目　录

第一章 C/C++概述

本章讨论C程序设计语言的基本特点、发展过程、应用范围以及C++面向对象的程序设计方法(OOP)。

1.1 C语言的起源

C语言最早是由Dennis Ritchie在DEC PDP-11上的UNIX操作系统环境下实现的。C语言的前身是由Martin Richards设计的BCPL语言,它深刻地影响了Ken Thompson设计的B语言的风格,后者在1970年发展成C语言。由于C语言在各种操作系统下实现起来非常方便灵活,因此各种C语言的实用版本纷纷出现。在不同版本上,C语言的源程序可以奇迹般地保持高度兼容,因此使得C语言广为流行。1987年美国国家标准局ANSI建立的ANSI标准被广为采用,现今所有主要的C编译器均已经实现或扩充了ANSI标准。在微机上最广泛流行的有Microsoft公司的Microsoft C/C++系列和Borland国际公司的Borland C/C++系列。本书内的全部源程序均采用Borland C++语言规范编写。

1.2 C语言的特点

1.2.1 C是一种中级程序设计语言

我们之所以把C语言称为一种中级程序设计语言,是因为C既有高级语言的一些特征,同时又具有和汇编语言相似的处理能力。典型的高级语言如FORTRAN、Pascal、Ada、BASIC、COBOL等等大都不支持对字节、字位和地址等的直接操作,而C语言则支持这些。C语言和其他所有的高级语言一样都支持数据类型这个概念,尽管C语言只有五种基本类型,但它并不像Pascal或Ada语言那样对类型要求那么严格。C语言几乎可以在所有类型之间相互转换。C语言可以对字位、字节、字和指针直接进行操作,这使得它很适合系统程序设计。C语言的另一特征是它的关键字很少,ANSI标准定义的仅有32个关键字。下面表1.1至表1.4列出的是ANSI C标准关键字表、Borland C++对ANSI C的关键字扩充表、C++中特有的关键字表以及Borland C++中的寄存器变量。最后我们在表1.5中总结了Borland C++的全部关键字。

1.2.2 C是一种结构化程序设计语言

不言而寓,C是一种结构化程序设计语言。C语言的主要结构就是函数,也就是C中的独立的子程序。在C语言中,函数被描写成一个块状结构,所有的程序动作在其内部进行。因而可以把程序每个任务的定义和编程独立实现,即实现程序的模块化。C语言结构化的另一典型特征就是使用代码块,即可以在相对独立的语句组中使用{ }来构成一个逻辑单位,

同时执行。使用代码块可以使算法更加清晰、优美和高效，也便于阅读。

表 1.1 ANSI C 关键字表

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			

表 1.2 Borland C++关键字扩充表

_asm	asm	_cdecl	cdecl	_cs
_ds	_es	_export	_far	far
_fastcall	huge	interrupt	_loadds	_near
_pascal	pascal	_saveregs	_seg	_ss

表 1.3 C++中特有的关键字

asm	operator	class	private	delete
protected	virtual	friend	public	inline
template	new	this		

表 1.4 Borland C++中的寄存器变量

_AH	_AL	_AX	_BH	_BL	_BX
_CH	_CL	_CX	_DH	_DL	_DX
_DI	_DS	_ES	_FLAGES	_SI	_SP
_SS					

表 1.5 全部 Borland C++关键字

_asm	asm	auto	break	case
_cdecl	cdecl	char	class	const
continue	_cs	default	delete	do
double	_ds	else	enum	_es
_export	extern	_far	far	_fastcall
float	for	friend	goto	_huge
huge	if	inline	int	_interrupt
interrupt	_loadds	long	_near	near
new	operator	_pascal	pascal	private
protected	public	register	return	_saveregs
_seg	short	signed	sizeof	_ss
static	struct	switch	template	this
typedef	union	unsigned	virtual	void
volatile	while			

所有的C语言程序都包含一个或多个函数。唯一不可缺少的是称为main()的主函数。它在程序开始执行时第一个被调用。在优秀的C语言程序中,main()函数实质上总是勾勒出程序的轮廓。这种轮廓即由一系列的函数调用组成。

1.2.3 C是面向程序员的语言

C语言完全是由程序员建立、影响和使用的语言。也就是说,C语言提供了程序员想要的一切:很少的限制,没有什么修饰,块结构,独立函数和极少的关键字。使用C语言可以使程序员达到使用汇编语言的执行效率,同时还保留了高级语言的许多特性,尤其是结构化特性、宜于移植和可读性。

随着C语言的流行,很多程序员利用它的可移植性和高效性开始编制各类实用程序。由于大多数机器上都支持C语言编译器,因此在一种机器上编制的程序可以不加修改或只要很少的修改就能运行在另一台机器上。这样节约的金钱和时间自不必说了。还有,每个C语言程序员都可以建立和维护自己特有的函数库,以适应他自己的风格并能应用于各种不同的程序中。本书中所提供的全部源程序均为笔者从自己收集和创建的函数库中精选出来的,读者可将其加入自己的C语言函数库中以便随时使用。

1.3 库、头文件、编译和链接

所有的C语言编译器都附带一个标准函数库,用来提供常用的操作。ANSI C规定了包含在库中的最小函数集。我所使用的Borland C++ 3.0/3.1编译器提供了大约600个函数和宏,用户可以在自己的C和C++程序中调用它们以执行各种任务,包括低级和高级I/O、串和文件操作、内存分配、过程控制、数据转换、数学计算、图形生成等等。

和其他C语言编译器一样,所有Borland C++库函数在一个或多个头文件中都给出原型声明。头文件也叫包含文件,提供库函数的原型声明。库函数所用数据类型和符号常量也定义在这些文件中,另外还有Borland C++和库函数定义的全程变量。

在许多C语言编译器中,编译和链接是一件比较繁琐费时的事。Borland C++编译器提供了两种简便的编译链接方法:MAKE实用程序和PROJECT项目管理器。后者主要用于集成环境调试,前者主要用于生成最后使用版本。具体使用方法可以参见Borland C++使用手册。

1.4 C语言的存储映像

一个编译好的C程序建立并使用四个独立的逻辑区域,这些内存区域用于不同目的(图1.1)。

第一个内存区域用来存放程序的代码部分,下一个区域用来存储全局变量,余下的两个部分分别称为栈和堆。栈在程序运行时起着很大的作用,它保留着函数调用的返回地址、函数的自变量以及局部变量。同时它也用来保存当前CPU的状态。而堆是一块空闲的内存区域,当你的程序要使用C语言的动态存储分配函数时就要用到。

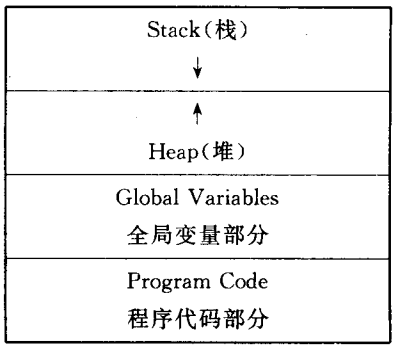

图 1.1 一般性的 C 程序存储映像

程序的确切内存分布可能随不同的编译器和不同的环境而变化。我们微机上最常使用的 80×86 类处理器的大多数编译器都用六种不同的方法组织内存的结构。这是因为 80×86 系列的段存储结构的原故。据此,程序设计人员就要根据内存使用情况选择最适合的模式,以减少代码量,增加程序运行速度。80×86 系列处理器的存储模型在本书的第二章中我们还要详细讨论。

1.5 面向对象的程序设计方法 OOP

近年来,面向对象的编程技术(object-oriented programming 简称 OOP)越来越受到程序设计人员的普遍关注。OOP 的目的十分明确:使编程更容易。随着用户要求程序的功能越来越多,程序的编制越来越复杂,对一个好的软件来说,这种应变能力就显得越来越重要。这就使得面向对象的语言如 C++等迅速发展起来。

最早具有面向对象程序设计思想的语言是 SIMULA 语言,随后又出现了 SMALLTALK 等。现在广泛应用的是美国 AT&T 公司 BELL 实验室的 B. Stroustrap 在 C 语言基础上扩充引进面向对象的程序设计思想发展的 C++。而后,美国 AT&T 公司以及 Microsoft 公司、Borland Inc. 公司等相继推出了自己的版本。在国内较流行的主要是后两家的 Microsoft C++系列和 Borland C++系列。Microsoft 公司最近又推出了 Microsoft C++系列的换代产品 Visual C++ 系列,Borland 公司也即将推出 Borland C++ V3. X 的后继版本 Borland C++ V4. 0。本书涉及的全部源程序均使用 Borland C++ V3. x 调试通过,一般不必或稍作修改就可以适用于其他的 C++版本。

1.5.1 OOP 技术要点

OOP 认为世界由各种对象(object)组成,任何事物都是对象,是某个对象类(class)的实例(instance)。复杂的对象可以由简单的对象以某种方式组成。其二,OOP 认为所有对象都可以划分成各种对象类,在每个对象类上都定义了一种方法(method),所谓方法就是施加于该类对象上的某种操作。对象和传统的数据有着本质的区别,它不像数据那样被动地等待对它执行某种操作,相反,它是进行数据处理的主体,必须发送信息(或消息)请求对象执行它的某个动作,处理它的私有的或公有的数据。在外界不能直接对它的私有数据执行操作。

其三,OOP 方法学认为,对象之间除了互相传递消息的联系之外,再没有其他联系。一切基于对象的信息和实现方法都被封装(encapsulation) 于相应的对象类的定义之中,在外界是不可见的。其四,对象类是按照基类、派生类的关系构成一个或多个层次结构系统。在这种层次结构中,上层对象类(即基类)所具有的性质可以被下层对象类继承(inheritance),除非在下层对象类中又对相应的属性作了重新描述(即再定义,又称重载- Overloading),这时要以新的属性为准。总而言之,OOP 技术就是以各种对象类来模拟现实世界的有力工具。

1.5.2 OOP 的基本组成部分

掌握 OOP 编程技术首先要弄清 OOP 的基本组成。在笔者看来,OOP 的基本组成部分实质上无外乎下面五种:

(1) 类(Classes);

(2) 对象(Objects);

(3) 实例变量(Instance Variables);

(4) 消息(Messages);

(5) 方法(Methods);

学习如何以面向对象方式编程的首要一点是弄清类与对象之间的区别。对象是数据及可以对这些数据施加的操作结合在一起所构成独立实体的总称。而类是一组具有相同数据和相同操作的对象的描述(或定义)。也就是说,类是对一组对象的概括,而每个对象都是某个类的一个具体实例。类是一种模板,用来定义一个对象。同时,对象本身并不包含实现操作的程序代码,这些代码是在类中定义的。

一个类包括两个基本组成部分:实例变量和方法。实例变量是由某个特定类所描述的一个对象,通常,对象和类的实例是同义词。方法就是对象所能执行的操作。在C++中我们称之为成员函数(member function)。它是类中定义的函数,描述对象执行操作的算法,或响应消息的方法。一句话,实例变量可视为数据分类,而方法表示函数。

一般说来,消息是要求某个对象执行类中所定义的某个操作的规格说明,通常它由三部分组成,即接受消息的对象、消息选择子(又称消息名)和零个或多个变元(实参)。

1.5.3 OOP 的四大特点

当使用 C++的面向对象编程时,必须弄清它的四大特征,即:封装、继承、多态和重载。

(一) 封装(*Encapsulation*)

简单地讲,封装就是将数据和对这些数据进行处理所需要的各种操作连接在一个根下的技术。封装给我们提供了两个重要的特征:

(1) 使得数据和函数在一个根下;

(2) 使得数据具有隐蔽能力。

例如,当我们编写数据库对象时,我们感兴趣的是怎样访问数据库中的这个对象,而不会去关心这些数据对象是如何存储在数据库系统中的。又如我们使用计算机及其操作系统,但是我们并不了解也不需要了解它们工作的细节。通常,封装有三个目的。第一,保护对象

的数据，只有该对象内部定义的函数过程才能访问它；第二，封装使我们运用这些数据更加方便，因为所有的操作只能通过该对象已定义过的接口；第三，封装可以隐蔽数据的内部存储机制和如何实现各种操作的细节。

（二）继承（*Inheritance*）

OOP 是一个建立类的等级的过程，这种等级的建立体现在它的继承上。也就是说，类可以从较简单普遍的类中继承特征，这种方式就如同生物学的分类。继承的主要特点有：

（1）在没有改变原始类的情况下，增加一些数据和代码；

（2）重复使用代码；

（3）改变一个类的性质。

继承的一个典型应用是，可以任意修改一个现有的类来产生一个与这个类有细微差别的新类，便于我们使代码大量的重用成为可能。

（三）多态性（*Polymorphism*）

多态性一词来源于希腊语："有多种形态"。多态性的一个重要特点是：在一个类中的对象可以有同样名称的不同方法，这些方法的代码可以完全不同。在 C++中，多态性使用虚函数来实现。虚函数可以使你在一个类的等级中使用相同函数的多个版本，在运行时决定所用的特定版本（称作迟后联编）。

（四）重载（*Overloading*）

重载也就是重载函数，这意味着我们可以有相同名字的函数，但使用时带有不同的数据类型。如，在 C++中，我们计算一个数的立方，可以不必像在 C 语言中那样分别根据类型写成不同的函数名，我们就可以这样说明：

```
int cube(int number);
float cube(float float_number);
double cube(double double_number);
```

只要参数表不同，对于给定的参数，C++能自动选定正确的函数。这就叫重载。

1.6 C++是 OOP 技术的优秀实现

上面我们讲了 OOP 技术的一些基本特点，而对这些 OOP 技术的实现我们选择了发展比较成熟的 C++。C++继承了 C 语言所具有的精华，又发展并融进了 OOP 的最新的进展，为我们建立现实世界系统的计算机模型提供了一个很自然而又功能强大的新的工具。本书中充分使用 Borland C++ 3.x 版的 C++新特征，详细描述了流行的图形模式、图像格式、内存驻留、图像截取和汉字处理等 C/C++处理的崭新的特征。

第二章　系统资源的调用

基本输入输出系统(BIOS)是计算机操作系统最低的软件成份。在个人计算机系统中，BIOS 通常由用于测试系统硬件并引导加载操作系统的例程和一组设备驱动程序组成。这些驱动程序为系统的各种 I/O 部件(如键盘、显示器、硬盘驱动器、软盘驱动器、实时时钟、并行口、串行口等) 和其他部件提供服务。在基本输入输出系统(BIOS)之上就形成了各种操作系统。在我国乃至世界 80×86 体系个人计算机系统中，由 Microsoft 公司研制的 PC-DOS 或 MS-DOS 操作系统(简称 DOS) 一直是最流行的操作系统。本章即将讲述 BIOS 和 DOS 的 C/C++调用方式。

2.1　80×86 体系结构

由于本书是一本讲述 Borland C/C++高级应用的指导性教材，在这里我们假定读者已经熟悉 80×86 体系结构和 80×86 汇编。不熟悉这些的读者可以先阅读有关 80×86 的书籍，如参考文献[5]。

2.2　中断向量

IBM PC 系列微机是以中断方式工作的。在 PC 内存低端的 1K 字节，由 BIOS 和 DOS 共同组成了 256 个中断向量(见表 2.1)，处理系统中所有的 I/O 操作和其他的服务。所有的应用程序都是使用 BIOS 和 DOS 提供的这些中断例程来访问系统的软硬件资源的。表 2.1 是在 80386 系统下使用 PCTOOLS 的 SI.EXE 测定的 DOS 6.0 的中断向量一览表。

表 2.1　软中断一览表

Number	Address	Label	Owner
00	4D51:00DD	Division by zero	SI.EXE
01	13B4:CFF3	Single Step	SI.EXE
02	0F57:0016	NMI / Parity Check	Stacks
03	13B4:CFF3	Breakpoint	SI.EXE
04	0070:06F4	Overflow (INTO)	SYSTEM
05	F000:FF54	Print Screen	BIOS
06	F000:7059	Invalid opcode (2/3/486)	BIOS
07	F000:7059	Coprocessor emulation (2/3/486)	BIOS
08	E015:17AC	Timer-tick Hardware Interrupt	SMARTDRV
09	E015:185A	Keyboard Hardware Interrupt	SMARTDRV
0A	F000:7059	Cascaded Interrupt Controller	BIOS

表 2.1 软中断一览表(续)

Number	Address	Label	Owner
0B	F000:7059	Asynchronous adapter	BIOS
0C	F000:7059	Asynchronous adapter	BIOS
0D	F000:7059	Segment overrun	BIOS
0E	0F57:00B7	Diskette hardware Interrupt	Stacks
0F	0070:06F4	Printer hardware Interrupt	SYSTEM
10	1094:0A36	Video Functions	MOUSE
11	F000:F84D	Equipment installed	BIOS
12	F000:F841	Memory size	BIOS
13	E015:17FC	Diskette/fixed disk	SMARTDRV
14	F000:E739	BIOS Asynchronous (COM Ports)	BIOS
15	E015:18D7	Cassette/Miscellaneous	SMARTDRV
16	F000:E82E	Keyboard	BIOS
17	F000:EFD2	Printer (LPT1,2,3)	BIOS
18	F000:FF53	ROM BASIC entry	BIOS
19	E015:18C7	Bootstrap loader	SMARTDRV
1A	F000:FE6E	Time of day get/set	BIOS
1B	3E0C:E56F	Keyboard control-break	SI. EXE
1C	F000:FF53	Auxillary timer-tick	BIOS
1D	F000:F0A4	Pointer: Video parameters	BIOS
1E	0000:0522	Pointer: Diskette parameters	SYSTEM
1F	C000:6C0C	Pointer: Extended Video Characters	SYSTEM
20	011C:1094	DOS program terminate	SYSTEM
21	E015:15FA	DOS function call	SMARTDRV
22	0FD1:02B1	Storage: DOS terminate Address	COMMAND
23	0FD1:014A	DOS control-break exit	COMMAND
24	52E2:0045	DOS critical error	SI. EXE
25	E015:1915	DOS absolute disk read	SMARTDRV
26	E015:195E	DOS absolute disk write	SMARTDRV
27	011C:10BC	DOS terminate & stay resident	SYSTEM
28	E015:15AE	DOS idle	SMARTDRV
29	0070:0762	DOS Fast Character to Screen	SYSTEM
2A	011C:10DA	Local Area Network	SYSTEM
2B-2D	011C:10DA	Local Area Network	SYSTEM
2E	0FD1:013F	Reserved for DOS	COMMAND
2F	E015:1368	Reserved for DOS	SMARTDRV
30	1C10:D0EA	Reserved for DOS	SI. EXE
31	F000:7001	Reserved for DOS	BIOS
32	011C:10DA	Reserved for DOS	SYSTEM
33	3E0C:2E64	Mouse driver	SI. EXE
34-3E	011C:10DA	Reserved for DOS	SYSTEM
3F	011C:10DA	Overlay Manager	SYSTEM
40	F000:EC59	Revectored INT 13h	BIOS

表 2.1 软中断一览表(续)

Number	Address	Label	Owner
41	F000:E4C1	pointer: Fixed disk parameters	BIOS
42	F000:F065	Revectored INT 10h (by EGA)	BIOS
43	C000:680C	pointer: EGA graphics font	SYSTEM
44	F000:7059	Novell Netware	BIOS
45	F000:7059	Reserved for DOS	BIOS
46	F000:E401	pointer: Fixed disk parameters	BIOS
47-49	F000:7059	Reserved	BIOS
4A	F000:7059	Real-Time Clock alarm interrupt	BIOS
4B	C801:10D7	Reserved	SYSTEM
4C-5F	F000:7059	Reserved	BIOS
60-66	0000:0000	User Programs	No Owner
67	02B9:02B0	Expanded Memory Manager	EMM386
68-6C	F000:7059	Reserved	BIOS
6D	C000:0BB4	Reserved	SYSTEM
6E-6F	F000:7059	Reserved	BIOS
70	0F57:0052	Real Time Clock	Stacks
71	F000:765F	IRQ9 Software diverted to IRQ2	BIOS
72	F000:7059	IRQ 10 Reserved	BIOS
73	F000:7059	IRQ 11 Reserved	BIOS
74	1094:173C	IRQ 12 Mouse / Reserved	MOUSE
75	F000:7650	IRQ 13 Numeric CoProcessor	BIOS
76	0F57:0117	IRQ 14 Fixed Disk Controller	Stacks
77	F000:7059	IRQ 15 Reserved	BIOS
78	0000:0000	Reserved	No Owner
79	0000:0000	Reserved	No Owner
7A	0000:0000	Novell Netware	No Owner
7B-7F	0000:0000	Reserved	No Owner
80-E3	0000:0000	BASIC	No Owner
E4	0300:01FE	BASIC Interpreter	EMM386
E5	0D0B:0101	BASIC Interpreter	DBLSPACE
E6	0BF6:1D68	BASIC Interpreter	DBLSPACE
E7	0000:0E0B	BASIC Interpreter	SYSTEM
E8	0300:0D0E	BASIC Interpreter	DBLSPACE
E9	0200:000B	BASIC Interpreter	SYSTEM
EA	0000:03E0	BASIC Interpreter	DOS
EB	0A8E:0000	BASIC Interpreter	DBLSPACE
EC	0046:C000	BASIC Interpreter	DBLSPACE
ED	F000:3A6C	BASIC Interpreter	BIOS
EE	0280:0246	BASIC Interpreter	HIMEM
F0	0000:0040	BASIC Interpreter	DOS
EF	6B8C:1100	BASIC Interpreter	Free Memory

表 2.1 软中断一览表(续)

Number	Address	Label	Owner
F1	020D:0EA3	User Programs	EMM386
F2	01F7:07C0	User Programs	HIMEM
F3	8000:0102	User Programs	Free Memory
F4	00FA:55F2	User Programs	DBLSPACE
F5	0202:52E8	User Programs	DBLSPACE
F6	0200:07C0	User Programs	HIMEM
F7	7BF6:000B	User Programs	Free Memory
F8	07C0:7BF6	User Programs	Stacks
F9	0200:0040	User Programs	SYSTEM
FA	7BF6:000B	User Programs	Free Memory
FB	0000:03F6	User Programs	DOS
FC	0004:0080	User Programs	DOS
FD	0246:0001	User Programs	SYSTEM
FE	36A8:3986	User Programs	SI.EXE
FF	0060:F000	User Programs	Stacks

2.3 改变中断向量

我们知道,中断向量存储在微机内存最前面的 400H 个字节中。每一个向量的长度是由四个字节组成的,这四个字节内所存放的是中断处理程序的首地址。其中前两个字节是中断处理程序首地址的偏移量(offset),后两个字节是中断处理程序首地址的段值(segment)。

中断向量一般有两种修改方法。一是可以直接设置中断向量的地址值,或是使用 DOS 所提供的系统调用设置中断向量的地址值。

2.3.1 直接设置中断向量

因为中断向量只是存放地址值的存储单元,因此我们可以直接地把地址值存放到中断向量的存储单元中。比如,下面这段程序是设置键盘中断向量 (int 9)的例子:

```
mov ax,0
mov es,ax
mov word   ptr es:24,offset keyboard
                              ;Install   offset of handler
mov word   ptr es:26,seg keyboard
                              ;Install segment of handler
```

图 2.1

在许多情况下上面这段程序都可以正确执行。但是如果上面的程序执行过程中突然击键的话,就可能会出现问题。而最遭的情况是发生在第三个 mov 指令执行完毕而第四个 mov 尚未执行时,键盘中断向量此时没有任何意义,这时的击键将造成整个系统死机。因此

我们在设置中断向量时，必须让中断无效，即屏蔽中断，如：

```
mov ax,0
mov es,ax
cli
mov word   ptr es:24,offset keyboard
                              ;Install   offset of handler
mov word   ptr es:26,seg keyboard
                              ;Install   segment of handler
sti
```

图 2.2

通常上面的作法是能够正确执行的，但是，CLI 这个指令无法屏蔽 NMI 中断(即不可屏蔽中断)，因此如果在程序执行时发生了 NMI 中断就不可预测了。但是 NMI 中断不能够发生在一个完整的指令当中，于是我们就有下面的作法：

```
mov word ptr kbd-ptr[0],offset keyboard
mov word   ptr kbd-ptr[2],seg keyboard
mov di,0
mov es,di                     ;Set es:di to destination
mov di,24                     ; pointer
mov si,offset   kbdptr        ;Set ds:si to source pointer
mov cx,2                      ;Set word count to 2
cld                           ;Set   direction to forward
cli                           ;Disable   interrupts
rep  movsw                    ;Copy the new vector
sti                           ;Enable   interrupts
.
.
kbdptr dd 0
```

图 2.3

上面的作法虽然比较复杂，但是对于所有的中断都有效。这是因为 rep 这个指令是根据寄存器 CX 所设置的次数来重复执行 movsw 指令的，因而整个指令就如同单一的指令一样，不能被 NMI 所中断。

2.3.2 使用 DOS 来设置中断向量

由于安全地设置中断向量需要一定的技巧，因此 DOS 提供了一项特殊的服务来帮助程序设计人员安全地获取和设置中断向量。这就是中断 INT 21H 中的功能 35H(获取中断向量)和 25H(设置中断向量)。这两个中断调用将根据你所提供的中断向量号码来获取和设置中断向量。以下就是获取并设置键盘中断向量 INT 9 的例子：

```
old_keyboard_vec dd 0
```

```
        .
        .
        mov al,9
        mov al,35H
        int  21H
        mov old_keyboard_vec,bx          ;es:bx of interrupts handler
        mov old_keyboard_vec[2],es
        mov bx,cs
        mov ds,bx                        ;ds:dx of new interrupts address
        mov dx,offset  new_keyboard_vec
        mov al,9
        mov ah,25H
        int  21H
        .
        .
new_keyboard_vec proc far
        ...
        iret
new_keyboard_vec endp
```

图 2.4

上面的程序主要是以新的键盘中断处理程序来取代原来系统中的键盘中断处理程序，同时保留原来系统中的键盘中断处理程序，以便以后随时可以恢复。

2.3.3 使用C来设置中断向量

使用C语言来获取和设置中断向量不是没有办法，而是方法很多。通常我们用到的主要有 int86()，int86x()，intdos()等函数及嵌入式汇编等方式，这里就不一一列举了。在下一节及以后的章节中读者将会看到很多这样的实例。

2.4 系统功能的调用

2.4.1 intdos()，int86()与 int86x()函数

这三个函数是C语言中系统功能调用最常使用的。本书中很多地方都使用它们调用系统功能。虽说使用嵌入式汇编也非常有效，但我们仍坚持使用各种C语言编译器都支持的上述三个函数调用方式，以加强程序代码的可移植性。

2.4.2 DOS.H 与 BIOS.H

在 Borland C++编译器中，系统资源的支持者主要通过 DOS.H 和 BIOS.H 体现出来。在头文件 BIOS.H 中定义了包括磁盘 I/O、串并口、键盘管理、打印机和屏幕驱动等等一系列函数和宏定义。图 2.5 定义了常用的 BIOS 汇编级功能调用数据定义代码。至于C语言的

相应定义读者可详细阅读 BIOS. H。

```
; FILENAME: BIOS. INC
;
; Description: This include file contains symbolic equates representing
; the BIOS function calls and their associated services.
VIDEO_SERVICE                   =   010h   ; BIOS Int 10h Video Service interrupt
INT10_SET_MODE                  =   000h   ; set video mode
INT10_SET_CURSOR_SHAPE          =   001h   ; set cursor shape
INT10_SET_CURSOR_POS            =   002h   ; set cursor position
INT10_READ_CURSOR               =   003h   ; get cursor position service
INT10_READ_LIGHT_PEN            =   004h   ; read light pen position
INT10_SELECT_DISPLAY_PAGE       =   005h   ; select display page
INT10_SCROLL_UP                 =   006h   ; scroll window up
INT10_SCROLL_DOWN               =   007h   ; scroll window down
INT10_READ_ATTR_CHAR            =   008h   ; read attribute\char
INT10_WRITE_ATTR_CHAR           =   009h   ; write attribute\char
INT10_WRITE_CHAR                =   00Ah   ; write character
INT10_SET_COLOR_PALETTE         =   00Bh   ; set color palette
INT10_WRITE_PIXEL               =   00Ch   ; write graphics pixel
INT10_READ_PIXEL                =   00Dh   ; read graphics pixel
INT10_WRITE_TTY                 =   00Eh   ; write text in tty mode
INT10_GET_MODE                  =   00Fh   ; Get video display mode
INT10_SET_PALETTE_REGS          =   010h   ; set palette registers
INT10_FONT_SIZE                 =   011h   ; determine the # of rows
INT10_WRITE_STRING              =   013h   ; write string
INT10_GET_VIDEO_BUFFER          =   0FEh   ; get cideo buffer
INT10_UPDATE_VIDEO_BUFFER       =   0FFh   ; update video buffer

FLOPPY_SERVICE                  =   013h   ; Floppy disk service
INT13_RESET_FLOPPY_DISK         =   000h   ; Reset the floppy disk controller
INT13_GET_SYSTEM_STATUS         =   001h   ; Get the status of the floppy disk controller
INT13_READ_FLOPPY_DISK          =   002h   ; Read from the floppy disk
INT13_WRITE_FLOPPY_DISK         =   003h   ; Write to the floppy disk
INT13_VERIFY_DISK_SECTORS       =   004h   ; Verify sectors on the disk
INT13_FORMAT_TRACK              =   005h   ; Format a track on the floppy

SERIAL_PORT_SERVICE             =   014h   ; BIOS serial port service
INT14_INITIALIZE_PORT           =   000h   ; Initialize the COM port
INT14_WRITE_CHAR                =   001h   ; Write a character to the COM port
INT14_READ_CHAR                 =   002h   ; Read a character from the COM port
INT14_STATUS                    =   003h   ; Get the COM port status

KEYBOARD_SERVICE                =   016h   ; BIOS Keyboard interrupt
INT16_READ_CHAR                 =   000h   ; Read the next character
```

```
INT16_KBD_STATUS              =   001h   ; Returns the keyboard status
INT16_KBD_FLAGS               =   002h   ; Returns the keyboard flags

PRINTER_SERVICE               =   017h   ; BIOS Printer interrupt
INT17_WRITE_CHAR              =   000h   ; Write a character to the printer
INT17_INIT_PRINTER_PORT       =   001h   ; Initialize the printer port
INT17_PRINTER_STATUS          =   002h   ; Return the printer status

INT22_TERMINATE               =   022h   ; Vector to terminate routine
INT23_CTRLC_HANDLER           =   023h   ; Vector to CTRL-C handler
INT24_CRITICAL_ERROR          =   024h   ; Vector to Critical Error handler
INT25_ABSOLUTE_READ           =   025h   ; Do an absolute disk read
INT25_ABSOLUTE_WRITE          =   026h   ; Do an absolute disk write
INT27_KEEP                    =   027h   ; Terminate and stay resident
INT2F_PRINT_SPOOLER           =   02Fh   ; Control the DOS print spooler

LIM_SERVICE                   =   067h   ; Lotus/Intel/Microsoft EMS service
INT67_GET_MANAGER_STATUS      =   001h   ; Test if hardware works
INT67_GET_PAGE_FRAME_SEG      =   002h   ; Get the segment address of the
                                         ; page frame
INT67_GET_PAGE_COUNT          =   003h   ; Get the number of pages
INT67_ALLOCATE_MEMORY         =   004h   ; Get a handle and allocate soem memory
INT67_MAP_MEMORY              =   005h   ; Map a page of memory into the page frame
INT67_RELEADE_MEMORY          =   006h   ; Release a handle and its associated memory
INT67_GET_VERSION             =   007h   ; Get the version number of the manager
INT67_SAVE_CONTEXT            =   008h   ; Save the mapping context
INT67_RESTORE_CONTEXT         =   009h   ; Restore the mapping context
INT67_GET_HANDLE_COUNT        =   00Ch   ; Get the number of handles
INT67_GET_PAGES_FOR_HANDLE    =   00Dh   ; Get the numebr of pages that belong
                                         ; to a handle
INT67_GET_PAGES_FOR_HANDLES   =   00Eh   ; Get the numebr of pages that belong
                                         ; to each handle
INT67_GET_SET_PAGE_MAP        =   00Fh   ; Get or set the settings of the page
                                         ;  mapping hardware
```

图 2.5 常用的 BIOS 汇编级功能调用数据定义

在头文件 DOS.H 中定义了使用各种系统资源的函数和宏，同样也包括磁盘文件管理、串并口、键盘管理、打印机、屏幕驱动等等一系列 DOS 提供的更强的函数功能调用。读者可详细阅读 DOS.H 以便深刻了解这些函数的广泛用途。图 2.6 定义了常用的 DOS 汇编级功能调用数据定义代码。