第一部分　单板机实验

第一节　TP801-A单板计算机的结构和原理

TP801-A 是以Z80-CPU为核心的单板微型计算机，其结构如图 1.1 所示。

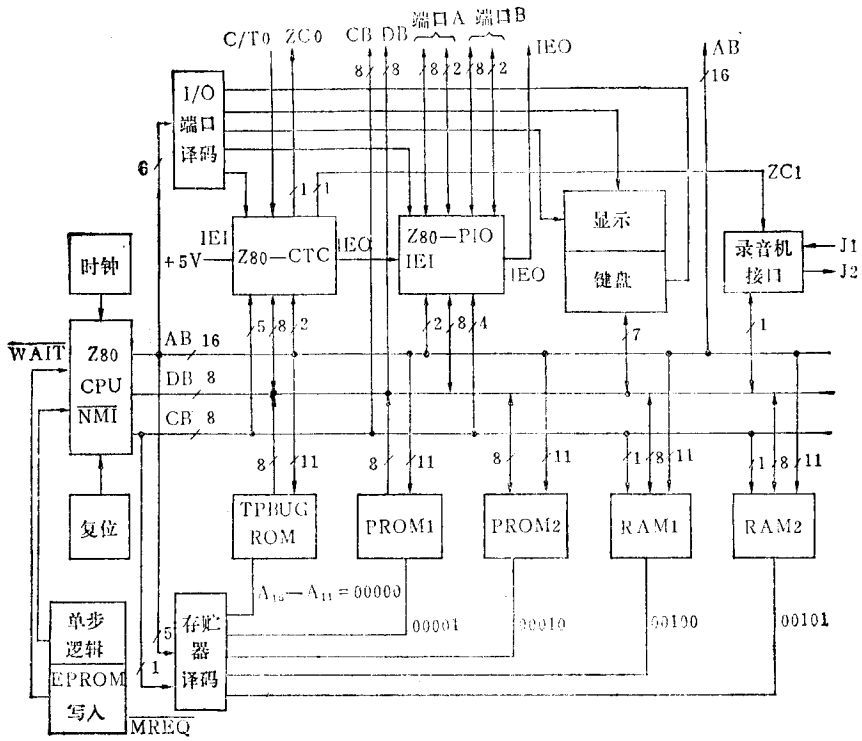


图1.1　TP801-A单板计算机原理框图

它以Z80-CPU 为微处理器，时钟频率为 2MHz。单板机上没有汇编程序，以机器语言进行操作，凡要在TP801 上运行的用户程序，必须经过手工汇编，把它们变为机器码才能在单板机上运行。

TP801 的存贮器分成两大部分：ROM 和 RAM。

ROM 又分为：ROM(其中驻有 TPBUG-A)，PROM1 和 PROM2。每一种都为 2K×8Bit 的片子，构成 2KROM。其中，ROM中存放系统的监控和调试程序TPBUG-A，它用于管理 TP801 的工作，也可用于调试用户的程序。

PROM1 中可以存放固化的用户程序。若TP801 用于一个固定的工作对象，则可把用户程序经过调试后就固化在 PROM1 中。若单板机的开关 S_2 合向下边 (详见后面介绍)，则当系统复位(上电即自动复位，或因按 RESET 按钮引起的复位)后，在经过适当初始化后，就可直接进入和执行固化在PROM1 中的用户程序，而不进入TPBUG-A。

PROM2 是一个 EPROM 片子，典型的为 Intel 2716。在TP801上，可以把在RAM中并经过调试的用户程序，写入到位于 PROM2 上的 EPROM 中。然后可以将此EPROM 片子插至 PROM1 位置，令单板机执行固化在其中的用户程序。

RAM又可分为RAM1和RAM2。由于上述的三种 ROM片子都是 2K × 8 Bit的，所以，RAM也以 2K字节作为一组，分别叫做 RAM1 和RAM2。TP801-A单板计算机的 RAM 容量基本配置是 4K字节，但尚可以按需要扩展，再扩展 4K字节是比较方便的。

RAM可用作用户的程序区和数据区，以及系统和用户的堆栈。RAM中有一部分空间用作系统的数据区。

Z80-CPU 有 16 条地址线，可寻址 64K 存贮空间。而在 TP801-A 中的存贮器，ROM占 6K 存贮空间，RAM 为 4-8K。那么，这些ROM和R AM 的地址是如何分配的呢？RAM 中的用户区是在什么地址范围内呢？这就必须分析一下存贮器的地址连线。TP801-A的存贮器结构如图 1.2 所示。

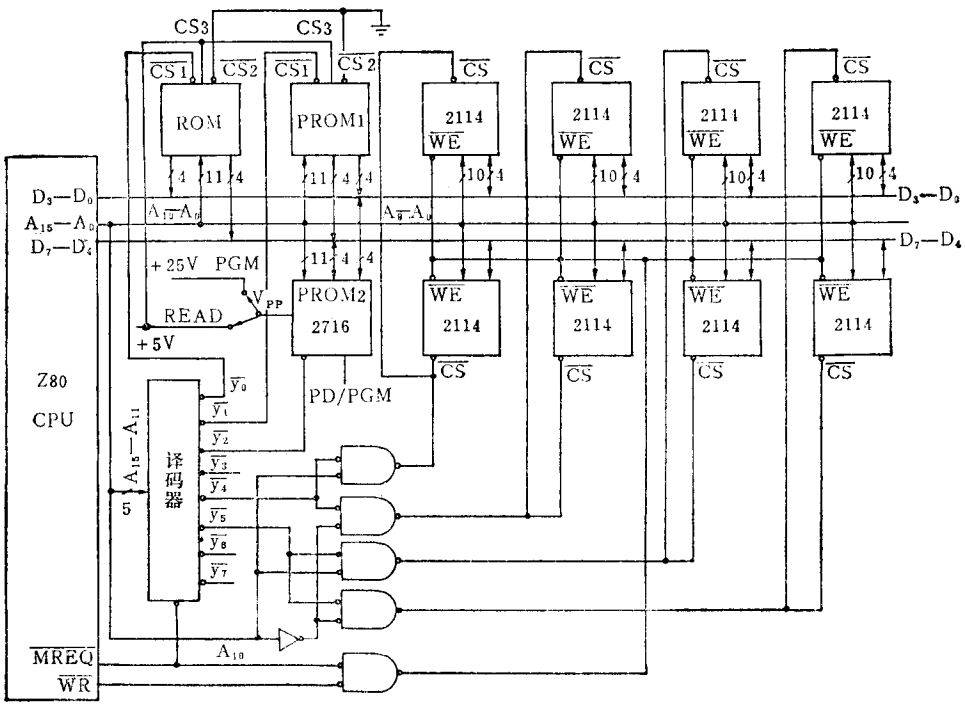
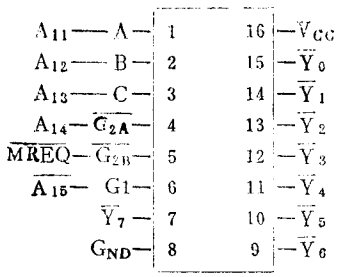


图1.2　TP801-A存贮器结构

ROM、PROM1和PROM2 都是2K × 8 Bit的片子，每个片子上有 11 条地址线，可直接区分片内的2^{11}=2K地址单元，它们可直接连至Z80-CPU地址总线的A_0-A_{10}。ROM和PROM1 的选片端都有 3 个，其中两个中一个为低电平有效的那端直接接地，而另一个高电平有效的那端直接接至＋ 5 V；另一个选片端连至高位地址线的译码输出。

CPU 的高位地址线 A_{15}-A_{11}，连至译码器，如图 1.3 所示。

$\overline{A_{15}}$ 连至选片端 G_1，而 A_{14} 连至选片端$\overline{G2A}$。故要选中此译码器片子，必须 A_{15}= A_{14}= 0 。由于 A_{13}、A_{12}、A_{11} 的不同组合，译码器的输出有效的情况如图 1.3 中的真



G_1	$\overline{G_{2A}}$	$\overline{G_{2B}}$	C	B	A	输出
1	0	0	0	0	0	$\overline{Y}_0=0$ 其余为1
1	0	0	0	0	1	$\overline{Y}_1=0$ 其余为1
1	0	0	0	1	0	$\overline{Y}_2=0$ 其余为1
1	0	0	0	1	1	$\overline{Y}_3=0$ 其余为1
1	0	0	1	0	0	$\overline{Y}_4=0$ 其余为1
1	0	0	1	0	1	$\overline{Y}_5=0$ 其余为1
1	0	0	1	1	0	$\overline{Y}_6=0$ 其余为1
1	0	0	1	1	1	$\overline{Y}_7=0$ 其余为1
不是上述情况			×	×	×	输出全为1

图1.3 RAM的译码电路

值表所示。

由此可确定各个ROM的地址范围：

ROM:	$A_{15}A_{14}A_{13}A_{12}A_{11}$	A_{10}——A_0	
最低地址	0 0 0 0, 0	000, 0000, 0000	0000H
最高地址	0 0 0 0, 0	111, 1111, 1111	07FFH
PROM1:	$A_{15}A_{14}A_{13}A_{12}A_{11}$	A_{10}——A_0	
最低地址	0 0 0 0, 1	000, 0000, 0000	0800H
最高地址	0 0 0 0, 1	111, 1111, 1111	0FFFH
PROM2:	$A_{15}A_{14}A_{13}A_{12}A_{11}$	A_{10}——A_0	
最低地址	0 0 0 1, 0	000, 0000, 0000	1000H
最高地址	0 0 0 1, 0	111, 1111, 1111	17FFH

RAM1和RAM2都是由2114片子组成。2114是1K×4Bit的片子，两片构成1K字节。故，RAM1和RAM2各由4片2114组成。每一个片子上有10条地址线，它们直接连至地址总线的A_0-A_9，可以区分1K地址。RAM1和RAM2由译码器的输出线$\overline{Y}_4$和$\overline{Y}_5$来区分。而RAM1和RAM2的两个1K字节就要由地址线A_{10}来区分。故，$\overline{Y}_4$与A_{10}选择RAM1中的2K地址，$\overline{Y}_5$与A_{10}选择RAM2中的2K地址。由此可以确定RAM的地址范围。

RAM1:

前1K地址	$A_{15}A_{14}A_{13}A_{12}A_{11}$	A_{10}	A_9——A_0	
最低地址	0 0 1 0, 0	0	00, 0000, 0000	2000H
最高地址	0 0 1 0, 0	0	11, 1111, 1111	23FFH
后1K地址	$A_{15}A_{14}A_{13}A_{12}A_{11}$	A_{10}	A_9——A_0	
最低地址	0 0 1 0, 0	1	00, 0000, 0000	2400H
最高地址	0 0 1 0, 0	1	11, 1111, 1111	27FFH
RAM2:	$A_{15}A_{14}A_{13}A_{12}A_{11}$	A_{10}	A_9——A_0	
前1K地址				
最低地址	0 0 1 0, 1	0	00, 0000, 0000	2800H
最高地址	0 0 1 0, 1	0	11, 1111, 1111	2BFFH
后1K地址				

最低地址　0 0 1 0，1　1　00，0000，0000　2C00H

最高地址　0 0 1 0，1　1　11，1111，1111　2FFFH

综合以上讨论，TP801-A中存贮器的地址分配如表 1-1 所示。

2114 片子与 2MHz 的 Z80-CPU 相连接，则即使在取指周期 2114 也能满足CPU的要求，不需要在 CPU 的基本时序中插入T_W状态。

TP801-A的监控程序需要利用 RAM 的一些空间作为它的数据区和堆栈。故RAM区域的分配如表 1-2 所示。

表1-1

地　址	器　件	A_{15}—A_{11}	A_{10}—A_0	译码器的有效输出
0000—07FFH	2K ROM	0 0 0 0 0	可变	$\overline{Y}_0$ = $\overline{CS0}$ = MON SEL
0800—0FFFH	2K PROM1	0 0 0 0 1	可变	$\overline{Y}_1$ = $\overline{CS1}$ = PROM1 SEL
1000—17FFH	2K PROM2	0 0 0 1 0	可变	$\overline{Y}_2$ = $\overline{CS2}$ = PROM2 SEL
1800—1FFFH	没用	0 0 0 1 1	可变	$\overline{Y}_3$ = $\overline{CS3}$　没用
2000—27FFH	2K RAM1	0 0 1 0 0	可变	$\overline{Y}_4$ = $\overline{CS4}$ = RAM1 SEL
2800—2FFFH	2K RAM2	0 0 1 0 1	可变	$\overline{Y}_5$ = $\overline{CS5}$ = RAM2 SEL
3000—37FFH	备用	0 0 1 1 0	可变	$\overline{Y}_6$ = $\overline{CS6}$　没用
3800—3FFFH	备用	0 0 1 1 1	可变	$\overline{Y}_7$ = $\overline{CS7}$　没用

表　1-2

地　址　空　间	用　途	字　节　数
2000—23FFH	RAM1的用户区	1K
2400—27FFH	RAM1的用户区	1K
2800—2BFFH	RAM2的用户区	1K
2C00—2F87H	RAM2的用户区	904
2F88—2F9FH	监控程序堆栈工作区	24
2FA0—2FB7H	用户程序寄存器存放区，用户程序堆栈区	24
2FB8—2FBFH	TP801-A 4个用户程序的入口地址	8
2FC0—2FFFH	TPBUG-A使用的RAM暂存区和断点表	64

TP801-A的 I/O设备及接口电路如图1.4所示。

接有一个Z80-CTC片子，其中通道 1、2 和 3 用于监控程序，通道 0 留给用户用。用户可根据需要把它编程为定时器或计数器用。

有一个 PIO 片子，TP801-A没有使用，故它的两个通道都可留给用户用。

TP801-A的典型输入是键盘，有 16 个 16 进制数字键 和 12 个 功 能 键，如 图 1.5 所示。

这些键的功能和使用方法在后面介绍。

TP801-A 的典型输出设备是 6 个 7 段显示管，可以显示机器码。通常左面 4 个 显示管显示内存单元的地址或 CPU 内部寄存器号，或端口地址；而右面两个，显示该单元或寄存器或外设端口的内容。

此外，TP801-A 还可以与盒式磁带机相连。可以把内存中的内容转贮到磁带上，或

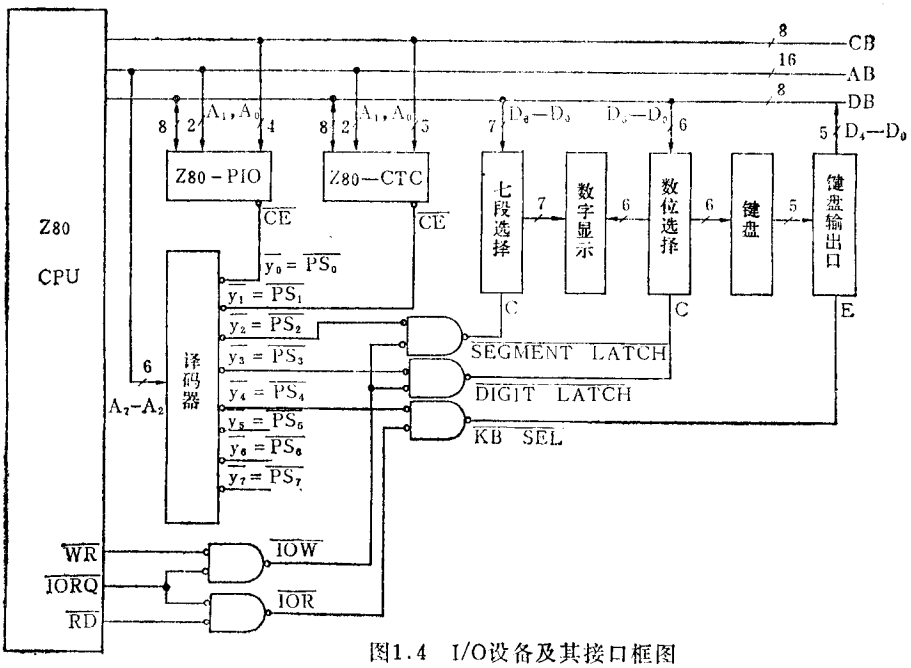


图1.4 I/O设备及其接口框图

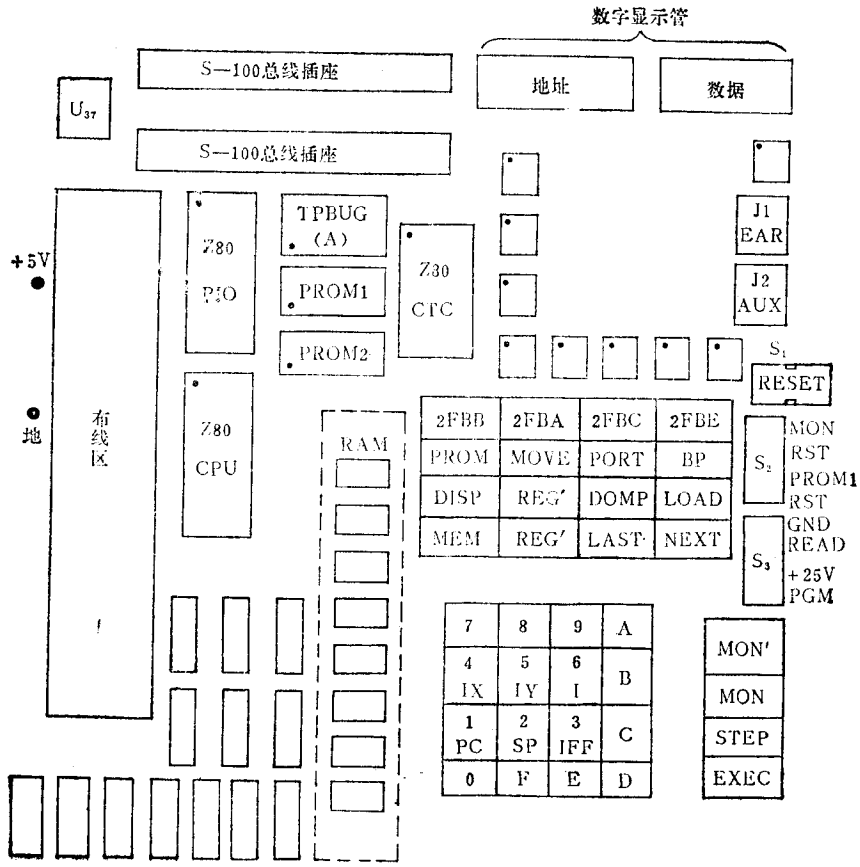


图1.5 TP801-A面板布置图

把磁带上记录的信息输入内存中。

所有上述这些I/O设备和接口电路，都要由地址总线的低8位来加以区别和选择。

Z80-PIO以及Z80-CTC，都要用地址总线的最低两位A_1和A_0来选择片内的不同端口。所以，要由地址总线的高6位——A_7-A_2经过译码器后作为选片信号。于是TP801-A的I/O设备和接口电路的端口地址如表1-3所示。

TP801-A是一个单板计算机，但任何一个计算机都必须在一个系统程序的管理下才能正常工作，在单板机中此程序固化在ROM中，称为监控程序，在TP801-A中即为固化在ROM中的TPBUG-A。

表 1-3

A_7—A_2	译码器输出	器件	A_1A_0	端口	端口地址
100000	$\overline{Y}_0$ = PSO = PIO SEL	Z80—PIO	00	端口A数据	80H
			01	端口B数据	81H
			10	端口A控制寄存器	82H
			11	端口B控制寄存器	83H
100001	$\overline{Y}_1$ = $\overline{PS1}$ = CTC SEL	Z80—CTC	00	通道(Ch)0	84H
			01	通道1	85H
			10	通道2	86H
			11	通道3	87H
100010	$\overline{Y}_2$ = $\overline{PS2}$ = SEGLH	74LS273（八锁存器）	××	七段选择（只写）	88—8BH
100011	$\overline{Y}_3$ = $\overline{PS3}$ = DIGLH	74LS273	××	数位选择（只写）	8C—8FH
100100	$\overline{Y}_4$ = $\overline{PS4}$ = KB SEL	74LS244（八缓冲器）	××	读键值（只读）	90—93H
100101	$\overline{Y}_5$ = $\overline{PS5}$	没使用			94—97H
100110	$\overline{Y}_6$ = PS6	没使用			98—9BH
100111	$\overline{Y}_7$ = $\overline{PS7}$	没使用			9C—9FH

第二节 TPBUG-A介绍

TPBUG-A是一个2K字节的监控和调试程序，它主要是由以下几部分程序组成。

1. 初始化程序；
2. 键盘输入及处理程序；
3. 显示程序；
4. 各个功能键的处理程序；
5. 公用子程序。

这些程序在教材的第十章中做了较详细的分析。

为了实验中和以后工作中的需要，我们在这儿详细介绍一下显示程序与键盘输入程序。

一、显示程序

(一) 数码显示管

TP801-A 中所用的是 5V 电源的七段数字显示管，每一段为一发光二极管，其结构及原理如图 1.6 所示。

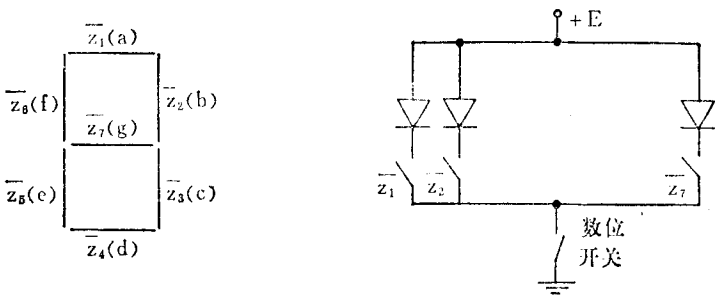


图1.6　显示管原理图

七段，共有七个段控制信号，以控制所显示的字形。当某一段的控制信号为低电平时，则相当于相应的触点闭合，这一段发光；整个管还有一个数位控制信号，也是低电平闭合，只有此触点闭合时，数字管才能发光。

七段用七位数即用 CPU 输出的D_6-D_0控制，D_7始终为 0，因而 16 进制数的显示编码如表 1-4 所示。

TP801-A 中用 6 个这样的数字管，可显示 6 个 16 进制数，其硬件电路如图 1.7 所示。

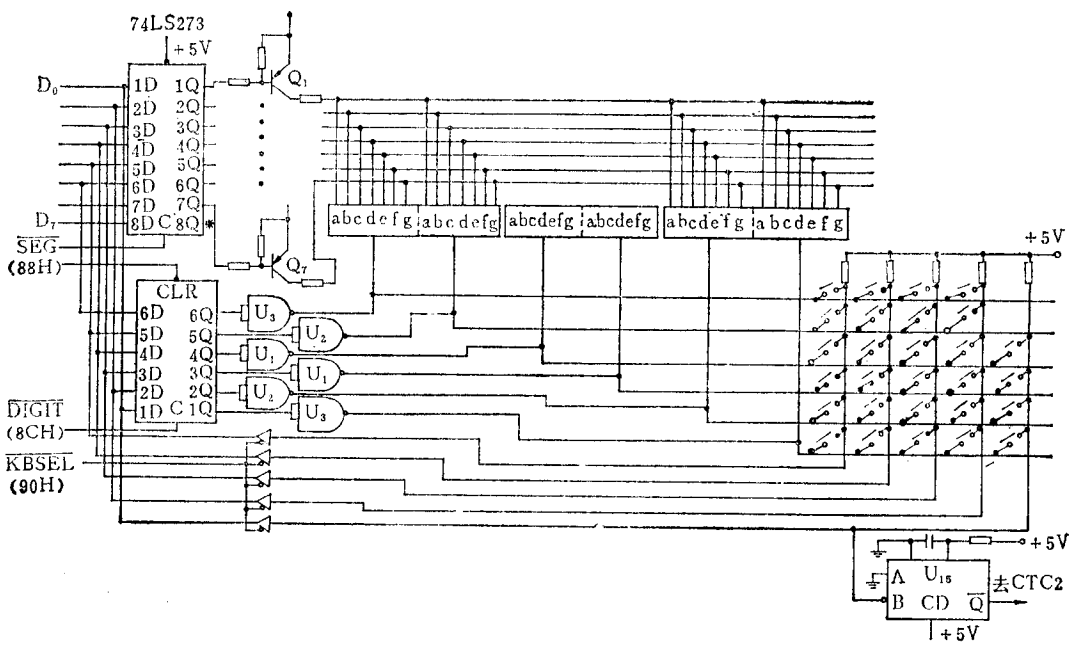


图1.7　键盘输入和显示硬件电路图

表 1-4

数 符	Z_8	Z_7	Z_6	Z_5	Z_4	Z_3	Z_2	Z_1	16进制字模编码
0	0	1	0	0	0	0	0	0	40H
1	0	1	1	1	1	0	0	1	79H
2	0	0	1	0	0	1	0	0	24H
3	0	0	1	1	0	0	0	0	30H
4	0	0	0	1	1	0	0	1	19H
5	0	0	0	1	0	0	1	0	12H
6	0	0	0	0	0	0	1	0	02H
7	0	1	1	1	1	0	0	0	78H
8	0	0	0	0	0	0	0	0	00H
9	0	0	0	1	1	0	0	0	18H
A	0	0	0	0	1	0	0	0	08H
b	0	0	0	0	0	0	1	1	03H
C	0	1	0	0	0	1	1	0	46H
d	0	0	1	0	0	0	0	1	21H
E	0	0	0	0	0	1	1	0	06H
F	0	0	0	0	1	1	1	0	0EH

要显示的字符的编码由数据总线送至锁存器 $\overline{SEG}$（端口地址88H）锁存，后经三极管 Q_1-Q_7，供给大的驱动电流，连到所有数字管相应的段。由于要显示的字模编码是同时连到所有显示管的，所以要使某一个显示管显示出数码，CPU还必须输出一个数位控制字（只有D_0-D_5有用），它由DIGIT锁存器锁存，以选择哪一个显示管工作。所以，要显示一个数码，首先要把这个数码转换成相应的字模编码，再输出给端口88H($\overline{SEG}$)；然后把控制哪一个数字管亮的数位控制字输出给端口8CH($\overline{DIGIT}$)。

（二）显示程序

要显示的数放在以DISMEM(在TP801-A中为2FF7H）单元开始的缓冲区中，总共为6个单元，由于每个数字显示管只能显示一位16进制数，故这6个存贮单元的高4位全为0，低4位为要显示的数的二进制数码。故若要显示一个存贮单元或某一个寄存

器的内容，则必须通过一个子程序把这一字节的高 4 位和低 4 位拆开，放至显示缓冲区的两个单元中。

要显示某一个16进制数，首先要把这个数转换为显示管段形的字模编码，这个转换是由软件通过搜索一个表格（字模表）来实现的。这个表格以 16 进制数的次序，把它们相应的字模编码顺序存放，我们把表格的起始地址送 IX，以要显示的数作为偏移量，由 IX 加上偏移量找到字模编码的地址，则从表格中取出来的即为相应的数的字模编码。

显示程序的流程图如图 1.8 所示。

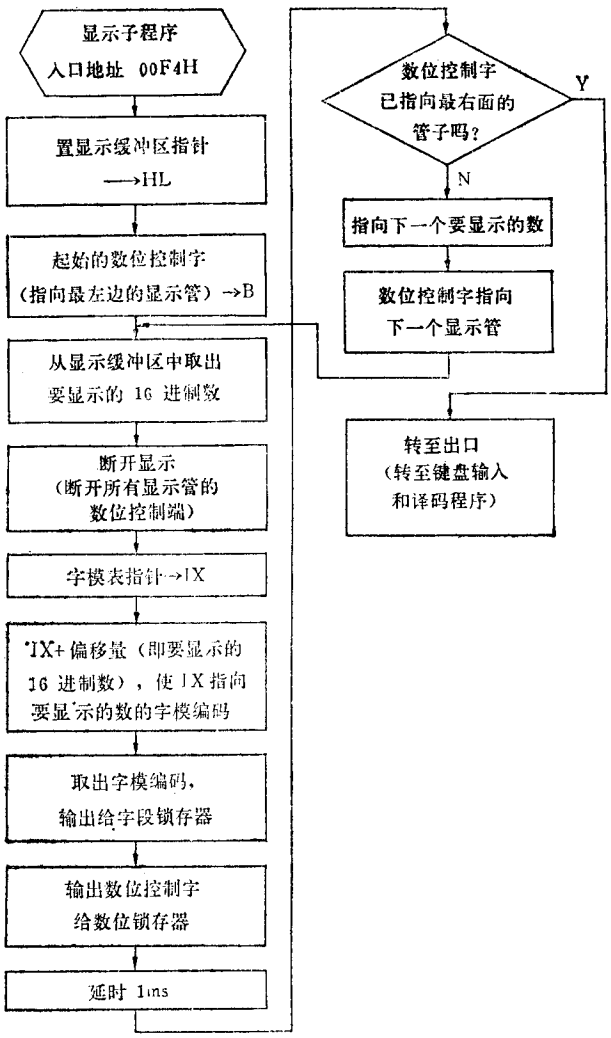


图1.8　显示程序流程图

显示程序为：

```
;          显示程序
; 功能：把数据从 DISMEM 开始的缓冲区送显示管。
```

```
; 输入：要显示的数放在以 DISMEM 开始的 6 个单元中。
; 输出：把预置的 6 个数输出以更新原有的显示。
; 所用的寄存器：HL 是指向 DISMEM 的指针
;               IX 是指向字模表的指针
;               B 是当前数位指示器
;               E保留要被显示的数
;               A和D是暂存寄存器
        ORG     00F4H
DISUP:  LD      HL, DISMEM      ; 指向显示缓冲区的起点
        LD      B, 20H          ; 指向最左边的显示管
DISUP1: LD      E, (HL)         ; 从显示缓冲区取数
        LD      D, 0
        LD      A, 0
        OUT     (DIGLH), A      ; 断开显示
        LD      IX, SEGPT       ; IX 指向字模表起点
        ADD     IX, DE          ; 指向要显示的数的字模
        LD      A, (IX+0)       ; 取出字模编码→A
        OUT     (SEGLH), A      ; 输出至段形锁存器
        LD      A, B
        OUT     (DIGLH), A      ; 输出至数位锁存器（控制是第几个
                                ; 数字显示管亮）
        LD      E, 45           ; 置 1ms 延时
DISUP2: DEC     E
        LD      A, 0
        CP      E
        JR      NZ, DISUP2
        LD      A, 01H
        CP      B               ; B = 1？（即是否已移至最右面的显
                                ; 示管）
        JR      Z, DISUP3       ; 是，转出口
        INC     HL              ; 指向下一个要显示数
        SRL     B               ; 移至下一个显示管
        JR      DISUP1          ; 循环
DISUP3: JP      DECKY           ; 转至键盘输入程序
DIGLH:  EQU     8CH
SEGLH:  EQU     88H
        ORG     07A6H
SEGPT:  DB      40H             ; 0
        DB      79H             ; 1
```

```
DB      24H             ; 2
DB      30H             ; 3
DB      19H             ; 4
DB      12H             ; 5
DB      02H             ; 6
DB      78H             ; 7
DB      00H             ; 8
DB      18H             ; 9
DB      08H             ; A
DB      03H             ; B
DB      46H             ; C
DB      21H             ; D
DB      06H             ; E
DB      0EH             ; F
DB      7FH             ; 空白
DB      0CH             ; 响应符 P
DB      5FH             ; 撇号′,′
```

二、键盘输入程序

在 TP801-A 中是用 16 个数字键和 12 个命令键输入的，但键盘上方的 8 个命令键又可分为上、下两档，每一个键可以有两种命令。每一个键相当于一个按钮，把所按的键转换成相应的数或命令是通过键盘输入与译码程序实现的。

（一）键盘输入硬件电路

键盘输入和显示的硬件电路见图 1.7。

数据总线的信号 D_0-D_5 经反相后输出作为行线（键盘的输入信号线），它由端口 $\overline{\text{DIGIT}}$（地址为 8CH）控制；5 条列线由＋5 V电源经 10K电阻引出（键盘的输出信号线），通过三态缓冲器引至数据总线的 D_4-D_0，由端口 $\overline{\text{KBSEL}}$（地址为 90H）控制。

在每一行与每一列之间接有一个键。

在工作时，程序按行扫描键盘，检查列的输出，由行信号与列信号的组合以确定是哪一个键闭合。即先让 $D_0=1$（反相后为低电平），D_1-$D_5=0$，此数据由端口 8CH输出，即使键盘的最下面一行为低电平，若在此行中有键闭合，则相应的列输出为 0（低电平），而其他列为 1。若这一行没有键闭合（即所有的列全为高电平），则再使上面一行为低电平，再检查列的输出，有键闭合的列输出为 0，否则为 1……，这样一行一行地扫描，由行的值与列的输出的配合，即可确定是哪一个键闭合。

例如，数码 0 闭合，则其对应的行值为 01H，而列的输出即为 0FH；数码 1 则为：行＝02H，列＝0FH；数码 9 则为：行＝04H，列＝1BH……如表 1-5 所示。

键盘输入程序首先检查是否有任一键闭合，若无任一键闭合则转至显示程序。当发现有某一键闭合时，则按行扫描键盘，每扫一行输入列值检查此行中是否有键闭合，若有键闭合，则转至键盘译码程序以确定此键值；若无键闭合，则扫描另一行……。程序

表 1-5

表格中的编码	键值	行值	列值
0FFH	0	B = 01	A = 0F
0EFH	1	B = 02	A = 0F
0F7H	2	B = 02	A = 17
0FBH	3	B = 02	A = 1B
0DFH	4	B = 04	A = 0F
0E7H	5	B = 04	A = 17
0EBH	6	B = 04	A = 1B
0CFH	7	B = 08	A = 0F
0D7H	8	B = 08	A = 17
0DBH	9	B = 08	A = 1B
0DDH	A	B = 08	A = 1D
0EDH	B	B = 04	A = 1D
0FDH	C	B = 02	A = 1D
0DH	D	B = 01	A = 1D
0BH	E	B = 01	A = 1B
07H	F	B = 01	A = 17
0EH	EXEC	B = 01	A = 1E
0FEH	SS	B = 02	A = 1E
0EEH	MON	B = 04	A = 1E
0DEH	MON′	B = 08	A = 1E
0CDH	NEXT LOAD	B = 10	A = 1D
0CBH	LAST DUMP	B = 10	A = 1B
0C7H	REG REG′	B = 10	A = 17
0BFH	MEM DISP	B = 10	A = 0F
0BDH	BP 2FBE	B = 20	A = 1D
0BBH	PORT 2FBC	B = 20	A = 1B
0B7H	MOVE 2FBA	B = 20	A = 17
0AFH	PROG 2FB8	B = 20	A = 0F

流程图如图 1.9 所示。

其程序为：

```
;          键盘译码和执行程序段
; 功能：熄灭数据显示，扫描键盘矩阵，搜索
; 有无键闭合，若有则插入 20ms 延时作为DEBOUNCE(消除键颤动的影响)
; 随后一次扫描一行，检查所有的列，找到闭
; 合的键值。若是数字键，则放到DISMEM中的下
; 一个空的单元（指针在KEYPTR中），若已输入两
; 个数置标志 DIG2，已输入 4 个数置标志 DIG4
; 当已输入了八个数，就调用修改内容子程序。
; 若输入的为命令键，就从转移表中（JPTAB）找
; 到相应命令的处理程序的入口。命令键的执
```

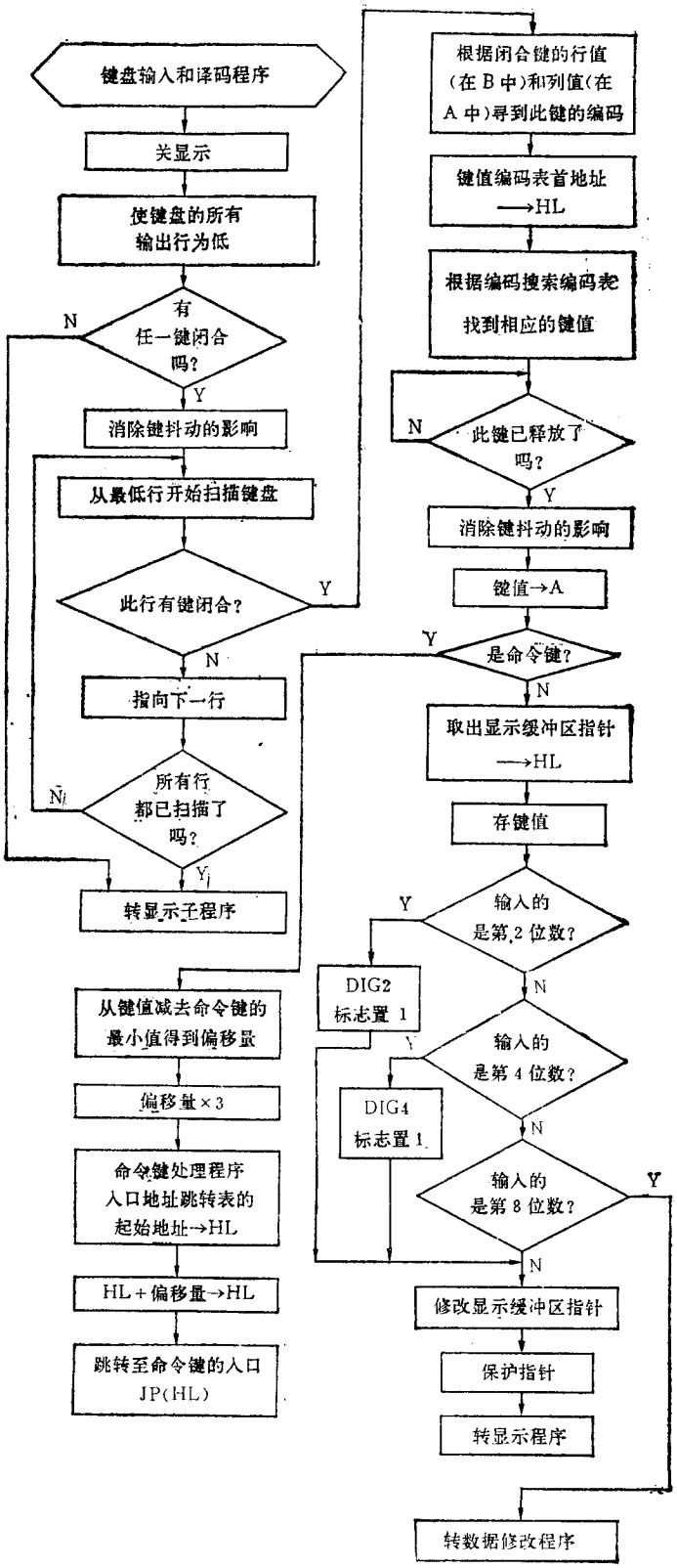


图1.9 键盘输入和译码程序流程图

```
;  行程序也包含在这个模块中。
;  所用的寄存器：
;          A——放键盘值；
;          B——扫描的行值；
;          C——暂存由行和列决定的编码；
;         HL——键值查阅表 KYTBL 指针；
;         HL——存贮器显示缓冲区指针。
                ORG     0123H
DECKY:          LD      A, 7FH          ; 断开显示
                OUT     (SEGLH), A
                LD      A, 3FH
                OUT     (DIGLH), A      ; 输出使所有行为低
                IN      A, (KBSEL)      ; 输入列数据
                AND     1FH             ; 屏蔽掉无用位
                CP      1FH             ; 有任一键闭合吗?
                JP      Z, DISUP        ; 无键闭合，转至显示程序
                CALL    D20MS           ; 有键闭合，调 20ms延时，作
                                        ; 为 DEBOUNCE
                LD      C, DIGLH        ; 端口地址→C
                LD      B, 01H          ; 置使第一行为低的值
KEYDN1:         OUT     (C), B          ; 使所选择的行为低
                IN      A, (KBSEL)      ; 输入列数据
                AND     1FH
                CP      1FH             ; 有键闭合吗?
                JR      NZ, KEYDN2      ; 有，转至键译码程序
                SLA     B               ; 无，选择下一行
                LD      A, 40H
                CP      B               ; 所有行都扫描了吗?
                JR      NZ, KEYDN1      ; 未完，循环
                JP      DISUP           ; 已完，转至显示程序
```

（二）键盘译码程序

如上所述，由行值与列值的结合可找到所按的键，但要找到相应的键值就要从表格中去寻找，为了便于寻找，先要由行值和列值转换为另一个表格内的编码，这可以通过以下的程序段来实现：

```
;          键盘译码程序
;   B中为行值，A中为列值。
KEYDN2:  LD     C, 00H
KEYDN3:  DEC    C
         SRL    B
```

```
        JR    NZ, KEYDN3    ; 若B未全移为0，则使C减1，直至B＝0
        SLA   C
        SLA   C
        SLA   C
        SLA   C
        ADD   A, C          ; 与A中的内容相结合，形成键值在
                            ; 表格中对应的编码
        LD    HL, KYTBL     ; 置键值转换表指针
KEYDN4: CP    (HL)          ; 查表
        JR    Z, KEYDN5     ; 找到，转至KEYDN5
        INC   HL
        INC   B             ; 修改键值
        JR    KEYDN4        ; 继续寻找，当在表中找到相应的编码
                            ; 时，在寄存器B中得到键值
KEYDN5: IN    A, (KBSEL)    ; 输入列值，检查键是否释放
        AND   1FH
        CP    1FH
        JR    NZ, KEYDN5    ; 未释放循环等待
        CALL  D20MS         ; 已释放，调用20ms作为DEBOUNCE
        LD    A, B          ; 键值→A
        CP    10H           ; 是命令键吗?
        JR    NC, KEYDN6    ; 是，转至命令键译码程序
        LD    HL, (KEYPTR)  ; 是数字键，把显示缓冲区地址→HL
        LD    (HL), B       ; 把键值存在缓冲区中
        OR    A             ; 清CY
        LD    BC, DSMEM1
        SBC   HL, BC        ; 已输入两个数?
        JR    Z, KEYDNA     ; 是，转至KEYDNA，置输入两个数标志
        OR    A
        LD    BC, DSMEM3
        LD    HL, (KEYPTR)
        SBC   HL, BC        ; 已输入4个数?
        JR    Z, KEYDN8     ; 是，转至KEYDN8，置输入4个数标志
        OR    A
        LD    BC, DSMEM7
        LD    HL, (KEYPTR)
        SBC   HL, BC        ; 已输入8个数?
        JR    Z, KEYDN9     ; 是，输至KEYDN9，置输入8个数标志
KEYDN7: LD    HL, (KEYPTR)
```

```
        INC   HL
        LD    (KEYPTR), HL ; 存指针
        JP    DISUP        ; 转至显示程序入口，等待输入新的键值
KEYDNA: LD    HL, DIG2
        INC   (HL)         ; 置输入两个数标志
        JR    KEYDN7
KEYDN8: LD    HL, DIG4
        INC   (HL)         ; 置输入四个数标志
        JR    KEYDN7
KEYDN9: CALL  ALTER        ; 输入修改数据到MEM，PORT或REG
        LD    HL, (KEYPTR)
        DEC   HL
        LD    (KEYPTR), HL
        JP    DISUP
; 从命令键转移表中，找到命令键处理程序的入口。
KEYDN6: SUB   10H          ; 得到偏移量
        LD    C, A
        ADD   A, C
        ADD   A, C         ; 偏移量×3
        LD    C, A
        LD    B, 0
        LD    A, (2FFF)    ; 取上、下档键标志
        BIT   0, A
        JR    NZ, DKB      ; 是下档键转至 DKB 处理
        LD    A, C
        CP    0AH
        JR    C, DKB
        LD    HL,JPTAB2-C  ; 设上档键指针
        ADD   HL, BC       ; 加偏移量找到所按命令键的入口
        JP    (HL)         ; 转至相应的命令键的入口
DKB:    LD    HL, JPTAB1   ; 置下档键指针
        ADD   HL, BC
        JP    (HL)         ; 转至入口
        ORG   01DCH
JPTAB1: JP    CCS1         ; EXEC
        JP    CCS2         ; SS
        JP    MON          ; MON
        JP    MON'         ; MON'
        JP    NEXT         ; NEXT
```

```
        JP      LAST            ; LAST
        JP      CCS6            ; REG
        JP      CCS8            ; MEM
        JP      CCS9            ; BP
        JP      CCS7            ; PORT
        JP      MOVE            ; MOVE
        JP      CCS12           ; PROG
        ORG     0205H
JPTAB2: JP      CCS11           ; LOAD
        JP      CCS10           ; DUMP
        JP      CCS5            ; REG'
        JP      DISP            ; DISP
        JP      USER4           ; 用户程序入口地址键 2FBE
        JP      USER3           ; 用户程序入口地址键 2FBC
        JP      USER2           ; 用户程序入口地址键 2FBA
        JP      USER1           ; 用户程序入口地址键 2FB8
        ORG     07DCH
KYTBL:  DB      0FFH            ; 0     B=01, A=0F
(键值转换表)DB  0EFH            ; 1     B=02, A=0F
        DB      0F7H            ; 2     B=02, A=17
        DB      0FBH            ; 3     B=02, A=1B
        DB      0DFH            ; 4     B=04, A=0F
        DB      0E7H            ; 5     B=04, A=17
        DB      0EBH            ; 6     B=04, A=1B
        DB      0CFH            ; 7     B=08, A=0F
        DB      0D7H            ; 8     B=08, A=17
        DB      0DBH            ; 9     B=08, A=1B
        DB      0DDH            ; A     B=08, A=1D
        DB      0EDH            ; B     B=04, A=1D
        DB      0FDH            ; C     B=02, A=1D
        DB      0DH             ; D     B=01, A=1D
        DB      0BH             ; E     B=01, A=1B
        DB      07H             ; F     B=01, A=17
        DB      0EH             ; EXEC  B=01, A=1E
        DB      0FEH            ; SS    B=02, A=1E
        DB      0EEH            ; MON   B=04, A=1E
        DB      0DEH            ; MON'  B=03, A=1E
        DB      0CDH            ; NEXT(LOAD)  B=10, A=1D
        DB      0CBH            ; LAST(DUMP)  B=10, A=1B
```