文轩科技

单片机那些事儿

中级篇——中断

残弈悟恩（刘平）

2014

官方淘宝店铺：http://shop109195762.taobao.com

郑重声明

《单片机那些事儿》以残弈悟恩亲自开发的 MGMC-V1.0 实验板为硬件平台，以残弈悟恩亲自编著的《深入浅出玩转 51 单片机》为辅助教程，以残弈悟恩亲自录制的《31 天环游单片机》为基础视频。

本资料以个人学习和工作的经验为素材，以单片机初学者、单片机项目开发者为对象，教大家如何走进单片机世界、如何达到开发工程项目为目的。限于时间和水平关系，资料中难免有过失之处，望各位高手批评指教，多多拍砖，拍累了，你们休息，我继续上路。

现已连载的方式免费共享于各大电子网站，供单片机新手们参考学习，可以自由下载传阅，但未经残弈悟恩允许，不得用于任何商业目的，若经发现，残弈悟恩将以愚公移山的精神追究到底。

有商家已经开始以此资料作为商业用途了，望大家注意，我就先来加个水印哦。

拍我、批我，找我聊人生、聊感情、聊技术的联系方式，一一公布如下：

- 助学活动

  EDA 助学小组：http://group.ednchina.com/GROUP_GRO_14273_3000002320.HTM

  ATE 助学小组：http//group.chinaaet.com/322

  Elecfans 助学活动：http://bbs.elecfans.com/jishu_444860_1_1.html

  活动更新详见论坛：www.ieeBase.net

- 31 天环游单片机视频

在线观看地址：http://study.chinaaet.com/course/6100000018

百度下载地址：http://pan.baidu.com/share/home?uk=2417018124#category/type=0

- 残弈悟恩【AET（网络名师）、EDN（博客专家）、Elecfans（社区之星）】

奋斗历程：http://bbs.elecfans.com/jishu_447902_1_1.html

博客：http://bbs.ednchina.com/BLOG_%E6%AE%8B%E5%BC%88%E6%82%9F%E6%81%A9_2001612.HTM?source=ednc_topnav

- 联系方式

个人邮箱：xymbmcu@163.com

单片机交流群：143406243

AD14/PADS9.5 交流群：10821147

版本：20140912（V1.0）　　制造者：残弈悟恩

让爱充满大地——花 1 秒时间，拯救 1 个人，传递 1 份爱

声明：只是残弈悟恩爱心的喷发，我得不到一分钱，各位不要多想，谢谢！

你知道吗？在非洲北边的某个地区，每一秒都有许许多多的人正在挨饿，每一天至少有一位儿童死于营养不足。你的一次点击就能让某位穷人得到 1.1 杯食物。当然你可以不相信有这样的链接或者是骗点击什么的。事实上，网站确实是帮穷人得 1.1 杯食物的，只要你点进去单机一下中间的黄色按钮，就会出来一系列介绍各种商品的网页（绝对免费的并且不会下载任何软件，也不会有电脑病毒），同时也会有人因为您的一次点击而得到 1.1 杯食物，食物是由商家提拱的，但爱心却是您献出的。如果你觉得残弈悟恩在忽悠大家，你不妨可以在网上查一下是真与假。

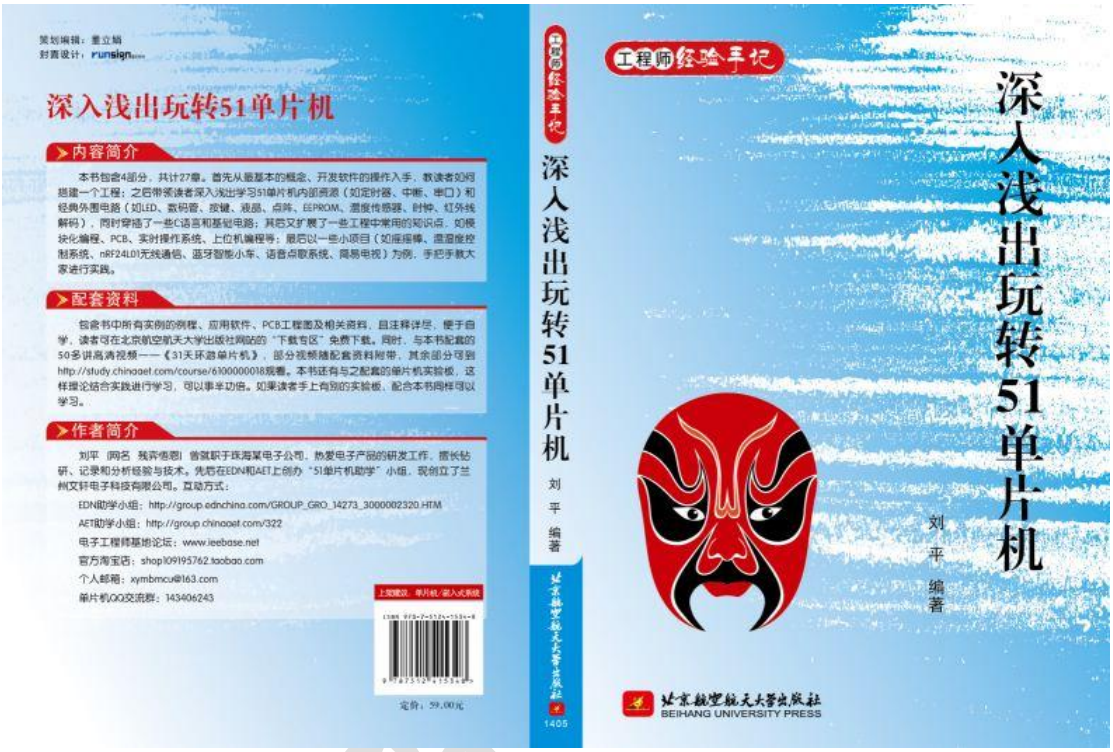
看到这本资料的朋友多数都是电子爱好者、单片机初学者，或者干电子这一行的，管你穷学生还是穷工人，只要能上网，只要愿花一秒种就可以了。人生在世，有两件事不能等：一、孝顺；二、行善。无论你是 LED 小灯、普通灯泡也好，还是荧光灯也吧，最重要就是要懂得用自身的光去照耀别人，光的强度并不重要。

点击链接：

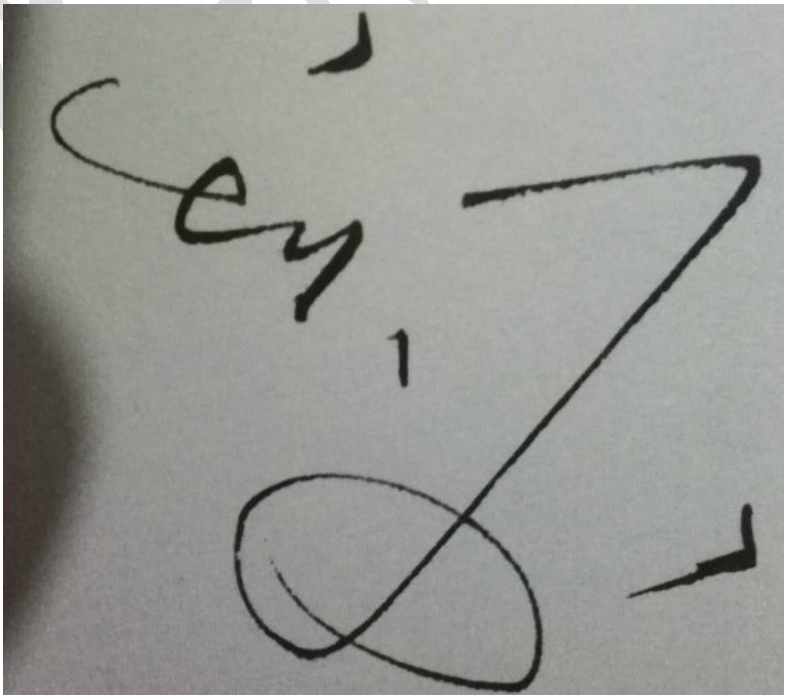
http://www.thehungersite.com/clickToGive/home.faces?siteId=1&link=ctg_ths_home_from_ths_thankyou_sitenav

特别说明

该资料是《深入浅出玩转 51 单片机》一书的“裁剪版、凌乱版”，完整版、整齐版请见由北京航天航空大学出版社于 2014 年 5 月权威出版的书籍——《深入浅出玩转 51 单片机》，这里便于大家直观了解，先贴封面一张，当当、卓越、京东、淘宝等网站随便一搜，就能找到此书的购买链接，这里不赘。



如果想要我亲笔签名的（字不好看，给您眼睛带来的创伤，望您海涵），可以点击签名图片进入购买链接地址。



第九章　中断

上一章已经说了，中断很重要，睁大眼睛、静下心来好好看，看完若还不会，我可管不着。

9.1 运算放大器

运算放大器，简称“运放”，英文描述为 Operation Amplifier（OP），是一种运用很广泛的线性集成电路，其种类繁多，在运用方面不但可对微弱信号进行放大，还可作为反相器、电压比较器、电压跟随器、积分器、微分器等，并可对信号做加、减运算，所以被称之为运算放大器。其符合表示如图 9-1 所示（左：国家标准规定的符合；右：国内外常用符合）。

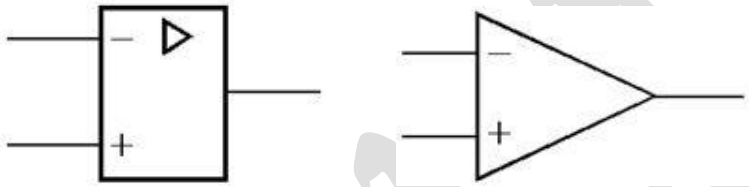


图 9-1 运算放大器的代表符合

9.1.1 负反馈

说到运放，其实有好多特性和参数，限于篇幅，就不一一列举了。这里有一很重要的概念—负反馈。

关于负反馈，残弈悟恩也给不出什么严格的定义。这里结合电路图来说明什么是负反馈，引入负反馈有何意义？

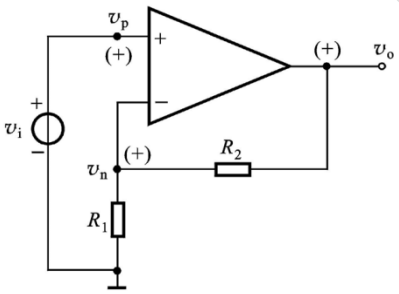


图 9-2 同相放大电路

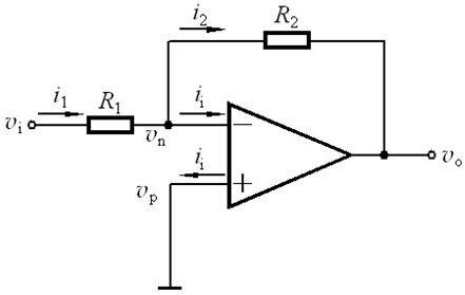


图 9-3 反相放大电路

电路如图 9-2 所示，输入信号电压 V_i（$=V_p$）加到运放的同相输入端“+”和地之间，输出电压 V_o 通过 R_1 和 R_2 的分压作用，得 $V_n = V_f = R_1V_o/(R_1 + R_2)$，作用于反相输入端“-”，所以 V_f 在此称为反馈电压。

当输入信号电压 V_i 的瞬时电位变化极性如图中的（+）号所示，由于输入信号电压 V_i（V_p）加到同相端，输出电压 V_o 的极性与 V_i 相同。反相输入端的电压 V_n 为反馈电压，其极性亦为（+），而静输入电压 $V_{id} = V_i - V_f = V_p - V_n$ 比无反馈时减小了，即 V_n 抵消了 V_i 的一部分，使放大电路的输出电压 V_o 减小了，因而这时引入的反馈时负反馈。

综上，负反馈作用是利用输出电压 V_o 通过反馈元件（R_1、R_2）对放大电路起自动调节作用，从而牵制了 V_o 的变化，最后达到输出稳定平衡。

9.1.2 同相放大电路

提供正电压增益的运算放大电路称之为同相放大，如图 9-2 所示。

在图 9-2 中，输出通过负反馈的作用，使 V_n 自动地跟踪 V_p，使 $V_p \approx V_n$，或 $V_{id} = V_p - V_n \approx 0$。这种现象称为虚假短路，简称**虚短**。

由于运放的输入电阻的阻值又很高，所以，运放两输入端的 $I_p = -I_n = (V_p - V_n) / R_i \approx 0$，这种现象称为虚断。注意：**虚短**是本质的，而**虚断**则是派生的。

9.1.3 反相放大电路

提供负电压增益的运算放大电路称之为反相放大，如图 9-3 所示。

图 9-3 中，输入电压 V_i 通过 R_1 作用于运放的反相端，R_2 跨接在运放的输出端和反相端之间，同相端接地。由虚短的概念可知，$V_n \approx V_p = 0$，因此反相输入端的电位接近于地电位，，故称**虚地**。虚地的存在是反相放大电路在闭环工作状态下的重要特征。

9.2 小秘图示定时器

假如你正在接电话或者打电话，突然有人打来了一个，当这个人是你的男朋友或者女朋友或者 BOSS 时，我想这个电话你不会不接吧，不接你小样可就摊上事了。此时你也许会说，噢！不好意思，咱们先聊到这里，我这儿有个紧要的事，等我处理完后给你回过去接着我们的话题继续聊，赶紧挂了电话处理领导电话后回来继续和前边的人聊。我想这个过程中实际上已近发生了一次中断。如果你不会单片机中断和定时器，就像小鸟不会飞那样，可想而知中断的重要了吧。下面我们来玩玩单片机的中断，你不仅要会玩，而且还要玩精，这样你才能达到高手的境界。

9.3 中断的一点点东西

9.3.1 原理说明

对于单片机来讲，在程序的执行过程中，由于某种外界的原因，必须终止当前执行的程序，而去执行相应的处理程序，待处理结束后，再回来继续执行被终止的程序，这个过程叫中断。对于单片机来说，突发的事情实在太多了。例如用户通过按键给单片机输入数据时，这对单片机本身来说是无法估计的事情，这些外部来的突发信号，一般就由单片机的外部中断来处理。外部中断其实就是一个管脚的状态改变所引起的。

其流程图如下图 9-4 所示。

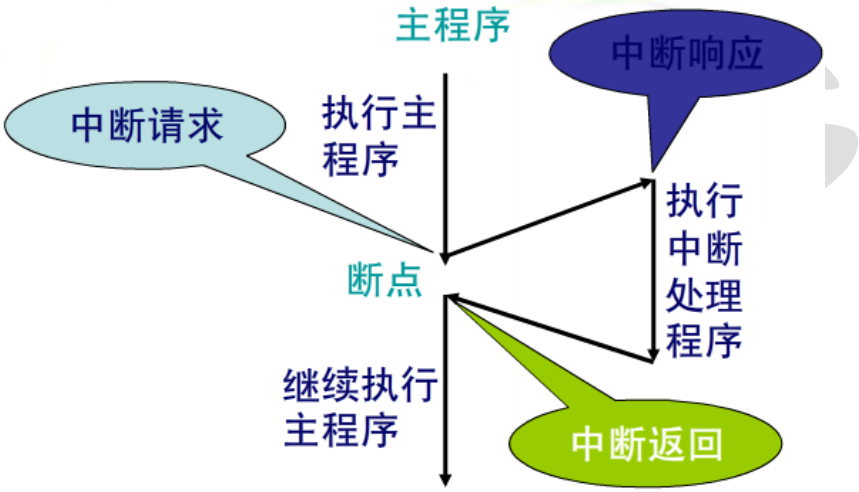


图 9-4 中断流程示意图

52 单片机有 6 个中断源，具有二个中断优先级，可实现二级中断服务程序的嵌套。每个中断源均可软件编程为高优先级或低优先级中断，允许或禁止向 CPU 请求中断。例如你现在正接着普通朋友的电话，这时你女朋友来电话了，之后你会向你普通朋友说，我们就聊到这儿吧，之后你会去接你女朋友的电话，在通话中，你的 BOSS 又来电话了，这时你就会考虑那个的电话重要，这个你考虑重要的过程就是所谓的—中断优先级，那若单片机同时有两个中断产生，单片机又是怎么执行的呢？着就取决于单片机内部的一个特殊功能寄存器（中断优先级寄存器）的设置，通过设置它，就相当于告诉单片机那个优先级高，那个优先级低，若不操作，就是按单片机默认的设置来执行（单片机自己有一套默认的优先级）。

52 单片机的 6 个中断源我们简述如下：

INT0—外部中断 0，引入端口：P3.2，触发方式：低电平、下降沿。

INT1—外部中断 1，引入端口：P3.3，触发方式：低电平、下降沿。

T0—定时器/计数器 0 中断，触发方式：T0 计数器记满归零。

T1—定时器/计数器 1 中断，触发方式：T1 计数器记满归零。

T2—定时器/计数器 2 中断，触发方式：T2 计数器记满归零。

TI/RI—串口中断，触发方式：串口完成一帧字符发送或接收完。

上面就是 52 单片机的 6 个中断源，接下来看看单片机默认的优先级、C 语言入口序号、

中断向量地址，具体如表 9-1 所示（此表可记可不记，但是必须会查）。

表 9-1 52 单片机中断源及优先级顺序

中断源	中断向量地址（汇编用）	优先级	序号（C 语言用）	中断请求标志位	中断允许控制位
INT0-外部中断 0	0003H	1(最高)	0	IE0	EX0/EA
定时器 0	000BH	2	1	TF0	ET0/EA
INT1-外部中断 1	0013H	3	2	IE1	EX1/EA
定时器 1	001BH	4	3	TF1	ET1/EA
串口中断	0023H	5	4	RI/TI	--
定时器 2	002BH	6（最低）	5	TF2	ET2/EA

在使用单片机中断的过程中，首先需要设置两个与中断有关的寄存器，两个寄存器简述如下。

① IE—中断允许寄存器

所谓的中断允许寄存器就是控制各个中断是开是关，要使用哪个中断，就必须将其对应位置 1，也即允许该中断。要上大学，首先得大学校门为你敞开吧（意味着你要考上，等价于学校允许你进入这个学校了）。IE 在特殊功能寄存器中，字节地址为：A8H，位地址分别是：AFH~A8H(由高到低)，由于该字节地址（A8）能被 8 整除（单片机中能被 8 整除的地址都可以位寻址），因而该地址可以位寻址，即可对该寄存器的每一位进行单独操作。IE 复位值为：0x00,各个位定义如表 9-2 所示。

表 9-2 IE:中断允许寄存器

Bit	D7	D6	D5	D4	D3	D2	D1	D0
Name	EA	--	ET2	ES	ET1	EX1	ET0	EX0
Address	AFH	--	ADH	ACH	ABH	AAH	A9H	A8H

EA:CPU 的总中断允许控制位，EA = 1，CPU 开放总中断；EA = 0，CPU 屏蔽所有中断申请。

对于新手们，总是不能理解，其实可以举个这样的例子：EA 就是校长，ET2、ES、ET1、EX1、ET0、EX0 分别是六个班主任，要给学生放假，首先校长同意，再班主任同意，这才能放假，若校长不放，班主任为了混饭肯定不敢放，若校长同意了，班主任不放，学生怕屁股疼，还是自己不敢给自己放假。

ET2：定时/计数器 T2 的溢出中断允许位。

ET2 = 1，允许 T2 中断；ET2 = 0，禁止 T2 中断。

ES:串行口中断允许位。

ES = 1，允许串行口中断；ES = 0，禁止串行口中断。

ET1：定时/计数器 T1 的溢出中断允许位。

ET1 = 1，允许 T1 中断；ET1 = 0，禁止 T1 中断。

EX1：外部中断 1 允许位。

EX1 = 1，允许外部中断 1 中断；EX1 = 0，禁止外部中断 1 中断。

ET0：定时/计数器 T0 的溢出中断允许位。

ET0 = 1，允许 T0 中断；ET0 = 0，禁止 T0 中断。

EX0：外部中断 0 允许位。

EX0 = 1，允许外部中断 0 中断；EX0 = 0，禁止外部中断 0 中断。

② IP--中断优先级寄存器

前面说过 52 单片机具有两个中断优先级，即高级优先级和低级优先级，可以实现两级中断嵌套。中断优先级在特殊功能寄存器中，可以通过设置实现各个中断属于中断的哪一级，该寄存器字节地址为：B8H，也能位寻址，即可对每一位单独操作。IP 复位值：0x00，各个位定义如表 9-3 所示。

表 9-3 IP:中断优先级寄存器

Bit	D7	D6	D5	D4	D3	D2	D1	D0
Name	--	--	--	PS	RT1	PX1	PT0	PX0
Address	--	--	--	BCH	BBH	BAH	B9H	B8H

PS:串行口中断优先级控制位。

PS = 1，串行口中断为高优先级中断；

PS = 0，串行口中断为低优先级中断。

PT1：定时/计数器 T1 中断优先级控制位。

PT1 = 1，T1 中断定义为高优先级中断；

PT1 = 0，T1 中断定义为低优先级中断。

PX1：外部中断 1 中断优先级控制位。

PX1 = 1，外部中断 1 中断定义为高优先级中断；

PX1 = 0，外部中断 1 中断定义为低优先级中断。

PT0：定时/计数器 T0 的溢出中断允许位。

PT0 = 1，T0 中断定义为高优先级中断；

PT0 = 0，T0 中断定义为低优先级中断。

看完这两个寄存器之后，网友们是不是又累了、困了，其实残弈悟恩比你们更累，大家都要坚持住，坚持就是胜利。两个表都可以不用背，但是一定要会查。

8.3.2 硬件设计

关于硬件，中断也是单片机内一个看不见、摸不着的东西，再说了这个硬件设计由单片机生产厂商来完成，读者们就不需要知道了，会用就 OK 了。顺便为大家贴一张中断示意图（见图 9-5），是不是很高端、大气、上档次啊。相信大家的实力，所以这里就不讲述了，自行理解哈。

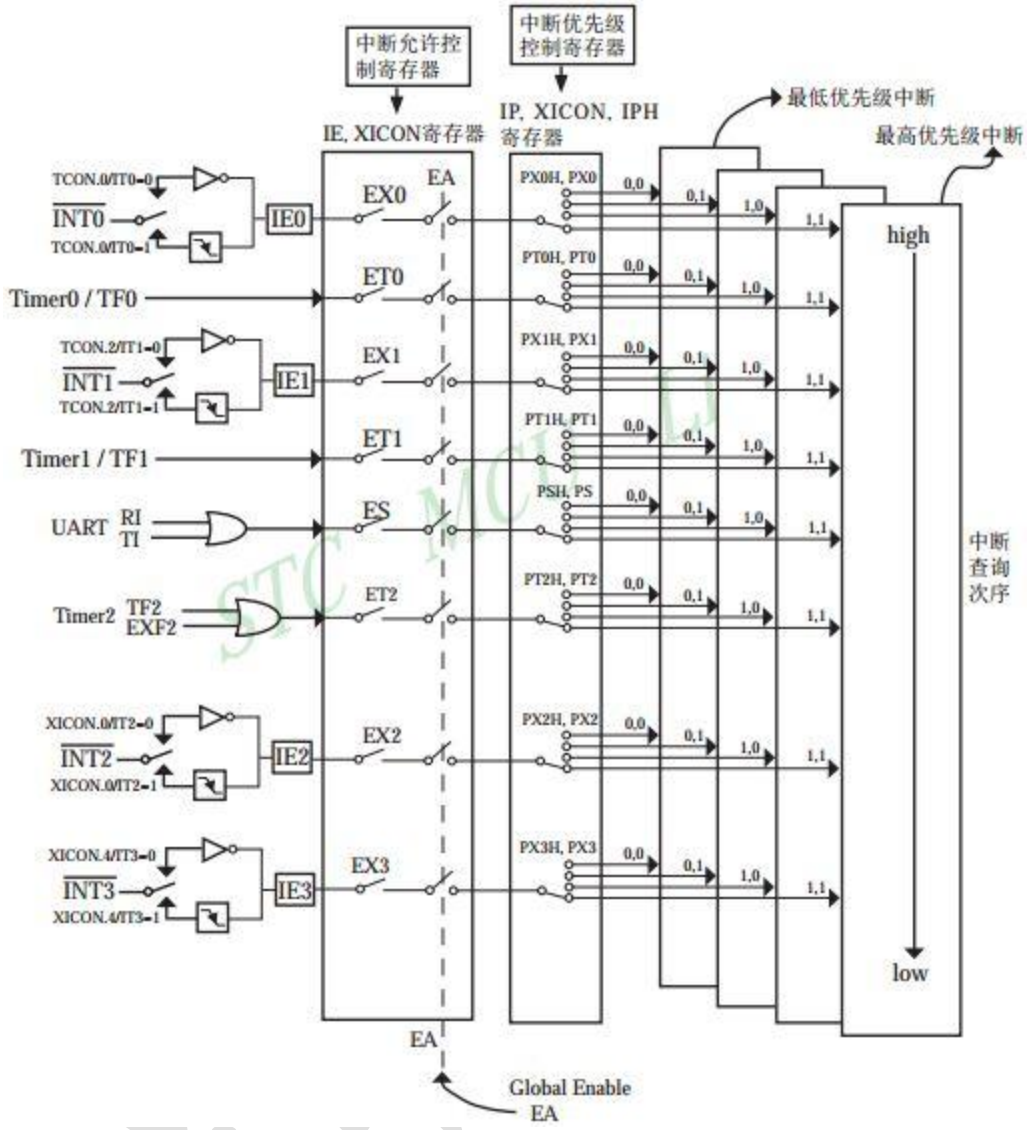


图 9-5 中断内部结构图

9.3.3 软件分析

在这里只是讲述了中断如何应用，并没有详细讲解整个中断的执行过程，其实中断的执行过程还是比较复杂，因而就摒弃没讲，若大家感兴趣，可以从汇编的角度入手，学习单片机的中断执行过程，残弈悟恩觉得那将是一次真正的挑战。

最后再说说中断服务程序的写法。

```
void 函数名(void) interrupt 中断号 (using 工作组)
{
    中断服务程序内容
}
```

关于中断函数说明如下几点，望大家采纳并铭记。

1) 中断函数无返回值，所以前面为 void。

2) 中断函数命名随便，但一定不能是 C 语言的关键字，如 if、case 等，残弈悟恩建议大家用：Timer0_ISR、EX0_ISR 等。

3) 中断函数不带任何参数，所以小括号内写了 void。

4) 中断函数的关键词 interrupt，一定要写且要正确无误。

5) 中断号见表 1 第四列，必须一一对应，不能错，错了伤不起。

后面的 using 工作组（），通常不写，具体依个人习惯，OK，中断就“忽悠”这么多，接下来看实例。

9.4 例说中断

借着实例，踏上神圣的中断之路，有没有信心？

实例 1“鬼火”灯

还记得上一张的定时器吗？还记得那几个说明吗？还记得残弈悟恩设计的“鬼火”灯吗？这节我们再用定时器+中断的方式来在 MGMC-V1.0 实验板重新点燃这盏明灯吧！

- 实验目的：如何通过中断的方式实现定时的目的
- 实验器材：电脑、MGMC-V1.0 开发板、Keil4、STC-ISP 软件
- 实验原理：前面有所讲述
- 实验步骤：

一、使用 Keil4 编译软件，新建工程，编写程序，并生成 hex 文件。

```
#include <reg52.h>
/* ******************************************************* */
// 宏定义
/* ******************************************************* */
#define uInt16 unsigned int
/* ******************************************************* */
// 函数名称：Timer0Init()
// 函数功能：定时器 0 初始化设置
// 入口参数：无
// 出口参数：无
/* ******************************************************* */
void Timer0Init(void)
{
    TMOD = 0x01; // 设置定时器 0 工作在模式 1 下
    TH0 = 0xDC;
    TL0 = 0x00;    // 赋初始值
    EA = 1;        // 开总中断（校长发言）
    ET0 = 1;       // 开定时器 0 中断（班主任说话）
    TR0 = 1;       // 开定时器 0
}
```

```
21. /* ******************************************************* */
22. // 函数名称：main()
23. // 函数功能：定时器初始化后进入死循环，等定时器中断
24. // 入口参数：无
25. // 出口参数：无
26. /* ******************************************************* */
27. void main(void)
28. {
29.     Timer0Init();
30.     while(1);
31. }
32. /* ******************************************************* */
33. // 函数名称：Timerj0_ISR()
34. // 函数功能：定时器中断服务，控制 LED 闪烁
35. // 入口参数：无
36. // 出口参数：无
37. /* ******************************************************* */
38. void Timer0_ISR(void) interrupt 1
39. {
40.     static uInt16 uiCounter = 0;        //只做一次定义
41.     TH0 = 0xDC;
42.     TL0 = 0x00;                         //定时器重新赋初值
43.     uiCounter++;                        //记录中断次数
44.     if(100 == uiCounter)
45.     {
46.         uiCounter = 0;
47.         P2 = ~ P2;
48.     }
49. }
```

二、使用 ISP 烧录程序，将 hex 文件烧录到 MGMC-V1.0 开发板中。

三、观察实验现象，你的“鬼火”灯能否吓唬猎人呢？

- 实验总结：

依旧分析上述程序，该程序有详细的注释，只简单提几句。17、18 行，要产生中断必须中断的各个控制位要使能，还记得要放假的事吗？校长、班主任都要同意吧；30 行，程序停止在这里，等待中断产生；38 行，中断函数比较特殊，即使写在主函数后面也不需要声明，并且不需要调用，但是格式必须遵循规定，无规矩是成不了方圆滴；40 行，局部变量，定义时要赋初值，可这个局部变量不简单，前面有个 “static”，意思是中断函数每过 1ms 就会执行一次，如果不定义成静态存储变量的形式，那么程序每过 1ms 进来就会对该变量赋一次初值 0，这样程序无论如何执行 43 行的“++”，那也永远加不到 100，当然此时也可以将其定义为全局变量，但有一点必须铭记，能不用全局变量时就最好不要用全局变量，那么两全其美的做法就是将其定义为一个静态的局部变量，这样，当每次中断执行到 43 行“++”时，就会在原来的基础之上加 1，这样就满足要求了。

实例 2 四位计数器伴随 8 盏“鬼火”灯

还记得数码管的动态显示原理吗？若不记得了，笔者就再带领读者们复习一下，前面已经提到过，数码管的动态显示，实质是利用人的视觉暂留效应来产生的。例如有 4 个数码管，要显示“0123”，显示过程是，先选中第一个数码管，让其显示“0”；再选中第二个让其显示“1”；接着选中第三个让其显示“2”；最后选中第四个数码管，让其显示“3”；只要这四个的显示间隔足够短就可以达到动态的显示效果。那足够短，究竟多少是足够短呢？也就是说完成一次全部数码管扫描的时间是多少？

答案是：10ms 以内。记得以前有句流行的广告语是这么说的：100Hz 无闪烁。什么意思，就是只要频率大于 100Hz，也即周期小于 10ms 就没有闪烁，这就是动态扫描的硬性指标。记得培训时有人问笔者，那再快点行不行，完全行，只是再快就没意义了，因为刷新速度越快，CPU 运行的负荷就越重，继而增加了单片机的功耗，这对构建节约型社会来说，是很不好的，^_^。

接下来就以一个实例来说明如何用定时器来刷新数码管，又如何用另一个定时器来实现 8 个 LED 小灯的闪烁。最后实现 4 位数码管做类似于秒表的功能，也即数码管上的数据每隔一秒加一，依次从“0000”显示到“9999”，再从“0000”继续开始，同时 8 个灯每隔 1S 闪烁一次。最后例程代码如下：

```
#include <reg52.h>
#define uInt16 unsigned int
#define uChar8 unsigned char
uChar8 code Disp_Tab[] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};
sbit SEG_SELECT = P1^7;
sbit BIT_SELECT = P1^6;
uInt16 g_uiNum = 0;                    //全局变量，存放秒的计数值
uChar8 g_ucRefresh = 0;                //用于刷新数码管
void TimerInit(void)
{
    TMOD = 0x11;        // 设置定时器 0、1 工作在模式 1 下
    TH0 = 0xFC;         // 为定时器 0 赋初始值
    TL0 = 0x66;         // 定时 1ms
    TH1 = 0x4C;         // 为定时器 1 赋初始值
    TL1 = 0x00;         // 定时 50ms
    EA = 1;             // 打开总中断
    ET0 = 1;            // 开定时器 0 的中断
    TR0 = 1;            // 启动定时器 0
    ET1 = 1;            // 打开定时器 1 的中断
    TR1 = 1;            // 启动定时器 1
}
void main(void)
{
    uChar8 QianNum,BaiNum,GeNum,ShiNum;
```

```
25.     TimerInit();
26.     while(1)
27.     {
28.         GeNum   = g_uiNum % 10;         /* 利用“/”和“%”来分离位 */
29.         ShiNum   = g_uiNum / 10 % 10;
30.         BaiNum   = g_uiNum / 100 % 10;
31.         QianNum = g_uiNum / 1000;
32.         switch(g_ucRefresh)
33.         {
34.             case 0:
35.                 BIT_SELECT = 1;P0 = 0xef;BIT_SELECT = 0;
36.                 SEG_SELECT = 1;P0 = Disp_Tab[QianNum];SEG_SELECT = 0;
37.             break;
38.             case 1:
39.                 BIT_SELECT = 1;P0 = 0xdf;BIT_SELECT = 0;
40.                 SEG_SELECT = 1;P0 = Disp_Tab[BaiNum];SEG_SELECT = 0;
41.             break;
42.             case 2:
43.                 BIT_SELECT = 1;P0 = 0xbf;BIT_SELECT = 0;
44.                 SEG_SELECT = 1;P0 = Disp_Tab[ShiNum];SEG_SELECT = 0;
45.             break;
46.             case 3:
47.                 BIT_SELECT = 1;P0 = 0x7f;BIT_SELECT = 0;
48.                 SEG_SELECT = 1;P0 = Disp_Tab[GeNum];SEG_SELECT = 0;
49.             break;
50.             default:P0 = 0x00; break;
51.         }
52.     }
53. }
54. void Timer0_ISR(void) interrupt 1
55. {
56.     static uInt16 uiCounter = 0;
57.     TH0 = 0xFC; TL0 = 0x66;
58.     uiCounter++;
59.     g_ucRefresh++;
60.     if(4 == g_ucRefresh)            // 总共 4 位数码管（0、1、2、3）
61.         g_ucRefresh = 0;
62.     if(1000 == uiCounter)           // 计 1000 次数，说明 1S 时间已到
63.     {
64.         uiCounter = 0;
65.         g_uiNum++;
66.         if(10000 == g_uiNum)    // 4 位数码管最多能显示 9999
67.             g_uiNum = 0;
68.     }
```

```
69.  }
70.  void Timer1_ISR(void) interrupt 3
71.  {
72.      uChar8 ucCounter;
73.      TH1 = 0x4C; TL1 = 0x00;
74.      ucCounter++;
75.      if(20 == ucCounter)          //  定时 20 * 50ms = 1s
76.      {
77.          ucCounter = 0;
78.          P2 = ~ P2;
79.      }
80.  }
```

该程序用了定时器 0 和定时器 1 分别来控制数码管和 LED 灯，时间基准分别为 1ms 和 50ms。28~31 行是用来分离 4 位数，这个在以后的数据处理中经常用到，望读者一定要理解并掌握计算的方法。其中数码管刷新并没有写成专门的函数，而是利用 1ms 的时间基准来刷新的，这样 4 个数码管占用的刷新周期为 4ms（小于前面提到的 10ms），这种思想能很好的解决由于延时函数带来的各种大大小小的问题（如 4 位数码管亮暗不同等）。所以对于单片机初学者，不是把别人的 Demo 程序下载到板子看看效果就行了，那样是学不会、玩不好单片机的，一定要沉下来、静下来，多思考、多总结别人的方法和经验。若遇到问题，首先要自己思考，静静的去想一想，若真的没法解决，当然可以去问别人，它山之石可以攻玉嘛，但一定不能遇到问题，自己都没想，就在网上发了一堆帖子，在各大论坛上留言，这样只能让自己变得很浮躁，当然就不会有进步了。

如果读者对程序要求不严格，或者做实验时不够仔细，那么此时或许会认为实验做到这步就很完美了。可对于要求严格的客户来说，或者说对于仔细的读者，可能会对此程序提出好多问题，例如，我们这里只用了后面四位数码管，可是前面的四位也会暗暗的随机显示一些数值，同时，实例 10 中提到的“鬼影”现象还是很讨厌的伴随着此程序，接下来就带着这些问题，再来深入的研究一下此实例，看能不能得到更加完美的实验效果呢？答案当然是肯定的，世上无难事，只怕有心人。只要读者有这个决心，就肯定能解决。

最后一个问题，56 行代码前为何加了 static，能不能省，为什么？答案前面章节见。

实例 3 请个大师来捉“鬼”—数码管的消隐

要消除数码管显示时的“鬼影”以及延时带来的种种问题，必须要从其产生的原理入手，只要这样，不仅能知其然，还能知其所以然。

数码管动态扫描中的“鬼影”现象，主要是由段选和位选的瞬态所产生，这里的瞬态也可理解为过度状态。在理论上，每个数码管显示时持续的时间为 1ms，1ms 之后，由于中断的原因，显示位会发生切换，如从第五个切换到第六个、第七个、第八个（前四个没用）。例如此时显示的数据为：1234。我们来详细看看这个切换过程，具体过程分析如下：

这里就从 34 行代码说起，在说之前，读者们需理清两个概念：

（1）C 语言代码是一句一句按顺序，从前往后执行的；

（2）单片机执行的速度是很快，但再快还是需要时间的。

进入第一个 case 语句（g_ucRefresh=0）之后，开“位”门，送位选数据，再关“位”们，整个过程很完美；之后开“段”门，送段选数据，再关“段”门，过程完美吗，NO！为何呢？因为执行完语句：SEG_SELECT = 1 后，“段”们已经开了，可此时 P0 口数据总线上的数值为：0xfe，这样，送出的段选数据就为：0xfe，刚也说了，单片机运行需要时间，那这段时间内第一个数码管就会显示：8。之后送入段选数据：0x06，这样，数码管才会显示 1。由此可见，扫描过程中所显示的数值“8”就不是我们想要的，因此在每次开“段”门之前加一句：P0 = 0x00（如下面代码中的：4、10、16、22 行），是很有必要的，这样，打开“段”门之后送过去的段选数据就是：0x00，也即 8 段数码管都灭。

在来看看，由 case“0”切换到 case“1”的过程。程序运行完：BIT_SELECT = 1 后，“位”门已经打开，可这时 P0 口数据总线上的数值显示“1”所用的段选数据，也就为：0x06，这样，则会选中：一、四、五、六、七、八位数码管，同样程序运行到真正送位选数据需要时间，那这段时间除了二、三位数码管不显示以外，别的六个都显示数字：1。之后才会执行语句：P0 = 0xdf。为了解决以上这个 BUG，需要在开“位”门之前加三行代码：SEG_SELECT = 1;P0 = 0x00;SEG_SELECT = 0（如下面代码的：2、8、14、20）。

增加了消隐之后的数码管扫描程序：

```
1.  case 0:
2.      SEG_SELECT = 1;P0 = 0x00;SEG_SELECT = 0;
3.      BIT_SELECT = 1;P0 = 0xef;BIT_SELECT = 0;
4.      P0 = 0x00;
5.      SEG_SELECT = 1;P0 = Disp_Tab[QianNum];SEG_SELECT = 0;
6.  break;
7.  case 1:
8.      SEG_SELECT = 1;P0 = 0x00;SEG_SELECT = 0;
9.      BIT_SELECT = 1;P0 = 0xdf;BIT_SELECT = 0;
10.     P0 = 0x00;
11.     SEG_SELECT = 1;P0 = Disp_Tab[BaiNum];SEG_SELECT = 0;
12. break;
13. case 2:
14.     SEG_SELECT = 1;P0 = 0x00;SEG_SELECT = 0;
15.     BIT_SELECT = 1;P0 = 0xbf;BIT_SELECT = 0;
16.     P0 = 0x00;
17.     SEG_SELECT = 1;P0 = Disp_Tab[ShiNum];SEG_SELECT = 0;
18. break;
19. case 3:
20.     SEG_SELECT = 1;P0 = 0x00;SEG_SELECT = 0;
21.     BIT_SELECT = 1;P0 = 0x7f;BIT_SELECT = 0;
22.     P0 = 0x00;
23.     SEG_SELECT = 1;P0 = Disp_Tab[GeNum];SEG_SELECT = 0;
24. break;
25. default:P0 = 0x00;
26. break;
```

程序经过重重修改之后，数码管显示亮度特别均匀，也没有一丝“鬼影”，同时也避开了延时函数，现在可以说做到了完美的要求。读者也看到了，程序的开发过程需要时间、需要思考，更需要实践的不断检验。由此可见，将理论与实践相结合，才能彰显玩好单片机之本色。可能这么说，对大家来说不够直观，那就贴一张图，一作说明，如图 9-6 所示。

图 9-6 数码管消隐对比图

9.5 延时版的消抖背后却埋藏着多少的深思——状态机法

或许读者并没有注意到实例 14~16 的缺陷，或许读者学的很“完美”，别人讲述的也很“自信”，但是这些“完美”、“自信”的背后却至少需要思考两个问题。

问题一：延时 10ms 是长是短呢？若睡大觉，多睡、少睡 5 小时都没什么，那么对于单片机呢，要知道单片机在 10ms 内能干好多事，这样势必会让单片机变成一个不守时的懒“人”。

问题二：判断按键释放是用 while(!Key1)，当然若是自己开发，自己用，或许知道按一下，松手才会执行后面的。可读者想过没，以后开发的产品 99.99%的不是自己用，而是别人用，那要是别人一把按下，再不松手，等着数值加加或者减减，可这一按，就直接按“死”了，因为程序会死在 while(!Key1)这里，这或许不是任何人想看到的吧！

鉴于以上情况，程序不得不深思，不得不重改。接下来读者就跟随残弈悟恩一起去探究它的“高级”法。或许还有比读者更先进、更完美的写法，但是笔者这里要讲述的状态机扫描法也值得读者好好一学，至少从如何改进程序、如何让程序健壮、如何节省单片机的 CPU 等方面，绝对有益而无害。

9.5.1 状态机简介

提到状态机，或许对一些新手们来说，会感到特别陌生。这种陌生其实对于大多数人来说都会有，就笔者而言，虽然以前在数字电路中看过（为何是看过，而不是听过，因为残弈自大二开始就从不去上课了）状态机，但是还没真正用过，之后玩 FPGA 时才好好学了这块知识，现在也算掌握了，接下来就和大家分享一下所谓的状态机。

有限状态机（FSM）思想广泛应用于硬件控制电路设计中，也是软件上常用的一种处理方法（软件上称为——有限消息机（FMM）。它把复杂的控制逻辑分解成有限个稳定状态，在每个状态上判断事件，将其变为离散数字处理，这样就符合计算机的工作特点。同时，因为有限状态机具有有限个状态，所以可以在实际的工程上实现。但这并不意味着其只能进行有限次的处理，相反，有限状态机是闭环系统，有限无穷，可以用有限的状态，处理无穷的事务。

状态机有 4 个要素：现态、条件、动作、次态。这样主要是为了理解状态机内在的因果关系。其中“现态”、“条件”是因，“动作”、“次态”是果。

（1）现态：指当前所处的状态。（2）条件：又称“事件”，触发状态转变的原因。

（3）动作：条件满足后执行的动作。（4）次态：条件满足后要迁往的新状态。

这么说或许有些模糊，那就举个生活中最简单的例子。假如人有三种状态：健康、感冒、康复中。触发的条件有：淋雨（T1）、看医生（T2）、休息（T3）。所以状态机就是健康-（T1）→感冒；感冒-（T2）→康复中；康复中-（T3）→健康，等等。正如这样，状态在不同的条件下跳转到自己或不同的状态。

接下来本应该给读者画一个状态图，限于篇幅，这里就不画了，到后面按键的检测中，再来张状态图，以做讲解。

9.5.2 状态机法的按键检测

说按键检测状态机前，先来上张具有连发功能的按键检测状态图（见图 9-7），接着分析此图，完了再来讲述如何将状态图装换为 C 语言代码。

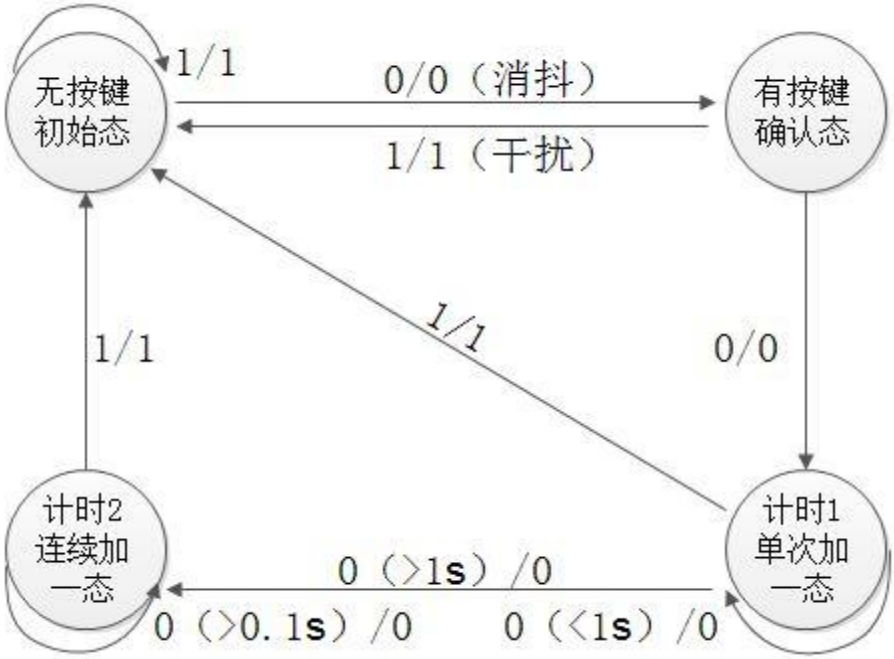


图 9-7 具有连发功能的按键检测状态图

接下来笔者笔者就详细来解释一下这个状态图。等这个状态图懂了，那么状态机甚至整个按键的检测过程就很清楚了。

在讲述之前，先来做几点说明。

说明一：现态、次态是相对的。例如初始态相对于确认态是现态，相对于单次加一态（连续加一态）就是次态。

说明二：图中的四个圈表示四种状态，也即按键就这四种有限状态。

说明三：带箭头的方向线指示状态转换的方向，当方向线的起点和终点都在同一个圆圈上时，则表示状态不变。

说明四：标在方向线旁斜线左、右两侧的二进制数分别表示状态转换前输入信号的逻辑值和相应的输出逻辑值。图中斜线前的 0 表示按键按下，1 表示按键未按下（或者释放）；斜线后的 0 表示按键按下后的电平状态为低电平，相反，1 表示高电平，也即按键未按下。

程序开始运行时，首先进来处于初始态（无按键按下），这时若按键未按下，则状态不

变，一直处于初始态。若此时按键状态值变为 0（低电平），说明有按键按下，但抖动是否消除，还需待定。但无论是否消除，肯定会进入确认态。进入之后，若没消除，则返回到初始态，若消除，则进入单次加一态，这时接着会判断按下的时间值，若时间小于 1s，键值加一，并返回到初始态；若判断此时按下的时间值大于 1s，则进入状态切换到连续加一态（连发态）。进入连发态后，键值每过 0.1s 就会自动加一，若此时按键释放，则就会进入初始态。这是不是恰好符合实际中按键的一般思路，当然是！

实例 4 独立按键的检测——状态机法

实例简介：当按下 MGMC-V1.0 实验板上的 K1 按键时，若按下到释放的时间小于 1s，则按一次，由 D1 亮变到 D2 亮，依次 D1~D8 循环点亮；若按下时间大于 1s，则按下之后每过 100ms，所亮灯有序的变化一次。如果该实例用实例 14 的方法来写，估计还真有难度，或者直接写不出来，笔者没试过，这个过程就留读者。有了上面的状态图分析，我们乘热打铁，赶紧来看看软件如何设计。老套路，先上源码，再来解释，具体源码如下：

```
1.  #include <reg52.h>
2.  typedef unsigned char uChar8;
3.  sbit KEY1 = P3^4;
4.  typedef enum KeyState{StateInit,StateAffirm,StateSingle,StateRepeat};
5.  /* ************************************************** */
6.  // 函数名称：KeyScan(void)
7.  // 函数功能：扫描按键
8.  // 入口参数：无
9.  // 出口参数：键值（num）
10. /* ************************************************** */
11. uChar8 KeyScan(void)
12. {
13.     static uChar8 KeyStateTemp = 0,KeyTime = 0;
14.     uChar8 num;
15.     bit KeyPressTemp;
16.     KeyPressTemp = KEY1;                    //读取 I/O 口的键值
17.     switch(KeyStateTemp)
18.     {
19.         case StateInit:                     //按键初始状态
20.             if(!KeyPressTemp)               //当按键按下，状态切换到确认态
21.                 KeyStateTemp = StateAffirm;
22.             break;
23.         case StateAffirm:                   //按键确认态
24.             if(!KeyPressTemp)               //抖动已经消除
25.             {
26.                 KeyTime = 0;
27.                 KeyStateTemp = StateSingle; //切换到单次触发态
28.             }
```

```
29.                 else KeyStateTemp = StateInit;      //还处于抖动状态，切换到初始态
30.                 break;
31.             case StateSingle:                       //按键单发态
32.                 if(KeyPressTemp)                    //按下时间小于 1s 且按键已经释放
33.                 {
34.                     KeyStateTemp = StateInit;       //按键释放，则回到初始态
35.                     num++;                          //键值加一
36.                     if(8 == num)    num = 0;
37.                 }
38.                 else if(++KeyTime > 100)            //按下时间大于 1s(100*10ms)
39.                 {
40.                     KeyStateTemp = StateRepeat;//状态切换到连发态
41.                     KeyTime = 0;
42.                 }
43.                 break;
44.             case StateRepeat:                       //按键连发态
45.                 if(KeyPressTemp)
46.                     KeyStateTemp = StateInit;       //按键释放，则进初始态
47.                 else                                //按键未释放
48.                 {
49.                     if(++KeyTime > 10)              //按键计时值大于 100ms（10*10ms）
50.                     {
51.                         KeyTime = 0;
52.                         num++;                      //键值每过 100ms 加一次
53.                         if(8 == num)    num = 0;
54.                     }
55.                     break;
56.                 }
57.                 break;
58.             default:   KeyStateTemp = KeyStateTemp = StateInit; break;
59.         }
60.         return num;
61.  }
62.  void Timer0Init(void)
63.  {
64.      TMOD = 0x01;                                   // 设置定时器 0 工作在模式 1 下
65.      TH0 = 0xDC;                                    // 定时 10ms
66.      TL0 = 0x00;                                    // 赋初始值
67.      TR0 = 1;                                       // 开定时器 0
68.  }
69.  /* ******************************************************* */
70.  // 函数名称：ExecuteKeyNum(void)
71.  // 函数功能：按键值来执行相应的动作
72.  // 入口参数：无
```

```
73.  // 出口参数：无
74.  /* ************************************************** */
75.  void ExecuteKeyNum(void)
76.  {
77.      static uChar8 KeyNum = 0;          //提个为题：这里的 static 能不能省略？为何？
78.      if(TF0)
79.      {
80.          TF0 = 0;
81.          TH0 = 0xDC; TL0 = 0x00;
82.          KeyNum = KeyScan();            //将 KeyScan()函数的返回值赋值给 KeyNum。
83.      }
84.      switch(KeyNum)
85.      {
86.          case 0: P2 = 0xfe; break;
87.          case 1: P2 = 0xfd; break;
88.          case 2: P2 = 0xfb; break;
89.          case 3: P2 = 0xf7; break;
90.          case 4: P2 = 0xef; break;
91.          case 5: P2 = 0xdf; break;
92.          case 6: P2 = 0xbf; break;
93.          case 7: P2 = 0x7f; break;
94.          default:P2 = 0xff; break;
95.      }
96.  }
97.  void main(void)
98.  {
99.      Timer0Init();
100.     while(1)
101.     {
102.         executeKeyNum();
103.     }
104. }
```

程序的细节，因为有详细的注释，残弈悟恩就不废话了。这里主要说明一下 KeyScan()函数中的各个条件判断。函数体中的各个判断条件（if、case、if...case）在这里统统可以理解为状态机中的“条件”，也即触发条件。正在运行的状态就是“现态”，当满足条件后待切换到的下一个状态就是“次态”。在每个状态中所执行的语句就是“动作”。这样完美的将状态机、定时器、按键结合到了一起，从而解决了该节刚开始提出的两个问题。当然以后应用中，该按键扫描程序还需整合，这个读者可以参考实例 44。笔者不可能将读者以后用的程序都写出来，这也不现实，但是若读者掌握了这种方法，以后肯定能开发出自己所需的好代码。

最后留读者一个作业：读者能不能将矩阵按键也用定时器+状态机的方法搞定？这个问题的具体答案，读者可以“追寻”残弈悟恩，或许还能给读者一个安心的小窝。

实例 5 矩阵按键的检测——状态机法

顺便也为大家把矩阵按键的状态机也附到这里，以便大家共享。

```
/* ********************************************************* */
// 版　权　　：兰州文轩电子科技有限公司
// 论　坛　　：电子工程师基地（www.ieebase.net）
// 交流群　　：143406243
// 版权所有，盗版必究。
/* ********************************************************* */
// 工　程　　:飞天开发板(MGMC-V2.0)
// 文件名　　: MatrixKeyMain.c
// 处理器　　: STC89C52RC
// 编译环境 : Keil4 C51
// 系统时钟 : 11.0592MHz
// 版　　本 : V1.0
// 生成日期 : 2014-02-25
// 修改日期 :
// 简单描述 : 基于状态机的矩阵按键实例
/* ********************************************************* */
#include <reg52.h>
/* ********************************************************* */
// 宏定义
/* ********************************************************* */
#define uInt16 unsigned int
#define uChar8 unsigned char

#define DATA P0                         //数据口
#define KEYPORT    P3                   //键盘接入端口
#define NOKEY 255                       //无键值标志
#define KEYMASK 0x0f                    //
/* ********************************************************* */
// 位定义
/* ********************************************************* */
sbit SEG_SELECT = P1^7;             //段选控制端
sbit BIT_SELECT = P1^6;             //位选控制端
/* ********************************************************* */
// 数组定义
/* ********************************************************* */
uChar8 code SEG_Tab[] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,
0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71,0x00};//段选显示表格
/* ********************************************************* */
// 全局变量定义
/* ********************************************************* */
```

```
41. uChar8 g_ucKeyNum = 16;                 //键值
42. uChar8 TimeCount = 0;                   //时间计数
43. bit TimeOK;                             //5ms 时间到标志
44. /* ******************************************************* */
45. // 函数名称：ScanKey()
46. // 函数功能：矩阵按键扫描
47. // 入口参数：无
48. // 出口参数：无
49. /* ******************************************************* */
50. uChar8 ScanKey(void)
51. {
52.     static uChar8 KeyState = 0;             //按键状态
53.     static uChar8 KeyValue;                 //临时键值
54.     static uChar8 KeyLine;                  //按键行值
55.     uChar8 KeyReturn = NOKEY;               //键值返回变量
56.     uChar8 i;                               //循环变量
57.     switch(KeyState)                        //状态判断
58.     {
59.         case 0:                                 //初始状态
60.         KeyLine = 0x10;
61.         for (i = 1;i <= 4;i++)
62.         {
63.             P3 = ~KeyLine;                  //第一行送低电平
64.             P3 = ~KeyLine;
65.             KeyValue = KEYMASK & P3;    //读取 P3 口列值
66.             if (KeyValue == KEYMASK)
67.             {
68.                 KeyLine <<= 1;
69.             }
70.             else
71.             {
72.                 KeyState++;
73.                 break;
74.             }
75.         }
76.         break;
77.
78.         case 1:                                 //确认状态
79.         if (KeyValue == (KEYMASK & P3))
80.         {
81.             switch(KeyLine | KeyValue)
82.             {
83.                 case 0x87:KeyReturn = 15;break;
84.                 case 0x8b:KeyReturn = 11;break;
```

```
85.                         case 0x8d:KeyReturn = 7;break;
86.                         case 0x8e:KeyReturn = 3;break;
87.
88.                         case 0x47:KeyReturn = 14;break;
89.                         case 0x4b:KeyReturn = 10;break;
90.                         case 0x4d:KeyReturn = 6;break;
91.                         case 0x4e:KeyReturn = 2;break;
92.
93.                         case 0x27:KeyReturn = 13;break;
94.                         case 0x2b:KeyReturn = 9;break;
95.                         case 0x2d:KeyReturn = 5;break;
96.                         case 0x2e:KeyReturn = 1;break;
97.
98.                         case 0x17:KeyReturn = 12;break;
99.                         case 0x1b:KeyReturn = 8;break;
100.                        case 0x1d:KeyReturn = 4;break;
101.                        case 0x1e:KeyReturn = 0;break;
102.                    }
103.                    KeyState++;
104.            }
105.            else
106.                    KeyState--;
107.            break;
108.
109.            case 2:                                    //完成态
110.            P3 = 0x0f;
111.            P3 = 0x0f;
112.            if((KEYMASK & P3) == KEYMASK)
113.            {
114.                    KeyState = 0;
115.            }
116.            break;
117.        }
118.        return KeyReturn;
119. }
120. /* ********************************************************* */
121. // 函数名称：Display()
122. // 函数功能：数码管刷新
123. // 入口参数：需显示的数字(ucVal)
124. // 出口参数：无
125. /* ********************************************************* */
126. void Display(uChar8 ucVal)
127. {
128.            BIT_SELECT = 1;
```

```
129.            DATA = 0x00;
130.            BIT_SELECT = 0;
131.            SEG_SELECT = 1;
132.            DATA = SEG_Tab[ucVal];
133.            SEG_SELECT = 0;
134.            DATA = 0x00;
135. }
136. /* ******************************************************* */
137. // 函数名称：Timer0Init()
138. // 函数功能：
139. // 入口参数：无
140. // 出口参数：无
141. /* ******************************************************* */
142. void Timer0Init(void)
143. {
144.     TMOD = 0x01;
145.     TH0 = 0xFC;
146.     TL0 = 0x66;                    //1ms 计时
147.     EA = 1;
148.     ET0 = 1;
149.     TR0 = 1;
150. }
151. /* ******************************************************* */
152. // 函数名称：main()
153. // 函数功能：扫描键值并显示键值
154. // 入口参数：无
155. // 出口参数：无
156. /* ******************************************************* */
157. void main(void)
158. {
159.     uChar8 KeyTemp;                //临时存放键值变量
160.     Timer0Init();
161.     while(1)
162.     {
163.         if (TimeOK)
164.         {
165.             KeyTemp = ScanKey();
166.             if (KeyTemp != NOKEY)
167.             {
168.                 g_ucKeyNum = KeyTemp;
169.             }
170.         }
171.     }
172. }
```

```
/* ******************************************************* */
// 函数名称：Timer0_ISR()
// 函数功能：定时器 0 中断服务
// 入口参数：无
// 出口参数：无
/* ******************************************************* */
void Timer0_ISR(void) interrupt 1
{
    TH0 = 0xFC;
    TL0 = 0x66;
    Display(g_ucKeyNum);
    if(TimeCount >= 5)
    {
        TimeCount = 0;
        TimeOK = 1;
    }
    else TimeCount++;
}
```

最后的最后，为大家总结一下中断。常用中断，一般有两种模式，一、定时器+中断（前面有好多实例了）；一种是外部中断（用来计数等），这个就留读者自行研究了。