单片机教程第十六课：计数器与定时器

一、计数概念的引入

从选票的统计谈起：画“正”。这就是计数，生活中计数的例子处处可见。例：录音机上的计数器、家里面用的电度表、汽车上的里程表等等，再举一个工业生产中的例子，线缆行业在电线生产出来之后要计米，也就是测量长度，怎么测法呢？用尺量？不现实，太长不说，要一边做一边量呢，怎么办呢？行业中有很巧妙的方法，用一个周长是 1 米的轮子，将电缆绕在上面一周，由线带轮转，这样轮转一周不就是线长 1 米嘛，所以只要记下轮转了多少圈，就可以知道走过的线有多长了。

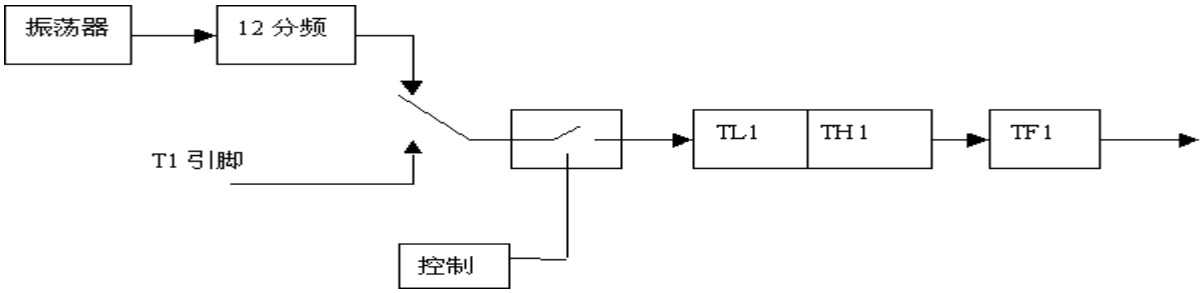
二、计数器的容量

从一个生活中的例子看起：一个水盆在水龙头下，水龙没关紧，水一滴滴地滴入盆中。水滴不断落下，盆的容量是有限的，过一段时间之后，水就会逐渐变满。录音机上的计数器最多只计到 999 ...那么单片机中的计数器有多大的容量呢？8031 单片机中有两个计数器，分别称之为 T0 和 T1，这两个计数器分别是由两个 8 位的 RAM 单元组成的，即每个计数器都是 16 位的计数器，最大的计数量是 65536。

三、定时

8031 中的计数器除了可以作为计数之用外，还可以用作时钟，时钟的用途当然很大，如打铃器，电视机定时关机，空调定时开关等等，那么计数器是如何作为定时器来用的呢？

一个闹钟，我将它定时在 1 个小时后闹响，换言之，也可以说是秒针走了（3600）次，所以时间就转化为秒针走的次数的，也就是计数的次数了，可见，计数的次数和时间之间的确十分相关。那么它们的关系是什么呢？那就是秒针每一次走动的时间正好是 1 秒。



结论：只要计数脉冲的间隔相等，则计数值就代表了时间的流逝。由此，单片机中的定时器和计数器是一个东西，只不过计数器是记录的外界发生的事情，而定时器则是由单片机提供一个非常稳定的计数源。那么提供组定时器的是计数源是什么呢？看图 1，原来就是由单片机的晶振经过 12 分频后获得的一个脉冲源。晶振的频率当然很准，所以这个计数脉冲的时间间隔也很准。问题：一个 12M 的晶振，它提供给计数器的脉冲时间间隔是多少呢？当然这很容易，就是 12M/12 等于 1M，也就是 1 个微秒。结论：计数脉冲的间隔与晶振有关，12M 的晶振，计数脉冲的间隔是 1 微秒。

四、溢出

让我们再来看水滴的例子，当水不断落下，盆中的水不断变满，最终有一滴水使得盆中的水满了。这时如果再有一滴水落下，就会发生什么现象？水会漫出来，用个术语来讲就是“溢出”。

水溢出是流到地上，而计数器溢出后将使得 TF0 变为“1”。至于 TF0 是什么我们稍后再谈。一旦 TF0 由 0 变成 1，就是产生了变化，产生了变化就会引发事件，就象定时的时间一到，闹钟就会响一样。至于会引发什么事件，我们下次课再介绍，现在我们来研究另一个问题：要有多少个计数脉冲才会使 TF0 由 0 变为 1。

五、任意定时及计数的方法

刚才已研究过，计数器的容量是 16 位，也就是最大的计数值到 65536，因此计数计到 65536 就会产生溢出。这个没有问题，问题是我们现实生活中，经常会有少于 65536 个计数值的要求，如包装线上，一打为 12 瓶，一瓶药片为 100 粒，怎么样来满足这个要求呢？

提示：如果是一个空的盆要 1 万滴水滴进去才会满，我在开始滴水之前就先放入一勺水，还需要 10000 滴嘛？对了，我们采用预置数的方法，我要计 100，那我就先放进 65436，再来 100 个脉冲，不就到了 65536 了吗。定时也是如此，每个脉冲是 1 微秒，则计满 65536 个脉冲需时 65.536 毫秒，但现在我只要 10 毫秒就可以了，怎么办？10 个毫秒为 10000 个微秒，所以，只要在计数器里面放进 55536 就可以了。

单片机教程第十七课：定时/计数器的方式控制字

从上一节我们已经得知，单片机中的定时/计数器都可以有多种用途，那么我怎样才能让它们工作于我所需要的用途呢？这就要通过定时/计数器的方式控制字来设置。

在单片机中有两个特殊功能寄存器与定时/计数有关，这就是 TMOD 和 TCON。顺便说一下，TMOD 和 TCON 是名称，我们在写程序时就可以直接用这个名称来指定它们，当然也可以直接用它们的地址 89H 和 88H 来指定它们（其实用名称也就是直接用地址，汇编软件帮你翻译一下而已）。

用于 T1				用于 T0			
GATE	$C/\overline{T}$	M1	M0	GATE	$C/\overline{T}$	M1	M0

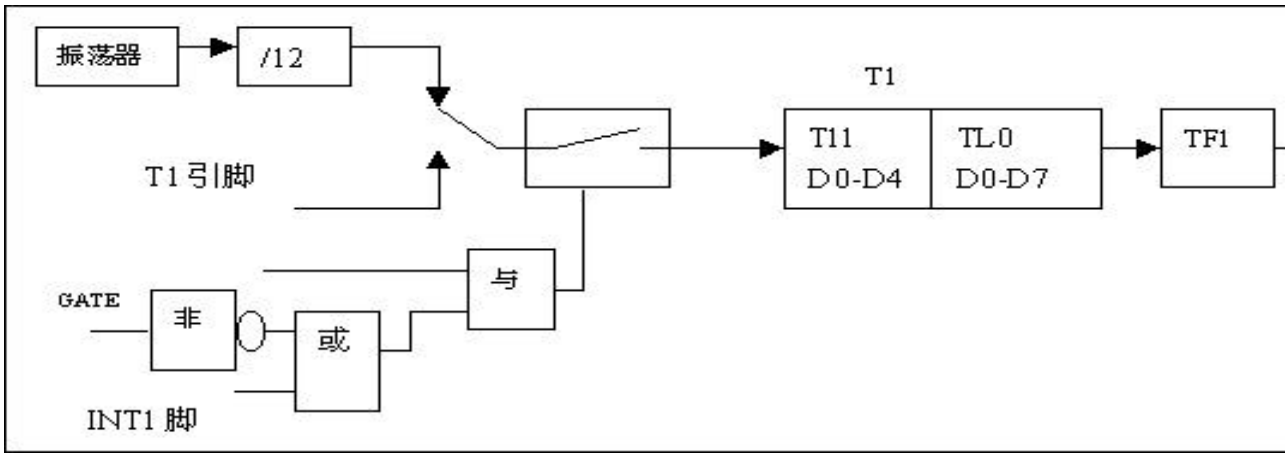
图 1（TMOD）

从图 1 中我们可以看出，TMOD 被分成两部份，每部份 4 位。分别用于控制 T1 和 T0，至于这里面是什么意思，我们下面介绍。

用于定时/计数器				用于中断			
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

图 2（TCON）

从图 2 中我们可以看出，TCON 也被分成两部份，高 4 位用于定时/计数器，低 4 位则用于中断（我们暂不管）。而 TF1（0）我们上节课已提到了，当计数溢出后 TF1（0）就由 0 变为 1。原来 TF1（0）在这儿！那么 TR0、TR1 又是什么呢？看上节课的图。



计数脉冲要进入计数器还真不容易，有层层关要通过，最起码，就是 TR0（1）要为 1，开关才能合上，脉冲才能过来。因此，TR0（1）称之为运行控制位，可用指令 SETB 来置位以启动计数器/定时器运行，用指令 CLR 来

关闭定时/计数器的工作，一切尽在自己的掌握中。

定时/计数器的四种工作方式

工作方式 0

定时器/计数器的工作方式 0 称之为 13 位定时/计数方式。它由 TL（1/0）的低 5 位和

TH（0/1）的 8 位构成 13 位的计数器，此时 TL（1/0）的高 3 位未用。

我们用这个图来讨论几个问题：

M1M0：定时/计数器一共有四种工作方式，就是用 M1M0 来控制的，2 位正好是四种组合。

C/T：前面我们说过，定时/计数器即可作定时用也可用计数用，到底作什么用，由我们根据需要自行决定，也说是决定权在我们&#0;&#0;编程者。如果 C/T 为 0 就是用作定时器（开关往上打），如果 C/T 为 1 就是用作计数器（开关往下打）。顺便提一下：一个定时/计数器同一时刻要么作定时用，要么作计数用，不能同时用的，这是个极普通的常识，几乎没有教材会提这一点，但很多初学者却会有此困惑。

GATE：看图，当我们选择了定时或计数工作方式后，定时/计数脉冲却不一定能到达计数器端，中间还有一个开关，显然这个开关不合上，计数脉冲就没法过去，那么开关什么时候过去呢？有两种情况

GATE=0，分析一下逻辑，GATE 非后是 1，进入或门，或门总是输出 1，和或门的另一个输入端 INT1 无关，在这种情况下，开关的打开、合上只取决于 TR1，只要 TR1 是 1，开关就合上，计数脉冲得以畅通无阻，而如果 TR1 等于 0 则开关打开，计数脉冲无法通过，因此定时/计数是否工作，只取决于 TR1。

GATE=1，在此种情况下，计数脉冲通路上的开关不仅要由 TR1 来控制，而且还要受到 INT1 引脚的控制，只有 TR1 为 1，且 INT1 引脚也是高电平，开关才合上，计数脉冲才得以通过。这个特性可以用来测量一个信号的高电平的宽度，想想看，怎么测？

为什么在这种模式下只用 13 位呢？干吗不用 16 位，这是为了和 51 机的前辈 48 系列兼容而设的一种工作式，如果你觉得用得不顺手，那就干脆用第二种工作方式。

工作方式 1

工作方式 1 是 16 位的定时/计数方式，将 M1M0 设为 01 即可，其它特性与工作方式 0 相同。

工作方式 2

在介绍这种式方式之前先让我们思考一个问题：上一次课我们提到过任意计数及任意定时的问题，比如我要计 1000 个数，可是 16 位的计数器要计到 65536 才满，怎么办呢？我们讨论后得出的办法是用预置数，先在计数器里放上 64536，再来 1000 个脉冲，不就行了吗？是的，但是计满了之后我们又该怎么办呢？要知道，计数总是不断重复的，流水线上计满后马上又要开始下一次计数，下一次的计数还是 1000 吗？当计满并溢出后，计数器里面的值变成了 0（为什么，可以参考前面课程的说明），因此下一次将要计满 65536 后才会溢出，这可不符合要求，怎么办？当然办法很简单，就是每次一溢出时执行一段程序（这通常是需要的，要不然要溢出干吗？）可以在这段程序中做把预置数 64536 送入计数器中的事情。所以采用工作方式 0 或 1 都要在溢出后做一个重置预置数的工作，做工作当然就得要时间，一般来说这点时间不算什么，可是有一些场合我们还是要计较的，所以就有了第三种工作方式&#0;&#0;自动再装入预置数的工作方式。

既然要自动得新装入预置数，那么预置数就得放在一个地方，要不然装什么呢？那么预置数放在什么地方呢？它放在 T（0/1）的高 8 位，那么这样高 8 位不就不能参与计数了吗？是的，在工作方式 2，只有低 8 位参与计数，而高 8 位不参与计数，用作预置数的存放，这样计数范围就小多了，当然做任可事总有代价的，关键是看值不值，如果我根本不需要计那么多数，那么就可以用这种方式。看图 4，每当计数溢出，就会打开 T（0/1）的高、低 8 位之间的开关，计预置数进入低 8 位。这是由硬件自动完成的，不需要由人工干预。

通常这种式作方式用于波特率发生器（我们将在串行接口中讲解），用于这种用途时，定时器就是为了提供一个时间基准。计数溢出后不需要做事情，要做的仅仅只有一件，就是重新装入预置数，再开始计数，而且中间不要任何延迟，可见这个任务用工作方式 2 来完成是最妙不过了。

工作方式 3

这种式作方式之下，定时/计数器 0 被拆成 2 个独立的定时/计数器来用。其中，TL0 可以构成 8 位的定时器或计数器的工作方式，而 TH0 则只能作为定时器来用。我们知道作定时、计数器来用，需要控制，计满后溢出需要有溢出标记，T0 被分成两个来用，那就要两套控制及、溢出标记了，从何而来呢？TL0 还是用原来的 T0 的标记，而 TH0 则借用 T1 的标记。如此 T1 不是无标记、控制可用了吗？是的。

一般情况处，只有在 T1 以工作方式 2 运行（当波特率发生器用）时，才让 T0 工作于方式 3 的。

定时器/计数器的定时/计数范围

工作方式 0：13 位定时/计数方式，因此，最多可以计到 2 的 13 次方，也就是 8192 次。

工作方式 1：16 位定时/计数方式，因此，最多可以计到 2 的 16 次方，也就是 65536 次。

工作方式 2 和工作方式 3，都是 8 位的定时/计数方式，因此，最多可以计到 2 的 8 次方，也说是 256 次。

预置值计算：用最大计数量减去需要的计数次数即可。

例：流水线上一个包装是 12 盒，要求每到 12 盒就产生一个动作，用单片机的工作方式 0 来控制，应当预置多大的值呢？对了，就是 8192-12=8180。

以上是计数，明白了这个道理，定时也是一样。这在前面的课程已提到，我们不再重复，请参考前面的例子。

单片机第十八课：中断系统

有关中断的概念

什么是中断，我们从一个生活中的例子引入。你正在家中看书，突然电话铃响了，你放下书本，去接电话，和来电话的人交谈，然后放下电话，回来继续看你的书。这就是生活中的“中断”的现象，就是正常的工作过程被外部的事件打断了。仔细研究一下生活中的中断，对于我们学习单片机的中断也很有好处。第一、什么可经引起中断，生活中很多事件可以引起中断：有人按了门铃了，电话铃响了，你的闹钟闹响了，你烧的水开了….等等诸如此类的事件，我们把可以引起中断的称之为中断源，单片机中也有一些可以引起中断的事件，8031中一共有 5 个：两个外部中断，两个计数/定时器中断，一个串行口中断。

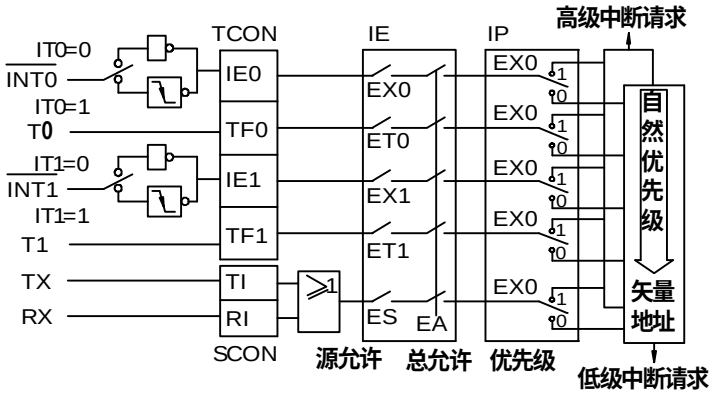
中断的嵌套与优先级处理

设想一下，我们正在看书，电话铃响了，同时又有人按了门铃，你该先做那样呢？如果你正是在等一个很重要的电话，你一般不会去理会门铃的，而反之，你正在等一个重要的客人，则可能就不会去理会电话了。如果不是这两者（即不等电话，也不是等人上门），你可能会按你通常的习惯去处理。总之这里存在一个优先级的问题，单片机中也是如此，也有优先级的问题。优先级的问题不仅仅发生在两个中断同时产生的情况，也发生在一个中断已产生，又有一个中断产生的情况，比如你正接电话，有人按门铃的情况，或你正开门与人交谈，又有电话响了情况。考虑一下我们会怎么办吧。

中断的响应过程

当有事件产生，进入中断之前我们必须先记住现在看书的第几页了，或拿一个书签放在当前页的位置，然后去处理不同的事情（因为处理完了，我们还要回来继续看书）：电话铃响我们要到放电话的地方去，门铃响我们要到门那边去，也说是不同的中断，我们要在不同的地点处理，而这个地点通常还是固定的。计算机中也是采用的这种方法，五个中断源，每个中断产生后都到一个固定的地方去找处理这个中断的程序，当然在去之前首先要保存下面将执行的指令的地址，以便处理完中断后回到原来的地方继续往下执行程序。具体地说，中断响应可以分为以下几个步骤：1、保护断点，即保存下一将要执行的指令的地址，就是把这个地址送入堆栈。2、寻找中断入口，根据 5 个不同的中断源所产生的中断，查找 5 个不同的入口地址。以上工作是由计算机自动完成的，与编程者无关。在这 5 个入口地址处存放有中断处理程序（这是程序编写时放在那儿的，如果没把中断程序放在那儿，就错了，中断程序就不能被执行到）。3、执行中断处理程序。4、中断返回：执行完中断指令后，就从中断处返回到主程序，继续执行。究竟单片机是怎么样找到中断程序所在位置，又怎么返回的呢？我们稍后再谈。

MCS-51 中断系统的结构：



如图所示，由与中断有关的特殊功能寄存器、中断入口、顺序查询逻辑电路等组成，包括 5 个中断请求源，4 个用于中断控制的寄存器 IE、IP、ECON 和 SCON 来控制中断 类弄、中断的开、关和各种中断源的优先级确定。

中断请求源：

（1）外部中断请求源

即外中断 0 和 1，经由外部引脚引入的，在单片机上有两个引脚，名称为 INT0、INT1，也就是 P3.2、P3.3 这两个引脚。在内部的 TCON 中有四位是与外中断有关的。IT0：INT0 触发方式控制位，可由软件进和置位和复位，IT0=0，INT0 为低电平触发方式，IT0=1，INT0 为负跳变触发方式。这两种方式的差异将在以后再谈。IE0：INT0 中断请求标志位。当有外部的中断请求时，这位就会置 1（这由硬件来完成），在 CPU 响应中断后，由硬件将 IE0 清 0。IT1、IE1 的用途和 IT0、IE0 相同。

内部中断请求源 TF0

定时器 T0 的溢出中断标记，当 T0 计数产生溢出时，由硬件置位 TF0。当 CPU 响应中断后，再由硬件将 TF0 清 0。TF1：与 TF0 类似。TI、RI：串行口发送、接收中断，在串口中再讲解。2、中断允许寄存器 IE 在 MCS－51 中断系统中，中断的允许或禁止是由片内可进行位寻址的 8 位中断允许寄存器 IE 来控制的。见下表 EAX

其中 EA 是总开关，如果它等于 0，则所有中断都不允许。ES－串行口中断允许 ET1－定时器 1 中断允许 EX1－外中断 1 中断允许。ET0－定时器 0 中断允许 EX0－外中断 0 中断允许。如果我们要设置允许外中断 1，定时器 1 中断允许，其它不允许，则 IE 可以是 EAX
即 8CH，当然，我们也可以用位操作指令 SETB EA
SETB ET1SETB EX1
来实现它。

五个中断源的自然优先级与中断服务入口地址外中断

0：0003H 定时器 0：000BH 外中断 1：0013H 定时器 1：001BH 串口 ：0023H 它们的自然优先级由高到低排列。写到这里，大家应当明白，为什么前面有一些程序一始我们这样写：
ORG 0000HLJMP START

ORG 0030H
START：。

这样写的目的，就是为了让出中断源所占用的向量地址。当然，在程序中没用中断时，直接从 0000H 开始写程序，在原理上并没有错，但在实际工作中最好不这样做。优先级：单片机采用了自然优先级和人工设置高、低优先级的策略，即可以由程序员设定那些中断是高优先级、哪些中断是低优先级，由于只有两级，必有一些中断处于同一级别，处于同一级别的，就由自然优先级确定。
开机时，每个中断都处于低优先级，我们可以用指令对优先级进行设置。看表 2 中断优先级中由中断优先级寄存器 IP 来高置的，IP 中某位设为 1，相应的中断就是高优先级，否则就是低优先级。
XX
X
PS
PT1
PX1
PT0
PX0
例：设有如下要求，将 T0、外中断 1 设为高优先级，其它为低优先级，求 IP 的值。IP 的首 3 位没用，可任意取值，设为 000，后面根据要求写就可以了 XX

因此，最终，IP 的值就是 06H。例：在上例中，如果 5 个中断请求同时发生，求中断响应的次序。响应次序为：定时器 0－＞外中断 1－＞外中断 0－＞实时器 1－＞串行中断。

MCS－51 的中断响应过程：

1、中断响应的条件

讲到这儿，我们依然对于计算机响应中断感到神奇，我们人可以响应外界的事件，是因为我们有多种“传感器“——眼、耳可以接受不同的信息，计算机是如何做到这点的呢？其实说穿了，一点都不希奇，MCS51 工作时，在每个机器周期中都会去查询一下各个中断标记，看他们是否是“1“，如果是 1，就说明有中断请求了，所以所谓中断，其实也是查询，不过是每个周期都查一下而已。这要换成人来说，就相当于你在看书的时候，每一秒钟都会抬起头来看一看，查问一下，是不是有人按门铃，是否有电话。。。。很蠢，不是吗？可计算机本来就是这样，它根本没人聪明。了解了上述中断的过程，就不难解中断响应的条件了。在下列三种情况之一时，CPU 将封锁对中断的响应：
（1）CPU 正在处理一个同级或更高级别的中断请求。
（2）现行的机器周期不是当前正执行指令的最后一个周期。我们知道，单片机有单周期、双周期、三周期指令，当前执行指令是单字节没有关系，如果是双字节或四字节的，就要等整条指令都执行完了，才能响应中断（因为中断查询是在每个机器周期都可能查到的）。
（3）当前正执行的指令是返回批令（RETI）或访问 IP、IE 寄存器的指令，则 CPU 至少再执行一条指令才应中断。这些都是与中断有关的，如果正访问 IP、IE 则可能会开、关中断或改变中断的优先级，而中断返回指令则说明本次中断还没有处理完，所以都要等本指令处理结束，再执行一条指令才可以响应中断。

2、中断响应过程

CPU 响应中断时，首先把当前指令的下一条指令（就是中断返回后将要执行的指令）的地址送入堆栈，然后根据中断标记，将相应的中断入口地址送入 PC，PC 是程序指针，CPU 取指令就根据 PC 中的值，PC 中是什么值，就会到什么地方去取指令，所以程序就会转到中断入口处继续执行。这些工作都是由硬件来完成的，不必我们去考虑。这里还有个问题，大家是否注意到，每个中断向量地址只间隔了 8 个单元，如 0003－000B，在如此少的空间中如何完成中断程序呢？很简单，你在中断处安排一个 LJMP 指令，不就可以把中断程序跳转到任何地方了吗？一个完整的主程序看起来应该是这样的：

```
ORG 0000HLJMP START
ORG 0003H
LJMP INT0 ；转外中断 0ORG 000BH
RETI ；没有用定时器 0 中断，在此放一条 RETI，万一“不小心“产生了中断，也不会有太大的后果。。
```

中断程序完成后，一定要执行一条 RETI 指令，执行这条指令后，CPU 将会把堆栈中保存着的地址取出，送回 PC，那么程序就会从主程序的中断处继续往下执行了。注意：CPU 所做的保护工作是很有限的，只保护了一个地址，而其它的所有东西都不保护，所以如果你在主程序中用到了如 A、PSW 等，在中断程序中又要用它们，还要保证回到主程序后这里面的数据还是没执行中断以前的数据，就得自己保护起来。

单片机教程第十九课：定时、中断练习一

在学单片机时我们第一个例子就是灯的闪烁，那是用延时程序做的，现在回想起来，这样做不很恰当，为什么呢？我们的主程序做了灯的闪烁，就不能再干其它的事了，难道单片机只能这样工作吗？当然不是，我们可以用定时器来实现灯的闪烁的功能。

例 1：查询方式

```
ORG 0000H
AJMP START
ORG 30H
START:
MOV P1,#0FFH ;关所 灯
MOV TMOD,#00000001B ;定时/计数器 0 工作于方式 1
MOV TH0,#15H
MOV TL0,#0A0H ;即数 5536
SETB TR0 ;定时/计数器 0 开始运行
LOOP:JBC TF0,NEXT ;如果 TF0 等于 1，则清 TF0 并转 NEXT 处
AJMP LOOP ;否则跳转到 LOOP 处运行
NEXT:CPL P1.0
MOV TH0,#15H
MOV TL0,#9FH;重置定时/计数器的初值
AJMP LOOP
END AJMP LOOP
END
```

键入程序，看到了什么？灯在闪烁了，这可是用定时器做的，不再是主程序的循环了。简单地分析一下程序，为什么用 JBC 呢？TF0 是定时/计数器 0 的溢出标记位，当定时器产生溢出后，该位由 0 变 1，所以查询该位就可知宇时时间是否已到。该位为 1 后，要用软件将标记位清 0，以便下一次定时是间到时该位由 0 变 1，所以用了 JBC 指令，该指位在判 1 转移的同时，还将该位清 0。

以上程序是可以实现灯的闪烁了，可是主程序除了让灯闪烁外，还是不能做其他的事啊！不，不对，我们可以在 LOOP：……和 AJMP LOOP 指令之间插入一些指令来做其他的事情，只要保证执行这些指令的时间少于定时时间就行了。那我们在用软件延时程序的时候不是也可以用一些指令来替代 DJNZ 吗？是的，但是那就要求你精确计算所用指令的时间，然后再减去相应的 DJNZ 循环次数，很不方便，而现在只要求所用指令的时间少于定时时间就行，显然要求低了。当然，这样的方法还是不好，所以我们常用以下的方法来实现。

程序 2：用中断实现

```
ORG 0000H
AJMP START
ORG 000BH ;定时器 0 的中断向量地址
AJMP TIME0 ;跳转到真正的定时器程序处
ORG 30H
START:
MOV P1,#0FFH ;关所 灯
```

```
MOV TMOD,#00000001B ;定时/计数器 0 工作于方式 1
MOV TH0,#15H
MOV TL0,#0A0H ;即数 5536
SETB EA ;开总中断允许
SETB ET0 ;开定时/计数器 0 允许
SETB TR0 ;定时/计数器 0 开始运行
LOOP: AJMP LOOP ;真正工作时,这里可写任意程序
TIME0: ;定时器 0 的中断处理程序
PUSH ACC
PUSH PSW ;将 PSW 和 ACC 推入堆栈保护
CPL P1.0
MOV TH0,#15H
MOV TL0,#0A0H ;重置定时常数
POP PSW
POP ACC
RETI
END
```

上面的例子中，定时时间一到，TF0 由 0 变 1，就会引发中断，CPU 将自动转至 000B 处寻找程序并执行，由于留给定时器中断的空间只有 8 个字节，显然不足以写下所有有中断处理程序，所以在 000B 处安排一条跳转指令，转到实际处理中断的程序处，这样，中断程序可以写在任意地方，也可以写任意长度了。进入定时中断后，首先要保存当前的一些状态，程序中只演示了保存存 ACC 和 PSW，实际工作中应该根据需要将可能会改变的单元的值都推入堆栈进行保护（本程序中实际不需保存护任何值，这里只作个演示）。

上面的两个程序运行后，我们发现灯的闪烁非常快，根本分辨不出来，只是视觉上感到灯有些晃动而已，为什么呢？我们可以计算一下，定时器中预置的数是 5536，所以每计 60000 个脉冲就是定时时间到，这 60000 个脉冲的时间是多少呢？我们的晶振是 12M，所以就是 60000 微秒，即 60 毫秒，因此速度是非常快的。如果我想实现一个 1S 的定时，该怎么办呢？在该晶振濒率下，最长的定时也就是 65。536 个毫秒啊！上面给出一个例子。

```
ORG 0000H
AJMP START
ORG 000BH ;定时器 0 的中断向量地址
AJMP TIME0 ;跳转到真正的定时器程序处
ORG 30H
START:
MOV P1,#0FFH ;关所 灯
MOV 30H,#00H ;软件计数器预清 0
MOV TMOD,#00000001B ;定时/计数器 0 工作于方式 1
MOV TH0,#3CH
MOV TL0,#0B0H ;即数 15536
SETB EA ;开总中断允许
SETB ET0 ;开定时/计数器 0 允许
SETB TR0 ;定时/计数器 0 开始运行
LOOP: AJMP LOOP ;真正工作时,这里可写任意程序
TIME0: ;定时器 0 的中断处理程序
```

```
PUSH ACC
PUSH PSW ;将 PSW 和 ACC 推入堆栈保护
INC 30H
MOV A,30H
CJNE A,#20,T_RET ;30H 单元中的值到了 20 了吗?
T_L1: CPL P1.0 ;到了,取反 P10
MOV 30H,#0 ;清软件计数器
T_RET:
MOV TH0,#15H
MOV TL0,#9FH ;重置定时常数
POP PSW
POP ACC
RETI
END
```

先自己分析一下，看看是怎么实现的？这里采用了软件计数器的概念，思路是这样的，先用定时/计数器 0 做一个 50 毫秒的定时器，定时是间到了以后并不是立即取反 P10，而是将软件计数器中的值加 1，如果软件计数器计到了 20，就取反 P10，并清掉软件计数器中的值，否则直接返回，这样，就变成了 20 次定时中断才取反一次 P10，因此定时时间就延长了成了 20*50 即 1000 毫秒了。

这个思路在工程中是非常有用的，有的时候我们需要若干个定时器，可 51 中总共才有 2 个，怎么办呢？其实，只要这几个定时的时间有一定的公约数，我们就可以用软件定时器加以实现，如我要实现 P10 口所接灯按 1S 每次，而 P11 口所接灯按 2S 每次闪烁，怎么实现呢？对了我们用两个计数器，一个在它计到 20 时，取反 P10，并清零，就如上面所示，另一个计到 40 取反 P11，然后清 0，不就行了吗？这部份的程序如下

```
ORG 0000H
AJMP START
ORG 000BH ;定时器 0 的中断向量地址
AJMP TIME0 ;跳转到真正的定时器程序处
ORG 30H
START:
MOV P1,#0FFH ;关所 灯
MOV 30H,#00H ;软件计数器预清 0
MOV TMOD,#00000001B ;定时/计数器 0 工作于方式 1
MOV TH0,#3CH
MOV TL0,#0B0H ;即数 15536
SETB EA ;开总中断允许
SETB ET0 ;开定时/计数器 0 允许
SETB TR0 ;定时/计数器 0 开始运行
LOOP: AJMP LOOP ;真正工作时,这里可写任意程序
TIME0: ;定时器 0 的中断处理程序
PUSH ACC
PUSH PSW ;将 PSW 和 ACC 推入堆栈保护
INC 30H
INC 31H ;两个计数器都加 1
```

```
MOV A,30H
CJNE A,#20,T_NEXT ;30H 单元中的值到了 20 了吗?
T_L1: CPL P1.0 ;到了,取反 P10
MOV 30H,#0 ;清软件计数器
T_NEXT:
MOV A,31H
CJNE A,#40,T_RET ;31h 单元中的值到 40 了吗?
T_L2:
CPL P1.1
MOV 31H,#0 ;到了,取反 P11,清计数器,返回
T_RET:
MOV TH0,#15H
MOV TL0,#9FH ;重置定时常数
POP PSW
POP ACC
RETI
END
```

程序一下载

```
ORG     0000H
AJMP    START
ORG     30H
START:
    MOV     P1,#0FFH    ;关所 灯
    MOV     TMOD,#00000001B ;定时/计数器 0 工作于方式 1
    MOV     TH0,#15H
    MOV     TL0,#0A0H   ;即数 5536
    SETB    TR0         ;定时/计数器 0 开始运行
LOOP:JBC    TF0,NEXT ;如果 TF0 等于 1，则清 TF0 并转 NEXT 处
    AJMP    LOOP            ;否则跳转到 LOOP 处运行
NEXT:CPL    P1.0
    MOV     TH0,#15H
    MOV     TL0,#9FH;重置定时/计数器的初值
    AJMP    LOOP
    END
```

程序二下载

```
ORG     0000H
AJMP    START
ORG     000BH   ;定时器 0 的中断向量地址
AJMP    TIME0   ;跳转到真正的定时器程序处
ORG     30H
START:
    MOV     P1,#0FFH    ;关所 灯
    MOV     TMOD,#00000001B ;定时/计数器 0 工作于方式 1
    MOV     TH0,#15H
```

```
    MOV     TL0,#0A0H   ;即数 5536
    SETB    EA  ;开总中断允许
    SETB    ET0 ;开定时/计数器 0 允许
SETB    TR0         ;定时/计数器 0 开始运行
LOOP:   AJMP    LOOP    ;真正工作时,这里可写任意程序
TIME0:              ;定时器 0 的中断处理程序
    PUSH    ACC
PUSH    PSW     ;将 PSW 和 ACC 推入堆栈保护
    CPL P1.0
    MOV     TH0,#15H
    MOV     TL0,#0A0H   ;重置定时常数
    POP PSW
    POP ACC
    RETI
END
```

程序三下载

```
ORG     0000H
AJMP    START
ORG     000BH   ;定时器 0 的中断向量地址
AJMP    TIME0   ;跳转到真正的定时器程序处
ORG     30H
START:
    MOV     P1,#0FFH    ;关所 灯
    MOV     30H,#00H    ;软件计数器预清 0
    MOV     TMOD,#00000001B ;定时/计数器 0 工作于方式 1
    MOV     TH0,#3CH
    MOV     TL0,#0B0H   ;即数 15536
    SETB    EA  ;开总中断允许
    SETB    ET0 ;开定时/计数器 0 允许
SETB    TR0         ;定时/计数器 0 开始运行
LOOP:   AJMP    LOOP    ;真正工作时,这里可写任意程序
TIME0:              ;定时器 0 的中断处理程序
    PUSH    ACC
PUSH    PSW     ;将 PSW 和 ACC 推入堆栈保护
    INC 30H
    MOV     A,30H
    CJNE    A,#20,T_RET ;30H 单元中的值到了 20 了吗?
T_L1:   CPL P1.0    ;到了,取反 P10
    MOV     30H,#0              ;清软件计数器
T_RET:
    MOV     TH0,#15H
    MOV     TL0,#9FH    ;重置定时常数
    POP PSW
    POP ACC
```

```
    RETI
END
```

程序四下载

```
ORG     0000H
AJMP    START
ORG     000BH   ;定时器 0 的中断向量地址
AJMP    TIME0   ;跳转到真正的定时器程序处
ORG     30H
START:
    MOV     P1,#0FFH    ;关所 灯
    MOV     30H,#00H    ;软件计数器预清 0
    MOV     TMOD,#00000001B ;定时/计数器 0 工作于方式 1
    MOV     TH0,#3CH
    MOV     TL0,#0B0H   ;即数 15536
    SETB    EA  ;开总中断允许
    SETB    ET0 ;开定时/计数器 0 允许
SETB    TR0             ;定时/计数器 0 开始运行
LOOP:   AJMP    LOOP    ;真正工作时,这里可写任意程序
TIME0:                  ;定时器 0 的中断处理程序
    PUSH    ACC
PUSH    PSW     ;将 PSW 和 ACC 推入堆栈保护
    INC 30H
    INC 31H ;两个计数器都加 1
    MOV     A,30H
    CJNE    A,#20,T_NEXT    ;30H 单元中的值到了 20 了吗?
T_L1:   CPL P1.0        ;到了,取反 P10
    MOV     30H,#0                  ;清软件计数器
T_NEXT:
    MOV     A,31H
    CJNE    A,#40,T_RET ;31h 单元中的值到 40 了吗?
T_L2:
    CPL P1.1
    MOV     31H,#0          ;到了,取反 P11,清计数器,返回
T_RET:
MOV     TH0,#15H
    MOV     TL0,#9FH    ;重置定时常数
    POP PSW
    POP ACC
    RETI
END
```

您能用定时器的方法实现前面讲的流水灯吗？试试看。

单片机教程第二十课：定时/计数器实验 2

前面我们做了定时器的实验，现在来看一看计数实验，在工作中计数通常会有两种要求：第一、将计数的值显示出来，第二、计数值到一定程度即中断报警。第一种如各种计数器、里程表，第二种如前面例中讲到的生产线上的计数。先看第一种吧。我们的硬件中是这样连线的：324 构成的振荡器连到定时/计数器 1 的外部引脚 T1 上面，我们就利用这个来做一个计数实验，要将计数的值显示出来，当然最好用数码管了，可我们还没讲到这一部份，为了避免把问题复杂化，我们用 P1 口的 8 个 LED 来显示计到的数据。

程序如下：

```
ORG 0000H
AJMP START
ORG 30H
START:
MOV SP,#5FH
MOV TMOD,#01000000B ;定时/计数器 1 作计数用,0 不用全置 0
SETB TR1 ;启动计数器 1 开始运行.
LOOP: MOV A,TL0
MOV P1,A
AJMP LOOP
END
```

在硬件上用线将 324 的输出与 T1 连通(印板上有焊盘)运行这种程序，注意将板按正确的位置放置（LM324 放在左手边，LED 排列是按从高位到低们排列）看到什么？随着 324 后接的 LED 的闪烁，单片机的 8 只 LED 也在不断变化，注意观察，是不是按二进制：

```
00000000
00000001
00000010
00000011
```

这样的顺序在变呢？这就对了，这就是 TL0 中的数据。

程序二：

```
ORG 0000H
AJMP START
ORG 001BH
AJMP TIMER1 ;定时器 1 的中断处理
ORG 30H
START: MOV SP,#5FH
MOV TMOD,#01010000B ;定时/计数器 1 作计数用,模式 1,0 不用全置 0
MOV TH1,#0FFH
MOV TL1,#0FAH ;预置值,要求每计到 6 个脉冲即为一个事件
SETB EA
SETB ET1 ;开总中断和定时器 1 中断允许
SETB TR1 ;启动计数器 1 开始运行.
AJMP $
```

```
TIMER1:
PUSH ACC
PUSH PSW
CPL P1.0 ;计数值到,即取反 P1.0
MOV TH1,#0FFH
MOV TL1,#0FAH ;重置计数初值
POP PSW
POP ACC
RETI
END
```

上面这个程序完成的工作很简单，就是在每 6 个脉冲到来后取反一次 P1。0，因此实验的结果应当是：LM324 后接的 LED 亮、灭 6 次，则 P1。0 口所接 LED 亮或灭一次。这实际就是我们上面讲的计数器的第二种应用。

程序三：外部中断实验

```
ORG 0000H
AJMP START
ORG 0003H ;外部中断地直入口
AJMP INT0
ORG 30H
START: MOV SP,#5FH
MOV P1,#0FFH ;灯全灭
MOV P3,#0FFH ;P3 口置高电平
SETB EA
SETB EX0
AJMP $
INT0:
PUSH ACC
PUSH PSW
CPL P1.0
POP PSW
POP ACC
RETI
END
```

本程序的功能很简单，按一次按键 1（接在 12 引脚上的）就引发一次中断 0，取反一次 P1。0，因此理论上按一下灯亮，按一下灯灭，但在实际做实验时，可能会发觉有时不“灵”，按了它没反应，但在大部份时候是对的，这是怎么回事呢？我们在讲解键盘时再作解释，这个程序本身是没有问题的。

```
ORG     0000H
AJMP    START
ORG     30H
START:
    MOV     SP,#5FH
        MOV     TMOD,#00000110B ;定时/计数器 1 作计数用,0 不用全置 0
    SETB    TR0     ;启动计数器 1 开始运行.
```

```
LOOP:
        MOV     A,TL0
        CPL A
    MOV     P1,A
    AJMP    LOOP
END

ORG     0000H
AJMP    START
ORG     000BH
AJMP    TIMER0 ;定时器 0 的中断处理
ORG     30H
START:
    MOV     SP,#5FH
    MOV     TMOD,#00000101B     ;定时/计数器 1 作计数用,模式 1,0 不用全置 0
    MOV     TH0,#0FFH
    MOV     TL0,#0FAH           ;预置值,要求每计到 6 个脉冲即为一个事件
    SETB    EA
SETB    ET0       ;开总中断和定时器 1 中断允许
    SETB    TR0       ;启动计数器 1 开始运行.
    AJMP    $
TIMER0:
    PUSH    ACC
    PUSH    PSW
    CPL     P1.0      ;计数值到,即取反 P1.0
    MOV     TH0,#0FFH
    MOV     TL0,#0FAH   ;重置计数初值
        POP     PSW
        POP     ACC
    RETI
END

ORG     0000H
AJMP    START
ORG     0003H    ;外部中断地直入口
AJMP    INT_0
ORG     30H
START:
    MOV     SP,#5FH
    MOV     P1,#0FFH      ;灯全灭
    MOV     P3,#0FFH      ;P3 口置高电平
    SETB    IT0
    SETB    EA
    SETB    EX0
```

```
    AJMP    $
INT_0:
    PUSH    ACC
    PUSH    PSW
    CPL P1.0
    POP PSW
    POP ACC
    RETI
END
```

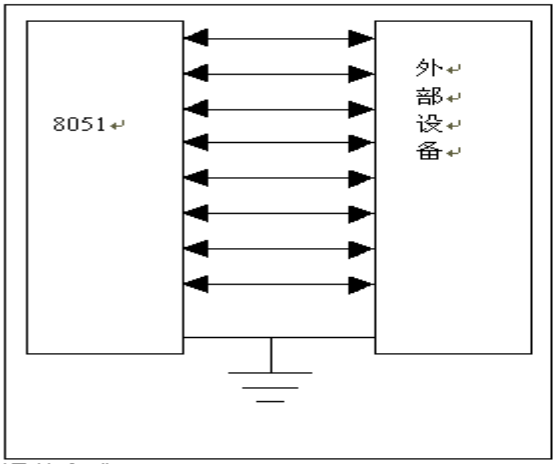
单片机教程第二十一课：串行接口

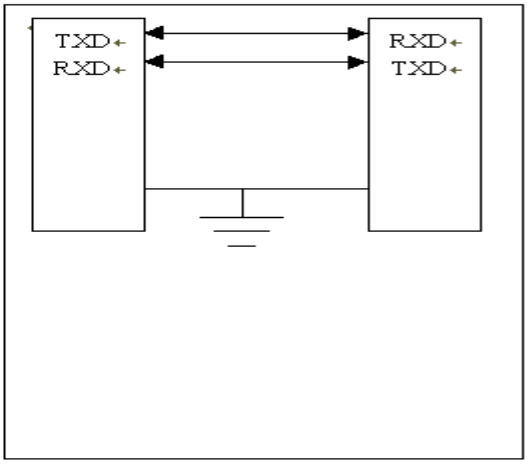
概述

串行接口的一般概念 单片机与外界进行信息交换称之为通讯。

8051 **单片机的通讯方式有两种：**

并行通讯:数据的各位同时发送或接收。 串行通讯:数据一位一位顺序发送或接收。参看下图：





串行通讯的方式：

异步通讯：它用一个起始位表示字符的开始，用停止位表示字符的结束。其每帧的格式如下：在一帧格式中，先是一个起始位 0，**然后是** 8 **个数据位，规定低位在前，高位在后，接下来是奇偶校验位（可以省略），最后是停止位** 1。**用这种格式表示字符，则字符可以一个接一个地传送。**

在异步通讯中，CPU **与外设之间必须有两项规定，即字符格式和波特率。字符格式的规定是双方能够在对同一种** 0 **和** 1 **的串理解成同一种意义。原则上字符格式可以由通讯的双方自由制定，但从通用、方便的角度出发，一般还是使用一些标准为好，如采用** ASCII **标准。波特率即数据传送的速率，其定义是每秒钟传送的二进制数的位数。例如，数据传送的速率是** 120 **字符/s，而每个字符如上述规定包含** 10 **数位，则传送波特率为** 1200 **波特。**

同步通讯：在同步通讯中，每个字符要用起始位和停止位作为字符开始和结束的标志，占用了时间；所以在数据块传递时，为了提高速度，常去掉这些标志，采用同步传送。由于数据块传递开始要用同步字符来指示，同时要求由时钟来实现发送端与接收端之间的同步，故硬件较复杂。

通讯方向：在串行通讯中，把通讯接口只能发送或接收的单向传送方法叫单工传送；而把数据在甲乙两机之间的双向传递，称之为双工传送。在双工传送方式中又分为半双工传送和全双工传送。半双工传送是两机之间不能同时进行发送和接收，任一时该，只能发或者只能收信息。

2．8051 **单片机的串行接口结构**

8051 **串行接口是一个可编程的全双工串行通讯接口。它可用作异步通讯方式（**UART**），与串行传送信息的外部设备相连接，或用于通过标准异步通讯协议进行全双工的** 8051 **多机系统也可以通过同步方式，使用** TTL **或** CMOS **移位寄存器来扩充** I/O **口。**

8051 单片机通过引脚 RXD（P3.0，串行数据接收端）和引脚 TXD（P3.1，串行数据发送端）与外界通讯。SBUF 是串行口缓冲寄存器，包括发送寄存器和接收寄存器。它们有相同名字和地址空间，但不会出现冲突，因为它们两个一个只能被 CPU 读出数据，一个只能被 CPU 写入数据。

串行口的控制与状态寄存器

串行口控制寄存器 SCON

它用于定义串行口的工作方式及实施接收和发送控制。字节地址为 98H，其各位定义如下表：

D7	D6	D5	D4	D3	D2	D1	D0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0、SM1：串行口工作方式选择位，其定义如下：

SM0、SM1	工作方式	功能描述	波特率
0 0	方式 0	8 位移位寄存器	Fosc/12
0 1	方式 1	10 位 UART	可变
1 0	方式 2	11 位 UART	Fosc/64 或 fosc/32
1 1	方式 3	11 位 UART	可变

其中 fosc 为晶振频率

SM2：多机通讯控制位。在方式 0 时，SM2 一定要等于 0。在方式 1 中，当（SM2）=1 则只有接收到有效停止位时，RI 才置 1。在方式 2 或方式 3 当（SM2）=1 且接收到的第九位数据 RB8=0 时，RI 才置 1。

REN：接收允许控制位。由软件置位以允许接收，又由软件清 0 来禁止接收。

TB8: 是要发送数据的第 9 位。在方式 2 或方式 3 中，要发送的第 9 位数据，根据需要由软件置 1 或清 0。例如，可约定作为奇偶校验位，或在多机通讯中作为区别地址帧或数据帧的标志位。

RB8：接收到的数据的第 9 位。在方式 0 中不使用 RB8。在方式 1 中，若（SM2）=0，RB8 为接收到的停止位。在方式 2 或方式 3 中，RB8 为接收到的第 9 位数据。

TI： 发送中断标志。在方式 0 中，第 8 位发送结束时，由硬件置位。在其它方式的发送停止位前，由硬件置位。TI 置位既表示一帧信息发送结束，同时也是申请中断，可根据需要，用软件查询的方法获得数据已发送完毕的信息，或用中断的方式来发送下一个数据。TI 必须用软件清 0。

RI： 接收中断标志位。在方式 0，当接收完第 8 位数据后，由硬件置位。在其它方式中，在接收到停止位的中间时刻由硬件置位（例外情况见于 SM2 的说明）。RI 置位表示一帧数据接收完毕，可用查询的方法获知或者用中断的方法获知。RI 也必须用软件清 0。

特殊功能寄存器 PCON

PCON 是为了在 CHMOS 的 80C51 单片机上实现电源控制而附加的。其中最高位是 SMOD。

串行口的工作方式

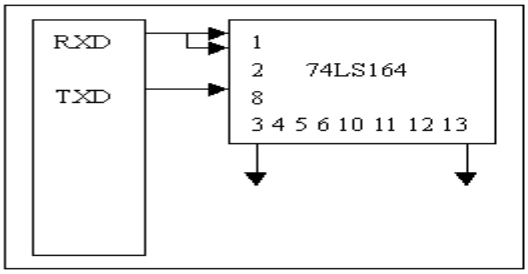
8051 单片机的全双工串行口可编程为 4 种工作方式，现分述如下：

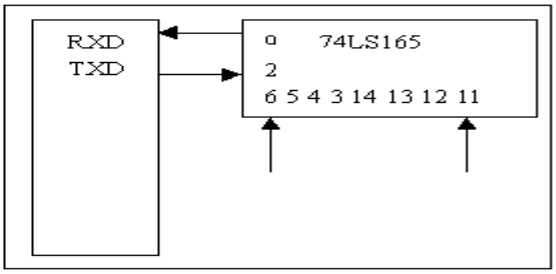
方式 0 为移位寄存器输入/输出方式。可外接移位寄存器以扩展 I/O 口，也可以外接同步输入/输出设备。8 位串行数据者是从 RXD 输入或输出，TXD 用来输出同步脉冲。

输出 串行数据从 RXD 引脚输出，TXD 引脚输出移位脉冲。CPU 将数据写入发送寄存器时，立即启动发送，将 8 位数据以 fos/12 的固定波特率从 RXD 输出，低位在前，高位在后。发送完一帧数据后，发送中断标志 TI 由硬件置位。

输入 当串行口以方式 0 接收时，先置位允许接收控制位 REN。此时，RXD 为串行数据输入端，TXD 仍为同步脉冲移位输出端。当（RI）=0 和（REN）=1 同时满足时，开始接收。

当接收到第 8 位数据时，将数据移入接收寄存器，并由硬件置位 RI。
下面两图分别是方式 0 扩展输出和输入的接线图。





方式 1 为波特率可变的 10 位异步通讯接口方式。发送或接收一帧信息，包括 1 个起始位 0，8 个数据位和 1 个停止位 1。
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 引脚输出，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 引脚，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；否则信息丢失。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。它可作为多机通讯中地址/数据信息的标志位，也可以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 引脚，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；否则信息丢失。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，在串行通讯中，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。

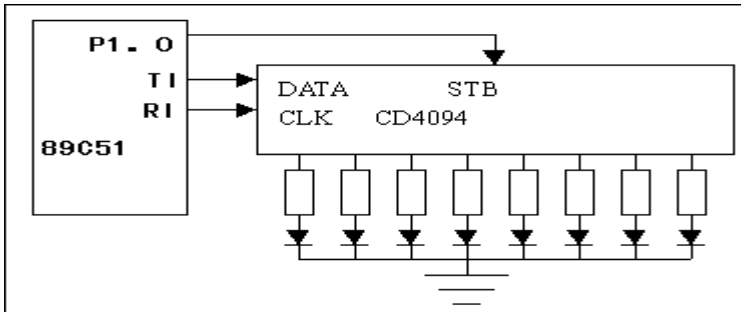
当时钟频率选用 11.0592MHZ 时，取易获得标准的波特率，所以很多单片机系统选用这个看起来“怪”的晶振就是这个道理。

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

单片机教程第二十二课：串行口应用编程实例

1． **串口方式** 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，就可以扩展一个并行口。**



例：用 8051 **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，可以靠** 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，在开始通讯之前，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 **就可以了。**

```
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
```

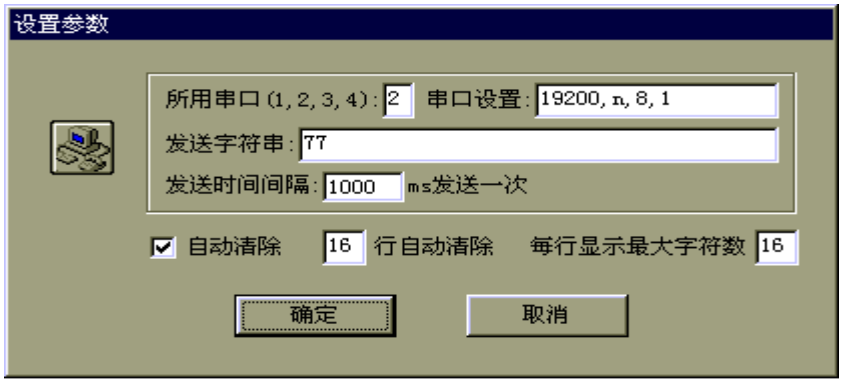
说明：DELAY **延时子程序可以用前面我们讲** P1 **口流水灯时用的延时子程序，这里就不给出了。**

二、异步通讯

```
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口相连就可以实验了。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。可以在 PC 端用串口精灵来做实验。串口精灵在我主页上有下载。运行串口精灵后，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口 2，如果你不是串口 2，请自行更改串口的设置。



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”就可以了。按此设置，实验板上应当有两只灯亮，6 只灯灭。大家可以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，也可以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
```

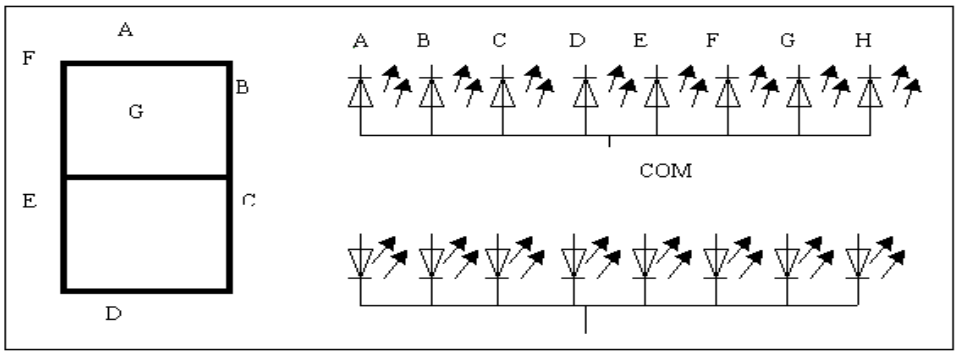
本程序没有写入发送程序，大家可以自行添加。

单片机教程第二十三课：LED 数码显示器的连接与编程

在单片机系统中，通常用 LED 数码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。

八段 LED 显示器

引入：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，可以拼成各种各样的图形，LED 显示器实际上也是这么一个东西。



八段 LED 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 显示器有两种不同的形式：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 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 显示器。如下图所示。`

共阴和共阳结构的 LED 显示器各笔划段名和安排位置是相同的。当二极管导通时，相应的笔划段发亮，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 位二进制码就可以表示欲显示字符的字形代码。例如，对于共阴 LED 显示器，当公共阴极接地（为零电平），而阳极 hgfedcba 各段为 0111011 时，显示器显示"P"字符，即对于共阴极 LED 显示器，“P”字符的字形码是 73H。如果是共阳 LED 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，常不按规则的方法去对应字段与位的关系，这时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子。

静态显示接口

在单片机应用系统中，显示器显示常用两种方法：静态显示和动态扫描显示。所谓静态显示，就是每一个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，使用这种方法单片机中 CPU 的开销小。可以提供单独锁存的 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 0 为移位寄存器方式，外接 6 片 74LS164 作为 6 位 LED 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 个引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 引脚）并行输出端分别接 LED 显示器的 hg---a 各段对应的引脚上。关于 74LS164 还可以作如下的介绍：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 引脚）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，要从后面进去一个人，前面必须要从前面走出去一个人才行。

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《初学单片机易掌握的概念》。务必搞懂，搞懂了这一点，你的级别就高过初学者，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 个数成相应的显示字形码，然后输出到显示器中显示。

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
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，也就是相应的输出位为 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）写出相应位的 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顺序，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，由此往往陷入困惑的境界。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。有兴趣的朋友还可以再看看我网站上“其他资料”中的“银行利率屏的设计”一文。

单片机教程第二十四课：动态扫描显示接口

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，所以我们就可以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方法，轮流控制各个显示器的 COM 端，使各个显示器轮流点亮。

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，实际上还可以扩充。它们的公共端则由 PNP 型三极管 8550 控制，显然，如果 8550 导通，则相应的数码管就可以亮，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。这样我们就可以通过控制 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
```

从上面的例子中可以看出，动态扫描显示必须由 CPU 不断地调用显示程序，才能保证持续不断的显示。

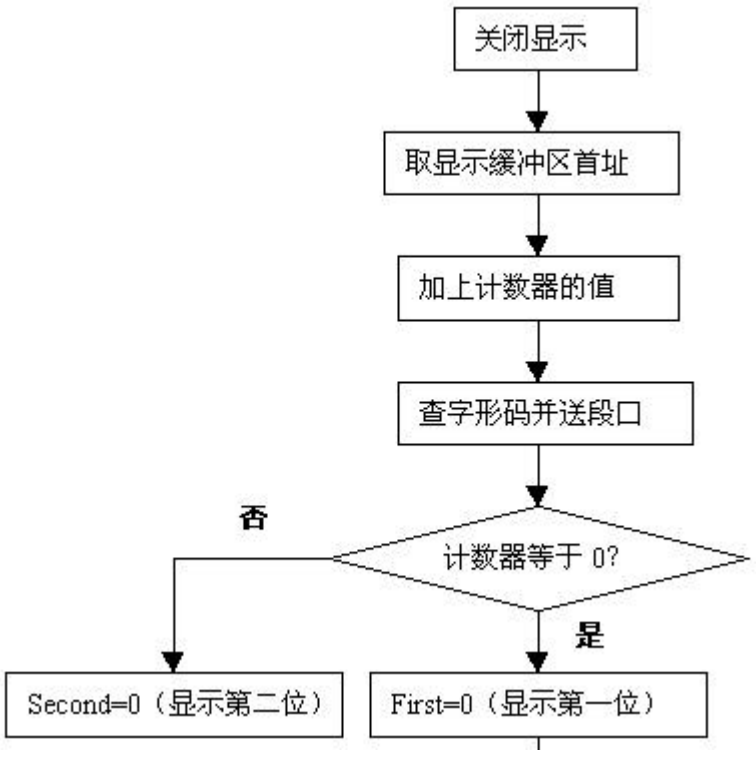
上面的这个程序可以实现数字的显示，但不太实用，为什么呢？这里仅是显示两个数字，并没有做其他的工作，因此，两个数码管轮流显示 1 毫秒，没有问题，实际的工作中，当然不可能只显示两个数字，还是要做其他的事情的，这样在二次调用显示程序之间的时间间隔就不一不定了，如果时间间隔比较长，就会使显示不连续。而实际工作中是很难保证所有工作都能在很短时间内完成的。况且这个显示程序也有点“浪费”，每个数码管显示都要占用 1 个毫秒的时间，这在很多合是不允许的，怎么办呢？我们可以借助于定时器，定时时间一到，产生中断，点亮一个数码管，然后马上返回，这个数码管就会一直亮到下一次定时时间到，而不用调用延时程序了，这段时间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

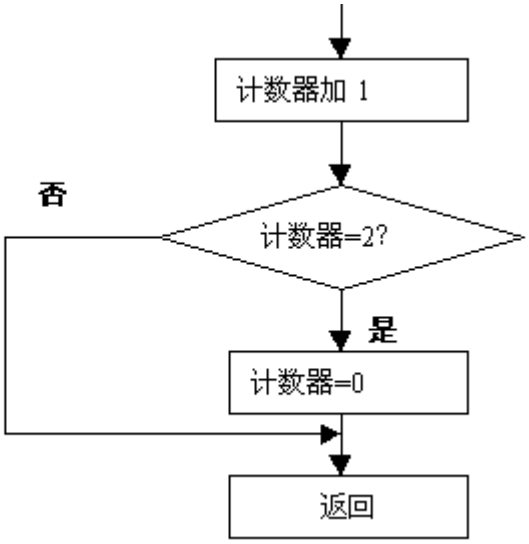
```
ORG 30H
START:
MOV SP,#5FH ;设置堆栈
MOV P1,#0FFH
MOV P0,#0FFH
MOV P2,#0FFH ;初始化，所显示器，LED 灭
MOV TMOD,#00000001B ;定时器 T0 工作于模式 1（16 位定时/计数模式）
MOV TH0,#HIGH(65536-2000)
MOV TL0,#LOW(65536-2000)
SETB TR0
SETB EA
SETB ET0
MOV Counter,#0 ;计数器初始化
MOV DISPBUFF,#0 ;第一位始终显示 0
MOV A,#0
LOOP:
MOV DISPBUFF+1,A ;第二位轮流显示 0-9
INC A
LCALL DELAY
CJNE A,#10,LOOP
MOV A,#0
AJMP LOOP ;在此中间可以按排任意程序，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 ;根据计数器的值取相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 ;否则显示第二位
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
```

从上面的程序可以看出，和静态显示相比，动态扫描的程序稍有点复杂，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可以显示更多位数了。下面给



出显示程序的流程图。



程序一

```
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
```

程序二

```
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
;主程序到此结束
DISP:
    PUSH    ACC             ;ACC 入栈
    PUSH    PSW             ;PSW 入栈
    MOV     TH0,#HIGH(65536-2000)
    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           ;根据计数器的值取相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
    CLRSECOND           ;否则显示第二位
    AJMP    DISPSECOND
DISPFIRST:
    CLRFIRST            ;显示第一位
```

```
DISPSECOND:
    INC Counter         ;计数器加 1
    MOV     A,Counter
    DEC     A           ;如果计数器计到 2 望，则让它回 0
    DEC     A
    JZ   RSTCOUNT
    AJMP    DISPNEXT
RSTCOUNT:
    MOV     Counter,#0      ;计数器的值只能是 0 或 1
DISPNEXT:

    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
```