输出 当 CPU 执行一条指令将数据写入发送缓冲 SBUF 时，就启动发送。串行数据从 TXD 管脚输出，发送完一帧数据后，就由硬件置位 TI。

输入 在（REN）=1 时，串行口采样 RXD 管脚，当采样到 1 至 0 的跳变时，确认是开始位 0，就开始接收一帧数据。只有当（RI）=0 且停止位为 1 或者（SM2）=0 时，停止位才进入 RB8，8 位数据才能进入接收寄存器，并由硬件置位中断标志 RI；不然信息丢失。所以在方式 1 接收时，应先用软件清零 RI 和 SM2 标志。

方式 2

方式月为固定波特率的 11 位 UART 方式。它比方式 1 增加了一位可程控为 1 或 0 的第 9 位数据。

输出: 发送的串行数据由 TXD 端输出一帧信息为 11 位，附加的第 9 位来自 SCON 寄存器的 TB8 位，用软件置位或复位。它可作为多机通信中地址/数据信息的标志位，也能作为数据的奇偶校验位。当 CPU 执行一条数据写入 SUBF 的指令时，就启动发送器发送。发送一帧信息后，置位中断标志 TI。

输入: 在（REN）=1 时，串行口采样 RXD 管脚，当采样到 1 至 0 的跳变时，确认是开始位 0，就开始接收一帧数据。在接收到附加的第 9 位数据后，当（RI）=0 或者（SM2）=0 时，第 9 位数据才进入 RB8，8 位数据才能进入接收寄存器，并由硬件置位中断标志 RI；不然信息丢失。且不置位 RI。再过一位时间后，不管上述条件时否满足，接收电路即行复位，并重新检测 RXD 上从 1 到 0 的跳变。

工作方式 3

方式 3 为波特率可变的 11 位 UART 方式。除波特率外，其余与方式 2 相同。

波特率选择

如前所述，在串行通信中，收发双方的数据传送率（波特率）要有一定的约定。在 8051 串行口的四种工作方式中，方式 0 和 2 的波特率是固定的，而方式 1 和 3 的波特率是可变的，由定时器 T1 的溢出率控制。

方式 0

方式 0 的波特率固定为主振频率的 1/12。

方式 2

方式 2 的波特率由 PCON 中的选择位 SMOD 来决定，可由下式表示：

波特率=2 的 SMOD 次方除以 64 再乘一个 fosc，也就是当 SMOD=1 时，波特率为 1/32fosc，当 SMOD=0 时，波特率为 1/64fosc

3．方式 1 和方式 3

定时器 T1 作为波特率发生器，其公式如下：

波特率= $\frac{2^{smod}}{32}\times$ 定时器 T1 溢出率

T1 溢出率= T1 计数率/产生溢出所需的周期数

式中 T1 计数率取决于它工作在定时器状态还是计数器状态。当工作于定时器状态时，T1 计数率为 fosc/12;当工作于计数器状态时，T1 计数率为外部输入频率，此频率应小于 fosc/24。产生溢出所需周期与定时器 T1 的工作方式、T1 的预置值有关。

定时器 T1 工作于方式 0：溢出所需周期数=8192-x

定时器 T1 工作于方式 1：溢出所需周期数=65536-x

定时器 T1 工作于方式 2：溢出所需周期数=256-x

因为方式 2 为自动重装入初值的 8 位定时器/计数器模式，所以用它来做波特率发生器最恰当。

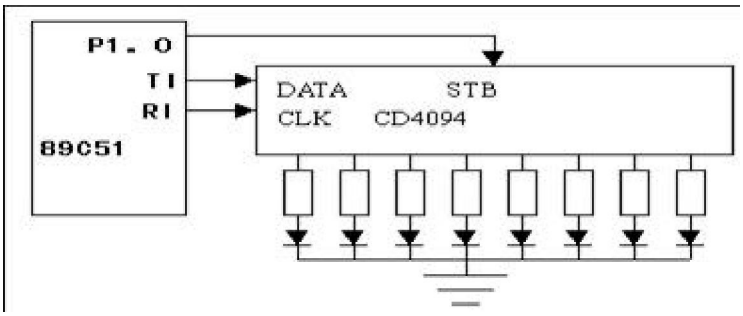
当时钟频率选用 11.0592MHZ 时，取易获得标准的波特率，所以很多单片机系统选用这个看起来“怪”的晶体震荡器就是这个道理。

下表列出了定时器 T1 工作于方式 2 常用波特率及初值。

常用波特率	Fosc(MHZ)	SMOD	TH1 初值
19200	11.0592	1	FDH
9600	11.0592	0	FDH
4800	11.0592	0	FAH
2400	11.0592	0	F4h
1200	11.0592	0	E8h

22 课:单片机串行口通信程序设计

1．串行口方式 0 应用编程 8051 单片机串行口方式 0 为移位寄存器方式，外接一个串入并出的移位寄存器，就能扩展一个并行口。



<单片机串行口通信程序设计硬件连接图>

例：用 8051 单片机串行口外接 CD4094 扩展 8 位并行输出口，如图所示，8 位并行口的各位都接一个发光二极管，要求发光管呈流水灯状态。 串行口方式 0 的数据传送可采用中断方式，也可采用查询方式，无论哪种方式，都要借助于 TI 或 RI 标志。串行发送时，能靠 TI 置位（发完一帧数据后）引起中断申请，在中断服务程序中发送下一帧数据，或者通过查询 TI 的状态，只要 TI 为 0 就继续查询，TI 为 1 就结束查询，发送下一帧数据。在串行接收时，则由 RI 引起中断或对 RI 查询来确定何时接收下一帧数据。无论采用什么方式，在开始通信之前，都要先对控制寄存器 SCON 进行初始化。在方式 0 中将，将 00H 送 SCON 就能了。

----------单片机串行口通信程序设计列子----------

```
ORG  2000H
START:  MOV  SCON,#00H  ;置串行口工作方式 0
MOV  A,#80H  ;最高位灯先亮
CLR  P1.0  ;关闭并行输出（避象传输过程中，各 LED 的"暗红"现象）
OUT0:  MOV  SBUF,A  ;开始串行输出
OUT1:  JNB  TI,OUT1  ;输出完否
CLR  TI  ;完了，清 TI 标志，以备下次发送
SETB  P1.0  ;打开并行口输出
ACALL  DELAY  ;延时一段时间
RR  A  ;循环右移
CLR  P1.0  ;关闭并行输出
JMP  OUT0  ;循环
```

说明：DELAY 延时子程序能用前面我们讲 P1 口流水灯时用的延时子程序，这里就不给出了。

二、串行口异步通信

```
org  0000H
AJMP  START
ORG  30H
START:
mov  SP,#5fh  ;
mov  TMOD,#20h  ;T1:  工作模式 2
mov  PCON,#80h  ;SMOD=1
mov  TH1,#0FDH  ;初始化波特率（参见表）
mov  SCON,#50h  ;Standard  UART  settings
MOV  R0,#0AAH  ;准备送出的数
SETB  REN  ;允许接收
SETB  TR1  ;T1 开始工作
WAIT:
MOV  A,R0
CPL  A
MOV  R0,A
MOV  SBUF,A
LCALL  DELAY
JBC  TI,WAIT1  ;如果 TI 等于 1，则清 TI 并转 WAIT1
AJMP  WAIT
WAIT1:  JBC  RI,READ  ;如果 RI 等于 1，则清 RI 并转 READ
AJMP  WAIT1
READ:
MOV  A,SBUF  ;将取得的数送 P1 口
MOV  P1,A
LJMP  WAIT
DELAY:  ;延时子程序
MOV  R7,#0ffH
DJNZ  R7,$
RET
END
```

将程序编译通过，写入芯片，插入实验板，用通读电缆将实验板与主机的串行口相连就能实验了。上面的程序功能很简单，就是每隔一段时间向主机轮流送数 55H 和 AAH，并把主机送去的数送到 P1 口。能在 PC 端用串行口精灵来做实验。串行口精灵在我主页上有下载。运行串行口精灵后，按主界面上的“设置参数”按钮进入“设置参数”对话框，按下面的参数进行设置。注意，我的机器上用的是串行口 2，如果你不是串行口 2，请自行更改串行口的设置。

设置完后，按确定返回主界面，注意右边有一个下拉列表，应当选中“按 16 进制”。然后按“开始发送”、“开始接收”就能了。按此设置，实验板上应当有两只灯亮，6 只灯灭。大家能自行更改设置参数中的发送字符如 55，00，FF 等等，观察灯的亮灭，并分析原因，也能在主界面上更改下拉列表中的“按 16 进制”为“按 10 进制”或“按 ASCII 字符”来观察现象，并仔细分析。这对于大家理解 16 进制、10 进制、ASCII 字符也是很有好处的。程序本身很简单，又有注释，这里就不详加说明了。

三、上述程序的中断版本

```
org  0000H
AJMP  START
org  0023h
AJMP  SERIAL  ;
ORG  30H
START:
mov  SP,#5fh  ;
mov  TMOD,#20h  ;T1:  工作模式 2
mov  PCON,#80h  ;SMOD=1
mov  TH1,#0FDH  ;初始化波特率（参见表）
mov  SCON,#50h  ;Standard  UART  settings
MOV  R0,#0AAH  ;准备送出的数
SETB  REN  ;允许接收
SETB  TR1  ;T1 开始工作
SETB  EA  ;开总中断
SETB  ES  ;开串行口中断
SJMP  $
SERIAL:
MOV  A,SBUF
MOV  P1,A
CLR  RI
RETI
END
```

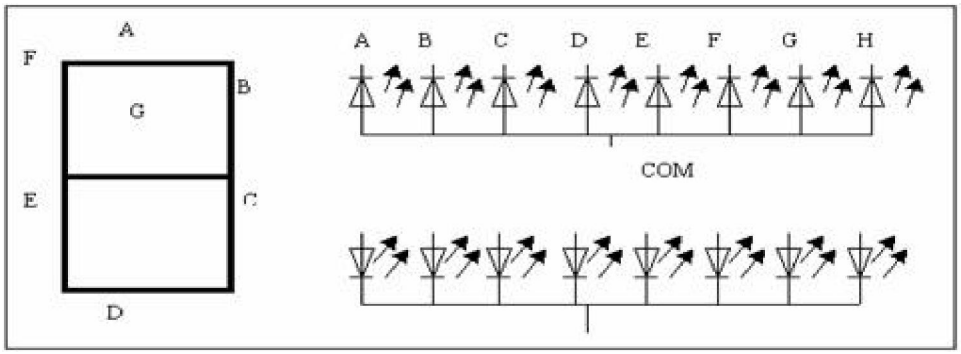
本程序没有写入发送程序，大家能自行添加。

23 课:LED 数码管静态显示接口与编程

在单片机系统中，常常用 LED 数码数码管显示器来显示各种数字或符号。由于它具有显示清晰、亮度高、使用电压低、寿命长的特点，因此使用非常广泛。

引言：还记得我们小时候玩的“火柴棒游戏”吗，几根火柴棒组合起来，能拼成各种各样的图形，LED 数码管显示器实际上也是这么一个东西。

八段 LED 数码管显示器



<单片机静态显示接口>

八段 LED 数码管显示器由 8 个发光二极管组成。基中 7 个长条形的发光管排列成“日”字形，另一个贺点形的发光管在数码管显示器的右下角作为显示小数点用，它能显示各种数字及部份英文字母。LED 数码管显示器有两种不一样的形式：一种是 8 个发光二极管的阳极都连在一起的，称之为共阳极 LED 数码管显示器；另一种是 8 个发光二极管的阴极都连在一起的，称之为共阴极 LED 数码管显示器。如下图所示。`

共阴和共阳结构的 LED 数码管显示器各笔划段名和安排位置是相同的。当二极管导通时，对应的笔划段发亮，由发亮的笔划段组合而显示的各种字符。8 个笔划段 hgfedcba 对应于一个字节（8 位）的 D7 D6 D5 D4 D3 D2 D1 D0,于是用 8 位二进制码就能表示欲显示字符的字形代码。例如，对于共阴 LED 数码管显示器，当公共阴极接地（为零电平），而阳极 hgfedcba 各段为 0111011 时，数码管显示器显示"P"字符，即对于共阴极 LED 数码管显示器，“P”字符的字形码是 73H。如果是共阳 LED 数码管显示器，公共阳极接高电平，显示“P”字符的字形代码应为 10001100（8CH）。这里必须注意的是：很多产品为方便接线，常不按规则的办法去对应字段与位的关系，这个时候字形码就必须根据接线来自行设计了，后面我们会给出一个例程。

在单片机应用系统中，数码管显示器显示常用两种办法：静态显示和动态扫描显示。所谓静态显示，就是每一个数码管显示器都要占用单独的具有锁存功能的 I/O 接口用于笔划段字形代码。这样单片机只要把要显示的字形代码发送到接口电路，就不用管它了，直到要显示新的数据时，再发送新的字形码，因此，使用这种办法单片机中 CPU 的开销小。能供给单独锁存的 I/O 接口电路很多，这里以常用的串并转换电路 74LS164 为例，介绍一种常用静态显示电路，以使大家对静态显示有一定的了解。

MCS-51 单片机串行口方式押为移们寄存器方式，外接 6 片 74LS164 作为 6 位 LED 数码管显示器的静态显示接口，把 8031 的 RXD 作为数据输出线，TXD 作为移位时钟脉冲。74LS164 为 TTL 单向 8 位移位寄存器，可实现串行输入，并行输出。其中 A、B（第 1、2 脚）

为串行数据输入端，2 个管脚按逻辑与运算规律输入信号，公一个输入信号时可并接。T（第 8 脚）为时钟输入端，可连接到串行口的 TXD 端。每一个时钟信号的上升沿加到 T 端时，移位寄存器移一位，8 个时钟脉冲过后，8 位二进制数全部移入 74LS164 中。R（第 9 脚）为复位端，当 R=0 时，移位寄存器各位复 0，只有当 R=1 时，时钟脉冲才起作用。Q1…Q8（第 3-6 和 10-13 管脚）并行输出端分别接 LED 数码管显示器的 hg---a 各段对应的管脚上。关于 74LS164 还能作如下的介绍：所谓时钟脉冲端，其实就是需要高、低、高、低的脉冲，不管这个脉冲是怎么来的，比如，我们用根电线，一端接 T，一端用手拿着，分别接高电平、低电平，那也是给出时钟脉冲，在 74LS164 获得时钟脉冲的瞬间（再讲清楚点，是在脉冲的沿），如果数据输入端（第 1，2 管脚）是高电平，则就会有一个 1 进入到 74LS164 的内部，如果数据输入端是低电平，则就会有一个 0 进入其内部。在给出了 8 个脉冲后，最先进入 74LS164 的第一个数据到达了最高位，然后再来一个脉冲会有什么发生呢？再来一个脉冲，第一个脉冲就会从最高位移出，就象车站排队买票，栏杆就那么长，要从后面进去一本人，前面必须要从前面走出去一本人才行。

搞清了这一点，下面让我们来看电路，6 片 7LS164 首尾相串，而时钟端则接在一起，这样，当输入 8 个脉冲时，从单片机 RXD 端输出的数据就进入到了第一片 74LS164 中了，而当第二个 8 个脉冲到来后，这个数据就进入了第二片 74LS164，而新的数据则进入了第一片 74LS164，这样，当第六个 8 个脉冲完成后，首次送出的数据被送到了最左面的 164 中，其他数据依次出现在第一、二、三、四、五片 74LS164 中。有个问题，在第一个脉冲到来时，除了第一片 74LS164 中接收数据外，其他各片在干吗呢？它们也在接收数据，因为它们的时钟端都是被接在一起的，可是数据还没有送到其他各片呢，它们在接收什么数据呢？。。。。。。其实所谓数据不过是一种说法而已，实际就是电平的高低，当第一个脉冲到来时，第一片 164 固然是从单片机接收数据了，而其它各片也接到前一片的 Q8 上，而 Q8 是一根电线，在数字电路中它只可能有两种状态：低电平或高电平，也就是“0”和“1”。所以它的下一片 74LS164 也相当于是在接收数据啊。只是接收的全部是 0 或 1 而已。这个问题放在这儿说明，可能有朋友不屑一顾，而有的朋友可能还是不清楚，这实际上涉及到数的本质的问题，如果不懂的，请仔细思考，并找一些数字电路的数，理解 164 的工作原理，再来看这个问题，或者去看看我的另一篇文章《27 课:关于单片机的一些基本概念》的文章。务必搞懂，搞懂了这一点，你的级别就高过开始学习者，可谓入门者了。

入口：把要显示的数分别放在显示缓冲区 60H-65H 共 6 个单元中，并且分别对应各个数码管 LED0-LED5。

出口：将预置在显示缓冲区中的 6 个数成对应的显示字形码，然后输出到数码管显示器中显示。

单片机 led 显示程序如下：

```
DISP:  MOV  SCON,#00H  ;初始化串行口方式 0

MOV  R1,#06H  ;显示 6 位数

MOV  R0,#65H  ;60H-65H 为显示缓冲区

MOV  DPTR,#SEGTAB  ;字形表的入口地址

LOOP:

MOV  A,@R0  ;取最高位的待显示数据
```

```
MOVC  A,@A+DPTR  ;查表获取字形码
MOV  SBUF,A  ;送串行口显示
DELAY:  JNB  TI,DELAY  ;等待发送完毕
CLR  TI  ;清发送标志
DEC  R0  ;指针下移一位，准备取下一个待显示数
DJNZ  R1,LOOP  ;直到 6 个数据全显示完。
RET
SETTAB:  ;字形表，前面有介绍，以后我们再介绍字形表的制作。
DB  03H  9FH  27H  0DH  99H  49H  41H  1FH  01H  09H  0FFH
;  0  1  2  3  4  5  6  7  8  9  消隐码
```

单片机显示测试用主程序

```
ORG  0000H
AJMP  START
ORG  30H
START:  MOV  SP,#6FH
MOV  65H,#0
MOV  64H,#1
MOV  63H,#2
MOV  62H,#3
MOV  61H,#4
MOV  60H,#5
LCALL  DISP
SJMP  $
```

如果按图示数码管排列，则以上主程序将显示的是 543210，想想看，如果要显示 012345 该怎样送数？

下面我们来分析一下字形表的制作问题。先就上述“标准”的图形来看吧。写出数据位和字形的对应关系并列一个表如下（设为共阳型，也就是对应的输出位为 0 时笔段亮）

如何，字形表会做了吧，就是这样列个表格，根据要求（0 亮或 1 亮）写出对应位的 0 和 1，就成了。做个练习，写出 A-F 的字形码吧。

如果为了接线方便而打乱了接线的次序，那么字形表又该如何接呢？也很简单，一样地列表啊。以新实验板为例，共阳型。接线如下：

P0.7 P0.6 P0.5 P0.4 P0.3 P0.2 P0.1 P0.0

C E H D G F A B

则字形码如下所示：

;0 00101000 28H

;1 01111110 7EH

;2 10100100 0A4H

;3 01100100 64H

;4 01110010 72H

;5 01100001 61H

;6 00100001 21H

;7 01111100 7CH

;8 00100000 20H

;9 01100000 60H

作为练习，大家写出 A-F 的字形代码。

本来这里是讲解单片机数码管显示器的静态接口的，到此应当可算结束了，但是我还想接着上面讲到的数的本质的问题再谈一点。单片机中有一些术语、名词本来是帮助我们理解事物的，但有时我们会被这些术语的相关语义所迷惑，以致不能进一步认清他们的本质，由此一般陷入困惑的境界。只有深入地了解了 74LS164 的工作特性，才能真正理解何谓串行的数据。

24 课:动态扫描显示接口电路及程序

在单片机系统中动态扫描显示接口是单片机中应用最为广泛的一种显示方式之一。其接口电路是把所有显示器的 8 个笔划段 a-h 同名端连在一起，而每一个显示器的公共极 COM 是各自独立地受 I/O 线控制。CPU 向字段输出口送出字形码时，所有显示器接收到相同的字形码，但究竟是那个显示器亮，则取决于 COM 端，而这一端是由 I/O 控制的，所以我们就能自行决定何时显示哪一位了。而所谓动态扫描就是指我们采用分时的办法，轮流控制各个显示器的 COM 端，使各个显示器轮流点亮。在 http://www.51hei.com 还有很多关于单片机显示接口的文章，大家可以参考一下

在轮流点亮扫描过程中，每位显示器的点亮时间是极为短暂的（约 1ms），但由于人的视觉暂留现象及发光二极管的余辉效应，尽管实际上各位显示器并非同时点亮，但只要扫描的速度足够快，给人的印象就是一组稳定的显示数据，不会有闪烁感。

下图所示就是我们的单片机实验板上的动态扫描接口。由 89c51 的 P0 口能灌入较大的电流，所以我们采用共阳的数码管，并且不用限流电阻，而只是用两只 1N4004 进行降压后给数码管供电，这里仅用了两只，实际上还能扩充。它们的公共端则由 PNP 型三极管 8550 控制，显然，如果 8550 导通，则对应的数码管就能亮，而如果 8550 截止，则对应的数码管就不可能亮，8550 是由 P2.7，P2.6 控制的。这样我们就能通过控制 P27、P26 达到控制某个数码管亮或灭的目的。

下面的这个单片机程序，就是用实验板上的数码管显示 0 和 1。

```
FIRST EQU P2.7 ;第一位数码管的位控制

SECOND EQU P2.6 ;第二位数码管的位控制

DISPBUFF EQU 5AH ;显示缓冲区为 5AH 和 5BH

ORG 0000H

AJMP START

ORG 30H

START:

MOV SP,#5FH ;设置堆栈

MOV P1,#0FFH

MOV P0,#0FFH

MOV P2,#0FFH ;初始化，所显示器，LED 灭

MOV DISPBUFF,#0 ;第一位显示 0

MOV DISPBUFF+1,#1 ;第二握显示 1

LOOP:

LCALL DISP ;调用显示程序
```

```
AJMP LOOP
;主程序到此结束
DISP:
PUSH ACC ;ACC 入栈
PUSH PSW ;PSW 入栈
MOV A,DISPBUFF ;取第一个待显示数
MOV DPTR,#DISPTAB ;字形表首地址
MOVC A,@A+DPTR ;取字形码
MOV P0,A ;将字形码送 P0 位（段口）
CLR FIRST ;开第一位显示器位口
LCALL DELAY ;延时 1 毫秒
SETB FIRST ;关闭第一位显示器（开始准备第二位的数据）
MOV A,DISPBUFF+1 ;取显示缓冲区的第二位
MOV DPTR,#DISPTAB
MOVC A,@A+DPTR
MOV P0,A ;将第二个字形码送 P0 口
CLR SECOND ;开第二位显示器
LCALL DELAY ;延时
SETB SECOND ;关第二位显示
POP PSW
POP ACC
RET
DELAY: ;延时 1 毫秒
PUSH PSW
SETB RS0
MOV R7,#50
D1: MOV R6,#10
D2: DJNZ R6,$
DJNZ R7,D1
POP PSW
```

```
RET

DISPTAB:DB 28H,7EH,0a4H,64H,72H,61H,21H,7CH,20H,60H

END
```

从上面的单片机例程中能看出，动态扫描显示必须由 CPU 持续地调用显示程序，才能保证持续持续的显示。

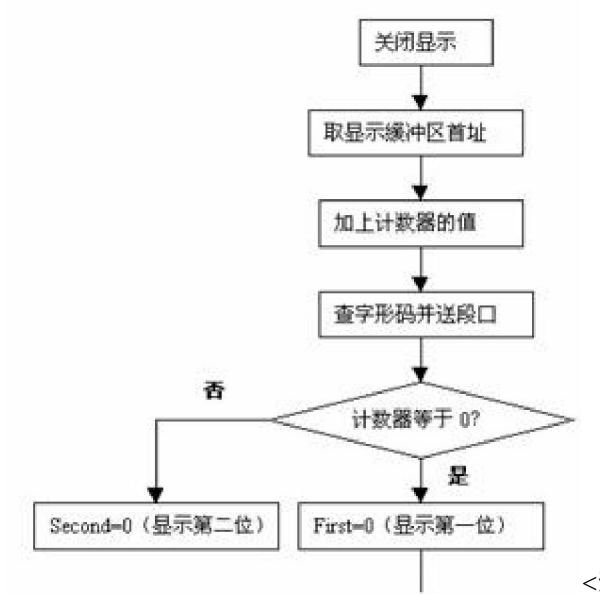
上面的这个程序能实现数字的显示，但不太实用，为什么呢？这里仅是显示两个数字，并没有做其他的工作，因此，两个数码管轮流显示 1 毫秒，没有问题，实际的工作中，当然不可能只显示两个数字，还是要做其他的事情的，这样在二次调用显示程序之间的时间间隔就不一不定了，如果时间间隔比较长，就会使显示不连续。而实际工作中是很难保证所有工作都能在很短时间内完成的。况且这个显示程序也有点“浪费”，每个数码管显示都要占用 1 个毫秒的时间，这在很多合是不允许的，怎么办呢？我们能借助于定时器，定时时间一到，产生中断，点亮一个数码管，然后马上返回，这个数码管就会一直亮到下一次定时时间到，而不用调用延时程序了，这段时间能留给主程序干其他的事。到下一次定时时间到则显示下一个数码管，这样就很少浪费了。

```
Counter EQU 59H ;计数器，显示程序通过它得知现正显示哪个数码管

FIRST EQU P2.7 ;第一位数码管的位控制

SECOND EQU P2.6 ;第二位数码管的位控制

DISPBUFF EQU 5AH ;显示缓冲区为 5AH 和 5BH

ORG 0000H

AJMP START

ORG 000BH ;定时器 T0 的入口

AJMP DISP ;显示程序

ORG 30H

START:

MOV SP,#5FH ;设置堆栈

MOV P1,#0FFH

MOV P0,#0FFH

MOV P2,#0FFH ;初始化，所显示器，LED 灭

MOV TMOD,#00000001B ;定时器 T0 工作于模式 1（16 位定时/计数模式）

MOV TH0,#HIGH(65536-2000)

MOV TL0,#LOW(65536-2000)

SETB TR0

SETB EA
```

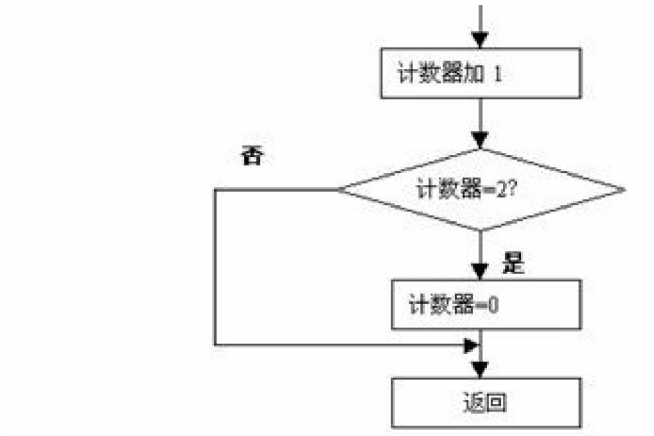
```
SETB ET0
MOV Counter,#0 ;计数器初始化
MOV DISPBUFF,#0 ;第一位始终显示 0
MOV A,#0
LOOP:
MOV DISPBUFF+1,A ;第二位轮流显示 0-9
INC A
LCALL DELAY
CJNE A,#10,LOOP
MOV A,#0
AJMP LOOP ;在此中间能按排任意程序，这里仅作示范。
;主程序到此结束
DISP: ;定时器 T0 的中断响应程序
PUSH ACC ;ACC 入栈
PUSH PSW ;PSW 入栈
MOV TH0,#HIGH(65536-2000) ;定时时间为 2000 个周期，约 2170 微秒（11.0592M）
MOV TL0,#LOW(65536-2000)
SETB FIRST
SETB SECOND ;关显示
MOV A,#DISPBUFF ;显示缓冲区首地址
ADD A,Counter
MOV R0,A
MOV A,@R0 ;根据计数器的值取对应的显示缓冲区的值
MOV DPTR,#DISPTAB ;字形表首地址
MOVC A,@A+DPTR ;取字形码
MOV P0,A ;将字形码送 P0 位（段口）
MOV A,Counter ;取计数器的值
JZ DISPFIRST ;如果是 0 则显示第一位
CLR SECOND ;不然显示第二位
AJMP DISPNEXT
```

```
DISPFIRST:
CLR FIRST ;显示第一位
DISPNEXT:
INC Counter ;计数器加 1
MOV A,Counter
DEC A ;如果计数器计到 2，则让它回 0
DEC A
JZ RSTCOUNT
AJMP DISPEXIT
RSTCOUNT:
MOV Counter,#0 ;计数器的值只能是 0 或 1
DISPEXIT:
POP PSW
POP ACC
RETI
DELAY: ;延时 130 毫秒
PUSH PSW
SETB RS0
MOV R7,#255
D1: MOV R6,#255
D2: NOP
NOP
NOP
NOP
DJNZ R6,D2
DJNZ R7,D1
POP PSW
RET
DISPTAB:DB 28H,7EH,0a4H,64H,72H,61H,21H,7CH,20H,60H
END
```

从上面的单片机程序能看出，动态显示和静态显示相比，程序稍有点复杂，不过，这是值得的。这个程序有一定的通用性，只要改变端口的值及计数器的值就能显示更多位数了。



下面给出显示程序的流程图。 <动态扫描程序框图>



25 课:单片机键盘接口程序设计

键盘是由若干按钮组成的开关矩阵，它是单片机系统中最常用的输入设备，用户能通过键盘向计算机输入指令、地址和数据。一般单片机系统中采和非编码键盘，非编码键盘是由软件来识别键盘上的闭合键，它具有结构简单，使用灵活等特点，因此被广泛应用于单片机系统。

按钮开关的抖动问题

组成键盘的按钮有触点式和非触点式两种，单片机中应用的一般是由机械触点组成的。在下图中， <键盘结构图>

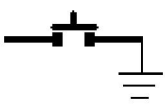
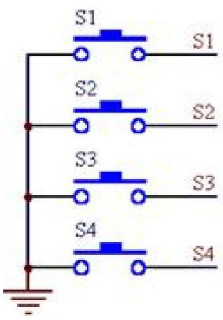


图 1

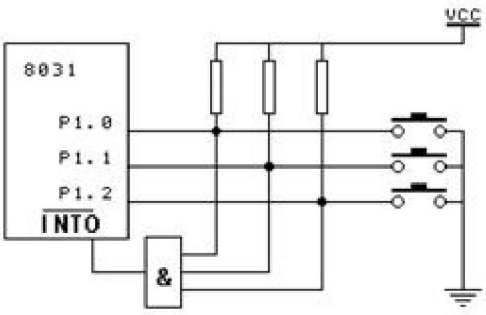
图 2

当开关 S 未被按下时，P1。0 输入为高电平，S 闭合后，P1。0 输入为低电平。由于按钮是机械触点，当机械触点断开、闭合时，会有抖动动，P1。0 输入端的波形如图 2 所示。这种抖动对于人来说是感觉不到的，但对计算机来说，则是完全能感应到的，因为计算机处理的速度是在微秒级，而机械抖动的时间至少是毫秒级，对计算机而言，这已是一个“漫长”的时间了。前面我们讲到中断时曾有个问题，就是说按钮有时灵，有时不灵，其实就是这个原因，你只按了一次按钮，可是计算机却已执行了多次中断的过程，如果执行的次数正好是奇数次，那么结果正如你所料，如果执行的次数是偶数次，那就不对了。

为使 CPU 能正确地读出 P1 口的状态，对每一次按钮只作一次响应，就必须考虑如何去除抖动，常用的去抖动的办法有两种：硬件办法和软件办法。单片机中常用软件法，因此，对于硬件办法我们不介绍。软件法其实很简单，就是在单片机获得 P1。0 口为低的信息后，不是立即认定 S1 已被按下，而是延时 10 毫秒或更长一些时间后再次检测 P1。0 口，如果仍为低，说明 S1 的确按下了，这实际上是避开了按钮按下时的抖动时间。而在检测到按钮释放后（P1。0 为高）再延时 5-10 个毫秒，消除后沿的抖动，然后再对键值处理。不过一般情况下，我们常常不对按钮释放的后沿进行处理，实践证明，也能满足一定的要求。当然，实际应用中，对按钮的要求也是千差万别，要根据不一样的需要来编制处理程序，但以上是消除键抖动的原则。 键盘与单片机的连接



<键盘连接>图 3



<单片机与键盘接口图>图 4

1、通过 1/0 口连接。将每个按钮的一端接到单片机的 I/O 口，另一端接地，这是最简单的办法，如图 3 所示是实验板上按钮的接法，四个按钮分别接到 P3.2 、P3.3、P3.4 和 P3.5。对于这种键各程序能采用持续查询的办法，功能就是：检测是否有键闭合，如有键闭合，则去除键抖动，判断键号并转入对应的键处理。下面给出一个例程。四个键定义如下：
P3.2：开始，按此键则灯开始流动（由上而下）
P3.3：停止，按此键则停止流动，所有灯为暗
P3.4：上，按此键则灯由上向下流动
P3.5：下，按此键则灯由下向上流动

```
UpDown EQU 00H ;上下行标志
StartEnd EQU 01H ;起动及停止标志

LAMPCODE EQU 21H ;存放流动的数据代码

ORG 0000H

AJMP MAIN

ORG 30H

MAIN:

MOV SP,#5FH

MOV P1,#0FFH

CLR UpDown ;启动时处于向上的状态

CLR StartEnd ;启动时处于停止状态

MOV LAMPCODE,#0FEH ;单灯流动的代码

LOOP:

ACALL KEY ;调用键盘程序

JNB F0,LNEXT ;如果无键按下，则继续

ACALL KEYPROC ;不然调用键盘处理程序

LNEXT:

ACALL LAMP ;调用灯显示程序

AJMP LOOP ;反复循环，主程序到此结束

DELAY:

MOV R7,#100

D1: MOV R6,#100

DJNZ R6,$

DJNZ R7,D1

RET;----------延时程序，键盘处理中调用
```

```
KEYPROC:
MOV A,B ;从 B 寄存器中获取键值
JB ACC.2,KeyStart ;分析键的代码，某位被按下，则该位为 1（因为在键盘程序中已取反）
JB ACC.3,KeyOver
JB ACC.4,KeyUp
JB ACC.5,KeyDown
AJMP KEY_RET
KeyStart:
SETB StartEnd ;第一个键按下后的处理
AJMP KEY_RET
KeyOver:
CLR StartEnd ;第二个键按下后的处理
AJMP KEY_RET
KeyUp: SETB UpDown ;第三个键按下后的处理
AJMP KEY_RET
KeyDown:
CLR UpDown ;第四个键按下后的处理
KEY_RET:RET
KEY:
CLR F0 ;清 F0，表示无键按下。
ORL P3,#00111100B ;将 P3 口的接有键的四位置 1
MOV A,P3 ;取 P3 的值
ORL A,#11000011B ;将其余 4 位置 1
CPL A ;取反
JZ K_RET ;如果为 0 则一定无键按下
ACALL DELAY ;不然延时去键抖
ORL P3,#00111100B
MOV A,P3
ORL A,#11000011B
CPL A
```

```
JZ K_RET
MOV B,A ;确实有键按下，将键值存入 B 中
SETB F0 ;设置有键按下的标志
K_RET:
ORL P3,#00111100B ;此处循环等待键的释放
MOV A,P3
ORL A,#11000011B
CPL A
JZ K_RET1 ;直到读取的数据取反后为 0 说明键释放了，才从键盘处理程序中返回
AJMP K_RET
K_RET1:
RET
D500MS: ;流水灯的延迟时间
PUSH PSW
SETB RS0
MOV R7,#200
D51: MOV R6,#250
D52: NOP
NOP
NOP
NOP
DJNZ R6,D52
DJNZ R7,D51
POP PSW
RET
LAMP:
JB StartEnd,LampStart ;如果 StartEnd=1，则启动
MOV P1,#0FFH
AJMP LAMPRET ;不然关闭所有显示，返回
LampStart:
```

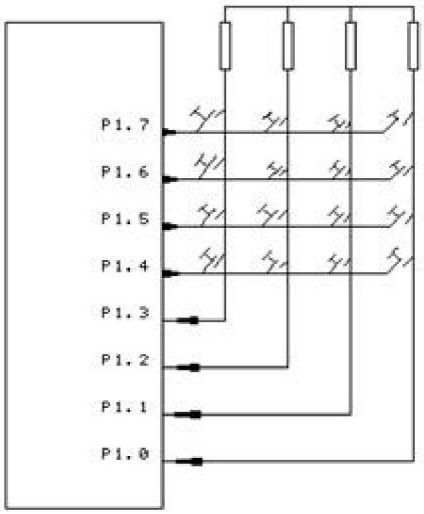
```
JB UpDown,LAMPUP ;如果 UpDown=1，则向上流动
MOV A,LAMPCODE
RL A ;实际就是左移位而已
MOV LAMPCODE,A
MOV P1,A
LCALL D500MS
AJMP LAMPRET
LAMPUP:
MOV A,LAMPCODE
RR A ;向下流动实际就是右移
MOV LAMPCODE,A
MOV P1,A
LCALL D500MS
LAMPRET:
RET
END
```

以上程序功能很简单，但它演示了一个单片机键盘处理程序的基本思路，程序本身很简单，也不很实用，实际工作中还会有好多要考虑的因素，比如主循环每次都调用灯的循环程序，会造成按钮反应“迟钝”，而如果一直按着键不放，则灯不会再流动，一直要到松开手为止，等等，大家能仔细考虑一下这些问题，再想想有什么好的解决办法。

2、采用中断方式：如图 4 所示。各个按钮都接到一个与非上，当有任何一个按钮按下时，都会使与门输出为低电平，从而引起单片机的中断，它的好处是不用在主程序中持续地循环查询，如果有键按下，单片机再去做对应的处理

26 课:矩阵式键盘接口技术及程序设计

在单片机系统中键盘中按钮数量较多时，为了减少 I/O 口的占用，常常将按钮排列成矩阵形式，如图 1 所示。在矩阵式键盘中，每条水平线和垂直线在交叉处不直接连通，而是通过一个按钮加以连接。这样，一个端口（如 P1 口）就能组成 4*4=16 个按钮，比之直接将端口线用于键盘多出了一倍，而且线数越多，区别越明显，比如再多加一条线就能组成 20 键的键盘，而直接用端口线则只能多出一键（9 键）。由此可见，在需要的键数比较多时，采用矩阵法来做键盘是合理的。



<单片机矩阵式键盘接口技术及编程接口图>

矩阵式结构的键盘显然比直接法要复杂一些，识别也要复杂一些，上图中，列线通过电阻接正电源，并将行线所接的单片机的 I/O 口作为输出端，而列线所接的 I/O 口则作为输入。这样，当按钮没有按下时，所有的输出端都是高电平，代表无键按下。行线输出是低电平，一旦有键按下，则输入线就会被拉低，这样，通过读入输入线的状态就可得知是否有键按下了。具体的识别及编程办法如下所述。

矩阵式键盘的按钮识别办法

确定矩阵式键盘上何键被按下介绍一种“行扫描法”。

行扫描法 行扫描法又称为逐行（或列）扫描查询法，是一种最常用的按钮识别办法，如上图所示键盘，介绍过程如下。

判断键盘中有无键按下 将全部行线 Y0-Y3 置低电平，然后检测列线的状态。只要有一列的电平为低，则表示键盘中有键被按下，而且闭合的键位于低电平线与 4 根行线相交叉的 4 个按钮之中。若所有列线均为高电平，则键盘中无键按下。

判断闭合键所在的位置 在确认有键按下后，即可进入确定具体闭合键的过程。其办法是：依次将行线置为低电平，即在置某根行线为低电平时，其它线为高电平。在确定某根行线位置为低电平后，再逐行检测各列线的电平状态。若某列为低，则该列线与置为低电平的行线交叉处的按钮就是闭合的按钮。

下面给出一个具体的例程：

图仍如上所示。8031 单片机的 P1 口用作键盘 I/O 口，键盘的列线接到 P1 口的低 4 位，键盘的行线接到 P1 口的高 4 位。列线 P1.0-P1.3 分别接有 4 个上拉电阻到正电源+5V，并把列线 P1.0-P1.3 设置为输入线，行线 P1.4-P.17 设置为输出线。4 根行线和 4 根列线形成 16 个相交点。

检测当前是否有键被按下。检测的办法是 P1.4-P1.7 输出全“0”，读取 P1.0-P1.3 的状态，若 P1.0-P1.3 为全“1”，则无键闭合，不然有键闭合。

去除键抖动。当检测到有键按下后，延时一段时间再做下一步的检测判断。

若有键被按下，应识别出是哪一个键闭合。办法是对键盘的行线进行扫描。P1.4-P1.7 按下述 4 种组合依次输出：

P1.7　1 1 1 0

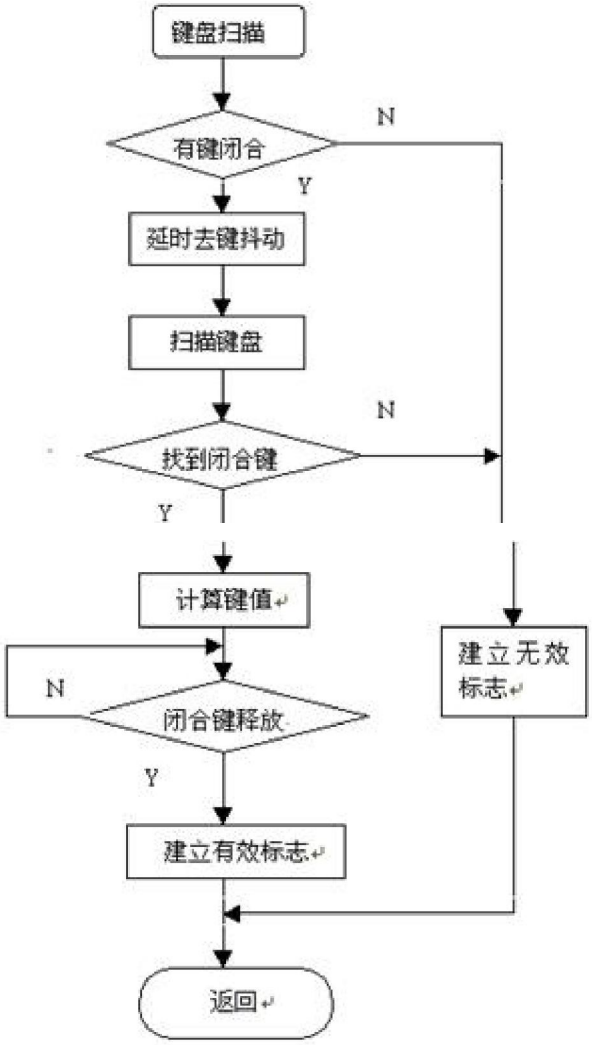
P1.6　1 1 0 1

P1.5　1 0 1 1

P1.4　0 1 1 1

在每组行输出时读取 P1.0-P1.3，若全为“1”，则表示为“0”这一行没有键闭合，不然有键闭合。由此得到闭合键的行值和列值，然后可采用计算法或查表法将闭合键的行值和列值转换成所定义的键值

为了保证键每闭合一次 CPU 仅作一次处理，必须却除键释放时的抖动。



《单片机矩阵式键盘接口技术及编程》

键盘扫描程序：

从以上分析得到单片机键盘扫描程序的流程图如图 2 所示。程序如下

```
SCAN: MOV P1,#0FH
MOV A,P1
ANL A,#0FH
CJNE A,#0FH,NEXT1
SJMP NEXT3
NEXT1: ACALL D20MS
MOV A,#0EFH
NEXT2: MOV R1,A
```

```
MOV P1,A
MOV A,P1
ANL A,#0FH
CJNE A,#0FH,KCODE;
MOV A,R1
SETB C
RLC A
JC NEXT2
NEXT3: MOV R0,#00H
RET
KCODE: MOV B,#0FBH
NEXT4: RRC A
INC B
JC NEXT4
MOV A,R1
SWAP A
NEXT5: RRC A
INC B
INC B
INC B
INC B
JC NEXT5
NEXT6: MOV A,P1
ANL A,#0FH
CJNE A,#0FH,NEXT6
MOV R0,#0FFH
RET
```

键盘处理程序就作这么一个简单的介绍，实际上，键盘、显示处理是很复杂的，它一般占到一个应用程序的大部份代码，可见其重要性，但说到，这种复杂并不来自于单片机的本身，而是来自于操作者的习惯等等问题，因此，在编写键盘处理程序之前，最好先把它从逻辑上理清，然后用适当的算法表示出来，最后再去写代码，这样，才能快速有效地写好代码。到本课为止，本教程暂告一个段落！

27 课:关于单片机的一些基本概念

随着电子技术的迅速发展，计算机已深入地渗透到我们的生活中，许多电子爱好者开始学习单片机知识，但单片机的内容比较抽象，相对电子爱好者已熟悉的模拟电路、数字电路，单片机中有一些新的概念，这些概念非常基本以至于一般作者不屑去谈，教材自然也不会很深入地讲解这些概念，但这些内容又是学习中必须要理解的，下面就结合本人的学习、教学经验，对这些最基本概念作一说明，希望对自学者有所帮助。

一、总线：我们知道，一个电路总是由元器件通过电线连接而成的，在模拟电路中，连线并不成为一个问题，因为各器件间一般是串行关系，各器件之间的连线并不很多，但计算机电路却不一样，它是以微处理器为核心，各器件都要与微处理器相连，各器件之间的工作必须相互协调，所以就需要的连线就很多了，如果仍如同模拟电路一样，在各微处理器和各器件间单独连线，则线的数量将多得惊人，所以在微处理机中引入了总线的概念，各个器件共同享用连线，所有器件的 8 根数据线全部接到 8 根公用的线上，即相当于各个器件并联起来，但仅这样还不行，如果有两个器件同时送出数据，一个为 0，一个为 1，那么，接收方接收到的究竟是什么呢？这种情况是不允许的，所以要通过控制线进行控制，使器件分时工作，任何时候只能有一个器件发送数据（能有多个器件同时接收）。器件的数据线也就被称为数据总线，器件所有的控制线被称为控制总线。在单片机内部或者外部存储器及其它器件中有存储单元，这些存储单元要被分配地址，才能使用，分配地址当然也是以电信号的形式给出的，由于存储单元比较多，所以，用于地址分配的线也较多，这些线被称为地址总线。

二、数据、地址、指令：之所以将这三者放在一起，是因为这三者的本质都是一样的——数字，或者说都是一串‘0’和‘1’组成的序列。换言之，地址、指令也都是数据。指令：由单片机芯片的设计者规定的一种数字，它与我们常用的指令助记符有着严格的一一对应关系，不能由单片机的开发者更改。地址：是寻找单片机内部、外部的存储单元、输入输出口的依据，内部单元的地址值已由芯片设计者规定好，不可更改，外部的单元能由单片机开发者自行决定，但有一些地址单元是一定要有的（详见程序的执行过程）。数据：这是由微处理机处理的对象，在各种不一样的应用电路中各不相同，一般而言，被处理的数据可能有这么几种情况：

1·地址（如 MOV DPTR，#1000H），即地址 1000H 送入 DPTR。

2·方式字或控制字（如 MOV TMOD，#3），3 即是控制字。

3·常数（如 MOV TH0，#10H）10H 即定时常数。

4·实际输出值（如 P1 口接彩灯，要灯全亮，则执行指令：MOV P1，#0FFH，要灯全暗，则执行指令：MOV P1，#00H）这里 0FFH 和 00H 都是实际输出值。又如用于 LED 的字形码，也是实际输出的值。

理解了地址、指令的本质，就不难理解程序运行过程中为什么会跑飞，会把数据当成指令来执行了。

三、P0 口、P2 口和 P3 的第二功能使用办法 开始学习时一般对 P0 口、P2 口和 P3 口的第二功能使用办法迷惑不解，认为第二功能和原功能之间要有一个切换的过程，或者说要有一条指令，事实上，各端口的第二功能完全是自动的，不需要用指令来转换。如 P3.6、P3.7 分别是 WR、RD 信号，当微片理机外接 RAM 或有外部 I/O 口时，它们被用作第二功能，不能作为通用 I/O 口使用，只要一微处理机一执行到 MOVX 指令，就会有对应的信号从 P3.