第5章 MCS-51单片机中断系统

CHAPTER 5

5.1 中断系统概述

5.1.1 中断及中断源

在计算机系统中，所谓中断(Interrupt)就是指CPU在执行程序的过程中，由于某一事件发生，要求CPU暂停正在执行的程序，而去执行相应的处理程序，待处理结束后，再返回到原来程序停止处继续执行。触发产生中断(Interrupt Arising)的事件称为中断源(Interrupt Source)。中断发生后，中断源向CPU发出的请求信号叫做中断请求(Interrupt Request)。CPU停止执行现行程序而处理中断称为中断响应(Interrupt Response)，CPU停止执行现行程序的间断处称为断点(Breakpoint)，CPU执行的与中断相关的处理程序称为中断处理程序或中断服务程序(Interrupt Service Routines)，处理过程即为中断处理(Interrupt Process)。中断系统是指完成中断处理的软件和硬件资源，它包括中断源的产生、中断判优、中断响应、中断查询、中断处理等过程，它是计算机系统一个重要的组成部件。一个典型的中断处理过程如图5.1所示。

CPU响应中断请求调用中断处理程序的过程与主程序调用子程序的过程相似(见图5.2)，但是，它们是不同的，主要区别在于：调用子程序时，调用哪个子程序、完成什么任务是程序设计时事先安排好的，采用子程序调用指令实现。中断事件发生是随机的，哪个事件发生、何时调用中断处理程序是事先无法确定的，在程序中无法事先安排调用指令，调用中断处理程序的过程是由硬件自动完成的。

触发中断的事件可以是计算机外部的，也可以是计算机内部的，归纳起来，中断源有以下几种情况。

(1) 外部设备发生某一事件，如打印机准备就绪、被控设备的参数超过限位阈值等。

(2) 计算机内部某个事件发生，如定时器/计数器溢出、串行口接收到一帧数据等。

(3) 计算机发生了故障引起中断。如系统电源掉电、运算溢出、系统出错等事件。

(4) 人为设置中断。在编程和调试时人为设置的中断事件，如单步执行、设置断点。

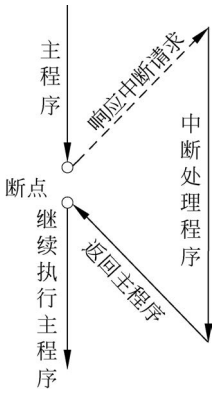


图 5.1　一个典型的中断处理过程

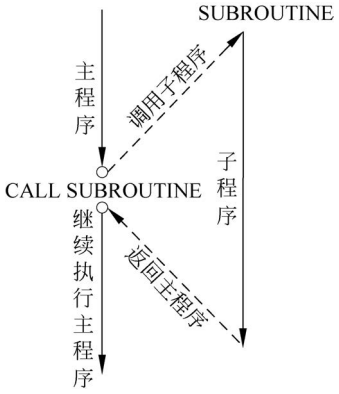


图 5.2　调用子程序的过程

5.1.2　中断技术的作用

计算机系统采用中断技术，可以提高 CPU 的工作效率和处理问题的灵活性，其作用体现在以下几个方面。

（1）解决了快速 CPU 和低速外设之间的速度匹配问题，使 CPU 和外设同时工作。CPU 的工作速度是微秒级的，而外设的工作速度一般在毫秒级以上。CPU 在启动外设工作后继续执行主程序，同时外设也在工作。当外设完成某项任务后，就发出中断申请，请求 CPU 中断它正在执行的程序，转去执行中断处理程序。中断处理完之后，CPU 恢复执行主程序，外设也继续工作。这样，可以启动多个外设同时工作，有效地提高了 CPU 的效率。

（2）可以实现实时处理。所谓实时控制，就是要求计算机能及时地响应被控对象提出的分析、计算和控制等请求，使被控对象保持在最佳工作状态，以达到预定的控制效果。在实时控制中，现场的各种参数、状态随时间和现场变化，可根据要求随时向 CPU 发出中断申请请求 CPU 处理，若中断条件满足，CPU 立即响应并处理。

（3）可以对突发故障及时处理。对于系统掉电、存储出错、运算溢出等难以预料的情况或故障，可由故障源向 CPU 发出中断请求，CPU 响应后，按照事先拟定处理预案进行处理。

（4）可以实现多任务资源共享。当一个 CPU 面对多项任务时，可能会出现在同一时刻几项任务同时要求 CPU 处理的情况，即资源竞争。中断技术是解决资源竞争的有效方法，它可以使多项任务共享一个资源，使得 CPU 能够分时完成多项任务。

5.2　MCS-51 单片机的中断系统

MCS-51 单片机有 5 个中断源，分为两个中断优先级，可以实现两级中断嵌套。CPU 对中断采用两级管理，用户可以根据需要来设定 CPU 是否开放中断，而且每个中断源都可以独立地设定为允许或禁止请求中断。每个中断源的优先级也可以独立地设定为高优先级或低优先级。MCS-51 单片机中断系统结构如图 5.3 所示。

MCS-51 单片机的 5 个中断源分别是两个外部事件中断、两个定时器/计数器计数溢出事件触发的中断和一个串行口缓冲器接收到或发送完数据触发的中断。

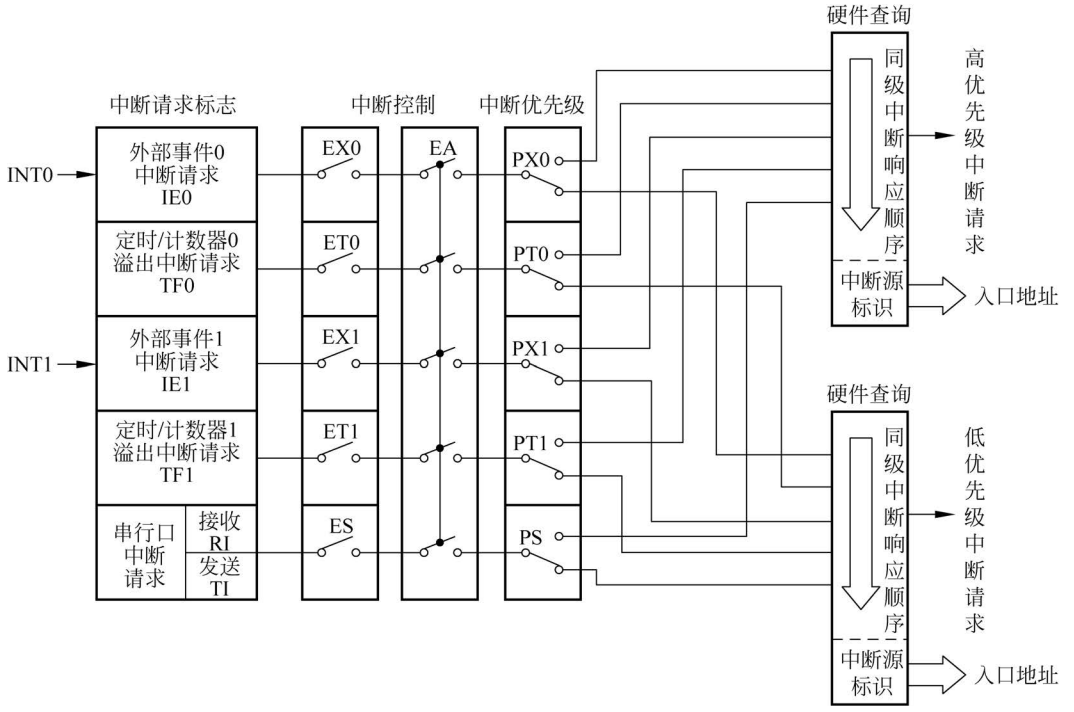


图 5.3　MCS-51 单片机中断系统结构

外部事件中断是由来自单片机外部的信号触发的，中断请求信号分别由引脚 $\overline{\text{INT0}}$(P3.2)和 $\overline{\text{INT1}}$(P3.3)引入。定时器/计数器溢出中断是计数器发生计数溢出而触发的中断。定时器/计数器溢出中断是为了实现定时或计数的需要而设置的。当计数器发生计数溢出时，意味着定时时间到或计数值已达到要求，向 CPU 请求中断。这是由单片机内部发出的中断请求。串行口中断是为单片机串行数据发送和接收的需要而设置的。当串行口接收或发送完一帧串行数据时，产生一个中断请求。这种请求也是在单片机内部发出的。

中断触发后，中断触发标志被登记在寄存器中，MCS-51 单片机没有专用的中断标志寄存器，它用两个特殊功能寄存器 TCON 和 SCON 分别登记 5 个中断源的中断触发标志，以此向 CPU 请求中断。CPU 开放中断与否、中断源是否允许中断由中断控制寄存器 IE 设定，而中断优先级由中断优先级寄存器 IP 中的位来设定。查询电路用来处理相同优先级时 CPU 响应中断请求的顺序，以实现硬件调用响应的中断处理程序。下面介绍 MCS-51 单片机的中断系统。

5.2.1　MCS-51 单片机的中断标志

MCS-51 单片机的 5 个中断源的中断请求标志如图 5.3 所示。中断标志位的状态为 1 时，表明对应的中断源触发了中断，产生了中断请求。这些标志位分别由两个特殊功能寄存器来存储，即定时器/计数器控制寄存器(Timer/Counter Control Register，TCON)和串行口控制寄存器(Serial Port Control Register，SCON)。中断系统在每个机器周期采样这些标志，并在下一个机器周期查询它们，以确定哪些中断源发出了中断请求，然后进行相应的处理。

1. 定时器/计数器控制寄存器(TCON)

定时器/计数器控制寄存器 TCON 锁存外部事件中断请求标志以及定时器/计数器的溢出标志。TCON 的地址为 88H，各位的位地址为 88H～8FH，TCON 寄存器的内容如图 5.4 所示。

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

图 5.4　TCON 寄存器的内容

1) 外部事件中断请求标志

IE0 为外部事件中断 $\overline{\text{INT0}}$ 的中断请求标志位，IE1 为外部事件中断 $\overline{\text{INT1}}$ 的中断请求标志位。当外部事件发生时，在单片机引脚 $\overline{\text{INT0}}$ 和 $\overline{\text{INT1}}$ 产生中断请求信号，IE0 和 IE1 分别由来自引脚 $\overline{\text{INT0}}$ 和 $\overline{\text{INT1}}$ 的外部中断请求信号触发。

外部事件中断可以由电平触发(Level Activated)或跳变触发(Transition Activated)，中断触发的方式取决于 TCON 寄存器中 IT0 和 IT1 的设定。IT0 和 IT1 分别为定义 $\overline{\text{INT0}}$ 和 $\overline{\text{INT1}}$ 引脚上中断请求信号的触发方式。

下面以外部事件中断 $\overline{\text{INT0}}$ 为例来说明外部事件中断触发机制。

当 IT0 为 1 时，设置外部事件中断 $\overline{\text{INT0}}$ 为跳变触发方式，如果在 $\overline{\text{INT0}}$ 引脚上出现高电平变为低电平的负跳变，IE0 位由硬件自动置 1，以此为标志向 CPU 请求中断。CPU 响应中断时，自动将中断请求标志 IE0 清零。

当 IT0 为 0 时，外部事件中断为电平触发方式。在电平触发方式时，中断请求标志由外部中断请求触发信号控制，如果在 $\overline{\text{INT0}}$ 引脚上为低电平时，中断标志位 IE0 位则由硬件置 1，以此为标志向 CPU 请求中断，如果在 $\overline{\text{INT0}}$ 引脚上为高电平时，中断标志位 IE0 位则被清零。

外部事件中断为跳变触发方式时，在两个连续的机器周期内，第 1 个机器周期在 $\overline{\text{INT0}}$ 引脚上检测到的中断请求信号为高电平，第二个机器周期检测到其为低电平，那么，由硬件自动把 IE0 置 1，向 CPU 请求中断。因此，在这种触发方式下，中断请求信号的高电平和低电平的持续时间应不少于 1 个机器周期，以保证跳变能够被检测到。CPU 响应中断，IE0 自动清零。

在单片机复位时，IT0 被清零，外部事件中断为电平触发方式。

外部事件中断 $\overline{\text{INT1}}$ 的中断触发机制与 $\overline{\text{INT0}}$ 类似。

2) 定时器/计数器溢出标志

TF0 为定时器/计数器 T0 的计数溢出标志位，TF1 为定时器/计数器 T1 的计数溢出标志位。

以定时器/计数器 T0 为例，定时器/计数器 T0 启动计数后，从初始值开始加 1 计数，当计数器计满后(计数器的所有位都为 1)，再计 1 次，计数器溢出，溢出标志位 TF0 由硬件自动置 1 并锁存，以此向 CPU 请求中断。CPU 响应计数器溢出中断后，标志位 TF0 被自动清零。

定时器/计数器 T1 的中断触发过程与 T0 类似。

2. 串行口控制寄存器(SCON)

串行口控制寄存器 SCON 锁存串行口发送结束标志和接收到数据的标志，不论哪个中

断标志有效都会触发串行口中断。SCON 的地址为 98H，各位的位地址为 98H～9FH，SCON 寄存器的内容如图 5.5 所示。

D7	D6	D5	D4	D3	D2	D1	D0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

图 5.5　SCON 寄存器的内容

1）串行口发送中断请求标志位：TI

当串行口发送缓冲器 SBUF 发送完一帧数据后，由硬件自动把 TI 置 1，以此向 CPU 请求中断。值得注意的是，在 CPU 响应中断时，TI 并不会被自动清零，必须由用户在中断处理程序中用软件清零；否则，CPU 将会陷入响应中断和中断处理当中。

2）串行口接收中断请求标志位：RI

当串行口接收缓冲器 SBUF 接收完一帧串行数据后，由硬件把 RI 置 1，以此向 CPU 请求中断。同样，在 CPU 响应中断时，RI 并不会被自动清零，必须由用户在中断处理程序中用软件清零；否则，CPU 将会陷入响应中断和中断处理当中，将会造成数据帧的丢失。

由于串行口接收和发送共享一个中断源，发送结束标志 TI 和接收到数据标志 RI 只要其中有一个被置 1，都会产生串行口中断请求。因此，在双工通信时，为了辨别哪一个触发了中断，首先必须在中断处理程序中检测 TI 和 RI 的状态，然后清除标志位（TI 或 RI），再进行相应的中断处理，以保证接收和发送的持续进行。

5.2.2　MCS-51 单片机的中断控制

如图 5.3 所示，MCS-51 单片机的中断控制分为两级，第一级通过 5 个中断允许控制位来确定屏蔽或者允许某个中断源的中断请求，第二级通过 1 个控制位来确定 CPU 开放或禁止中断。MCS-51 单片机提供了一个专门的特殊功能寄存器——中断允许寄存器（Interrupt Enable Register，IE）来保存这些中断允许控制位。IE 寄存器的地址为 0A8H，寄存器中各位的位地址为 0A8H ～0AFH。IE 寄存器的内容如图 5.6 所示。

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

图 5.6　IE 寄存器的内容

1. CPU 的中断控制位：EA

EA 为 MCS-51 单片机的 CPU 中断控制位。

当 EA＝0，禁止所有中断源中断 CPU 工作，即禁止 CPU 响应任何中断请求。

当 EA＝1，允许中断源中断 CPU 工作，即开放 CPU 的中断响应功能。CPU 开放中断后，每个中断源可以独立地设置为禁止或允许中断。

2. 外部中断允许控制位：EX0 和 EX1

EX0 为外部事件中断 $\overline{\text{INT0}}$ 的中断允许控制位。

当 EX0＝0，禁止外部事件中断 CPU 工作，即屏蔽了中断请求，即使外部事件发生，中断标志 IE0 置 1，CPU 也不会响应。

当 EX0＝1，允许外部事件中断 CPU 工作。在 CPU 开放中断（EA 为 1）且该控制位为 1 的情况下，外部事件发生时 IE0 被置 1，向 CPU 请求中断。

EX1 为外部事件中断 $\overline{\text{INT1}}$ 的中断允许控制位。外部事件中断 $\overline{\text{INT1}}$ 的中断控制过程与 $\overline{\text{INT0}}$ 类似。

3. 定时器/计数器溢出中断允许控制位：ET0 和 ET1

ET0 为定时器/计数器 T0 溢出中断允许控制位。

当 ET0＝0，禁止定时器/计数器计数溢出时中断 CPU 工作。在这种情况下，即使计数器计数溢出，溢出标志位 TF0 置 1，CPU 也不会响应这个中断请求。TF0 可作为查询计数器是否溢出的测试标志。

当 ET0＝1，允许定时器/计数器计数溢出时中断 CPU 工作。在 CPU 开放中断且该控制位为 1 的情况下，一旦计数器计数溢出就会把 TF0 置 1，向 CPU 请求中断。

ET1 为定时器/计数器 T1 溢出中断允许控制位。定时器/计数器 T1 溢出中断控制过程与 T0 类似。

4. 串行中断允许控制位：ES

ES 为串行口的中断允许控制位。当 ES＝0 时，禁止串行口中断，即使接收缓冲器接收到数据把接收中断标志 RI 置位，或者发送缓冲器发送完数据把发送中断标志 TI 置位，CPU 也不会响应这个中断请求。标志位 RI 和 TI 可以作为接收和发送结束的测试标志位。当 ES＝1 时，允许串行口中断，RI 和 TI 二者中只要有一个为 1，即可向 CPU 请求中断。

MCS-51 单片机复位后，IE 被清零，因此，复位后所有中断都是被禁止的。若需要允许中断，必须根据需要重新设置 IE。

例 5.1　一个单片机应用系统要求外部事件 $\overline{\text{INT1}}$、定时器/计数器 T0 以及串行口具有中断功能，如何设定 IE 寄存器？

因为应用系统要求 $\overline{\text{INT1}}$、T0 以及串行口具有中断功能，首先必须使 CPU 开放中断，因此，EA 应设置为 1，$\overline{\text{INT1}}$、T0 以及串行口对应的中断控制位 EX1、ET0 和 ES 也应为 1。程序如下：

```
IE = 0x96;
```

或：

```
EA = 1;     ;CPU 开放中断
EX = 1;     ;允许 INT1 中断
ET = 1;     ;允许定时器/计数器 T0 溢出中断
ES = 1;     ;允许串行口中断
```

5.2.3　MCS-51 单片机的中断优先级

当多个中断源同时请求中断，或者 CPU 正在处理一个中断，又有了新的中断请求，对于上述情形 MCS-51 单片机该如何处理呢？MCS-51 单片机设置了一个中断优先级寄存器(Interrupt Priority Register，IP)用于设置中断源的优先级，每个中断源可以被设置为高优先级或者低优先级(见图 5.3)，可以实现两级中断嵌套。对于上述问题，MCS-51 单片机有以下处理原则。

(1) 多个中断源同时向 CPU 请求中断时，首先响应高优先级中断源的中断请求。

(2) 当 CPU 正在处理一个中断，又有新的中断请求时，如果新的中断请求的优先级高于正在处理的中断，则 CPU 暂停正在执行的中断处理，去响应新的中断请求，即高优先级

中断请求可以中断低优先级的中断处理，从而实现中断嵌套，如图 5.7 所示。如果是新的中断请求的优先级与正在处理的中断相同，或者其优先级低于正在处理的中断，那么，CPU 不会响应这个中断请求，将继续执行中断处理，待本次中断处理结束返回后，再对新的中断请求进行处理，也就是说，相同优先级和低优先级的中断请求不能中断高优先级或相同优先级的中断处理。

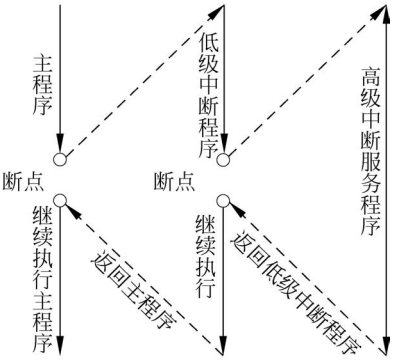


图 5.7　中断嵌套

中断优先级寄存器 IP 寄存器的地址为 0B8H，各位的位地址为 0B8H～0BFH。IP 寄存器的内容如图 5.8 所示。

D7	D6	D5	D4	D3	D2	D1	D0
—	—	—	PS	PT1	PX1	PT0	PX0

图 5.8　IP 寄存器的内容

（1）PX0 为外部事件中断 $\overline{\text{INT0}}$ 的优先级设定位；PX0＝0，该中断源为低优先级，PX0＝1，该中断源为高优先级。

（2）PT0 为定时器/计数器 T0 的中断优先级设定位；PT0＝0，该中断源为低优先级，PT0＝1，该中断源为高优先级。

（3）PX1 为外部事件中断 $\overline{\text{INT1}}$ 的优先级设定位；PX1＝0，该中断源为低优先级，PX1＝1，该中断源为高优先级。

（4）PT1 为定时器/计数器 T1 的中断优先级设定位；PT1＝0，该中断源为低优先级，PT1＝1，该中断源为高优先级。

（5）PS 为串行口中断的优先级设定位。PS＝0，该中断源为低优先级；PS＝1，该中断源为高优先级。

单片机复位时，IP 被清零，所有中断源被默认为低优先级中断。在应用系统设计时，可以根据需要把所用中断源设置为高优先级或低优先级中断。

例 5.2　把定时器/计数器 T0 和串行口中断源设置为高优先级。

设置优先级的程序如下：

```
IP = 0x12;
```

或：

```
PT0 = 1;    //设置定时器/计数器 T0 为高优先级中断源
PS = 1;     //设置串行口为高优先级中断源
```

如果有多个相同优先级的中断源同时向 CPU 请求中断，这时 CPU 该如何应对呢？如图 5.3 所示，MCS-51 单片机的中断系统设立了一个硬件查询电路，由中断系统内部的查询顺序来确定 CPU 优先响应哪一个中断请求。优先级相同时，5 个中断源的优先级由高到低排列顺序见表 5.1。

表 5.1　优先级相同时的中断优先级排列顺序

序号	中　断　源	中断请求标志	优先级
1	外部事件中断 $\overline{\text{INT0}}$	IE0	最高 ↓ 最低
2	定时器/计数器 T0 溢出中断	TF0	
3	外部事件中断 $\overline{\text{INT1}}$	IE1	
4	定时器/计数器 T1 溢出中断	TF1	
5	串行口接收和发送中断	RI 和 TI	

5.2.4　MCS-51 中断响应及处理过程

在程序中设置了 CPU 的中断控制位和中断允许控制位以后，当中断源触发中断时，相应的中断标志位被置 1。MCS-51 单片机的中断系统在每一个机器周期对所有中断标志位的状态进行采样，并在随后的一个机器周期查询这些中断标志，以确定哪一个中断源请求中断。如果中断系统检测到某个中断标志为 1，则表明该中断源向 CPU 发出了中断请求。但是，MCS-51 单片机的 CPU 响应中断请求是有条件的，如果此时不存在下列 3 种情形，CPU 将响应这个中断请求，立即产生一个硬件调用，使程序转移到相应的中断处理程序入口地址处调用中断处理程序，进行中断处理。3 种情形如下。

（1）CPU 正在处理相同优先级或高优先级的中断。

（2）当前的机器周期不是指令的最后一个机器周期。

（3）正在执行的指令是中断返回指令 RETI 或者是访问特殊功能寄存器 IE 或 IP 的指令。

CPU 响应中断时，必须是在一条指令执行结束之后。另外，CPU 执行 RETI 指令和对寄存器 IE 和 IP 访问的指令时，即使指令执行结束也不会立即响应，必须至少再执行一条指令方可响应中断请求。

对于有的中断源，CPU 在响应时会自动清除中断请求标志，如外部事件中断 $\overline{\text{INT0}}$ 和 $\overline{\text{INT1}}$ 跳变触发方式时的中断请求标志 IE0 和 IE1，定时器/计数器溢出的中断标志 TF0 和 TF1。

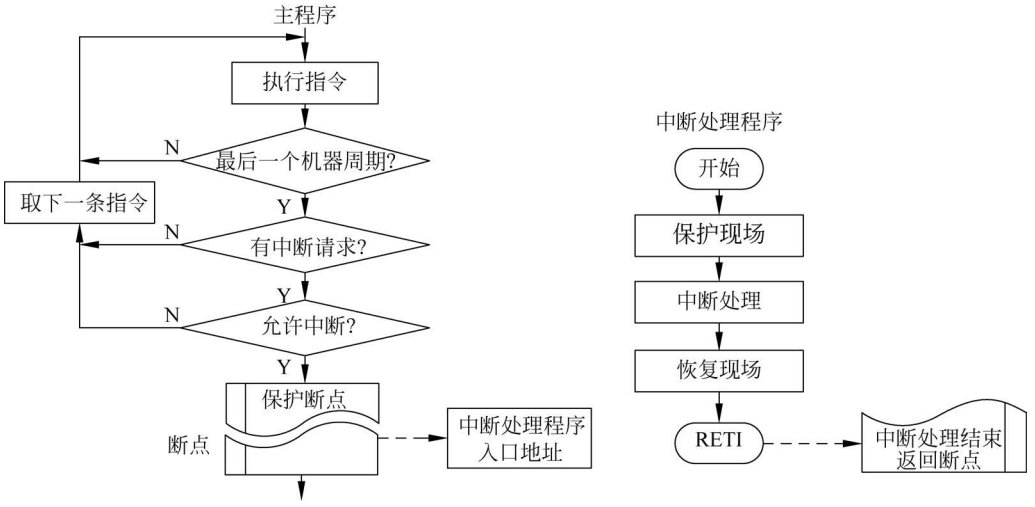
CPU 响应中断请求时，中断系统会根据中断源的优先级把相应的高优先级触发器或低优先级触发器置 1，以封锁相同优先级和低级优先级的中断请求；然后由硬件自动把当前程序计数器 PC 的内容（即断点）压入堆栈保护，并且把相应的中断处理程序入口地址装入程序计数器 PC，使程序转移到中断处理程序。

中断处理程序是根据处理中断事件的预案而设计的应用程序，即某个中断触发以后需要完成哪些任务。MCS-51 单片机各中断源的中断处理程序入口地址是固定的，见表 5.2。一旦 CPU 响应了某个中断请求，就会直接转到相应的入口地址去执行程序。

表 5.2　各中断源的中断处理程序入口地址和编号

序　　号	中　断　源	入 口 地 址	中 断 编 号
1	外部事件中断 $\overline{\text{INT0}}$	0003H	0
2	定时器/计数器 T0 溢出中断	000BH	1
3	外部事件中断 $\overline{\text{INT1}}$	0013H	2
4	定时器/计数器 T1 溢出中断	001BH	3
5	串行口接收和发送中断	0023H	4

MCS-51 单片机 CPU 的中断响应过程可用图 5.9 描述。

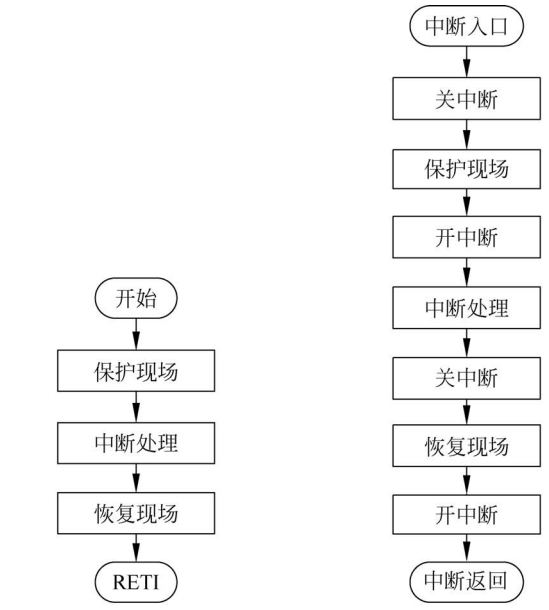


(a) MCS-51单片机CPU响应中断的过程　　(b) 中断处理过程

图 5.9　MCS-51 单片机 CPU 的中断响应过程

应用系统中只包含一个优先级中断源时，中断处理程序结构如图 5.10 所示。

值得注意的是，虽然 CPU 响应中断时，自动进行了置位高或低优先级触发器、保护断点、装入中断入口地址到 PC 等操作，但并没有关中断的操作。因此，应用系统包含高低优先级中断源时，为了防止高优先级中断响应干扰现场保护和恢复，中断嵌套程序设计时低优先级中断处理程序如图 5.10(b)所示，而高优先级中断处理程序的结构则无须考虑低中断处理的干扰，其结构图为图 5.10(a)。



(a) 中断处理程序的一般结构　　(b) 低优先级中断处理程序结构

图 5.10　中断处理程序结构

当包含多个相同优先级的中断源时，高、低优先级中断处理可采用图 5.10 的两种结构，但必须保证各个中断处理程序使用的单元、寄存器不冲突，除非有必要。

由第 4 章可知，在 C51 语言中，中断处理程序被作为一种特殊的函数——中断函数。由于中断函数既不传递参数，又没有返回值，其常用格式为：

```
void 函数名(void) interrupt n using m
{
  函数体;
}
```

interrupt n 指出中断源的编号(见表 5.2)，using m 用来指定该中断函数所用的工作寄存器组，m 为 0～3。

中断处理程序被编译器编译产生目标代码时，会做如下处理。

(1) 根据中断源编号，自动生成跳转中断入口地址的指令。

(2) 生成保护现场的代码，实现 ACC、B、DPH、DPL 和 PSW 的内容的入栈保护。

(3) 如果中断函数没有用 using 属性指定该函数要使用的工作寄存器组，则生成保护所有工作寄存器的代码，把工作寄存器内容保护在堆栈中。如果使用 using 属性指定，则不产生代码。

(4) 生成恢复现场的代码，在退出中断函数之前，需要恢复保护在堆栈中的工作寄存器和特殊寄存器内容。

(5) 生成中断返回指令(RETI)代码

下面用一个定时器/计数器 T0 中断处理程序的例子来说明生成目标代码的原理。该函数使用了工作寄存器组 3。为了便于说明，把语句所在行的行号标记在其左侧。C51 语言源文件如下：

```
1         extern bit alarm;
2         int alarm_count;
3
4
5         void falarm(void) interrupt 1 using 3 {
6          alarm_count * = 2;
7          alarm = 1;
8        }
```

编译后产生的代码文件为：

```
   ; FUNCTION falarm(BEGIN)
0000 C0E0        PUSH   ACC
0002 C0D0        PUSH   PSW
                     ; SOURCE LINE # 5
                     ; SOURCE LINE # 6
0004 E500        MOV    A,alarm_count + 01H
0006 25E0        ADD    A,ACC
0008 F500        MOV    alarm_count + 01H,A
000A E500        MOV    A,alarm_count
000C 33          RLC    A
000D F500        MOV    alarm_count,A
                     ; SOURCE LINE # 7
000F D200        SETB   alarm
                     ; SOURCE LINE # 8
```

```
0011 D0D0        POP   PSW
0013 D0E0        POP   ACC
0015 32          RETI
     ; FUNCTION falarm(END)
```

从生成的代码文件可以看出，编译后的目标代码由保护现场、中断处理、恢复现场和中断返回 RETI 几部分构成，同时生成了跳转到中断入口地址的指令。另外，由于指定了工作寄存器组，在代码中没有保护工作寄存器的代码。

在应用中断函数时应注意以下几点。

（1）中断函数不能传递参数。如果想要中断函数有返回值，则该函数不能使用 using 属性，因为要返回的值是存放在工作寄存器中的，返回时要恢复到函数调用之前工作寄存器组，会导致错误的返回值。如果中断函数用工作寄存器传递参数，同样也不能使用 using 属性，因为中断函数切换工作寄存器组会丢失这些参数。例如在一个中断函数 ISR1 中调用函数 FUN1，程序如下：

```
void ISR1(void) interrupt 0 using 1
{
    函数体;
}
```

函数 FUN1 声明如下：

```
void FUN1(void)
{
    函数体;
}
```

ISR1 被指定使用工作寄存器区 1，假设 FUN1 在编译时被默认使用工作寄存器区 0，由于 8051 单片机没有工作寄存器到工作寄存器的传送指令。因此，编译器只能对此生成工作寄存器到存储器单元的传送指令。编译器可根据所用的工作寄存器区计算出它的 8 个工作寄存器的单元地址。例如，计算得到函数 FUN1 使用的工作寄存器区 0 的 8 个工作寄存器是 0x00～0x07。如果所选的工作寄存器区不是 0 区，则该函数编译时可能会覆盖这些单元，导致传递的数据丢失，函数的运行结果不正确。为了保证中断函数正确调用 FUN1 函数，可使用 registerbank 编译指令指定 FUN1 使用工作寄存器区 1，这样编译时在函数中不生成切换工作寄存器区的代码。

（2）中断函数不能作为普通的自定义函数调用。直接调用中断过程是无意义的，中断函数退出时 CPU 执行了 RETI 指令，意味着 CPU 响应了一次无中断请求的中断处理，可能导致不可预料的执行结果。另外，也不能通过函数指针调用中断函数。

（3）如果中断函数调用了其他函数，则被调用的函数所用工作寄存器须与中断函数一致。另外，在这种情况下，被调用的函数最好设置为可重入的，这是由于中断是随机的，应用程序运行时有可能会出现中断函数所调用的函数被嵌套调用的情形。其次，中断函数最好写在应程序尾部，且禁止使用 extern 存储类型说明，以防其他程序调用。

（4）编译器编译时，进入中断函数，寄存器 ACC、B、PSW 会入栈保护，中断处理结束退出时这些寄存器内容会被从堆栈中恢复。如果有属性 using n，在中断处理程序中，先对 PSW 内容入栈保护，再修改 PSW 中寄存器选择位指定中断处理程序所用的工作寄存器组。

（5）如果应用系统使用了多个中断源，这些中断源的优先级相同，那么它们的中断函数

可以共享同一个工作寄存器组，因为这些中断函数执行时不会被其他中断源的中断请求所中断。以此类推，在应用系统中存在不同优先级的中断源时，可以给具有相同中断优先级的所有中断函数分配给相同的工作寄存器区，这样可减少应用程序所需的工作寄存器区和存储空间。

5.3　外部事件中断及应用

5.3.1　外部事件中断的响应时间

1. 触发方式

MCS-51 单片机的外部事件中断请求是通过 $\overline{\text{INT0}}$ 和 $\overline{\text{INT1}}$ 引脚引入的，通过软件设置触发方式，中断触发方式既可以为电平触发方式，也可以为跳变触发方式，在使用外部事件中断源时，中断触发信号必须与触发方式协调一致，使产生中断触发信号的电路满足以下要求。

（1）电平触发方式时，由于中断请求标志完全由引脚 $\overline{\text{INT0}}$ 或 $\overline{\text{INT1}}$ 的电平控制，虽然在每个机器周期中断系统都采样中断标志位的状态，但由于单片机只能在执行完一个指令后响应中断请求，因此，$\overline{\text{INT0}}$ 或 $\overline{\text{INT1}}$ 引脚上的中断请求信号必须保持足够长的时间，直到中断实际发生为止；否则，会丢失中断请求。在系统存在多级中断时，必须重视触发信号的有效时间。另外，中断触发信号低电平的维持时间也不能太长，在 CPU 响应中断，进入子程序后，即可撤除本次中断请求信号。

（2）跳变触发方式时，在 $\overline{\text{INT0}}$ 或 $\overline{\text{INT1}}$ 引脚上的高电平和低电平保持时间必须不少于 1 个机器周期。由于 CPU 响应中断时，自动把中断请求标志清零，撤除了本次中断请求，因此，在设计时一般不考虑中断请求信号的撤除问题。

2. 中断响应时间

中断响应时间是指从中断请求标志位置 1 到 CPU 开始执行中断服务程序的第一条指令所持续的时间。CPU 并非每时每刻都对每一个中断请求予以响应；另外，不同的中断请求其响应时间也是不同的，因此，中断响应时间形成的过程较为复杂。本节以外部事件中断为例，说明 MCS-51 单片机的中断响应所需的时间，以便在程序设计时能合理地估算程序的运行时间，进一步提升程序的运行效率。

1）中断请求立即被 CPU 响应

CPU 在每个机器周期采样 $\overline{\text{INT0}}$ 和 $\overline{\text{INT1}}$ 引脚上的电平，如果中断请求有效，则把相应中断请求标志位置位，然后在下一个机器周期再查询中断请求标志位的状态，这就意味着中断请求信号的低电平至少应维持一个机器周期。这时，如果满足中断响应条件，则 CPU 响应中断请求，在下一个机器周期执行硬件调用，使程序转入中断入口地址处。该调用指令执行时间是两个机器周期，因此，外部中断响应时间至少需要三个机器周期，这是最短的中断响应时间。

2）中断响应条件不满足，中断请求没有被 CPU 立即响应

虽然中断触发，中断请求标志位被置 1，但是由于中断响应的 3 个条件不能满足，响应被阻断，中断响应时间被延长，假如以下几种情形。

（1）如果此时一个相同优先级或高优先级的中断正在处理执行，则附加的等待时间取

决于正在执行的中断处理程序的执行时间。

（2）如果正在执行的一条指令还没有到最后一个机器周期，则附加的等待时间为 1～3 个机器周期（因为指令最长的执行时间为 4 个机器周期：乘法、除法指令）。

（3）如果正在执行的指令是 RETI 指令或访问 IE、IP 的指令，则附加的等待时间在 5 个机器周期之内（最多用一个机器周期完成当前指令，再加上最多 4 个机器周期完成下一条指令）。

综上所述，如果系统中只有一个中断源，中断响应时间为 3～8 个机器周期。如果有多个中断或多级中断处理嵌套时，中断响应时间与相同优先级或高优先级的中断处理程序的执行时间有关。

5.3.2 外部事件中断源的应用

1. 外部事件中断方法的选择

单片机应用系统需要处理大量的输入信号，那么，对这些信号处理时是采用查询方式还是采用中断处理方式呢？所谓查询就是让计算机不断地对输入信号进行检测、判断和处理。针对这个问题，应该从系统本身的要求出发，考虑以下两方面的因素。

（1）考察应用系统对输入信号状态变化的反应快慢程度，如果应用系统的最大响应时间较小，那么，对输入信号最好采用中断方法。不管应用系统对输入信号的查询速度有多么迅速，其平均响应时间一般大于中断方法的响应时间。

（2）考察输入信号状态变化的最小持续时间；如果信号触发频率接近指令周期频率的 1/10，那么，最好采用中断方法；否则，查询时需要采用较小的查询循环。

除此之外，应用系统中有多个输入信号，每一个输入都要求用中断方法处理。对于这种情况，要么采用中断源共享的方法，使它们共享 MCS-51 单片机仅有的两个中断源，要么从整个系统的设计要求出发，把其中的一些相对不重要的输入采用查询方式处理。

2. 外部事件中断的初始化及中断处理程序编程步骤

1）中断系统初始化

在主程序对中断系统初始化时，需完成以下设置。

（1）设置外部事件中断请求信号的触发方式。

① 电平触发：$IT_x=0$，$x=0,1$。由于单片机复位后，IT1 和 IT0 被清零，默认为电平触发方式，因此，有时可以在程序中被省略。

② 跳变触发方式：$IT_z=1$，$x=0,1$。

（2）开放 CPU 中断：EA＝1。

（3）设置外部事件中断允许控制位：EX0＝1 或 EX1＝1。

（4）如果有中断嵌套处理，设置中断源的优先级。

① 设置外部事件中断源为高优先级：PX0＝1 或 PX1＝1。

② 设置外部事件中断源为低优先级：PX0＝0 或 PX1＝0。由于单片机复位后，IP 被清零，默认所有中断源为低优先级，因此，有时在程序中被省略。

在主程序中，对中断系统初始化时，也可以采用下列形式设置中断允许控制位和中断源的优先级：

```
IE = 0x81;          //开中断，开外部事件 0 中断
IP = 0x01;          //设置外部事件 0 中断源为高优先级
```

2）中断处理程序编程

中断处理程序是根据处理外部事件的具体要求而设计的程序。C51 语言有专门的中断函数。程序设计时，可以参考图 5.10 的结构。

3. 外部事件采用跳变触发方式请求中断

例 5.3　单片机应用系统如图 5.11 所示，P1 口为输出口，外接 8 个指示灯 L0～L7。系统工作时，指示灯 L0～L7 逐个被点亮。在逐个点亮 L0～L7 的过程中，当开关 K 被扳动时，则暂停逐个点亮的操作，L0～L7 全部点亮并闪烁 10 次。闪烁完成后，从暂停前的灯位开始继续逐个点亮的操作。

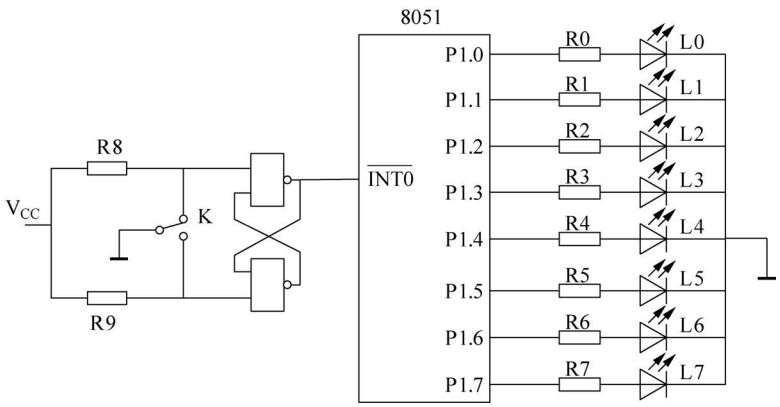


图 5.11　例 4.1 单片机应用系统

为了实现开关 K 扳动 1 次仅在 $\overline{\text{INT0}}$ 引脚产生 1 次高电平到低电平的负跳变，用 R-S 触发器设计了消除开关触点机械抖动的电路。K 每扳动 1 次，产生 1 次中断请求，根据题意，设计的程序如下：

（1）主程序：

```
#include <reg51.h>                                    //头文件
#define DataPort P1                                   //定义端口 P1
unsigned char code   dofly_DuanMa[8] = {0x01,0x03,0x07,0x0f,
                                0x1f,0x3f,0x7f,0xff};  //逐个点亮显示码
unsigned char code dofly_10[2] = {0xff,0x00};         //全亮、全灭显示码
void Delay(unsigned int t);                           //延时函数声明
void Display(unsigned char Num);
/* ------------------------------------------------
                      主函数
------------------------------------------------ */
void Main(void)
{
    unsigned int j;
    EX0 = 1;                                          //允许外部中断 INT0 中断
    IT0 = 1;                                          //设置 INT0 中断源为跳变触发
    EA = 1;                                           //开放 CPU 中断
    while(1)
      {                                               //正常工作状态，逐个点亮 LED
       j++;
       if(j == 8)                                     //显示位置
        {
```

```
            j = 0;                                  //显示位置复位
          }
       Display(j);                                  //显示
      }
}
```

（2）延时子程序：

```
/* ------------------------------------------------
延时函数,含有输入参数 unsigned int t,无返回值
unsigned int 是定义无符号整形变量,其值的范围是 0～65535
------------------------------------------------ */
void Delay(unsigned int t)
{
  while(t--);
}
```

（3）显示子程序：

```
/* ------------------------------------------------
  显示函数,Num 表示显示位置
------------------------------------------------ */
void Display(unsigned char Num)
{
    DataPort = 0;                                   //全灭,消隐,防止有交替重影
    DataPort = dofly_DuanMa[Num];                   //取显示码并显示
    Delay(50000);                                   //状态维持
}
```

（4）中断处理程序：

```
/*************************************************************************
 * 名称 : Outside_Int1()
 * 功能 : 外部中断 0 的中断处理
 * 输入 : 无
 * 输出 : 无
*************************************************************************/
void Outside_Int1(void) interrupt 0
{
   unsigned char i;
     Delay(2);
      for(i = 0;i < 10;i++)
       {  DataPort = dofly_10[0];                   //取显示码,全亮
          Delay(50000);                             //状态维持
          DataPort = dofly_10[1];                   //取显示码,全灭
          Delay(50000);                             //状态维持
       }
  Delay(30);
}
```

4. 外部事件采用电平触发方式请求中断

例 5.4 如图 5.12 所示，P1.0～P1.3 为输出，外接指示灯 L0～L3，P1.7～P1.4 为输入，外接开关 K0～K3，欲采用外部中断控制方式实现按钮开关 K0～K3 分别控制指示灯 L0～L3，按钮开关 S 每闭合一次，外部中断触发一次，程序改变一次指示灯的显示状态。

在图 5.12 中，当按键 S 每按动一次，在 D 触发器的 CLK 端产生一个正脉冲，D 触发器

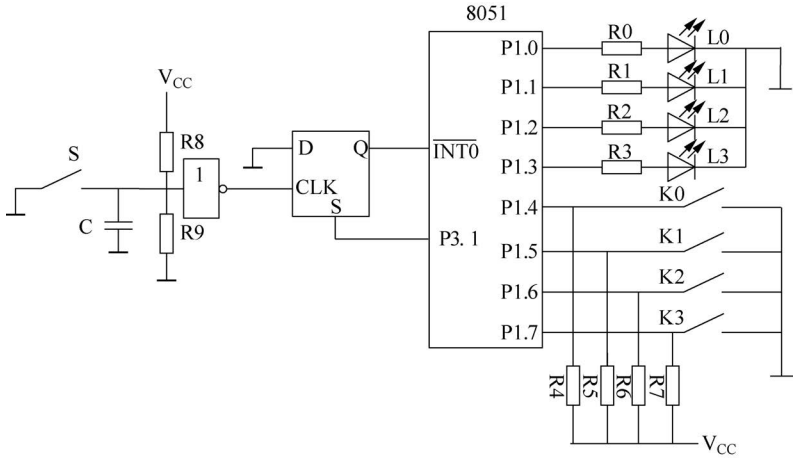


图 5.12　例 5.4 应用系统图

翻转，Q 端输出低电平并锁存，产生中断请求信号。由于电平触发时，CPU 响应中断后，中断标志 IE0 并不能自动清除，需要用外部手段撤除中断请求信号，因此，在进入中断处理程序后，应通过软件撤除本次的中断请求信号，程序中 P3.1 为 0 时，D 触发器的置位端 S 为 1，使 D 触发器的 Q 输出高电平，撤除了本次中断请求。这样，下一次按动 S 时，可以触发新的中断。根据题意，程序如下：

（1）主程序：

```
#include <reg51.h>
sbit KEY0 = P1^4;                    //定义按键 K0 输入端口
sbit KEY1 = P1^5;                    //定义按键 K1 输入端口
sbit KEY2 = P1^6;                    //定义按键 K2 输入端口
sbit KEY3 = P1^7;                    //定义按键 K3 输入端口
sbit LED0 = P1^0;                    //定义 led0 输出端口
sbit LED1 = P1^1;                    //定义 led1 输出端口
sbit LED2 = P1^2;                    //定义 led2 输出端口
sbit LED3 = P1^3;                    //定义 led3 输出端口
sbit SetQ1 = P3^1;                   //定义 D 触发器的 S 端,清除中断请求
sbit SetQ2 = P3^2;                   //定义 INT0 端口
                                     /* 主函数 */
void Main(void)
{
    EX0 = 1;                         //允许外部中断 INT0 中断
    IT0 = 0;                         //设置 INT0 中断源为电平触发方式
    EA = 1;                          //开放 CPU 中断
    PX0 = 0;                         //设置 INT0 中断源为低优先级
    SetQ1 = 1;
    SetQ2 = 1;
while(1)                             //主循环
  {}
}
```

（2）中断处理程序：

```
/* 外部中断 INT0 的中断处理 */
void Outside_Int1(void) interrupt 0
```

```
{
    SetQ1 = 0;                          //清除外部中断请求信号
    LED0 = !KEY0;                       //读入 K0 状态,控制 LED0
    LED1 = !KEY1;                       //读入 K1 状态,控制 LED1
    LED2 = !KEY2;                       //读入 K2 状态,控制 LED2
    LED3 = !KEY3;                       //读入 K3 状态,控制 LED3
    SetQ1 = 1;                          //复位 D 触发器
}
```

对于跳变触发的外部中断 $\overline{\text{INT0}}$ 或 $\overline{\text{INT1}}$，CPU 在响应中断后由硬件自动清除其中断标志位 IE0 或 IE1，无须采取其他措施。而采用电平触发时，中断请求撤除方法较复杂。因为电平触发时，CPU 在响应中断后，硬件不会自动清除 IE0 或 IE1，也不能用软件将其清除，它是由 $\overline{\text{INT0}}$ 或 $\overline{\text{INT1}}$ 引脚上外部中断请求信号直接决定的。所以，在 CPU 响应中断后，必须立即撤除 $\overline{\text{INT0}}$ 或 $\overline{\text{INT1}}$ 引脚上的低电平。否则，就会引起重复中断。

在图 5.12 中，用开关 S 闭合的事件触发外部中断，当 S 闭合时，在触发器的输入端 CLK 产生一个正脉冲，请求信号不直接加在 $\overline{\text{INT0}}$ 引脚上，而是加在 D 触发器的 CLK 端。由于 D 端接地，当外部中断请求的正脉冲信号出现在 CLK 端时，Q 端输出为 0 并锁存，$\overline{\text{INT0}}$ 为低，外部中断向单片机发出中断请求。CPU 响应中断请求后，进行中断处理，在中断处理程序中用软件来撤除外部中断请求。“SetQ1＝0”使 P3.1 为 0，使 D 触发器输出置位，Q 端输出为 1，从而撤除中断请求。而“SetQ1＝1”使 P3.1 变为 1，其目的是使 D 触发器的输出 Q 受 CLK 控制，新的外部中断请求信号能够向 CPU 再次申请中断。语句“SetQ＝1”是必不可少的；否则，将无法再次形成新的外部中断。

5. 同时使用两个外部中断源

例 5.5 单片机应用系统如图 5.13 所示，P1 口为输出口，外接 8 个指示灯 L0～L7。要求实现下面的要求：

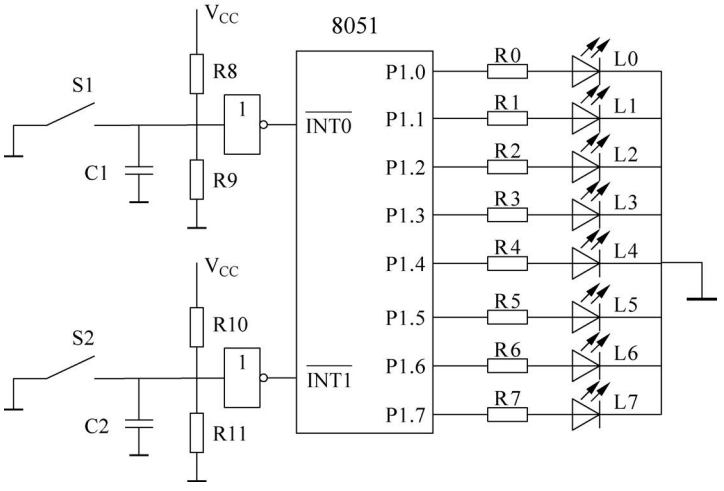


图 5.13 同时使用两个外部中断源的单片机应用系统

（1）系统工作时，指示灯 L0～L7 以 3 个指示灯为一组循环显示。

（2）当 S1 按下时，暂时中断 3 灯循环方式，熄灭全部指示灯，从指示灯 L0 开始逐个点亮并保持，直至 L0～L7 全部点亮，然后熄灭，重复上述过程 5 次后退出，继续 3 灯循环显示

模式。

（3）当 S2 按下时，暂时中断 3 灯循环方式，全部指示灯 L0～L7 闪烁显示 10 次后退出，继续 3 灯循环显示模式。

在本例中，S1 和 S2 具有相同的中断优先级，当两个按钮同时按下时，优先响应 S1 的请求；正在处理其中一个时，不会响应另外一个。根据题目要求，同时使用两个外部中断源的程序如下：

（1）主程序：

```
#include <reg51.h>
#define DataPort P1                                    //定义 P1 口代号为 DataPort
unsigned char code dofly_DuanMa[8] = {0x07,0x0e,0x1c,0x38,
                  0x70,0xe0,0xc1,0x83};                //正常工作方式时的 LED 显示控制码
unsigned char code dofly_S1[2] = {0xff,0x00};          //工作方式 1 的显示控制码
unsigned char code dofly_S0[9] = {0x00,0x01,0x03,0x07,
                 0x0f,0x1f,0x3f,0x7f,0xff};            //工作方式 2 的显示控制码
void Delay(unsigned int t);                            //函数声明
void Display(unsigned char Num);
/*   主函数 */

void Main(void)
{
   unsigned int j = 0;
   EX0 = 1;                                            //允许外部中断 INT0 中断
   EX1 = 1;                                            //允许外部中断 INT1 中断
   IT0 = 1;                                            //设置 INT0 中断源为跳变触发方式
   IT1 = 1;                                            //设置 INT1 中断源为跳变触发方式
   EA = 1;                                             //开放 CPU 中断
   IP = 0;
   while(1)                                            //正常工作方式
   {
      j++;
      if(j == 8)                                       //LED 显示位置
      {
         j = 0;                                        //复位 LED 显示位置
      }
         Display(j);                                   //显示
    }
}
```

（2）延时子程序：

```
/* 延时函数,t——延时参数 */

void Delay(unsigned int t)
{
    unsigned char t0;
    while(t--)
    {
        for(t0 = 0;t0 < 255;t0++);
    }
}
```

（3）显示子程序：

```
/* LED 显示函数,Num 为显示位置 */
void Display(unsigned char Num)
{
      DataPort = 0;                                    //消隐
      DataPort = dofly_DuanMa[Num];                    //LED 显示
      Delay(100);                                      //LED 显示状态维持
}
```

(4) $\overline{\text{INT0}}$ 中断源的中断处理程序：

```
/* 外部中断 INT0 的中断处理——工作方式 1 */
void Outside_Int1(void) interrupt 0
{
   unsigned char i;
   unsigned char j;
   for(j = 0;j < 5;j++)                                //8 个 LED 逐个点亮,循环 5 次
       {
         for(i = 0;i < 9;i++)
           {
                DataPort = dofly_S0[i];                //取 LED 显示码显示
                Delay(100);                            //LED 显示状态维持
           }
       }
}
```

(5) $\overline{\text{INT1}}$ 中断源的中断处理程序：

```
/* 外部中断 INT1 的中断处理——工作方式 2 */
void Outside_Int2(void) interrupt 2
{
   unsigned char i;
   for(i = 0;i < 10;i++)       ;                       //亮、暗交替 10 次
   {
     DataPort = dofly_S1[0];                           //全亮
     Delay(100);                                       //
     DataPort = dofly_S1[1];                           //全灭
     Delay(100);                                       //
   }
}
```

6. 两级中断嵌套处理

例 5.6 单片机应用系统如图 5.13 所示，P1 口为输出口，外接 8 个指示灯 L0～L7。要求实现如下要求。

(1) 系统工作时，L0～L7 以 3 个指示灯为一组循环显示。

(2) 当 S1 按下时，暂时中断 3 灯循环方式，熄灭全部指示灯，从 L0 开始逐个点亮并保持，直至 L0～L7 全部点亮，然后熄灭，重复上述过程 5 次后退出，继续 3 灯循环显示模式。

(3) 不论在(1)和(2)哪种运行方式下，只要 S2 按下，暂时中断当前的显示方式，全部指示灯闪烁显示 10 次后退出，继续运行以前的显示方式。

在本例中，设置按钮 S2 产生的中断请求为高优先级的，任何时候只要 S2 按下，必须立即响应这个中断请求，此时需要设置 $\overline{\text{INT1}}$ 中断源的优先级为高优先级，即 PX1＝1。因此，将例 5.5 程序中的 IP＝0 语句改为 IP＝0x04，其他代码不变，就可实现两级中断嵌套处理的要求。

5.3.3　外部事件中断源的扩展

如果系统中有多个外部事件，可以采用中断源共享的方法，使多个中断源共同使用MCS-51单片机的两个外部事件中断源。

例 5.7　电梯是大型建筑不可缺少的运输工具。在运行过程中，有以下几种情况需要控制系统立即处理。

(1) 当测速传感器检测到电梯超过额定运行速度时，控制系统应立即切断控制回路电源。

(2) 当电梯运行到接近底层和顶层时，安装在电梯轿箱上的撞弓装置撞击到强迫减速开关时，控制系统应强制电梯减速运行。

(3) 强制减速后仍然不能停车，当上限或下限限位开关有效时，应切断整个系统的电源。

(4) 当发生意外情况时，按下紧急停止按钮，电梯紧急制动停车。

(5) 当电路欠压时或电网电压波动时，为了避免控制回路误动作，应切断控制回路电源。

(6) 曳引电机过载时，应进行过载保护，切断控制回路电源。

因为电梯是载人和运货的工具，从(1)～(6)的紧急程度和事故后果来看，(3)的紧急程度最高，(2)次之，其他情况的紧急程度排列顺序依次为：(1)，(5)，(6)，(4)。若采用中断方式处理这些事件，必须进行中断源的扩展以共享单片机的两个外部中断源，中断源扩展电路如图5.14所示。图5.14中，所有中断采用电平触发形式，中断请求信号一直保持，以确保中断处理的实现，除非控制系统自行撤除中断请求信号，中断处理才会终止。上/下限限位开关闭合触发的事件中断由 $\overline{\text{INT0}}$ 引入，该中断源的优先级最高，不论在何种情况下，只要上/下限限位开关闭合，CPU立即响应。其余几个中断源由 $\overline{\text{INT1}}$ 引入，通过与门把它们综合，共享MCS-51单片机的外部事件中断 $\overline{\text{INT1}}$，它们的状态分别由P1.0～P1.4反馈到单片机，通过程序查询它们的状态来判别哪一个事件触发了中断，然后再进行相应的中断处理。程序查询的先后顺序由这些事件的紧急程度(即优先级)确定，其中优先级最高的，程序首先查询，优先级最低的，程序最后查询。

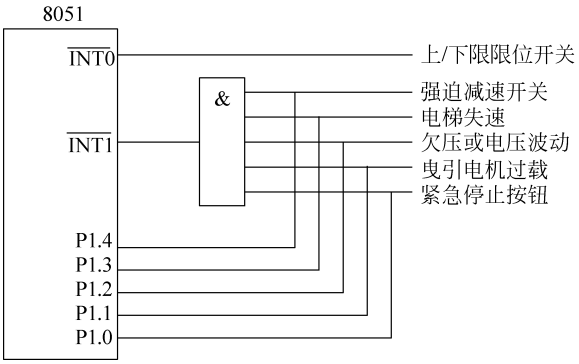


图 5.14　外部事件中断源的扩展电路

根据上述要求和分析，给出电梯控制系统多中断处理的主要程序的实现方法如下：

```
EX0 = 1;                                    //
EX1 = 1;                                    //
IT0 = 1;                                    //设置 INT0 中断源为跳变触发方式
IT1 = 1;                                    //设置 INT1 中断源为跳变触发方式
EA = 1;                                     //
```

```
IP = 0x04;                                          //
```

（1）主程序：

```
    #include <reg51.h>
          sbit   P14 = P1^4
          sbit   P13 = P1^3
          sbit   P12 = P1^2
          sbit   P11 = P1^1
          sbit   P10 = P1^0
/* ---------------------------- 主函数 ---------------------------------- */
  void main(void)
     {
          IT0  = 0;                                 //设置 INT0 中断源为电平触发方式
          IT1  = 0;                                 //设置 INT1 中断源为电平触发方式
          EA  =  1;                                 //开放 CPU 中断
          EX0  =  1;                                //允许外部中断 INT0 中断
          EX1  =  1;                                //允许外部中断 INT1 中断
          PX0  =  1;                                //设置 INT0 中断源优先级为高优先级
          PX1  = 0;                                 //设置 INT1 中断源优先级为高优先级
          while(1)                                  //正常工作方式
           {
               主处理程序;
           }
     }
```

（2）$\overline{\text{INT0}}$ 中断处理程序：

```
/* 外部中断 INT0 的中断处理 */
void Outside_Int1(void) interrupt 0
{
    切断系统电源控制程序;
}
```

（3）$\overline{\text{INT1}}$ 中断处理程序：

```
/* 外部中断 INT0 的中断处理 */
void Outside_Int2(void) interrupt 2
 {
    if(P14 == 0){转强制换速处理程序;}
    else if(P13 == 0){失速处理程序;}
    else if(P12 == 0){电源故障处理程序;}
    else if(P11 == 0){电机过载处理程序;}
    else if(P10 == 0){急停按钮处理程序;}
 }
```

5.4 本章小结

在 CPU 执行程序的过程中，由于某种原因要求 CPU 暂时停止正在执行的程序，转去执行相应的处理程序，待处理结束后，再返回到暂停处继续执行，这个过程称为中断。

CPU 响应中断请求调用中断处理程序的过程与主程序调用子程序的主要区别在于：子程序调用是用户设计程序时事先安排好的，采用子程序调用指令实现的；而中断事件发生是随机的，调用中断处理程序的过程是由硬件自动完成的。

MCS-51 单片机具有 5 个中断源：两个外部事件中断($\overline{\text{INT0}}$ 和 $\overline{\text{INT1}}$)、两个定时器/计数器中断和一个串行口中断。它们可以设为两个中断优先级，实现两级中断嵌套。CPU 对中断采用两级管理，用户可以根据需要来设定 CPU 是否开放中断，而且每个中断源都可以独立地设定为允许和禁止中断请求。另外，中断源的优先级也可以独立地设定为高或低优先级。

MCS-51 单片机的 CPU 响应中断请求是有条件的，如果此时不存在下列 3 种情形。

(1) CPU 正在处理相同优先级或高优先级的中断。

(2) 当前的机器周期不是指令的最后一个机器周期。

(3) 正在执行的指令是中断返回 RETI 或者是对寄存器 IE 或 IP 的写入操作指令。

那么，CPU 立即响应这个中断请求，直接转移到相应的中断处理程序入口地址处，进行中断处理。

MCS-51 单片机的外部事件中断的请求(触发)信号由 $\overline{\text{INT}x}$($\overline{\text{INT0}}$ 或 $\overline{\text{INT1}}$)引脚引入单片机的中断系统，中断触发方式可以为电平触发方式或跳变触发方式，通过编程设置 TCON 中的控制位 IT0 和 IT1 实现。

若外部事件中断为电平触发方式时，$\overline{\text{INT}x}$ 引脚的低电平必须保持到 CPU 响应该中断时为止，并且必须在本次中断处理返回以前变为高电平，以撤除本次中断请求信号；否则，如果中断请求信号没有撤除，中断返回后又再次响应该中断，CPU 将陷入无休止的中断响应和中断处理当中。外部事件中断为跳变触发方式时，在 $\overline{\text{INT}x}$ 引脚上出现负跳变，则硬件把中断请求标志 IE0 或 IE1 置位，发出中断请求。CPU 响应中断时，自动把中断请求标志清零，撤除本次中断请求。

中断系统应用时，必须对中断系统初始化，设置中断请求信号的触发方式、中断允许控制位、中断源的优先级等。中断处理程序是根据处理外部事件的具体要求而设计的程序，实现中断嵌套处理时，虽然高优先级中断源可以中断低优先级的中断处理，但不能干扰低优先级的现场保护和恢复。

5.5　复习思考题

一、选择题

1. 在计算机系统中，中断系统是指(　　)。

A. 实现中断处理的硬件电路　　B. 实现中断处理的程序

C. A 和 B　　D. 触发中断的事件

2. 中断处理程序是(　　)。

A. 用户编写的处理事件的应用程序　　B. CPU 内部嵌入的硬件程序

C. 被中断停止执行的程序　　D. 查询中断是否触发的程序

3. MCS-51 单片机的中断系统的中断源有(　　)。

A. 5 个　　B. 6 个　　C. 2 个　　D. 3 个

4. MCS-51 单片机禁止和允许中断源中断使用的寄存器是(　　)。

A. TCON　　B. PSW　　C. IP　　D. IE

5. MCS-51 单片机设定中断源优先级使用的寄存器是(　　)。

A. TCON　　B. PSW　　C. IP　　D. IE

6. 外部事件中断源 $\overline{\text{INT0}}$ 请求中断时，登记该中断请求标志的寄存器是（　　）。

A. SCON　　B. PSW　　C. TCON　　D. EX0

7. 下面哪种设置，可以使中断源 $\overline{\text{INT1}}$ 以跳变方式触发中断？（　　）

A. IT0＝1　　B. IT0＝0　　C. IT1＝1　　D. IT1＝0

8. 单片机的串行口接收到一帧数据后，登记标志位 TI＝1 的寄存器是（　　）。

A. SCON　　B. PSW　　C. TCON　　D. SBUF

9. 在电平触发方式下，中断系统把标志位 IE1 置 1 的前提是：在 $\overline{\text{INT1}}$ 引脚上采集到的有效信号是（　　）。

A. 高电平　　B. 低电平

C. 高电平变为低电平　　D. 低电平变为高电平

10. 在跳变触发方式下，中断系统把标志位 IE0 置 1 的前提是：在 $\overline{\text{INT0}}$ 引脚上采集到的有效信号是（　　）。

A. 高电平　　B. 低电平

C. 高电平变为低电平　　D. 低电平变为高电平

11. MCS-51 单片机定时器/计数器 T0 溢出时，被置 1 的标志位是（　　）。

A. IE0　　B. RI　　C. TF0　　D. TF1

12. 下列中断源请求中断被响应后，其中断请求标志位被自动清零的是（　　）。

A. $\overline{\text{INT0}}$ 以电平方式触发中断　　B. $\overline{\text{INT0}}$ 以跳变方式触发中断

C. 串行口发送完一帧数据　　D. 定时器/计数器 T0 计数溢出

13. 在 MCS-51 单片机中，CPU 响应中断后，需要外电路实现中断请求清除的是（　　）。

A. 定时器/计数器 T1 溢出触发的中断

B. $\overline{\text{INT0}}$ 以跳变方式触发的中断

C. 串行口接收到一帧数据触发的中断

D. $\overline{\text{INT1}}$ 以电平方式触发的中断

14. 在 MCS-51 单片机中，CPU 响应中断后，需要用软件清除中断请求标志的是（　　）。

A. 定时器/计数器 T1 溢出触发的中断

B. $\overline{\text{INT0}}$ 以跳变方式触发的中断

C. 串行口接收到一帧数据触发的中断

D. $\overline{\text{INT1}}$ 以电平方式触发的中断

15. 如果 IP 内容为 0010100B，则中断源优先级最高的是（　　）。

A. $\overline{\text{INT0}}$　　B. 串行口

C. 定时器/计数器 T1　　D. $\overline{\text{INT1}}$

16. 外部事件 1 对应的中断处理程序入口地址为（　　）。

A. 0003H　　B. 000BH　　C. 0013H　　D. 001BH

17. 单片机应用系统使用了外部事件中断源 $\overline{\text{INT0}}$，拟采用跳变触发方式，下面设置指令不正确的是（　　）。

A. IT0＝0　　B. IT0＝1　　C. TCON＝0x11　　D. TCON＝0x01

18. 在单片机应用系统中，拟用外部事件中断源 $\overline{\text{INT0}}$ 和定时器/计数器溢出 T1 中断源，下面设置不正确的是（　　）。

A. IE＝0x89　　B. IP＝0x89

C. IE＝0x9b　　D. EA＝1，EX0＝1，ET1＝1；

19. 应用系统使用了3个中断源：$\overline{\text{INT0}}$、定时器/计数器T1和串行口，优先级顺序为定时器/计数器T1、串行口、$\overline{\text{INT0}}$，下面设置正确的是(　　)。

A. IP＝0x11　　B. IP＝0x18　　C. IE＝0x01　　D. IP＝0x08

20. 初始化时，禁止CPU中断和$\overline{\text{INT1}}$程序是：

A. EA＝0；　　B. IE＝0x84；　　C. IP＝0x81；　　D. PX1＝0；

二、思考题

1. 在计算机系统中，什么是中断、中断源、断点和中断处理？

2. 在计算机系统中，中断处理和子程序调用有什么不同？

3. MCS-51单片机提供了哪几种中断源？在中断管理上如何控制？各个中断源中断优先级的高低如何确定？

4. MCS-51单片机响应中断的条件是什么？

5. MCS-51单片机的CPU响应多个中断请求时，如何处理多个中断同时请求的问题？

6. MCS-51单片机是如何分配中断处理程序入口地址的？

7. 简述MCS-51单片机的中断响应过程。

8. 在应用系统中只包含一个优先级的中断处理时，给出中断处理程序的一般结构。

9. 如果应用系统包含了两个优先级的中断处理，高、低优先级的中断处理程序结构有什么不同？

10. 对于输入信号检测来说，中断处理方式和程序查询方式有什么不同？

三、程序设计

1. 在图5.11电路中，通常情况下，L0～L7依次循环显示，每扳动一次开关K，L0～L7以两灯为1组循环显示1次。用中断方式实现上述要求。

2. 如图5.15所示，P1.0～P1.3为输出，外接指示灯L0～L3，P1.7～P1.4为输入，外接开关K0～K3，欲采用外部中断控制方式实现按开关K0～K3闭合状态分别控制指示灯L0～L3的状态，外部中断每触发一次，程序改变1次指示灯的显示状态。要求用跳变触发方式。

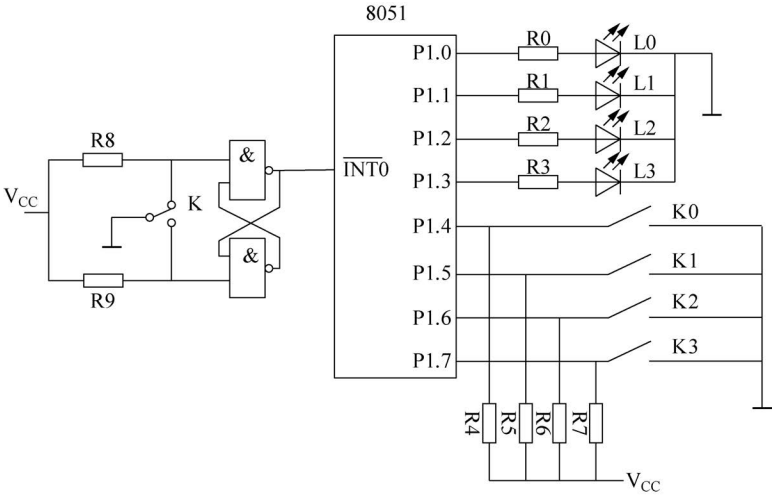


图5.15　习题2应用系统原理图

3. 如图 5.16 为一个应用系统，单片机通过 P1 口与智能传感器相连，STB 为传感器输出的选通信号，传感器每从 DB 输出一个 7 位二进制数据后（最高位是 0），就从 STB 输出一个负脉冲，8051 单片机读取的数据存储在内部 RAM 中，如果读取的数据超过 7 位（最高位为 1）的次数超过 20 次，则终止从传感器读数。采用中断方式实现数据接收功能。

4. 路灯控制器如图 5.17 所示，夜晚路灯 L1 自动启动，白天路灯 L1 自动熄灭。采用中断方式实现路灯的自动控制。图 5.17 中，VL 为光敏三极管，有光照射时，VL 导通，无光照射时，VL 截止。

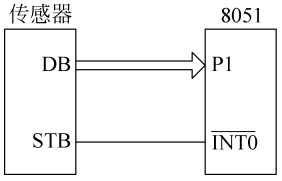


图 5.16　习题 3 应用系统原理图

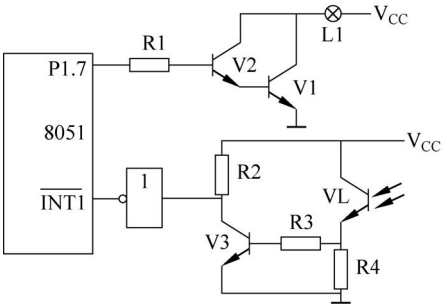


图 5.17　习题 4 的路灯控制器图

5. 图 5.18 为单片机应用系统，4 个外部扩展中断源 EXINT0～EXINT3 共享外部事件中断 $\overline{\text{INT0}}$，当其中有一个或几个出现高电平时向单片机发出中断请求。设它们的优先级顺序为 EXINT0→EXINT3，中断源 EXINT0～EXINT3 的中断处理程序分别为 PREX0，PREX1，PREX2 和 PREX3，请用中断方式实现上述要求。

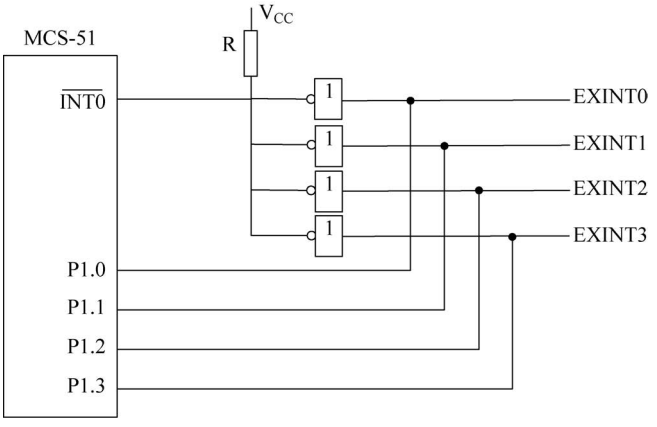


图 5.18　习题 5 的单片机应用系统原理图

6. 一个控制系统中有 5 台外围设备需要集中监控，出现故障时需要立即处理，设备 1～5 的故障状态信号分别为 EX1～EX5，其中，设备 1 和设备 2 的故障危害性大，设备 3～设备 5 的故障为一般性故障，危害较小。请用 MCS-51 单片机中断方式实现上述设备的监控，设计电路并编程，设相应的中断处理子程序为 Ex1Pro～Ex5Pro。

7. 单片机应用系统如图 5.19 所示，P1 口外接 8 个指示灯 L0～L7。要求实现下面的要求。

（1）一般情况下，指示灯 L0～L7 以 100ms 的间隔闪烁。

（2）S0，S1，S2 为 3 种显示模式，当 S0，S1，S2 被按下时，暂时中断闪烁方式，熄灭全部指示灯，进入相应的显示模式：

① 当按下 S0 时，从指示灯 L0 开始逐个点亮并保持 200ms，直至 L0～L7 全部点亮，然后熄灭，重复上述过程 10 次后退出。

② 当按下 S1 时，从指示灯 L0 开始，每个点亮 200ms 后熄灭，重复上述过程 10 次后退出。

③ 当按下 S2 时，从指示灯 L7 开始以 3 个为一组点亮并保持，直至 L7～L0 全部点亮，然后熄灭，重复上述过程 10 次后退出。

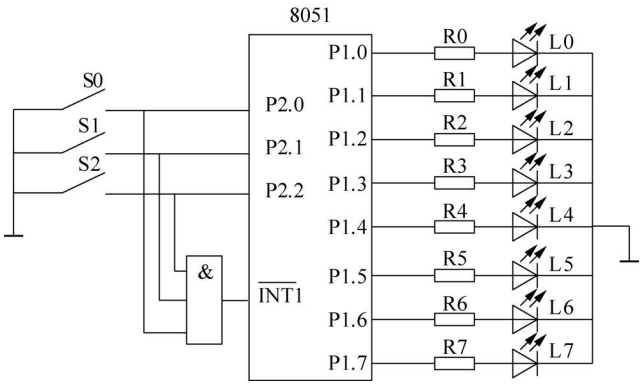


图 5.19　习题 7 的单片机应用系统原理图

8. 在图 5.20 单片机应用系统中，A，B 两路检测信号分别从 P3.2（$\overline{INT0}$）和 P3.3（$\overline{INT1}$）引入单片机，通常情况下，当 A，B 为高电平时，表示系统工作正常，指示灯 L1 亮；当 A 出现低电平时，指示灯 L1 灭，L2 以 500ms 的间隔闪烁，除非 A 再次变为高电平，系统恢复正常。无论在什么情况下，只要 B 出现低电平，关闭指示灯 L1，L2 以 200ms 的间隔闪烁，同时蜂鸣器 BUZ 以 200ms 的间隔鸣叫，除非 B 再次变为高电平，系统恢复正常。采用中断方式实现以上监控功能。

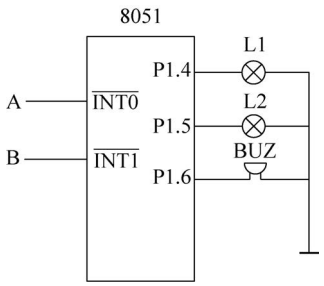


图 5.20　单片机应用系统原理图