第 8 课，定时器中断跑马灯

在第 3 课，我们用指令延时方式实现了跑马灯。这里我们用定时器方式再次实现，定时器方式有效率高，定时准确等优点。

一个编程经验是，所有的中断都要尽快的运行和退出，中断服务程序越短越好，这样才不至于干扰主程序的工作和其他中断的运行。

也就是，我们应该尽量把程序代码从中断服务函数里搬出来。

对于定时器的中断的工作方式，我们可以建立一个全局的标记，在中断里置这个标记，然后就退出。在主程序里检查到这个标记之后，就运行相关的程序。对于 CPU 任务比较多的项目来说，这种工作方式可以获得最佳的工作效率。当然，对于非常实时的应用要求，，比如时钟，还是建议在中断里做完，因为使用标记的方式时，主程序可能太忙而造成错过标记信号，就是这个标记还没有开始处理呢，下一个又来了。熟练的程序员还是可以避开这些异常的情况的。

在我们的这个例程中，前一课的 1 秒钟输出信号，被换成了一个全局标记。在主程序中去检查这个标记，再清 0 标记和处理相应的工作。

这一课的跑马灯输出方式也改变了，我们采用查表的方式，将要点亮的灯预先设置好，到了时间，就一起送到 P1 口。这样，程序的执行效率会更高。

下面请认真学习和分析例程：

以下是例程，请打开 lesson8 目录的工程，内容是一样的。

——

```
#define uchar unsigned char //定义一下方便使用
#define uint    unsigned int
#define ulong unsigned long
#include <reg52.h>          //包括一个 52 标准内核的头文件

sbit P10 = P1^0; //头文件中没有定义的 IO 就要自己来定义了
sbit P11 = P1^1;
sbit P12 = P1^2;
sbit P13 = P1^3;
bit ldelay=0;     //长定时溢出标记,预置是 0

char code dx516[3] _at_ 0x003b;//这是为了仿真设置的
//定时器中断方式的跑马灯
void main(void)      //  主程序
{
    uchar code ledp[4]={0xfe,0xfd,0xfb,0xf7};//预定的写入 P1 的值
    uchar ledi;         //用来指示显示顺序

    RCAP2H =0x10; //赋 T2 的预置值 0x1000，溢出 30 次就是 1 秒钟
    RCAP2L =0x00;
    TR2=1;       //启动定时器
    ET2=1;       //打开定时器 2 中断
```

```
    EA=1;        //打开总中断

    while(1)   //主程序循环
    {
        if(ldelay)       //发现有时间溢出标记，进入处理
        {
            ldelay=0;   //清除标记
            P1=ledp[ledi]; //读出一个值送到 P1 口
            ledi++;          //指向下一个
            if(ledi==4)ledi=0;  //到了最后一个灯就换到第一个
        }
    }
}
//定时器 2 中断
timer0() interrupt 5
{
    static uchar t;
    TF2=0;
    t++;
    if(t==30) //T2 的预置值 0x1000，溢出 30 次就是 1 秒钟,晶振 22118400HZ
    {
        t=0;
        ldelay=1;//每次长时间的溢出，就置一个标记，以便主程序处理
    }
}
```

————————————————————————————

编译，进入仿真，看结果。可以看到 4 个灯以精确的 1 秒的速度不断循环跑动。

作业：

现在的灯是从左到右跑动，请改为从右到左跑动。