2015-2016学年春季学期

第3章　C51语言编程基础

内容概要

已掌握标准C语言前提下，初步介绍如何使用C51来编写AT89S51单片机的应用程序。

C51是在标准C的基础上，根据单片机存储器硬件结构及内部资源，扩展了相应的数据类型和变量，而C51在语法规定、程序结构与设计方法上，都与标准C相同。

本章重点介绍C51对标准C所扩展的部分，并通过一些例程来介绍C51的程序设计思想。

3.1 编程语言Keil C51简介

目前51系列单片机编程的C语言都采用Keil C51(简称C51)，Keil C51是在标准C语言基础上发展起来的。

3.1.1 Keil C51简介

C语言是美国国家标准协会（ANSI）制定的编程语言标准，1987年ANSI公布87 ANSI C，即标准C语言。

Keil C51语言是在ANSI C的基础上针对51单片机的硬件特点进行的扩展，并向51单片机上移植，经过多年努力，C51语言已经成为公认的高效、简洁而又贴近51单片机硬件的实用高级编程语言。

目前大多数的51单片机用户都在使用C51语言来进行程序设计。

用C51进行单片机软件开发，有如下优点：

（1）可读性好

C51语言程序比汇编语言程序的可读性好，因而编程效率高，程序便于修改。

（2）模块化开发与资源共享

用C51开发出来的程序模块可以不经修改，直接被其他项目所用，这使得开发者能够很好地利用已有的大量的标准C程序资源与丰富的库函数，减少重复劳动。

（3）可移植性好

为某种型号单片机开发的C语言程序，只需将与硬件相关之处和编译连接的参数进行适当修改，就可以方便地移植到其他型号的单片机上。例如，为51单片机编写的程序通过改写头文件以及少量的程序行，就可以方便地移植到PIC单片机上。

（4）生成的代码效率高

当前较好的C51语言编译系统编译出来的代码效率只比直接使用汇编语言低20%左右，如果使用优化编译选项，效果会更好。

不同的嵌入式处理器的C编译系统与标准C的不同之处，主要是它们所针对的嵌入式处理器的硬件系统不同。

Keil C51的基本语法与标准C相同，但对标准C进行了扩展。

深入理解Keil C51对标准C的扩展部分是掌握Keil C51的关键之一。

3.1.2 C51与标准C的比较

（1）库函数的不同

由于标准C的中的部分库函数不适于嵌入式处理器系统，因此被排除在Keil C51之外，如字符屏幕和图形函数。

有一些库函数可以继续使用，但这些库函数都必须针对51单片机的硬件特点来作出相应的开发，与标准C库函数的构成与用法有很大的不同。例如库函数printf和scanf，在标准C中，这两个函数通常用于屏幕打印和接收字符，而在Keil C51中，它们主要用于串行口数据的收发。

（2）数据类型的不同

51系列单片机包含位操作空间和丰富的位操作指令，因此Keil C51与ANSI C相比又扩展了4种类型，以便能够灵活地进行操作。

（3）C51语言的变量存储模式与标准C语言中的变量存储模式数据不同

标准C语言最早是为通用计算机设计的，在通用计算机中只有一个程序和数据统一寻址的内存空间。

C51语言中变量的存储模式与51单片机的存储器紧密相关。

（4）数据存储类型的不同

51系列单片机存储区可分为片内数据存储区、片外数据存储区和程序存储区。

- 片内数据存储区可分为3个不同的C51存储类型：data，idata和bdata；
- 片外数据存储区可分为2个不同的C51存储类型：xdata和pdata；
- 程序存储区只能读不能写，可在片内或片外，C51语言提供了code存储类型来访问程序存储区。

标准C并没有提供这部分存储器的地址范围的定义。此外，对于AT89C51单片机中大量的特殊功能寄存器也没有定义。

（5）标准C语言没有处理单片机中断的定义

C51语言中有专门的中断函数。

（6）C51语言与标准C语言的输入/输出处理不一样

C51语言中的输入/输出是通过51单片机的串行口来完成的，输入/输出指令执行前必须对串行口进行初始化操作。

（7）头文件的不同

C51语言头文件必须把51单片机内部的外设硬件资源（如定时器、中断、I/O等）相应的功能寄存器写入头文件。

51系列单片机厂家有多个，它们的差异在于内部资源如定时器、中断、I/O等数量以及功能的不同，而对使用者来说，只需要将相应的功能寄存器的头文件加载在程序内，就可实现所具有的功能。因此，Keil C51系列的头文件集中体现了各系列芯片的不同资源及功能。

（8）程序结构的差异

由于51单片机的硬件资源有限，它的编译系统不允许太多的程序嵌套。

标准C所具备的递归特性不被Keil C51支持，在C51中，要使用递归特性，必须用reentrant进行声明才能使用。

但是从数据运算操作、程序控制语句以及函数的使用上来说，Keil C51与标准C几乎没有什么明显的差别。

如果程序设计者具备了有关标准C的编程基础，只要注意Keil C51与标准C的不同之处，并熟悉AT89S51单片机的硬件结构，就能够较快地掌握Keil C51的编程。

3.2 Keil C51的开发环境

Keil C51是德国Keil software公司开发的用于51系列单片机的C51语言开发软件。Keil C51在兼容ANSI C的基础上，又增加很多与51单片机硬件相关的编译特性，使得开发51系列单片机程序更为方便和快捷，程序代码运行速度快，所需存储器空间小，完全可以和汇编语言相媲美。它支持众多的MCS-51架构的芯片，同时集编辑、编译、仿真等功能于一体，具有强大的软件调试功能，是众多的单片机应用开发软件中最优秀的软件之一。

3.2.1 集成开发环境Keil μVision4简介

Keil Software公司推出的Keil μVision4是一款基于Windows的软件平台，它是一种用于51单片机的集成开发环境（IDE—Intergrated Development Environment）。μVision3提供了对基于8051内核的各种型号单片机的支持，完全兼容先前的Keil μVision2和Keil μVision3版本。

目前当前较新的版本为Keil μVision4，版本Keil C51 V9.xx。

μVision4内部集成了源程序编辑器，并允许用户在编辑源文件时就可设置程序调试断点，便于在程序调试过程中快速检查和修改程序。

此外，μVision4还支持软件模拟仿真（Simulator）和用户目标板调试(Monitor51)两种工作方式。在软件模拟仿真方式下不需任何51单片机及其外围硬件即可完成用户程序仿真调试。

Keil C51的库函数含有100多种功能，其中大多数是可再入的。函数库支持所有的ANSI C的程序。库函数中的程序还为硬件提供特殊指令，例如nop、testbit、rol、ror等，方便了应用程序的开发。

Keil μVision的串口调试器软件comdebug.exe，用于在电脑端能够看到单片机发出的数据，该软件无需安装，可直接在当前位置运行这个软件。

可到有关搜索网站输入关键词“**串口调试助手**”，找到一个合适的下载网站，可即下载最新版本。

3.2.2 Keil μVision4软件的安装、启动和运行

1. 软件安装

Keil μVision4的安装，同大多数软件安装一样，根据提示进行。安装完毕后，可在桌面上看到Keil μVision4软件的快捷图标。

2. 软件启动

点击桌面上的Keil μVision4软件的快捷图标，即可启动该软件，几秒后，出现编辑界面。

3. 软件的运行

（1）建立一个新工程

Keil μVision把用户每一个应用程序设计**都当作一个项目**，用项目管理的方法把一个应用程序设计中所需要用到的、互相关联的程序链接在同一项目中。这样，打开一个项目时，所需的关联程序也都跟着进入了调试窗口，方便用户对项目中各个程序的编写、调试和存储。

用户也可能开发多个项目，每个项目用到了相同或不同的程序文件和库文件，采用项目管理，很容易区分不同项目中所用到的程序文件和库文件。因此，在编写一个新的应用程序前，先建立项目的良好习惯。

在编辑界面下，首先要建立一个点击“Project”菜单，选择下拉式菜单中的“New Project”，弹出文件对话窗口，选择要保存的路径，在“文件名”中输入一个程序项目名称，保存后的文件扩展名为“.uvproj”，这是Keil μVision4项目文件的扩展名，以后可直接点击此文件就可打开先前做的项目。（Keil μVision2和μVision3为“uv2”）

点击“保存”后，这是会弹出一个对话框，要求选择单片机的型号，用户可根据所使用的单片机来选择。Keil μVision4支持几乎所有的51内核的单片机。

开始编写第一个程序。点击“File”菜单，再在下拉菜单中单击“New”选项。此时光标在编辑窗口里闪烁，这时，用户可以输入代码了。

输入完毕，单击菜单上的“File”，在下拉菜单中单击“Save”，在“文件名”栏的编辑框中，键入文件名，同时，必须键入正确的扩展名。

注意，如果用C语言编写程序，则扩展名为“.c”；如果用汇编语言编写程序，则扩展名必须为“.asm”。然后，单击“保存”按钮。

3.3 C51语言程序设计基础

（★）指令系统的寻址方式

寻址方式就是在指令中说明操作数所在地址的方法，共7种寻址方式。

1. 寄存器寻址方式

操作数在寄存器中

```
MOV  A，Rn
```

表示把寄存器Rn的内容传送给累加器A。

寻址范围

- 4组通用工作寄存区共32个工作寄存器；
- 部分特殊功能寄存器，例如A、B 以及数据指针寄存器DPTR等。

2. 直接寻址方式

操作数直接以单元地址的形式给出：

```
MOV  A，40H
```

寻址范围

- 内部RAM的128个单元
- 特殊功能寄存器（SFR）。除了以单元地址的形式外，可用寄存器符号的形式给出；如：

MOV A，80H 与 MOV A，P0是等价的。

直接寻址方式是访问特殊功能寄存器的唯一寻址方式！

3. 寄存器间接寻址方式

寄存器中存放的是操作数的地址；

访问片内RAM或片外RAM的低256字节时，只能采用R0或R1作为间址寄存器：

```
MOV  A，@Ri
```

若Ri中的内容为40H，则把片内RAM中40H单元的内容送到A。

寻址范围

- 访问片内RAM低128单元，其通用形式为@Ri；
- 对片外RAM的64K字节的间接寻址，例如：

```
MOVX  A，@DPTR
```

- 片外RAM的低256字节，例如：

```
MOVX  A，@Ri
```

- 堆栈区。堆栈操作指令PUSH（压栈）和POP（出栈）使用堆栈指针（SP）作间址寄存器。

4. 立即寻址方式

操作数在指令中直接给出，需在操作数前加前缀标志“#”。

```
MOV  A，#40H
```

5. 基址寄存器加变址寄存器间址寻址方式

本寻址方式是以DPTR或PC作基址寄存器，以累加器A作为变址寄存器，如

```
MOVC A，@A+DPTR
```

若A的原有内容为05H，DPTR的内容为0400H，该指令执行的结果是把程序存储器地址为0405H单元的内容传送给A。

本寻址方式是专门针对程序存储器的寻址方式，寻址范围可达到64KB。

6. 位寻址方式

MCS-51有位处理功能，可以对数据位进行操作，例如：

```
MOV  C，40H
```

是把位40H的值送到进位位C。

寻址范围

- 内部RAM中的位寻址区；
- 特殊功能寄存器中的可寻址位；

7. 相对寻址方式

在相对寻址的转移指令中，给出了地址偏移量，以“rel”表示，即把PC的当前值加上偏移量就构成了程序转移的目的地址：

目的地址 = 转移指令所在的地址 + 转移指令的字节数 + rel

偏移量rel是一带符号的8位二进制数补码数，范围是：

–128 ~ +127

- 向地址增加方向最大可转移(127+转移指令字节)个单元地址；
- 向地址减少方向最大可转移(128–转移指令字节)个单元地址。

3.3.1 C51语言中的数据类型和存储类型

1. 数据类型

数据是单片机操作的对象，是具有一定格式的数字或数值，数据的不同格式称为数据类型。

Keil C51的基本数据类型如表3-1所示。针对AT89S51单片机的硬件特点，C51在标准C的基础上，**扩展了4种数据类型**（见表中最后4行）。

注意：扩展的4种数据类型，不能使用指针对它们存取。

表3-1 Keil C51支持的数据类型(★)

数据类型	位数	字节数	取值范围
signed char	8	1	-128~+127，有符号字符变量
unsigned char	8	1	0~255，无符号字符变量
signed int	16	2	-32768~+32767，有符号整型数
unsigned int	16	2	0~65535，无符号整型数
signed long	32	4	-2147483648~+2147483647，有符号长整型数
unsigned long	32	4	0~+4294967295，无符号长整型数
float	32	4	±3.402823 E+38，浮点数(精确到7位)
double	64	8	±1.175494E-308，浮点数(精确到15位)
*	24	1~3	对象指针
bit	1		0或1
sfr	8	1	0~255
sfr16	16	2	0~65535
sbit	1		可进行位寻址的特殊功能寄存器的某位的绝对地址

2. C51的扩展数据类型

（1）位变量bit

bit的值可以是1（true），也可以是0（false）。

（2）特殊功能寄存器SFR

AT89S51特殊功能寄存器在片内RAM区的80H~FFH之间，“sfr” 数据类型占用一个内存单元。利用它可访问AT89S51内部的所有特殊功能寄存器。

例如：sfr P1=0x90这一语句定义P1口在片内的寄存器，在后面语句中可用“P1=0xff”(使P1的所有引脚输出为高电平)之类的语句来操作特殊功能寄存器。

（3）特殊功能寄存器sfr16

“sfr16”数据类型占用两个内存单元。sfr16和sfr一样用于操作特殊功能寄存器。所不同的是它用于操作占两个字节的特殊功能寄存器。

例如： sfr16 DPTR=0x82语句定义了片内16位数据指针寄存器DPTR，其低8位字节地址为82H，高8位字节地址为83H，在后面的语句中可以对DPTR进行操作。

（4）特殊功能位 sbit

sbit 是指AT89S51片内特殊功能寄存器的可寻址位。

例如：

```
sfr  PSW = 0xd0;      /*定义PSW寄存器地址为0xd0*/
sbit PSW ^2 = 0xd2;   /*定义OV位为PSW.2*/
```

符号“^”前面是特殊功能寄存器的名字，“^”的后面数字定义特殊功能寄存器可寻址位在寄存器中的位置，取值必须是0~7。

注意，不要把bit与sbit混淆。

- bit用来定义普通的位变量，值只能是二进制的0或1；
- sbit定义的是特殊功能寄存器的可寻址位，其值是可进行位寻址的特殊功能寄存器的位绝对地址，例如PSW寄存器OV位的绝对地址0xd2。

3. 数据的存储类型

在讨论C51的数据类型时，必须同时提及它的存储类型，以及它与51单片机存储器结构的关系，这是由于C51定义的任何数据类型必须以一定的方式定位在51单片机的某一存储区中。

51系列单片机存储区可分为片内数据存储区、片外数据存储区和程序存储区。

访问片外数据存储区比片内数据存储区慢，原因是片外数据存储区是通过数据指针加载地址来间接寻址的。

表3-2 C51语言存储类型与8051存储空间的对应关系(★)

存储类型	与存储空间的对应关系
data	片内RAM直接寻址区，位于片内RAM的低128字节
bdata	片内RAM位寻址区，位于20H~2FH空间，允许位访问与字节访问
idata	片内RAM间接寻址的存储区
pdata	片外RAM的一个分页寻址区，每页256B
xdata	片外RAM全部空间，大小为64KB
code	程序存储区的64KB空间

(1) DATA区

- DATA区寻址是最快的，因此把经常使用的变量放在DATA区。
- DATA区存储空间有限，除包含程序变量外，还包括堆栈和寄存器组。
- 声明中存储类型标识符为：data。
- 位于片内RAM的低128字节，可直接寻址。

声明方式：

```
unsigned char data      system_status = 0;
unsigned int data       unit_id[8];
char data               inp_string[20];
```

标准变量和用户自声明变量都可存储在DATA区中，只要不超出DATA区的范围。

由于C51使用默认的寄存器组来传递参数，因此DATA区至少失去了8字节空间。

当内部堆栈溢出时，由于51单片机没有报错机制，程序会被复位，因此要预留较大的堆栈空间防止堆栈溢出。

(2) BDATA区

- BDATA区实质是DATA区中的位寻址区，在这个区中声明变量就可以进行位寻址。
- 声明中存储类型标识符为：bdata。
- 字节地址为20H~2FH，片内RAM可位寻址的16字节存储区的128个位。

BDATA区声明位变量和使用位变量：

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

C51编译器不允许在BDATA区声明float和double型的变量！

(3) IDATA区

- IDATA区使用寄存器做为指针来进行间接寻址，常用来存放使用比较频繁的变量。
- 与外部存储器寻址相比，它的指令执行周期和代码长度相对较短。
- 声明中存储类型标识符为：idata。
- 字节地址为00H~FFH，片内RAM的所有地址单元，只能间接寻址，速度比直接寻址慢。

声明方式

```
unsigned char idata     system_status = 0;
unsigned int idata      unit_id[8];
char idata              inp_string[16];
float idata             out_value;
```


（4）PDATA区和XDATA区

- PDATA区和XDATA区位于片外存储器。
- 声明中存储类型标识符分别为：pdata和xdata。
- PDATA区为片外数据存储器页，一页为256字节。因此PDATA区只有256字节。
- XDATA区最多可达64KB，对应的xdata存储类型标识符可指定片外数据存储器的任何地址。
- 对PDATA区的寻址要比对XDATA区的寻址快。因为对PDATA区寻址只需装入8位地址，而对XDATA区寻址需装入16位地址。

声明方式

```
unsigned char xdata     system_status = 0;
unsigned int pdata      unit_id[8];
char xdata              inp_string[16];
float pdata             out_value;
```

注意：外部数据存储器和外部I/O口是**统一编址**的，外部数据存储器地址段除了包括存储器地址以外，还包括外部I/O口的地址！

（5）程序存储器CODE区

- 声明中存储类型标识符为：code。
- 储存的数据是不可改变的，外部程序存储器的64KB空间。

```
unsigned char code      a[4] = {0x00,0x01,0x02,0x03};
```


表3-3 C51存储类型及其大小与值域(★)

存储类型	与存储空间的对应关系	数据长度（bit）	值域范围
data	片内RAM直接寻址区，位于片内RAM的低128字节	8	0~255
bdata	片内RAM位寻址区，位于20H~2FH空间，允许位访问与字节访问	1	
idata	片内RAM间接寻址的存储区	8	0~255
pdata	片外RAM的一个分页寻址区，每页256B	8	0~255
xdata	片外RAM全部空间，大小为64KB	16	0~65535
code	程序存储区的64KB空间	16	0~65535

对单片机编程，正确地定义数据类型以及存储类型，是所有编程者在编程前都需要首先考虑的问题。

在资源有限的条件下，如何节省存储单元并保证运行效率，是对开发者的一个考验。

只有对C51中的各种数据类型以及存储类型非常熟练的掌握，才能运用自如。

定义变量类型应考虑的问题

程序运行时该变量可能的取值范围，是否有负值，绝对值有多大，以及相应需要的存储空间大小。

在够用的情况下，尽量选择8位即一个字节的char型，特别是unsigned char。对于51系列这样的定点机而言，浮点类型变量将明显增加运算时间和程序长度，如果可以的话，尽量使用灵活巧妙的算法来避免浮点变量的引入。

定义数据的存储类型通常遵循的原则

只要条件满足，

- 尽量选择内部直接寻址的存储类型data；
- 然后选择idata，即内部间接寻址；
- 对于那些经常使用的变量要使用内部寻址；
- 在内部数据存储器数量有限或不能满足要求的情况下才使用外部数据存储器；
- 选择外部数据存储器可先选择pdata类型，最后选用xdata类型。

需注意，扩展片外存储器，原理上虽很简单，但在实际开发中，很多时候，会带来不必要的麻烦，如可能降低系统稳定性、增加成本、拉长开发和调试周期等，推荐充分利用片内存储空间。

另外，通常的单片机应用都是面对小型的控制，代码比较短，对于程序存储区的大小要求很低，常常是片内RAM很紧张而片内Flash ROM很富裕，因此如果实时性要求不高，可考虑使用宏，以及将一些子函数的常量数据做成数据表，放置在程序存储区，当程序运行时，进入子函数动态调用下载至RAM即可，退出子函数后立即释放该内存空间。

4. 数据存储模式

如果定义（声明）变量时略去存储类型标识符，编译器会自动默认存储类型。

默认的存储类型受SMALL、COMPACT和LARGE存储模式限制。

如声明

```
char var1;
```

SMALL模式：DATA区；

COMPACT模式：PDATA区；

LARGE模式：XDATA区。

在固定的存储器地址上进行变量的传递，是C51的标准特征之一。

- SMALL模式：在片内数据存储器中完成；
- COMPACT模式和LARGE模式：允许在片外数据存储器中完成；

C51支持混合模式，如在LARGE模式下，生成的程序可以将一些函数放在SMALL模式中，目的是加快执行速度。

（1）SMALL模式

- 该模式下，所有变量都默认位于51单片机的片内数据存储器，这与使用data指定数据存储器类型是相同的。
- 该模式下，变量访问的效率高，但所有数据对象和堆栈必须使用片内RAM。

（2）COMPACT模式

- 使用该模式时，所有变量都默认位于片外数据存储器的1页内，这与使用pdata指定数据存储器类型是相同的。该存储器类型适用于变量不超过256字节的情况，这是由数据指针寻址的寻址方式决定的。
- 与SMALL模式相比，变量访问的效率比较低，对变量的访问速度也会慢一些，但比LARGE模式快。

（3）LARGE模式

- 使用该模式时，所有变量都默认位于片外数据存储器，使用数据指针进行寻址。
- 该模式下，通过数据指针访问片外数据存储器的效率比较低，特别是当变量为2字节或更多字节时，要比SMALL模式和LARGE模式产生更多的代码。

3.3.2 C51语言的特殊功能寄存器及位变量定义

1．特殊功能寄存器（SFR）的C51定义

- C51语言允许通过关键字sfr、sbit或直接引用编译器提供的头文件来对特殊功能寄存器（SFR）进行访问；
- 51单片机的特殊功能寄存器分布在片内RAM的高128字节中；
- 对SFR的访问只能使用直接寻址的方式。

（1）使用关键字sfr

为了能直接访问特殊功能寄存器，C51提供了一种定义方法，引入关键字sfr，语法为：

sfr 特殊功能寄存器名称 = 特殊功能寄存器地址；

```
sfr  IE = 0xA8;        /*中断允许寄存器地址为A8H*/
sfr  TCON = 0x88;      /*定时器/计数器控制寄存器地址为88H*/
sfr  SCON = 0x98;      /*串行口控制寄存器地址为98H*/
```

如果要访问16位SFR，可以使用关键字sfr16，16位SFR的低字节地址必须作为“sfr16”的定义地址，如

```
sfr16  DPTR = 0x82;  /* DPTR的低8位地址为82H，高8位地址为83H */
```


（2）通过头文件访问SFR

- 各种衍生的51单片机的SFR的数量和类型有时是不同的，可以通过访问头文件的方式来访问SFR 。
- 为方便起见，C51把51单片机常用的SFR和其中的可寻址位进行了定义，放在reg51.h(或reg5*.h)头文件中。
- 使用时，只需在使用前用预处理指令

  #include < reg51.h >

  把该头文件包含到程序中，就可以直接使用其中的SFR名称和其中的可寻址位名称。
- 用户可通过文本编辑器自行对头文件进行增减。


```
#include   <reg51.h>     /* 头文件为51型单片机的头文件 */

void  main( void )
{
   TL0 = 0xF0;          /* 给定时器T0的低字节TL0设置时间常数 */
   TH0 = 0x3F;          /* 给定时器T0的高字节TH0设置时间常数 */
   TR0 = 1;             /* 启动定时器0 */
   ……
   ……
   ……
}
```


（3）SFR中的位定义

对SFR中的可寻址位的访问，要使用关键字来定义可寻址位，共有3种方法：

① **sbit 位名 = 特殊功能寄存器^位置；**

```
sfr   PSW = 0xD0;  /* 定义PSW寄存器的字节地址为D0H */
sbit  CY = PSW^7;  /* 定义CY位为PSW.7，地址为D7H */
sbit  OV = PSW^2;  /* 定义OV位为PSW.2，地址为D2H */
```

② **sbit 位名 = 字节地址^位置；**

```
sbit  CY = 0xD0^7;  /* 定义CY位地址为D7H */
sbit  OV = 0xD0^2;  /* 定义OV位地址为D2H */
```


③ **sbit 位名 = 位地址；**

将位的绝对地址赋给变量，位地址必须在0x80~0xFF。

```
sbit  CY = 0xD7;    /* 定义CY位为PSW.7，地址为D0H */
sbit  OV = 0xD2;    /* 定义OV位为PSW.2，地址为D2H */
```

【例】片内I/O口P1口的各寻址位的定义如下：

```
sfr   P1 = 0x90;
sbit  P1_7 = P1^7;
sbit  P1_6 = P1^6;
……
sbit  P1_0 = P1^0;
```

2 .位变量的C51定义

（1）位变量的C51定义方法

由于AT89C51能够进行位操作，C51扩展了“bit”数据类型用来定义位变量，这是C51与标准C的不同之处。

C51通过关键字“bit”来定义位变量，格式为：

bit bit-name；

如：

```
bit   ov_flag;          /* 将ov_flag定义为位变量 */
bit   lock_pointer;     /*将lock_pointer定义为位变量*/
```


（2）函数可以包含类型为bit的参数，也可将其作为返回值

C51程序函数可以包含类型为“bit”的参数，也可将其作为返回值。

如：

```
bit  func(bit b0, bit b1);     /* 位变量b0，b1作为函数func的参数 */
{
      ……
      return(b1);              /* 位变量b1作为函数的返回值 */
}
```


（3）位变量定义的限制

位变量不能用来定义指针和数组。

如：

```
bit  *ptr;          /* 错误，不能用位变量来定义指针*/
bit  a-array[ ] ;   /* 错误，不能用位变量来定义数组*/
```

在定义位变量时，允许定义存储类型，位变量都被放入一个位段，此段总是位于AT89S51片内RAM中，因此其存储器类型限制为bdata，data 或idata，如果将位变量定义成其他类型都会在编译时出错。

3.3.3 C51语言的绝对地址访问

如何对51单片机的片内RAM、片外RAM及I/O进行访问，C51提供了两种比较常用的访问绝对地址的方法。

1. 绝对宏

- C51编译器提供了一组宏定义来对CODE、DATA、PDATA和XDATA空间进行绝对寻址。
- 在程序中，使用“ #include <absacc.h> ”来对头文件absacc.h中声明的宏来访问绝对地址，包括CBYTE、CWORD、DBYTE、CWORD、XBYTE、XWORD、PBYTE和PWORD。

CBYTE ：以字节形式对CODE区寻址；
CWORD ：以字形式对CODE区寻址；
DBYTE ：以字节形式对DATA区寻址；
CWORD ：以字形式对DATA区寻址；
XBYTE ：以字节形式对XDATA区寻址；
XWORD ：以字形式对XDATA区寻址；
PBYTE ：以字节形式对PDATA区寻址；
PWORD ：以字形式对PDATA区寻址。

```
#include   <absacc.h>
#define    PORTA   XBYTE[0xFFC0]
        /* 将PORTA定义为外部I/O口，地址为0xFFC0，长度8位 */
#define    NRAM    DBYTE[0x50]
        /* 将NRAM定义为片内RAM，地址为0x50，长度8位 */
```


【例】片内RAM、片外RAM及I/O的定义程序：

```
#include   <absacc.h>
#define    PORTA   XBYTE[0xFFC0]
        /* 将PORTA定义为外部I/O口，地址为0xFFC0，长度8位 */
#define    NRAM    DBYTE[0x40]
        /* 将NRAM定义为片内RAM，地址为0x40，长度8位 */
void main( void )
{
  PORTA = 0x3D;
        /* 将数据3DH写入地址为0xFFC0的外部I/O端口PORTA */
  NRAM  = 0x01;
        /* 将数据01H写入片内RAM的40H单元 */
}
```

2. _at_关键字

- 使用关键字"_at_"可对指定的存储器空间的绝对地址进行访问。
- 格式为

【存储器类型】 数据类型说明符 变量名_at_地址常数

其中，

- ✓ 存储器类型为C51能识别的数据类型；
- ✓ 数据类型说明符为C51支持的数据类型；
- ✓ 地址常数用于指定变量的绝对地址，必须位于有效的存储器空间之内；
- ✓ 使用_at_定义的变量必须为全局变量。

【例】使用关键字_at_实现绝对地址的访问

```
void main( void )
{
  data unsigned char y1_at_0x50;
       /* 在DATA区定义字节变量y1，地址为50H */
  xdata unsigned int  y2_at_0x4000;
       /* 在XDATA区定义字变量y2，地址为4000H */
  y1 = 0xFF;
  y1 = 0x1234;
  ……
  while(1);
}
```


【例】将片外RAM 地址从2000H开始的连续20字节单元清0

```
xdata unsigned char buffer[20]_at_0x2000;

void main( void )
{
  unsigned char i;
  for( i=0; i<20; i++ )
  {
    buffer[i] = 0;
  }
}
```


3.3.4 C51的基本运算

1. 算术运算符

表3-4 算术运算符及其说明

符号	说明
+	加法运算
–	减法运算
*	乘法运算
/	除法运算
%	取模运算
++	自增1
––	自减1

- 对于“/”和“%”，这两个符号都涉及除法运算；
- “/”运算是取商；
- “%” 运算为取余数。

如“5/3”的结果（商）为1，而“5%3”的结果（余数）为2。

自增和自减运算符是使变量自动加1或减1；

自增和自减运算符放在变量前和变量之后是不同的。

- ✓ ++i，--i：在使用i之前，先使i值加（减）1。
- ✓ i++，i--：在使用i之后，再使i值加（减）1。

如：若i=4，则执行x=++i时，先使i加1，再引用结果，即x=5，运算结果为i=5，x=5。

如：若i=4，则执行x=i++时，先引用i值，即x=4，再使i加1，运算结果为i=5，x=4。

2. 逻辑运算符

表3-5　逻辑运算符及其说明

符号	说明
&&	逻辑与
II	逻辑或
I	逻辑非

73

3. 关系运算符

判断两个数之间的关系。

表3-6 关系运算符及其说明

符号	说明
>	大于
<	小于
>=	大于或等于
<=	小于或等于
==	等于
!=	不等于

74

4. 位运算

想要改变I/0口得某一位的值而不影响其它位，若I/0口可位寻址，可直接解决；若只能字节操作（如外扩的I/0口），要想实现单独的位控，需采用位操作。

表3-7 位运算其说明

符号	说明
&	位逻辑与
I	位逻辑或
^	位异或
~	位取反
<<	位左移
>>	位右移

75

【例】编写程序将扩展的某I/O口PORTA（只能字节操作）的PORTA.5清0，PORTA.1置为1。

```
#include   <absacc.h>
#define    PORTA   XBYTE[0xFFC0]

void main( void )
 {
   ……
   PORTA = ( PORTA & 0xDF ) | 0x02;
   ……
}
```

76

5. 赋值、指针和取值运算符

指针是C语言中一个十分重要的概念。

C语言中提供了两个专门用于指针和取地址运算符。

表3-8 赋值、指针和取值运算及其说明

符号	说明
=	赋值
*	取内容
&	取变量的地址

77

形式：

变量 = *指针变量;

指针变量 = &目标变量;

取内容运算：是将**指针变量**所指向的**目标变量的值**赋给左边的变量；

取地址运算：是将**目标变量的地址**赋给左边的变量；

注：指针变量中只能存放**地址**（即**指针型数据**），不能将非指针类型的数据赋值给一个指针变量。

78

3.3.5 C51的分支与循环程序结构

程序结构上可把程序分为三类，即顺序、分支和循环结构。

顺序结构是程序的基本结构，程序自上而下，从main（）的函数开始一直到程序运行结束，程序只有一条路可走，没有其他的路径可以选择。

顺序结构比较简单和便于理解，重点介绍分支结构和循环结构。

1. 分支控制语句

实现分支控制的语句有：if语句和switch语句。

(1) if语句

只有两条分支的时候，判定给定的条件是否满足，根据判定结果决定执行两种操作之一。

```
if（条件表达式） {语句；}
```

if语句可以嵌套使用！

C51提供3种形式的if语句。

形式1

```
if（条件表达式） {语句；}
```

如

```
if( x>y )
{
   max = x;
   min = y;
}
```


形式2

```
if（条件表达式） {语句1；}
else {语句2；}
```

如

```
if( x>y )
{
   max = x;
}
else
{
   min = x;
}
```

本形式相当于双分支选择结构。

形式3

```
if(条件表达式1） {语句1；}
else if(条件表达式2) {语句2；}
else if(条件表达式3) {语句3；}
……
else {语句n；}
```

如

```
if( x>100 )      { y = 1; }
else if( x>50 )  { y = 2; }
else if( x>30 )  { y = 3; }
else if( x>20 )  { y = 4; }
else             { y = 5; }
```

本形式相当于串行多分支选择结构。

(2) switch语句

if语句只有两个分支可供选择，而switch语句是多分支选择语句，在分支较多时的情况下使用switch语句。

形式为

```
switch(条件表达式1)
{
        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n：{语句n；}     break；
        default ：{语句n+1；}
}
```

说明

- 每一个case的常量表达式必须互不相同，否则将出现混乱；
- 各个case和default出现的次序，不影响程序的执行结果；
- switch括号内表达式的值与某case后面的常量表达式的值相同时，就执行它后面的语句，遇到break则退出switch语句。若所有case语句的常量表达式均不匹配时，执行default项语句。

85

注意

- 每个switch分支必须有一个break语句，否则程序并不能跳出switch，就会继续执行case后面的case语句。如果看一下上述结构的程序对应的汇编语言源程序可看到，每一条break语句对应了汇编语言中的一条SJMP指令，而没有SJMP指令程序会继续向下执行，并不能跳出分支选择语句。
- 实际上case(0)，case(1) ……只是确定了分支的地址，真正的判断是在switch语句开始的。
- 最后一个分支可以不加break，结束后直接退出。

86

【例】

```
input : keynum = keyscan( )
switch( keynum );break;
{
   case 1 : key1( );
   case 2 : key2( );
   case 3 : key3( );
   case 4 : key4( );
      ……
   default : goto input
}
```

87

2. 循环控制语句

许多实用程序都包含循环程序，熟练掌握和运用循环结构的程序设计是C51语言程序设计的基本要求。

实现循环结构的语句有3种：

- while语句
- do-while语句
- for语句

88

（1）while语句

格式为：

```
while（表达式）
{
   循环体语句；
}
```

表达式是while循环是否继续的条件，如果表达式为‘真’，就重复执行循环体语句；反之，则退出循环体。

89

while语句循环的特点在于，循环条件的验证在循环体的起始位置，要想执行重复操作，必须首先进行循环条件的测试，如条件不成立，则循环体内的重复操作一次也不执行。

例如：

```
while( ( P1& 0x80 ) ==0 )
{

}
```

while中的条件语句对P1口的P1.7进行测试，如果为低电平‘0’，则继续测试下去（等待），一旦P1.7变为高电平‘1’，则循环中止。

90

（2）do while语句

格式为：

```
do
{
   循环体语句；
}
while（表达式）；
```

do-while语句在执行完循环体后再判断条件，决定循环体是否继续执行。如果表达式非0，则继续执行那个循环体语句，直至表达式为0为止。

91

- do-while语句构成的循环与while语句构成的循环十分相似，区别在于while语句执行循环体前先判断条件， do-while语句执行完循环体后再判断条件。
- 和while循环一样，在do-while循环体中，一定要有能够退出循环的操作（避免死循环）！
- 使用较少，较多使用while循环。

92

【例】实型数组sample中有10个采样值，编写程序段，要求返回其平均值（平均值滤波）。

```
float avg( float *sample )
{
   float sum = 0;
   char n=0;
   do
   {
      sum += sample[n];
      n++;
   }while( n<10 );
   return( sum/10 );
}
```

93

（3）for语句

- 3种循环中，使用最为频繁。
- 不仅可以用于循环次数已知的情况，也可用于循环次数不确定，只给出循环条件的情况，可完全替代while语句。

格式为：

```
for (循环体初始化；循环体执行条件；循环体执行后操作)
{
   循环体语句；
}
```

94

for是C51的关键字，其后的括号通常有3个表达式，执行过程为：

1. 执行“初值设定表达式”；
2. 判断“执行条件表达式”，满足则继续，不满足则退出循环；
3. 执行一次循环体；
4. 计算“更新表达式”，转向步骤2；
5. 结束循环。

95

特例

1. for语句中3个表达式全部为空

```
for（；；）
{
   循环体语句；
}
```

无限循环，相当于while(1)。

96

特例

2. for语句中，第1个表达式为空

```
for（ ; i<100; i++ ）
{
   sum = sum+i;
}
```

对i不赋初值。

97

特例

3. for语句中，第2个表达式为空

```
for（ i=1; ; i++ ）
{
   sum = sum+i;
}
```

不判断循环条件，认为表达式2始终为‘真’，无限循环。

98

特例

4. for语句中，第1个和第3个表达式为空

```
for（ ; i<=100; ）
{
   sum = sum+i;
   i++;
}
```

99

特例

5. for语句中，没有循环体语句

```
int a = 1000;
for（ i=1; i<a; i++ ）
{
}
```

典型应用是软件延时（非优化条件下）。

在程序设计中，经常用到时间延迟，可用循环结构实现，即消耗掉一段已知时间。

100

AT89S51的指令执行时间是靠一定数量的时钟周期来计时的，如果用12MHz晶振，则12个机器周期为1us。

【例】编写一个延时1ms的程序。

```
void delayms( unsigned char j )
{
   unsigned char i;
   while(j--)
   {
      for( i=0; i<125; i++)
      { ; }
   }
}
```

for内部循环大约延时8us，与编译器有关，不是特别精确。

101

当i=0时，编译器要多花一个机器周期对for循环初始化？

因为在使用立即数时，单片机需要在代码空间（程序存储器）中为该立即数申请一个存储单元，用来存放该立即数，作为MOV指令的操作数；而累加器A是单片机中的寄存器，使用A可以节省一个字节的存储空间，从而实现以时间换取空间。

102

- 关于循环，需说明的是，在无操作系统的控制器和处理器上运行的程序，主体通常采用轮询方式，即把所有的操作包含在一个while(1){ }中。
- 无限循环在面向通用计算机的软件设计中是不被允许的，而在嵌入式系统软件设计中，则由于其硬件构成和使用需求，常常采用这种无限循环。

103

【例】求1到100之间整数的和。

```
#include <reg51.h>
#include <stdio.h>
main( )
{
   int  nVar1, nSum；
   for(nVar1=0, nSum=1；nSum<=100；nSum++)
      nVar1 += nCount；        /*累加求和*/
   while(1)；
}
```

104

【例】无限循环的结构实现。

方法1

```
while(1)
{
   代码段；
}
```

方法2

```
for( ; ; )
{
   代码段；
}
```

105

方法3

```
do
{
   代码段；
} while(1);
```

106

3. break语句、continue语句和goto语句

在循环体的执行过程中，

- ✓ 如果在满足循环判定条件的情况下跳出代码段，可以使用break语句或和continue语句；
- ✓ 如果要从任意地方跳转到某指定处，可使用goto语句。

107

(1) break语句

在循环结构中，可使用break语句跳出本层循环体，从而马上结束本层循环。

【例】

```
void main( void )
{
   int  i, sum=0;
   for (i=1; i<=10; i++)
   {
      sum = sum+i;
      if( sum>5 ) break;
   }
}
```

108

(2) continue语句

作用和用法与break语句相似，区别在于：

- 循环体遇到break语句，马上结束本层循环，跳出本层循环体；
- 循环体遇到continue语句，马上结束本次循环，不跳出本层循环体，直接尝试下一次循环。

109

【例】 输出整数1到100的累加值，但要求跳过所有个位为3的数。

思考：

- 在循环中加入判断，个位为3的数跳过不加；
- 用‘求余’运算判断个位是否为3；

110

```
void main( void )
{
   int  i, sum=0;
   for (i=1; i<=10; i++)
   {
     if( i%10==3 )
        continue;
     sum = sum+i;
   }
}
```

111

(3) goto语句

无条件转移语句，当执行goto语句时，将程序指针跳转到goto给出的下一条代码。

格式为：

goto 标号；

112

【例】求1到100之间整数的和。

```
#include <reg51.h>
#include <stdio.h>
void main( void )
{
   int  i, sum=0;

   sumadd:
        sum = sum+i;
        i++;
        if(i<101)
         { goto sumadd; }
}
```

113

goto语句在C51中经常用于无条件跳转某条必须执行的语句以及用于在死循环程序中退出循环。

为避免跳转时引发错误，也为了程序阅读方便，

慎重使用goto语句！

114

3.3.6 C51的指针

C51支持基于存储器的指针和一般指针两种指针类型；

当定义一个指针变量未声明其类型时，默认为一般指针；

基于存储器的指针类型由C51源代码中的存储类型决定，用这种指针可以高效访问对象，而且只需要1个字节（data *、idata *、pdata *）或2个字节（code *、xdata *）。

一般指针占用3字节：

- 1字节存储器类型

  存储器类型决定了对象所用的8051的存储空间；

- 2字节偏移量

偏移量指向实际地址。

一般指针可以访问任何变量而不管它在8051存储器的指针位置。

1.基于存储器的指针

定义一个指针时给出它所指对象的存储类型，则该指针是基于存储器的指针；

基于存储器的指针以存储类型为变量，形式为：

数据类型　存储类型1　*(存储类型2)　指针变量；

其中：

存储类型1：指针指向数据的存储类型；

存储类型2：指针自身的存储类型；

【例】

```
char xdata  *px;
```

定义了一个指向XDATA区的字符类型数据的指针，指针本身在默认的存储区，长度为2字节，值为0~0xFFFF。

【例】

```
char xdata  *data px;
```

定义了一个指向XDATA区的字符类型数据的指针，指针本身在DATA区，长度为2字节，值为0~0xFFFF。

【例】

```
data  char xdata  *px;
```

存储类型2可放在定义的开头，也可放在定义的指针变量之前。

C51的所有数据类型都和8051的存储器类型相关，所有用于一般指针的操作均可用于基于存储器的指针。

```
char xdata  *px;
/* px指向一个片外RAM的字符变量，px本身在默认的存储器中，由编译模式决定，占用2字节 */
char xdata  *data py;
/* px指向一个片外RAM的字符变量，px本身在默认的存储器中，由编译模式决定，占用2字节 */
```


2.一般指针

一般指针的声明符合标准C，但在C51中可以声明指针的存储类型。

一般指针形式为：

数据类型　存储类型1　*(存储类型2)　指针变量；

```
char  *px;
```

未声明px所指向变量的存储类型，px本身存储在编译模式决定的默认的存储区中，占用3字节。

一般指针包括3字节：1字节存储器类型和2字节偏移量，形式为：

地址	+0	+1	+2
存储内容	存储器类型	偏移量高位	偏移量地位

一般指针

存储器类型	data idata bdata	xdata	pdata	code
存储内容	0x00	0x11	0xFE	0xFF

存储器类型编码

121

例如以xdata型的0x1234地址作为指针表示，形式为：

地址	+0	+1	+2
存储内容	0x01	0x12	0x34

注意：当常数作指针时，必须注意正确定义存储器类型和偏移。

122

例如，将常数值0x41写入地址0x8000的片外RAM。

```
#define  XBYTE     ( ( char* )0x10000L )
XBYTE[0x8000] = 0x41;
```

XBYTE定义为((char*)0x10000L)，易知0x10000L为一般指针，其存储类型为1，偏移量为0000。

XBYTE被定义成为指向xdata零地址的指针，而XBYTE[0x8000]是片外RAM 0x8000的绝对地址。

注意：C51编译器不检查指针常数，用户应选择有意义的值，利用指针可对内存地址直接操作。

123

3.4 C51语言的函数

函数是一个完成一定相关功能的代码段。在高级语言中，函数和“子程序”、“过程”描述相似的事情。

在C51语言中使用术语“函数”。

C51中函数的数量没有限制，但必须至少有1个函数，以main为名，称为主函数。

主函数是唯一的，整个程序从主函数开始执行。

用户还可建立和使用库函数。

124

3.4.1 函数的分类

从结构上划分，函数分为：

- 主函数main ( )
- 普通函数

对普通函数，从用户使用的角度划分有两种：

- 标准库函数
- 用户自定义函数

125

（1）标准库函数

Keil C51具有功能强大、资源丰富的标准库函数，由C51编译器提供。

进行程序设计时，应该善于充分利用这些功能强大、资源丰富的标准库函数，提高编程效率。

126

用户可以直接调用C51的库函数而不需要为这个函数写任何代码，只需要包含具有该函数说明的头文件即可。

例如调用输出函数printf时，要求程序在调用输出库函数前包含以下的include 命令：

```
#include  <stdio.h>
```

127

（2）用户自定义函数

用户根据自己需要所编写的函数。如前面举例的Delay函数。编写时，需要注意以下几点。

函数的首部（函数的第1行），包括函数名、函数类型、函数属性、函数参数（形式参数）名、参数类型。

例如：

```
void Delay( unsigned int   cnt )
```

128

函数体，即函数首部下面的花括号“{ }”内的部分。如果一个函数体内有多个花括号，则最外层的一对“{ }”为函数体的范围；

C51区分大小写，例如Delay与delay，编译时是不同的两个名称；

每个语句最后必须有一个分号，分号是C语句的必要组成部分。

129

从函数的定义的形式上划分可以有三种形式：无参函数、有参函数和空函数。

（1）无参函数

此种函数在被调用时，既无参数输入，也不返回结果给调用函数，只是为完成某种操作而编写的。格式为：

```
返回值类型标识符  函数名( )
{
   函数体;
}
```

无参函数一般不带返回值，返回值类型标识符可以省略。

130

（2）有参函数

调用此种函数时，必须提供实际的输入函数，必须说明与实际参数一一对应的形式参数，并在函数结束时返回结果，供调用它的函数使用。

形式为：

```
返回值类型标识符  函数名( 形参列表 )
{
   函数体;
}
```

131

【例】定义一个函数max( )，用于求两个数中大数。

```
int   max(int  a, int  b)
{
   if( a>b )   return (a);
   else        return (b);
}
```

其中，参数a 、 b为形式参数（形参）， return( ) 为返回值语句。

132

（3）空函数

函数体内无语句，是空白的。

调用空函数时，什么工作也不做，不起任何作用。

定义空函数的目的，并不是为了执行某种操作，而是为以后程序功能的扩充。

程序最初设计时，往往只涉及最基本的功能模块的函数，其他模块的功能函数可以在以后补上。因此先将非基本模块的功能函数定义成空函数，用一个空语句“；”占好位置，并写好注释，以后再用一个编好的函数代替它。

形式为：

返回值类型标识符 函数名( )
{
}

例如：

```
float min( )
{     }/* 空函数，预订占座 */
```


3.4.2 函数的调用

在一个函数中需要用到某个函数的功能时，就调用该函数。

调用者成为主调函数，被调用者称为被调函数。

（1）函数调用的一般形式

函数名（实际参数1，实际参数2，…）

若被调函数是有参函数，则主调函数必须把被调函数所需的参数传递给被调函数。

（2）函数调用的方式

① 简单调用作为函数调用语句把被调函数的函数名作为主调函数的一个语句。

```
print_message( );
```

此时，并不要求被调函数返回结果数值，只要求被调函数完成某种操作。

② 被调函数结果作为表达式的一个运算对象。

```
result = 2 + max(a, b);
```

被调函数以一个运算对象出现在表达式中，要求被调函数带有return( )语句，返回结果数值参与表达式的运算。

③ 被调函数作为另一个主调函数的实际参数。

```
result = min( max(a, b), c );
```

被调函数max( )的返回结果数值作为主调函数max( )的实参。

（3）两种特殊的函数调用

- 嵌套调用

  是在被调用的函数中又调用其他函数的语句。

- 递归调用

  函数的递归调用就是一个函数在其函数体内有调用自身。

  再入函数是一种可以在函数体内直接或间接调用自身的一种函数，在Keil C51中递归函数必须是可重入的，可重入的函数需要加上reentrant。

139

再入函数有以下几点规定：

- 再入函数不能包括位操作以及51单片机的位寻址区；
- 在同一个程序中可以定义和使用不同存储模式的再入函数， 任何模式的再入函数不能调用不同模式的再入函数；
- 在参数的传递上，实际参数可以传递给间接调用的再入函数。

140

（4）对调入函数的说明

当一个函数调用另一个函数时，必须具备以下条件：

- 被调函数必须已经存在；
- 如果程序中已使用库函数，或使用了不在同一文件中的其它函数，要在程序开头处使用‘#include’包含语句；
- 如果程序中使用了同一个文件中的自定义函数，应根据主调函数和被调函数在文件中的位置，决定是否对被调函数作出说明。

141

a. 如果被调函数在主调函数之后，应在被调函数调用之前，对被调函数及的返回值类型作出说明；
b. 如果被调函数在主调函数之前，则不用对被调函数作出说明；
c. 如果在所有函数定义之前，在文件的开头处，在函数的外部已经说明了函数的类型，则在主调函数中不用对被调函数作出说明。

142

3.4.3 中断服务函数（★）

由于标准C没有单片机中断的定义，C51进行了扩展。增加了一个扩展关键字‘interrupt’。

使用interrupt可将一个函数定义为中断服务函数。

编译时，C51将对中断服务程序自动添加现场保护、现场恢复等程序段，方便用户使用。

143

中断服务函数的一般形式为：

函数类型 函数名 (形参表) interrupt n using n

- 关键字interrupt是中断号，对于51单片机，n的取值为0~4；
- 关键字using后面的n是所选择的寄存器组，using是一个可选项，可以省略。如果没有使用关键字using指明寄存器组，中断函数中的所有工作寄存器的内容将被保存在堆栈中。

144

3.4.4 变量及存储方式

1. 变量

（1）局部变量

若某一函数内部存在的变量，只在该函数内部有效。

（2）全局变量

在整个源文件中都存在的变量，所有函数都可以访问；

在全局变量声明前的函数或声明文件之外的源文件访问全局变量，要使用关键字extern进行说明。

145

- 由于全局变量一直存在，占用了大量的内存单元，而且加大了程序的耦合性，不利于程序的移植和复用；
- 全局变量可以使用关键字static进行定义。给变量只能在变量定义的源文件中使用，不能被其他源文件引用，称为静态全局变量；

146

2. 变量的存储方式

单片机的存储区间可以分为程序存储区、静态存储区和动态存储区。

数据存放在静态存储区或动态存储区；

- 全局变量存放在静态存储区， 程序开始运行时就给其分配存储空间；
- 局部变量存放在动态存储区，在进入拥有该变量的函数时，才给其分配存储空间。

147

3.4.5 宏定义和文件包含

1.宏定义

宏定义语句属于C51的预处理指令，可使变量的书写简化，增加程序可读性、可维护性和可移植性。

宏定义分为简单的宏定义和带参数的宏定义。

（1）简单的宏定义

#define 宏替换名 宏替换体

宏定义可以出现在程序的任何地方。

148

例如

```
#define   uchar      unsigned char
#define   UCHAR      unsigned char
```

编译时可由C51编译器把unsigned char用uchar替代；

```
#define   uchar      unsigned char
#define   uint       unsigned int
#define   gain       4
```

（2）带参数的宏定义

#define 宏替换名(形参) 带形参的宏替换体

149

2.文件包含

文件包含是指一个文件将另一个文件包含进去。

格式为：

#include <文件名> 或 #include “文件名”

区别为：

- 采用<文件名>时，在头文件目录中查找指定文件；
- 采用“文件名”时，在当前的目录中查找指定文件。

要调用C51编译器提供的库函数，必须在文件开头该文件包含进来。

150

3.4.6 库函数

C51强大的功能及其高效率的重要体现之一，在于提供了丰富的可直接调用的库函数。

使用库函数可使程序代码简单、结构清晰、易于调试和维护。

① 特殊功能寄存器包含文件<reg5x.h>；
② 绝对地址包含文件<absacc.h> ；
③ 输入/输出流函数，位于<stdio.h> ；
④ 动态内存分配函数，位于<stdlib.h> ；
⑤ 缓冲区处理函数，位于<string.h> ；
⑥ ……

思考题及习题

1. C51在标准C的基础上，扩展了哪几种数据类型？
2. C51有哪几种数据存储类型？其中idata, code, xdata, pdata各对应AT89S51的那些存储空间？
3. 说明3种数据存储模式SMALL、COMPACT与LARGE之间的区别？
4. 编写C51程序，将片外2000H为首地址的连续10个单元的内容，读入到片内40H到49H单元。
5. do-while构成的循环与while循环的区别是什么？