51 单片机汇编语言教程：15 课:单片机位操作指令

前面那些流水灯的例程，我们已经习惯了“位”一位就是一盏灯的亮和灭，而我们学的指令却全都是用“字节”来介绍的：字节的移动、加法、减法、逻辑运算、移位等等。用字节来处理一些数学问题，比如说：控制冰箱的温度、电视的音量等等很直观，能直接用数值来表在。可是如果用它来控制一些开关的打开和合上，灯的亮和灭，就有些不直接了，记得我们上次课上的流水灯的例程吗？我们知道送往 P1 口的数值后并不能马上知道哪个灯亮和来灭，而是要化成二进制才知道。工业中有很多场合需要处理这类开关输出，继电器吸合，用字节来处理就显示有些麻烦，所以在 8031 单片机中特意引入一个位处理机制。

位寻址区

在 8031 中，有一部份 RAM 和一部份 SFR 是具有位寻址功能的，也就是说这些 RAM 的每一个位都有自己的地址，能直接用这个地址来对此进行操作。

内部 RAM 的 20H-2FH 这 16 个字节，就是 8031 的位寻址区。看图 1。可见这里面的每一个 RAM 中的每个位我们都可能直接用位地址来找到它们，而不必用字节地址，然后再用逻辑指令的方式。

能位寻址的特殊功能寄存器

8031 中有一些 SFR 是能进行位寻址的，这些 SFR 的特点是其字节地址均可被 8 整除，如 A 累加器，B 寄存器、PSW、IP（中断优先级控制寄存器）、IE（中断允许控制寄存器）、SCON（串行口控制寄存器）、TCON（定时器/计数器控制寄存器）、P0-P3（I/O 端口锁存器）。以上的一些 SFR 我们还不熟，等我们讲解相关内容时再作详细解释。

位操作指令

MCS-51 单片机的硬件结构中，有一个位处理器（又称布尔处理器），它有一套位变量处理的指令集。在进行位处理时，CY（就是我们前面讲的进位位）称“位累加器”。有自己的位 RAM，也就是我们刚讲的内部 RAM 的 20H-2FH 这 16 个字节单元即 128 个位单元，还有自己的位 I/O 空间（即
P0.0…..P0.7,P1.0…….P1.7,P2.0……..P2.7,P3.0……..P3.7）。当然在物理实体上它们与原来的以字节寻址用的 RAM,及端口是完全相同的,或者说这些 RAM 及端口都能有两种使用办法。

位传送指令

MOV C，BIT

MOV BIT，C

这组指令的功能是实现位累加器（CY）和其它位地址之间的数据传递。

例：MOV P1.0,CY ;将 CY 中的状态送到 P1.0 管脚上去（如果是做算术运算，我们就能通过观察知道现在 CY 是多少啦）。

MOV P1.0,CY ;将 P1.0 的状态送给 CY。

位修正指令

位清 0 指令

CLR C ;使 CY=0

CLR bit ;使指令的位地址等于 0。例：CLR P1.0 ;即使 P1.0 变为 0

位置 1 指令

SETB C ;使 CY=1

SETB bit ;使指定的位地址等于 1。例：SETB P1.0 ;使 P.0 变为 1

位取反指令

CPL C ;使 CY 等于原来的相反的值，由 1 变为 0，由 0 变为 1。

CPL bit ;使指定的位的值等于原来相反的值，由 0 变为 1，由 1 变为 0。

例：CPL P1.0

以我们做过的实验为例，如果原来灯是亮的，则执行本指令后灯灭，反之原来灯是灭的，执行本指令后灯亮。

位逻辑运算指令

位与指令

ANL C,bit ;CY 与指定的位地址的值相与，结果送回 CY

ANL C,/bit ;先将指定的位地址中的值取出后取反，再和 CY 相与，结果送回 CY，但注意，指定的位地址中的值本身并不发生变化。

例：ANL C,/P1.0

设执行本指令前，CY=1，P1.0 等于 1（灯灭），则执行完本指令后 CY=0，而 P1.0 也是等于 1。

可用下列程序验证：

```
ORG 0000H

AJMP START

ORG 30H

START： MOV SP，#5FH

MOV P1，#0FFH

SETB C

ANL C，/P1.0

MOV P1.1,C ;将做完的结果送 P1.1,结果应当是 P1.1 上的灯亮，而 P1.0 上的灯还是不亮
```

位或指令

```
ORL C,bit

ORL C,/bit
```

这个的功能大家自行分析吧，然后对照上面的例程，编一个验证程序，看看你相得对吗？

位条件转移指令

判 CY 转移指令

```
JC rel

JNC rel
```

第一条指令的功能是如果 CY 等于 1 就转移，如果不等于 1 就次序执行。那么转移到什么地方去呢？我们能这样理解：JC 标号，如果等于 1 就转到标号处执行。这条指令我们在上节课中已讲到，不再重复。

第二条指令则和第一条指令相反，即如果 CY=0 就转移，不等于 0 就次序执行，当然，我们也同样理解： JNC 标号

判位变量转移指令

```
JB bit,rel

JNB bit,rel
```

第一条指令是如果指定的bit位中的值是1，则转移，不然次序执行。同样，我们能这样理解这条指令：JB bit,标号

第二条指令请大家先自行分析

下面我们举个例程说明：

```
ORG OOOOH

LJMP START

ORG 3OH

START：MOV SP，#5FH

MOV P1，#OFFH

MOV P3，#OFFH

L1: JNB P3.2,L2 ;P3.2上接有一只按钮，它按下时，P3.2=0

JNB P3.3,L3 ;P3.3上接有一只按钮，它按下时，P3.3=0

LJM P L1

L2: MOV P1,#OOH

LJMP L1

L3: MOV P1,#OFFH

LJMP L1

END
```

把上面的例程写入片子，看看有什么现象……

按下接在P3.2上的按钮，P1口的灯全亮了，松开或再按，灯并不熄灭，然后按下接在P3.3上的按钮，灯就全灭了。这像什么？这不就是工业现场经常用到的“启动”、“停止”的功能吗？

怎么做到的呢？一开始，将OFFH送入P3口，这样，P3的所有引线都处于高电平，然后执行L1，如果P3.2是高电平（键没有按下），则次序执行JNB P3.3,L3语句，同样，如果P3.3是高电平（键没有按下），则次序执行LJMP L1语句。这样就不停地检测P3.2、P3.3，如果有一次P3.2上的按钮按下去了，则转移到L2，执行MOV P1，#OOH，使灯全亮，然后又转去L1，再次循环，直到检测到P3.3

为 0，则转 L3，执行 MOV P1，#0FFH，例灯全灭，再转去 L1，如此循环不已。大家能否稍加改动，将本程序用 JB 指令改写？

51 单片机汇编语言教程：16 课:单片机定时器与计数器

一、计数概念的引入

从选票的统计谈起：画“正”。这就是计数，生活中计数的例程处处可见。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程，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？行业中有很巧妙的办法，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，就能知道走过的线有多长了。

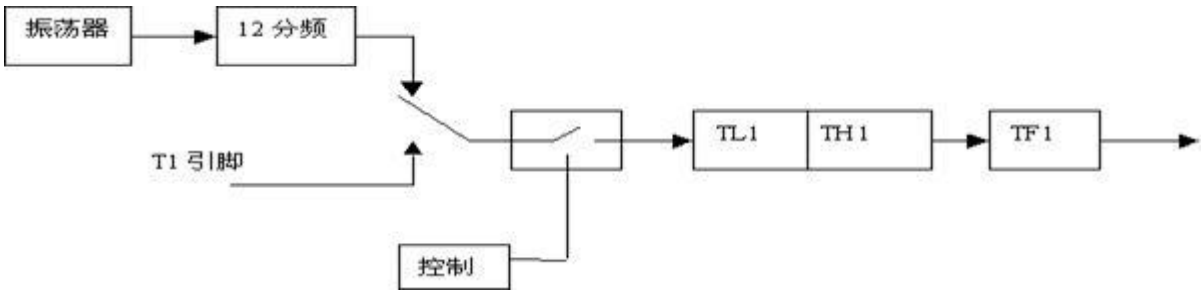
二、计数器的容量

从一个生活中的例程看起：一个水盆在水龙头下，水龙没关紧，水一滴滴地滴入盆中。水滴持续落下，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 中的计数器除了能作为计数之用外，还能用作时钟，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，也能说是秒针走了（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。



<单片机定时器记数器结构>

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供给一个非常稳定的计数源。那么供给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体震荡器经过 12 分频后获得的一个脉冲源。晶体震荡器的频率当然很准，所以这个计数脉冲的时间间隔也很准。问题：一个 12M 的晶体震荡器，它供给给计数器的脉冲时间间隔是多少呢？当然这很不难，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体震荡器有关，12M 的晶体震荡器，计数脉冲的间隔是 1 微秒。

四、溢出

让我们再来看水滴的例程，当水持续落下，盆中的水持续变满，最终有一滴水使得盆中的水满了。这个时候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、任意定时及计数的办法　　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，我们采用预置数的办法，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 毫秒就能了，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 就能了。

51 单片机汇编语言教程：17 课:单片机定时器/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/计数器都能有多种用途，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，我们在写程序时就能直接用这个名称来指定它们，当然也能直接用它们的地址 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）

<TMOD 结构>

从图 1 中我们能看出，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。

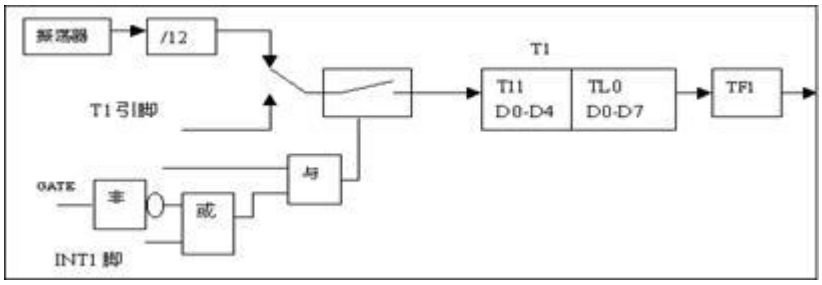
用于定时/计数器				用于中断			
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

图 2（TCON）

<TCON 结构>

从图 2 中我们能看出，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不难，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。



<单片机定时器/计数器结构>

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 位组成 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，但很多开始学习者却会有此困惑。

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 管脚的控制，只有 TR1 为 1，且 INT1 管脚也是高电平，开关才合上，计数脉冲才得以通过。这个特性能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，计数总是持续重复的，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，能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（这常常是需要的，要不然要溢出干吗？）能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，那么就能用这种方式。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。

常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，定时器就是为了供给一个时间基准。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 能组成 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，最多能计到 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，最多能计到 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，最多能计到 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，请参考前面的例程。

51 单片机汇编语言教程：18 课:单片机的中断系统

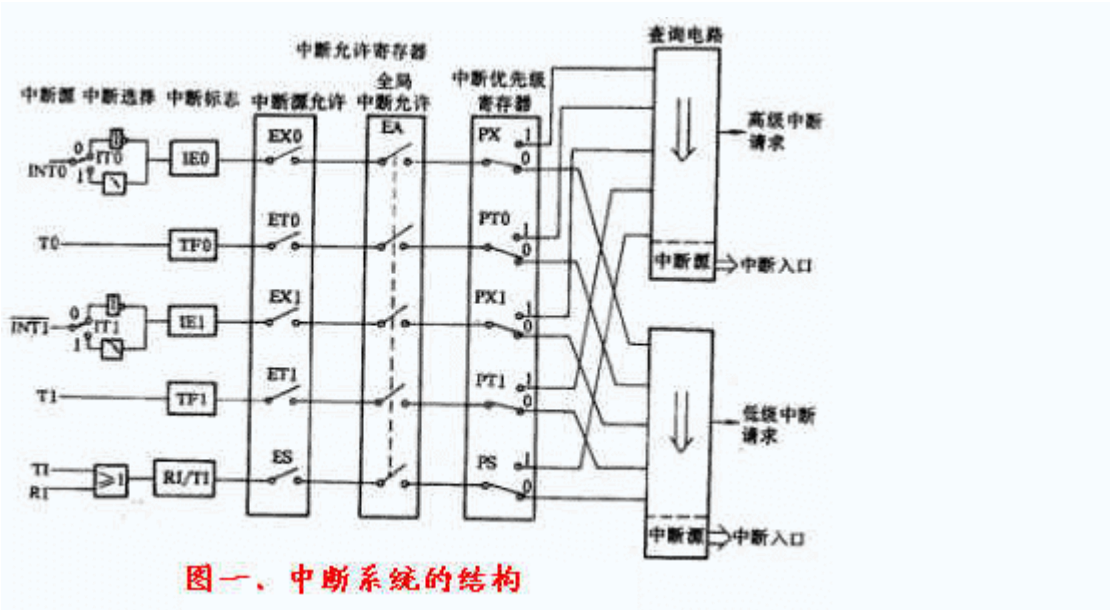
有关单片机中断系统的概念：什么是中断，我们从一个生活中的例程引入。你正在家中看书，突然电话铃响了，你放下书本，去接电话，和来电话的人交谈，然后放下电话，回来继续看你的书。这就是生活中的“中断”的现象，就是正常的工作过程被外部的事件打断了。仔细研究一下生活中的中断，对于我们学习单片机的中断也很有好处。

第一、什么可经引起中断，生活中很多事件能引起中断：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，我们把能引起中断的称之为中断源，单片机中也有一些能引起中断的事件，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。

第二、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），你可能会按你常常的习惯去处理。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。

第三、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，然后去处理不一样的事情（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，也说是不一样的中断，我们要在不一样的地点处理，而这个地点常常还是固定的。计算机中也是采用的这种办法，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，中断响应能分为以下几个步骤：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 个不一样的中断源所产生的中断，查找 5 个不一样的入口地址。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 单片机中断系统的结构：
5 个中断源的符号、名称及产生的条件如下。
INT0：外部中断 0，由 P3．2 端口线引入，低电平或下跳沿引起。
INT1：外部中断 1，由 P3．3 端口线引入，低电平或下跳沿引起。
T0：定时器／计数器 0 中断，由 T0 计满回零引起。
T1：定时器／计数器 1 中断，由 T1 计满回零引起。
TI／RI：串行 I／O 中断，串行端口完成一帧字符发送／接收后引起。
整个中断系统的结构框图见下图一所示。



图一、中断系统的结构

<51 单片机中断系统结构>

如图所示，由与中断有关的特殊功能寄存器、中断入口、次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，经由外部管脚引入的，在单片机上有两个管脚，名称为 INT0、INT1，也就是 P3.2、P3.3 这两个管脚。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。（2）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，在串行口中再讲解。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 能是 EAX

即 8CH，当然，我们也能用位操作指令 SETB EA

SETB ET1SETB EX1

来实现它。3、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 串行口：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 003OH

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，即能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，我们能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，对应的中断就是高优先级，不然就是低优先级。

XX

X

PS

PT1

PX1

PTO

PXO

例：设有如下要求，将 TO、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，后面根据要求写就能了 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，我们人能响应外界的事件，是因为我们有多种“传感器“－－眼、耳能接受不一样的信息，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：

CPU 正在处理一个同级或更高级别的中断请求。

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）。

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，再执行一条指令才能响应中断。

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，将对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，不就能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 OOOOHLJMP START

ORG OOO3H

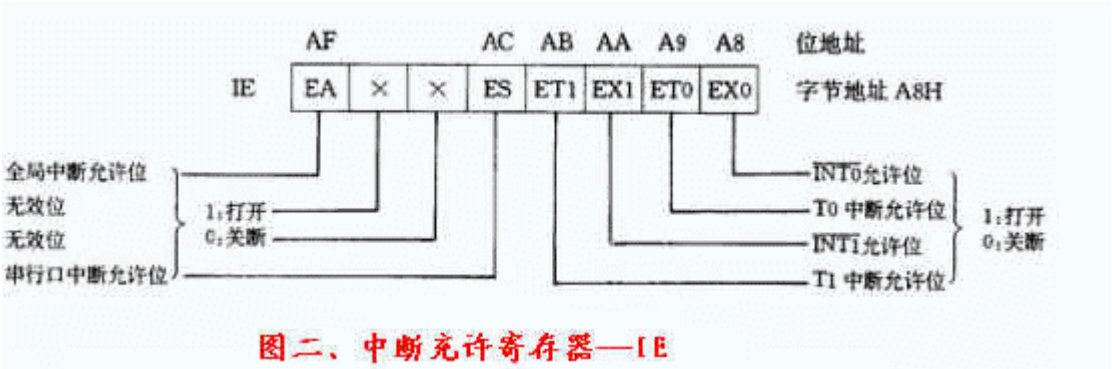
LJMP INTO ；转外中断 OORG OOOBH

RETI ；没有用定时器 O 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。

中断系统的控制寄存器：
中断系统有两个控制寄存器 IE 和 IP，它们分别用来设定各个中断源的打开 / 关闭和中断优先级。此外，在 TCON 中另有 4 位用于选择引起外部中断的条件并作为标志位。

1．中断允许寄存器--IE
IE 在特殊功能寄存器中，字节地址为 A8H，位地址(由低位到高位)分别是 A8H-AFH。
IE 用来打开或关断各中断源的中断请求，基本格式如下图二所示：



图二、中断允许寄存器—IE

EA：全局中断允许位。EA＝0，关闭全部中断；EA＝1，打开全局中断控制，在此条件下，由各个中断控制位确定相应中断的打开或关闭。

×：无效位。

ES：串行 I／O 中断允许位。ES＝1，打开串行 I／O 中断；ES＝0，关闭串行 I／O 中断。

ET1；定时器／计数器 1 中断允许位。ET1＝1，打开 T1 中断；ET1＝0，关闭 T1 中断。

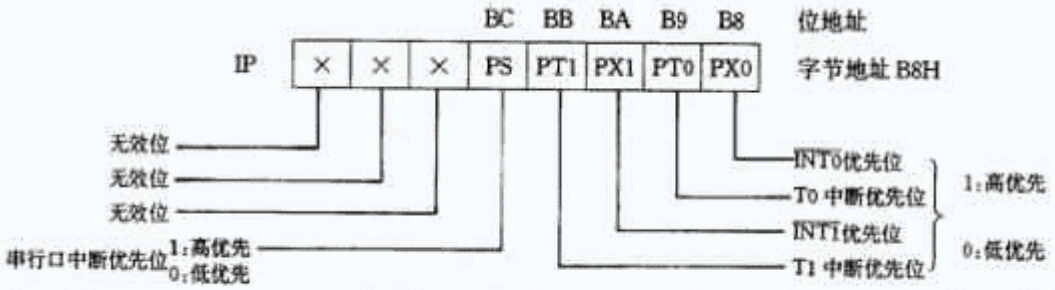
EX1：外部中断 1 中断允许位。EX1＝1，打开 INT1；EX1＝0，关闭 INT1。

ET0：定时器／计数器 0 中断允许位。ET0＝1，打开 T0 中断；ET0＝0，关闭 T0 中断。

EX0：外部中断 0 中断允许位。Ex0＝1，打开 INT0;EX0=0,关闭 INT0.

中断优先寄存器--IP：

IP 在特殊功能寄存器中，字节地址为 B8H，位地址(由低位到高位)分别是 B8H 一 BFH,IP 用来设定各个中断源属于两级中断中的哪一级，IP 的基本格式如下图三所示：



图三、中断优先寄存器——IP

×：无效位。

PS：串行 I／O 中断优先级控制位。PS＝1，高优先级；PS＝0，低优先级。

PT1：定时器／计数器 1 中断优先级控制位。PT1＝1，高优先级；PT1＝0，低优先级。

Px1：外部中断 1 中断优先级控制位。Px1＝1，高优先级；PX1＝0，低优先级。

PT0：定时器／计数器 o 中断优先级控制位。PT0＝1，高优先级；PT0＝0，低优先级。

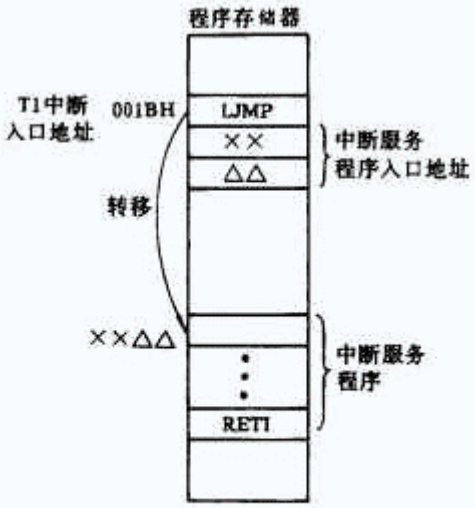
Px0：外部中断 0 中断优先级控制位。Px0＝1，高优先级；Px0＝0，伤优先级。

在 MCS-51 单片机系列中，高级中断能够打断低级中断以形成中断嵌套；同级中断之间，或低级对高级中断则不能形成中断嵌套。若几个同级中断同时向 CPU 请求中断响应，则 CPU 按如下顺序确定响应的先后顺序：

INT0 一 T0---INT1 一 T1 一 RI / T1.

中断的响应过程

若某个中断源通过编程设置，处于被打开的状态，并满足中断响应的条件，而且①当前正在执行的那条指令已被执行完
1、当前末响应同级或高级中断
2、不是在操作 IE，IP 中断控制寄存器或执行 REH 指令则单片机响应此中断。
在正常的情况下，从中断请求信号有效开始，到中断得到响应，通常需要 3 个机器周期到 8 个机器周期。中断得到响应后，自动清除中断请求标志(对串行 I / O 端口的中断标志，要用软件清除)，将断点即程序计数器之值(PC)压入堆栈(以备恢复用)；然后把相应的中断入口地址装入 PC，使程序转入到相应的中断服务程序中去执行。
各个中断源在程序存储器中的中断入口地址如下：
中断源 入口地址
INT0(外部中断 0) 0003H
TF0(T0 中断) 000BH
INT1(外部中断 1) 0013H
TF1(T1 中断) 001BH
RI / TI(串行口中断) 0023H
由于各个中断入口地址相隔甚近，不便于存放各个较长的中断服务程序，故通常在中断入口地址开始的二三个单元中，安排一条转移类指令，以转入到安排在那儿的中断服务程序。以 T1 中断为例，其过程下如图四所示。
由于 5 个中断源各有其中断请求标志 0，TF0，IE1，TF1 以及 RI / TI，在中断源满足中断请求的条件下，各标志自动置 1，以向 CPU 请求中断。如果某一中断源提出中断请求后，CPU 不能立即响应，只要该中断请求标志不被软件人为清除，中断请求的状态就将一直保持,直到 CPU 响应了中断为止,对串行口中断而言，这一过程与其它 4 个中断的不同之处在于；即使 CPU 响应了中断，其中断标志 RI / TI 也不会自动清零，必须在中断服务程序中设置清除 RI / TI 的指令后，才会再一次地提出中断请求。
CPU 的现场保护和恢复必须由被响应的相应中断服务程序去完成，当执行 RETI 中断返回指令后，断点值自动从栈顶 2 字节弹出，并装入 PC 寄存器，使 CPU 继续执行被打断了的程序。
下面给出一个应用定时器中断的实例。
现要求编制一段程序，使 P1．0 端口线上输出周期为 2ms 的方波脉冲。设单片机晶振频率 Fosc＝6MHZ.
1、方法：利用定时器 T0 作 1ms 定时，达到定时值后引起中断，在中断服务程序中，使 P1．0 的状态取一次反，并再次定时 1ms。
2、定时初值：机器周期 MC＝12/fosc＝2us。所以定时 1ms 所需的机器周期个数为 500D，亦即 01F4H。设 T0 为工作方式 1(16 位方式)，则定时初值是(01F4H)求补＝FE0CH



图四
由中断入口地址进入中断服务程序

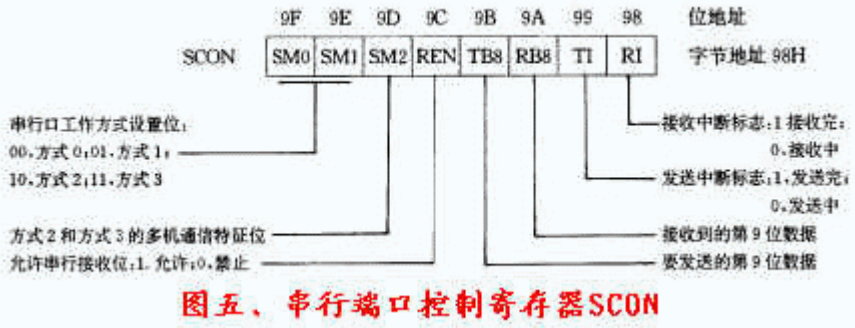
START：	MOV TMOD，#01H	；T0 为定时器状态，工作方式 1
	MOV TL0，#0CH	；T0 的低位定时初值
	MOV TH0，#0FEH	；T0 的高位定时初值
	MOV TCON，#10H	；打开 T0
	SETB ET0	；1ET0，即允许 T0 中断
	SETB EA	；1EA，即充许全局中断
	AJMP $	；动态暂存
000BH：	AJMP IST0	；转入 T0 中断服务程序入口地址 IST0
IST0：	MOV TL0，#0CH	；重置定时器初值
	MOV TH0，#0FEH	；重置定时器初值
	CPL P1.0	；P1.0 取反
	RET1	；中断返回

串行端口的控制寄存器：

串行端口共有 2 个控制寄存器 SCON 和 PCON，用以设置串行端口的工作方式、接收 / 发送的运行状态、接收 / 发送数据的特征、波特率的大小，以及作为运行的中断标志等。

①串行口控制寄存器 SCON

SCON 的字节地址是 98H，位地址(由低位到高位)分别是 98H 一 9FH。SCON 的格式如图五所示。

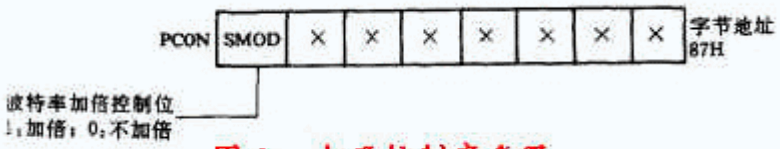


图五、串行端口控制寄存器SCON

SMo，SM1：
串行口工作方式控制位。
00--方式 0；01--方式 1；
10--方式 2；11--方式 3。
SM2：
仅用于方式 2 和方式 3 的多机通讯控制位
发送机 SM2＝1(要求程控设置)。
当为方式 2 或方式 3 时：
接收机 SM2＝1 时，若 RB8＝1，可引起串行接收中断；若 RB8＝0，不引起串行接收中断。SM2＝0 时，若 RB8＝1，可引起串行接收中断；若 RB8＝0，亦可引起串行接收中断。
REN:
串行接收允许位。
0--禁止接收；1--允许接收。
TB8:
在方式 2，3 中，TB8 是发送机要发送的第 9 位数据。
RB8:
在方式 2，3 中，RB8 是接收机接收到的第 9 位数据，该数据正好来自发送机的 TB8。
TI:
发送中断标志位。发送前必须用软件清零，发送过程中 TI 保持零电平，发送完一帧数据后，由硬件自动置 1。如要再发送，必须用软件再清零。
RI:
接收中断标志位。接收前，必须用软件清零，接收过程中 RI 保持零电平，接收完一帧数据后，由片内硬件自动置 1。如要再接收，必须用软件再清零。

电源控制寄存器 PCON

PCON 的字节地址为 87H，无位地址，PCON 的格式如图六所示。需指出的是，对 80C31 单片机而言，PCON 还有几位有效控制位。



图六、电源控制寄存器

SMOD：波特率加倍位。在计算串行方式 1，2，3 的波特率时；0---不加倍；1---加倍。

串行中断的应用特点：

8031 单片机的串行 I／O 端口是一个中断源，有两个中断标志 RI 和 TI，RI 用于接收，TI 用于发送。
串行端口无论在何种工作方式下，发送／接收前都必须对 TI／RI 清零。当一帧数据发送／接收完后，TI/RI 自动置 1，如要再发送／接收，必须先用软件将其清除。
在串行中断被打开的条件下，对方式 0 和方式 1 来说，一帧数据发送／接收完后，除置位 TI／RI 外，还会引起串行中断请求，并执行串行中侧目务程序。但对方式 2 和方式 3 的接收机而言，还要视 SM2 和 RB8 的状态，才可确定 RI 是否被置位以及串行中断的开放：
SM2 RB8 接收机中断标志与中断状态
0 1 激活 RI，引起中断
1 0 不激活 RI，不引起中断
1 1 激活 RI，引起中断
单片机正是利用方式 2，3 的这一特点，实现多机间的通信。串行端口的常用应用方法见相关章节。

波特率的确定：

对方式 0 来说，波特率已固定成 fosc／12，随着外部晶振的频率不同，波特率亦不相同。常用的 fosc 有 12MHz 和 6MHz，所以波特率相应为 1000×103 和 500×103 位／s。在此方式下，数据将自动地按固定的波特率发送／接收，完全不用设置。
对方式 2 而言，波特率的计算式为 2SMOD•fosc／64。当 SMOD＝0 时，波特率为 fm／64；当 SMOD＝1 时，波特率为 fosc／32。在此方式下，程控设置 SMOD 位的状态后，波特率就确定了，不需要再作其它设置。
对方式 1 和方式 3 来说，波特率的计算式为 2SMOD／32×T1 溢出率，根据 SMOD 状态位的不同，波特率有 Tl／32 溢出率和 T1／16 溢出率两种。由于 T1 溢出率的设置是方便的，因而波特率的选择将十分灵活。
前已叙及，定时器 Tl 有 4 种工作方式，为了得到其溢出率，而又不必进入中断服务程序，往往使 T1 设置在工作方式 2 的运行状态，也就是 8 位自动加入时间常数的方式。由于在这种方式下，T1 的溢出率(次／秒)计算式可表达成：

$$T1\ 溢出率=\frac{f_{OSC}}{12(256-x)}$$

式中 x 为设置的定时初值，于是波特率(位/秒)表达式为：

$$BR=\frac{2^{SMOD}}{32}\times\frac{f_{OSC}}{12(256-x)}$$

由上式可见，选取不同的 x 初值，就可得到不同的波特率。

把上式变换一下形式，就可根据所要求的波特率 BR，算出 T1 定时器初值的大小来，其计算式为：

$$x=256-\frac{2^{SMOD}\cdot f_{OSC}}{384\cdot BR}$$

例如，以产生 1200 位/秒为例，求定时器 T1 定时初值 x 的大小。

设 $f_{OSC}=6MHz$，SMOD=0，则

$$1200/秒=\frac{1}{32}\times\frac{6\times10^{6}Hz}{12(256-x)}$$

解得

$$x=243D=F3H$$

下面一段主程序和中断服务程序，是利用串行方式 1 从数据 00H 开始连续不断增大地串行发送一片数据的程序例。设单片机晶振的频率为 6MHZ，波特率为 1200 位 / 秒。

ORG 2000H	;1200 位/秒的定时器初值
MOV TL1,#0F3H	
MOV TH1,#0F3H	;使 SMOD=0
MOV PCON,#00H	;T1 方式 2
MOV TMOD,#20H	
SETB EA	
CLR ET1	;关闭 T1 中断
SETB ES	;开串行中断
SETB TR1	;开 T1 定时
MOV SCON,#40H	;串行方式 1
CLR A	
MOV SBUF,A	;串行发送
JNB T1,$	;等待发送完
CLR T1,	;清标志
SJMP $	
ORG 0023H	;串行中断入口地址
MOV SBUF,A	;连续发送
JNB T1,$	
INC A	
CLR T1	
RET1	;中断返回

51 单片机汇编语言教程：19 课:单片机定时器、中断试验

我们在学单片机时我们第一个例程就是灯的闪烁，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，我们能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 ;不然跳转到 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。

以上程序是能实现灯的闪烁了，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，我们能在 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能用一些指令来替代 DJNZ 吗？是的，但是那就要求你精确计算所用指令的时间，然后再减去对应的 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，这样的办法还是不好，所以我们常用以下的办法来实现。

程序 2：用中断实现

```
ORG 0000H     ,http://www.51hei.com

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
```

上面的例程中，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，中断程序能写在任意地方，也能写任意长度了。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）。

上面的两个单片机程序运行后，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？我们能计算一下，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？我们的晶体震荡器是 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？在该晶体震荡器濒率下，最长的定时也就是 65。536 个毫秒啊！上面给出一个例程。

```
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，不然直接返回，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，我们就能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
```

您能用定时器的办法实现前面讲的流水灯吗？试试看。

51 单片机汇编语言教程：20 课:单片机定时/计数器实验

前面我们做了定时器的实验，现在来看一看计数实验，在工作中计数常常会有两种要求：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 组成的振荡器连到定时/计数器 1 的外部管脚 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 ,http://www.51hei.com

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 也在持续变化,注意观察,是不是按二进制：

00000000

00000001

00000010

00000011

这样的次序在变呢？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
```

上面这个单片机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 OOOOH

AJMP START

ORG OOO3H ;外部中断地直入口

AJMP INTO

ORG 3OH

START: MOV SP,#5FH

MOV P1,#OFFH ;灯全灭

MOV P3,#OFFH ;P3 口置高电平

SETB EA

SETB EXO

AJMP $

INTO:

PUSH ACC

PUSH PSW

CPL P1.O

POP PSW

POP ACC

RETI

END
```

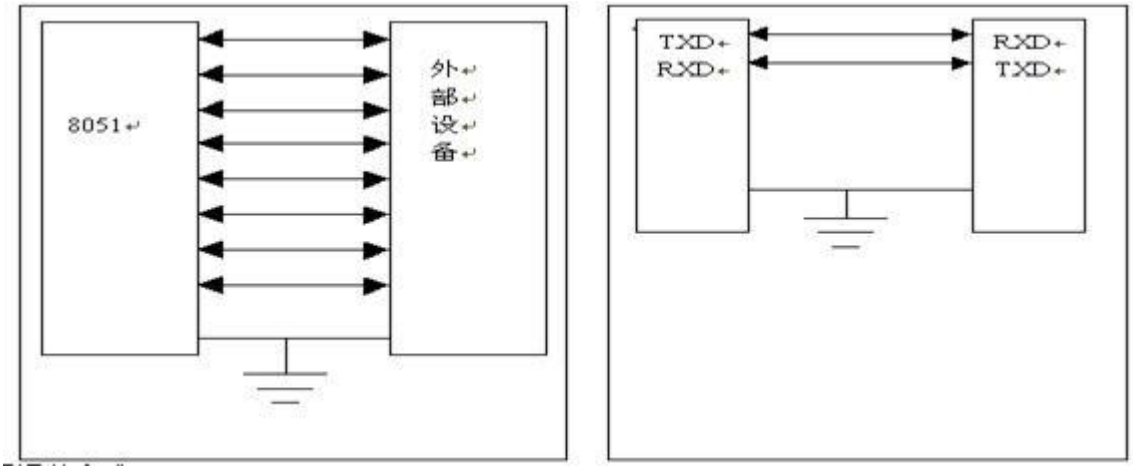
本程序的功能很简单，按一次按钮 1（接在 12 管脚上的）就引发一次中断 0，取反一次 P1。0，因此理论上按一下灯亮，按一下灯灭，但在实际做实验时，可能会发觉有时不“灵”，按了它没反应，但在大部份时候是对的，这是怎么回事呢？我们在讲解键盘时再作解释，这个程序本身是没有问题的。

51 单片机汇编语言教程：21 课:单片机串行口介绍

介绍：串行口是单片机与外界进行信息交换的工具。

8051 单片机的通信方式有两种：

并行通信:数据的各位同时发送或接收。 串行通信:数据一位一位次序发送或接收。参看下图：



<单片机串行通信>
<并行通信>

串行通信的方式：

异步通信：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（能省略），最后是停止位 1。用这种格式表示字符，则字符能一个接一个地传送。

在异步通信中，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。原则上字符格式能由通信的双方自由制定，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。

同步通信：在同步通信中，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。

通信方向：在串行通信中，把通信接口只能发送或接收的单向传送办法叫单工传送；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 单片机串行接口是一个可编程的全双工串行通信接口。它可用作异步通信方式（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信协议进行全双工的 8051 多机系统也能通过同步方式，使用 TTL 或 CMOS 移位寄存器来扩充 I/O 口。

8051 单片机通过管脚 RXD（P3.0，串行数据接收端）和管脚 TXD（P3.1，串行数据发送端）与外界通信。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 为晶体震荡器频率

SM2：多机通信控制位。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，或在多机通信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，用软件查询的办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，可用查询的办法获知或者用中断的办法获知。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

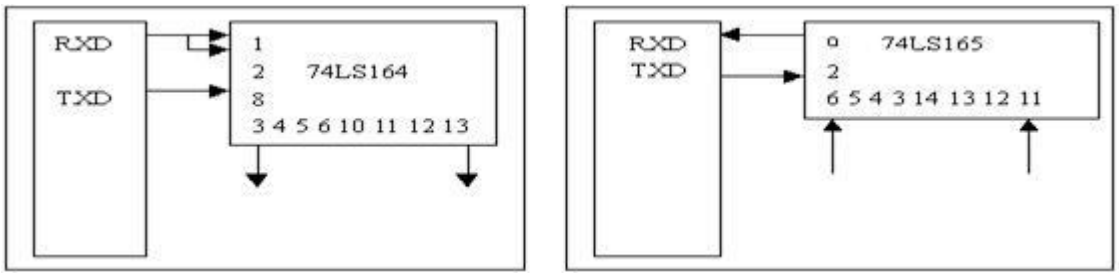
8051 单片机的全双工串行口可编程为 4 种工作方式，现分述如下：

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，也能外接同步输入/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 管脚输出，TXD 管脚输出移位脉冲。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。



<单片机串行口接线图>

方式 1 为波特率可变的 10 位异步通信接口方式。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 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。

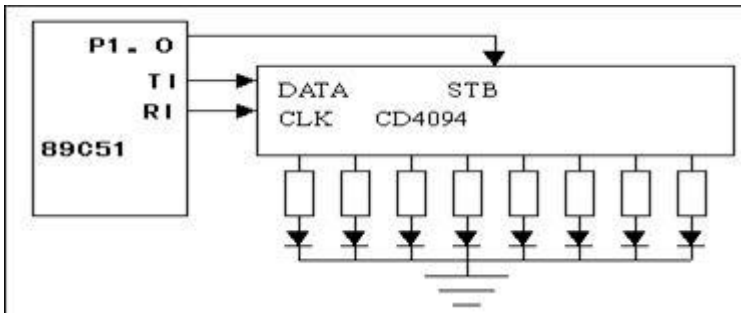
下表列出了定时器 T1 工作于方式 2 常用波特率及初值。

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

<<上一贴　　下一

51 单片机汇编语言教程：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。

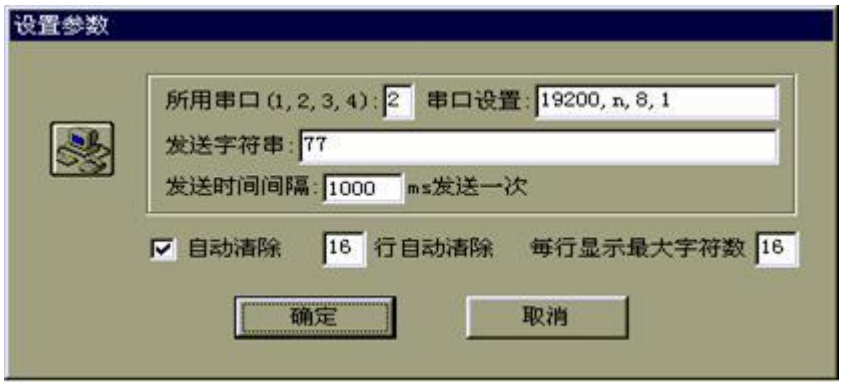
二、串行口异步通信

```
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。