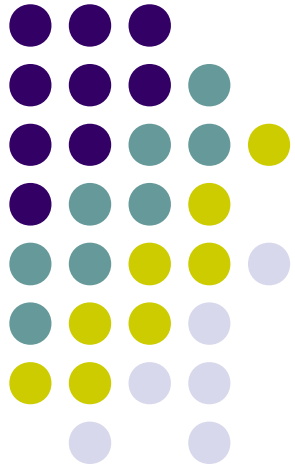
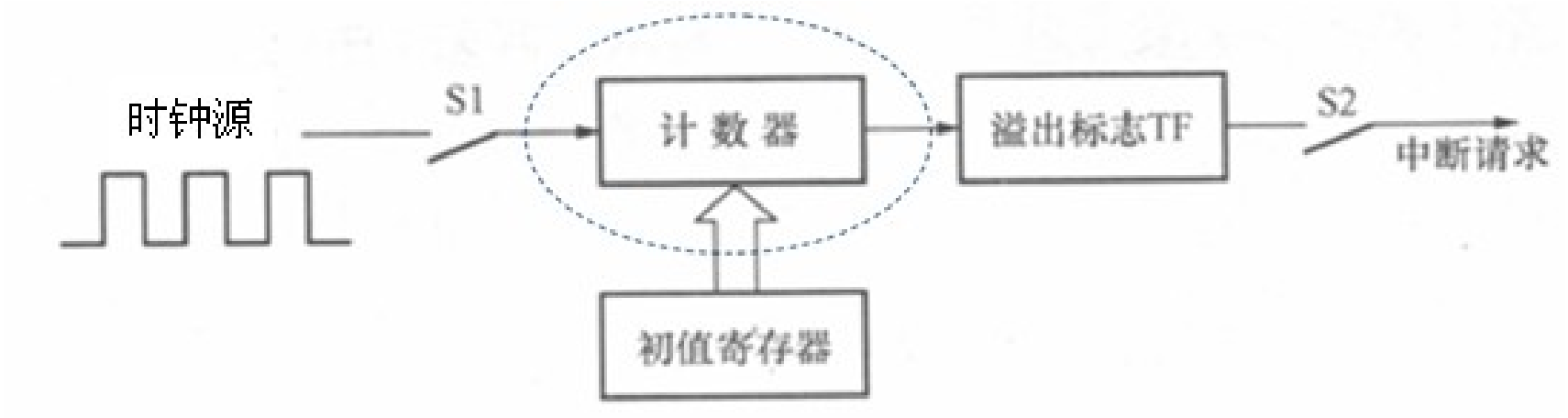
项目 5　定时与中断系统设计

本章内容

- 定时 / 计数器
- 中断系统



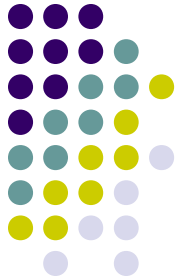
定时／计数器工作原理



S1：启动或停止计数器工作

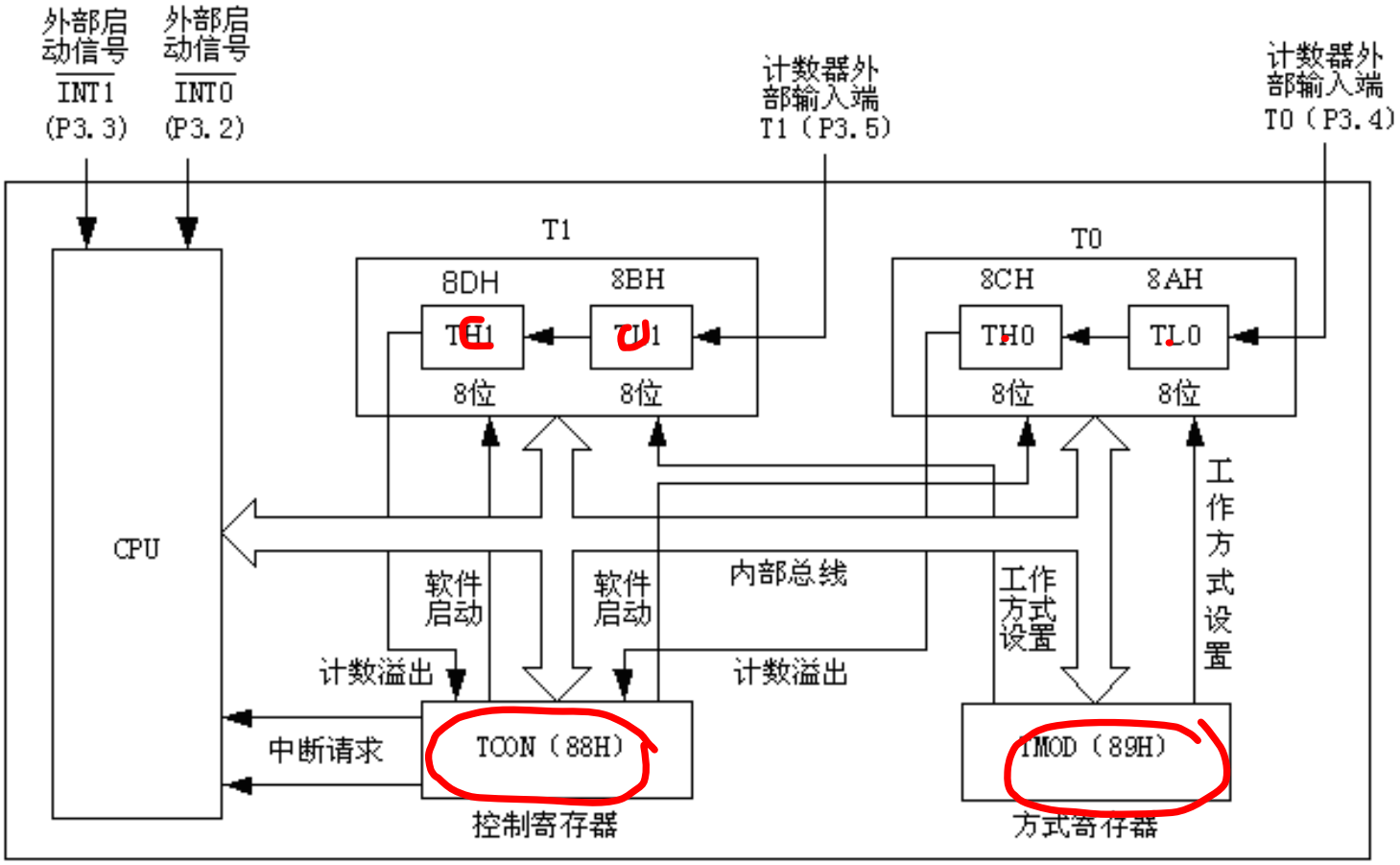
S2：允许或禁止溢出中断

单片机定时 / 计数器



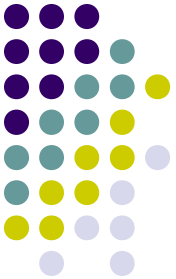
8051 单片机内部有两个 16 位的可编程定时 / 计数器，称为 T0 （ T0 ）和 T1 （ T1 ）

单片机定时／计数器



8051 定时器 / 计数器逻辑结构

单片机定时/计数器



设置定时/计数器工作方式

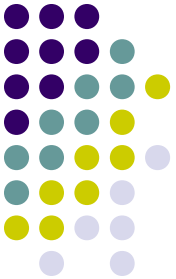
- 通过对方式寄存器 TMOD 的设置，确定相应的定时/计数器是定时功能还是计数功能，工作方式以及启动方法。
- 定时/计数器工作方式有四种：方式 0、方式 1、方式 2 和方式 3。
- 定时/计数器启动方式有两种：软件启动和硬软件共同启动。除了从控制寄存器 TCON 发出的软件启动信号外，还有外部启动信号引脚，这两个引脚也是单片机的外部中断输入引脚。

单片机定时／计数器

设置计数初值

- T0 、 T1 是 16 位加法计数器，分别由两个 8 位专用寄存器组成， T0 由 TH0 和 TL0 组成， T1 由 TH1 和 TL1 组成。 TL0 、 TL1 、 TH0 、 TH1 的访问地址依次为 8AH~8DH ，每个寄存器均可被单独访问，因此可以被设置为 8 位、 13 位或 16 位计数器使用。
- 在计数器允许的计数范围内，计数器可以从任何值开始计数，对于加 1 计数器，当计到最大值时（对于 8 位计数器，当计数值从 255 再加 1 时，计数值变为 0 ），产生溢出。
- 定时／计数器允许用户编程设定开始计数的数值，称为赋初值。初值不同，则计数器产生溢出时，计数个数也不同。例如：对于 8 位计数器，当初值设为 100 时，再加 1 计数 156 个，计数器就产生溢出；当初值设为 200 时，再加 1 计数 56 个，计数器产生溢出。

单片机定时/计数器



启动定时/计数器

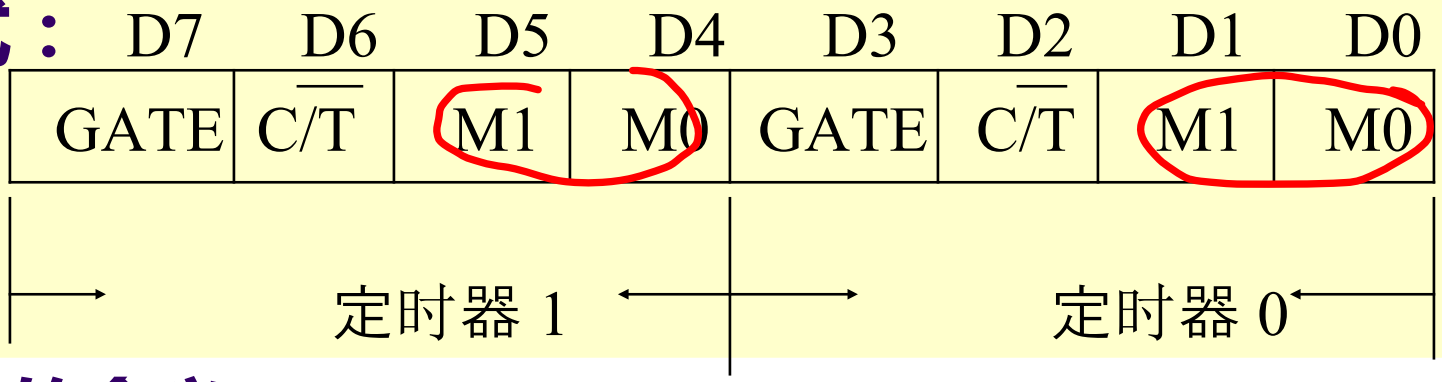
根据设置的定时/计数器启动方式，启动定时/计数器。如果采用软件启动，则需要把控制寄存器中的 TR0 或 TR1 置 1；如果采用硬软共同启动方式，不仅需要把控制寄存器中的 TR0 或 TR1 置 1，还需要相应外部启动信号为高电平。

定时器的方式寄存器 TMOD

作用： TMOD 用来确定两个定时器的工作方式。低半字节设置定时器 T0，高半字节设置定时器 T1 。

字节地址： 89H，不可以位寻址。

格式：



各位的含义：

$C/\overline{T}$：功能选择位。 0 为定时器方式； 1 为计数器方式。

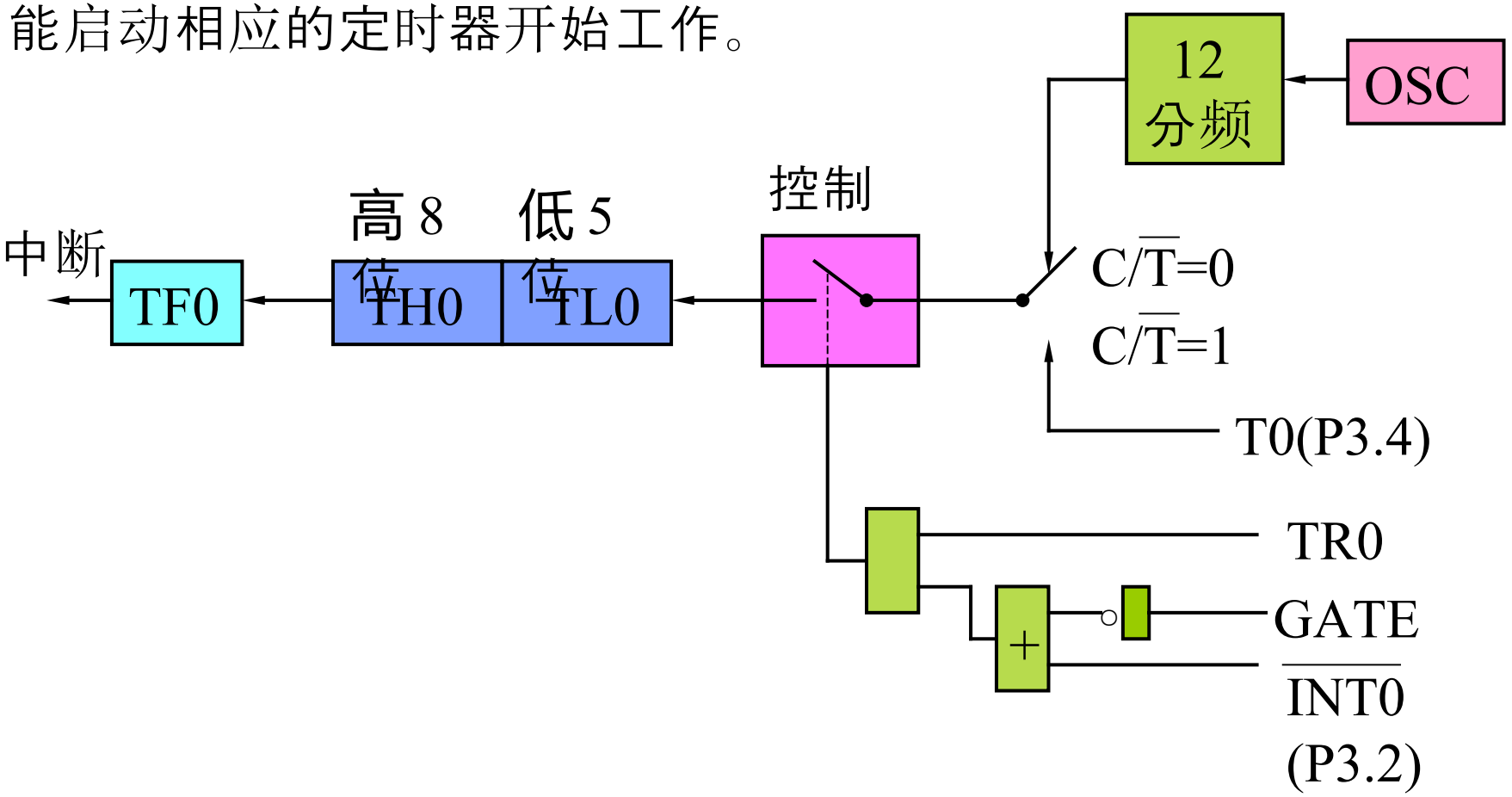
M1， M0：方式选择位。可以选择为四种工作方式 0 、 1 、 2 、 3 之 1 。

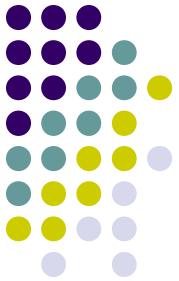
四种工作方式的区别后面讲解。

GATE：门控位。

0 ：只要软件控制位 TR0 或 TR1 置 1 即可启动定时器开始工作；

1 ：只有INT0或INT1引脚为高电平，且TR0或TR1置1时， 才能启动相应的定时器开始工作。





例如：设定时器 T0 为定时工作方式，要求用软件启动

定时器 T0 工作，按方式 1 工作；定时器 T1 为计数

工作方式，要求软件启动，工作方式为方式 2 。

则根据 TMOD 各位的定义可知，其控制字为：

格式：

D7	D6	D5	D4	D3	D2	D1	D0
GATE	$\overline{C/T}$	M1	M0	GATE	$\overline{C/T}$	M1	M0
0	1	1	0	0	0	0	1

即控制字为 61H，其对应为：

TMOD=0x61

定时器的控制寄存器 TCON

作用： TCON 用来控制两个定时器的启动、停止，表明定时器的溢出、中断情况。

字节地址： **0x**88，可以位寻址。系统复位时，所有位均清零。

格式：

D7	D6	D5	D4	D3	D2	D1	D0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

各位的含义： TCON 中的低 4 位与中断有关，在中断章节中讨论。

TF1 (8FH)：定时器 1 溢出标志。计满后自动置 1 。

TR1 (8EH)：定时器 1 运行控制位。由软件清零关闭定时器 1 。

当 GATE=0 时，TR1 软件置 1 即启动定时器 1 。（SETB TR1）

定时器的四种工作方式

方式的选择：根据 M1 ， M0 来选择。

00 ：方式 0　01 ：方式 1　10 ： 方式 2　11 ： 方式 3

主要特点：

方式 0 ：13 位定时器。　TH0 的 8 位 +TL0 的低 5 位

方式 1 ： 16 位定时器。　TH0 的 8 位 +TL0 的 8 位

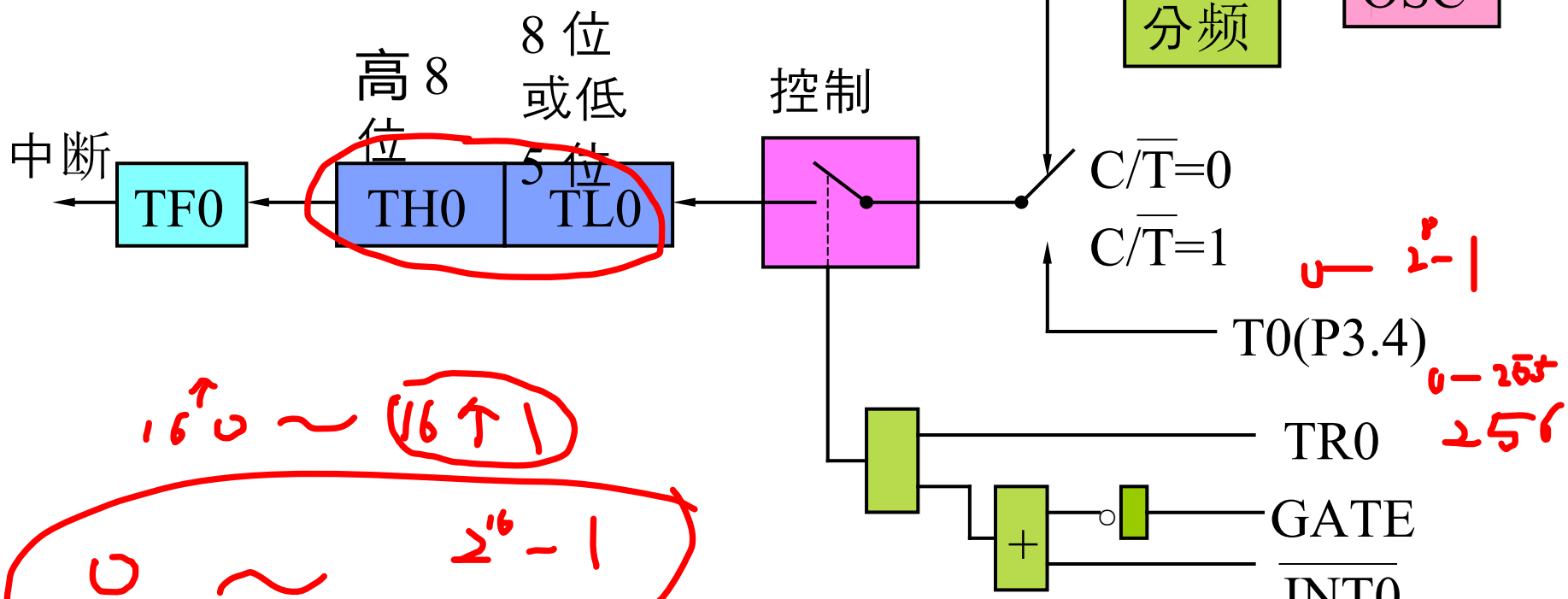
方式 2 ：能重复置初始值的 8 位定时器 。　TL0 和 TH0 必须赋相同的值。

方式 3 ：只适用于定时器 0 ， T0 被拆成两个独立的 8 位定时器 TL0,TH0 。

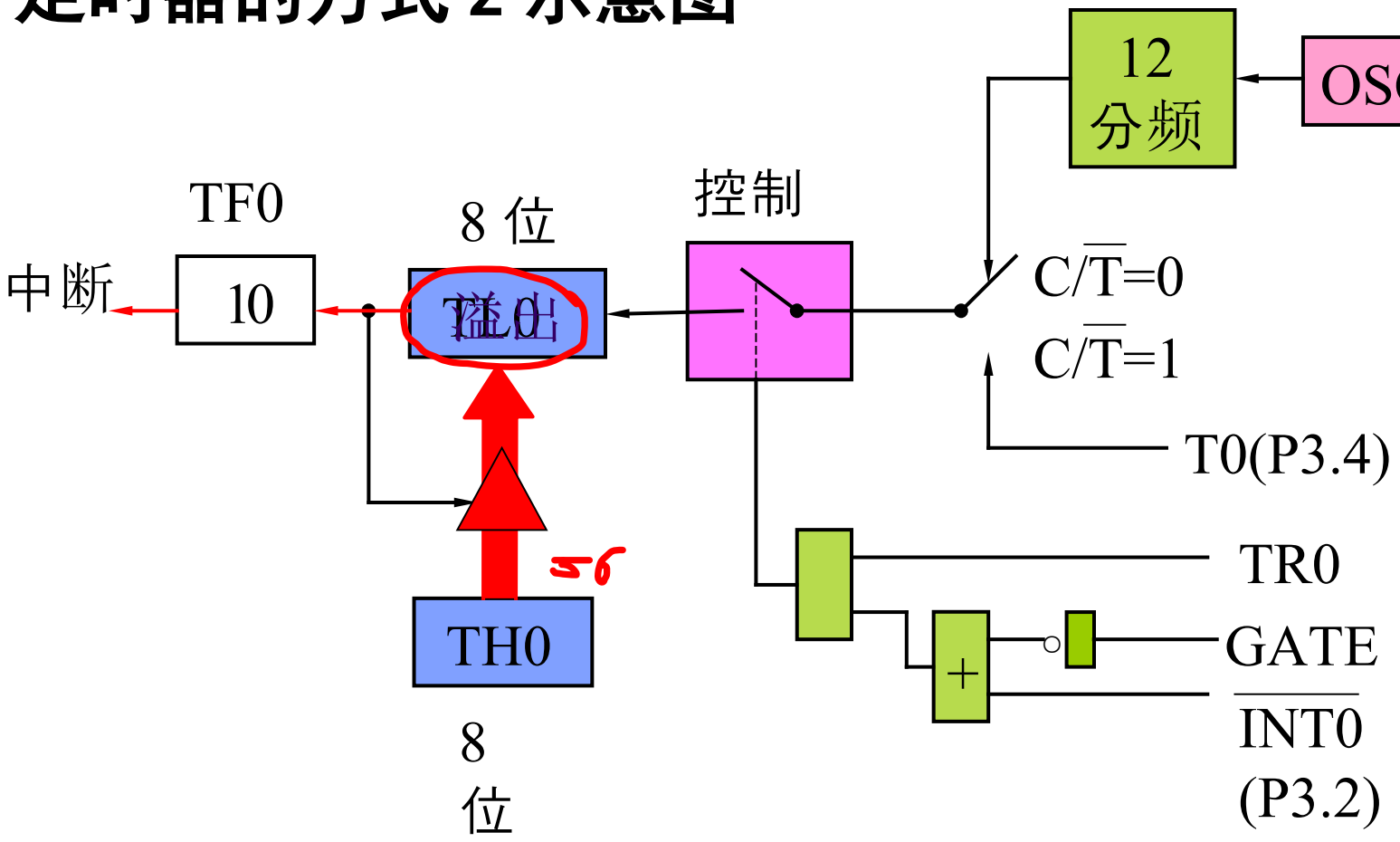
其中： **TL0 与方式 0 、 1 相同，可定时或计数**。用定时器 T0 的

GATE 、 C/T 、 TR0 、 TF0 、 T0 、和 INT0 控制

定时器的方式 0 、 1 示意图



定时器的方式 2 示意图


12
分频
OSC
TF0
中断
10
8 位
TL0
溢出
TH0
8
位
控制
C/T=0
C/T=1
T0(P3.4)
TR0
GATE
INT0
(P3.2)

定时器的初始值的计算

对于不同的工作方式，计数器位数不同，故最大计数值 M 也不同：

方式 0 ： $M=2^{13}=8192$

方式 1 ： $M=2^{16}=65536$

方式 2 ： $M=2^{8}=256$

方式 3 ：定时器 0 分为 2 个 8 位计数器，每个 M 均为 256 。

因为定时 / 计数器是作加 1 计数，并在计满溢出时产生中断，因此初值 X 的计算如下：

$$X = M - \text{计数值}$$

计算出来的结果 X 转换为 16 进制数后分别写入 TL0 （ TL1 ）、 TH0 （ TH1 ）。

注意！方式 0 时初始值写入时，对于 TL 不用的高 3 位应填入 0 ！

赋初值

- 定时计数器 0 ， 定时 1s, 采用方式 1 ， 12HZM,50ms
- Void delay()
- {
- TMOD=0x01; //00000001
- For （ i=0,i<20;i++ ）
- {
- TH0=(65536-50000)/256;// (65536-50000)>>8
- TL0=(65536-50000)%256; // (65536-50000)&0x00ff;
- TR0=1;
- While(!TF0) ;
- TF1=0;
-
-
- }
- }

赋初值

- 采用方式 1， 12MHZ, 定时 50ms
- // 最大的定时范围 65536us
- // 需要的计算个数为 50ms/1us
- TH0=(65536-50000)/256;// (65536-50000)>>8
- TL0=(65536-50000)%256; // (65536-50000)&0x00ff;
-

- 采用方式 0， 12MHZ, 定时 5ms

- 采用方式 2， 12MHZ, 定时 5ms

举 例 1

用 T1 、工作方式 0 实现 1 秒延时函数，晶振频率为 12MHz 。

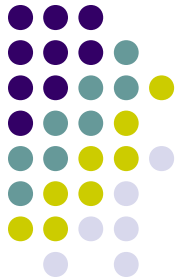
方式 0 采用 13 位计数器，其最大定时时间为： 8192×1μs = 8.192ms ，因此，定时时间不可能象任务 7 中一样选择 50ms ，可选择定时时间为 5ms ，再循环 200 次。

定时时间为 5ms ，则计数值为 5ms/1μs =5000 ， T1 的初值为：

X = M – 计数值 = 8192 – 5000 = 3192 = C78H = 0110001111000B TH1=**01100011**

TL1=000**11000**

13 位计数器中 TL1 的高 3 位未用，填写 0 ， TH1 占高 8 位，所以， X 的实际填写值应为：



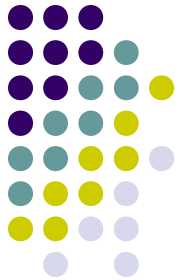
- 工作方式 0 实现 5000us 秒延时函数，晶振频率为 12MHz 。
- 方式 0 采用 13 位计数器 :
- 计数范围 :8192
- 定时范围： 8192* 机器周期（ 1us ）

定 5000us　1s/ 5000us=200 次

需要的计数值： 5000us/1us=5000 次

初值： 8192-5000=3192%32

5%8=5　10000001



- **定时计数器 1，工作方式 0 实现 1 秒延时函数，晶振频率为 12MHz 。**
- **方式 0 采用 13 位计数器：**
- TMOD=0x00;　//00000000B

举　例 1

用 T1 方式 0 实现任务 7 中 1 秒延时函数如下：

```
void delay1s()
{
  unsigned char  i;
  TMOD=0x00;                    //  置 T1 为工作方式 0
  for(i=0;i<0xc8;i++){          //  设置 200 次循环次数
   TH1=0x63;                              //  设置定时器初值
   TL1=0x18;
   TR1=1;                                 //  启动 T1
   while(!TF1);                 //  查询计数是否溢出，即定时 5ms
时间到， TF1=1
   TF1=0;                                          // 5ms 定时时间到
，将定时器溢出标志位 TF1 清零
   }
}
```

举　例 2

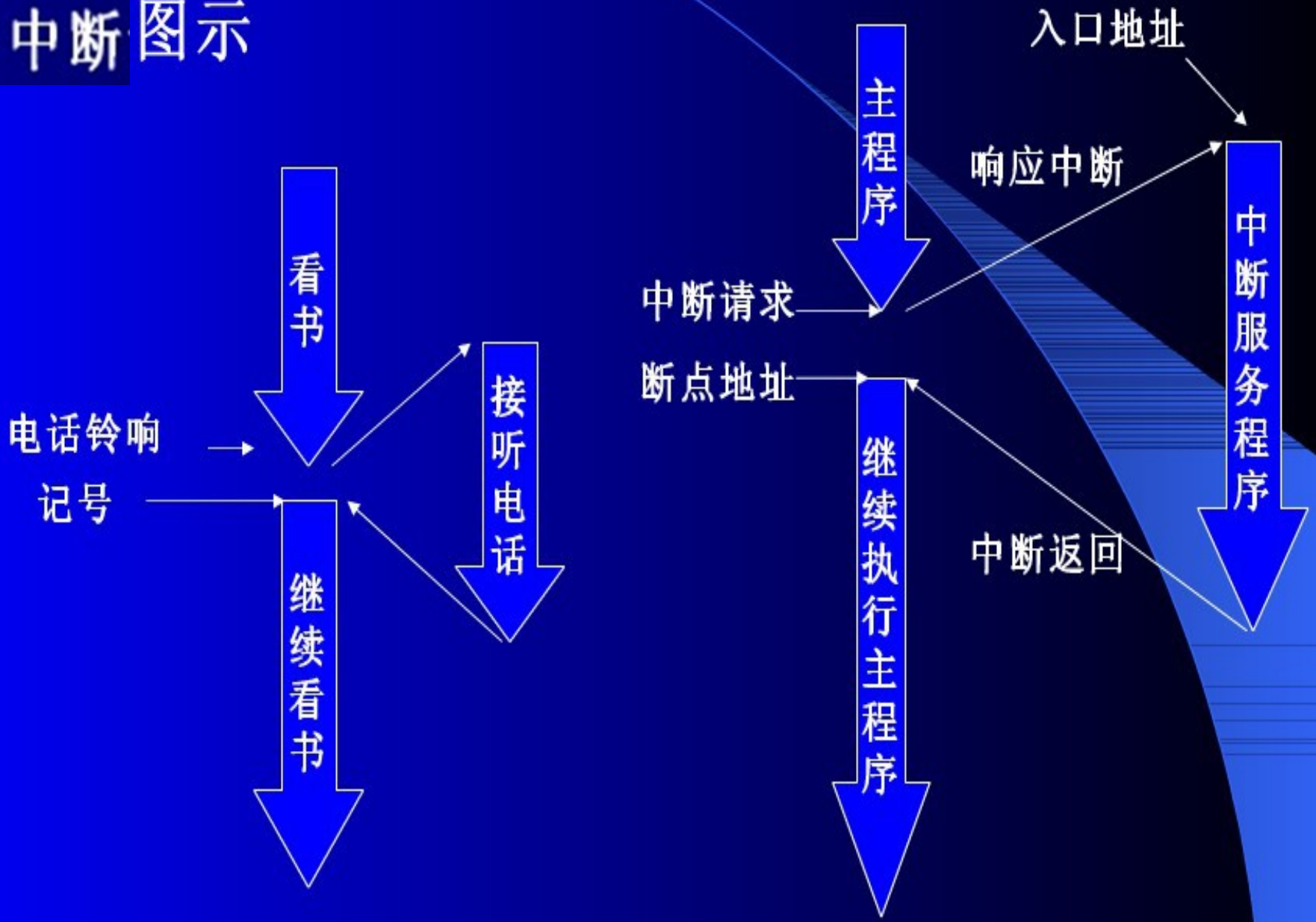
用 T1 、工作方式 2 实现 1 秒延时，晶振频率为 12MHz 。因工作方式 2 是 8 位计数器，其最大定时时间为：256×1μs = 256μs ，为实现 1 秒延时，可选择定时时间为 250μs ，再循环 4000 次。定时时间选定后，可确定计数值为 250 ，则 T1 的初值为： X = M – 计数值 =256 – 250 = 6 = 6H 。采用 T1 方式 2 工作，因此， TMOD =0x20 。

举　例 2

用定时器工作方式 2 实现的 1 秒延时函数如下：

```
void delay1s()
{
   unsigned int  i;// i 取值范围为 0 ～ 4000，因此不能定义成 unsigned char
   TMOD=0x20;                              //  设置 T1 为方式 2
   TH1=6;      //  设置定时器初值，放在 for 循环之外
   TL1=6;
  for(i=0;i<4000;i++){        //  设置 4000 次循环次数
   TR1=1;                                  //  启动 T1
   while(!TF1); //  查询计数是否溢出，即定时 250μs 时间到， TF1=1
   TF1=0;  // 250μs 定时时间到，将定时器溢出标志位 TF1 清零
   }
}
```

中断图示


看书
电话铃响
记号
接听电话
继续看书
主程序
中断请求
断点地址
响应中断
入口地址
中断服务程序
继续执行主程序
中断返回

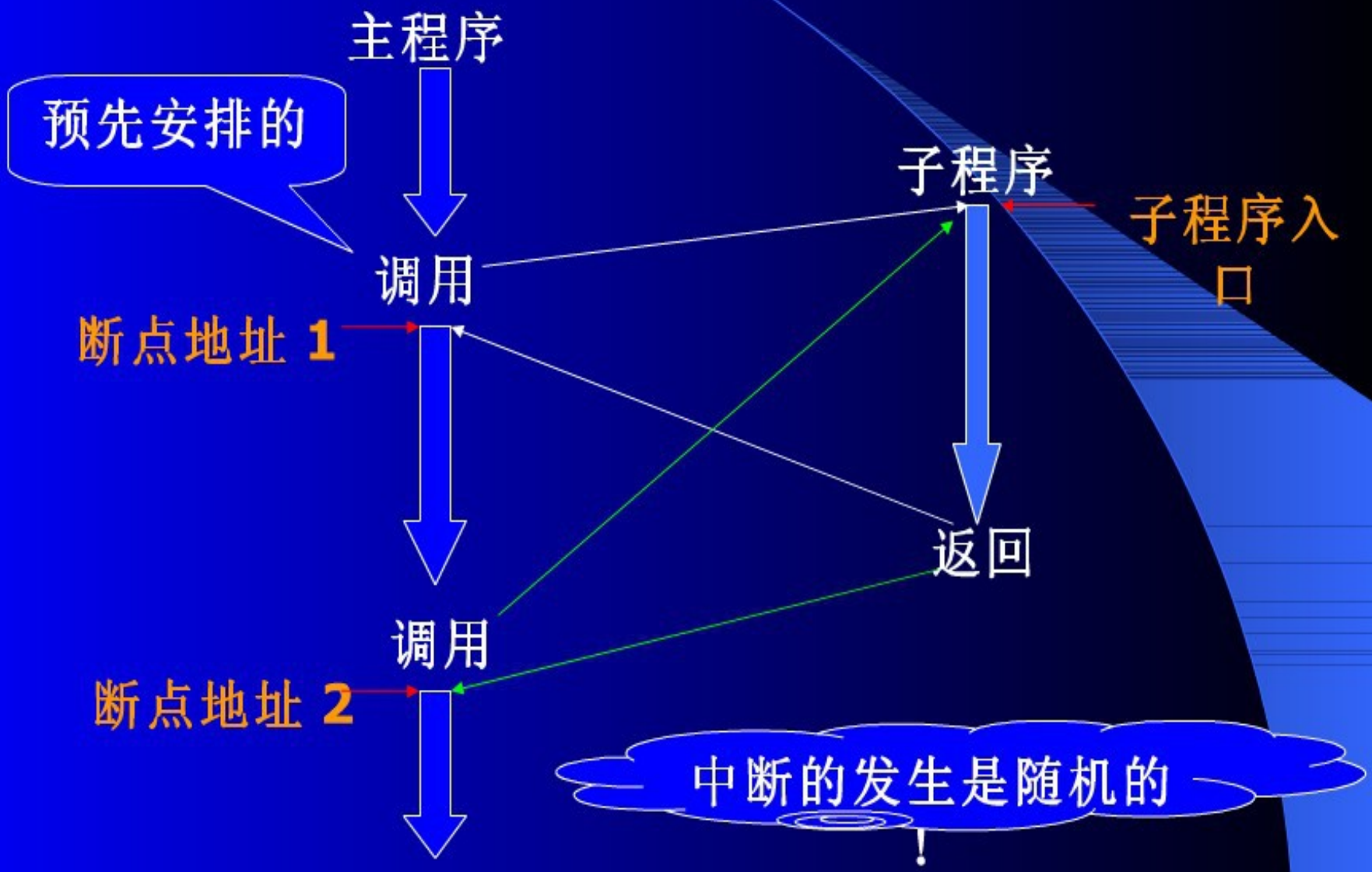
什么是中断

中断是指通过硬件来改变 CPU 的运行方向。计算机在执行程序的过程中，外部设备向 CPU 发出中断请求信号，要求 CPU 暂时中断当前程序的执行而转去执行相应的处理程序，待处理程序执行完毕后，再继续执行原来被中断的程序。这种程序在执行过程中由于外界的原因而被中间打断的情况称为“中断”。

中断基本概念

（１）中断服务程序： CPU 响应中断后，转去执行相应的处理程序，该处理程序通常称之为中断服务程序。
（２）主程序：原来正常运行的程序称为主程序。
（３）断点：主程序被断开的位置（或地址）称为断点。
（４）中断源：引起中断的原因，或能发出中断申请的来源，称为中断源。
（５）中断请求：中断源要求服务的请求称为中断请求（或中断申请。

中断与子程序调用



小提示

中断函数的调用过程类似于一般函数调用，区别在于：

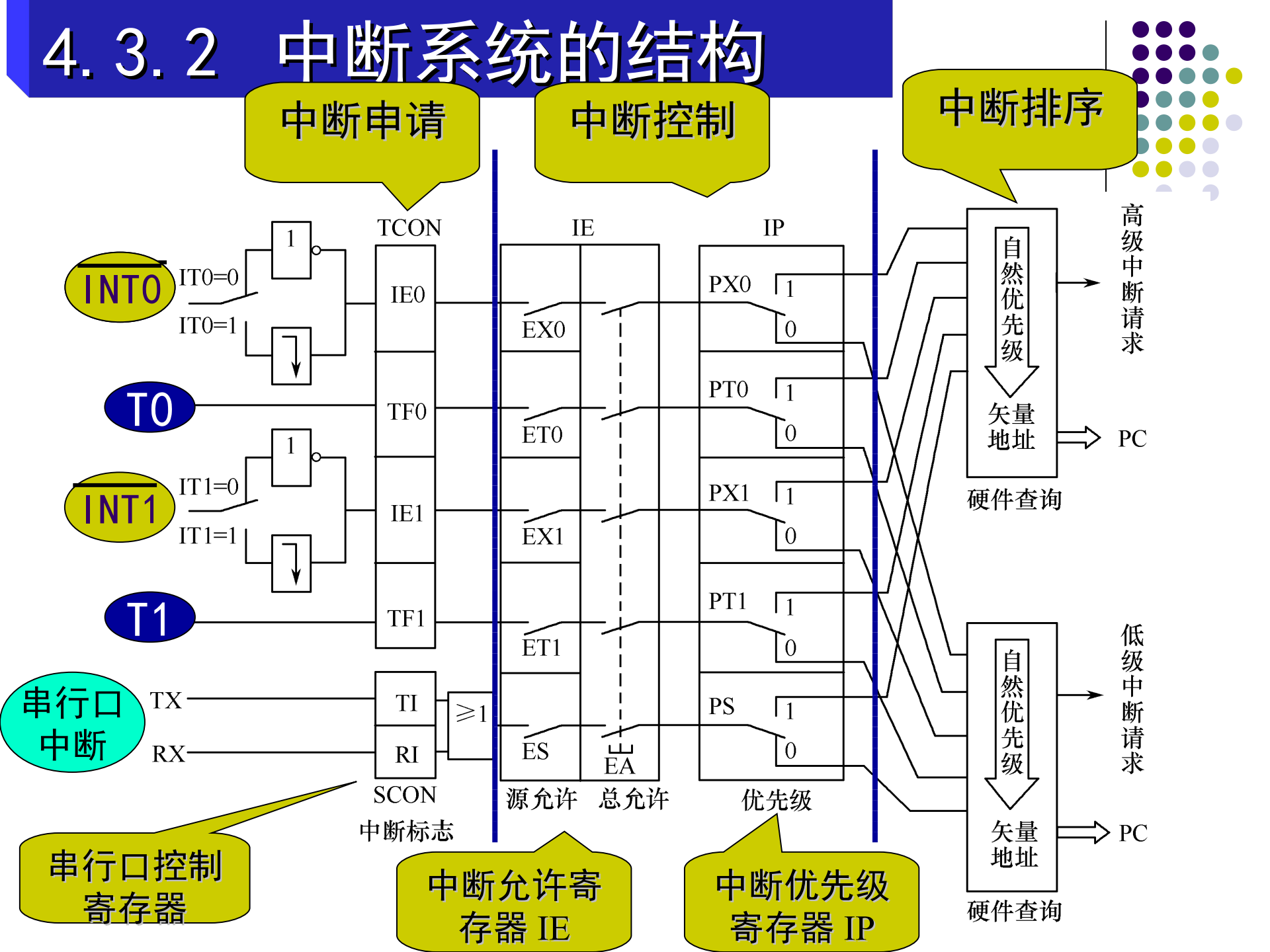
何时调用一般函数在程序中是事先安排好的；而何时调用中断函数事先却无法确定，因为中断的发生时由外部因素决定的，程序中无法事先安排调用语句。

因此，调用中断函数的过程是由硬件自动完成的。

中断特点

同步工作
异常处理
实时处理

4.3.2 中断系统的结构


中断申请
中断控制
中断排序
TCON
IE
IP
INT0
IT0=0
IT0=1
1
IE0
TF0
T0
INT1
IT1=0
IT1=1
IE1
TF1
T1
串行口中断
TX
RX
TI
RI
≥1
SCON
中断标志
EX0
ET0
EX1
ET1
ES
EA
源允许
总允许
PX0
PT0
PX1
PT1
PS
优先级
自然优先级
矢量地址
硬件查询
高级中断请求
低级中断请求
PC
串行口控制寄存器
中断允许寄存器 IE
中断优先级寄存器 IP

序号	中断源		中断引起原因	中断标志位		位名称	说明
1	T1	定时器1中断	定时计数器1计数回零溢出	TF1	T1溢出中断标志	TCON.7	T1被启动计数后，从初值开始加1计数，计满溢出后由硬件置位TF1，同时向CPU发出中断请求，此标志一直保持到CPU响应中断后才由硬件自动清0。也可由软件查询该标志，并由软件清0。前述的定时器编程都是采用查询方式实现。
2	T0	定时器0中断	定时计数器0计数回零溢出	TF0	T0溢出中断标志	TCON.5	T0被启动计数后，从初值开始加1计数，计满溢出后由硬件置位TF0，同时向CPU发出中断请求，此标志一直保持到CPU响应中断后才由硬件自动清0。也可由软件查询该标志，并由软件清0。
3	INT1	外部中断1	P3.3引脚的低电平或下降沿信号	IE1	中断标志	TCON.3	IE1=1，外部中断1向CPU申请中断。
				IT1	*中断触发方式控制位*	*TCON.2*	*当IT1=0，外部中断1控制为电平触发方式；当IT1=1，外部中断1控制为边沿（下降沿）触发方式。*
4	INT0	外部中断0	P3.2引脚的低电平或下降沿信号	IE0	中断标志	TCON.1	IE0=1，外部中断0向CPU申请中断。
				IT0	*中断触发方式控制位*	*TCON.0*	*当IT0=0，外部中断0控制为电平触发方式；当IT0=1，外部中断0控制为边沿（下降沿）触发方式。*
5	TI/RI	串行口中断	串行通信完成一帧数据发送或接收引起中断	TI	串行发送中断标志	SCON.1	CPU将数据写入发送缓冲器SBUF时，启动发送，每发送完一个串行帧，硬件都使TI置位；但CPU响应中断时并不自动清除TI，必须由软件清除。
				RI	串行接收中断标志	SCON.0	当串行口允许接收时，每接收完一个串行帧，硬件都使RI置位；同样，CPU在响应中断时不会自动清除RI，必须由软件清除。

当中断源向 CPU 申请中断时，相应中断标志位由硬件自动置 1，当响应中断请求后，该如何撤除这些中断标志位。

1. 对于 T0， T1 溢出中断和边沿触发的外部中断， CPU 在响应中断后即由硬件自动清除其中断标志位 TF0， TF1 或 IE0， IE1，无须采取其他措施。

2、对于串行口中断， CPU 在响应中断后，硬件不能自动清除中断请求标志位 TI 或 RI，必须在中断服务程序中用软件将其清除。

中断的开放和禁止

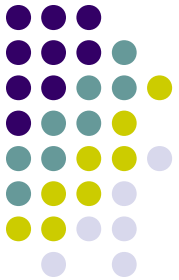
MCS-51 系列单片机的 5 个中断源都是可屏蔽中断，中断系统内部设有一个专用寄存器 IE，用于控制 CPU 对各中断源的开放或屏蔽。 IE 寄存器格式如下：

IE(A8H)

D7	D6	D5	D4	D3	D2	D1	D0
EA	×	×	ES	ET1	EX1	ET0	EX0

中断的开放和禁止

中断允许位		位名称	说明
EA	总中断允许控制位	IE.7	EA = 1，开放所有中断，各中断源的允许和禁止可通过相应的中断允许位单独加以控制； EA = 0，禁止所有中断。
ES	串行口中断允许位	IE.4	ES = 1，允许串行口中断； ES = 0 禁止串行口中断。
ET1	T1 中断允许位	IE.3	ET1 = 1，允许 T1 中断； ET1 = 0，禁止 T1 中断。
EX1	）中断允许位	IE.2	EX1 = 1，允许外部中断 1 中断； EX1 = 0，禁止外部中断 1 中断。
ET0	T0 中断允许位	IE.1	ET0 = 1，允许 T0 中断； ET0 = 0，禁止 T0 中断。
EX0	）中断允许位	IE.0	EX0 = 1，允许外部中断 0 中断； EX0 = 0，禁止外部中断 0 中断。



8051 单片机系统复位后， IE 寄存器中各中断允许位均被清零，即禁止所有中断

例如： EA=1; // 开放中断总允许位

EX0=1; // 开放外部中断 0 允许位

上面的例题也可以用一条语句来实现：

IE=0x81 ； // 寄存器 IE=1000 0001B 同时开放中断总允许位和外部中断 0 允许位。

中断优先级

- MCS-51 系列单片机有两个中断优先级：高优先级和低优先级。
- 每个中断源都可以通过设置中断优先级寄存器 IP 确定为高优先级中断或低优先级中断，实现二级嵌套。同一优先级别的中断源可能不止一个，因此，也需要进行优先权排队。同一优先级别的中断源采用自然优先级。
- 中断优先级寄存器 IP，用于锁存各中断源优先级控制位。 IP 中的每一位均可由软件来置 1 或清 0， 1 表示高优先级， 0 表示低优先级。
- 复位后， IP 低 5 位全清零，默认为低优先级

	D7	D6	D5	D4	D3	D2	D1	D0	
IP	–	–	–	PS	PT1	PX1	PT0	PX0	B8H
位地址	–	–	–	BCH	BBH	BAH	B9H	B8H	

图5-7

在同时收到几个同一优先级的中断请求时，优先响应哪一个中断，取决于内部的查询顺序。查询顺序如下：

中断查询次序

中断源	中断级别
外部中断 0	最高
T0 溢出中断	↓
外部中断 1	
T1 溢出中断	
串行口中断	最低

中断优先级

- 中断优先级寄存器 IP，用于锁存各中断源优先级控制位。 IP 中的每一位均可由软件来置 1 或清 0， 1 表示高优先级， 0 表示低优先级。
- 复位后， IP 低 5 位全清零，默认为低优先级

	D7	D6	D5	D4	D3	D2	D1	D0	
IP	–	–	–	PS	PT1	PX1	PT0	PX0	B8H
位地址	–	–	–	BCH	BBH	BAH	B9H	B8H	

图5-7

中断优先级

中断优先级控制位		位名称	说明
PS	串行口中断优先控制位	IP.4	PS = 1，设定串行口为高优先级中断；PS = 0，设定串行口为低优先级中断。
PT1	定时器 T1 中断优先控制位	IP.3	PT1 = 1，设定定时器 T1 为高优先级中断；PT1 = 0，设定定时器 T1 为低优先级中断。
PX1	外部中断 1 中断优先控制位	IP.2	PX1 = 1，设定外部中断 1 为高优先级中断；PX1 = 0，设定外部中断 1 为低优先级中断。
PT0	T0 中断优先控制位	IP.1	PT0 = 1，设定定时器 T0 为高优先级中断；PT0 = 0，设定定时器 T0 为低优先级中断。
PX0	外部中断 0 中断优先控制位	IP.0	PX0 = 1，设定外部中断 0 为高优先级中断；PX0 = 0，设定外部中断 0 为低优先级中断。

中断处理过程

中断响应

- 中断响应是指 CPU 对中断源中断请求的响应。 CPU 并非任何时刻都能响应中断请求，而是在满足所有中断响应条件、且不存在任何一种中断阻断情况时才会响应。
- CPU 响应中断的条件有：①有中断源发出中断请求；②中断总允许位 EA 置 1；③申请中断的中断源允许位置 1 。
- CPU 响应中断的阻断情况有：① CPU 正在响应同级或更高优先级的中断；②当前指令未执行完；③正在执行中断返回或访问寄存器 IE 和 IP 。

中断处理过程

中断响应过程就是自动调用并执行中断函数的过程。

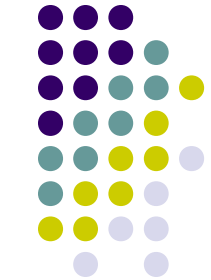
C51 编译器支持在 C 源程序中直接以函数形式编写中断服务程序。常用的中断函数定义语法如下：

```
void  函数名（）  interrupt  n
```

其中 n 为中断类型号， C51 编译器允许 0 ～ 31 个中断， n 取值范围 0 ～ 31 。下面给出了 8051 控制器所提供的 5 个中断源所对应的中断类型号和中断服务程序入口地址：

中断源	**n**	**入口地址**
外部中断 0	0	0003H
定时 / 计数器 0	1	000BH
外部中断 1	2	0013H
定时 / 计数器 1	3	001BH
串行口	4	0023H

小提示



1. 不能进行参数传递；
2. 无返回值；
3. 在任何情况下，不能调用中断函数；
4. 可以再中断函数定义中使用 using 指令指定当前使用的寄存器组，格式如下：

   void　函数名（）　interrupt　n　[using m]
5. 在中断函数中调用的函数所使用的寄存器组必须与中断函数相同；

中断处理过程

中断响应时间

中断响应时间是指从中断请求标志位置位到 CPU 开始执行中断服务程序的第一条语句所需要的时间。

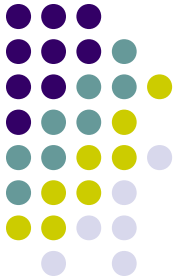
1 ）中断请求不被阻断的情况

外部中断响应时间至少需要 3 个机器周期，这是最短的中断响应时间。一般来说，若系统中只有一个中断源，则中断响应时间为 3 ～ 8 个机器周期。

2 ）中断请求被阻断的情况

如果系统不满足所有中断响应条件、或者存在任何一种中断阻断情况，那么中断请求将被阻断，中断响应时间将会延长。

任务 10　模拟交通灯控制



任务要求：

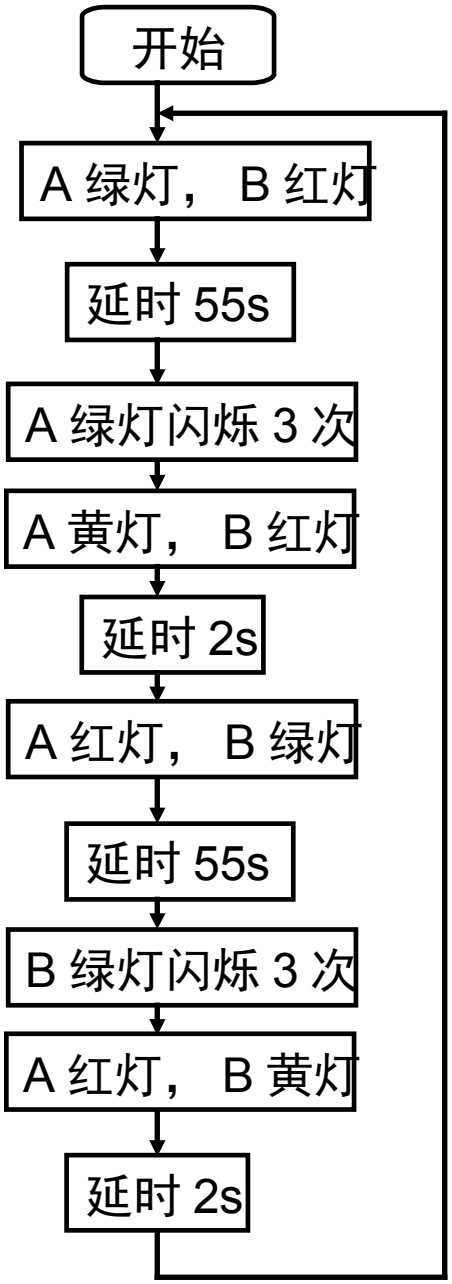
① 正常情况下双方向轮流点亮交通灯，交通灯的状态如表 4.8 所示；

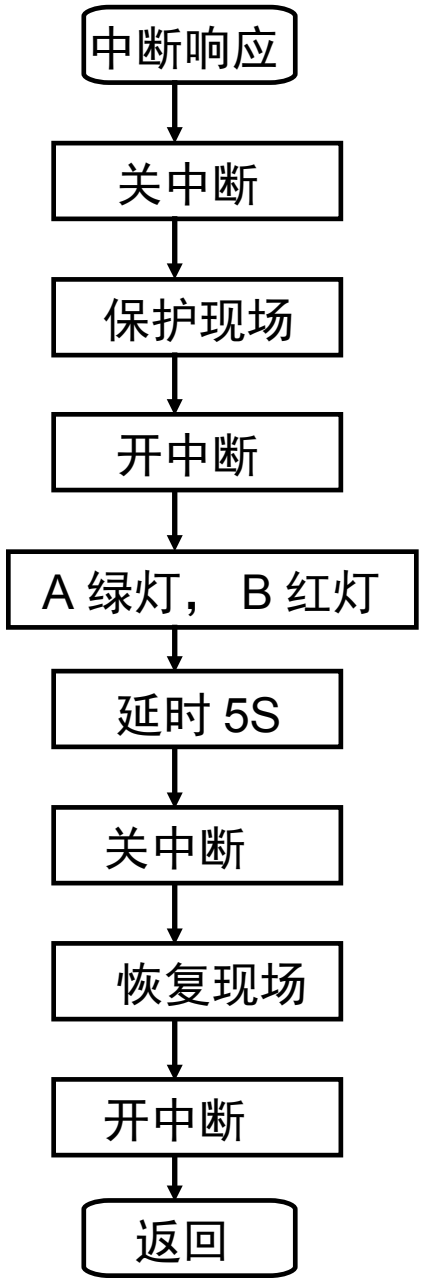
② 特殊情况时， A 道放行；

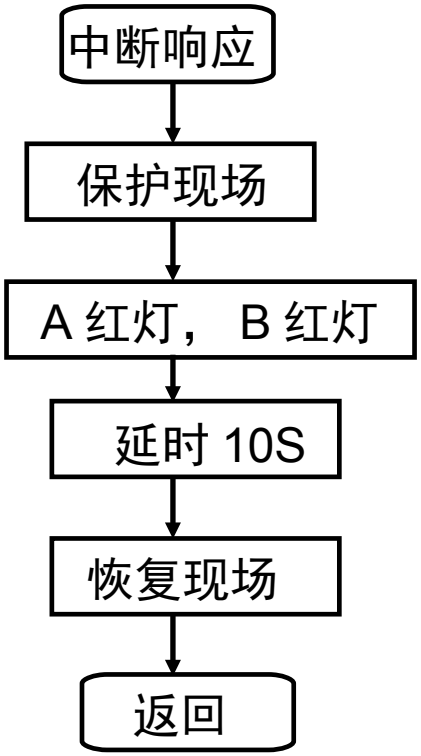
③ 有紧急车辆通过时， A 、 B 道均为红灯。紧急情况优先级高于特殊情况。

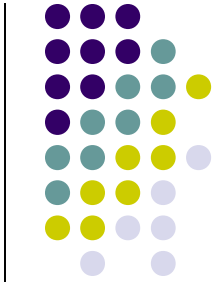
东西方向（简称 A 方向）			南北方向（简称 B 方向）			状态说明
红 灯	黄 灯	绿 灯	红 灯	黄 灯	绿 灯	
灭	灭	亮	亮	灭	灭	A 方向通行，B 方向禁行
灭	灭	闪烁	亮	灭	灭	A 方向警告，B 方向禁行
灭	亮	灭	亮	灭	灭	A 方向警告，B 方向禁行
亮	灭	灭	灭	灭	亮	A 方向禁行，B 方向通行
亮	灭	灭	灭	灭	闪烁	A 方向禁行，B 方向警告
亮	灭	灭	灭	亮	灭	A 方向禁行，B 方向警告

P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	P1 端口数据	状态说明
A 红灯	A 黄灯	A 绿 灯	B 红灯	B 黄灯	B 绿灯		
1	1	0	0	1	1	F3H	状态 1：A 通行，B 禁行
1	1	0,1 交替变换	0	1	1		状态 2：A 绿灯闪，B 禁行
1	0	1	0	1	1	EBH	状态 3：A 警告，B 禁行
0	1	1	1	1	0	DEH	状态 4：A 禁行，B 通行
0	1	1	1	1	0,1 交替变换		状态 5：A 禁行，B 绿灯闪
0	1	1	1	0	1	DDH	状态 6：A 禁行，B 警告


开始
A 绿灯， B 红灯
延时 55s
A 绿灯闪烁 3 次
A 黄灯， B 红灯
延时 2s
A 红灯， B 绿灯
延时 55s
B 绿灯闪烁 3 次
A 红灯， B 黄灯
延时 2s

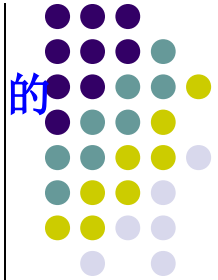

中断响应
关中断
保护现场
开中断
A 绿灯， B 红灯
延时 5S
关中断
恢复现场
开中断
返回


中断响应
保护现场
A 红灯， B 红灯
延时 10S
恢复现场
返回

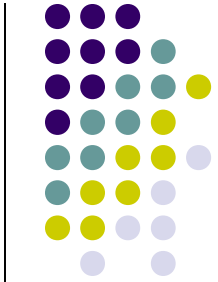


```
// 功能：交通灯控制程序
#include <REG51.H>
unsigned char t0,t1;                          // 定义全局变量，用来保存延时时间循环次数
// 函数名： delay0_5s1
// 函数功能：用 T1 的方式 1 编制 0.5 秒延时程序，假定系统采用 12MHz 晶振，定
//  时器 1 、工作方式 1 定时 50ms ，再循环 10 次即可定时到 0.5 秒
// 形式参数：无
// 返回值：无
void delay0_5s1()
{
  for(t0=0;t0<0x0a;t0++)    // 采用全局变量 t0 作为循环控制变量
  {
    TH1=0x3c;               // 设置定时器初值
    TL1=0xb0;
    TR1=1;                  // 启动 T1
    while(!TF1);            // 查询计数是否溢出，即定时 50ms 时间到， TF1=1
    TF1=0;                  // 50ms 定时时间到，将定时器溢出标志位 TF1 清零
  }
}
// 函数名： delay_t1
// 函数功能：实现 0.5 秒～ 128 秒延时
// 形式参数： unsigned char t;
//         延时时间为 0.5 秒 ×t
// 返回值：无
void delay_t1(unsigned char t)
{
  for(t1=0;t1<t;t1++)                         // 采用全局变量 t0 作为循环控制变量
    delay0_5s1();
}
```

```
// 函数： int_0
// 函数功能：外部中断 0 中断函数，紧急情况处理，当 CPU 响应外部中断 0 的
// 中断请求时，自动执行该函数，实现两个方向红灯同时亮 10 秒
// 形式参数：无
// 返回值：无
void int_0() interrupt 0          // 紧急情况中断
{
   unsigned char i,j,k,l,m;
   i=P1;          // 保护现场，暂存 P1 口、 t0 、 t1 、 TH1 、 TH0
   j=t0;
   k=t1;
   l=TH1;
   m=TH0;
   P1=0xdb;                       // 两个方向都是红灯
   delay_t1(20);                  // 延时 10 秒
   P1=i;          // 恢复现场，恢复进入中断前 P1 口、 t0 、 t1 、 TH1 、 TH0
   t0=j;
   t1=k;
   TH1=l;
   TH0=m;
}
```



```
// 函数： int_1
// 函数功能：外部中断 1 中断函数，特殊情况处理，当 CPU 响应外部中断 1 的中断请求时，自动执行该函数，实现 A 道放行 5 秒
// 形式参数：无
// 返回值：无
void int_1() interrupt 2  // 特殊情况中断
{
   unsigned char i,j,k,l,m;
   EA=0;                                  // 关中断
   i=P1;                                  // 保护现场，暂存 P1 口、 t0 、 t1 、 TH1 、 TH0
   j=t0;
   k=t1;
   l=TH1;
   m=TH0;
   EA=1;                                  // 开中断
   P1=0xf3;                               // A 道放行
   delay_t1(10);                          // 延时 5 秒
   EA=0;                                  // 关中断
   P1=i;          // 恢复现场，恢复进入中断前 P1 口、 t0 、 t1 、 TH1 、 TH0
   t0=j;
   t1=k;
   TH1=l;
   TH0=m;
   EA=1;                                  // 开中断
}
```



```
void main()                                  // 主函数
{
  unsigned char k;
  TMOD=0x10;                                 // T1 工作在方式 1
  EA=1;                                      // 开放总中断允许位
  EX0=1;                                     // 开外部中断 0 中断允许位
  IT0=1;                                     // 设置外部中断 0 为下降沿触发
  EX1=1;                                     // 开外部中断 1 中断允许位
  IT1=1;                                     // 设置外部中断 1 为下降沿触发
  while(1) {
    P1=0xf3;                                 // A 绿灯， B 红灯，延时 5 秒
    delay_t1(10);
    for(k=0;k<3;k++){                        // A 绿灯闪烁 3 次
    P1=0xf3;
    delay0_5s1();                // 延时 0.5 秒
    P1=0xfb;
    delay0_5s1();                // 延时 0.5 秒
        }
  P1=0xeb;                                   // A 黄灯， B 红灯，延时 2 秒
  delay_t1(4);
  P1=0xde;                                   // A 红灯， B 绿灯，延时 5 秒
  delay_t1(10);
  for(k=0;k<3;k++)               // B 绿灯闪烁 3 次
  {
    P1=0xde;
    delay0_5s1();                // 延时 0.5 秒
    P1=0xdf;
    delay0_5s1();                // 延时 0.5 秒
  }
  P1=0xdd;                                   // A 红灯， B 黄灯，延时 2 秒
  delay_t1(4);
      }
}
```