的四种常用器件（CPU，PIO，CTC和SIO）的主要特性和操作说明。第八章介绍了A／D转换器0809的特性和使用说明。

本书前六章选自北京工业大学出版的《微处理机实用教材——基础知识和Z80器件的使用》，后二章为编译材料。由于水平有限，错误在所不免，恳请读者指正。

株洲市电子研究所

香港金山公司

1982年5月

再版附言

本版对第七章做了较大的增补和充实，其它各章仅订正了初版中的错误。

编　者

1983年3月

目　录

第一章 基础知识

1.1 数制及不同数制间的换算

一、十进制数

在日常生活中，人们最熟悉的是十进制数。它有十个不同的数字：0，1，2，3，4，5，6，7，8，9。在表示数时，这些处于不同的位置（或数位）的数字代表的意义是不同的。例如1001，表示一千零一。我们称这是一个四位（十进制）数。

一般地讲，任何十进制数

D3 D2 D1 D0

都可以写成基数十的各次幂的和式，即

$$D3\ D2\ D1\ D0 = D3\times10^{3}+D2\times10^{2}+D1\times10^{1}+D0\times10^{0}$$

可见同样一个数字，放在最高位与最低位的含义是不同的，D3 有表示 10^3 的权，D0 有表示 10^0 的权。上式我们又称为按权展开式。

二、二进制数

在电子计算机中通常并不采用十进制数，而是采用二进制数。因为电子计算机中使用高电平和低电平来表示两个不同的数码：0，1。一个二进制数的按权展开式如下：

$$B3\ B2\ B1\ B0 = B3\times2^{3}+B2\times2^{2}+B1\times2^{1}+B0\times2^{0}$$

在这种数制中，1001不表示一千零一，而是表示九。

三、十六进制数

这种数制中有十六个不同的数字：0，1，2，3，4，5，6，7，8，9，A（相当于十进数中的10），B（11），C（12），D（13），E（14），F（15）。

它的按权展开式如下：

$$H3\ H2\ H1\ H0 = H3\times16^{3}+H2\times16^{2}+H1\times16^{1}+H0\times16^{0}$$

在微型计算机中经常使用这种十六进制数制，其理由如下：

1.它与二进制数之间的转换比较方便。例如二进制数

1001 1100B *

$$=(1\times2^{3}+1\times2^{0})\times16^{1}+(1\times2^{3}+1\times2^{2})\times16^{0}$$

$$=9CH$$

---

* B（Binary）表示二进制数。同样D（Decimal）和H（Hexadecimal）分别表示十进制数和十六进制数。

即每四个二进制数对应于一个十六进制数。微型机中的二进制数通常是8位或8位的整倍数（16，24，32位），则相应的十六进制数为2，4，6，8位。

2.使用二进制，书写太长，不易记忆，且念起来不易懂。而使用十六进制可以弥补上述缺点。

四、八进制数

它的特性与16进制数相似，并且在近代微型机中较少使用，故不予介绍。

五、二—十进制数（BCD码）

在这种数制中，用二进制数来表示十进制数字。例如

$97D=10010111BCD$

但须注意，这种数制与二进制不同，它们的数位的权不同。

$$10010111=(1\times2^3+1\times2^0)\times10^1+(1\times2^2+1\times2^1+1\times2^0)\times10^0$$

这种数制的优点是既照顾了人们使用十进制数的习惯，又考虑到计算机的特点。缺点是不便于运算。由于微型计算机中的运算比较简单，因此经常采用这种数制，特别是微型机化仪器的输入和输出。

六、不同数制间的换算

1.二翻十。即把二进制数换算成十进制数。这种方法比较简单，利用二进制数的按权展开式，将二进制数按权相加，就得到等值的十进制数。

例：　$1101B=1\times2^3+1\times2^2+0\times2^1+1\times2^0=13D$

2.十翻二。即把十进制数换算成二进制数。现举例说明其方法：

例：　求13D的二进制数

$13\div2=6$	余数1	（最低位）
$6\div2=3$	余数0	
$3\div2=1$	余数1	
$1\div2=0$	余数1	（最高位）

结果：1101 B

在微型计算机中常有一些实用的子程序用于进行这类运算。

1.2 逻辑电路

通常一个逻辑电路有一或两个以上的输入，一个输出。不同性质的逻辑电路，输出与输入之间有不同的关系，这种关系称为逻辑函数。输入或输出的状态只有两种：高电平和低电平。通常，我们用逻辑1（即有效）来表示高电平，逻辑0（即无效）表示低电平，这称为正逻辑，大多数微型机均使用正逻辑。反之，用逻辑1表示低电平，逻辑0表示高电平，称为负逻辑，少数微型机如Intel 4004／4040使用负逻辑。本书中使用正逻辑。

下面介绍经常用的逻辑电路。

一、与门（AND gate）

1.符号：如图1.1（a）所示。

2.含义：只有当所有输入（A和B）都为1状态时，输出y才为1。

3.逻辑式：y＝A×B或y＝AB。

4.真值表（逻辑函数的另一种表示法）：如表1.1所示。

二、或门（OR gate）

1.符号：如图1.1（b）所示。

2.含义：只要输入中有一个为逻辑1，输出即为逻辑1。

3.逻辑式：y＝A＋B

4.真值表：如表1.1所示。

表 1.1 常用逻辑电路真值表

输入		输出				
A	B	与门	或门	非门（$Y=\overline{A}$）	异门	同门
0	0	0	0	1	0	1
0	1	0	1	1	1	0
1	0	0	1	0	1	0
1	1	1	1	0	0	1

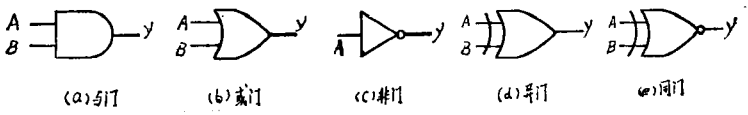


图 1.1 常用逻辑电路

三、非门（反相器—inverter）

1.符号：如图1.1（c）所示。

2.含义：输出与输入状态相反。

3.逻辑式：$Y=\overline{A}$

4.真值表：如表1.1所示。

四、异门（eXclusive—OR gate）

1.符号：如图1.1（d）所示。

2.含义：输入信号中有奇数个1，输出为逻辑1。反之，为逻辑0。

3.逻辑式：y＝A⊕B

4.真值表：如表1.1所示。

五、同门（eXclusive—NOR gate）

1.符号：如图1.1（e）所示。

2.含义：输入信号中有偶数个1，输出为逻辑1。反之，为逻辑0

3.逻辑式：$y=A\oplus B$或$y=\overline{A\oplus B}$

4.真值表：如表1.1所示。

六、摩根定理

摩根定理是逻辑运算中最常用的定理。即

$$\overline{A\,B}=\overline{A}+\overline{B}$$

$$\overline{A+B}=\overline{A}\cdot\overline{B}$$

读者可以用真值表来证明这个定理。此定理表明，可以把逻辑“与”运算转换成逻辑“或”运算。反之亦然。

1.3 运算电路 (Arithmetic circuit)

一、一位（二进制）数加法与半加法器（Half adder）

两个一位的二进制数A与B相加，有四种情况：

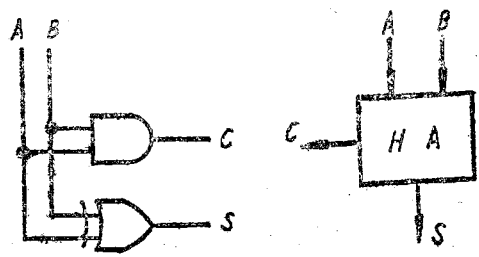
	(1)	(2)	(3)	(4)
A	0	0	1	1
+B	+0	+1	+0	+1
C S	0 0	0 1	0 1	1 0
	↓ ↓	↓ ↓	↓ ↓	↓ ↓
	C S	C S	C S	C S

其结果如上，其中S是本位和，C是（对下一位的）进位。由上节可知：

$$S=A\oplus B$$

$$C=A\times B$$

由此可得逻辑图，如图1.2（a）所示。其方框图如图1.2（b）所示，通常叫做半加法器。



(a) 电路　　(b) 框图

图 1.2 半加法器

二、多位（二进制）数加法与全加法器（Full adder）

先介绍多位数相加的过程。设有两个4位的二进制数A和B，A＝1011，B＝1001，试求A＋B＝？

下面按照小学生的习惯进行演算，即逐位进行运算。先从最低位开始，$A_0+B_0=C_1S_1$，S_0是本位和，C_1是对下一位的进位。其次$C_1+A_1+B_1=C_2S_1$，依此类推。

Ci	1 ←	0 ←	1 ←	1 ←	1
Ai		1	0	1	1
+Bi		1	0	0	
	1	0	1	1	
Si	↑ 0	↑ 1	↑ 0	↑ 0	↑ 0
	C_4 ↑	C_3 ↑	C_2 ↑	C_1 ↑	
	S_3	S_2	S_1	S_0	

可见，多位数相加与一位数相加的

差别在于：前者为三个一位数相加，后者只有两个一位数相加。三个一位数相加有八种情况，其结果如表1.2所示。其逻辑式如下：

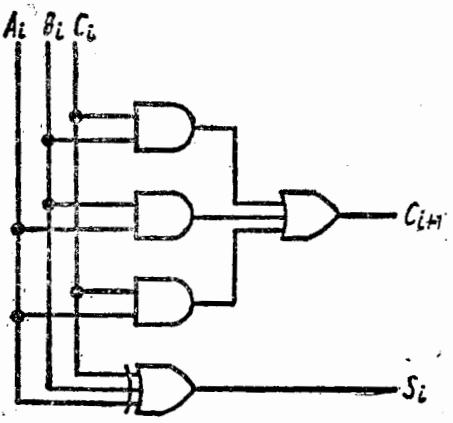
$$S_i = A_i \oplus B_i \oplus C_i$$

$$C_{i+1} = A_i B_i + B_i C_i + C_i A_i$$

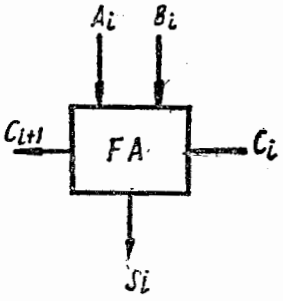
表 1.2 全加法器真值表

输入			输出	
A_i	B_i	C_i	C_{i+1}	S_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

图1.3（a）示出其逻辑图，图1.3（b）示出其方框图，通常叫做全加法器。由此可以得到两个4位数相加的二进制加法器，如图1.4（a）所示。将此图简化可得图1.4（b）所示的方框图。



(a) 电路

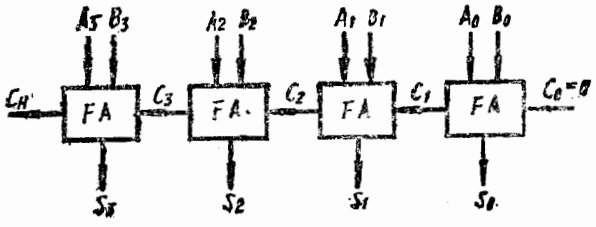


(b) 框图

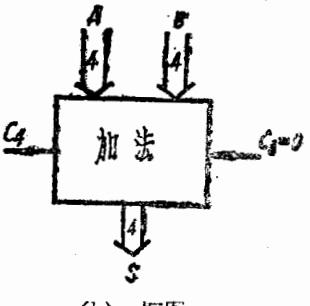
图1.3 全加法器

三、二进制数减法与补码运算

按照小学生的演算法则进行二进制数减法并不困难，但要用电子路线来实现减法要比加法麻烦得多。因此人们提出一种新的表示数的形式——补码。



(a) 电路



(b) 框图

图 1.4 二进制加法器

如前所述，四个二进制数字可以表示16个十进制数—0到15。现在重新规定其含义，用来表示另外16个数，如表1.3所示。0到7八个数的表示法与以前相同；所不同的是它能表

表1.3 数的补码表示

数	补 码
-8	1000
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

示负数。以－1为例，表中用1111来表示。数1111加1得10000，最高（第五）位1将被丢失而不起作用。所以，数1111补上一个1得0000，即数1111对0000来说缺1，因而可将1111看作－1，其他类推。在补码表示法中，最高位可以看作符号位，0表示正，1表示负。由此，利用补码进行减法运算可以将减法转换为加法，例如

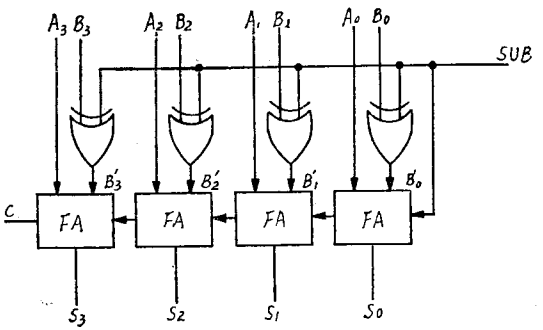
5－3＝5＋(-3)＝？其运算过程如下：

$$\begin{array}{r} 5 \\ +(-3) \\ \hline 2 \end{array} \qquad \begin{array}{r} 0101 \\ +1101 \\ \hline 10010 \end{array}$$

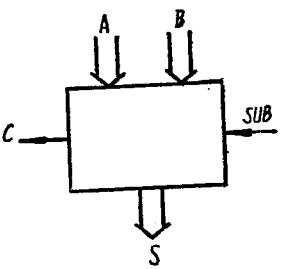
↑ 丢失

但是如何将3（0011）转换成-3（1101）？简单的说，就是将0011进行“求反加1”运算：

求反　　0 0 1 1
　　↓ ↓ ↓ ↓
　　1 1 0 0



(a) 电路



(b) 框图

图1.5 加法/减法器

加1

$$\begin{array}{r} 1100 \\ +\quad\ 1 \\ \hline 1101 \end{array}$$

在电路上又如何实现？图1.5（a）示出了利用补码运算的加法／减法器。当进行加法运算时，令SUB＝0，则$C_0=0$，且$B'_i = B_i$（i=0—3），它与图1.4的二进制加法器相同。当进行减法运算时，令SUB＝1，则$C_0=1$，且$B'_i=\overline{B_i}$（i＝0—3）。即通过异门将减数B求反，利用$C_0=1$，得到加1操作。图1.5（b）示出其方框图。

1.3 触发器与寄存器(Flip—flop and Register)

一、触发器

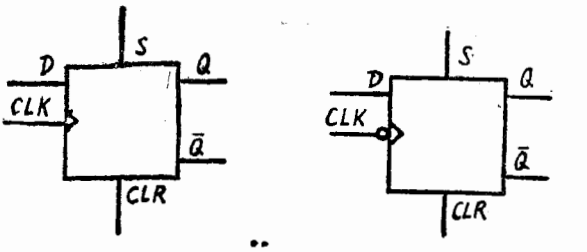
触发器是一种常用的数字电路。它可以作为一种无触点开关，也可以存放一个（二进

制）位（bit）的信息。常用的触发器有：

1.D触发器

1）符号：如图1.6（a）所示。图中输入信号有D—数据，和CLK—时钟，CLR—复位（清零）信号，S—置位（置1）信号；输出信号有Q和$\overline{Q}$。

2）操作：如表1.4所示。当CLK正跳变时，Q=D；当CLK不是正跳变时，Q保持原状。CLR=1时，Q=0；当S=1时，Q=1。



(a)CLK正跳变有效　　(b)CLK负跳变有效

图1.6　D触发器

表 1.4

输入		输出
CLK	D	Q
↑	1	1
↑	0	0
其他	×	原状

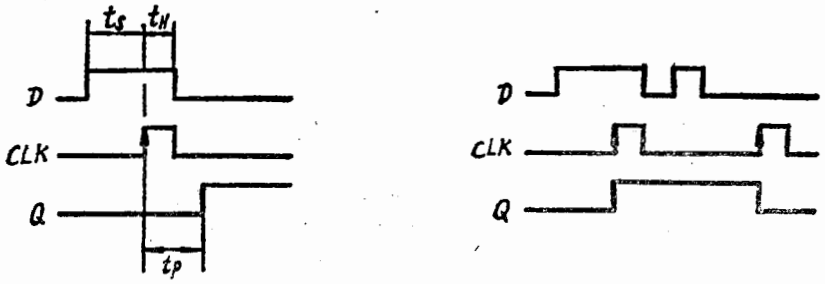
图1.7示出D触发器操作的时间图。图1.7（a）表明，数据线D的建立必须领先于CLK正跳变，这段时间称为建立（Setup）时间t_S。数据线D的撤除必须滞后于CLK正跳变，这段时间称为保持（Hold）时间t_H。如果不满足上述条件，D触发器就不能正确动作。输出信号Q的变化滞后于CLK正跳变，这段时间称为传播延迟（Propagation delay）t_P 在以后的叙述中，将忽略这些特性。图1.7（b）示出了不考虑t_P的操作时间图。

图1.6（b）示出另一种D触发器，与前者的差别仅在于CLK端上多一个小园圈。在数字电路中，小园圈表示反相，因而这种D触发器的CLK信号在负跳变时为有效。同理，若CLK，S端上也加上小园圈，则这些信号在0电平为有效。

2.JK触发器

1）符号：如图1.8所示。J和K为控制输入端。

2）操作：表1.5示出了当CLK发生正跳变时输出与输入的关系



(a)考虑t_P　　(b)不考虑t_P

图 1.7　D触发器的操作时间图

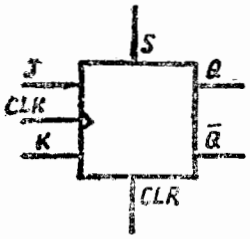


图1.8 JK触发器

表 1.5

输入		输出
J	K	Q
0	0	不变
0	1	0
1	0	1
1	1	翻转

二、寄存器

寄存器由若干个（通常是4，8，16个等）触发器构成。它可以存放一个4位、8位、或16位的字，此外还能执行加1、减1、移位和其他的操作。

1.计数器

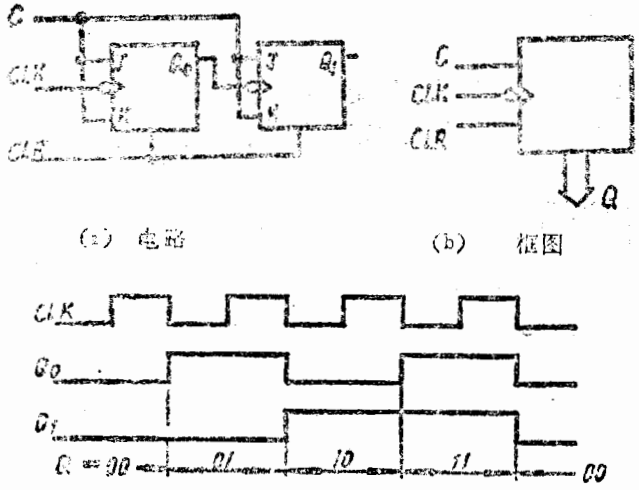
1）电路图及方框图：如图1.9（a）（b）所示。

2）操作：当CLR＝1时，所有触发器被复位，即$Q_1=0$、$Q_0=0$。当C＝0时，所有J、K端都为0，施加时钟脉冲CLK并不能改变Q的状态，即Q保持原状。当C＝1时，所有J、K端都为1，因而每一个时钟脉冲都使计数器加1，其时间图如图1.9(C)所示。可知C是控制输入端，控制计数器是否对CLK进行计数。

2.缓冲与移位寄存器

1）电路图：如图1.10所示。

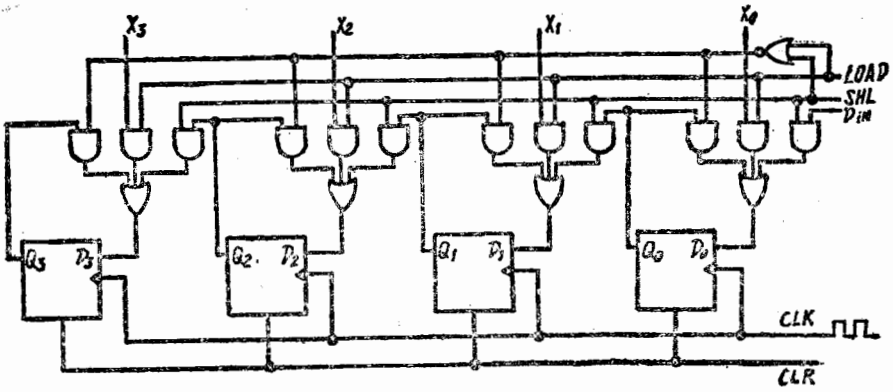
2）操作：当CLR＝1时，所有触发器被复位，Q＝0。当LOAD＝1时，在CLK正跳变瞬间，将X装入Q，即Q＝X。当SHL＝1时，在CLK正跳变瞬间，Q向左移位，即$D_{in}\rightarrow Q_0$，$Q_0\rightarrow Q_1$，$Q_1\rightarrow Q_2$，$Q_2\rightarrow Q_3$，Q_3丢失。当LOAD＝0和SHL＝0时，在CLK正跳变瞬间，Q保持不变。



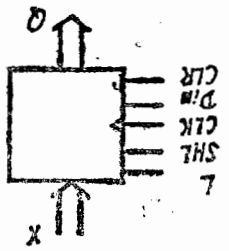
(a) 电路　　(b) 框图

(c) 时间图

图 1.9 計数器



(a) 电路



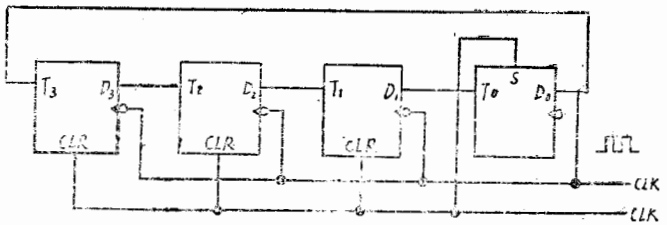
(b) 框图

图1.10 缓冲与移位寄存器

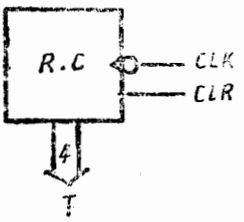
3.环形计数器

1）电路图：如图1.11（a）（b）所示。

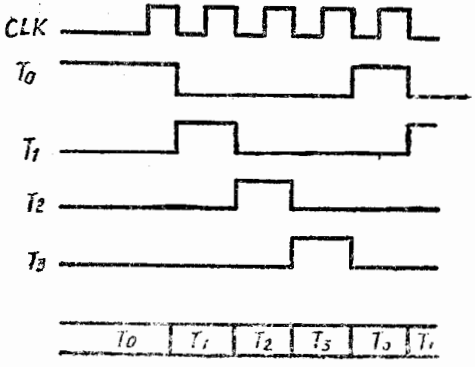
2）操作：当CLR＝1时，$T_3=0$，$T_2=0$，$T_1=0$，$T_0=1$，即 T＝0001。当送入时钟脉冲CLK时，每出现一次正跳变，T依次为0010→0100→1000→0001→……。



(a) 电路



(b) 框图



(c) 时间图

图1.11 环形计数器

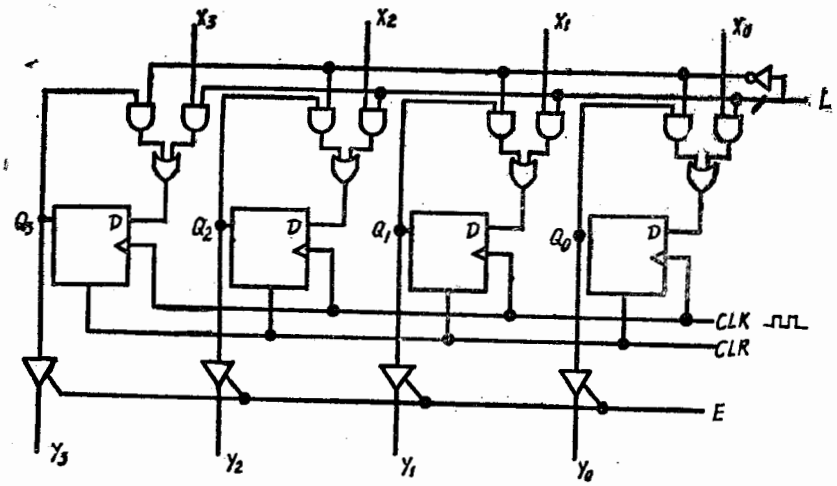
三、三态输出寄存器

由于一般的数字器件只有两个状态1和0，所以每条信号线只能由一个器件来驱动，从而使信息传输线的数目大为增多。为了减少信息传输线的数目，简化控制系统，将属于不同来源的信息或数据在一组统一的传输线上分时（Multiplexed）传送到不同的目的地，这组传输线（通常是8条或16条）称之为总线（Bus）。在某一时刻，只能有一个器件驱动总线，这个器件的输出可以呈1或0。其他器件不能呈此二状态，它们必须呈第三种状态——高阻抗，即浮动状态。也就是说，好像它们的输出被开关所断开，对总线状态不起作用。

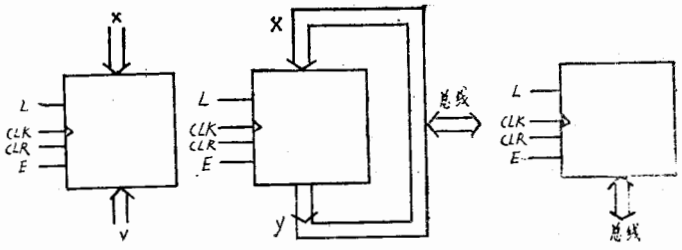
为了将普通器件的二态输出更改为三态输出，通常使用图1.12所示的三态开关。当E＝1时，Dout＝Din，此时Dout线由该器件来驱动，当E＝0时，Dout呈高阻抗状态，该器件对它不起作用，而是由其他器件驱动此线。

图 1.12　三态开关

图1.13示出具有三态输出的缓冲寄存器。当E＝1时，输出Y＝Q；当E＝0时，输出Y＝高阻抗，与各个触发器的状态Q无关。对于具有三态输出的缓冲寄存器，可以将其数据输出线和数据输入线合并在一起，以便挂到总线上去，如图1.13（C）所示。



(a)　电路



(b)　框图　　(c)具有数据总线的寄存器

图1.13　具有三态输出的缓冲寄存器

图1.14示出了四个寄存器挂在一条总线W上。如果控制信号中只有E_A、L_B等于1，其他均为0，则在CLK正跳变时，寄存器A的内容将装入寄存器B。因为只有$E_A = 1$，其他器件的E信号为0，总线的状态由A来决定。换句话说，A驱动总线。因为只有$L_B = 1$，总线上内容只装入B。通俗地说，E是“放出”控制信号，L是“装入”信号。

如果只有$E_B = 1$，$L_C = 1$，$L_D = 1$，其他控制信号均为0，则在CLK正跳变时，寄存器B的内容将装入寄存器C和D。

对于某一时刻，只能有一个器件的E信号有效，否则将有多个器件“争夺”总线而发生误操作。

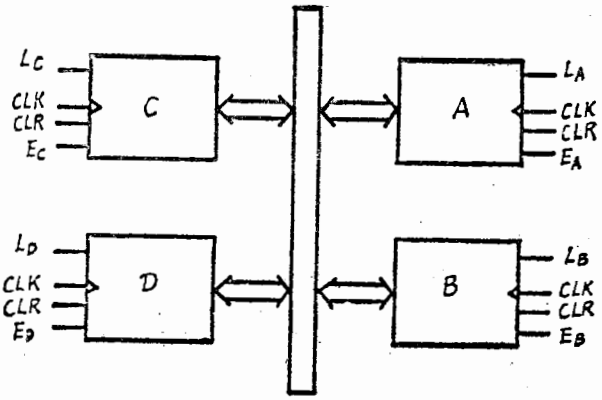


图 1.14 挂到总线上的寄存器组

1.5 存储器(Memory)

存储器是由若干个寄存器组成，每个寄存器存放一个二进制字（在微型计算机中，通常是8位称做一个字节）。存储器用来存放程序和数据。有多种介质可以用来制作存储器，如半导体、磁芯、磁盘、磁带等等。微型计算机的内存储器（简称内存）主要使用半导体存储器，本节内容仅限于这种半导体存储器。

存储器可以分为下列两大类：

1.只读存储器—ROM　它用来存放程序、表和常数。它的内容只能被微处理器读出，而不能被微处理器改变，也不会因断开电源而丢失。ROM通常又分为下列三种：

1）ROM　它的内容由芯片制造厂使用掩模编程而被永久性地固定下来。由于其工作可靠，在大量生产时价格低廉，在产品已被定型而大量生产时，常常使用ROM。

2）PROM（可编程序只读存储器）　它的内容由用户利用PROM写入器（或称编程器）而被固定下来，一旦被编定，就不能再被改变，只允许对未曾编程的位进行编程。在小批量生产时，常使用PROM。

3）EPROM　（可擦除的可编程序只读存储器）它的内容被用户编定后，可以用紫外线擦掉而再次被编程，虽然EPROM的可靠性不如前两者，但是由于它的灵活性，常被使用于产品的研制阶段。

近年来，又出现一些其它类型的ROM，如EAROM，EEPROM，此处不作介绍。

2.读写存储器—RAM　它的内容可以由微处理器写入和读出。RAM 可分为两种：

1）静态RAM　每位信息存放在一个触发器中，只要电源有电，其信息能一直被保持。

2）动态RAM　它利用门—基片电容上的电荷来存放信息。这种电荷将在几毫秒内耗散，因而每隔两毫秒需对动态RAM的内容刷新一次。

动态 RAM 的优点是器件密度高，价格低，待机功耗低。它的缺点是必须有刷新电路，每个动态RAM芯片的字长仅为一位，因而 8 位微型计算机（即使内存容量很小）要求最少使用八个芯片。

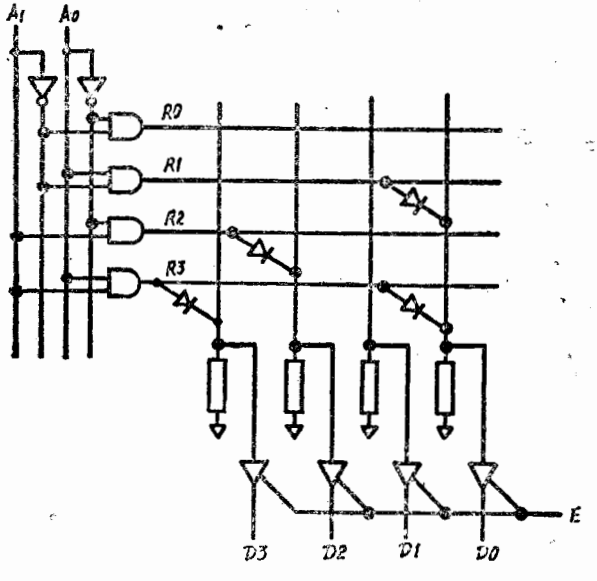
一、只读存储器ROM

只读存储器的原理图如图1.15（a）所示。每一行可以看作一个存储单元，用来存放一个二进制字。图中共有四行。即这一存储器有四个单元。可以存放四个二进制字。图中共有四列。表示每个字的字长为 4 位。

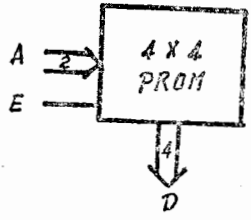
如果图中的地址线A 1 =0、A0=1、E= 1，则第二条线R1为高电平，从而通过二极管而使D0为高电平，D1、D2、D3相应的列中没有插入二极管，故为低电平，即：

$$D=D3D2D1D0=0001$$

其他线R0，R2，R3都为低电平，故对输出D没有影响。同理，A1，A0呈不同值时选中不同的水平线呈高电平，从而使数据线上呈现不同单元的内容。有二极管的位，相应于逻辑 1；反之，没有二极管的位，相应于逻辑 0 。



(a)　电路



(b)　框图

A	D
0 0	0 0 0 0
0 1	0 0 0 1
1 0	0 1 0 0
1 1	1 0 0 1

(c)　存储单元的内容表

图1.15　只读存储器的原理图

如果图中矩阵中二极管的有无可由用户来决定，而且可以再次被改变，则可以认为这是一种EPROM 的模型。

图1.15（b）示出它的方框图。数据输出线D的位数表示字长，通常是8位；地址输入线A的位数N决定存储字数，字数$=2^N$。

图1.15（c）用表格来表示ROM。在该图中，ROM的内容$R=(A1AO)^2$。可见，这样的ROM可以用作平方的表格。推而广之，ROM还可以存放各种函数（包括逻辑运算函数）以及其他的常数等。

图1.16示出了目前常用的2716型EPROM（2K×8）的引脚图。在24根引脚中，有11根地址线，8根数据输出线，以及Vcc和地线。这些引脚易于理解，不再进一步解释。现介绍其余三根引脚的作用，如表1.6所示。当$V_{PP}=+5$ V时，为读数方式；当$V_{PP}=+25$V时，为编程（即写入— programming）方式。

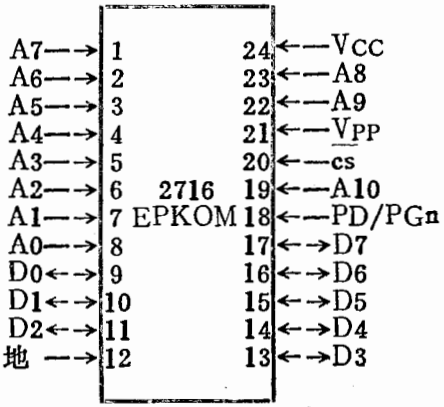


图1.16　2716型EPROM（2K×8）的示脚图

表1.6　工作方式选择

方式 \ 引脚	PD/PGM (18)	$\overline{CS}$ (20)	V_{PP} (21)	数据线状态
读	0	0	+5 V	DOUT
未选中	×	1	+5 V	高阻抗
待机	1	×	+5 V	高阻抗
编程	⎍	1	+25V	DIN
校验编程内容	0	0	+25V	DOUT
禁止编程	0	1	+25V	高阻抗

下面扼要介绍这几种工作方式：

1.读——此时可将其中某一个单元的内容读至数据线上。

2.未选中——因为$\overline{CS}=1$，数据输出线呈高阻抗，即该芯片不起任何作用。

3.待机（Power Down）——当PD/PGM＝1时，芯片处于待机方式。这种方式与不选中相似，唯一的差别在于：待机方式的功耗仅为最大运行功耗的四分之一。

编程——若要对EPROM某个单元进行写入，则应对PD/PGM引脚输入一个正脉冲，脉冲宽度50毫秒左右，对2K个单元的写入编程总共需要100秒左右。由于施加的脉冲电平与TTL兼容，不需要施加高电压脉冲，因而不使用专门装置，而用单板机这样的简单系统即能

编程。

5.校验编程内容——在编程完毕后，将其中的内容读出并进行比较，以决定编程的内容有否出错。此方式与读方式相似。

6.禁止编程——此时禁止将数据线上内容写入EPROM。

二、读写存储器RAM

图1.17示出了2114型静态RAM（1K×4）的引脚图。在18根引脚中，有10根地址线，4根数据（I/O）线，2根为Vcc和地线。这些引脚的作用易于理解，不再进一步解释。表1.7说明了$\overline{CS}$和$\overline{WE}$的作用。

表1.7 RAM的操作

$\overline{CS}$*	$\overline{WE}$	操作	数据线状态
1	×	保持原状	高阻抗
0	1	读	读出单元的内容
0	0	写	需要写入的内容

*或称为ME信号

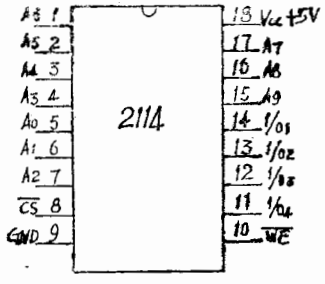


图1.17 2114型静态RAM(IK×4)的引脚图

第二章 模型计算机

在学习微型计算机和计算机的过程中，往往会有难于入门的感觉，其原因在于“太烦”。为此本章介绍一种尽可能简单的模型机，在第六节还对此机作必要的补充，以使初学者能在不到10个学时内，了解一个计算机的全貌，同时也掌握一些现代微型计算机中的重要概念，为进一步应用微型计算机技术创造条件。

2.1 模型计算机的结构

模型计算机的结构如图2.1所示。它是由算术及逻辑运算部件ALU、寄存器（存储器可看作多个寄存器）挂到总线上，并配以必要的控制电路而构成。这条总线既传送数据，也传送地址。如果寄存器的输出未接至总线，则为二态的；否则就是三态的。

下面简要介绍各个部件的作用。

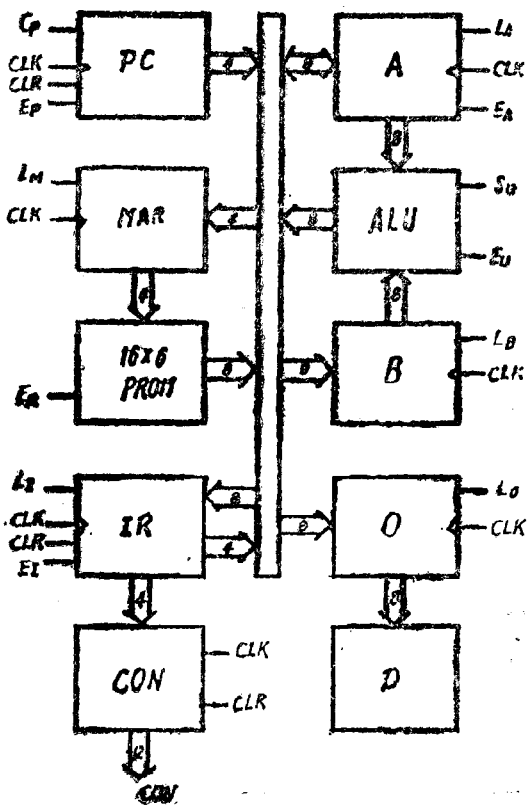


图2.1 模型计算机的结构

1.可编程序只读存储器PROM

PROM中存有16个8位字，即其容量为16×8位。它的内容和地址都可用16进制数来表示，如图2.2所示。在上面的单元中存放指令（R_0—R4），下面的单元中存放数据（R9—RB）。

当E_R为高电平时，16个PROM存储单元中有一个被读到总线上，这个单元的地址由存储器地址寄存器MAR来决定。

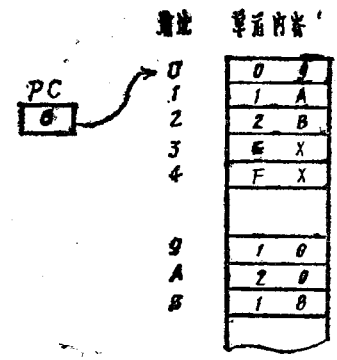


图 2.2 PROM 的内容表

2.存储器地址寄存器MAR 这是一个 4 位寄存器，存放被访问存储单元的地址。其内容可以来自程序计数器PC，也可以来自指令寄存器IR中的低 4 位（指令中的地址码）。它的二态输出送往PROM。

3.指令寄存器IR这是一个 8 位寄存器。计算机在执行程序时，将指令从PROM中取出，放入IR。8 位的指令码由两部分组成：高 4 位是操作码，表示该指令所应执行的操作，此信息送向CON单元，但不受 EI 信号的控制；低 4 位是地址码，表示该指令执行的操作中所需要的数据在哪个存储单元，当E_I L_M 有效时，此信息将送向MAR。

4.累加器A 这也是一个 8 位寄存器，用来存放运算的中间结果。它有两个输出：一是送往算术逻辑单元ALU的二态输出，其不受E_A 信号的控制；另一是送往总线的三态输出。

5.B寄存器它也是一个 8 位寄存器，接受总线上送来的加数或减数，并有一个送往 ALU 的二态输出，以供ALU 进行运算。

6.算术逻辑运算单元ALU这是一个加法器/减法器（见1.4节），S_U 电平的低或高决定它进行加或减

当$S_U = 0$时，ALU=A+B

当$S_U = 1$时，ALU=A−B

ALU是一种异步（无时钟的）电路，A和B对它的输入都是二态的，它随时对累加器 A和 B 寄存器进行运算。当$E_U = 1$时，它的内容方出现在总线上。

7.输出寄存器O这是一个 8 位寄存器。当计算机运算结束时，运算结果存放在累加器 A 中，需要将它传送到输出寄存器O，以便送往外部。

8.二进制显示器D它是连接到输出寄存器O的 8 个发光二极管（LED），用来显示输出寄存器O的内容。

9.程序计数器PC这是一个 4 位计数器。它被用来指示当前该执行指令的地址，如图2.2所示。计算机在复位后，它的内容为0000。每当一条指令从存储器中被取出后，PC内容自动增 1。因而，在每个取（指令）周期的起始时，PC中保存当前指令的地址。如图 2.2所示，PC=0000，表明该执行0000单元中的指令09。

10.控制单元CON通过有节奏地送出12个控制信号和时钟信号，它指挥计算机有节奏地执行指令所要求的操作。详细介绍见2.4节。

上述十个部件可进一步划分为下列几个部分。

1.运算器—包括ALU、A、B。它的任务是执行算术的和逻辑的运算。

2.控制器—包括PC、IR、CON。它按一定的顺序从存储器取出程序的一条指令，加以“解释”，发出操作命令。此外它还要安排一定的操作步骤，使计算机中各部分在得到操作命令后，按一定的节拍，有条不紊地操作。控制器是计算机工作自动化的保证。

上述两部分又可统称为CPU（中央处理机）。笼统地讲，微处理器就是CPU。

3.存储器—包括PROM，MAR。它用来存放指令和数据。与运算器和控制器直接联系的存储器称为内存。

上述三部分是计算机的主要组成部分。

4.输出设备—包括O和D。用来显示所得到的计算结果。

2.2 指令系统和程序设计简介

计算机与一般电子设备不同，它可以利用上节介绍的硬件，针对不同的问题进行相应的程序设计，以完成不同任务。然后将此程序的指令逐条送入存储器，并启动计算机运行。

用户在编制程序时，必须熟悉计算机的指令系统，即计算机能执行的基本操作。

一、指令系统

模型机的指令系统如表2.1所示，它共有五条指令。指令码由8位二进制数组成。

1.LDA×（0 0 0 0 ××××）

这是一条加载累加器A的指令。它将存储器中某单元的内容装入累加器A。指令码的高4位0 0 0 0表示操作的性质，称为操作码；指令码的低4位××××表示要加载A的数的地址，即操作数的地址，称为地址码；表2.1中所示的LDA9（00001001）指令表示将存储器中9单元的内容R9装入累加器A。

用二进制数（如00001001）表示指令能够被计算机“理解”，所以称为“机器语言”。在书写过程中，使用二进制数既烦，又易出错，故常使用16进制数（如09H）。但是，利用机器语言编写的程序很不直观，如09H这一数据，很难看出它是指令还是一个数。即使知道它是指令的话，也很难知道它是什么指令。因此人们就想办法，利用一些意义明确的符号来表示指令。例如09H指令，可以用“LDA9”来表示。由LDA可知，这是一条加载累加器A指令（LoaD Accumulator）这种符号（LDA）含义明显，便于记忆，故称之为助记符（MNEmonic）。在此基础上发展出一种具有一定语法规则的符号语言，称为汇编语言。遗憾的是，对于同样的指令，不同的计算机有不同的助记符。

表 2.1 模型计算机的指令系统

汇编语言		机器语言				操作及说明
		二进制		十六进制		
助记符	操作数	操作码	地址码	操作码	地址码	
LLA	9	0 0 0 0	1 0 0 1	0	9	A←R9
ADD	A	0 0 0 1	1 0 1 0	1	A	A←A+RA
SUB	B	0 0 1 0	1 0 1 1	2	B	A←A−RB
OUT		1 1 1 0	××××	E	×	0←A
HLT		1 1 1 1	××××	F	×	停止

2.ADD 0 0 0 0 ××××

它将累加器A的内容加上存储器中某指定单元的内容，其结果仍存放在累加器A。同样，指令码的低4位表示操作数的地址。

3.SUB 0 0 0 0 ××××

它将累加器A的内容减去存储器中某指定单元的内容，其结果仍存放在累加器A中。同样，指令码的低4位表示操作数的地址。

以上三条指令统称为访问存储器指令，因为这些指令的操作都与存储器中某指定单元内

的操作数有关。

4.OUT　1 1 1 0 × × × ×

它将累加器A的内容送往输出寄存器O。指令码的低4位可为任意值，指令的操作与此值无关，即不涉及存储器中存放的操作数。

5.HLT　1 1 1 1 × × × ×

它停止计算机的运行。同样。指令码的低4位可为任意值，指令的操作与此值无关，即不涉及存储器中存放的操作数。

二、程序设计简介

程序是一个人为编定的指令序列，用来解决用户提出的某个问题。编写程序的过程称为程序设计。下面我们举例说明，如何编写程序，使计算机来计算16D＋32D－24D＝？

首先，将这些需要运算的数据安排在存储器的数据区，通常安排在高地址单元中，如表2.2所示。本例中这三个操作数存放在R9，RA，RB。

由于计算机往往从0单元开始并顺序执行指令，因此第一条指令安排在0单元中。其它指令顺序放在R1，R2，R3，R4中，如表2.2所示。

0单元：LDA　9　将R9单元的内容16D装入累加器A，执行完后，A＝16D。

1单元：ADD　A　将RA单元的内容32D加到累加器A。结果存入累加器A。A＝16D＋32D＝48D。

2单元：SUBB　将累加器A中的内容减去RB单元的内容24D，结果存入A。A＝48D－24D＝24D

3单元：OUT　将累加器A中的内容送入输出寄存器O。O＝A＝24D。

4单元：HLT　使计算机停止运行。在本模型机中，将停止时钟脉冲的发出。而在Z80微型机中，则是不停地进行空操作。

操作结果如表2.2所示。

在存储器中，存放指令区必须与存放数据区分开，以免顺序执行指令时，误将数据当作指令来执行，从而产生荒谬的结果。

大多数微型计算机都能将汇编语言翻译成机器语言，这种过程称为汇编过程（Assembly），使用的工具称为汇编程序或汇编器（Assembler）。

表2.2　程　序　设　计　举　例

地　址	机　器　代　码	汇　编　语　言	操　作　结　果
0	0 9	LDA　9	A＝10H
1	1 A	ADD　A	A＝30H
2	2 B	SUB　B	A＝18H
3	E×	OUT	O＝18H
4	F×	HLT	
9	10		
A	20		
B	18		

机器语言和汇编语言都与机器本身有关，是一种面向机器的低级语言。使用这种语言编制程序不够方便，且不能在机器间交流程序。因而人们又不断设计出面向问题／或面向过程的高级语言（本书不作介绍）。

2.3 指令的执行过程

一条指令的操作不是一下子就完成的，而是如同人们做操那样，在好几个节拍内完成的。每一个节拍完成一个微操作，这些微操作的集合即是指令所要求的操作。本机中每条指令需要六个节拍来完成，正如人们做操需要八个节拍一样。如表2.3所示，前三拍为取周期，即将指令从存储器中取出。T0拍将PC的内容送MAR，因为当前要执行的指令的地址放在PC中。T1拍将MAR指定的存储器单元的内容（指令码）取出，放在IR中。T2拍使PC内容增1，为执行下一条指令作好准备。取周期的微操作与要执行的指令的种类无关，五条指令在T0—T2节拍内执行的微操作相同。后三拍为执行周期，其微操作因指令不同而各异，有些指令只需一个或两个节拍即可完成其要求的操作。例如ADDA指令，T3拍将IR中低4位（即地址码）送MAR。T4拍将存储器中的加数（其地址由MAR规定）送B寄存器，此时ALU中的内容为累加器A和B寄存器相加的和。T5拍将ALU中的和送累加器A。其他指令的具体过程见表2.3，不再赘述。

不同的指令在不同的节拍内要执行不同的微操作。为此，在此节拍内除了需要有CLK正跳变外，尚需有相应的控制信号，如表2.3所示。

2.4 控制单元CON的设计

计算机能有条不紊地工作，全靠CON的指挥。

计算机对CON的要求如下：

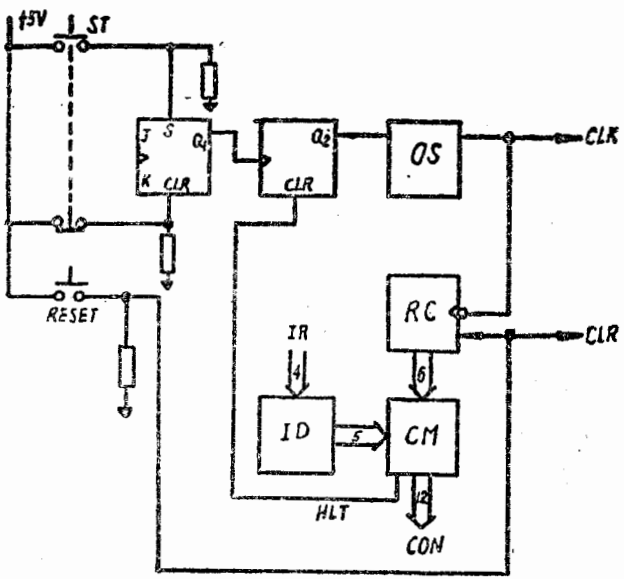


图2.3 控制单元CON的电路

表 2.3

指 令 的 执 行 过 程

指令 \ 节拍	指令周期					
	取周期			执行周期		
	T0	T1	T2	T3	T4	T5
LDA 9 0000 1001	MAR←PC LM, Ep	IR←R(MAR) L1, ER	PC←PC+1 Cp	MAR←IR LM, E1	A←R(MAR) LA, EB	——
ADD A 0001 1010	MAR←PC LM, Ep	IR←R(MAR) L1, ER	PC←PC+1 Cp	MAR←IR LM, E1	B←R(MAR) LB, ER	A←A+B LA, EU
SUB B 0010 1011	MAR←PC LM, Ep	IR←R(MAR) L1, ER	PC←PC+1 Cp	MAR←IR LM, E1	B←R(MAR) LB, ER	A←A−B LA, EU, SU
OUT 1110 ××××	MAR←PC LM, Ep	IR←R(MAR) L1, ER	PC←PC+1 Cp	O←A LO, EA	——	——
HLT 1111 ××××	MAR←PC LM, Ep	IR←R(MAR) L1, ER	PC←PC+1 Cp	—— HLT	——	——

1.产生清零脉冲CLR；

2.产生时钟脉冲CLK；

3.循环产生六个不同的节拍T0—T5；

4.在不同指令的不同节拍内产生所需的控制信号；

5.启动计算机，使之从0H单元开始执行程序。

按照上述要求设计的控制单元CON的电路如图2.3所示。其中各个部件的功能如下：

1.复位按钮RESET　按下此按钮，即向计算机其他部件发CLR信号。

2.启动按钮ST和时钟电路　当按下ST按钮时，Q1信号由低变高，从而使Q2信号由低变高。当Q2为高电平时，CLK上不停地发出时钟脉冲，当Q2为低电平时，CLK保持低电平，没有出现时钟脉冲。

3.环形计数器RC　其结构与图1.11相似，所不同的是它由六个触发器构成。当CLR信号有效时，T0呈高电平，以后每当出现CLK负跳变时，轮流使T1，T2，……为高电平。

4.指令译码器ID　如图2.4所示，其输入信号来自指令寄存器IR的高4位。当I7—I4=0001时，输出信号ADD应为高电平。其余输出信号均为低电平。余此类推。

5.控制矩阵CM　这个部件用来在不同指令的不同节拍内产生不同的控制信号（又称不同的控制字CON），如表2.3所示。或者说，它使输入信号I_i和T_i与输出信号CON之间成一定的逻辑函数关系CON=f（I_i，T_i），表2.3即为这种函数的真值表。

图2.5为控制矩阵的电路图。以L_M信号为例，在下列四种情况中均需出现L_M信号。

1）T0节拍：不管什么指令，只要T0线为高电平。则L_M^1为高电平。L_M为高电平。

2）LDA指令的T3节拍：此时LDA=1，T3=1由此$L_M^2=1$ $L_M=1$

3）ADD指令的T3节拍：同理，亦可得$L_M^3=1$，$L_M=1$。

4）SUB指令的T3节拍：此时亦可得$L_M^4=1$，$L_M=1$。

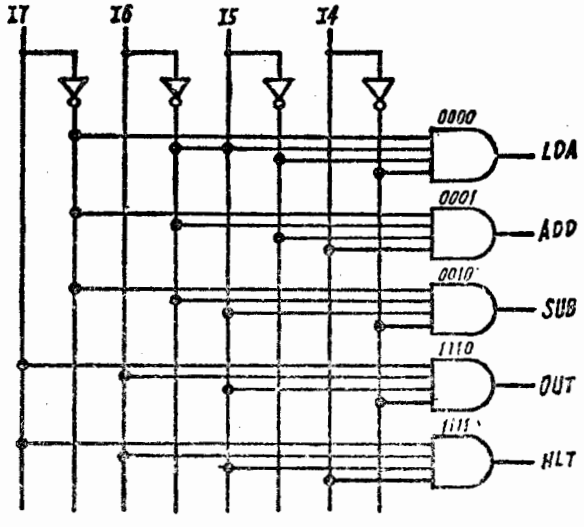


图 2.4　指令譯碼器 ID 的电路

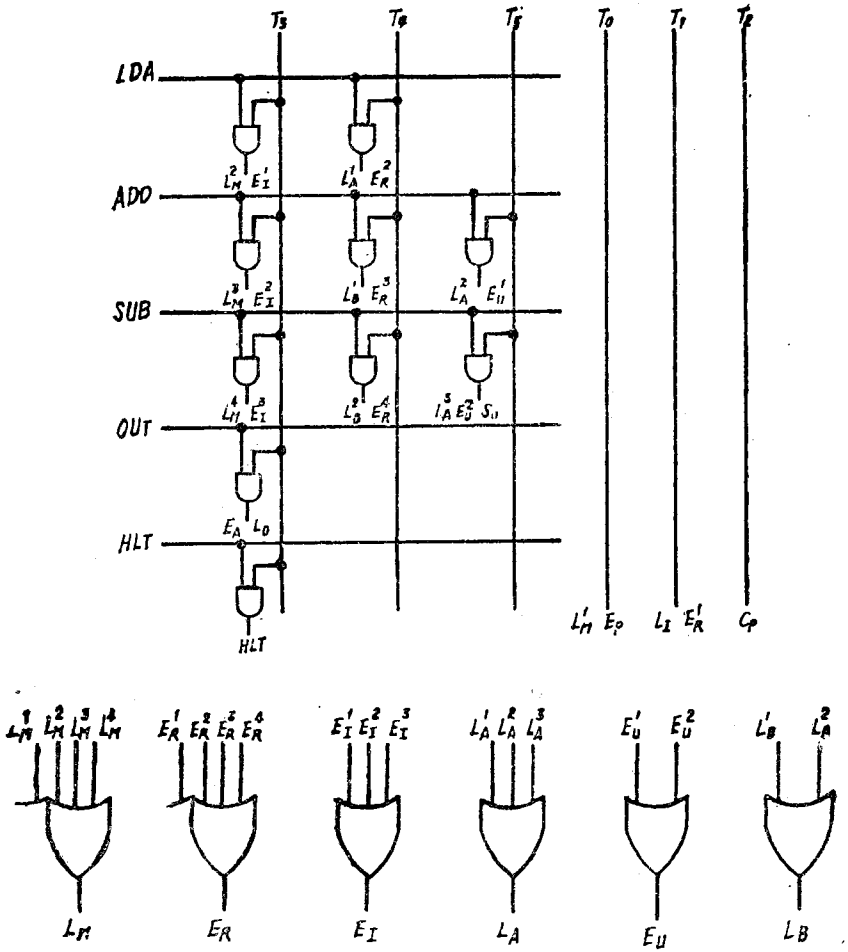


图 2.5 控制矩阵 CM 的电路

2.5 模型计算机的操作

PROM的内容如表2.2所示。模型计算机操作的时间图如图2.6所示。

首先压下RESET按钮，发出CLR信号，使环形计数器的T0成高电平，并使PC和IR为0000。接着压下ST启动按钮，CLK不停地发脉冲，环形计数器的T0—T5反复地轮流出现高电平，即轮流出现不同的节拍。值得注意的是，环形计数器电平的转换发生在CLK的负跳变瞬间。下面叙述此后发生的过程：

1.T0期间：L_M，E_P 为高电平，在CLK正跳变瞬间，MAR←PC，所以MAR=0000。

2.T1期间：L_I，E_R 为高电平，在CLK正跳变瞬间，IR←R(MAR)，所以IR=09H。

3.T2期间：C_P 为高电平，在CLK正跳变瞬时，PC←PC+1，所以PC=0001。

4.T3期间：L_M，E_I 为高电平（因为指令寄存器IR的高4位为0000，指令译码器的LDA线成高电平，从而使控制矩阵的L_M，E_I 为高电平），在CLK正跳变瞬间，MAR←IR，所以MAR=1001。

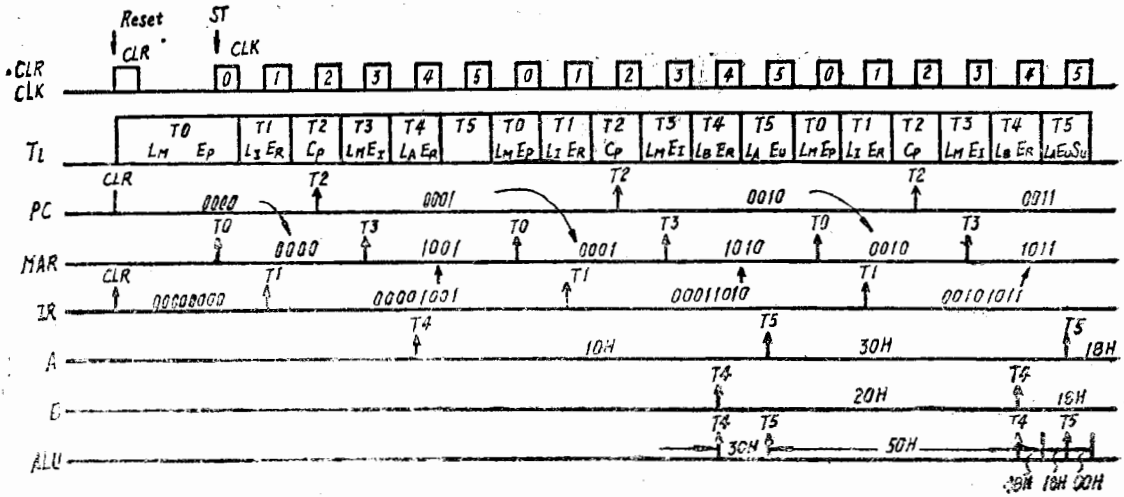


图2.6 模型计算机操作的时间图

5. T4期间：L_A E_R 为高电平，在CLK正跳变瞬间，A←R（MAR），所以A=R9=0001，0000=10H。

6. T5期间：没有发出控制信号，不进行任何操作。

此后，又不断出现T0—T5以执行后继单元的指令，其过程相似，不再赘述。

顺便指出，某些时刻ALU中的内容毫无意义，只要它不影响计算机的正确操作即可。

2.6 扩充模型机*

以上叙述的模型计算机十分简单，非常便于了解计算机工作的全貌及其主要特点。但是它有相当的局限性，有一些重要概念无法介绍。为此，设计了下述的扩充模型机。

一、结构与指令系统

如图2.7所示，其结构与前述的模型计算机相似，仅有如下的差别：

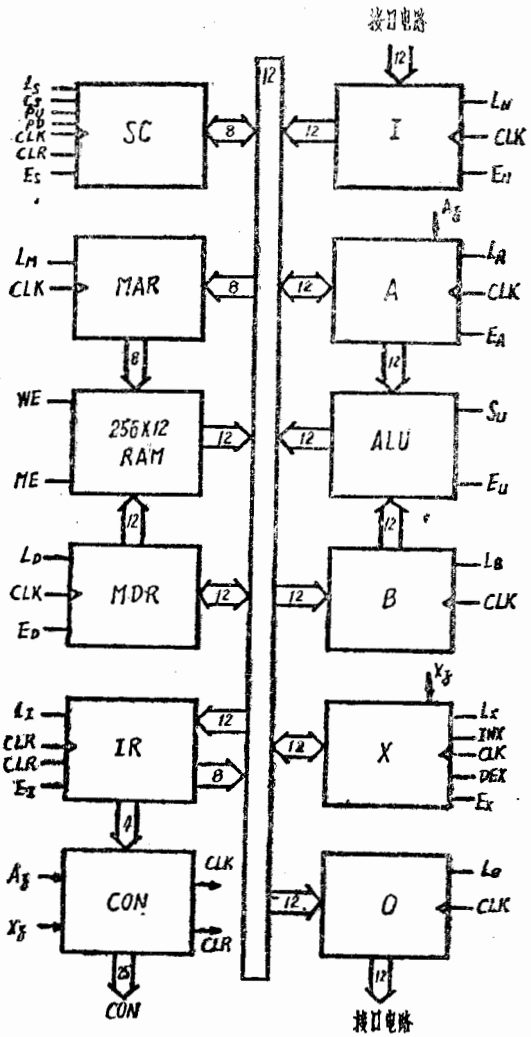


图2.7扩充模型机的结构→

*在教学中本节可省略不讲

1.存储器为RAM而不是PROM，容量256×12，为了便于写入，添加一个存储器数据寄存器MDR。

2.用SC(栈)代替PC，SC内有四个寄存器SC(0)，SC(1)，SC(2)，SC(3)，用一个2位的可逆计数器（可以进行加1和减1）SP来选择其中一个SC(SP)进行上述PC的工作。SP称为栈指示器。如图2.8所示。

例如，若SP=00，则选中SC(0)进行PC的工作若SP=10，则选中SC(2)进行PC的工作。其余类推。

PU信号用来使SP在CLK正跳变时加1，PD信号用来使SP在CLK正跳变时减1。

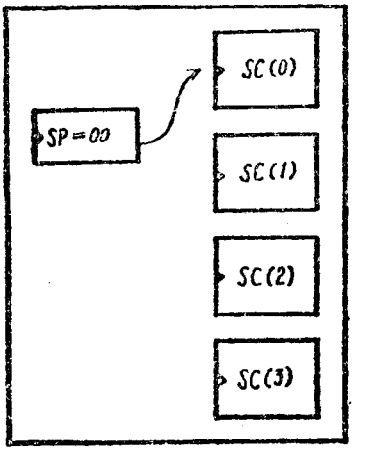


图 2.8 栈SC的結构示意图

3.增加了变址寄存器X。若信号NX=1，则X在CLK正跳变时加1，若信号DEX=1，则X在CLK正跳变时减1。它的作用将在下面予以介绍。

4.增加了输入寄存器I。外部设备通过I将信息送入RAM。

5.总线为12位。

扩充模型机的指令系统见表2.4。指令中的地址码使用一个任意数字，以便阐明指令的操作。

表2.4 扩充模型机的指令系统

汇编语言		机器代码 二进制		十六进制	操作及说明
助记符	操作数	操作码	地址码		
LDA	32	0 0 0 0	0 0 1 1	0 3 2	A←R3 2
ADD	33	0 0 0 1	0 0 1 1	1 3 3	A←A+R3 3
SUB	34	0 0 1 0	0 0 1 1	2 3 4	A←A−R3 4
STA	3B	0 0 1 1	0 0 1 1	3 3 B	R3B←A
NOP		0 1 0 0	× × × ×	4 × ×	不操作
LDX	43	0 1 0 1	0 1 0 0	5 4 3	X←R4 3
DEX		0 1 1 0	× × × ×	6 × ×	X←X−1
INX		0 1 1 1	× × × ×	7 × ×	X←X+1
JMP	56	1 0 0 0	0 1 0 1	8 5 6	PC←56
CLA		1 0 0 1	× × × ×	9 × ×	A←0 0 0
JIZ	5E	1 0 1 0	0 1 0 1	A 5 E	若X=0PC←5 E
JMS	63	1 0 1 1	0 1 1 0	B 6 3	SP←SP+1 SC(SP)←63
BRB		1 1 0 0	× × × ×	C × ×	SP←SP−1
INP		1 1 0 1	× × × ×	D × ×	A←I
OUT		1 1 1 0	× × × ×	E × ×	O←A
HLT		1 1 1 1	× × × ×	F × ×	停

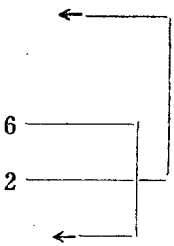
二、程序设计举例

例1. 试问下列程序中R2—R4的指令段执行多少次？

R0	NOP	
R1	LDX	A
R2	DEX	
R3	NOP	
R4	JIZ	6
R5	JMP	2
R6	NOP	
R7	HLT	
RA	03D	

解：共执行三次

			第一次	第二次	第三次
R0	NOP		V		
R1	LDX	A	X= 3		
R2	DEX		X= 2	X= 1	X= 0
R3	NOP		二次 V	V	V
R4	JIZ	6	V	V	V
R5	JMP	2	V	V	
R6	NOP				V
R7	HLT				V
RA	03D				



例2. 试设计 3 × 8 =？ 的程序

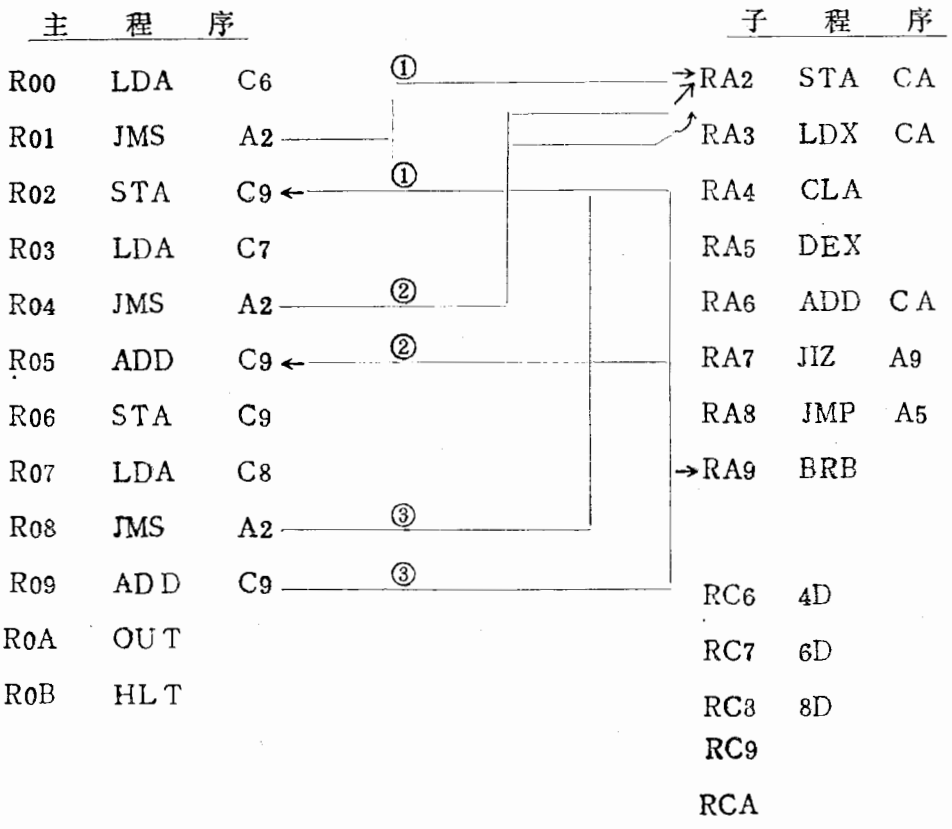
解：

R0	CLA	
R1	LDX	A
R2	DEX	
R3	ADD	B
R4	JIZ	6
R5	JMP	2
R6	OUT	
R7	HLT	
RA	3D	
RB	8D	

改变例1中R0，R3，R6单元中指令即得。

例3.试设计一程序以求 $4^2+6^2+8^2=?$

解：



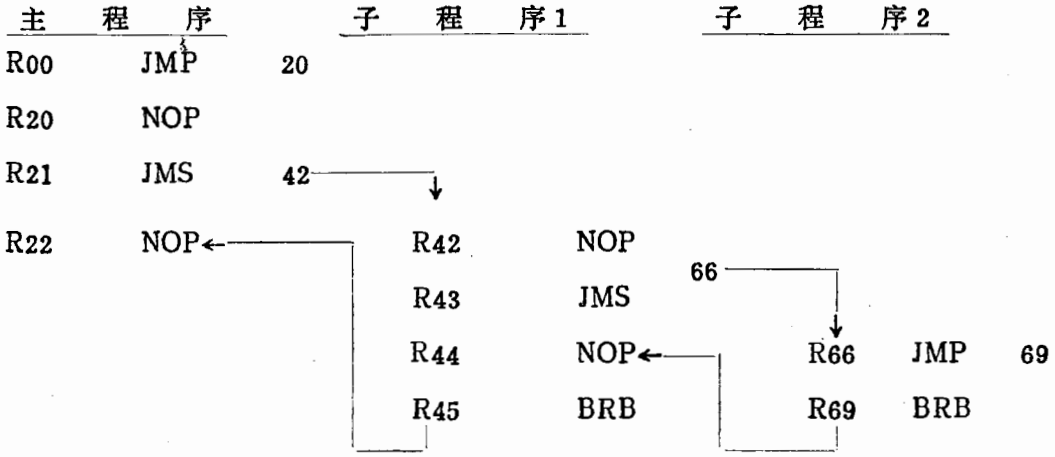
题中有三次要求一个数的平方，为此编写一个求平方的子程序（从RA2单元开始）。每当求平方时调用此程序即可，而不必重复三次编写“求平方”的程序，从而可简化程序设计，并节省存储单元的空间。

子程序中最后一条指令BRB并不要求给出地址码（即使能给出，也是一个变化值，在编写程序时不易表示），那末子程序是如何回到主程序中刚才离开的地方？将在下面予以介绍。

三、栈的结构S C

SC中有四个寄存器SC(0)，SC(1)，SC(2)，SC(3)。它们中的任何一个都可象PC那样工作。每个节拍内只能由SP来选中一个进行PC的工作。例如SP＝00，则由SC（0）进行PC的工作。如果SP＝10，则由SC（2）进行PC的工作。SP是一个二位的可逆计数器。当PU＝1时，在CLK正跳变时SP增1；反之，当PD＝1时，在CLK正跳变时SP减1。在复位时SP＝0。

下面分析一个例子说明其工作过程。其程序如下所示。



这个程序的执行过程如下表所示：

节 / 指令	T0 MAR←SC	T1 IR←R(MAR)	T2 SC←SC+1	T3	T4	T5
R00 JMP 20	MAR=SC(0) =00	IR=R00 =820	SC(0)=01	SC(0)=20		
R20 NOP	MAR=SC(0) =20	IR=R20 =4××	SC(0)=21			
R21 JMS 42	MAR=SC(0) =21	IR=R21 =B42	SC(0)=22	SP←SP+1 SP ‖ 01	SC(1)=42	
R42 NOP	MAR=SC(1) =42	IR=R42 =4××	SC(1)=43			
R43 JMS 66	MAR=SC(1) =43	IR=R43 =B66	SC(1)=44	SP←SP+1 SP=10	SC(2)=66	
R66 JMP 69	MAR=SC(2) =66	IR=R66 =869	SC(2)=67	SC(2)=69		
R69 BRB	MAR=SC(2) =69	IR=R69 =C××	SC(2)=6A	SP←SP−1 SP=01		
R44 NOP	MAR=SC(1) =44	IR=R44 =4××	SC(1)=45			
R45 BRB	MAR=SC(1) =45	IR=R45 =C××	SC(1)=46	SP←SP−1 SP=00		
R22 NOP	MAR=SC(0) =22	IR=R22 =4××	SC(0)=23			

当主程序执行 R21 单元“转子”（转子程序）JMS 指令时，主程序下一条要执行 的指令的地址保存在SC（0）中，此后暂停使用SC（0）。在执行子程序时，使用SC(1)作为程序计数器；当执行R45单元中BRB指令时，只要恢复使用 SC(0)作为程序计数 器，即 可找到返回的地址。子程序 1 转子程序 2，以及返回的过程与此相同。 SC中有四个计数器，因此子程序可嵌套三级。这种结构是执行“转子”指令时，从硬件上保护程序计数器的内容，Intel4040即是采用这种结构。现代大多数微型机都在RAM中开辟一个栈区，将程序 计 数 器的内容保护在栈中，从而可以使SC中只有一个计数器PC，如同模型机那样。这种方法的优点是：子程序可以嵌套很多级，并且还可以将CPU中一些工作寄存器的内容保 护 进 栈，以免在执行子程序时破坏主程序运算中所得到的中间结果。详细过程将在下章叙述。

2.7 模型计算机系统的简化

图2.1所示的模型计算机可以看作由三个部件组成，每个部件制做在一个芯片 内，也 就是三个芯片即可构成一个计算机系统，如图2.9所示。这三个芯片为：

1.CPU—包含A，ALU，B，PC，IR，CON

2.存储器—PROM，MAR

3.输入输出—I，O

这种模型计算机传送数据和地址分时地（Multiplexed）共用一条总线。但只有极 少 数微型机（如Intel 4004／4040和F8）使用这种工作方式，大多数微型机使用两条总线分别传送数据和地址。后者的优点是可以缩短微型机的指令周期，即提高其运算速度；缺 点 是 芯片上，使用更多的引脚从而增大器件的外形尺寸。简而言之，微型机系统就是若 干 个 芯 片（CPU，RAM，I／O）挂到三条（数据、地址、控制）总线上而构成。如图2.10 所示。

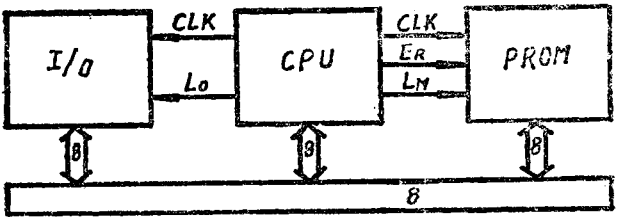
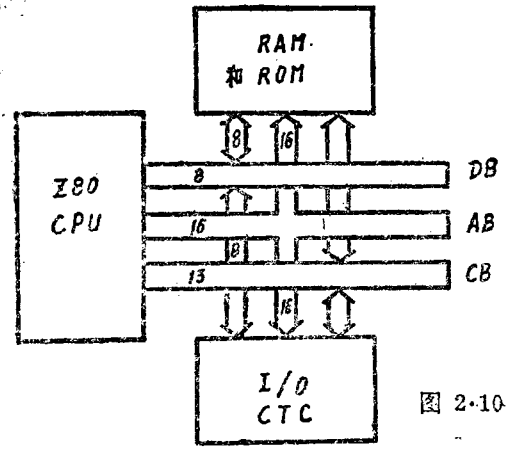


图 2.9 模型机的简化图



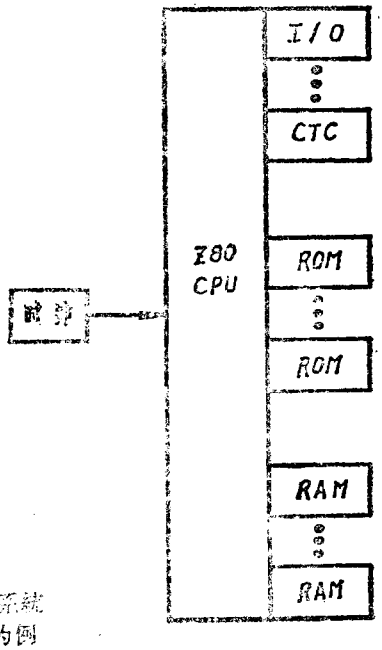


图 2·10 一般的微型机系统
——以 Z80 为例

第三章　Z80—CPU的结构

CPU是中央处理单元（Central Prccessing Unit）的简称。它是微型计算机的中枢，担负处理数据和控制整个微型计算机系统的使命。

Z80—CPU是指由美国Zilog公司原型设计生产的微型计算机系统中的超大规模集成电路CPU芯片。根据厂家所制定的指令系统，在使用这种芯片时用户可以直接编写程序。

为了研究Z80—CPU的结构，我们需要建立它的程序模型，从编写程序的角度来观察组成CPU的各部分。为了掌握Z80程序的编写方法，我们将指令系统视为CPU的传递函数，逐条理解其中每个指令的执行在CPU硬件结构中所引起的变化。

3.1　CPU在微型计算机系统中的作用

因为CPU的结构和指令系统是根据对于CPU功能的要求来设计的，所以在具体介绍Z80—CPU之前，有必要先考察它在整个微型计算机系统中的作用。

如图3.1所示的微型计算机系统，除CPU外，还有存储器和输入/输出器件(I/O bevice)通由$\overline{CPU}$表示。CPU的功能可以概括为对数据的**运算**（CPU内部）、**传送**（CPU内部以及与$\overline{CPU}$之间）、以及CPU对它本身及对$\overline{CPU}$的**控制**。所有上述三种功能都是通过对于数据字或控制字的传送和处理来实现的。这些字的每8位（bit）字长叫做一字节（byte），每4位字叫做半字节（nibble）

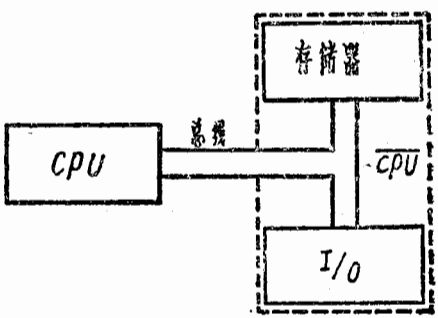


图 3.1　微型計算机系統的簡化框图

在CPU内部的数据传送主要是通过软件设计（程序编写）来操纵的，在CPU和$\overline{CPU}$之间的数据传送还需要配合硬件设计（电路接口）来实现。

3.2　Z80—CPU的结构

Z80—CPU是在8080A微处理器的基础上改进设计的。主要的改进是、

1、减少芯片数目。为了有效使用8080ACPU，还需配用8224时钟发生器和8228系统控制器。Z80—CPU将三种芯片的功能综合在一起，减少了芯片数目，为用户提供便利。

2、减少电源种类。8080A需要三种电源（＋5V，－5V和＋12V），Z80—CPU只需要一种＋5V电源。

3、其他功能方面主要是指令系统功能的加强，包括更强有力的中断功能，变址寻址功

能，数据块传送功能，位操作功能，动态存储器刷新功能等。

Z80—CPU的结构如图3.2所示，主要由寄存器、运算器、指令译码及定时和控制几个部分组成。

一、寄 存 器

在CPU的整个功能中，寄存器扮演最重要的角色。寄存器实质上是一个小的存储器，它由一定数目的二态存储单元所组成，本身具有一定的地址以便加以识别。

在Z80—CPU中，包含有208位可供用户使用的寄存器，全部用静态RAM实现。它们分为专用和通用两类，其程序模型如图3.3所示。

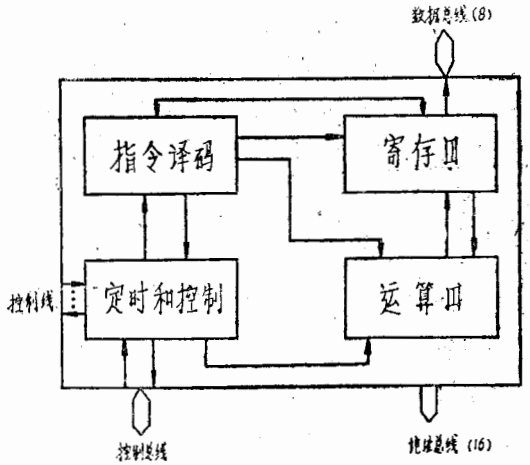


图3.2 Z80—CPU的简化框图

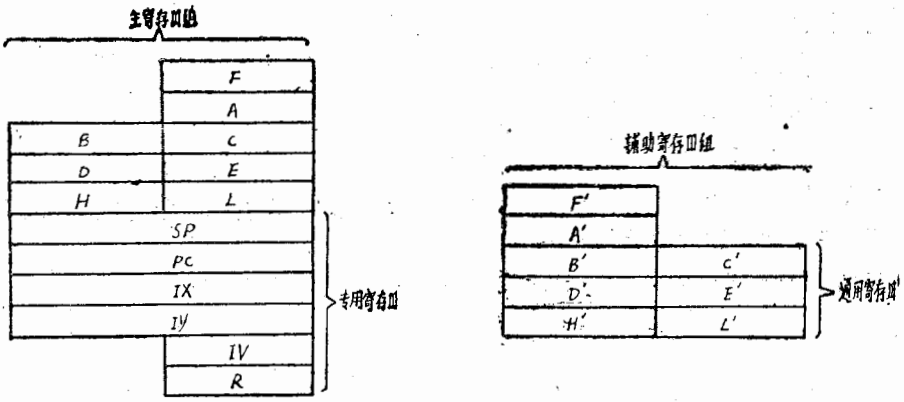


图3.3 Z80—CPU的寄存器

Z80—CPU中的专用寄存器包括：

1、程序计数器（Program Counter，简称PC）。其内容是要执行的下一指令字节所在的存储单元的地址。一个指令周期开始时，CPU将PC的内容放在地址总线上，使CPU从存储器中取出指令的第一个字节。CPU续而使PC内容增1，使PC指向相续的指令字节。

2.栈指示器(Stack pointer，简称SP)。用于贮存某一指向特定存储单元地址的寄存器

叫做指示器。指示器的内容，即所贮存的地址叫做指针。

栈是一种暂存数据（或地址）的存储区，好象一个货栈。组成栈的一些存储单元，象一落碟子，有规律地排列着。最先进栈的数据字构成栈底，栈的另一端为栈顶(Top Of Stack，简称TOS)。对于用户来说，只有最后进栈的一个数据字，即处于栈顶的一个数据字才能被存取（见图3.4）。

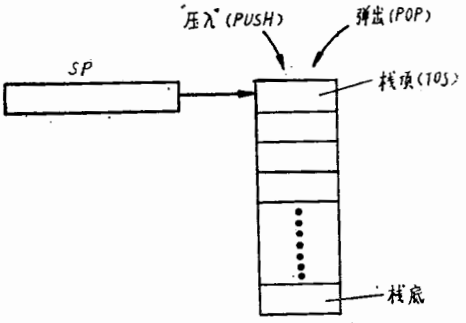


图3.4 栈与操作

从软件的角度看，栈是一种数据结构。其中各元素的内容和数目都在动态地变化，但变化方式不是随意的，而是先进后出（First In Last Out，简称FILO）的。所有的元素进栈或出栈都要通过栈顶。中间的元素不能跳出这个序列。因此栈也叫做下推表。

在栈操作中，栈中的各元素实际上并未移动。唯一变化发生在栈指示器SP中。SP是一个专用16位寄存器，用于贮存栈顶地址。当一个数据字进栈时，需将SP增1（或减1）栈顶上升，数据字存放在SP增量后所指向的新栈顶。这种操作叫做"压入"（PUSH）。如果要从栈中取出数据，则最先取出已经处于栈顶的数据字，然后将SP减1（或增1），降下栈顶，并依此类推。这种操作叫做"弹出"（POP）。

3.变址寄存器（Index Register）。在Z80—CPU中，设有两个完全相同的变址寄存器IX和IY。这是一种贮存16位地址的寄存器。其内容不仅可以在程序控制下循环增1或减1，而且能与指令中包含的一个操作数（叫做偏移，Offset）相加，形成一个新的有效地址，指向所要访问的单元（见图3.5）。在处理数组和表时，使用这种寄存器尤为便利。

4.中断矢量寄存器（Interrupt Vector register，简称IV或I）。使微型计算机暂停正常程序流程，接受输入信号的请求去执行一定的服务子程序（称为中断服务程序），然后再返回原来的程序流程，这种工作方式叫做中断（见图3.6）。

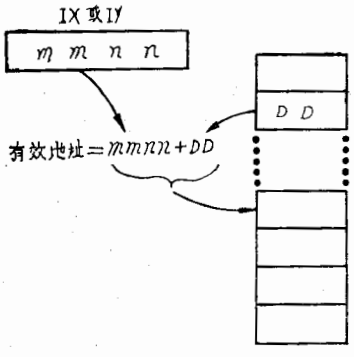


图3.5 变址操作示意图

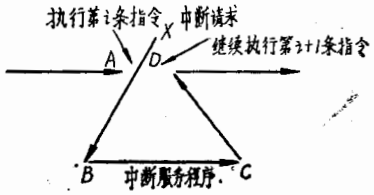


图3.6 中断操作示意图

为了能够准确返回原来的程序流程，需要自动将中断前一时刻各CPU寄存器的内容保护起来，称为保护现场；而在执行完中断服务程序之后再恢复这一现场。此外，为了使CPU在响应中断后能自动转向中断服务程序，需要形成中断服务程序的入口地址，也叫做中断矢量。在Z80—CPU中，为用户提供了形成中断矢量的三种方式，方式0，方式1和方式2。在方式2中，借助于3位的中断矢量寄存器IV来形成中断矢量是最完备的一种。如图3.7所示，当Z80—CPU工作在这种方式时，需要制定一个16位中断矢量地址表。这个表可以位于可寻址的存储器中的任何部位。其中各登记项所组成的16位地址标识各中断服务程序中第一条可执行的指令，即入口地址。当CPU在方式2的条件下响应中断时，请求中断的外部设备应将一个中断响应矢量的低8位放在数据总线上去。Z80—CPU将IV的内容与这个矢量结合起来，形成一个16位地址。用这个地址，就能够访问中断地址矢量表，进入所需要的中断服务程序。

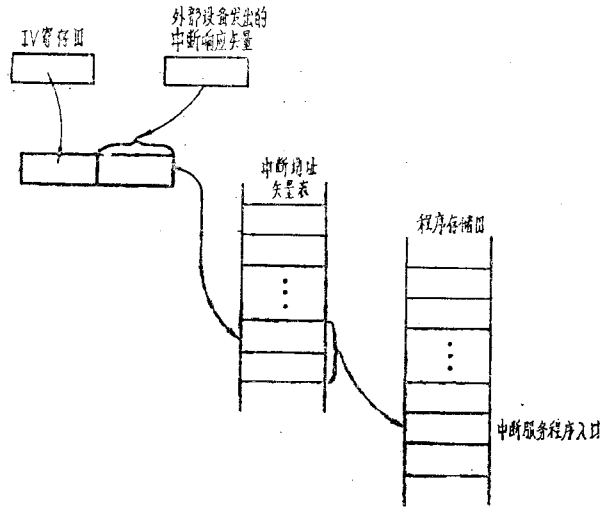


图3.7　中断矢量的形成

5.存储器刷新计数器（memory Refresh counter，简称R）。在Z80微形计算机系统中，允许采用十分廉价的动态RAM，但是，除非大约每2 ms进行一次刷新，否则在这种存储器中所贮存的信息就会消失，为了解决这个问题，需要能够周期性地访问动态RAM，在每次访问中都将各存储单元内容重新写入动态刷新电路。在Z80—CPU中的存储器刷新计数器就是为此而设置的。

6.累加器（Accumulator，简称A或Acc）和标志寄存器（Flag register）。在CPU中累加器是使用最频繁的一种寄存器。在CPU完成各种运算期间，累加器作为中间结果的暂存器；在各种8位算术运算中，总有一个操作数存放在累加器中。CPU还利用累加器来实现逻辑操作，移位及某些指令所要求的特殊操作。

标志寄存器由六个1位标志触发器组成(图3.8)。它们反映CPU内部的某种操作状态，保存或反映运算的部分结果；有的标志还能输送补充加数（如CY标志）。在Z80—CPU中，设有两类标志。一类主要为CPU形成判定操作提供测试条件，叫做测试标志；另一类主要用于BCD运算，叫做非测试标志。测标志用来作为执行转移、调用或返回指令的条件。根据对某个标志位状态测试的结果是1还是0，就能够决定是否执行相应的操作。测试标志包

括：

1)进位标志（CarrY,简称C或CY）。如果最后一次运算从最高位（MSB）产生进位，则将其存入此标志。在做减法时，最高位产生借位时此标志也能置1。此外，各种移位指令也能影响此标志。

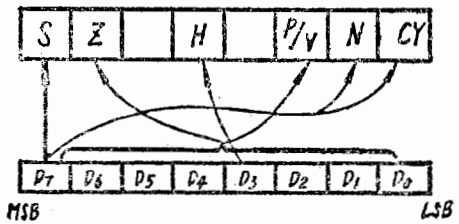


图 3.8 Z80—CPU的标志寄存器

2)零标志（Zero，简称Z）。如果最后一次运算结果为零，则此标志置1；否则置0。

3)符号标志（Sign，简称S）。在有正负号的运算中，用数据字的最高位表示正负号。如果最后一次运算的结果为负数，S标志置1；否则置0。

4)奇偶／溢出标志（Parity／oVerflow，简称P／V或P／O）。这个标志主要的用途有两个：一是标识逻辑运算结果的奇偶性，如果结果中有偶数个1，则P／V标志1；否则置0。二是标识算术运算是否发生二的补码溢出，如果发生溢出，则此标志置1；否则置0。

对于一个数据字节，如用最高位表示正负号，则有7个有效位，能表示-128—+127范围内的数。如果运算的结果超出了这个数值范围，就会发生溢出，以两数M，N相加为例，设其最高位分别为M_7，N_7；结果为R、其最高位为R_7，则有下列真值表：

M7	N7	R7	P/V
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

若两个数同号，和的符号却不同，说明发生了溢出

若M，N具有不同符号，不会发生溢出

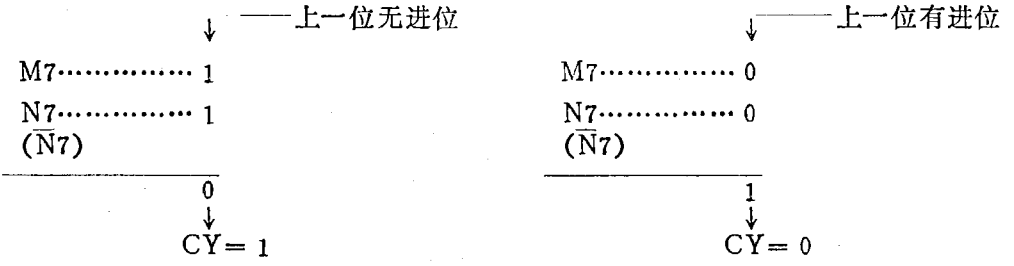
若两数同号，和的符号相同，说明没有发生溢出

判别减法是否发生溢出，要看参加运算数的符号相反的情况，其真值表如下：

M7	N7	R7	P／V
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

若M，N具有不同符号，差的符号与被减数相反，说明已发生溢出

对加法来说，由上述真值表可见



故可列出产生溢出的条件为：

上一位有进位送入最高(符号)位，从最高位无进位输出；

上一位无进位送入最高(符号)位，从最高位有进位输出。

将此条件写成表达式：　　$P/V=d_{bcy}+CY$

试中d_{bcy}表示从和的第6位向最高（第7）位的进位。

Z80—CPU的非测试标志有两个：

1)半进位标志（Half carry简称H）。当低位半字节向高位半字节产生BCD进位或借位时，此标志置1。这个标志用于校正BCD加法或减法的结果。

2)减标志（Negate,简称N,此标志也叫做加／减标志，Add／Subtract flag)。在BCD运算时，十进制调整指令DAA利用此标志来区别加法和减法。（前面进行的一次）操作是加法，N置0；（前面进行的一次操作）是减法，N置1。

累加器和标志位都可以由一些特定的指令来直接访问或受到影响。对于某几种涉及寄存器对的指令（如PUSH，POP指令），累加器和标志寄存器被视为一个寄在器对，叫做程序状态字（Program Status Word)或处理器状态字(Proceasor Status Word)。简称PSW。在Z80—CPU中，有两组独立的累加器A，A′和标志寄存器F，F′：组成PSW和PSW′。

Z80—CPU的通用寄存器也叫做工作寄存器（Working registers）。它们具有多种功能，并能在程序的直接操纵下工作。利用通用寄在器，可以暂存参加运算的操作数和中间结果，避免中间结果在存储器和累加器之间来回传送。从这个意义上讲，它们也叫做次级累加器。另外，通用寄存器也可以存放数据地址，作为数据计数器使用。这时就需组成寄存器对，作为专门指向数据存储区的指示器。

在Z80—CPU中有原组独立的8位通用寄存器，主寄存器组（main register set)B，C，D，E，H，L可以进行8位操作，也可以组成“对”进行某些16位操作。辅助寄存器组(alternative register set)B′，C′，D′，E′，H′，L′的排列与主寄存器组一一对应。如前所述，当Z80—CPU响应中断后需要保护现场；待中断服务程序结束后需要恢复现场。如果只有一级中断申请的话，就不必再设栈区。只用一条指令就能将主寄存器内容送入辅助寄存器，然后在需要时再交换回来。这样就能大大简化中断操作。因此，辅助寄存器也可以看成是设置在CPU内部的栈区。

二、运　算　器

运算器也叫做算术和逻辑单元（Arithmetic & Logic Unit，简称ALU）。其主要功

能是完成各种算术和逻辑运算。Z80—CPU除具有基本的加、减、比较、增1、减1“与”“或”、“异”等运算功能外，还具有丰富的移位功能以及对单个位的处理（对于各寄存器的任一位的置、复位、测试）功能。

三、指令译码及定时和控制

这是整个CPU的控制中枢，与将程序存储器送来的指令翻译成控制信号，以产生所需要的动作；它使CPU能够找到为指令所要求的贮存在存储器和寄存器中的地址或数据；它向ALU提供控制输入，使之执行指令所规定的运算；它监视外部控制信号，并产生适当的响应；它为存储器和I／0器件提供状态，控制和定时信号等。总之，它的功能是在正确的时刻，将信息准确地送往某个目的地。它动作的依据是该时刻执行的指令，它所产生的作用是发往CPU和CPU各部分的控制和定时信号（见图3.9）。

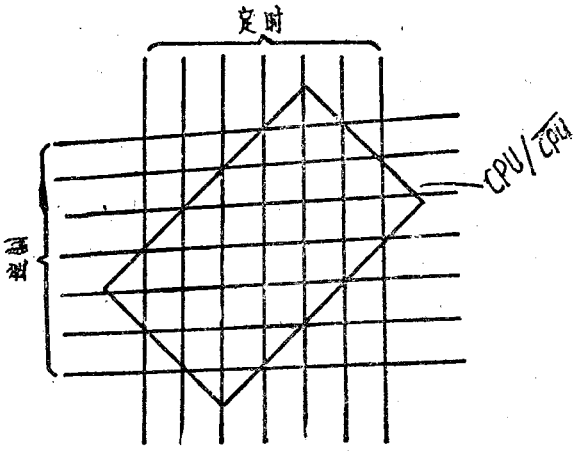


图 3.9　控制的器功能

Z80—CPU的动作需要精确定时。基本定时脉冲由外部振荡器产生，接至CPU的φ输入端。如图3.10所示，在两个定时脉冲上升沿之间的持续时间是一个时钟周期(Clock period)，它等于机器处于一种状态的持继时间，因此也叫做T状态（T state）或T周期（T cycle）。CPU实现某种规定的基本操作所需时间叫做机器周期（Machine cycle）或M周期，一般为3或4个T状态。有些指令在执行时自动插入或$\overline{\text{CPU}}$通过$\overline{\text{WAIT}}$信号插入等待（WAIT）状态，这时就可能有5至6个T状态。一条指令从取出到执行完毕的持续时间叫做指令周期

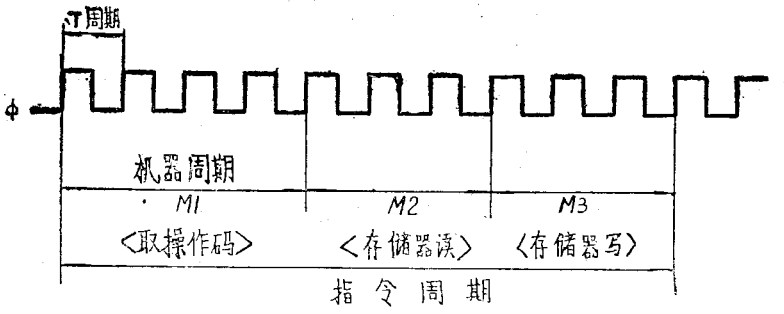


图3.10　Z80的定时波形

（Instruction cycle）。根据指令内容的不同，Z80—CPU执行一条指令可能需要一至六个机器周期。

四、总线

在CPU内部各单元之间的数据传送是通过内部总线实现的。对于用户来说，有三条总线使CPU与$\overline{\text{CPU}}$相联系。

1、数据总线（Data Bus，简称DB）。它是8位双向总线。担负CPU与$\overline{\text{CPU}}$的数据传输。

2、地址总线（Address Bus，简称AB）。它是由CPU向外发出的16位单向总线，总共可以选择$2^{16}=65536$个不同的地址。

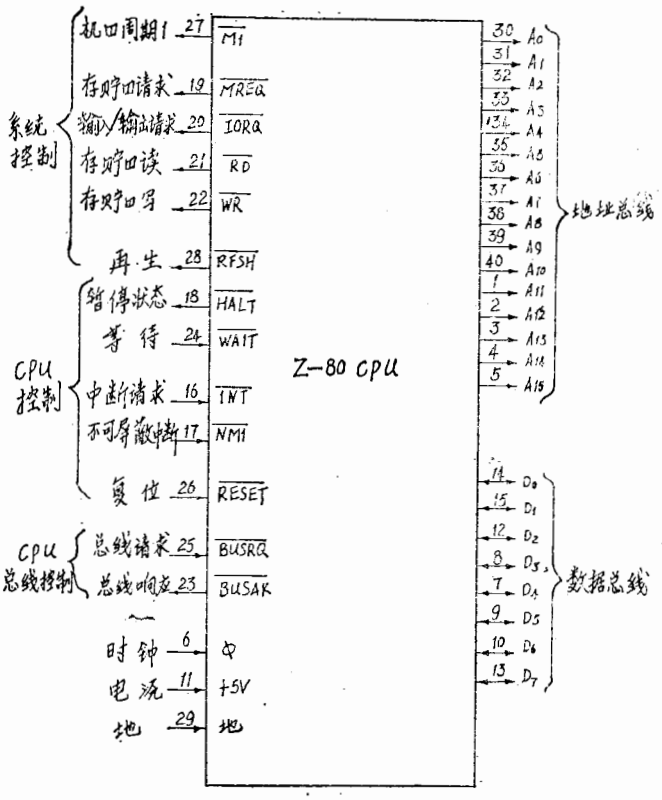


图3.11　Z80—的引脚图

3、控制总线（三态线）及定时、控制线（输入或输出线）见图3.11的引脚图。其中，定时、控制信号引线分为三组：系统控制（6个），CPU控制（5个）和CPU总线控制（2个）。所有涉及这些引线的信号都是低电平有效的。

涉及系统控制方面的信号控制CPU／$\overline{\text{CPU}}$之间的数据传送。其中的信号包括：

$\overline{M_1}$（Machine cycle 1，机器周期1）。输出。它标示在一条指令的执行期间处于取指令的机周期。

$\overline{\text{MREQ}}$(Memory REQuest，存储器请求)。三态输出。此信号标示地址线上存在一个

用于存储器读或写操作的有效地址码。

$\overline{IORQ}$（I/OReQuest，输入/输出请求）。三态输出。标示此时地址总线的低8位A_0—A_7含有一个有效的I/O口地址。中断响应操作在M_1，有效期间出现。IORQ也被用于中断响应。$\overline{M_1}$和$\overline{IORQ}$同时存在表示中断已被接受，在数据总线上有一个中断响应矢量存在。

$\overline{RD}$（memory ReaD，存储器读）。三态输出。标示CPU希望从存储器或I/O设备中读出数据。

$\overline{WR}$（memory WRite，存储器写）。三态输出。标示出CPU希望向存储器或I/O设备写入数据。

$\overline{RESH}$（ReFreSH，刷新）。输出。此信号标示出地址总线的低7位A_0—A_6是动态存储器的刷新地址。当前的$\overline{MREQ}$信号被用于动态存储器的刷新。

涉及CPU控制方面的信号影响CPU操作的进程。其中的信号包括：

$\overline{HALT}$（HALT state，暂停状态）。输出。在执行一条HALT指令后，$\overline{HALT}$输出低电平。这时CPU进入暂停状态。在此期间，它连续执行NOP指令，以维持暂停状态，又不中断对动态存储器的刷新。

$\overline{WAIT}$（MAIT state，等待状态）。输入。等效于8080A的READY输入。如果存储器或I/O设备在指定时间内来不及响应CPU的访问请求，则它们将$\overline{WAIT}$输入拉成低电平。作为对$\overline{WAIT}$的响应，CPU进入等待状态。

$\overline{INT}$（INTerrupt request中断请求）。输入。由I/O设备产生的中断请求信号。

$\overline{NMI}$（NonMaskable Interrupt requet，不可屏蔽中断请求）。输入。负跳滑触发中断请求信号。与INT相象，只是比前者具有更高的优先权。不受中断允许触发器状态的影响，因此不能被禁止。

$\overline{RESET}$(RESET,复位)。复位控制输入。使CPU进入起始状态，使PC，IV和R寄存器清零，置中断方式为0，并禁止通过INT输入的中断请求。所有的三态总线信号都被置入浮动状态。

涉及CPU总线控制方面的信号有两个。当外部设备请求占用微型计算机的数据、地址总线和三态控制总线时，外部设备将$\overline{BUSRQ}$（BUS ReQuest，总线请求）输入拉成低电平。在现行机器周期结束时，CPU将使所有的三态总线进入浮动状态，并用$\overline{BUSAK}$（BUS Acknowledge，总线响应）输出低电平来表示接受外部设备对占用结线的请求。这一对信号多用于实现直接存储器存取（DMA）。

在图3.11中，ϕ为时钟输入端，+5 V是CPU要求的电源，GND为接地端。

数据、地址和控制总线都是三态的。当它们处于浮动状态时，外部设备占用对总线的控制。

第四章　Z80—CPU指令系统

指令是CPU借以控制CPU内部各单元以及CPU各部分协调动作的命，CPU所具有的全部指令组成指令系统。因此，指令系统实际上全面描述了CPU的功能。

为了操作微型计算机实现某项指定任务，按照一定的规则排列的指令序列叫做程序。根据一定的算法，从指令系统中选取所需的指令，赋予一定参数后加以排序，就得到需要的程序。这个过程叫做编写程序。

对于计算机本身，用二进制代码书写的程序最易于接受。但对于用户，却宁可用符号代替指令码。在微型计算机的一般应用中，最广泛使用汇编语言指令。用这种指令编写的程序，叫做汇编语言源程序。这种程序中的各条指令经过一种专用程序——汇编程序被翻译成二进制代码的形式，叫做结果代码。整个源程序也就被翻译成为用二进制代码表示的目标程序。这种翻译过程叫做汇编。

如前所述，Z80是在8080A基础上设计的。在它的指令系统中，除包括了8080A的全部78条指令外，还增加了80条指令，总共有158条。

汇编语言指令是用汇编语句形式书写的。为了能编好程序，首先必须了解语句话法。

指令所载有的实际信息与指令代码的格式关系很大。在学习指令系统时，每条指令的代码格式也是需要我们了解的。

每种实际微型计算机的指令系统，都是由生产厂家制定的。因此，总是有差异的。为了尽快掌握一种微型计算机的指令系统，重要的不在于单纯背诵各条指令，而在于熟练指令的分类，善于剖析典型指令的格式和操作内容。这样，才能达到举一反三，事半功倍的良好效果。

4.1　指令的语句语法、代码格式和分类

一、语　句　语　法

汇编语言指令是按照下列规则编定的。任何指令都具有四个独立的和不同的部分，叫做字段（field）。例如

①	②	③	④
ALTR3B:	LD	A, (DSMEM7)	; GET NEW VAIUE

是从某程序中取出的一条指令，其中

①标号段（label field）：它是一个名字，用来标识16位的指令地址。在程序中的一条指令是否具有标号，视需要而定。也就是说，标号是任选的（Optional）。标号中打头的字符可以是字母表中的字母、符号等，但不得用阿拉伯数字。在标号的末尾需接一个冒号“：”。

②操作码段（OPeration code field）：它规定所要实现的操作。为了便于记忆，一种操作是用该操作的（英文）名称中的几个字母表示的，叫做助记符（mnemonic）。例如，助记符LD就是LoaD（传送，加载，装入）的简称；JP表示JumP（转移）操作；CP表示

ComPare（比较）操作等。

③操作数段（OPerand field）：即参与操作的数据。它与操作码一起，确定指令所要执行的操作。根据操作码段的要求，操作数段可以空白，具有一项，或者具有用逗号分开的两项。

在Z80指令中，有四种操作数：

寄存器（register）

寄存器对（register pair）

立即数据（immediate data）

16位存储器地址（16 bit memory address）

这些操作数可以有以下几种表示方式：

十六进制数据（Hexadecimal data）

十进制数据（Decimal data）

八进制数据（Octal data）

二进制数据（Binary data）

当前程序计数器 $（current Program counter）

ASCII常数（ASCII constant）

由标号标识的地址（address specified by label）

表达式（expression）

④注释段（Comment field），对于这一字段，唯一的要求是用分号起头。注释用来说明该指令在整个程序中的作用。这个字段不产生结果代码，对机器的工作无影响。

二、指令代码的格式

Z80指令存放在程序存储器（一般用ROM，PROM，EPROM，亦可用RAM）中相继的单元中。每个单元是一个字节，具有唯一的16位地址(见图4.1)。在Z80指令系统中，根据指令内容的不同，一条指令的长度可以是一、二、三或四个字节。第一字节或第一、二字节一定表示操作码。操作数如果存在的话，与操作码一起包含在第一、二字节中，或者单独包含于第三、四字节之中，如图4.2所示的几种情况。

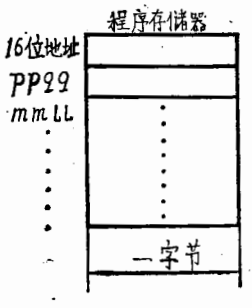


图 4.1 程序存储器的结构

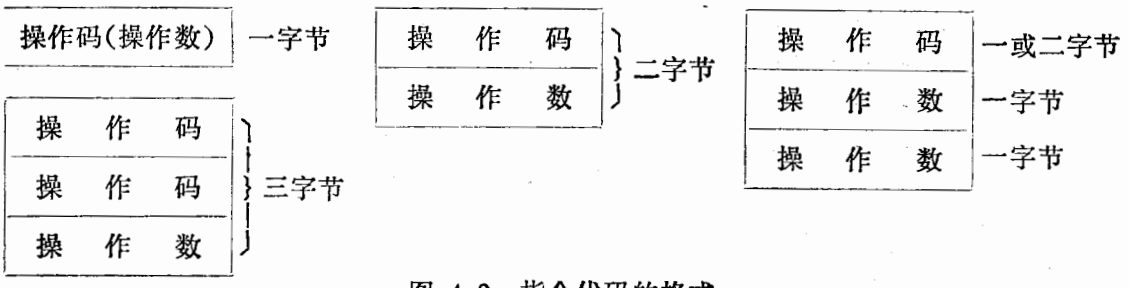


图 4.2 指令代码的格式

三、指令的分类

根据指令所实现的操作，可以将指令分为：

1、**数据传送指令**：其中包括8位传送指令组；16位传送指令组；交换指令组；数据块传送和查找指令组和输入／输出指令组。

2、**数据操作指令**：其中包括8位算术和逻辑指令组；16位算术指令组；通用算术指令组和循环与移位指令组。

3、**程序控制指令**：其中包括转移指令组和子程序调用和返回指令组。

4、**CPU控制和位操作指令**：其中包括CPU中断控制指令组；CPU其他控制指令组和位操作指令组。

4.2 数据传送指令

数据传送指令是最常用，也是最基本的一类指令。其共同特点是所执行的操作是在CPU寄存器之间或CPU寄存器与储存器之间传送数据；指令必须指明数据传送的源和目的地；且源的内容不因执行传送指令而发生变化。尽管各种传送指令所实现的操作实质上都是相同的，但根据功能可以分为传送、交换、查找、I/O等指令组；根据操作数的长度不同又可分为8位传送指令组，16位传送指令组。下面分别介绍。

一、8位传送指令组

这组指令的一般形式为

LD dst_8 src_8 （LoaD sorrce to destination）

所执行的操作是，将源内容传送到目的地。所传送的内容都是8位数据。这个源可以是某个寄存器，令

src_8 = r′ = A, B, C, D, E, H, L

及 dst_8 = r = A, B, C, D, E, H, L

则有一字节指令 LD r, r′

其代码格式为

0	1	d	d	d	s	s	s

其中 000——dst或src＝B

001——dst或src＝C

010——dst或src＝D

011——dst或src＝E

100——dst或src＝H

101——dst或src＝L

111——dst或src＝A

例如，寄存器H的内容为8AH，记作H＝8AH，E＝10H，则执行指令

LDH，E后

H＝10H，E＝10H

式H＝10H中左端的H表示“寄存器H”，右端的H表示10为十六进制数。

传送的源可以是一个8位数据。因为这个数据就包含在指令之中并直接参加操作，所以叫做立即数。若用n表示0——255范围内的一字节立即数，则有

LD r, n

表示将立即数n传送到存器r，记作

r←n

传送的源或目的地可以是一个指针所指向的存储单元。例如（HL）表示由寄存器对HL内容作为指针所指向的存储单元的内容；（IX+d）表示由变址寄存器IX的内容与某个位移量d（displacement）相加形成的指针所指向的存储单元的内容。这样，我们有以下指令

LD r, (HL),	r←(HL);
LD (HL), r,	(HL)←r;
LD r, (IX+d)	r←(IX+d);
LD r, (IY+d)	r←(IY+d);
LD (IX+d), r,	(IX+d)←r;
LD (IY+d), r,	(IY+d)←r.

有的指令以某个寄存器对的内容为指针，在所指向的存储单元和累加器之间传送数据，这样的指令有

LD A, (BC),	A←(BC);
LD A, (DE),	A←(DE);
LD (BC), A,	(BC)←A;
LD (DE), A,	(DE)←A.

刷新寄存器R和中断矢量寄存器1只有8位，它们与累加器之间可以传送数据

LD A, R,	A←R;
LD A, I,	A←I;
LD R, A,	R←A;
LD I, A,	I←A.

也可以在指令中用一个16位数据规定单元的地址，在此单元和累加器之间进行8位数据传送。

LD A, (nn),	A←(nn);
LD (nn), A,	(nn)←A.

二、16位传送指令组

这组指令的一般形式为

LD dst_{16}, src_{16}

其中dst_{16}和src_{16}是某寄存器对的(16位)内容或以二字节数值nn和nn+1为地址的相继单元的(16位)内容。各条指令的操作数（dst_{16}和src_{16}的具体含义）详见《Z80微型电脑袖珍设计手册》附表（以下简称"附表"）。这里要指出，有几条指令的操作段包含8位和16位两种操作数。这时的操作仍是16位的，分两步完成。例如

LD HL, (nn)

其中 HL——寄存器对HL，16位

(nn)——16位地址nn所指定的存储单元内容，8位

此指令所实现的操作为

H←(nn+1)——将地址为nn+1的单元内容装入H；

L←(nn)——将地址为nn的单元内容装入L。

在16位传送指令组中，还有6条指令涉及栈操作。先看进栈。源是qq（寄存器对AF,BC,DE,HL，中的任意一个），IX或IY。以PUSHqq为例，它所执行的操作，是将寄存器对qq的内容从栈顶压入栈中。其操作如图4.3所示，记作

PUSHqq

(sp−2)←qqL

(sp−1)←qqH

sp←Sp−2

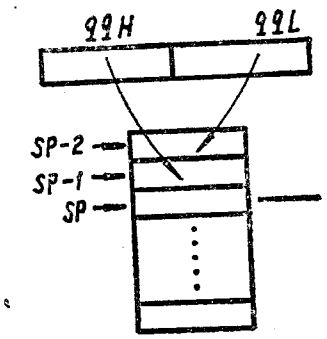


图 4.3 PUSH指令的执行

再看出栈指令POPqq。它所执行的操作是将栈顶内容依次弹出，分别送入寄存器对qq的高位和低位中。其操作如图4.4所示，记作

POPqq

qqH←(sp+1)

qqL←(sp)

sp←sp+2

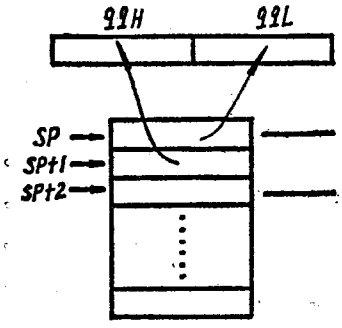


图 4.4 POP指令的执行

三、交换指令组

这组指令的功能是交换源和目的地内容，可以简化某些传送操作。尤其是交换主、辅寄存器内容的指令EXX和EX，AF，AF′，在单级中断保护现场时可代替繁冗的栈操作。交换指令组共有下列六条指令：

指令	主要用途
EX. DE,HL(DE←→HL)	交换以便建立指针 (EXchange to set the pointer)
EX AF,AF′(AF←→AF′	交换以便保护现场 (EXchange to save status)
EXX (BC←→BC′; DE←→DE′; HL←—HL′)	
EX (SP), HL (H←←(SP+1); L←→(SP))	交换以便保护指针
EX(SP), IX (IXH←→(SP+1); IXL←→(SP))	进栈并取出栈顶内容作为新指针 (EXchange to save pointer in stack and set previous top of stack as new pointer)
EX(SP),IY (IYH←→(SP+1); IYL←→(SP))	

四、数据块传送和查找指令组

利用数据块传送指令能将多达65536字节的数据在两个存储器缓冲区之间传送。这两个缓冲区可以位于存储器中的任何部位。在这组指令操作中，HL寄存器对指向源缓冲区，DE寄存器对指向目的地缓冲区，BC寄存器对作为字节计数，如图4.5所示。

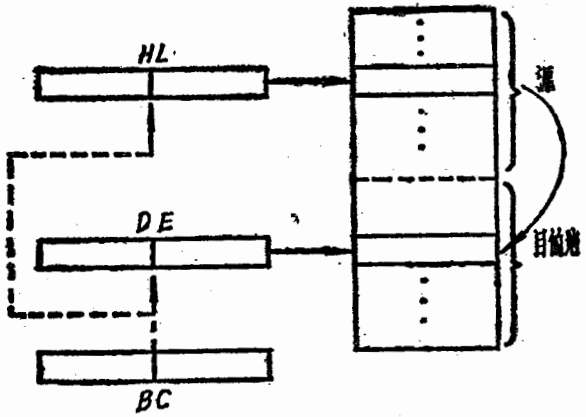


图 4.5 数据块传送指令

数据块传送指令有多种传送方式：

1、增址型，即每传送一个数据字节后，源／目的地地址增1，数据从低地址向高地址传送；

2、减址型，即每传送一个数据字节后，源／目的地地址减1，数据从高地址向低地址传送。

两种型式的指令都能再分为单步型和循环型两种。单步型指令只执行一次操作。循环型指令重复同一个操作，直至某个测试条件得到满足为止。现分述如下：

1)增址单步指令LDI。在存储单元间传送数据，目的地和源地址增量（transfer data between memory locations, Increment destination and source addresses）。记作

(DE)←(HL)，DE←DE+1，HL←HL+1，BC←BC−1

例如：BC=0007H，DE=2222H，HL=1111H，(1111H)=88H，执行指令LDI后

HL=1112H，DE=2223H，(2222H)=88H，(1111H)=88H，BC=0006H

2)增址循环指令LDIR。这条指令与LDI完成相同的操作。只是计数寄存器BC不断自动减1，每次都按新地址传送数据，直至BC=0，操作自动停止。

例如：HL=1111H，DE=2222H，BC=0003H，各单元内容为

(1111H)=88H　　(2222H)=66H

(1112H)=36H　　(2223H)=59H

(1113H)=A5H　　(2224H)=FCH

执行LDIR后，寄存器对和各存储单元内容变为

HL=1114H，DE=2225H，BC=0000H，

(1111H)=88H　　(2222H)=88H

(1112H)=36H　　(2223H)=36H

(1113H)=A5H　　(2224H)=A5H

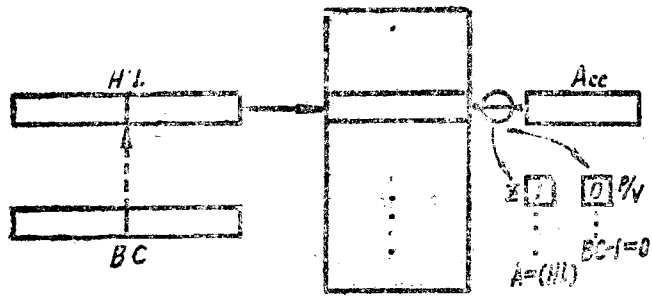
3)减址单步指令LDD。在存储单元之间传送数据，目的地和源地址减量（transfer data between memory locations, Decrement destination and source addresses）。记作

(DE)←(HL)，DE←DE−1，HL←HL−1，BC←BC−1

4)减址循环指令LDDR。这条指令与LDD相对应，计数值BC自动减1，直至BC=0时，操作自动停止。

还应指出，Z80的数据块传送指令同样适用于动态存储器。在执行这类指令时，动态存储器被不断自动刷新。

当我们对一个大存储器的缓冲区进行搜索，以便找到其中的某个字节时，Z80数据块查找指令能够提供极大便利。查找指令能使存储器缓冲区逐字节地与累加器内容相比较。"HL寄存器对"作为缓冲区的存储单元地址指示器，BC寄存器对作为字节计数。在查找操作中，每一步比较的结果主要反映在Z和P／V标志中，见图4.6



与数据块传送指令相仿，这组指令也有增址、减址、单步、循环之分。下面分别介绍。

5)增址单步指令CPI。比较累加器和存储单元，增量地址，减量字节计数器（Compar accumulator with memory location, Increment address, decrement byte Counter）。记作

A←(HL)，HL←HL+1

BC←BC−1

这条指令对标志的影响如下：

Z——等于1。若A=(HL)；

　　等于0，若A≠(HL)。

P／V——等于1，若BC−1≠0；

　　等于0，若BC−1=0。

其他标志

S——等于1，若比较结果为负（A−(HL)＜0），

　　等于0，若比较结果为正。

H——等于1，若第4位有借位；

　　等于0，若第4位无借位，

N——置1。

CY——不受影响

6)增址循环指令CPIR。比较累加器和存储单元，增量地址，减量字节计数器。继续执行直至比较的双方一致或者字节计数器等于零为止（ComPare accumulator with memory location, Increment address, decrement byte counter, continue until match is found or dyte counter is zeRo）。记作

A←（HL），　　HL←HL+1

BC←BC−1，

重复执行直至A=HL或BC=0

例如：HL=1111H，　　A=F3H，　　BC=0004H

且若　（1111H）=52H

（1112H）=00H

（1113H）=F3H

（1114H）=75H

（1115H）=00H

执行CPIR指令后

HL=1113H，　　BC=2，　　P/V=1，　　Z=1

又如：　A=3FH，　　则执行CPIR指令后

HL=1115H，　　BC=0000H，　　P/V=0，　　Z=0

减址型查找指令也有两条：CPD和CPDR，这里不再赘述。

五、输入输出I/O出指令组

Z80向用户提供三组I/O指令。

1．标准8080输入和输出指令：

1）INA，（n）。输入到累加器（INput to accumulator）。传送数据的源是以n=0～255作为设备号（device number）所指定的外部设备（I/O口），目的地是CPU累加器A。在执行此指令时，操作数n放在地址总线的低8位A_0至A_7，以便从最多256个口中选择一个口。与此同时，累加器的内容出现在地址总线的高8位A_8至A_{15}。然后，所选中I/O口的一字节数据被放到数据总线上并写入累加器中。

例如，累加器A=23H，I/O口（01H）=7BH

执行指令　　IN A，（01H）后

A=7BH

2）OUT（n），A。从累加器输出（OUTput from accumulator）。传送数据的源是累加器，目的地是设备号n所指定的外部设备（I/O口）。在执行此指令时，操作数n放在地址总线的低8位A_0至A_7，以便从最多256个口中选择一个口。与此同时，累加器的内容出现在地址总线的高8位A_8至A_{15}。然后，累加器内容被放到数据总线上并写入指令操作数n所选中的I/O口中。

2．寄存器间接寻址输入和输出指令：

1）IN r，（C）。输入至寄存器（INput to register）。传送数据的源是以寄存器C

内容作为设备号所选中的I／O口，目的地是寄存器r＝B，C，D，E，H，L，A。在执行此指令时，寄存器C的内容放在地址总线的低8位A_0至A_7，以便从最多256个口中选择一个I／O口。寄存器B的内容放在地址总线的高8位A_8至A_{15}。然后，所选中I／O口的一字节数据被放到数据总线上并写入指令操作数r所指定的CPU寄存器中，当一段程序中多次访问同一I／O口时，这种指令使用起来特别方便。

2）OUT（C），r。从寄存器输出（OUTput from register）。完成操作（C）←r，例如

（C）＝1FH，　　（H）＝AAH

执行指令OUT（C），H后设备号为1FH的I／O口内容为（1FH）＝AAH。

此指令的详细操作过程类似于上列指令，不再介绍。

3．数据块传送输入和输出指令：

这组指令的操作方式与数据块传送指令相似，只是作为传送一方的源或目的地不是存储器而是I／O口，这组指令也有增址、减址、单步、循环之分。下面简要介绍。

1）增址循环输入指令INIR。输入到存储器并增量指针，直至字节计数器为0（INputto memory and Increment pointer until byte counter is zeRo）。记作

（HL）←（C），　　HL←HL＋1，　　B←B－1

详细过程是：寄存器C的内容被放到地址总线的低8位A_0至A_7以便在最多256个I／O口中选择一个。字节计数器B的内容被放到地址总线的高8位A_8A至A_{15}。然后，所选中I／O口的一字节数据放到数据总线上并写入CPU。接着，HL寄存器对的内容被放到地址总线上，以便将输入的数据字节送入所指向的存储单元。此后，HL增1，字节计数器减1。若减1后B＝0，则此指令结束；若B≠0，则PC－2，重复执行此指令，如果执行此指令之前将B置为0则根据变补作减法的原理，将输入256个字节。

2）减址循环输入指令INDR。其操作记为

（HL）←（C），　　HL←HL－1，　　B←B－1

相应的一对输出指令为OTIR和OTDR，这里不再赘述。

3）增址单步输入指令INI。它的执行只完成一个操作序列

（HL）←（C），　　HL←HL＋1，　　B←B－1

此外，也有IND，OUTI，OUTD三条指令，详见附表。

4.3　数据操作指令

这类指令主要对CPU寄存器中的数据进行算术或逻辑操作，可以分为以下四组：

一、8位算术和逻辑指令组：实现加（ADD），带进位加（ADd with Carry），减（SUBtract），带进位减（SuBtract with Carry），“与”（AND），“或”（OR），“异”（eXclusvie OR），比较（ComPare），增量（INCrement），减量(DECrement）等十一种操作。上述指令的共同特点（INC和DEC指令除外）是：

- 都是针对累加器与某个指定寄存器、储存单元或立即数据之间进行操作的；
- 除了CP指令对累加器没有影响外，其他指令操作的结果均放入累加器中；
- 作为将定操作的结果，将对标志寄存器发生某种影响。

下面介绍各指令。

1）ADD。源字与累加器内容二进制加(binary ADD source word with contents of accumulator)。这条指令因操作数不同具有下列形式：

ADD / ADC } A, r　　r=A, B, C, D, E, H, L

所执行的操作是　　A←A+r(+CY)

ADD / ADC } A, n　　n=0～255(立即数)

所执行的操作是　　A←A+n(+CY)

ADD / ADC } A, (HL)

所执行的操作是　　A←A+(HL)+(CY)

ADD / ADC } A, (IX+d) / (IY+d)

所执行的操作是　　$A \leftarrow A + \begin{pmatrix}(IX+d)\\(IY+d)\end{pmatrix}(+CY)$

这里 $\begin{pmatrix}(IX+d)\\(IY+d)\end{pmatrix}$ 是变址寄存器内容 $\begin{matrix}IX\\IY\end{matrix}$ 作为基地址与偏移d相加形成的有效地址所指向的单元

2）SUB。累加器内容与源字二进制减(binary SUBtract source word from contents of accumulator)。这条指令因操作数不同具有下列形式：

指令		操作	
SUB / (SBC)	r, n, (HL), (IX+d), (IY+d)	A←A−	r, n, (HL), (IX+d), (IY+d) (−CY)

3）AND。寄存器、立即数据或存储单元和累加器逻辑“与”(logcal AND register, immediate data or memory location with accumulator)。这条指令因操作数不同具有下列形式：

指令		操作	
AND	r, n, (HL), (IX+d), (IY+d)	A←A∧	r, n, (HL), (IX+d), (IY+d)

4）OR。寄存器、立即数据或存储单元和累加器逻辑“或”；(logical OR register,

immediate bata or memory location with accumulator）。这条指令因操作数不同具有下列形式：

指令		操作	
OR	r	A←—A∨	r
	n		n
	(HL)		(HL)
	(IX+d)		(IX+d)
	(IY+d)		(IY+d)

5）XOR。寄存器、立即数据或存储单元和累加器逻辑“异”（logical eXclusive OR register, immediate data or memory location with accumulator）。这条指令因操作不同具有下列形式：

指令		操作	
XOR	r	A←—A⊕	r
	n		n
	(HL)		(HL)
	(IX+d)		(IX+d)
	(IY+d)		(IY+d)

6）CP。寄存器、立即数据或存储器单元和累加器内容相比较（Compare register, immediate data or memory location with contents of accumulator）。这条指因操作不同具有下列形式：

指令		操作	
CP	r	A—	r
	n		n
	(HL)		(HL)
	(IX+d)		(IX+d)
	(IY+d)		(IY+d)

在执行此指令时，累加器与源数据不变，比较结果反映在标志位。例如

(IX+d)=A0H, A=E3H

执行CP(IX+d)后，各标志位根据运算结果受到不同的影响，如图4.7所示

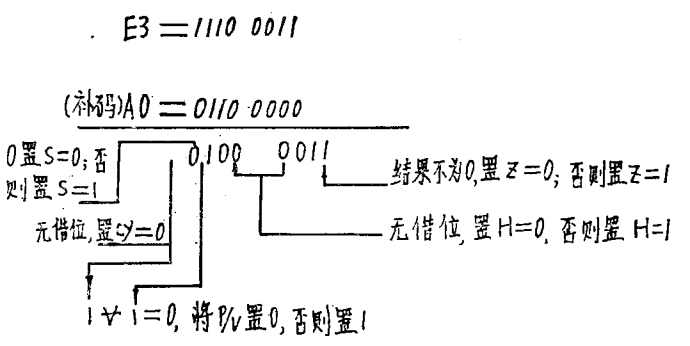


图4.7 CP指令举例

7）INC。增量寄存器或存储单元（INCrement register or memory location）。这条指令因操作数不同具有下列形式：

指令	操作
INC r	r←r+1
INC (HL)	(HL)←(HL)+1
INC (IX+d)	(IX+d)←(IX+b)+1
INC (IY+d)	(IY+d)←(IY+b)+1

S，Z，H，P／V标志将受到影响，详见附表。

8）DEC。减量寄存器或存储单元（DECrement register or memory location）。其操作数与INC指令相同，只是完成“－1”操作，详见附表。

二、16位算术指令组：这组指令一共三种，加、减法指令的操作是在HL寄存器对与某一寄存器之间进行的。增减量指令是针对某个寄存器对进行的。

1）ADDHL，ss。HL与寄存器对相加（ADD register pair to H and L）。记作

$$HL \leftarrow HL + ss$$

其中ss＝BC，DE，HL，SP

2）ADC HL，ss。HL与寄存器对带进位相加（ADd register pair with Carry to H and L）。记作

$$HL \leftarrow HL + ss + CY$$

3）SBC HL，ss。HL与寄存器对带进位减（SuBtract register pair with Carry from and pL）。记作

$$HL \leftarrow HL - ss - CY$$

4）INCss。增量指定寄存器对的内容（INCrement contents of specified register pair）。记作

$$ss \leftarrow ss + 1$$

5）DECss。减量指定寄存器对的内容（DECrement contents of specified registei Hair）。记作

$$ss \leftarrow ss - 1$$

三、通用算术指令组：这组指令的操作对象是PSW，在某些算术运算中，它们起重要辅助作用。

1）DAA。十进制调整累加器（Decimal Adjust Accumulator）。将二进制加法自动调整成BCD加法。这条指令利用进位CY标志和半进位H标志，使BCD加法得到正确结果。

在Z80的汇编语言程序中，利用DAA指令与算术运算指令组成十进制运算复合指令组ADD DAA ADC DAA，INC DAA，SUB DAA，SBC DAA，DEC DAA NEC DAA等

效于对 BCD 的源进行运算，产生BCD的结果。例如

A=39H,　　B=47H

执行ADD B

DAA

后，A=86H，而不是A=80H。

2）CPL。累加器变反（ComPLement the accumulator）。记作

$$A \leftarrow \overline{A}$$

3）NEG。累加器内容变补（NEGate contents of accumulator）。

即　　A←0—A

或写成　　$A \leftarrow \overline{A}+1$

4）CCF。进位标志变反（Complemeut Carry Flag）。记作

$$CY \leftarrow \overline{CY}$$

5）SCF。置位进位标志（Set Carry Flag）。记作

$$CY \leftarrow 1$$

四、循环移位和移位指令组：循环移位指令寄存器形成环形工作方式。其最高位直接与最低位相连。原始数据的各个位保持不变，唯一发生变化的是各个位的位置以及进位标志的内容。这类指令多用于实现乘、除法和计数等程序中。

移位指令实现普通的移位操作。利用移位操作，可以将数据在串行和并行两种形式之间进行转换，实现对数据的定标和规格化；实现对数据的拚装和分离；实现乘、除法，比较位组合格式等。

Z80向用户提供六种基本类型的九组循环移位和移位指令。简要介绍如下：

1）RLC。寄存器、累加器或存储单元向左循环移动（Rotate register, accumulator or memory location Left Circolar）。这条指令因操作数不同具有下列形式：

RLC { r, HL) } ——二字节
RLC { (IX+d), (IY+d) } ——四字节
RLCA ——一字节

所执行的操作如图4.8所示。第0位移入第1位；原来看第1位移入第2位，并依此类推。第7位移入进位标志CY和第0位。

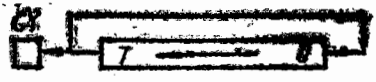


图 4.8　RLC指令的操作

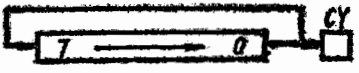


图4.9　RRC指令的操作

2）RRC。寄在器、累加器或存储单元向右循环移位(Rotater register, accumulator or memory location Right Circular)。其操作数与RLC指令相同，所执行的操作如图4.9所示。

3) RL。寄存器、累加器或存储单元通过进位位向左循环移位(Rotate register, accumulator or memory location Left thru carry)。这条指令因操作数不同具有下列形式：

RL { r, (HL) } ——二字节
RL { (IX+b), (IY+d) } ——四字节
RLA ——————— 一字节

所执行的操作如图4.10所示。

4) RR。寄存器、累加器或存储单元通过进位位向右循环移位(Rotate register, accumulator or memory location Right thru carry)。其操作数与RL指令相同，所执行的操作如图4.11所示。

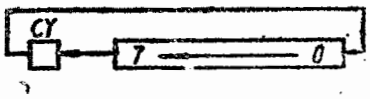


图4.10 RL指令的操作

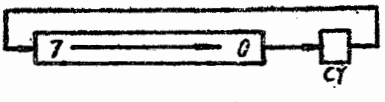


图4.11 RR指令的操作

5) RLD。一个二——十进制数字在累加器和存储单元之间向左循环移位(Rotate one BCD digit Left between the accumulator and memory location)。所执行的操作如图4.12所示。存储单元(HL)的低4位内容移入同一单元的高4位(图中①)。该高位原先的内容移入累加器的低4位(图中②)。累加器低4位原先的内容移入(HL)的低4位(图中的③)。累加器的高4位不受影响。

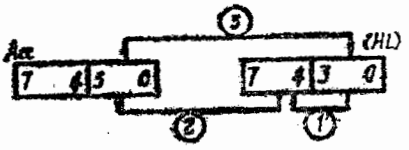


图4.12 RLD指令的操作

6) RRD。一个二——十进制数字在累加器和存储单元之间向右循环移位(Rotate one BCD digit Right between the accumulator and memory location)。所执行的操作如图4.13所示。

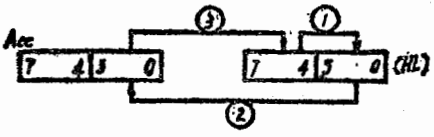


图4.13 RRD指令的操作

7) SRL。寄存器或存储单元向右逻辑移位(Shift register or memory location Right Logical)。这条指令因操作数不同具有下列形式：

SRL { r, (HL), (IX+d), (IY+d) }

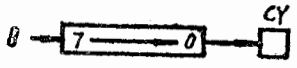


图4.14 SRL指令的操作

所执行的操作如图4.14所示。数据向右移1位，留下的最高位被清零。

8 ）SLA。寄存器或存储单元向左算术移位（Shift register or memory location Left Arithmetic）。此指令的操作数与SRL指令相同，所执行的操作如图4.15所示。可见，这实质上是一条向左的逻辑移位指令。

图4.15 SLA指令的操作

9 ）SRA。寄存器或存储单元向右算术移位（Shift register or memory location Right Arithmetic）。此指令的操作数与SLA指令相同，其操作如图4.16所示。在移位时保留数据最高位（符号位），并将其依次传送到下一位。这种操作也叫做符号延展（Sign extension），如图4.17。

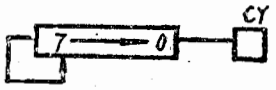


图4.16 SRA指令的操作

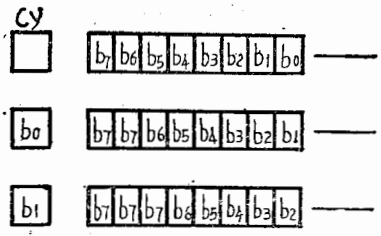


图4.17 SRA指令的操作

4.4 程序控制指令

这类指令控制和改变程序的正常进程。包括：

1、无条件转移指令（unconditional JumP）。该指令通过对程序计数器的操作，使正在执行的指令序列转向新的地址继续执行下去。

2、条件转移指令（JumP on condition）。该指令使CPU能重复执行一个指令序列，测试字符，识别错误，检查外部设备的状态等。条件转移指令是使CPU能够进行判定的主要手段。可以说CPU的智能化很大程度上依赖于这条指令。

3、子程序操作指令。它也是要实现程序的转移。但它与普通转移指令的区别在于执行所转移的程序后还需能从原来的断点处返回程序流程。下面介绍各类程序控制指令。

一、转移指令组

转移指令又可分为以下四组：

1、标准8080转移指令。除所用助记符不同外，其功能完全雷同于8080A的转移指令。

1 ）$\underset{\text{label}}{\text{JP}}^{\text{nn}}$。转移到由操作数所指定的指令（JumP to the instruction identified in the operand）。这是三字节指令，第一字节规定操作码；二、三字节规定操作数，它可以是一个16位地址nn也可以是一个表示16位地址的标号。这条指令所执行的操作为

PC←（第三字节）（第二字节）

使程序转到操作数所指定的地址。

2）JPcc, $\begin{matrix}nn\\label\end{matrix}$。如果条件满足，则转移到操作数所指定的地址（JumP to address dentified in the operand if condition is satisfied）。

cc是条件，包括：

cc=NZ——非零（NonZero），	标志Z=0
cc=Z ——零（Zero），	标志Z=1
cc=NC——没有进位（NonCarry），	标志CY=0
cc=C ——有进位（Carry），	标志CY=1
cc=PO——奇偶性奇（Parity Odd），	标志P/V=0
cc=PE——奇偶性偶（Parity Even），	标志P/V=1
cc=P ——符号为正（sign Positive），	标志S=0
cc=N ——符号为负（sign Negative），	标志S=1

2、变址转移。

JP $\begin{cases}(HL)\\(IX)\\(IY)\end{cases}$。转移到由16位寄存器内容所指定的地址（Jump to address sepcified by contents of 16 bit register）。记作

$$PC \leftarrow \begin{cases}HL\\IX\\IY\end{cases}$$

寄存器对HL，IX，IY都是用来贮存指针的。又由于它们可以进行+1/-1操作，因此利用它们的内容作为可变转移地址十分便利。

3．二字节转移。

二字节转移指令多用于“长距离”转移。如果转移的目标就在现行指令附近，一般采用相对于现行指令的二字节转移指令，其中也包括无条件和有条件转移两种。

1）JR e-2。相对于程序计数器现在的内容转移（Jump Relative to present contents of program counter）。操作数中的e是偏移。这里要注意，偏移e是从相对转移指令本身的第一字节（操作码）开始算起的，如图4.18所示。转移的范围是从(e-2)min+2=-128+2=-126个字节到(e-2)max+2=+127+2=129个字节。

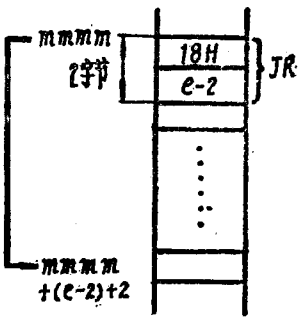


图 4.18 JR指令的操作

2）JR $\begin{matrix}C\\NC\\Z\\NZ\end{matrix}$, e-2。若操作数中所规定的条件得到满足，相对于程序计数器现在的内容转移(Jump Relative to present contents of program counter if conditions specified in the operand are satisfied)。相对转移的条件是C(CY=1)，NC(CY=0)，Z(Z=1)和NZ(Z=0)。

4．循环控制转移指令。

在实际问题中，往往需要多次重复执行一段程序，直至某一条件得到满足为止。这一条件可以由一个预置的计数值逐次减1实现。当计数值为0时，条件得到满足，程序循环结束。为了便于实现这类操作，Z80提供一条专用的循环控制转移指令：

DJNZ e— 2．寄存器B 减1，若寄存器B不为零，相对于程序计数器现在的内容转移(Decrement register B, Jump relative to presnt contents of program counter if register B Not Zero)。

DJNZ的典型使用方式如图4.19所示。

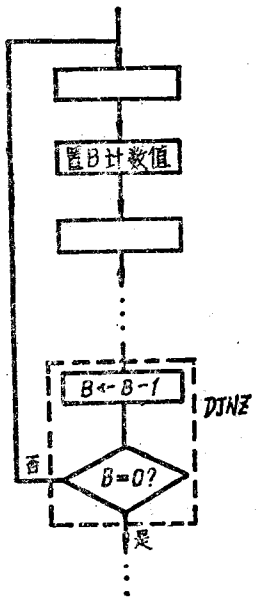


图 4.19 DJNZ指令的操作

二、子程序调用和返回指令组

调用一是种转移，它将程序计数器原先的内容保护起来，以便在执行完所调用的子程序之后，从断点返回原来的程序流程。一般情况下，调用和返回指令是成对编排的，也分为无条件和条件调用／返回两类。下面简要介绍。

1）CALL $\overset{nn}{label}$。调用以nn或标号为起始地址的子程序（CALL the subroutine entered with nn or label）。记作

$$(SP-1)\leftarrow PCH$$
$$(SP-2)\leftarrow PCL$$
$$PC\leftarrow nn$$

此指令将PC的当前内容压入栈内，然后将CALL指令的操作数段所给出的子程序入口地址装入PC，以便转去执行子程序。

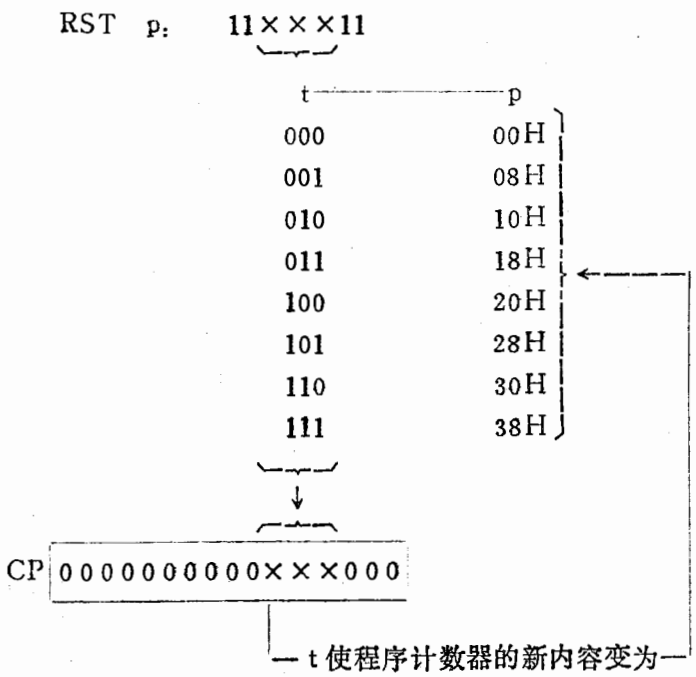
3．RET。从子程序返回（RETurn from subroutine）。记作

$$PCL\leftarrow(SP)$$
$$PC_H\leftarrow(SP+1)$$
$$SP\leftarrow(SP+2)$$

此指令将原先进栈的断点地址弹回计数器而返回原程序流程。

调用和返回指令也可按一定测试条件进行。相应的指令为CALL cc，nn和RET，详见附表。此外，还有一条特殊的一节调用指令，常常由外部设备硬件来提供，即

RST p。重新启动（ReStarT）。它是一个一字节子程序调用指令，最常用于中断的场合，其结果代码为



能够形成八个中断服务程序入口地址。

执行RST p指令时，先将PC保护进栈，然后将p装入PC_L，PC_H为00，转向中断服务程序。

4.5 CPU控制和位操作指令

一、CPU中断控制指令组

中断的目的是使外部设备能够以一定的方式暂停CPU的操作，并迫使CPU执行一个服务子程序。待子程序完成后，CPU就返回原先的程序流程。Z80提供三组中断控制指令，现分述如下：

1、中断允许／禁止。Z80具有两种中断输入，软件可屏蔽中断INT（maskadle INTerrupt）和不可屏蔽中断NMI（Non-Maskable Interrupt）。NMI是为那些必须要求响应的很重要的功能服务的。INT则能够由程序员有选择地允许或禁止。其控制是通过对中断许允触发器IFF（Interrupt Flip-Flop）的操作实现的。在Z80—CPU中，有两个这样的触发器FF1和IFF2。IFF1的状态用来禁止中断，而IFF2用来暂存IFF1的内容。

1）EI。允许（开）中断（Enable Interrupt）。此指令会使IFF1和IFF2都进入置位状态（状态1）。当CPU接受中断后，IFF1和IFF2自动复位，从而禁止后继的中断申请，直至程序中出现一个新的EI指令为止。

2）DI。禁止（关）中断（Disable Interrupt）。当执行此指令后，可屏蔽中断请求被禁止，I／O设备向CPU发出的$\overline{INT}$输入无效。上述状态一直维持到出现EI指令为止。

2、中断返回。

1）RETN。从不可屏蔽中断返回（RETurn from Non-maskable interrupt）。用于不可屏蔽中断服务程序结束时，此指令执行无条件返回，其功能与RET指令相同。就是说，

先前进栈的PC值从栈中弹出，然后SP←SP＋2。另外，此指令恢复中断允许逻辑，即IFF2的状态被传送到IFF1，使之恢复到接受NMI之前所具有的中断状态。

2）RETI。从中断返回（RETurn from Interrupt）。此指令与RET指令的功能完全相同，只是Z80外部接口器件将识别此指令，并用它标示现行中断服务程序已经执行完毕。

3、中断方式选择，可屏蔽中断具有方式0，1，2三种方式：

1）方式0（MODE 0）。执行“IM 0（中断方式0，InterruPt Mode 0）”指令，进入方式0的准备条件是：$\overline{\text{RESET}}$信号已加至CPU，使后者完成初始化或IM 0指令已执行完毕；IFF1处于置位状态，即已开中断，这时如果没有总线请求或不可屏蔽中断请求，则此中断请求将被接受。CPU将发出一个$\overline{M_1}$和$\overline{IOKQ}$信号，然后进入等待状态。

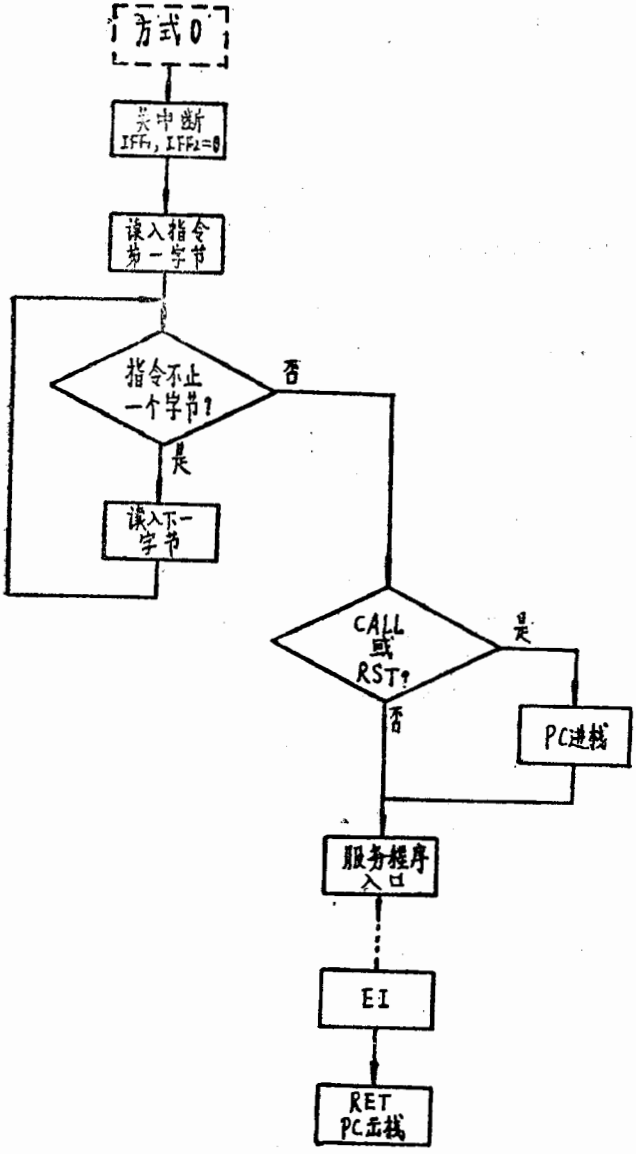


图4.20 中断方式0的操作

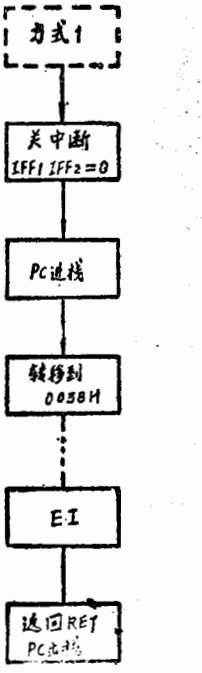


图4.21 中断方式1的操作

图4.22 中断方式2的操作

请求中断的外部设备应能识别$\overline{W_1}$和$\overline{IORQ}$的同时出现，并将一条指令放在数据总线上去。一般情况下，所放的是RST或CALL指令。这两条指令都自动将PC保护行栈，然后转到中断服务程序的入口。

一但中断开始，所有后继的中断都被禁止，IFF1和IFF2自动复位的。这时，程序中应安排一条EI指令，以便充许优先级较高的其他备设发出中断请求。在中断服务程序结尾应安排RET指令，使中断结束返回原先的程序流程。方式0的操作如图4.20所示。

2）方式1（MODE1）。执行“IM1（中断方式1，Inteffupt Mode1)”指令。CPU响应中断时自动执行一条RST指令，转向单元0038H.可见，这种中断方式所需外部硬件最少。中断方式1的操作见图4.21

3、方式2（MODE2）。执行“IM2(中断方式2，Interrupt Mobe2)”指令。它也叫做矢量的中断方式。中断矢量的低8位由请求中断的外部设备提供。方式2的操作见图4.22。

二、CPU其他控制指令组

1）NOP。不操作（NO oPeration）。执行此指令时除程序计数器增1和动态存储器刷新外，CPU不实现任何操作。

2）HALT。暂停(HALT)。它使CPU停止操作，重复执行NOP指令，直至接到中断请求或复位命令后才重新开始进入程序流程。

三、位操作指令组

这类指令使我们能对寄存器或存储单元中的指定位进行测试或置位、复位操作。例如在信号处理的应用中，一个单独信号往往作为一个二进制位的实体存在，这时位操作就具有特别重要的意义。Z80的位操作指令有以下几组：

1）BIT b, r
b, (HL)
b, (IX+d)
b, (IY+d)

测试寄序器或存储单元中的一位，结果放在零标志中(test BIT in register or memory location, the result is put in zero flag)。指令中第二个操作数指定被测试位所在的源字节，第一个操作数指定测试位的位号（b的代码为3位）。

2、SET b, r
b, (HL)
b, (IX+d)
b, (IY+d)

置位寄存器或存储单元中的指定位b（SET bit b of register or memony location）记作

rb ← 1
(HL) b ← 1
(IX+d)b ← 1

(IY+d)b ← 1

3) RES b, r

b, (HL)

b, (IX+d)

b, (IY+d)

复位寄存器或存储单元中的指定位 b (RESet bit b of register or memory location)。

第五章　并行输入/输出接口芯片
Z80 PIO

5.1 概　　述

一个微型计算机，除CPU和RAM、ROM外，尚需要输入／输出（I／O）芯片，才能使系统正式进行操作。这些I／O芯片可分为两大类；

1.通用I／O芯片。其中主要有并行I／O、串行I／O、DMA、可编程中断控制器、计数／定时器等等。

2.专用I／O芯片。其中主要有软磁盘控制器、CRT控制器、键盘接口、键盘／显示接口、数据编码等等。

Z80—PIO（以下简称为PIO）是一个可以用程序来变更其工作方式的器件，它具有16根输入／输出（又称为I／O）线。这些I／O线可分成两个8位的I／O口，每个I／O口配有二根联络线（handshaking），用以控制数据的传送。

PIO能为外部设备与Z80—CPU之间提供一个TTL兼容的接口。利用CPU的指令变更PIO的工作方式，从而能与各种各样的外部设备——大多数的键盘、纸带读入器和凿孔机、打印机、PROM写入器等——相接而不需其他外加电路。

当使用PIO时，外部设备与CPU之间的数据传送均在IM_2（方式2）中断控制下实现。

5.2 PIO的方框图及引脚

PIO的方框图示于图5.1。每个口的I／O逻辑是由六个寄存器和“联络”控制逻辑所组成，如图5.2所示。六个寄存器为：8位数据输入寄存器；8位数据输出寄存器；2位方式控制寄存器；8位屏蔽寄存器；8位I／O选择寄存器；2位屏蔽控制寄存器。

前两种寄存器系统称为数据寄存器，后四种寄存器统称为控制寄存器。这些控制寄存器的作用将在5.3节中叙述。

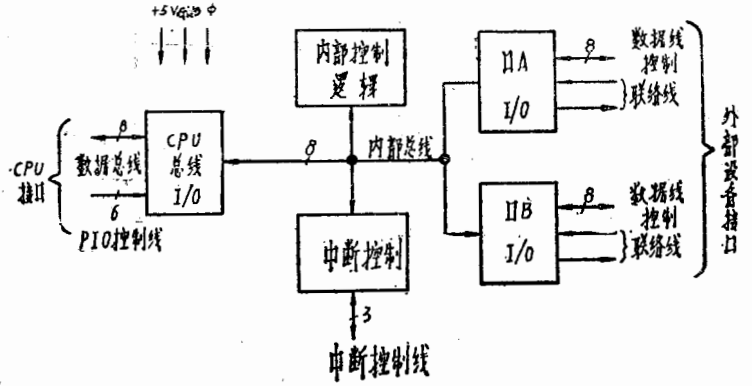
PIO的引脚布置如图5.3所示。这些引脚的作用如下：

1.D7—D0为数据总线（双向，三态）。这个总线用来在CPU和PIO之间传送数据和命令。

2.$\overline{CE}$为芯片允许（输入，低电平有效）。只有当此信号为低电平时，才允许CPU访问该PIO芯片。

3.B／$\overline{A}$为选择口A或口B（输入）。当该信号为低电平时，选中口A；当为高电平时，选中口B。通常将此引脚接至CPU地址总线的A0位。

4.C／$\overline{D}$为选择数据寄存器或控制寄存器（输入）当该信号为低电平时，访问数据寄存



←图5.1 PIO的框图

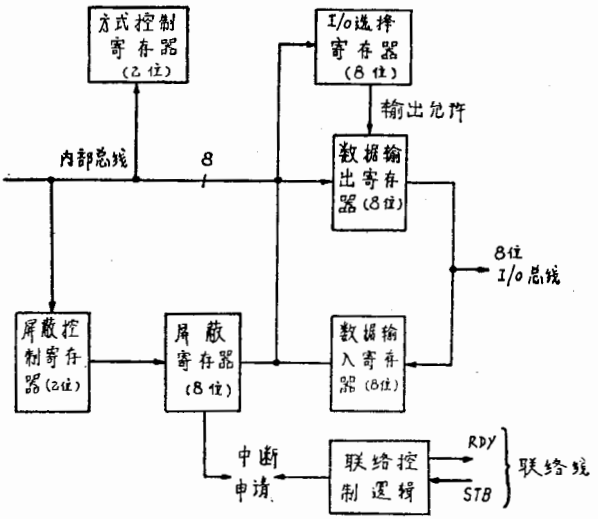
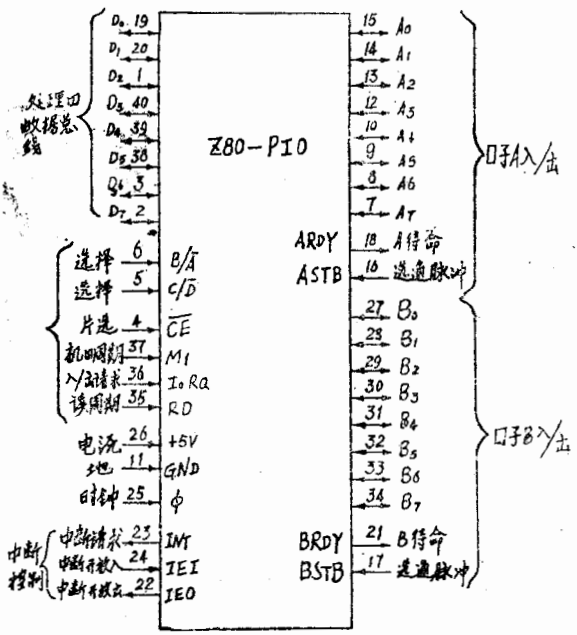


图5.2 口的I/O框图→



←图5.3 PIO的引脚布置图

器，即数据总线被用来传送数据；当该信号为高电平时，访问控制寄存器，即CPU通过数据总线向PIO写入的信息是一个命令。通常将此引脚接至CPU地址总线的A1位。

从以上三个信号的含义，可将PIO的选择逻辑归纳如表5.1所示。

表5.1　　PIO的选择逻辑表

引脚			选中单元	单元地址
$\overline{CE}$(*)	B/A(AO)	C/D(A_1)		
0(A7—A2=100000)	0	0	口A的数据寄存器	80H
0(A7—A2=100000)	0	1	口A的控制寄存器	82H
0(A7—A2=100000)	1	0	口B的数据寄存器	81H
0(A7—A2=100000)	1	1	口B的控制寄存器	83H
C1(A7—A2≠100000)	×	×	芯片未被选中	

(*)CMC—80的PIO1是按此地址连接的。不同的微型机系统或不同的PIO芯片，A7—A2可为不同值。

表5.1中×表示可为任意值，80H为十六进制数80。如果地址总线A7—A2 = 100000时，E被译码成逻辑0。如果A7—A2为其他值时，$\overline{CE}$ = 1。在此情况下，PIO占有四个外部设备地址80H，81H，82H，83H。

5.$\overline{M1}$ 为CPU的取指令机器周期信号（输入，低电平有效）。

6. $\overline{IORQ}$ 为CPU的I/O请求信号（输入，低电平有效）。

7.$\overline{RD}$ 为CPU的读周期状态（输入，低电平有效）。

上述三种控制信号的作用如表5.2所示：

表5.2　　PIO控制信号的作用

引脚			功能解释
$\overline{M1}$	$\overline{IORQ}$	$\overline{RD}$	
0	0	0	没有作用
0	0	1	中断响应
0	1	0	检查中断服务程序是否结束
0	1	1	复　位
1	0	0	CPU从PIO读出
1	0	1	从CPU写入PIO
1	1	0	没有作用
1	1	1	没有作用

8.IEI为中断允许输入信号（输入，高电平有效）。当用了一个以上的中断源器件时，本信号被用来构成优先权中断链，若本引脚为高电平，表示CPU目前没有为优先权（比本芯片）更高的其他器件服务。此时允许芯片向CPU请求中断。

9.IEO为中断允许输出信号（输出，高电平有效）。只有IEI为高电平，且CPU没有为本芯片的中断服务时，本信号才为高电平。若CPU为本芯片或中断优先权更高的芯片的中断服务时，本信号均为低电平，从而阻止优先权较低的器件发出中断请求。

10.$\overline{\text{INT}}$为中断请求（输出，漏极开路，低电平有效）。当PIO向CPU发出中断请求时$\overline{\text{INT}}$有效。

下面顺便介绍链形中断的处理过程。图5.4表示了典型的嵌袋中断的序列。在此序列中，口2A首先发中断请求（在同一芯片中，口A的中断优先权高于口B）并被接受。当口2A正在被处理中，一个优先权更高的口IB发出中断请求，并被接受。接着对优先权较高的口1B的服务程序执行完毕，并且执行一条RETI指令，通知此口这一服务程序已结束，口IB的IEO端的电位变高。接着完成优先权较低的2A口的服务程序。

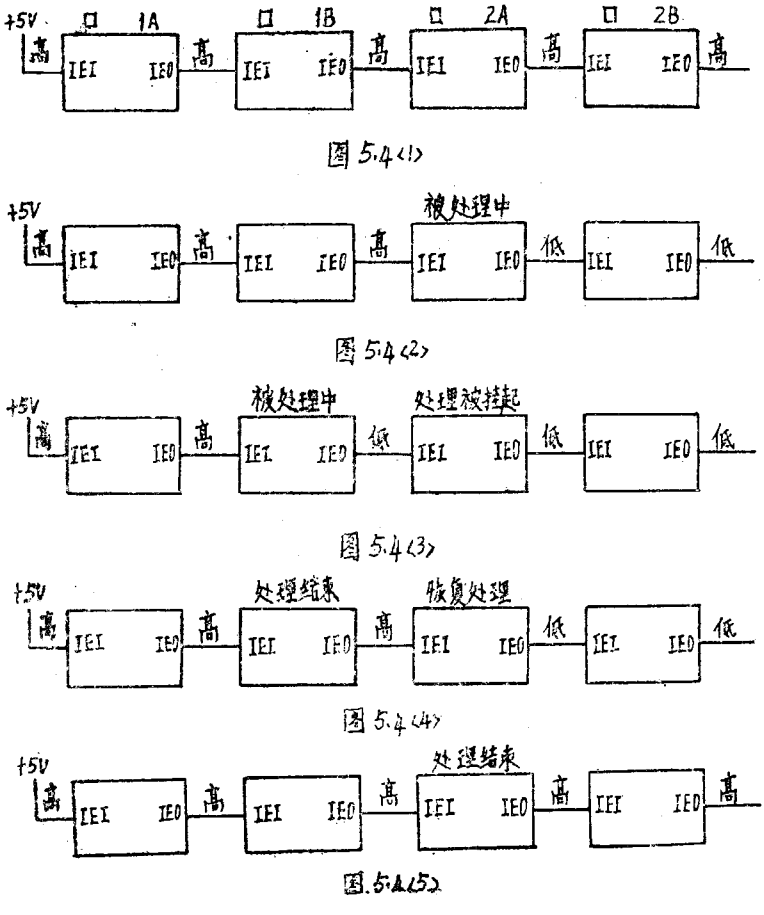


图5.4 链形中断处理

11.PA7—PA0为口A的总线（双向，三态）。用来在口A和外部设备之间传送数据和控制信息。

12.$\overline{\text{A STB}}$为（来自外部设备的）口A选通脉冲（输入，低电平有效）。此信号的作用与口A的工作方式有关。将在下节介绍。

13.ARDY为寄存器A待命（输出，高电平有效）。此信号的作用与口A的工作方式有

关。将在下节介绍。

14.PB7—PB0为口B的总线（双向，三态）。用来在口B和外部设备之间传送数据和控制信息。口B总线能在1.5伏下提供1.5毫安电流，以便驱动复合晶体管。

15.$\overline{\text{B STB}}$ 为（来自外部设备的）口 B 选通脉冲（输入，低电平有效）。此信号的作用与口B和口A的工作方式有关，将在下节介绍。

16.B RDY为寄存器B待命（输出，高电平有效）。此信号的作用与口 B 和口 A 的工作方式有关，将在下节介绍。

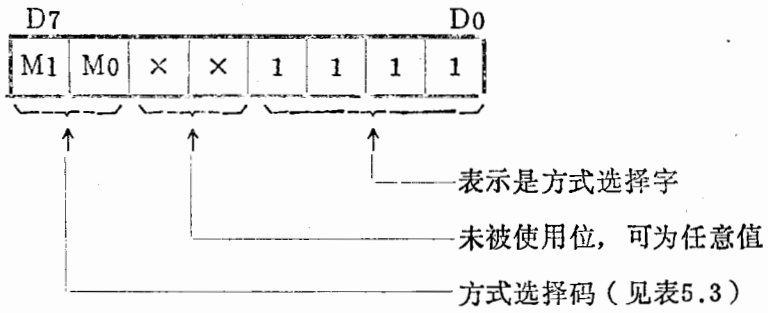
17.其他：ϕ为时钟；＋5 V为电源；GND为地。

5.3 PIO的操作说明

每片PIO占有四个外部设备地址，即有四个可寻址单元能被访问，如图 5.5 所示。图中的设备地址采用CMC—80微型电脑中所使用的 PIO1 的具体值。

PIO有两个口：口A和口B。每个口有两个可寻址单元，一是控制寄存器，另一个是数据寄存器。写入控制寄存器的控制字有六种，其中有三种仅用于工作方式 3，将在方式 3 中介绍。下面介绍另外三种（也是重要的）控制字的格式及含义。由于这些控制字都是写入同一地址的控制寄存器，因而需要规定一些特征来区别这些控制字。

1.方式选择字



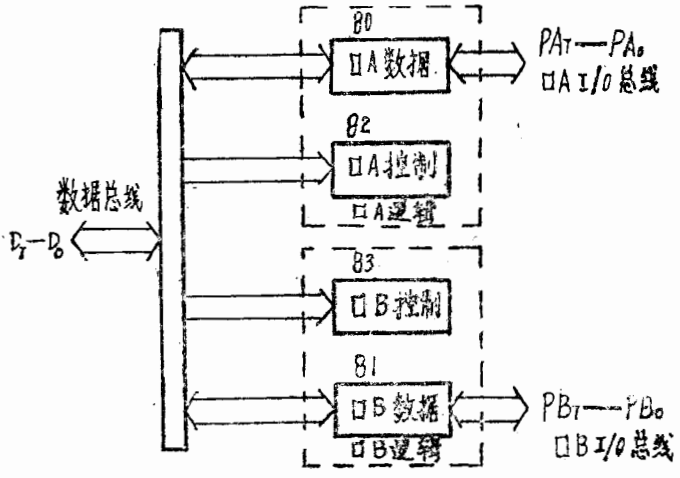


图5.5 PIO有四个可寻址单元

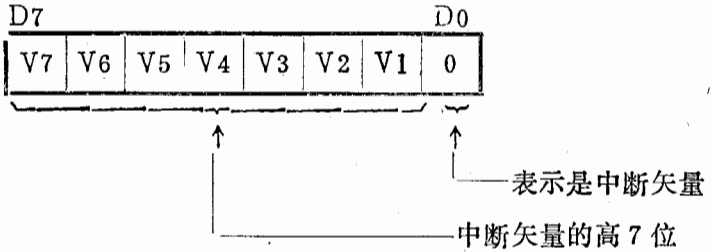
PIO有四种操作方式，如表5.3所示。具体操作方式由方式选择字中的M1M0来决定。

表5.3　　PIO的操作方式

M1	M0	操作方式	说明
0	0	方式 0	带连络的输出
0	1	方式 1	带连络的输入
1	0	方式 2	带连络的双向I/O*
1	1	方式 3	位控操作方式

*只有口A能工作在方式2，此时口B的联络线被口A所占用，因而口B只能工作在方式3。详细解释见后。

2.中断矢量



假设CPU中断矢量寄存器I＝23H，又设口A的V7－V1＝0100000，则PIO的中断矢量＝40H，两者合成一个完整的中断矢量2340H。如表5.4所示，由2340和2341单元中找到中断服务程序的起始地址2200H。因为中断服务的起始地址占用两个单元，因而中断矢量可以安排成都是偶数，如2340H，2342H，2344H等，即它的最低位必然为0。由此可以与其他控制字相区别。

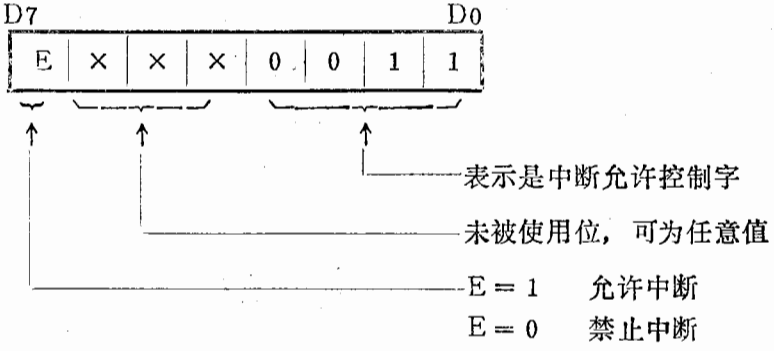
表5.4　中断服务程序起始地址表(举例)

2340	0	0
	2	2
2342	1	B
	2	2
2344		

口A中断服务程序的起始地址

口B中断服务程序的起始地址

3.中断允许控制字



现分别介绍四种操作方式

一、方式0—输出

以口A为例，结合下面的初始化程序，说明PIO如何被设定成这种工作方式并进行工作。

```
LD    A, 23H        设定中断矢量
LD    I, A
LD    A, 40
OUT   (82), A
LD    A, 0FH        设定为工作方式0
OUT   (82), A
LD    A, 83H        允许PIO请求中断
OUT   (82), A
IM    2             设置中断方式IM2
EI                  开中断
LD    A, (HL)
OUT   (80), A       向口A输出一个数据
```

其过程如下：

1.设定中矢量为2340H。

2.设定口A为操作方式0。

3.对PIO口A开中断，也就是使其中断允许触发器置位。

4.设定CPU的中断方式为IM2，并开CPU的中断。

5.CPU执行一条输出指令OUT(80)，A即开始方式0的输出周期，如图5.6所示，CPU执行输出指令期间。对PIO产生—$\overline{WR}$*脉冲，并用它将累加器A中的数据，锁存在口A的数据输出寄存器中。之后$\overline{WR}$*电平变高，在下一个ϕ下降沿使A RDY电平变高，告知外部设备口A中已有数据可供使用。这一段时间如图中T1所示。此后A RDY将保持有效，

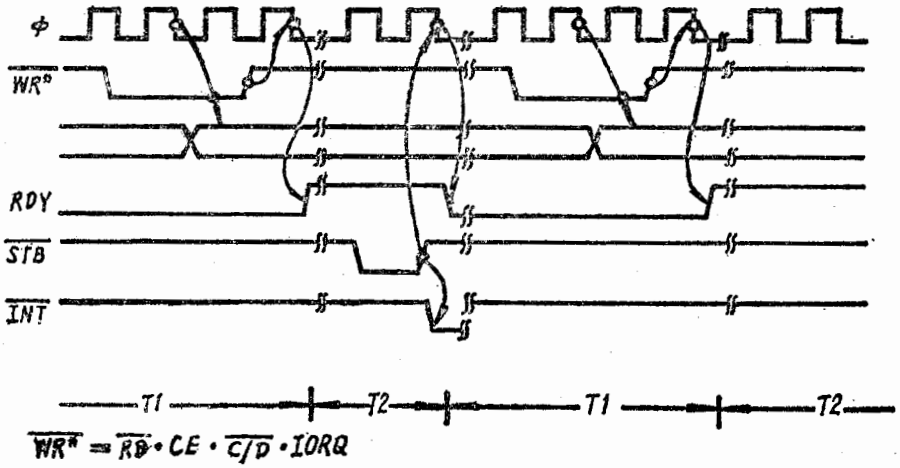


图 5.6 方式0的时间图

CPU转入正常程序流程。当外部设备在$\overline{\text{A STB}}$线上发出一个负脉冲，将口A中数据取走后（即$\overline{\text{A STB}}$出现一个上升沿），自动产生中断请求，即INT电平变低。这一时间段如图中 T2 所示，此后，如果CPU响应此中断请求，口A将中断矢量40与CPU1寄存器中的23 合成一个完整的中断矢量2340H，在2340和2341单元中取出口A中断服务程序的起始地址，例如，如表5.4所示为2200。接着执行起始地址为2200的中断服务程序，CPU进行相应的操作，并又向口A 输出一个数据，此后的过程与上相同。

由上可见，外部设备与CPU之间的全部数据传送是在中断控制下实现的。尽管这种传送周期可以很长，但是它占用CPU的时间很少，因而很少影响CPU的正常程序流程的进行。

二、方式1—输入

以口A为例，图5.7示出了一个输入周期的时间图。这一周期是在CPU执行了一次读数之后，由外部设备利用$\overline{\text{A STB}}$来启动的。当$\overline{\text{A STB}}$呈低电平时，从外部设备将数据装入口A内的数据输入寄存器，$\overline{\text{A STB}}$的上升沿将使INT电平变低。假如这时中断允许触发器已被置位，且这一器件的中断请求是优先权最高的，则下φ的下降沿将使A RDY电平变低，表示数据输入寄存器已满，禁止再送数来，之后，CPU在执行中断服务程序过程中，从口A读取数据（发生在$\overline{\text{RD}}$ *上出现一个负脉冲时），在$\overline{\text{RD}}$ *信号上升沿的下一个φ的下降沿时，使A RDY又变高，从而允许外部设备将新的数据装入口A。

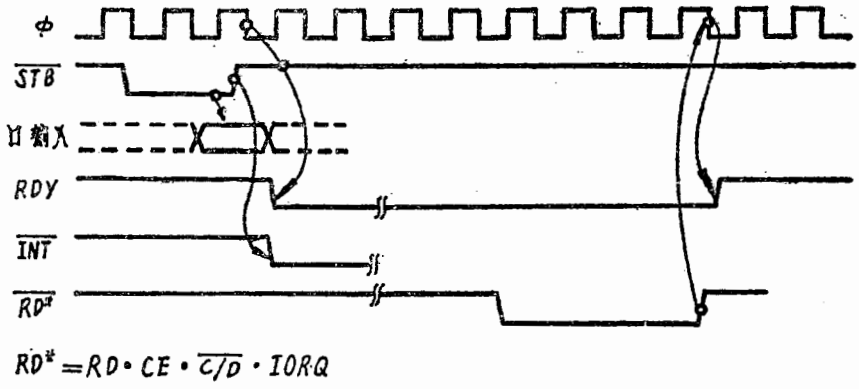


图 5.7　方式1的时间图

三、方式2—双向操作

这一方式只有口A可使用。它只不过是方式0和方式1的组合而已，但必须使用四根联络线，因而除了使用口A的二根联络线$\overline{\text{A STB}}$和A RDY作为输出的联络控制，还需要占用口B的二根联络线$\overline{\text{B STB}}$和B RDY作为口A输入的联络控制用。在这种情况下，口B只能工作在方式3，因为方式3不需使用联络线（见后）。

图5.8示出了这一方式的时间图。它与以上对方式0和方式1所介绍的情况几乎是相同的。其间的差别为：在方式2中，只有当$\overline{\text{A STB}}$为低电平时，才允许数据送到口A的总线上，值得注意的是，必须将口A和口B的中断允许触发器置位（即开中断），才能实现中断

驱动的双向传送。

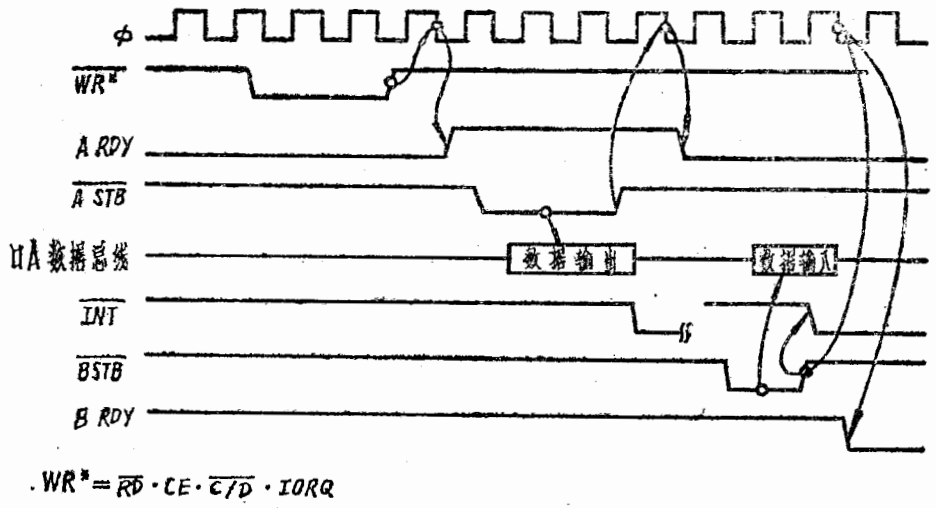


图 5.8 方式2的时间图

四、方式3—位控方式（以口B为例）

这一控制方式并不使用联络信号。在这方式中，可由程序规定口的某些线为输出线，另一些线为输入线。并可由程序规定，在外部设备中出现指定状态的情况时向 CPU 发出中断请求。

使用这一控制方式，可在任何时刻执行常规的口子写入或读出，在读PIO时，送回CPU的数据由两部分组成：一部分是数据输出寄存器中相应于被指定为输出位的内容；另一部分是数据输入寄存器中相应于被指定为输入位的内容。

现结合下面的初始化程序来说明PIO如何被设定成这种工作方式并进行工作。

```
LD    A, 23H      设定中断矢量
LD    I, A
LD    A, CFH      设定方式3。下面送入的控制字不论其格式如何,必然是一个
OUT   (83), A     I/O选择字
LD    A, 29H      送入I/O选择字
OUT   (83), A
LD    A, 42H      设定中断矢量低字节
OUT   (83), A
LD    A, B7H      设定中断控制字
OUT   (83), A
LD    A, D6H      设定屏蔽字
OUT   (83), A
IM    2
EI
```

下面对只在方式3中使用的控制字进行解释：

1.I/O选择字。此字紧跟在设定为工作方式3的控制字之后写入，它被锁存在前述的I/O选择寄存器中。

D7							D0
I/O7	I/O6	I/O5	I/O4	I/O3	I/O2	I/O1	I/O0

I/O=1　该位为输入

I/O=0　该位为输出

2.中断控制字。该字的D6D5锁存在屏蔽控制寄存器中。

D7				D0			
				0	1	1	1

表示此为中断控制字

D4=1　表示下一个送入的控制字为中断屏蔽字

D4=0　其他情况

D5=1　被监视的信号高电平有效

D5=0　被监视的信号低电平有效

D6=1　对被监视的信号进行“与”操作

D6=0　对被监视的信号进行“或”操作

D7=1　开中断

D7=0　关中断

3.屏蔽字。此字紧跟在中断屏蔽字（且D4=1）之后写入，它被锁存在屏蔽寄存器中。

D7							D0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

MB=0　该位受监视

MB=1　该位的监视被屏蔽

五、程序设定PIO工作方式的步骤

1.中断矢量

D7							D0
V7	V6	V5	V4	V3	V2	V1	0

表示此控制字为中断矢量

2.设置方式

D7							D0
M1	M0	×	×	1	1	1	1

表示此为方式控制字

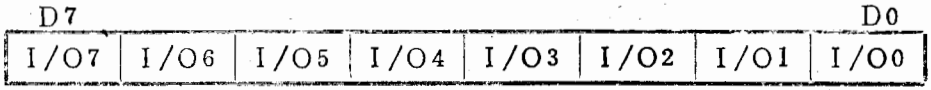
00——输出

01——输入

10——双向

11——位控

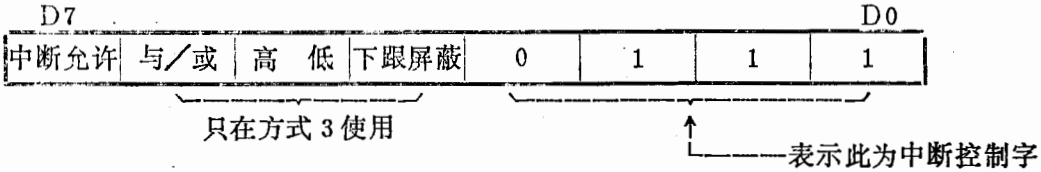
在选择方式3（D7D6=11）时下一控制字必须是I/O选择字：



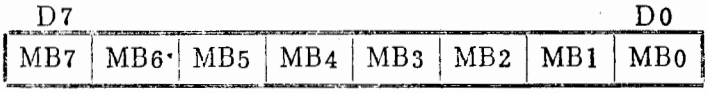
I/O=1　　使该位为输入

I/O=0　　使该位为输出

3.设置中断控制



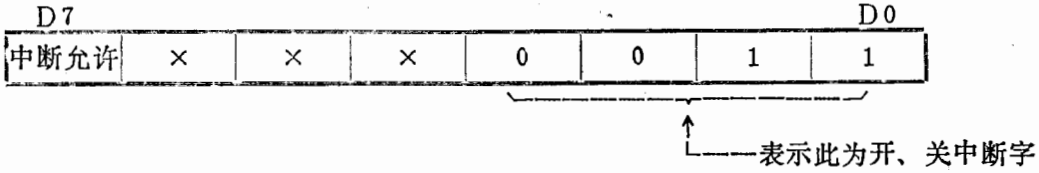
若下跟屏蔽位（D4）为1，写入口的下一个控制字必须是屏蔽字：



MB=0　　该位受监视

MB=1　　该位的监视被屏蔽

此外，口的中断允许触发器可以用下列命令来置1或置0，而不会修改中断控制字的其它部分。



5.4 PIO的使用

本节以位控方式的应用为例，说明PIO的使用及其电路连接。

图5.9示出了一个典型的控制方式的应用。假设要监视一个工业加工过程，任何不正常工作情况的发生，都将报告以Z80—CPO为中心所组成的控制系统。这过程的控制字具有下列内容：

1.方式控制字

1	1	×	×	1	1	1	1

选用方式3，下一个必为I/O选择字：

2. I/O选择字

0	0	1	0	1	0	0	1

PB0、PB3、PB5作为输入线。

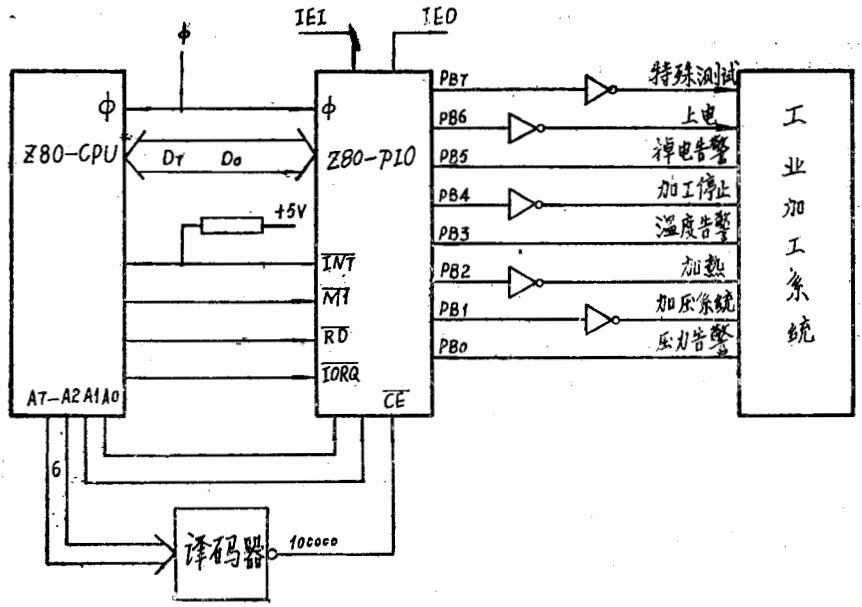


图5.9 位控方式的电路连接

3.中断矢量

0	1	0	0	0	0	1	0

4.中断控制字

1	0	1	1	0	1	1	1

表示对受监视的输入线进行“或”操作，并认为高电平有效。下面写入的控制字必须是一个屏蔽字。

5.屏蔽字、

1	1	0	1	0	1	1	0

表示PB0、PB3和PB5线受监视。若其中任一根线为高电平都将产生中断请求。要求CPU进行告警处理。

此外，口B通过其输出线，向工业加工系统发出命令，指示它进行相应的操作。

第六章　计数器/定时器芯片

Z80—CTC

6.1 概　述

在微型计算机化（Microcomputer—based）仪器和设备中，经常有一些定时和计数的要求，例如要求微型机驱动步进马达一类的电力机械，即要求微型计算机产生具有数毫秒脉宽的脉冲。微型计算机实现这类要求的具体方法有下列三种：

1.利用等待循环（Wait loop）。这种方法的缺点是束缚了CPU，使CPU 在等待循环中不能为其他事件服务。

2.利用单冲电路（One shot）。如图6.1所示，通过微型机的输出口产生正（或负）跳变，使单稳电路产生一脉冲。这种方法的缺点是脉冲宽度与单冲电路中的电路时间常数RC（即图中的电阻与电容的乘积）有关，一经设定，不能由程序来更改。另外，单冲电路使微型计算机设备的调试带来很大的麻烦。

3.使用可编程序的计数/定时器，现代流行的微型计算机中可包有这样的芯片，它们可由程序来设定脉冲的个数、频率、波形等。

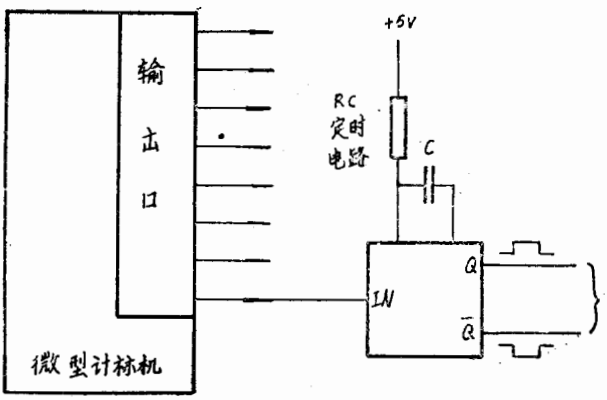


图 6.1　由单冲电路来产生脉冲

Z80—CTC计数器／定时器（以下简称CTC）是一种具有四个独立通道的可编程序器件。它有下列两个功能：

1.定时功能。它能定时地发出脉冲（并能在发脉冲的同时请求中断）。脉冲的时间间隔、脉冲序列的起始终了、产生脉冲的方式以及脉冲的个数均可由程序来设定。

2.计算功能。它能对外界事件进行计数，当达到程序规定的数值时，向CPU 请求中断（并可输出一脉冲）。

6.2 CTC的方框图及引脚

CTC的方框图示于图6.2。它的每个通道都具有各自的中断矢量。0号通道具有最高的中断优先级。每个通道的方框图如图6.3所示。它由两个8位的寄存器、两个8位的计数器以及控制逻辑线路所组成。两个寄存器是时间常数寄存器和通道控制寄存器。两个计数器是（CPU可访问读的）减1计数器和定标器（仅用于定时器工作式）。它们的功能将在下节操作说明中介绍。

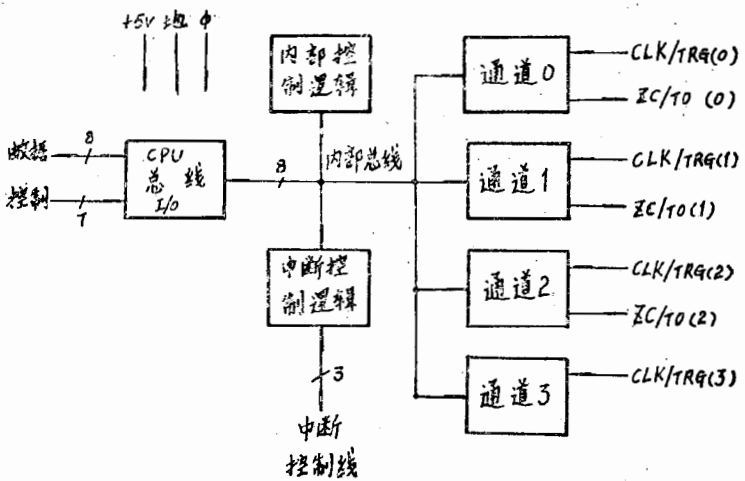


图 6.2 CTC的框图

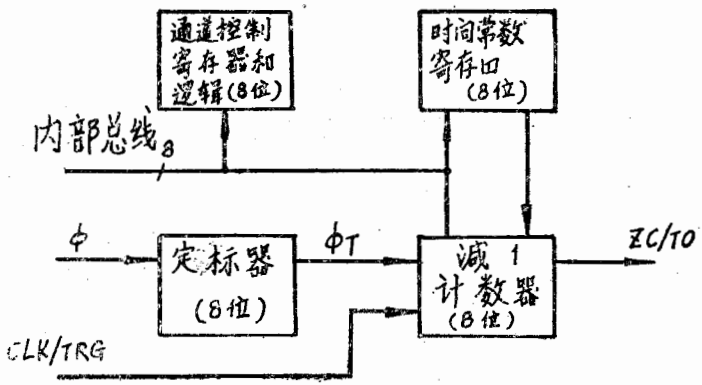


图 6.3 CTC通道的框图

CTC的引脚布置图6.4所示。这些引脚的作用如下：

1. D7—D0为Z80—CPU数据总线（双向，三态）。此总线用来在CPU和CTC之间传送数据和命令。

2. $\overline{CE}$为芯片允许（输入，低电平有效）。只有当此信号为低电平时，才允许CPU访

问本CTC芯片。通常，这个信号是根据地址总线低8位中的A7—A2进行译码得出。在CMC—80微型电脑系统中，当A7—A2＝100011时才选中CTC。

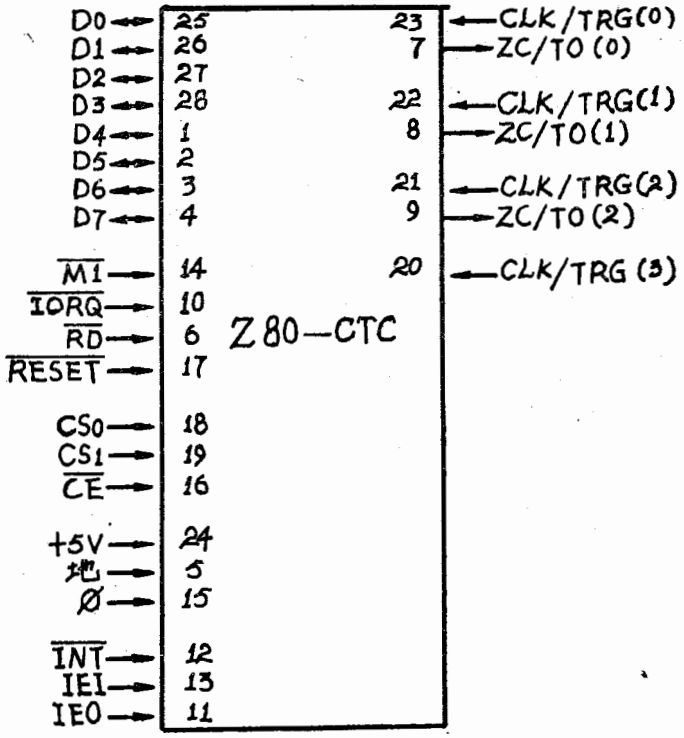


图 6.4 CTC的引脚布置图

3．CS1，CS0为通道选择（输入）。通常这两个引脚接至地址总线的A1，A0位，从而形成一个两位二进制地址码，以便在四个CTC独立通道中择一个通道。其选择逻辑如表6．1所示。

表 6.1　　CTC的选择逻辑

引脚			选中单元	单元地址
$\overline{CE}$	CS1 (A1)	CS0 (A1)		
0(A7—A2=100011)	0	0	通 道 0	8CH
0(A7—A2=100011)	0	1	通 道 1	8DH
0(A7—A2=100011)	1	0	通 道 2	8EH
0(A7—A2=100011)	1	1	通 道 3	8FH
1(A7—A2≠100011)	×	×	芯片未被选中	

4.$\overline{M1}$为CPU的取指令机器周期信号（输入，低电平有效）。

5.$\overline{IORQ}$为CPU的I／O请求信号（输入，低电平有效）。

6.$\overline{RD}$为CPU的读周期状态（输入，低电平有效）。

上述三种控制信号的作用如表6.2所示。

7.IEI为中断允许输入信号（输入，高电平有效）。

8.IEO为中断允许输出信号（输出，高电平有效）。

9.$\overline{INT}$为中断请求（输出，漏极开路，低电平有效）

表 6.2　　CTC 控 制 信 号 的 作 用

引脚			功能解释
$\overline{M1}$	$\overline{IORQ}$	$\overline{RD}$	
0	0	1	中断响应
0	1	0	检查中断服务程序是否结束
1	0	0	CPU从CTC读出
1	0	1	从CPU写入CTC
其	他	值	没有作用

上述三种控制信号的作用在第五章已经介绍，此处不再重复。顺便指出，通道0具有最高的中断优先级，依次为通道1，2，3。

10.$\overline{RESET}$为复位信号（输入，低电平有效）。这个信号终止所有通道的工作。禁止CTC产生中断请求，并禁止ZC／TO输出正脉冲。IEO重复IEI的状态，同时CTC的数据总线输出驱动器变为高阻状态。

11.CLK／TRG（3—0）)或称C／T3—0）为四个通道的“外部时钟／定时器触发”脉冲（输入，可由用户规定其为正跳变或负跳变有效）。它们作用将在下节介绍。

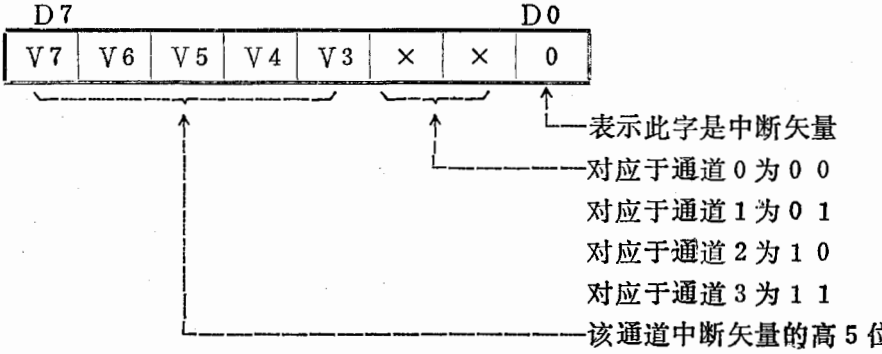
12.ZC／TO(2—0)（或称ZC2—0)为三个通道（通道3除外）的“回零／时间到”脉冲（输出）。它们的作用将在下节介绍。

13.其他：ϕ为时钟；　+5V为电源；　GND为地。

6.3　CTC的操作说明

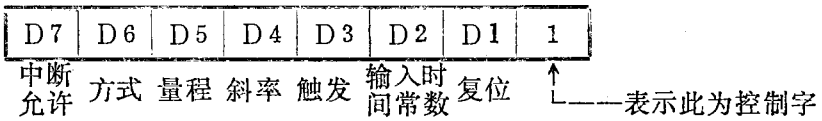
CTC的每个通道都有两种工作方式——定时器和计数器。从CPU向CTC写入的字有三种——控制字、时间常数和中断矢量。每个通道只占用一个外部设备地址，而不象PIO的一个口占用两个外部设备地址——一个用来传送数据，另一个用来传送控制字。因而需要规定一些特征来区别上述三种字。

CTC的中断矢量格式如下：



中断矢量的高五位在CTC 程序设计时写入通道0（中断矢量只需、也只能写入通道0），下二位将由中断控制逻辑提供对应于工作通道的二进制编码，如上述中断矢量格式所示。

控制字的格式如下：



D7＝1　允许通道中断

D7＝0　禁止通道中断

D6＝1　通道选择计数器方式工作

D6＝0　通道选择定时器方式工作

D5＝1　表示定标器系数为 256

D5＝0　表示定标器系数为 16

（D5仅在定时器方式时才有意义）

D4＝1　表示CLK/TRG的上升沿为有效

D4＝0　表示CLK/TRG的下降沿为有效

D3＝1　由CLK/TRG 来启动定时器的工作

D3＝0　由装入时间常数来启动定时器的工作

（D3仅在定时器方式时才有意义）

D2＝1　下一个写入的字是时间常数（或初始常数）

D2＝0　下一个写入的字不是时间常数

D1＝1　立即停止通道的工作，ZC/TO不起作用，并禁止通道中断逻辑。

D1＝0　每当减 1 计数器到达零时，时间常数寄存器立即将其内容装入减 1 计数器，通道继续现行操作。

可见时间常数是紧跟在控制字（且D 2＝ 1 ）之后写入，而不须用特征值来表示。

下面介绍CTC的两种操作方式。

一、定时器工作方式

例1. 由程序（装入时间常数）来启动定时器工作（以通道 0 为例）。

程序设计

```
LD      A, 23H      设定中断矢量
LD      I, A
LD      A, 50H
OUT     (8C), A
LD      A, 85H      设定工作方式
OUT     (8C), A
LD      A, 03H      装入时间常数
OUT     (8C), A
IM      2
EI
```

定时器工作的时间图如图6.5所示。

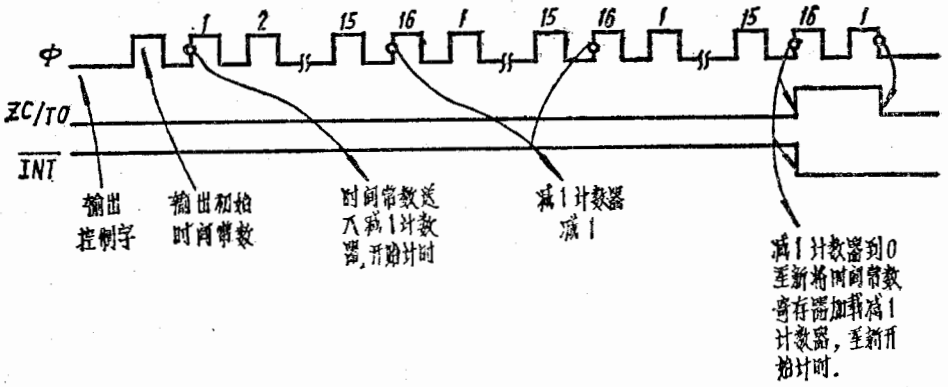


图 6.5 由程序控制的定时器的时間图

定时器的工作过程是：当时间常数03H装入时间常数寄存器后，定时器开始工作，时间常数寄存器将其内容装入减1计数器，定标器对时钟φ进行计数，本例中每输入16个φ（因为控制字D5＝0），定标器输出一个φT（见图6.3）使减1计数器减1，定标器输出第三个φT时，减1计数器由1减为0，ZC/TO发出一正脉冲，INT电平变低，向CPU请求中断。本例中因为控制字D1＝0，时间常数寄存器将其内容再一次装入减1计数器，并重复上述操作。可见ZC/TO上每隔48个φ出现一个正脉冲，直到写入一个停止其操作的控制字（D1＝1）为止。

本例中ZC/TO上发出正脉冲的时间间隔可以由程序来设定。发脉冲的起始和终止也由程序设定。

例2.用外界CLK/TRG来启动定时器的工作（以通道0为例）。

程　序　设　计

```
LD      A, 23H      设定中断矢量
LD      I, A
LD      A, 40H
OUT     (8C), A
LD      A, BDH      设计工作方式
OUT     (8C), A
LD      A, 03H      装入时间常数
OUT     (8C), A
IM      2
IE
```

定时器工作的时间图如图6.6所示

定时的器工作过程加下：当时间常数03H装入时间常数寄存器后，定时器等待CLK／TRG信号的到来。一旦出现CLK/TRG信号（本例中控制字D4＝1，故正跳变有效），定时器开始工作，其后的过程与例1基本相同。所不同的是，其一，每当出现第256个φ后(而

不是第16个，因为控制字中D5＝１），减１计数器减１；其二，当减１计数器到零后，ZC/TO上出现正脉冲。INT电平变低，定时器停止工作。只有当再一次出现CLK/TRG信号时，才能再次启动定时器的工作。

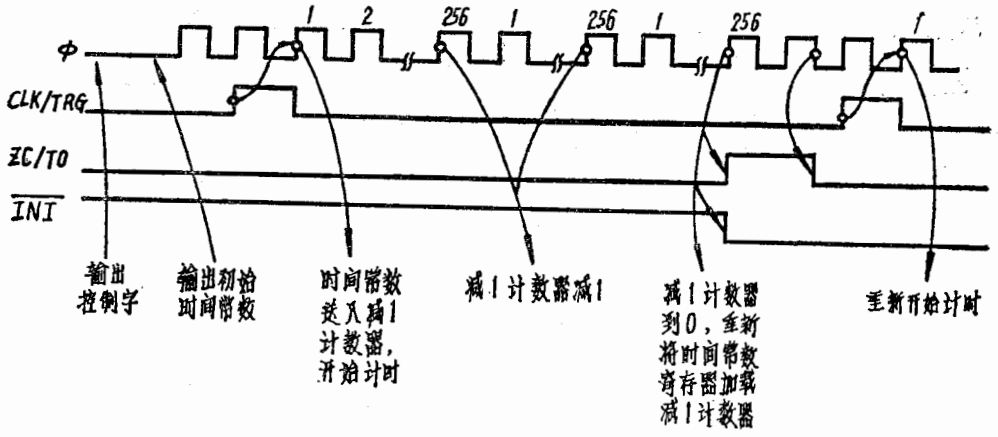


图 6.6　CLK/TRG 由外界控制的定时器的时间图

二、计数器工作方式

这种工作方式主要用于对外界（异步）事件进行计数。当到达规定数值后，ZC/TO发出脉冲，并使INT电平变低。这种工作方式的时间图如图6.7所示。

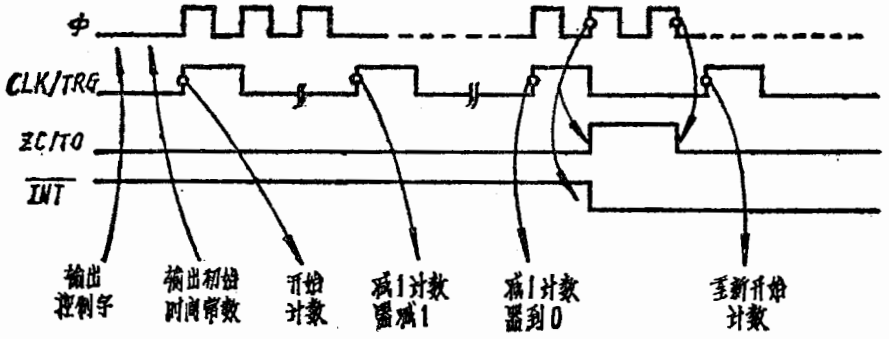


图 6.7　按計数器方式操作的时間图

```
                程　序　设　计
LD      A, 23H      设定中断矢量
LD      I, A
LD      A, 40H
OUT     (8C), A
LD      A, D5H      设定工作方式
OUT     (8C), A
LD      A, 03H      装入初始常数
OUT     (8C), A
IM      2
EI
```

计数器的工作过程是：首先输入控制字，规定按计数器方式工作等等。其次输入初始常数，计算器开始工作，时间常数寄存器将其中的初始常数装入减1计数器。之后每当输入一个CLK/TRC信号（表示发生一次外界事件），减1计数器即减1。直到减1计数器到达零时，ZC/TO发出一正脉冲，INT 电平变低，时间常数寄存器再次将初始常数装入减1计数器。以后的过程为重复上述计数操作。

6.4 CTC的硬件连接

图6.8示出了CPU与CTC的硬件连接。它和PIO的连接图相似，此处不再赘述。

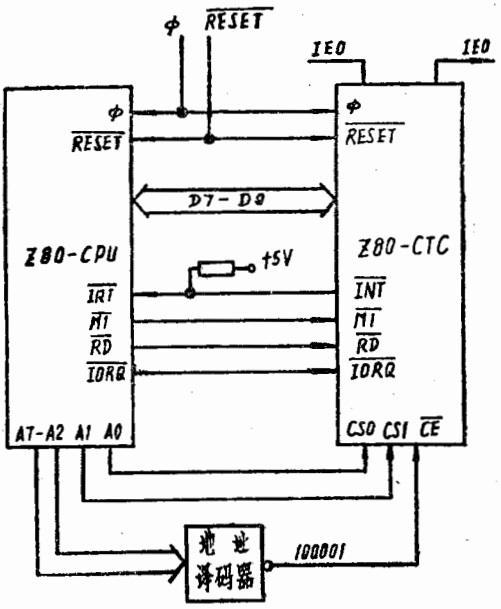


图 6.8 CTC与CPU的硬件連接图

第七章　串行输入输出接口芯片

Z80—SIO

7.1　概　　述

Z80—SIO是具有两个独立发送和接收串行数据通道的器件，每个通道都可通过编程而适配于异步，同步或SDLC通讯方式，两个通道的工作方式可以相异，但每个通道的发送和接收须为同一通讯方式。

Z80—SIO内的逻辑对SDLC和HDLC方式没有区别，用户可采用软件控制来适应这两种规程的细微差别。

SIO是Z80系列的芯片之一。它与CPU的接口界面包括双向数据总线，控制信号，中断信号和器件选择／寻址逻辑；它与数据通讯设备的接口界面包括数据传输，时钟、输入输出控制和调制解调控制信号。

Z80—SIO采用40引脚双列直插封装。芯片为NMOS工艺。所有输入和输出引脚与TTL兼容。采用单一正5伏电源。

7.2　SIO的引脚信号及寄存器寻址

SIO的引脚信号分为两类，一类是与CPU接口的信号，另一类是输出／输出的通道信号，我们先来看前一类。

双向数据总线DO～D7是SIO与CPU并行交换数据的通路，所有的控制码，状态字，发送的数据字和接收的数据字都由此进行传送。

$\overline{CE}$是主片选信号；

当CPU与SIO交换数据时，B/$\overline{A}$为低时表示有选择A通道，为高时表示选择B通道；C/$\overline{D}$为低时表示选择发送或接收缓冲器，为高时表示选择控制或状态寄存器；

使用SIO时应由CPU地址总线译码产生$\overline{CE}$，B/$\overline{A}$和C/$\overline{D}$，每片SIO占4个I/O口地址或存储单元地址，通常把地址线A0接到B/$\overline{A}$上，A1接到C/$\overline{D}$上，而由A2～A7(A15)译码后接到$\overline{CE}$上，例如，在CMC—80微型电脑中，A7～A2为100010时产生CE信号，寻址情况如下：

CMC—80 I/o地址	C/$\overline{D}$	B/$\overline{A}$	选通寄存器
88H	0	0	A通道发送/接收缓冲器
89H	0	1	B通道发送/接收缓冲器
8AH	1	0	A通道控制/状态寄存器
8BH	1	1	B通道控制/状态寄存器

虽然CPU对SIO寻址只有四个寄存器，但芯片内部的结构要复杂得多，从用户的观点来看SIO的逻辑如图7.1所示。可以看到，发送逻辑是单级缓冲，而接收逻辑是三级缓冲，它们组

成先进先出栈结构，每级缓冲器都有自己的出错状态标志。控制寄存器（常称写寄存器）每个通道有七个或八个；状态寄存器（常称读寄存器）每个通道有两个或三个。

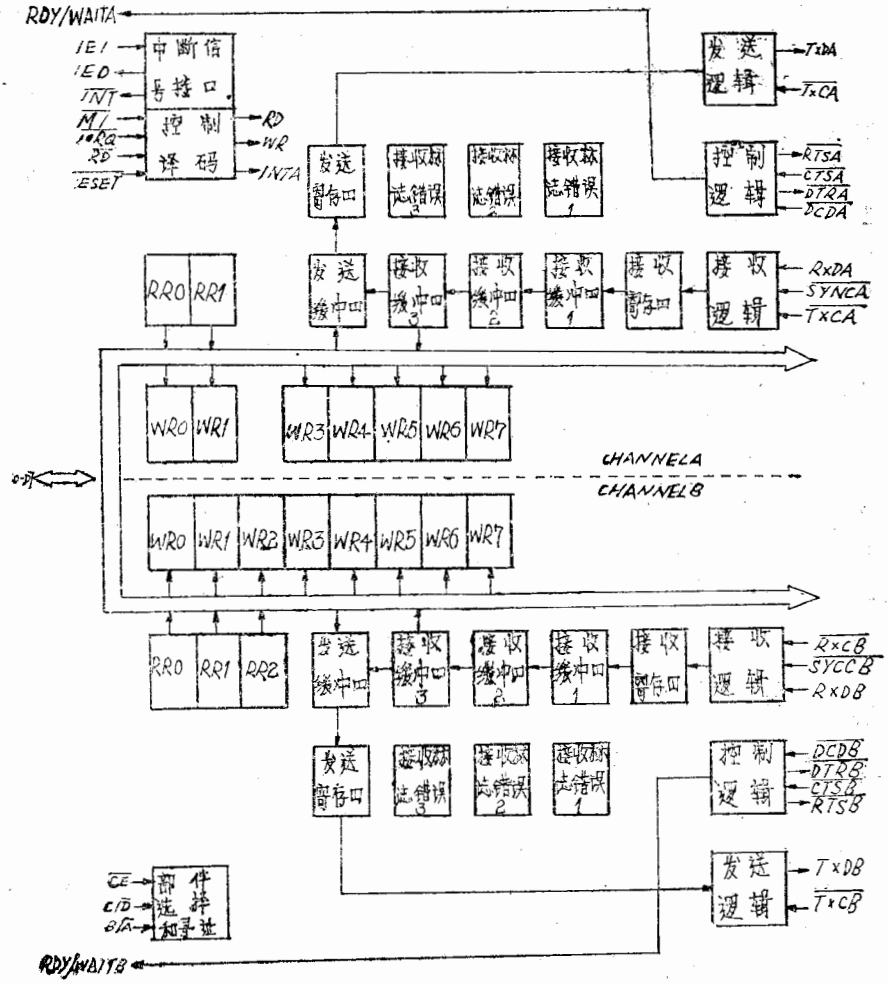


图7.1　SIO逻辑框图

当CPU访问读／写寄存器时，在一般情况下读写的是RR0或WR0，写寄存器WR0中有三位用来指出下次访问的读写寄存器序号。当CPU要从RR1或RR2里读取状态，或者要向WR1～WR7写入控制字时，必须对SIO进行两次访问：第一次把选定访问的寄存器序号写入WR0的低三位，第二次访问才能真正达到目的。写入WR0的序号仅对一次访问有效，此后又自动返回到选择WR0或RR0一般情况。

例如，考虑如下的指令序列

```
LD      A, 3            取00000011送累加器
OUT     (8BH), A        送SIO, B通道写寄存器WR0
LD      A, DATA         取数据DATA送累加器
OUT     (8BH), A        写入B通道WR3
```

第一条指令把03H取到累加器。然后把它写到B通道WRO内，使得它的低三位为011，下一次写入B通道控制寄存器就会写到WR 3 里面去。

第三条指令取到累加器的数据DATA由末一条OUT指令送往WR 3，这条指令一经执行，通道B的逻辑就立刻返回到选择WR0或RR0，以便再读写其它的寄存器。

由于Z80—CPU有成组输入／输出指令，因此可用一条指令向写寄存器写入一系列的控制字。例如取下列数据存放在连续的存储单元里：

1	×××××100	选择WR 4
2	NNNNNNNN	写入WR 4 的控制字
3	×××××001	选择WR 1
4	MMMMMMMM	写入WR 1 的控制字
5	×××××010	选择WR 2
6	PPPPPPPP	写入WR 2 的控制字
7	×××××011	选择WR 3
8	RRRRRRRR	写入WR 3 的控制字
9	×××××101	选择WR 5
10	SSSSSSSS	写入WR 5 的控制字
11	×××××110	选择WR 6
12	TTTTTTTT	写入WR 6 的控制字
13	×××××111	选择WR 7
14	YYYYYYYY	写入WR 7 的控制字

其中 M、N、P、R、S、T、V和×表示用户根据需要指定的二进制数。则利用OTIR指令就可以一次把这十四个数据分别写到八个写寄存器里，其中第1、3、5、7、9、11和13个字节自动写入WR0，它们指定下一个字节写入的寄存器序号。第2、4、6、8、10、12和14个字节就写入前一字节指定的寄存器里，接着又自动返回到选择WR0的状态。

信号状态($\overline{CE}=0$)			访问寄存器情况
C／$\overline{D}$	B／$\overline{A}$	$\overline{RD}$	
0	0	0	由A 通道读缓冲栈顶读取
0	0	1	写入A 通道发送缓冲器
0	1	0	由B 通道读缓冲栈顶读取
0	1	1	写入B 通道发送缓冲器
1	0	0	读取A 通道RR0,RR1*
1	1	1	写入A通道WR0,WR1,WR3,WR4,WR5,WR6,WR7*
1	1	0	读取B通道RR0,RR1*RR2**
1	1	1	写入B通道WR0,WR1,WR2,WR3,WR4,WR5,WR6,WR7*

*RR0～RR2，WR0～WR7由WR0的低三位所写入的数所选定，如：

WR0 [× × ×]

读出				写入
RR0	0	0	0	WR0
RR1	0	0	1	WR1
RR2	0	1	0	WR2
	0	1	1	WR3
	1	0	0	WR4
	1	0	1	WR5
	1	1	0	WR6
	1	1	1	WR7

*RR2仅能由通道读出

表7.1 Z80—SIO寄存器寻址

用户通过对SIO的每个通道写寄存器写入适当的代码和从读寄存器读出状态来控制它按需求进行工作，在微计算机存在储器和SIO之间并行传送数据时，可以使用直接存储访问方式，也可以利用中断逻辑。

SIO与CPU之间的另一类信号与总线控制有关，包括$\overline{M1}$，$\overline{RD}$和$\overline{IORQ}$，它们的逻辑组合对SIO产生读，写和中断应答的控制作用。其情况如表7.2所示。

控制输入			Z80→SIO功能解释(设$\overline{CE}=0$)
$\overline{M1}$	$\overline{RD}$	$\overline{IORQ}$	
0	0	0	禁用状态，可能引起故障
0	0	1	检查是否有RETI指令，判断中断服务的结束
0	1	0	中断应答
0	1	1	不　用
1	0	0	CPU读，SIO输出送CPU
1	0	1	不　用
1	1	0	CPU写，CPU输出送SIO
1	1	1	不　用

表7.2 SIO对CPU控制信号的反应

SIO还有两条$\overline{WAIT}/\overline{RDY}$控制信号线（每个通道一个），它们可以通过编程成如向CPU申请等待的信号，或者作为DMA请求服务信号。我们先来看它作为等待信号的情况。当I/o指令要与发送缓冲器或接收缓冲器传送数据时，$\overline{WAIT}/\overline{RDY}$可以在读或写机器周期里变成低电压平，从而使CPU进入等待状态，直到所选定的缓冲器可以交换信息时为止。当CPU接收到低电平的$\overline{WAIT}$信号时，它就在读或写机器周期里自动插入一连串的等待时钟周期，一直到$\overline{WAIT}$信号变成高电平为止，可参见CPU的说明。

等待是很有用的信号，它允许CPU既不采用中断方式，也不使用查询方式而与SOI交换成组的信息。当CPU企图从尚未得到接收数据的缓冲寄存器里读取信息时，SIO将输出低电平的$\overline{WAIT}$信号，直到读访求的数据就位为止。类似的，当CPU向SIO的发送逻辑写入数据时，如果发送缓冲器空，则可以接受它；如果前次写入的数据尚未移出，则$\overline{WAIT}$信号变低，使CPU的写／输出机器周期里插入等待周期，直到发送缓冲器可以接收新的数据。

SIO输入输出串行的数据，而CPU与它进行并行的数据交换，等待逻辑保证CPU读取的一定是有效的数据，而仅当通道准备接收数据时CPU才会向它写数。这些功能都需通过向SIO编程才能据需要实现。

外部通讯设备与微电脑系统并行传输成组数据更有效的途经是采用存储器直接访问（DMA）逻辑，适当的编程可以使$\overline{WAIT}/\overline{RDY}$引脚产生DMA机器周期申请信号，送往Z80—DMA—类DMA控制器去。Z80—SIO没有接收DMA应答信号的输入脚，此外，为了完成所需的数据传输，DMA控制器需产生适当的C/$\overline{D}$、B/$\overline{A}$、$\overline{CE}$、$\overline{RD}$、$\overline{M1}$和$\overline{IORQ}$信号送往SIO。

每个通道的$\overline{WAIT}/\overline{RDY}$引脚或是编程如向CPU请求等待，或是向DMA控制器申请服务，它不能同时实现两种功能；另外，这个引脚或是为接收逻辑使用，或是编程为发送逻辑服务，二者亦不可得兼。但A、B两通道各自的$\overline{WAIT}/\overline{RDY}$是完全独立的，对一者的编程不影响另一者的使用方式。

如果用户也不采用DMA方式来实现通讯设备通过SIO与电脑内存的数据交换，则还可以利用中断逻辑。如果一个通道同时完成接收和发送的功能（全双工），既可以使用中断服务处理两方面的数据传输；也可以对一向传输采用中断逻辑，而另一向靠DMA进行管理，这当然要靠对DMA控制器适当编程才能实现。

SIO的中断逻辑信号包括中断请求$\overline{INT}$，中断优先健信号IEI和IEO，以及前面介绍过的中断应答复合信号$\overline{M1}$和$\overline{IORQ}$。

中断请求和应答信号是容易理解的。SIO通过输出$\overline{INT}$低电平信号向CPU申请中断，CPU使$\overline{M1}$和$\overline{IORQ}$同时为低电平表示这一中断申请已被接受。中断优先健的构造在介绍PIO一章中已经叙述过了，在未申请中断时，中断健里任何器件的输出信号IEO与其输入信号IEI具有同一逻辑电平；器件的IEI输入高电平时它的中断申请逻辑才被选通；当SIO申请中断时，它的IEO输出变低，从而闭锁了健下面优先权较低器件的中断申请逻辑，只有在CPU执行了一条“中断返回”指令（RETI）或向SIO发出了一条“复位中断健逻辑”命令后，SIO的IEO输出才回升为高电平。

上述中断优先健逻辑对中断应答信号的接收也起了控制作用，当CPU送出中断应答信号时，请求中断服务储器件中仅有优先权最高者才能接收这个信号，其它低优先权的器件中断机构都被封锁，直到高优先权器件送出有率的IEO信号为止。更高优先权的器件当然可以接收中间插入的应答信号。

SIO还需要有时钟信号ϕ以同步内部操作，此外，无庸赘言，还有电源和地线的引脚。最后一个重要的信号是复位（$\overline{RESET}$），由外面送给SIO的$\overline{RESET}$信号至少要在一个ϕ时钟周期内保持低电平，它产生如下的效果：

1）TxDA和TxDB输出为高电平（号传状态）；

2）调制解调控制输出均为高；

3）所有寄存器内容均复位，所有中断被禁止，两个通道的发送接收逻辑停止工作；

4）数据总线进入浮动状态。

SIO也可以只使一个通道复位而另一个通道不受影响，其控制代码在后面讨论写寄存，时详述。

当对整个SIO器件或它的一个通道施行复位时，悬挂起来的外部/状态中断被清除，但当再次对SIO器件初始化时，这一中断或许又会产生出来，（外部／状态中断是作为一个范畴处理的一组中断条件，（后面将详细说明）因此，复位后应对每个通道送两次“复位外部／状态中断命令。

以上是SIO与微电脑系统内部的信息交换信号，下面再来看它的串行I/o接口信号。

SIO的每个通道理应有九个信号，它们包括串行发送信号TxD和串行接收信号RxD；发送时钟$\overline{TxC}$和接收时钟$\overline{RxC}$，同步信号$\overline{SYNC}$和四个调制解调控制信号：请求发送$\overline{RTS}$（输出）；清除发送$\overline{CTS}$（输入）；数据载波检测$\overline{DCD}$（输入）和数据终端就储$\overline{DTR}$（输出）。

发送的串行数据在$\overline{TxC}$的下降沿（高到低）产生跳变，而接收的串行数据在$\overline{RxC}$的上升沿（低到高）取样。

$\overline{SYNC}$信号在外同步通讯方式时作为写符边界同步的输入信号，在内同步和SDLC通讯方式时它作为找到同步时的输出信号，在异步通讯方式时它是一般的输入引脚。

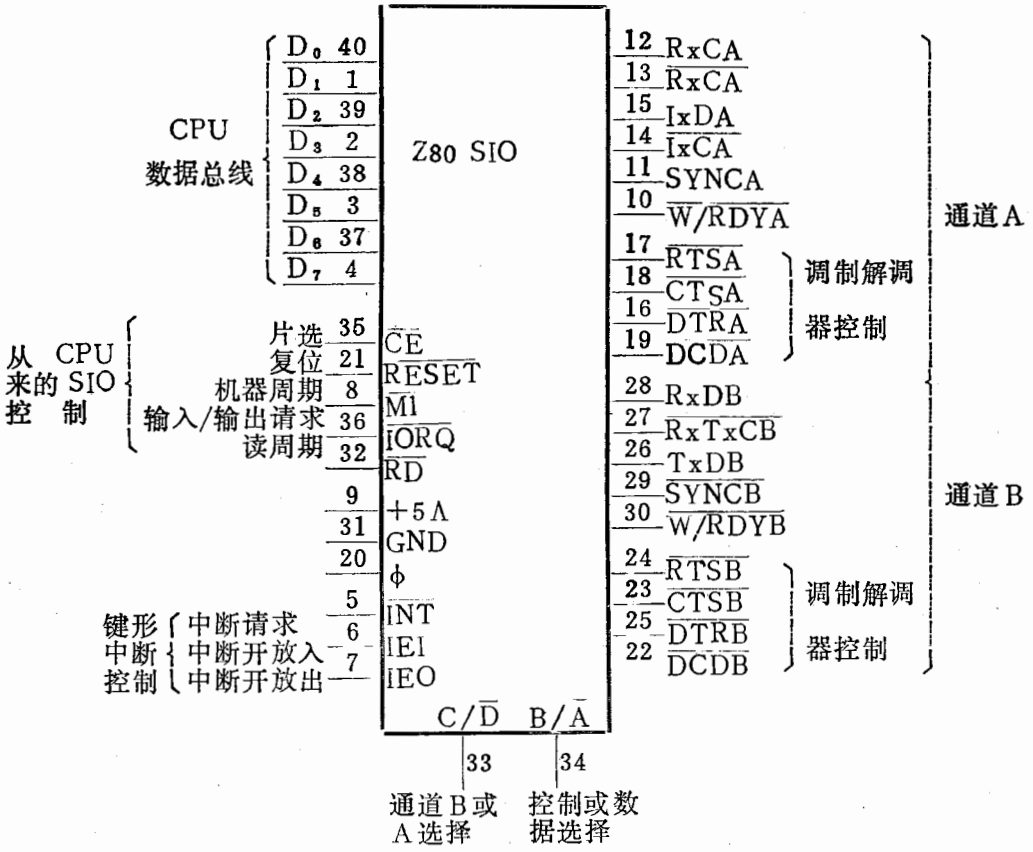


图 7.2 SIO引脚布置图

由于封装引脚的限制，B通道的I/o信号只有八个，SIO由于所牺牲的信号不同而有三种封装型是，SIO/ϕ的$\overline{TxCB}$和$\overline{RxCB}$共用一个引脚；SIO/1牺牲了$\overline{DTRB}$；而SIO/2牺牲了$\overline{SYNCB}$其引脚见图7.2。

7.3 SIO的读写寄存器

系统的CPU是通过向SIO的写寄存器输出控制字和由读寄存器输入状态字来控制它的操作的。表7.3列举了Z80—SIO写寄存器的内容。

下面讨论Z80—SIO的读写寄存器时常遇到称为“外部／状态”中断的一组中断条件，它包括：

1）由$\overline{DCD}$、$\overline{CTS}$或$\overline{SYNC}$引脚上的信号变化产生的中断；

2）发送脱空和信息结束（Underrun and end of message）中断；

3）由断开或异常结束（Break or abort）产生的中断。关于这类中断后文还要更详细地介绍。

写寄存器0用来指定下次访问读写寄存器时所选定的读或写寄存器序号，它也用来发出一些常用的控制命令。

装入WS0的0、1、2位的数值用来指定下次访问本通道读写寄存器时而选定的对象，这一批理正在7.2节详细说明过了。

WS0的3、4、5位选定七个命令之一。000不产生任何操作，其它七个命令分别是

1）发送异常结束（Abort）。001使得此通道发出一个SDLD异常结束字符，输出八位继续的逻辑1。在此之前输出的也许是逻辑0，也许已经有最多五个1，（数据在连续五个1后面就会插入0）所以加上前面可能存在的1，SDLC异常结束字符就变成总共连续的八个到十三个1。

2）复位外部／状态中断。010使得所有已发生的外部／状态中断逻辑及读寄存器内相应的状态位复位，包括已被接受和未被接受的都在其内。

3）通道复位。011产生与前述的$\overline{RESET}$信号同样的复位操作，但只复位本通道逻辑，而$\overline{RESET}$信号复位两个通道。本命令占用四个ϕ期周。

4）首字中断后允许再度中断。在程序指定SIO接收器只在接收首字时中断的方式下，当第一个接收字符产生中断请求后，相继到达的数据都不再引起接收中断，100命令使得此后又产生一次首字中断。

5）复位挂起的发送中断。在程序指定SIO发送器每当缓冲器空时请求中断的方式下，如果不向发送缓冲器里送数，则这一中断请求反复出现——每当发送逻辑从发送缓冲器里取数而发现未送时就申请一次中断。101命令使这种中断被锁闭起来，直到CPU再向发送缓冲器送入下一数据后才能再度发出发送中断。

6）出错复位命令。110复位本通道的奇偶，溢出，格式和CRC出错状态位（在读寄存器RR1内），这一命令使此类中断再次开放。

7）中断返回，111命令使已被接收的SIO中断请求复位，这一命令只在A通道的WR0内存在。如果此片SIO内没有其它被挂起来的中断请求，则其IEO引脚输出高电平，使得

中断优先链里较低级器件的中断逻辑开放”如果本片SIO内还有挂起来的中断，则其最高优先权者提出申请，IEO仍为低电平。这一命令主要使SIO用于非Z80系统，在Z80系统内当然可以使用RETI指令。

WR 0的6和7位产生三个命令。0 0不产生作用，三个命令是

1）0 1复位接收循环冗余码（CRC）校验逻辑。

2）1 0复位发送循环冗余码（CRC）产生逻辑，在处理新的一帧SDLC信息或一组同步数据时，通常对接收和发送的CRC逻辑都要进行复位，对同步方式复位为00H，对SDLC方式复位为FFH。

3）1 1复位发送脱空和信息结束（Underrun and end of message）中断。这一常用命令复位RR 0的第6位，再度开放这种中断，后面还会详细讨论这一命令的作用。

写寄存器1用来开放各类中断和指定$\overline{\text{WAIT}}/\overline{\text{READY}}$引脚的功能：

WR 1位0开放一组称为“外部/状态”的中断，

WR 1位1开放发送缓冲器空中断，每当数据由发送缓冲器送入发送移位寄存器时即产生这一中断，而空的发送缓冲器自身并不产生中断请求。

WR 1位2仅在B通道内有效，当这位置1时，SIO在中断请求被接受时送给CPU的中断向量低字节可根据该中断的性质而取八值之一。这一机理后文还要详述。

WR 1的3和4位控制接收中断逻辑，可有如下的选择：

1）可以禁止所有接收中断；

2）可以仅在接收了第一个数据字节后请求中断，这常用来初始化DMA控制器，也可用在查询或等待式的工作方式的启动上。

3）可以在每个接收字符装配完毕后都请求中断。如果通过WR 1位2使状态影响向量，则此时对接收数据字的状态是否影响向量仍可再作选择，对于高可靠的高速数据块接收可以不考虑出错中断的问题，否则就要使出错影响中断向量。

WR 1的5、6、7位控制本通道$\overline{\text{WAIT}}/\overline{\text{RDY}}$引脚的作用，位7选通或禁止这一信号的使用；位5决定这一信号用于发送还是接收，它无法同时为两方面服务；位6确定这一信号用于使CPU等待还是向DMA控制器发出DMA请求。$\overline{\text{WAIT}}/\overline{\text{RDY}}$的使用7.2已有略述，表7.4列举了这一信号与WR 1的5、6、7位设置的关系。

写寄存器2只在通道B里具有。当SIO申请中断被CPU接受时，它就把B通道WR 2的内容送上数据总线作为中断向量的低位字节，如果B通道WR 1位2选通了状态影响向量，则WR 2内的1、2、3位会由于引起中断的条件而进行相应的修正。如果无须使用中断逻辑，则不可以不向WR 2内写入控制字；但无论在那个通道使用中断，则必须先向B通道WR 2内写入中断向量。关于中断的类型及各自的向量后文还将详细叙述。

写寄存器3主要用于接收逻辑的控制。

WR 3位0是通道接收逻辑的选通标志。

WR 3位1用于在同步通讯接收数据流中剔除同步字符。如果这位写0，则同步字符和其它数据一样被传送给CPU，这位仅对同步方式有意义，但在异步和SDLC通讯方式时这位应置0。

WR 3位2仅用于SDLC方式。此位置1时只有地址段与本通道WR 6的内容相一致的

表 7.3　　SIO写寄存器的内容

寄存器	位序 7 6 5 4 3 2 1 0	方式 异步	同步	SDLC	功能
WR0 指针和主控命令	AA BBB CCC				
		×	×	×	—下次访问寄存器的指针 000—WR0／RR0　001—WR1／RR1 010-WR2/RR2　011-WR3　100-WR4 101-WR5　110-WR6　111-WR7
		×	×	×	—0 0 0 无操作
				×	—0 0 1 发送SDLC异常结束(abort)字符
		×	×	×	—0 1 0 复位 RR0 的中断状态位，重新开放外部状态中断
		×	×	×	—0 1 1 通道复位
		×	×	×	—1 0 0 首字中断后允许再次中断
		×	×	×	—1 0 1 发送缓冲器空后闭锁这一中断，直到下次再向发送缓冲器送数时再开放
		×	×	×	—1 1 0 复位RR1的奇偶、溢出、格式和CRC出错标志位
		×	×	×	—1 1 1 复位中断优先权逻辑(通道A)
		×	×	×	—0 0 无操作
			×	×	—0 1 复位接收CRC校验器
			×	×	—1 0 复位发送CRC产生器
		×	×	×	复位RR0的发送脱空(Vnderrun)和信息结束中断标志位
WR1 中断·等待和就绪选通	A B C DD E F G				
		×	×	×	—开放外部／状态中断
		×	×	×	—开放发送缓冲器空中断
		×	×	×	—状态影响向量(通道B)
		×	×	×	—0 0 禁止接收中断
		×	×	×	—0 1 首字接收中断
		×	×	×	—1 0 每字接收中断，特定接收条件影响向量
		×	×	×	—1 1 每字接收中断，不修改向量
		×	×	×	—$\overline{\text{WAIT}}$／$\overline{\text{RDY}}$用于接收／发送
		×	×	×	—$\overline{\text{WAIT}}$／$\overline{\text{RDY}}$用于DMA等待
		×	×	×	—选通$\overline{\text{WAIT}}$／$\overline{\text{RDY}}$

寄存器	位序 7 6 5 4 3 2 1 0	方式 异步	同步	SDLC	功能
WR2 中断向量	A A A A B B B A	×	×	×	—中断向量低八位，第3.2.1位可选择受中断状态影响而修改
WR3 接收控制	A A B C D E F G				
	→	×	×	×	—允许接收
	→		×		—剔除接收的同步字符
	→			×	—仅接收有效地址帧
	→		×	×	—选通接收CRC校验器
	→		×	×	—进入同步／标志搜索状态
	→	×	×	×	—由$\overline{CTS}$和$\overline{DCD}$选通发送和接收
	→	×	×	×	—每个接收字符的位数：0 0—5位；1 0—6位；0 1—7位 1 1—8位。
WR4 工作方式控制	A A B B C C D E				
	→	×	×		—选通奇偶校验
	→	×	×		—选定偶／奇校验
	→		×	×	—0 0 同步或SDLC方式
	→	×			—0 1、1 0、或1 1异步方式，停止位数为1，1½，2
	→		×		—0 0 单同步方式
	→		×		—0 1 双同步方式
	→			×	—1 0 SDLC方式
	→		×		—1 1 外同步方式
	→	×	×	×	—0 0 × 1 对时钟TCx和RCx分
	→	×			—0 1 ×16 频，使码元周期为钟周
	→	×			—1 0 ×32 期的倍数选择
	→	×			—1 1 ×64
WR5 发送控制	A B B C D E F G				
	→		×	×	—选通发送CRC产生器
	→	×	×	×	—RTS输出选通
	→		×	×	—CRC多项式选择CRC—16／SDLC
	→	×	×	×	—允许发送
	→	×	×		—送空位(SPACE)，使Tx D为低电平
	→	×	×	×	—每个发送字符位数：0 0—5位或更少 1 0—6位；0 1—7位；1 1—8位；
	→	×	×	×	—DTR输出选通

寄存器	位序 7 6 5 4 3 2 1 0	方式 异步	同步	SDLC	功能
WR6	AAAAAAAA		×		—单同步发送同步字符；双同步首同步字符；外同步发送和接收中剔除的同步字符
				×	—本机地址字段
WR7	AAAAAAAA		×		—单同步接收同步字符；双同步次同步字符
				×	—标志字段(01111110)

表 7.4　　SIO $\overline{\text{WAIT}}/\overline{\text{RDY}}$ 信号值表

WR1位 7	WR1位 6	WR1位 5	$\overline{\text{WAIT}}/\overline{\text{RDY}}$ 信号电平
0	0	×	浮动状态
0	1	×	高电平
1	0	0	当发仍缓冲器满而实行写入时 $\overline{\text{WAIT}}$为低电平，其它情况下为浮动状态
1	0	1	当接收缓冲器空而CPU实行读出时 $\overline{\text{WAIT}}$为低电平，其它情况下为浮动状态
1	1	0	当发送缓冲器满时$\overline{\text{RDY}}$为高电平 当发送缓冲器空时$\overline{\text{RDY}}$为低电平
1	1	1	当接收缓冲器空时$\overline{\text{RDY}}$为高电平 当接收缓冲器有数据时$\overline{\text{RDY}}$为低电平

信息帧和具有通用地址FFH的信息帧才被接收，其它信息帧全部拒收；这位置0则接收所有的信息帧。在异步或同步方式时这位应置0。

WR3位3用于在接收数据流中有选择地取一部分来进行CRC校验计算。当WR3位3置1时就对接收的数据进行CRC计算，但SIO的CRC计算比接收延迟一个字的时间，也就是说，一个接收字仅在它由接收缓冲器1送到接收缓冲器2时才开始对它进行CRC计算，所以，程序有一个接收字的时间来查询所接收的字符并关闭对它进行的CRC计算（把WR3位3清0）；同样也有一个字接收的时间来查询下一个接收字并决定对它打开CRC计算（对WR3位3置1）。

WR3位4与1使接收逻辑进入搜索状态，在同步方式中查找同步字符，在SDLC方式中停止接收本帧信息而查寻标志字符。

WR3位5用来选通调制解调控制信号$\overline{DCD}$和$\overline{CTS}$当这位为1时，接收逻辑仅当$\overline{DCD}$为低时才接收数据，而只当$\overline{CTS}$为低时发送逻辑才开始工作。如果这位为0，则只要在WR3位0和WR5位3写入1，接收和发送就分别开始工作了。

WR3位6和7用来确定接收逻辑把串行数位组装成字的位数，对于面向字的异步或同步通讯格程这是容易理解的，对面向位的SDLC格程，也需要把其信息段按输入位的顺序组装成一个个的字才能送给CPU。

如果选择每个字8位，则8位数据装成一个字，异步和同步方式中虽然可进行奇偶校验，但奇偶位被丢弃而不能装配进数据字里。

对于每个字只有7位、6位或5位的情形，则组装的字是低位（右）对准的，对于异步或同步通讯并选用了奇偶校验时，奇偶校验位被装配在最高有效数位的左面，其它的空位一概组装如1，形式如下：

每字8位	DDDDDDDD
每字7位	PDDDDDDD
每字6位	1PDDDDDD
每字5位	11PDDDDD

其中D表示接收的二进制数位，P表示奇偶位，当不选用奇偶校验或SDLC方式时，P＝1。

写寄存器4确定通道接收和发送的基本工作方式，在初始化SIO通道时应首先对WR4写入控制字（WR2可以先于WR4写入而不会发生其它问题）。

WR4位0和1控制发送和接收的奇偶校验。在SDLC方式中并不能自动地排除此二位所指定的奇偶性状态，但SDLC规程没有奇偶校验的内容，所以须在WR4位0写0。异步和同步通讯里对奇偶性的处理已见前述。

WR4位2、3、4和5选择异步，同步或SDLC工作方式，位2和3还指定异步方式每字所带的停止位数。位4和5在同步通讯中还指定同步的选用：包括单同步字，双同步字或外同步。

WR4位6和7决定取样钟频。这主要用在异步通讯方式上，通常在异步方式中对外来钟信号每16个，32个或64个脉冲取样一位串行数据，如果在异步规程中也在每个钟脉冲取样数位，则称为“伪同步”或“准同步”。

写寄存器5主要用于发送逻辑的控制。

WR5位0用于开放发送循环冗余码产生器，位2用于确定使用哪个循环冗余多项式进行计算。通常同步方式使用CRC—16多项式，SDLC方式使用SDLC多项式。通讯规程规定同步或SDLC方式应采用CRC校验，在异步通讯中不要试图去计算CRC字。

WR5位1控制调制解调器联络输出信号$\overline{RTS}$，在同步或SDLC方式下，$\overline{RTS}$输出状态与WR5位1状态互补：在异步方式下，当WR5位1复位为0时，$\overline{RTS}$信号仅在现行字符发送完毕而且发送缓冲器为空时才变成高电平，当WR5位1置1时，$\overline{RTS}$总为低电平。WR5位3是发送逻辑选通信号，当需要发送时此位须得置1。

WR5位4在异步或同步通讯中用于产生异常结束（断开）信号，当此位置1时，发送输出TxD立刻被锁定为低电平（空位，SPACE）。如果要发送逻辑正常工作，须得向这位送0。

当在任何工作方式下发送出断开命令时，发送缓冲器和发送移位寄存器里的数据即丢失，但若在同步方式下正在发送同步字符的半途送入了这个命令，则输出TxD在发送完同步字符后才开始成为空位状态；如果正在发送CRC码时送入断开命令，则同步字符取代CRC码输出完毕后成为断开状态。

WR5的5和6位确定发送每字符的位数，这个位数不一定要和本通道接收逻辑的字位数一致。如果选择每字6或7位，则发送逻辑对送入发送缓冲器的8位数据的高位不予发送，果选择每字5位，实际上则是每字“5位或更少”，对送入发送缓冲器的数据字节，如果高三位全零，则不发送它们，如果在速续三位的左边（高位）还有（连续的）1，则这些1也不发送。所以，当指定每字5位的时候，据根实际发送字的位数组装的字节形式如下：

每字5位	0 0 0 D D D D D
每字4位	1 0 0 0 D D D D
每字3位	1 1 0 0 0 D D D
每字2位	1 1 1 0 0 0 D D
每字1位	1 1 1 1 0 0 0 D

其中D表示待发送的字位。

WR5位7控制调制解调联络信号$\overline{\text{DTR}}$的输出$\overline{\text{DTR}}$状态与WR5位7状态互补。

写寄存器6和写寄存器7存放同步通讯和SDLC通讯所需的字符。

在同步方式里，WR4的4、5位决定使用一个或两个同步字符，如果是单同步，则发送的同步字符放在WR6内，而接收时用来搜索的同步字符写在WR7内；如果是双同步，则第一个同步字符放在WR6内，第二个同步字符放在WR7里。

在SDLC方式时，标志字符应放入WR7内，而本站地址段放在WR6里。发送时SIO自动发送标志，而CPU的程序控制在开标志之后所发送的地址段，控制段和信息段的内容。WR6里的内容仅用于接收逻辑，接收电路在输入数位流里先查找WR7的内容以检测标志，如果WR3位又规定进行地址检索，则它在标志之后核对接收来的地址段内容是否与WR6的内容相符，以决定对此帧信息的接收或拒收。

下面转向考察读寄存器。

读寄存器0（RR0）包含在发送或接收操作过程中SIO的状态；读寄存器1（RR1）包含在发送或接收告一段落时所用到的信息；读寄存器2（RR2）里放的是写到WR2里的中断向量，如果选择状态影响向量时，则它的内容也作相应的修正，RR2仅存在于B通道。

控制SIO的工作可以利用中断逻辑，也可以使用查询的方法，周期地读出每个通道RR0

的内容以监视其运行，看是否发生了需要照料的事情。当然采用中断逻辑要效率高得多，可以在申请中断后，在中断服务程序里再来读RR0的内容。

RR0的位0在接收缓冲器栈内有一个或更多的字时便置1。由图7.1可以看到，输入有三级缓冲栈，接收移位寄存器将装配好的字送入缓冲器1，然后再上升到缓冲器2，再进入缓冲器3。当CPU读取数据时，缓冲栈里已装有接收数据的最高位缓冲器的内容被送上总线。所以，只有栈里至少有一个未被读取的接收字，RR0的0位便置为1。

接收缓冲器1、2、3各伴有自己的接收错误缓冲，当读取RR1的内容时，栈顶接收字的奇偶性、格式和溢出错误状态位被送出。接收逻辑保持每个接收字与它的错误标记相伴传送。应当注意，当CPU读取了一个接收字时，接收逻辑就刷去了它的错误标记，所以应当在读取数据之前读它的出错状态，如果先读了接收字，那末再读出的是下一个字的出错标志，而不是已读取接收字的状态了。

RR 0位1是总的中断请求状态，这一位仅在通道A里有意义，无论A或B通道发生了中断请求，通道A的RR 0位1被置1。

RR 0位2在发送缓冲器空的时候被置1，这不是出错状态，它只是简单地标明发送逻辑等待下一个字送来以便接着发送。

RR 0位3表示在下列诸原因引起中断请求时调制能调联络输入信号$\overline{DCD}$的互补状态：

1）$\overline{DCD}$发生电平变化

2）$\overline{CTS}$发生电平变化

3）$\overline{SYNC}$作为输入信号发生电平变化

4）检测到输入的断开（Break）或异常结束（Abort状态。

5）出现发送脱空（Underrun）或信息结束中断，如果有两种或更多的上述原因出现的中断请求尚未服务，则RR 0位3取最先出现中断请求时$\overline{DCD}$状态的反码。

RR 0位4对$\overline{SYNC}$脚作为输入或作为输出时代表不同的意义，在异步通讯或外同步通讯时，如果未发生中断，RR 0位4表示$\overline{SYNC}$输入信号的状态（反码）；如果发生了前述5类的外部／状态中断，则RR 0位4保持中断产生时的状态不再变化，直到不再有待服务的外部／状态中断为止。在内同步或SDLC通讯时，当复位或因WR 3位4写1而强迫进入同步检索状态后，RR 0位4置1，找到同步字符或标志字符后这一位清0。

RR 0位5表示外部／状态中断发生时$\overline{CTS}$信号的状态（反码），其情况与RR 0位3相同。

RR0位6，与写寄存器0位6和7为11的命令一起，提供了对发送逻辑脱空（underrun）和信息结束的控制。

复位后RR0位6置1，这位权能使用WR 0的6.7两位写入11来清0。

如果在发送数据前RR0位6清为0，则发送逻辑把脱空（underrun）解释为信息结束的标志。在SDLC方式中，脱空将使发送逻辑把CRC的现行值发送出去，然后接着发关标志（close flag）。此时RR 0的位2和4清0，通知CPU一帧信息已发送完毕，等待新的数据信息。在同步方式中，脱空也使得CRC码发送出去，继而发送同步字符，RR 0位2清0，准备好发送新的数据信息。

表 7.5　　SIO 读寄存器的内容

寄存器	位序 7 6 5 4 3 2 1 0	方式 异步	同步	SDLC	功能
RR0 接收和发送逻辑状态	A B C D E F G H	×	×	×	接收缓冲器里有接收字符
		×	×	×	SIO有挂起的中断(仅A通道有意义)
		×	×	×	发送缓冲器空
		×	×	×	发生外部／状态中断时DCD的状态
		×	×		发生外部／状态中断时SYNC的状态
			×	×	搜索同步状态(内同步或SDLC方式)
		×	×	×	发生外部状态／中断时CTS的状态
		×	×	×	复位或发送脱空／信息结束时置 1
					WR0－6.7位写入11时清。
		×			断开(break)检测到时置 1
			×		CRC比较的结果
				×	异常结束(pabort)检测到时置 1
RR0 操作结束和错误状态	A B C D E E E F	×			全字发送完毕，发送缓冲器空
				×	最后一装配字里有效的信息位数，详见表7.6
		×	×		接收缓冲器栈顶字的奇偶错误
		×	×	×	接收缓冲器溢出(overrun)
		×			接收缓冲器栈顶字格式错
			×	×	接收CRC校验错
				×	一帧接收结束
RR2	A A A A B B B A	×	×	×	写入WR2的中断向量，其1.2.3位可能修改过，RR2仅通道B具有

如果发送数据时让RR 0 位 6 为 1、则在同步方时发送脱空引起插入同步字符，而 CRC 码不被发出，也不建立信息结束标志；RR 0 位 2 保持为 1 指出是发生了脱空而并非信息结束。SDLC 方式中不允许脱空。

RR 0 位 7 当异步接收到断开 break 或SDLC接收到异常结束 (abort)时置 1。

读寄存器 1 包容在发送或接收操作终结时的状态信息。

RR 1 位 0 仅用于异步方式，当发送缓冲器和发送移位寄存器均变空时此位置 1。在同步或SDLC方式时这位恒为 1。

RR 1 位 1、2、3 仅用于SDLC方式，它指明最末一个装配字里包含的帧信息段 位数。

因为SDLC方式允许其信息段中的数位任意多少，所以末字的有效位数也是不固定的。RR1位1、2、3所代表的意义随每字装配位数的不同而异，表7.6列举了所有情况下它们所指明的状态。

RR1位4.5.6是可以引起中断的出错标志位，其意义见表7.5。

RR1位7仅用于SDLC方式，当一帧的闭标志收到时此位置1。

7.4 SIO的中断逻辑

Z80—SIO是设计用在Z80—CPU系统里的通讯接口器件。它符合CPU中断响应方式2的联络要求，把CPU发来的$\overline{M1}$和$\overline{IORQ}$同时为低的信息解释为中断应答信号，如果此时它正在申请中断而且IEI输入为高，它就接收这一应答信号把读寄存器RR2的内容送上数据总线，以便CPU接收它作为申请中断器件的标识——中断向量的低八位。同时释放申请中断线$\overline{INT}$，以便其它器件通过$\overline{INT}$向CPU申请中断。

RR1			8位/字		7位/字		6位/字		5位/字	
3	2	1	L	S	L	S	S	T	S	T
1	0	0	0	3	0	1	0	5	0	2
0	1	0	0	4	0	2	0	6	0	3
1	1	0	0	5	0	3	1	6	0	4
0	0	1	0	6	0	4	2	6	0	5
1	0	1	0	7	0	5	3	6	1	5
0	1	1	0	8	0	6	4	6	×	×
1	1	1	1	8	0	7	×	×	×	×
0	0	0	2	8	×	×	×	×	×	×

其中：L表示最末的一个组装字
S表示倒数第二个组装字
T表示倒数第三个组装字，当出现T时L恒为零
X表示不会发生的条件

表7.6 接收SDLC信息帧时末字里的有效数位

但SIO在请求中断的同时IEO输出低电平，CPU响应中断并不使这一引脚恢复高电平，它禁止在中断链里优先级别比它低的器件再向CPU申请中断。直到向写寄存器WR0的位3：4：5写入“复位中断优先权逻辑”命令，或Z80—CPU执行了RETI指令，或此器件被复位时，IEO才恢复为高电平。

Z80—SIO器件内有六个中断级别，是：

1）（最高优先权）通道A接收中断
2）通道A发送中断
3）通道A外部／状态中断
4）通道B发接收中断
5）通道发送中断
6）通道B外部／状态中断（最低优先权）。

表 7.7 SIO 中断总表

类型	中断	优先权	向量位 3 2 1	方式 异步	方式 同步	方式 SDLC	允许	中断产生条件	中断状态	响应
发送中断	发送缓冲器A空	2	1 0 0	×	×	×	WR1 −1=1	由发送缓冲器送往移位寄存器；脱空时CRC送出	RR1-2=1 或 RRO-0=1 （异步）	输出下一个字节到发送缓冲器。WR0-543=101禁止此中断继续产生，直到向发送缓冲器再写入下个字，脱空时，无CRC输出则RR0-2置位，CRC输出则RR0-2复位
	发送缓冲器B空	5	0 0 0	×	×	×				
	发送脱空A	3	1 0 1		×	×	WR1 −0=1	脱空后，在CRC输出前或同步/标志输出后	RRO-6=1	RR0−6置位则插入同步/标志字；复位RR0−6则脱空时插入CRC字
	发送脱空B	6	0 0 1		×	×				
	CRC发送A	3	1 0 1		×	×	WR1 −0=1 WR5 −0=1 RR0 −6=0	脱空后，CRC开始发送时	RRO-6=1 RRO-2=0	脱空后仅当RR0−6复位时发送CRC，CRC输出后产生发送缓冲器空中断，启动下次发送或停止发送
	CRC发送B	6	0 0 1		×	×				
	同步字符发送A	3	1 0 1		×		WR1 −0=1 WR4 −3.2 =00 RR0 −6=1	脱空后，同步字符开始发送时	RRO-6=1 RRO-2=1	当RR0−6为1时发送脱空则发送同步字符作为同步插入
	同步字符发送B	6	0 0 1		×					
	标志发送A	3	1 0 1			×	WR1 −0=1 WR4 −5.4 =10	脱空后标志开始发送时	RRO-6=1 RRO-2=0 如 RRO-2=1 出错	SDLC脱空后发送标志，如RR0−6复位，先发CRC再发标志，可接发下帧或停止发送如RR0−6置位不发，CRC引起错误
	标志发送B	6	0 0 1			×				
	$\overline{\text{CTS}}$ A	3	1 0 1	×	×	×	WR1 −0=1 WR3 −5=1	$\overline{\text{CTS}}$或$\overline{\text{DCD}}$的任何电平变化	RRO−3 RRO−5	$\overline{\text{CTS}}$和$\overline{\text{DCD}}$正常为低如运行时变高则异常结束，$\overline{\text{CTS}}$常用于发送，$\overline{\text{DCD}}$常用于接收
	$\overline{\text{CTS}}$ B	6	0 0 1	×	×	×				
	$\overline{\text{DCD}}$ A	3	1 0 1	×	×	×				
	$\overline{\text{DCD}}$ B	6	0 0 1	×	×	×				
	$\overline{\text{SYNC}}$ A	3	1 0 1	×	×		WR1 −0=1	$\overline{\text{SYNC}}$作为输入的电平变化	RRO−4	同步输入或未定义的开关量输入
	$\overline{\text{SYNC}}$ B	6	0 0 1	×	×					
	接收首字A	1	1 1 0	×	×	×	WR1 −4.3 =01	接收首字进入接收缓冲器	RRO-0=1	读入此字复位状态但禁止继续中断，对WR0-5.43送100允再次许中断
	接收首字B	4	0 1 0	×	×	×				

类型	中断	优先权	向量位 3 2 1	方式 异步	方式 同步	方式 SDLC	允许	中断产生条件	中断状态	响应
接收中	每一接收字A	1	1 1 0	× ×		×	WR1 −4.3 =10或=11	一个或几个接收字在缓冲器内	RR0-0=1	每个新字移入接收缓冲器时中断，RR0-0保持为1到全部字被读取为止
	每一接收字B	4	0 1 0							
	奇偶性A	1	1 1 1	× ×			WR1 −4.3 ≠00 WR4 −0=1	接收缓冲器内奇偶出错	RR1-4=1	奇偶状态位反映接收缓冲器栈顶字的状态 WR0−5.4.3=110复位此位
	奇偶性B	4	0 1 1	× ×						
	溢出A	1	1 1 1	× ×		×	WR1 −4.3 ≠00	第四个接收字符进入缓冲器	RR1-5=1	第四个接收字冲掉了第三个而保留第一、二个不变 WR0-5.4.3=110复位此位
	溢出B	4	0 1 1	× ×		×				
	格式错A	1	1 1 1	×			WR1 −4.3 ≠00	现字符格式出错	RR1-6=1	检查现在组装的字符格式出错 WR−5.4.3为110复位挂此位
	格式错B	4	0 1 1	×						
	帧结束A	1	1 1 1			×	WR1 −4.3 ≠00	一帧SDLC成功接收	RR1-7=1	WR0−5.4.3=110复位，准备接收下帧或停止接收
	帧结束B	4	0 1 1			×				
	断开A	3	1 0 1	×			WR1 −0=1 WR4 −3.2 ≠00	检测到断开或断开恢复	RR0-7=1	异步输入被挂起来，程序进行处理
	断开B	6	0 0 1	×						
	异常结束A	3	1 0 1			×	WR1 −0=1 WR4 −5.4 =10	SDLC输入检测到异常结束字符	RR0-7=1	SDLC输入异常结果，程序进行处理
	异常结束B	6	0 0 1			×				
	$\overline{\text{SYNC}}$A	3	1 0 1	× ×			WR1 −0=1	SYNC作为输入脚的跳变	RR0−4	外同步的字符边界信号，其它为未定义的开关量输入
	$\overline{\text{SYNC}}$B	6	0 0 1	× ×						
	$\overline{\text{DCD}}$A	3	1 0 1	× ×		×	WR1 −0=1	$\overline{\text{DCD}}$或$\overline{\text{CTS}}$的任何电平变化	RR0−3 RR0−5	系统故障或控制信号由程序进行处理
	$\overline{\text{DCD}}$B	6	0 0 1	× ×		×				
	$\overline{\text{CTS}}$A	3	1 0 1	× ×		×				
	$\overline{\text{CTS}}$B	6	0 0 1	× ×		×				

各种中断所包含的范围见表7.7。如果允许状态影响向量，则各种中断会修改中断向量的第3、2、1位，因之可以产生八种向量地址，每个通道的接收中断具有两个向量，其它种类的中断均各有一个向量。两个接收向量用以区分接收数据和该数据的出错状态。

表7.7的“方式”一栏标明各种中断具有意义的工作方式。某列上的“×”表示在此种通迅规程中发送或接收数据应该考虑此类中断。没有×的地方并不一定就不发生这类中断，而是意味着这种中断条件没有应用意义，因此应当禁止它产生，Z80－SIO也通常不引起这些中断。

表7.7中“允许”一栏标示每种中断能够产生的控制字写入条件。与标示条件相反时便禁止了中断，当有数个条件时，它们的同时成立是产生中断的必要条件。

实际中断申请触发的契机在“中断产生条件”一栏内注明。应仔细研究这一列以确切弄清中断请求产生的时刻，有些中断产生的条件并非是显而易见的。例如，“发送缓冲器空”中断并不由空的发送缓冲器产生，而是在数据由发送缓冲器向发送移位寄存器传送时产生的，所以，如果从未向发送缓冲器装入任何数据的则不会产生“发送缓冲器空”中断，虽然此时发送缓冲器肯定是空的。然而，一旦”发送缓冲器空”中断出现了，除非向WR0位5、4，3写入101复位这一中断，或者实际向发送缓冲器守入数据，否则这一中断总是持续地出现。

表7.7里“中断状态”一栏指明确认各类中断所需的标志，一般来说这是容易明白的。只需对奇偶性，溢出和格式这三种接收出错中断略加解释。这些出错状态是保存在与接收数据缓冲栈相伴的错误状态缓冲栈里的，读寄存器1的4、5、6位里反映的是接收栈顶数据所对应的状态。当读取接收数据的时侯，接收逻辑也刷新了出错缓冲器的内容，使之反映下一个接收数据的状态。

例如在异步通讯时，考虑如下的状态

									溢出	格式	奇偶	
接收缓冲器3	0	1	1	0	0	0	1	1	0	0	0	出错缓冲器3
接收缓冲器2	1	1	1	0	0	0	1	1	0	0	1	出错缓冲器2
接收缓冲器1	1	0	0	0	0	1	0	0	1	0	0	出错缓冲器1

在接收栈顶（缓冲器3）里的数据无错误，次一个字节有奇偶性错误。它们都会依次产生接收中断，中断向量可以标志是正确的接收还遇到了错误状态。也就是说，接收出错并不产生额外的中断请求，它只是在通常的接收中断时修改中断向量。

由奇偶出错中断请求出现后又已接收了两个或更多的数据字，因为栈里第三个字标示有溢出状态，这意味着至少有一个字溢出并丢失了。

响应上述中断请求的服务程序对第一个字节将检测到正确的接收状态，对次一字节将发现奇偶性出错，而在第三个字节发现有溢出。应当注意，中断服务程序须得在读取字之前读状态，如果读取了接收字后再想读状态，则读的是缓冲器里下一个字的状态了。

表7.7里“响应”一栏说明中断的含意和中断服务程序可能的响应情况，下面我们统观一下SIO在三种通讯方式中的中断情况。

SIO 发送和接收逻辑绝大部分在三种工作方式中是共同的，对于中断处理亦如是，我们首先来看这些共同的部分。

$\overline{DCD}$和$\overline{CTS}$可分别用来启动接收和发送逻辑，如果程序指定它们自动开放，那么当接收逻辑已被编程开放时，它等待$\overline{DCD}$变成低电平时才检则RxD的接收数据流，同样，当发送逻辑编程开放时，它等待$\overline{CTS}$变低才通过TxD发送数据。

注意当自动开放$\overline{DCD}$和$\overline{CTS}$时并不等于就自动开放了接收和发送逻辑，接收和发送逻辑有其各自的控制位（接收在WR3位0，发送在WR5位3），如果控制位未置1，相应的$\overline{DCD}$和／或$\overline{CTS}$信号对通讯工作不能起作用。

$\overline{DCD}$和$\overline{CTS}$是否用于通讯的自动开放与它们的电平跳变引起中断请求无关。如果允许外部／状态中断，则$\overline{DCD}$和$\overline{CTS}$的跳变将引起中断申请，而不管它们是否自动开放接收和发送逻辑。

$\overline{DCD}$和$\overline{CTS}$的中断服务需求取决于它们是否用于自动开放接收和发送操作，如果用于自动开放，则$\overline{DCD}$的中断请求表示接收开始，$\overline{CTS}$中断请求开始发送。此后在所有数据接收或发送完毕之后，$\overline{DCD}$或$\overline{CTS}$请求中断则表示接收或发送已完全成功。但若电平跳变在通信中途产生，意味着线路另一端的设备已断开，Z80—SIO 的发送或接收也须中途结束。如果$\overline{DCD}$和$\overline{CTS}$不用于接收和发送的自动开放，则它们成为一般的控制输入，外部环境可用之通过SIO 发出中断请求。

如果希望接收字符装配好后产生中断，有两种选择：

1）可以只在接收到信息首字后产生中断，而嗣后的接收字均不申请中断。在表7.7里这称为“接收首字”中断。

2）可以在每一个接收字装配好后都产生中断。

如果采用接收首字中断，则若不执行一特定命令以复位这类中断时，对接踵而来的接收字符不再产生中断。但奇偶性，格式和溢出错中断对相继的接收字仍为有效。

在异步通讯方式中，当接收逻辑检测到高电平的传号信号变低时，认为这是“断开”信号。它常常用来唤起串行 I/O设备的注意。接收逻辑检测到断开后便发出中断请求，此后当RxD 由低变到高时，又产生中断以标志断开的结束。断开中断属于外部／状态中断之一种，断开和断开的结束用同一个状态位（RR0位7）和同一个复位命令处理，所以在断开中断时，须执行一条复位外部／状态中断的命令，否则断开结束时就不能产生中断了。

Z80—SIO发送逻辑可以发送断开信号，当对 WR5 位 4 写入 1 时该通道就输出断开，对该位再写入 0 就结束了断开状态。

每当数据字节由发送缓冲器传送到发送移位寄存器时发送逻辑就产生中断请求，这是要求向发送缓冲器输出下一个字节的信号，发送时奇偶性、格式、溢出错等均无意义，发送溢出不能产生中断；而若发送时奇偶性或格式出错，则说明此器件工作不正常。

在发送缓冲器空中断之后，你可以在发送逻辑输出整个字符的期间内向缓冲器装入下一个字，若在此期间内未向发送缓冲器输出任何数据字节，则出现了“脱空(underrun)”状态。所以，发送缓冲器空中断意味着发送逻辑从发送缓冲器取了一个字并正通过 TxD 向外发送，而脱空中断意味着当发送逻辑回过来取下一个输出字时发现发送缓冲器竟然是空的。

脱空仅在同步和SDLC方式才有意义，异步方式中无所谓“脱空”，因为它允许发送字之间有任意长时间的传号状态。Z80—SIO同步和SDLC发送逻辑用脱空状态来结束信息的发送和达到特定的控制目的，详见后述。

DCD、$\overline{\text{CTS}}$和$\overline{\text{SYNC}}$输入在整个发送和接收通讯过程中都被监测着，如果允许外部／状态中断，则它们中任何一个产生电平变化（高变低或低变高）就申请中断。当$\overline{\text{SYNC}}$作为输出时，其电平跳变不产生中断请求。

7.5 SIO的编程和工作方式

和其它I/O接口一样，控制SIO操作的程序包括三部分：初始化，运行和结束。初始化就是向每个通道的写寄存器写入适当的控制字以指定其工作方式。

SIO控制程序的运行部分是很简单的，为了在CPU和SIO之间传送数据，可以选用如下的方式：

1）可以简单地使用I/O指令来发送和／或接收并行数据。此时需将$\overline{\text{WAIT}}$/$\overline{\text{RDY}}$信号编程为$\overline{\text{WAIT}}$请求信号，可参见前面（7.2）对该信号引脚的详细讨论。

2）可以用查询的方法来进行程序控制的输入输出，读寄存器RR0位2标示发送逻辑是否在等待下一个数据，而RR0位0标示接收缓冲器里是否有可传送给CPU的有效接收数据。

3）可以利用中断逻辑控制数据的传送，用发送缓冲器空中断服务程序将下一个数据字节送往发送逻辑，而在接收字符可用中断服务程序里从接收逻辑读进接收数据字节。

4）也可以使用DMA逻辑来传送数据，但应注意只能或者用在发送方面、或者用在接收方面不能对一个通道同时采用DMA逻辑来发送和接收并行的数据。采用DMA时，须将$\overline{\text{WAIT}}$/$\overline{\text{RDY}}$信号编程为DMA请求。如果要通过DMA接收数据，则应将SIO编程为接收首字中断，在服务程序里启动DMA控制器。如果通过DMA发送数据，或者可以在启动发送同时初始化DMA控制器，或者在用$\overline{\text{CTS}}$启动发送时，在$\overline{\text{CTS}}$中断服务程序里初始化DMA控制器。

对于各种工作方式，通讯结束的状态是一样的：发送或接收操作或是正确完成、或是出现错误。如果成功结束，则可以启动下一次传送，也可以关闭发送或接收逻辑。如果检测到出错，通常中止发送或接收，在发送时可以本机重发，接收时则要通报发送端重发。

同步和SDLC方式使用专门的通讯字。在同步方式中要用同步字符，它通常取01101001的形式。在SDLC方式中使用形为01111110的标志字符，还涉及到地址字段。同步、标志和地址字节存放在写寄存器WR6和WR7里，我们在这里对这两个寄存器的功能还要补充说明两点：

我们在同步和SDLC方式中可以使用同步、标志和地址字的概念，SIO逻辑从WR6和WR7里取这些通讯字，例如，在SDLC方式中从WR7里取标志。但并没有从WR7里取数并校核它是否确为规范的标志字符的逻辑，无论需要输出或在输入数据流里搜索标志，通道逻辑从WR7里取字并把它就当作有效的标志。类似地，同步通讯方式时从WR6和WR7里取同步字，而并不校验它们是否符合任何规程标准。在双同步情形下这可能是有益的，例如，发送脱空时，可以不必发送两个一样的同步字符，而在WR6里放个特定的DLE（数据传送漏

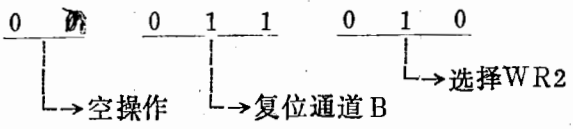
失）字符，在WR7里放同步字符，这样在脱空时就输出了DLE—SYNC（数据传送漏失——同步）字符对。

当发送逻辑从WR6或WR7中取字符时，可以看成字符先由WR6或WR7传送到发送缓冲器，然后再传送到发送移位寄存器。因此，在同步或SDLC方式中输出同步、标志等专用字符时，正象输出其它数据字一样，也会产生发送缓冲器空中断和发送脱空中断。但毕竟略有所不同：当发送缓冲器装满后则WR6或WR7的内容不被送入发送缓冲器内，发送缓冲器也能够由CPU送入新的发送数冲而冲消插入的同步字符。

7.5 .1 SIO异步操作

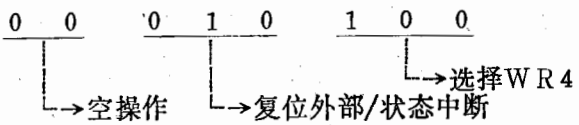
异步发送和接收操作的初始化过程逻辑见图 7.3。首先，须将通道初始化为异步工作方式，下面我们看初始化通道B的步骤：

1）假定 SIO 全部采用中断逻辑方式工作，首先，复位通道 B 并准备向 WR2 写中断向量：

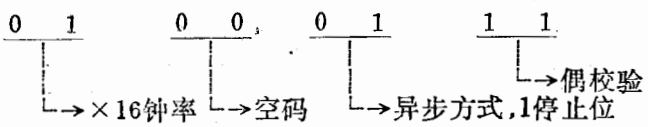


2）向WR2输出中断向量。

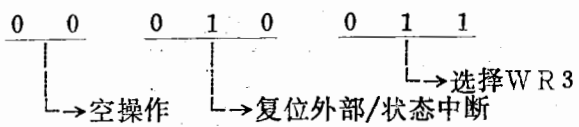
3）在(2)中访问WR2后，通道逻辑自动转回WR0、向WR0写控制字复位外部／状态中断，并准备向WR4写控制字：



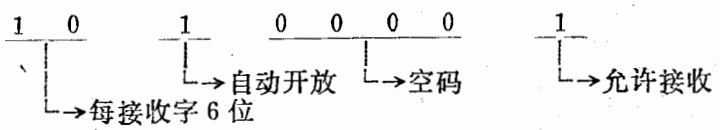
4）向WR4写控制字选定异步方式，指定奇偶校验方式、停止位格式、钟分频率：



5）向WR0 再次写入复位外部／状态中断命令，并准备向WR3写控制字：

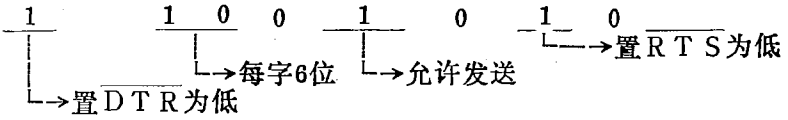


6）向WR3写控制字指定接收参数



7）向WR0写05H，选择WR5

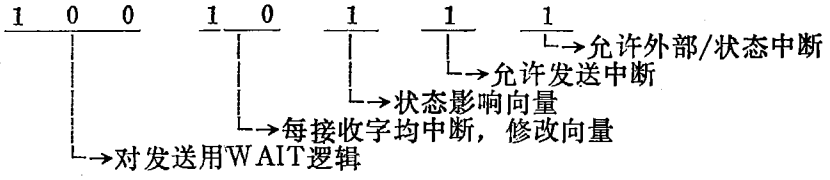
8）向WR5写控制字指定发送操作方式，设定$\overline{RTS}$和$\overline{DTR}$输出信号：



其它各位为空码。

9）向WR0写01H，选择WR1

10）向WR1写控制字控制中断和$\overline{WAIT}/\overline{RDY}$逻辑：



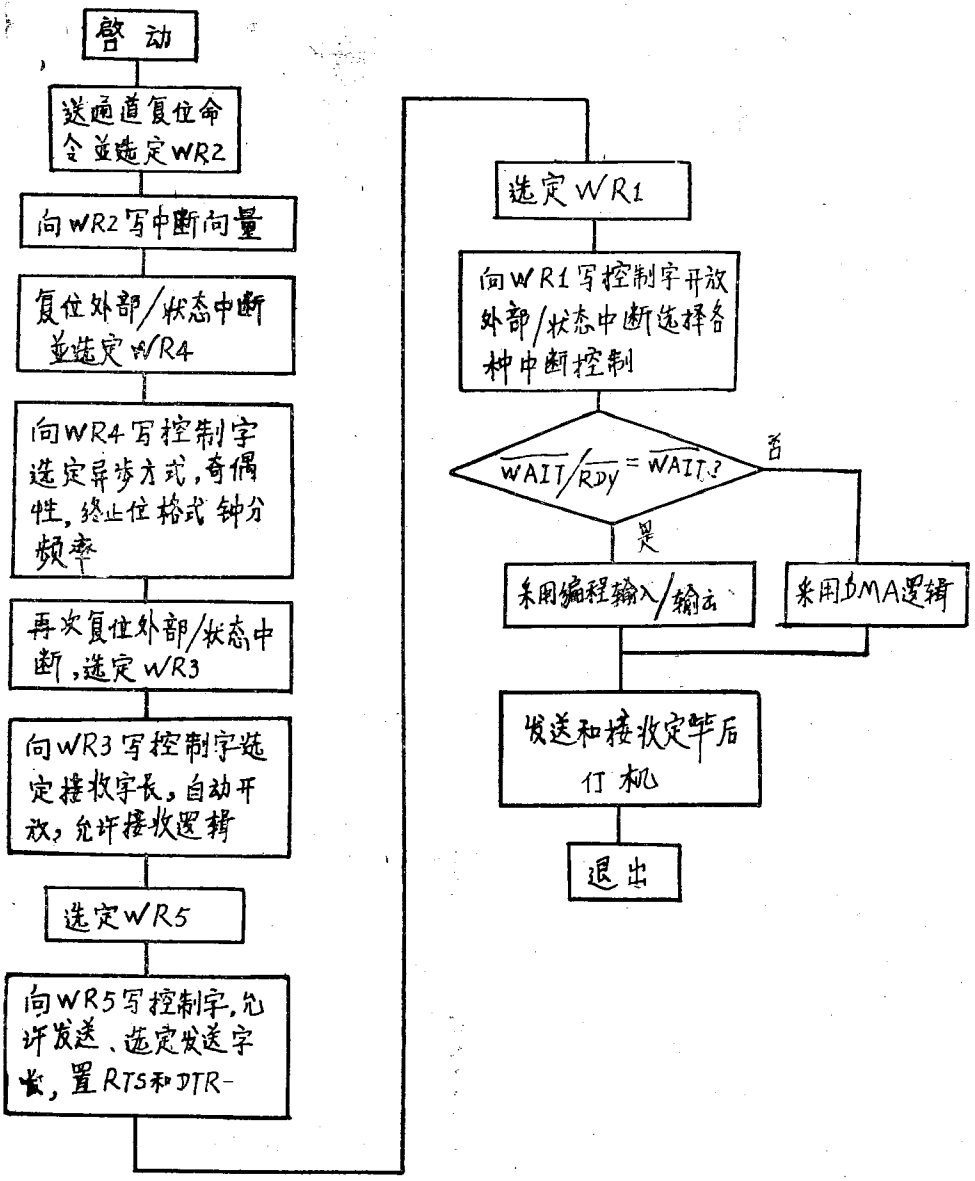


图 7·3 SIO异步操作编程逻辑

这样，异步发送和接收就可以开始了。在发送字符开始之前，TxD向外输出高电平的传号信号。

在第6步中指定由$\overline{CTS}$和$\overline{DCD}$控制通讯操作，所以虽然随时可把待发送的首字送往发送逻辑，但在$\overline{CTS}$输入为低之前它将停留在发送缓冲器里。一旦$\overline{CTS}$变低，数据就按指定的格式由缓冲器输出来。如果不指定自动开放，则数首字一写入发送缓冲器就立刻发送出去。

在异步发送和接收中不使用SYNG信号，该引脚成为可引起中断请求的外部控制信号。

接收逻辑在常规情况下由RxD检查到高电平信号。由于$\overline{DCD}$指定自动开放，在$\overline{DCD}$变低前接收逻辑对RxD信号没有响应，当$\overline{DCD}$不用来自动开放时，接收逻辑一当开始工作就监测RxD输入，当RxD首次变低时，接收逻辑识别出起始位并开始装配输入的数据。

7.5 2. SIO同步通讯操作

在同步工作方式的初始化过程中不但需要如前段对异步方式所列举的程序步骤，而且还要增添向WR6；以及可能还要向向WR7；写入同步字符的程序步。

同步通讯方式包括外同步、同步单步和双同步三种类型，它们各自对WR6和WR7的使用有所不同，下表列出了在三种类型下对WR6和WR7内容的使用情况。

工作方式	WR6内容	WR7内容
单同步	发送的同步字符	接收时用的来同步字符
双同步	接收和发送的同步字首次节	接收和发送的同步字次字节
外同步	发送时插入和接收时自动剔除的同步写符	不用

当把一个通道初始化为同步通讯工作方式时，使用的逻辑相似于图7.3对异步通讯所阐述的步骤，以下只对同步通讯的一些特殊点进行说明。

开始工作时，同步接收逻辑处于搜索（Hunt）状态，以期于输入数据流中找到字符边界。如果使用外同步，则$\overline{SYNC}$作为输入控制信号，应在同步字符收到后下一接收字首位到来之前，置为低电平。此时如果允许外部／状态中断，则就会产生外同步中断申请。

事实上，在外同步方式时，建立同步与$\overline{SYNC}$输入引起的中断请求并无必然联系，只要允许外部／状态中断，则$\overline{SYNC}$输入的电平跳变总会请求中断，把某个$\overline{SYNC}$中断请求解释为初始同步中断是程序逻辑处理的问题。

如果接收逻辑工作在单同步或双同步方式，则当在接收数据流中搜索到单字节或双字节同步字符时就自动进入了字符同步状态。此时$\overline{SYNC}$引脚变为输出信号，在同步字符检测到时输出低脉冲。当最初的同步字符检测到之后，接收逻辑仍在数据流中检测相继的同步字符，每次检测到就使$\overline{SYNC}$输出低电平脉冲。$\overline{SYNC}$作为输出时其电平跳变不申请中断。当第一个接收字符装配完毕后接收逻辑便产生它的首次中断请求。

如果选择外同步方式，则无须使用WR6里的同步字符去检测信息的开始，因为$\overline{SYNC}$信

号控制了接收逻辑的起始工作，它也可以作为一段通讯结束的标帜。但可能仍然需用WS6里的同步字以剔除插入在一段通讯字里的同步字符。

接收逻辑对输入数据流连续地进行循环冗余码计算并与正确的结果进行比较，校验的正确与否在读寄存器RR1位6上反映出来。

当检测到或隐含意味着信息结束时接收逻辑并不自动停止接收和装配字，如果要停止接收，须向WR3位0输出0以禁止接收。

接收逻辑的工作方式是很直观的：从接收数据流里按选定方式组装成为数据字节，然后送往接收缓冲器，同步字符并不一定要自动由接收数据流中剔除，如果需要自动剔除的话则要在WR3位1写入1。无论在初始化过程中选择或不选择自动剔除，开放或关闭这一功能是随时可以进行的，如果开放这个功能，接收逻辑把装配好的数据节与WR7的内容进行比较（无论它的内容是什么），如果符合，则接收的字符被剔除掉。

可以选择标准的循环冗余校验算法（循环多项式CRC—16：$X^{16}+X^{15}+X^2+1$），也可以选择标准的SDLC循环冗余校验算法（循环多项式：$X^{16}+X^{12}+X^5+1$），但接收和发送须采用同一算法。可以对所有通讯字进行CRC计算，也可以只对某些通讯字进行这种计算。

如果选择由$\overline{CTS}$和$\overline{DCD}$自动开放，则仅当$\overline{CTS}$由高变低时同步发送逻辑才开始发送工作，否则一当允许发送时发送逻辑就开始工作，可以选择单同步、双同步或外同步发送工作方式。如果选用外同步方式，则在发送开始之前或非信息结束的脱空（Underrun）之后，装在WR6里的同步字符被接续地发送出去。选用单同步方式时情况也一样。而在双同步方式里，WR6的内容，继而WR7的内容，在信息的开始和结束以及脱空时被发送出去。

值得再次强调，无论在信息的开始，结束或插在中间，发送同步字符似乎是通过了发送缓冲器，然而，如果缓冲器不空，则发送的是它的内容而不是WR6或WR7里的同步字符。

当一个数据字节由发送缓冲器送往发送移位寄存器时，发送逻辑产生发送缓冲器空中断请求，如果允许这一中断的话，则可以在服务程序里向通道送入下一个发送字节。脱空（Underrun）和发送结束共占一个状态和中断位（RR0位6）。最初，脱空／信息结束（RR0—6）位置1，如果保留这一状态不变，则每当发生脱空时，可以产生中断请求（如果允许时），并在确实是脱空的状态下，就插入发送一个或更多的同步字符。如果将RR0位6清零，则发送逻辑把下一次的脱空当作信息段的结束，它先把计算的循环冗余字发送出去，然后接着发送一个或多个同步字符。每次同步字输出时仍产生发送缓冲器空中断请求。

7.5 3. SIO的SDLC通讯操作

把一个通道初始化为SDLC工作方式的编程步骤也与图7.3对异步方式所描述的相同，下面仅对SDLC的特殊方面进行说明。

应当向WR6里写入地址段字节，而向WR7里写入标准的标志节（01111110）。SDLC发送逻辑在每帧信息的开始和结尾输出WR7的内容无论它是什么，SDLC接收逻辑把输入数据流与WR7的内容进行比较以检测每帧的开始标志和关标志。

如果选定了按地址的接收方式，SDLC接收逻辑可以把一帧的开标志之后的接收字节与WR6的内容进行比较，如果相符，或者该接收字是OFFH，则接收这一帧，否则就不接收它。

接收逻辑检测到标志字就毫不含糊地判定一帧的结束，因为SDLC规程不允许脱空，所以没有描入的标志字。于是，在开标志之后再检测到的标志字就一定是关标志，关标志之前的两个字节是循环冗余校验字，接收逻辑把它与自己的计算结果相比较。

如果SDLC接收逻辑检测到异常结束（Abort）字（01111111），则产生异常结束中断请求，但在此之前已接收的一帧的前半部分由程序决定接收或拒绝，在异常结束产生时，SDLC接收逻辑产生中断请求，停止接收数据，开始搜寻下一个标志字。

在开标志之后的第一个数据字节是地址段，接收到地址段后就产生首字中断。当接收逻辑检测到一帧的关标志时产生特别的帧结束中断。

SDLC发送逻辑象同步发送逻辑一样利用脱空状态位来终止一个信息帧。注意如果没有向写寄存器WR0位6和7写入11，则SDLC发送逻辑在产生脱空出错时输出一个标字，然后跟着一个异常结束字。因为对方终端可能把在脱空之前输出的两个字节误当作更前面数据的CRC字，所以异常结束字可以防止以为接收到一个完全有效信息帧的错误。正常通讯时应使SDLC发送逻辑把脱空处理为信息的结束。

第八章　模拟数字转换芯片

ADC 0809

8.1 概　　述

客观世界的许多变化量是连续性的，例如温度、压力、电压、电流、相角等，要使用数字计算机对它们进行测量、计算、控制等，就需要先把这些连续变化的量转变为离散的数字量，通称为模（拟）数（字）转换（AD变换）。

ADC0809（以下简称ADC）是进行模数转换的集成电路器件，它具有八个模拟量输入线（八通道），可在程序控制下选择其中之一进行模数转换，得到八位二进制数字量，送到微型电脑的数据总线上，供CPU进行处理。

ADC0809是National半导体公司的产品，不属于Z80的芯片系列，虽然它的设计能与多数微计算机的大部分总线兼容，但不具备Z80系列所要求的中断键和中断申请机构．把它纳入Z80微型电脑中则需特别加以考虑，可以不使用中断而采用程序查询的方式进行服务，也可以借助Z80系列的I/O接口芯片而纳入中断系统当中，CWC—80系统采取了后一种方法。

8.2 ADC的方框图及引脚

ADC的方框图示于图8.1。八个模拟输入信号在多路开关地址寄存器控制下经8.1模拟

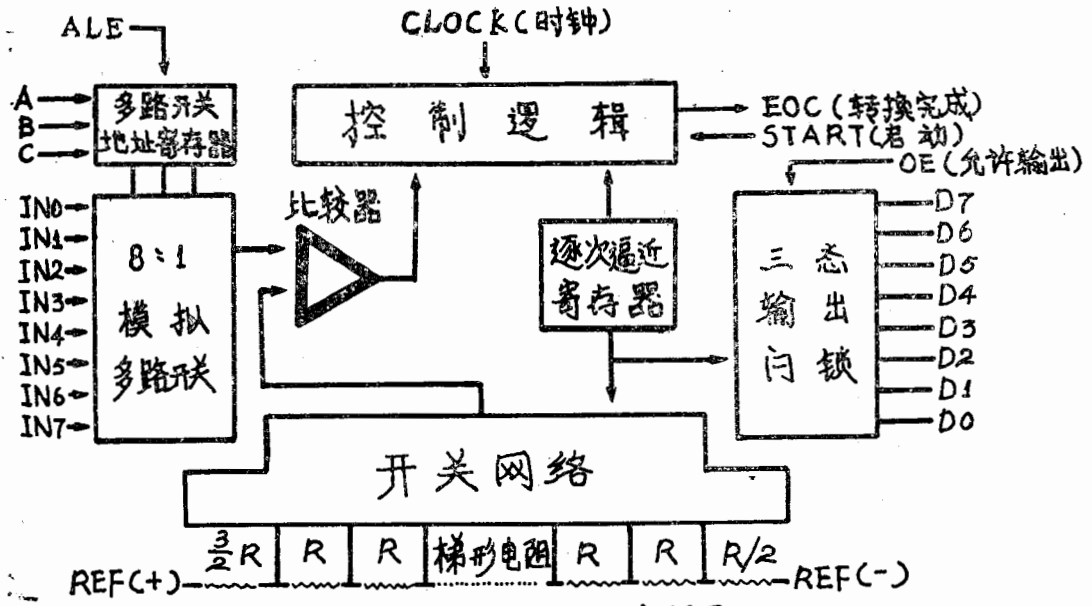


图 8.1 ADC0809的方框图

多路开关选通其中之一送往比较器，在控制逻辑作用下，逐次逼近寄存器内的数值不断变化（增加），使开关网络输出的电压也随之而上升，通过此较器鉴别出两个电压相第或最接近的状态，并将此时的通道寄存器内容送往输出闩锁，便可使转换后的数字量送往CPU以待处理。

ADC的引脚布置如图8.2所示，这些引脚的作用如下：

1.IN0—IN7为模拟量输入线，待转换的模拟电压量由此引入系统。

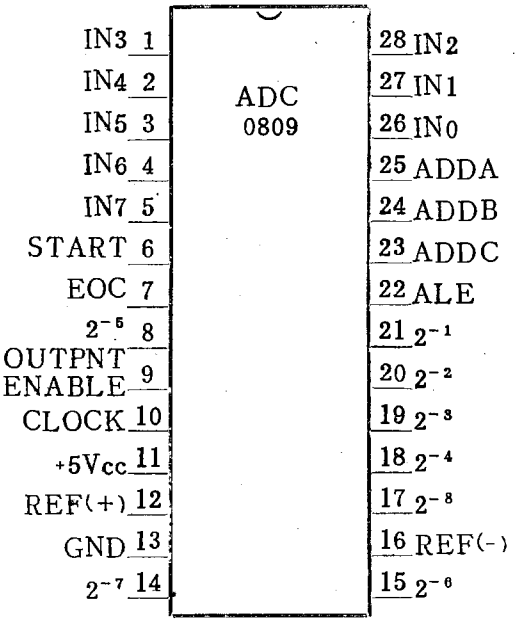


图8.2 ADC0809的引脚布置图

2.D0—D7 为数据输出线(三态)。用来把转换后的数字量送到微型电脑的数据总线上。

3.A、B、C、为输入多路开关地址选择线，表8.1所示是地址线状态与选择通道的关系。

引脚				选中输入	地址元单
A_7-A_3(*)	C(A2)	B(A1)	A(A0)		
10011	0	0	0	IN1	9 8
10011	0	0	1	IN2	9 9
10011	0	1	0	IN2	9A
10011	0	1	1	IN3	9B
10011	1	0	0	IN4	9C
10011	1	0	1	IN5	9D
10011	1	1	0	IN6	9E
10011	1	1	1	IN7	9F

表8.1 ADC的地址选择逻辑

(*)CMC—80的ADC是按此地址连接的，不同的微型电脑A7—A3可为不同值。

4.ALE为地址闩锁输入线，在该信号的上升沿处把A、B、C 三选择线的状态闩锁入寄存器里。

5.START为启动转换输入线，在该信号的上升沿清除ADC的内寄存器，而在下降沿启动内部控制逻辑开始A／D转换工作。

6.EOC为转换完成输出线，该信号上升沿表示内部A／D转换已完成。

7.OE为允许输出信号输入线，在OE为1电平时，数据线D0—D7脱离三态，把内部转换的数据送上数据总线。

8.CLOCK为转换定时时钟输入线，转换速度取决于钟频，通常其频率不能高于640KH，此时转换速度约为100uS

9.REF(+)和REF(−)为模拟信号参考电压输入线。它们可以与本机的电源 Vcc和地脱离，但REF(−)不得为负值，REF(+)不得高于Vcc，$\frac{1}{2}$〔REF(+)+REF(−)〕与$\frac{1}{2}$Vcc之差不大于0.1伏。

10.其它Vcc为电源（+5V），GND为地。

8.3 ADC的作操说明

ADC的操作比较简单，先用ALE和A、B、C输入线把要进行转换的通道地址闩锁入ADC芯片，然后发出启动（START）信号，则ADC自动进行转换，转换完成后，EOC线输出逻辑1.可以利用它向CPU发出中断申请，也可以等待CPU查询该信号再进行服务，CPU 通过OE信号便可读出转换的数据3。

ADC与各种微处理器连接的方法是多种多样的。在CMC—80微型电脑中，用门电路组合逻辑指定通道地址，启动转换和在完成转换后读入数据，而转换完成信号EOC接到CTC的CLK／TRG3引线，可以利用它向CPU申请中断。例如要将通道5的模拟量转换后送到累加器A内，中断服务程序LDAD入口地址在3706和3707单元，则可编制如下程序来进行A/D 转换工作。

```
LD      HL, LDAD        送中断服务程序入口地址
LD      (3706H), HL
LD      A, 37H          送中断向量到CTC
LD      I, A
LD      A, 00H
OUT     (8CH), A
LD      A, 0D5H         设定工作方式
OUT     (8FH), A
LD      A, 01H          装入计数值为1
OUT     (8FH), A
OUT     (9DH), A        输出通道地址9D，指定转换IN5，A的值不起作用
IM          2
EI
HALT                    暂停，等待中断
```

中断服务程序为

```
LDAD    IN      A, (9DH)
        RETI
```

则便可完成所要求的A／D转换工作。