



第9章

单片机C51语言程序设计



9.1 单片机的C语言概述

- (1) 上手快。
- (2) 无须考虑底层细节。
- (3) 程序可读性好，易于维护。
- (4) 提高了开发效率。
- (5) 代码可扩展性、可重用性好。
- (6) 可移植性好。

- (1) 代码效率不如汇编语言，需要占用较多的单片机存储器资源。
- (2) 代码执行效率往往不如优化过的汇编语言代码，即使经过C语言编译器优化过也如此。
- (3) 代码调试、代码跟踪不如汇编语言调试直观。

9.2 C51入门

9.2.1 C51的数据类型

表 9-1 C51 支持的数据类型

数据类型	说明	长度	值域
unsigned char	无符号字符型	单字节	0~255
signed char	有符号字符型	单字节	-128~+127
unsigned int	无符号整型	双字节	0~65535
signed int	有符号整型	双字节	-32768~+32767
unsigned long	无符号长整型	四字节	0~4294967295
signed long	有符号长整型	四字节	-2147483648~+2147483647
float	浮点型	四字节	$\pm 1.175494\text{E}-38 \sim \pm 3.402823\text{E}+38$
*	指针型	1~3 字节	对象的地址
bit	位标量	位	0 或 1
sfr	8 位特殊功能寄存器型	单字节	0x80~0xff
sfr16	16 位特殊功能寄存器型	双字节	0~65535
sbit	可寻址位	位	0 或 1



9.2 C51入门

9.2.1 C51的数据类型

表 9-2 存储器类型的编码值

存储类型 I	Idata/data/bdata	xdata	pdata	Code
编码值	0x00	0x01	0xFE	0xFF

9.2 C51入门

9.2.2 特殊功能寄存器

(1) sfr。

(2) sfr16。

(3) sbit。

① **sbit**位变量名=特殊功能寄存器名^位位置。

```
sfr_name^int_constant。
```

```
sfr PSW=0xD0; //声明PSW 为特殊功能寄存器，地址为0xD0
```

```
sfr IE=0xA8;
```

```
sbit OV=PSW^2;
```

```
sbit EA=IE^7; //指定IE的第7位为EA，即中断允许
```

② **sbit**位变量名=字节地址名^位位置。

```
int_constant^int_constant。
```

```
sbit OV=0xD0^2;
```

```
sbit CY=0xD0^7;
```

```
sbit EA=IE^7; //指定IE 的第7 位为EA，即中断允许
```

③ **sbit**位变量名=位地址。

```
sbit OV=0xD2;
```

```
sbit CY=0xD7;
```

```
sbit EA=0xAF;
```

9.2 C51入门

9.2.3 C51的存储类型

表 9-3 存储区描述

存储类型	描述	地址
data	RAM的低128B,可在一个周期内直接寻址	00H~7FH
bdata	DATA区可字节、位混合寻址的16B区	20H~2FH
idata	RAM区的高128B,必须采用间接寻址	00H~FFH
pdata	外部存储区的256B,用MOVX @R0指令访问	00H~FFH
xdata	片外RAM(64KB),用MOVX @DPTR指令访问	0000H~FFFFH
code	程序存储区(64KB),用MOVC @A+DPTR指令访问	0000H~FFFFH



9.2 C51入门

9.2.3 C51的存储类型

1. DATA区

```
unsigned char data system_status=0;
unsigned int data unit_id[2];
char data inp_string[16];
float data outp_value;
mytype data new_var;
```

2. BDATA区


```
unsigned char bdata status_bute;
unsigned int bdata status_word;
unsigned long bdata status_dword;
sbit stat_flag=status_byte^4;
if (status_word^15)
{
...
}
stat_flag=1;
```



9.2 C51入门

9.2.3 C51的存储类型

```
typedef union{                                //声明联合体类型
unsigned long lvalue;                          //长整形32位
float fvalue;                                  //浮点数32位
}bit_float;                                    //联合体名
bit_float bdata myfloat;                       //在BDATA区中声明联合体
sbit float_ld=myfloat^31                      //声明位变量名
```

```
unsigned char data byte_status=0x43;
unsigned char bdata bit_status=0x43;
sbit status_3=bit_status^3;
bit use_bit_status (void);
bit use_byte_status (void);
void main ()
{
    unsigned char temp=0;
    if (use_bit_status ())
    {temp++; }
    if (use_byte_status ())
    {temp++; }
    if (use_bitnum_status ())
    {temp++; }
    bit use_bit_status (void)
    {return (bit) (status_3); }
    bit use_bitnum_status (void)
    {return (bit) (status^3); }
    bit use_byte_status (void)
    {return byte_status&0x04; }
```

```
//声明一个字节宽状态寄存器
//声明一个可位寻址状态寄存器
//把bit_status的第3位设为位变量
```

```
//如果第3位置位， temp加1
```

```
//如果第3位置位， temp再加1
```

```
//如果第3位置位， temp再加1
```



9.2 C51入门

9.2.3 C51的存储类型

3. IDATA区

```
unsigned char idata system_status=0;           //定义无符号字符型变量
unsigned int idata unit_id[2];                 //定义无符号整型数组
char idata inp_string[16];                    //定义无符号字符型数组
float idata outp_value;                       //定义浮点型变量
```

9.2 C51入门

4. PDATA区和XDATA区

```
unsigned char xdata system_status=0; //定义无符号字符型变量
unsigned int pdata unit_id[2]; //定义无符号整型数组，有两个数据单元
char xdata inp_string[16]; //定义有符号字符型数组，有16个数据单元
float pdata outp_value; //定义浮点型变量
```

```
#include<reg51.h>
unsigned char pdata inp_reg1; //在pdata定义无符号字符型变量
unsigned char xdata inp_reg2; //在xdata定义无符号字符型变量
void main () {
inp_reg1=P1; //对变量赋值，用MOVX @R0指令
inp_reg2=P3 //对变量赋值，用MOVX @DPTR指令
}
```

```
inp_byte=XBYTE[0x8500]; //从地址8500H 读一字节
inp_word=XWORD[0x400]; //从地址4000H 读一字节
C=* ((char xdata*) 0x0000); //从地址0000H 读一字节
XBYTE[0x7500]=out_val; //写一字节到7500H
```



9.2 C51入门

5. CODE区

```
unsigned char code a[ ]=  
    {0x00, 0x02, 0x03, 0x04, 0x05, 0x06,  
    0x07, 0x08, 0x09, 0x010, 0x011, 0x012,  
    0x013, 0x014, 0x015} ;
```

9.2 C51入门

9.2.4 C51的运算符及表达式

表 9-4 运算符的优先级别和结合性

优 先 级	运 算 符	含 义	操作数个数	结 合 性
1	() [] -> 与	圆括号 下标运算符 结构体成员运算符		从左至右
2	! ~ ++ -- (类型) * & Size of	逻辑非运算符 按位取反运算符 自增运算符 自减运算符 强制类型转换运算符 指针运算符 取地址运算符 长度运算符	1 (单目运算符)	从右至左



3	* / %	乘法运算符 除法运算符 取余运算符	2 (双目运算符)	从左至右
4	+ -	加法运算符 减法运算符	3 (双目运算符)	
5	<< >>	左移运算符 右移运算符	2 (双目运算符)	
6	< <= > >=	关系运算符	2 (双目运算符)	
7	== !=	等于运算符 不等于运算符	2 (双目运算符)	
8	&	按位与运算符	2 (双目运算符)	
9	^	按位异或运算符	2 (双目运算符)	
10		按位或运算符	2 (双目运算符)	
11	&&	逻辑与运算符	2 (双目运算符)	
12		逻辑或运算符	2 (双目运算符)	
13	?:	条件运算符	3 (三目运算符)	从右至左
14	= += -= *= /= %= >>= <<= &= ^= =	赋值及复合 赋值运算符	2 (双目运算符)	
15	,	逗号运算符		从左至右

9.2 C51入门

9.2.5 C51的流程控制语句

【例 9-1】查表求 3 的平方值。

```
main ( )  
{  
    Char code table[6]={0,1,4,9,16,25};  
    int p =03H, x;  
    x = table[p];  
}
```

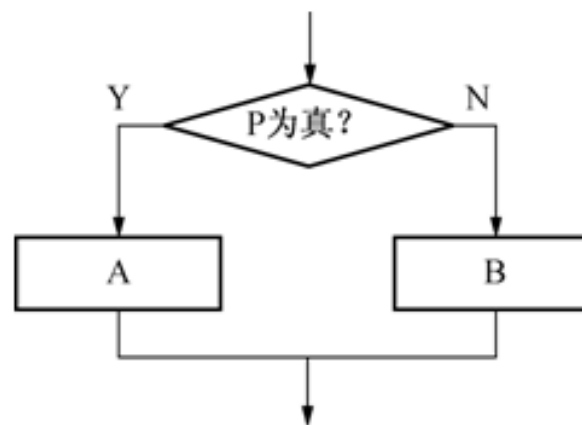


图 9-1 选择结构



9.2 C51入门

9.2.5 C51的流程控制语句

```
if (表达式)
{语句; }
```

形式 1

```
if (表达式) {语句; }
```

形式 2

```
if (表达式)
    {语句1; }
else
    {语句2; }
```

形式 3

```
if (表达式)
    {语句1; }
Else if
    {语句2; }
Else if
    {语句3; }
...
else
    {语句n; }
```




9.2 C51入门

9.2.5 C51的流程控制语句

```
switch (表达式)
{
case 常量表达式1: {语句1; } break;
case 常量表达式2: {语句2; } break;
case 常量表达式3: {语句3; } break;
...
case 常量表达式n: {语句n; } break;
default: {语句n+1; }
}
```



9.2 C51入门

1. while循环语句

```
while (条件表达式)
    {语句; } //循环体
```

【例 9-2】用 while 语句结构求 1~100 的和。

```
main ( )
{
    int i, sum;
    sum=0;
    while (i<101)
    {
        sum=sum+i           /*注意{ }不能省略, 否则跳不出循环体 */
        i++;
    }
    printf ("sum=% d",sum)
}
```

9.2 C51入门

2. do-while循环语句

```
do {语句; } //循环体  
while {条件表达式}
```

【例 9-3】用 do-while 语句求 1~100 的和。

```
main ( )  
{  
    int i, sum;  
    sum=0;  
    do  
    { sum=sum+i /*注意{ }不能省略, 否则跳不出循环体 */  
      i++;  
    }  
    while (i<101);  
    printf ("sum=% d",sum)  
}
```

【例 9-4】使用 do-while 语句的延时程序。

```
void delay ( )  
{  
    int x=20000;  
    do { x=x-1;  
      } while (x>1);  
}
```



9.2 C51入门

3. for循环语句

```
for (初值设定表达式; 循环条件表达式; 条件更新表达式)
    {语句; }    //循环体
```

【例 9-5】用 for 语句求 1~100 的和。

```
main ( )
{
    int i, sum;
    sum=0;
    for (i=0; i<101; i++)
        sum=sum+i;
    printf ("sum=% d",sum);
}
```



9.2 C51入门

4. 转移语句

goto

break

continue

return



9.2 C51入门

9.2.6 函数与C51中断服务函数

```
函数类型 函数名称 (形式参数表)
{
    函数体
}
```

1. 重入函数

```
int calc (char i, int b) reentrant
{
    int x;
    x = table [i];
    return (x * b);
}
```



9.2 C51入门

2. 函数使用指定的寄存器组 `using n`

`void` 函数标识符 (形参表) `using n`

```
{  push      psw
    mov      psw, #与寄存器组号n有关的常量
    |
    pop      psw
}
```

3. 函数使用指定的存储模式

类型说明符 函数标识符 (形参表) 存储模式修饰符 {small, compact, large}

```
extern int func (int i, int j)    large;    //修饰为大模式
```



9.2 C51入门

4. C51中断服务程序

```
void 函数标识符 (void) interrupt m
```

```
unsigned int interruptcnt;
unsigned char second;
void timer0 (void) interrupt 1 using 2
{
    if (++interruptcnt == 4000)
    {
        second++;           //计数到4000
        interruptcnt = 0;   //清除中断计数器
    }
}
```




9.2 C51入门

9.2.7 指针与指定地址的存储器访问

```
unsigned char *my_ptr, *another_ptr;
unsigned int *int_ptr;
float *float_ptr;
time_str *time_ptr;

My_ptr=&char_val;
Int_ptr=&int_array[10];
Time_str=&oldtime;
```

```
Time_ptr= (time_str *) (0x10000L); //指向地址0
Time_ptr++; //指向地址5
```

```
time_ptr=oldtime_ptr //两个指针指向同一地址
*int_ptr=0x4500 //把0x4500 赋给int_ptr 所指的变量
```

```
time_ptr->days=234;
*time_ptr.hour=12;
```

```
struct bst_node{
    unsigned char name[20]; //存储姓名
    struct bst_node *left, right; //分别指向左右子树的指针
};
```



9.2 C51入门

9.2.7 指针与指定地址的存储器访问

```
#define CBYTE ((unsigned char volatile code *) 0)
#define DBYTE ((unsigned char volatile data *) 0)
#define PBYTE ((unsigned char volatile pdata *) 0)
#define XBYTE ((unsigned char volatile xdata *) 0)
```

```
#include <absacc.h>
```

```
...
```

```
in_byte = XBYTE[0x8000];
```

```
//从地址8000H读一个字节
```

```
in_word = XWORD[0x4200];
```

```
//从地址4200H读一个字节
```

```
cp = *((char xdata *) 0x0010);
```

```
//从地址0010读一个字节
```

```
XBYTE[0x6500] = out_byte;
```

```
//写一个字节到6500H
```



9.2 C51入门

9.2.8 51应用要点归纳

- (1) **C51**支持位操作，而标准**C**不能。在编程中尽量使用位变量，可以节省内部**RAM**单元。
- (2) 用**C51**编程需注重对系统资源的理解。可以通过多看编译生成的**.m51** 文件来了解程序中资源的利用情况。
- (3) **C51**程序中应用的各种算法要精简，不要对系统构成过重的负担。
- (4) 用**C51**编程时要合理使用堆栈资源。
- (5) **C51**编程时不要使用复杂的数据结构，例如复杂结构体、函数指针等。
- (6) **C51**程序编译时，要合理使用编译模式，同时选择合适的优化等级。
- (7) 如果使用**C51**程序来操作单片机内部或者外围的硬件设备，需要熟悉那些硬件外设的编程模型，注意硬件时序特性，以提高编程效率。



9.3 C51编程举例

9.3.1 C51程序实现I/O端口的操作

【例 9-6】使 P1 口先输出 0xDB，调用延时程序 delay () 后，再使 P1 口输出 0xB6，然后再调用延时程序 delay ()，如此循环往复。程序如下：

```
#include <reg51.h>
void delay (void);
void main (void)
{
    delay ();                //调用延时子程序
    do{                      //置P1口状态为11011011
        P1=0xDB;
        delay ();           //延时
        P1=0x6D;            //置P1口状态为01101101
        delay ();           //延时
        P1=0xB6;            //置P1口状态为10110110
        delay ();           //延时
    } while (1);
}
void delay (void)
{
    volatile int x=20000;
    do { x=x-1; } while (x>1);
}
```

9.3 C51编程举例

9.3.2 C51实现内部定时器操作

【例 9-7】设 $f_{osc}=12\text{MHz}$ ，要求在 P1.0 脚上输出周期为 2ms 的方波，并采用定时器查询方式。

解：周期为 2ms 的方波要求定时间隔为 1ms。机器周期为 $12/f_{osc}=1\mu\text{s}$ ； $1\text{ms} = 1000\mu\text{s} = 1000$ 个机器周期，用定时器 T0 的方式 1 编程。程序如下：

```
#include <reg51.h>
sbit P1_0=P1^0;
void main (void)
{
    TMOD=0x01;           //T0的方式1
    TR0=1;               //启动定时器T0
    for (;;)
    {
        TH0= -1000/256;  //装计数器初值
        TL0= -1000%256;
        do {} while (!TF0); //查询等待TF0位
        P1_0=!P1_0;      //查询时间到，P1.0变反
        TF0=0;           //软件清除TF0位
    }
}
```

【例 9-8】设 fosc=12MHz，要求在 P1.0 脚上输出周期为 2ms 的方波，并采用定时器中断方式。

解：此例采用中断方式，用定时器 T0 方式 2。程序如下：

```
#include <stdio.h>
#include <reg52.h>
unsigned char intcycle;           //中断循环次数计数器intcycle
sbit P1_0=P1^0;
//T0中断服务子程序；每250μs中断一次，当晶振频率为12MHz
timer0 () interrupt 1 using 1     //T0中断向量000BH, Reg Bank 1
{
    if (++intcycle == 4) {        //1 msec = 4* 250usec cycle
        intcycle = 0;
        P1_0=!P1_0;
    }
}
tinit () {                       //设置T0方式2, 允许中断
    TH0 = -250;                  //装计数器初值
    TL0 = -250;
    TMOD = TMOD | 0x02;         //选择模式2
    TR0 = 1;                    //启动T0
    ET0 = 1;                    //允许T0中断
    EA = 1;                    //允许总中断
}
void main (void)
{
    tinit ();                   //初始化T0
    while (1) {}
}
```

9.3 C51编程举例

9.3.3 C51实现简易交通灯控制

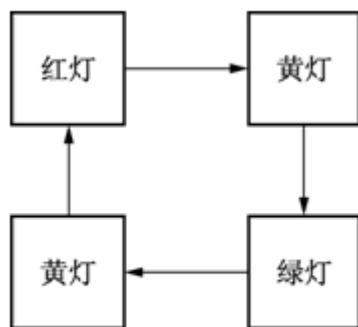


图 9-2 交通灯工作过程

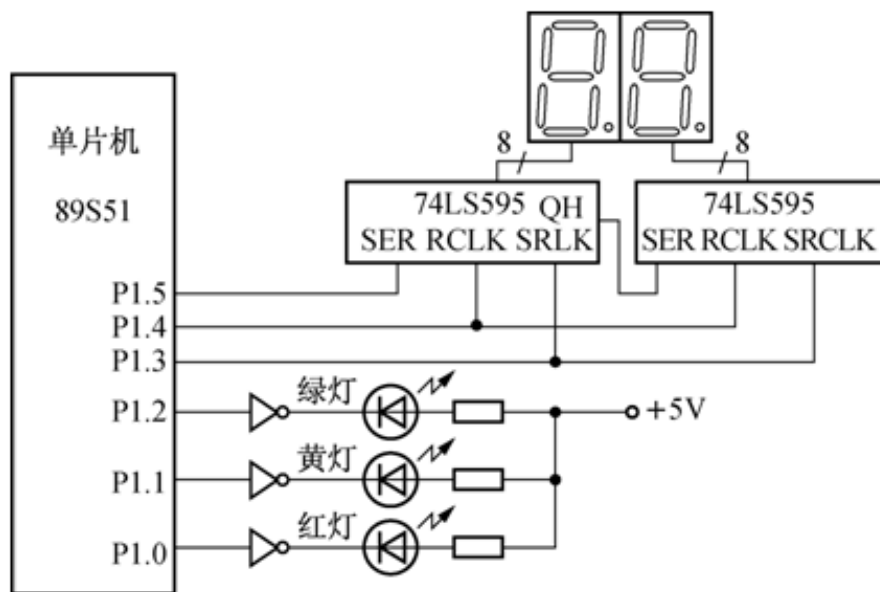


图 9-3 交通灯控制电路结构图

```

#include <reg51.h>
#define uchar unsigned char
#define uint unsigned int
#define True 1
#define False 0
#define TH_SET 0x4C
#define TL_SET 0x00          //11.0592MHz 晶振下，定时50ms 时间的计数初始值
#define RTime 50            //红灯时间
#define GTime 60            //绿灯时间
#define YTime 5             //黄灯时间
sbit RED=P1^0; sbit GREEN=P1^1; sbit YELLOW=P1^2;
sbit SER=P1^5; sbit RCLK=P1^4; sbit SRCLK=P1^3;
bdata uchar sendata;
sbit sendbit_0=sendata^0;
bit TimeOutFlag;
uchar DispBuf[2],IntCount,Status,LightTime;
uchar code disptab[11]={0xc0,0xf9b,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,
0x90};
//以上是数码显示表，c0H-"0",f9H-"1",a4H-"2",b0H-"3",99H-"4",92H-"5"
//,82H-"6",f8H-"7",80H-"8",90H-"9"; 对应共阳数码管：dp g f e d c b a

```

接下页

接上页

```
void PrintToLed (uchar *buf);           //送数码管显示函数声明
void Timer0_SVR () interrupt 1 using 1 //定时/计数器0 中断服务程序
{
    TH0=TH_SET;
    TL0=TL_SET;                          //重载计数初值
    IntCount++;
    if (IntCount ==20)                   //中断次数到
    { IntCount=0;                          //中断次数清零
      TimeOutFlag=True;                   //置定时1s到标志
    }
}
void main (void)
{ EA = 0;                                 //关中断控制器
  TMOD = 0x01;                            //定时/计数器0方式1定时
  TH0 = TH_SET;
  TL0 = TL_SET;                            //载入计数初值
  ET0 = 1;                                 //开定时器0 中断
  TimeOutFlag=False;                       //清定时1s 到标志
  IntCount =0;                             //中断计数清0
  TR0=1;                                   //启动定时器
  EA=1;                                    //开中断控制器
  LightTime=RTime;                         //置入红灯时间
  RED=1; YELLOW=0; GREEN=0;              //点亮红灯
  Status=1;                               //置当前状态为1
  while (True)
```

接下页

接上页

```
{
    if ( TimeOutFlag ==True )           //1s 定时到
    { TimeOutFlag=False;                //清定时1s 到标志
      LightTime--;                       //时间减1
      if ( Status==1 && LightTime==0)    //熄灭红灯，开黄灯，并载入黄灯时间
      { RED=0; YELLOW=1; GREEN=0; LightTime=YTime; Status=2; }
      if ( Status==2 && LightTime==0)    //熄灭黄灯，开绿灯，并载入绿灯时间
      { RED=0; YELLOW=0; GREEN=1; LightTime=GTime; Status=3; }
      if ( Status==3 && LightTime==0)    //熄灭绿灯，开黄灯，并载入黄灯时间
      { RED=0; YELLOW=1; GREEN=0; LightTime=YTime; Status=4; }
      if ( Status==4 && LightTime==0)    //熄灭黄灯，开红灯，并载入红灯时间
      { RED=1; YELLOW=0; GREEN=0; LightTime=RTime; Status=1; }
    }
    DispBuf[0] = (uchar) (LightTime/10); //取时间的十位
    DispBuf[1] = (uchar) (LightTime%10); //取时间的个位
    PrintToLed (DispBuf);               //送数码管显示
}
}
void PrintToLed (uchar *buf)
{ uchar dispbit, disptime;
  RCLK=0;                               //置74LS595 为移位状态
  for (dispbit=0; dispbit<2; dispbit++) //两个数码管
  { sendata=disptab[buf[dispbit]];      //查表
    for (disptime=0; disptime<8; disptime++) //送8段数码
    { SRCLK=0;
      SER=sendbit_0;                    //送数据
      SRCLK=1;                          //时钟上升沿送数据
      sendata=sendata>>1;               //右移1位
    }
  }
  RCLK=1;                               //数据锁存
}
```



9.3 C51编程举例

9.3.4 C51实现串口操作

【例 9-10】

```
#include <reg52.h>
#include <stdio.h>
void main (void)
{
    SCON = 0x50;           //串口方式1, 允许接收
    TMOD = 0x20;          //定时器1定时方式2
    TCON = 0x40;          //设定时器1开始计数
    TH1 = 0xfd;           //11.0592MHz 9600波特率
    TL1 = 0xfd;
    TI = 1;
    TR1 = 1;              //启动定时器
    while (1)
    {
        printf ("This is a test!\n"); //UART输出
    }
}
```

9.4 Keil C51集成开发环境

9.4.1 Keil C51的编译流程

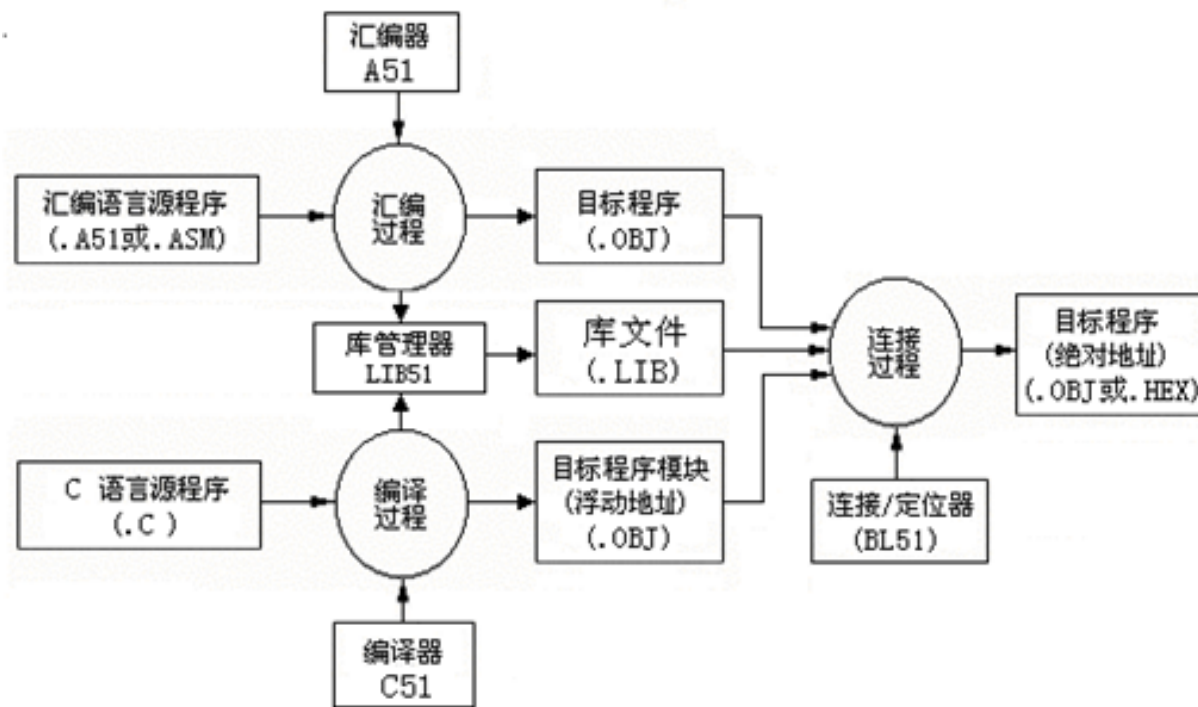


图 9-4 Keil C51 的编译器及编译过程

9.4 Keil C51集成开发环境

9.4.2 创建工程

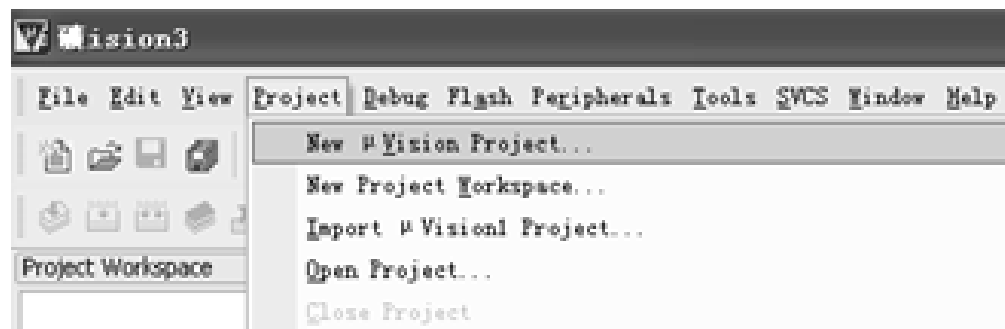


图 9-5 Project 菜单

9.4 Keil C51集成开发环境

9.4.2 创建工程



图 9-6 Create New Project ”对话框

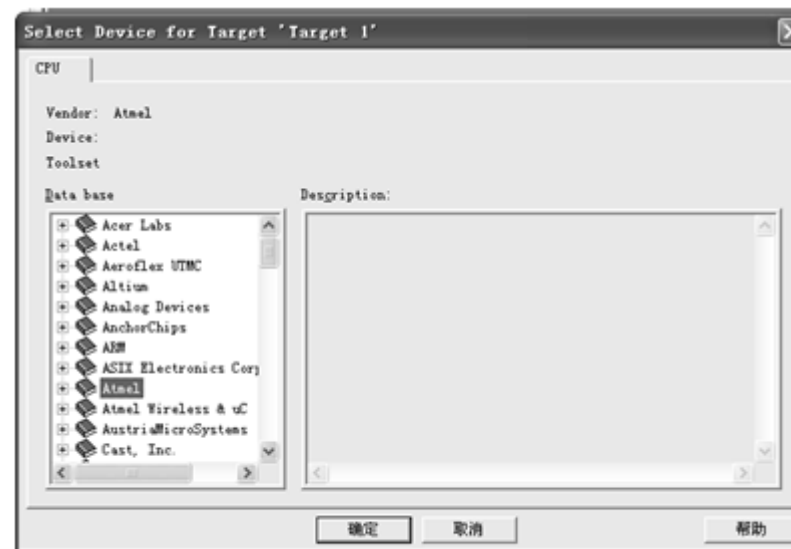


图 9-7 选取芯片



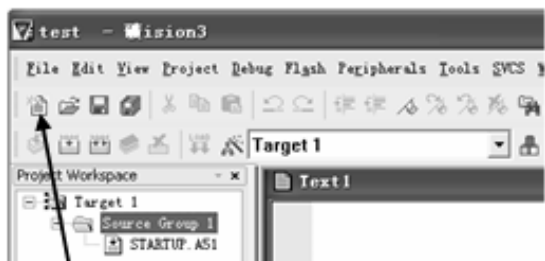
9.4 Keil C51集成开发环境

9.4.3 输入C源文件

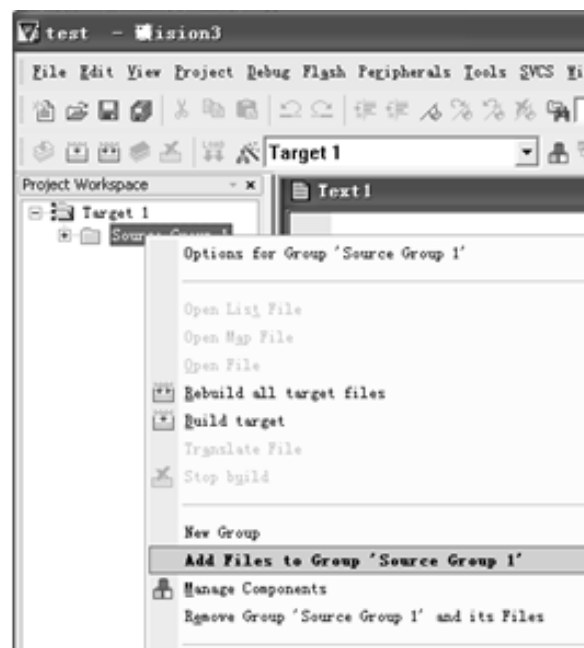
```
#include<AT89X51.H>
#include<stdio.h>
void main ()
{
    SCON=0x50;        //串行口方式1, 允许接收
    TMOD=0x20;        //定时器1定时方式2
    TCON=0x40;        //设定定时器1开始计数
    TH1=0xE8;         //11.0592MHz, 1200bps
    TL1=0xE8;
    TI=1;
    TR1=1;            //启动定时器
    While (1)
    {
        printf ("Hello World\n"); //显示Hello World!
    }
}
```

9.4 Keil C51集成开发环境

9.4.3 输入C源文件



①



②

图 9-8 新建程序文件

图 9-9 把文件加入到项目文件组中

9.4 Keil C51集成开发环境

9.4.4 C程序编译

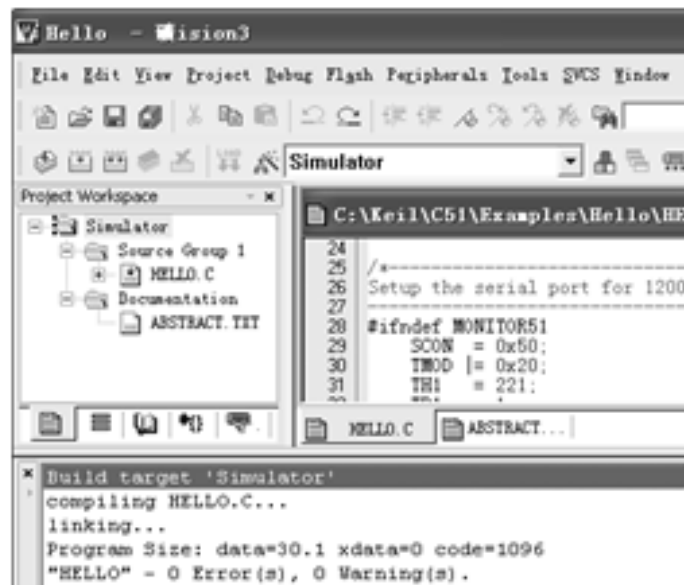


图 9-10 编译程序

9.4 Keil C51集成开发环境

9.4.5 程序调试

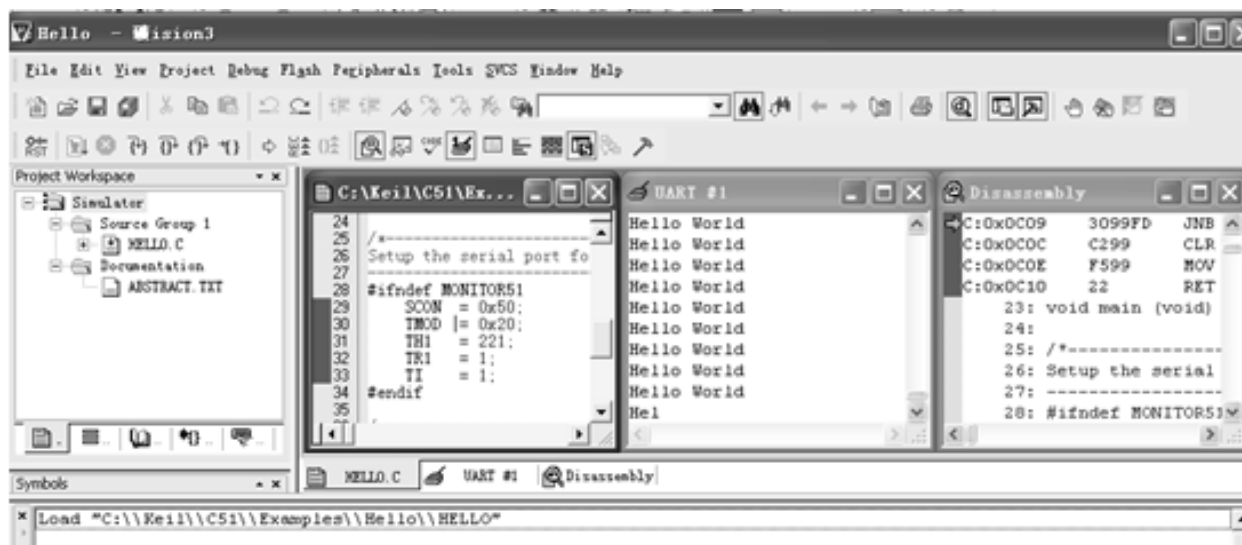


图 9-11 调试运行程序

9.4 Keil C51集成开发环境

9.4.6 生成HEX目标文件

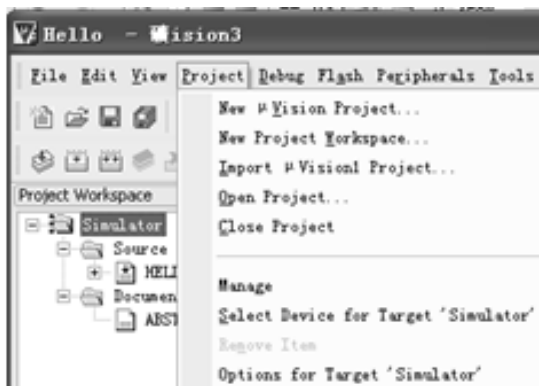


图 9-12 项目功能菜单图



图 9-13 项目选项窗口

```
Build target 'Simulator'
compiling HELLO.C...
linking...
Program Size: data=30.1 xdata=0 code=1096
"HELLO" - 0 Error(s), 0 Warning(s).
```

图 9-14 编译信息窗口



9.5 C语言与汇编语言的混合编程

9.5.1 C51程序中嵌入汇编代码

```
#pragma asm
mov P2, #0x30
...
...
# pragma endasm
```

； 嵌入的汇编语言代码

```
#include <reg51.h>
extern unsigned char code newval[256];
void func1 (unsigned char param) {
    unsigned char temp;
    temp=newval[param];
    temp*=4;
    temp/=7;
    #pragma asm
        MOV P1, R7 ; 输出temp中的数
        NOP
        NOP
        MOV P1, #0
    #pragma endasm
}
```

9.5 C语言与汇编语言的混合编程

9.5.1 C51程序中嵌入汇编代码

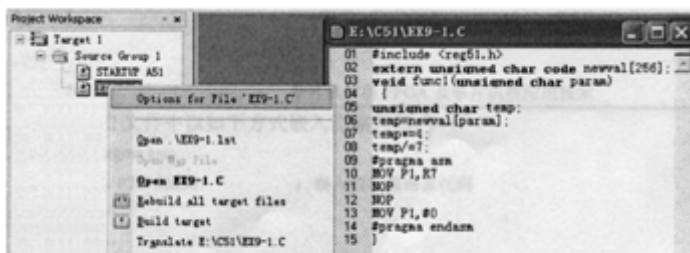


图 9-15 在 Keil C51 中导入 C 文件

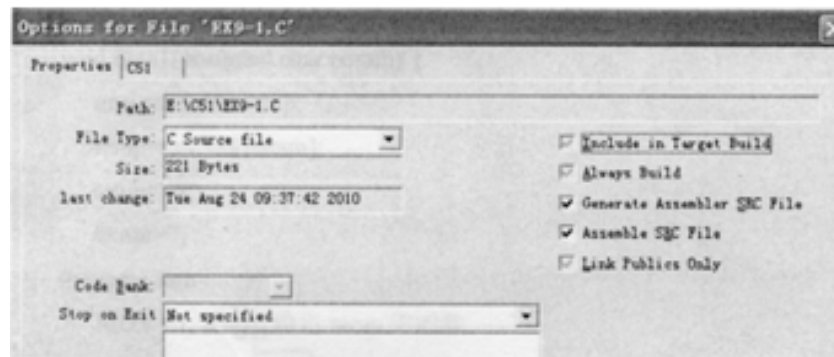


图 9-16 选择生成 SRC 文件

9.5 C语言与汇编语言的混合编程

9.5.2 C51程序中调用汇编子程序

表 9-5 C51 调用汇编子程序中的函数名转换

C51 中的函数声明	转换成汇编子程序中的函数名	说 明
void func (void)	FUNC	无参数传递或参数传递不通过寄存器的函数，函数名只需转换成大写形式
void func (char)	_FUNC	通过寄存器的函数，函数名前加“_”
void func (void) reentrant	_? FUNC	对于重入函数，函数名前加“_?”

表 9-6 参数传递中所使用的寄存器

参 数 类 型	char	int	long、float	一 般 指 针
第一个参数	R7	R6、R7	R4~R7	R1、R2、R3
第二个参数	R5	R4、R5	R4~R7	R1、R2、R3
第三个参数	R3	R2、R3	无	R1、R2、R3



9.5 C语言与汇编语言的混合编程

9.5.2 C51程序中调用汇编子程序

表 9-7 参数返回值所使用的寄存器

返回值	寄存器	说明
bit	C	进位标志
(unsigned) char	R7	R7
(unsigned) int	R6、R7	高位在 R6，低位在 R7
(unsigned) long	R4~R7	高位在 R4，低位在 R7
float	R4~R7	32 位 IEEE 格式，指数和符号在 R7
指针	R1、R2、R3	R3 存储器类型，高位在 R2，低位在 R1



9.5 C语言与汇编语言的混合编程

9.5.2 C51程序中调用汇编子程序

```
#include<REG51.H> //声明所调用的汇编程序（采用了extern）
extern unsigned mymult (unsigned char, unsigned char);
main (void)
{
    Char j;
    j= mymult (5, 73);
}
```

```
        PUBLIC _MYMULT           ; 带参数的函数声明
        PROC SEGMENT CODE       ; 定义PROC为再定位程序段
        RSEG PROC               ; 定义PROC为当前段
        _ MYMULT:
        MOV A, R7                ; 调入第一参数
        MOV B, R5                ; 调入第二参数
        MUL AB
        MOV R6, B                ; 返回运算结果
        MOV R7, A
        RET
        END
```