项目 1　报警灯的闪烁控制

1.1　任务一：预备知识

1.1.1　知识链接

1．预备知识

（1）十六进制是二进制的简短表示形式。

（2）十进制中的 0～15 分别用十六进制表示为 0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F。

（3）熟练掌握二进制与十六进制之间的转换。进制转换表如表 1-1 所示。

表 1-1　进制转换表

十　进　制	二　进　制	十 六 进 制	十　进　制	二　进　制	十 六 进 制
0	0000	0	9	1001	9
1	0001	1	10	1010	A
2	0010	2	11	1011	B
3	0011	3	12	1100	C
4	0100	4	13	1101	D
5	0101	5	14	1110	E
6	0110	6	15	1111	F
7	0111	7	16	10000	10
8	1000	8			

规律：把 1 个 4 位二进制数转换成 1 个十六进制数，一般先把二进制数转换成十进制数，再把十进制数转换成十六进制数。

2．C51 知识

C 语言作为一种非常方便的语言得到了广泛的支持，很多硬件开发都使用 C 语言编程，如各种单片机、DSP、ARM 等。C 语言程序本身不依赖于机器硬件系统，基本上不需要修

改就可以将程序从不同的单片机中移植过来。C 语言提供了很多数学函数并支持浮点运算，开发效率高，故可以缩短开发时间，增加程序的可读性和可维护性。

1）C51 与 ASM51 相比的优点

C51 与 ASM51 相比具有如下优点。

（1）不要求了解单片机的指令系统，仅要求对 8051 的存储器结构有初步认识。

（2）寄存器分配、不同存储器的寻址及数据类型等细节可由编译器管理。

（3）程序有规范的结构，可分成不同的函数，这种方式可使程序结构化。

（4）提供的库包含许多标准子程序，具有较强的数据处理能力。

（5）具有方便的模块化编程技术，可容易地移植已编好的程序。

2）C51 包含的头文件

C51 包含的头文件通常有 reg51.h、reg52.h、math.h、ctype.h、stdio.h、stdlib.h 及 absacc.h 等，其中针对 51 单片机编程的头文件有 reg51.h 和 reg52.h（定义特殊功能寄存器和位寄存器），另外常用的数学运算头文件有 math.h（定义常用的数学运算）等。

书写格式范例：#include"reg51.h"或#include<reg52.h>。

3）C 语言的函数结构

一个 C 语言的函数由如下两部分组成。

（1）函数首部，即函数的第一行，包括函数名、函数类型、函数属性、函数参数（形参）、参数类型。一个函数名后面必须跟一对圆括号。

（2）函数体，即函数首部下面的大括号“{}”内的部分。如果一个函数内有多个大括号，那么最外层的一对“{}”为函数体的范围。函数体一般包括声明部分（在这部分定义所用到的变量）、执行部分（这部分由若干条语句组成）。

在以下情况中也可以没有声明部分，甚至既没有声明部分，也没有执行部分。

例如：

```
void mDelay()
{}
```

这是一个空函数，也是合法的。

在编写程序时，可以利用空函数。例如，主程序需要调用一个延时函数，首先弄清楚主程序的框架结构并编译通过，然后利用空函数把架子搭起来，里面的细节（如具体延时多少、如何延时）可以在以后慢慢填写，这样在主程序中就可以调用它了。

4）主函数体

一个 C 语言程序总是从 main 函数开始执行，不论 main()放在什么位置。

main 函数格式：void main()。

main 函数特点：无返回值，大多数情况下无参数。

任何一个 C 语言程序有且仅有一个 main 函数，它是整个程序开始执行的入口。

例如：

```
void main()
  { 总程序从这里开始执行；
    其他语句；
  }
```

每条语句和数据定义的最后必须有一个分号，分号是 C 语句的必要组成部分，是一条语句的结束标志。

C 语言的书写格式自由，可以在一行写多条语句，也可以把一条语句写在多行，没有行号（可以有标号），对书写的缩进没有要求。但是建议读者按照一定的规范来写，这样可以给自己带来方便。

另外，可以用/*…*/的形式为 C 语言程序的任何一部分做注释，从“/*”开始，一直到“*/”为止，中间任何内容都被认为是注释。因此，在书写特别是修改源程序时要注意，若删掉一个“*/”，那么从这里开始一直到下一个“*/”中的全部内容都将被认为是注释。

3．C 语言中的数据类型、常量及变量

1）数据类型

首先简单介绍 C 语言的标识符和关键字。标识符用来标识源程序中某个对象的名字，这些对象可以是语句、数据类型、函数、变量及数组等。C 语言是区分字母大小写的一种高级语言，如果定时器 1 被定义为"Timer1"，那么程序中的"TIMER1"是与其定义完全不同的标识符。

标识符由字符串、数字和下画线等组成，注意第一个字符必须是字母或下画线，如"1Timer"是错误的，编译时会有错误提示。有些编译系统专用的标识符以下画线开头，因此一般不要以下画线开头命名标识符。标识符在命名时应当简单且含义清晰，这样有助于理解程序。C51 编译器只支持标识符的前 32 位为有效标识，这在一般情况下也足够使用了。

关键字则是编程语言保留的特殊标识符，它们具有固定的名称和含义，在程序编写中不允许标识符与关键字相同。KEIL μVISION2 中的关键字除了包括 ANSI C 标准的 32 个关键字，还根据 51 单片机的特点扩展了相关的关键字。其实在 KEIL μVISION2 的文本编辑器中编写 C 语言程序，系统可以以不同的颜色显示保留字，默认颜色为天蓝色。

表 1-2 中列出了 KEIL μVISION2 C51 编译器所支持的数据类型。在标准 C 语言中基本的数据类型为 char、int、short、long、float 和 double，而在 C51 编译器中 int 和 short 相同，float 和 double 相同，它们的具体定义如下。

表 1-2　KEIL μVISION2 C51 编译器所支持的数据类型

数 据 类 型	长　　度	值　　域
unsigned char	单字节	0～255
signed char	单字节	−128～+127

续表

数 据 类 型	长　　度	值　　域
unsigned int	双字节	0～65535
signed int	双字节	-32768～+32767
unsigned long	4 字节	0～4294967295
signed long	4 字节	-2147483648～+2147483647
float	4 字节	±1.175494E-38～±3.402823E+38
*	1～3 字节	对象的地址
bit	位	0 或 1
sfr	单字节	0～255
sfr16	双字节	0～65535
sbit	位	0 或 1

（1）char（字符类型）。

char 字符类型长度为单字节，通常用于定义处理字符数据的变量或常量，分为无符号字符类型 unsigned char 和有符号字符类型 signed char，默认值为 signed char。unsigned char 用字节中所有的位来表示数值，可以表达的数值范围是 0～255。signed char 用字节中最高位字节表示数据的符号，“0”表示正数，“1”表示负数，负数用补码表示，其所能表示的数值范围是-128～+127。unsigned char 常用于处理 ASCII 字符和小于或等于 255 的整型数。

正数的补码与原码相同，负二进制数的补码等于它的绝对值按位取反后加 1。

（2）int（整型）。

int 整型长度为双字节，用于存放一个双字节数据，分为有符号整型 signed int 和无符号整型 unsigned int，默认值为 signed int。signed int 可以表示的数值范围是-32768～+32767，字节中最高位表示数据的符号，“0”表示正数，“1”表示负数。unsigned int 表示的数值范围是 0～65535。

（3）long（长整型）。

long 长整型长度为 4 字节，用于存放一个 4 字节数据，分为有符号长整型 signed long 和无符号长整型 unsigned long，默认值为 signed long。signed long 表示的数值范围是-2147483648～+2147483647，字节中最高位表示数据的符号，“0”表示正数，“1”表示负数。unsigned long 表示的数值范围是 0～4294967295。

（4）float（浮点型）。

float 浮点型在十进制中具有 7 位有效数字，是符合 IEEE-754 标准的单精度浮点型数据，长度为 4 字节。因浮点型数据的结构较复杂，在后面的章节中再进行详细的讨论。

（5）*（指针型）。

指针型数据本身就是一个变量，该变量用于存放指向另一个数据的地址。这个指针变量要占据一定的内存单元，对不同的处理器其长度也不尽相同，在 C51 编译器中它的长度

一般为 1～3 字节。指针变量也具有类型。

（6）bit（位标量）。

bit 位标量是 C51 编译器的一种扩充数据类型，利用它可以定义一个位标量，但不能定义位指针，也不能定义位数组。它的值是一个二进制位，不是 0 就是 1，与一些高级语言中的 Boolean 类型中的 True 和 False 类似。

C51 存储器类型中有一个 bdata 存储器类型，这个是指可位寻址的数据存储器，位于单片机的可位寻址区中，可以将要求可位寻址的数据定义为 bdata。例如：

```
unsigned char bdata ib；//在可位寻址区定义 unsigned char 的变量 ib
int bdata ab[2]；       //在可位寻址区定义数组 ab[2]，这些也称为可位寻址对象
sbit ib7=ib^7；         //用关键字 sbit 定义位变量来独立访问可位寻址对象的其中一位
sbit ab12=ab[1]^12；
```

操作符“^”后面的位号的最大值取决于指定的基址类型 char0~7、int0~15、long0~31。

（7）sfr（特殊功能寄存器）。

sfr 也是 C51 编译器的一种扩充数据类型，占用一个内存单元，值域为 0～255。利用它可以访问 51 单片机内部的所有特殊功能寄存器。例如，sfr P1 =0x90 语句可将 P1 定义为 P1 端口在片内的寄存器，在后面的语句中可用类似于 P1＝255（对 P1 端口的所有引脚置高电平）的语句来操作特殊功能寄存器。

（8）sfr16（16 位特殊功能寄存器）。

sfr16 占用两个内存单元，值域为 0～65535，也是 C51 编译器的一种扩充数据类型。sfr16 和 sfr 一样用于操作特殊功能寄存器，不同之处是它用于操作占 2 字节的特殊功能寄存器，如定时器 T0 和 T1。

（9）sbit（可寻址位）。

sbit 同样是 C51 编译器中的一种扩充数据类型，利用它可以访问芯片内部的 RAM 中的可寻址位或特殊功能寄存器中的可寻址位。

下文为 sfr、sfr16、sbit 定义变量的方法。sfr 和 sfr16 可以直接对 51 单片机的特殊功能寄存器进行定义，定义方法如下。

sfr 特殊功能寄存器名=特殊功能寄存器地址常数。

sfr16 特殊功能寄存器名= 特殊功能寄存器地址常数。

可以按如下方法定义 AT89C51 的 P1 口。

```
sfr P1 =0x90；          //定义 P1 I/O 口，其地址为 90H
```

sfr 关键字后面是一个要定义寄存器的名字，可任意选取，但要符合标识符的命名规则，名字最好有一定的含义。例如，P1 口可以用 P1 命名，这样程序会变得好读很多；等号后面必须是常数，不允许有带运算符的表达式，而且该常数必须在特殊功能寄存器的地址范围之内（80H～FFH），具体可查看书中的相关表。sfr 是用来定义 8 位的特殊功能寄存器，

而 sfr16 则是用来定义 16 位的特殊功能寄存器。例如，8052 的 T2 定时器，可以定义如下：

```
sfr16 T2 =0xCC；//这里定义 8052 定时器 T2，地址为 TL2=CCH，TH2=CDH
```

用 sfr16 定义 16 位特殊功能寄存器时，等号后面是它的低位地址，高位地址一定要位于物理低位地址之上。注意不能用于定时器 T0 和 T1 的定义。

sbit 可定义可位寻址对象，如访问特殊功能寄存器中的某位。其实这样的应用是经常要用的。例如，访问 P1 口中的第 2 个引脚 P1.1 可以按照如下方法去定义。

（1）sbit 位变量名＝位地址。

```
//这样是把位的绝对地址赋给位变量。同 sfr 一样，sbit 的位地址必须位于 80H～FFH
sbit P1_1 = 0x91；
```

（2）sbit 位变量名＝特殊功能寄存器名^位号。

```
sft P1 =0x90；
//先定义一个特殊功能寄存器名，再指定位变量名所在的位置。可寻址位位于特殊功能寄存器中时可
//采用这种方法
sbit P1_1 =P1 ^ 1；
```

（3）sbit 位变量名＝字节地址^位号。

```
//这种方法其实和（2）是一样的，只是把特殊功能寄存器的位地址直接用常数表示
sbit P1_1 =0x90 ^ 1；
```

2）常量

常量的数据类型只有整型、浮点型、字符型、字符串型和转义字符。

（1）整型常量可以表示为十进制，如 123、0、−89 等；表示为十六进制时则以 0x 开头，如 0x34、−0x3B 等。如果常量为长整型，那么就在数字后面加字母 L，如 104L、034L 等。

（2）浮点型常量可分为十进制和指数表示形式。十进制由数字和小数点组成，如 0.888、3345.345、0.0 等，整数或小数部分为 0，0 可以省略但必须有小数点。指数表示形式为[±]数字[.数字]e[±]数字，[]中的内容为可选项，内容根据具体情况可有可无，但其余部分必须有，如 125e3、7e9、−3.0e−3。

（3）字符型常量是单引号内的字符，如'a'和'd'等。不可以显示的控制字符可以在该字符前面加一个反斜杠“\”组成专用转义字符，常用转义字符表如表 1-3 所示。

表 1-3　常用转义字符表

转 义 字 符	含　义	ASCII 码（十六/十进制）
\o	空字符（NULL）	00H/0
\n	换行符（LF）	0AH/10
\r	回车符（CR）	0DH/13
\t	水平制表符（HT）	09H/9
\b	退格符（BS）	08H/8
\f	换页符（FF）	0CH/12

续表

转 义 字 符	含　义	ASCII 码（十六/十进制）
\'	单引号	27H/39
\"	双引号	22H/34
\\	反斜杠	5CH/92

（4）字符串型常量由双引号内的字符组成，如"test"和"OK"等。当引号内没有字符时，为空字符串。在使用特殊字符时同样要使用转义字符，如双引号。在 C 语言中，字符串型常量是作为字符类型数组来处理的，在存储字符串时，系统会在字符串尾部加上转义字符“\o”以作为该字符串的结束符。字符串型常量"A"和字符型常量'A'是不同的，前者在存储时多占用 1 字节。

（5）转义字符的值是一个二进制数。

常量可用在不必改变值的场合，如固定的数据表中、字库中等。下文将说明几种常量的定义方式。

```
   #difine False 0x0;            //用预定义语句可以定义常量
   #difine True 0x1;             //这里定义 False 为 0，True 为 1
                                 //在程序中用到 False 编译时自动用 0 替换，同理 True 替换为 1
   unsigned int code a=100;      //这一句用 code 把 a 定义在程序存储器中并赋值
   const unsigned int c=100;     /* unsigned const int c=100; 用 const 定义 c 为无
符号 int 常量并赋值*/
```

以上两句的值都保存在程序存储器中，而程序存储器在运行中是不允许被修改的，因此如果在这两句后面用了类似 a=110, a++这样的赋值语句，那么编译时将会出错。

3）变量

要在程序中使用变量必须先用标识符作为变量名，并指出所用的数据类型和存储模式，这样编译系统才能为变量分配相应的存储空间。定义一个变量的格式为

［存储种类］数据类型［存储器类型］变量名表

在定义格式中除了数据类型和变量名表是必要的内容，其他内容都是可选项。存储种类有 4 种：自动（auto）、外部（extern）、静态（static）和寄存器（register），默认类型为自动（auto）。

存储器类型的说明就是指定该变量在 C51 硬件系统中所使用的存储区域，并在编译时准确定位， KEIL μVISION2 所能识别的存储器类型如表 1-4 所示。注意在 AT89C51 芯片中 RAM 只有低 128 字节，而位于 80H～FFH 的高 128 字节在 AT89C52 芯片中才有用，否则就会和特殊寄存器地址重叠。

表 1-4　KEIL μVISION2 所能识别的存储器类型

存储器类型	说　明
data	直接访问内部数据存储器（128 字节），访问速度最快

续表

存储器类型	说　明
bdata	可位寻址内部数据存储器（16 字节），允许位与字节混合访问
idata	间接访问内部数据存储器（256 字节），允许访问全部内部地址
pdata	分页访问外部数据存储器（256 字节），用 MOVX@Ri 指令访问
xdata	外部数据存储器（64KB），用 MOVX@DPTR 指令访问
code	程序存储器（64KB），用 MOVC@A+DPTR 指令访问

如果省略存储器类型，系统会按编译模式 SMALL、COMPACT 或 LARGE 所规定的默认存储器类型去指定变量的存储区域。无论什么存储模式都可以声明变量在 8051 任何存储区范围，然而把最常用的命令（如循环计数器和队列索引）放在内部数据区可以显著提高系统性能。另外，需要指出的是，变量的存储种类与存储器类型完全无关。

SMALL 存储模式把所有函数变量和局部数据段放在 8051 系统的内部数据存储区，这使访问数据非常快，但 SMALL 存储模式的地址空间因此受限。在编写小型的应用程序时，变量和数据放在 data 内部数据存储器中是很好的，因为访问速度快，但在较大的应用程序中 data 最好只存放小的变量、数据或常用的变量（如循环计数、数据索引），而大的数据则放置在别的存储区域。

COMPACT 存储模式中的所有函数和程序变量及局部数据段都定位在8051 系统的外部数据存储区。外部数据存储区最多可有 256 字节（一页），在本模式中外部数据存储区的短地址用@R0/R1 表示。

LARGE 存储模式所有的函数和程序变量及局部数据段定位在 8051 系统的外部数据存储区。外部数据存储区最多可有 64KB，因此用 DPTR 数据指针访问数据。

另外，用以重新定义数据类型的语句是 typedef，这是一条很好用的语句，如 int 类型用关键字 integer 来定义，可以写为：

```
typedef int integer;
integer a,b;
```

在编译这两条语句时，其实是先把 integer 定义为 int，在后面的语句中遇到 integer 就用 int 置换，integer 就等于 int，因此 a、b 也就被定义为 int。typedef 不能直接用来定义变量，它只是对已有数据类型进行名字上的置换，而不是产生一个新的数据类型。

下面两条语句就是一个错误的例子：

```
typedef int integer;
integer = 100;
```

使用 typedef 可以方便程序的移植并简化较长的数据类型定义。用 typedef 还可以定义结构类型，这一点在后面详细解说结构类型时再一并说明。

typedef 的语法是：typedef　已有的数据类型名　新的数据类型名。

4．LED 小灯和蜂鸣器的控制

1）蜂鸣器的介绍

（1）蜂鸣器的作用。蜂鸣器是一种一体化结构的电子讯响器，采用直流电压供电的方式，广泛应用于计算机、打印机、复印机、报警器、电子玩具、汽车电子设备、电话机及定时器等电子产品中作为发声器件。

（2）蜂鸣器的分类。蜂鸣器主要分为压电式蜂鸣器和电磁式蜂鸣器两种类型。

（3）蜂鸣器的电路图形符号。蜂鸣器在电路中用字母“H”或“HA”（旧标准用“FM”“LB”“JD”等）表示。

2）蜂鸣器的结构原理

（1）压电式蜂鸣器主要由多谐振荡器、压电蜂鸣片、阻抗匹配器、共鸣箱及外壳等组成。部分压电式蜂鸣器外壳上还装有 LED。多谐振荡器由晶体管或集成电路构成。当接通电源后（1.5～15V 直流工作电压），多谐振荡器起振，输出 1.5～2.5kHz 的音频信号，阻抗匹配器推动压电蜂鸣片发声。压电蜂鸣片由锆钛酸铅或铌镁酸铅压电陶瓷材料制成。在陶瓷片的两面镀上银电极，经极化和老化处理后，再将其与黄铜片或不锈钢片黏在一起。

（2）电磁式蜂鸣器由振荡器、电磁线圈、磁铁、振动膜片及外壳等组成。接通电源后，振荡器产生的音频信号电流通过电磁线圈，使电磁线圈产生磁场，从而使振动膜片在电磁线圈和磁铁的相互作用下，周期性地振动发声。

3）发声程序

通过在 P1.0 输出一个音频范围的方波（见图 1-1），驱动实验板上的蜂鸣器发出蜂鸣声，其中，延时子程序的作用是使输出的方波频率在人耳听觉能力之内，如果没有这个延时子程序，输出的频率将大大超出人耳的听觉能力，人将不能听到声音。更改延时常数，可以改变输出频率，也就可以调整蜂鸣器的音调。

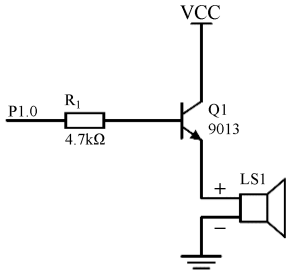

图 1-1　蜂鸣器原理图

例题：警报声程序（C51）。

```
sbit SPK = P1^0;
unsigned char frq;                  //频率
void delayms(unsigned char ms); //延时函数
main()
{TMOD = 0x01;                       //方式 1，16 位定时/计数
```

```
    frq = 0x00;                          //初值
    TH0 = 0x00;
    //初始计数初值为 0x00FF，计到 0xFFFF 下一个即 0x0000 时，进入中断
    TL0 = 0xFF;
    TR0 = 1;                             //开始计数
    IE = 0x82;                           //总中断，定时器 0 中断允许
    while(1)
    {frq++;
    delayms(1);          /*此处的延时是为了控制 frq 频率变量增加的快慢，以此来控制方波的
占空比*/
      }
    }
    void timer0( ) interrupt 1 using 1
    {/*每次进入中断程序时，frq 频率变量都是不同的，范围是 0x00H～0xFFH，这样每次定时的值都
不一样*/
    TH0 = 0xFE;                          //注意此处的初值是 0xFEXX(XX 为 frq)
    TL0 = frq;
    SPK = ~SPK;                          //反转方波
    }
    void delayms(unsigned char ms)
    {unsigned char i;
    while(ms--)
    for(i = 0; i < 120; i++);
    }
    #include<reg52.h>
    #define uint unsigned int
    #define uchar unsigned char
    sbit D1=P1^0;
    void delay();
    void delay(uint);
    void main()
    {
     while(1)
     {
         D1=0;
         delay(100);
         D1=1;
         delay(100);
     }
    }
    /*
    void delay()
    {
     uint x,y;
     for(x=100;x>0;x--)
         for(y=600;y>0;y--);
    }*/
```

```
void delay(uint z)
{
 uint x;
 uchar y;
 for(x=z;x>0;x--)
     for(y=20;y>0;y--);
}
```

1.1.2　任务分析

C语言是一种编译型程序设计语言，它兼顾了多种高级语言的特点，并具备汇编语言的功能。目前，使用 C 语言进行程序设计已经成为软件开发的一个主流。用 C 语言开发系统可以大大缩短开发周期，明显增强程序的可读性，便于改进、扩充和移植。而针对 8051 单片机的 C 语言日趋成熟，并成为专业化的实用高级语言。

1.1.3　任务实施

1．准备工作

1）工具准备

（1）多媒体教室。

（2）计算机。

2）知识准备

（1）数制转换的知识。

（2）51 单片机 C 语言程序的数据类型。

（3）51 单片机 C 语言程序的常量和变量。

2．进行简单的跑马灯实验，项目名为 RunLED2

```
//这里没有使用预定义文件，而是自己定义特殊功能寄存器，之前使用的预定义文件其实就是这个作用
sfr P1 =0x90;
sbit P1_0 =P1 ^ 0;
sbit P1_7 =0x90 ^ 7;
sbit P1_1 =0x91;          //这里分别定义 P1 端口和引脚 P10、P11、P17
void main(void)
{
unsigned int a;
unsigned char b;
do{
for (a=0;a<50000;a++)
```

```
    P1_0 =0;                    //点亮 P1_0
    for (a=0;a<50000;a++)
    P1_7 =0;                    //点亮 P1_7
    for (b=0;b<255;b++)
    {
    for (a=0;a<10000;a++)
    P1 =b;                      //用 b 的值来做跑马灯的花样
    }
    P1 =255;                    //熄灭 P1 上的 LED
    for (b=0;b<255;b++)
    {
    for (a=0;a<10000;a++)       //P1_1 闪烁
    P1_1 =0;
    for (a=0;a<10000;a++)
    P1_1 =1;
    }
    while(1);
    }
```

本任务使学生掌握了 C51 程序的基本知识、数据类型、常量及变量的定义和使用，为助学生将来学习 51 单片机 C 语言程序控制打下了基础。

1.1.4 思考与练习

（1）请写出-29、32、-75 的补码。

（2）使用 C51 编程时，定义的变量没有指定存储器类型，默认的存储器类型是什么？这个存储器类型共有多大的存储空间？

（3）程序中要使用引脚 P1.0 和引脚 P1.3，如何定义？

1.2 任务二：51 单片机的结构、引脚功能、最小系统电路图

1.2.1 知识链接

1．单片机的概念

在一片集成电路芯片上集成微处理器、存储器、I/O 口电路，从而构成单芯片微型计算

机，即单片机。Intel 公司推出了 MCS-51 系列单片机：集成 8 位 CPU、4KB ROM、128B RAM、4 个 8 位并行 I/O 口、1 个全双工串行 I/O 口、2 个 16 位定时/计数器，寻址范围为 64KB，并有控制功能较强的布尔处理器。

2．MCS-51 单片机的结构特点

MCS-51 单片机的结构如下。

（1）内 ROM：4KB。

（2）内 RAM：128B。

（3）外 ROM：64KB。

（4）外 RAM：64KB。

（5）I/O 线：32 根。

（6）定时/计数器：两个 16 位可编程定时/计数器。

（7）串行口：全双工，两个。

（8）寄存器区：工作寄存器区，在 128B 内 RAM 中，分 4 个区。

（9）中断源：5 源中断，2 级优先。

（10）堆栈：最深 128B。

（11）布尔处理机：位处理机，某位单独处理。

（12）指令系统：5 大类，111 条。

图 1-2 中间的一条双横线是 MCS-51 单片机的内部总线，其他部件都是通过内部的总线与 CPU 相连接。下面对 MCS-51 单片机内部的单个部件进行讲解。

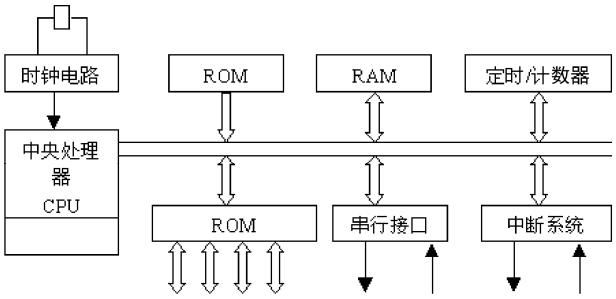

图 1-2　MCS-51 单片机结构图

总线（Bus）：总线是计算机各部件之间传送信息的公共通道。单片机中有内部总线和外部总线两类，内部总线是 CPU 内部之间的连线，外部总线是 CPU 与其他部件之间的连线。外部总线有 3 种：数据总线 DB（Data Bus）、地址总线 AB（Address Bus）和控制总线 CB（Control Bus）。

中央处理器（CPU）：MCS-51 单片机的 CPU 能处理 8 位二进制数或代码。CPU 是单片机的核心部件，包含运算器、控制器及若干个寄存器等部件。

内部数据存储器（RAM）：MCS-51 单片机芯片共有 256 个 RAM 单元，其中后 128 个单元被专用寄存器占用（稍后详解），前 128 个单位作为寄存器供用户使用，用于存放可读写的数据。因此，通常所说的内部数据存储器就是指前 128 个单元，简称内部 RAM，地址范围为 00H～FFH（256B），是一个多用、多功能数据存储器，有数据存储、通用工作寄存器、堆栈、位地址等空间。

内部程序存储器（ROM）：MCS-51 单片机内部有 4KB/8KB 的 ROM（51 系列单片机的 ROM 为 4KB，52 系列单片机的 ROM 为 8KB）用于存放程序、原始数据或表格，因此称为程序存储器，简称内部 RAM，地址范围为 0000H～FFFFH（64KB）。单片机产品系列表如表 1-5 所示。

表 1-5　单片机产品系列表

功能配置 / 芯片类型		ROM	EPROM	RAM	定时/计数器	I/O 口		中断源
						并行	串行	
51	8031	—	—	128KB	2×16 位	4×8 位	1	5
	8051	4KB	—	128KB	2×16 位	4×8 位	1	5
	8751	—	4KB	128KB	2×16 位	4×8 位	1	5
52	8032	—	—	256KB	3×16 位	4×8 位	1	6
	8052	8KB	—	256KB	3×16 位	4×8 位	1	6
	8752	—	8KB	256KB	3×16 位	4×8 位	1	6

定时/计数器：51 系列单片机共有两个 16 位的定时/计数器（52 系列单片机共有 3 个 16 位的定时/计数器）以实现定时或计数功能，并以其定时或计数结果对计算机进行控制。定时靠内部的分频时钟频率计数实现；当作计数器时，对 P3.4（T0）或 P3.5（T1）端口的低电平脉冲计数。

并行 I/O 口：MCS-51 单片机共有 4 个 8 位的 I/O 口（P0、P1、P2、P3）以实现数据的输入/输出。具体功能将在后面章节详细论述，MCS-51 单片机引脚如图 1-3 所示。

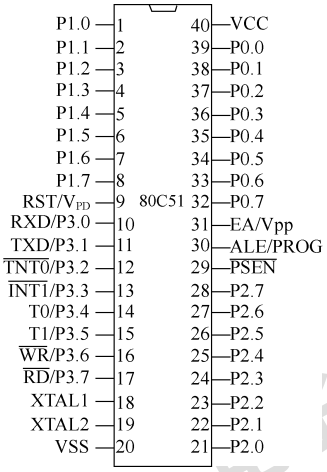

图 1-3　MCS-51 单片机引脚

串行 I/O 口：MCS-51 单片机有一个可编程的全双工的串行 I/O 口，以实现单片机和其他设备之间的串行数据传送。该串行 I/O 口功能较强，既可作为全双工异步通信收发器，也可作为移位器。RXD（P3.0）为接收端口，TXD（P3.1）为发送端口。

中断控制系统：MCS-51 单片机的中断功能较强，可满足不同控制应用的需要。51 系列单片机有 5 个中断源（52 系列单片机有 6 个中断源），即 2 个外中断源，2 个定时中断源，2 个串行中断源。全部中断源可分为高级和低级两个优先级别，优先级别的设置也将在后面进行详细的讲解。

定时与控制部件：MCS-51 单片机内部有一个高增益的反相放大器，其输入端为 XTAL1，输出端为 XTAL2。MCS-51 单片机内部有时钟电路，但石英晶体和微调电容需要外接。时钟电路为单片机产生时钟脉冲序列。

3．MCS-51 **单片机的引脚功能**

MCS-51 单片机是标准的 40 引脚双列直插式集成电路芯片，引脚分布请参照图 1-3。

P0.0～P0.7：P0 口 8 位双向口线（在引脚的 32～39 号端子）。

P1.0～P1.7：P1 口 8 位双向口线（在引脚的 1～8 号端子）。

P2.0～P2.7：P2 口 8 位双向口线（在引脚的 21～28 号端子）。

P3.0～P3.7：P3 口 8 位双向口线（在引脚的 10～17 号端子）。

P0 口有如下 3 个功能。

（1）外部扩展存储器时，当作数据总线使用（D0～D7 为数据总线接口）。

（2）外部扩展存储器时，当作地址总线使用（A0～A7 为低 8 位地址总线接口）。

（3）不扩展时，可当作一般的 I/O 口使用，但内部无上拉电阻，作为 I/O 口时应在外部接上拉电阻。

P1 口只当作 I/O 口使用，其内部有上拉电阻。

P2 口有如下两个功能。

（1）外部扩展存储器时，当作地址总线使用。

（2）当作一般 I/O 口使用时，其内部有上拉电阻。

P3 口有如下两个功能。

除可作为 I/O 口使用以外（其内部有上拉电阻），还有一些特殊功能，可由特殊功能寄存器来设置。

P3.0：RXD 串行 I/O 口输入。

P3.1：TXD 串行 I/O 口输出。

P3.2：INT0 外部中断 0 输入。

P3.3：INT1 外部中断 1 输入。

P3.4：T0 定时器 0 外部输入。

P3.5：T1 定时器 1 外部输入。

P3.6：WR 外部写控制。

P3.7：RD 外部读控制。

有 EPROM 的单片机芯片（如 8751）为写入程序提供专门的编程脉冲和编程电源，这些信号也是以信号引脚的形式提供的。

ALE/PROG（引脚 30）地址锁存允许/片内 EPROM 编程脉冲：ALE 为地址锁存控制信号，在系统扩展时，ALE 用于把 P0 口的输出低 8 位地址送锁存器锁存起来，以实现低位地址和数据的隔离。如果在 8051 扩展 2KB EEPROM 电路，ALE 与 74LS373 锁存器的 G 相连接，当 CPU 对外部进行存取时，用以锁住地址的低位地址，即 P0 口输出。ALE 是以晶振六分之一的固定频率输出的正脉冲，当系统中未使用外部存储器时，ALE 也会有六分之一的固定频率输出，因此可作为外部时钟或外部定时脉冲。PROG 功能：片内有 EPROM 的芯片，在 EPROM 编程期间，此引脚输入编程脉冲。

PSEN 外部程序存储器读选通信号：在读外部 ROM 时 PSEN 低电平有效，以实现外部 ROM 单元的读操作。

（1）内部 ROM 读取时，PSEN 不动作。

（2）外部 ROM 读取时，在每个机器周期会动作两次。

（3）外部 RAM 读取时，两个 PSEN 脉冲被跳过，不会输出。

（4）外接 ROM 时，与 ROM 的引脚 OE 相连接。

EA/VPP（引脚 31）访问程序存储器控制信号。

（1）接高电平时：CPU 先读取内部程序存储器（ROM），当读取内部程序存储器超过 0FFFH（8051）或 1FFFH（8052）时自动读取外部 ROM。

（2）接低电平时：CPU 读取外部程序存储器（ROM）。

（3）8751 烧写内部 EPROM 时，利用此引脚输入 21V 的烧写电压。

RST 复位信号：当输入的信号连续 2 个机器周期以上为高电平时即为有效，用以完成单片机的复位初始化操作。

XTAL1 和 XTAL2 外接晶振引脚。当使用芯片内部时钟时，这两个引脚用于外接石英晶体和微调电容；当使用外部时钟时，用于接外部时钟脉冲信号。

VCC：输入+5V 电源。

VSS：GND 接地，如图 1-4 所示。

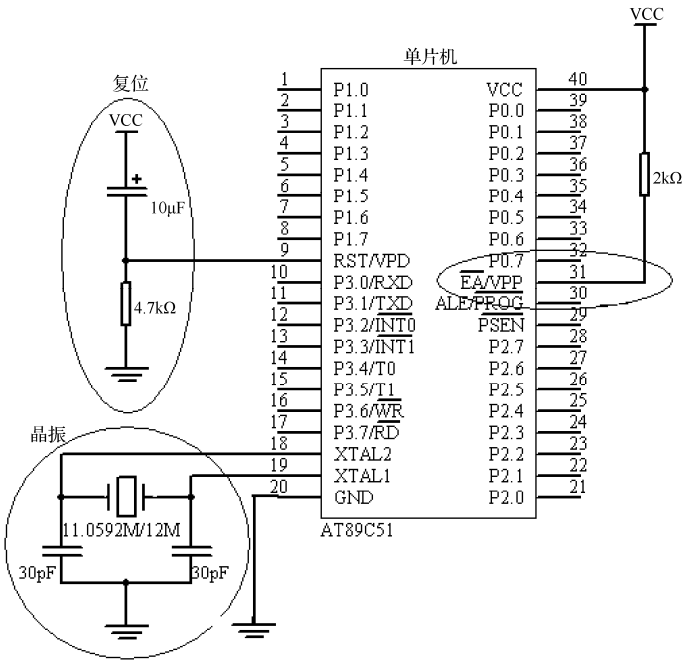

图 1-4　51 单片机最小系统图

4．MCS-51 单片机的存储器结构

MCS-51 单片机在物理结构上有 4 个存储空间（见图 1-5）：片内程序存储器、片外程序存储器、片内数据存储器、片外数据存储器。然而，逻辑上从用户的角度考虑，8051 单片机有 3 个存储空间：片内外统一编址的 64KB 的程序存储器地址空间；256B 的片内数据存储器的地址空间；64KB 片外数据存储器的地址空间。在访问 3 个不同的逻辑空间时，应采用不同形式的指令（具体内容在后面的指令系统学习中将会讲解），以产生不同的存储器空间的选通信号。

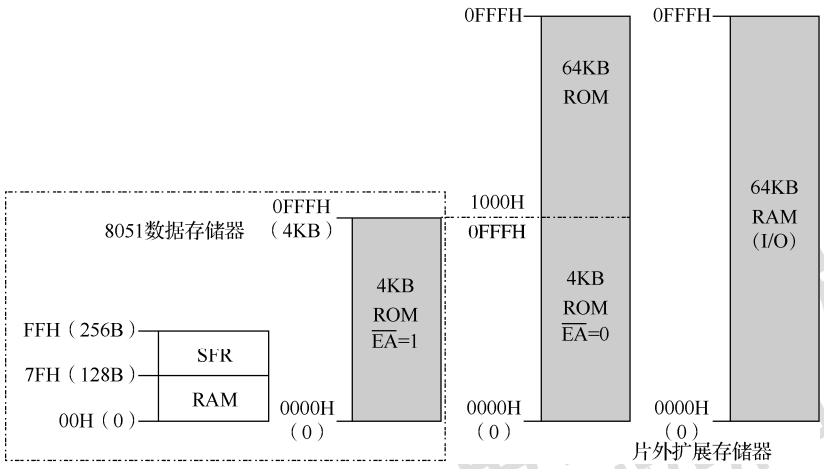

图 1-5　存储器结构

1）程序存储器

一个微处理器之所以能够聪明地执行某种任务，是因为它们除了有强大的硬件，还可以加载运行的软件。其实微处理器并不聪明，它们只是完全按照人们预先编写的程序执行而已。设计人员编写的程序存放在微处理器的程序存储器中，俗称只读程序存储器。程序相当于在微处理器处理问题时发布的一系列命令，其实程序和数据一样，都是由机器码组成的代码串，只是程序代码存放于程序存储器中。

MCS-51 单片机具有 64KB 的程序存储器寻址空间，用于存放用户程序、数据和表格等信息。对于内部无程序存储器的 8031 单片机，它的程序存储器必须外接，空间地址为 64KB，此时单片机的端必须接地，强制 CPU 从外部程序存储器读取程序。对于内部有程序存储器的 8051 等单片机，正常运行时接高电平，使 CPU 先从内部的程序存储器中读取程序，当 PC 值超过内部 ROM 的容量时，会自动转向外部的程序存储器读取程序。

当 EA=1 时，程序从片内程序存储器开始执行，当 PC 值超过片内 ROM 容量时会自动转向外部 ROM 空间。当 EA=0 时，程序从外部存储器开始执行，如前面提到的片内无程序存储器的 8031 单片机，在实际应用中就要把 8031 单片机的引脚接为低电平。

8051 单片机内有 4KB 的程序存储单元，其地址为 0000H～0FFFH，单片机启动复位后，程序计数器的内容为 0000H，因此系统将从 0000H 单元开始执行程序。但在程序存储单元中有些特殊的单元，在使用中应加以注意。

其中一组特殊的单元是 0000H～0002H，系统复位后，PC 为 0000H，单片机从 0000H 单元开始执行程序，如果单片机不是从 0000H 单元开始执行的，那么应该在这 3 个单元中存放一条无条件转移指令，让 CPU 直接去执行用户指定的程序。

另一组特殊的单元是 0003H～002AH，这 40 个单元各有用途，它们被均匀地分为 5 段，其定义如下。

0003H～000AH：外部中断 0 中断地址区。

000BH～0012H：定时/计数器 0 中断地址区。

0013H～001AH：外部中断 1 中断地址区。

001BH～0022H：定时/计数器 1 中断地址区。

0023H～002AH：串行中断地址区。

可见以上 40 个单元是专门用于存放中断处理程序的地址单元，中断响应后，按中断的类型自动转到各自的中断区去执行程序。从上文可以看出，每个中断服务程序只有 8 个字节单元，用 8 个字节来存放一个中断服务程序显然是不可能的。因此，以上地址单元不能用于存放程序的其他内容，只能存放中断服务程序。但通常情况下，我们在中断响应的地址区安放一条无条件转移指令，来指向程序存储器的其他真正存放中断服务程序的空间去执行，这样中断响应后，CPU 读到这条转移指令便会转向其他地方继续执行中断服务程序。

由上述内容可得，0000H～0002H 只有 3 个存储单元，这 3 个存储单元在程序存放时存放不了实际意义的程序。通常在实际编写程序时在这里安排一条 ORG 指令，利用 ORG 指令跳转到从 0033H 开始的用户 ROM 区域，再来安排程序语言。从 0033H 开始的用户 ROM 区域用户可以通过 ORG 指令任意安排，但在应用中需要注意，不要超过实际的存储空间，不然程序就会找不到。

2）数据存储器

数据存储器也称为随机存取数据存储器，可分为内部数据存储器和外部数据存储器。MCS-51 单片机内部 RAM 有 128B 或 256B 的用户数据存储空间（不同的型号有区别），片外最多可扩展 64KB 的 RAM，构成两个地址空间。访问片内 RAM 用“MOV”指令，访问片外 RAM 用“MOVX”指令，片内和片外 RAM 是用于存放执行的中间结果和过程数据的。MCS-51 单片机的数据存储器均可读写，部分单元还可以位寻址。

MCS-51 单片机的内部数据存储器在物理上和逻辑上都可分为两个地址空间，即数据存储器区（低 128 单元）和特殊功能寄存器区（高 128 单元），如图 1-6 所示。这两个空间是相连的，从用户角度而言，低 128 单元才是真正的数据存储器，下面将对此进行详细讲解。

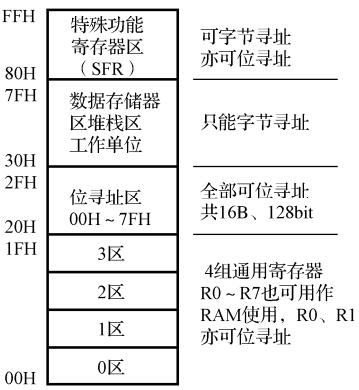

图 1-6　内部存储器的分布

片内数据存储器为 8 位地址，因此可寻址的最大范围为 256 个单元地址。对片外数据存储器采用间接寻址方式，R0、R1 和 DPTR 都可以作为间接寻址寄存器，R0、R1 是 8 位的寄存器，即 R0、R1 的寻址范围最大为 256 个单元，而 DPTR 是 16 位地址指针，寻址范围可达 64KB。也就是说在寻址片外数据存储器时，如果寻址范围超过了 256B，就不能用 R0、R1 作为间接寻址寄存器，而必须用 DPTR 寄存器作为间接寻址寄存器。从图 1-5 中可以看到，8051 单片机片内 RAM 共有 256 个单元（00H～FFH），这 256 个单元共分为两部分，一部分是 00H～7FH 单元(共 128B)为用户数据 RAM;另一部分是 80H～FFH 单元(共 128B)，为特殊寄存器（SFR）单元，从图 1-5 中可清楚地看出它们的结构分布。

（1）通用寄存器区（00H～1FH）。00H～1FH 共 32 个单元，被均匀地分为 4 块，每块包含 8 个 8 位寄存器，均以 R0～R7 来命名，通常称这些寄存器为通用寄存器。这 4 块中

的寄存器都称为 R0～R7，那么如何在程序中区分和使用它们呢？Intel 工程师安排了一个寄存器——程序状态字寄存器（PSW）来管理它们，CPU 只要定义这个 PSW 的 D3 和 D4 位（RS0 和 RS1），即可选中这 4 组通用寄存器。工作寄存器分布如表 1-6 所示。如果程序中并不需要用 4 组通用寄存器，那么可选中第 0 组工作寄存器，其余部分可用作一般的数据缓冲器。

表 1-6　工作寄存器分布

组	RS1	RS0	R0	R1	R2	R3	R4	R5	R6	R7
0	0	0	00H	01H	02H	03H	04H	05H	06H	07H
1	0	1	08H	09H	0AH	0BH	0CH	0DH	0EH	0FH
2	1	0	10H	11H	12H	13H	14H	15H	16H	17H
3	1	1	18H	19H	1AH	1BH	1CH	1DH	1EH	1FH

（2）位寻址区（20H～2FH）。片内 RAM 的 20H～2FH 单元为位寻址区，既可以作为一般单元用字节寻址，又可以对它们的位进行寻址。位寻址区共有 16B，128bit，位地址为 00H～7FH，RAM 位寻址区分布如表 1-7 所示。CPU 能直接寻址这些位，执行置“1”、清“0”、求“反”、转移、传送和逻辑等操作。通常称 MCS-51 单片机具有布尔处理功能，布尔处理的存储空间指的就是这些位寻址区。

表 1-7　RAM位寻址区分布

单元地址	MSB			位地址				LSB
2FH	7FH	7EH	7DH	7CH	7BH	7AH	79H	78H
2EH	77H	76H	75H	74H	73H	72H	71H	70H
2DH	6FH	6EH	6DH	6CH	6BH	6AH	69H	68H
2CH	67H	66H	65H	64H	63H	62H	61H	60H
2BH	5FH	5EH	5DH	5CH	5BH	5AH	59H	58H
2AH	57H	56H	55H	54H	53H	52H	51H	50H
29H	4FH	4EH	4DH	4CH	4BH	4AH	49H	48H
28H	47H	46H	45H	44H	43H	42H	41H	40H
27H	3FH	3EH	3DH	3CH	3BH	3AH	39H	38H
26H	37H	36H	35H	34H	33H	32H	31H	30H
25H	2FH	2EH	2DH	2CH	2BH	2AH	29H	28H
24H	27H	26H	25H	24H	23H	22H	21H	20H
23H	1FH	1EH	1DH	1CH	1BH	1AH	19H	18H
22H	17H	16H	15H	14H	13H	12H	11H	10H
21H	0FH	0EH	0DH	0CH	0BH	0AH	09H	08H
20H	07H	06H	05H	04H	03H	02H	01H	00H

（3）用户 RAM 区（30H～7FH）。在片内 RAM 低 128 单元中，通用寄存器占用 32 个单元，位寻址区占用 16 个单元，剩下的 80 个单元是供用户使用的一般 RAM 区，地址单元为 30H～7FH。对这部分区域的使用不进行任何规定和限制，但应说明的是，堆栈一般开

辟在这个区域。

（4）特殊功能寄存器区（高 128 单元）。特殊功能寄存器（SFR）也称专用寄存器，可反映 MCS-51 单片机的运行状态，很多功能是通过特殊功能寄存器来定义和控制程序的执行的。MCS-51 单片机有 21 个特殊功能寄存器，它们被离散地分布在内部 RAM 的 80H～FFH 地址中，这些寄存功能已经有了专门的规定，用户不能修改其结构。下面对其主要的特殊功能寄存器和程序计数器 PC（Program Counter）进行一些简单的介绍，如表 1-8 所示。

表 1-8　特殊功能寄存器

标 识 符 号	地　　址	寄存器名称
ACC	0E0H	累加器
B	0F0H	寄存器
PSW	0D0H	程序状态字
SP	81H	堆栈指针
DPTR	82H、83H	数据指针（16 位）含 DPL 和 DPH
IE	0A8H	中断允许控制寄存器
IP	0B8H	中断优先控制寄存器
P0	80H	I/O 口 0 寄存器
P1	90H	I/O 口 1 寄存器
P2	0A0H	I/O 口 2 寄存器
P3	0B0H	I/O 口 3 寄存器
PCON	87H	电源控制及波特率选择寄存器
SCON	98H	串行口控制寄存器
SBUF	99H	串行数据缓冲寄存器
TCON	88H	定时控制寄存器
TMOD	89H	定时/计数器方式选择寄存器
TL0	8AH	定时/计数器 0 低 8 位
TH0	8CH	定时/计数器 0 高 8 位
TL1	8BH	定时/计数器 1 低 8 位
TH1	8DH	定时/计数器 1 高 8 位

① 程序计数器（PC）。程序计数器在物理上是独立的，它不属于特殊内部数据存储器块。PC 是一个 16bit 的计数器，用于存放一条要执行的指令地址，寻址范围为 64KB，PC 有自动加 1 功能，即完成了一条指令的执行后，其内容自动加 1。PC 本身没有地址，因而不可寻址，用户无法对它进行读写，但是可以通过转移、调用、返回等指令改变其内容，以控制程序按要求去执行。

② 累加器（ACC）。累加器是一个最常用的专用寄存器，大部分单操作数指令的一个操作数取自累加器，很多双操作数指令中的一个操作数也取自累加器。加、减、乘、除运算的指令，以及运算结果都存放于累加器或 AB 寄存器对中。大部分的数据操作都会通过

累加器进行，它就像一个交通要道。在程序比较复杂的运算中，累加器会制约软件运行效率，但它的功能较多，地位也十分重要，以至于有些后来制作出的单片机集成了多累加器结构，或者使用寄存器阵列来代替累加器，即赋予更多寄存器以累加器的功能，以解决累加器的“交通堵塞”问题，提高单片机的软件运行效率。

③ 寄存器（B）。在乘法和除法指令中，乘法指令中的两个操作数分别取自累加器和寄存器，其结果存放于 AB 寄存器对中。在除法指令中，被除数取自累加器，除数取自寄存器，结果中的商存放在累加器中，余数存放在寄存器中。

④ 程序状态字（PSW）。程序状态字是一个 8 位寄存器，用于存放程序运行的状态信息，这个寄存器的一些位可由软件设置，有些位则由硬件运行时自动设置。寄存器的各位定义如下，其中 PSW.1 是保留位，未使用。程序状态字的功能说明如表 1-9 所示，其各个位的定义介绍如下。

表 1-9　程序状态字的功能说明

位序	PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.0
位标志	CY	AC	F0	RS1	RS0	OV	—	P

- PSW.7（CY）为进位标志，此位有两个功能：一是在执行某算数运算时，存放进位标志，可被硬件或软件置位或清零；二是在位操作中作为累加位使用。
- PSW.6（AC）为辅助进位标志，在进行加、减运算时若低 4 位向高 4 位进位或借位，则 AC 置位，否则被清零。AC 也常用于十进制调整。
- PSW.5（F0）为用户标志位，供用户设置的标志位。
- PSW.4、PSW.3（RS1 和 RS0）为寄存器组选择位，如表 1-6 所示。
- PSW.2（OV）为溢出标志。在带符号的加、减运算中，超出累加器所能表示的符号数有效范围（−128～+127）时，即产生溢出。如果 OV=1，那么表明运算结果错误，如果 OV=0，那么表明运算结果正确。

执行加法指令时，当位 6 向位 7 进位，而位 7 不向 C 进位时，OV=1；或者位 6 不向位 7 进位，而位 7 向 C 进位时，同样 OV=1。

在乘法指令中乘积超过 255 时，OV=1，表明乘积在 AB 寄存器对中。若 OV=0，则说明乘积没有超过 255，乘积只在累加器中。除法指令，OV=1，表示除数为 0，运算不被执行，否则 OV=0。

- PSW.0（P）为奇偶校验位。声明累加器的奇偶性，每个指令周期都由硬件来置位或清零，若累加器中值为 1 的位数是奇数，则 P 置位，否则清零。

⑤ 数据指针（DPTR）。数据指针为 16 位寄存器，编程时既可以按 16 位寄存器来使用，也可以按两个 8 位寄存器来使用，即高位字节寄存器 DPH 和低位字节寄存器 DPL。

DPTR 主要用来保存 16 位地址，当对 64KB 外部数据存储器寻址时，可作为间址寄存

器使用，此时使用如下两条指令：

```
MOVX    A, @DPTR
MOVX    @DPTR, A
```

在访问程序存储器时，DPTR 可用作基址寄存器，采用基址+变址寻址的方式访问程序存储器，这条指令常用于读取程序存储器内的表格数据。

```
MOVC    A, @A+@DPTR
```

⑥ 堆栈指针（SP）。堆栈是一种数据结构，它是一个 8 位寄存器（见图 1-7），用来指示堆栈顶部在内部 RAM 中的位置。系统复位后，SP 的初始值为 07H，堆栈实际上从 08H 开始。但从 RAM 的结构分布中可知，08H～1FH 属于 1～3 工作寄存器区，若编程时需要用到这些数据单元，则必须对堆栈指针 SP 进行初始化，原则上设在任何一个区域均可，但一般设在 30H～1FH 较为适宜。

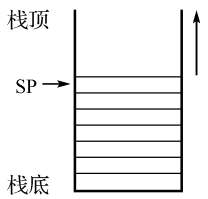

图 1-7　堆栈的分布

将数据写入堆栈称为入栈（PUSH，有些文献也称为插入运算或压入），从堆栈中取出数据称为出栈（POP，也称为删除运算或弹出）。堆栈最主要的特征是“后进先出”，即最先入栈的数据放在堆栈的最底部，而最后入栈的数据放在堆栈的顶部，因此最后入栈的数据是最先出栈的。这和往一个箱里存放书本一样，若想取出最先放入箱底部的书，则必须先取走最上层的书。

那么堆栈有何用途呢？设立堆栈是为了在中断操作和子程序调用时可以保存数据，即常说的断点保护和现场保护。微处理器无论是转入子程序还是转入中断服务程序，在执行完程序后，还是要回到主程序中来，在转入子程序或中断服务程序前，必须先将现场的数据保存，否则返回时，CPU 就不知道原来的程序执行到哪一步，原来的中间结果为多少。因此，在转入执行其他子程序前，先将需要保存的数据压入堆栈中保存，以备返回时再复原当时的数据，供主程序继续执行。

在转入子程序或中断服务程序时，需要保存的数据可能有若干个，都需要一一保留。如果微处理器要进行多重子程序或中断服务程序嵌套，那么需要保存的数据就更多，这要求堆栈要有相当的容量，否则会造成堆栈溢出，丢失应备份的数据。轻则使运算和执行结果错误，重则使整个程序紊乱。

MCS-51 单片机的堆栈是在 RAM 中开辟的，即堆栈要占据一定的 RAM 存储单元。同时，MCS-51 单片机的堆栈可以由用户设置，SP 的初始值不同，堆栈的位置则不同；不

同的设计人员，使用的堆栈区不同；不同的应用要求，堆栈要求的容量也有所不同。堆栈的操作只有两种，即进栈和出栈，但不管是向堆栈写入数据还是从堆栈读出数据，都是对栈顶单元进行的，SP 就是实时指示出栈顶的位置（即地址）。在子程序调用和中断服务程序响应的开始和结束期间，CPU 都是根据 SP 指示的地址与相应的 RAM 存储单元交换数据的。

堆栈的操作有两种方式。第一种是自动方式，即在中断服务程序响应或子程序调用时，返回地址自动进栈。当需要返回执行主程序时，返回的地址自动交给 PC，以保证程序从断点处继续执行，这种方式不需要编程人员干预。第二种方式是人工指令方式，使用专有的堆栈操作指令进行进栈、出栈操作，也只有两条指令：进栈操作为 PUSH 指令，在中断服务程序或子程序调用时作为现场保护；出栈操作为 POP 指令，用于子程序完成时为主程序恢复现场。

⑦ I/O 口专用寄存器（P0、P1、P2、P3）。I/O 口寄存器 P0、P1、P2 和 P3 分别是 MCS-51 单片机的 4 组 I/O 口锁存器。MCS-51 单片机并没有专门的 I/O 口操作指令，而是把 I/O 口也当作一般的寄存器来使用，数据传送都统一使用 MOV 指令来进行，这样做的好处在于，4 组 I/O 口还可以当作寄存器直接寻址方式参与其他操作。

⑧ 定时/计数器（TL0、TH0、TL1 和 TH1）。MCS-51 单片机中有两个 16 位的定时/计数器 T0 和 T1，它们由 4 个 8 位寄存器组成，两个 16 位定时/计数器却是完全独立的。可以单独对这 4 个寄存器进行寻址，但不能把 T0 和 T1 当作 16 位寄存器来使用。

⑨ 定时/计数器方式选择寄存器（TMOD）。TMOD 是一个专用寄存器，用于控制两个定时/计数器的工作方式，TMOD 可以用字节传送指令设置其内容，但不能位寻址，各位的定义如表 1-10 所示，更详细的内容，将在下文讲解。

表 1-10　定时/计数器工作方式控制寄存器

位序	D7	D6	D5	D4	D3	D2	D1	D0
位标志	GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
	定时/计数器 1				定时/计数器 0			

⑩ 串行数据缓冲寄存器（SBUF）。串行数据缓冲寄存器用来存放需要发送和接收的数据，它由两个独立的寄存器组成，一个是发送缓冲器，另一个是接收缓冲器，要发送和接收的操作其实都是对串行数据缓冲寄存器进行的。

除了以上简述的几个专用寄存，还有 IP、IE、TCON、SCON 和 PCON 等特殊功能寄存器，这几个特殊功能寄存器主要用于中断和定时，将在下文详细说明。

1.2.2　任务分析

利用 AT89C51 单片机控制报警灯，需要使用 I/O 端口，编写的控制程序涉及单片机内部数据存储区和程序存储区，以及部分特殊功能寄存器的定义和使用。

1.2.3　任务实施

1．准备工作

1）工具准备

（1）多媒体教室。

（2）常用开发软件，如 KEIL C51、Proteus 等。

（3）计算机，每人一组，每组一台。

2）知识准备

（1）8051 单片机的外部引脚及功能。

（2）8051 单片机的存储器的分布及功能。

2．任务要求

掌握 51 单片机的结构及引脚功能、最小系统电路图。掌握用单片机来控制一个 LED 小灯闪烁的原理及简单程序的编写，参考图 1-7。

利用所学知识，编写 C 语言程序控制 LED 小灯以不同的方式闪烁。

参考程序：

```
#include <reg52.h>
#define uchar unsigned char
#define uint unsigned int   //float
sbit LED = P1^0;            //P1.0
void DelayMS(uint x)
{
    uchar i;
 while(x--)
 {
    for(i=120;i>0;i--);     //1ms
 }
}
void main()
{
    while(1)
 {
```

```
        LED = ~LED;
        DelayMS(150);           //150ms
    }
}
```

利用 51 单片机控制一个 LED 小灯的电路图如图 1-8 所示。

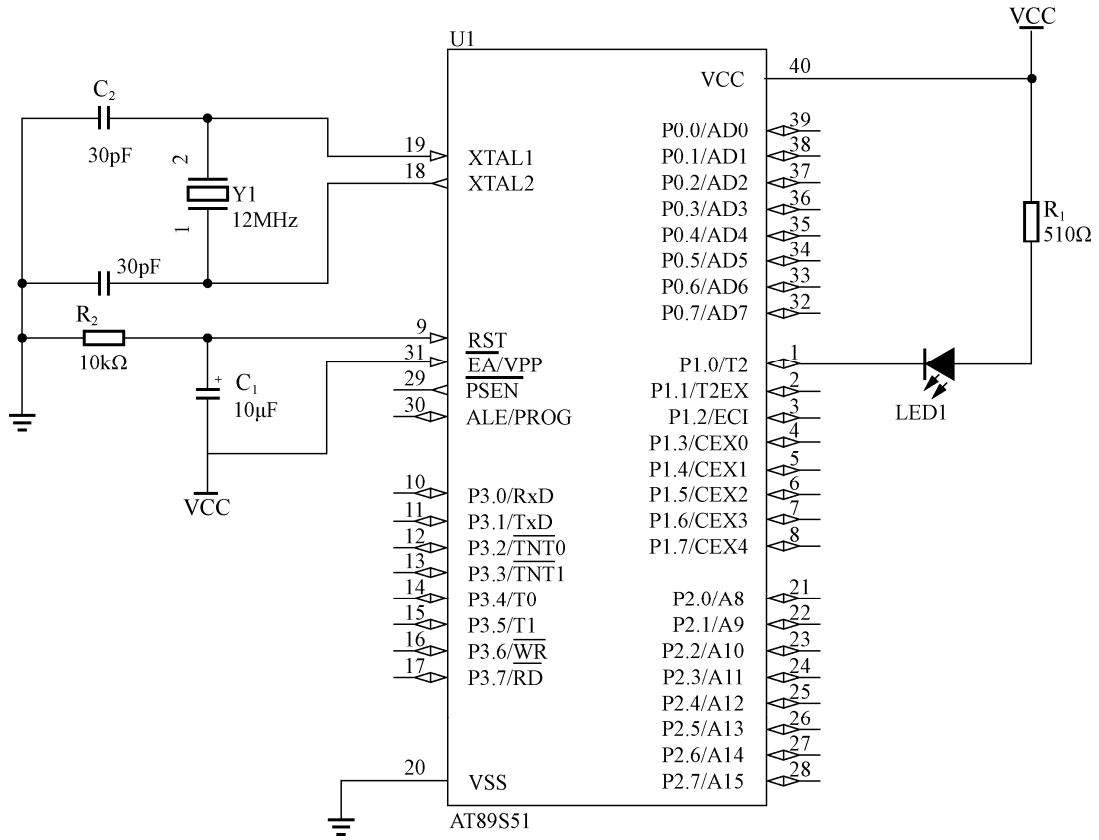

图 1-8　利用 51 单片机控制一个 LED 小灯的电路图

1.2.4　归纳总结

本任务让学生对 51 系列单片机的内部结构及引脚有了系统的认识，对自己需要编程的引脚有了理性的认识。

1.2.5　思考与练习

（1）MCS-51 单片机的片内总体结构中的 9 个部件分别是什么？各起什么作用？这 9

个部件之间有什么联系？

（2）怎样通过 MCS-51 单片机片内总体结构图来区分 8051、8751、8031 芯片？

1.3　任务三：KEIL C51 软件的操作及点亮一个 LED

1.3.1　知识链接

1．KEIL C51 **软件的使用**

使用 C 语言肯定要用到 C 编译器，以便把写好的 C 程序编译为机器码，这样单片机才能执行编写好的程序。KEIL μVISION2 是众多单片机应用开发软件中较好的软件之一，它支持众多不同公司的 MCS-51 架构的芯片，集编辑、编译、仿真等功能于一体，同时还支持 PLM、汇编和 C 语言的程序设计。它的界面和常用的微软 Visual C++的界面相似，界面友好、易学易用，在调试程序、软件仿真方面也有很强大的功能。因此，很多开发 51 单片机的工程师或普通的单片机爱好者，都十分喜欢它。

以上简单介绍了 KEIL C51 软件，它如何安装呢？

KEIL C51 软件是一个商业软件，普通爱好者可以到 KEIL 中国代理周立功公司的网站上下载一份能编译 2KB 的 Demo 版软件，该软件基本可以满足一般的个人学习和小型应用的开发（安装的方法和普通软件相当这里不再进行介绍）。

安装好后，双击软件图标运行 KEIL C51 软件，运行几秒后，出现如图 1-9 所示的屏幕。然后，按下面的步骤建立第一个项目。

图 1-9　启动时的屏幕

（1）单击“Project”菜单，在弹出的下拉菜单中选择“New Project”选项，如图 1-10 所示。接着弹出一个标准的 Windows 文件对话框（见图 1-11），在“文件名”的文本框中输入第一个 C 语言程序项目名称，这里用“test”，保存后的文件扩展名为 Uv2，这是 KEIL

μVISION2 项目文件扩展名，直接单击此文件也可以打开先前做的项目。

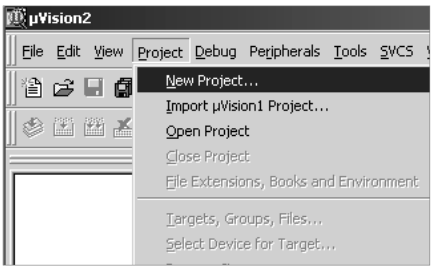

图 1-10　选择“New Project”选项

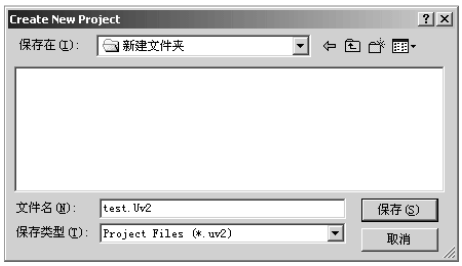

图 1-11　文件对话框

（2）选择需要的单片机，这里选择常用的 Atmel 公司的 AT89C51，如图 1-12 所示。

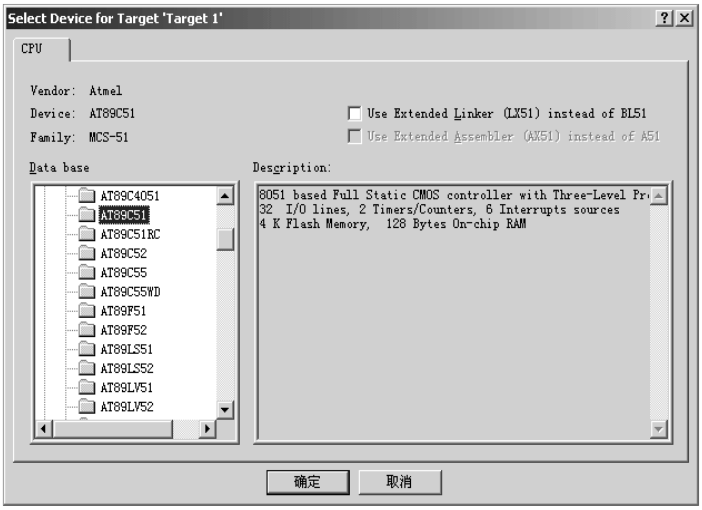

图 1-12　选取单片机

（3）先在项目中创建新程序文件或加入旧程序文件。如果没有现成的程序，那么就要新建一个程序文件。KEIL 中有一些程序的 Demo，这里还是以一个 C 语言程序为例来介绍如何新建一个 C 语言程序及如何将其加到第一个项目中。单击图 1-13 中 1 对应的新建文件快捷按钮，在 2 中出现一个新的文本编辑窗口，这个操作也可以通过单击“File”→“New”菜单或按“Ctrl+N”快捷键来实现。当光标已经出现在文本编辑窗口中时，可以输入程序。下面是经典的一段程序：

```
#include <AT89X51.H>
#include <stdio.h>
sbit D2=P1^0;
unsigned int a;
void main()
{
  D2=0;
}
```

这段程序的功能是令引脚 P1.0 的 LED 小灯亮，那么如何把它加到项目中，又如何编译试运行呢？

（4）单击图 1-13 中 3 对应的功能按钮保存新建的程序，也可以通过单击“File”→“Save”菜单或按“Ctrl+S”快捷键保存程序。因为是新文件所以保存时会弹出类似图 1-11 中的对话框，把第一个程序命名为 Test1.c，保存在项目所在的目录中，这时会发现程序单词有了不同的颜色，这说明 KEIL 的 C 语法检查生效了。如图 1-14 所示，在屏幕左边的“Source Group1”文件夹图标上右击，弹出菜单，在这里可以进行增加或减少项目文件等操作。单击“Add Files to Group 'Source Group 1'”弹出文件窗口，选择刚刚保存的文件，单击“ADD”按钮，关闭对话框，程序文件就加到项目中了。这时在“Source Group1”文件夹图标左边出现一个小“+”号，说明文件组中有了文件，单击它可以展开查看。

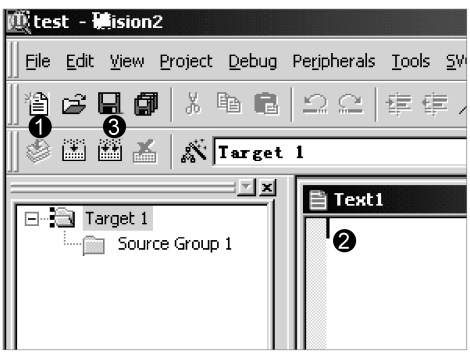

图 1-13　新建程序文件

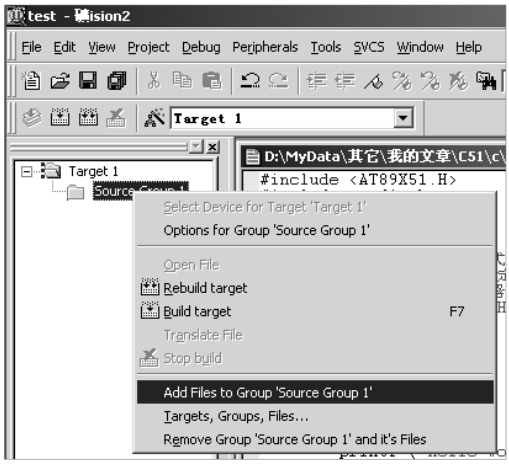

图 1-14　把文件加入项目文件组中

（5）C 语言程序文件已被加到项目中，下面为编译运行操作。这个项目只是用来学习新建程序项目和编译运行仿真，因此使用软件默认的编译设置，不会生成用于芯片烧写的 HEX 文件。

图 1-15 中 1、2、3 对应的功能按钮都是编译按钮，1 对应的功能按钮用于编译单个文件；2 对应的功能按钮是编译当前项目，如果先前编译过一次之后文件没有做过编辑改动，这时再单击该按钮是不会重新编译的；3 对应的功能按钮是重新编译，每单击一次均会再次编译链接一次，不管程序是否有改动；在 3 右边的是停止编译按钮，只有单击了前 3 个功能按钮中的任意一个，停止编译按钮才会生效；5 是“Project”菜单中的 3 个编译按钮；在 4 中可以看到编译的错误信息和使用的系统资源情况等，以后可以通过查看 4 来查错；6 的左边有一个形状类似放大镜的按钮，这是开启/关闭调试模式按钮，它也存在于“Debug”菜单“Start\Stop Debug Session”中，其对应的快捷键为“Ctrl+F5”。

图 1-15　编译程序

（6）进入调试模式，软件窗口样式如图 1-16 所示。图 1-16 中 1 为运行按钮，当程序处于停止状态时此按钮才有效。2 为停止按钮，程序处于运行状态时此按钮才有效。3 是复位按钮，模拟芯片的复位，程序回到最开头处执行。单击 4 可以打开 5 串行调试窗口，在这个窗口中可以看到从 51 芯片的串行 I/O 口输入/输出的字符。这些功能在菜单中也有，这里不再赘述，其他功能会在后文中介绍。首先单击 4 打开串行调试窗口，再单击运行键，这时就可以看到串行调试窗口中不断出现“Hello World!”，这样就完成了第一个项目。最后停止程序运行回到文件编辑模式中，先单击停止按钮，再单击开启/关闭调试模式按钮。

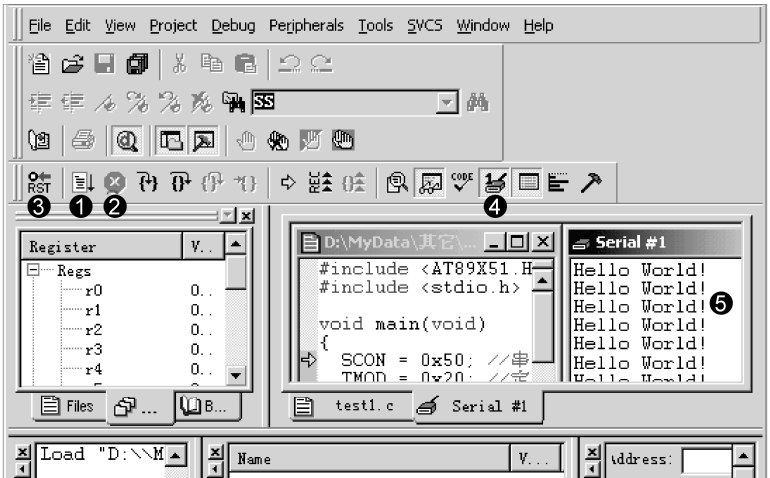

图 1-16　调试运行程序

（7）找到 test.Uv2 的文件就可以打开先前的项目，然后右击图 1-17 中的项目文件夹，弹出项目功能菜单，选择“Options for Target'Target1'”选项，弹出项目选项设置窗口。同样先选中项目文件夹图标，这时在“Project”菜单中也有一样的菜单可选。打开项目选项窗口，转到“Output”选项，如图 1-18 所示。图 1-18 中 1 对应的是选择编译输出的路径，2 对应

的是设置编译输出生成的文件名，3 对应的是决定是否要创建 HEX 文件，选中它就可以将 HEX 文件输出到指定的路径。再将它重新编译一次，很快在编译信息窗口中就会显示已将 HEX 文件创建到指定的路径了，如图 1-19 所示。

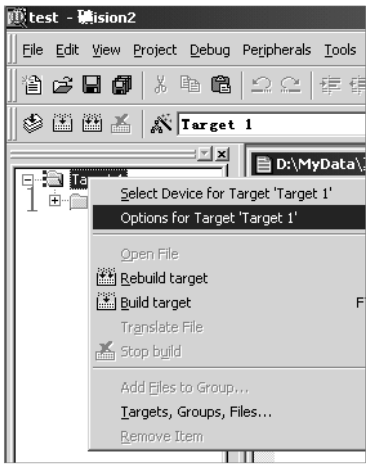

图 1-17　项目功能菜单

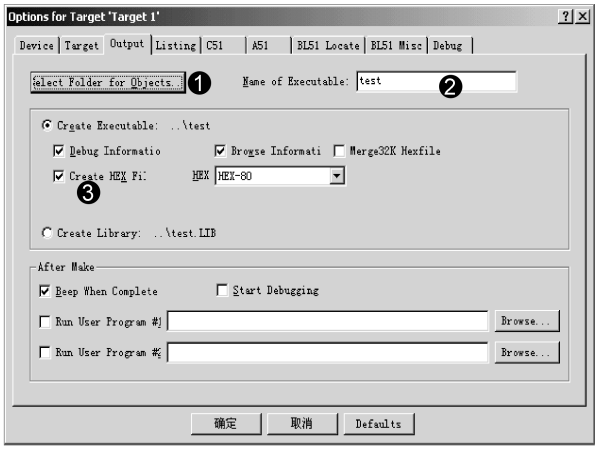

图 1-18　项目选项窗口

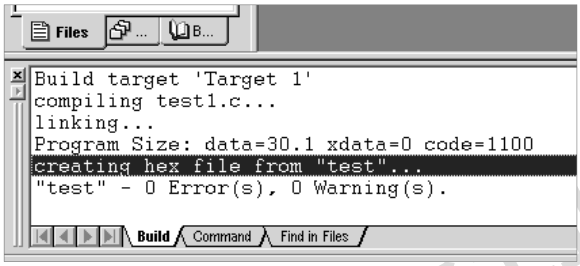

图 1-19　编译信息窗口

2．Proteus **软件的使用**

Proteus 软件是英国 Labcenter Electronics 公司出版的 EDA 工具软件（该软件的中国总代理为广州市风标电子技术有限公司）。它不仅具有其他 EDA 工具软件具有的仿真功能，还具有仿真单片机及外围器件功能，是目前比较好的仿真单片机及外围器件的工具，受到了单片机爱好者、从事单片机教学的教师、致力于单片机开发应用的科技工作者的青睐。同时，Proteus 软件还是一个巨大的教学资源，可以用于模拟电路与数字电路的教学实验、单片机与嵌入式系统的教学实验、微控制器系统的综合实验、创新实验与毕业设计等。

Proteus 软件可以为广大学生打造一个口袋实验室。在 Proteus 家族里，Proteus 8.0 版本的出现具有重要的意义，它也与其他版本有很明显的区别，将对其进行详细介绍。

1）创建新工程

在已经安装 Proteus 8.0 版本的前提下，选择并打开 Proteus 8 Professional，先新建一个工程，因为本教程与 PCB 绘制教程相关联，所以此时的工程是一个带有原理图、PCB 和源代码编译部分的工程。

如图 1-20 和图 1-21 所示，单击“File”菜单，选择“New Project”选项，将出现新建工程向导部分，在此可以设置文件名（Name）和保存路径（Path）。

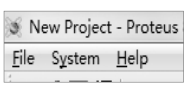

图 1-20　新建工程

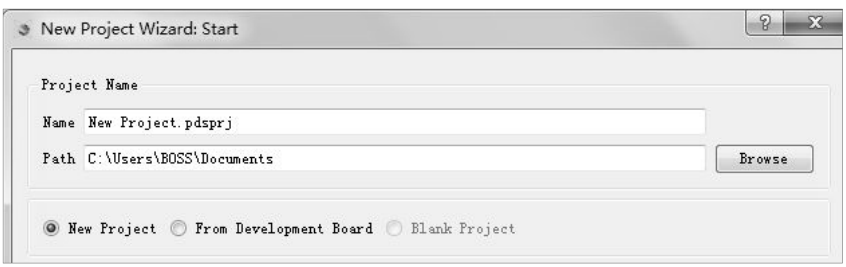

图 1-21　设置文件名和保存路径

单击“Next”按钮，在出现的窗口的顶部选项卡中，选择“Create a schematic from the selected template”（从选中的模板中创建原理图）选项，然后选择“DEFAULT”（默认）选项。如果不需要绘制原理图，可直接选择“Do not create a schematic” 选项。

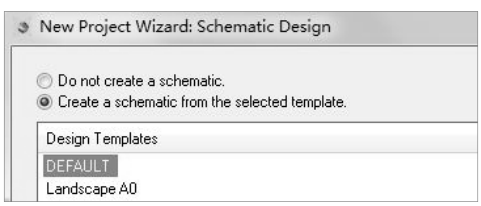

图 1-22　创建原理图

继续单击“Next”按钮，选择“Create a PCB layout from the selected template”（从选择的模板中创建 PCB 设计）选项，然后继续选择“DEFAULT”（默认）选项。如果不需要进

行 PCB 设计，可直接选择“Do not create a PCB layout”选项，如图 1-23 所示。

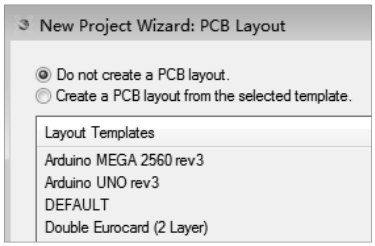

图 1-23　设置创建 PCB 制版图

因为本书只讲解需要仿真的设计，所以继续单击“Next”按钮，在仿真页面选择“Create Firmware Project”选项，并设置 Family（系列下拉列表）为“8051”，Contoller（控制器）下拉列表为“AT89C51”，Compiler 下拉列表（编译器）为“ASEM-51（Proteus）”，也就是在此设计外部代码编译器。如果不需要进行仿真，那么可直接选择“No Firmware Project”选项，如图 1-24 所示。

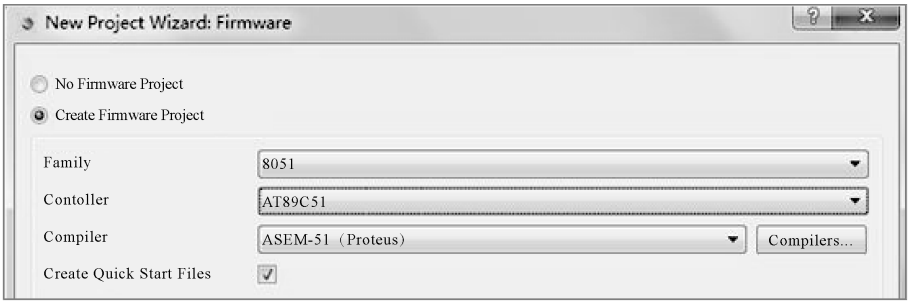

图 1-24　设置创建 Firmware Project

单击“Next”按钮，新工程创建完毕。此时的新工程包含原理图设计部分、PCB 绘制部分和源代码部分。

2）绘制原理图

（1）打开 P 元器件库，如图 1-25 所示。

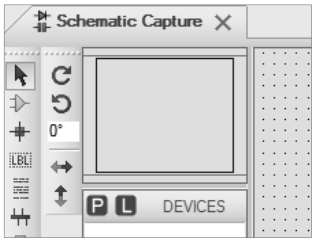

图 1-25　打开 P 元器件库

（2）在元器件库中输入关键字如图 1-26 所示。

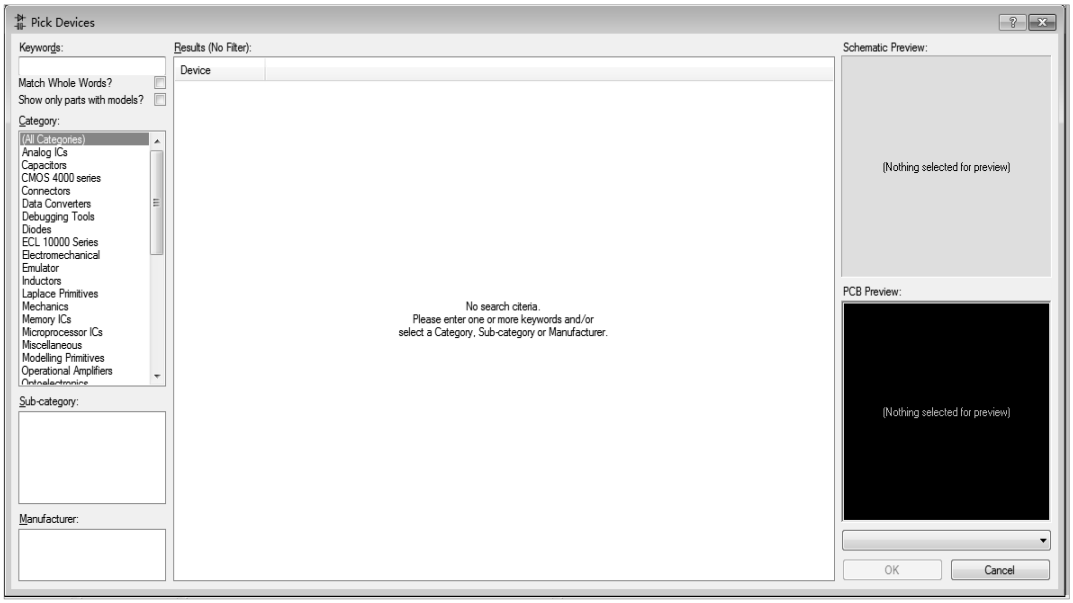

图 1-26　在元器件库中输入关键字

（3）找到元器件并双击，选中它到工作区域，如图 1-27 所示。

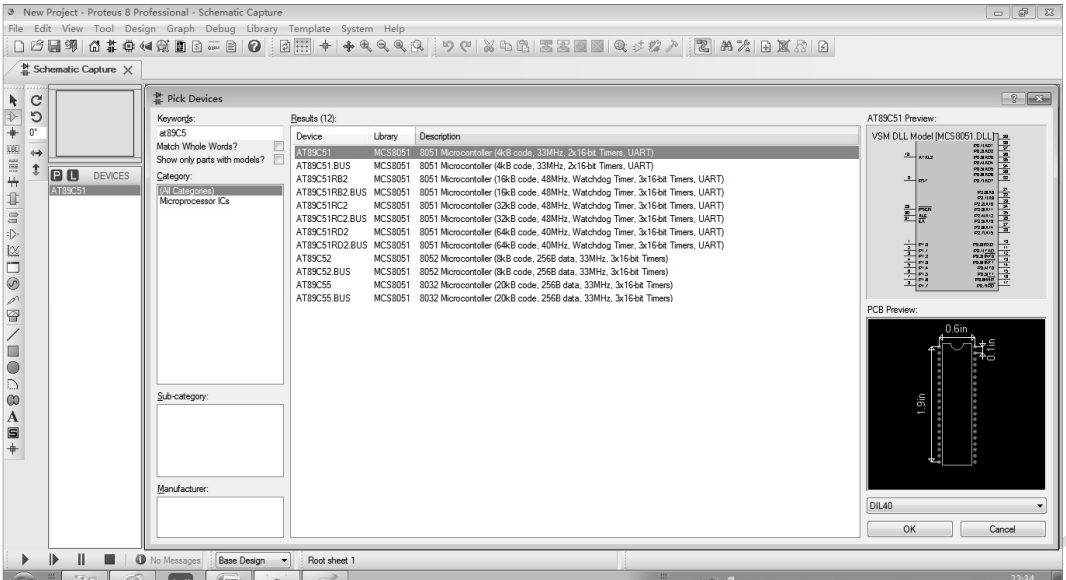

图 1-27　选择元器件

（4）在绘图区域内选中元器件，并将其旋转到合适的位置，再将鼠标指针放在绘图区域里单击，出现元器件的影子，将其放在合适的位置，再单击，元器件就摆放好了，如图 1-28 和图 1-29 所示。

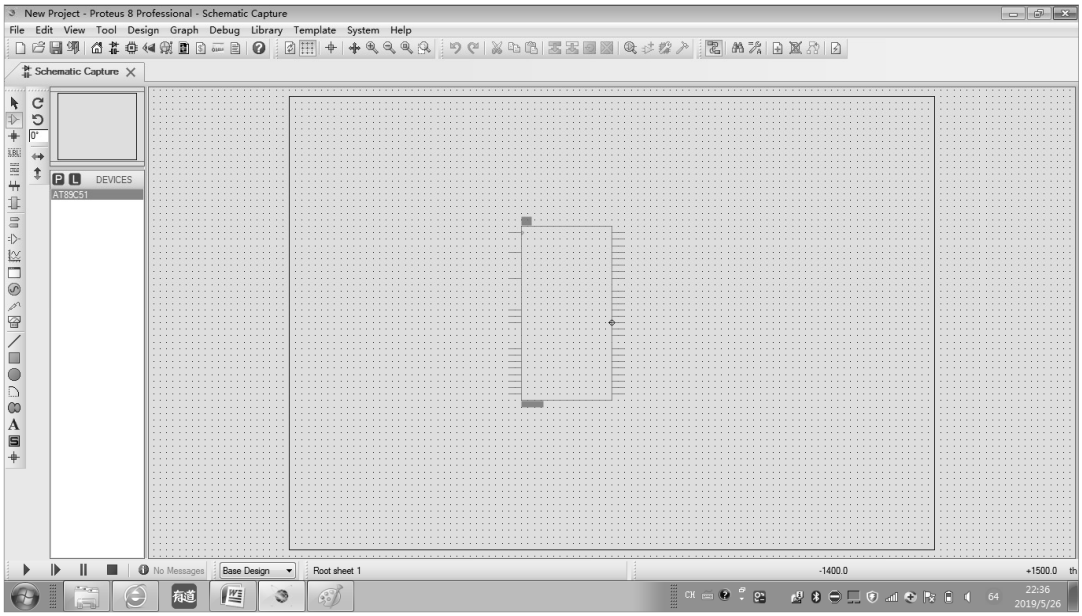

图 1-28　元器件选择位置

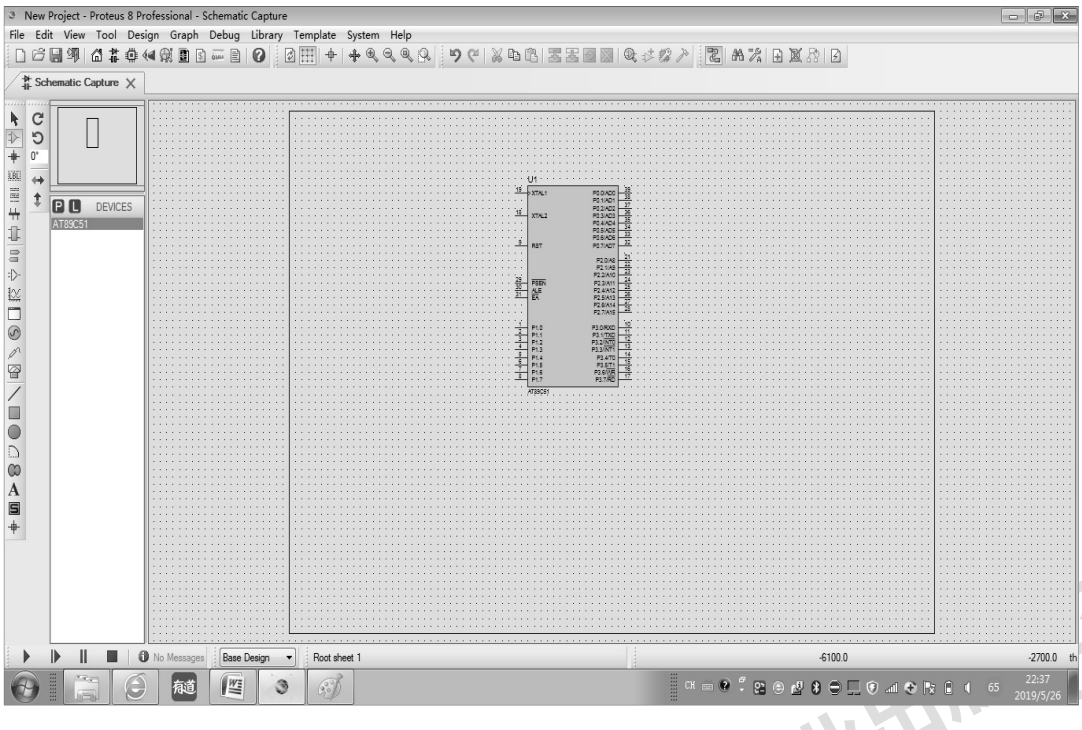

图 1-29　摆放元器件

（5）连接引脚，将鼠标指针移到需要导通的两个引脚中的一根上并单击，此时会出现导线，然后将鼠标指针移到另一个引脚上，再单击，引脚连接完成，如图 1-30 所示。

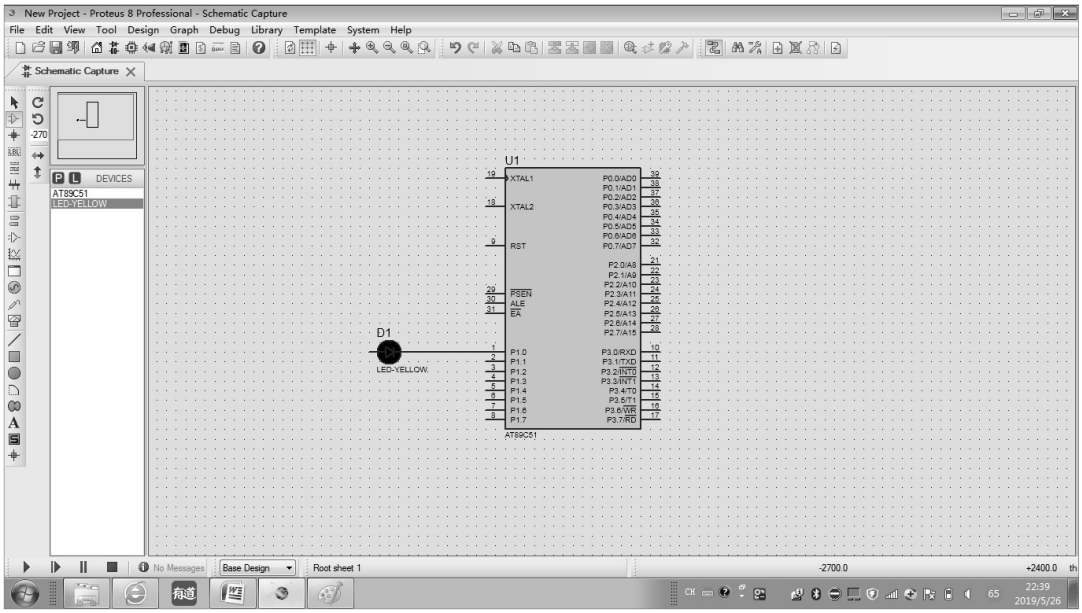

图 1-30　连接引脚

（6）双击单片机芯片，加载 KEIL C51 编译生成的 HEX 文件，并加载到“Program File”文本框内，如图 1-31 所示。

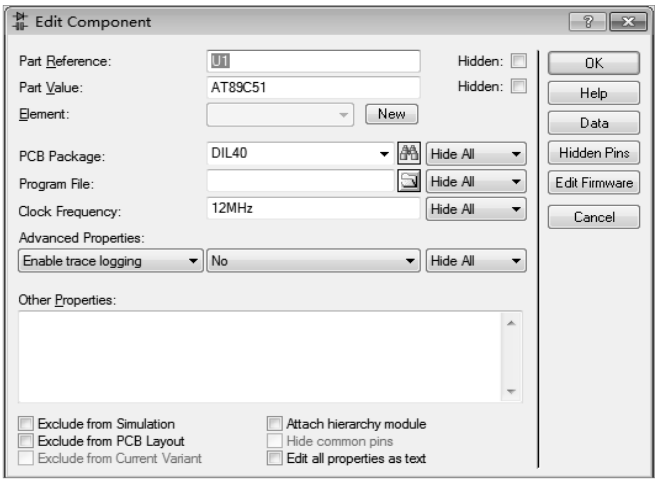

图 1-31　加载程序

（7）仿真，观察动态效果，如图 1-32 所示。

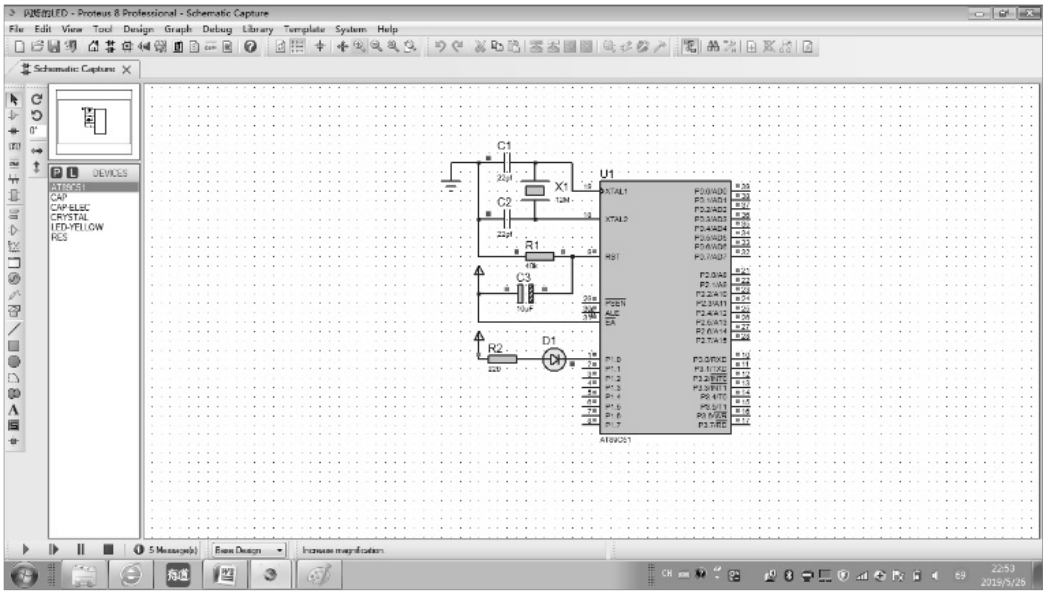

图 1-32　动态图

1.3.2　任务分析

先安装和使用 KEIL C51 软件，了解 51 单片机的结构，在对引脚 P1 的功能有了详细了解后，进行编程，实现将接在引脚 P1.0 的一个 LED 小灯点亮的功能。

1.3.3　任务实施

1．准备工作

1）工具准备

（1）多媒体教室。

（2）常用开发软件，如 KEIL C51、Proteus、CodeWarrior 等。

（3）单片机实验仪器，2~3 人一组，每组一台。

（4）计算机，2~3 人一组，每组一台。

2）知识准备

（1）单片机的基本原理。

（2）C 语言程序设计原理。

2．任务要求

掌握 KEIL C51 软件的基本操作，新建项目和程序，调试程序，生成目标程序，完成单片机点亮一个 LED 小灯的硬件电路设计和软件设计。

AT89C51 最小化系统的电路图如图 1-33 所示。

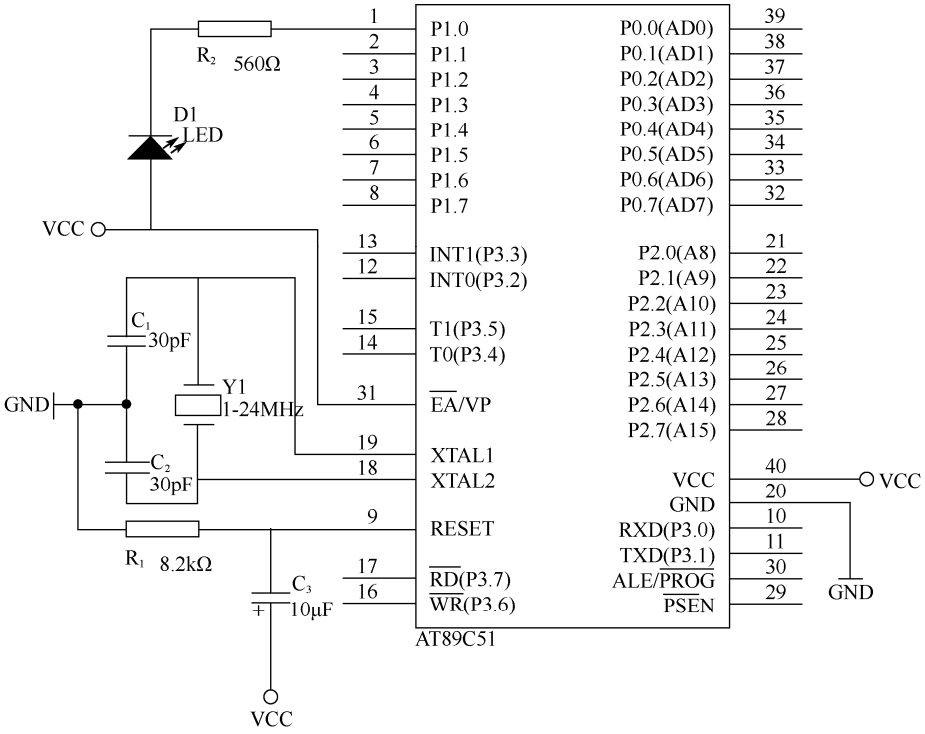

图 1-33　AT89C51 最小化系统的电路图

参考程序：

```
#include<reg52.h>
sbit D2=P1^2;
unsigned int a;
void main()
{
    D2=0;
/*   for(a=0;a<=10000;a++){};
 D2=1;
 for(a=0;a<=10000;a++){};*/
}
```

KEIL C51 软件编译器所支持的注释语句，一种是以“//”符号开始的语句，符号之后的语句都被视为注释，直到有回车换行；另一种是在“/*”符号之内的为注释，注释不会被 C 编译器编译。刚开始时 LED 小灯不亮（因为上电复位后所有的 I/O 口都置引脚 1 为高电平），延时一段时间［for (a=0; a<50000; a++)这句在运行］后 LED 小灯亮，再延时，LED 小

灯灭，然后交替亮、灭。至此，第一个真正的小应用就做完了。若没有这样的效果则需要认真检查电路或编译烧写的步骤，将 HEX 文件烧写入实验板，观察实验效果。

1.3.4　归纳总结

本任务令学生初步学习了一些 KEIL μVISION2 的项目文件的创建、编译、运行和软件仿真的基本操作方法，其中一些功能的快捷键的使用将在实际的开发应用中大大提高工作效率。