MCS-51单片机C语言应用程序开发技术及实例

赵远航　郑志辉　主编

航空工业出版社

MCS-51 单片机 C 语言应用程序

开发技术及实例

赵远航　郑志辉　主编
徐建平　刘春虹　审校

航空工业出版社

1996

内 容 提 要

C 语言是一种流行的结构化程序设计语言，它不仅具有高级语言的各种特性，而且能直接操作硬件单元。因此，使用C 语言开发单片机应用程序不仅能极大地缩短开发周期，而且使得程序易于维护和移植。本书详细介绍了目前微型计算机上最流行的单片机开发工具C-51。全书共分 4 章，第 1 章介绍了 C-51 的安装方法及系统需求；第 2 章通过实例详尽介绍了 C-51 的编译连接方法；第 3 章为 C-51 详细参考，该章全面介绍了 C-51 的存储模式、数据类型、配置、连接、堆栈、中断、与汇编语言的接口等；第 4 章介绍了 C-51 的命令行选择项、数据类型、C 库函数、头文件、扩展语言关键字、#include 文件、段及编译信息等。

本书内容全面、语言流畅、实例丰富，适合于从事单片机开发工作的科技人员和有关人员参考。

图书在版编目(CIP)数据

MCS-51 单片机 C 语言应用程序开发技术及实例/赵远航，郑志辉主编．—北京：航空工业出版社，1996.10
ISBN 7-80046-959-X

Ⅰ.M… Ⅱ.①赵… ②郑… Ⅲ.单片微型计算机-C 语言-程序设计-软件开发 Ⅳ.TP368.1

中国版本图书馆 CIP 数据核字(96)第 17158 号

航空工业出版社出版发行
（北京市安定门外小关东里 14 号 100029）
北京通县向阳印刷厂印刷　　全国各地新华书店经售
1996 年 10 月第 1 版　　1996 年 10 月第 1 次印刷
开本：787×1092 1/16　　印张：9　　字数：220 千字
印数：1—1600　　定价：26.80 元

前　言

众所周知，随着单片机集成度的提高，其应用范围越来越广。但是，同使用汇编语言开发软件一样，对单片机软件的开发始终存在着一个开发复杂、移植和维护困难的问题。今天，这个问题终于可以用C-51解决了。这就是说，广大单片机开发人员终于可以使用C语言进行单片机开发了。

C语言是一种流行的结构化程序设计语言，它不仅具有高级语言的各种特性，而且能直接操作硬件单元。因此，使用C语言开发单片机应用程序不仅能极大地缩短开发周期，而且使得程序易于维护和移植。本书详细介绍了目前微型计算机上最流行的单片机开发工具C-51。全书共分4章，第1章介绍了C-51的安装方法及系统需求；第2章通过实例详尽介绍了C-51的编译连接方法；第3章为C-51详细参考，该章全面介绍了C-51的存储模式、数据类型、配置、连接、堆栈、中断、与汇编语言的接口等；第4章介绍了C-51的命令行选择项、数据类型、C库函数、头文件、扩展语言关键字、#include文件、段及编译信息等。

本书由赵远航和郑志辉主编，参与本书编写工作的还有刘志强、韦杭、李海林、王小华、叶光明、郭福林、张强华等，本书由徐建平、刘春虹审校。

由于时间仓促，不当之处在所难免，尚希读者批评指正。

编　者

1996年6月

目　录

第 1 章　安装 C-51

单片机的软件开发同使用汇编语言开发软件一样，始终存在着开发复杂、移植和维护困难的问题。今天，这个问题终于解决了，这就是 C-51，也就是说，广大单片机开发人员终于可以使用 C 语言进行单片机开发了。

本章讲述如何安装和运行 C-51 开发工具集。

1.1　系统要求

IBM PC 版本的 C-51 开发工具集包括两张 5 1/4 英寸 1.2 兆 MS-DOS 格式软盘。它要求运行在 MS-DOS(或 PC DOS)系统 2.11-4.01 版本下，或者 OS/2 1.0 以上版本；硬盘最小自由空间为 1.5 兆字节，以编译及存放有关程序；至少 512K 的随机存储器 RAM 空间(建议用 640K 空间)，其中至少要有 450K 可用空间用来编译和连接。

要想利用 80286、80386、80486PC 机的扩充内存来进行 C 编译和连接，则要求使用 C-51 工具集提供的用来编译和连接的 .DXT 文件，而不使用缺省的 .EXE 文件。

1.2　C-51 文件类型

表 1.1 列出了 C-51 创建和使用的扩充文件类型。

表 1.1　文件类型及其扩展类型

*.EXE	MS-DOS 可执行文件
*.DXT	被重新命名的“*.exe”可执行文件，在配备扩充存储器的 80286、80386 或 80486 机型中运行
*.C	C 语言源文件
*.H	C 语言包含(“#include”)文件
*.S03	8051 汇编语言源文件
*.INC	8051 汇编语言“include”文件
*.R03	编译和汇编输出的 C 运行时程序库文件和目标文件
*.A03	连接输出的 8051 可执行文件
*.LST	编译和连接输出的缺省表文件
*.MAP	为连接表(“图”)文件而增设的推荐文件
*.XCL	连接命令行选择项的命令文件
*.DOC	补充文档的正文文件
*.BAT	控制编译和连接的批处理文件
*.00T	为连接可执行输出文件而增设的缺省文件

1.3　C-51 安装

用户可利用两种方法来安装 C-51，一是通过 INSTALL.EXE 程序，二是手工安装。下面我们分别对其作简单介绍。

1.3.1 使用 INSTALL.EXE 安装 C-51 文件

要运行安装工具，请把 C-51 第一张盘插入 A 驱动器，转到 A 驱动器并输入：

```
A:\>INSTALL
```

INSTALL.EXE 可以把 C-51 文件安装在单目录下或多目录下。缺省目录名如表 1.2 所示，用户也可在安装时修改它们。

表 1.2 C-51 安装目录和文件

目录	内容
单目录：	
\ARCH	所有文件
多目录：	
\ARCH\BIN	可执行文件(.EXE)
\ARCH\LIB	库(.LIB)
\ARCH\SOURCE	C 和汇编源文件(.C、.S03、.INC、.XCL、.DOC)
\ARCH\INCLUDE	"#include"文件(.H)

如果确认为多目录配置，则 INSTALL.EXE 工具对 AUTOEXEC.BAT 文件做如下修改：

```
set C_INCLUDE=INCLUDE 文件所在路径\
set XLINK_DFLIDIR=LIB 文件所在路径\
```

这些命令将设置搜索 INCLUDE 和 LIB 文件的路径。

1.3.2 手工安装 C-51 开发工具集

手工安装 C-51 十分简单，用户只需将所有文件拷入一个目录(单目录安装)或多个目录(多目录安装)。如果用户采用的是多目录安装，则还需设置环境变量(如上节所示)。

1.4 环境变量

本节解释几个环境变量的使用。这些环境变量通过操作系统可确定编译程序、连接程序和库程序的路径及其他信息，从而使我们不必在每次命令行中打出全部路径和其他信息。

DOS 和其他操作系统提供了宿主内存的"环境"区来确定名称("环境变量")及相关正文串。这样，当一个程序(像 C-51 编译程序)在运行时，可检查某个特定变量的"环境"，从中识别和读出一些通过正文串所提供的环境特定信息。

例如，可在 AUTOEXEC.BAT 文件中用下面的 DOS 命令告诉编译程序到哪儿去查找"#include"文件。

```
set C_INCLUDE=c:\arch\include\
```

当编译一个有"#include"文件的 C 源文件时，编译程序首先在当前目录中查找 #include 文件。如果没有找到文件，编译程序就检查操作系统环境来看看是否设置了"C INCLUDE"变量。如果找到上述变量的话，编译程序将会通过 C_INCLUDE 变量查找 #include 文件。

C_INCLUDE 路径必须在串尾包含一个所示的反斜线符("\")，因为此路径串被逐字放到包含文件名中。

可利用分号把各个路径分开的方法来确定 C INCLUDE 的多路径。如下所示：

```
c:\ARCHIMED>set C_INCLUDE=drive1:\path1\;driver2:\path2\
```

1.4.1 连接程序的目录模块搜索路径 XLINK_DFLTDIR

XLINK 连接程序为变量 XLINK_DFLTDIR(XLINK,DeFauLT DIRectory)检查 DOS 环境。XLINK_DFLTDIR 提供了一个目录,在此目录中可搜索目标文件,如 C 函数库文件 CL8051 * .R03,编译程序和连接程序创建的其他可重写的目标模块。

1.4.2 通用环境变量 QCC8051 和 XLINK_ENVPAR

另外,C-51 和 XLINK 还提供了"通用"环境变量。C-51 将检查 QCC8051 的环境设置,而 XLINK 通常检查 XLINK_ENVPAR 的环境设置。

因为编译程序通常产生混和的 C 程序和汇编程序,所以在每次编译时总是需要在编译程序命令行上加上-L-q 选择项:

```
C:\ARCHIMED\SOURCE>C-51 -L -q myfile
```

如果在 AUTOEXEC.BAT 文件中增加如下行:

```
Set QCC8051 = -L -q
```

-L -q 会一直"自动地"被增加到 C-51 命令行的开始部分,用户只要输入

```
C:\ARCHIMED\SOURCE>C-51 myfile
```

而得到的结果命令行为

```
C:\ARCHIMED\SOURCE > C-51 -L -q myfile
```

由于 XLINK 几乎总是被用作"连接程序控制文件",它包括 XLINK 所要求的众多命令行选项,因此,XLINK_ENVPAR 并不常用。

第2章 C-51概述

本章将讲解如何编译和连接第一个程序、配置启动例行程序和I/O目标、使用C-51版本4.0扩充语言关键字、以及连接C和汇编语言多模块。

下面给出几个范例的摘要。与这些范例有关的源文件均在软盘中提供。

第一个范例(2.1节)将编译一个C-51工具提供的简单C程序。实例程序访问一个#include文件且定义了几个数据工具,它可用来了解C-51软件是否安装到计算机上,以及是否创建了一个C和汇编的混合语言列表文件。

第二个范例(2.2节)告诉用户如何连接在第一步中编译好的程序,这可通过使用连接程序控制文件LNK8051.XCL加一些命令行选择项进行。第二个连接程序控制文件TUT1.XCL是为此范例定制的,它告诉用户怎样为一个特定的应用程序定制一个连接程序控制文件。

第三个范例(2.3节)告诉用户如何修改缺省CSTARTUP运行时的初始代码,这是在复位或加电后马上执行的代码。另外,可修改被printf()调用的putchar()单字符I/O函数来向I/O目标发出每个字符,这种传送一般采用缺省8051嵌入式串行I/O端口进行。

用户要做的修改将把串行I/O端口初始代码加到CSTARTUP例程中,并把制表符到空格的转换加到putch()例程。

然后,用XLIB库程序把已修改的版本替换紧凑内存模式的C运行库CL8051C.R03。这将使得用新的版本替换缺省版本来连接。

第四个范例(2.4节)讲述了众多C-51 4.0扩充关键字,其中包括interrupt关键字、sfr数据类型、内存区说明符data和xdata。

用户还将学到怎样使用扩充语言的#pragma等价格式,它允许使用许多扩充特性,但却使程序保持了ANSI C的兼容性。

最后一个范例(2.5节)将编译两个C文件并汇编一个汇编语言文件。这些程序描述了ANSI C新增的关键字的使用方法,并告诉用户如何从C语言中把参数传入或传出汇编语言例程。

然后用库XLIB把三个目标模块放到一个单库文件中。最后,把模块从库文件中连接到一个单符号输出文件(用AOMF8051格式)。

2.1 编译一个简单的C程序

在此范例中,将编译一个C-51工具提供的简单C程序。范例程序访问了一个#include文件并定义了一些数据工具。这用来检测C-51软件是否安装到计算机上,是否创建了一个C和汇编的混合语言列表文件,然后再进行详细的说明。

了解混合列表文件是很重要的,因为它表明了在C-51开发环境中一些很重要的概念,如可重定位段和汇编程序支持命令。

2.1.1 创建 C 源文件 TUT1.C

首先要做的是创建一个源文件。用户可以自己键入程序 2.1，也可使用系统提供的文件 TUT1.C。

TUT1.C 是由几个简单的变量声明和几个赋值语句组成的一个短小 C 程序。变量的名称已详细地拼出，用户在查看映像文件的汇编语言部分时能较容易地区别它们。

程序 2.1 TUT1.C

```
/*
TUT1.C
C-51 4.0 的第一个例子包括一些变量定义和赋值语句
该例子讲述了编译命令行的行使用及编译列表文件格式
编译命令行为:C-51 -ms -L -q -p tut1
其含义为:小存储器模式、混合 C 和汇编语言列表文件、可编程代码
作者/日期
*/
unsigned char uninit_char_var;
const char array_of_char[]="const_chars_in_array";
char * pointer_to_string="string";
int init_int_var=1, uninit_int_var;

void main(void)
{
unsigned int local_var=7;
uninit_unsig_char_var=0x55;
pointer_to_string=array_of_const_char;
init_int_var+=uninit_int_var;
/* ANSI 要求 uninit 变量设置为 0 */
}
```

下一步是编译源文件。如果做下一步有问题，则可能是因安装错误而致。

2.1.2 编译 C 源文件

使用下列命令行编译 TUT1.C。用户也可使用批处理文件 TUT1.BAT 来编译和连接此范例。

编译程序将创建一个名为 TUT1.R03 的目标文件和一个名为 TUT1.LST 的列表文件。R03 文件扩展名中的 R 代表“可重定位”，03 是 8051 系列模块的扩展名。其命令为：

```
C\ARCHIMED>C-51 TUT1 -ms -L -q -p
```

依据上面的操作步骤，编译程序调用包含了源程序名(采用 .C 扩展名)，后面跟着四个编译程序选择项。

-ms	指定内存模式为小模式
-L	指定生成 .LST 列表文件
-q	将助记符包含在 .LST 列表文件中
-p	生成 PROM 代码

2.1.3 关于列表文件 TUT1.LST

列表文件通常包含下面内容：

(1) 引言。
(2) 模块、段、变量和函数声明。
(3) C 语言及其汇编语言转化。
(4) 变量和初值定义。
(5) 错误警告。
(6) 模块产生的代码段长度声明。

有关源行号和地址的说明可在文件中找到。在文件中,C 源行号放在每个 C 源语句的前面,反斜杠\放在汇编语言语句的前面。跟在反斜杠后面的第一个数字是在"段"中的语句的相对地址,语句放在这些段中。

跟在反斜杠后面的其余(若存在)数字部分是对应相对地址的实 8051 机器代码。

2.1.4 范例 1 概述

在本节的范例 1 中,用户调用了带-l 和-q 选择项的 C-51 编译程序。C-51 编译程序创建两个输出文件,一个是带 .R03 扩展名的目标文件,另一个为 .LST 扩展名的列表文件。C 和汇编混合结果列表文件的讨论指出了编译程序是如何使用段和汇编支持命令的。

2.2 连接一个简单的 C 程序

在本节的范例中将告诉用户如何使用 XLINK.EXE 连接程序来连接在范例 1 中已编译的程序。用户将了解通用连接程序控制文件 LINK8051.XCL 怎样用一系列命令行选择项来控制连接过程。第二个连接控制文件 TUT1.XCL 详细地告诉用户如何为特殊应用定制连接程序控制文件。

连接程序后(使用 TUT1.XCL),检查产生的连接程序映像文件 TUT1.MAP(连接程序"映像"文件)。连接程序映像文件包含了各段的绝对物理地址及其中的数据和代码目标。使用连接程序选择项将创建一个用小内存模式的"PROMmable(可编程)"无符号文件。

因为此例没有涉及配置问题,所以,程序即使编译和连接,仍有可能不能在硬件上正确运行。有关特殊硬件配置的详细情况,参见第 3 章 3.7 节。

2.2.1 XLINK 连接程序

连接是代码模块和编译程序要求的 C 运行库函数相结合的过程。连接程序把物理地址分配给程序和库模块中定义的代码和数据目标,并且创建一个输出文件,使用户能下装到 PROM 或仿真器(通常通过宿主机的串行口)。

XLINK 连接程序是有效而灵活的,并要求提供指令以完成定义项目的连接。这些指令通过连接程序命令行给出,并包含:

(1)连接 ROM 和 RAM 段的地址。
(2)连接的模块名。
(3)连接模块的 C 运行库文件名。
(4)连接程序映像(.MAP)文件的格式。
(5)要求的输出文件格式(超过 20 个变量)。
(6)输出文件名和映像(.MAP)文件名。

连接一个 C 程序的命令行需要更多的任选择项，所以连接程序通常从与“连接程序控制文件”有关的批文件中读取这些指令(缺省文件扩展名为 .XCL 的 XCL 文件)。

为使用 XCL 连接程序控制文件，则调用带-f 连接程序命令选择项的连接程序。-f<文件名[.XCL]>(带文件内容的扩展命令行)连接程序选择项使连接程序从带有缺省文件扩展名 .XCL 的已命名文件中读取命令行选择项。它将把操作系统的 EOF(End Of File)字符解释为命令行的结束。

命令行选择项在命令行和 .XCL 文件之间分离开来。这样用户可以创建一个普通连接控制文件，它依赖于在命令行中使用更多的选择项。否则用户可以创建一个完整的连接程序控制文件，它包括工具的所有相关信息。下面列出了这两种技术的示例。

2.2.2 以通用连接程序控制文件 LINK8051.XCL 连接

连接 TUT1.R03 文件，使用下面所示命令行：

```
D:\ARCH>xlink -f link8051 cl8051s tut1 -o tut1.hex -x -l tut1.map
```

完整的命令行由连接程序控制文件 LINK8051.XCL 的内容加上连接模块名，以及命令行中指定的输出文件名三部分组成。

文件 CL8051S.R03 是小模式下的 C 运行库文件。任何时候，连接 C 程序都必须包含运行库文件。六种内存模式都有这样的文件，该名字的最后一位作为内存模式名的首字母，其他部分相同。因为已完成的 TUT1.C 使用小模式(使用 C-51 -MS 命令选择项)，故必须使用 CL8051S.R03。

-X(创建映像文件)，-L(命名映像文件)，以及-O(命名输出文件)选择项在命令行中显式地使用。连接后，通过目录列表可核实 .HEX 文件与 .MAP 文件的存在。

2.2.3 定制 XLINK 连接程序控制(.XCL)文件

完整的连接程序控制文件可以很容易地利用拷贝 LNK8051.XCL 的方法来创建，重新对它命名，然后为工具定制选择项。

下面的示例使用名为 TUT1.XCL 的已修改的连接程序控制文件，它在盘中已提供。文件的内容如下所示。关键点以 items 标明(“item1”，“item2”，等等)，下节是有关这些标志的解释。

程序 2.2 连接程序控制文件示例 1：TUT1.XCL

```
-!
TUT1.XCL 为辅导第一示例连接控制文件
它基于示例连接程序控制文件 LNK8051.XCL
连接程序命令行:LINK -f tut1
-!
-! 定义 CPU 类型：(item 1) -!
-c8051
-! 选择区 0 作为缺省寄存器区：(item 2) -!
-D_R=0
-! 指定 XDATA 段开始地址：(item 3) -!
-Z(XDATA)C_ARGX,X_UDATA,X_IDATA,NO_INIT,ECSTR=0
-! 指定 CODE 段开始地址：(item 4) -!
-Z(CODE)INTVEC,RECODE,D_CDATA,I_CDATA,X_CDATA=0
-Z(CODE)C_ICALL,C_RECFN,CSTR,CCSTR,CODE,CONST
-! 指定 DATA 段开始地址：(item 5) -!
```

```
-Z(DATA)C_ARGD,D_UDATA,D_IDATA=30
-! IDATA 段紧接 DATA 段：(item 6) -!
-Z(IDATA)C_ARGI,I_UDATA,I_IDATA,CSTACK
-! 指定位可寻址段开始地址：(item 7) -!
-Z(BIT)C_ARGB,BITCARS=0
-! 指定 printf()格式程序：(item 8) -!
-e_small_write=_formatted_write
-! 指定 scanf()格式程序：(item 9) -!
-e_medium_read=_formatted_read
-! 列出被连接的模块：(item 10) -!
tut1
-! 列出被连接的 C 运行库文件：(item 11) -!
cl8051s
-! 禁止覆盖检查：(item 12) -!
-Z
-! 指定输出文件格式：(item 13) -!
-FINTEL-STANDARD
-! 指定输出文件名：(item 14) -!
-o tut1.hex
-! 产生连接程序列表映像文件：(item 15) -!
-x
-! 命名连接程序映像文件：(item 16) -!
-l tut1.map
```

2.2.4 连接程序映像文件(.MAP)

映像文件通常包含下列内容：

(1) 引言。

(2) 模块映像。

(3) 段跳转表。

(4) 错误和警告。

-x(产生交叉引用文件)连接程序选择项有许多格式，而每一种格式都提供了不同的映像文件格式。

2.2.5 范例 2 概述

范例 2 用连接程序控制文件 LNK8051.XCL 来连接在范例 1 中创建的示例程序。说明部分详细分析了连接程序控制文件中各项内容，并讨论了如何修改它们。

连接程序控制文件使连接程序创建两个输出文件：绝对执行文件 TUT1.A03 和映像文件 TUT1.MAP。

2.3 修改 CSTARTUP.S03 和 PUTCHAR.C

当 CSTARTUP 运行时，初始化代码的缺省值以及 putchar()和 getchar()低级输入/输出程序在大多数应用中已经完全能够满足要求。但是，也可以为特定的项目修改它们。在此例中，将对缺省 CSTARTUP 和 putchar()程序作简单的修改，并用紧凑内存模式重新汇编和编译它们。

将要做的修改是，把串行端口初始化代码增加到 CSTARTUP 程序中，并把制表符转换成空格增加到 putchar()函数中。然后，用 XLIB 库程序以紧凑内存模式把修改后的版本重新放到 C 运行库 CL8051.R03。这样，新版本将代替缺省版本被连接。

2.3.1 修改 CSTARTUP 并把它重新放入库中

汇编语言文件 CSTARTUP.S03 包含了"引导"代码的源代码，它在处理器复位或加电后立即执行。CSTARTUP 代码定义和设置堆栈，把缺省寄存器存储区安装到 8051 的 PSW 寄存器中，把初始化变量设置成程序所希望的初始值，最后把控制交给 main()，在这里执行第一个 C 指令。CSTARTUP 还包含 exit()函数代码，当程序 main()的最后一条指令完成后，控制就传给它。

CSTARTUP 中要修改的项如下：

(1)改变堆栈长度(缺省为 30 字节)。

(2)若没有要求初始化的代码，则删除它。

(3)把监控复位指令增加到变量初始化代码中(若使用监视计时器)。

(4)若不使用自动向量产生任选择项(或者如果用汇编语言写句柄)，则把入口增加到中断向量表中。

(5)把代码增加到 exit()函数中(若不希望死循环)。

(6)增加其他代码以启动任何硬件。

在此例中，用户将做上面列出的最后一个修改，增加代码以初始化 8051 的内部串行口，使得它可以被 C-51 开发工具集所提供的缺省 putchar()和 getchar()函数来访问它。printf()和 scanf()函数也使用 putchar()和 getchar()作为它们的低层输出，所以对它们的使用也是依靠此设置的。

如果用户愿意，也可以从 C 源程序中初始化串行口，正像第 3 章 3.7.6 节描述 PUTCHAR.C 那样。对于此例，代替增加串行口初始化代码，可以作上面所列出的任何其他修改，然后跟上其余的指令来汇编和替换库文件中 CSTARTUP。有关上面列出的修改的更详细的信息，请参考第 3 章 3.7.3 节。

把串行口初始化代码增加到 CSTARTUP.S03 中

把文件 CSTARTUP.S03 放到编辑程序中。此文件包含将在处理程序复位和调用 main 之间执行的所有代码。

现在，对 CSTARTUP.S03 进行修改。在此描述的修改已按惯例作用于文件 TUT3STRT.S03，有关修改版本的其余正文是 TUT3STRT.S03。

开始，会发现 CSTARTUP.S03 中下面内容：

```
;------------------------------------------------------------
;if hardware must be initiated from assembty or if internpts
;should be on when reaching main,this is theplace to insert
;such code
;------------------------------------------------------------
```

(如果硬件必须利用汇编语言初始化，或如果到达 main 时，中断必须被打开，应插入下面的代码。)

在它下面插入几行：

```
MOV SCON,#52H          ;设置 model,使能串行口接收
```

```
                        ;设置 TI 位容许第一个字符发送
MOV TMOD,#20H           ;设置定时器 1 为自动重载
MOV TCON,#69H           ;打开定时器 1
MOV TH1,#F3H            ;定时器重载值为 1200 波特@12MHz
```

这些行要依据自己的应用来修改。此例文件 TUT2STRT. S03 包含了增加的这些行。然后,汇编已修改的 CSTARTUP。

汇编 CSTARTUP. S03

CSTARTUP. S03 包括所有内存模式的所有代码。这就是,在此文件中使用的条件汇编指令,它允许利用此文件创建目标模块的不同版本,指定所使用的特定内存模式。这样做是因为不同的内存模式,其 CSTARTUP 代码略有不同。

例如,对于 CSTARTUP. S03 下列片段中成对出现的 IF/ELSE 条件汇编指令代码,如果符号"bawked mode"定义成 1,那么它只能汇编成目标代码的一部分。

```
IF banked mode
EXTERN    ? X_CALL_L18
MOV       A,#BYTE3 main
MOV       DPTR,#main
LCALL     ? X_CALL_L18          ;main()
ELSE
LCALL     main                  ;main()
ENDIF
```

一个称为 DEFMODEL. INC 的汇编包含文件控制这些条件汇编子句,可参考 CSTARTUP. S03 文件的顶端:

```
;...
NAME CSTARTUP
$ defmodel. inc             ;定义存储器模式
EXTERN    ? C_EXIT          ;程序执行完毕后转向一位置
EXTERN    _R                ;寄存器组(0,8,16 or 24)
EXTERN    main              ;通常的第一个 C 函数
;...
```

可以从提供的六个包含文件中选择一个,然后用拷贝的方法来创建文件 DEFMODEL. INC。这六个文件称为"DEFM * . INC",这里 * 代表要汇编的内存模式的首字母。没有缺省文件 DEFMODEL. INC。

在此情况下,可用紧凑内存模式,所以必须拷贝文件 DEFMC. INC 到 DEFMODEL. INC,如下所示:

```
C:\ARCHIMED>COPYdefmc. ine defmodel. inc
```

DEFMC. INC 的内容如程序 2. 3 所示。

程序 2. 3　DEFMC. INC 的内容

```
;-DEFMC. INC-
;
;创建一紧凑模式的 include 文件
;
;Version: 4. 00 [IANR]
;
local_DATA      DEFINE    1
local_IDATA     DEFINE    0
local_XDATA     DEFINE    0
global_DATA     DEFINE    0
```

```
global_IDATA.   DEFINE    0
global_XDATA    DEFINE    1
banked_mode     DEFINE    0
                LSTCND    +
```

或者，可以重新声明：

```
$ defmodel. inc      ;定义存储器模式
$ defmc. inc         ;设置紧凑内存模式
```

或者，仔细阅览文件，以 INC 文件为基础，删除每个对使用的内存模式来说不正确的条件声明。

然后，使用下述命令，在与 CSTARTUP. S03 相同的目录下用 DEFMODEL. INC 文件重新汇编 CSTARTUP. S03。

```
C:\ARCHIMED>a8051 tut3strt
```

这将在 ARCHIMED 目录下创建一个目标文件 TUT3STRT. R03。如果还想创建一个映像文件，以便检查条件汇编声明完成的结果。此外，还可以在命令行中提供映像文件名。例如，下面命令行创建 . R03 和 . LST 文件：

```
C:\ARCHIMED>a8051 tut3strt tut3strt
```

注意，即使文件称为 TUT3STRT，如果模块名在 NAME 汇编指令中定义为“CSTARTUP”。其次，模块名在这种情况下为 CSTARTUP，正如 CSTARTUP. S03 开始的 NAME 汇编指令中规定的那样，通常是很敏感的。但是，无论什么文件名（这里是 TUT3STRT. S03），只有操作系统支持敏感文件名时，它才是敏感的。MS-DOS 不支持敏感文件名。

然后，把新 CSTARTUP 替换到 C 运行库中。

把 CSTARTUP 重新放入 C 运行库文件

为了用连接程序连接修改版的 CSTARTUP，而不是 C 运行库文件 CL8051. R03 提供的缺省文件，使用 XLIB 管理程序的 'replace module' 命令，拷贝新 CSTARTUP 目标模块覆盖 CL8051C. R03 以前存在的文件。

用 XLIB 中的 replace module 命令替换模块是不可逆的！在使用此命令前，请做一个要修改的库档案文件副本（在这里是 CL8051C. R03）。

XLIB 库管理程序有它自己的命令提示符（*）。为调用库管理程序，在 DOS 提示符下键入 XLIB：

```
C:\ARCHIMED\SOURCE>xlib
```

用下面命令把 STARTUP 文件替换到运行库文件中：

```
* def CPU 8051
* rep mod tut3strt cl8051c
* exit
```

XLIB 命令可以缩写，所以在命令序列中，替换模块命令 replac modnle 缩写为 rep mod，定义 CPU 命令 define CPU 缩写为 def cpu。

DOS 批处理文件 TUT3. BAT 包含用来汇编 TUT3STRT. S03 的命令行任选择项（并编译下一节所描述的 TUT3CHAR. C），然后调用库管理程序 XLIB。

一旦完成这些操作，可在任何时候用连接程序控制文件指定 C 运行库文件 CL8051，以连接程序，CSTARTUP 新版本将包含在连接程序输出文件中。

另外，有一种等价的方法，可用代码连接 CSTARTUP，它在第 3 章解释。

例如，作为另外一种通过库文件连接修改的 CSTARTUP 模块的方法，可用 XLIB 中的清除一模块命令来清除现存的库文件中的 CSTARTUP 模块，然后增加名 TUT3STRT 到模块表中来显式地连接 CSTARTUP 模块，此模块表是连接程序控制文件要连接的。

或者，可用 XLINK 的任选择项-C（条件安装）连接新 CSTARTUP 模块以禁止从库文件 CSTARTUP 中自动装入，然后显式地把包含已经修改的 CSTARTUP 模块的文件连接到连接程序控制文件中。

下面的例子告诉读者怎样修改 PUTCHAR.C 并把它替换到库中。

2.3.2 修改 PUTCHAR.C 并把它替换到库中

ANSI C 的 putchar()和 getchar()单字符输入/输出函数各自通过预先定义的 I/O 目标，每次发送和接收一个字符。C-51 缺省的目标是 8051 嵌入式串行口。此外，ANSI C 格式 I/O 函数 printf()和 scanf()各自调用 putchar()和 getchar()，每次它们发送或接收一个字符。

如果依据应用需要不同的 I/O 目标，或者如果涉及一个不同的串行 I/O 协议（例如中断驱动代替缺省的 polling 版本），必须修改缺省 putchar()和 getchar()。因为这些原因，所以，提供了 C 库源文件 PUTCHAR.C 和 GETCHAR.C。

为了使缺省的 putchar()和 getchar()工作，必须初始化并使能串行口，可利用 C 或汇编语言函数进行。本章的 1.1 节告诉用户怎样为此目的，增加一个串行口初始化函数至 CSTARTUP 代码。

此例修改较少，即只是让缺省 putchar()函数把制表符翻译成普通空格。目标是使用户熟悉 putchar()函数并熟悉修改它的技术，然后重新编译它，并把它替换到库中。修改 getchar()是一个简单的过程。

中断驱动串行口例程是 C 用户的好课题，C-51 至少有一个中断驱动串行口配置完整例子。

将制表符到空格符的转换增加到 putchar.c

把文件 PUTCHAR.C 装入编辑程序。用户将对它做的改变已方便地在 TUT2CHAR.C 文件中做到。正文的其余部分和修改的版本有关，如 TUT3CHAR.C。修改 PUTCHAR.C 文件，使得它如程序 2.4 所示。

程序 2.4 对 PUTCHAR.C 所作修改

```
/*            -TUT3CHAR.C-
The ANSI "putchar" function as modified in
Tutorial #3:Modifying CSTARTUP and putchar()
This putchar convers tabls to spaces.
*/
#include <stdio.h>
#include <io51.h>
#define TAB         0x09     /* 字符 ASCII 代码 */
#define SPACE       0x20
#define LF          0x0A
#define CR          0x0D
static void low_level_put(char c)
{
    do {} while (! TI);
    TI=0;
```