微处理机实用教材

TP801

Z80单板计算机基础知识

北京工业大学微处理机应用研究室编

前　言

自从一九七一年第一个微处理器 Intel4004 问世以来，微处理器/微型计算机得到了飞速的发展。近年来，每年有一亿多只微处理器投放市场，这种新型器件几乎渗透到各个技术领域。

我国广大技术人员迫切需要掌握和推广应用这种技术。为此，我们曾先后在校内外面向教师和技术人员举办了几次讲座，并为我校工业自动化系七九届研究生开设了《微型计算机化设计》课程 。在此基础上，我们配合 TP801—Z80 单板计算机的研制、生产，编写了这本《微处理器实用教材——基础知识和 Z80器件的使用》，作为《TP801—Z80 单板计算机使用手册》的补充，供初学者学习和广大TP801 的用户参考使用。

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，它们分别只具有五条和十六条指令。通过这一章的学习，读者可以大致了解微型计算机总体结构的全貌以及其中各主要部件的工作原理。在第二种模型计算机中还引入了子程序的调用过程以及栈的概念。第三至第六章介绍 Z80 系列中的三种常用器件(CPU,PIO和CTC)的主要特性和操作说明。

本书和《TP801—Z80 单板计算机使用手册》配合，将作为 TP801 用户的培训教材，或供初学者自学选用。讲授这两本书约需 40～60 学时，再配合进 行 适量的实验课程，可使学员获得使用微型计算机的初步知识和技能。

本书第一、二、五、六章由徐家栋编写，第三、四章由颜超编写。侯伯文参加了本书的修改定稿工作。由于我们的水平有限，书中错误在所难免，恳切希望读者指正。

北京工业大学工业自动化系
微型机应用与自动化研究室
一九八一年四月

目　录

第一章　基础知识

1.1　数制及不同数制間的换算

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 不是表示一千零一，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（相应于十进制数中的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*

---

* 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。

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

$= 9CH$

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、二一十进制数（BCD码）

在这种数制中，用二进制数来表示十进制数字。例如

$97D = 10010111\ BCD$

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：1101B

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：$y = A \times B$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。

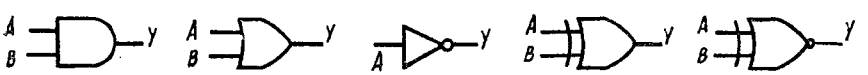
2．含义：只要输入中有一个为逻辑1，输出即为逻辑1。

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

輸　入		輸　出				
A	B	与門	或門	非門 $(Y = \overline{A})$	异　門	同　門
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



(a) 与門　(b) 或門　(c) 非門　(d) 异門　(e) 同門

图 1.1　常用邏輯电路

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：$y = A \oplus B$

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：$y = A \odot B$ 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

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

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

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。在工程上，可以使用单一的与非门来实现各种逻辑运算。

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：

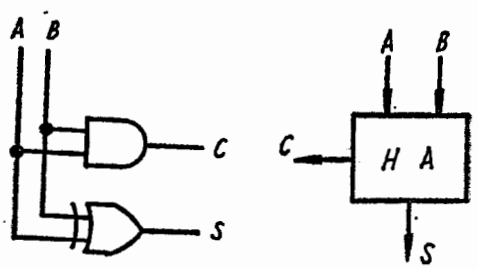
	（1）	（2）	（3）	（4）
A	0	0	1	1
+ B	+ 0	+ 1	+ 0	+ 1
C S	0 0	0 1	0 1	1 0
	↓ ↓	↓ ↓	↓ ↓	↓ ↓
	C S	C S	C S	C S

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 = ?

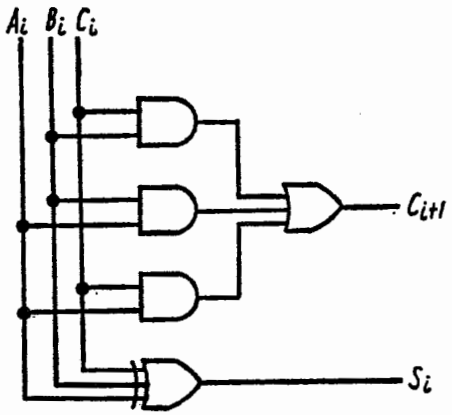
下面按照小学生的习惯进行演算，即逐位进行运算。先从最低位开始，$A_0 + B_0 = C_1 S_0$，S_0 是本位和，C_1 是对下一位的进位。其次 $C_1 + A_1 + B_1 = C_2 S_1$，依此类推。

C_i	1←		0←		1←		1←	
A_i			1		0		1	1
$+B_i$			1		0		0	1
S_i		1		0		1		1
		↑	0	↑	1	↑	0	↑ 0
		C_4	↑	C_3	↑	C_2	↑	C_1 ↑
			S_3		S_2		S_1	S_0

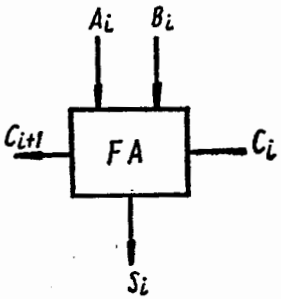
可见，多位数相加与一位数相加的差别在于：前者为三个一位数相加，后者只有两个一位数相加。三个一位数相加有八种情况，其结果如表1.2所示。其逻辑式如下：

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

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



(a) 电路



(b) 框图

图 1.3 全加法器

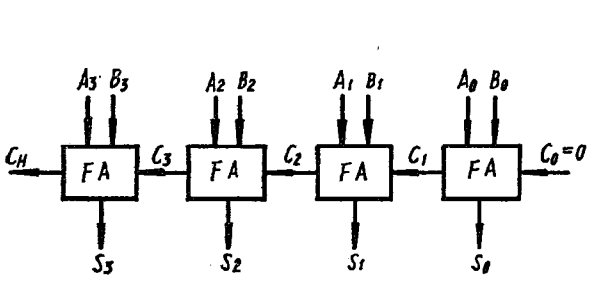
表 1.2 全加法器真值表

輸入			輸出	
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

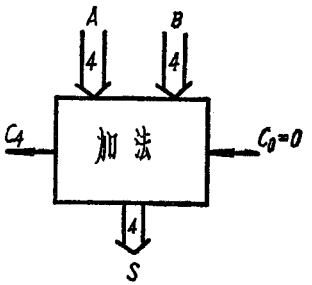
图1.3（a）示出其逻辑图，图1.3（b）示出其方框图，通常叫做全加法器。由此可以得到两个4位数相加的二进制加法器，如图1.4（a）所示。将此图简化可得图1.4（b）所示的方框图。

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，但要用电子线路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

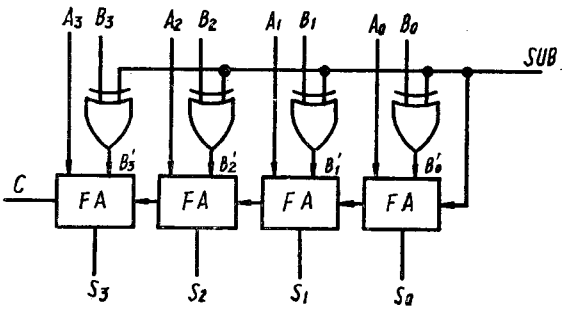
但是如何将 3 (0011) 转换 成 -3 (1101) ？简单地说，就是将 0011 进行“求反加 1 ”运算：

求反 0 0 1 1

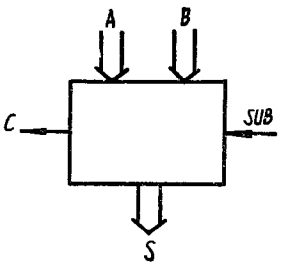
↓ ↓ ↓ ↓

1 1 0 0

<table>
<caption>表 1.3 数的补码表示</caption>
<tr><th>数</th><th>补 码</th></tr>
<tr><td>-8</td><td>1000</td></tr>
<tr><td>-7</td><td>1001</td></tr>
<tr><td>-6</td><td>1010</td></tr>
<tr><td>-5</td><td>1011</td></tr>
<tr><td>-4</td><td>1100</td></tr>
<tr><td>-3</td><td>1101</td></tr>
<tr><td>-2</td><td>1110</td></tr>
<tr><td>-1</td><td>1111</td></tr>
<tr><td>0</td><td>0000</td></tr>
<tr><td>1</td><td>0001</td></tr>
<tr><td>2</td><td>0010</td></tr>
<tr><td>3</td><td>0011</td></tr>
<tr><td>4</td><td>0100</td></tr>
<tr><td>5</td><td>0101</td></tr>
<tr><td>6</td><td>0110</td></tr>
<tr><td>7</td><td>0111</td></tr>
</table>



(a) 电路



(b) 框图

图 1.5 加法/减法器

加1

$$\begin{array}{r} 1\ 1\ 0\ 0 \\ +\qquad 1 \\ \hline 1\ 1\ 0\ 1 \end{array}$$

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$B_i'=\overline{B_i}$（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。

1.4 触发器与寄存器(Flip-flop and Register)

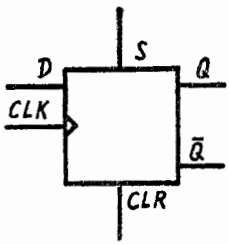
一、触发器

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：

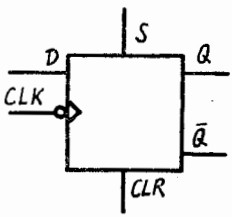
1．D触发器

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。当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

輸入		輸出
CLK	D	Q
↑	1	1
↑	0	0
其他	×	原状

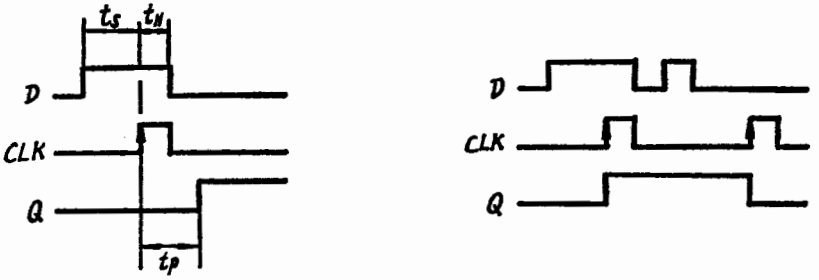
图1.7示出D触发器操作的时间图。图1.7（a）表明，数据线D的建立必须领先于CLK正跳变，这段时间称为建立（Setup）时间t_s。数据线D的撤除必须滞后于CLK正跳变，这段时间称为保持（Hold）时间t_H。如果不满足上述条件，D触发器就不能正确动作。输出信号Q的变化滞后于CLK正跳变，这段时间称为传播延迟（Propagation delay）t_P。在以后的叙述中，将忽略这些特性。图1.7（b）示出了不考虑t_P的操作时间图。

图1.6（b）示出另一种D触发器，与前者的差别仅在于CLK端上多一个小圆圈。在数字电路中，小圆圈表示反相，因而这种D触发器的CLK信号在负跳变时为有效。同理，若CLR，S端上也加上小圆圈，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触发器的操作时間图

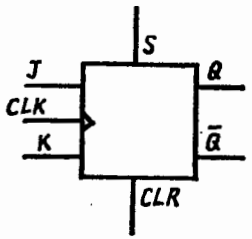


图 1.8　JK触发器

表 1.5

輸　入		輸　出
J	K	Q
0	0	不　变
0	1	0
1	0	1
1	1	翻　轉

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

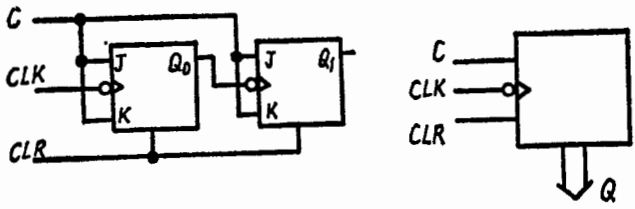
1）电路图及方框图：如图1.9 (a)　(b) 所示。

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。

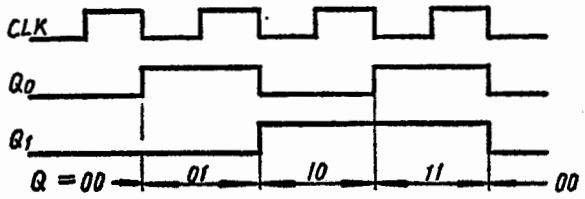
2．缓冲与移位寄存器

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。

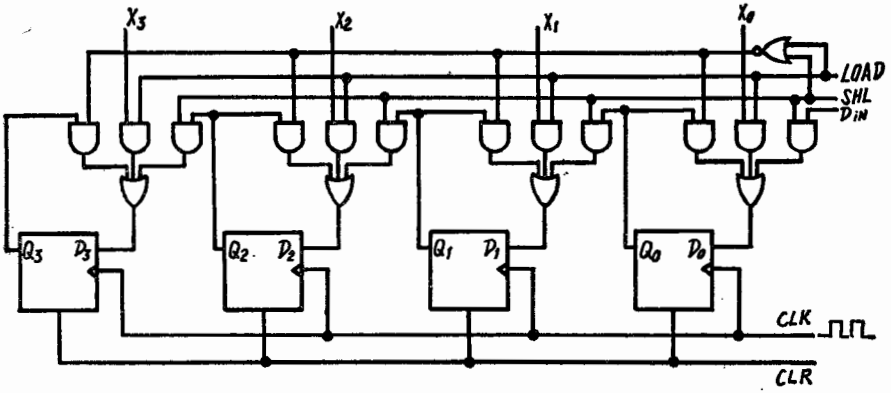


(a) 电路　　(b) 框图

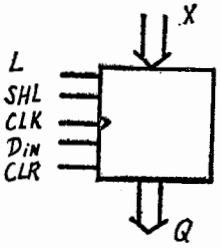


(c) 时間图

图 1.9 計数器



(a) 电路



(b) 框图

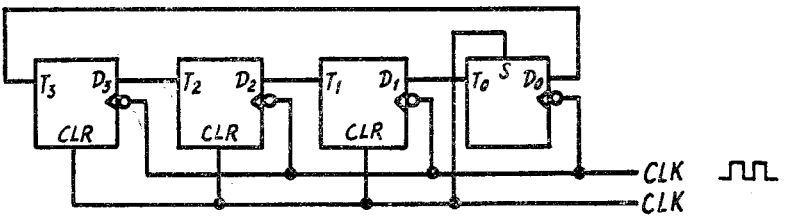
图 1.10 緩冲与移位寄存器

3．环形计数器

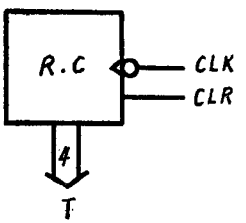
1）电路图：如图 1.11 (a) (b) 所示。

2）操作：当CLR = 1时，$T_3 = 0$，$T_2 = 0$，$T_1 = 0$，$T_0 = 1$，即T = 0001。当送

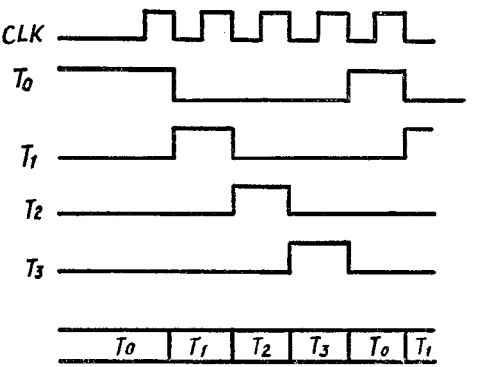
入时钟脉冲 CLK 时，每出现一次正跳变，T 依次为 0010→0100→1000→0001→……。如图 1.11（c）所示。



（a） 电路



（b） 框图



（c） 时間图

图 1.11 环形計数器

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，$D_{out} = D_{in}$，此时 D_{out} 线由该器件来驱动。当 E = 0 时，D_{out} 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。

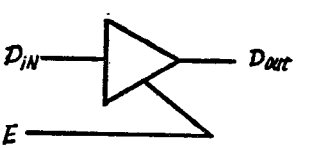
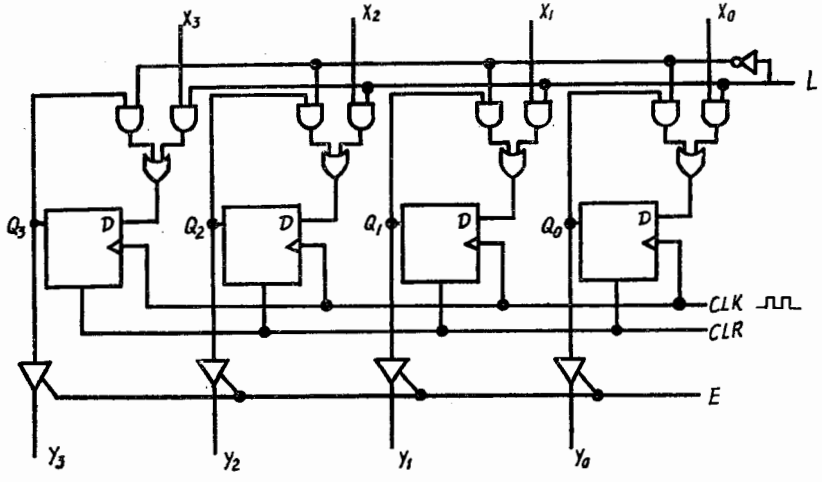


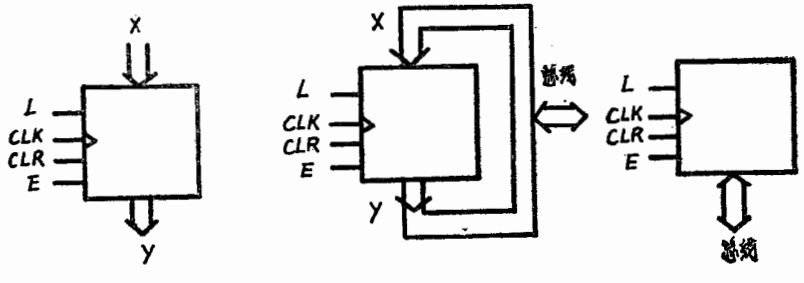
图 1.12 三态开关

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 具有三态輸出的緩冲寄存器

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。

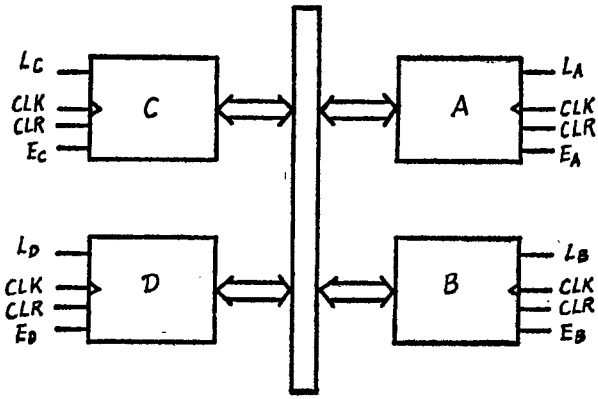


图 1.14　挂到总线上的寄存器組

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，又出现一些其他类型的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。

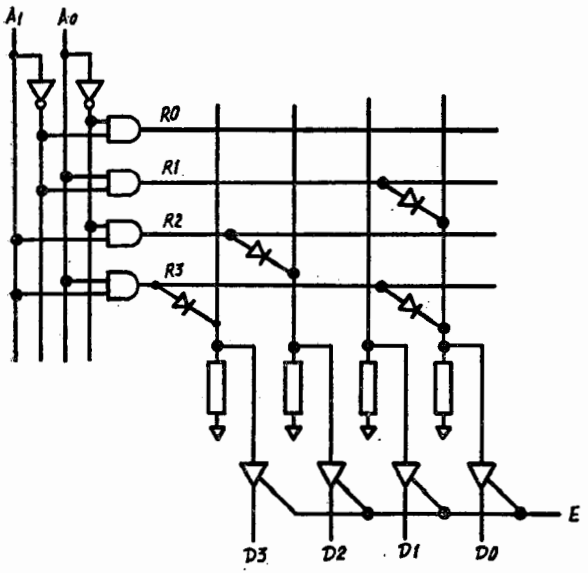
一、只读存储器 ROM

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。

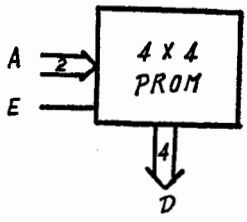
如果图中的地址线 A1 = 0、A0 = 1、E = 1，则第二条线 R1 为高电平，从而通过二极管而使 D0 为高电平，D1、D2、D3 相应的列中没有插入二极管，故为低电平，即

$$D = D3D2D1D0 = 0001$$

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 只讀存儲器的原理图

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 = $(AIAO)^2$。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 28 根引脚中，有 11 根地址线，8 根数据输出线，以及 V_{CC} 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} = +5V 时，为读数方式；当 V_{PP} = +25V 时，为编程（即写入—Programming）方式。

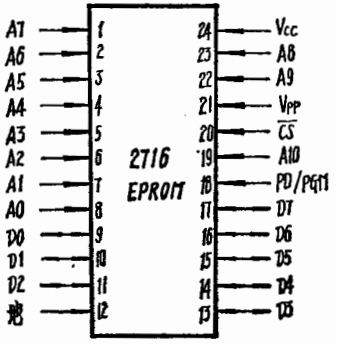


图 1.16　2716 型 EPROM(2K×8)的引脚图

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
讀	0	0	+5V	D_{OUT}
未选中	×	1	+5V	高阻抗
待机	1	×	+5V	高阻抗
編程	⎍	1	+25V	D_{IN}
校驗編程内容	0	0	+25V	D_{OUT}
禁止編程	0	1	+25V	高限抗

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。

4．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，脉冲宽度为 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，因而不必使用专门装置，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 V_{cc} 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	讀	讀出单元的内容
0	0	写	需要写入的内容

* 或称为 ME 信号

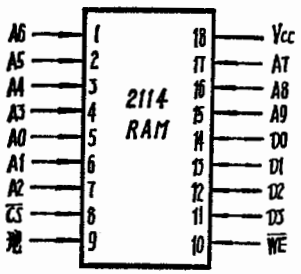


图 1.17 2114 型靜态 RAM (IK×4) 的引脚图

第二章　模型計算机

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扩充，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　模型計算机的結构

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。

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（R0—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。

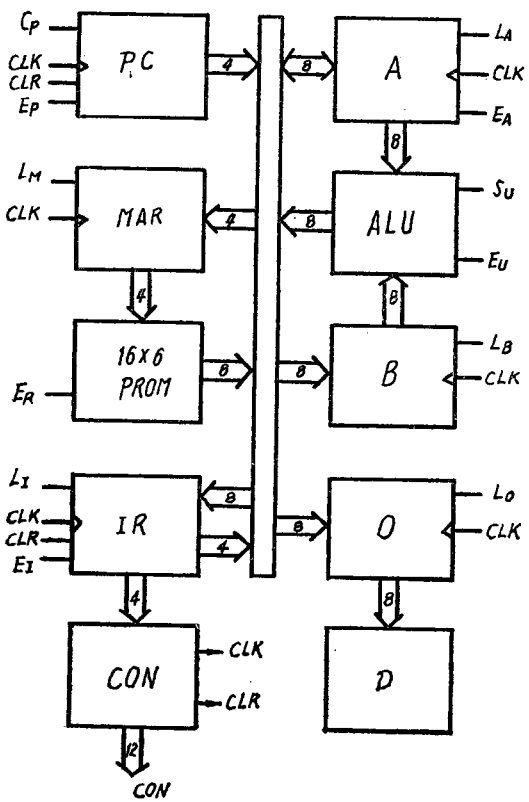


图 2.1　模型計算机的結构

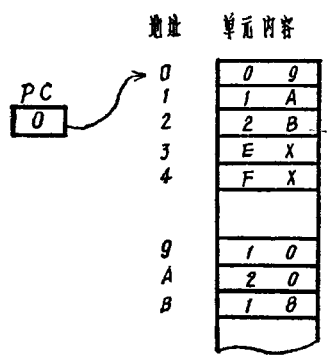


图 2.2　PROM 的内容表

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E_I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的指令的地址，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 指令系統和程序設計简介

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。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 LDA 9（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 09 H）。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“LDA 9”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

汇编語言		机器語言				操作及說明
		二進制		十六進制		
助記符	操作数	操作碼	地址碼	操作碼	地址碼	
LDA	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	×	O←A
HLT		1 1 1 1	× × × ×	F	×	停止

2. ADD　0 0 0 1 × × × ×

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 1 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。

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：SUB B　将累加器 A 中的内容减去 RB，结果存入 A。A = 48 D − 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

表 2.2　　程 序 设 计 举 例

地　址	机 器 代 碼	汇 編 語 言	操 作 結 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	1 0		
A	2 0		
B	1 8		

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）。

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和/或面向过程的高级语言（本书中不作介绍）。

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 ADD A 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 的設計

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：

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：

表 2.3　　指　令　的　执　行　过　程

节拍 / 指令	指令周期					
	取周期			执行周期		
	T0	T1	T2	T3	T4	T5
LDA 9 0000 1001	MAR←PC L_M, E_P	IR←R(MAR) L_I, E_R	PC←PC+1 C_P	MAR←IR L_M, E_I	A←R(MAR) L_A, E_B	——
ADD A 0001 1010	MAR←PC L_M, E_P	IR←R(MAR) L_I, E_R	PC←PC+1 C_P	MAR←IR L_M, E_I	B←R(MAR) L_B, E_R	A←A+B L_A, E_U
SUB B 0010 1011	MAR←PC L_M, E_P	IR←R(MAR) L_I, E_R	PC←PC+1 C_P	MAR←IR L_M, E_I	B←R(MAR) L_B, E_R	A←A−B L_A, E_U, S_U
OUT 1110 ××××	MAR←PC L_M, E_P	IR←R(MAR) L_I, E_R	PC←PC+1 C_P	O←A L_O, E_A	——	——
HLT 1111 ××××	MAR←PC L_M, E_P	IR←R(MAR) L_I, E_R	PC←PC+1 C_P	—— HLT	——	——

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。

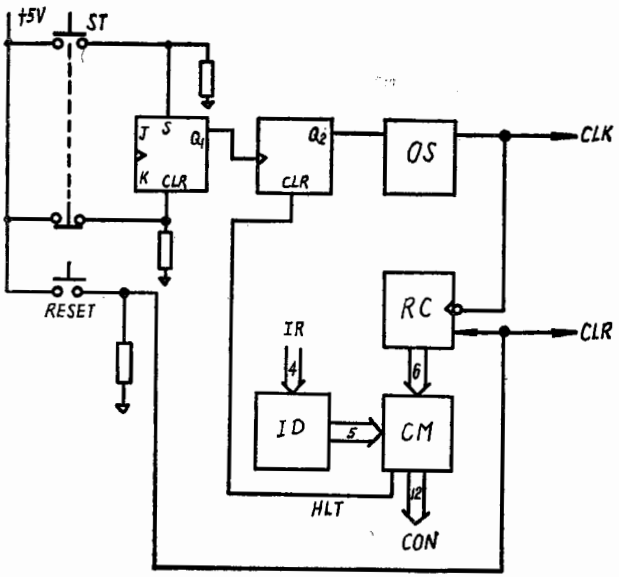


图 2.3　控制单元 CON 的电路

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 $L_M^4=1$，$L_M=1$。

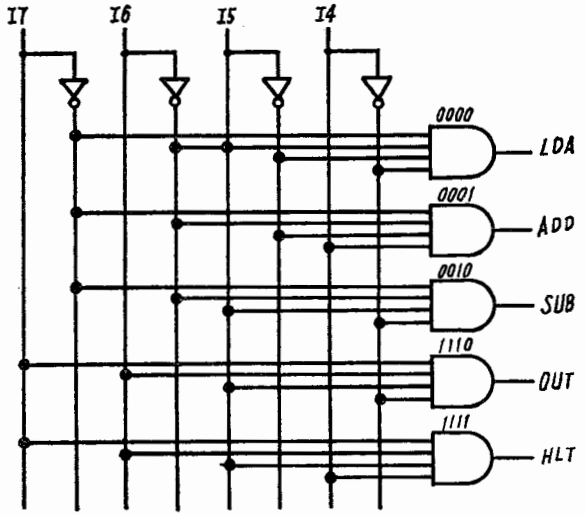


图 2.4　指令譯碼器 ID 的电路

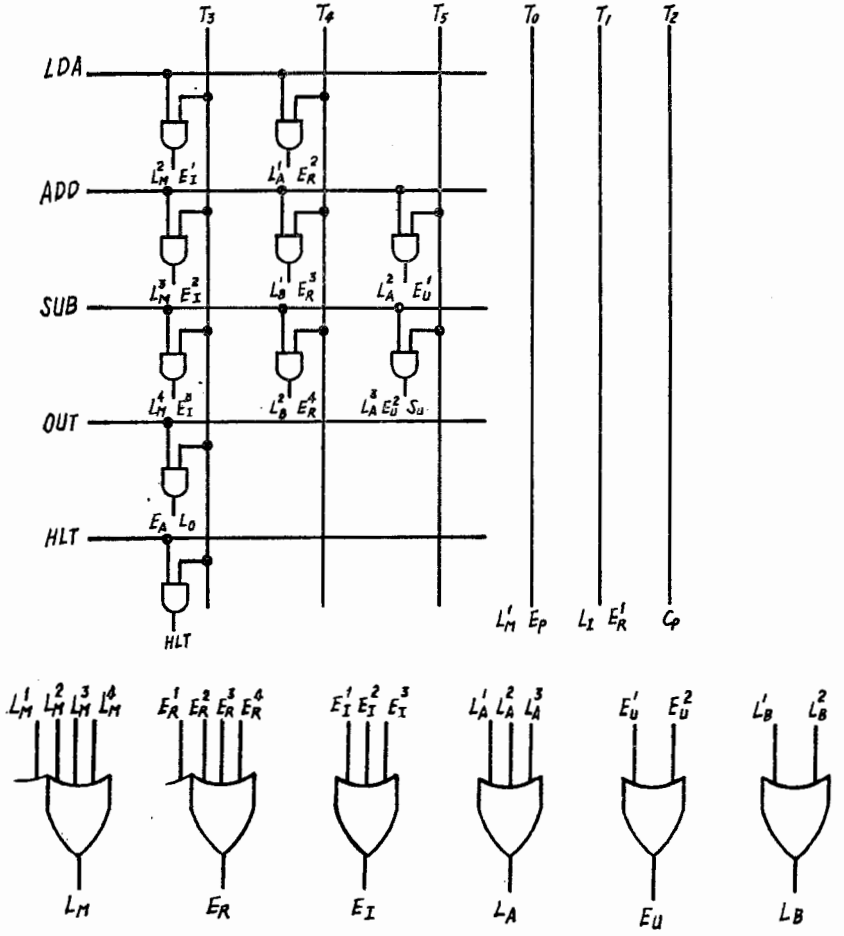


图 2.5　控制矩陣 CM 的电路

2.5 模型計算机的操作

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 正跳变瞬间，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。

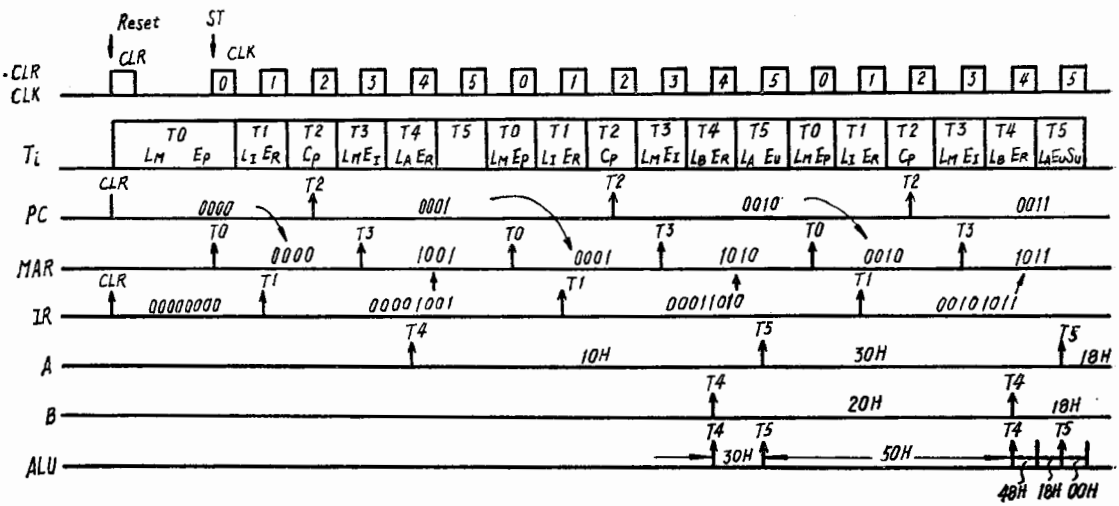


图 2.6 模型計算机操作的时間图

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：

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

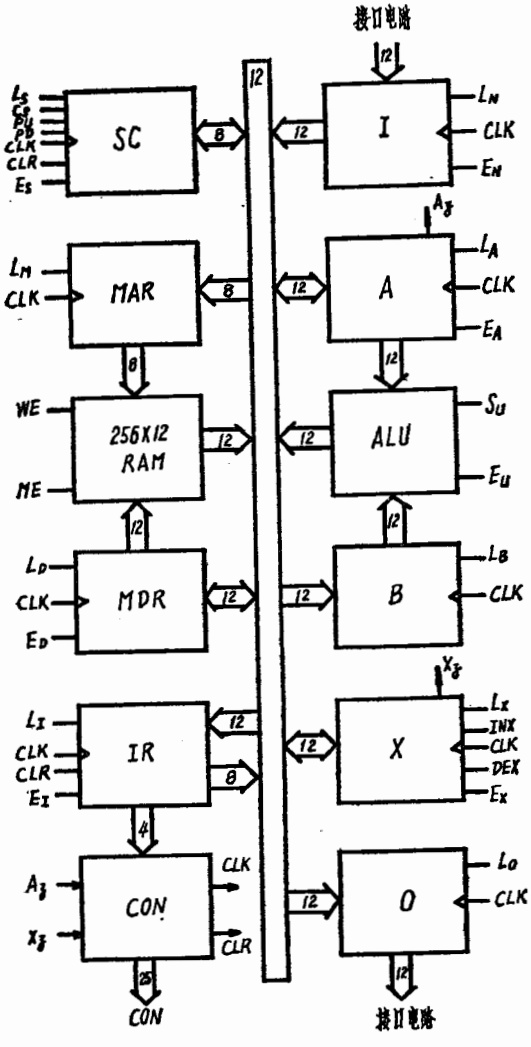


图 2.7 扩充模型机的結构

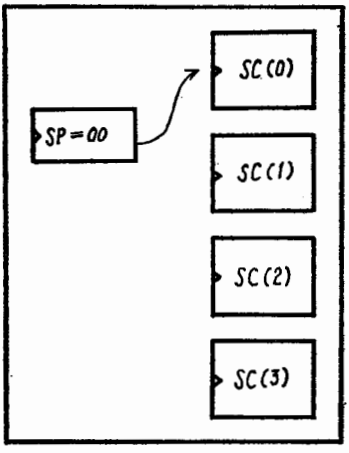


图 2.8 栈 SC 的結构示意图

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。

3．增加了变址寄存器 X。若信号

* 在教学中本节可省略不講

INX = 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		十六进制	操作及说明
		二进制			
助记符	操作数	操作码	地址码		
LDA	32	0000	0011 0010	032	A←R32
ADD	33	0001	0011 0011	133	A←A+R33
SUB	34	0010	0011 0100	234	A←A−R34
STA	3B	0011	0011 1011	33B	R3B←A
NOP		0100	×××× ××××	4××	不操作
LDX	43	0101	0100 0011	543	X←R43
DEX		0110	×××× ××××	6××	X←X−1
INX		0111	×××× ××××	7××	X←X+1
JMP	56	1000	0101 0110	856	PC←56
CLA		1001	×××× ××××	9××	A←000
JIZ	5E	1010	0101 1110	A5E	若X=0 PC←5E
JMS	63	1011	0110 0011	B63	SP←SP+1 SC(SP)←63
BRB		1100	×××× ××××	C××	SP←SP−1
INP		1101	×××× ××××	D××	A←I
OUT		1110	×××× ××××	E××	O←A
HLT		1111	×××× ××××	F××	停

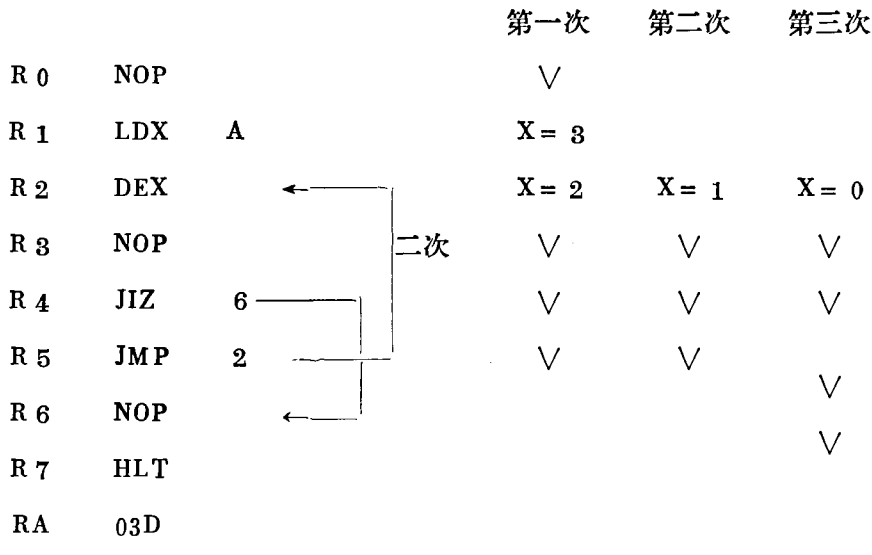
二、程序设计举例

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



例2．试设计 $3\times8=?$ 的程序

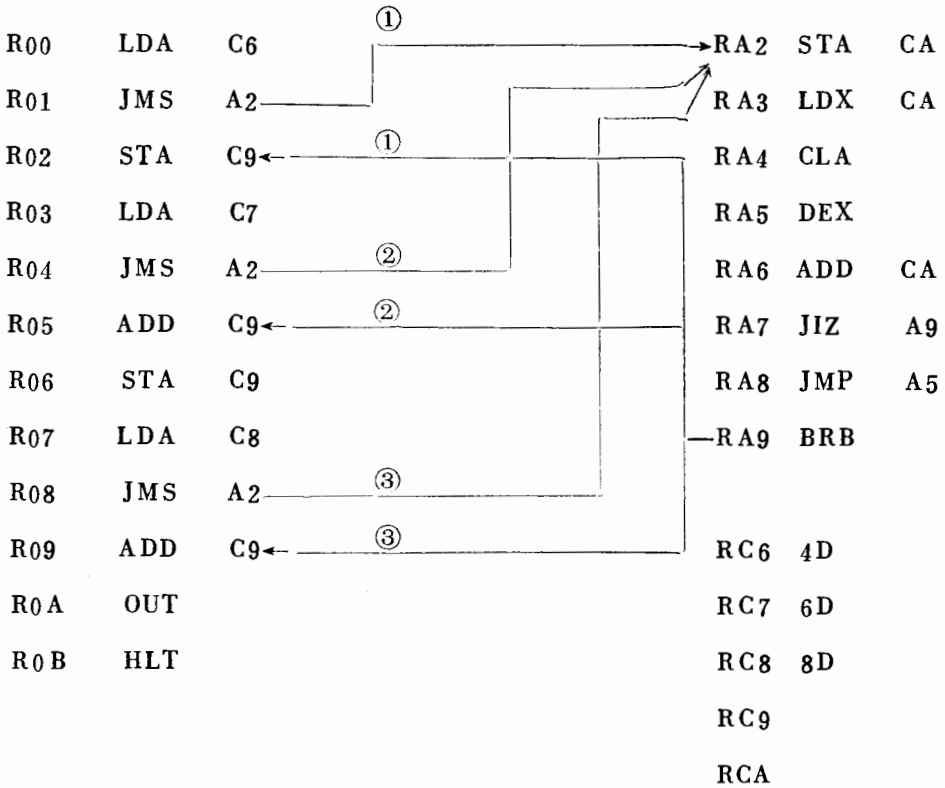
解：

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：



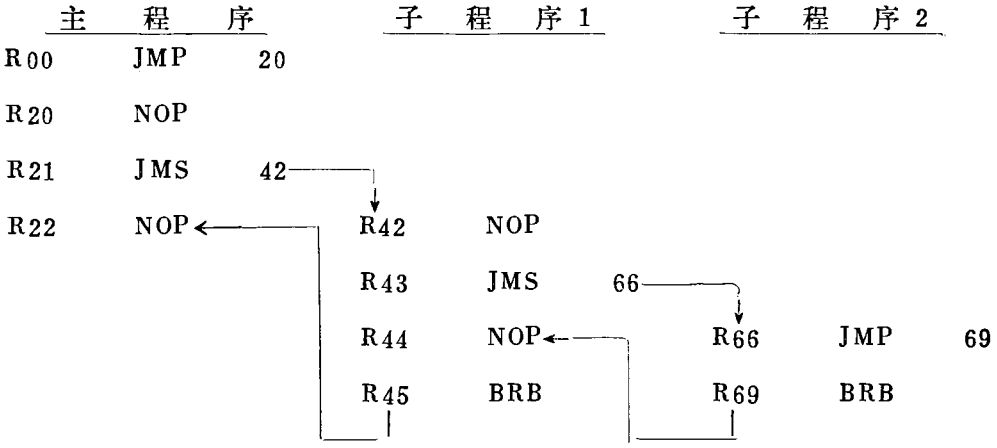
题中有三次要求一个数的平方，为此编写一个求平方的子程序（从RA2单元开始）。每当求平方时调用此子程序即可，而不必重复三次编写“求平方”的程序，从而可简化程序设计，并节省存储单元的空间。

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SC

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：

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

当主程序执行 R21 单元中“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 1 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，Intel 4040 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 模型計算机系統的简化

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。

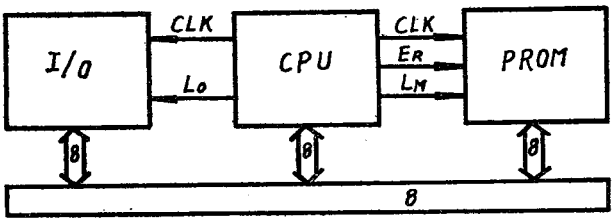
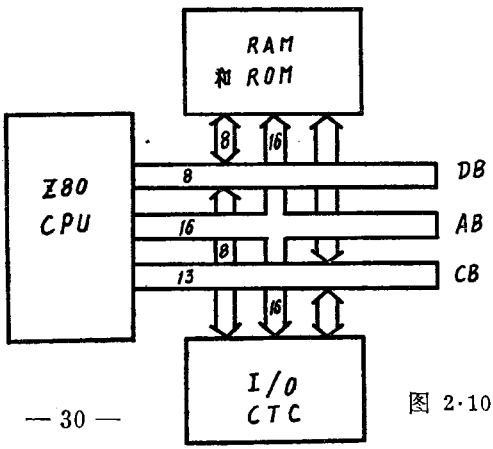


图 2.9　模型机的简化图



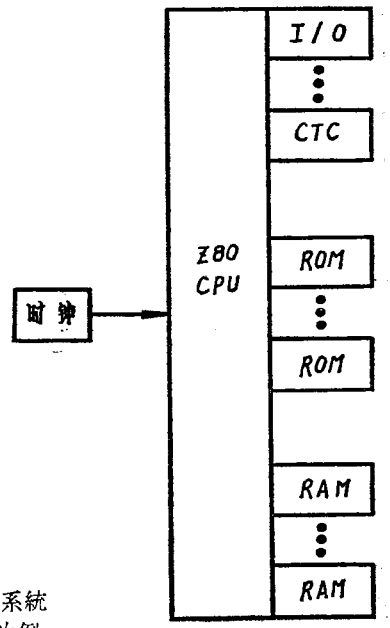


图 2·10　一般的微型机系統
——以 Z80 为例

第三章　Z80-CPU的结构

CPU是中央处理单元(Central Processing 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在微型計算机系統中的作用

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，有必要先来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 device)，通由$\overline{CPU}$表示。CPU的功能可以概括为对数据的**读、写**(CPU与$\overline{CPU}$之间)、**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**。所有上述四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)。

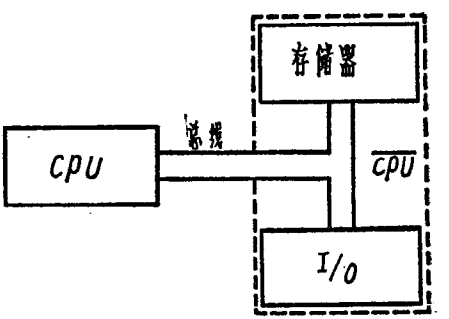


图3.1　微型計算机系統的簡化框图

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$\overline{CPU}$之间的数据传送主要是通过硬件设计(电路接口)来实现的。

3.2　Z80-CPU的結构

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。

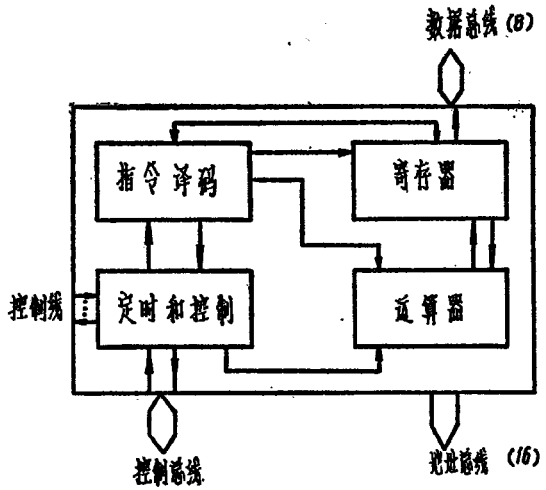


图 3.2　Z80—CPU的简化框图

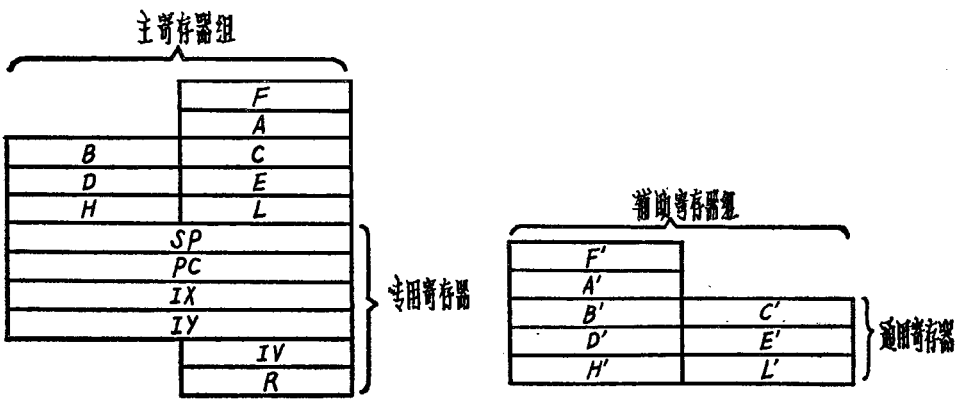


图 3.3　Z80—CPU的寄存器

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 继而使 PC 内容增 1，使 PC 指向相继的指令字节。

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）。

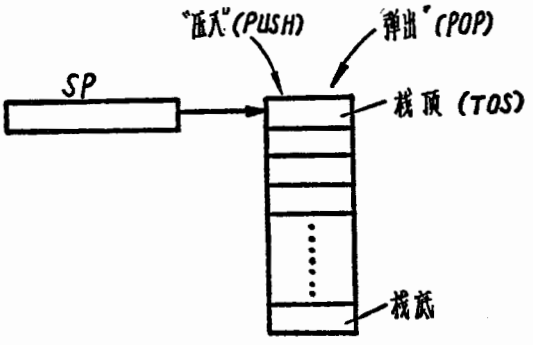


图 3.4 栈与栈操作

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）。

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 8 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

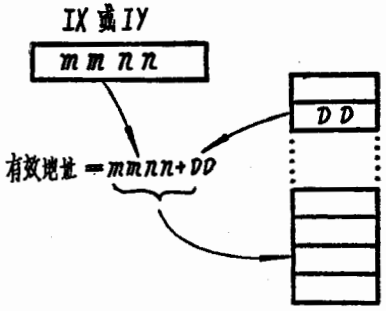


图 3.5 变址操作示意图

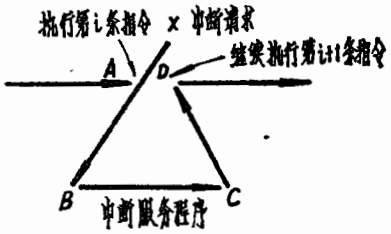


图 3.6 中断操作示意图

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。

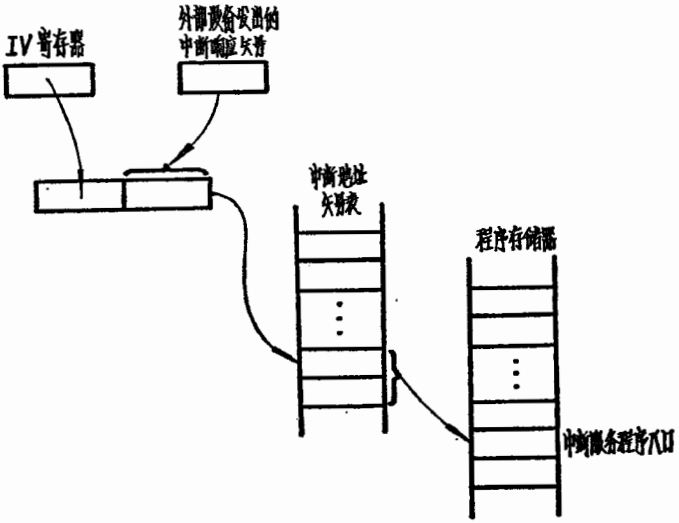


图 3.7 中断矢量的形成

5. 存储器刷新计数器 (memory Refresh counter, 简称R)。在Z 80微型计算机系统中，允许采用十分廉价的动态RAM。但是，除非大约每2ms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的动态刷新电路。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。测试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。

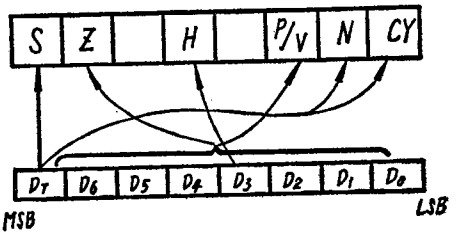


图 3.8 Z80—CPU 的标志寄存器

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）。这个标志标识运算结果的奇偶性，如果结果中有偶数个1，则P/V标志置1；否则置0。这个标志还标识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。

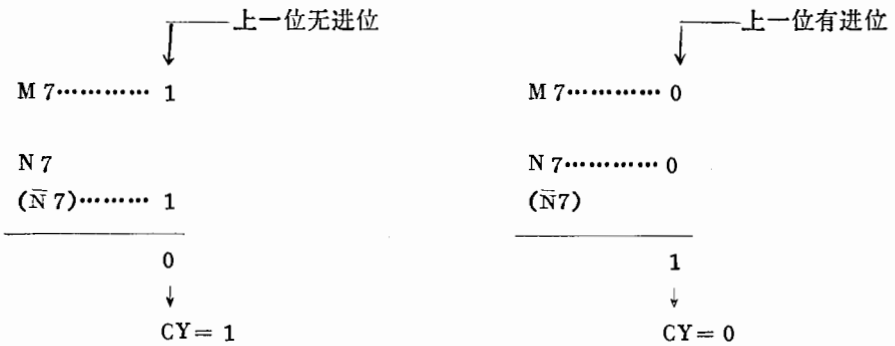
对于一个数据字节，如用最高位表示正负号，则有7个有效位，能表示-128—+127范围内的数。如果运算的结果超出了这个数值范围，就会发生溢出。以两数M，N相加为例，设其最高位分别为M_7，N_7；结果为R，其最高位为R_7，则有下列真值表：

M7	$\overline{N7}$	R7	P/V	
0	0	0	0	
0	0	1	1	若两个数同号，和的符号却不同，說明发生了溢出
0	1	0	0	
0	1	1	0	若M，N具有不同符号，不会发生溢出
1	0	0	0	
1	0	1	0	
1	1	0	1	
1	1	1	0	若两数同号，和的符号相同，說明沒有发生溢出

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	1	若 M，N 具有不同符号，差的符号与被减数相反，說明已发生溢出
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	0	
1	1	0	1	
1	1	1	0	

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：

$$P/V = d_{bcy} + CY$$

式中 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存器对，叫做程序状态字（Program Status Word）或处理器状态字（Processor 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。利用通用寄存器，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中有两组独立的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。它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/O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$\overline{\text{CPU}}$各部分的控制和定时信号（见图3.9）。

Z80—CPU的动作是周期性的，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续时间，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（WATI）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（Instruction cycle）。根据指令内容的不同，Z80

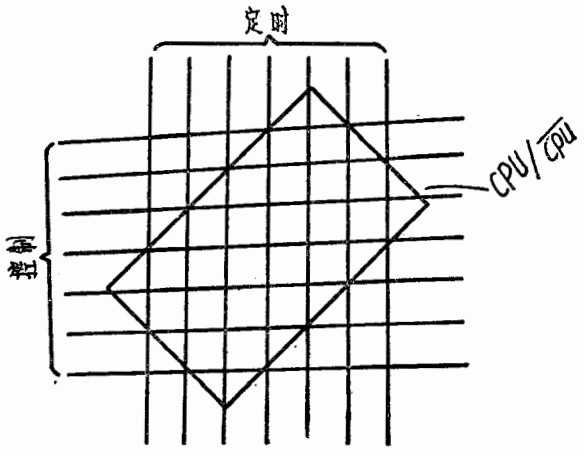


图 3.9 控制的器功能

--CPU 执行一条指令可能需要一至六个机器周期。

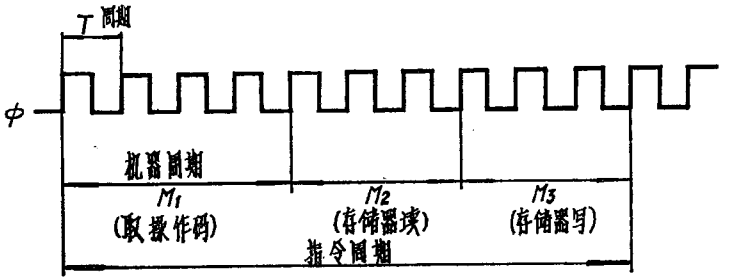


图 3.10 Z80 的定时波形

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。

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。

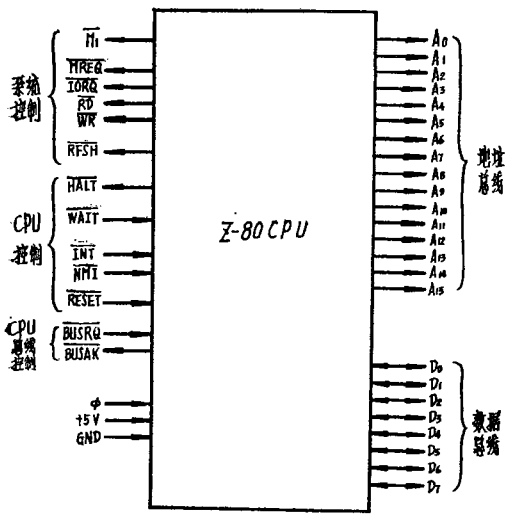


图 3.11 Z80—CPU 的引脚图

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：

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器周期。

$\overline{\text{MREQ}}$（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。

$\overline{\text{IORQ}}$（I/O ReQuest，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。$\overline{\text{IORQ}}$ 也被用于中断响应。$\overline{M_1}$和 $\overline{\text{IORQ}}$ 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。

$\overline{\text{RD}}$（memory ReaD，存储器读）。三态输出。标志 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。

$\overline{RFSH}$(ReFre SH，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。

$\overline{WAIT}$(WAIT 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 request，不可屏蔽中断请求)。输入。负跳沿触发中断请求信号。与$\overline{INT}$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)。标准复位控制输入。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$\overline{INT}$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总线的请求。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，+5V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$\overline{\text{CPU}}$各部分协调动作的命令，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。

为了操纵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。这个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，首先必须了解语句语法。

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，而在于谙练指令的分类，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、代碼格式和分类

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 VALUE

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

的末尾须接一个冒号“：”。

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。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Z 80 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

Z 80 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。

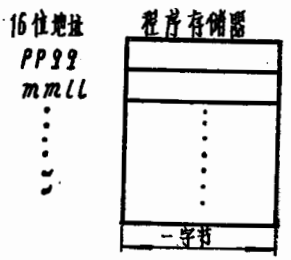


图 4.1 程序存储器的结构

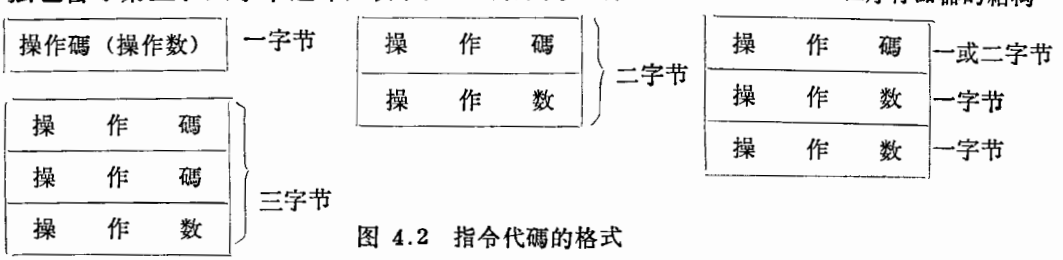


图 4.2 指令代碼的格式

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寄存器与存储器之间传送数据；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 source 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

LD H, 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 传送到寄存器 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 I 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  dst16, src16
```

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

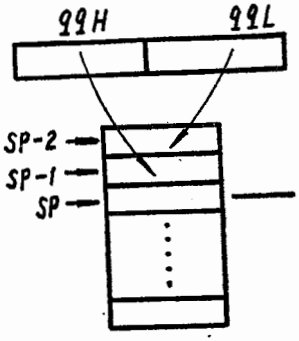


图 4.3 PUSH指令的执行

PUSHqq

(sp−2) ←qqL

(sp−1) ←qqH

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

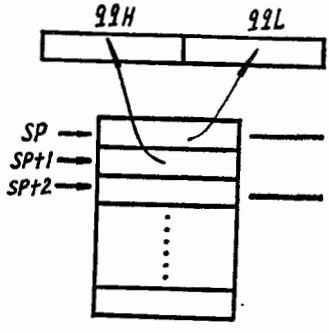


图 4.4 POP指令的执行

POPqq

qqH← (sp+1)

qqL← (sp)

sp←sp+2

三、交换指令组

这组指令的功能是交换源和目的地的内容，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 （接45页）

EX°(SP), IX $\begin{pmatrix} IXH \longleftrightarrow (SP+1) \\ IXL \longleftrightarrow (SP) \end{pmatrix}$ 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 $\begin{pmatrix} IYH \longleftrightarrow (SP+1) \\ IYL \longleftrightarrow (SP) \end{pmatrix}$

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。在这组指令的操作中，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。

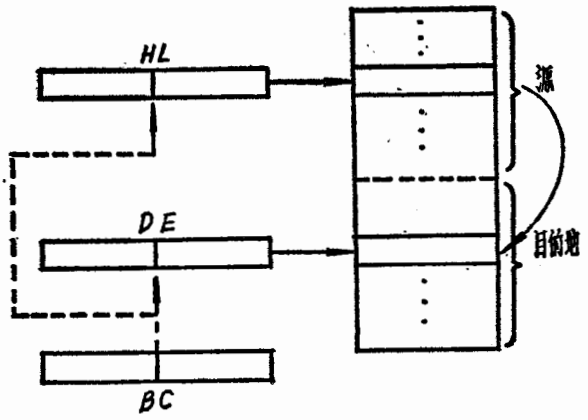


图 4.5 数据块传送指令

数据块传送指令有两种传送方式：

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及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,

以及

(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及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。

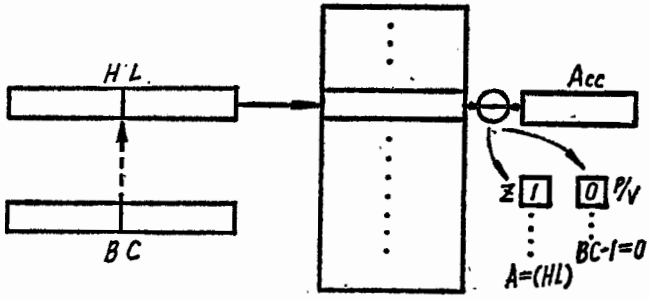


图 4.6　数据块查找指令

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 (Compare 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 byte 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）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）IN A，(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（0IH）=7BH

执行指令　　IN A，（0IH）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

$$(C)=1FH,\qquad (H)=AAH$$

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（INput to memory and Increment pointer until byte counter is zeRo）。记作

$$(HL)\leftarrow(C),\qquad HL\leftarrow HL+1,\qquad B\leftarrow B-1$$

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_8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

$$(HL)\leftarrow(C),\qquad HL\leftarrow HL-1,\qquad B\leftarrow B-1$$

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

$$(HL)\leftarrow(C),\qquad HL\leftarrow HL+1,\qquad B\leftarrow B-1$$

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”(eXclusive 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 相令除外）是：

- 都是针对累加器与某个指定寄存器、存储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；
- 作为特定操作的结果，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：

$$\left.\begin{matrix}\text{ADD}\\ \text{ADC}\end{matrix}\right\}\ \text{A, r}\qquad r=A,\ B,\ C,\ D,\ E,\ H,\ L,$$

所执行的操作是 $A\leftarrow A+r\ (+CY)$

$$\left.\begin{matrix}\text{ADD}\\ \text{ADC}\end{matrix}\right\}\ \text{A, n}\qquad n=0\sim255\ (\text{立即数})$$

所执行的操作是 $A\leftarrow A+n(+CY)$

$$\left.\begin{matrix}\text{ADD}\\ \text{ADC}\end{matrix}\right\}\ \text{A, (HL)}$$

所执行的操作是 $A\leftarrow A+(HL)+(CY)$

$$\left.\begin{matrix}\text{ADD}\\ \text{ADC}\end{matrix}\right\}\ \text{A,}\quad \begin{matrix}(IX+d)\\ (IY+d)\end{matrix}$$

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

这里 $\begin{matrix}(IX+d)\\ (IY+d)\end{matrix}$ 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”(logical 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 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
OR r / n / (HL) / (IX+d) / (IY+d)	A ← A∧ r / n / (HL) / (IX+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)。这条指令因操作数不同具有下列形式：

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

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

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

$$(IX+d)=A0H,\qquad A=E3H$$

执行 CP(IX+d)后，各标志位根据运算结果受到不同影响，如图 4.7 所示。

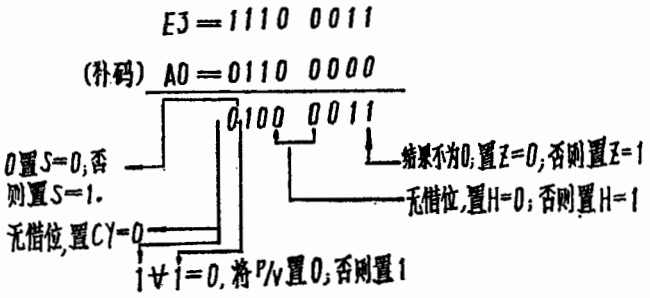


图4.7 CP 指令举例

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+d)+1
INC (IY+d)	(IY+d)←(IY+d)+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 指令的相同，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) ADD HL, 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 H and L)。记作

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

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

$$ss \leftarrow ss + 1$$

5) DEC ss。减量指定寄存器对的内容(DECrement contents of specified register pair)。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，NEG 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。累加器内容变负(NEGate contents of accumulator)。它等效于取累加器内容的补码，即

$$A \leftarrow 0 - A$$

或写成

$$A \leftarrow \overline{A} + 1$$

4) CCF。进位标志变反(Complement 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：循环移位指令使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，可以使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；实现对数据的拼装和分离；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位(Rotate register,accumulator or memory location Left Circular)。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；原来的第 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。

2) RRC。寄存器、累加器或存储单元向右循环移位(Rotate 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。

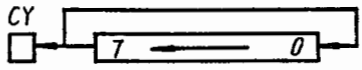


图 4.8 RLC 指令的操作

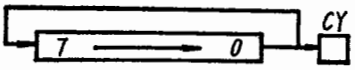


图 4.9 RRC 指令的操作

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+d), (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。

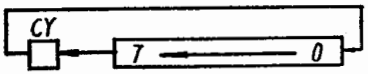


图 4.10 RL 指令的操作

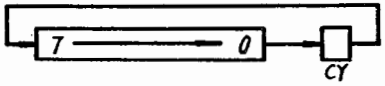


图 4.11 RR 指令的操作

5) RLD。一个二——十进制数字在累加器和存储单元之间向右循环移位(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 4 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。

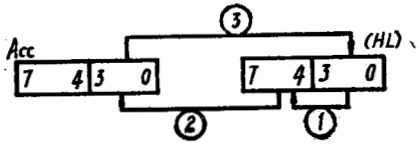


图 4.12 RLD 指令的操作

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。

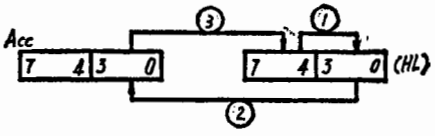


图 4.13 RRD指令的操作

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:

$$\text{SRL}\begin{cases} \text{r} \\ \text{(HL)} \\ \text{(IX+d)} \\ \text{(IY+d)} \end{cases}$$

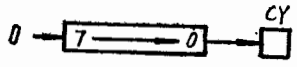


图 4.14 SRL指令的操作

所执行的操作如图4.14所示。数据向左移位,留下的空位都被清零。

8) SLA。寄存器或存储单元向左算术移位(Shif 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 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。

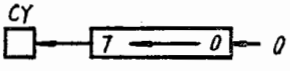


图 4.16 SRA指令的操作

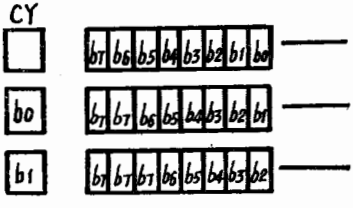


图 4.17 SRA指令的操作

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) JP $\begin{matrix}nn\\label\end{matrix}$。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) JP cc, $\begin{matrix}nn\\label\end{matrix}$。如果条件满足，则转移到操作数所指定的地址(JumP to address identified 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 specified by contents of 16 bit register)。记作

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

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。

三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 con-

tents of program counter)。操作数中的e是偏移。这里要注意，转移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。

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)。

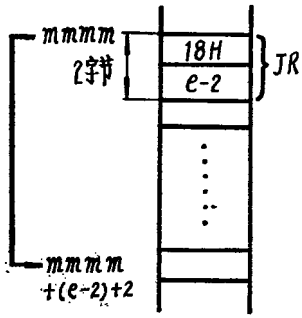


图 4.18 JR指令的操作

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 present contents of program counter if register B Not Zero)。

DJNZ的典型使用方式如图4.19所示。

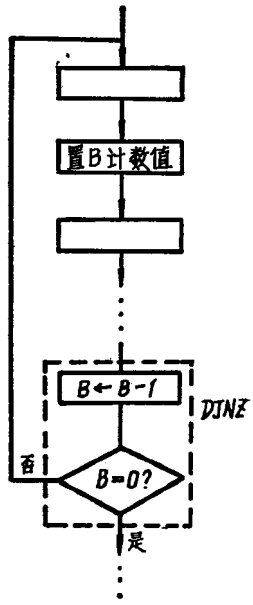


图 4.19 DJNZ指令的操作

二、子程序调用和返回指令组

调用是一种转移，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 $\begin{matrix}nn\\label\end{matrix}$。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

$$PC_L\leftarrow(SP)$$

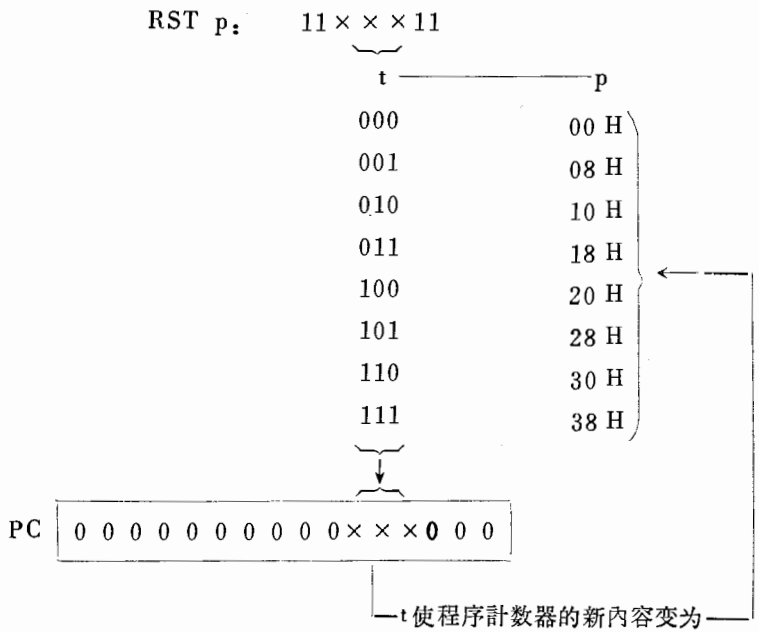
$$PC_H \leftarrow (SP+1)$$

$$SP \leftarrow (SP+2)$$

此指令将原先进栈的断点地址弹回程序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，还有一条特殊的一字节调用指令，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 (maskable 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允许触发器IFF(Interrupt Flip-Flop)的操作实现的。在Z80—CPU中，有两个这样的触发器IFF1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 Z 80 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：$\overline{RESET}$ 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 $\overline{IORQ}$ 信号，然后进入等待状态。

请求中断的外部设备应能识别 $\overline{M_1}$ 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 保护进栈，然后转到中断服务程序的入口。

一旦中断开始，所有后继的中断都被禁止，IFF1 和 IFF2 自动复位。这时，程序中应安排一条 EI 指令，以便允许优先级较高的其他设备发出中断请求。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(MODE 1)。执行“IM 1。中断方式 1 (Interrupt Mode 1)”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(MODE 2)。执行“IM 2。中断方式 2(Interrupt Mode 2)”指令。它也叫做矢量中断方式。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。

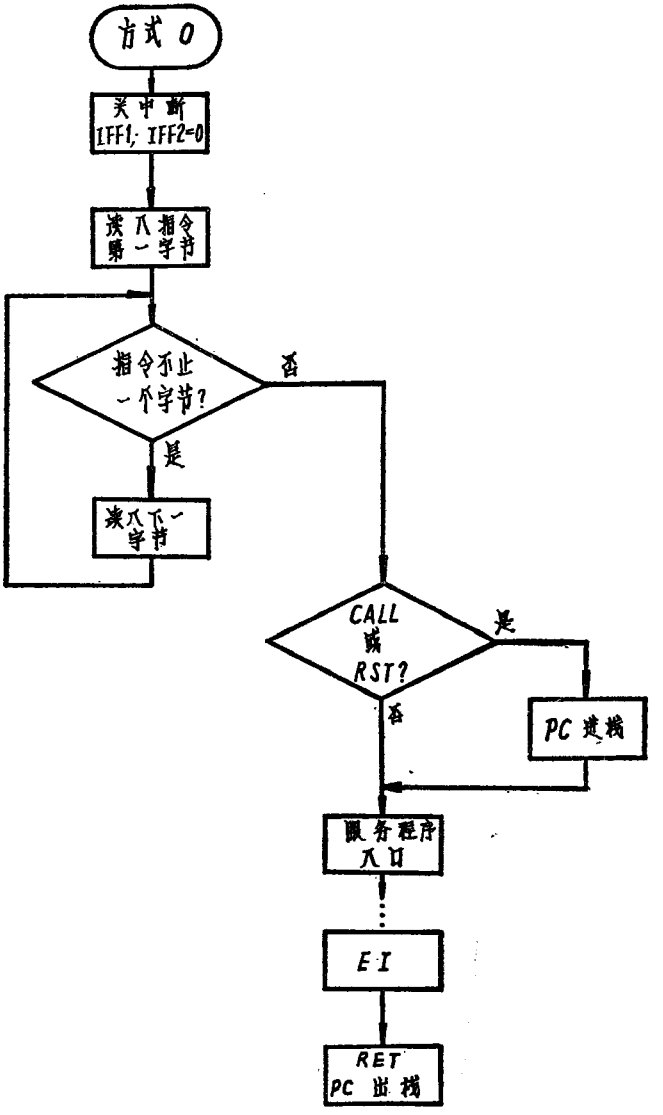


图 4.20 中断方式 0 的操作

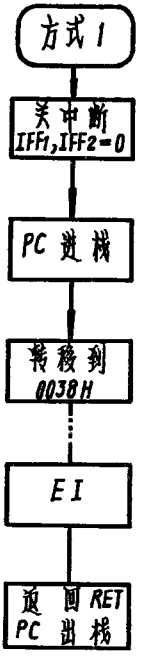


图 4.21 中断方式 1 的操作

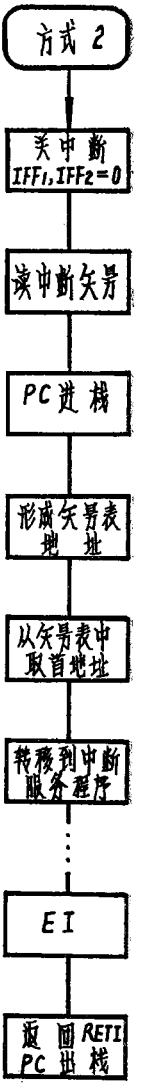


图 4.22 中断方式 2 的操作

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)

测试寄存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

$r_b \leftarrow 1$
$(HL)_b \leftarrow 1$
$(IX + d)_b \leftarrow 1$
$(IY + d)_b \leftarrow 1$

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。

本书在第五、六两章中只介绍 Z80 系列中最常用的两种芯片：Z80-PIO 和 Z80-CTC。

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、纸带读入机和凿孔机、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 IM2(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。

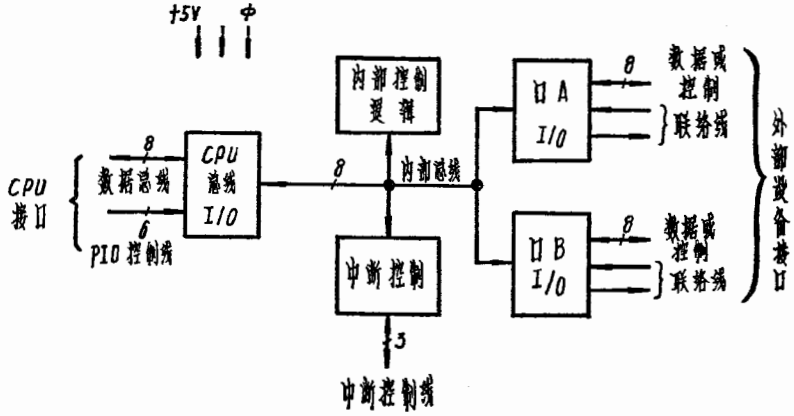


图 5.1 PIO的框图

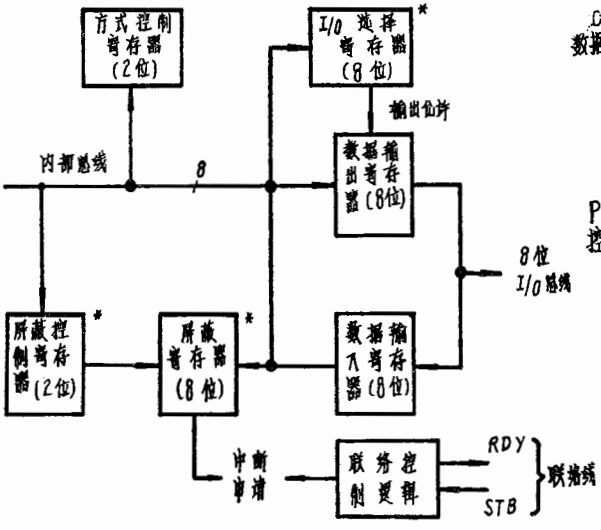


图 5.2 口的 I/O 框图

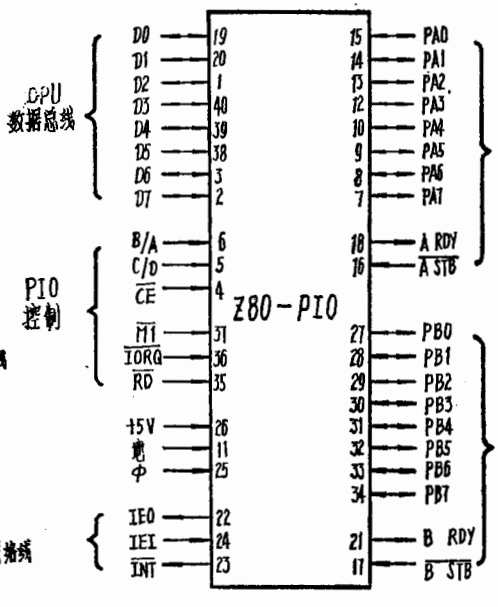


图 5.3 PIO 的引脚布置图

3. B/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/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

引脚			选中单元	单元地址
$\overline{CE}$ (*)	B/A(A0)	C/D(A1)		
0 (A7—A2=100000)	0	0	口 A 的数据寄存器	80 H
0 (A7—A2=100000)	0	1	口 A 的控制寄存器	82 H
0 (A7—A2=100000)	1	0	口 B 的数据寄存器	81 H
0 (A7—A2=100000)	1	1	口 B 的控制寄存器	83 H
1 (A7—A2≠100000)	×	×	芯片未被选中	

(*) TP—801 是按此地址連接的。不同的微型机系統或不同的 PIO 芯片，A7—A2可为不同值。

表 5.1 中×表示可为任意值，80H 表示为十六进制数 80 。如果地址总线 A7—A2 = 100000 时，$\overline{CE}$ 被译码成逻辑 0 。如果 A7—A2 为其他值时，$\overline{CE}$ = 1 。在这种情况下，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	沒有作用
0	0	1	中断响应
0	1	0	检查中断服务程序是否結束
0	1	1	复　位
1	0	0	CPU 从 PIO 讀出
1	0	1	从CPU写入PIO
1	1	0	沒有作用
1	1	1	沒有作用

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. $\overline{INT}$ 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，$\overline{INT}$ 有效。

下面顺便介绍链形中断的处理过程。图 5.4 表示了典型的嵌套中断的序列。在此序

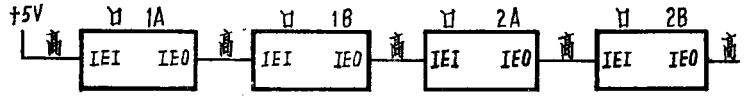


图 5.4 (1)

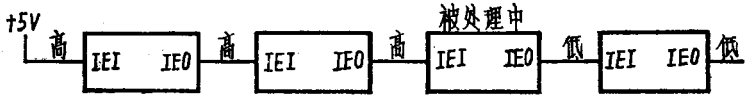


图 5.4 (2)

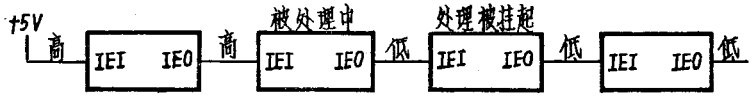


图 5.4 (3)

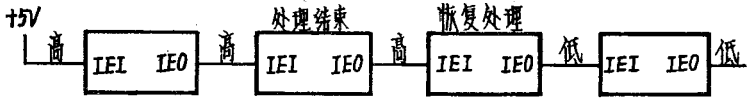


图 5.4 (4)

图 5.4 (5)

图 5.4 鏈形中断处理

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 1B 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 1B 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。

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. $\overline{A\ STB}$ 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. A RDY 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。

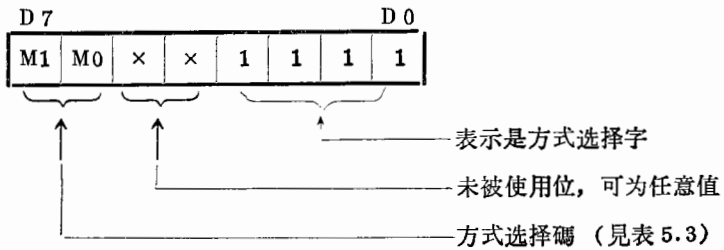
17. 其他：ϕ 为时钟；+5V 为电源；GND 为地。

5.3 PIO 的操作說明

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 TP801 单板计算机中所使用的具体值。

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(也是更重要的)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



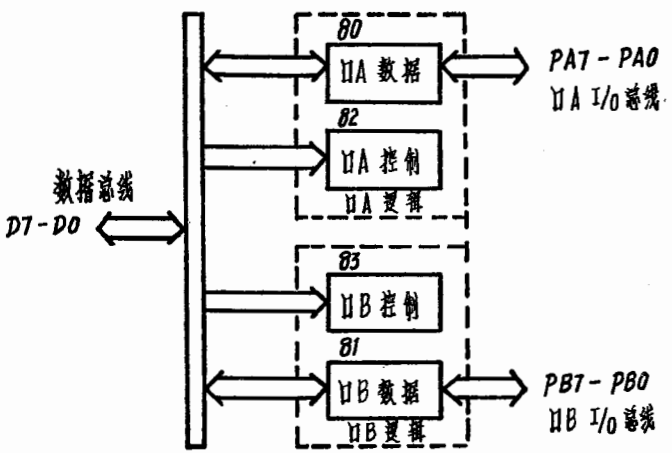


图 5.5 PIO 有四个可寻址单元

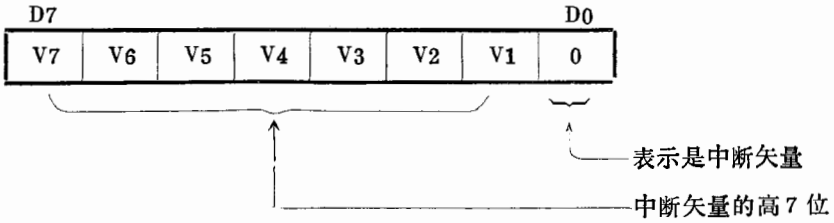
PIO 有四种操作方式，如表 5.3 所示。具体操作方式由方式选择字中的 M1M0 来决定。

表 5.3　　PIO　的　操　作　方　式

M1	M0	操作方式	說明
0	0	方式 0	带联絡的輸出
0	1	方式 1	带联絡的輸入
1	0	方式 2	带联絡的双向 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的联絡綫被口A所占用，因而口B只能工作在方式3。詳細解释見后。

2．中断矢量

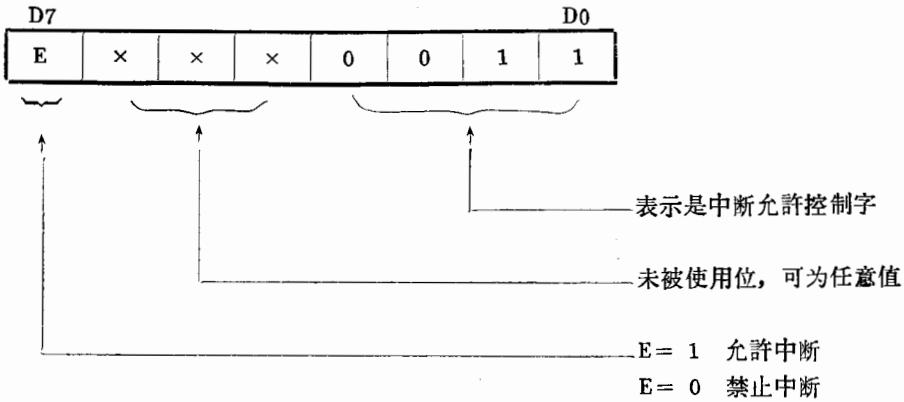


假设 CPU 中断矢量寄存器 I = 23H，又设口 A的 V7－V1 = 0100000，则 PIO 的中断矢量 = 40H，两者合成一个完整的中断矢量2340H。如表5.4所示，由2340和2341单元中找到中断服务程序的起始地址2200H。因为中断服务程序的起始地址占用两个单元，因而中断矢量可以安排成都是偶数，如 2340H，2342H，2344H 等，即它的最低位必然为 0 。由此可以与其他控制字相区别。

表 5.4　中断服务程序起始地址表(举例)

2340	0	0	口 A中断服务程序的起始地址
	2	2	
2342	1	B	口 B中断服务程序的起始地址
	2	2	
2344			

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,40H
OUT   (82),A
LD    A,OFH       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. 设定中断矢量为 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 产生一$\overline{WR}^*$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

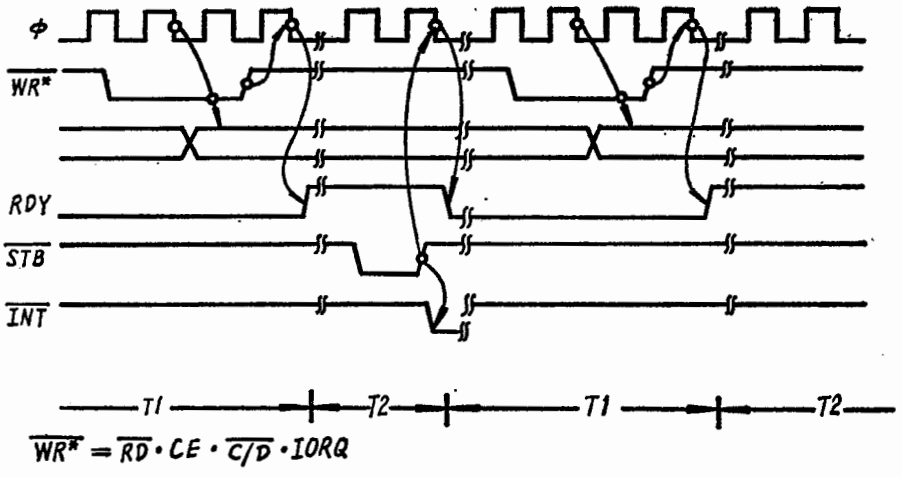


图 5.6 方式 0 的时间图

变高，告知外部设备口 A 中已有数据可供使用。这一时间段如图中 T1 所示。此后 A - RDY 将保持有效，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$\overline{\text{INT}}$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 CPU I 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 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$\overline{\text{INT}}$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，则下一个 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 。

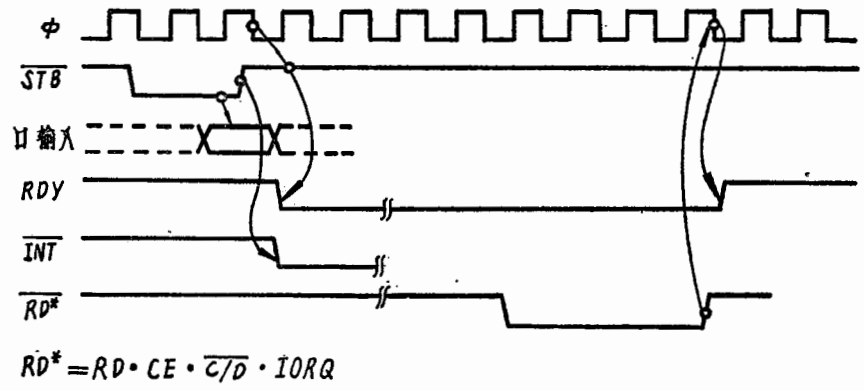


图 5.7　方式 1 的时間图

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。

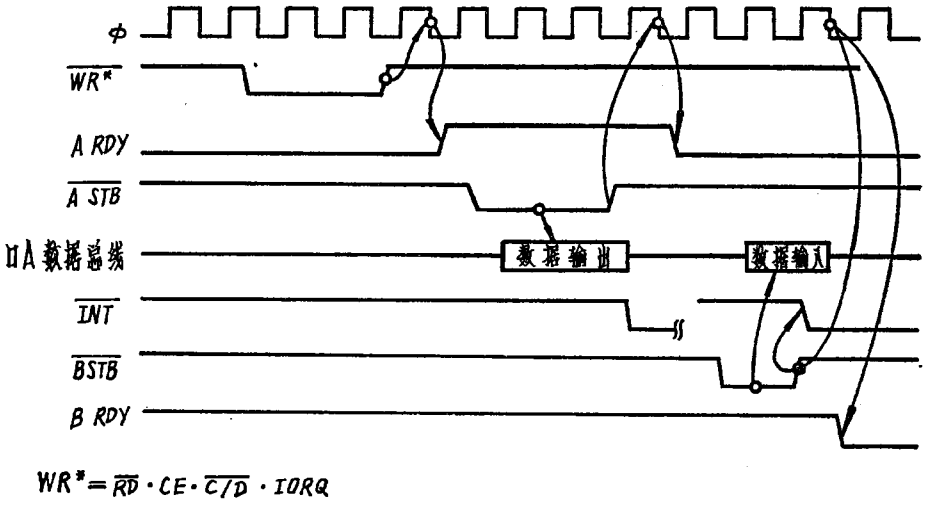


图 5.8　方式 2 的时間图

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。

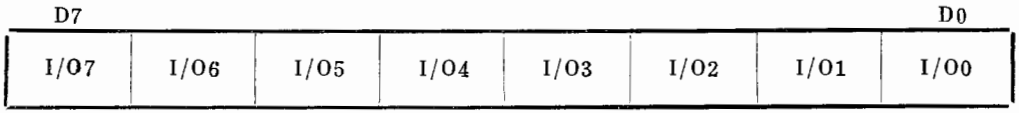
现结合下面的初始化程序来说明 PIO 如何被设定成这种工作方式并进行工作。

```
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：

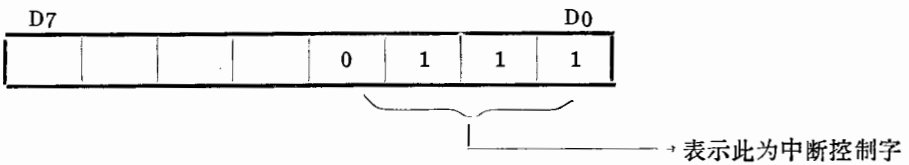
1．I/O选择字。此字紧跟在设定为工作方式3的控制字之后写入，它被锁存在前述的I/O选择寄存器中。



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。



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

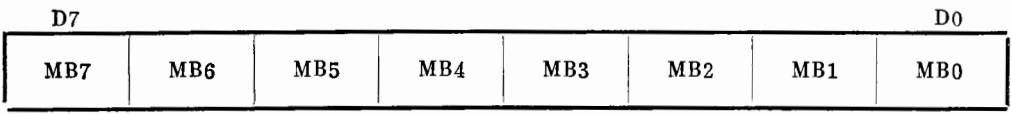
D6 = 1　对被监视的信号进行“与”操作

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。

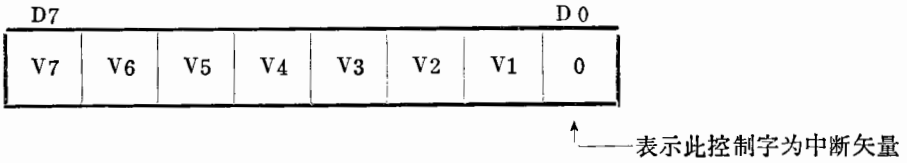


MB = 0　该位受监视

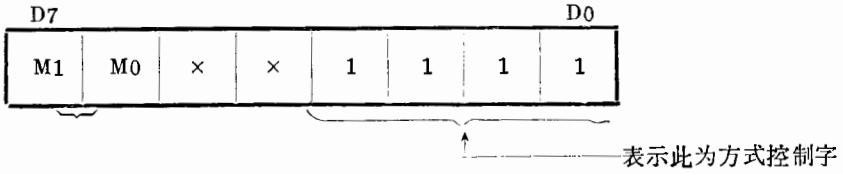
MB = 1　该位的监视被屏蔽

五、程序设定 PIO 工作方式的步骤

1．中断矢量

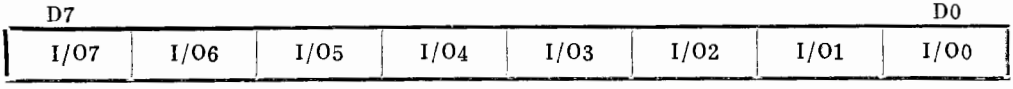


2．设置方式



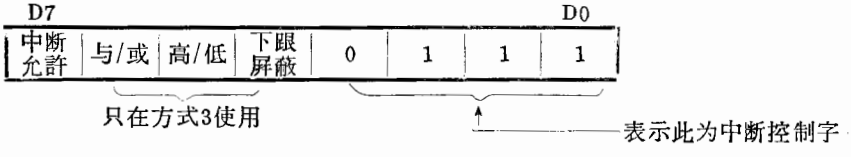
00——输出
01——输入
10——双向
11——位控

在选用方式 3（D7D6 = 11）时下一控制字必须是 I/O 选择字：

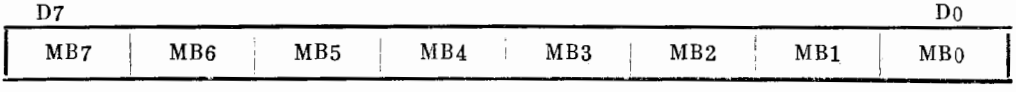


I/O = 1　　使该位为输入
I/O = 0　　使该位为输出

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他部分。

D7							D0
中断允許	×	×	×	0	0	1	1

↑——表示此为开、关中断字

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-CPU 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。

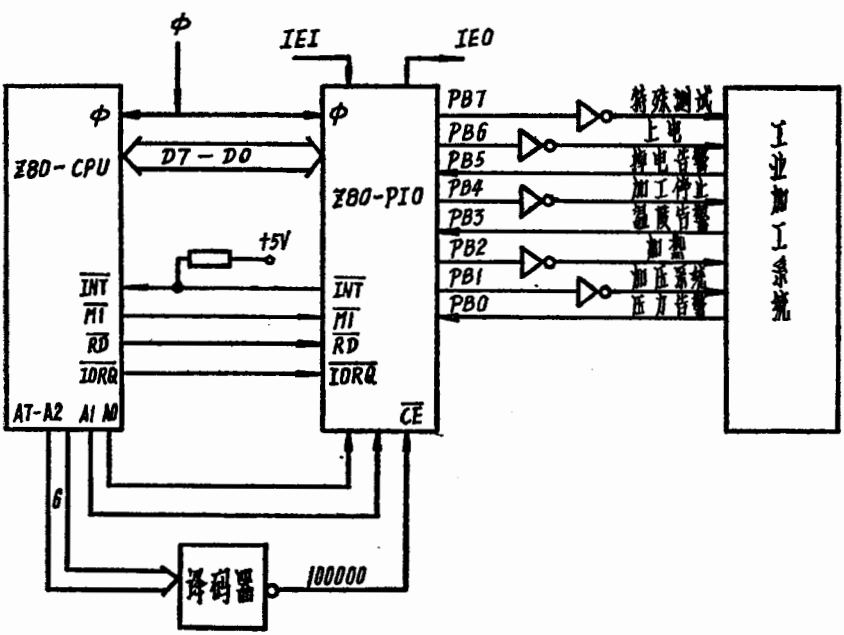


图 5.9　位控方式的电路連接

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　計数器/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化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都可包含有这样的芯片，它们可由程序来设定脉冲的个数、频率、波形等。

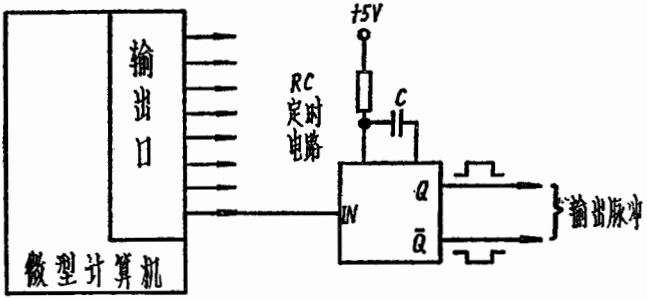


图 6.1　由单冲电路来产生脉冲

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、脉冲序列的起始和终了、产生脉冲的方式以及脉冲的个数均可由程序来设定。

2．计数功能。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

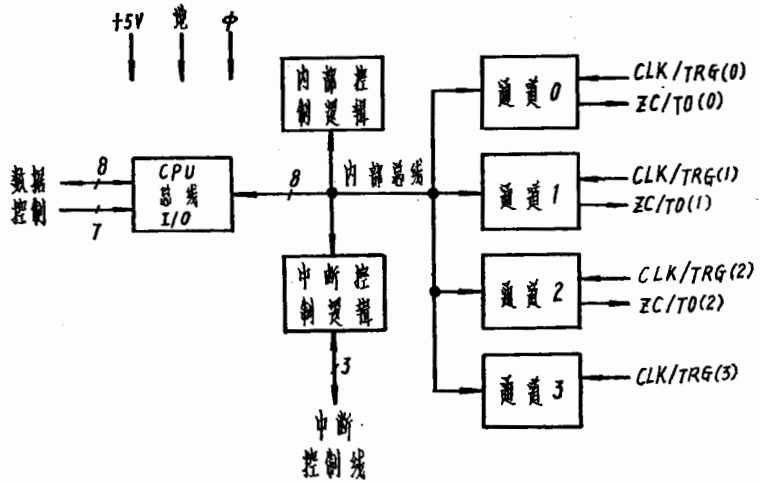


图 6.2 CTC 的框图

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（仅用于定时器工作方式）。它们的功能将在下节操作说明中介绍。

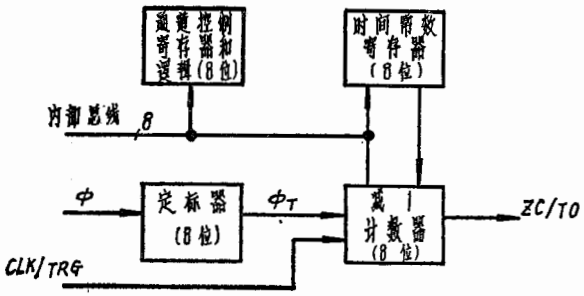


图 6.3 CTC 通道的框图

CTC 的引脚布置如图 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TP 801单板机系统中，当A7—A2 = 100001 时，才选中 CTC。

3．CS_1，CS_0 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 独立通道中选择一个通道。其选择逻辑如表 6.1 所示。

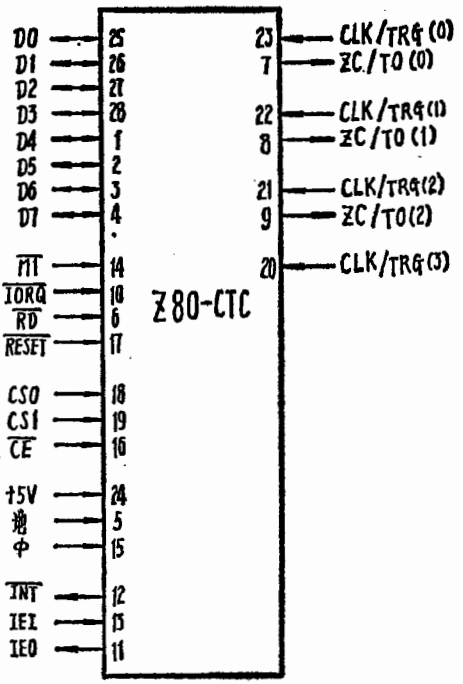


图 6.4　CTC 的引脚布置图

表 6.1　　PIO 的选择逻辑

引脚			选中单元	单元地址
$\overline{CE}$	CS 1 (A *)	CS 0 (A 1)		
0　(A7—A2＝100001)	1	0	通道 0	84 H
0　(A7—A2＝100001)	0	1	通道 1	85 H
0　(A7—A2＝100001)	1	0	通道 2	86 H
0　(A7—A2＝100001)	1	1	通道 3	87 H
1　(A7—A2≠100001)	×	×	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　　PIO 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結束
1	0	0	CPU从CTC讀出
1	0	1	从CPU写入 CTC
其　他　值			沒有作用

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。IE0 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）。它们的作用将在下节介绍。

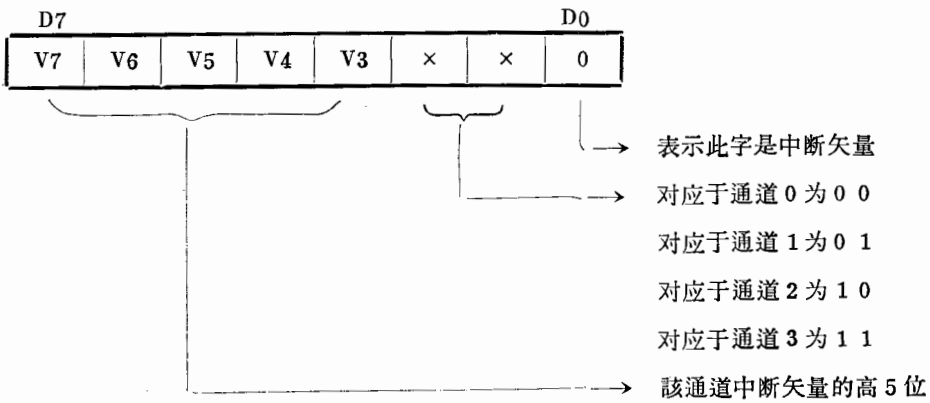
12. ZC/TO(2－0)（或称ZC2－0）为三个通道(通道 3 除外）的“回零/时间到”脉冲（输出）。它们的作用将在下节介绍。

13. 其他：ϕ为时钟；＋5v为电源；GND 为地。

6.3 CTC 的操作說明

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的中断矢量的格式如下：



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	D6	D5	D4	D3	D2	D1	1
中断允許	方式	量程	斜率	触发	輸入时間常数	复位	→表示此为控制字

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D2 = 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   (84),A
LD    A,85H     设定工作方式
OUT   (84),A
LD    A,03H     装入时间常数
```

```
OUT   (84),A
IM    2
EI
```

定时器工作的时间图如图 6.5 所示。

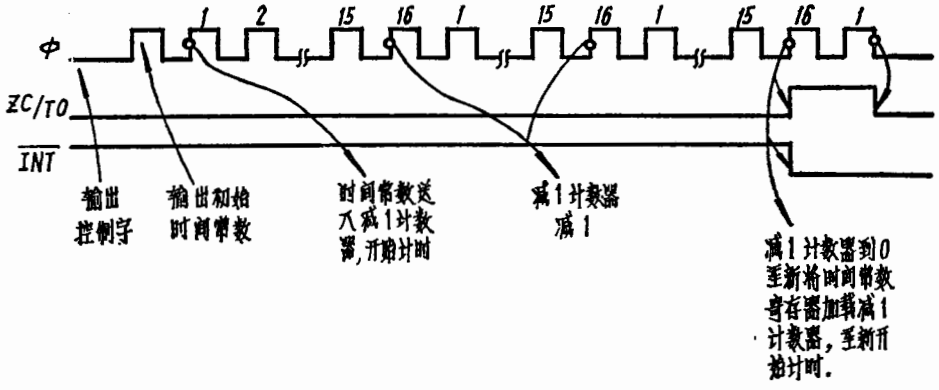


图 6.5　由程序控制的定时器的时间图

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 ϕ_T（见图 6.3）使减 1 计数器减 1，定标器输出第三个 ϕ_T 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。发脉冲的起始和终了也由程序设定。

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  (84),A
LD   A,BDH     设定工作方式
OUT  (84),A
LD   A,03H     装入时间常数
OUT  (84),A
IM   2
IE
```

定时器工作的时间图如图 6.6 所示。

定时器的工作过程如下：当时间常数 03H 装入时间常数寄存器后，定时器等待 LK/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

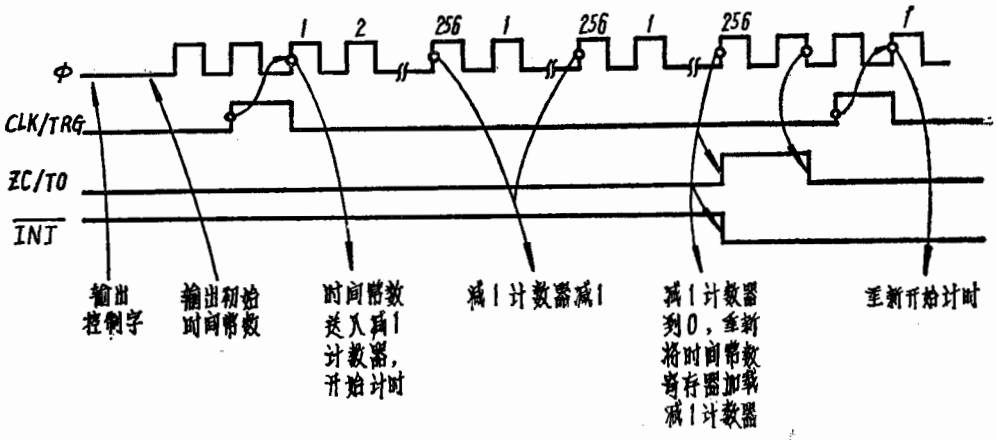


图 6.6　CLK/TRG 由外界控制的定时器的时間图

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。

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。

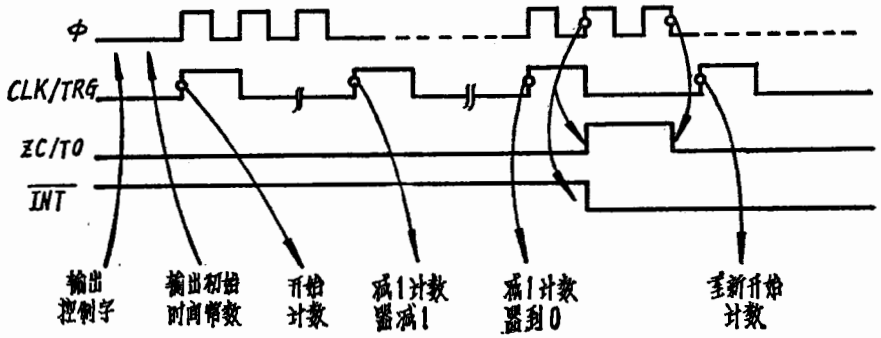


图 6.7　按計数器方式操作的時間图

程　序　设　计

```
LD   A,23H      设定中断矢量
LD   I,A
```

```
LD   A,40H
OUT (84),A
LD   A,D5H     设定工作方式
OUT (84),A
LD   A,03H     装入初始常数
OUT (84),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，计数器开始工作，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/TRG 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，重复上述计数操作。

6.4 CTC 的硬件連接

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。

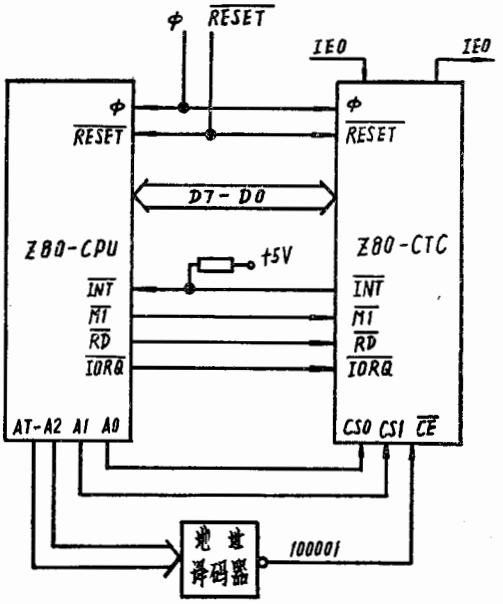


图 6.8　CTC 与 CPU 的硬件連接图