1. 闪烁灯

1. 实验任务

如图 4.1.1 所示：在 P1.0 端口上接一个发光二极管 L1，使 L1 在不停地一亮一灭，一亮一灭的时间间隔为 0.2 秒。

2. 电路原理图

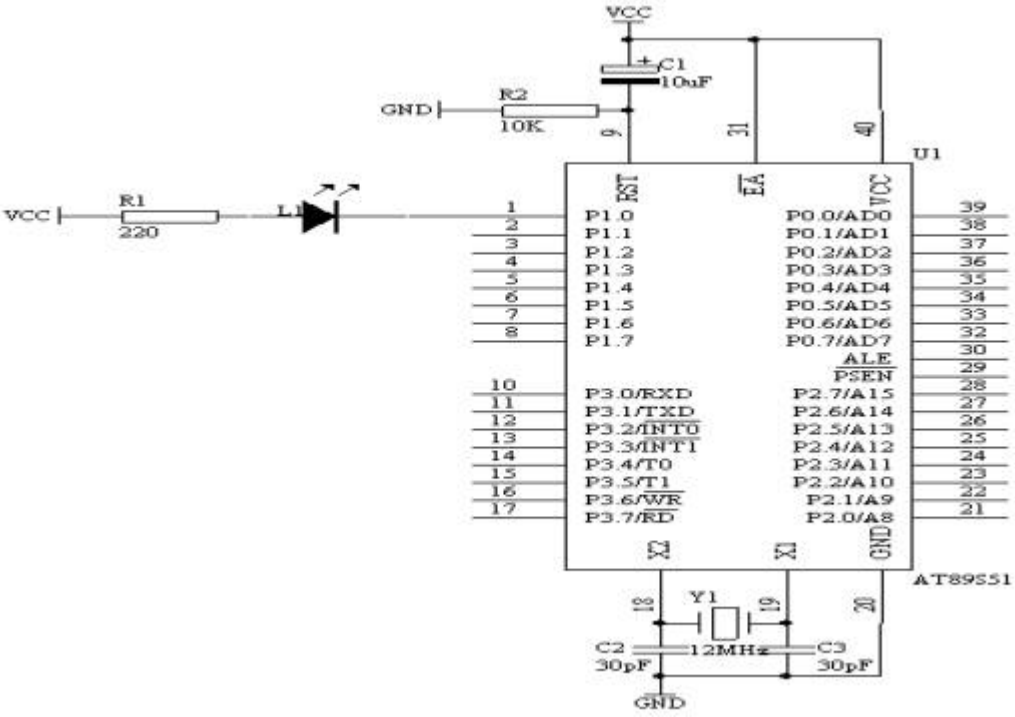


图 4.1.1

3. 系统板上硬件连线

把“单片机系统”区域中的 P1.0 端口用导线连接到“八路发光二极管指示模块”区域中的 L1 端口上。

4. 程序设计内容

（1）. 延时程序的设计方法

作为单片机的指令的执行的时间是很短，数量大微秒级，因此，我们要求的闪烁时间间隔为 0.2 秒，相对于微秒来说，相差太大，所以我们在执行某一指令时，插入延时程序，来达到我们的要求，但这样的延时程序是如何设计呢？下面具体介绍其原理：

如图 4.1.1 所示的石英晶体为 12MHz，因此，1 个机器周期为 1 微秒

机器周期 微秒

MOV R6,#20 2 个机器周期 2

D1: MOV R7,#248 2 个机器周期 2 2＋2×248＝498 20×

DJNZ R7,$ 2 个机器周期 2×248 498

DJNZ R6,D1 2 个机器周期 2×20＝40

10002

因此，上面的延时程序时间为 10.002ms。

由以上可知，当 R6＝10、R7＝248 时，延时 5ms，R6＝20、R7＝248 时，延时 10ms,以此为基本的计时单位。如本实验要求 0.2 秒＝200ms，10ms×R5＝200ms，则 R5＝20，延时子程序如下：

```
DELAY: MOV R5,#20
D1: MOV R6,#20
D2: MOV R7,#248
DJNZ R7,$
DJNZ R6,D2
DJNZ R5,D1
RET
```

（2）． 输出控制

如图 1 所示，当P1.0 端口输出高电平，即P1.0＝1 时，根据发光二极管的单向导电性可知，这时发光二极管 L1 熄灭；当P1.0 端口输出低电平，即 P1.0＝0 时，发光二极管 L1 亮；我们可以使用 SETB P1.0 指令使 P1.0 端口输出高电平，使用 CLR P1.0 指令使 P1.0 端口输出低电平。

5． 程序框图

如图4.1.2所示

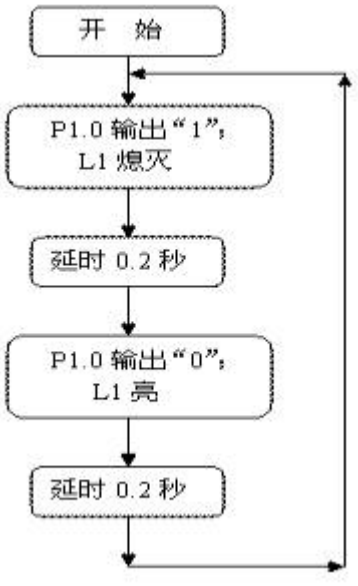


图 4.1.2

6. 汇编源程序

```
ORG 0
START: CLR P1.0
LCALL DELAY
SETB P1.0
LCALL DELAY
LJMP START
DELAY: MOV R5,#20 ;延时子程序，延时 0.2 秒
D1: MOV R6,#20
D2: MOV R7,#248
DJNZ R7,$
DJNZ R6,D2
DJNZ R5,D1
RET
END
```

7. C 语言源程序

```
#include <AT89X51.H>
sbit L1=P1^0;

void delay02s(void) //延时 0.2 秒子程序
{
unsigned char i,j,k;
for(i=20;i>0;i--)
for(j=20;j>0;j--)
for(k=248;k>0;k--);
}


void main(void)
{
while(1)
{
L1=0;
delay02s();
```

```
L1=1;
delay02s();
}
}
```

2. 模拟开关灯

1. 实验任务

如图 4.2.1 所示，监视开关 K1（接在 P3.0 端口上），用发光二极管 L1（接在单片机 P1.0 端口上）显示开关状态，如果开关合上，L1 亮，开关打开，L1 熄灭。

2. 电路原理图

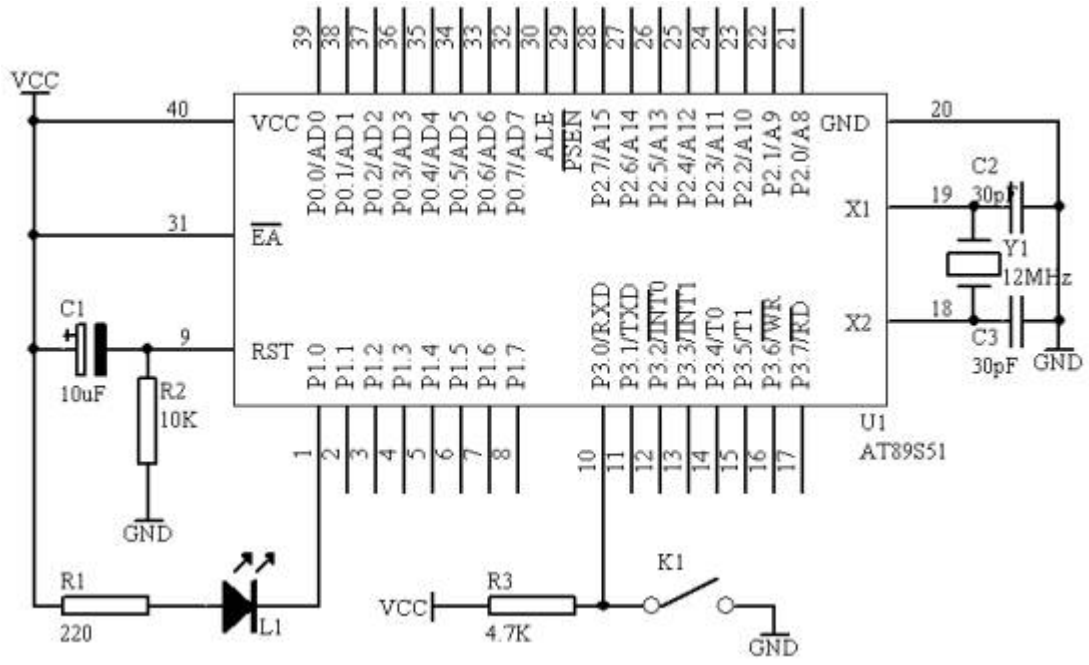


图 4.2.1

3. 系统板上硬件连线

（1）. 把“单片机系统”区域中的 P1.0 端口用导线连接到“八路发光二极管指示模块”区域中的 L1 端口上；

（2）. 把“单片机系统”区域中的 P3.0 端口用导线连接到“四路拨动开关”区域中的 K1 端口上；

4. 程序设计内容

（1）． 开关状态的检测过程

单片机对开关状态的检测相对于单片机来说，是从单片机的 P3.0 端口输入信号，而输入的信号只有高电平和低电平两种，当拨开开关 K1 拨上去，即输入高电平，相当开关断开，当拨动开关 K1 拨下去，即输入低电平，相当开关闭合。单片机可以采用 JB　BIT，REL 或者是 JNB　BIT，REL 指令来完成对开关状态的检测即可。

（2）． 输出控制

如图 3 所示，当 P1.0 端口输出高电平，即 P1.0＝1 时，根据发光二极管的单向导电性可知，这时发光二极管 L1 熄灭；当 P1.0 端口输出低电平，即 P1.0＝0 时，发光二极管 L1 亮；我们可以使用 SETB　P1.0 指令使 P1.0 端口输出高电平，使用 CLR　P1.0 指令使 P1.0 端口输出低电平。

5． 程序框图

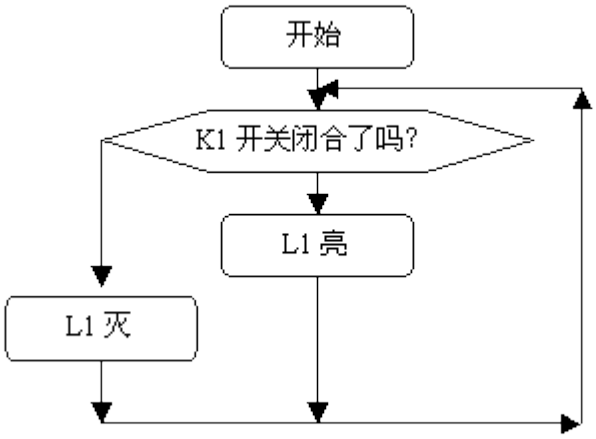


图 4.2.2

6． 汇编源程序 ORG 00H

```
START: JB P3.0,LIG
CLR P1.0
SJMP START
LIG: SETB P1.0
SJMP START
END
```

7． C 语言源程序

```
#include <AT89X51.H>
sbit K1=P3^0;
sbit L1=P1^0;
```

```
void main(void)
{
while(1)
{
if(K1==0)
{
L1=0; //灯亮
}
else
{
L1=1; //灯灭
}
}
}
```

3. 多路开关状态指示

1. 实验任务

如图 4.3.1 所示，AT89S51 单片机的 P1.0－P1.3 接四个发光二极管 L1－L4, P1.4－P1.7 接了四个开关 K1－K4，编程将开关的状态反映到发光二极管上。（开关闭合，对应的灯亮，开关断开，对应的灯灭）。

2. 电路原理图

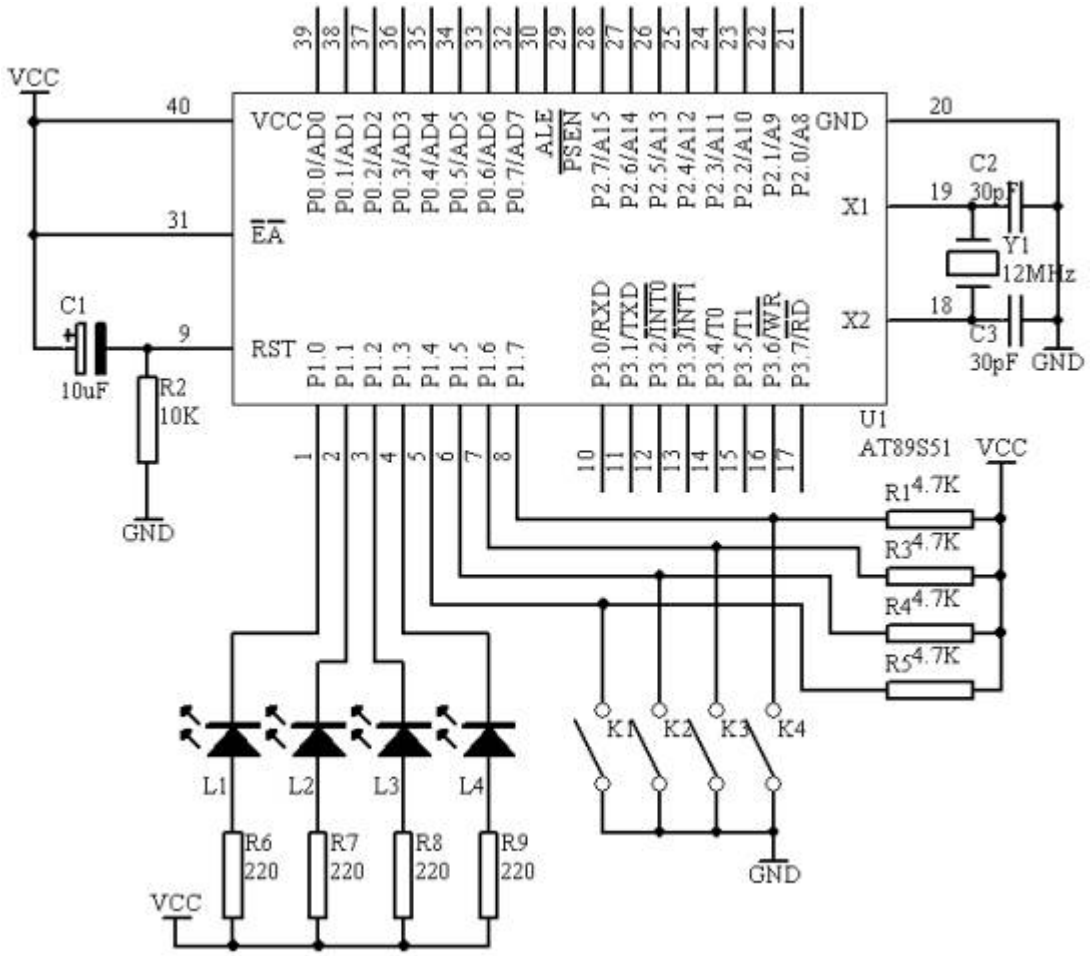


图 4.3.1

3. 系统板上硬件连线

（1. 把“单片机系统”区域中的 P1.0－P1.3 用导线连接到“八路发光二极管指示模块”区域中的 L1－L4 端口上；

（2. 把“单片机系统”区域中的 P1.4－P1.7 用导线连接到“四路拨动开关”区域中的 K1－K4 端口上；

4. 程序设计内容

（1. 开关状态检测

对于开关状态检测，相对单片机来说，是输入关系，我们可轮流检测每个开关状态，根据每个开关的状态让相应的发光二极管指示，可以采用 JB　P1.X，REL 或 JNB　P1.X，REL 指令来完成；也可以一次性检测四路开关状态，然后让其指示，可以采用 MOV　A，P1 指令一次把 P1 端口的状态全部读入，然后取高 4 位的状态来指示。

（2． 输出控制

根据开关的状态，由发光二极管 L1－L4 来指示，我们可以用 SETB P1.X 和 CLR P1.X 指令来完成，也可以采用 MOV P1，＃1111XXXXB 方法一次指示。

5． 程序框图

读 P1 口数据到 A_{CC} 中

A_{CC} 内容右移 4 次

A_{CC} 内容与 F0H 相或

A_{CC} 内容送入 P1 口

<![endif]-->

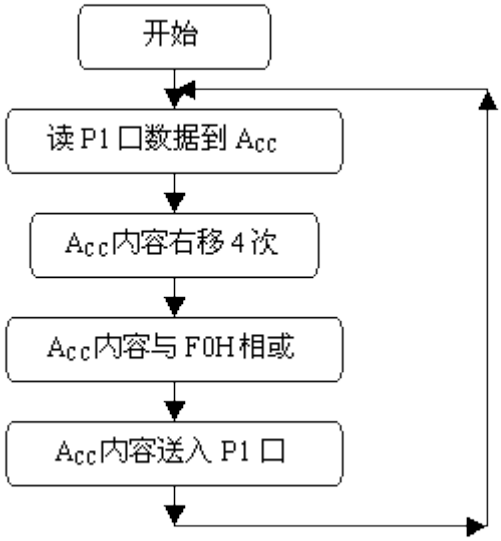


图 4.3.2

6． 方法一（汇编源程序）

```
ORG 00H
START: MOV A,P1
ANL A,#0F0H
RR A
RR A
RR A
RR A
ORl A,#0F0H
MOV P1,A
SJMP START
```

```
END
```
7. 方法一（C 语言源程序）
```
#include <AT89X51.H>
unsigned char temp;

void main(void)
{
while(1)
{
temp=P1>>4;
temp=temp | 0xf0;
P1=temp;
}
}
```
8. 方法二（汇编源程序）
```
ORG 00H
START: JB P1.4,NEXT1
CLR P1.0
SJMP NEX1
NEXT1: SETB P1.0
NEX1: JB P1.5,NEXT2
CLR P1.1
SJMP NEX2
NEXT2: SETB P1.1
NEX2: JB P1.6,NEXT3
CLR P1.2
SJMP NEX3
NEXT3: SETB P1.2
NEX3: JB P1.7,NEXT4
CLR P1.3
SJMP NEX4
NEXT4: SETB P1.3
NEX4: SJMP START
END
```
9. 方法二（C 语言源程序）
```
#include <AT89X51.H>

void main(void)
{
while(1)
{
if(P1_4==0)
{
P1_0=0;
```

```
}
else
{
P1_0=1;
}
if(P1_5==0)
{
P1_1=0;
}
else
{
P1_1=1;
}
if(P1_6==0)
{
P1_2=0;
}
else
{
P1_2=1;
}
if(P1_7==0)
{
P1_3=0;
}
else
{
P1_3=1;
}
}
}
```

4. 广告灯的左移右移

1. 实验任务

做单一灯的左移右移，硬件电路如图 4.4.1 所示，八个发光二极管 L1－L8 分别接在单片机的 P1.0－P1.7 接口上，输出“0”时，发光二极管亮，开始时 P1.0→P1.1→P1.2→P1.3→┅→P1.7→P1.6→┅→P1.0 亮，重复循环。

2. 电路原理图

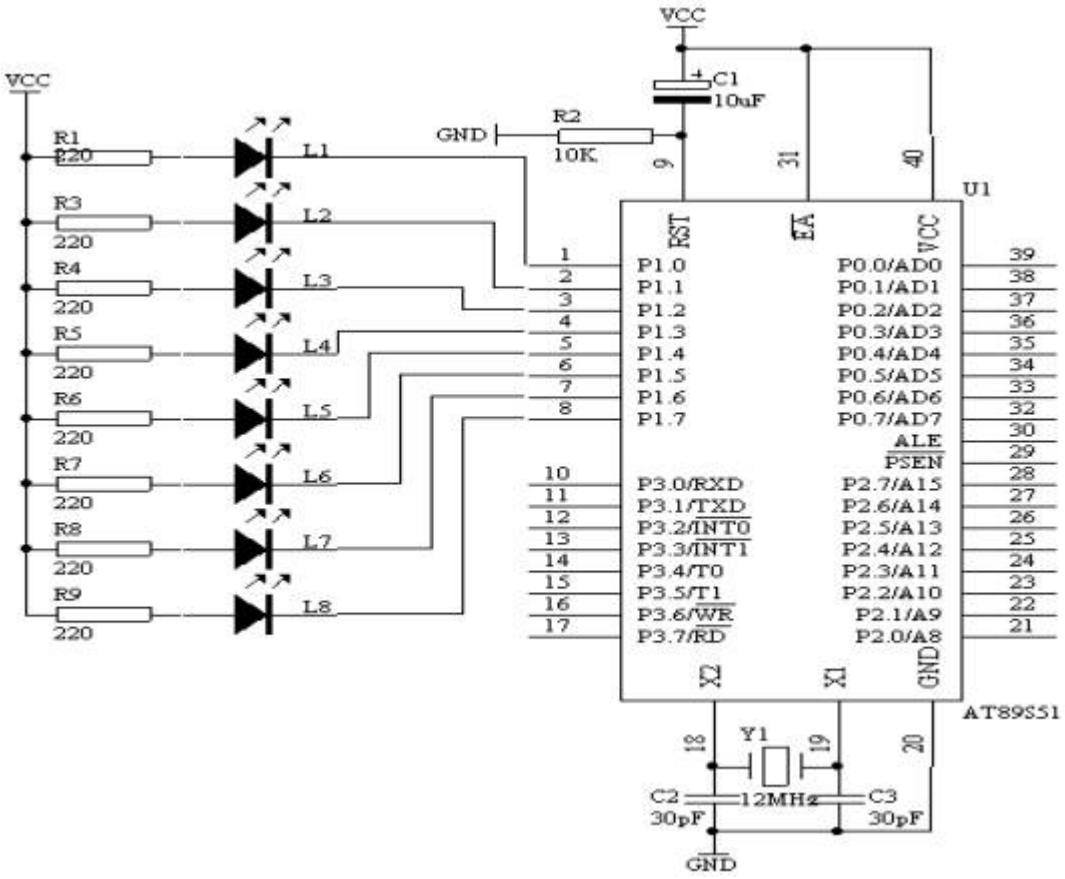


图 4.4.1

3. 系统板上硬件连线

把“单片机系统”区域中的 P1.0－P1.7 用 8 芯排线连接到“八路发光二极管指示模块”区域中的 L1－L8 端口上，要求：P1.0 对应着 L1，P1.1 对应着 L2，……，P1.7 对应着 L8。

4. 程序设计内容

我们可以运用输出端口指令 MOV P1，A 或 MOV P1，＃DATA，只要给累加器值或常数值，然后执行上述的指令，即可达到输出控制的动作。

每次送出的数据是不同，具体的数据如下表 1 所示 ：

P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	说明
L8	L7	L6	L5	L4	L3	L2	L1	
1	1	1	1	1	1	1	0	L1 亮
1	1	1	1	1	1	0	1	L2 亮
1	1	1	1	1	0	1	1	L3 亮
1	1	1	1	0	1	1	1	L4 亮
1	1	1	0	1	1	1	1	L5 亮

1	1	0	1	1	1	1	1	L6 亮
1	0	1	1	1	1	1	1	L7 亮
0	1	1	1	1	1	1	1	L8 亮

表 1

5. 程序框图

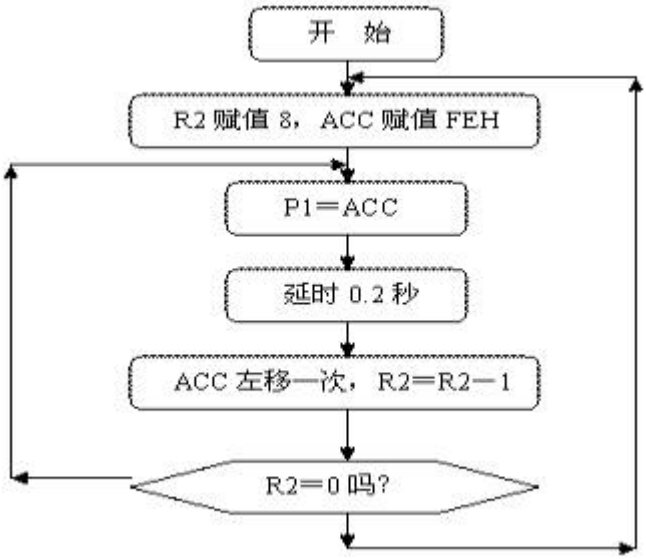


图 4.4.2

6. 汇编源程序

```
ORG 0
START: MOV R2,#8
MOV A,#0FEH
SETB C
LOOP: MOV P1,A
LCALL DELAY
RLC A
DJNZ R2,LOOP
MOV R2,#8
LOOP1: MOV P1,A
LCALL DELAY
RRC A
DJNZ R2,LOOP1
LJMP START
DELAY: MOV R5,#20 ;
D1: MOV R6,#20
D2: MOV R7,#248
DJNZ R7,$
DJNZ R6,D2
DJNZ R5,D1
```

```
RET
END
```

7. C 语言源程序

```
#include <AT89X51.H>
unsigned char i;
unsigned char temp;
unsigned char a,b;

void delay(void)
{
unsigned char m,n,s;
for(m=20;m>0;m--)
for(n=20;n>0;n--)
for(s=248;s>0;s--);
}
void main(void)
{
while(1)
{
temp=0xfe;
P1=temp;
delay();
for(i=1;i<8;i++)
{
a=temp<<i;
b=temp>>(8-i);
P1=a|b;
delay();
}
for(i=1;i<8;i++)
{
a=temp>>i;
b=temp<<(8-i);
P1=a|b;
delay();
}
}
}
```

5. 广告灯（利用取表方式）

1. 实验任务

利用取表的方法，使端口 P1 做单一灯的变化：左移 2 次，右移 2 次，闪烁 2 次（延时的时间 0.2 秒）。

2. 电路原理图

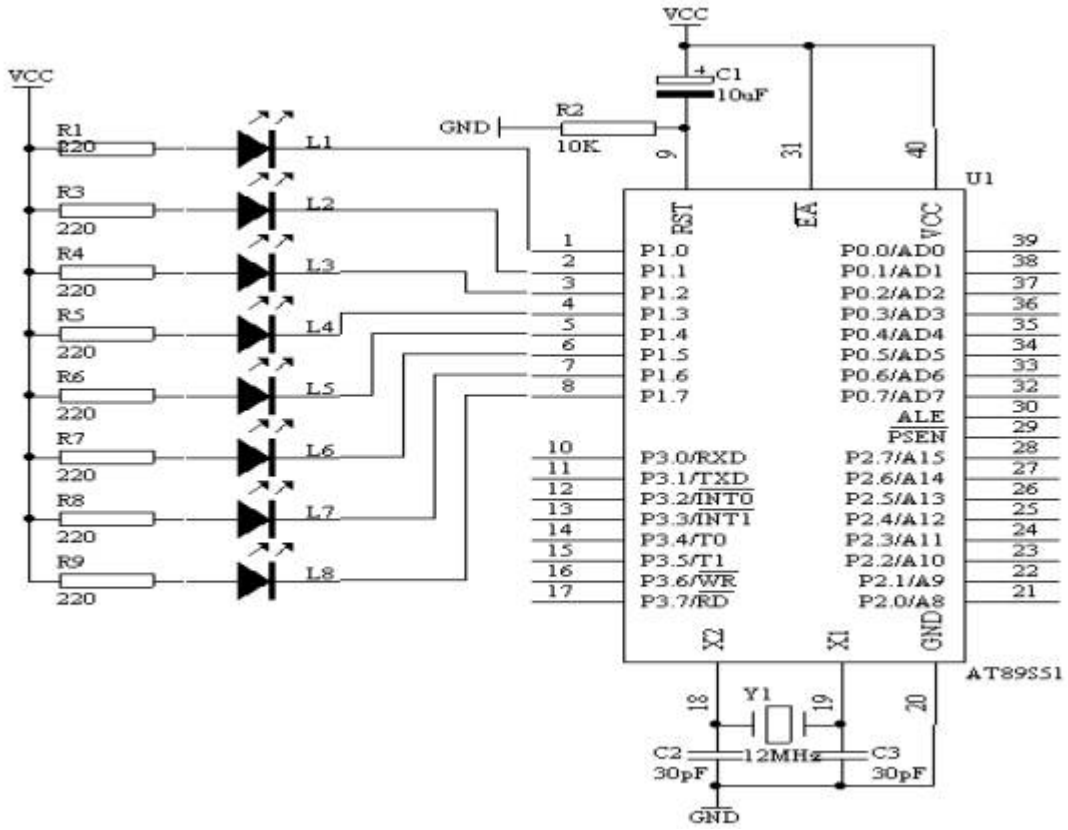


图 4.5.1

3. 系统板上硬件连线

把“单片机系统”区域中的 P1.0—P1.7 用 8 芯排线连接到“八路发光二极管指示模块”区域中的 L1—L8 端口上，要求：P1.0 对应着 L1，P1.1 对应着 L2，……，P1.7 对应着 L8。

4. 程序设计内容

在用表格进行程序设计的时候，要用以下的指令来完成

（1）. 利用 MOV DPTR，＃DATA16 的指令来使数据指针寄存器指到表的开头。

（2）. 利用 MOVC A，@A＋DPTR 的指令，根据累加器的值再加上 DPTR 的值，就可以使程序计数器 PC 指到表格内所要取出的数据。

因此，只要把控制码建成一个表，而利用 MOVC　A，@A＋DPTR 做取码的操作，就可方便地处理一些复杂的控制动作，取表过程如下图所示：

5.　程序框图

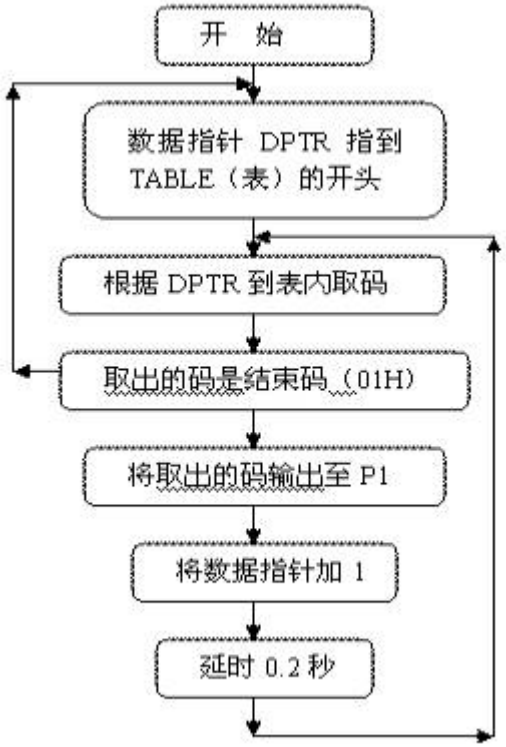


图 4.5.2

6.　汇编源程序

```
ORG 0
START: MOV DPTR,#TABLE
LOOP: CLR A
MOVC A,@A+DPTR
CJNE A,#01H,LOOP1
JMP START
LOOP1: MOV P1,A
MOV R3,#20
LCALL DELAY
INC DPTR
JMP LOOP
DELAY: MOV R4,#20
D1: MOV R5,#248
DJNZ R5,$
DJNZ R4,D1
DJNZ R3,DELAY
```

```
RET
TABLE: DB 0FEH,0FDH,0FBH,0F7H
DB 0EFH,0DFH,0BFH,07FH
DB 0FEH,0FDH,0FBH,0F7H
DB 0EFH,0DFH,0BFH,07FH
DB 07FH,0BFH,0DFH,0EFH
DB 0F7H,0FBH,0FDH,0FEH
DB 07FH,0BFH,0DFH,0EFH
DB 0F7H,0FBH,0FDH,0FEH
DB 00H, 0FFH,00H, 0FFH
DB 01H
END
```

7. C 语言源程序

```
#include <AT89X51.H>
unsigned char code table[]={0xfe,0xfd,0xfb,0xf7,
0xef,0xdf,0xbf,0x7f,
0xfe,0xfd,0xfb,0xf7,
0xef,0xdf,0xbf,0x7f,
0x7f,0xbf,0xdf,0xef,
0xf7,0xfb,0xfd,0xfe,
0x7f,0xbf,0xdf,0xef,
0xf7,0xfb,0xfd,0xfe,
0x00,0xff,0x00,0xff,
0x01};
unsigned char i;

void delay(void)
{
unsigned char m,n,s;
for(m=20;m>0;m--)
for(n=20;n>0;n--)
for(s=248;s>0;s--);
}

void main(void)
{
while(1)
{
if(table[i]!=0x01)
{
P1=table[i];
i++;
delay();
}
```

```
else
{
i=0;
}
}
}
```

6. 报警产生器

1. 实验任务

用 P1.0 输出 1KHz 和 500Hz 的音频信号驱动扬声器，作报警信号，要求 1KHz 信号响 100ms，500Hz 信号响 200ms,交替进行，P1.7 接一开关进行控制，当开关合上响报警信号，当开关断开告警信号停止，编出程序。

2. 电路原理图

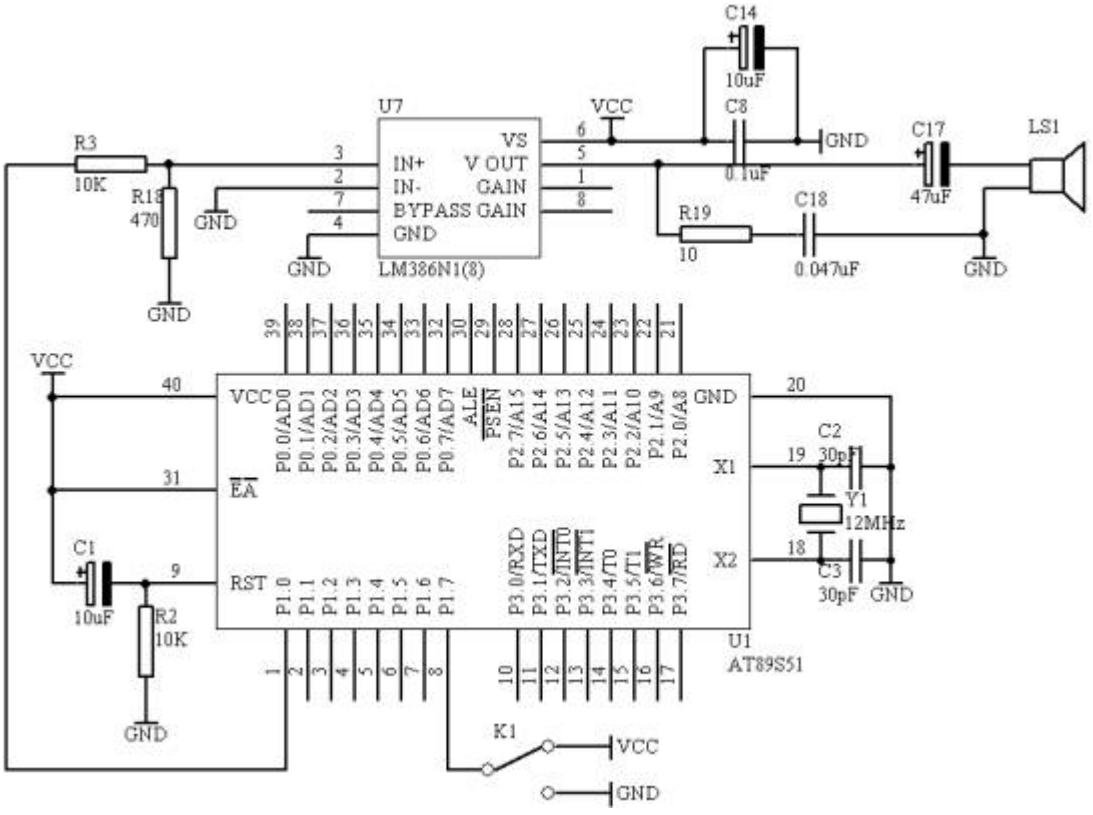


图 4.6.1

3. 系统板上硬件连线

（1. 把“单片机系统”区域中的 P1.0 端口用导线连接到“音频放大模块”区域中的 SPK IN 端口上；

（2. 在“音频放大模块”区域中的 SPK OUT 端口上接上一个 8 欧的或者是 16 欧的喇叭；

（3. 把“单片机系统”区域中的 P1.7/RD 端口用导线连接到“四路拨动开关”区域中的 K1 端口上；

4. 程序设计内容

（1. 信号产生的方法

500Hz 信号周期为 2ms，信号电平为每 1ms 变反 1 次，1KHz 的信号周期为 1ms，信号电平每 500us 变反 1 次；

5. 程序框图

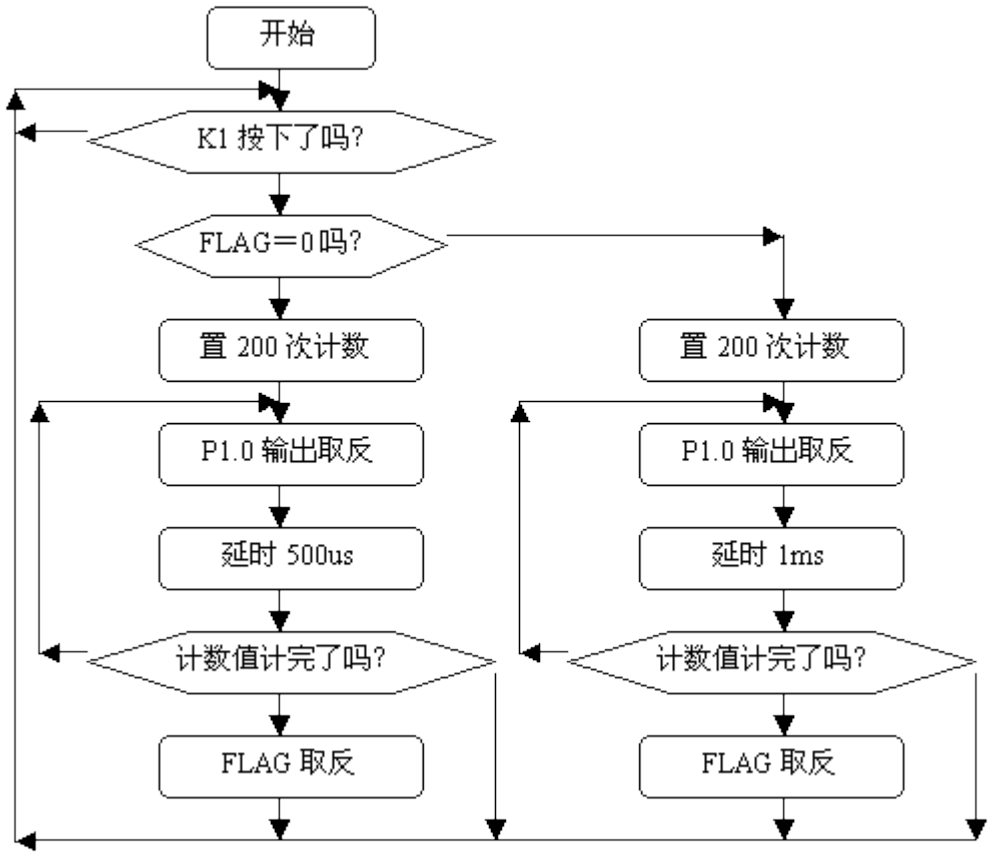


图 4.6.2

6. 汇编源程序

```
FLAG BIT 00H
ORG 00H
START: JB P1.7,START
JNB FLAG,NEXT
MOV R2,#200
DV: CPL P1.0
LCALL DELY500
LCALL DELY500
DJNZ R2,DV
CPL FLAG
NEXT: MOV R2,#200
DV1: CPL P1.0
LCALL DELY500
DJNZ R2,DV1
CPL FLAG
SJMP START
DELY500: MOV R7,#250
LOOP: NOP
DJNZ R7,LOOP
RET
END
```

7. C 语言源程序

```
#include <AT89X51.H>
#include <INTRINS.H>

bit flag;
unsigned char count;

void dely500(void)
{
unsigned char i;
for(i=250;i>0;i--)
{
_nop_();
}
}

void main(void)
{
while(1)
{
if(P1_7==0)
{
```

```
for(count=200;count>0;count--)
{
P1_0=~P1_0;
dely500();
}
for(count=200;count>0;count--)
{
P1_0=~P1_0;
dely500();
dely500();
}
}
}
```

7. I/O 并行口直接驱动 LED 显示

1. 实验任务

如图 13 所示，利用 AT89S51 单片机的 P0 端口的 P0.0－P0.7 连接到一个共阴数码管的 a－h 的笔段上，数码管的公共端接地。在数码管上循环显示 0－9 数字，时间间隔 0.2 秒。

2. 电路原理图

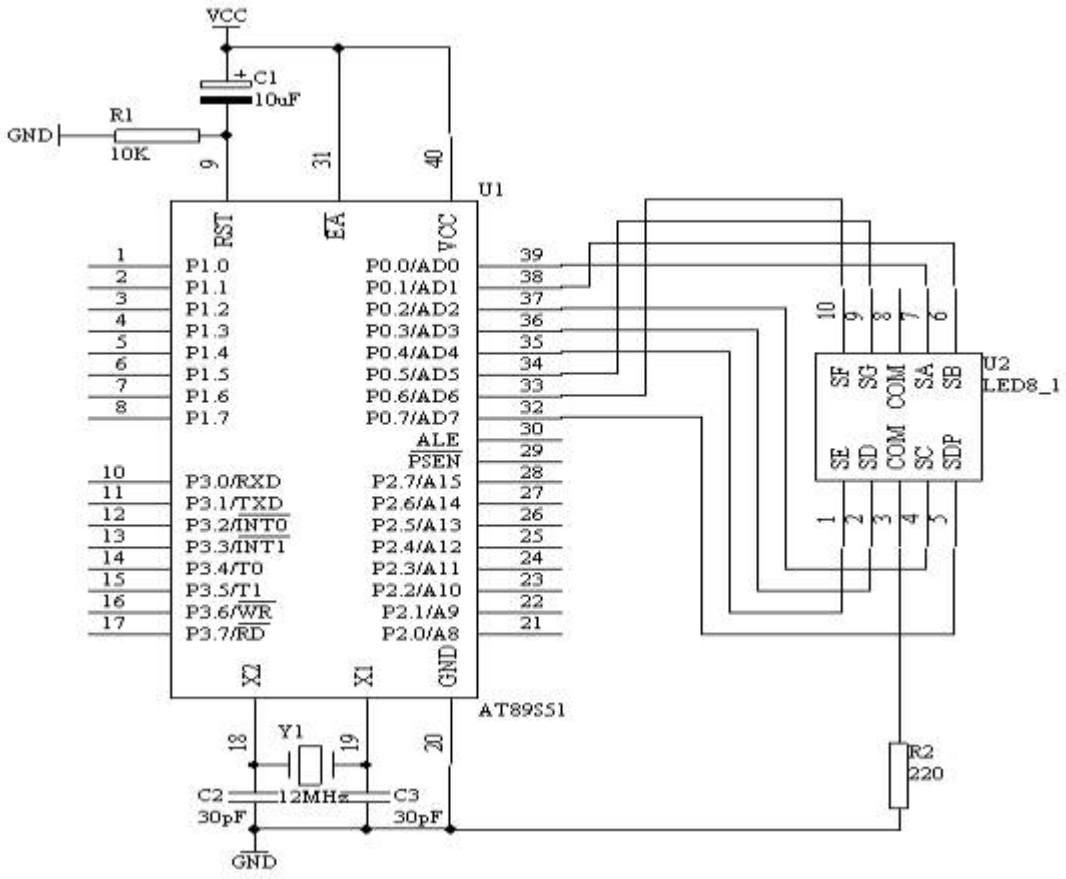


图 4.7.1

3. 系统板上硬件连线

把“单片机系统”区域中的 P0.0/AD0－P0.7/AD7 端口用 8 芯排线连接到“四路静态数码显示模块”区域中的任一个数码管的 a－h 端口上；要求：P0.0/AD0 与 a 相连，P0.1/AD1 与 b 相连，P0.2/AD2 与 c 相连，……，P0.7/AD7 与 h 相连。

4. 程序设计内容

（1. LED 数码显示原理

七段 LED 显示器内部由七个条形发光二极管和一个小圆点发光二极管组成，根据各管的极管的接线形式，可分成共阴极型和共阳极型。

LED 数码管的 g~a 七个发光二极管因加正电压而发亮，因加零电压而不以发亮，不同亮暗的组合就能形成不同的字形，这种组合称之为字形码，下面给出共阴极的字形码见表 2

“0”	3FH		“8”	7FH	
“1”	06H		“9”	6FH	
“2”	5BH		“A”	77H	

“3”	4FH		“b”	7CH	
“4”	66H		“C”	39H	
“5”	6DH		“d”	5EH	
“6”	7DH		“E”	79H	
“7”	07H		“F”	71H	

（2. 由于显示的数字 0－9 的字形码没有规律可循，只能采用查表的方式来完成我们所需的要求了。这样我们按着数字 0－9 的顺序，把每个数字的笔段代码按顺序排好!建立的表格如下所示:TABLE DB 3FH,06H,5BH,4FH,66H,6DH,7DH,07H，7FH，6FH

5.程序框图

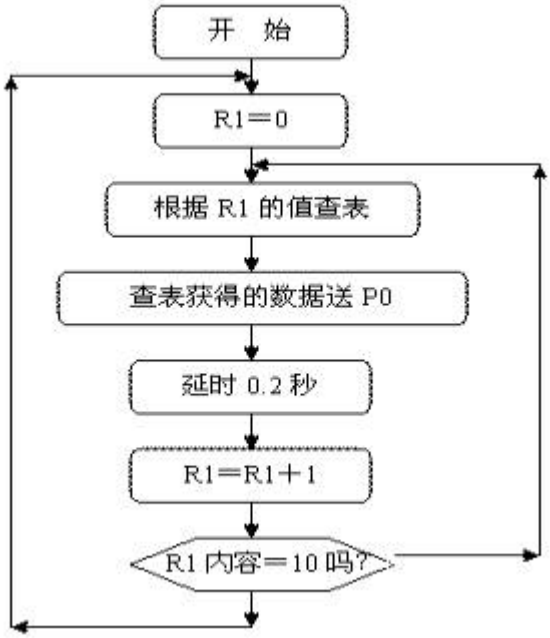


图 4.7.2

6. 汇编源程序

```
ORG 0
START: MOV R1,#00H
NEXT: MOV A,R1
MOV DPTR,#TABLE
MOVC A,@A+DPTR
MOV P0,A
LCALL DELAY
INC R1
CJNE R1,#10,NEXT
LJMP START
DELAY: MOV R5,#20
D2: MOV R6,#20
D1: MOV R7,#248
```

```
DJNZ R7,$
DJNZ R6,D1
DJNZ R5,D2
RET
TABLE: DB 3FH,06H,5BH,4FH,66H,6DH,7DH,07H,7FH,6FH
END
```

7. C 语言源程序

```
#include <AT89X51.H>
unsigned char code table[]={0x3f,0x06,0x5b,0x4f,0x66,
0x6d,0x7d,0x07,0x7f,0x6f};
unsigned char dispcount;

void delay02s(void)
{
unsigned char i,j,k;
for(i=20;i>0;i--)
for(j=20;j>0;j--)
for(k=248;k>0;k--);
}

void main(void)
{
while(1)
{
for(dispcount=0;dispcount<10;dispcount++)
{
P0=table[dispcount];
delay02s();
}
}
}
```

8. 按键识别方法之一

1. 实验任务

每按下一次开关 SP1，计数值加 1，通过 AT89S51 单片机的 P1 端口的 P1.0 到 P1.3 显示出其的二进制计数值。

2. 电路原理图

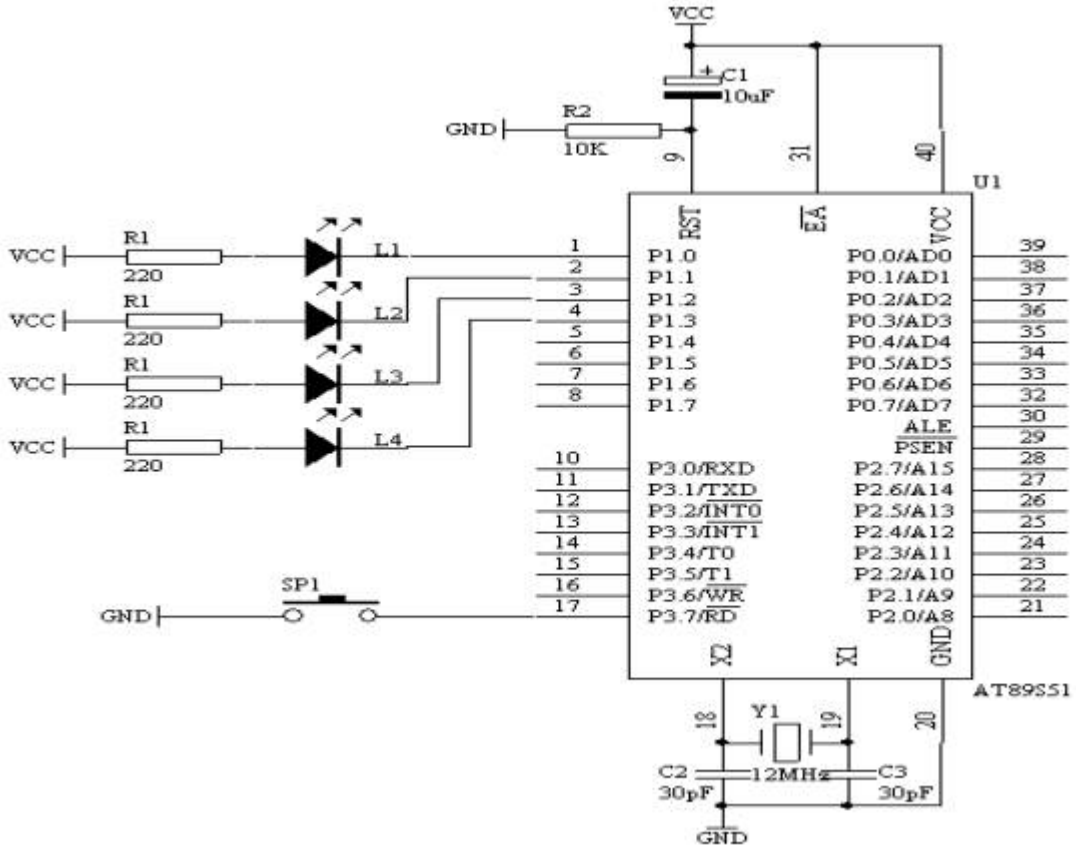


图 4.8.1

3. 系统板上硬件连线

（1. 把“单片机系统”区域中的 P3.7/RD 端口连接到“独立式键盘”区域中的 SP1 端口上；

（2. 把“单片机系统”区域中的 P1.0－P1.4 端口用 8 芯排线连接到“八路发光二极管指示模块”区域中的“L1－L8”端口上；要求，P1.0 连接到 L1，P1.1 连接到 L2，P1.2 连接到 L3，P1.3 连接到 L4 上。

4. 程序设计方法

（1. 其实，作为一个按键从没有按下到按下以及释放是一个完整的过程，也就是说，当我们按下一个按键时，总希望某个命令只执行一次，而在按键按下的 过程中，不要有干扰进来，因为，在按下的过程中，一旦有干扰过来，可能造成误触发过程，这并不是我们所想要的。因此在按键按下的时候，

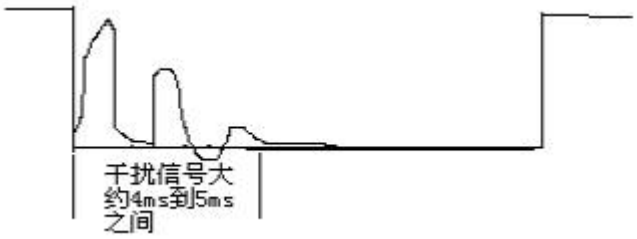


图 4.8.2

要把我们手上的干扰信号以及按键的机械接触等干扰信号给滤除掉，一般情况下，我们可以采用电容来滤除掉这些干扰信号，但实际上，会增加硬件成本及硬件电路的体积，这是我们不希望，总得有个办法解决这个问题，因此我们可以采用软件滤波的方法去除这些干扰

信号，一般情况下，一个按键按下的时候，总是在按下的时刻存在着一定的干扰信号，按下之后就基本上进入了稳定的状态。具体的一个按键从按下到释放的全过程的信号图如上图所示：

从图中可以看出，我们在程序设计时，从按键被识别按下之后，延时 5ms 以上，从而避开了干扰信号区域，我们再来检测一次，看按键是否真得已经按下，若真得已经按下，这时肯定输出为低电平，若这时检测到的是高电平，证明刚才是由于干扰信号引起的误触发，CPU 就认为是误触发信号而舍弃这次的按键识别过程。从而提高了系统的可靠性。

由于要求每按下一次，命令被执行一次，直到下一次再按下的时候，再执行一次命令，因此从按键被识别出来之后，我们就可以执行这次的命令，所以要有一个等待按键释放的过程，显然释放的过程，就是使其恢复成高电平状态。

（1. 对于按键识别的指令，我们依然选择如下指令 JB　BIT，REL 指令是用来检测 BIT 是否为高电平，若 BIT＝1，则程序转向 REL 处执行程序，否则就继续向下执行程序。或者是　JNB　BIT，REL 指令是用来检测 BIT 是否为低电平，若 BIT＝0，则程序转向 REL 处执行程序，否则就继续向下执行程序。

（2． 但对程序设计过程中按键识别过程的框图如右图所示：

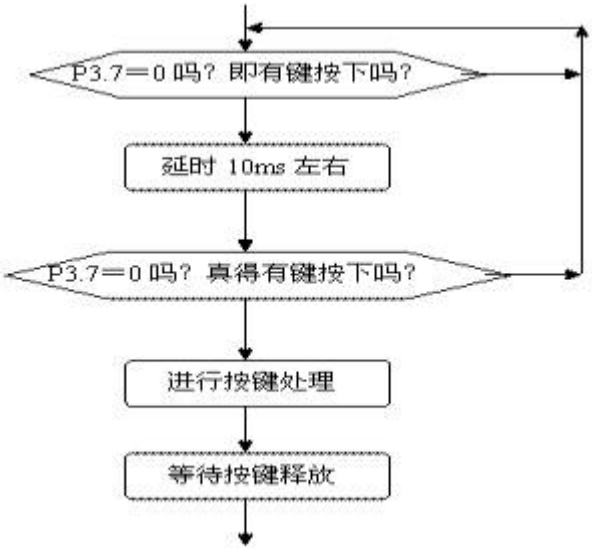


图 4.8.3

5． 程序框图

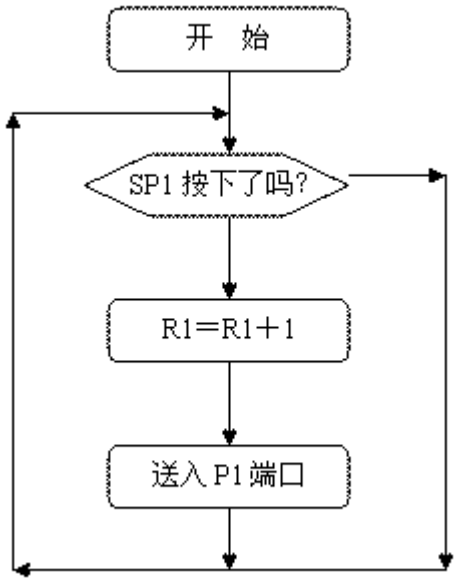


图 4.8.4

6． 汇编源程序

```
ORG 0
START: MOV R1,#00H ;初始化 R1 为 0，表示从 0 开始计数
MOV A,R1 ;
CPL A ;取反指令
MOV P1,A ;送出 P1 端口由发光二极管显示
REL: JNB P3.7,REL ;判断 SP1 是否按下
```

```
LCALL DELAY10MS ;若按下，则延时 10ms 左右
JNB P3.7,REL ;再判断 SP1 是否真得按下
INC R1 ;若真得按下，则进行按键处理，使
MOV A,R1 ;计数内容加 1，并送出 P1 端口由
CPL A ;发光二极管显示
MOV P1,A ;
JNB P3.7,$ ;等待 SP1 释放
SJMP REL ;继续对 K1 按键扫描
DELAY10MS: MOV R6,#20 ;延时 10ms 子程序
L1: MOV R7,#248
DJNZ R7,$
DJNZ R6,L1
RET
END
```

7. C 语言源程序

```
#include <AT89X51.H>

unsigned char count;

void delay10ms(void)
{
unsigned char i,j;
for(i=20;i>0;i--)
for(j=248;j>0;j--);
}

void main(void)
{
while(1)
{
if(P3_7==0)
{
delay10ms();
if(P3_7==0)
{
count++;
if(count==16)
{
count=0;
}
```

```
P1=~count;
while(P3_7==0);
}
}
}
}
```

9. 一键多功能按键识别技术

1. 实验任务

如图 4.9.1 所示，开关 SP1 接在 P3.7/RD 管脚上，在 AT89S51 单片机的 P1 端口接有四个发光二极管，上电的时候，L1 接在 P1.0 管脚上的发光二极管在闪烁，当每一次按下开关 SP1 的时候，L2 接在 P1.1 管脚上的发光二极管在闪烁，再按下开关 SP1 的时候，L3 接在 P1.2 管脚上的发光二极管在闪烁，再按下开关 SP1 的时候，L4 接在 P1.3 管脚上的发光二极管在闪烁，再按下开关 SP1 的时候，又轮到 L1 在闪烁了，如此轮流下去。

2. 电路原理图

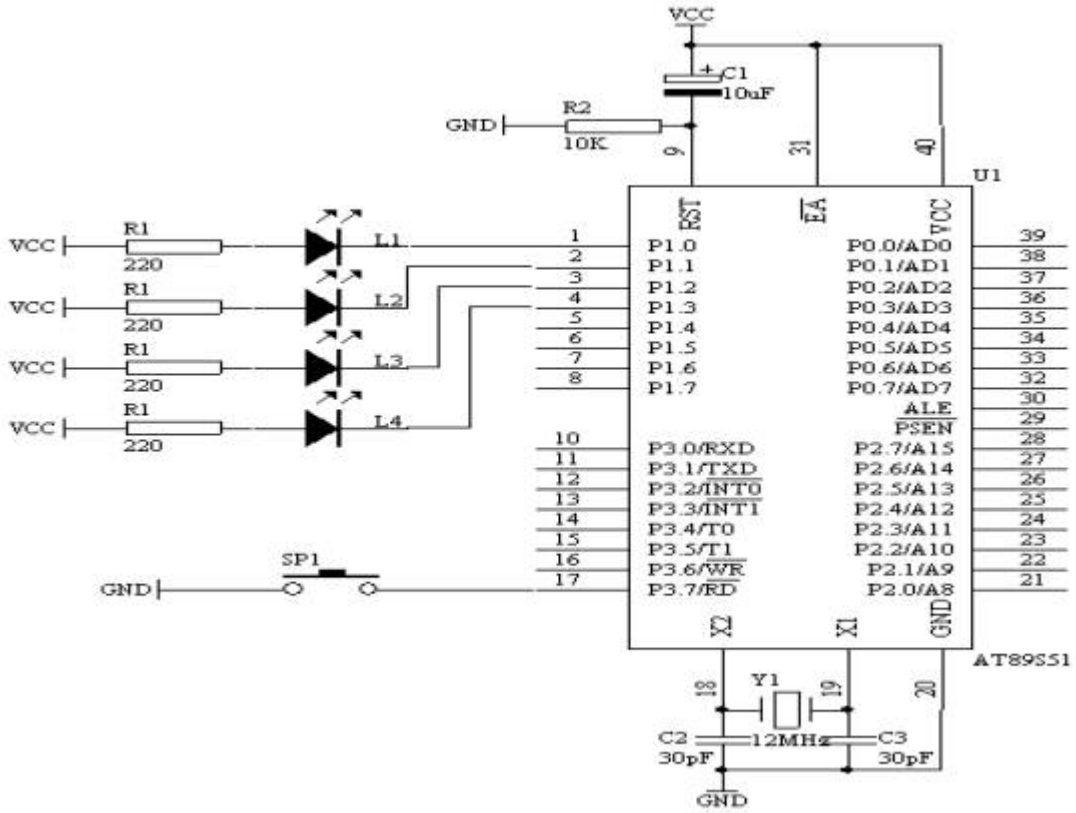


图 4.9.1

3．系统板上硬件连线

（1． 把“单片机系统”区域中的 P3.7/RD 端口连接到“独立式键盘”区域中的 SP1 端口上；

（2. 把“单片机系统”区域中的 P1.0－P1.4 端口用 8 芯排线连接到“八路发光二极管指示模块”区域中的“L1－L8”端口上；要求，P1.0 连接到 L1，P1.1 连接到 L2，P1.2 连接到 L3，P1.3 连接到 L4 上。

4．程序设计方法

（1． 设计思想由来

在我们生活中，我们很容易通过这个叫张三，那个叫李四，另外一个是王五；那是因为每个人有不同的名子，我们就很快认出，同样，对于要通过一个按键来识别每种不同的功能，我们给每个不同的功能模块用不同的 ID 号标识，这样，每按下一次按键，ID 的值是不相同的，所以单片机就很容易识别不同功能的身份了。

（2． 设计方法

从上面的要求我们可以看出，L1 到 L4 发光二极管在每个时刻的闪烁的时间是受开关 SP1 来控制，我们给 L1 到 L4 闪烁的时段定义出不同的 ID 号，当L1 在闪烁时，ID＝0；当 L2 在闪烁时，ID＝1；当 L3 在闪烁时，ID＝2；当 L4 在闪烁时，ID＝3；很显然，只要每次按下开关 K1 时，分别给出不同的 ID 号我们就能够完成上面的任务了。下面给出有关程序设计的框图。

5．程序框图

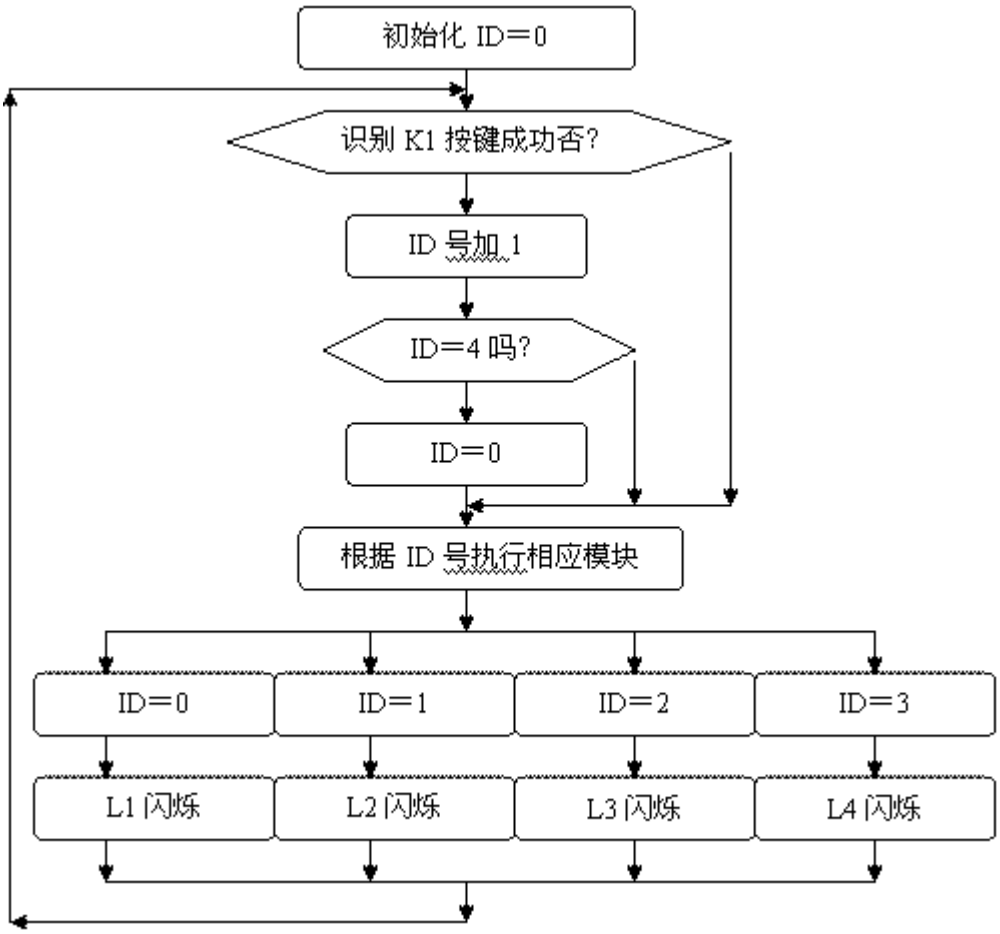


图 4.9.2

6. 汇编源程序

```
ID EQU 30H
SP1 BIT P3.7
L1 BIT P1.0
L2 BIT P1.1
L3 BIT P1.2
L4 BIT P1.3
ORG 0
MOV ID,#00H
START: JB K1,REL
LCALL DELAY10MS
JB K1,REL
INC ID
MOV A,ID
CJNE A,#04,REL
MOV ID,#00H
REL: JNB K1,$
MOV A,ID
```

```
CJNE A,#00H,IS0
CPL L1
LCALL DELAY
SJMP START
IS0: CJNE A,#01H,IS1
CPL L2
LCALL DELAY
SJMP START
IS1: CJNE A,#02H,IS2
CPL L3
LCALL DELAY
SJMP START
IS2: CJNE A,#03H,IS3
CPL L4
LCALL DELAY
SJMP START
IS3: LJMP START
DELAY10MS: MOV R6,#20
LOOP1: MOV R7,#248
DJNZ R7,$
DJNZ R6,LOOP1
RET
DELAY: MOV R5,#20
LOOP2: LCALL DELAY10MS
DJNZ R5,LOOP2
RET
END
```

7. C 语言源程序

```
#include <AT89X51.H>
unsigned char ID;
void delay10ms(void)
{
unsigned char i,j;
for(i=20;i>0;i--)
for(j=248;j>0;j--);
}

void delay02s(void)
{
unsigned char i;
for(i=20;i>0;i--)
{delay10ms();
}
}
```

```
void main(void)
{ while(1)
{ if(P3_7==0)
{delay10ms();
if(P3_7==0)
{
ID++;
if(ID==4)
{
ID=0;
}
while(P3_7==0);
}
}
switch(ID)
{ case 0:
P1_0=~P1_0;
delay02s();
break;
case 1:
P1_1=~P1_1;
delay02s();
break;
case 2:
P1_2=~P1_2;
delay02s();
break;
case 3:
P1_3=~P1_3;
delay02s();
break;
}
}
}
```

10. 00－99 计数器

1. 实验任务

利用 AT89S51 单片机来制作一个手动计数器，在 AT89S51 单片机的 P3.7 管脚接一个轻触开关，作为手动计数的按钮，用单片机的 P2.0－P2.7 接一个共阴数码管，作为 00－99 计数的个位数显示，用单片机的 P0.0－P0.7 接一个共阴数码管，作为 00－99 计数的十位数显示；硬件电路图如图 19 所示。

2. 电路原理图

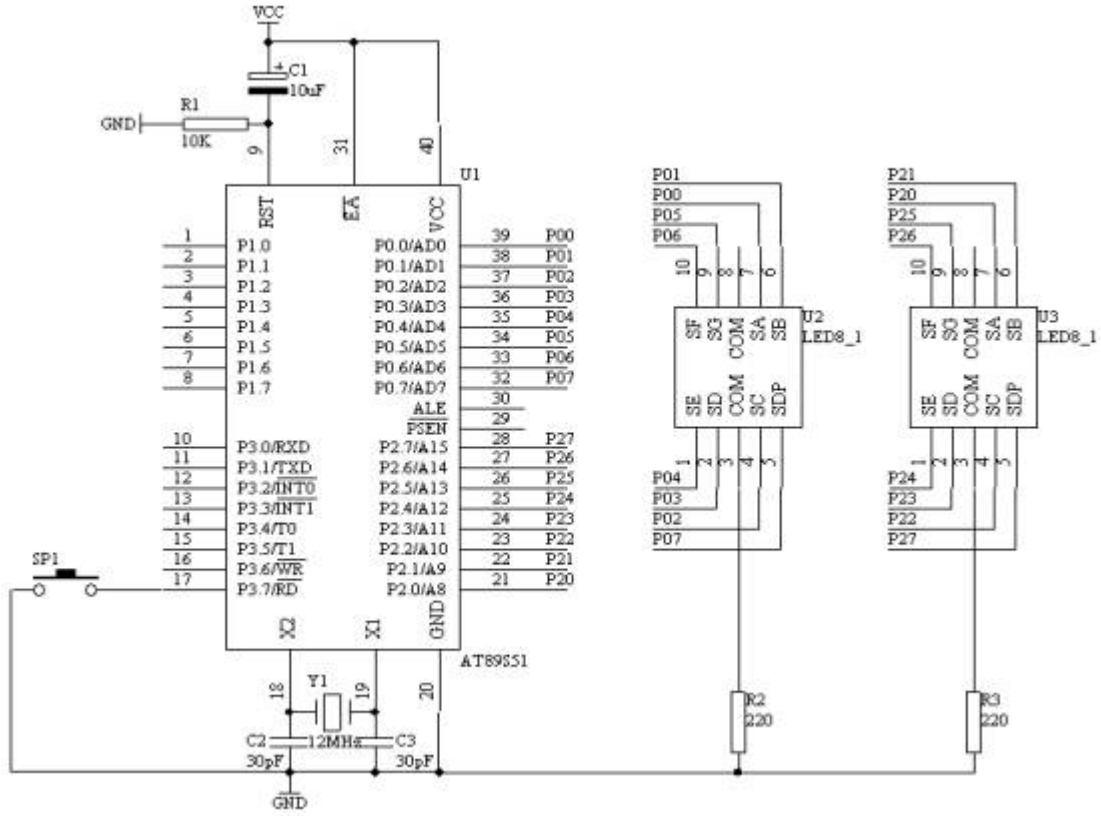


图 4.10.1

3. 系统板上硬件连线

（1. 把“单片机系统”区域中的 P0.0/AD0－P0.7/AD7 端口用 8 芯排线连接到“四路静态数码显示模块”区域中的任一个 a－h 端口上；要求：P0.0/AD0 对应着 a，P0.1/AD1 对应着 b，……，P0.7/AD7 对应着 h。

（2. 把“单片机系统”区域中的P2.0/A8－P2.7/A15 端口用8 芯排线连接到“四路静态数码显示模块”区域中的任一个数码管的 a－h 端口上；

（3. 把“单片机系统”区域中的 P3.7/RD 端口用导线连接到“独立式键盘”区域中的 SP1 端口上；

4. 程序设计内容

（1. 单片机对按键的识别的过程处理

（2. 单片机对正确识别的按键进行计数，计数满时，又从零开始计数；

（3. 单片机对计的数值要进行数码显示，计得的数是十进数，含有十位和个位，我们要把十位和个位拆开分别送出这样的十位和个位数值到对应的数码管上显示。如何拆开十位和个位我们可以把所计得的数值对 10 求余，即可得个位数字，对 10 整除，即可得到十位数字了。

（4. 通过查表方式，分别显示出个位和十位数字。

5. 程序框图

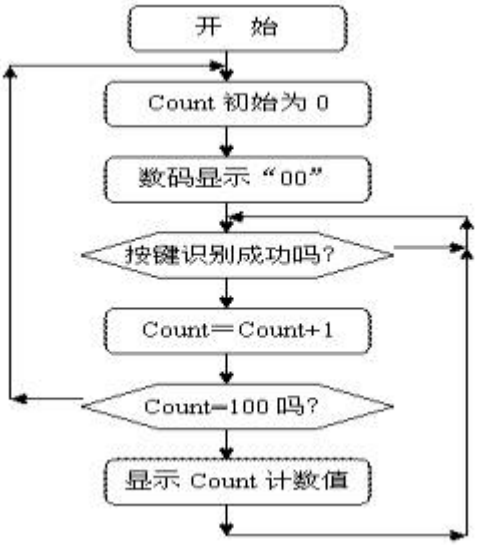


图 4.10.2

6. 汇编源程序

```
Count EQU 30H
SP1 BIT P3.7
ORG 0
START: MOV Count,#00H
NEXT: MOV A,Count
MOV B,#10
DIV AB
MOV DPTR,#TABLE
MOVC A,@A+DPTR
MOV P0,A
MOV A,B
MOVC A,@A+DPTR
MOV P2,A
WT: JNB SP1,WT
WAIT: JB SP1,WAIT
LCALL DELY10MS
JB SP1,WAIT
INC Count
MOV A,Count
CJNE A,#100,NEXT
```

```
LJMP START
DELY10MS: MOV R6,#20
D1: MOV R7,#248
DJNZ R7,$
DJNZ R6,D1
RET
TABLE: DB 3FH,06H,5BH,4FH,66H,6DH,7DH,07H,7FH,6FH
END
```

7. C 语言源程序

```
#include <AT89X51.H>
unsigned char code table[]={0x3f,0x06,0x5b,0x4f,0x66,
0x6d,0x7d,0x07,0x7f,0x6f};
unsigned char Count;

void delay10ms(void)
{
unsigned char i,j;
for(i=20;i>0;i--)
for(j=248;j>0;j--);
}

void main(void)
{
Count=0;
P0=table[Count/10];
P2=table[Count%10];
while(1)
{
if(P3_7==0)
{
delay10ms();
if(P3_7==0)
{
Count++;
if(Count==100)
{
Count=0;
}
P0=table[Count/10];
P2=table[Count%10];
while(P3_7==0);
}
}
}
```

}

11. 00－59 秒计时器（利用软件延时）

1. 实验任务

如下图所示，在 AT89S51 单片机的 P0 和 P2 端口分别接有两个共阴数码管，P0 口驱动显示秒时间的十位，而 P2 口驱动显示秒时间的个位。

2. 电路原理图

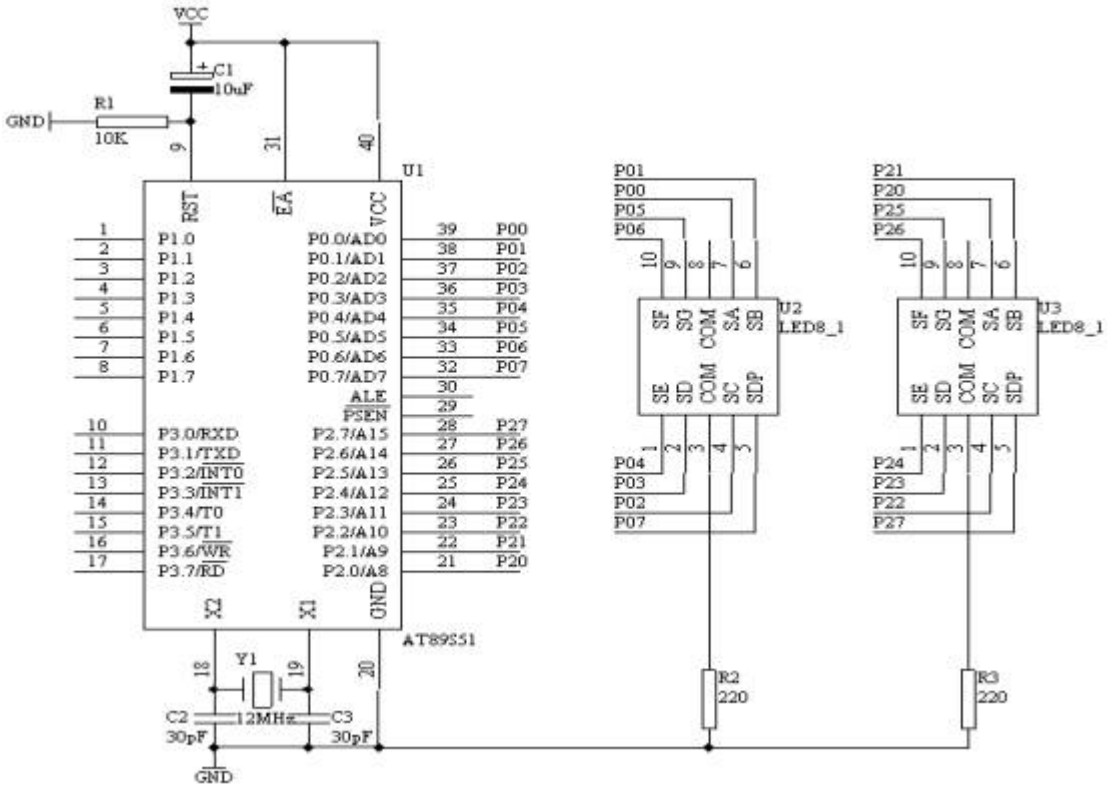


图 4.11.1

3. 系统板上硬件连线

（1. 把“单片机系统”区域中的 P0.0/AD0－P0.7/AD7 端口用 8 芯排线连接到“四路静态数码显示模块”区域中的任一个 a－h 端口上；要求：P0.0/AD0 对应着 a，P0.1/AD1 对应着 b，……，P0.7/AD7 对应着 h。

（2. 把“单片机系统”区域中的P2.0/A8－P2.7/A15 端口用8 芯排线连接到“四路静态数码显示模块”区域中的任一个 a－h 端口上；要求：P2.0/A8 对应着 a，P2.1/A9 对应着 b，……，P2.7/A15 对应着 h。

4. 程序设计内容

（1. 在设计过程中我们用一个存储单元作为秒计数单元，当一秒钟到来时，就让秒计数单元加 1，当秒计数达到 60 时，就自动返回到 0，重新秒计数。

（2. 对于秒计数单元中的数据要把它十位数和个数分开，方法仍采用对 10 整除和对 10 求余。

（3. 在数码上显示，仍通过查表的方式完成。

（4. 一秒时间的产生在这里我们采用软件精确延时的方法来完成，经过精确计算得到 1 秒时间为 1.002 秒。

```
DELY1S: MOV R5,#100

D2: MOV R6,#20

D1: MOV R7,#248

DJNZ R7,$

DJNZ R6,D1

DJNZ R5,D2

RET
```

5. 程序框图

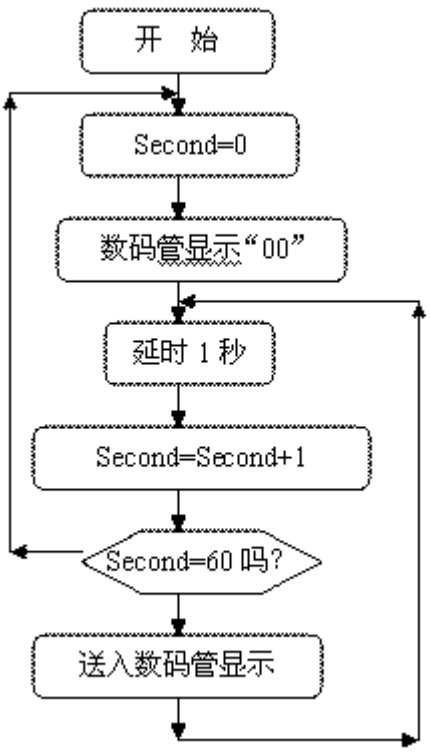


图 4.11.2

6. 汇编源程序

```
Second EQU 30H
ORG 0
START: MOV Second,#00H
NEXT: MOV A,Second
MOV B,#10
DIV AB
MOV DPTR,#TABLE
MOVC A,@A+DPTR
MOV P0,A
MOV A,B
MOVC A,@A+DPTR
MOV P2,A
LCALL DELY1S
INC Second
MOV A,Second
CJNE A,#60,NEXT
LJMP START
DELY1S: MOV R5,#100
D2: MOV R6,#20
D1: MOV R7,#248
DJNZ R7,$
DJNZ R6,D1
DJNZ R5,D2
RET
TABLE: DB 3FH,06H,5BH,4FH,66H,6DH,7DH,07H,7FH,6FH
END
```

7. C语言源程序

```
#include <AT89X51.H>
unsigned char code table[]={0x3f,0x06,0x5b,0x4f,0x66,
0x6d,0x7d,0x07,0x7f,0x6f};
unsigned char Second;

void delay1s(void)
{
unsigned char i,j,k;
for(k=100;k>0;k--)
for(i=20;i>0;i--)
for(j=248;j>0;j--);
}


void main(void)
{
```

```
Second=0;
P0=table[Second/10];
P2=table[Second%10];
while(1)
{
delay1s();
Second++;
if(Second==60)
{
Second=0;
}
P0=table[Second/10];
P2=table[Second%10];
}
}
```

12. 可预置可逆 4 位计数器

1. 实验任务

利用 AT89S51 单片机的 P1.0－P1.3 接四个发光二极管 L1－L4，用来指示当前计数的数据；用 P1.4－P1.7 作为预置数据的输入端，接四个拨动开关 K1－K4，用 P3.6/WR 和 P3.7/RD 端口接两个轻触开关，用来作加计数和减计数开关。具体的电路原理图如下图所示

2. 电路原理图

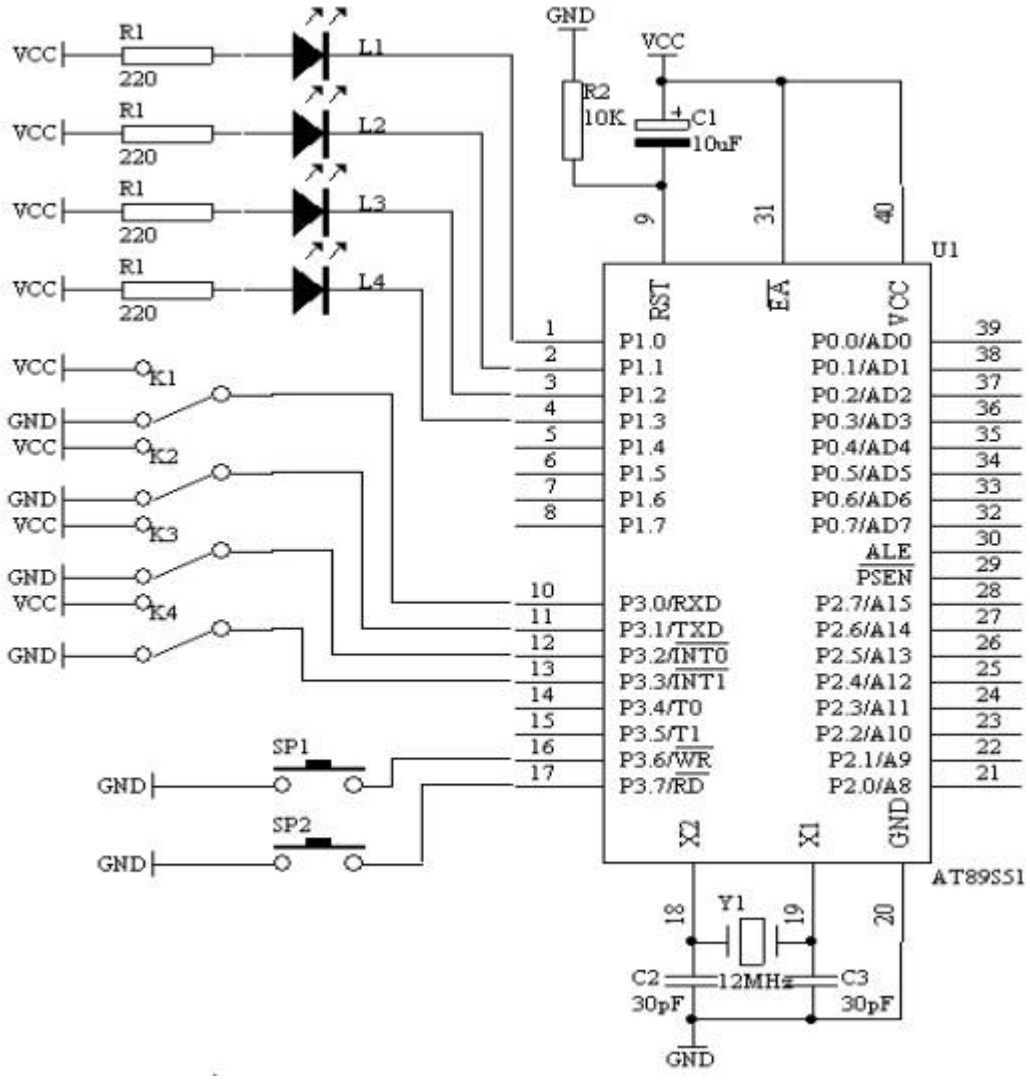


图 4.12.1

3. 系统板上硬件连线

（1. 把“单片机系统”区域中的 P1.0－P1.3 端口用 8 芯排线连接到“八路发光二极管指示模块”区域中的 L1－L4 上；要求：P1.0 对应着 L1，P1.1 对应着 L2，P1.2 对应着 L3，P1.3 对应着 L4；

（2. 把“单片机系统”区域中的 P3.0/RXD，P3.1/TXD，P3.2/INT0，P3.3/INT1 用导线连接到“四路拨动开关”区域中的 K1－K4 上；

（3. 把“单片机系统”区域中的 P3.6/WR，P3.7/RD 用导线连接到“独立式键盘”区域中的 SP1 和 SP2 上；

4. 程序设计内容

（1. 两个独立式按键识别的处理过程；

（2． 预置初值读取的问题

（3． LED 输出指示

5． 程序框图

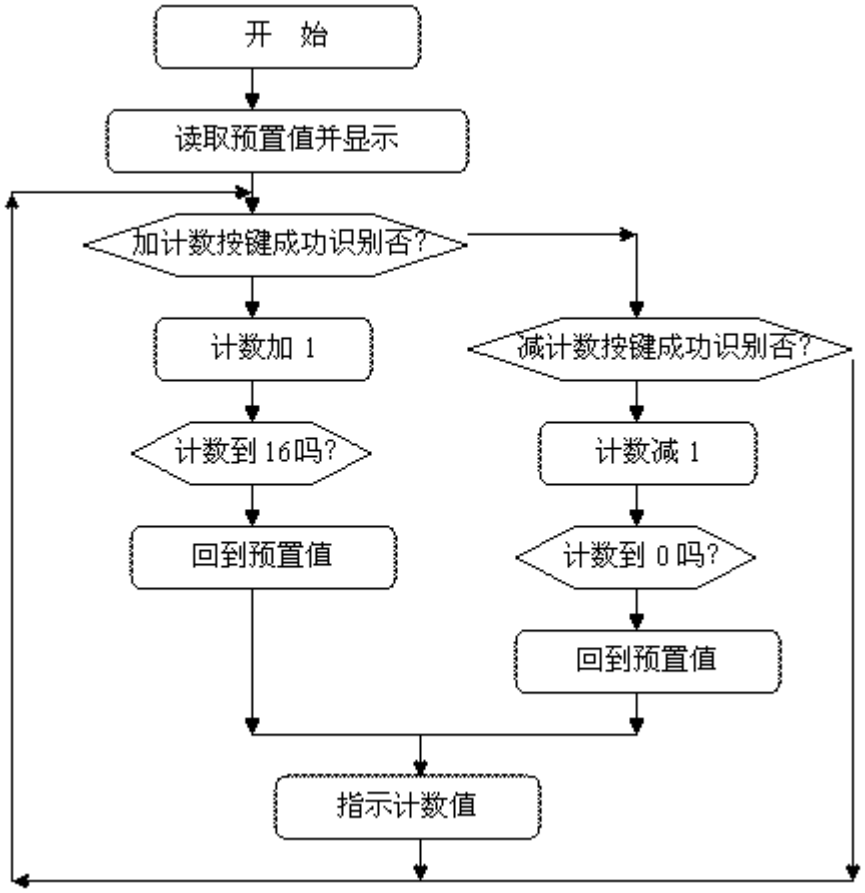


图 4.12.2

6． 汇编源程序

```
COUNT EQU 30H
ORG 00H
START: MOV A,P3
ANL A,#0FH
MOV COUNT,A
MOV P1,A
SK2: JB P3.6,SK1
LCALL DELY10MS
JB P3.6,SK1
INC COUNT
MOV A,COUNT
CJNE A,#16,NEXT
MOV A,P3
```

```
ANL A,#0FH
MOV COUNT,A
NEXT: MOV P1,A
WAIT: JNB P3.6,WAIT
LJMP SK2
SK1: JB P3.7,SK2
LCALL DELY10MS
JB P3.7,SK2
DEC COUNT
MOV A,COUNT
CJNE A,#0FFH,NEX
MOV A,P3
ANL A,#0FH
MOV COUNT,A
NEX: MOV P1,A
WAIT2: JNB P3.7,WAIT2
LJMP SK2
DELY10MS: MOV R6,#20
MOV R7,#248
D1: DJNZ R7,$
DJNZ R6,D1
RET
END
```

7. C 语言源程序

```
#include <AT89X51.H>

unsigned char curcount;

void delay10ms(void)
{
unsigned char i,j;
for(i=20;i>0;i--)
for(j=248;j>0;j--);
}


void main(void)
{
curcount=P3 & 0x0f;
P1=~curcount;
while(1)
{
if(P3_6==0)
{
```

```
delay10ms();
if(P3_6==0)
{
if(curcount>=15)
{
curcount=15;
}
else
{
curcount++;
}
P1=~curcount;
while(P3_6==0);
}
}
if(P3_7==0)
{
delay10ms();
if(P3_7==0)
{
if(curcount<=0)
{
curcount=0;
}
else
{
curcount--;
}
P1=~curcount;
while(P3_7==0);
}
}
}
}
```

13. 动态数码显示技术

1. 实验任务

如图 4.13.1 所示，P0 端口接动态数码管的字形码笔段，P2 端口接动态数码管的数位选择端，P1.7 接一个开关，当开关接高电平时，显示“12345”字样；当开关接低电平时，显示“HELLO”字样。

2. 电路原理图

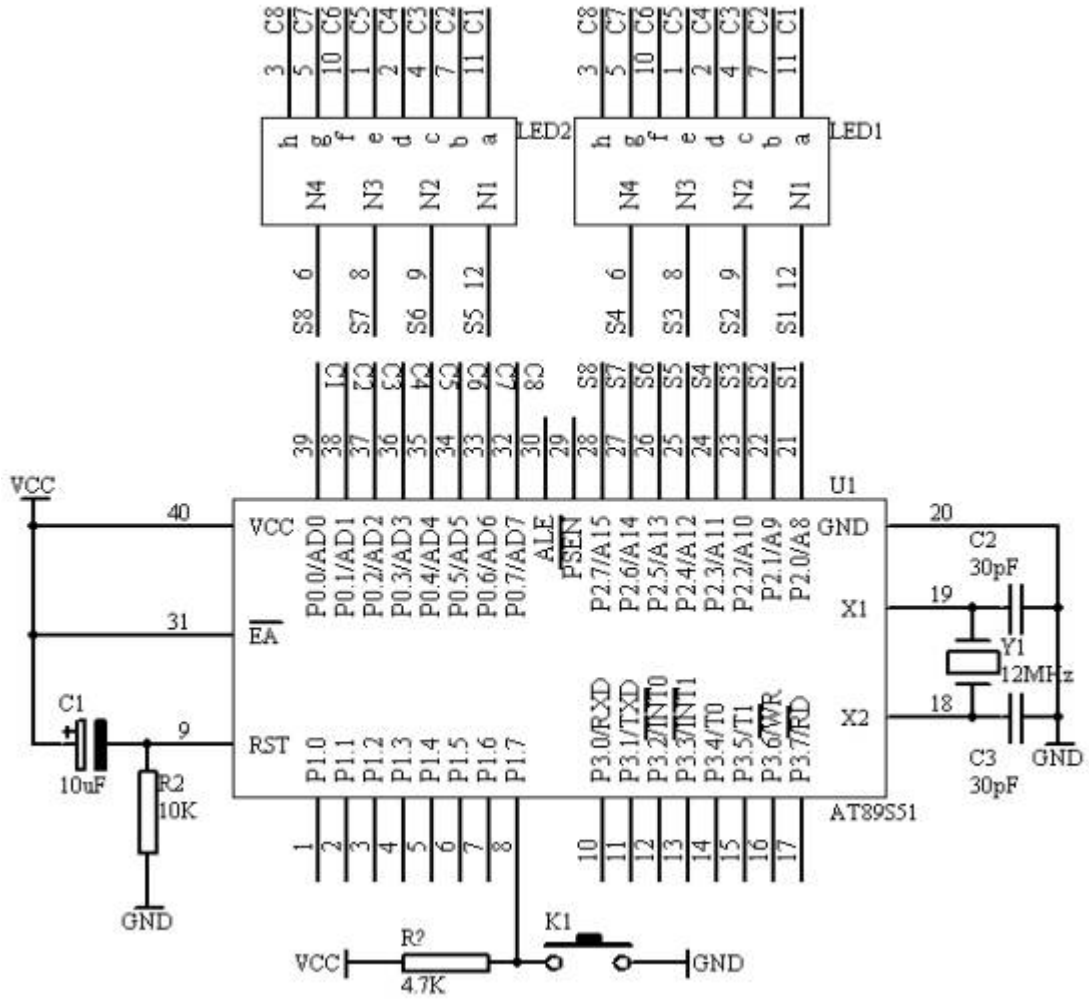


图 4.13.1

3. 系统板上硬件连线

（1. 把“单片机系统”区域中的 P0.0/AD0－P0.7/AD7 用 8 芯排线连接到“动态数码显示”区域中的 a－h 端口上；

（2. 把“单片机系统”区域中的 P2.0/A8－P2.7/A15 用 8 芯排线连接到“动态数码显示”区域中的 S1－S8 端口上；

（3. 把“单片机系统”区域中的 P1.7 端口用导线连接到“独立式键盘”区域中的 SP1 端口上；

4. 程序设计内容

（1. 动态扫描方法

动态接口采用各数码管循环轮流显示的方法，当循环显示频率较高时，利用人眼的暂留特性，看不出闪烁显示现象，这种显示需要一个接口完成字形码的输出（字形选择），另一接口完成各数码管的轮流点亮（数位选择）。

（2. 在进行数码显示的时候，要对显示单元开辟 8 个显示缓冲区，每个显示缓冲区装有显示的不同数据即可。

（3. 对于显示的字形码数据我们采用查表方法来完成。

5. 程序框图

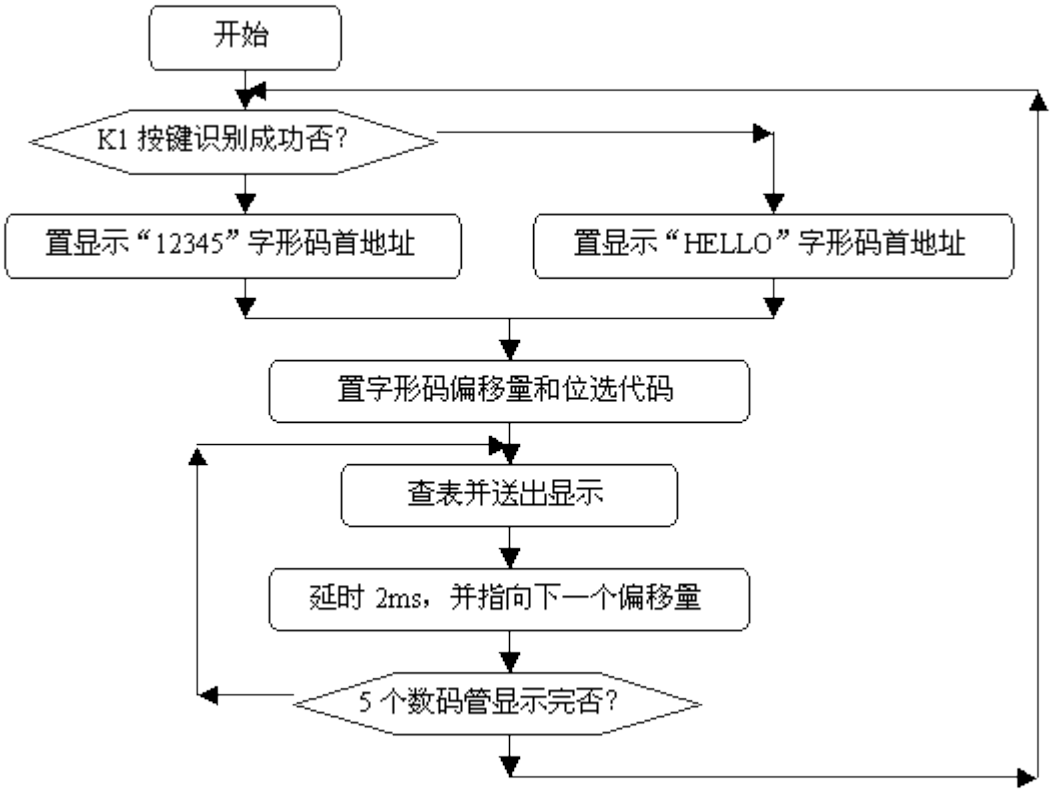


图 4.13.2

6. 汇编源程序

```
ORG 00H
START: JB P1.7,DIR1
MOV DPTR,#TABLE1
SJMP DIR
DIR1: MOV DPTR,#TABLE2
DIR: MOV R0,#00H
MOV R1,#01H
NEXT: MOV A,R0
MOVC A,@A+DPTR
```

```
MOV P0,A
MOV A,R1
MOV P2,A
LCALL DAY
INC R0
RL A
MOV R1,A
CJNE R1,#0DFH,NEXT
SJMP START
DAY: MOV R6,#4
D1: MOV R7,#248
DJNZ R7,$
DJNZ R6,D1
RET
TABLE1: DB 06H,5BH,4FH,66H,6DH
TABLE2: DB 78H,79H,38H,38H,3FH
END
```

7. C 语言源程序

```
#include <AT89X51.H>

unsigned char code table1[]={0x06,0x5b,0x4f,0x66,0x6d};
unsigned char code table2[]={0x78,0x79,0x38,0x38,0x3f};
unsigned char i;
unsigned char a,b;
unsigned char temp;

void main(void)
{
while(1)
{
temp=0xfe;
for(i=0;i<5;i++)
{
if(P1_7==1)
{
P0=table1[i];
}
else
{
P0=table2[i];
}
P2=temp;
a=temp<<(i+1);
b=temp>>(7-i);
```

```
temp=a|b;
          for(a=4;a>0;a--)
for(b=248;b>0;b--);
}
}
```

14. 4×4 矩阵式键盘识别技术

1. 实验任务

如图 4.14.2 所示，用AT89S51 的并行口 P1 接 4×4 矩阵键盘，以P1.0—P1.3 作输入线，以P1.4—P1.7 作输出线；在数码管上显示每个按键的“0—F”序号。对应的按键的序号排列如图 4.14.1 所示

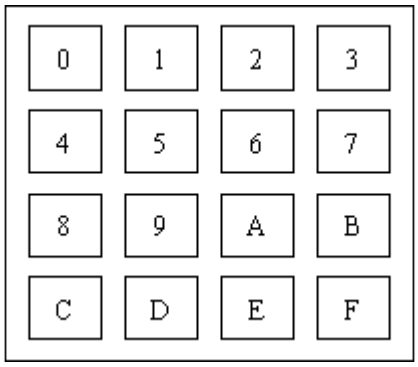


图 4.14.1

2. 硬件电路原理图

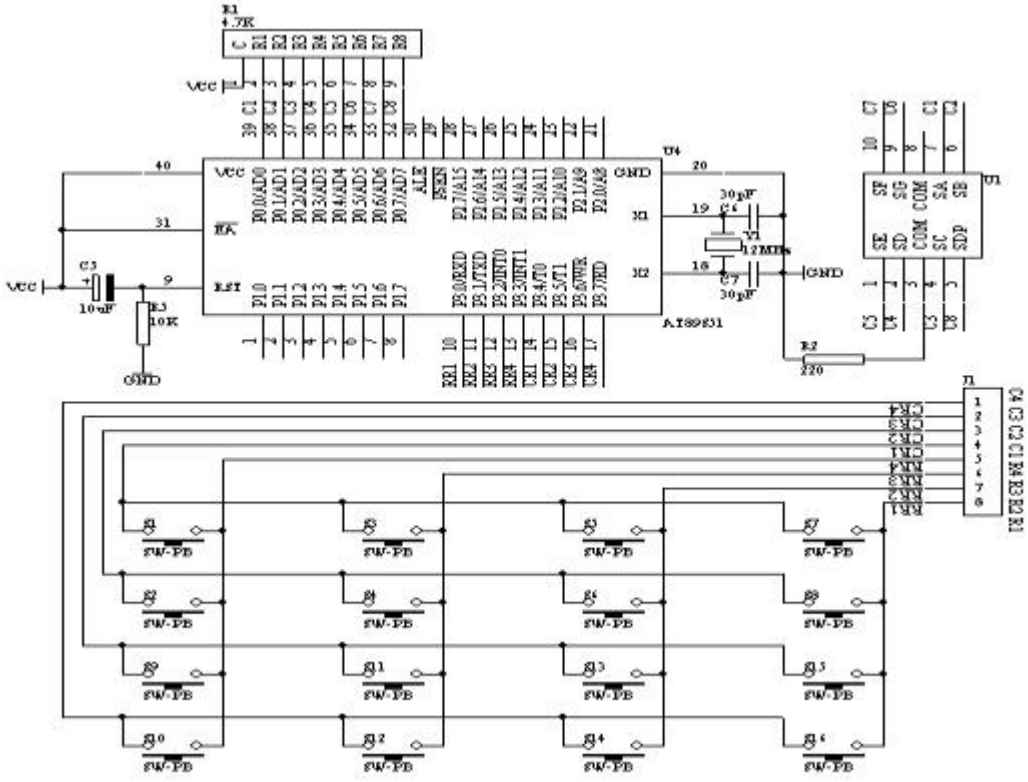


图 4.14.2

3. 系统板上硬件连线

（1. 把“单片机系统“区域中的 P3.0－P3.7 端口用 8 芯排线连接到“4X4 行列式键盘”区域中的 C1－C4　R1－R4 端口上；

（2. 把“单片机系统”区域中的 P0.0/AD0－P0.7/AD7 端口用 8 芯排线连接到“四路静态数码显示模块”区域中的任一个 a－h 端口上；要求：P0.0/AD0 对应着 a，P0.1/AD1 对应着 b，……，P0.7/AD7 对应着 h。

4. 程序设计内容

（1. 4×4 矩阵键盘识别处理

（2. 每个按键有它的行值和列值　，行值和列值的组合就是识别这个按键的编码。矩阵的行线和列线分别通过两并行接口和 CPU 通信。每个按键的状态同样需变成数字量“0”和“1”，开关的一端（列线）通过电阻接 VCC，而接地是通过程序输出数字“0”实现的。键盘处理程序的任务是：确定有无键按下，判断哪一个键按下，键的功能是什么；还要消除按键在闭合或断开时的抖动。两个并行口中，一个输出扫描码，使按键逐行动态接地，另一个并行口输入按键状态，由行扫描值和回馈信号共同形成键编码而识别按键，通过软件查表，查出该键的功能。

5. 程序框图

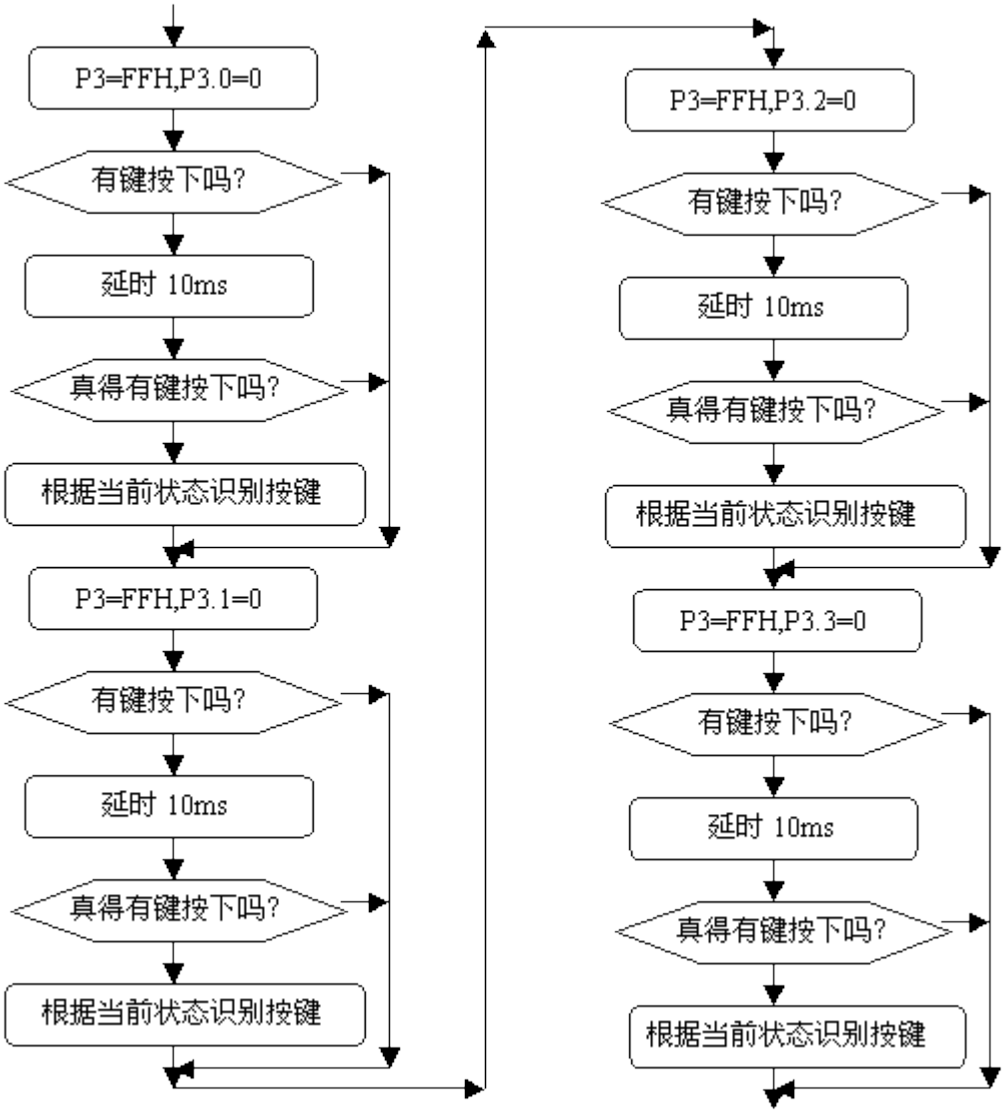


图 4.14.3

6. 汇编源程序

```
KEYBUF EQU 30H
ORG 00H
START: MOV KEYBUF,#2
WAIT:
MOV P3,#0FFH
CLR P3.4
MOV A,P3
ANL A,#0FH
XRL A,#0FH
JZ NOKEY1
LCALL DELY10MS
```

```
MOV A,P3
ANL A,#0FH
XRL A,#0FH
JZ NOKEY1
MOV A,P3
ANL A,#0FH
CJNE A,#0EH,NK1
MOV KEYBUF,#0
LJMP DK1
NK1: CJNE A,#0DH,NK2
MOV KEYBUF,#1
LJMP DK1
NK2: CJNE A,#0BH,NK3
MOV KEYBUF,#2
LJMP DK1
NK3: CJNE A,#07H,NK4
MOV KEYBUF,#3
LJMP DK1
NK4: NOP
DK1:
MOV A,KEYBUF
MOV DPTR,#TABLE
MOVC A,@A+DPTR
MOV P0,A

DK1A: MOV A,P3
ANL A,#0FH
XRL A,#0FH
JNZ DK1A
NOKEY1:
MOV P3,#0FFH
CLR P3.5
MOV A,P3
ANL A,#0FH
XRL A,#0FH
JZ NOKEY2
LCALL DELY10MS
MOV A,P3
ANL A,#0FH
XRL A,#0FH
JZ NOKEY2
MOV A,P3
ANL A,#0FH
CJNE A,#0EH,NK5
```

```
MOV KEYBUF,#4
LJMP DK2
NK5: CJNE A,#0DH,NK6
MOV KEYBUF,#5
LJMP DK2
NK6: CJNE A,#0BH,NK7
MOV KEYBUF,#6
LJMP DK2
NK7: CJNE A,#07H,NK8
MOV KEYBUF,#7
LJMP DK2
NK8: NOP
DK2:
MOV A,KEYBUF
MOV DPTR,#TABLE
MOVC A,@A+DPTR
MOV P0,A

DK2A: MOV A,P3
ANL A,#0FH
XRL A,#0FH
JNZ DK2A
NOKEY2:
MOV P3,#0FFH
CLR P3.6
MOV A,P3
ANL A,#0FH
XRL A,#0FH
JZ NOKEY3
LCALL DELY10MS
MOV A,P3
ANL A,#0FH
XRL A,#0FH
JZ NOKEY3
MOV A,P3
ANL A,#0FH
CJNE A,#0EH,NK9
MOV KEYBUF,#8
LJMP DK3
NK9: CJNE A,#0DH,NK10
MOV KEYBUF,#9
LJMP DK3
NK10: CJNE A,#0BH,NK11
MOV KEYBUF,#10
```

```
LJMP DK3
NK11: CJNE A,#07H,NK12
MOV KEYBUF,#11
LJMP DK3
NK12: NOP
DK3:
MOV A,KEYBUF
MOV DPTR,#TABLE
MOVC A,@A+DPTR
MOV P0,A

DK3A: MOV A,P3
ANL A,#0FH
XRL A,#0FH
JNZ DK3A
NOKEY3:
MOV P3,#0FFH
CLR P3.7
MOV A,P3
ANL A,#0FH
XRL A,#0FH
JZ NOKEY4
LCALL DELY10MS
MOV A,P3
ANL A,#0FH
XRL A,#0FH
JZ NOKEY4
MOV A,P3
ANL A,#0FH
CJNE A,#0EH,NK13
MOV KEYBUF,#12
LJMP DK4
NK13: CJNE A,#0DH,NK14
MOV KEYBUF,#13
LJMP DK4
NK14: CJNE A,#0BH,NK15
MOV KEYBUF,#14
LJMP DK4
NK15: CJNE A,#07H,NK16
MOV KEYBUF,#15
LJMP DK4
NK16: NOP
DK4:
MOV A,KEYBUF
```

```
MOV DPTR,#TABLE
MOVC A,@A+DPTR
MOV P0,A

DK4A: MOV A,P3
ANL A,#0FH
XRL A,#0FH
JNZ DK4A
NOKEY4:
LJMP WAIT
DELY10MS:
MOV R6,#10
D1: MOV R7,#248
DJNZ R7,$
DJNZ R6,D1
RET
TABLE: DB 3FH,06H,5BH,4FH,66H,6DH,7DH,07H
DB 7FH,6FH,77H,7CH,39H,5EH,79H,71H
END
```

7. C 语言源程序

```
#include <AT89X51.H>
unsigned char code table[]={0x3f,0x06,0x5b,0x4f,
0x66,0x6d,0x7d,0x07,
0x7f,0x6f,0x77,0x7c,
0x39,0x5e,0x79,0x71};
unsigned char temp;
unsigned char key;
unsigned char i,j;


void main(void)
{
while(1)
{
P3=0xff;
P3_4=0;
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
for(i=50;i>0;i--)
for(j=200;j>0;j--);
temp=P3;
temp=temp & 0x0f;
```

```
if (temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
switch(temp)
{
case 0x0e:
key=7;
break;
case 0x0d:
key=8;
break;
case 0x0b:
key=9;
break;
case 0x07:
key=10;
break;
}
temp=P3;
P1_0=~P1_0;
P0=table[key];
temp=temp & 0x0f;
while(temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
}
}
}

P3=0xff;
P3_5=0;
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
for(i=50;i>0;i--)
for(j=200;j>0;j--);
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
temp=P3;
```

```
temp=temp & 0x0f;
switch(temp)
{
case 0x0e:
key=4;
break;
case 0x0d:
key=5;
break;
case 0x0b:
key=6;
break;
case 0x07:
key=11;
break;
}
temp=P3;
P1_0=~P1_0;
P0=table[key];
temp=temp & 0x0f;
while(temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
}
}
}

P3=0xff;
P3_6=0;
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
for(i=50;i>0;i--)
for(j=200;j>0;j--);
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
switch(temp)
{
```

```
case 0x0e:
key=1;
break;
case 0x0d:
key=2;
break;
case 0x0b:
key=3;
break;
case 0x07:
key=12;
break;
}
temp=P3;
P1_0=~P1_0;
P0=table[key];
temp=temp & 0x0f;
while(temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
}
}
}

P3=0xff;
P3_7=0;
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
for(i=50;i>0;i--)
for(j=200;j>0;j--);
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
switch(temp)
{
case 0x0e:
key=0;
break;
```

```
case 0x0d:
key=13;
break;
case 0x0b:
key=14;
break;
case 0x07:
key=15;
break;
}
temp=P3;
P1_0=~P1_0;
P0=table[key];
temp=temp & 0x0f;
while(temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
}
}
}
}
}
```

15. 定时计数器 T0 作定时应用技术（一）

1. 实验任务

用 AT89S51 单片机的定时/计数器 T0 产生一秒的定时时间，作为秒计数时间，当一秒产生时，秒计数加 1，秒计数到 60 时，自动从 0 开始。硬件电路如下图所示

2. 电路原理图

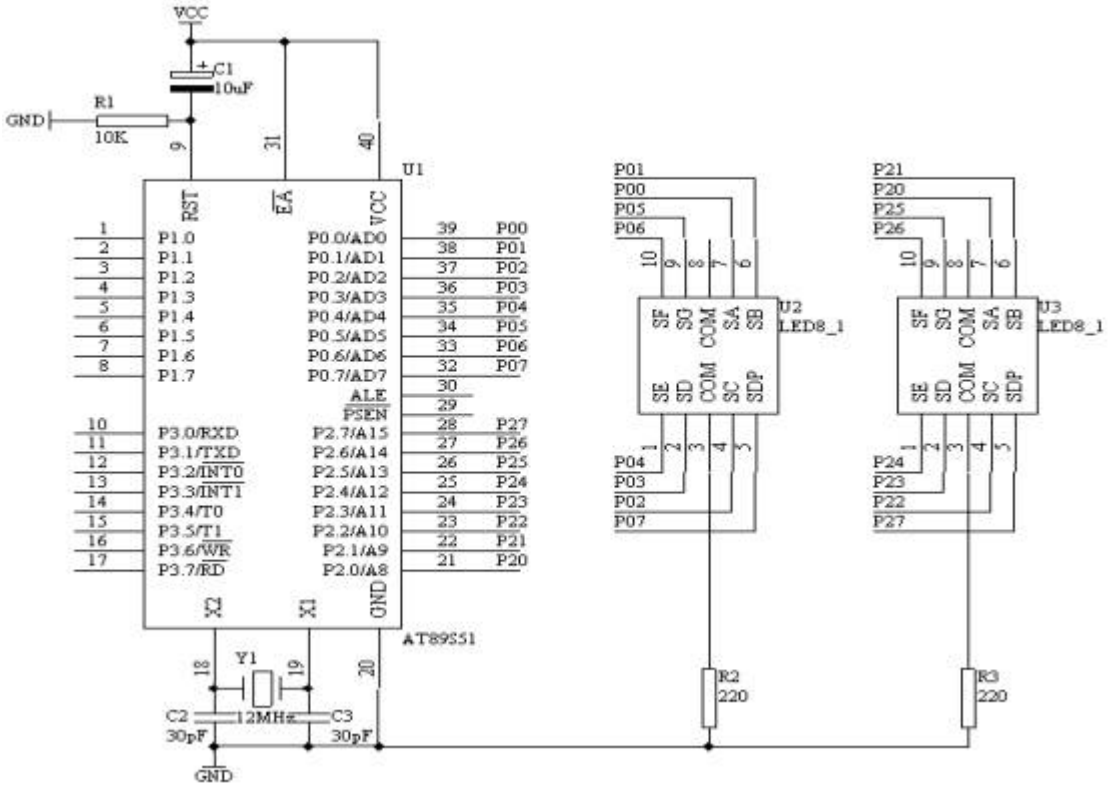


图 4.15.1

3. 系统板上硬件连线

（1. 把“单片机系统”区域中的 P0.0/AD0－P0.7/AD7 端口用 8 芯排线连接到“四路静态数码显示模块”区域中的任一个 a－h 端口上；要求：P0.0/AD0 对应着 a，P0.1/AD1 对应着 b，……，P0.7/AD7 对应着 h。

（2. 把“单片机系统”区域中的P2.0/A8－P2.7/A15 端口用8 芯排线连接到“四路静态数码显示模块”区域中的任一个 a－h 端口上；要求：P2.0/A8 对应着 a，P2.1/A9 对应着 b，……，P2.7/A15 对应着 h。

4. 程序设计内容

AT89S51 单片机的内部 16 位定时/计数器是一个可编程定时/计数器，它既可以工作在 13 位定时方式，也可以工作在 16 位定时方式和 8 位定时方式。只要通过设置特殊功能寄存器 TMOD，即可完成。定时/计数器何时工作也是通过软件来设定 TCON 特殊功能寄存器来完成的。

现在我们选择 16 位定时工作方式，对于 T0 来说，最大定时也只有 65536us，即 65.536ms，无法达到我们所需要的 1 秒的定时，因此，我们必须通过软件来处理这个问题，假设我们取 T0 的最大定时为 50ms，即要定时 1 秒需要经过 20 次的 50ms 的定时。对于这 20 次我们就可以采用软件的方法来统计了。

因此，我们设定 TMOD＝00000001B，即 TMOD＝01H

下面我们要给 T0 定时/计数器的 TH0，TL0 装入预置初值，通过下面的公式可以计算出

TH0＝（216－50000）　/　256

TL0＝（216－50000）　MOD　256

当 T0 在工作的时候，我们如何得知 50ms 的定时时间已到，这回我们通过检测 TCON 特殊功能寄存器中的 TF0 标志位，如果 TF0＝1 表示定时时间已到。

5.　程序框图

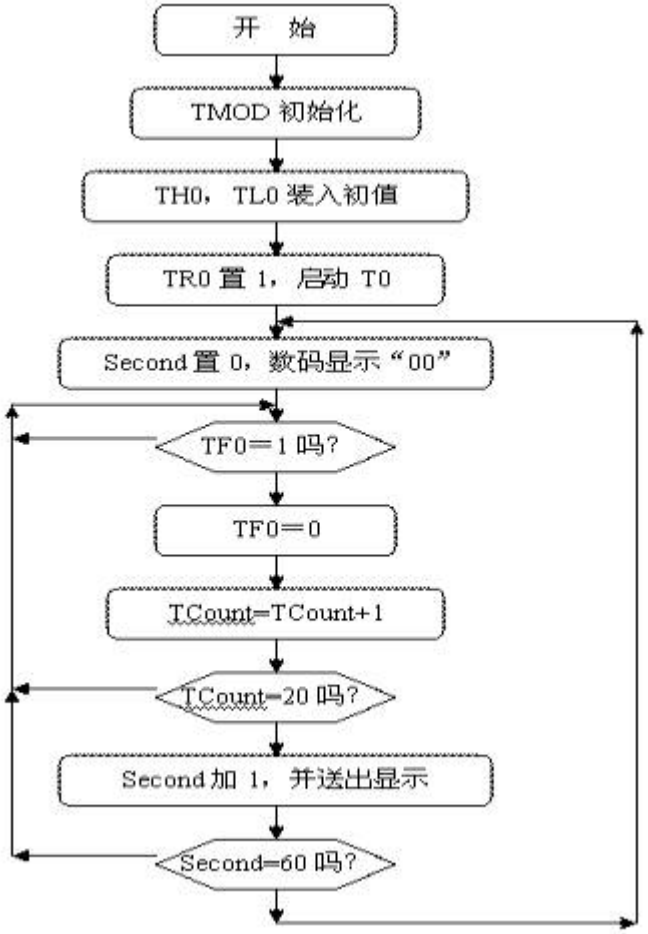


图 4.15.2

6.　汇编源程序（查询法）

```
SECOND EQU 30H
TCOUNT EQU 31H
ORG 00H
START: MOV SECOND,#00H
MOV TCOUNT,#00H
```

```
MOV TMOD,#01H
MOV TH0,#(65536-50000) / 256
MOV TL0,#(65536-50000) MOD 256
SETB TR0
DISP: MOV A,SECOND
MOV B,#10
DIV AB
MOV DPTR,#TABLE
MOVC A,@A+DPTR
MOV P0,A
MOV A,B
MOVC A,@A+DPTR
MOV P2,A
WAIT: JNB TF0,WAIT
CLR TF0
MOV TH0,#(65536-50000) / 256
MOV TL0,#(65536-50000) MOD 256
INC TCOUNT
MOV A,TCOUNT
CJNE A,#20,NEXT
MOV TCOUNT,#00H
INC SECOND
MOV A,SECOND
CJNE A,#60,NEX
MOV SECOND,#00H
NEX: LJMP DISP
NEXT: LJMP WAIT
TABLE: DB 3FH,06H,5BH,4FH,66H,6DH,7DH,07H,7FH,6FH
END
```

7. C 语言源程序（查询法）

```
#include <AT89X51.H>

unsigned char code dispcode[]={0x3f,0x06,0x5b,0x4f,
0x66,0x6d,0x7d,0x07,
0x7f,0x6f,0x77,0x7c,
0x39,0x5e,0x79,0x71,0x00};
unsigned char second;
unsigned char tcount;

void main(void)
{
TMOD=0x01;
TH0=(65536-50000)/256;
TL0=(65536-50000)%256;
```

```
TR0=1;
tcount=0;
second=0;
P0=dispcode[second/10];
P2=dispcode[second%10];
while(1)
{
if(TF0==1)
{
tcount++;
if(tcount==20)
{
tcount=0;
second++;
if(second==60)
{
second=0;
}
P0=dispcode[second/10];
P2=dispcode[second%10];
}
TF0=0;
TH0=(65536-50000)/256;
TL0=(65536-50000)%256;
}
}
}
```

1. 汇编源程序（中断法）

```
SECOND EQU 30H
TCOUNT EQU 31H
ORG 00H
LJMP START
ORG 0BH
LJMP INT0X
START: MOV SECOND,#00H
MOV A,SECOND
MOV B,#10
DIV AB
MOV DPTR,#TABLE
MOVC A,@A+DPTR
MOV P0,A
MOV A,B
MOVC A,@A+DPTR
MOV P2,A
```

```
MOV TCOUNT,#00H
MOV TMOD,#01H
MOV TH0,#(65536-50000) / 256
MOV TL0,#(65536-50000) MOD 256
SETB TR0
SETB ET0
SETB EA
SJMP $
INT0X:
MOV TH0,#(65536-50000) / 256
MOV TL0,#(65536-50000) MOD 256
INC TCOUNT
MOV A,TCOUNT
CJNE A,#20,NEXT
MOV TCOUNT,#00H
INC SECOND
MOV A,SECOND
CJNE A,#60,NEX
MOV SECOND,#00H
NEX: MOV A,SECOND
MOV B,#10
DIV AB
MOV DPTR,#TABLE
MOVC A,@A+DPTR
MOV P0,A
MOV A,B
MOVC A,@A+DPTR
MOV P2,A
NEXT: RETI

TABLE: DB 3FH,06H,5BH,4FH,66H,6DH,7DH,07H,7FH,6FH
END
```

2. C 语言源程序（中断法）

```
#include <AT89X51.H>

unsigned char code dispcode[]={0x3f,0x06,0x5b,0x4f,
0x66,0x6d,0x7d,0x07,
0x7f,0x6f,0x77,0x7c,
0x39,0x5e,0x79,0x71,0x00};
unsigned char second;
unsigned char tcount;

void main(void)
{
```

```
TMOD=0x01;
TH0=(65536-50000)/256;
TL0=(65536-50000)%256;
TR0=1;
ET0=1;
EA=1;
tcount=0;
second=0;
P0=dispcode[second/10];
P2=dispcode[second%10];
while(1);
}

void t0(void) interrupt 1 using 0
{
tcount++;
if(tcount==20)
{
tcount=0;
second++;
if(second==60)
{
second=0;
}
P0=dispcode[second/10];
P2=dispcode[second%10];
}
TH0=(65536-50000)/256;
TL0=(65536-50000)%256;
}
```

16. 定时计数器 T0 作定时应用技术（二）

1. 实验任务

用 AT89S51 的定时/计数器 T0 产生 2 秒钟的定时，每当 2 秒定时到来时，更换指示灯闪烁，每个指示闪烁的频率为 0.2 秒，也就是说，开始 L1 指示灯以 0.2 秒的速率闪烁，当 2 秒定时到来之后，L2 开始以 0.2 秒的速率闪烁，如此循环下去。0.2 秒的闪烁速率也由定时/计数器 T0 来完成。

2. 电路原理图

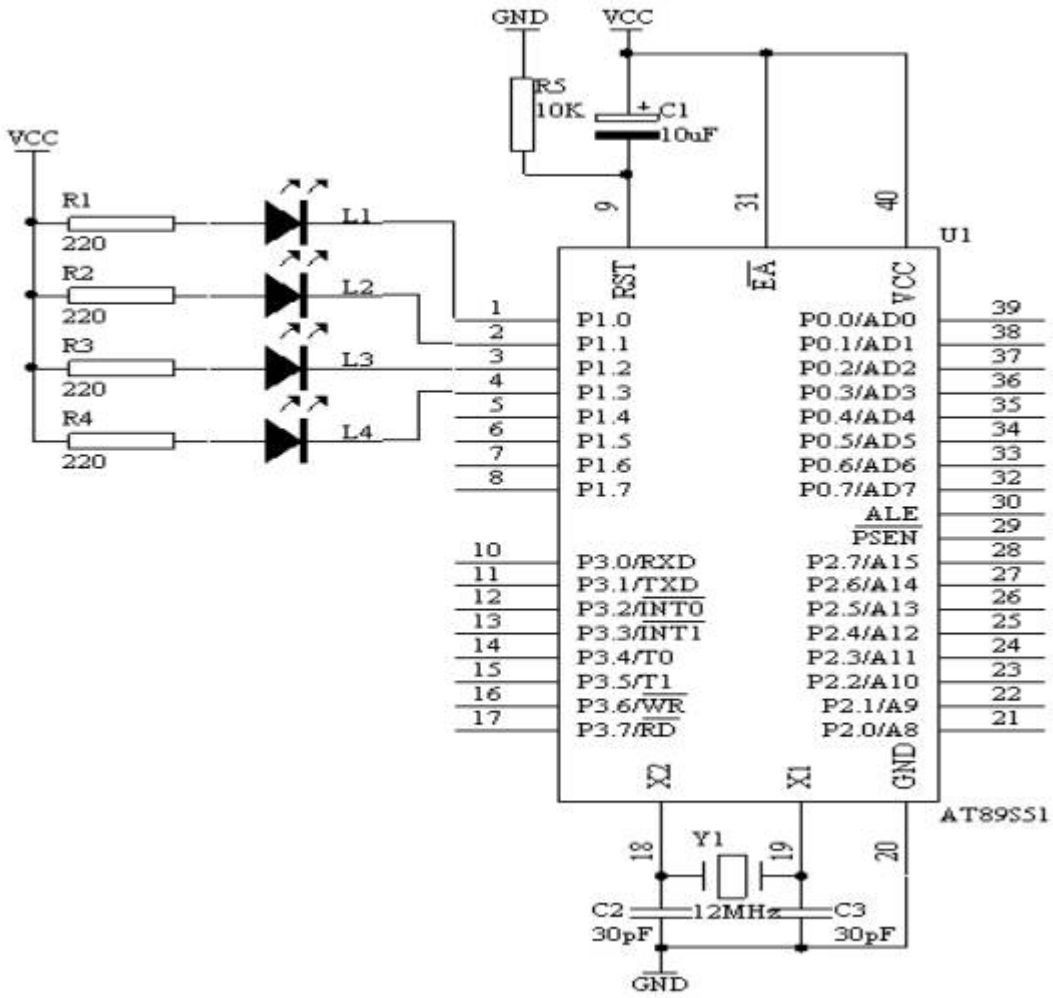


图 4.16.1

3. 系统板硬件连线

（1. 把“单片机系统”区域中的 P1.0－P1.3 用导线连接到“八路发光二极管指示模块”区域中的 L1－L4 上

4. 程序设计内容

（1. 由于采用中断方式来完成，因此，对于中断源必须它的中断入口地址，对于定时/计数器 T0 来说，中断入口地址为 000BH，因此在中断入口地方加入长跳转指令来执行中断服务程序。书写汇编源程序格式如下所示：

```
ORG   00H
LJMP   START
ORG   0BH   ;定时/计数器 T0 中断入口地址
LJMP INT_T0
START: NOP   ;主程序开始
.
```

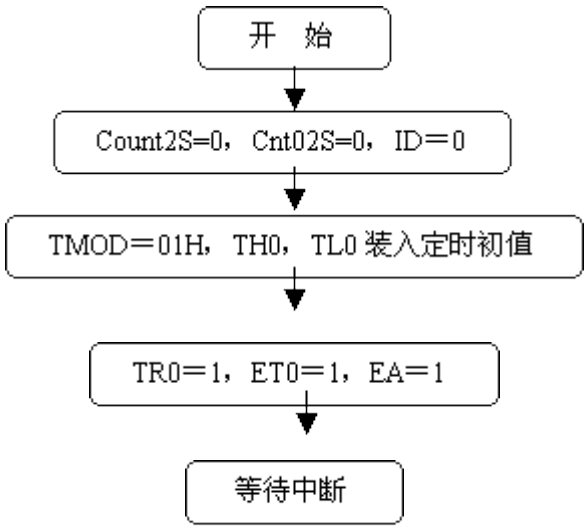
```
.

INT_T0: PUSH ACC   ;定时/计数器 T0 中断服务程序
PUSH PSW
.
.
POP PSW
POP ACC
RETI   ;中断服务程序返回
END
```

（2. 定时 2 秒，采用 16 位定时 50ms，共定时 40 次才可达到 2 秒，每 50ms 产生一中断，定时的 40 次数在中断服务程序中完成，同样 0.2 秒的定时，需要 4 次才可达到 0.2 秒。对于中断程序，在主程序中要对中断开中断。

（3. 由于每次 2 秒定时到时，L1－L4 要交替闪烁。采用 ID 来号来识别。当 ID＝0 时，L1 在闪烁，当 ID＝1 时，L2 在闪烁；当 ID＝2 时，L3 在闪烁；当 ID＝3 时，L4 在闪烁

5. 程序框图

T0 中断服务程序框图



主程序框图

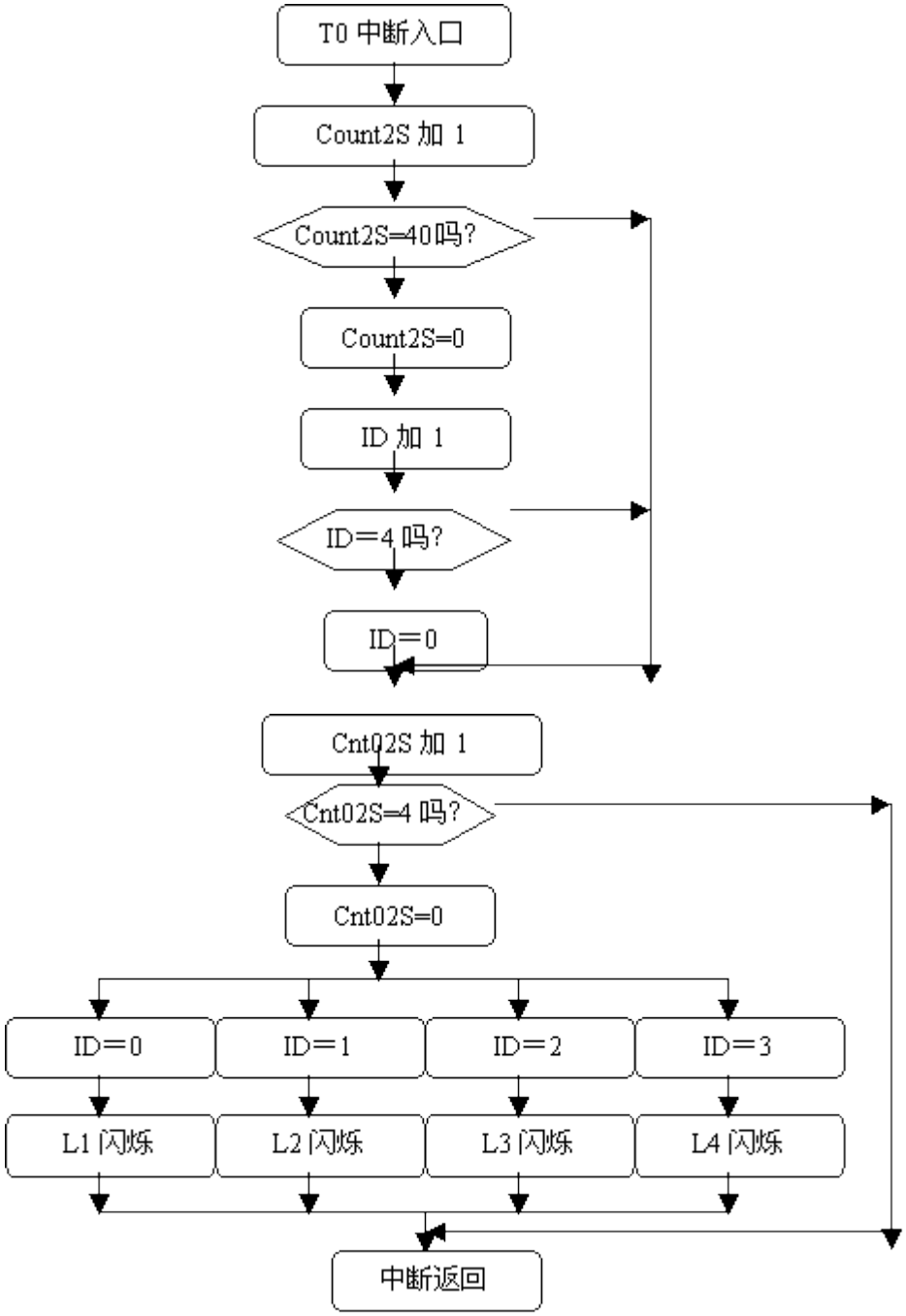


图 4.16.2

6. 汇编源程序

6. 汇编源程序

```
TCOUNT2S EQU 30H
TCNT02S EQU 31H
ID EQU 32H
ORG 00H
LJMP START
ORG 0BH
```

```
LJMP INT_T0
START: MOV TCOUNT2S,#00H
MOV TCNT02S,#00H
MOV ID,#00H
MOV TMOD,#01H
MOV TH0,#(65536-50000) / 256
MOV TL0,#(65536-50000) MOD 256
SETB TR0
SETB ET0
SETB EA
SJMP $
INT_T0: MOV TH0,#(65536-50000) / 256
MOV TL0,#(65536-50000) MOD 256
INC TCOUNT2S
MOV A,TCOUNT2S
CJNE A,#40,NEXT
MOV TCOUNT2S,#00H
INC ID
MOV A,ID
CJNE A,#04H,NEXT
MOV ID,#00H
NEXT: INC TCNT02S
MOV A,TCNT02S
CJNE A,#4,DONE
MOV TCNT02S,#00H
MOV A,ID
CJNE A,#00H,SID1
CPL P1.0
SJMP DONE
SID1: CJNE A,#01H,SID2
CPL P1.1
SJMP DONE
SID2: CJNE A,#02H,SID3
CPL P1.2
SJMP DONE
SID3: CJNE A,#03H,SID4
CPL P1.3
SID4: SJMP DONE
DONE: RETI
END
```

7. C 语言源程序

```
#include <AT89X51.H>

unsigned char tcount2s;
```

```
unsigned char tcount02s;
unsigned char ID;

void main(void)
{
TMOD=0x01;
TH0=(65536-50000)/256;
TL0=(65536-50000)%256;
TR0=1;
ET0=1;
EA=1;

while(1);
}

void t0(void) interrupt 1 using 0
{
tcount2s++;
if(tcount2s==40)
{
tcount2s=0;
ID++;
if(ID==4)
{
ID=0;
}
}
tcount02s++;
if(tcount02s==4)
{
tcount02s=0;
switch(ID)
{
case 0:
P1_0=~P1_0;
break;
case 1:
P1_1=~P1_1;
break;
case 2:
P1_2=~P1_2;
break;
case 3:
P1_3=~P1_3;
```

```
break;
}
}
}
```

17. 99 秒马表设计

1. 实验任务

（1. 开始时，显示“00”，第 1 次按下 SP1 后就开始计时。
（2. 第 2 次按 SP1 后，计时停止。
（3. 第 3 次按 SP1 后，计时归零。

2. 电路原理图

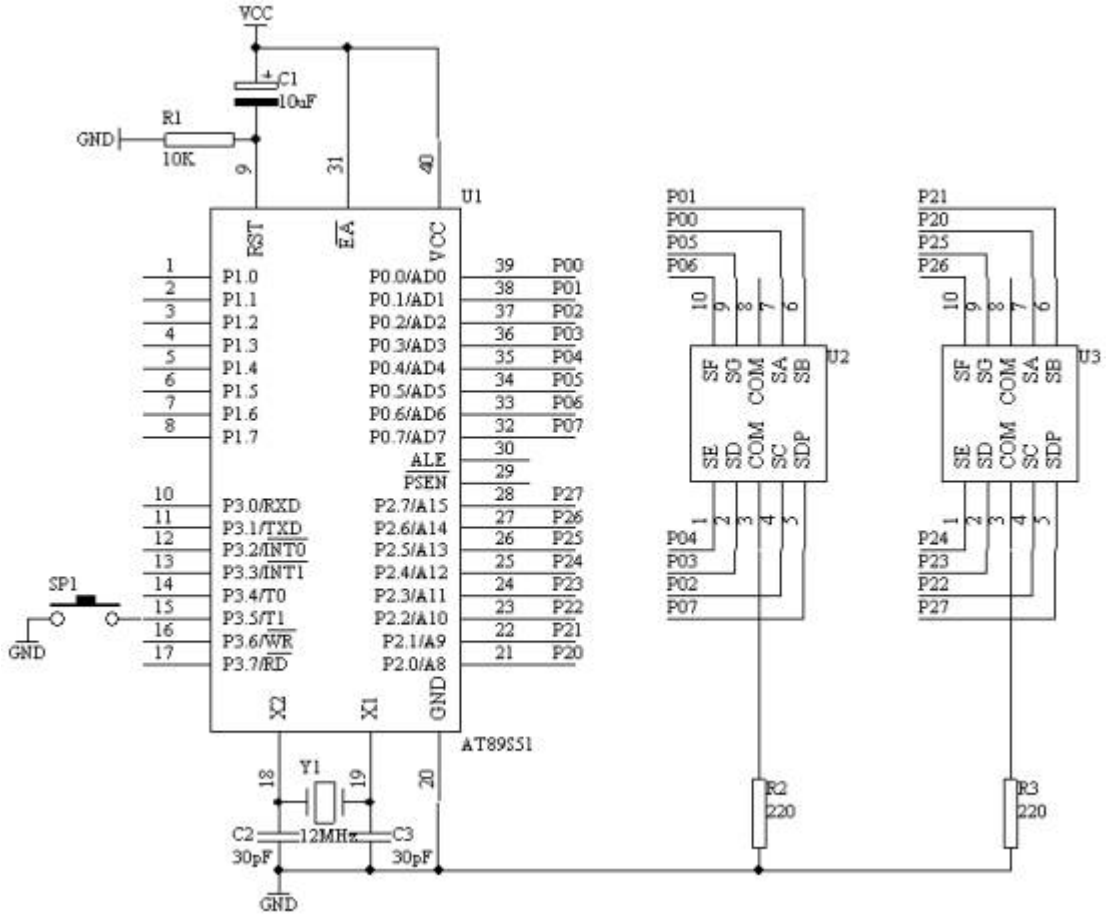


图 4.17.1

3. 系统板上硬件连线

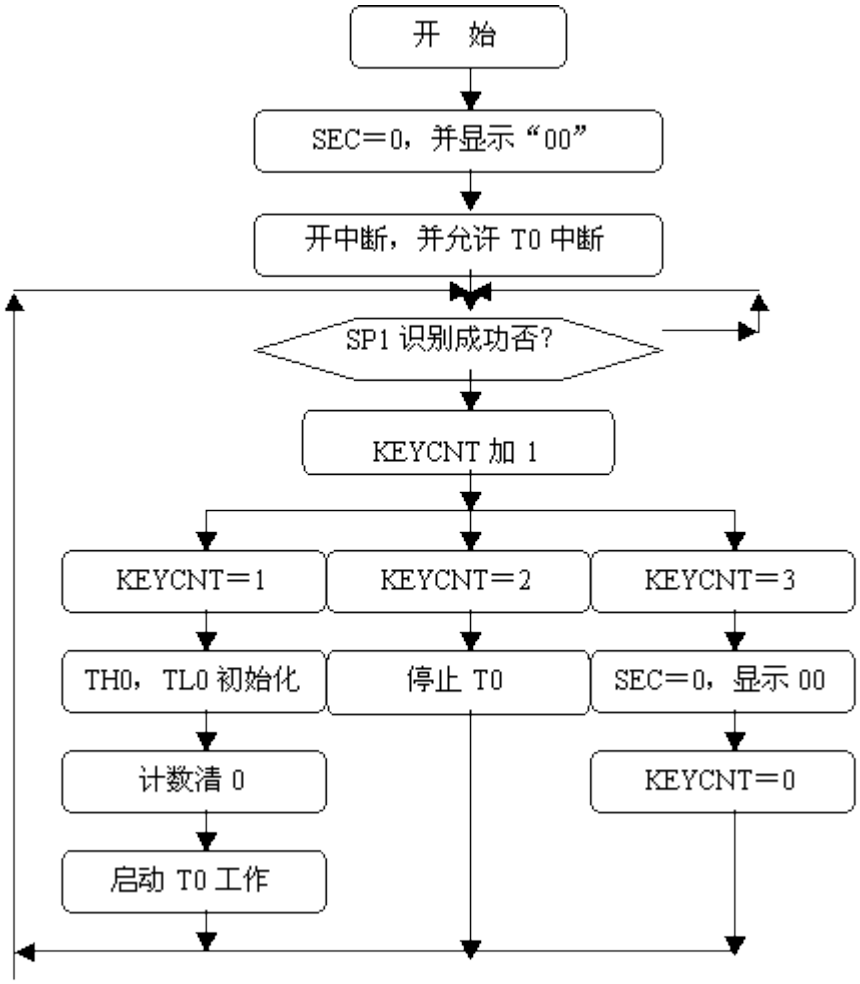
（1. 把“单片机系统”区域中的 P0.0/AD0－P0.7/AD7 端口用 8 芯排线连接到“四路静态数码显示模块”区域中的任一个 a－h 端口上；要求：P0.0/AD0 对应着 a，P0.1/AD1 对应着 b，……，P0.7/AD7 对应着 h。

（2. 把“单片机系统”区域中的P2.0/A8－P2.7/A15端口用8芯排线连接到“四路静态数码显示模块”区域中的任一个a－h端口上；要求：P2.0/A8对应着a，P2.1/A9对应着b，……，P2.7/A15对应着h。

（3. 把“单片机系统“区域中的P3.5/T1用导线连接到”独立式键盘“区域中的SP1端口上；

4. 程序框图

主程序框图



T0中断服务程序框图

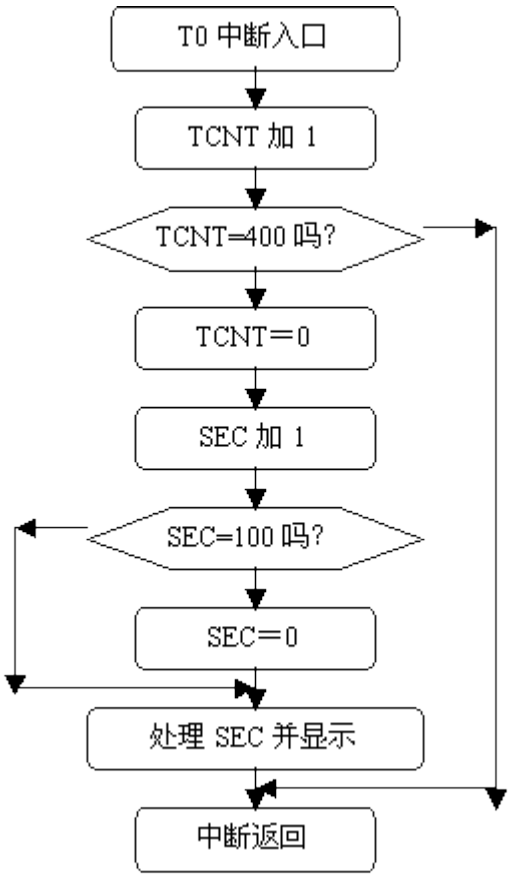


图 4.17.2

5. 汇编源程序

```
TCNTA EQU 30H
TCNTB EQU 31H
SEC EQU 32H
KEYCNT EQU 33H
SP1 BIT P3.5
ORG 00H
LJMP START
ORG 0BH
LJMP INT_T0
START: MOV KEYCNT,#00H
MOV SEC,#00H
MOV A,SEC
MOV B,#10
DIV AB
MOV DPTR,#TABLE
MOVC A,@A+DPTR
MOV P0,A
MOV A,B
MOV DPTR,#TABLE
```

```
MOVC A,@A+DPTR
MOV P2,A
MOV TMOD,#02H
SETB ET0
SETB EA
WT: JB SP1,WT
LCALL DELY10MS
JB SP1,WT
INC KEYCNT
MOV A,KEYCNT
CJNE A,#01H,KN1
SETB TR0
MOV TH0,#06H
MOV TL0,#06H
MOV TCNTA,#00H
MOV TCNTB,#00H
LJMP DKN
KN1: CJNE A,#02H,KN2
CLR TR0
LJMP DKN
KN2: CJNE A,#03H,DKN
MOV SEC,#00H
MOV A,SEC
MOV B,#10
DIV AB
MOV DPTR,#TABLE
MOVC A,@A+DPTR
MOV P0,A
MOV A,B
MOV DPTR,#TABLE
MOVC A,@A+DPTR
MOV P2,A
MOV KEYCNT,#00H
DKN: JNB SP1,$
LJMP WT
DELY10MS:
MOV R6,#20
D1: MOV R7,#248
DJNZ R7,$
DJNZ R6,D1
RET
INT_T0:
INC TCNTA
MOV A,TCNTA
```

```
CJNE A,#100,NEXT
MOV TCNTA,#00H
INC TCNTB
MOV A,TCNTB
CJNE A,#4,NEXT
MOV TCNTB,#00H
INC SEC
MOV A,SEC
CJNE A,#100,DONE
MOV SEC,#00H
DONE: MOV A,SEC
MOV B,#10
DIV AB
MOV DPTR,#TABLE
MOVC A,@A+DPTR
MOV P0,A
MOV A,B
MOV DPTR,#TABLE
MOVC A,@A+DPTR
MOV P2,A
NEXT: RETI
TABLE: DB 3FH,06H,5BH,4FH,66H,6DH,7DH,07H,7FH,6FH
END
```

6. C 语言源程序

```
#include <AT89X51.H>

unsigned char code dispcode[]={0x3f,0x06,0x5b,0x4f,
0x66,0x6d,0x7d,0x07,
0x7f,0x6f,0x77,0x7c,
0x39,0x5e,0x79,0x71,0x00};
unsigned char second;
unsigned char keycnt;
unsigned int tcnt;

void main(void)
{
unsigned char i,j;

TMOD=0x02;
ET0=1;
EA=1;
second=0;
P0=dispcode[second/10];
P2=dispcode[second%10];
```

```
while(1)
{
if(P3_5==0)
{
for(i=20;i>0;i--)
for(j=248;j>0;j--);
if(P3_5==0)
{
keycnt++;
switch(keycnt)
{
case 1:
TH0=0x06;
TL0=0x06;
TR0=1;
break;
case 2:
TR0=0;
break;
case 3:
keycnt=0;
second=0;
P0=dispcode[second/10];
P2=dispcode[second%10];
break;
}
while(P3_5==0);
}
}
}
}

void t0(void) interrupt 1 using 0
{
tcnt++;
if(tcnt==400)
{
tcnt=0;
second++;
if(second==100)
{
second=0;
}
P0=dispcode[second/10];
```

```
P2=dispcode[second%10];
}
}
```

18. “嘀、嘀、……”报警声

1. 实验任务

用 AT89S51 单片机产生“嘀、嘀、…”报警声从 P1.0 端口输出，产生频率为 1KHz，根据上面图可知：1KHZ 方波从 P1.0 输出 0.2 秒，接着 0.2 秒从 P1.0 输出电平信号，如此循环下去，就形成我们所需的报警声了。

2. 电路原理图

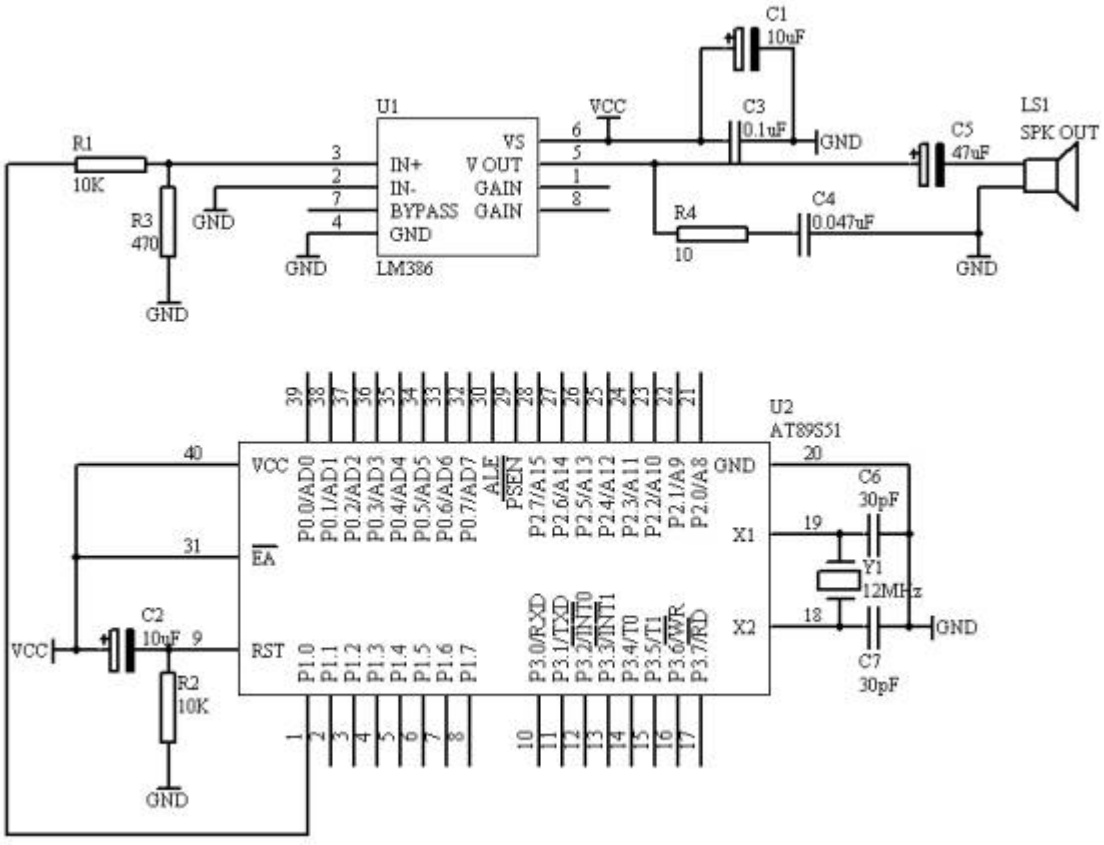


图 4.18.1

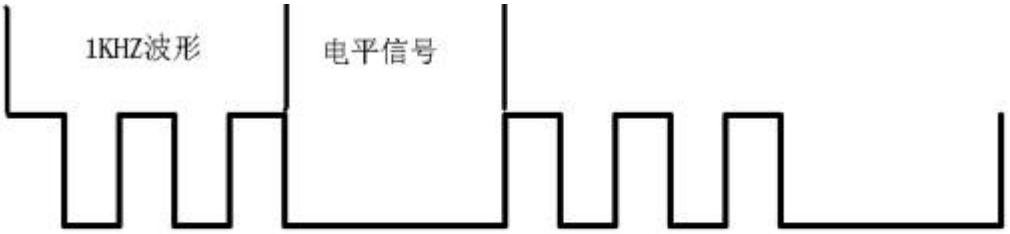
3. 系统板硬件连线

（1. 把“单片机系统”区域中的 P1.0 端口用导线连接到“音频放大模块”区域中的 SPK IN 端口上，

（2. 在“音频放大模块”区域中的 SPK OUT 端口上接上一个 8 欧或者是 16 欧的喇叭；

4. 程序设计方法

（1. 生活中我们常常到各种各样的报警声，例如“嘀、嘀、…”就是常见的一种声音报警声，但对于这种报警声，嘀 0.2 秒钟，然后断 0.2 秒钟，如此循环下去，假设嘀声的频率为 1KHz，则报警声时序图如下图所示：



上述波形信号如何用单片机来产生呢？

（2. 由于要产生上面的信号，我们把上面的信号分成两部分，一部分为 1KHZ 方波，占用时间为 0.2 秒；另一部分为电平，也是占用 0.2 秒；因此，我们利用单片机的定时/计数器 T0 作为定时，可以定时 0.2 秒；同时，也要用单片机产生 1KHZ 的方波，对于 1KHZ 的方波信号周期为 1ms，高电平占用 0.5ms，低电平占用 0.5ms，因此也采用定时器 T0 来完成 0.5ms 的定时；最后，可以选定定时/计数器 T0 的定时时间为 0.5ms，而要定时 0.2 秒则是 0.5ms 的 400 倍，也就是说以 0.5ms 定时 400 次就达到 0.2 秒的定时时间了。

5. 程序框图

主程序框图

中断服务程序框图

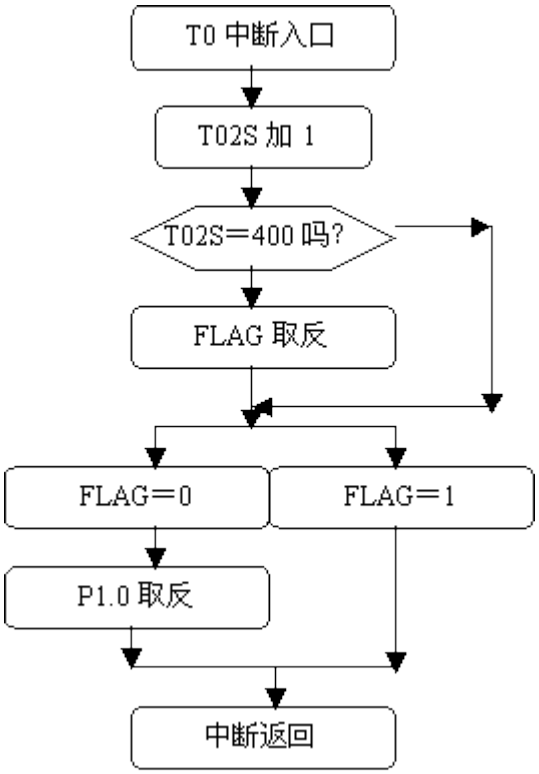


图 4.18.2

6. 汇编源程序

```
T02SA EQU 30H
T02SB EQU 31H
FLAG BIT 00H
ORG 00H
LJMP START
ORG 0BH
LJMP INT_T0
START: MOV T02SA,#00H
MOV T02SB,#00H
CLR FLAG
MOV TMOD,#01H
MOV TH0,#(65536-500) / 256
MOV TL0,#(65536-500) MOD 256
SETB TR0
SETB ET0
SETB EA
SJMP $
INT_T0:
MOV TH0,#(65536-500) / 256
MOV TL0,#(65536-500) MOD 256
INC T02SA
```

```
MOV A,T02SA
CJNE A,#100,NEXT
INC T02SB
MOV A,T02SB
CJNE A,#04H,NEXT
MOV T02SA,#00H
MOV T02SB,#00H
CPL FLAG
NEXT: JB FLAG,DONE
CPL P1.0
DONE: RETI
END
```

7. C 语言源程序

```
#include <AT89X51.H>
unsigned int t02s;
unsigned char t05ms;
bit flag;

void main(void)
{
TMOD=0x01;
TH0=(65536-500)/256;
TL0=(65536-500)%256;
TR0=1;
ET0=1;
EA=1;
while(1);
}

void t0(void) interrupt 1 using 0
{
TH0=(65536-500)/256;
TL0=(65536-500)%256;
t02s++;
if(t02s==400)
{
t02s=0;
flag=~flag;
}
if(flag==0)
{
P1_0=~P1_0;
}
}
```

19. “叮咚”门铃

1. 实验任务

当按下开关 SP1，AT89S51 单片机产生“叮咚”声从 P1.0 端口输出到 LM386，经过放大之后送入喇叭。

2. 电路原理图

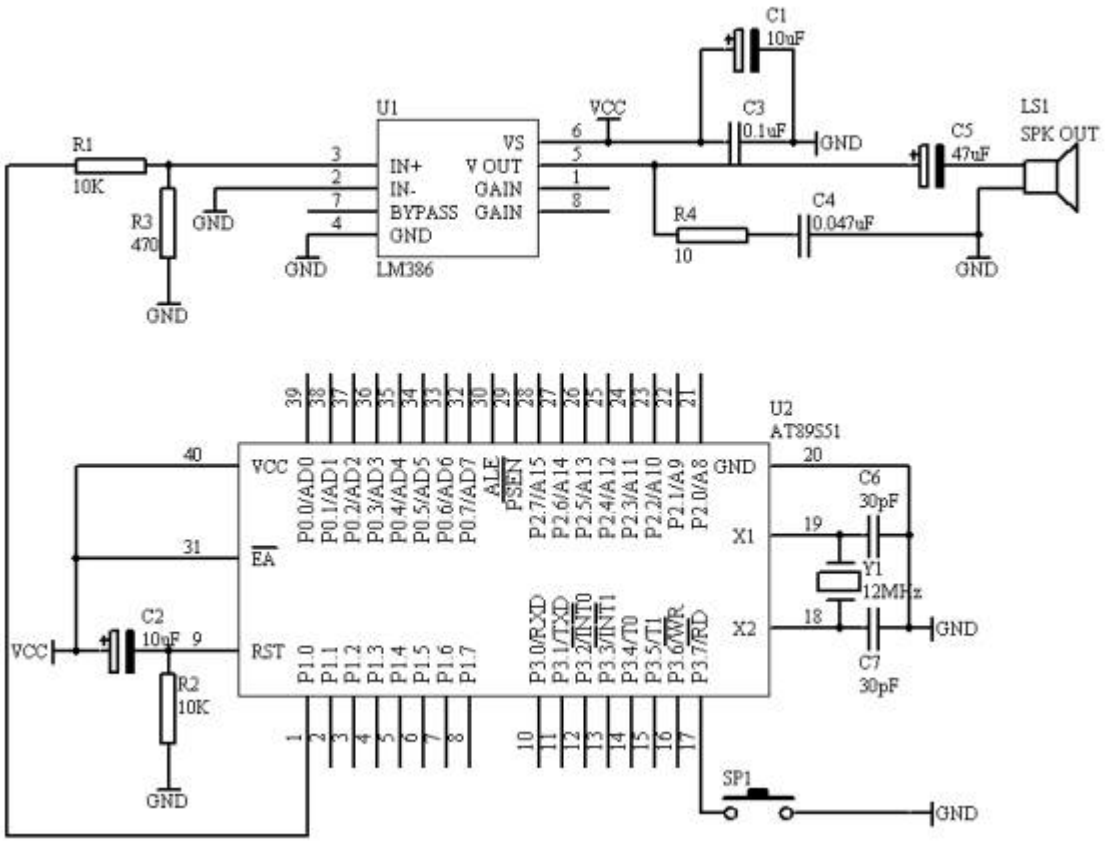


图 4.19.1

3. 系统板上硬件连线

（1. 把“单片机系统”区域中的 P1.0 端口用导线连接到“音频放大模块”区域中的 SPK IN 端口上；

（2. 在“音频放大模块”区域中的 SPK OUT 端口上接上一个 8 欧或者是 16 欧的喇叭；

（3. 把“单片机系统”区域中的 P3.7/RD 端口用导线连接到“独立式键盘”区域中的 SP1 端口上；

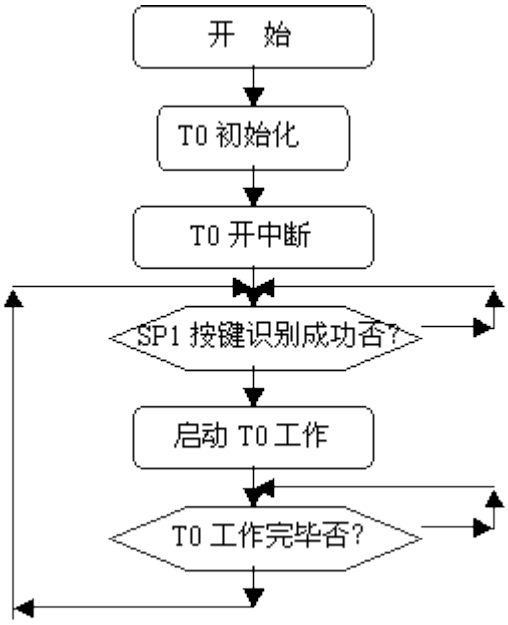
4. 程序设计方法

（1. 我们用单片机实定时/计数器 T0 来产生 700HZ 和 500HZ 的频率，根据定时/计数器 T0，我们取定时 250us，因此，700HZ 的频率要经过 3 次 250us 的定时，而 500HZ 的频率要经过 4 次 250us 的定时。

（2. 在设计过程，只有当按下 SP1 之后，才启动 T0 开始工作，当T0 工作完毕，回到最初状态。

（3. “叮”和“咚”声音各占用 0.5 秒，因此定时/计数器 T0 要完成 0.5 秒的定时，对于以 250us 为基准定时 2000 次才可以。

5. 程序框图



主程序框图

T0 中断服务程序框图

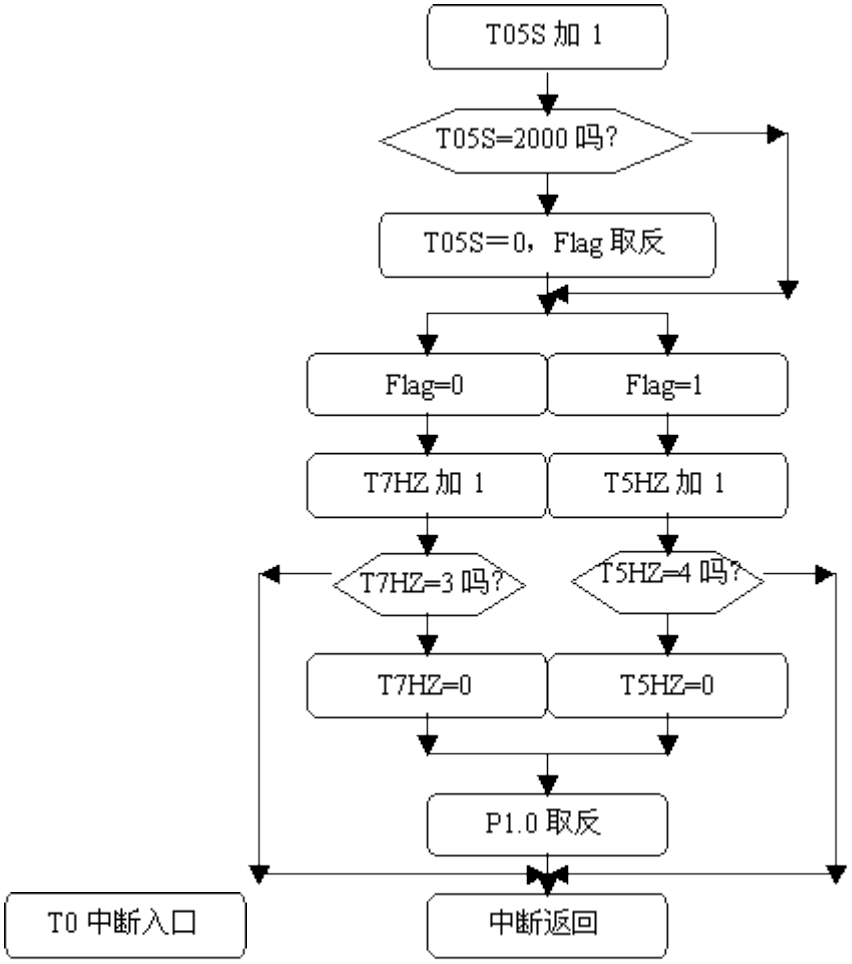


图 4.19.2

6. 汇编源程序

```
T5HZ EQU 30H
T7HZ EQU 31H
T05SA EQU 32H
T05SB EQU 33H
FLAG BIT 00H
STOP BIT 01H
SP1 BIT P3.7
ORG 00H
LJMP START
ORG 0BH
LJMP INT_T0
START: MOV TMOD,#02H
MOV TH0,#06H
MOV TL0,#06H
SETB ET0
SETB EA
NSP: JB SP1,NSP
```

```
LCALL DELY10MS
JB SP1,NSP
SETB TR0
MOV T5HZ,#00H
MOV T7HZ,#00H
MOV T05SA,#00H
MOV T05SB,#00H
CLR FLAG
CLR STOP
JNB STOP,$
LJMP NSP
DELY10MS: MOV R6,#20
D1: MOV R7,#248
DJNZ R7,$
DJNZ R6,D1
RET
INT_T0: INC T05SA
MOV A,T05SA
CJNE A,#100,NEXT
MOV T05SA,#00H
INC T05SB
MOV A,T05SB
CJNE A,#20,NEXT
MOV T05SB,#00H
JB FLAG,STP
CPL FLAG
LJMP NEXT
STP: SETB STOP
CLR TR0
LJMP DONE
NEXT: JB FLAG,S5HZ
INC T7HZ
MOV A,T7HZ
CJNE A,#03H,DONE
MOV T7HZ,#00H
CPL P1.0
LJMP DONE
S5HZ: INC T5HZ
MOV A,T5HZ
CJNE A,#04H,DONE
MOV T5HZ,#00H
CPL P1.0
LJMP DONE
DONE: RETI
```

```
END
```

7. C 语言源程序

```
#include <AT89X51.H>
unsigned char t5hz;
unsigned char t7hz;
unsigned int tcnt;

bit stop;
bit flag;

void main(void)
{
unsigned char i,j;

TMOD=0x02;
TH0=0x06;
TL0=0x06;
ET0=1;
EA=1;

while(1)
{
if(P3_7==0)
{
for(i=10;i>0;i--)
for(j=248;j>0;j--);
if(P3_7==0)
{
t5hz=0;
t7hz=0;
tcnt=0;
flag=0;
stop=0;
TR0=1;
while(stop==0);
}
}
}
}

void t0(void) interrupt 1 using 0
{
tcnt++;
```

```
if(tcnt==2000)
{
tcnt=0;
if(flag==0)
{
flag=~flag;
}
else
{
stop=1;
TR0=0;
}
}
if(flag==0)
{
t7hz++;
if(t7hz==3)
{
t7hz=0;
P1_0=~P1_0;
}
}
else
{
t5hz++;
if(t5hz==4)
{
t5hz=0;
P1_0=~P1_0;
}
}
}
```

20. 数字钟（★）

1. 实验任务

（1. 开机时，显示 12：00：00 的时间开始计时；

（2. P0.0/AD0 控制“秒”的调整，每按一次加 1 秒；

（3. P0.1/AD1 控制“分”的调整，每按一次加 1 分；

（4. P0.2/AD2 控制“时”的调整，每按一次加 1 个小时；

2. 电路原理图

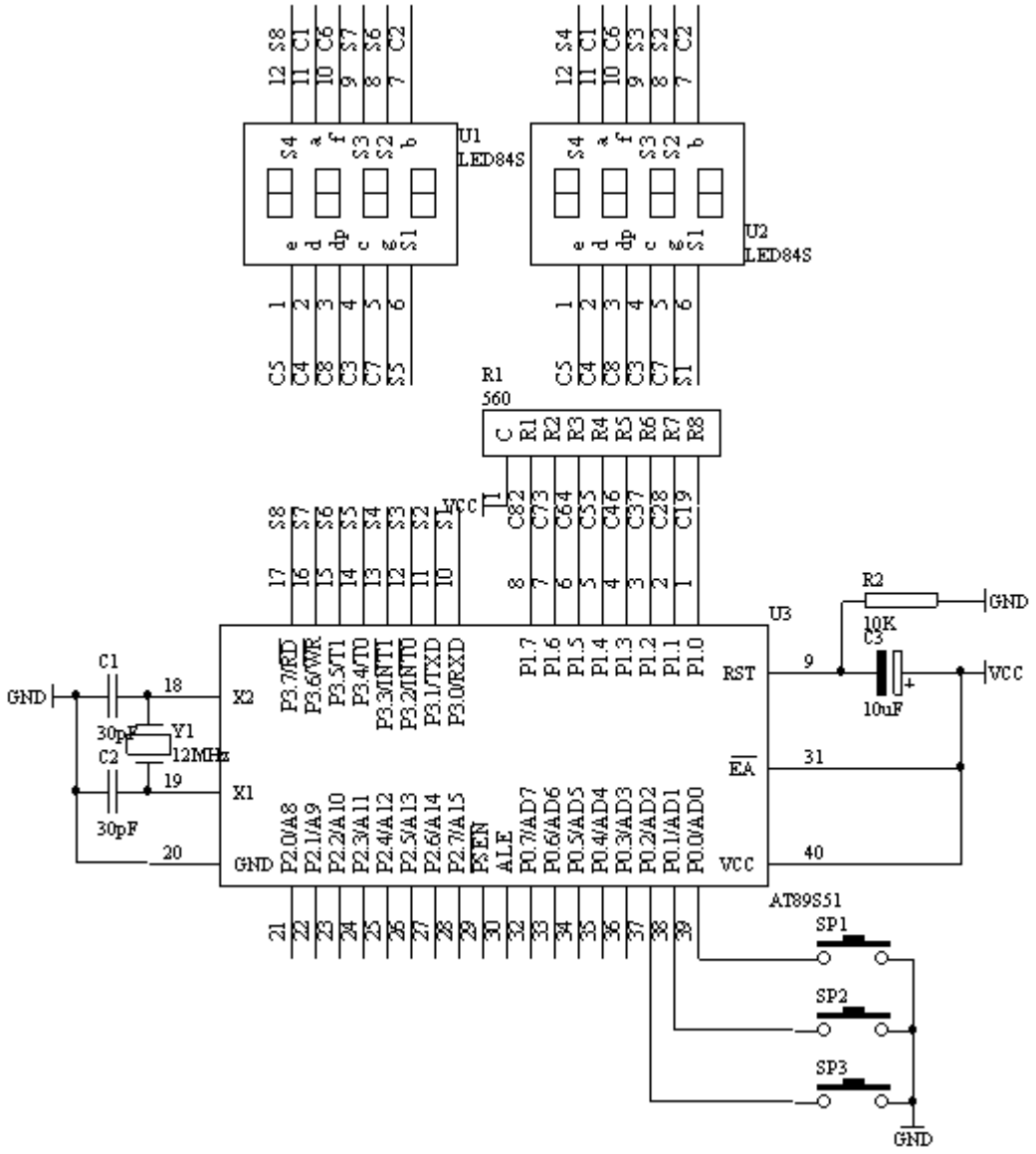


图 4.20.1

3. 系统板上硬件连线

（1. 把“单片机系统”区域中的 P1.0－P1.7 端口用 8 芯排线连接到“动态数码显示”区域中的 A－H 端口上；

（2. 把“单片机系统：区域中的 P3.0－P3.7 端口用 8 芯排线连接到“动态数码显示”区域中的 S1－S8 端口上；

（3. 把“单片机系统”区域中的 P0.0/AD0、P0.1/AD1、P0.2/AD2 端口分别用导线连接到“独立式键盘”区域中的 SP3、SP2、SP1 端口上；

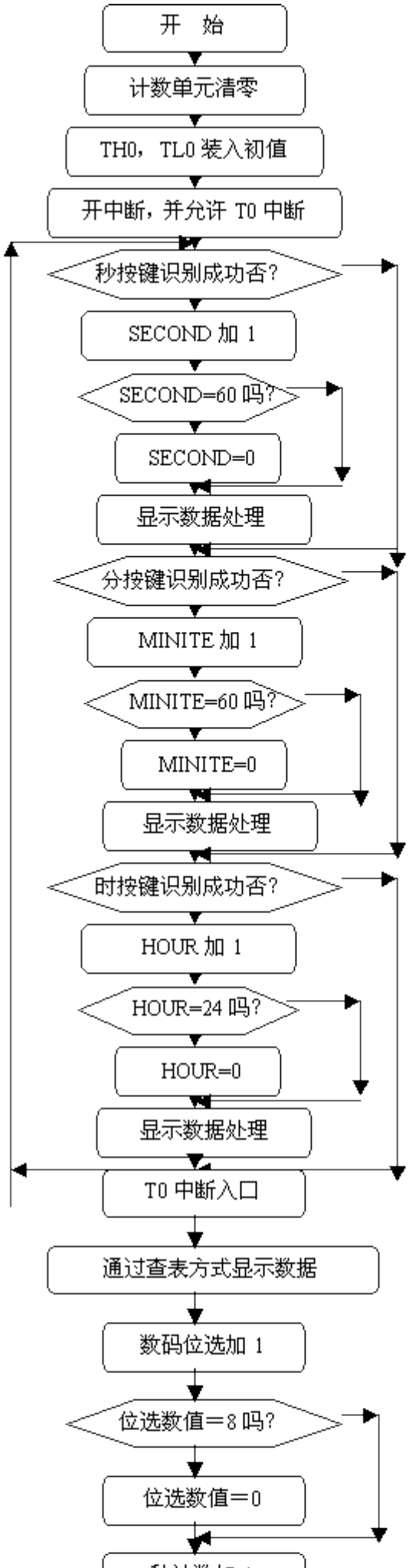
4. 相关基本知识

（1. 动态数码显示的方法

（2. 独立式按键识别过程

（3. “时”，“分”，“秒”数据送出显示处理方法

5. 程序框图


开　始
计数单元清零
TH0，TL0 装入初值
开中断，并允许 T0 中断
秒按键识别成功否?
SECOND 加 1
SECOND=60 吗?
SECOND=0
显示数据处理
分按键识别成功否?
MINITE 加 1
MINITE=60 吗?
MINITE=0
显示数据处理
时按键识别成功否?
HOUR 加 1
HOUR=24 吗?
HOUR=0
显示数据处理
T0 中断入口
通过查表方式显示数据
数码位选加 1
位选数值＝8 吗?
位选数值＝0

6. 汇编源程序

```
SECOND EQU 30H
MINITE EQU 31H
HOUR EQU 32H
HOURK BIT P0.0
MINITEK BIT P0.1
SECONDK BIT P0.2
DISPBUF EQU 40H
DISPBIT EQU 48H
T2SCNTA EQU 49H
T2SCNTB EQU 4AH
TEMP EQU 4BH

ORG 00H
LJMP START
ORG 0BH
LJMP INT_T0
START: MOV SECOND,#00H
MOV MINITE,#00H
MOV HOUR,#12
MOV DISPBIT,#00H
MOV T2SCNTA,#00H
MOV T2SCNTB,#00H
MOV TEMP,#0FEH
LCALL DISP
MOV TMOD,#01H
MOV TH0,#(65536-2000) / 256
MOV TL0,#(65536-2000) MOD 256
SETB TR0
SETB ET0
SETB EA
WT: JB SECONDK,NK1
LCALL DELY10MS
JB SECONDK,NK1
INC SECOND
MOV A,SECOND
CJNE A,#60,NS60
MOV SECOND,#00H
NS60: LCALL DISP
JNB SECONDK,$
NK1: JB MINITEK,NK2
LCALL DELY10MS
JB MINITEK,NK2
INC MINITE
```

```
MOV A,MINITE
CJNE A,#60,NM60
MOV MINITE,#00H
NM60: LCALL DISP
JNB MINITEK,$
NK2: JB HOURK,NK3
LCALL DELY10MS
JB HOURK,NK3
INC HOUR
MOV A,HOUR
CJNE A,#24,NH24
MOV HOUR,#00H
NH24: LCALL DISP
JNB HOURK,$
NK3: LJMP WT
DELY10MS:
MOV R6,#10
D1: MOV R7,#248
DJNZ R7,$
DJNZ R6,D1
RET
DISP:
MOV A,#DISPBUF
ADD A,#8
DEC A
MOV R1,A
MOV A,HOUR
MOV B,#10
DIV AB
MOV @R1,A
DEC R1
MOV A,B
MOV @R1,A
DEC R1
MOV A,#10
MOV@R1,A
DEC R1
MOV A,MINITE
MOV B,#10
DIV AB
MOV @R1,A
DEC R1
MOV A,B
MOV @R1,A
```

```
DEC R1
MOV A,#10
MOV@R1,A
DEC R1
MOV A,SECOND
MOV B,#10
DIV AB
MOV @R1,A
DEC R1
MOV A,B
MOV @R1,A
DEC R1
RET
INT_T0:
MOV TH0,#(65536-2000) / 256
MOV TL0,#(65536-2000) MOD 256
MOV A,#DISPBUF
ADD A,DISPBIT
MOV R0,A
MOV A,@R0
MOV DPTR,#TABLE
MOVC A,@A+DPTR
MOV P1,A
MOV A,DISPBIT
MOV DPTR,#TAB
MOVC A,@A+DPTR
MOV P3,A
INC DISPBIT
MOV A,DISPBIT
CJNE A,#08H,KNA
MOV DISPBIT,#00H
KNA: INC T2SCNTA
MOV A,T2SCNTA
CJNE A,#100,DONE
MOV T2SCNTA,#00H
INC T2SCNTB
MOV A,T2SCNTB
CJNE A,#05H,DONE
MOV T2SCNTB,#00H
INC SECOND
MOV A,SECOND
CJNE A,#60,NEXT
MOV SECOND,#00H
INC MINITE
```

```
MOV A,MINITE
CJNE A,#60,NEXT
MOV MINITE,#00H
INC HOUR
MOV A,HOUR
CJNE A,#24,NEXT
MOV HOUR,#00H
NEXT: LCALL DISP
DONE: RETI
TABLE: DB 3FH,06H,5BH,4FH,66H,6DH,7DH,07H,7FH,6FH,40H
TAB: DB 0FEH,0FDH,0FBH,0F7H,0EFH,0DFH,0BFH,07FH
END
```

7. C 语言源程序

```
#include <AT89X51.H>
unsigned char code dispcode[]={0x3f,0x06,0x5b,0x4f,
0x66,0x6d,0x7d,0x07,
0x7f,0x6f,0x77,0x7c,
0x39,0x5e,0x79,0x71,0x00};
unsigned char dispbitcode[]={0xfe,0xfd,0xfb,0xf7,
0xef,0xdf,0xbf,0x7f};
unsigned char dispbuf[8]={0,0,16,0,0,16,0,0};
unsigned char dispbitcnt;

unsigned char second;
unsigned char minite;
unsigned char hour;
unsigned int tcnt;
unsigned char mstcnt;

unsigned char i,j;

void main(void)
{
TMOD=0x02;
TH0=0x06;
TL0=0x06;
TR0=1;
ET0=1;
EA=1;

while(1)
{
if(P0_0==0)
```

```
{
for(i=5;i>0;i--)
for(j=248;j>0;j--);
if(P0_0==0)
{
second++;
if(second==60)
{
second=0;
}
dispbuf[0]=second%10;
dispbuf[1]=second/10;
while(P0_0==0);
}
}
if(P0_1==0)
{
for(i=5;i>0;i--)
for(j=248;j>0;j--);
if(P0_1==0)
{
minite++;
if(minite==60)
{
minite=0;
}
dispbuf[3]=minite%10;
dispbuf[4]=minite/10;
while(P0_1==0);
}
}
if(P0_2==0)
{
for(i=5;i>0;i--)
for(j=248;j>0;j--);
if(P0_2==0)
{
hour++;
if(hour==24)
{
hour=0;
}
dispbuf[6]=hour%10;
dispbuf[7]=hour/10;
```

```
while(P0_2==0);
}
}
}
}
void t0(void) interrupt 1 using 0
{
mstcnt++;
if(mstcnt==8)
{
mstcnt=0;
P1=dispcode[dispbuf[dispbitcnt]];
P3=dispbitcode[dispbitcnt];
dispbitcnt++;
if(dispbitcnt==8)
{
dispbitcnt=0;
}
}
tcnt++;
if(tcnt==4000)
{
tcnt=0;
second++;
if(second==60)
{
second=0;
minite++;
if(minite==60)
{
minite=0;
hour++;
if(hour==24)
{
hour=0;
}
}
}
dispbuf[0]=second%10;
dispbuf[1]=second/10;
dispbuf[3]=minite%10;
dispbuf[4]=minite/10;
dispbuf[6]=hour%10;
dispbuf[7]=hour/10;
```

```
}
}
```

21. 拉幕式数码显示技术

1. 实验任务

用 AT89S51 单片机的 P0.0/AD0－P0.7/AD7 端口接数码管的 a－h 端，8 位数码管的 S1－S8 通过 74LS138 译码器的 Y0－Y7 来控制选通每个数码管的位选端。AT89S51 单片机的 P1.0－P1.2 控制 74LS138 的 A，B，C 端子。在 8 位数码管上从右向左循环显示“12345678”。能够比较平滑地看到拉幕的效果。

2. 电路原理图

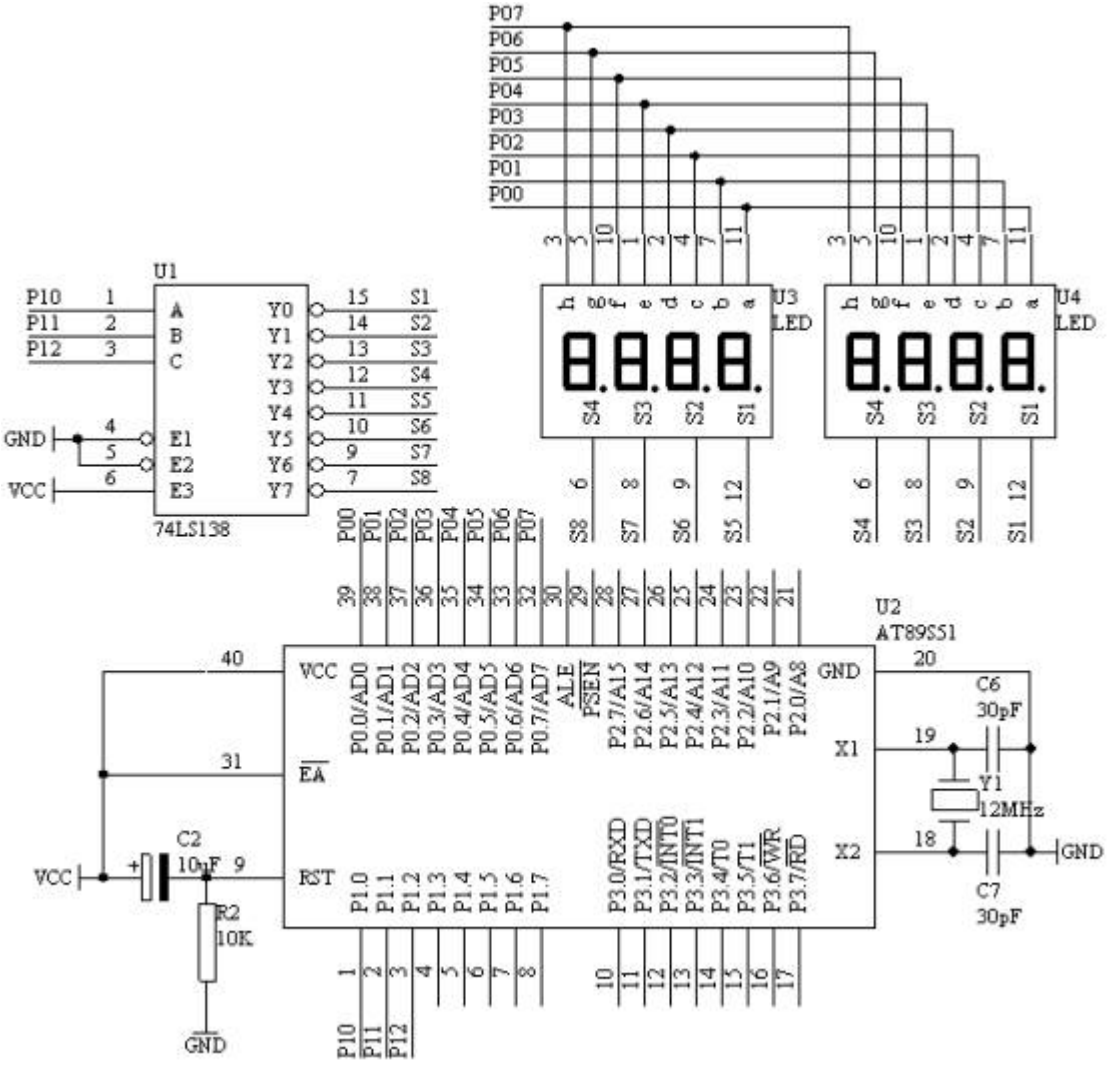


图 4.21.1

3. 系统板上硬件连线

（1. 把“单片机系统”区域中的 P0.0/AD0－P0.7/AD7 用 8 芯排线连接到“动态数码显示”区域中的 a－h 端口上；

（2. 把“三八译码模块”区域中的 Y0－Y7 用 8 芯排线连接到“动态数码显示”区域中的 S1－S8 端口上；

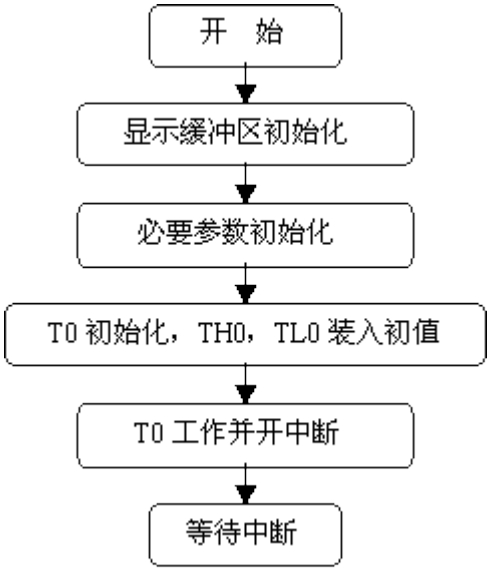
（3. 把“单片机系统”区域中的 P1.0－P1.2 端口用 3 根导线连接到“三八译码模块”区域中的 A、B、C“端口上；

4. 程序设计方法

（1. 动态数码显示技术；如何进行动态扫描，由于一次只能让一个数码管显示，因此，要显示 8 位的数据，必须经过让数码管一个一个轮流显示才可以，同时每个数码管显示的时间大约在 1ms 到 4ms 之间，所以为了保证正确显示，我必须每隔 1ms，就得刷新一个数码管。而这刷新时间我们采用单片机的定时/计数器 T0 来控制，每定时 1ms 对数码管刷新一次，T0 采用方式 2。

（2. 在进行数码显示的时候，要对显示单元开辟 8 个显示缓冲区，每个显示缓冲区装有显示的不同数据即可。

5. 程序框图



主程序框图

中断服务程序框图

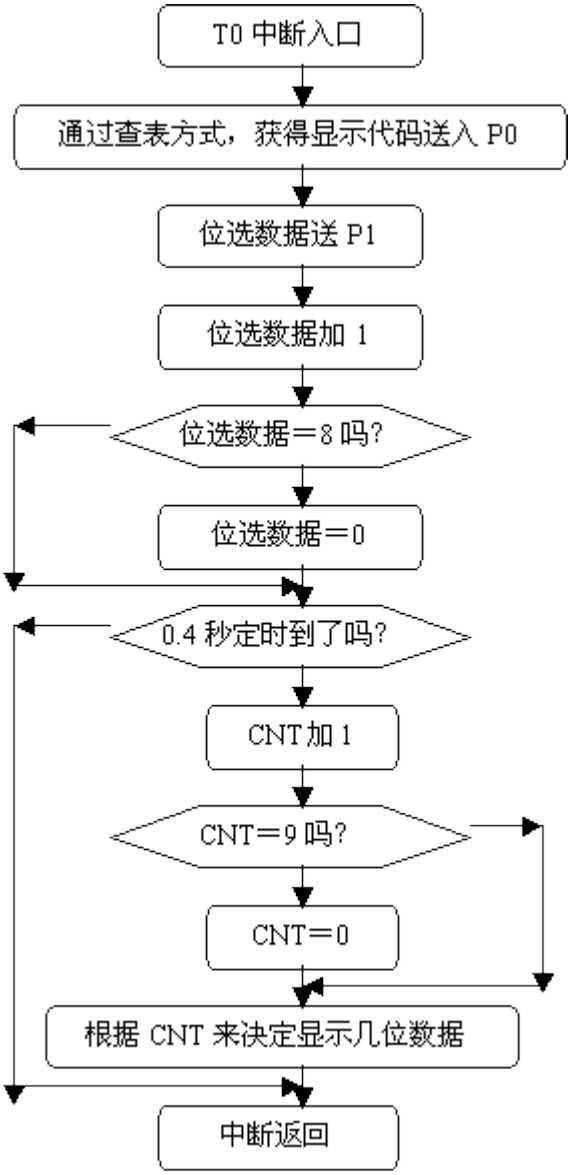


图 4.21.2

6. 汇编源程序

```
DISPBUF EQU 30H
DISPCNT EQU 38H
DISPBIT EQU 39H
T1CNTA EQU 3AH
T1CNTB EQU 3BH
CNT EQU 3CH

ORG 00H
LJMP START
ORG 0BH
LJMP INT_T0
```

```
START: MOV DISPCNT,#8
MOV A,#10
MOV R1,#DISPBUF
LP: MOV @R1,A
INC R1
DJNZ DISPCNT,LP
MOV DISPBIT,#00H
MOV T1CNTA,#00H
MOV T1CNTB,#00H
MOV CNT,#00H
MOV TMOD,#01H
MOV TH0,#(65536-1000) / 256
MOV TL0,#(65536-1000) MOD 256
SETB TR0
SETB ET0
SETB EA
SJMP $

INT_T0:
MOV TH0,#(65536-1000) / 256
MOV TL0,#(65536-1000) MOD 256
MOV A,DISPBIT
ADD A,#DISPBUF
MOV R0,A
MOV A,@R0
MOV DPTR,#TABLE
MOVC A,@A+DPTR
MOV P0,A
MOV A,P1
ANL A,#0F8H
ADD A,DISPBIT
MOV P1,A
INC DISPBIT
MOV A,DISPBIT
CJNE A,#08H,NEXT
MOV DISPBIT,#00H
NEXT: INC T1CNTA
MOV A,T1CNTA
CJNE A,#50,LL1
MOV T1CNTA,#00H
INC T1CNTB
MOV A,T1CNTB
CJNE A,#8,LL1
MOV T1CNTB,#00H
```

```
INC CNT
MOV A,CNT
CJNE A,#9,LLX
MOV CNT,#00H
MOV A,CNT
LLX: CJNE A,#01H,NEX1
MOV 30H,#8
LL1: LJMP DONE
NEX1: CJNE A,#02H,NEX2
MOV 31H,#8
MOV 30H,#8
LJMP DONE
NEX2: CJNE A,#03H,NEX3
MOV 32H,#8
MOV 31H,#8
MOV 30H,#8
LJMP DONE
NEX3: CJNE A,#04H,NEX4
MOV 33H,#8
MOV 32H,#8
MOV 31H,#8
MOV 30H,#8
LJMP DONE
NEX4: CJNE A,#05H,NEX5
MOV 34H,#8
MOV 33H,#8
MOV 32H,#8
MOV 31H,#8
MOV 30H,#8
LJMP DONE
NEX5: CJNE A,#06H,NEX6
MOV 35H,#8
MOV 34H,#8
MOV 33H,#8
MOV 32H,#8
MOV 31H,#8
MOV 30H,#8
LJMP DONE
NEX6: CJNE A,#07H,NEX7
MOV 36H,#8
MOV 35H,#8
MOV 34H,#8
MOV 33H,#8
MOV 32H,#8
```

```
MOV 31H,#8
MOV 30H,#8
LJMP DONE
NEX7: CJNE A,#08H,NEX8
MOV 37H,#8
MOV 36H,#8
MOV 35H,#8
MOV 34H,#8
MOV 33H,#8
MOV 32H,#8
MOV 31H,#8
MOV 30H,#8
LJMP DONE
NEX8: CJNE A,#00H,DONE
MOV 37H,#10
MOV 36H,#10
MOV 35H,#10
MOV 34H,#10
MOV 33H,#10
MOV 32H,#10
MOV 31H,#10
MOV 30H,#10
LL: LJMP DONE
DONE: RETI
TABLE: DB 3FH,06H,5BH,4FH,66H,6DH,7DH,07H,7FH,6FH,00H
END
```

7. C 语言源程序

```
#include <AT89X51.H>
unsigned char code dispcode[]={0x3f,0x06,0x5b,0x4f,
0x66,0x6d,0x7d,0x07,
0x7f,0x6f,0x77,0x7c,
0x39,0x5e,0x79,0x71,0x00};
unsigned char dispbitcode[]={0xf8,0xf9,0xfa,0xfb,
0xfc,0xfd,0xfe,0xff};
unsigned char dispbuf[8]={16,16,16,16,16,16,16,16};
unsigned char dispbitcnt;
unsigned int t02scnt;
unsigned char t5mscnt;
unsigned char u;
unsigned char i;

void main(void)
{
TMOD=0x02;
```

```
TH0=0x06;
TL0=0x06;
TR0=1;
ET0=1;
EA=1;
while(1);
}

void t0(void) interrupt 1 using 0
{
t5mscnt++;
if(t5mscnt==4)
{
t5mscnt=0;
P0=dispcode[dispbuf[dispbitcnt]];
P1=dispbitcode[dispbitcnt];
dispbitcnt++;
if(dispbitcnt==8)
{
dispbitcnt=0;
}
}
t02scnt++;
if(t02scnt==1600)
{
t02scnt=0;
u++;
if(u==9)
{
u=0;
}
for(i=0;i<8;i++)
{
dispbuf[i]=16;
}
for(i=0;i<u;i++)
{
dispbuf[i]=8;
}
}
}
```

22. 电子琴

1. 实验任务

（1. 由 4X4 组成 16 个按钮矩阵，设计成 16 个音。

（2. 可随意弹奏想要表达的音乐。

2. 电路原理图

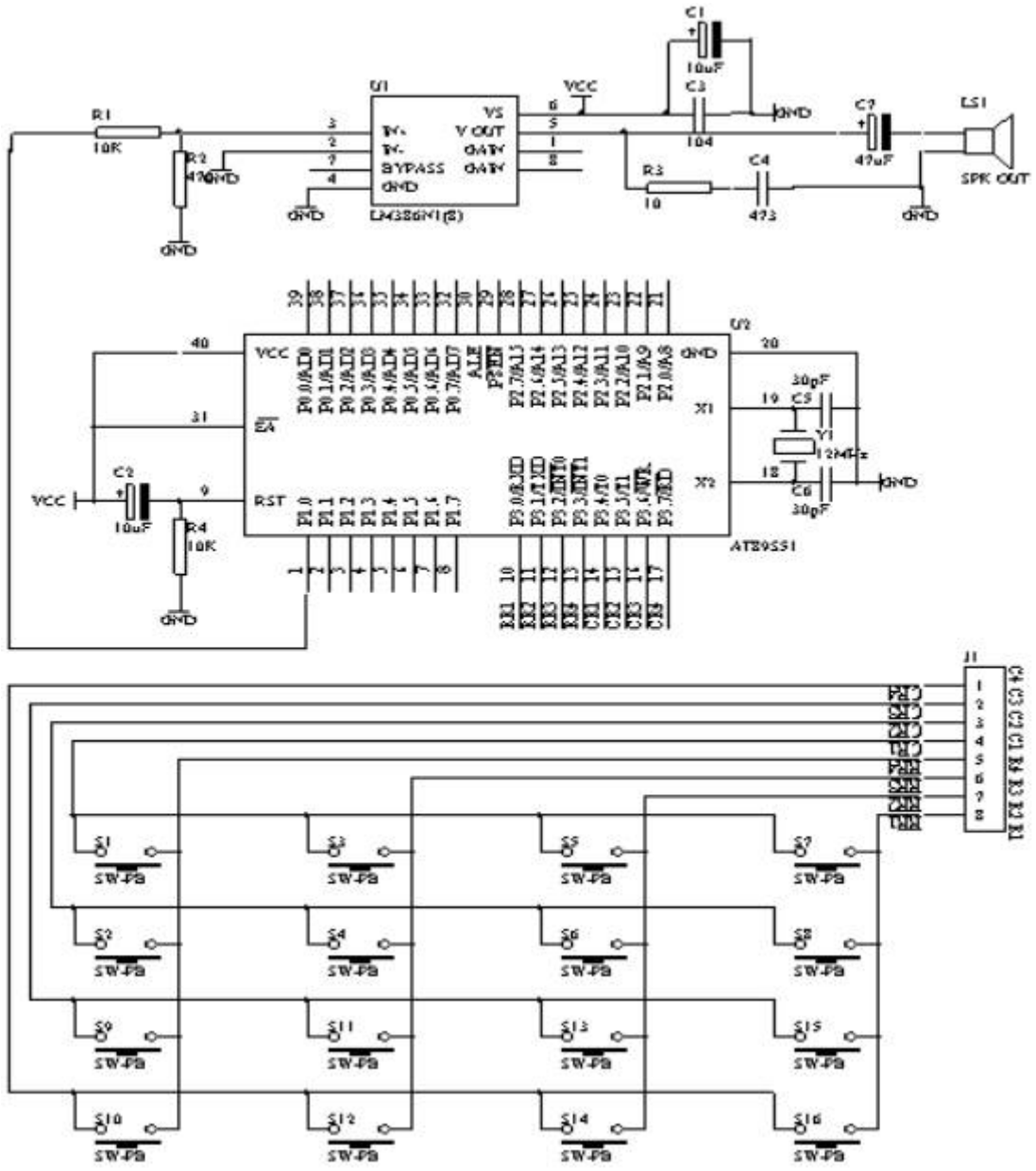


图 4.22.1

3. 系统板硬件连线

（1. 把“单片机系统”区域中的 P1.0 端口用导线连接到“音频放大模块”区域中的 SPK IN 端口上；

（2. 把“单片机系统“区域中的 P3.0－P3.7 端口用 8 芯排线连接到“4X4 行列式键盘”区域中的 C1－C4 R1－R4 端口上；

4. 相关程序内容

（1. 4X4 行列式键盘识别；

（2. 音乐产生的方法；

一首音乐是许多不同的音阶组成的，而每个音阶对应着不同的频率，这样我们就可以利用不同的频率的组合，即可构成我们所想要的音乐了，当然对于单片机来产生不同的频率非常方便，我们可以利用单片机的定时/计数器 T0 来产生这样方波频率信号，因此，我们只要把一首歌曲的音阶对应频率关系弄正确即可。现在以单片机 12MHZ 晶振为例，例出高中低音符与单片机计数 T0 相关的计数值如下表所示

音符	频率（HZ）	简谱码（T 值）	音符	频率（HZ）	简谱码（T 值）
低 1 DO	262	63628	# 4 FA#	740	64860
#1 DO#	277	63731	中 5 SO	784	64898
低 2 RE	294	63835	# 5 SO#	831	64934
#2 RE#	311	63928	中 6 LA	880	64968
低 3 M	330	64021	# 6	932	64994
低 4 FA	349	64103	中 7 SI	988	65030
# 4 FA#	370	64185	高 1 DO	1046	65058
低 5 SO	392	64260	# 1 DO#	1109	65085
# 5 SO#	415	64331	高 2 RE	1175	65110
低 6 LA	440	64400	# 2 RE#	1245	65134
# 6	466	64463	高 3 M	1318	65157
低 7 SI	494	64524	高 4 FA	1397	65178
中 1 DO	523	64580	# 4 FA#	1480	65198
# 1 DO#	554	64633	高 5 SO	1568	65217
中 2 RE	587	64684	# 5 SO#	1661	65235
# 2 RE#	622	64732	高 6 LA	1760	65252
中 3 M	659	64777	# 6	1865	65268
中 4 FA	698	64820	高 7 SI	1967	65283

下面我们要为这个音符建立一个表格，有助于单片机通过查表的方式来获得相应的数据

低音 0－19 之间，中音在 20－39 之间，高音在 40－59 之间

```
TABLE: DW 0,63628,63835,64021,64103,64260,64400,64524,0,0

DW 0,63731,63928,0,64185,64331,64463,0,0,0
```

DW 0,64580,64684,64777,64820,64898,64968,65030,0,0

DW 0,64633,64732,0,64860,64934,64994,0,0,0

DW 0,65058,65110,65157,65178,65217,65252,65283,0,0

DW 0,65085,65134,0,65198,65235,65268,0,0,0

DW 0

2、音乐的音拍，一个节拍为单位（C 调）

曲调值	DELAY	曲调值	DELAY
调 4/4	125ms	调 4/4	62ms
调 3/4	187ms	调 3/4	94ms
调 2/4	250ms	调 2/4	125ms

对于不同的曲调我们也可以用单片机的另外一个定时/计数器来完成。

下面就用 AT89S51 单片机产生一首“生日快乐”歌曲来说明单片机如何产生的。

在这个程序中用到了两个定时/计数器来完成的。其中 T0 用来产生音符频率，T1 用来产生音拍。

5. 程序框图

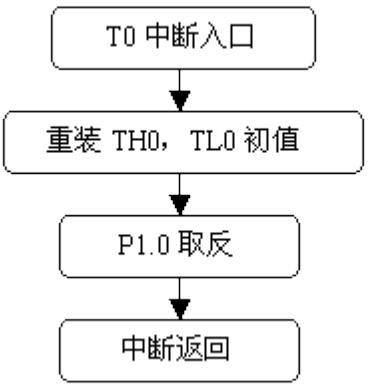


图 4.22.2

6. 汇编源程序

```
KEYBUF EQU 30H
STH0 EQU 31H
STL0 EQU 32H
TEMP EQU 33H
```

```
ORG 00H
LJMP START
ORG 0BH
LJMP INT_T0
START: MOV TMOD,#01H
SETB ET0
SETB EA
WAIT:
MOV P3,#0FFH
CLR P3.4
MOV A,P3
ANL A,#0FH
XRL A,#0FH
JZ NOKEY1
LCALL DELY10MS
MOV A,P3
ANL A,#0FH
XRL A,#0FH
JZ NOKEY1
MOV A,P3
ANL A,#0FH
CJNE A,#0EH,NK1
MOV KEYBUF,#0
LJMP DK1
NK1: CJNE A,#0DH,NK2
MOV KEYBUF,#1
LJMP DK1
NK2: CJNE A,#0BH,NK3
MOV KEYBUF,#2
LJMP DK1
NK3: CJNE A,#07H,NK4
MOV KEYBUF,#3
LJMP DK1
NK4: NOP
DK1:
MOV A,KEYBUF
MOV DPTR,#TABLE
MOVC A,@A+DPTR
MOV P0,A
MOV A,KEYBUF
MOV B,#2
MUL AB
MOV TEMP,A
MOV DPTR,#TABLE1
```

```
MOVC A,@A+DPTR
MOV STH0,A
MOV TH0,A
INC TEMP
MOV A,TEMP
MOVC A,@A+DPTR
MOV STL0,A
MOV TL0,A
SETB TR0

DK1A: MOV A,P3
ANL A,#0FH
XRL A,#0FH
JNZ DK1A
CLR TR0
NOKEY1:
MOV P3,#0FFH
CLR P3.5
MOV A,P3
ANL A,#0FH
XRL A,#0FH
JZ NOKEY2
LCALL DELY10MS
MOV A,P3
ANL A,#0FH
XRL A,#0FH
JZ NOKEY2
MOV A,P3
ANL A,#0FH
CJNE A,#0EH,NK5
MOV KEYBUF,#4
LJMP DK2
NK5: CJNE A,#0DH,NK6
MOV KEYBUF,#5
LJMP DK2
NK6: CJNE A,#0BH,NK7
MOV KEYBUF,#6
LJMP DK2
NK7: CJNE A,#07H,NK8
MOV KEYBUF,#7
LJMP DK2
NK8: NOP
DK2:
MOV A,KEYBUF
```

```
MOV DPTR,#TABLE
MOVC A,@A+DPTR
MOV P0,A
MOV A,KEYBUF
MOV B,#2
MUL AB
MOV TEMP,A
MOV DPTR,#TABLE1
MOVC A,@A+DPTR
MOV STH0,A
MOV TH0,A
INC TEMP
MOV A,TEMP
MOVC A,@A+DPTR
MOV STL0,A
MOV TL0,A
SETB TR0


DK2A: MOV A,P3
ANL A,#0FH
XRL A,#0FH
JNZ DK2A
CLR TR0
NOKEY2:
MOV P3,#0FFH
CLR P3.6
MOV A,P3
ANL A,#0FH
XRL A,#0FH
JZ NOKEY3
LCALL DELY10MS
MOV A,P3
ANL A,#0FH
XRL A,#0FH
JZ NOKEY3
MOV A,P3
ANL A,#0FH
CJNE A,#0EH,NK9
MOV KEYBUF,#8
LJMP DK3
NK9: CJNE A,#0DH,NK10
MOV KEYBUF,#9
LJMP DK3
```

```
NK10: CJNE A,#0BH,NK11
MOV KEYBUF,#10
LJMP DK3
NK11: CJNE A,#07H,NK12
MOV KEYBUF,#11
LJMP DK3
NK12: NOP
DK3:
MOV A,KEYBUF
MOV DPTR,#TABLE
MOVC A,@A+DPTR
MOV P0,A
MOV A,KEYBUF
MOV B,#2
MUL AB
MOV TEMP,A
MOV DPTR,#TABLE1
MOVC A,@A+DPTR
MOV STH0,A
MOV TH0,A
INC TEMP
MOV A,TEMP
MOVC A,@A+DPTR
MOV STL0,A
MOV TL0,A
SETB TR0

DK3A: MOV A,P3
ANL A,#0FH
XRL A,#0FH
JNZ DK3A
CLR TR0
NOKEY3:
MOV P3,#0FFH
CLR P3.7
MOV A,P3
ANL A,#0FH
XRL A,#0FH
JZ NOKEY4
LCALL DELY10MS
MOV A,P3
ANL A,#0FH
XRL A,#0FH
JZ NOKEY4
```

```
MOV A,P3
ANL A,#0FH
CJNE A,#0EH,NK13
MOV KEYBUF,#12
LJMP DK4
NK13: CJNE A,#0DH,NK14
MOV KEYBUF,#13
LJMP DK4
NK14: CJNE A,#0BH,NK15
MOV KEYBUF,#14
LJMP DK4
NK15: CJNE A,#07H,NK16
MOV KEYBUF,#15
LJMP DK4
NK16: NOP
DK4:
MOV A,KEYBUF
MOV DPTR,#TABLE
MOVC A,@A+DPTR
MOV P0,A
MOV A,KEYBUF
MOV B,#2
MUL AB
MOV TEMP,A
MOV DPTR,#TABLE1
MOVC A,@A+DPTR
MOV STH0,A
MOV TH0,A
INC TEMP
MOV A,TEMP
MOVC A,@A+DPTR
MOV STL0,A
MOV TL0,A
SETB TR0

DK4A: MOV A,P3
ANL A,#0FH
XRL A,#0FH
JNZ DK4A
CLR TR0
NOKEY4:
LJMP WAIT
DELY10MS:
MOV R6,#10
```

```
D1: MOV R7,#248
DJNZ R7,$
DJNZ R6,D1
RET
INT_T0:
MOV TH0,STH0
MOV TL0,STL0
CPL P1.0
RETI
TABLE: DB 3FH,06H,5BH,4FH,66H,6DH,7DH,07H
DB 7FH,6FH,77H,7CH,39H,5EH,79H,71H

TABLE1: DW 64021,64103,64260,64400
DW 64524,64580,64684,64777
DW 64820,64898,64968,65030
DW 65058,65110,65157,65178
END
```

7. C 语言源程序

```
#include <AT89X51.H>
unsigned char code table[]={0x3f,0x06,0x5b,0x4f,
0x66,0x6d,0x7d,0x07,
0x7f,0x6f,0x77,0x7c,
0x39,0x5e,0x79,0x71};
unsigned char temp;
unsigned char key;
unsigned char i,j;
unsigned char STH0;
unsigned char STL0;
unsigned int code tab[]={64021,64103,64260,64400,
64524,64580,64684,64777,
64820,64898,64968,65030,
65058,65110,65157,65178};

void main(void)
{
TMOD=0x01;
ET0=1;
EA=1;

while(1)
{
P3=0xff;
P3_4=0;
temp=P3;
```

```
temp=temp & 0x0f;
if (temp!=0x0f)
{
for(i=50;i>0;i--)
for(j=200;j>0;j--);
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
switch(temp)
{
case 0x0e:
key=0;
break;
case 0x0d:
key=1;
break;
case 0x0b:
key=2;
break;
case 0x07:
key=3;
break;
}
temp=P3;
P1_0=~P1_0;
P0=table[key];
STH0=tab[key]/256;
STL0=tab[key]%256;
TR0=1;
temp=temp & 0x0f;
while(temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
}
TR0=0;
}
}

P3=0xff;
P3_5=0;
```

```
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
for(i=50;i>0;i--)
for(j=200;j>0;j--);
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
switch(temp)
{
case 0x0e:
key=4;
break;
case 0x0d:
key=5;
break;
case 0x0b:
key=6;
break;
case 0x07:
key=7;
break;
}
temp=P3;
P1_0=~P1_0;
P0=table[key];
STH0=tab[key]/256;
STL0=tab[key]%256;
TR0=1;
temp=temp & 0x0f;
while(temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
}
TR0=0;
}
}

P3=0xff;
```

```
P3_6=0;
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
for(i=50;i>0;i--)
for(j=200;j>0;j--);
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
switch(temp)
{
case 0x0e:
key=8;
break;
case 0x0d:
key=9;
break;
case 0x0b:
key=10;
break;
case 0x07:
key=11;
break;
}
temp=P3;
P1_0=~P1_0;
P0=table[key];
STH0=tab[key]/256;
STL0=tab[key]%256;
TR0=1;
temp=temp & 0x0f;
while(temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
}
TR0=0;
}
}
```

```
P3=0xff;
P3_7=0;
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
for(i=50;i>0;i--)
for(j=200;j>0;j--);
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
switch(temp)
{
case 0x0e:
key=12;
break;
case 0x0d:
key=13;
break;
case 0x0b:
key=14;
break;
case 0x07:
key=15;
break;
}
temp=P3;
P1_0=~P1_0;
P0=table[key];
STH0=tab[key]/256;
STL0=tab[key]%256;
TR0=1;
temp=temp & 0x0f;
while(temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
}
TR0=0;
}
}
```

```
}
}

void t0(void) interrupt 1 using 0
{
TH0=STH0;
TL0=STL0;
P1_0=~P1_0;
}
```

23. 模拟计算器数字输入及显示

1. 实验任务

（1. 开机时，显示“0”

（2. 第一次按下时，显示“D1”；第二次按下时，显示“D1D2”；第三按下时，显示“D1D2D3”，8 个全显示完毕，再按下按键下时，给出“嘀”提示音。

2. 电路原理图

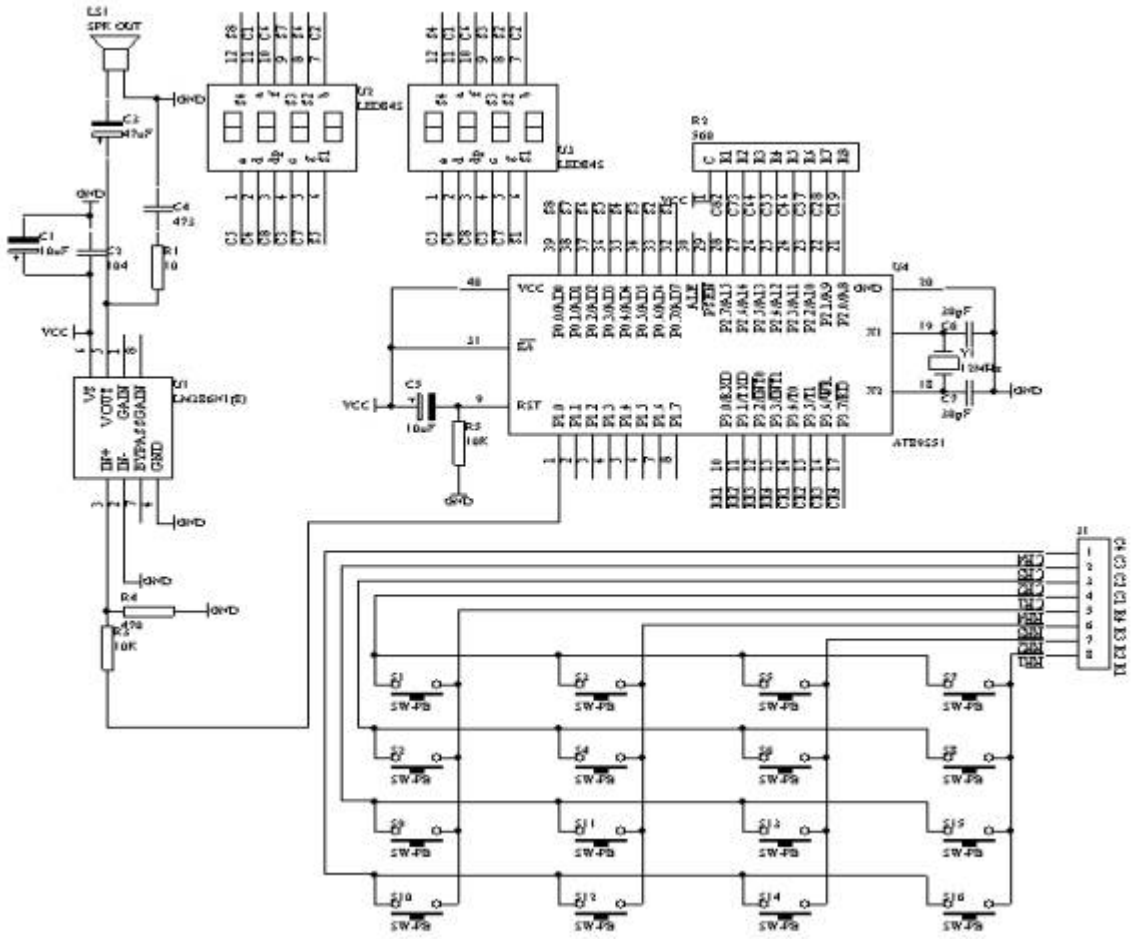


图 4.23.1

3. 系统板上硬件连线

（1. 把“单片机系统”区域中的 P1.0 端口用导线连接到“音频放大模块”区域中的 SPK IN 端口上；

（2. 把“单片机系统“区域中的 P3.0－P3.7 端口用 8 芯排线连接到“4X4 行列式键盘”区域中的 C1－C4 R1－R4 端口上；

（3. 把“单片机系统”区域中的 P0.0－P0.7 端口用 8 芯排线连接到“动态数码显示”区域中的 A－H 端口上；

（4. 把“单片机系统：区域中的 P2.0－P2.7 端口用 8 芯排线连接到“动态数码显示”区域中的 S1－S8 端口上；

4. 相关程序设计内容

（1. 行列式键盘输入及按键功能设定；

（2. 动态数码显示；

（3. 数码显示方式处理；

5. 汇编源程序

（略）

6. C 语言源程序

```
#include <AT89X51.H>

unsigned char code
dispcode[]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,
0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71,0x00};
unsigned char code
dispbitcode[]={0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f};
unsigned char dispbuf[8]={0,16,16,16,16,16,16,16};
unsigned char dispbitcount;
unsigned char temp;
unsigned char i,j;
unsigned char key;
unsigned char keypos;
bit alarmflag;

void change(unsigned char *p,unsigned char count)
{
while(count>0)
{
```

```
*(p+count)=*(p+count-1);
count--;
}
}

void main(void)
{
TMOD=0x01;
TH0=(65536-4000) / 256;
TL0=(65536-4000) % 256;
TR0=1;
ET0=1;
EA=1;

while(1)
{
P3=0xff;
P3_4=0;
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
for(i=50;i>0;i--)
for(j=200;j>0;j--);
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
switch(temp)
{
case 0x0e:
key=7;
break;
case 0x0d:
key=8;
break;
case 0x0b:
key=9;
break;
case 0x07:
key=10;
break;
```

```
}
if ((key>=0) && (key<10))
{
keypos++;
if(keypos<8)
{
change(dispbuf,keypos);
dispbuf[0]=key;
}
else
{
keypos=8;
alarmflag=1;
}
}
temp=P3;
P1_0=~P1_0;
temp=temp & 0x0f;
while(temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
}
alarmflag=0;
}
}

P3=0xff;
P3_5=0;
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
for(i=50;i>0;i--)
for(j=200;j>0;j--);
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
switch(temp)
{
case 0x0e:
```

```
key=4;
break;
case 0x0d:
key=5;
break;
case 0x0b:
key=6;
break;
case 0x07:
key=11;
break;
}
if ((key>=0) && (key<10))
{
keypos++;
if(keypos<8)
{
change(dispbuf,keypos);
dispbuf[0]=key;
}
else
{
keypos=8;
alarmflag=1;
}
}
temp=P3;
P1_0=~P1_0;
temp=temp & 0x0f;
while(temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
}
alarmflag=0;
}
}

P3=0xff;
P3_6=0;
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
```

```
for(i=50;i>0;i--)
for(j=200;j>0;j--);
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
switch(temp)
{
case 0x0e:
key=1;
break;
case 0x0d:
key=2;
break;
case 0x0b:
key=3;
break;
case 0x07:
key=12;
break;
}
if ((key>=0) && (key<10))
{
keypos++;
if(keypos<8)
{
change(dispbuf,keypos);
dispbuf[0]=key;
}
else
{
keypos=8;
alarmflag=1;
}
}
temp=P3;
P1_0=~P1_0;
temp=temp & 0x0f;
while(temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
```

```
}
alarmflag=0;
}
}

P3=0xff;
P3_7=0;
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
for(i=50;i>0;i--)
for(j=200;j>0;j--);
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
switch(temp)
{
case 0x0e:
key=0;
break;
case 0x0d:
key=13;
break;
case 0x0b:
key=14;
break;
case 0x07:
key=15;
break;
}
if ((key>=0) && (key<10))
{
keypos++;
if(keypos<8)
{
change(dispbuf,keypos);
dispbuf[0]=key;
}
else
{
```

```
keypos=8;
alarmflag=1;
}
}
temp=P3;
P1_0=~P1_0;
temp=temp & 0x0f;
while(temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
}
alarmflag=0;
}
}

}
}

void t0(void) interrupt 1 using 0
{
TH0=(65536-4000) / 256;
TL0=(65536-4000) % 256;
P0=dispcode[dispbuf[dispbitcount]];
P2=dispbitcode[dispbitcount];
dispbitcount++;
if (dispbitcount==8)
{
dispbitcount=0;
}
if (alarmflag==1)
{
P1_1=~P1_1;
}
}
```

24. 8X8 LED 点阵显示技术

1. 实验任务

在 8X8 LED 点阵上显示柱形，让其先从左到右平滑移动三次，其次从右到左平滑移动三次，再次从上到下平滑移动三次，最后从下到上平滑移动三次，如此循环下去。

2. 电路原理图

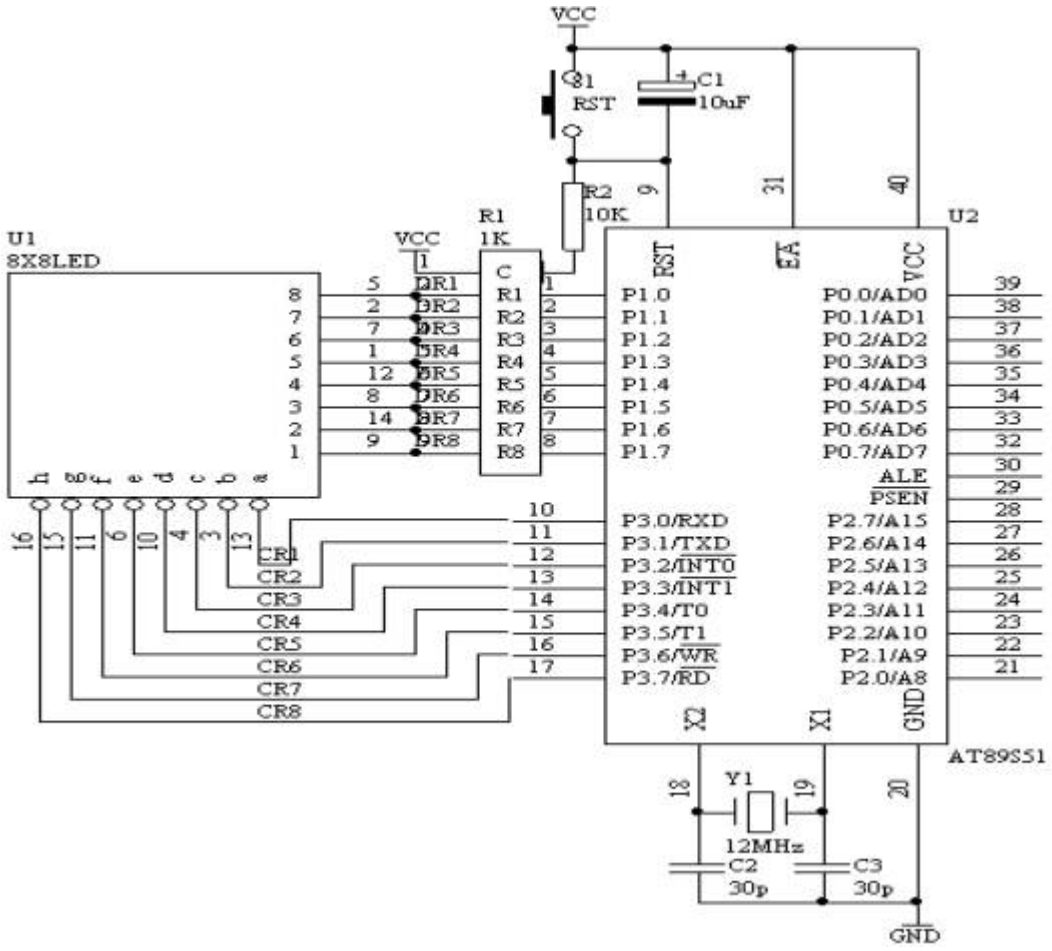


图 4.24.1

3. 硬件电路连线

(1). 把“单片机系统”区域中的 P1 端口用 8 芯排芯连接到“点阵模块”区域中的“DR1－DR8”端口上；

(2). 把“单片机系统”区域中的 P3 端口用 8 芯排芯连接到“点阵模块”区域中的“DC1－DC8”端口上；

4. 程序设计内容

(1). 8X8 点阵 LED 工作原理说明

8X8 点阵 LED 结构如下图所示

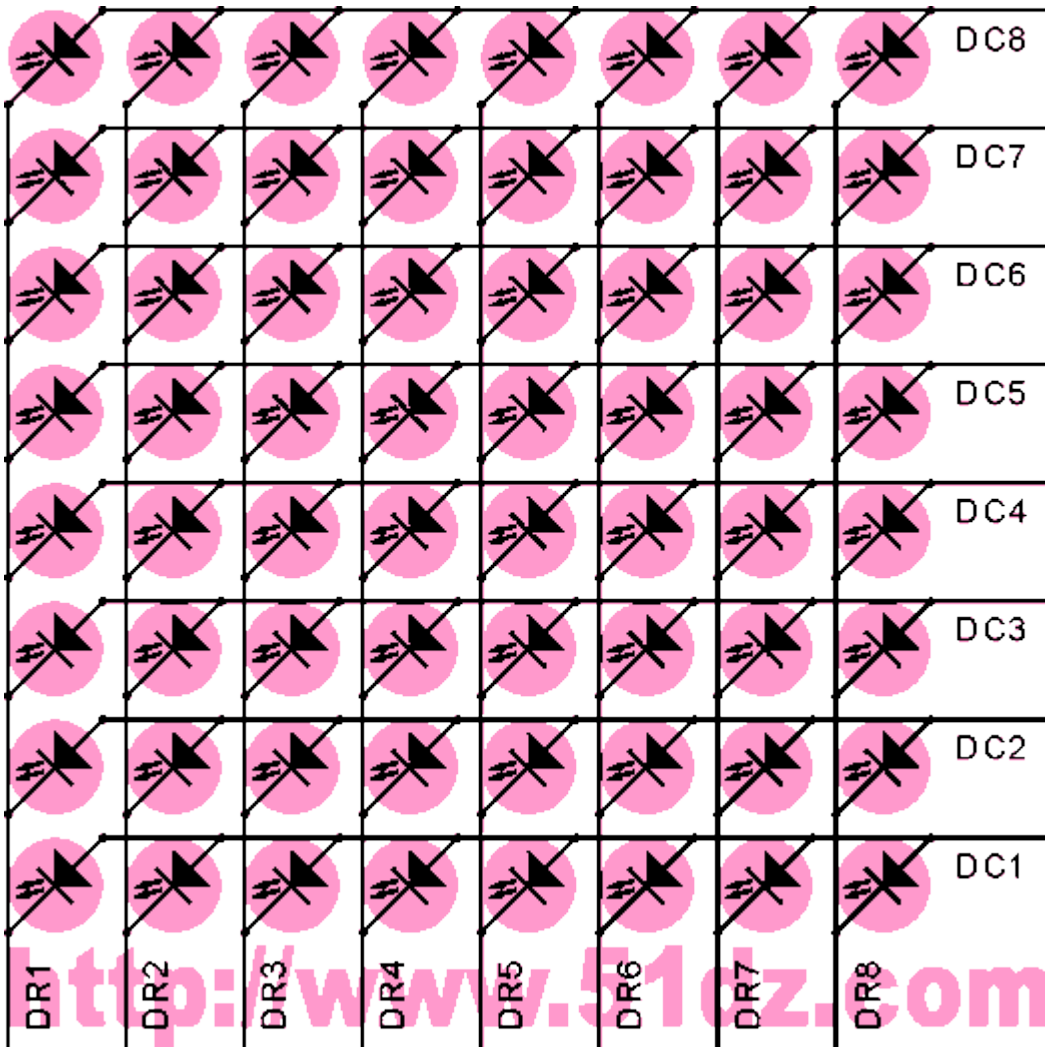


图 4.24.2

从图 4.24.2 中可以看出，8X8 点阵共需要 64 个发光二极管组成，且每个发光二极管是放置在行线和列线的交叉点上，当对应的某一列置 1 电平，某一行置 0 电平，则相应的二极管就亮；因此要实现一根柱形的亮法，如图 49 所示，对应的一列为一根竖柱，或者对应的一行为一根横柱，因此实现柱的亮的方法如下所述：

一根竖柱：对应的列置 1，而行则采用扫描的方法来实现。

一根横柱：对应的行置 0，而列则采用扫描的方法来实现。

5．汇编源程序

```
ORG 00H
START: NOP
MOV R3,#3
LOP2: MOV R4,#8
```

```
MOV R2,#0
LOP1: MOV P1,#0FFH
MOV DPTR,#TABA
MOV A,R2
MOVC A,@A+DPTR
MOV P3,A
INC R2
LCALL DELAY
DJNZ R4,LOP1
DJNZ R3,LOP2

MOV R3,#3
LOP4: MOV R4,#8
MOV R2,#7
LOP3: MOV P1,#0FFH
MOV DPTR,#TABA
MOV A,R2
MOVC A,@A+DPTR
MOV P3,A
DEC R2
LCALL DELAY
DJNZ R4,LOP3
DJNZ R3,LOP4

MOV R3,#3
LOP6: MOV R4,#8
MOV R2,#0
LOP5: MOV P3,#00H
MOV DPTR,#TABB
MOV A,R2
MOVC A,@A+DPTR
MOV P1,A
INC R2
LCALL DELAY
DJNZ R4,LOP5
DJNZ R3,LOP6

MOV R3,#3
LOP8: MOV R4,#8
MOV R2,#7
LOP7: MOV P3,#00H
MOV DPTR,#TABB
MOV A,R2
MOVC A,@A+DPTR
```

```
MOV P1,A
DEC R2
LCALL DELAY
DJNZ R4,LOP7
DJNZ R3,LOP8
LJMP START

DELAY: MOV R5,#10
D2: MOV R6,#20
D1: MOV R7,#248
DJNZ R7,$
DJNZ R6,D1
DJNZ R5,D2
RET

TABA: DB 0FEH,0FDH,0FBH,0F7H,0EFH,0DFH,0BFH,07FH
TABB: DB 01H,02H,04H,08H,10H,20H,40H,80H
END
```

6. C 语言源程序

```
#include <AT89X52.H>

unsigned char code taba[]={0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f};
unsigned char code tabb[]={0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};

void delay(void)
{
unsigned char i,j;

for(i=10;i>0;i--)
for(j=248;j>0;j--);
}

void delay1(void)
{
unsigned char i,j,k;

for(k=10;k>0;k--)
for(i=20;i>0;i--)
for(j=248;j>0;j--);

}
```

```
void main(void)
{
unsigned char i,j;

while(1)
{
for(j=0;j<3;j++)    //from left to right 3 time
{
for(i=0;i<8;i++)
{
P3=taba[i];
P1=0xff;
delay1();
}
}

for(j=0;j<3;j++)    //from right to left 3 time
{
for(i=0;i<8;i++)
{
P3=taba[7-i];
P1=0xff;
delay1();
}
}

for(j=0;j<3;j++)    //from top to bottom 3 time
{
for(i=0;i<8;i++)
{
P3=0x00;
P1=tabb[7-i];
delay1();
}
}

for(j=0;j<3;j++)    //from bottom to top 3 time
{
for(i=0;i<8;i++)
{
P3=0x00;
P1=tabb[i];
delay1();
}
```

```
}

}
}
```

25. 点阵式 LED“0－9”数字显示技术

1. 实验任务

利用 8X8 点阵显示数字 0 到 9 的数字。

2. 电路原理图

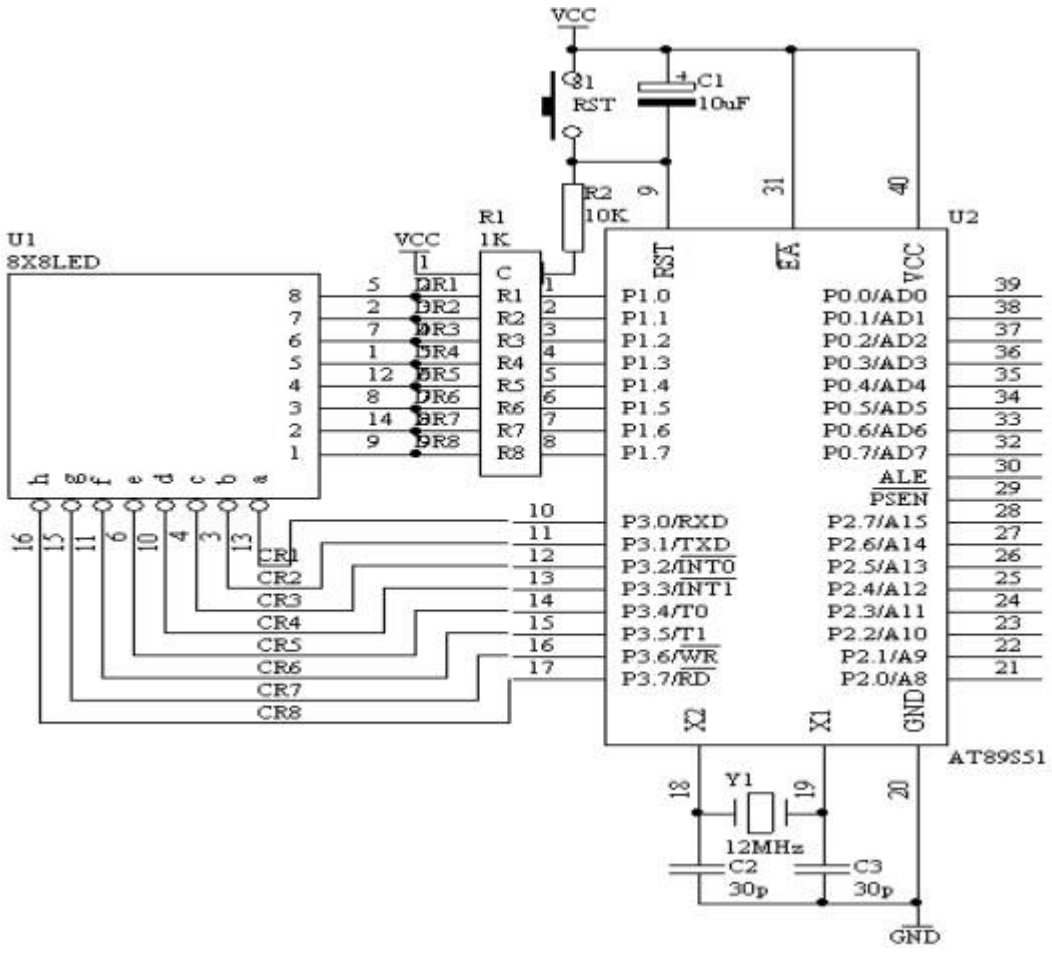


图 4.25.1

3. 硬件系统连线

(1). 把“单片机系统”区域中的 P1 端口用 8 芯排芯连接到“点阵模块”区域中的“DR1－DR8”端口上；

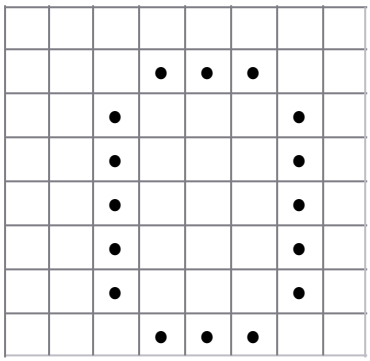
(2). 把“单片机系统”区域中的 P3 端口用 8 芯排芯连接到“点阵模块”区域中的“DC1－DC8”端口上；

4. 程序设计内容

(1). 数字 0－9 点阵显示代码的形成

如下图所示，假设显示数字“0”

1 2 3 4 5 6 7 8



00 00 3E 41 41 41 3E 00

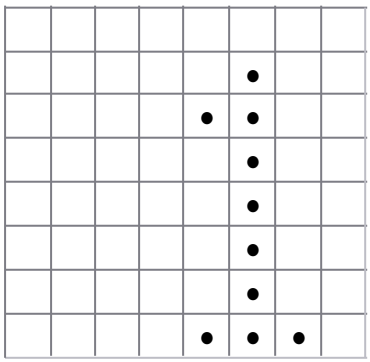
因此，形成的列代码为 00H，00H，3EH，41H，41H，3EH，00H，00H；只要把这些代码分别送到相应的列线上面，即可实现“0”的数字显示。

送显示代码过程如下所示

送第一列线代码到 P3 端口，同时置第一行线为“0”，其它行线为“1”，延时 2ms 左右，送第二列线代码到 P3 端口，同时置第二行线为“0”，其它行线为“1”，延时 2ms 左右，如此下去，直到送完最后一列代码，又从头开始送。

数字“1”代码建立如下图所示

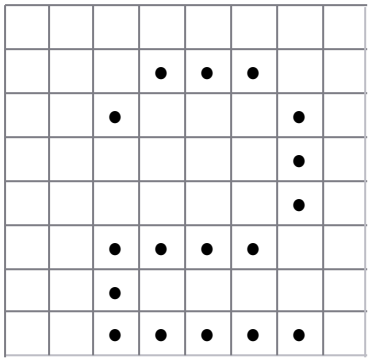
1 2 3 4 5 6 7 8



其显示代码为 00H，00H，00H，00H，21H，7FH，01H，00H

数字“2”代码建立如下图所示

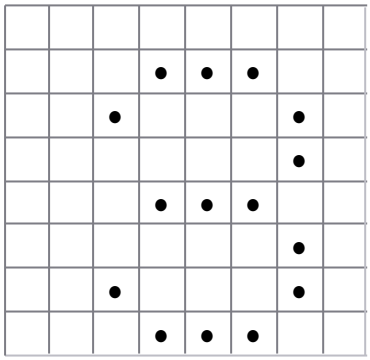
1 2 3 4 5 6 7 8



00H，00H，27H，45H，45H，45H，39H，00H

数字“3”代码建立如下图所示

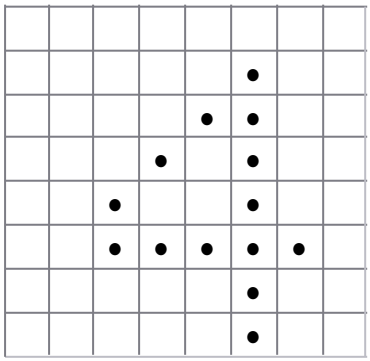
1 2 3 4 5 6 7 8



00H，00H，22H，49H，49H，49H，36H，00H

数字“4”代码建立如下图所示

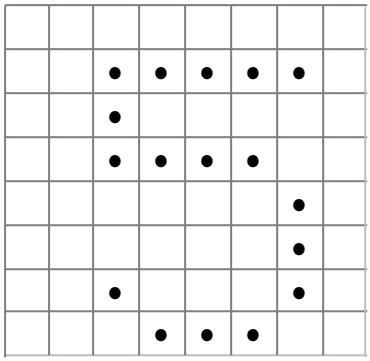
1 2 3 4 5 6 7 8



00H，00H，0CH，14H，24H，7FH，04H，00H

数字“5”代码建立如下图所示

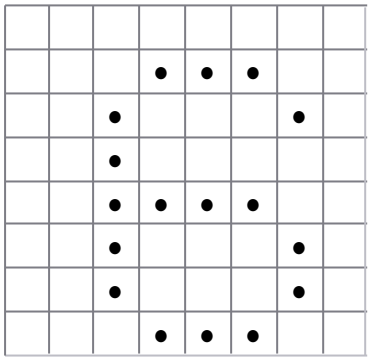
1 2 3 4 5 6 7 8



00H，00H，72H，51H，51H，51H，4EH，00H

数字“6”代码建立如下图所示

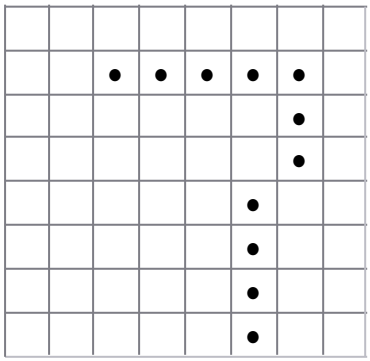
1 2 3 4 5 6 7 8



00H，00H，3EH，49H，49H，49H，26H，00H

数字“7”代码建立如下图所示

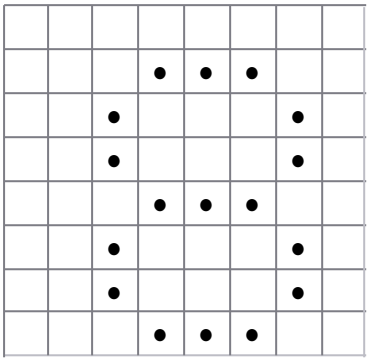
1 2 3 4 5 6 7 8



00H，00H，40H，40H，40H，4FH，70H，00H

数字“8”代码建立如下图所示

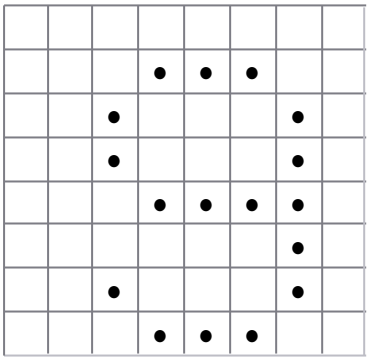
1 2 3 4 5 6 7 8



00H，00H，36H，49H，49H，49H，36H，00H

数字“9”代码建立如下图所示

1 2 3 4 5 6 7 8



00H，00H，32H，49H，49H，49H，3EH，00H

5. 汇编源程序

```
TIM EQU 30H
CNTA EQU 31H
CNTB EQU 32H

ORG 00H
LJMP START
ORG 0BH
LJMP TOX
ORG 30H
START: MOV TIM,#00H
MOV CNTA,#00H
MOV CNTB,#00H
MOV TMOD,#01H
```

```
MOV TH0,#(65536-4000)/256
MOV TL0,#(65536-4000) MOD 256
SETB TR0
SETB ET0
SETB EA
SJMP $

TOX:
MOV TH0,#(65536-4000)/256
MOV TL0,#(65536-4000) MOD 256
MOV DPTR,#TAB
MOV A,CNTA
MOVC A,@A+DPTR
MOV P3,A
MOV DPTR,#DIGIT
MOV A,CNTB
MOV B,#8
MUL AB
ADD A,CNTA
MOVC A,@A+DPTR
MOV P1,A
INC CNTA
MOV A,CNTA
CJNE A,#8,NEXT
MOV CNTA,#00H
NEXT: INC TIM
MOV A,TIM
CJNE A,#250,NEX
MOV TIM,#00H
INC CNTB
MOV A,CNTB
CJNE A,#10,NEX
MOV CNTB,#00H
NEX: RETI

TAB: DB 0FEH,0FDH,0FBH,0F7H,0EFH,0DFH,0BFH,07FH
DIGIT: DB 00H,00H,3EH,41H,41H,41H,3EH,00H
DB 00H,00H,00H,00H,21H,7FH,01H,00H
DB 00H,00H,27H,45H,45H,45H,39H,00H
DB 00H,00H,22H,49H,49H,49H,36H,00H
DB 00H,00H,0CH,14H,24H,7FH,04H,00H
DB 00H,00H,72H,51H,51H,51H,4EH,00H
DB 00H,00H,3EH,49H,49H,49H,26H,00H
DB 00H,00H,40H,40H,40H,4FH,70H,00H
```

```
DB 00H,00H,36H,49H,49H,49H,36H,00H
DB 00H,00H,32H,49H,49H,49H,3EH,00H
END
```

6. C 语言源程序

```
#include <AT89X52.H>
unsigned char code tab[]={0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f};
unsigned char code digittab[10][8]={
{0x00,0x00,0x3e,0x41,0x41,0x41,0x3e,0x00}, //0
{0x00,0x00,0x00,0x00,0x21,0x7f,0x01,0x00}, //1
{0x00,0x00,0x27,0x45,0x45,0x45,0x39,0x00}, //2
{0x00,0x00,0x22,0x49,0x49,0x49,0x36,0x00}, //3
{0x00,0x00,0x0c,0x14,0x24,0x7f,0x04,0x00}, //4
{0x00,0x00,0x72,0x51,0x51,0x51,0x4e,0x00}, //5
{0x00,0x00,0x3e,0x49,0x49,0x49,0x26,0x00}, //6
{0x00,0x00,0x40,0x40,0x40,0x4f,0x70,0x00}, //7
{0x00,0x00,0x36,0x49,0x49,0x49,0x36,0x00}, //8
{0x00,0x00,0x32,0x49,0x49,0x49,0x3e,0x00} //9
};

unsigned int timecount;
unsigned char cnta;
unsigned char cntb;

void main(void)
{
TMOD=0x01;
TH0=(65536-3000)/256;
TL0=(65536-3000)%256;
TR0=1;
ET0=1;
EA=1;
while(1)
{;
}
}

void t0(void) interrupt 1 using 0
{
TH0=(65536-3000)/256;
TL0=(65536-3000)%256;
P3=tab[cnta];
P1=digittab[cntb][cnta];
cnta++;
```

```
if(cnta==8)
{
cnta=0;
}
timecount++;
if(timecount==333)
{
timecount=0;
cntb++;
if(cntb==10)
{
cntb=0;
}
}
}
```

26. 点阵式 LED 简单图形显示技术

1. 实验任务

在 8X8 点阵式 LED 显示“★”、“●”和心形图，通过按键来选择要显示的图形。

2. 电路原理图

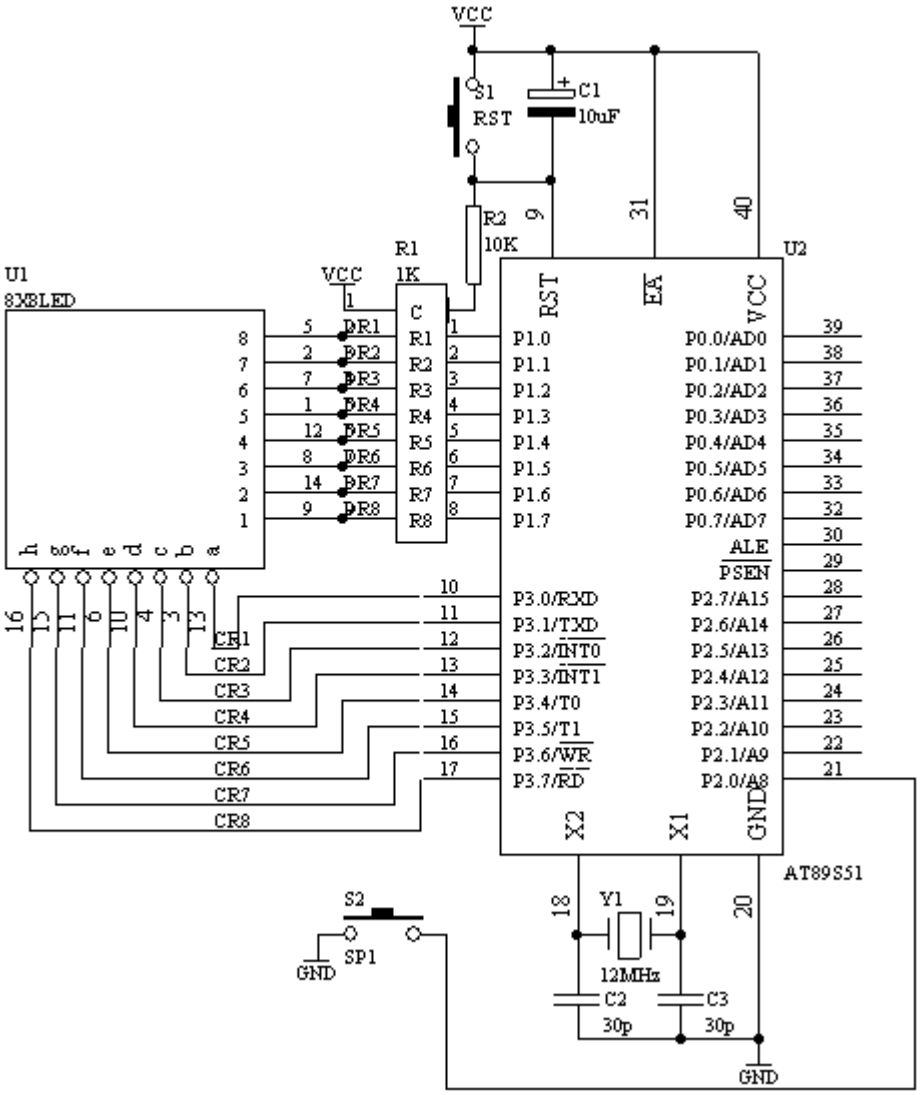


图 4.26.1

3. 硬件系统连线

(1). 把“单片机系统”区域中的 P1 端口用 8 芯排芯连接到“点阵模块”区域中的“DR1－DR8”端口上；

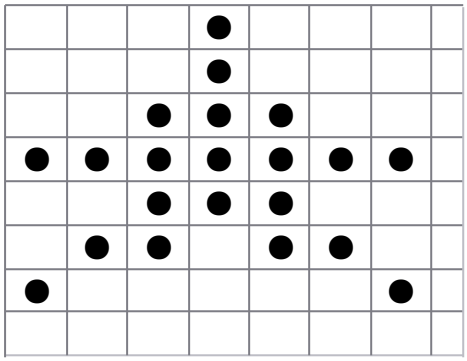
(2). 把“单片机系统”区域中的 P3 端口用 8 芯排芯连接到“点阵模块”区域中的“DC1－DC8”端口上；

(3). 把“单片机系统”区域中的 P2.0/A8 端子用导线连接到“独立式键盘”区域中的 SP1 端子上；

4. 程序设计内容

(1). “★”在 8X8LED 点阵上显示图如下图所示

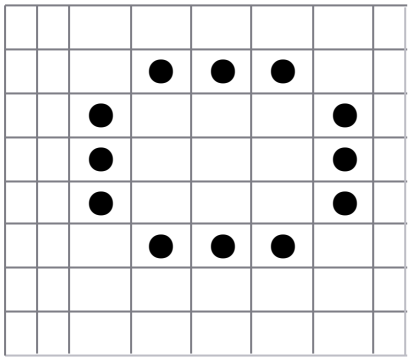
1 2 3 4 5 6 7 8



12H，14H，3CH，48H，3CH，14H，12H，00H

(2). “●”在 8X8LED 点阵上显示图如下图所示

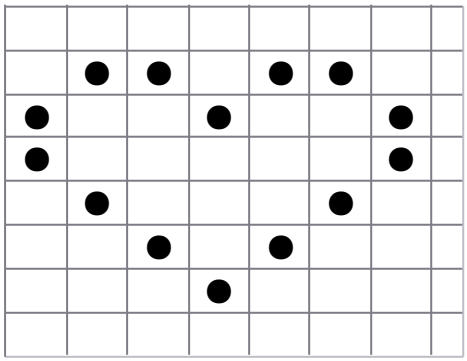
1 2 3 4 5 6 7 8



00H，00H，38H，44H，44H，44H，38H，00H

(3). 心形图在 8X8LED 点阵上显示图如下图所示

1 2 3 4 5 6 7 8



30H，48H，44H，22H，44H，48H，30H，00H

5. 汇编源程序

```
CNTA EQU 30H
COUNT EQU 31H

ORG 00H
LJMP START
ORG 0BH
LJMP T0X
ORG 30H
START: MOV CNTA,#00H
MOV COUNT,#00H
MOV TMOD,#01H
MOV TH0,#(65536-4000) / 256
MOV TL0,#(65536-4000) MOD 256
SETB TR0
SETB ET0
SETB EA
WT: JB P2.0,WT
MOV R6,#5
MOV R7,#248
D1: DJNZ R7,$
DJNZ R6,D1
JB P2.0,WT
INC COUNT
MOV A,COUNT
CJNE A,#03H,NEXT
MOV COUNT,#00H
NEXT: JNB P2.0,$
SJMP WT

T0X: NOP
MOV TH0,#(65536-4000) / 256
MOV TL0,#(65536-4000) MOD 256
MOV DPTR,#TAB
MOV A,CNTA
MOVC A,@A+DPTR
MOV P3,A
MOV DPTR,#GRAPH
MOV A,COUNT
MOV B,#8
MUL AB
ADD A,CNTA
MOVC A,@A+DPTR
MOV P1,A
```

```
INC CNTA
MOV A,CNTA
CJNE A,#8,NEX
MOV CNTA,#00H
NEX: RETI

TAB: DB 0FEH,0FDH,0FBH,0F7H,0EFH,0DFH,0BFH,07FH
GRAPH: DB 12H,14H,3CH,48H,3CH,14H,12H,00H
DB 00H,00H,38H,44H,44H,44H,38H,00H
DB 30H,48H,44H,22H,44H,48H,30H,00H
END
```

6. C 语言源程序

```
#include <AT89X52.H>

unsigned char code tab[]={0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f};
unsigned char code
graph[3][8]={{0x12,0x14,0x3c,0x48,0x3c,0x14,0x12,0x00},
{0x00,0x00,0x38,0x44,0x44,0x44,0x38,0x00},
{0x30,0x48,0x44,0x22,0x44,0x48,0x30,0x00}
};
unsigned char count;
unsigned char cnta;

void main(void)
{
unsigned char i,j;

TMOD=0x01;
TH0=(65536-4000)/256;
TL0=(65536-4000)%256;
TR0=1;
ET0=1;
EA=1;

while(1)
{
if(P2_0==0)
{
for(i=5;i>0;i--)
for(j=248;j>0;j--);
if(P2_0==0)
{
count++;
```

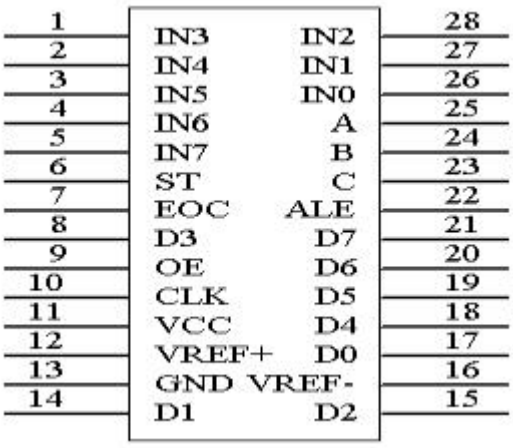
```
if(count==3)
{
count=0;
}
while(P2_0==0);
}
}
}
}

void t0(void) interrupt 1 using 0
{
TH0=(65536-4000)/256;
TL0=(65536-4000)%256;
P3=tab[cnta];
P1=graph[count][cnta];
cnta++;
if(cnta==8)
{
cnta=0;
}
}
```

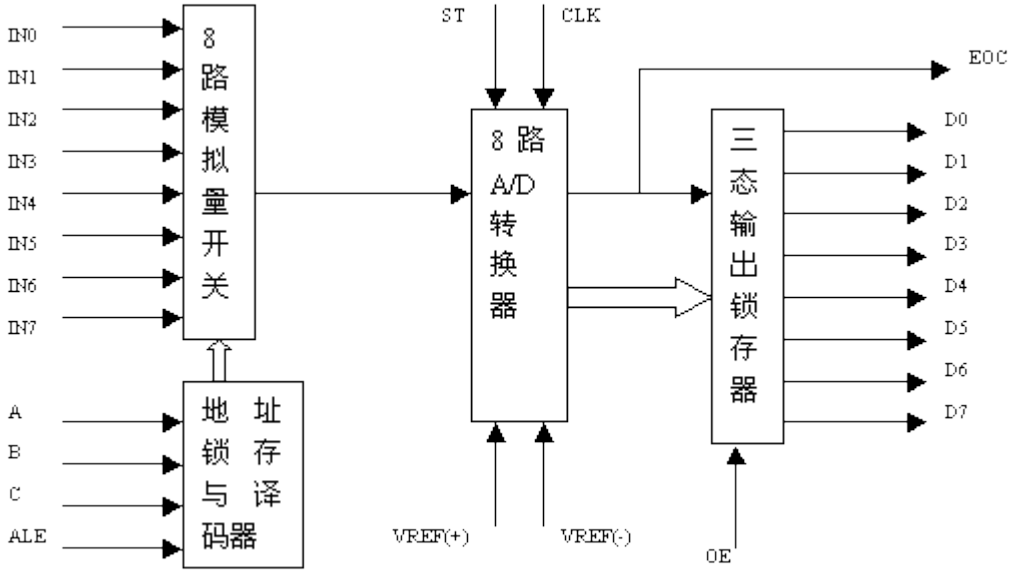
27. ADC0809A/D 转换器基本应用技术

1. 基本知识

ADC0809 是带有 8 位 A/D 转换器、8 路多路开关以及微处理机兼容的控制逻辑的 CMOS 组件。它是逐次逼近式 A/D 转换器，可以和单片机直接接口。

（1）. ADC0809 的内部逻辑结构





由上图可知，ADC0809 由一个 8 路模拟开关、一个地址锁存与译码器、一个 A/D 转换器和一个三态输出锁存器组成。多路开关可选通 8 个模拟通道，允许 8 路模拟量分时输入，共用 A/D 转换器进行转换。三态输出锁器用于锁存 A/D 转换完的数字量，当 OE 端为高电平时，才可以从三态输出锁存器取走转换完的数据。

（2）． 引脚结构

IN0－IN7：8 条模拟量输入通道

ADC0809 对输入模拟量要求：信号单极性，电压范围是 0－5V，若信号太小，必须进行放大；输入的模拟量在转换过程中应该保持不变，如若模拟量变化太快，则需在输入前增加采样保持电路。

地址输入和控制线：4 条

ALE 为地址锁存允许输入线，高电平有效。当 ALE 线为高电平时，地址锁存与译码器将 A，B，C 三条地址线的地址信号进行锁存，经译码后被选中的通道的模拟量进转换器进行转换。A，B 和 C 为地址输入线，用于选通 IN0－IN7 上的一路模拟量输入。通道选择表如下表所示。

C	B	A	选择的通道
0	0	0	IN0
0	0	1	IN1
0	1	0	IN2
0	1	1	IN3
1	0	0	IN4
1	0	1	IN5
1	1	0	IN6

1	1	1	IN7

数字量输出及控制线：11 条

ST 为转换启动信号。当 ST 上跳沿时，所有内部寄存器清零；下跳沿时，开始进行 A/D 转换；在转换期间，ST 应保持低电平。EOC 为转换结束信号。当 EOC 为高电平时，表明转换结束；否则，表明正在进行 A/D 转换。OE 为输出允许信号，用于控制三条输出锁存器向单片机输出转换得到的数据。OE＝1，输出转换得到的数据；OE＝0，输出数据线呈高阻状态。D7－D0 为数字量输出线。

CLK 为时钟输入信号线。因 ADC0809 的内部没有时钟电路，所需时钟信号必须由外界提供，通常使用频率为 500KHZ，

VREF（＋），VREF（－）为参考电压输入。

2. ADC0809 应用说明

（1）. ADC0809 内部带有输出锁存器，可以与 AT89S51 单片机直接相连。

（2）. 初始化时，使 ST 和 OE 信号全为低电平。

（3）. 送要转换的哪一通道的地址到 A，B，C 端口上。

（4）. 在 ST 端给出一个至少有 100ns 宽的正脉冲信号。

（5）. 是否转换完毕，我们根据 EOC 信号来判断。

（6）. 当 EOC 变为高电平时，这时给 OE 为高电平，转换的数据就输出给单片机了。

3. 实验任务

如下图所示，从 ADC0809 的通道 IN3 输入 0－5V 之间的模拟量，通过 ADC0809 转换成数字量在数码管上以十进制形成显示出来。ADC0809 的 VREF 接＋5V 电压。

4. 电路原理图

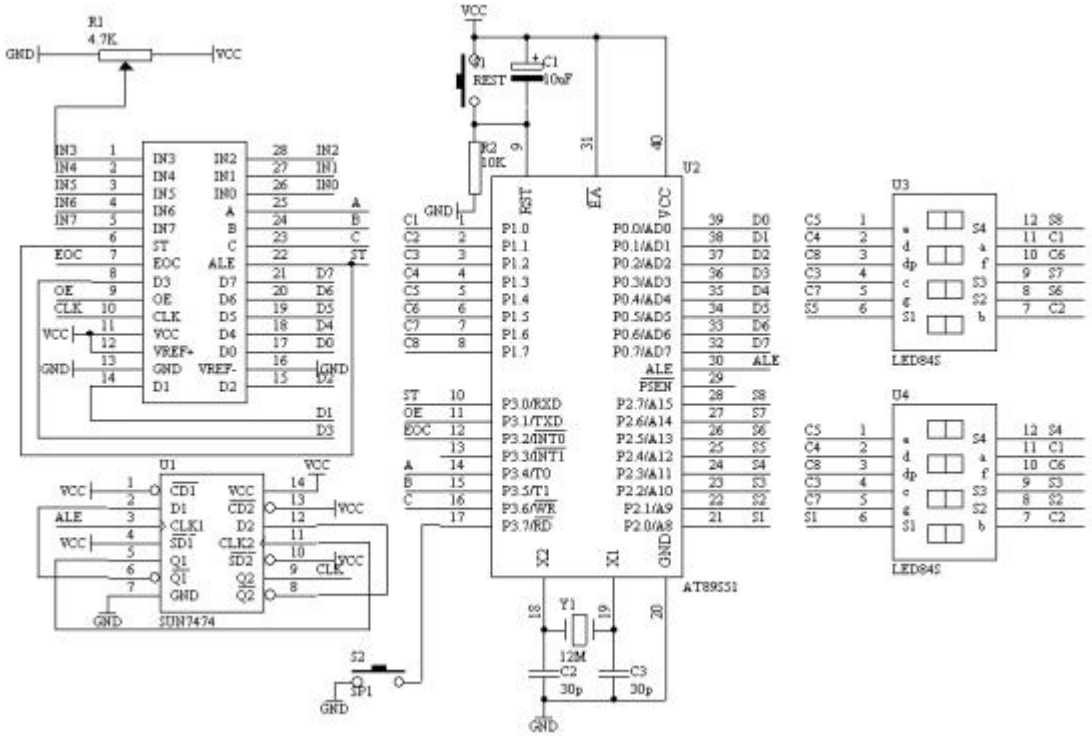


图 1.27.1

5. 系统板上硬件连线

（1）. 把“单片机系统板”区域中的 P1 端口的 P1.0—P1.7 用 8 芯排线连接到“动态数码显示”区域中的 A B C D E F G H 端口上，作为数码管的笔段驱动。

（2）. 把“单片机系统板”区域中的 P2 端口的 P2.0—P2.7 用 8 芯排线连接到“动态数码显示”区域中的 S1 S2 S3 S4 S5 S6 S7 S8 端口上，作为数码管的位段选择。

（3）. 把“单片机系统板”区域中的 P0 端口的 P0.0—P0.7 用 8 芯排线连接到“模数转换模块”区域中的 D0D1D2D3D4D5D6D7 端口上，A/D 转换完毕的数据输入到单片机的 P0 端口

（4）. 把“模数转换模块”区域中的 VREF 端子用导线连接到“电源模块”区域中的 VCC 端子上；

（5）. 把“模数转换模块”区域中的 A2A1A0 端子用导线连接到“单片机系统”区域中的 P3.4 P3.5 P3.6 端子上；

（6）. 把“模数转换模块”区域中的 ST 端子用导线连接到“单片机系统”区域中的 P3.0 端子上；

（7）. 把“模数转换模块”区域中的 OE 端子用导线连接到“单片机系统”区域中的 P3.1 端子上；

（8）. 把“模数转换模块”区域中的 EOC 端子用导线连接到“单片机系统”区域中的 P3.2 端子上；

（9）. 把“模数转换模块”区域中的 CLK 端子用导线连接到“分频模块”区域中的 /4 端子上；

（10）. 把“分频模块”区域中的 CK IN 端子用导线连接到“单片机系统”区域中的 ALE 端子上；

（11）. 把“模数转换模块”区域中的 IN3 端子用导线连接到“三路可调压模块”区域中的 VR1 端子上；

6. 程序设计内容

（1）. 进行 A/D 转换时，采用查询 EOC 的标志信号来检测 A/D 转换是否完毕，若完毕则把数据通过 P0 端口读入，经过数据处理之后在数码管上显示。

（2）. 进行 A/D 转换之前，要启动转换的方法：

ABC＝110 选择第三通道

ST＝0，ST＝1，ST＝0 产生启动转换的正脉冲信号

7. 汇编源程序

```
CH EQU 30H
DPCNT EQU 31H
DPBUF EQU 33H
GDATA EQU 32H
ST BIT P3.0
OE BIT P3.1
EOC BIT P3.2

ORG 00H
LJMP START
ORG 0BH
LJMP T0X
ORG 30H
START: MOV CH,#0BCH
MOV DPCNT,#00H
MOV R1,#DPCNT
MOV R7,#5
```

```
MOV A,#10
MOV R0,#DPBUF
LOP: MOV @R0,A
INC R0
DJNZ R7,LOP
MOV @R0,#00H
INC R0
MOV @R0,#00H
INC R0
MOV @R0,#00H
MOV TMOD,#01H
MOV TH0,#(65536-4000)/256
MOV TL0,#(65536-4000) MOD 256
SETB TR0
SETB ET0
SETB EA
WT: CLR ST
SETB ST
CLR ST
WAIT: JNB EOC,WAIT
SETB OE
MOV GDATA,P0
CLR OE
MOV A,GDATA
MOV B,#100
DIV AB
MOV 33H,A
MOV A,B
MOV B,#10
DIV AB
MOV 34H,A
MOV 35H,B
SJMP WT
TOX: NOP
MOV TH0,#(65536-4000)/256
MOV TL0,#(65536-4000) MOD 256
MOV DPTR,#DPCD
MOV A,DPCNT
ADD A,#DPBUF
MOV R0,A
MOV A,@R0
MOVC A,@A+DPTR
MOV P1,A
MOV DPTR,#DPBT
```

```
MOV A,DPCNT
MOVC A,@A+DPTR
MOV P2,A
INC DPCNT
MOV A,DPCNT
CJNE A,#8,NEXT
MOV DPCNT,#00H
NEXT: RETI
DPCD: DB 3FH,06H,5BH,4FH,66H
DB 6DH,7DH,07H,7FH,6FH,00H
DPBT: DB 0FEH,0FDH,0FBH,0F7H
DB 0EFH,0DFH,0BFH,07FH
END
```

8. C 语言源程序

```
#include <AT89X52.H>
unsigned char code dispbitcode[]={0xfe,0xfd,0xfb,0xf7,
  0xef,0xdf,0xbf,0x7f};
unsigned char code dispcode[]={0x3f,0x06,0x5b,0x4f,0x66,
0x6d,0x7d,0x07,0x7f,0x6f,0x00};
unsigned char dispbuf[8]={10,10,10,10,10,0,0,0};
unsigned char dispcount;

sbit ST=P3^0;
sbit OE=P3^1;
sbit EOC=P3^2;
unsigned char channel=0xbc;//IN3
unsigned char getdata;

void main(void)
{
TMOD=0x01;
TH0=(65536-4000)/256;
TL0=(65536-4000)%256;
TR0=1;
ET0=1;
EA=1;

P3=channel;

while(1)
{
ST=0;
ST=1;
```

```
ST=0;
while(EOC==0);
OE=1;
getdata=P0;
OE=0;
dispbuf[2]=getdata/100;
getdata=getdata%10;
dispbuf[1]=getdata/10;
dispbuf[0]=getdata%10;
}
}

void t0(void) interrupt 1 using 0
{
TH0=(65536-4000)/256;
TL0=(65536-4000)%256;
P1=dispcode[dispbuf[dispcount]];
P2=dispbitcode[dispcount];
dispcount++;
if(dispcount==8)
{
dispcount=0;
}
}
```

28. 数字电压表

1. 实验任务

利用单片机 AT89S51 与 ADC0809 设计一个数字电压表，能够测量 0－5V 之间的直流电压值，四位数码显示，但要求使用的元器件数目最少。

2. 电路原理图

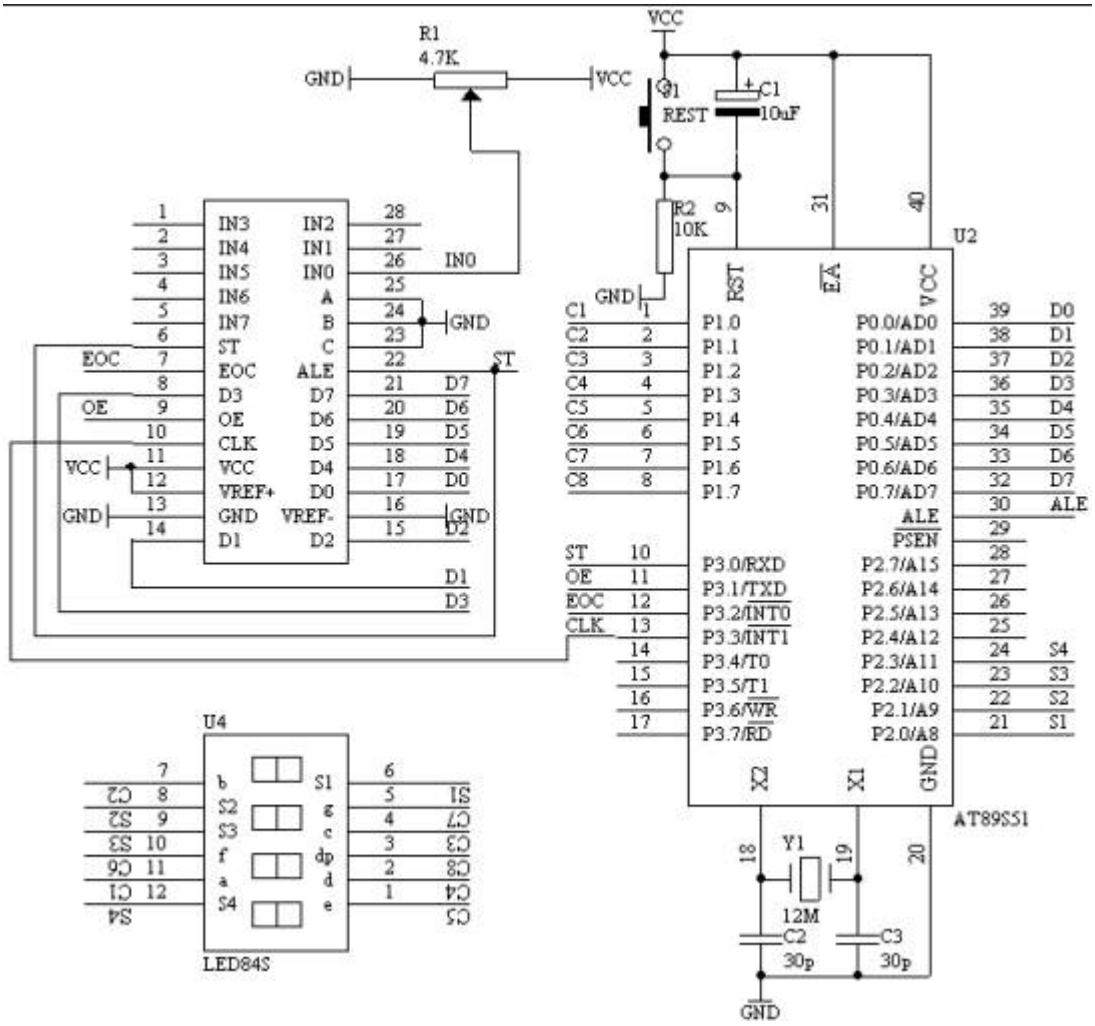


图 1.28.1

3. 系统板上硬件连线

a) 把“单片机系统”区域中的 P1.0－P1.7 与“动态数码显示”区域中的 ABCDEFGH 端口用 8 芯排线连接。

b) 把“单片机系统”区域中的 P2.0－P2.7 与“动态数码显示”区域中的 S1S2S3S4S5S6S7S8 端口用 8 芯排线连接。

c) 把“单片机系统”区域中的 P3.0 与“模数转换模块”区域中的 ST 端子用导线相连接。

d) 把“单片机系统”区域中的 P3.1 与“模数转换模块”区域中的 OE 端子用导线相连接。

e) 把“单片机系统”区域中的 P3.2 与“模数转换模块”区域中的 EOC 端子用导线相连接。

f) 把“单片机系统”区域中的P3.3与“模数转换模块”区域中的CLK端子用导线相连接。

g) 把“模数转换模块”区域中的A2A1A0端子用导线连接到“电源模块”区域中的GND端子上。

h) 把“模数转换模块”区域中的IN0端子用导线连接到“三路可调电压模块”区域中的VR1端子上。

i) 把“单片机系统”区域中的P0.0－P0.7用8芯排线连接到“模数转换模块”区域中的D0D1D2D3D4D5D6D7端子上。

4. 程序设计内容

i. 由于ADC0809在进行A/D转换时需要有CLK信号，而此时的ADC0809的CLK是接在AT89S51单片机的P3.3端口上，也就是要求从P3.3输出CLK信号供ADC0809使用。因此产生CLK信号的方法就得用软件来产生了。

ii. 由于ADC0809的参考电压VREF＝VCC，所以转换之后的数据要经过数据处理，在数码管上显示出电压值。实际显示的电压值(D/256*VREF)

5. 汇编源程序

（略）

6. C语言源程序

```
#include <AT89X52.H>

unsigned char code dispbitcode[]={0xfe,0xfd,0xfb,0xf7,
0xef,0xdf,0xbf,0x7f};
unsigned char code dispcode[]={0x3f,0x06,0x5b,0x4f,0x66,
0x6d,0x7d,0x07,0x7f,0x6f,0x00};
unsigned char dispbuf[8]={10,10,10,10,0,0,0,0};
unsigned char dispcount;
unsigned char getdata;
unsigned int temp;
unsigned char i;

sbit ST=P3^0;
sbit OE=P3^1;
sbit EOC=P3^2;
sbit CLK=P3^3;


void main(void)
```

```
{
ST=0;
OE=0;
ET0=1;
ET1=1;
EA=1;
TMOD=0x12;
TH0=216;
TL0=216;
TH1=(65536-4000)/256;
TL1=(65536-4000)%256;
TR1=1;
TR0=1;
ST=1;
ST=0;
while(1)
{
if(EOC==1)
{
OE=1;
getdata=P0;
OE=0;
temp=getdata*235;
temp=temp/128;
i=5;
dispbuf[0]=10;
dispbuf[1]=10;
dispbuf[2]=10;
dispbuf[3]=10;
dispbuf[4]=10;
dispbuf[5]=0;
dispbuf[6]=0;
dispbuf[7]=0;
while(temp/10)
{
dispbuf[i]=temp%10;
temp=temp/10;
i++;
}
dispbuf[i]=temp;
ST=1;
ST=0;
}
}
```

```
}

void t0(void) interrupt 1 using 0
{
CLK=~CLK;
}

void t1(void) interrupt 3 using 0
{
TH1=(65536-4000)/256;
TL1=(65536-4000)%256;
P1=dispcode[dispbuf[dispcount]];
P2=dispbitcode[dispcount];
if(dispcount==7)
{
P1=P1 | 0x80;
}
dispcount++;
if(dispcount==8)
{
dispcount=0;
}
}
```

29. 两点间温度控制

1. 实验任务

用可调电阻调节电压值作为模拟温度的输入量，当温度低于 30℃时，发出长嘀报警声和光报警，当温度高于 60℃时，发出短嘀报警声和光报警。测量的温度范围在 0－99℃。

2. 电路原理图

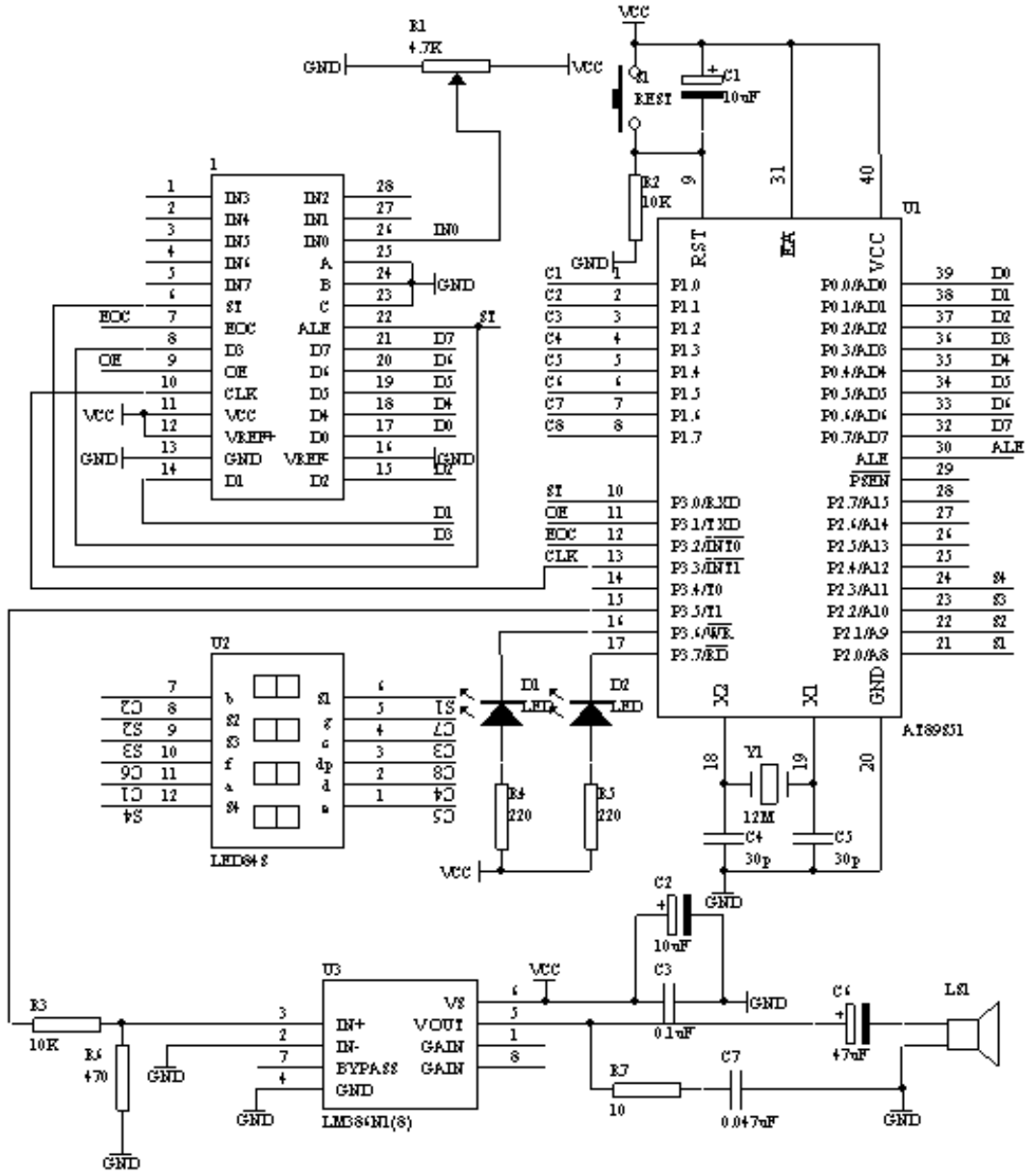


图 4.29.1

3. 系统板上硬件连线

a) 把“单片机系统”区域中的 P1.0－P1.7 与“动态数码显示”区域中的 ABCDEFGH 端口用 8 芯排线连接。

b) 把“单片机系统”区域中的 P2.0－P2.7 与“动态数码显示”区域中的 S1S2S3S4S5S6S7S8 端口用 8 芯排线连接。

c) 把“单片机系统”区域中的 P3.0 与“模数转换模块”区域中的 ST 端子用导线相连接。

d) 把“单片机系统”区域中的P3.1与“模数转换模块”区域中的OE端子用导线相连接。

e) 把“单片机系统”区域中的P3.2与“模数转换模块”区域中的EOC端子用导线相连接。

f) 把“单片机系统”区域中的P3.3与“模数转换模块”区域中的CLK端子用导线相连接。

g) 把“模数转换模块”区域中的A2A1A0端子用导线连接到“电源模块”区域中的GND端子上。

h) 把“模数转换模块”区域中的IN0端子用导线连接到“三路可调电压模块”区域中的VR1端子上。

i) 把“单片机系统”区域中的P0.0－P0.7用8芯排线连接到“模数转换模块”区域中的D0D1D2D3D4D5D6D7端子上。

j) 把“单片机系统”区域中的P3.6、P3.7用导线分别连接到“八路发光二极管指示模块”区域中的L1、L2上。

k) 把“单片机系统”区域中的P3.5用导线连接到“音频放大模块”区域中的SPK IN端口上。

l) 把“音频放大模块“区域中的SPK OUT插入音频喇叭。

4. 汇编源程序

（略）

5. C语言源程序

```
#include <AT89X52.H>

unsigned char code dispbitcode[]={0xfe,0xfd,0xfb,0xf7,
0xef,0xdf,0xbf,0x7f};
unsigned char code dispcode[]={0x3f,0x06,0x5b,0x4f,0x66,
0x6d,0x7d,0x07,0x7f,0x6f,0x00};
unsigned char dispbuf[8]={10,10,10,10,10,10,0,0};
unsigned char dispcount;
unsigned char getdata;
unsigned int temp;
unsigned char i;

sbit ST=P3^0;
sbit OE=P3^1;
sbit EOC=P3^2;
sbit CLK=P3^3;
```

```
sbit LED1=P3^6;
sbit LED2=P3^7;
sbit SPK=P3^5;
bit lowflag;
bit highflag;
unsigned int cnta;
unsigned int cntb;
bit alarmflag;


void main(void)
{
ST=0;
OE=0;
TMOD=0x12;
TH0=0x216;
TL0=0x216;
TH1=(65536-500)/256;
TL1=(65536-500)%256;
TR1=1;
TR0=1;
ET0=1;
ET1=1;
EA=1;
ST=1;
ST=0;
while(1)
{
if((lowflag==1) &&(highflag==0))
{
LED1=0;
LED2=1;
}
else if((highflag==1) && (lowflag==0))
{
LED1=1;
LED2=0;
}
else
{
LED1=1;
LED2=1;
}
}
```

```
}

void t0(void) interrupt 1 using 0
{
CLK=~CLK;
}

void t1(void) interrupt 3 using 0
{

TH1=(65536-500)/256;
TL1=(65536-500)%256;

if(EOC==1)
{
OE=1;
getdata=P0;
OE=0;
temp=getdata*25;
temp=temp/64;
i=6;
dispbuf[0]=10;
dispbuf[1]=10;
dispbuf[2]=10;
dispbuf[3]=10;
dispbuf[4]=10;
dispbuf[5]=10;
dispbuf[6]=0;
dispbuf[7]=0;
while(temp/10)
{
dispbuf[i]=temp%10;
temp=temp/10;
i++;
}
dispbuf[i]=temp;
if(getdata<77)
{
lowflag=1;
highflag=0;
}
else if(getdata>153)
{
lowflag=0;
```

```
highflag=1;
}
else
{
lowflag=0;
highflag=0;
}
ST=1;
ST=0;
}

P1=dispcode[dispbuf[dispcount]];
P2=dispbitcode[dispcount];
dispcount++;
if(dispcount==8)
{
dispcount=0;
}

if((lowflag==1) && (highflag==0))
{
cnta++;
if(cnta==800)
{
cnta=0;
alarmflag=~alarmflag;
}
if(alarmflag==1)
{
SPK=~SPK;
}
}
else if((lowflag==0) && (highflag==1))
{
cntb++;
if(cntb==400)
{
cntb=0;
alarmflag=~alarmflag;
}
if(alarmflag==1)
{
SPK=~SPK;
}
```

```
}
else
{
alarmflag=0;
cnta=0;
cntb=0;
}
}
```

30. 四位数数字温度计

1. 温度传感器 AD590 基本知识

AD590 产生的电流与绝对温度成正比，它可接收的工作电压为 4V－30V，检测的温度范围为－55℃－＋150℃，它有非常好的线性输出性能，温度每增加 1℃，其电流增加 1uA。

AD590 温度与电流的关系如下表所示

摄氏温度	AD590 电流	经 10KΩ电压
0℃	273.2 uA	2.732V
10℃	283.2 uA	2.832 V
20℃	293.2 uA	2.932 V
30℃	303.2 uA	3.032 V
40℃	313.2 uA	3.132 V
50℃	323.2 uA	3.232 V
60℃	333.2 uA	3.332 V
100℃	373.2 uA	3.732 V

AD590 引脚图

2. 实验任务

利用 AD590 温度传感器完成温度的测量，把转换的温度值的模拟量送入 ADC0809 的其中一个通道进行 A/D 转换，将转换的结果进行温度值变换之后送入数码管显示。

3. 电路原理图

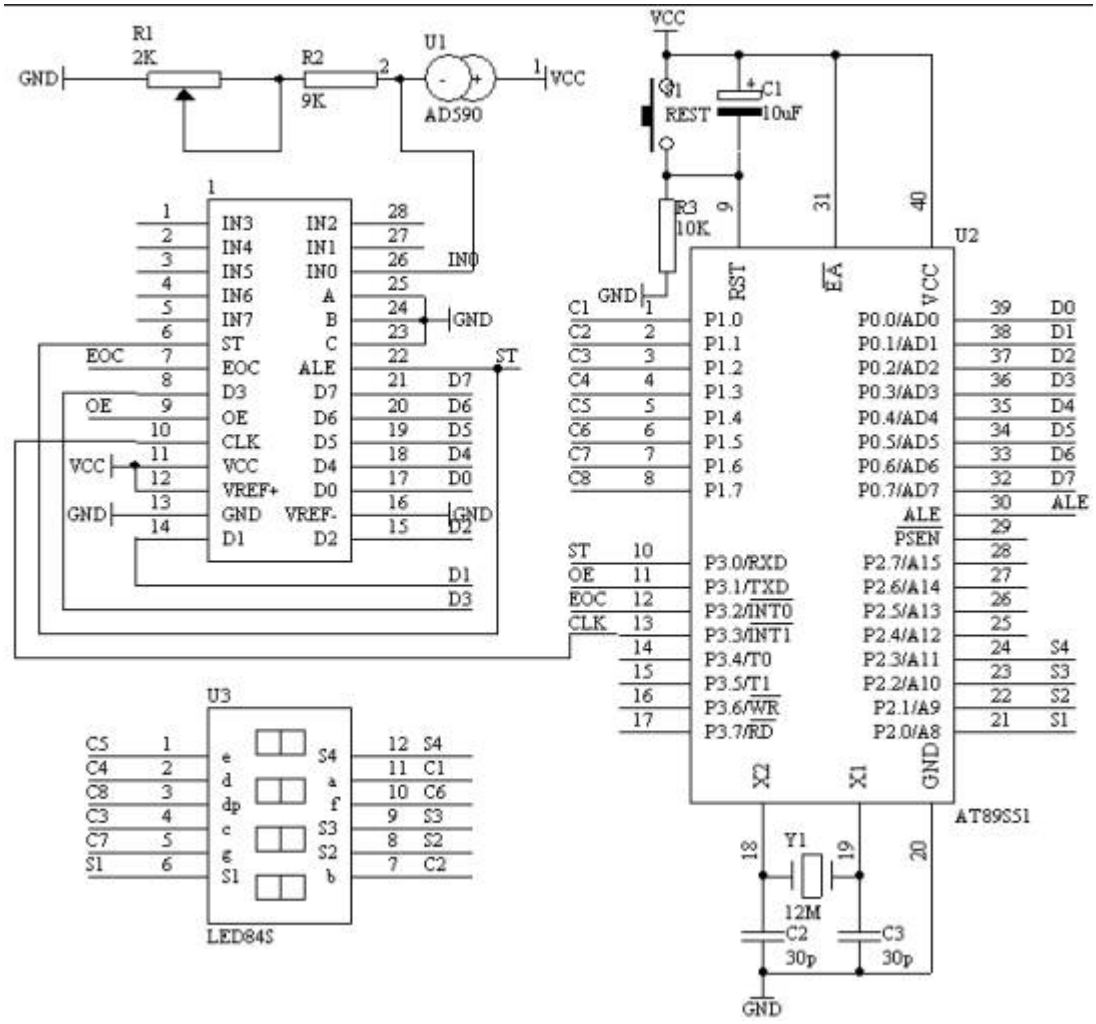


图 4.30.1

4. 系统板上硬件连线

（1）. 把“单片机系统”区域中的 P1.0－P1.7 与“动态数码显示”区域中的 ABCDEFGH 端口用 8 芯排线连接。

（2）. 把“单片机系统”区域中的 P2.0－P2.7 与“动态数码显示”区域中的 S1S2S3S4S5S6S7S8 端口用 8 芯排线连接。

（3）. 把“单片机系统”区域中的 P3.0 与“模数转换模块”区域中的 ST 端子用导线相连接。

（4）. 把“单片机系统”区域中的 P3.1 与“模数转换模块”区域中的 OE 端子用导线相连接。

（5）. 把“单片机系统”区域中的 P3.2 与“模数转换模块”区域中的 EOC 端子用导线相连接。

（6）． 把“单片机系统”区域中的 P3.3 与“模数转换模块”区域中的 CLK 端子用导线相连接。

（7）． 把“模数转换模块”区域中的 A2A1A0 端子用导线连接到“电源模块”区域中的 GND 端子上。

（8）． 把“模数转换模块”区域中的 IN0 端子用导线连接到自制的 AD590 电路上。

（9）． 把“单片机系统”区域中的 P0.0－P0.7 用 8 芯排线连接到“模数转换模块”区域中的 D0D1D2D3D4D5D6D7 端子上。

5． 程序设计内容

（1）． ADC0809 的 CLK 信号由单片机的 P3.3 管脚提供

（2）． 由于 AD590 的温度变化范围在－55℃－＋150℃之间，经过 10KΩ之后采样到的电压变化在 2.182V－4.232V 之间，不超过 5V 电压所表示的范围，因此参考电压取电源电压 VCC，（实测 VCC＝4.70V）。由此可计算出经过 A/D 转换之后的摄氏温度显示的数据为：

如果（D*2350/128）＜2732，则显示的温度值为－（2732－（D*2350/128））

如果（D*2350/128）≥2732，则显示的温度值为＋（（D*2350/128）－2732）

6． 汇编源程序

（略）

7．C 语言源程序

```
#include <AT89X52.H>
#include <ctype.h>

unsigned char code dispbitcode[]={0xfe,0xfd,0xfb,0xf7,
0xef,0xdf,0xbf,0x7f};
unsigned char code dispcode[]={0x3f,0x06,0x5b,0x4f,0x66,
0x6d,0x7d,0x07,0x7f,0x6f,0x00,0x40};
unsigned char dispbuf[8]={10,10,10,10,10,10,0,0};
unsigned char dispcount;
unsigned char getdata;
unsigned long temp;
unsigned char i;
bit sflag;

sbit ST=P3^0;
```

```
sbit OE=P3^1;
sbit EOC=P3^2;
sbit CLK=P3^3;
sbit LED1=P3^6;
sbit LED2=P3^7;
sbit SPK=P3^5;


void main(void)
{
ST=0;
OE=0;
TMOD=0x12;
TH0=0x216;
TL0=0x216;
TH1=(65536-4000)/256;
TL1=(65536-4000)%256;
TR1=1;
TR0=1;
ET0=1;
ET1=1;
EA=1;
ST=1;
ST=0;
getdata=148;
while(1)
{
;
}
}

void t0(void) interrupt 1 using 0
{
CLK=~CLK;
}

void t1(void) interrupt 3 using 0
{

TH1=(65536-4000)/256;
TL1=(65536-4000)%256;

if(EOC==1)
{
```

```
OE=1;
getdata=P0;
OE=0;
temp=(getdata*2350);
temp=temp/128;
if(temp<2732)
{
temp=2732-temp;
sflag=1;
}
else
{
temp=temp-2732;
sflag=0;
}
i=3;
dispbuf[0]=10;
dispbuf[1]=10;
dispbuf[2]=10;
if(sflag==1)
{
dispbuf[7]=11;
}
else
{
dispbuf[7]=10;
}
dispbuf[3]=0;
dispbuf[4]=0;
dispbuf[5]=0;
dispbuf[6]=0;
while(temp/10)
{
dispbuf[i]=temp%10;
temp=temp/10;
i++;
}
dispbuf[i]=temp;
ST=1;
ST=0;
}

P1=dispcode[dispbuf[dispcount]];
P2=dispbitcode[dispcount];
```

```
dispcount++;
if(dispcount==8)
{
dispcount=0;
}
}
```

31. 6 位数显频率计数器

1. 实验任务

利用 AT89S51 单片机的 T0、T1 的定时计数器功能，来完成对输入的信号进行频率计数，计数的频率结果通过 8 位动态数码管显示出来。要求能够对0－250KHZ 的信号频率进行准确计数，计数误差不超过±1HZ。

2. 电路原理图

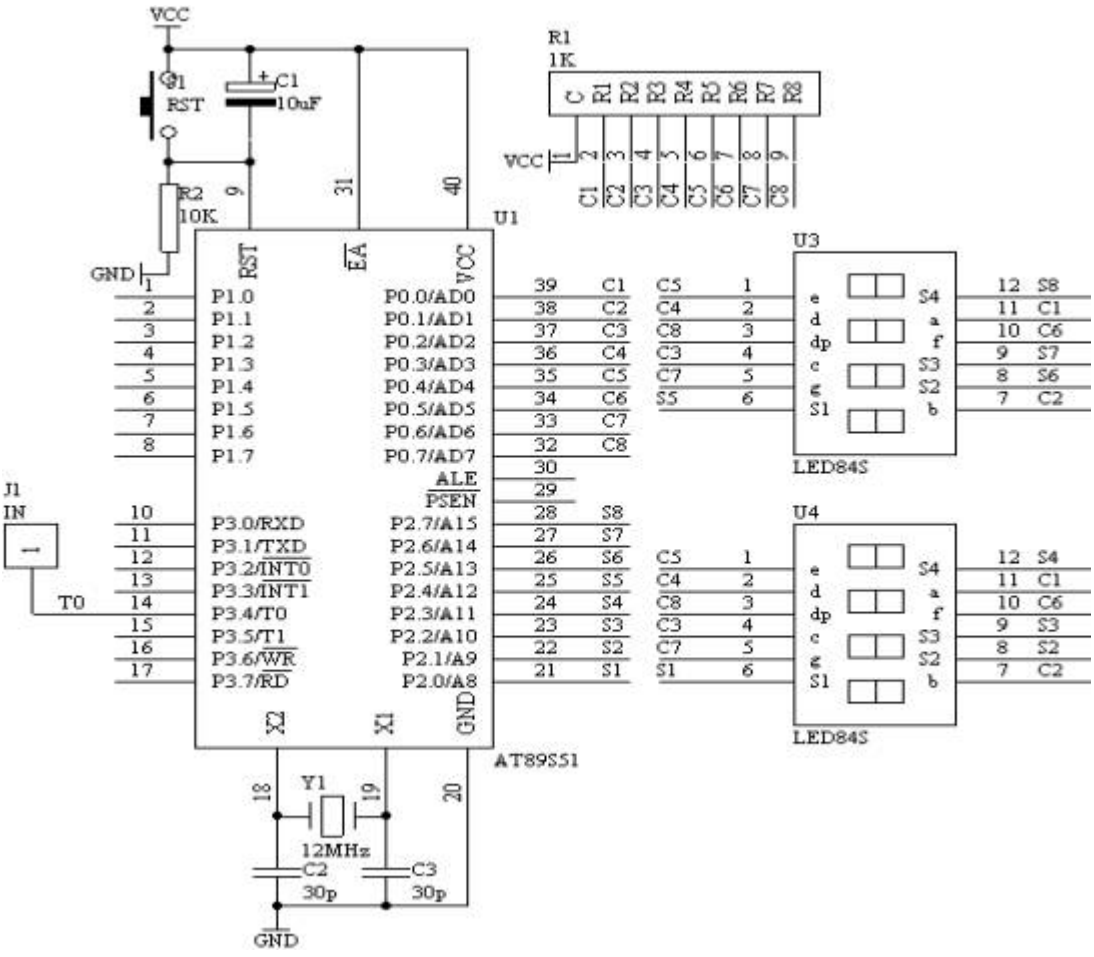


图 4.31.1

3. 系统板上硬件连线

（1）. 把“单片机系统”区域中的 P0.0－P0.7 与“动态数码显示”区域中的 ABCDEFGH 端口用 8 芯排线连接。

（2）. 把“单片机系统”区域中的 P2.0－P2.7 与“动态数码显示”区域中的 S1S2S3S4S5S6S7S8 端口用 8 芯排线连接。

（3）. 把“单片机系统”区域中的 P3.4（T0）端子用导线连接到“频率产生器”区域中的 WAVE 端子上。

4. 程序设计内容

（1）. 定时/计数器 T0 和 T1 的工作方式设置，由图可知，T0 是工作在计数状态下，对输入的频率信号进行计数，但对工作在计数状态下的 T0，最大计数值为 fOSC/24，由于 fOSC＝12MHz，因此：T0 的最大计数频率为 250KHz。对于频率的概念就是在一秒只数脉冲的个数，即为频率值。所以 T1 工作在定时状态下，每定时 1 秒中到，就停止 T0 的计数，而从 T0 的计数单元中读取计数的数值，然后进行数据处理。送到数码管显示出来。

（2）. T1 工作在定时状态下，最大定时时间为 65ms，达不到 1 秒的定时，所以采用定时 50ms，共定时 20 次，即可完成 1 秒的定时功能。

5. C 语言源程序

```
#include <AT89X52.H>
unsigned char code
dispbit[]={0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f};
unsigned char code dispcode[]={0x3f,0x06,0x5b,0x4f,0x66,
0x6d,0x7d,0x07,0x7f,0x6f,0x00,0x40};
unsigned char dispbuf[8]={0,0,0,0,0,0,10,10};
unsigned char temp[8];
unsigned char dispcount;
unsigned char T0count;
unsigned char timecount;
bit flag;
unsigned long x;

void main(void)
{
unsigned char i;

TMOD=0x15;
TH0=0;
TL0=0;
```

```
TH1=(65536-4000)/256;
TL1=(65536-4000)%256;
TR1=1;
TR0=1;
ET0=1;
ET1=1;
EA=1;

while(1)
{
if(flag==1)
{
flag=0;
x=T0count*65536+TH0*256+TL0;
for(i=0;i<8;i++)
{
 temp[i]=0;
}
i=0;
while(x/10)
{
temp[i]=x%10;
x=x/10;
i++;
}
temp[i]=x;
for(i=0;i<6;i++)
{
dispbuf[i]=temp[i];
}
timecount=0;
T0count=0;
TH0=0;
TL0=0;
TR0=1;
}
}
}
void t0(void) interrupt 1 using 0
{
T0count++;
}

void t1(void) interrupt 3 using 0
```

```
{
TH1=(65536-4000)/256;
TL1=(65536-4000)%256;
timecount++;
if(timecount==250)
{
TR0=0;
timecount=0;
flag=1;
}
P0=dispcode[dispbuf[dispcount]];
P2=dispbit[dispcount];
dispcount++;
if(dispcount==8)
{
dispcount=0;
}
}
```

32. 电子密码锁设计

1. 实验任务

根据设定好的密码，采用二个按键实现密码的输入功能，当密码输入正确之后，锁就打开，如果输入的三次的密码不正确，就锁定按键 3 秒钟，同时发现报警声，直到没有按键按下 3 种后，才打开按键锁定功能；否则在 3 秒钟内仍有按键按下，就重新锁定按键 3 秒时间并报警。

2. 电路原理图

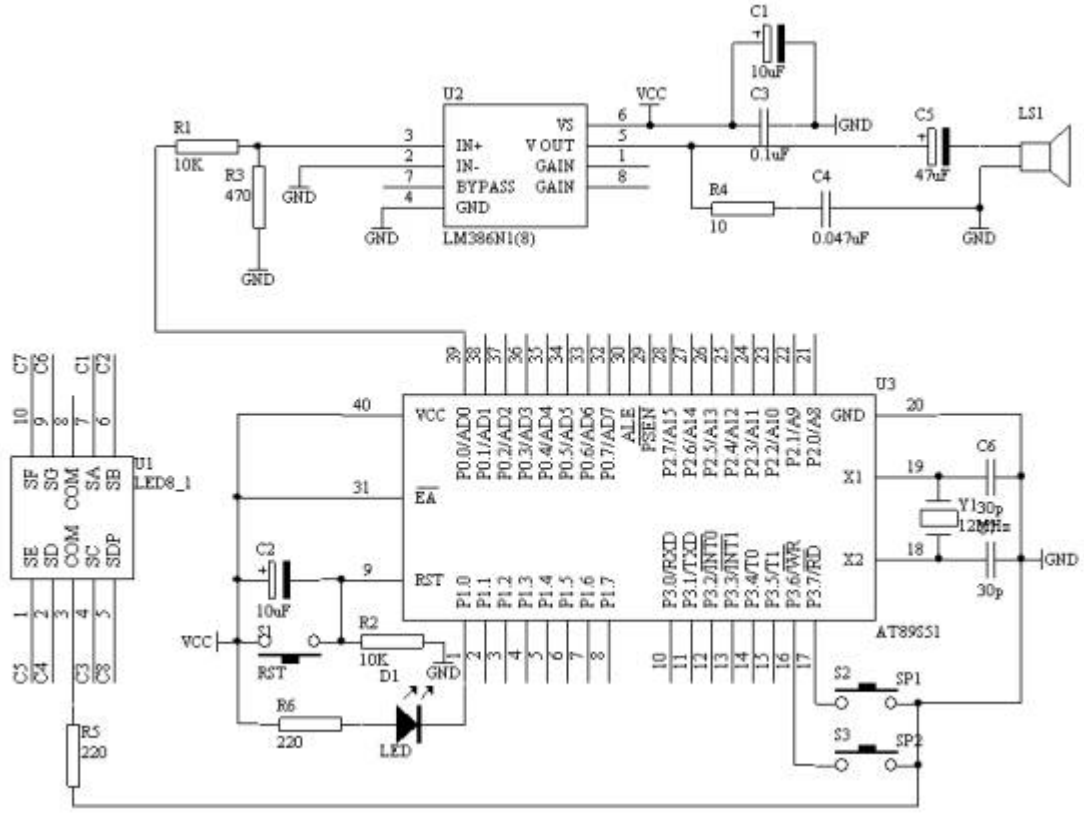


图 4.32.1

3．系统板上硬件连线

（1）． 把“单片机系统”区域中的 P0.0/AD0 用导线连接到“音频放大模块”区域中的 SPK IN 端子上；

（2）． 把“音频放大模块”区域中的 SPK OUT 端子接喇叭和；

（3）． 把“单片机系统”区域中的 P2.0/A8－P2.7/A15 用 8 芯排线连接到“四路静态数码显示”区域中的任一个 ABCDEFGH 端子上；

（4）. 把“单片机系统“区域中的 P1.0 用导线连接到“八路发光二极管模块”区域中的 L1 端子上；

（5）． 把“单片机系统”区域中的 P3.6/WR、P3.7/RD 用导线连接到“独立式键盘”区域中的 SP1 和 SP2 端子上；

4．程序设计内容

（1）． 密码的设定，在此程序中密码是固定在程序存储器 ROM 中，假设预设的密码为“12345”共 5 位密码。

（2）． 密码的输入问题：

由于采用两个按键来完成密码的输入，那么其中一个按键为功能键，另一个按键为数字键。在输入过程中，首先输入密码的长度，接着根据密码的长度输入密码的位数，直到所有长度的密码都已经输入完毕；或者输入确认功能键之后，才能完成密码的输入过程。进入密码的判断比较处理状态并给出相应的处理过程。

（3）． 按键禁止功能：初始化时，是允许按键输入密码，当有按键按下并开始进入按键识别状态时，按键禁止功能被激活，但启动的状态在 3 次密码输入不正确的情况下发生的。

5．C 语言源程序

```
#include <AT89X52.H>

unsigned char code ps[]={1,2,3,4,5};
unsigned char code dispcode[]={0x3f,0x06,0x5b,0x4f,0x66,
0x6d,0x7d,0x07,0x7f,0x6f,0x00,0x40};

unsigned char pslen=9;
unsigned char templen;
unsigned char digit;
unsigned char funcount;
unsigned char digitcount;
unsigned char psbuf[9];
bit cmpflag;
bit hibitflag;
bit errorflag;
bit rightflag;
unsigned int second3;
unsigned int aa;
unsigned int bb;
bit alarmflag;
bit exchangeflag;
unsigned int cc;
unsigned int dd;
bit okflag;
unsigned char oka;
unsigned char okb;

void main(void)
{
 unsigned char i,j;
P2=dispcode[digitcount];
TMOD=0x01;
TH0=(65536-500)/256;
TL0=(65536-500)%256;
```

```
TR0=1;
ET0=1;
EA=1;

while(1)
{
if(cmpflag==0)
{
if(P3_6==0) //function key
{
for(i=10;i>0;i--)
for(j=248;j>0;j--);
if(P3_6==0)
{
if(hibitflag==0)
{
funcount++;
if(funcount==pslen+2)
{
 funcount=0;
cmpflag=1;
}
P1=dispcode[funcount];
}
else
{
second3=0;
}
while(P3_6==0);
}
}

if(P3_7==0) //digit key
{
for(i=10;i>0;i--)
for(j=248;j>0;j--);
if(P3_7==0)
{
 if(hibitflag==0)
{
digitcount++;
if(digitcount==10)
{
digitcount=0;
```

```
}
 P2=dispcode[digitcount];
if(funcount==1)
{
pslen=digitcount;
templen=pslen;
}
else if(funcount>1)
{
psbuf[funcount-2]=digitcount;
}
}
else
{
second3=0;
}
while(P3_7==0);
}
}
}
else
{
cmpflag=0;
for(i=0;i<pslen;i++)
{
if(ps[i]!=psbuf[i])
{
hibitflag=1;
 i=pslen;
errorflag=1;
rightflag=0;
cmpflag=0;
second3=0;
goto a;
}
}
cc=0;
errorflag=0;
 rightflag=1;
hibitflag=0;
a: cmpflag=0;
}
}
}
```

```
void t0(void) interrupt 1 using 0
{
TH0=(65536-500)/256;
TL0=(65536-500)%256;

if((errorflag==1) && (rightflag==0))
{
bb++;
if(bb==800)
{
 bb=0;
alarmflag=~alarmflag;
}
if(alarmflag==1)
{
P0_0=~P0_0;
}

aa++;
if(aa==800)
{
aa=0;
P0_1=~P0_1;
}
second3++;
if(second3==6400)
{
second3=0;
hibitflag=0;
errorflag=0;
rightflag=0;
cmpflag=0;
P0_1=1;
alarmflag=0;
bb=0;
aa=0;
}
}

if((errorflag==0) && (rightflag==1))
{
 P0_1=0;
cc++;
```

```
if(cc<1000)
{
okflag=1;
}
else if(cc<2000)
{
okflag=0;
}
else
{
errorflag=0;
rightflag=0;
hibitflag=0;
cmpflag=0;
P0_1=1;
cc=0;
oka=0;
okb=0;
okflag=0;
P0_0=1;
}
if(okflag==1)
{
oka++;
if(oka==2)
{
oka=0;
P0_0=~P0_0;
}
}
else
{
okb++;
if(okb==3)
{
okb=0;
P0_0=~P0_0;
}
}
}
}
```

33. 4×4 键盘及 8 位数码管显示构成的电子密码锁

1. 实验任务

用 4×4 组成 0－9 数字键及确认键。

用 8 位数码管组成显示电路提示信息，当输入密码时，只显示“8.”，当密码位数输入完毕按下确认键时，对输入的密码与设定的密码进行比较，若密码正确，则门开，此处用 LED 发光二极管亮一秒钟做为提示，同时发出“叮咚”声；若密码不正确，禁止按键输入 3 秒，同时发出“嘀、嘀”报警声；若在 3 秒之内仍有按键按下，则禁止按键输入 3 秒被重新禁止。

2. 电路原理图

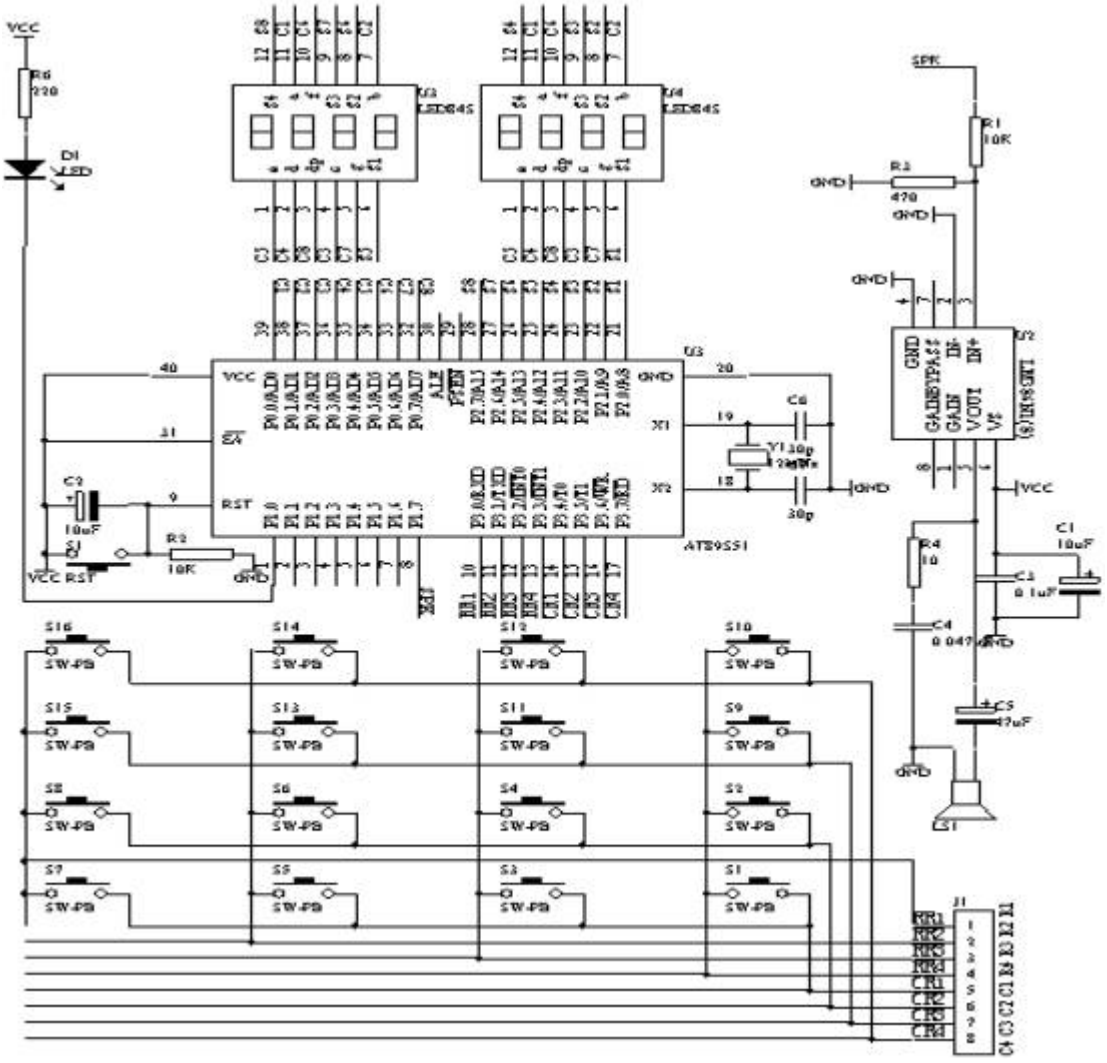
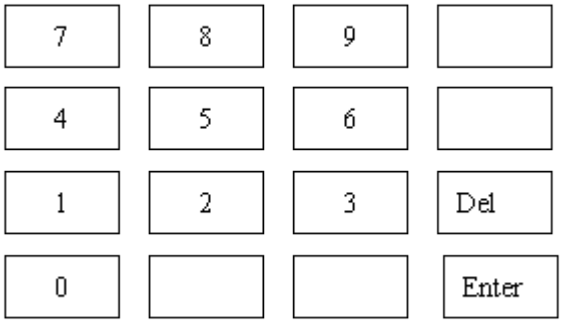


图 4.33.1

3. 系统板上硬件连线

（1）. 把“单片机系统”区域中的 P0.0－P0.7 用 8 芯排线连接到“动态数码显示”区域中的 ABCDEFGH 端子上。



（2）. 把“单片机系统“区域中的 P2.0－P2.7 用8芯排线连接到“动态数码显示”区域中的 S1S2S3S4S5S6S7S8 端子上。

（3）. 把“单片机系统”区域中的 P3.0－P3.7 用8芯排线连接到“4×4 行列式键盘”区域中的 R1R2R3R4C1C2C3C4 端子上。

（4）. 把“单片机系统”区域中的 P1.0 用导线连接到“八路发光二极管模块”区域中的 L2 端子上。

（5）. 把“单片机系统”区域中的 P1.7 用导线连接到“音频放大模块”区域中的 SPK IN 端子上。

（6）. 把“音频放大模块”区域中的 SPK OUT 接到喇叭上。

4. 程序设计内容

（1）. 4×4 行列式键盘识别技术：有关这方面内容前面已经讨论过，这里不再重复。

（2）. 8 位数码显示，初始化时，显示“P　　”，接着输入最大 6 位数的密码，当密码输入完后，按下确认键，进行密码比较，然后给出相应的信息。在输入密码过程中，显示器只显示“8.”。当数字输入超过 6 个时，给出报警信息。在密码输入过程中，若输入错误，可以利用“DEL”键删除刚才输入的错误的数字。

（3）. 4×4 行列式键盘的按键功能分布图如图 4.33.2 所示：

图 4.33.2

5. C 语言源程序

```
#include <AT89X52.H>

unsigned char ps[]={1,2,3,4,5};

unsigned char code dispbit[]={0xfe,0xfd,0xfb,0xf7,
0xef,0xdf,0xbf,0x7f};
```

```
unsigned char code dispcode[]={0x3f,0x06,0x5b,0x4f,0x66,
0x6d,0x7d,0x07,0x7f,0x6f,
0x77,0x7c,0x39,0x5e,0x79,0x71,
0x00,0x40,0x73,0xff};
unsigned char dispbuf[8]={18,16,16,16,16,16,16,16};
unsigned char dispcount;
unsigned char flashcount;
unsigned char temp;
unsigned char key;
unsigned char keycount;
unsigned char pslen=5;
unsigned char getps[6];
bit keyoverflag;
bit errorflag;
bit rightflag;
unsigned int second3;
unsigned int aa,bb;
unsigned int cc;
bit okflag;
bit alarmflag;
bit hibitflag;
unsigned char oka,okb;

void main(void)
{
unsigned char i,j;

TMOD=0x01;
TH0=(65536-500)/256;
TL0=(65536-500)%256;
TR0=1;
ET0=1;
EA=1;

while(1)
{
P3=0xff;
P3_4=0;
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
for(i=10;i>0;i--)
for(j=248;j>0;j--);
```

```
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
switch(temp)
{
case 0x0e:
key=7;
break;
case 0x0d:
key=8;
break;
case 0x0b:
key=9;
break;
case 0x07:
key=10;
break;
}
temp=P3;
P1_1=~P1_1;
if((key>=0) && (key<10))
{
if(keycount<6)
{
getps[keycount]=key;
dispbuf[keycount+2]=19;
}
keycount++;
if(keycount==6)
{
keycount=6;
}
else if(keycount>6)
{
keycount=6;
keyoverflag=1;//key overflow
}
}
else if(key==12)//delete key
{
if(keycount>0)
```

```
{
keycount--;
getps[keycount]=0;
dispbuf[keycount+2]=16;
}
else
{
keyoverflag=1;
}
}
else if(key==15)//enter key
{
if(keycount!=pslen)
{
errorflag=1;
rightflag=0;
second3=0;
}
else
{
for(i=0;i<keycount;i++)
{
if(getps[i]!=ps[i])
{
i=keycount;
errorflag=1;
rightflag=0;
second3=0;
goto a;
}
}
errorflag=0;
rightflag=1;
a: i=keycount;
}
}
temp=temp & 0x0f;
while(temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
}
keyoverflag=0;//?????????
}
```

```
}
P3=0xff;
P3_5=0;
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
for(i=10;i>0;i--)
for(j=248;j>0;j--);
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
switch(temp)
{
case 0x0e:
key=4;
break;
case 0x0d:
key=5;
break;
case 0x0b:
key=6;
break;
case 0x07:
key=11;
break;
}
temp=P3;
P1_1=~P1_1;
if((key>=0) && (key<10))
{
if(keycount<6)
{
getps[keycount]=key;
dispbuf[keycount+2]=19;
}
keycount++;
if(keycount==6)
{
keycount=6;
}
```

```
else if(keycount>6)
{
keycount=6;
keyoverflag=1;//key overflow
}
}
else if(key==12)//delete key
{
if(keycount>0)
{
keycount--;
getps[keycount]=0;
dispbuf[keycount+2]=16;
}
else
{
keyoverflag=1;
}
}
else if(key==15)//enter key
{
if(keycount!=pslen)
{
errorflag=1;
rightflag=0;
second3=0;
}
else
{
for(i=0;i<keycount;i++)
{
if(getps[i]!=ps[i])
{
i=keycount;
errorflag=1;
rightflag=0;
second3=0;
goto a4;
}
}
errorflag=0;
rightflag=1;
a4: i=keycount;
}
```

```
}
temp=temp & 0x0f;
while(temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
}
keyoverflag=0;//?????????
}
}

P3=0xff;
P3_6=0;
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
for(i=10;i>0;i--)
for(j=248;j>0;j--);
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
switch(temp)
{
case 0x0e:
key=1;
break;
case 0x0d:
key=2;
break;
case 0x0b:
key=3;
break;
case 0x07:
key=12;
break;
}
temp=P3;
P1_1=~P1_1;
if((key>=0) && (key<10))
{
```

```
if(keycount<6)
{
getps[keycount]=key;
dispbuf[keycount+2]=19;
}
keycount++;
if(keycount==6)
{
keycount=6;
}
else if(keycount>6)
{
keycount=6;
keyoverflag=1;//key overflow
}
}
else if(key==12)//delete key
{
if(keycount>0)
{
keycount--;
getps[keycount]=0;
dispbuf[keycount+2]=16;
}
else
{
keyoverflag=1;
}
}
else if(key==15)//enter key
{
if(keycount!=pslen)
{
errorflag=1;
rightflag=0;
second3=0;
}
else
{
for(i=0;i<keycount;i++)
{
if(getps[i]!=ps[i])
{
i=keycount;
```

```
errorflag=1;
rightflag=0;
second3=0;
goto a3;
}
}
errorflag=0;
rightflag=1;
a3: i=keycount;
}
}
temp=temp & 0x0f;
while(temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
}
keyoverflag=0;//?????????
}
}

P3=0xff;
P3_7=0;
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
for(i=10;i>0;i--)
for(j=248;j>0;j--);
temp=P3;
temp=temp & 0x0f;
if (temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
switch(temp)
{
case 0x0e:
key=0;
break;
case 0x0d:
key=13;
break;
case 0x0b:
```

```
key=14;
break;
case 0x07:
key=15;
break;
}
temp=P3;
P1_1=~P1_1;
if((key>=0) && (key<10))
{
if(keycount<6)
{
getps[keycount]=key;
dispbuf[keycount+2]=19;
}
keycount++;
if(keycount==6)
{
keycount=6;
}
else if(keycount>6)
{
keycount=6;
keyoverflag=1;//key overflow
}
}
else if(key==12)//delete key
{
if(keycount>0)
{
keycount--;
getps[keycount]=0;
dispbuf[keycount+2]=16;
}
else
{
keyoverflag=1;
}
}
else if(key==15)//enter key
{
if(keycount!=pslen)
{
errorflag=1;
```

```
rightflag=0;
second3=0;
}
else
{
for(i=0;i<keycount;i++)
{
if(getps[i]!=ps[i])
{
i=keycount;
errorflag=1;
rightflag=0;
second3=0;
goto a2;
}
}
errorflag=0;
rightflag=1;
a2: i=keycount;
}
}
temp=temp & 0x0f;
while(temp!=0x0f)
{
temp=P3;
temp=temp & 0x0f;
}
keyoverflag=0;//?????????
}
}
}
}
void t0(void) interrupt 1 using 0
{
TH0=(65536-500)/256;
TL0=(65536-500)%256;

flashcount++;
if(flashcount==8)
{
flashcount=0;
P0=dispcode[dispbuf[dispcount]];
P2=dispbit[dispcount];
dispcount++;
```

```
if(dispcount==8)
{
dispcount=0;
}
}

if((errorflag==1) && (rightflag==0))
{
bb++;
if(bb==800)
{
bb=0;
alarmflag=~alarmflag;
}
if(alarmflag==1)//sound alarm signal
{
P1_7=~P1_7;
}

aa++;
if(aa==800)//light alarm signal
{
aa=0;
P1_0=~P1_0;
}
second3++;
if(second3==6400)
{
second3=0;
errorflag=0;
rightflag=0;
alarmflag=0;
bb=0;
aa=0;
}
}
else if((errorflag==0) && (rightflag==1))
{
P1_0=0;
cc++;
if(cc<1000)
{
okflag=1;
}
```

```
else if(cc<2000)
{
okflag=0;
}
else
{
errorflag=0;
rightflag=0;
P1_7=1;
cc=0;
oka=0;
okb=0;
okflag=0;
P1_0=1;
}
if(okflag==1)
{
oka++;
if(oka==2)
{
oka=0;
P1_7=~P1_7;
}
}
else
{
okb++;
if(okb==3)
{
okb=0;
P1_7=~P1_7;
}
}
}

if(keyoverflag==1)
{
P1_7=~P1_7;
}
}
```

34. 带有存储器功能的数字温度计－DS1624 技术应用

1. DS1624 基本原理

DS1624 是美国 DALLAS 公司生产的集成了测量系统和存储器于一体的芯片。数字接口电路简单，与 I2C 总线兼容，且可以使用一片控制器控制多达 8 片的 DS1624。其数字温度输出达 13 位，精度为 0.03125℃。DS1624 可工作在最低 2.7V 电压下，适用于低功耗应用系统。

（1）． DS1624 基本特性

◆ 无需外围元件即可测量温度

◆ 测量范围为－55℃～＋125℃，精度为 0.03125℃

◆ 测量温度的结果以 13 位数字量（两字节传输）给出

◆ 测量温度的典型转换时间为 1 秒

◆ 集成了 256 字节的 E2PROM 非易性存储器

◆ 数据的读出和写入通过一个 2－线（I2C）串行接口完成

◆ 采用 8 脚 DIP 或 SOIC 封装，如图 2.34.1

图 2.34.1

（2）． 引脚描述及功能方框图

其引脚描述如表 1 所示：

表 1 DS1624 引脚描述

引脚	符号	描述
1	SDA	2－线（I²C）串行数据输入/输出
2	SCL	2－线（I²C）串行时钟端
3	NC	未连接
4	GND	地
5	A2	片选地址输入 A2
6	A1	片选地址输入 A1
7	A0	片选地址输入 A0
8	V_{DD}	电源端（+2.7V～+5.5V）

DS1624 的功能结构图如图 4.34.2 所示：

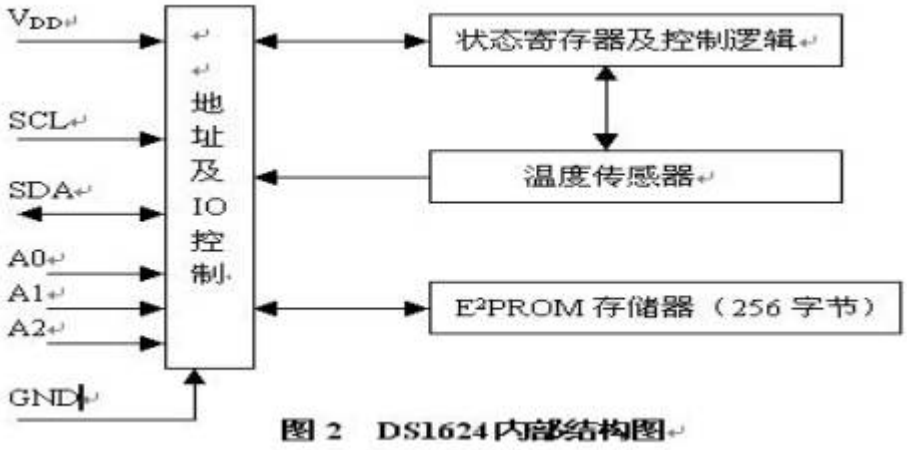


图 2　DS1624 内部结构图

图 4.34.2

（3）．　DS1624 工作原理

温度测量

图 4.34.3 是温度测量的原理结构图

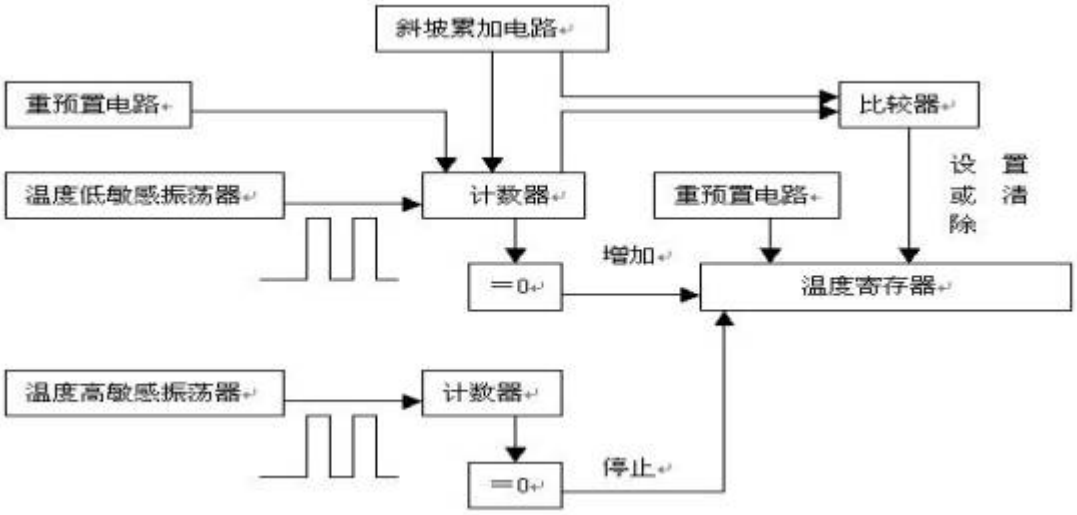


图 4.34.3　温度测量的原理结构图

DS1624 在测量温度时使用了独有的在线温度测量技术。它通过在一个由对温度高度敏感的振荡器决定的计数周期内对温度低敏感的振荡器时钟脉冲的计数值的计算来测量温度。DS1624 在计数器中预置了一个初值,它相当于－55℃。如果计数周期结束之前计数器达到 0，已预置了此初值的温度寄存器中的数字就会增加，从而表明温度高于－55℃。

与此同时，计数器斜坡累加电路被重新预置一个值，然后计数器重新对时钟计数，直到计数值为 0。

通过改变增加的每 1℃内的计数器的计数，斜坡累加电路可以补偿振荡器的非线性误差，以提高精度，任意温度下计数器的值和每一斜坡累加电路的值对应的计数次数须为已知。

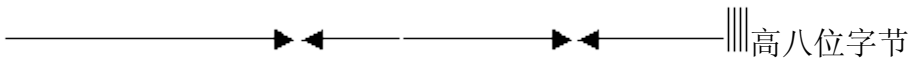
DS1624 通过这些计算可以得到 0.03125℃的精度，温度输出为 13 位，在发出读温度值请求后还会输出两位补偿值。表 2 给出了所测的温度和输出数据的关系。这些数据可通过 2 线制串行口连续输出，MSB 在前，LSB 在后。

表 2　温度与输出数据关系表

温度	数字量输出（二进制）	数字量输出（十六进制）
＋125℃	0111，1101，0000，0000	7D00H
＋25.0625℃	0001，1001，0001，0000	1910H
＋0.5℃	0000，0000，1000，0000	0080H
＋0℃	0000，0000，0000，0000	0000H
-0.5℃	1111，1111，1000，0000	FF80H
-25.0625℃	1110，0110，1111，0000	E6F0H
-55℃	1100，1001，0000，0000	C900H

由于数据在总线上传输时 MSB 在前，所以 DS1624 读出的数据可以是一个字节(分辨率为 1℃)，也可以是两个字节，第二个字节包含的最低位为 0.03125℃。

表 2 是 13 位温度寄存器中存储温度值的数据格式



低八位字节

S	B14	B13	B12	B11	B10	B9	B8		B7	B6	B5	B4	B3	0	0	0

表 3　温度值的数据存储格式

其中 S－为符号位，当 S＝0 时，表示当前的测量的温度为正的温度；当 S＝1 时，表示当前的测量的温度为负的温度。B14－B3 为当前测量的温度值。最低三位被设置为 0。

DS1624 工作方式

DS1621 的工作方式是由片上的配置/状态寄存器来决定的，如表 4，该寄存器的定义如下：

表 4　配置/状态寄存器格式

DONE	1	0	0	1	0	1	1SHOT

其中 DONE 为转换完成位，温度转换结束时置 1，正在进行转换时为 0；1SHOT 为温度转换模式选择。1SHOT 为 1 时为单次转换模式，DS1624 在收到启动温度转换命令 EEH 后进行一次温度转换。1SHOT 为 0 时为连续转换模式，此时 DS1624 将连续进行温度转换，并将最近一次的结果保存在温度寄存器中。该位为非易失性的。

片内 256 字节存储器操作

控制器对 DS1624 的存储器编程有两种模式：一种是字节编程模式，另一种是页编程模式。

在字节编程模式中，主控制器发送地址和一个字节的数据到 DS1624。

在主器件发出开始(START)信号以后，主器件发送写控制字节即 1001A2A1A00（其中 R/W 控制位为低电平“0”）。指示从接收器被寻址，DS1624 接收后应答，再由主器件发送访问存储器指令（17H）后，DS1624 接收后应答，接着由主器件发送的下一个字节字地址将被写入到 DS1624 的地址指针。主器件接收到来自 DS1624 的另一个确认信号以后，发送数据字节，并写入到寻址的存储地址。DS1624 再次发出确认信号，同时主器件产生停止条件 STOP，启动内部写周期。在内部写周期 DS1624 将不产生确认信号。

在页编程模式中，如同字节写方式，先将控制字节、访问存储器指令（17H)、字地址发送到 DS1624，接着发 N 个数据字节，其中以 8 个字节为一个页面。主器件发送不多于一个页面字节的数据字节到 DS1624，这些数据字节暂存在片内页面缓存器中，在主器件发送停止信号以后写入到存储器。接收每一个字节以后，低位顺序地址指针在内部加 1。高位顺序字地址保持为常数。如果主器件在产生停止条件以前要发送多于一页字的数据，地址计数器将会循环，并且先接收到的数据将被覆盖。像字节写操作一样，一旦停止条件被接收到，则内部写周期将开始。

存储器的读操作

在这种模式下，主器件可以从 DS1624 的 EEPROM 中读取数据。主器件在发送开始信号之后，主器件首先发送写控制字节 1001A2A1A00，主器件接收到 DS1624 应答之后，发送访问存储器的指令（17H），收到 DS1624 的应答之后，接着发送字地址将被被写入到 DS1624 的地址指针。这时 DS1624 发送应答信号之后，主器件并没有发送停止信号，而是重新发送 START 开始信号，接着又发送读控制字节 1001A2A1A01，主器件接收到 DS1624 应答之后，开始接收 DS1624 送出来的数据，主器件每接收完一个字节的数据之后，都要发送一个应答信号给 DS1624，直到主器件发送一个非应答信号或停止条件来结束 DS1624 的数据发送过程。

DS1624 的指令集

数据和控制信息的写入读出是以表 5 和表 6 所示的方式进行的。在写入信息时，主器件输出从器件(即 DS1624)的地址，同时 R/W 位置 0。接收到响应位后，

总线上的主器件发出一个命令地址，DS1624 接收此地址后，产生响应位，主器件就向它发送数据。如果要对它进行读操作，主器件除了发出命令地址外，还要产生一个重复的启动条件和命令字节，此时 R/W 位为 1，读操作开始。下面对它们的命令进行说明。

访问存储器指令［17H］：该指令是对 DS1624 的 EEPROM 进行访问，发送该指令之后，下一个字节就是被访问存储器的字地址数据。

访问设置寄存器指令［ACH］：如果 R/W 位置 0，将写入数据到设置寄存器。发出请求后，接下来的一个字节被写入。 如果 R/W 位置 1，将读出存在寄存器中的值。

读温度值指令［AAH］：即读出最后一个测温结果。DS1624 产生两个字节，即为寄存器内的结果。

开始测温指令［EEH］：此命令将开始一次温度的测量，不需再输入数据。在单次测量模式下，可在进行转换的同时使 DS1624 保持闲置状态。在连续模式下，将启动连续测温。

停止测温指令［22H］：该命令将停止温度的测量，不需再输入数据。此命令可用来停止连续测温模式。发出请求后，当前温度测量结束，然后 DS1624 保持闲置状态。直到下一个开始测温的请求发出才继续进行连续测量。

表 5　主机对 DS1624 写操作通信格式

I2C 通信开始	主器件发送控制字节（DS1624 地址和写操作）	DS1624 应答	主器件发送访问 DS1624 的指令	DS1624 应答	主器件发送的数据字节	DS1624 应答	I2C 通信停止

表 6　主机对 DS1624 读操作通信格式

I2C 通信开始	主器件发送控制字节（DS1624 地址和写操作）	DS1624 应答	主器件发送访问 DS1624 的指令	DS1624 应答	I2C 通信开始	主器件发送控制字节（DS1624 地址和读操作）	DS1624 应答	数据字节 0	主机应答	数据字节 1	主机非应答	I2C 通信停止

2. 实验任务

用一片 DS1624 完成本地数字温度的测量，并通过 8 位数码管显示出测量的温度值。其硬件电路图如图 4.34.4 所示

3. 电路原理图

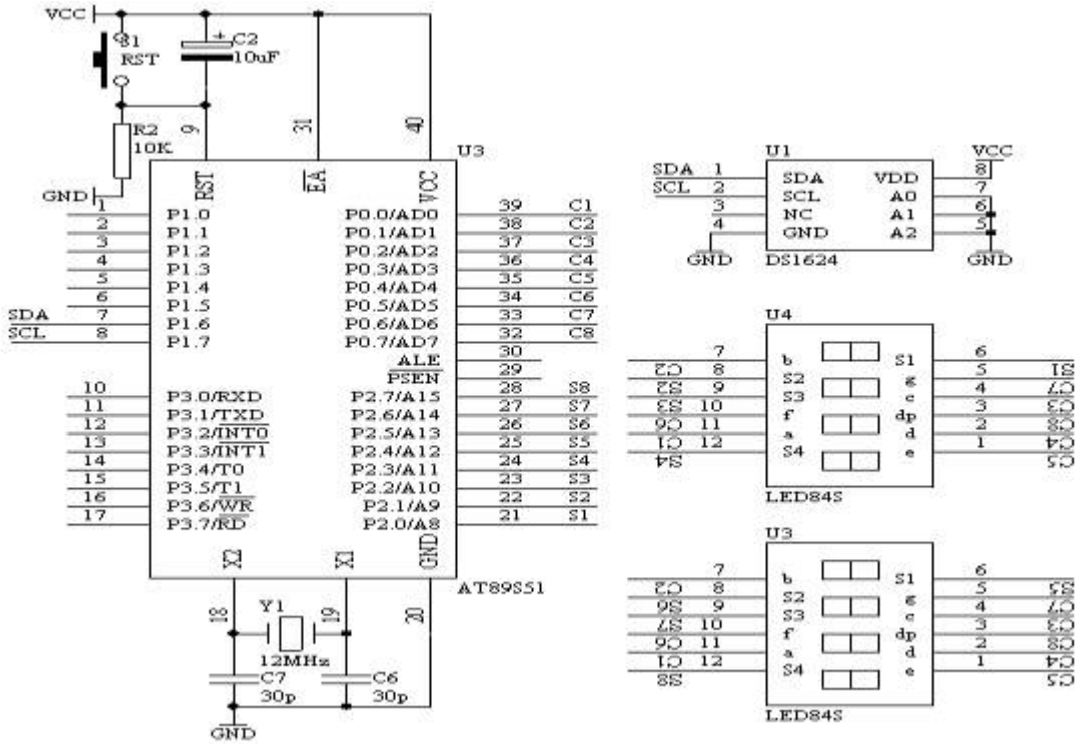


图 4.34.4

4. 系统板上硬件连线

（1）. 把“单片机系统”区域中的 P0.0—P0.7 用 8 芯排线连接到“动态数码显示”区域中的 ABCDEFGH 端子上。

（2）. 把“单片机系统”区域中的 P2.0—P2.7 用 8 芯排线连接到“动态数码显示”区域中的 S1S2S3S4S5S6S7S8 端子上。

（3）. 把 DS1624 芯片插入到“二线总线模块”区域中的 8 脚集成座上，注意芯片不插反。

（4）. 把“二线总线模块”区域中的 PIN1 PIN2 分别用导线连接到“单片机系统”区域中的 P1.6 和 P1.7 端子上。

（5）. 把“二线总线模块”区域中的 PIN4 PIN5 PIN6 分别用导线连接到“电源模块”区域中的 GND 端子上。

5. 程序设计内容

（1）. 由于 DS1624 是 I2C 总线结构的串行数据传送，它只需要 SDA 和 SCL 两根线完成数据的传送过程。因此，我们在进行程序设计的时候，也得按着 I2C 协议来对 DS1624 芯片数据访问。有关 I2C 协议参看有关资料，这里不详述。对于 AT89S51 单片机本身没有 I2C 硬件资源，所以必须用软件来模拟 I2C 协议过程。

（2）. 要从 DS1624 中读取温度值，首先启动 DS1624 的内部温度 A/D 开始转换，对应着有相应的命令用来启动开始温度转换，有关 DS1624 的指令集参考前面的叙述。一般情况下，DS1624 经过一次温度的变换，需要经过 1 秒钟左右的时间，所以等待 1 秒钟后，即可读取内部的温度值，对于读取的温度值，仍然通过 DS1624 的指令集来完成温度的读取。但所有有数据的传送过程必须遵循 I2C 协议。

6. C 语言源程序

```
#include <AT89X52.H>
#include <INTRINS.H>
unsigned char code displaybit[]={0xfe,0xfd,0xfb,0xf7,
0xef,0xdf,0xbf,0x7f};
unsigned char code displaycode[]={0x3f,0x06,0x5b,0x4f,
0x66,0x6d,0x7d,0x07,
0x7f,0x6f,0x77,0x7c,
0x39,0x5e,0x79,0x71,0x00};

unsigned char code dotcode[32]={0,3,6,9,12,16,19,22,
25,28,31,34,38,41,44,48,
50,53,56,59,63,66,69,72,
75,78,81,84,88,91,94,97};
sbit SDA=P1^6;
sbit SCL=P1^7;

unsigned char displaybuffer[8]={0,1,2,3,4,5,6,7};
unsigned char eepromdata[8];
unsigned char temperdata[2];

unsigned char timecount;
unsigned char displaycount;

bit secondflag=0;
unsigned char secondcount=0;
unsigned char retn;
unsigned int result;
unsigned char x;
unsigned int k;
unsigned int ks;
```

```
void delay(void);
void delay10ms(void);
void i_start(void);
void i_stop(void);
void i_init(void);
void i_ack(void);
bit i_clock(void);
bit i_send(unsigned char i_data);
unsigned char i_receive(void);
bit start_temperature_T(void);
bit read_temperature_T(unsigned char *p);
void delay(void)
{
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
}

void delay10ms(void)
{
unsigned int i;
for(i=0;i<1000;i++)
{
delay();
}
}

void i_start(void)
{
SCL=1;
delay();
SDA=0;
delay();
SCL=0;
delay();
}

void i_stop(void)
{
SDA=0;
```

```
delay();
SCL=1;
delay();
SDA=1;
delay();
SCL=0;
delay();
}
void i_init(void)
{
SCL=0;
i_stop();
}

void i_ack(void)
{
SDA=0;
i_clock();
SDA=1;
}

bit i_clock(void)
{
bit sample;

SCL=1;
delay();
sample=SDA;
_nop_();
_nop_();
SCL=0;
delay();
return(sample);
}

bit i_send(unsigned char i_data)
{
unsigned char i;

for(i=0;i<8;i++)
{
SDA=(bit)(i_data & 0x80);
i_data=i_data<<1;
i_clock();
```

```
}
SDA=1;
return(~i_clock());
}
unsigned char i_receive(void)
{
unsigned char i_data=0;
unsigned char i;

for(i=0;i<8;i++)
{
i_data*=2;
if(i_clock()) i_data++;
}
return(i_data);
}

bit start_temperature_T(void)
{
i_start();
if(i_send(0x90))
{
if(i_send(0xee))
{
i_stop();
delay();
return(1);
}
else
{
i_stop();
delay();
return(0);
}
}
else
{
i_stop();
delay();
return(0);
}
}

bit read_temperature_T(unsigned char *p)
```

```
{
i_start();
if(i_send(0x90))
{
if(i_send(0xaa))
{
i_start();
if(i_send(0x91))
{
*(p+1)=i_receive();
i_ack();
*p=i_receive();
i_stop();
delay();
return(1);
}
else
{
i_stop();
delay();
return(0);
}
}
else
{
i_stop();
delay();
return(0);
}
}
else
{
i_stop();
delay();
return(0);
}
}

void main(void)
{
P1=0xff;
timecount=0;
displaycount=0;
TMOD=0x21;
```

```
TH1=0x06;
TL1=0x06;
TR1=1;
ET1=1;
ET0=1;
EA=1;

if(start_temperature_T()) //向 DS1624 发送启动 A/D 温度转换命令，成功则启动 T0 定时 1s。
{
secondflag=0;
secondcount=0;
TH0=55536/256;
TL0=55536%256;
TR0=1;
}
while(1)
{
if(secondflag==1)
{
secondflag=0;
TR0=0;
if(read_temperature_T(temperdata)) //T0 定时 1s 时间到，读取 DS1624 的温度值
{
for(x=0;x<8;x++)
{
displaybuffer[x]=16;
}
x=2;
result=temperdata[1]; //将读取的温度值进行数据处理，并送到显示缓冲区
while(result/10)
{
displaybuffer[x]=result%10;
result=result/10;
x++;
}
displaybuffer[x]=result;
result=temperdata[0];
result=result>>3;
displaybuffer[0]=(dotcode[result])%10;
displaybuffer[1]=(dotcode[result])/10;
if(start_temperature_T()) //温度值数据处理完毕，重新启动 DS1624 开
```

```
始温度转换
{
secondflag=0;
secondcount=0;
TH0=55536/256;
TL0=55536%256;
TR0=1;
}
}
}
}
}
}
void t0(void) interrupt 1 using 0 //T0 用于定时 1s 时间到
{
secondcount++;
if(secondcount==100)
{
secondcount=0;
secondflag=1;
}
TH0=55536/256;
TL0=55536%256;
}
void t1(void) interrupt 3 using 0 //T1 定时 1ms 用数码管的动态刷新
{
timecount++;
if(timecount==4) //T1 定时 1ms 到
{
timecount=0;
if (displaycount==5)
{
P0=(displaycode[displaybuffer[7-displaycount]] | 0x80); //在该位
同时还要显示小数点
}
else
{
P0=displaycode[displaybuffer[7-displaycount]];
}
P2=displaybit[displaycount];
displaycount++;
if(displaycount==8)
{
displaycount=0;
}
```

}
}

35. DS18B20 数字温度计使用

1．DS18B20 基本知识

DS18B20 数字温度计是 DALLAS 公司生产的 1－Wire，即单总线器件，具有线路简单，体积小的特点。因此用它来组成一个测温系统，具有线路简单，在一根通信线，可以挂很多这样的数字温度计，十分方便。

1、DS18B20 产品的特点

（1）、只要求一个端口即可实现通信。

（2）、在 DS18B20 中的每个器件上都有独一无二的序列号。

（3）、实际应用中不需要外部任何元器件即可实现测温。

（4）、测量温度范围在－55。C 到＋125。C 之间。

（5）、数字温度计的分辨率用户可以从 9 位到 12 位选择。

（6）、内部有温度上、下限告警设置。

2、DS18B20 的引脚介绍

TO－92 封装的 DS18B20 的引脚排列见图 1，其引脚功能描述见表 1。

（底视图）图 1

表 1　DS18B20 详细引脚功能描述

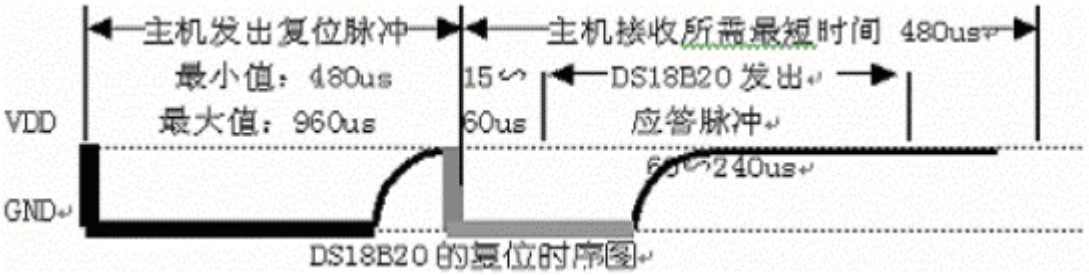
序号	名称	引脚功能描述
1	GND	地信号
2	DQ	数据输入/输出引脚。开漏单总线接口引脚。当被用着在寄生电源下，也可以向器件提供电源。
3	VDD	可选择的 VDD 引脚。当工作于寄生电源时，此引脚必须接地。

3．DS18B20 的使用方法

由于 DS18B20 采用的是 1－Wire 总线协议方式，即在一根数据线实现数据的双向传输，而对 AT89S51 单片机来说，硬件上并不支持单总线协议，因此，我们必须采用软件的方法来模拟单总线的协议时序来完成对 DS18B20 芯片的访问。

由于 DS18B20 是在一根 I/O 线上读写数据，因此，对读写的数据位有着严格的时序要求。DS18B20 有严格的通信协议来保证各位数据传输的正确性和完整性。该协议定义了几种信号的时序：初始化时序、读时序、写时序。所有时序都是将主机作为主设备，单总线器件作为从设备。而每一次命令和数据的传输都是从主机主动启动写时序开始，如果要求单总线器件回送数据，在进行写命令后，主机需启动读时序完成数据接收。数据和命令的传输都是低位在先。

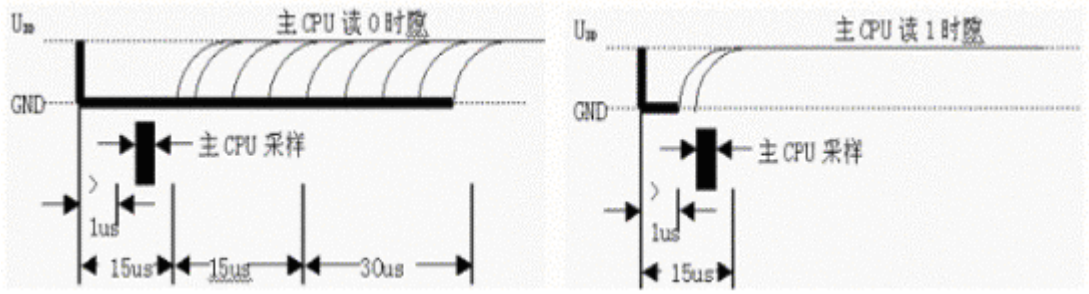
DS18B20 的复位时序



DS18B20 的读时序

对于 DS18B20 的读时序分为读 0 时序和读 1 时序两个过程。

对于 DS18B20 的读时隙是从主机把单总线拉低之后，在 15 秒之内就得释放单总线，以让 DS18B20 把数据传输到单总线上。DS18B20 在完成一个读时序过程，至少需要 60us 才能完成。

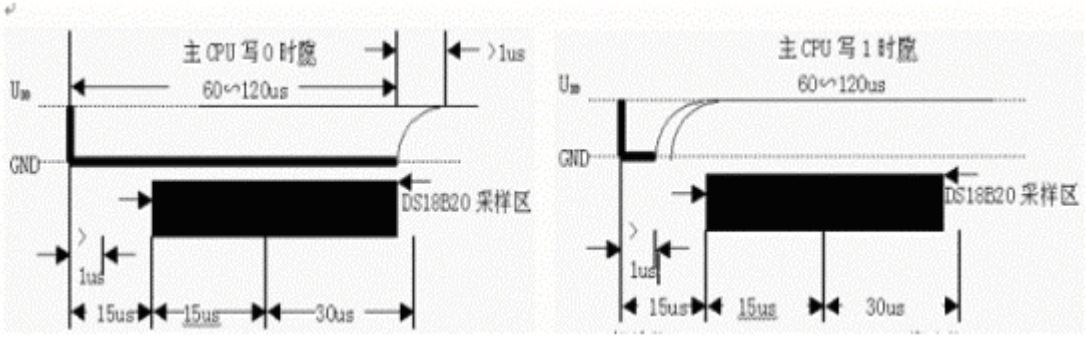


DS18B20 的写时序

对于 DS18B20 的写时序仍然分为写 0 时序和写 1 时序两个过程。

对于 DS18B20 写 0 时序和写 1 时序的要求不同，当要写 0 时序时，单总线要被拉低至少 60us，保证 DS18B20 能够在 15us 到 45us 之间能够正确地采样 IO 总

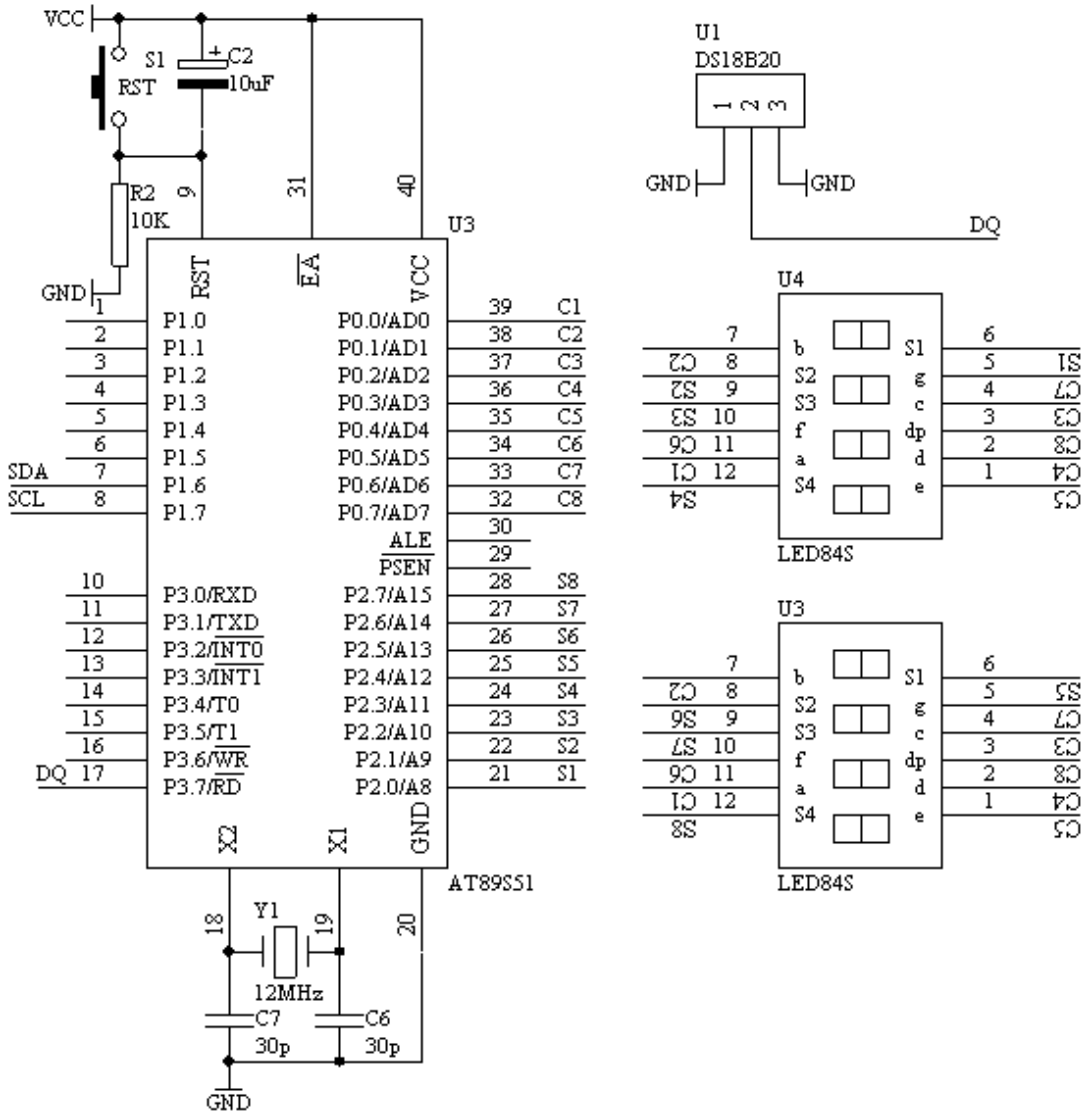
线上的“0”电平，当要写 1 时序时，单总线被拉低之后，在 15us 之内就得释放单总线。



4. 实验任务

用一片 DS18B20 构成测温系统，测量的温度精度达到 0.1 度，测量的温度的范围在－20 度到＋100 度之间，用 8 位数码管显示出来。

5. 电路原理图



6. 系统板上硬件连线

（1）. 把“单片机系统”区域中的 P0.0－P0.7 用 8 芯排线连接到“动态数码显示”区域中的 ABCDEFGH 端子上。

（2）. 把“单片机系统”区域中的 P2.0－P2.7 用 8 芯排线连接到“动态数码显示”区域中的 S1S2S3S4S5S6S7S8 端子上。

（3）. 把 DS18B20 芯片插入“四路单总线”区域中的任一个插座中，注意电源与地信号不要接反。

（4）. 把“四路单总线”区域中的对应的 DQ 端子连接到“单片机系统”区域中的 P3.7/RD 端子上。

7. C语言源程序

```
#include <AT89X52.H>
#include <INTRINS.h>

unsigned char code displaybit[]={0xfe,0xfd,0xfb,0xf7,
0xef,0xdf,0xbf,0x7f};
unsigned char code displaycode[]={0x3f,0x06,0x5b,0x4f,
0x66,0x6d,0x7d,0x07,
0x7f,0x6f,0x77,0x7c,
0x39,0x5e,0x79,0x71,0x00,0x40};
unsigned char code dotcode[32]={0,3,6,9,12,16,19,22,
25,28,31,34,38,41,44,48,
50,53,56,59,63,66,69,72,
75,78,81,84,88,91,94,97};
unsigned char displaycount;
unsigned char displaybuf[8]={16,16,16,16,16,16,16,16};
unsigned char timecount;
unsigned char readdata[8];
sbit DQ=P3^7;
bit sflag;

bit resetpulse(void)
{
unsigned char i;

DQ=0;
for(i=255;i>0;i--);
DQ=1;
for(i=60;i>0;i--);
return(DQ);
for(i=200;i>0;i--);
}

void writecommandtods18b20(unsigned char command)
{
unsigned char i;
unsigned char j;

for(i=0;i<8;i++)
{
if((command & 0x01)==0)
{
DQ=0;
for(j=35;j>0;j--);
```

```
DQ=1;
}
else
{
DQ=0;
for(j=2;j>0;j--);
DQ=1;
for(j=33;j>0;j--);
}
command=_cror_(command,1);
}
}

unsigned char readdatafromds18b20(void)
{
unsigned char i;
unsigned char j;
unsigned char temp;

temp=0;
for(i=0;i<8;i++)
{
temp=_cror_(temp,1);
DQ=0;
_nop_();
_nop_();
DQ=1;
for(j=10;j>0;j--);
if(DQ==1)
{
temp=temp | 0x80;
}
else
{
temp=temp | 0x00;
}
for(j=200;j>0;j--);
}
return(temp);
}

void main(void)
{
TMOD=0x01;
```

```
TH0=(65536-4000)/256;
TL0=(65536-4000)%256;
ET0=1;
EA=1;

while(resetpulse());
writecommandtods18b20(0xcc);
writecommandtods18b20(0x44);
TR0=1;
while(1)
{
;
}
}

void t0(void) interrupt 1 using 0
{
unsigned char x;
unsigned int result;

TH0=(65536-4000)/256;
TL0=(65536-4000)%256;
if(displaycount==2)
{
P0=displaycode[displaybuf[displaycount]] | 0x80;
}
else
{
P0=displaycode[displaybuf[displaycount]];
}
P2=displaybit[displaycount];
displaycount++;
if(displaycount==8)
{
displaycount=0;
}

timecount++;
if(timecount==150)
{
timecount=0;
while(resetpulse());
writecommandtods18b20(0xcc);
writecommandtods18b20(0xbe);
```

```
readdata[0]=readdatafromds18b20();
readdata[1]=readdatafromds18b20();
for(x=0;x<8;x++)
{
displaybuf[x]=16;
}
sflag=0;
if((readdata[1] & 0xf8)!=0x00)
{
sflag=1;
readdata[1]=~readdata[1];
readdata[0]=~readdata[0];
result=readdata[0]+1;
readdata[0]=result;
if(result>255)
{
readdata[1]++;
}
}
readdata[1]=readdata[1]<<4;
readdata[1]=readdata[1] & 0x70;
x=readdata[0];
x=x>>4;
x=x & 0x0f;
readdata[1]=readdata[1] | x;
x=2;
result=readdata[1];
while(result/10)
{
displaybuf[x]=result%10;
result=result/10;
x++;
}
displaybuf[x]=result;
if(sflag==1)
{
displaybuf[x+1]=17;
}
x=readdata[0] & 0x0f;
x=x<<1;
displaybuf[0]=(dotcode[x])%10;
displaybuf[1]=(dotcode[x])/10;
while(resetpulse());
writecommandtods18b20(0xcc);
```

```
writecommandtods18b20(0x44);
}
}
```