第八十五节： 定时中断的寄存器配置。

【85.1　寄存器配置的本质。】

单片机内部可供我们选择的资源非常丰富，有定时器，有串口，有外部中断，等等。这些丰富的资源，就像你进入一家超市，你只需选择你所需要的东西就可以了，所以配置寄存器的关键在于选择，所谓选择就是往寄存器里面做填空题，单片机系统内部再根据你的“选择清单”，去启动对应的资源。那么我们怎么知道某个型号的单片机内部有哪些资源呢？看该型号“单片机的说明书”呀，“单片机的说明书”就是我们通常所说的“芯片的 datasheet”，或者说是“芯片的数据手册”，这些资料单片机厂家会提供的。

跟单片机打交道，其实跟人打交道没什么区别，你要让单片机按照你的“意愿”走，你首先要把你的“意愿”表达清楚，这个“意愿”就是信息，信息要具备准确性和唯一性，不能模凌两可。比如，现在要让单片机“每 1ms 产生一次中断”，你想想你可能需要给单片机提供哪些信息？

（1）51 单片机有 2 个定时器，一个是 0 号定时器，一个是 1 号定时器，我们要面临“二选一”的选择，本例子中用的是“0 号定时器”。

（2）0 号定时器内部又有 4 种工作方式：方式 0，方式 1，方式 2，方式 3，本例子中用的是“方式 1”。

（3）定时器到底多长时间中断一次，这就涉及到填充与中断时间有关的寄存器的数值，该数值是跟时间成比例关系，本例子中配置的是 1ms 中断，就要填充对应的数值。

（4）默认状态下，定时器是不会被开启的，如果要开启，这里就是涉及到定时器的“开关”，本例子要开启此开关。

（5）定时器时间到了就要产生中断，中断也有“总开关”和“定时器的局部开关”，这两个开关都必须同时打开，中断才会有效。

要配置定时器“每 1ms 产生一次中断”，大概就上述这些信息，根据这些信息提示，下面开始讲解一下寄存器的具体内容。

【85.2　定时器/计数器的模式控制寄存器 TMOD。】

寄存器 TMOD 是一个 8 位的特殊变量，里面每一位都代表了不同的功能选择。根据芯片的说明书，TMOD 的 8 位从左到右依次对应从 D7 到 D0（左高位，右低位），定义如下：

```
GATE    C/T    M1     M0    GATE    C/T    M1     M0
```

仔细观察，发现左 4 位与右 4 位是对称的，分别都是“GATE，C/T ， M1 ， M0”，左 4 位控制的是“定时器 1”，右 4 位控制的是“定时器 0”，因为本例子用的是“定时器 0”，因此“定时器 1”的左 4 位都设置为 0 的默认数值，我们只需重点关注右 4 位的“定时器 0”即可。

GATE：定时器是否受“其它外部开关”的影响的标志位。定时器的开启或者停止，受到两个开关的影响，第一个开关是“自身原配开关”，第二个开关是“其它外部开关”。GATE 取 1 代表定时器受“其它外部开关”的影响，取 0 代表定时器不受“其它外部开关”的影响。本例子中，定时器只受到“自身原配开关”的影响，而不受到“其它外部开关”的影响，因此，GATE 取 0。

C/T：定时器有两种模式，该位取 1 代表“计数器模式”，取 0 代表“定时器模式”。本例子是“定时器模式”，因此，C/T 取 0。

M1 与 M0：工作方式的选择。M1 与 M0 这两位的 01 搭配，可以有 4 种组合（00,01,10,11），每一种组合就代表一种工作方式。本例子选用“方式 1”，因此 M1 与 M0 取“01”的组合。

综上所述，TMOD 的配置代码是：TMOD=0x01;

【85.3　决定时间长度的寄存器 TH0 与 TL0。】

TH0 与 TL0，T 代表定时器英文单词 TIME 的 T，H 代表高位，L 代表低位，0 代表定时器 0。

TH0 是一个 8 位宽度的寄存器，TL0 也是一个 8 位宽度的寄存器，两者合并起来成为一个整体，实际上就是一个 16 位宽度的寄存器，TH0 是高 8 位，TL0 是低 8 位，它们合并后的数值范围是：0 到 65535。该 16 位寄存器取值越大，定时中断一次的时间反倒越小，为什么？TH0 与 TL0 的初始值，就像一个水桶里装的水。如果这个桶是空桶（取值为 0），“雨水”想把这个桶“滴满溢出”所需要的时间就很大。如果里面已经装了大半的水（取值为大于 32767），“雨水”想把这个桶“滴满溢出”所需要的时间就很小。这里的关键词“滴满溢出”的“滴”与“满溢出”，“滴”的速度是由单片机晶振决定的，而“满溢出”一次就代表产生一次中断，执行完中断函数在即将返回主函数之前，我们重新装入特定容量的水（重装初值），为下一次的“滴满溢出”做准备，依次循环，从而连续不断地产生间歇的定时中断。

配置中断时间的大小是需要经验的，因为，每次定时中断的时间太长，就意味着时间的可分度太粗，而如果每次定时中断的时间太短，则会产生很频繁的中断，势必会影响主函数 main()的执行效率，而且累记中断次数的时间误差也会增大。因此，配置中断时间是需要经验的，根据经验，定时中断取 1ms 一次，是几乎所有单片机项目的最佳选择，按我的理解，“1ms 定时中断一次”已经是单片机界公认的一种“标配”。

要配置 1ms 定时中断，TH0 与 TL0 如何取值？刚才提到一个形象的例子“桶，滴，满溢出”。TH0 与 TL0 的最大取值范围是 65535，可以理解成为最大 65535“滴”，如果超过 65535“滴”(比如加 1“滴”后变成 65536“滴”)就会“满溢出”，从而产生一次中断（65536 是中断发生的临界值）。而“滴一次的时间”就刚好是单片机执行“一次单指令的时间”，“一次单指令的时间”等于 12 个晶振周期，比如 12MHz 的晶振，晶振周期是（1/12000000）秒，而“一次单指令的时间”就等于 12 乘以（1/12000000）秒，等于 0.000001 秒，也就是 1us。1us“滴”一次，要产生 1ms 的时间就需要“滴”1000 次。“满溢出”的前提条件是“桶里”一共需要装入 65536 滴才溢出，因此，在 12MHz 的晶振下要产生 1ms 的定时中断，TH0 与 TL0 的初值应该是 64536（65536 减去 1000 等于 64536），而 64536 变成十六进制 0xfc17，再分解到高 8 位 TH0 为 0xfc，低 8 位 TL0 为 0x17。

刚才的例子是假如晶振在 12MHz 的情况下所计算出来的结果，而本教程所用的晶振是 11.0592MHz，根据 11.0592MHz 产生 1ms 的定时中断，TH0 与 TL0 应该取值多少？根据刚才的计算方式:

```
初值=[溢出值]-([0.001 秒]/([晶振周期的 12 个]*([1 秒]/[晶振频率])))
初值=65536-(0.001/(12*(1/11059200)))
初值=65536-922        (注：922 是 921.6 的四舍五入)
初值=64614
初值=64614
初值=0xfc66
初值 TH0=0xfc
初值 TL0=0x66
```

【85.4　中断的总开关 EA 与局部开关 ET0。】

EA：中断的总开关。宽度是 1 位的位变量。此开关如果取 0，就会强行屏蔽所有的中断，因此，只要用到中断，此开关必须取 1。

ET0：专门针对定时器 0 中断的局部开关。宽度是 1 位的位变量。此开关如果取 0，则会屏蔽定时器 0 的中断，如果取 1 则允许定时器 0 中断。如果要定时器 0 能产生中断，那么总开关 EA 与 ET0 必须同时都打开（都取 1），两者缺一不可。

【85.5　定时器 0 的“自身原配开关”TR0。】

TR0：定时器的“自身原配开关”。宽度是 1 位的位变量。很多初学者会把 EA，ET0，TR0 三者搞不清。定时器可以工作在“查询标志位”和“中断”这两种状态，也就是说在没有中断的情况下定时器也可以单独使用的。TR0 是定时器 0 自身的发动引擎，要不要把这个发动引擎所产生的能量传输到中断的渠道，则取决于中断开关 EA 和 ET0。TR0 是源头开关，EA 是中断总渠道开关，ET0 是中断分支渠道的定时器 0 开关。TR0 取 1 表示启动定时器 0，取 0 表示关闭定时器 0。

【85.6　定时器 0 的中断函数的书写格式。】

```
void 函数名() interrupt 1
 {
      ...中断程序内容;
      ...此处省去若干代码
      ...中断程序内容;
      ...最后面的代码，要记得重装 TH0 与 TL0 的初值;
}
```

函数名可以随便取，只要不是编译器已经征用的关键字。这里的 1 是定时器 0 的中断号。不同的中断号代表不同类型的中断，至于哪类中断对应哪个中断号，大家可以查找相关书籍和资料。本节用的定时器 0 处于工作方式 1 的情况下，在即将退出中断之前，需要重装 TH0 与 TL0 的初始值。

【85.7　寄存器的名字来源。】

前面讲的寄存器都有固定的名字，而且这些名字都是唯一的，拼写的时候少一个字母或者多一个字母，C 编译器都会报错不让你通过，因此问题来了，初学者刚接触一款单片机的时候，如何知道某个寄存器它特定的唯一的名字？有两个来源。

第一个来源，可以打开 C 编译器的某个头文件（.h 格式）查看这些寄存器的名字。比如 51 单片机可以查看 REG52.H 这个头文件。如何打开 REG52.H 这个文件？在 keil 源代码编辑器界面下，选中上面 REG52.H 这几个字符，在右键弹出的菜单下点击 Open ducument“REG52.H”即可。

第二个来源是直接参考一些现成的范例程序，这些范例程序网上很多，有的是原厂提供的，有的是热心网友的分享，有的是技术书籍或者学习板开发板厂家提供的。

【85.8　如何快速配置寄存器。】

建议一边阅读芯片的数据手册，一边参考一些现成的范例程序，这些范例程序网上很多，有的是原厂提供的，有的是热心网友的分享，有的是技术书籍或者学习板开发板厂家提供的。

【85.9　练习例程。】

现在编写一个定时中断程序，让两个 LED 灯闪烁，一个是在主函数里用累计主循环次数的方式实现（P0.0 控制），另一个是在定时中断函数里用累计定时中断次数的方式实现（P0.1 控制）。这两个闪烁的 LED 灯，一个在 main 函数，一个是在中断函数，两路任务互不干涉独立运行，并行处理的“雏形”略显出来。

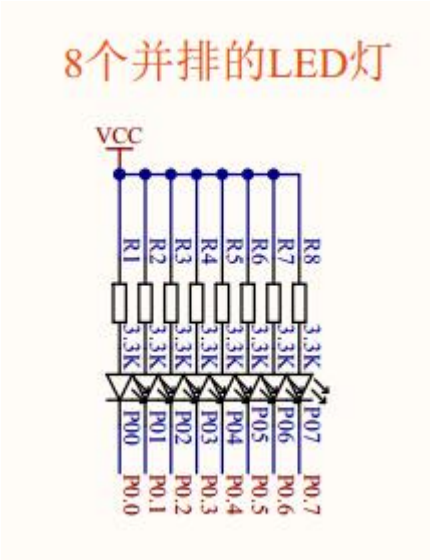

图 85.9.1　灌入式驱动 8 个 LED

```
#include "REG52.H"

#define  CYCLE_SUM   5000   //主循环的次数

#define  INTERRUPT_SUM   500   //中断的次数

sbit P0_0=P0^0;  //在主循环里的 LED 灯
sbit P0_1=P0^1;  //在定时中断里的 LED 灯

unsigned char Gu8CycleStep=0;
unsigned long Gu32CycleCnt=0;      //累计主循环的计数器

unsigned char Gu8InterruptStep=0;
unsigned long Gu32InterruptCnt=0;  //累计定时中断次数的计数器

void main()
{
    TMOD=0x01;  //设置定时器 0 为工作方式 1
    TH0=0xfc;   //产生 1ms 中断的 TH0 初始值
    TL0=0x66;   //产生 1ms 中断的 TL0 初始值
    EA=1;       //开总中断
    ET0=1;      //允许定时 0 的中断
    TR0=1;      //启动定时 0 的中断

    while(1)  //主循环
    {
       switch(Gu8CycleStep)
```

```
       {
           case 0:
               Gu32CycleCnt++;
               if(Gu32CycleCnt>=CYCLE_SUM)
                {
                    Gu32CycleCnt=0;
                    P0_0=0;  //主循环的 LED 灯亮。
                    Gu8CycleStep=1;
                }
               break;

           case 1:
               Gu32CycleCnt++;
               if(Gu32CycleCnt>=CYCLE_SUM)
                {
                    Gu32CycleCnt=0;
                    P0_0=1;   //主循环的 LED 灯灭。
                    Gu8CycleStep=0;
                }
               break;
        }
   }
}


void T0_time() interrupt 1    //定时器 0 的中断函数，每 1ms 单片机自动执行一次此函数
{
       switch(Gu8InterruptStep)
       {
           case 0:
               Gu32InterruptCnt++;   //累计中断次数的次数
               if(Gu32InterruptCnt>=INTERRUPT_SUM) //次数达到设定值就跳到下一步骤
                {
                    Gu32InterruptCnt=0;     //及时清零计数器，为下一步骤的新一轮计数准备
                    P0_1=0;  //定时中断的 LED 灯亮。
                    Gu8InterruptStep=1;  //跳到下一步骤
                }
               break;

           case 1:
               Gu32InterruptCnt++;   //累计中断次数的次数
               if(Gu32InterruptCnt>=INTERRUPT_SUM)  //次数达到设定值就返回上一步骤
                {
                    Gu32InterruptCnt=0;     //及时清零计数器，为返回上一步骤的新一轮计数准备
```

```
                P0_1=1;  //定时中断的 LED 灯灭。
                Gu8InterruptStep=0;  //返回到上一个步骤
            }
            break;
        }

        TH0=0xfc;   //重装初值，不能忘。
        TL0=0x66;   //重装初值，不能忘。
}
```