

科海培训中心系列教材

C 语言实践(一)

——C 语言的 DOS 系统程序设计

吕 强 杨季文 等 编著

清华大学出版社

(京)新登字 158 号

## 内 容 简 介

面向 UNIX 系统、服务于 UNIX 系统,也诞生于 UNIX 系统的研制过程中,是 C 语言最“本色”的特征;而应用于 DOS 系统,研制与开发 DOS 应用软件,又是当前 C 语言应用最为热门的领域。前者,决定了 C 语言的正规教学始终不能也不当脱开 UNIX 的背景;而后者,又决定了 C 语言的实践,必须面向最广阔的 DOS 天地。

本书正是基于此“实践”之目的,通过大量深入浅出的实例,彻底把 C 语言置身于 DOS 系统的大背景下来讲解 C 语言的。书中所涉及的 C 与汇编语言混合编程, DOS 系统资源的 C 语言访问,用 C 编写 TSR 程序、图形程序等等,都是用 C 语言编写 DOS 程序时极为关键的技术问题。

本书适于各类计算机及相关专业学生学习使用,更是各行业电脑爱好者不可多得的实用性科技读物。

**版权所有,盗版必究。**

**本书封面贴有清华大学出版社激光防伪标志,无标志者不得销售。**

### C 语言实践(一)

#### ——C 语言的 DOS 系统程序设计

吕 强 杨季文 等 编著

☆

清华大学出版社出版

北京 清华园

北京市朝阳区科普印刷厂印刷

新华书店总店科技发行所发行

☆

开本:787×1092 1/16 印张:22.75 字数:565 千字

1994 年 6 月第 1 版 1994 年 6 月第 1 次印刷

印数:00001—10000

ISBN 7-302-01567-8/TP·655

定价:25.00 元

# 前 言

开发 DOS 的原始语言环境是汇编语言(8086 系列),DOS 的许多概念是用汇编语言来表达的。C 语言作为系统开发的工具首先是面向 UNIX 的,UNIX 许多命令的语法体系有着 C 语言的影子。于是,DOS 的开发和 C 编程过去一直各自独立前进。

本书编者长期在 DOS 环境下以汇编语言辛勤耕耘着,随着工作面的不断拓宽和工作层次的逐步深入,使用汇编语言来与 DOS 交往越来越感到捉襟见肘。于是,开始以 C 语言来做一些外围工作,由于感到轻松许多,进而产生了直接用 C 开发 DOS 的想法。经过一段时期的摸索与实践,终于渐渐入门且获益颇多。因而编者希望与同行共同切磋,以达到得心应手操纵 C 为 DOS 服务的目的。

本书的主要目的是介绍如何使用 C 语言来实现开发 DOS,同时也介绍使用 C 编程的方法和技巧。懂得语言的语法,并不意味着能写出优秀的程序,好比精通汉语的人未必能成为作家一样。学习如何编程的最好方法也许就是阅读大量的程序例子,去芜存菁,提高自己。为此,本书给出了许多程序实例。

第一章是绪言,综述本书宗旨和前提。第二章阐述 C 语言和其他语言的接口,特别是与汇编的接口,它是本书的语言基础。第三章介绍 C 对 DOS 提供的软件资源的访问。第四章叙述 C 的 TSR 设计。第五章叙述驱动程序的 C 实现。第六章介绍制作键盘训练软件的实例。第七章给出一个图形格式转换器。第八章详述一个图元编辑系统。第九章介绍对串行口的编程。第十章给出对扬声器编程的实例。

本书是集体合作的成果。第一、四、五、七、八章由吕强编写,第九、十章由杨季文编写,第二、三章由吴子沂编写,第六章由陈时飏编写。另外,邵斌、洪光等参与了第七、八章的程序编写和调试工作。全书由吕强和杨季文统稿定稿,钱培德教授审阅了全稿。

由于编者水平所限,再加上时间仓促,书中谬误之处在所难免,恳请读者批评指正。

编 者

1993 年 8 月于苏州大学

# 目 录

<b>第一章 绪言</b> .....	1
<b>第二章 C语言与低、高级语言的接口</b> .....	3
2.1 C语言混合编程基础 .....	3
2.1.1 混合编程概述 .....	3
2.1.2 C语言的编译模式 .....	4
2.1.3 C语言外部接口约定原则 .....	7
2.2 C语言与汇编语言程序接口 .....	7
2.2.1 C编译程序的调用约定 .....	7
2.2.2 两种参数传递方式 .....	9
2.2.3 汇编子程序的编写格式与要求.....	10
2.2.4 从C中调用汇编函数 .....	13
2.2.5 建立汇编语言框架.....	20
2.2.6 从汇编程序中调用C语言函数 .....	25
2.2.7 C语言与汇编语言程序混合调用实例.....	28
2.3 C语言与PASCAL语言程序接口 .....	34
2.3.1 PASCAL语言简介 .....	34
2.3.2 C与PASCAL的接口程序设计基础 .....	34
2.3.3 C与PASCAL的调用约定 .....	36
2.3.4 C与PASCAL的接口程序设计举例 .....	36
2.3.5 连接C和PASCAL模块 .....	37
2.4 C语言与PROLOG语言程序接口 .....	38
2.4.1 PROLOG语言简介 .....	38
2.4.2 PROLOG混合编程基础 .....	38
2.4.3 C与PROLOG的混合编程规则 .....	40
2.4.4 C与PROLOG混合编程举例 .....	40
2.4.5 连接C和PROLOG模块 .....	41
2.5 C语言与BASIC语言程序接口 .....	42
2.5.1 BASIC语言混合编程基础 .....	42
2.5.2 BASIC调用C .....	43
2.5.3 C调用BASIC .....	44
<b>第三章 C语言与DOS操作系统的接口</b> .....	46
3.1 伪变量.....	46

3.1.1	伪变量的引入	46
3.1.2	伪变量的使用	47
3.1.3	伪变量应用举例	47
3.2	直接插入汇编代码	49
3.2.1	关键词 asm 或 _asm	49
3.2.2	asm 和 _asm 的指令集	51
3.2.3	汇编代码对 C 代码的引用	53
3.2.4	编译过程	56
3.2.5	程序举例	56
3.3	C 与 BIOS 接口	57
3.3.1	C 语言中提供的函数	57
3.3.2	C 语言对 ROM BIOS 显示驱动服务的调用	59
3.3.3	C 语言对 ROM BIOS 磁盘服务的调用	64
3.3.4	C 语言对 ROM BIOS 键盘服务的调用	66
3.4	C 与 DOS 接口	68
3.4.1	C 语言中提供的函数	68
3.4.2	C 语言对 DOS 功能服务的调用	70
<b>第四章</b>	<b>用 C 写 TSR 程序</b>	<b>73</b>
4.1	TSR 的一般讨论	73
4.1.1	概述	73
4.1.2	抢占式接管和链接式接管	73
4.1.3	TSR 编程应考虑的问题	74
4.2	与 TSR 相关的 DOS 功能调用和中断	76
4.2.1	DOS 功能调用	76
4.2.2	与 TSR 相关的中断	79
4.3	用 C 实现 TSR	80
4.3.1	中断函数	80
4.3.2	激活 TSR 的外部条件	82
4.3.3	激活 TSR 的内部条件	83
4.3.4	栈切换	84
4.3.5	保护 DOS 数据区	85
4.3.6	PSP 和 DTA 的切换	86
4.3.7	TSR 的应用部分	87
4.3.8	TSR 的撤离	88
4.3.9	TSR 的通信	89
4.3.10	TSR 的调试	90
4.3.11	确定 TSR 占用的内存	90

<b>第五章 用 C 写设备驱动程序</b> .....	92
5.1 概述 .....	92
5.2 DOS 对设备驱动程序的管理和请求 .....	92
5.2.1 设备驱动程序在 DOS 中的层次 .....	92
5.2.2 DOS 管理设备驱动程序的数据结构 .....	93
5.2.3 设备驱动程序的分类 .....	93
5.2.4 DOS 对设备的请求 .....	94
5.2.5 DOS 对设备驱动程序的调用 .....	96
5.3 设备驱动程序的 C 描述 .....	100
5.3.1 各种主要变量 .....	100
5.3.2 strategy 过程 .....	102
5.3.3 interrupt 过程 .....	102
5.3.4 各个命令处理函数 .....	103
5.4 在 C 环境下实现驱动程序 .....	105
5.4.1 数据在先 .....	106
5.4.2 标注结尾函数 .....	108
5.4.3 驱动程序的栈 .....	109
5.4.4 数据段的切换 .....	110
5.4.5 生成驱动程序的过程 .....	111
5.5 一个设备驱动程序的 C 框架清单 .....	112
5.5.1 预处理文本清单 .....	112
5.5.2 块设备驱动程序框架程序 .....	115
<b>第六章 汉字输入法演示系统的设计与实现</b> .....	124
6.1 系统概述 .....	124
6.1.1 系统开发的目的 .....	124
6.1.2 系统综述 .....	124
6.2 系统运行环境的设计与实现 .....	125
6.2.1 系统运行环境的设计 .....	125
6.2.2 系统 INT 10H 的实现 .....	126
6.3 演示模块的设计与实现 .....	135
6.3.1 演示模块的设计 .....	135
6.3.2 演示前的准备工作 .....	136
6.3.3 演示模块的实现 .....	143
6.3.4 演示模块的源程序 .....	147
<b>第七章 图形格式转换器</b> .....	167
7.1 概述 .....	167
7.2 图形格式简介 .....	167

7.2.1	WPS 桌面印刷系统的 SPT 格式 .....	167
7.2.2	MS-WINDOWS 的 BMP 格式 .....	168
7.2.3	Zsoft 的 PCX 格式 .....	169
7.2.4	STORYBOARD 的 PIC 格式 .....	171
7.2.5	CorelDraw 的 EPS 格式 .....	174
7.3	图形格式的转换模型 .....	174
7.3.1	总体设想 .....	175
7.3.2	图形文件的内存模式 .....	178
7.4	格式转换技术的应用 .....	180
7.5	一个转换器 CONVERT.C 程序清单 .....	181
<b>第八章</b>	<b>图元编辑</b> .....	<b>200</b>
8.1	概述 .....	200
8.2	用户界面设计 .....	200
8.2.1	系统文件说明 .....	200
8.2.2	图文编辑程序的使用方法 .....	201
8.2.3	如何将 MEM.OBJ 连入 C 语言的库中 .....	206
8.2.4	图元调用函数说明 .....	207
8.2.5	软件包的演示说明 .....	211
8.3	主要数据结构设计 .....	212
8.3.1	编辑板的数据结构 .....	212
8.3.2	显示板的数据结构 .....	212
8.3.3	图元在内存中的存储映像 .....	213
8.3.4	图元库的结构 .....	213
8.4	部分源程序示例 .....	214
<b>第九章</b>	<b>串行口的编程</b> .....	<b>279</b>
9.1	引言 .....	279
9.1.1	数据异步串行的发送和接收 .....	279
9.1.2	RS-232C 接口 .....	280
9.1.3	UART 内部寄存器定义 .....	280
9.1.4	有关的硬件中断及其处理 .....	286
9.2	利用 BIOS 串行通信管理程序 .....	288
9.2.1	BIOS 串行通信管理程序的功能 .....	288
9.2.2	利用 INT86 函数调用 BIOS 串行口管理程序的 C 函数 .....	290
9.2.3	一个简单的接收发送程序 .....	291
9.2.4	利用 BIOSCOM 函数实现的接收发送程序 .....	294
9.3	直接操纵异步串行通信口 .....	298
9.3.1	获取串行口的工作状态 .....	298

9.3.2	设置串行口的工作参数 .....	299
9.3.3	查询方式的发送和接收 .....	301
9.3.4	串行口测试程序 CTCOM.C .....	304
9.4	一个简单的终端仿真程序 .....	323
9.4.1	终端仿真程序 TERMINAL.C .....	323
9.4.2	再论串行口中断处理程序 .....	330
9.5	一个简单的文件传送程序 .....	331
<b>第十章</b>	<b>声音</b> .....	<b>346</b>
10.1	引言 .....	346
10.2	声音函数 .....	347
10.2.1	产生声音函数 .....	347
10.2.2	关闭声音函数 .....	348
10.2.3	延时 .....	348
10.3	实例 .....	349
10.3.1	听力测试程序 .....	349
10.3.2	音响模拟程序 .....	350
10.3.3	简单音乐演奏程序 .....	351
<b>参考文献</b>	.....	<b>354</b>

# 第一章 绪 言

DOS 系统的程序设计,是指为 DOS 开发一些实用程序,这些实用程序并不是简单地建立在 DOS 上的应用层次,它们至少要使用 DOS 层内部的功能,更多的是穿过 DOS, 直接访问 BIOS,甚至直接同硬件交往。因此,这样的实用程序是与 DOS 并级的,可以认为是对 DOS 的再开发或扩充。

诚然,实现 DOS 系统程序的“最佳”工作语言是汇编语言,这里的“最佳”是就时空效率而言的。汇编语言是做这类工作最省硬件资源的语言环境。然而,用汇编语言编制的系统程序可读性差,维护困难,开发环境也较弱。简言之,用汇编语言开发系统程序难度高,周期长。

为什么不用一种高级语言来完成这项工作呢? 答案很简单,时空效率太差。高级语言几乎与系统程序设计无缘。但是 C 语言在这方面有所突破,随着 UNIX 系统的成功,它已成为系统程序设计的新宠儿。

其实,把 C 语言称为中级语言更为确切一些,因为它既有高级语言丰富的控制结构和简洁的表达能力,又有类似汇编语言的数据类型,所以,C 成为良好的系统程序开发工具是理所当然的。具体地讲,C 能够使我们把汇编观点的数据放在高级语言的控制结构中处理,这正是系统程序设计人员求之不得的。而其他高级语言距机器太远,程序员需要有个对应的转换过程。一方面,C 语言用指针把汇编语言单一的数据类型扩充为丰富多采的数据类型,且这些数据类型与汇编语言的数据观点一致;另一方面,C 把汇编语言几乎空白的控制结构扩展为正规的高级语言控制结构,加上 C 语言的开发环境比汇编语言要高几个档次,所以, C 正逐步地成为开发 DOS 的重要言语环境。

但是,并不是说用 C 就一定能较好地完成开发 DOS 的任务,因为它毕竟具备高级语言的特征,整个开发环境是建立在 DOS 之上的。更关键的是,C 屏蔽了程序员对机器的直接控制和对 DOS 的请求,程序员只是面向 C 语言编译器,由编译连接器把程序员的请求分配给 DOS 实现。这对于 DOS 之上的应用开发来说是合理的、正确的,但对 DOS 层次上的系统程序来说却是障碍。我们的经验是:尽量少用函数,多使用控制结构来自行表达算法。因为一旦使用了函数,就将失去对程序的控制,时空效率也大为降低。但是,少用函数并不是说要禁止使用函数,只是要少用、慎用,或者应了解一下编译器对欲使用函数的处理。

用 C 语言进行 DOS 系统开发还有一个主要困难:代码和数据在内存的分配对程序员是透明的。这一点对 DOS 开发有利有弊。有利的是,我们不必为数据和代码的安排而花费额外精力,有弊的是,有时 DOS 的某些概念(或者说 Intel 芯片的概念)是用汇编语言来描述的,例如,设备驱动程序、中断等,要在这方面进行开发就回避不了内存使用的问题。

尽管使用 C 语言开发 DOS 有这样的“后顾之忧”,但是,我们的经验表明,用 C 开发 DOS 还是值得的,由此而得到的好处足以补偿损失,因为我们只要解决与 DOS 的接口即可。就开发工作本身来说,用 C 语言和汇编语言来实现,其难易程度、可靠性和强壮性等方面有天壤之别。问题是,能否保证用 C 语言开发出 DOS 要求的系统程序(质量好坏暂且不提)? 幸运的是,实用的 C 语言版本都提供嵌入汇编和方便地连接汇编的功能,对于不满足

我们要求的 C 程序段,可以用汇编语言来实现。所以,系统程序员充其量编写一个 C 的汇编版程序,当然这是极限情况。

本书的编者用 C 为 DOS 开发了许多系统程序,有的只是我们自己科研工作中需要的工具,有的极有商品化的价值。这类开发工作越多,我们就越感到,用 C 语言进行 DOS 系统程序开发是值得提倡的。

DOS 上实用的 C 分为两大版别,一是 Borland 公司的 Turbo C,一是 Microsoft 公司的 C。本书采用 Borland 的 Turbo C,书中所有程序均在 Turbo C 2.0 上通过。除非特别声明,本书中的 C 就是指 Turbo C 2.0,这是作者在准备书稿时的流行环境,故建议读者阅读本书时参见 Borland C++ AF 版本。

本书中的程序还大量使用了 DOS“未公开”的功能。这是一个敏感的问题,这方面的是是非非已超出本书的讨论范围。不过作者认为,进行 DOS 系统开发,没有这些“未公开”功能的支持,是永远也开发不好的。事实上,有些所谓“未公开”功能已经公开了。关于这方面的情况,请参考书末的参考文献。使用“未公开”功能要影响程序的稳定性和通用性。除非特别声明,涉及到“未公开”功能的程序均在 DOS 3.3 上通过。

书中给出了大量 C 源程序,既侧重于 DOS 的再开发,又兼顾了 C 语言的高级程序设计,这对于 DOS 程序员和学习 C 语言的读者都是有所帮助的。书中所有源程序均存放在软盘片上,感兴趣的读者可与清华大学出版社软件部(电话 2594891)或北京科海培训中心(电话 2569289)联系。

## 第二章 C 语言与低、高级语言的接口

本章主要介绍 C 语言与汇编语言的调用接口,并介绍 C 语言与其他高级语言(如 PASCAL,PROLOG,BASIC)的调用接口。

### 2.1 C 语言混合编程基础

#### 2.1.1 混合编程概述

作为一种中级语言,C 语言毫无疑问是杰出的,它的工作方法独特,为用户提供了很强的功能和灵活性。然而,没有一种语言是十全十美的,在实际开发和编制软件过程中,人们常常需要使用多种语言混合编程,从而充分利用各种语言的特色,使开发和编程工作达到事半功倍的效果。

在进行混合语言编程时,一项作业将被分成若干功能模块,每个模块以函数或过程的形式存在,针对每一功能模块的特点选用适合的语言独立编程。例如,涉及低级处理和中断操作的功能模块通常选用汇编语言编制,而涉及推理和问题搜索的功能模块通常选用 PROLOG 语言编制。每个模块编制好后,就要使用相应的语言编译程序对其进行编译形成目标文件,最后将多个目标文件连接在一起形成一个完整的可执行文件。整个工作过程如图 2-1 所示。

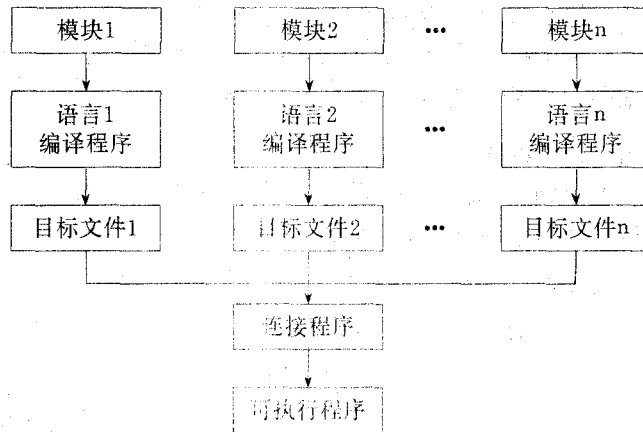


图 2-1 混合编程工作过程

在进行混合语言程序设计时,要注意两个关键问题:

(1) 要选择一个恰当的主模块。主模块是主程序所在的模块。要形成最后的可执行代码,没有主程序是不行的。一般,总是选择一个最大的模块作为主模块。这样,在进行混合编程时,总会有一个主语言,当用所选主语言编程很难实现或无法实现其功能时,才选用其他合适的语言针对该功能独立编程。而某些语言由于自身编译的限制,使其在进行混合编程

时,必须以主模块身份存在,这个问题将在后面详细讨论。

(2) 要严格遵守语言之间的调用约定,即建立语言之间的正确接口。不同的语言之间或多或少存在着差异,如数据类型、函数参数的传递方式等等,这就要求在进行混合编程时,除了掌握所用语言之外,还必须熟悉各种语言之间的命名约定、参数传送协议以及调用约定,并在实际编程中遵守这些要求。

## 2.1.2 C 语言的编译模式

在进行 C 语言混合编程前,首先要了解 C 语言的各种编译模式。这是因为,选择适当的编译模式对提高混合编程效率是很重要的,有时这甚至是混合编程成功与否的决定因素。

Microsoft C 与 Turbo C 都提供了六种编译模式,下面将逐一介绍这六种模式。

### 1. 微模式 (Tiny Model)

在该模式下,所有 4 个段寄存器 (CS、DS、SS、ES) 都为同一个值。程序和数据、堆栈都在同一段内,即所有的寻址都是 16 位。相应地,C 源程序中的所有指针都是近指针 (NEAR),所有的调用都是近调用。这种模式所生成的程序,在连接时加参数 /t 可以转化为 .COM 文件。

### 2. 小模式 (Small Model)

在小模式中,数据段和代码段分离,即最大可用空间为 128K,这种模式适合于大多数程序。由于寻址仍是 16 位,所以所有调用仍是近调用,所有指针仍是近指针,但它同时允许对个别不在代码段内的函数有 far 关键字来调用,对个别不在数据段内的数据也可以用 far 或 huge 关键字来修正指针。

### 3. 中模式 (Medium Model)

中模式中允许有多个代码段,但数据段仍只有一个。代码段采用 20 位寻址,数据段仍采用 16 位寻址。在一个代码段中缺省调用是近调用,代码段之间的缺省调用是远调用,但也可使用 near 关键字来推翻缺省约定。通常,一个 C 函数就形成一个独立的代码段。

### 4. 紧凑模式 (Compact Model)

紧凑模式与中模式相反,它只有一个代码段,却允许有多个数据段,也就是说代码仍是以 16 位进行寻址,而数据却要用 20 位进行寻址。

### 5. 大模式 (Large Model)

大模式允许有多个数据段和代码段,但全部静态数据仍限制在一个段 (64K) 内。

### 6. 巨模式 (Huge Model)

巨模式与大模式的唯一不同之处是,前者不再要求静态数据必须在一个段内。

显然,这六种编译模式适用不同的场合,其中微模式所产生的文件执行速度最快,而巨模式所产生的文件执行速度最慢。在进行单纯的 C 语言程序设计时,如果代码段和数据段都不是很大 (64K 以内),则通常选用小模式,这时所产生的文件执行速度接近于微模式。但在进行接口程序设计时,数据和代码都有可能不在同一段内,因而要选择中模式甚至大模式进行编译。

这六种编译模式对应着六种存储模式,其在内存中的分配情况如图 2-2~图 2-7 所示。

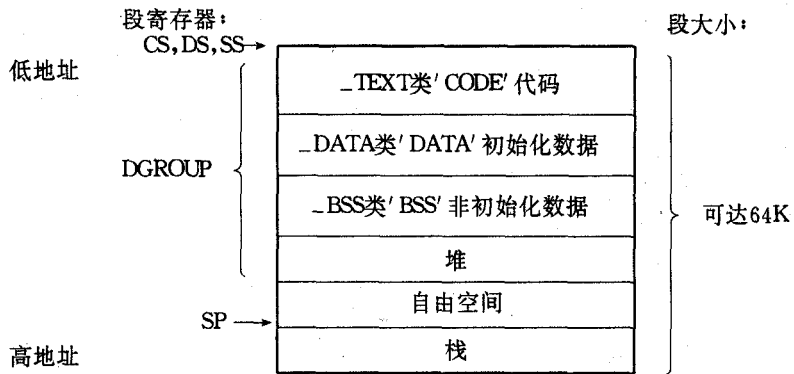


图 2-2

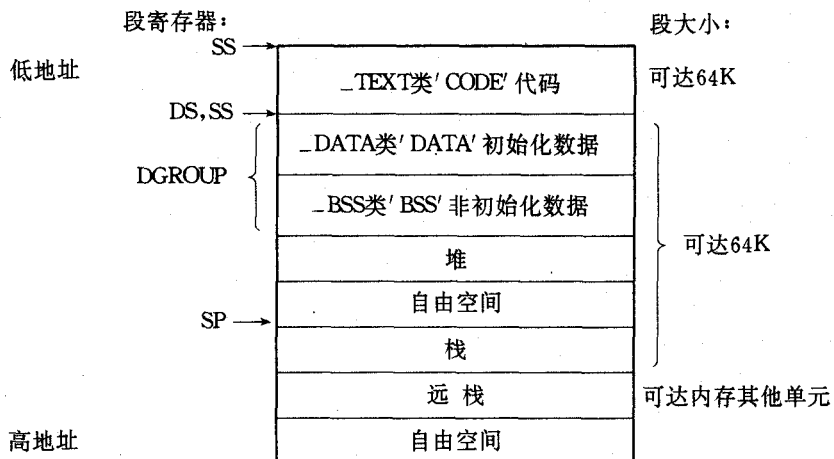


图 2-3

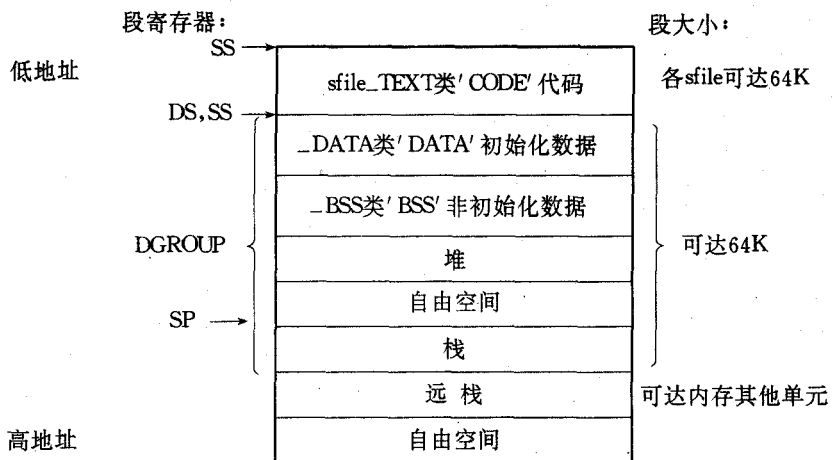


图 2-4

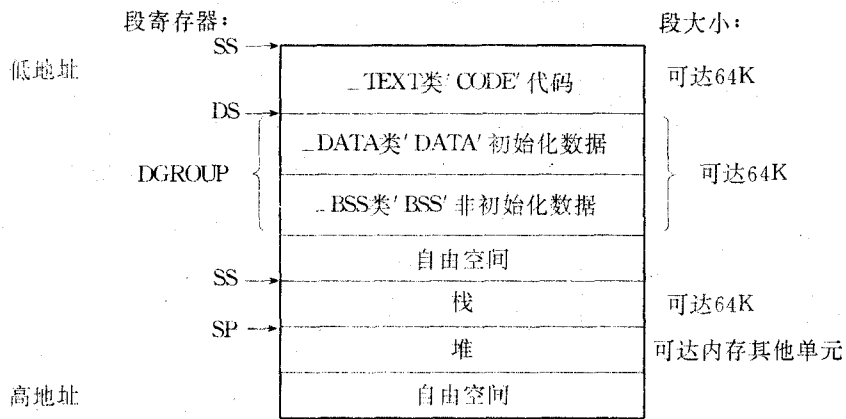


图 2-5

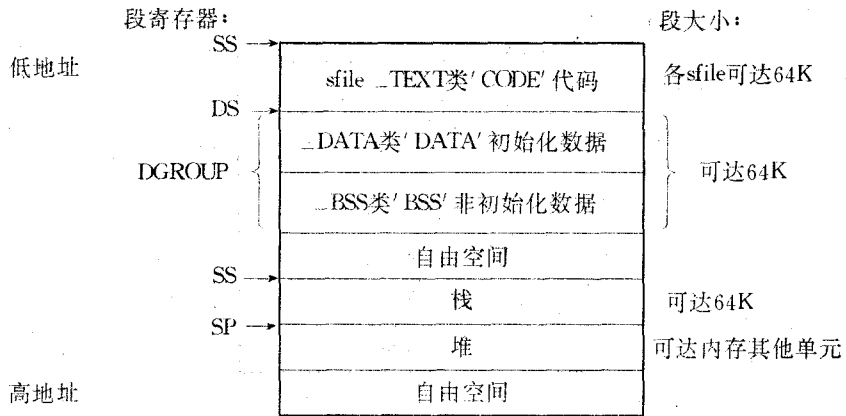


图 2-6

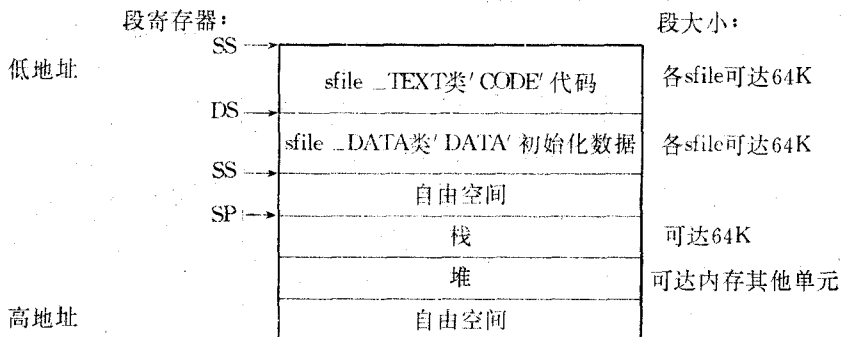


图 2-7

### 2.1.3 C 语言外部接口约定原则

#### 1. C 语言关键词 extern

extern 用于说明一个变量或函数是外部的,即该变量或函数存在于另一个独立的程序中。格式如下:

```
extern <变量名>;
```

```
extern <函数原型>;
```

例如说明:

```
extern int outv;
```

```
extern void outf(int a);
```

表明 outv 是一个外部变量,而 outf 函数是一个外部函数。外部变量和外部函数只要说明正确,C 程序就可按内部变量来使用或按内部函数的调用方式来调用。

#### 2. C 语言的编译模式

要符合与其接口的语言的要求。一般情况下,C 语言与高级语言(如 PASCAL,PROLOG 和 BASIC 等)所采用的编译模式通常是大模式。

以上介绍的是用 C 语言进行接口程序设计时通常要遵守的几条原则。事实上,当 C 语言与某种具体语言混合编程时,会有更多的接口设计原则要遵守。在本章后面几节中,将详细介绍 C 语言与汇编语言的混合编程规则,并将介绍 C 语言与 PASCAL、PROLOG 和 BASIC 三种高级语言的接口程序设计原则。

## 2.2 C 语言与汇编语言程序接口

汇编语言程序与高级语言相比,有几个主要特点:它能产生最快的可执行目标代码,程序更为紧凑,且节省内存空间,能直接控制计算机硬件(如显示器、打印机、存储器、磁盘等)的操作,完成一些高级语言难以做或无法做的工作,以及可直接与操作系统进行接口等等。把汇编程序模块和用户的 C 程序结合起来,这必将增强编程的灵活性,提高程序的执行速度和效率。

### 2.2.1 C 编译程序的调用约定

调用约定是指 C 编译程序将其信息传递给被调函数并从被调函数返回值的方法。通常包括:

- (1) 要传递给被调函数的参数放在何处。
- (2) 用什么汇编指令来调用函数。
- (3) 被调函数得到控制权后系统处于何种状态,哪些寄存器可以破坏,哪些寄存器需要保存。
- (4) 被调函数的返回值(如果有的话)应存放在何处。

在一般情况下,C 语言是利用堆栈将参数传递给被调函数的。如果参数是七种内部数据类型之一(即 char,short int,int,long int,unsigned int,float,double)或者是一个结构,那么数

据的实际值就被置于栈顶。如果参数是一个数组,那么就把数组的地址置于栈顶。表 2-1 给出了各种变量在栈中所占用的字节数。

表 2-1 C 函数在传递变量时各种数据类型在堆栈中所占字节数

类 型	字 节 数
char	2
short	2
signed char	2
signed short	2
unsigned char	2
unsigned short	2
int	2
signed int	2
unsigned int	2
long	4
unsigned long	4
float	8
double	8
(near)pointer	2(offset)
(far)pointer	4(segment and offset)

Microsoft C 与 Turbo C 参数是由右至左顺序压入栈中(除非用 Pascal 参数的传递方法,这将在下面叙述),接着压入调用函数的返回地址。函数调用是通过 call 指令实现的。可能是近调用,也可能是远调用,这要依 C 程序所选模式而定。如果是 near 近调用,则只需压入偏移地址;如果是 far 远调用,则需压入段地址及偏移地址。

被调函数开始执行时,便从栈中取出参数的值进行相应的运算,而当被调函数运行结束时,函数将返回值传给调用函数(如果有的话),这个值通常被放在 AX 寄存器或 DX 寄存器中。表 2-2 给出了 C 函数的返回值的寄存器使用情况。其中,float、double、struct、union 的回送方法是将值放入静态数据区,然后返回其地址指针;在小模式下,地址指针放入 AX 中,大模式下地址指针 DX : AX 中。

表 2-2 C 函数返回值的寄存器使用情况

类 型	寄存器与含义
char	AX
unsigned char	AX
short	AX
unsigned short	AX
int	AX
unsigned int	AX
long	低位在 AX 中,高位在 DX 中
unsigned long	低位在 AX 中,高位在 DX 中
float & double	指向静态存储区,指针在 AX 中
struct & union	指向静态存储区,指针在 AX 中

类 型	寄存器与含义
(near)pointer	偏移量在 AX 中
(far)pointer	偏移量在 AX 中,段值在 DX 中

那么,在进入用汇编语言编写的被调函数时,哪些寄存器需要保存呢?首先 BP(基指针)必须压入栈中保存,然后 SP(堆栈指针)的当前值放入 BP 中,以便子程序中可以使用堆栈,任意存取数据。另外两个需要考虑的寄存器是 SI 和 DI。如果在汇编子程序中需使用它们,则在子程序入口处亦需保存它们。这样,在汇编语言子程序返回之前,需恢复 BP、SI 和 DI,并且让堆栈指针复位。

### 2.2.2 两种参数传递方式

C 语言支持两种函数参数传递方式:标准 C 语言方式和 Pascal 语言方式。区别如下:

(1) C 语言方式函数的参数是按自右至左的次序压栈,而 Pascal 语言方式函数的参数是自左至右压栈。

(2) C 语言方式无需知道栈上所压参数的个数,它只假设所有参数均在栈中,而 Pascal 语言方式需知道向其传递的参数个数。

(3) C 语言方式在被调子程序中无需将参数弹出栈,只需在主程序中调用指令后加入 add sp,n(n 为参数所占字节数)指令或其他指令来调整栈。而 Pascal 语言方式是在被调子程序中根据参数的个数,在子程序最后用 ret n 清栈,这样,在主程序中,调用指令后不再出现调整栈操作。

假设有函数:func(int p1,int p2),有下列 C 主函数调用:

```
extern func (int a,int b);
main ()
{
    int a=8
    long b=0x1234abcd;
    func (a,b);
    :
}
```

用 C 语言方式,栈的内容如下:

```
sp+06 : 1234H
sp+04 : abcdH      ;b=p2
sp+02 : 0008H      ;a=p1
sp      : * * * *      ; * * * * 为返回地址(假设为小模式)
```

上述主函数经编译后产生的汇编代码为:

```
mov si,8                ;置 a=8
mov word ptr[bp-2],1234H ;置 b=0x1234abcd
mov word ptr [bp-4],abcdH ;
push word ptr [bp-2]     ;压入 b
push word ptr [bp-4]     ;
push si                  ;压入 a
call near ptr _func      ;调用 func(压入地址)
```