第 5 章　单片机的内部资源及应用

51 单片机有丰富的内部资源可供用户使用，主要是并行口、中断系统、定时/计数器和串行口。这些内部资源均可用 C51 语言对其控制。本章主要包括以下内容：

（1）介绍 AT89S51 单片机的并行 I/O 口的功能及结构，并以单片机控制连接在 I/O 口上的发光二极管闪烁为实例，介绍并行 I/O 口的操作方法。

（2）介绍 AT89S51 单片机的中断系统的功能及结构，并以可控流水灯为实例，介绍中断的操作方法。

（3）单片机应用于检测、控制及智能仪器等领域时，常需要实时时钟来实现定时或延时控制，也常需要计数器对外界事件进行计数。本章将介绍 AT89S51 单片机的定时/计数器的功能及结构，并以 1 s 流水灯为实例，介绍定时/计数器的操作方法。

（4）AT89S51 单片机内部除含有 4 个并行 I/O 接口外，还有一个串行通信 I/O 口，通过该串行口可以实现与其他计算机系统的串行通信。本章将通过实例介绍串行口的应用。

【教学导航】

教	知识重点	1. 单片机并行 I/O 口的结构、功能及操作方法； 2. 单片机中断系统、中断相关寄存器的功能及中断程序的编写； 3. 单片机定时器结构、工作方式及应用； 4. 串行口的结构、工作方式及波特率的设置； 5. 单片机之间、单片机与 PC 机之间的串行通信
	知识难点	1. 并行 I/O 口的结构和操作； 2. 中断的概念及中断程序的编写； 3. 串行口的结构、工作方式
	推荐教学方式	从具体实例入手，掌握单片机 I/O 口的操作方法；通过单片机流水灯的设计掌握定时器、中断系统的原理；通过单片机双机通信及单片机与 PC 机通信的设计，掌握单片机通信技术
	建议学时	12～16 学时
学	推荐学习方法	在理论学习中加强实践操作，以加深理解单片机 I/O 口的结构及操作方法；体验定时器、中断系统的编程技巧；掌握单片机双机通信的原理
	必须掌握的理论知识	1. 并行 I/O 口的结构和功能； 2. 单片机中断系统的组成； 3. 单片机定时器的功能； 4. 单片机中断和定时器的程序设计方法；

		5. 串行口的结构、工作方式及波特率的设置
	必须掌握的技能	1. 并行 I/O 口的 C51 编程； 2. 单片机中断和定时器程序的调试方法； 3. 单片机之间、单片机与 PC 机之间的串行通信

5.1 并行 I/O 端口

AT89S51 单片机共有 4 个 8 位的并行 I/O 口，分别用 P0、P1、P2、P3 表示。每个端口都包含 1 个锁存器，1 个输出驱动器和输入缓冲器。实际上它们已被归入专用寄存器之列，并且具有字节寻址和位寻址功能。

在访问片外扩展存储器时，低 8 位地址和数据由 P0 口分时传送，高 8 位地址由 P2 口传送。在无片外扩展存储器的系统中，这 4 个端口的每一位均可作为双向的 I/O 端口使用。

AT89S51 单片机的 4 个 I/O 端口都是 8 位双向端口，这些端口在结构和特性上是基本相同的，但又各具特点，以下分别介绍。

5.1.1 P1 口

P1 口是一个 8 位、可位寻址的准双向口，用作通用 I/O 口。

1. P1 口的结构

P1 口的电路结构如图 5.1 所示，电路中包含有 1 个数据输出锁存器、2 个三态数据输入缓冲器、1 个数据输出的驱动电路。其中，输出驱动电路由内部上拉电阻与场效应管共同组成。

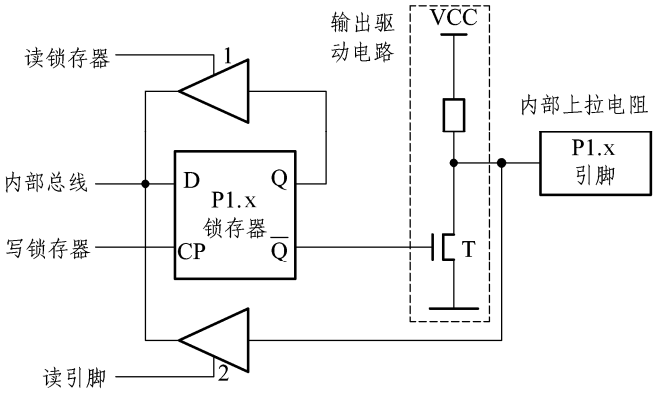

图 5.1 P1 口某位结构图

2. P1 口作为通用 I/O 口使用

1）P1 口用作输出口

当 P1 口作为输出口使用时，CPU 经内部总线将数据送入锁存器。当输出 1 时，CPU 对 P1.x 位锁存器写入高电平“1”，内部写脉冲加在锁存器时钟端 CP 上，锁存数据到 Q、$\overline{Q}$ 端，$\overline{Q}$ =“0”，使输出驱动器的场效应管 T 截止，该位的输出引脚由内部上拉电阻拉成高电平，P1.x 输出为 1。同理，当输出 0 时，$\overline{Q}$ =“1”，使输出驱动器的场效应管 T 导通，P1.x 输出为 0。

2）P1 口用作输入口

P1 口用作输入口时分为读引脚和读锁存器。

读引脚：读取芯片引脚的数据，实际上就是读出外部电路的输入信息。这时使用图 5.1 中下方的数据缓冲器 2，CPU 使“读引脚”为高电平“1”，三态数据输入缓冲器 2 导通，将 P1 口的电平读入到内部总线。

另外，从图 5.1 中还可以看出，在读入端口引脚数据时，由于输出驱动 T 并接在引脚上，如果 T 导通，就会将输入的高电平拉成低电平，从而产生误读。所以，在端口进行输入操作前，应先向端口锁存器写入“1”，即首先向锁存器写高电平“1”，使输出场效应管 T 截止，引脚处于悬浮状态，可作高阻抗输入。

读锁存器：先从锁存器中读取数据，然后进行处理，最后将处理后的结果重新写入锁存器中。这类指令称为“读—修改—写”指令。例如下面的 C51 语句：

```
P1=P1&0x0f;                    //将 P1 口的高 4 位引脚清 0，并从 P1 口输出
```

执行该语句时，分为“读—修改—写”三步。首先读入 P1 口锁存器中的数据；然后与 0x0f 进行“逻辑与”操作，即将所读入数据的高 4 位清 0；最后再将结果送回 P1 口（写操作）。对于这类“读—修改—写”语句，不直接读引脚而读锁存器是为了避免可能出现的错误。因为在端口已处于输出状态时，若端口的负载恰好是一个晶体管的基极，则导通了的 PN 结会把端口引脚的高电平拉低，这样直接读引脚就会把本来的“1”误读为“0”，但若从锁存器 Q 端读，就能避免这样的错误，得到正确的数据。

5.1.2　P2 口

P2 口是一个 8 位、可位寻址的准双向口，它有两种功能，一是在不需要片外 ROM 和 RAM 扩展时，用作通用 I/O 口，其功能与原理与 P1 口相同。二是当系统扩展片外 ROM 和 RAM 时，由 P2 口输出高 8 位地址（低 8 位地址由 P0 口输出）。

1. P2 口的结构

P2 口的电路结构如图 5.2 所示。P2 口某位的结构与 P1 口类似，驱动部分与 P1 口相同，但比 P1 口多了一个 MUX 开关和转换控制部分。在控制信号的作用下，MUX 开关可以分别接通锁存器输出或地址线。

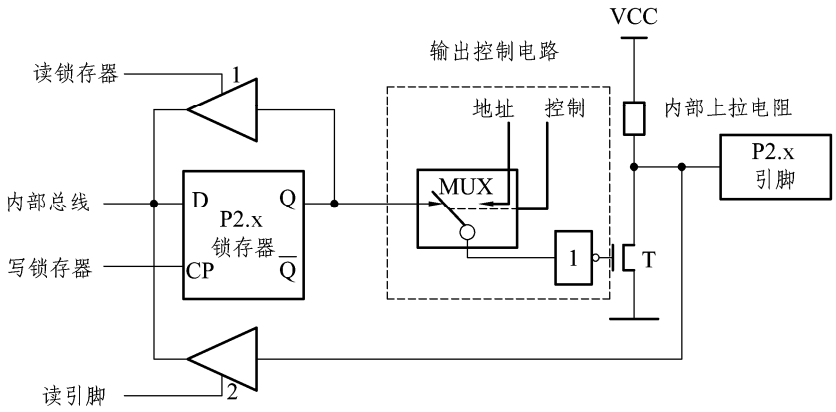

图 5.2 P2 口某位结构图

2. P2 口作为通用 I/O 口使用

1）P2 口用作输出口

当 P2 口作为输出口使用时，内部的控制信号为低电平，使 MUX 开关接通锁存器 Q 端的输出通路。当输出 1 时，CPU 经内部总线对 P2.x 位锁存器写入高电平“1”，内部写脉冲加在锁存器时钟端 CP 上，锁存数据到 Q、$\overline{Q}$ 端，经 MUX 和反相器后使输出驱动器的场效应管 T 截止，该位的输出引脚由内部上拉电阻拉成高电平，P2.x 输出为 1。同理，当输出 0 时，使输出驱动器的场效应管 T 导通，P2.x 输出为 0。

2）P2 口用作输入口

P2 口用作输入口时分为读引脚和读锁存器，与 P1 口原理相同。

3. P2 口作为地址总线使用

当系统扩展片外 ROM 和 RAM 时，由 P2 口输出高 8 位地址（低 8 位地址由 P0 口输出，原理见下节）。此时，在 CPU 的控制下使内部的控制信号为高电平，MUX 开关转向内部地址线一端。因为访问片外 ROM 和 RAM 的操作往往接连不断，所以，P2 口要不断送出高 8 位地址，此时 P2 口无法再用作通用 I/O 口。

5.1.3 P0 口

P0 口既可作为通用 I/O 端口使用，又可作地址/数据总线使用。作通用 I/O 输出时，输出级属开漏电路，必须外接 10 kΩ上拉电阻，才有高电平输出；作通用 I/O 输入时，必须先向对应的锁存器写入“1”，使 T2 截止，不影响输入电平。当 P0 口被地址/数据总线占用时，就无法再作通用 I/O 口使用了。

1. P0 口的结构

P0 口的电路结构如图 5.3 所示。电路中包含有 1 个数据输出锁存器、2 个三态数据输入缓

冲器、1 个数据输出的驱动电路和 1 个输出控制电路。驱动电路由上拉场效应管 T1 和驱动场效应管 T2 组成，其工作状态受控制电路“与”门、反相器和转换开关 MUX 控制。

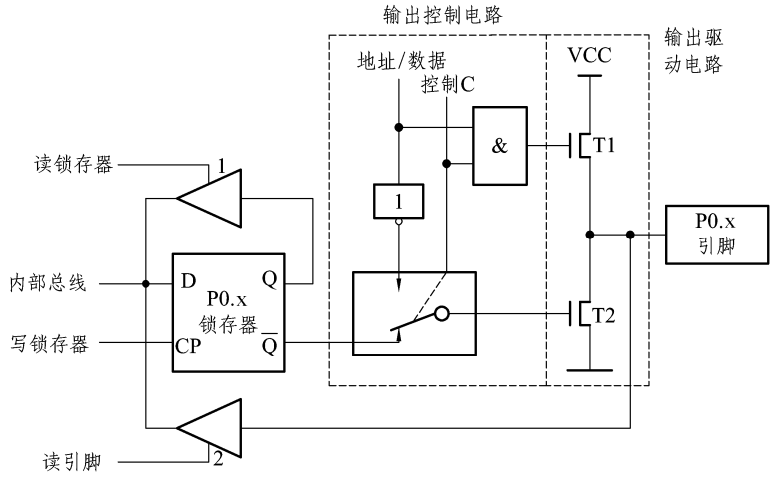

图 5.3　P0 口某位结构图

2. P0 口作为通用 I/O 口使用

当 AT89S51 单片机组成的系统无外扩存储器，CPU 对片内存储器和 I/O 口读/写时，由硬件自动使控制线 C 为低电平，封锁“与”门，使输出级中的上拉场效应管 T1 处于截止状态。开关 MUX 拨向 $\overline{Q}$ 输出端位置，它把输出级 T2 与锁存器的 $\overline{Q}$ 端接通。

1）P0 口用作输出口

当 CPU 向端口输出数据（执行输出指令）时，写脉冲加在锁存器的 CP 上，这样，与内部总线相连的 D 端的数据取反后就出现在 $\overline{Q}$ 端上，又经输出驱动器的场效应管 T2 反相，在 P0 端口上出现的数据正好是内部总线的数据。但要注意，因 T1 处于截止状态，输出级是漏极开路的开漏电路，因此，P0 口应外接 10 kΩ的上拉电阻，才能输出高电平。

2）P0 口用作输入口

P0 口用作输入口时分为读引脚和读锁存器，与 P1 口原理相同。

但要注意，当 P0 口进行通用 I/O 输入时，必须先向电路中的锁存器写入“1”，使 T2 截止，以避免锁存器为“0”状态时对引脚读入的干扰。

3. P0 口作为地址/数据总线使用

当 AT89S51 单片机扩展外存储器（ROM 或 RAM），CPU 对片外存储器读/写时，P0 口在 CPU 的控制信号管理下分时复用，作为外存储器的地址总线和数据总线。P0 口分时复用作为外存储器的地址/数据总线后，就不能作为通用 I/O 使用了。

1）P0 口用作输出地址/数据总线

由内部硬件自动使控制线 C 为高电平，“与门”解锁，MUX 开关把 CPU 内部“地址/数据”线经反相器与驱动场效应管 T2 栅极接通，输出地址/数据信号。从图 5.3 可以看到，上下两个 FET 处于反相，构成推拉式输出电路（T1 导通时上拉，T2 导通时下拉），使负载能力大为提高。所以，只有 P0 口的输出可驱动 8 个 LS 型 TTL 负载。

2）P0 口作输入数据总线

由内部硬件自动使控制线 C 为低电平，使 T1 处于截止状态，开关 MUX 拨向 $\overline{Q}$。CPU 向锁存器写高电平“1”，使输出场效应管 T2 截止，“读引脚”为高电平“1”，则外存储器数据直接从引脚通过输入缓冲器读入内部总线。

5.1.4 P3 口

1. P3 口的结构

P3 口的电路结构如图 5.4 所示。P3 口某位的结构与 P1 口类似，驱动部分与 P1 口相同，不同的是增加了一个与非门用于第二功能控制逻辑。因此 P3 口既可作为通用 I/O 口，还可作为第二功能口。

2. P3 口作为通用 I/O 口使用

当把 P3 口作为通用 I/O 口使用时，“第二输出功能”端保持高电平，工作原理与 P1 口类似。

1）P3 口用作输出口

当 P3 口作为输出口使用时，“第二输出功能”端保持高电平，打开“与非”门，所以，D 锁存器输出端 Q 的状态可通过“与非”门送至场效应管 T 输出。

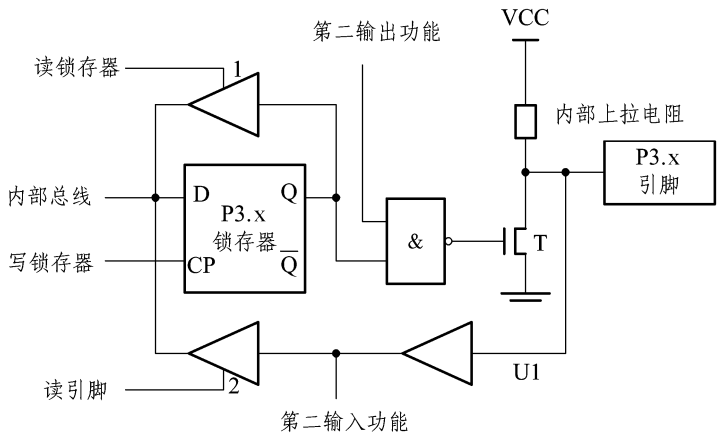

图 5.4 P3 口某位结构图

2）P3 口用作输入口

当 P3 口作为输入口使用（即 CPU 读引脚状态）时，同 P0～P2 口一样，应由软件向口锁存器先写入“1”，即使 D 锁存器 Q 端保持为 1，“与非”门输出为 0，FET 场效应管 T 截止，引脚端可作为高阻输入。当 CPU 发出读命令时，使三态缓冲器 2 上的“读引脚”信号有效，三态缓冲器 2 开通，于是引脚的状态经缓冲器 U1（常开的）、三态缓冲器 2 送到 CPU 内部总线。

3. P3 口用作第二功能口

当端口用于第二功能时，8 个引脚可按位独立定义，见第 1 章表 1.4 所示。

P3 口的特点在于为适应引脚信号第二功能的需要，增加了第二功能控制逻辑。由于第二功能信号有输入和输出两类，因此分两种情况说明。

当某位被用作第二功能时，该位的 D 锁存器 Q 端应被内部硬件自动置 1，使“与非”门对“第二输出功能端”是畅通的。“第二输出功能”端可为表 1.4 中的 TXD、$\overline{WR}$ 和 $\overline{RD}$ 三个第二输出功能引脚。例如，某一位被选择为 $\overline{RD}$ 功能，则该位的“第二输出功能”端上的 $\overline{RD}$ 控制信号状态通过“与非”门和 T 输出到引脚端。

由于 D 锁存器 Q 端已被置 1，“第二输出功能”端不用作第二功能输出时也保持为 1，所以 T 截止，该位引脚为高阻输入。此时，第二输入功能为 RXD、$\overline{INT0}$、$\overline{INT1}$、T0 和 T1。由于端口不作为通用 I/O 口，因此，“读引脚”信号无效，三态缓冲器 2 不导通。此时，某位引脚的第二输入功能信号（如 RXD）经缓冲器 3 送入“第二输入功能端”。

5.1.5　I/O 端口的负载能力和接口要求

综上所述，P0 口的输出级与 P1～P3 口的输出级在结构上是不同的，因此，它们的负载能力和接口要求也各不相同。

（1）P0 口与其他口不同，它的输出级无上拉电阻。当把它用作通用 I/O 口时，输出级是开漏电路，故用其输出去驱动 NMOS 输入时须外接上拉电阻。用作输入时，应先向口锁存器写 1。把它当作地址/数据总线时（片外扩展 ROM 或 RAM 的情况），则无须外接上拉电阻。P0 口的每一位输出可驱动 8 个 LS 型 TTL 负载。

（2）P1～P3 口的输出级接有内部上拉负载电阻，它们的每一位输出可驱动 4 个 LS 型 TTL 负载。作为输入口时，任何 TTL 或 NMOS 电路都能以正常的方式驱动 AT89S51 单片机（CHMOS）的 P1～P3 口。由于它们的输出级具有上拉电阻，所以也可以被集电极开路（OC 门）或漏极开路所驱动，而无须外接上拉电阻。

对于 AT89S51 单片机（CHMOS），端口只能提供几毫安的输出电流，故当作输出口去驱动一个普通晶体管的基极（或 TTL 电路输入端）时，应在端口与晶体管基极间串联一个电阻，以限制高电平输出时的电流。

P1～P3 口也都是准双向口。作为输入时，必须先对相应端口锁存器写入“1”。

小提示

（1）P0～P3 口均可作为通用 I/O 口使用。同时，P0 口还能在系统扩展时分时复用，作为低 8 位地址总线和数据总线，P2 口还能在系统扩展时用作高 8 位地址总线，P3 口还可以作为第二功能使用。

（2）P1～P3 口作为输出口使用时，无须再外接上拉电阻。但 P0 口作为输出口使用时，由于 T1 截止，输出电路是漏极开路电路，所以必须外接上拉电阻。

（3）当 P0～P3 口作为通用 I/O 口的输入口使用时，应区分读引脚和读端口。读引脚时，必须先向锁存器写入“1”，以避免锁存器为“0”状态时对引脚读入的干扰。

5.1.6 I/O 端口的应用举例

1. 输出口的应用

例 5.1 单片机 P1 口的 P1.0 接有一个 LED，如图 5.5 所示，利用单片机控制一个 LED 不断地闪烁。

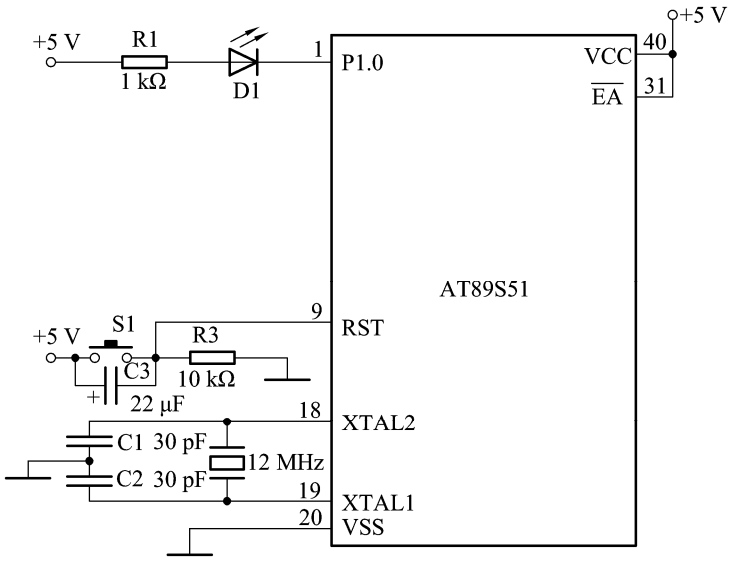

图 5.5 单片机控制一个 LED 闪烁的电路

分析：当 P1.0 引脚输出低电平时，LED 点亮；当 P1.0 引脚输出高电平时，LED 熄灭。控制一个 LED 不断闪烁的参考源程序如下：

```
//*********************************************************************************
//程序：ex5_1.c
//功能：一个 LED 不断地闪烁
//*********************************************************************************
```

```
#include<reg51.h>                    /*包含头文件 reg51.h*/
#define uint unsigned int            /*宏定义后方便书写*/
sbit LED0=P1^0;                      //定义 P1.0 引脚位名称为 LED0
void DelayMS(uint x);                //延时函数声明
void main()                          //主程序
{
    while(1)
    {
        LED0=~LED0;
        DelayMS(250);                //延时函数调用
    }
}
//*****************************************************************************
//函数名：DelayMS
//函数功能：软件延时函数
//形式参数：uint x；x 控制空循环的外循环次数，共循环 x*120 次
//*****************************************************************************
void DelayMS(uint x)
{
    uint j,k;                        //定义无符号字符型变量 j 和 k
    for(k=0;k<x;k++)                 //双重 for 循环语句实现软件延时
        for(j=0;j<120;j++);          //循环体为空循环
}
```

小经验

（1）对于每个函数都要有注释，在函数的前面注释该函数的名称、参数、参数的值域、返回值、功能、设计思路、注意事项等。在商业代码中注释比程序长的情况很常见。在代码维护、调试和排错时，若修改了代码，要养成立即修改注释的习惯。

（2）宏定义后的注释使用/* */，而不要用//，以避免某些版本的编译器在代码中将宏定义连同注释全部替换而造成错误。

（3）通常情况下，Keil C 编译器提供了多种型号的 51 系列单片机的特殊功能寄存器和部分可寻址 SFR 的位定义的头文件 reg51.h，只要在程序中包含了该头文件，就可以直接使用已定义的特殊功能寄存器和可寻址位。在 C51 语言中，用户可以在程序中通过关键字 sfr、sbit 来定义没有定义的特殊功能寄存器、可寻址位，从而在程序中直接访问它们。

```
例如：sbit LED0=P1^0；              //定义 P1 口的第 0 位 P1.0 接一个 LED0
```

（4）在例 5.1 的程序中，函数 DelayMS（ ）的定义放在主函数 main（ ）之后，因此，必

须先声明才能调用。如果函数 DelayMS ( ) 放在主函数 main ( ) 之前，无须进行 DelayMS ( ) 函数的声明，可在 main ( ) 函数直接调用。

例 5.2　单片机的 P1 口经过芯片 74LS240(八路反相器)分别连接了 8 个 LED，且 LED 的阴极并接在一起接地，如图 5.6 所示。利用单片机控制 8 个 LED 顺序点亮：首先从上到下依次点亮，循环三次；然后，从下到上依次点亮，循环三次；再从上到下依次点亮，循环三次……如此循环不断，产生一种动态的流水灯的效果。

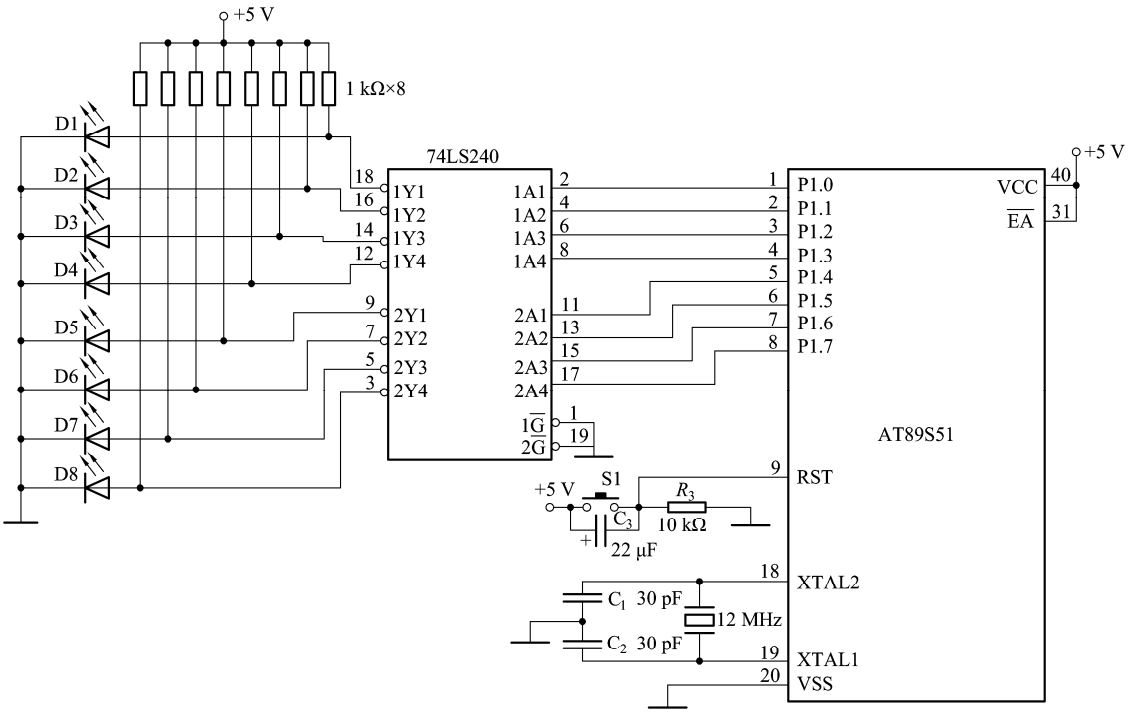

图 5.6　单片机控制 8 个 LED 发光二极管顺序点亮电路

小经验

为了增大单片机端口的扇出电流，提高负载能力，在单片机输出接口电路中经常会使用集成驱动芯片、缓冲与锁存芯片，例如 74LS245 或集电极开路电路 74LS06、74LS07 等。

分析：当 P1 口的引脚输出低电平“0”时，经 74LS240 反相后输出高电平，相应的 LED 被点亮；当 P1 口的引脚输出高电平“1”时，经 74LS240 反相后输出低电平，相应的 LED 熄灭。P1 口引脚的电平状态如表 5.1 所示。

表 5.1　P1 口引脚的电平状态

显示状态	引脚输出数据								P1 口输出数据
	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	
复位状态（全灭）	1	1	1	1	1	1	1	1	FFH
D1 亮	1	1	1	1	1	1	1	0	FEH
D2 亮	1	1	1	1	1	1	0	1	FDH
D3 亮	1	1	1	1	1	0	1	1	FBH
D4 亮	1	1	1	1	0	1	1	1	F7H
D5 亮	1	1	1	0	1	1	1	1	EFH
D6 亮	1	1	0	1	1	1	1	1	DFH
D7 亮	1	0	1	1	1	1	1	1	BFH
D8 亮	0	1	1	1	1	1	1	1	7FH

控制 8 个 LED 顺序点亮的参考源程序如下：

```
//*****************************************************************************
//程序：ex5_2.c
//功能：采用循环结构实现的流水灯控制程序
//*****************************************************************************
#include<reg51.h>                    /*包含头文件 REG51.H*/
#define uchar unsigned char          /*宏定义后方便书写*/
#define uint unsigned int            /*宏定义后方便书写*/
#define LED P1                       /*定义 8 个 LED 接至 P1 口*/
void DelayMS(uint x);                //延时函数声明
void LED_Down(uint x);               //单只 LED 依次灯下移点亮函数声明
void LED_Up(uint x);                 //单只 LED 依次灯上移点亮函数声明
void main()                          //主程序
{
    while(1)
    {
        LED_Down(3);                 //单只 LED 依次灯下移点亮三圈
        LED_Up(3);                   //单只 LED 依次灯上移点亮三圈
    }
}
//*****************************************************************************
//函数名：LED_Down
//函数功能：单只 LED 依次灯下移点亮
//形式参数：uint x；下移点亮 x 圈
```

```
//*****************************************************************************
void LED_Down(uint x)
{
    uchar i,j;
    for(i=0;i<x;i++)                    //单只 LED 依次灯下移点亮 x 圈
    {
        LED=0xfe;                       //最上边的 LED 亮
        for(j=0;j<8;j++)
        {
            DelayMS(250);               //延时函数调用
            LED=(LED<<1)|0x01;          //下移 1 位后，将 LSB 设为 1，点亮下一个 LED
        }
    }
}
//*****************************************************************************
//函数名：LED_Up
//函数功能：单只 LED 依次灯上移点亮
//形式参数：uint x；上移点亮 x 圈
//*****************************************************************************
void LED_Up(uint x)
{
    uchar i,j;
    for(i=0;i<x;i++)                    //单只 LED 依次灯上移点亮 x 圈
    {
        LED=0x7f;                       //最下边的 LED 亮
        for(j=0;j<8;j++)
        {
            DelayMS(250);               //延时函数调用
            LED=(LED>>1)|0x80;          //上移 1 位后，将 MSB 设为 1，点亮上一个 LED
        }
    }
}
void DelayMS(uint x)                    //延时函数，见例 5.1，此处略
```

2. 输入口的应用

例 5.3　单片机的 P1.0、P1.1 接两个 LED，模拟汽车左、右转向灯；P3.0、P3.1 接两个拨动开关 S0、S1，模拟驾驶员发出左、右转命令。模拟控制系统电路如图 5.7 所示。汽车转向灯显示状态如表 5.2 所示。

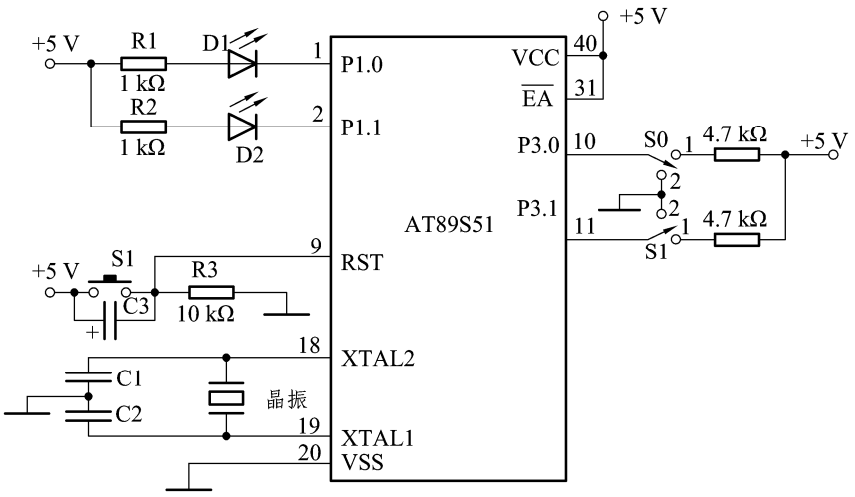

图 5.7　模拟汽车左右转向灯控制系统电路

表 5.2　汽车转向灯显示状态表

P3 口的状态		转向灯显示状态		驾驶员发出的命令
P3.1（右转向开关 S1）	P3.0（左转向开关 S0）	右转灯（P1.1）	左转灯（P1.0）	
1	1	灭（1）	灭（1）	驾驶员未发出命令
0	1	闪烁（0）	灭（1）	驾驶员发出右转命令
1	0	灭（1）	闪烁（0）	驾驶员发出左转命令
0	0	闪烁（0）	闪烁（0）	驾驶员发出汽车故障命令

模拟汽车转向灯控制的参考源程序如下：

```
//*****************************************************************************
//程序：ex5_3.c
//功能：采用 switch 语句实现的模拟汽车转向灯控制程序
//*****************************************************************************
#include<reg51.h>                          /*包含头文件 REG51.H*/
#define uchar unsigned char                /*宏定义后方便书写*/
#define uint unsigned int                  /*宏定义后方便书写*/
sbit Lift_LED=P1^0;                        //定义 P1.0 引脚位名称为 Lift_LED
sbit Right_LED=P1^1;                       //定义 P1.1 引脚位名称为 Right_LED
void DelayMS(uint x);                      //延时函数声明
void main()                                //主程序
{
    uchar Temp, KEY_Status;                //定义开关的状态变量 KEY_Status
    P3=0xff ;                              //P3 口作为输入口，必须先置全 1
    while(1)
    {
```

```
            Temp=P3;                                    //读取 P3 口的状态
            KEY_Status=Temp &0x03;                      //屏蔽掉 P3 口高 6 位，取开关 S0 和 S1 的状态
            switch(KEY_Status)
            {
              case 0: Right_LED=0; Lift_LED=0;break;    //如果 P3.1P3.0=00，则点亮左、右转向灯
              case 1: Right_LED=0; break;               //如果 P3.1P3.0=01，则点亮右转向灯
              case 2: Lift_LED=0; break;                //如果 P3.1P3.0=10，则点亮左转向灯
              default: ;                                //空语句，什么都不做
            }
            DelayMS(200);                               //延时
            Lift_LED=1;                                 //左转灯回到熄灭状态
            Right_LED=1;                                //右转灯回到熄灭状态
            DelayMS (200);                              //延时
        }
    }
    void DelayMS(uint x)                                //延时函数，见例 5.1，此处略
```

5.2 中断系统

单片机系统的运行同其他微机系统一样，CPU 不断地与外部输入/输出设备交换信息。CPU 与外部设备交换信息通常有以下几种方式：

（1）程序控制传送方式，又分为无条件传送方式和查询传送方式；

（2）中断传送方式；

（3）直接存储器存取（DMA）方式。

在单片机系统中，中断方式是一种非常重要的数据输入/输出方式。实时控制、故障自动处理、单片机与外围设备间的数据传送往往采用中断系统。中断系统的应用大大提高了单片机工作效率。下面对其作较详细的介绍。

5.2.1 中断系统基本知识

1. 中断的概念

先通过一个日常生活中的例子来说明中断的含义。假如你正在编写单片机程序，手机响了，这时，你放下手中的编程工作，去接电话。通话完毕，再继续写程序。这个例子就表现了中断及其处理过程：手机铃声使你暂时中止当前编写程序的工作，而去处理更为紧急的事情（接电话），把更紧急的事情处理完毕之后，再回头来继续处理原来的事情（编写程序）。在这个例子中，手机铃声称为“中断请求”，你暂停编程去接电话叫作“中断响应”，接电话的过程就是“中断处

理”。

中断是通过硬件来改变 CPU 的运行方向。计算机在执行程序的过程中，若出现某个特殊情况（或称为“事件”），由服务对象向 CPU 发出中断请求信号，要求 CPU 暂时中断当前程序的执行，而转去执行处理这一事件的处理程序，处理完毕之后再回到原来程序的“中断点”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打断的情况称为“中断”，如图 5.8 所示。下面给出几个与中断有关的概念。

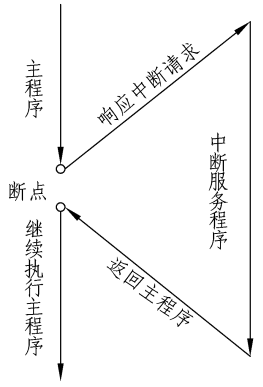

图 5.8　中断响应过程流程图

（1）主程序：CPU 原来正常运行的程序称为主程序。

（2）断点：主程序被断开的位置（或地址）称为断点。

（3）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。

（4）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）。

（5）中断服务程序：CPU 响应中断请求后，转去执行相应的处理程序，该处理程序称为中断服务或中断处理子程序。

> 小提示
>
> 调用中断服务程序的过程类似于调用子程序，其区别在于调用子程序在主程序中是事先安排好的，而何时调用中断服务程序事先却无法确定，因为“中断”的发生是由外部因素决定的，程序中无法事先安排调用指令。因此，调用中断服务程序的过程是由硬件自动完成的。

2. 中断的特点

1）分时操作

中断可以解决快速的 CPU 与慢速的外设之间的矛盾，使 CPU 和外设同步工作。CPU 在启动外设工作后继续执行主程序，同时外设也在工作，每当外设做完一件事就发出中断申请，请求 CPU 中断它正在执行的程序，转去执行中断服务程序（一般情况是处理输入输出数据）；中断处理完之后，CPU 恢复执行主程序，外设也继续工作。这样，CPU 可启动多个外设同时工作，大大地提高了 CPU 的效率。

2）实时处理

在实时控制中，现场的各种参数、信息均随时间和现场而变化。这些外界变量可根据要求随时向 CPU 发出中断申请，请求 CPU 及时处理，如中断条件满足，CPU 马上就会响应进行相应的处理，从而实现实时处理。

3）故障处理

针对难以预料的情况或故障，如掉电、存储出错、运算溢出等，可通过中断系统由故障源