声霸原理与应用

The Sound Blaster Book

Josha Munnik
Eric Oostendorp

敬万钧　袁宏春　译
刘锦德　校

SYBEX
電子工業出版社

SYBEX

声霸原理与应用

近期推出

SYBEX公司最新图书中译本书目

- Windows 3.1 使用大全
- Windows 高级程序开发工具
- Windows 环境下的 Turbo Pascal 程序员指南
- Windows 排错必读
- Windows 疑难详解
- DOS 6 从入门到精通
- FoxPro 2.5 从入门到精通
- AutoCAD 12 使用大全
- 精通 Norton Utilities 6.0
- 硬盘长寿指南
- 计算机初学者指南
- 掌握Microsoft Visual C++编程
- 精通Windows NT 编程技术
- PC 维修大全
- Windows、Word和Excel的简明指南
- 跟我学 Visual Basic和程序设计
- Windows 应用程序集粹
- 妙语话DOS
- 语声化PC机

以下为英文版

- Windows NT速查手册
- 最新袖珍计算机词典
- FoxPro 2.5速查手册

北京美迪亚电子信息有限公司

北京市海淀区万寿路甲15号院南小楼三层
电话：821.2166，821.2988
邮编：100036

书号：ISBN 7-5053-2710-0/TP・852
定价：46.00元

The Sound Blaster Book

Josha Munnik Eric Oostendorp

声霸——原理与应用

敬万钧 袁宏春 译

刘锦德校

電子工業出版社

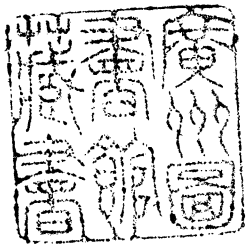
The Sound Blaster Book

Josha Munnik　Eric Oostendorp

声霸——原理与应用

敬万钧　袁宏春　译

刘锦德校



SYBEX

電子工業出版社

（京）新登字055号

内 容 提 要

本书为声霸卡的专籍，是为声霸卡的应用人员而写的内容，其包括了声霸卡的硬件、软件基础及应用编程。全书由两大部分组成，共分十一章。分别介绍了声霸卡的硬、软件基础，用声霸卡作曲，声霸卡的括充设备，芯比的编程，数字声霸卡处理器的编程，MIDI编程等等。本书还包含七个附录，详细介绍了声霸16。全书理论结合实际，深入浅出，是声霸卡使用者的必备工具，也是教学、开发、维护及设计工作的参考资料。

声霸——原理与应用

敬万钧　袁宏春　　译
刘锦德　校
责任编辑　　子　杉

电子工业出版社出版（北京市万寿路）
电子工业出版社发行　　各地新华书店经销
北京市顺义县天竺颖华印刷厂　印刷
开本 787×1092 毫米　1/16　印张：23　字数：542 千字
1995 年 2 月第 1 版　　1995 年 2 月第 1 次印刷
印数：5000 册　　定价：46.00 元
ISBN 7-5053-2710-0/TP・652

出 版 说 明

计算机科学技术日新月异。为了引进国外最新计算机技术，提高我国计算机应用与开发的水平，中国电子工业出版社与美国万国图文有限公司合资兴办的北京美迪亚电子信息有限公司取得了美国SYBEX公司的独家版权代理。SYBEX公司授权本公司通过电子工业出版社等出版机构全权负责在中国大陆出版该公司的中文版和英文版图书。现在与广大读者见面的是最近推出的第一批图书。今后我们还将陆续推出SYBEX公司的最新计算机图书和软件，为广大读者提供更好的服务，传递更多的信息。

美国SYBEX公司是世界著名的计算机图书出版商，该公司自1976年创办开始，其宗旨就是通过出版有效的、高质量的图书向计算机用户介绍实用技巧。我们优选翻译出版的图书是SYBEX公司的最新计算机图书，并采用了该公司提供的电子排版文件，从而提高质量并大大缩短了图书的出版时间，从根本上改变了以往翻译版图书要落后原版书较长的“时差”现象，这在电子技术日新月异的时代具有深远意义。

北京美迪亚电子信息有限公司

1994年11月

致　谢

首先，对我们在Ultra Force Development的同事——Arjan Brusse，Miche Hooymans，Eric Soonius和Remco de Berk表示感谢，感谢他们的批评指导、洞察力和对我们工作的支持！

还要感谢我们的父母，在写这本书的过程中，他们给了许多有益的帮助。

我们感谢Walop Electronics B.V.在我们的工作当中，提供了必要的硬件，特别是Ferry ten Brink，给了我们许多帮助，我们感谢每一位为此书出过力和给予过帮助的人。特别是弗兰克·万·托勒（Frank van Tol）以及Black Pearl Music的雇员。

最后，我们感谢SYBEX和印刷厂的全体人员，没有他们，这本书也不可能问世，还要特别感谢Tim Tnlly为声霸16一节编写了有关材料。

目　录

绪　言

长时间以来，机内扬声器是PC机的唯一声音源。不幸的是，机内扬声器所能作的节目只是由一些简单的短促声所组成。直到1987年，声音适配卡开始出现在市场上，PC机用户才有了机会来听取和产生真实的声音。

在1987年，AdLib音乐合成卡问世，很快它就成了计算机游戏的标准卡。除了图形能力之外，声音效果和音乐也成了计算机游戏的一部份，这得归功于这一个精心制作的新卡。

1989年，Creative Labs 的Sound Blaster第一版问市。由于AdLib是建立在合成器芯片之上，这限制了它的声音能力；而Sound Blaster对产生声音提供了更强的能力。使用Sound Blaster，你可以数字化地记录和播放声音。感谢Sound Blaster的数字能力，使你可以在程序中使用语言和声音效果。

越来越多的游戏和程序支持Sound Blaster。Sound Blaster已从声音卡市场上取代了AdLib。

Sound Blaster的初版之后接着就出现了Sound Blaster Pro，它增强了立体声能力，并且能和CD-ROM相结合；Sound Blaster MCV，它用于IBM PS/2 50 型和更高型；Sound Blaster 16，这是Sound Blaster卡的最新和最先进的版本。Microsoft已选择Sound Blaster Pro作为它的标准多媒体声音卡。

注解　本书详细地介绍了Sound Blaster和Sound Blaster Pro。由于在写本书时，Sound Blaster 16刚发布，因此，有关它的讨论放在附录A。关于SoundBlast MCV，因为它只为PS/2而设计，这是一种失策，由于使用它的人很少，所以本书未包含它的内容。

从本书你会学到什么？

本书是为想要开发Sound Blaster卡的创造潜力的用户而写的，它的内容极为广泛，包括从安装的基本知识，到定时器芯片、FM芯片和CMS芯片的编程。

全书由两部份组成："Sound Blaster硬件和软件基础"和"编程Sound Blaster"，第二部份比第一部份长得多。

以下逐章地介绍在**第一部份**会见到的内容：

- 第一章，"安装Sound Blaster硬件和软件"，说明如何安装你的Sound Blaster卡，如何测试它，以及使它兼容于你的系统。本章也简要地介绍了随卡带来的　软件。
- 第二章，"Sound Blaster卡一览"，详细地介绍形成Sound Blaster卡的5个扩充卡-CMS游戏卡，AdLib音乐合成器卡，数字声音处理（DSP）卡，　MIDI卡以及游戏杆卡。
- 第三章，"用Sound Blaster制作音乐"　说明了用Sound Blaster作曲的某些最重要的方面。本章也介绍一些创作音乐的程序。
- 第四章，"Sound Blaster扩充件"，讨论了你可用的各种扩充件。包括MIDI连接盒、CMS芯片、以及CD-ROM驱动器。

- 第五章，“了解MIDI接口”，说明什么是MIDI，并仔细地介绍了连接MIDI设备、MIDI设备本身、以及MIDI软件。

第二部份为：

- 第六章，“编程定时器芯片”，说明了Sound Blaster的定时器芯片的功能、定时器通道、以及定时器端口和计数器。
- 第七章，“编程FM芯片”，本章对AdLib用户特别有用，因为它不仅揭示了CMF格式，而且也揭示了ROL格式。加之它叙述了如何编程FM芯片使它产生声音，因此，也就说明了AdLib如何产生声音。
- 第八章，“编程CMS芯片”，讨论了编程CMS芯片以及使用CMS驱动程序。本章研究了CMS芯片功能、如何用CMS芯片产生声音和噪音，以及如何编程音调、音符和音符持续时间。
- 第九章，“编程数字声音处理器”，介绍DSP。讨论了Sound Blaster Pro的立体声处理器，包含了样本文件如何构成的内容，并描述了CT-VOICE 驱动程序。本章还叙述了不使用驱动程序如何记录和播放样本的问题。
- 第十章，“用MIDI编程”，综合性地讨论了MIDI语言，以及实际的MIDI文件格式。它说明了信息如何通过Sound Blaster传送以及从MIDI端口输出。它也说明了如何把MIDI信息送到外部合成器模块。
- 第十一章，“Sound Blaster Pro的混合器芯片”，说明了如何改变Sound Blaster Pro的音量设置，以及如何设置Sound Blaster Pro各个部份（包括Line In（线入），Mic In（麦克风输入）以及CD-ROM In （CD-ROM输入））的输入和输出等级。

本书还含有七个附录：

- 附录A，“Sound Blaster 16的硬件和软件”说明了如何安装Sound Blaster 16卡，以及如何使用所附加的软件。本附录还介绍了SB16的某些增强，包括Wave Blaster和Creative WaveStudio。
- 附录B，“DMA、IRQ以及I/O地址”，说明了DMA通道、IRQ号和I/O地址的意义，也说明了它们与Sound Blaster如何相互作用。
- 附录C，“Sound Blaster端口地址”，列出了控制Sound Blaster各个部份的寄存器的端口地址。
- 附录D，“混合器芯片寄存器”列出了控制混合器芯片操作的各寄存器。
- 附录E，“DSP命令”，讨论了所有的DSP命令。
- 附录F，“MIDI设备制造厂商的ID码”，列举了各MIDI设备制造厂的ID码。这些码子被用在系统专有的信息中。
- 附录G，“MIDI的状态和数据字节”，列出了各种MIDI信息的字节值。

Sound Blaster的每个部份都有其自己独特的性能和设置，这使得有可能将声音卡的每一部份处理为一独立的章节。虽然各部份之间是独立的，但这并不意味着你不能组合它们。例如，通过使用组合FM芯片和DSP的办法，你可以采用采样、声音、低音鼓、或者利用以上三者来充实丰富FM音乐。进而，你也可以使用CMS芯片以增加立体声效果。

关于程序清单的说明

在本书的各章，特别是6到11章，含有一些程序清单。程序清单用Pascal， C，以及汇编语言写成。汇编程序通常是作为C和Pascal程序的补充。有些时候举例用汇编提供，其理由很简单，是因为汇编语言有较高的速度。

如果你不是C或Pascal的行家，也不必着急；如果你不是汇编的专家也完全不必担心，因为绝大多数例子是以独立的库或程序单元的形式出现。库是由许多附以简要说明的函数所组成。对这些库，你可以使用它的各个函数，而不必知道它们如何建立以及它们如何工作。

除了库之外，本书也提供了使用某库函数的简单示例程序，并且说明了你自己能如何使用这些函数。

各个程序使用的是Turbo C，Turbo Pascal和Turbo Assembler，其原因非常简单，因为我们（作者）熟悉这些编程环境。偶而我们利用了这些环境所提供的便利点（包括在Pascal程序单元中）。但是，一个普通的程序员都会发现，将这些程序转换成自己的Pascal或C编译程序是不困难的。所有的程序都是相当容易并且是直观的，没有采用复杂的技术或技巧。

使用程序清单的提示

在你能使用程序之前，你必须把程序清单送入该编译器环境。

以下是键入这些清单时，需要遵守的某些有用的提示：

- 不要键入注解，这可以节省你一半的键入时间。除非你为了说明的目的，才键入注解。
- 在Turbo Pascal中，你可以都使用小写字母送入程序，以节省时间，因为Turbo Pascal是不分大、小写字母的。但是在你都使用小写字母时，你的程序清单会难于理解一些。
- C是要区分大、小写字母的。在C中，函数和变量是小写字母，而宏名用大写字母。在C中，当你键入程序代码时，就必须考虑此种情况。

在本书中的习惯用法

写本书的目的是为了帮助读者能快捷而容易地学习并参考Sound Blaster编程的技术。为达此目的，我们采取了下列作法：

- 包含了很多的标题头，以便你能很快地查找信息。
- 包含了大量的表格，每个表格都提供了关于Sound Blaster的简明信息。
- 在需要帮助你学习使用Sound Blaster、它的扩充件、以及其软件程序的地方，都包含了说明。

为你的方便，我们在本书中也包含了注解、提示和警告。

注解　注解将告诉你，要了解有关讨论题目进一步的信息到本书何处去找。

提示　提示让你深入了解Sound Blaster的使用。

警告 **警告是告诉你，此时你必须要作关键性的决定或选择。（例如说）后者可能会影响到程序运行的好坏，或者影响插卡产生声音的质量。**

第一部份　Sound Blaster硬件和软件基础

第一章　安装Sound Blaster硬件和软件

本章内容为：

为你的系统选用地址、端口和连接器设置
安装Sound Blaster卡到你的计算机
测试Sound Blaster，看其构成是否恰当
安装Sound Blaster软件

本章介绍如何在你的PC机上安装Sound Blaster卡和Sound Blaster软件,告诉你如何测试卡,以确认你已作了正确的安装,说明如何使此卡兼容于你的系统。本章还简要地说明了同Sound Blaster一起所附加的软件，使你了解Sound Blaster是如何工作的。

安装Sound Blaster需要进行三步：正确选择卡的设置、安装卡到PC机、以及测试Sound Blaster。我们首先说明选择正确的设置。

注解　对于Sound Blaster和Sound Blaster Pro安装指南所不同的地方,我们将对其加以说明，除了特别指出外，Sound Blaster和Sound Blaster Pro的安装技术是相同的。

注解　在写本书时，Creative Labs介绍了它的最新产品——Sound Blaster 16。有关安装Sound Blaster 16卡和Sound Blaster 16硬件的内容见附录A。

为你的系统选择跳接器设置

为进行卡的设置，需使用跳接器。跳接器是一种小的塑料－金属块，用来连接卡上成对的插针。用选择跳接器的办法，构成在你系统上能工作的Sound Blaster。

对Sound Blaster的所有设置都使用跳接器在卡本身上进行。你不能用软件改变设置，所以在安装Sound Blaster之前，你必须考虑你的系统如何构造。如果你所作的设置同你的机器的其他卡相冲突，在你把Sound Blaster安装在机器上之前，你是发现不了的。在发现冲突时，你必须取出卡，并改变设置。本章的以下部份说明如何进行跳接器设置，以及选择或不选择哪个设置,以使Sound Blaster在你的系统上工作。

如何改变跳接器设置

在你选用设置之前，你需要知道跳接器是如何工作的，因为所有的设置都是用跳接器来作的。

图1.1示出了三种跳接器设置——选用，未选用,以及未选用但此时是把跳接块套在一根插针上以保存它。

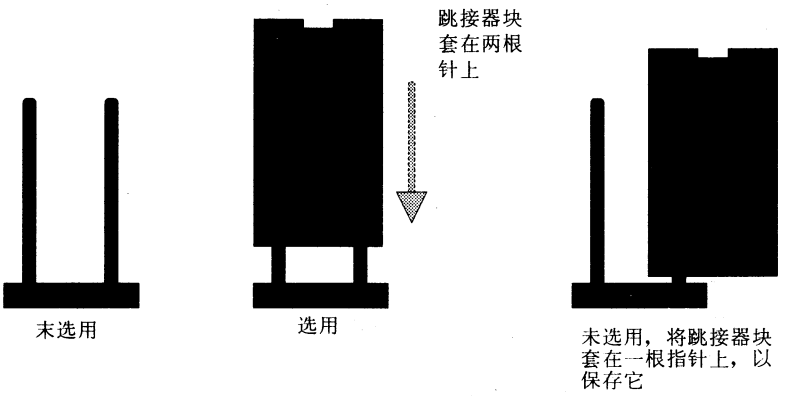


图1.1 跳接器设置。当塑料块套于两根插针上时，该跳接器被选用

- 当其塑料和金属块不连接两插针时，跳接器就未选用。当该跳接器未选用时，你的卡就未用相应的设置。
- 当塑料-金属块套在两插针上时，跳接器就被选用了。实际上，当跳接器被设置为此种方式时，上述两根插针就“连通”而起作用。当接通时，该设置就为你的系统所使用。

提示 为保存跳接器块以备你以后需要时使用，把跳接器块套于一根插针上，而不是套在两根上。此种作法，是让该设置不被选用,但是，你还是有跳接器块，以备以后你你需要时使用。

构造Sound Blaster的跳接器设置

图1.2示出在Sound Blaster卡上的跳接器设置。设置有四组，它们为：

- I/O地址
- IRQ号
- 游戏杆
- DMA设置

你可以对每一组选择一种设置。

注解 IRQ、DMA、和I/O地址设置在附录B中详细说明。

图1.3给出Sound Blaster Pro卡。对Sound Blaster Pro卡,IRQ设置借助于DACK和DRQ跳接器。卡上有一个Speaker Input(扬声器输入)跳接器、一个Remote Speaker Enable (遥控扬声器允许)跳接器以及一个CD-ROM音频输入跳接器。让我们逐一来看各个设置。

I/O地址跳接器

I/O(输入/输出)地址跳接器建立Sound Blaster卡的基本端口，同Sound Blaster 卡的所有通讯皆通过此地址进行。缺省端口地址为220H，但如果此地址已为其他的卡所使用，你就必须为Sound Blaster选用另外的端口。

I/O地址端口值必须在210H和260H之间。

Sound Blaster Pro I/O地址 对Sound Blaster Pro，如果你不能使用缺省端口地址220H的话，则只能选择240H作为替换端口。但是，在你决定使用240H端口地址之前，我们建议你先改变另外卡的端口一试。因为Sound Blaster的有些软件是对220H端口来写的，它们不允许你改变端口地址。大多数设备是不使用220H端口的，所以，你多半不必改为240H。

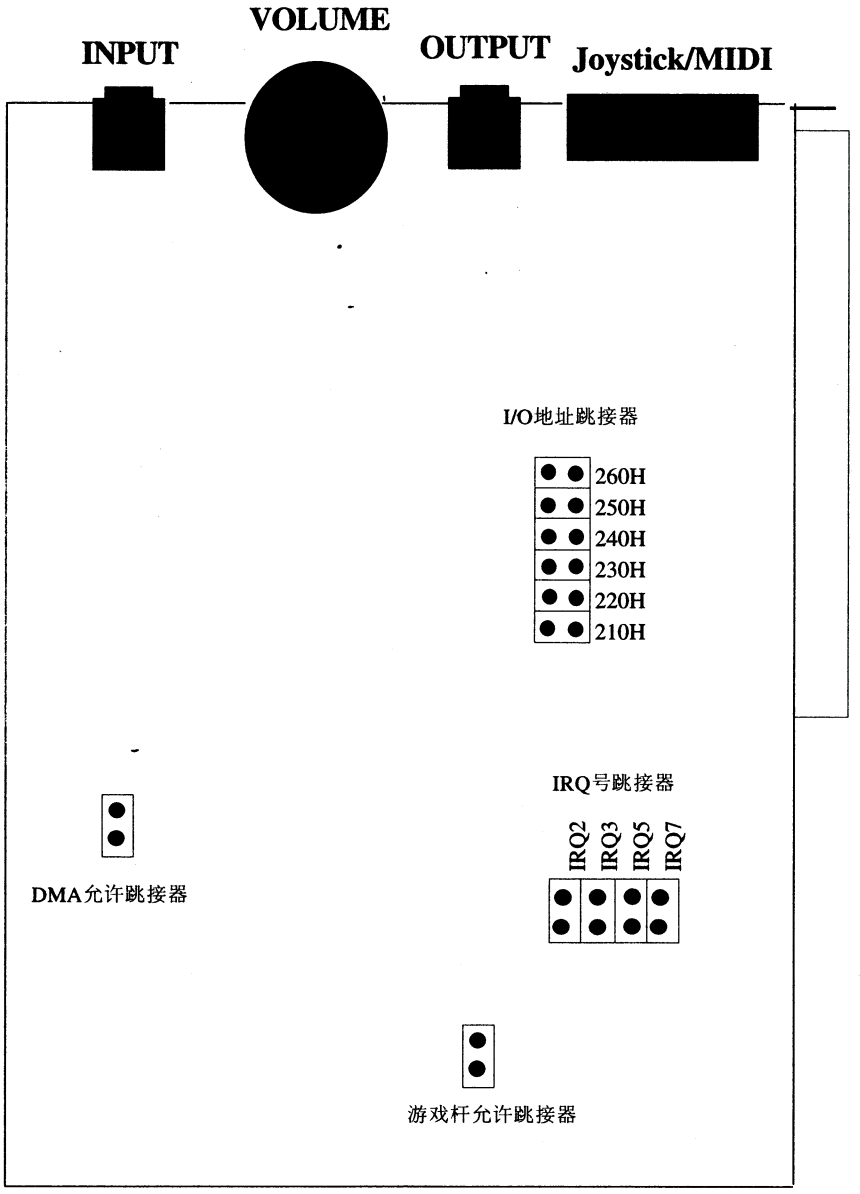


图1.2 Sound Blaster卡的设置

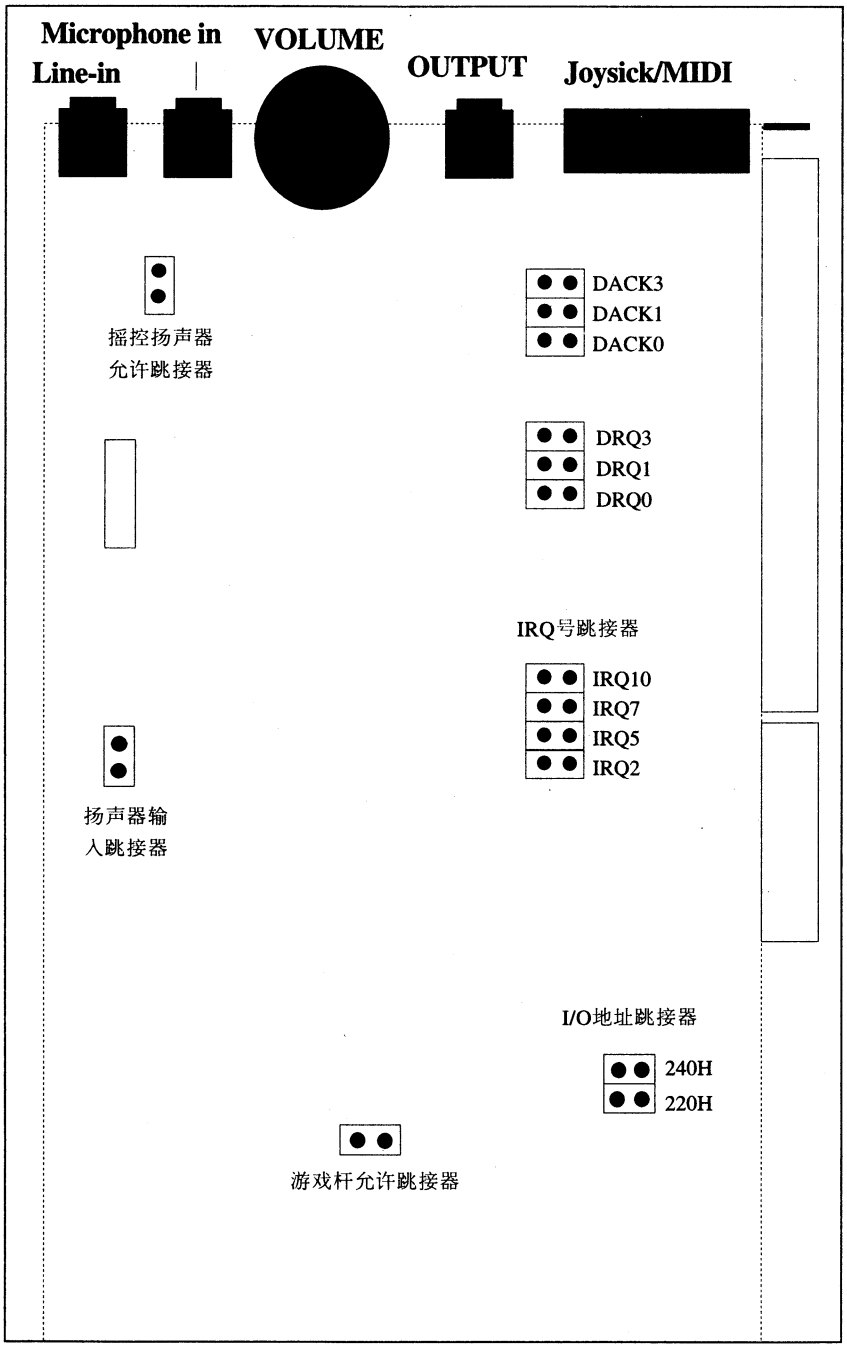


图1.3 Sound Blaster Pro卡的设置

中断（IRQ）跳接器

中断(IRQ)跳接器为Sound Blaster卡设置硬件中断号。IRQ代表中断请求。另外的设备可

能已在使用某个IRQ号，所以你必须小心地为Sound Blaster卡选择中断号。缺省号为IRQ7。

表1.1说明了各可用的中断号。Sound Blaster和Sound Blaster Pro两者都可以使用2、5、和7，但是，只有Sound Blaster也可以使用3，只有Sound Blaster Pro可以使用10。

如果你同时使用打印机和声霸卡，则你应该选择除缺省值7之外的其他设置。如果你用的是PS/2，使用7之外的其他设置。

提示　尽量避免使用IRQ10，10是新的中断号，老的Sound Blaster软件不支持它。

DMA Enable（DMA允许）跳接器

Sound Blaster用直接存贮器访问(DMA)通道作为声音输入和输出。此跳接器在Sound Blaster上才有，Sound Blaster Pro不具有DMA Enable跳接器，而是缺省使用DMA。

表1.1 Sound Blaster IRQ号

IRQ号	说明
2	供AT使用于菊花链。
3	供COM2使用(若此端口存在并使用的话)。*
5	供某些XT用于硬盘以及并行口。
7	缺省值，供并行口使用。在Tandy 1000上，此号为内部使用，所以不能选用。
10	待用。**

*　只供Sound Blaster，而不供Sound Blaster Pro所用。

**　只供Sound Blaster Pro所用。

PC-XT DMA通道　PC-XT有四个DMA通道：

0	用于存贮器刷新
1	留待其他卡选用
2	供软盘驱动器用
3	供硬盘使用

PC-AT和PS/2 DMA通道　PC-AT和PS/2有8个DMA通道：

0	留给其他卡选用
1	留给其他卡选用
2	供软盘驱动器使用
3	供硬盘使用
4	用作使用通道5，6，7的通路。
5	留给其他卡选用
6	留给其他卡选用
7	留给其他卡选用

注解　DMA Enable跳接器只在Sound Blaster才用，Sound Blaster Pro不用。

开通DMA跳接器缩短了声音输入、输出所需要的处理时间。如果DMA通道已为另外的卡长期使用，你可以去掉跳接器来断开DMA通道。

但是，要记住，只是在有实际的声音输入和输出时，Sound Blaster才使用DMA通道;其余时间，通道是自由的，可由其他卡使用。所以，如果你的设备不是频繁地使用DMA通道时，你还是可以开通DMA跳接器的。

如果你觉得开通DMA跳接器可能会与你的其他卡相冲突，可尝试改变其他卡上的DMA通道。但是，因为多数软件都用DMA通道产生声音输入和输出，我们建议，为了你能使用这些软件，不要去掉DMA跳接器。多数设备使用DMA通道有别于Sound Blaster所使用，所以实际上你不必移去此跳接器。

Joystick Enable（游戏杆允许）跳接器

所有的Sound Blaster卡都有一个游戏杆连接器。如果你已有一个游戏杆接在机器上，就去掉Sound Blaster的Joystick Enable跳接器。因为两个游戏杆会相互冲突，你必须断开它们中的一个。

注解　不连接Joystick Enable跳接器并不影响MIDI的连接。在断开游戏杆端口后，你还可以使用MIDI设备。

Sound Blaster Pro的设置

Sound Blaster Pro多了三个设置，它们是：DACK和DRQ设置、Remote Speaker Enable（遥控扬声器允许）（RSPKEN）跳接器、以及Speaker Input（扬声器输入）跳接器。

DACK和DRQ设置　对Sound Blaster Pro，DMA缺省为DMA1 (DACK1和DRQ1)。DMA为自动地开通。但是，要记住，只是在有实际的声音输入或输出时，Sound Blaster才使用DMA通道。两个设备可以共用一个DMA通道，只要它们不在精确的同一时刻使用它就行。

如果你觉得使用DMA选择跳接器会同你的其他卡相冲突,请尝试改变其他卡的DMA通道。因为为Sound Blaster写的大多数软件都是用DMA通道1来产生声音输入和输出的，所以我们极力建议不要改变其DMA跳接器的设置。如果DMA通道之一要为其他的设备长期使用，则就让其使用通道0。

如果你要改变DMA设置，则必须调整DMA请求线（DRQ）和DMA应答线（DACK）的设置（见图1.3）。对通道0选择DRQ0和DCK0。你必须选用这些DACK和DRQ的设置。

Speaker Input（扬声器输入）跳接器　你可以将你的PC机母板上的扬声器的输出连接到Sound Blaster卡上的扬声器输入跳接器。如果你这样作了，则你的PC机的所有声音将通过Sound Blaster来放音，这意味着你可以调节你的PC机所发声音的音量。但是，作此种连接要求焊接，这或许应由专业者来作。扬声器的地线连到跳接器的右插针，而5V电压连到跳接器的左插针。

Remote Speaker Enable（遥控扬声器允许）跳接器　如果将你的PC机母板的扬声器输出连到Sound Blaster的扬声器输入 产生了问题的话，你可以用遥控扬声器允许跳接器来断掉它。

安装Sound Blaster卡

当你已设置好卡后，就可以安装它了，其步骤如下：

1. 关掉PC电源，但是不去掉电源电缆——因为你需要将PC保持接地。
2. 卸去机壳螺钉。如果做此事需要帮助的话，请参阅你的计算机用户手册。
3. 慢慢地移去空插槽的盖子，在你移去盖子时，不要碰到任何电缆。
4. 连接内部扬声器到Sound Blaster Pro (如果你选用的话)。
5. 把卡插入空槽中，并上好顶端螺丝。Sound Blaster Pro是16位的卡，所以只能安装在PC-AT的16位槽。图1.4示出了8位和16位槽的区别。Sound Blaster可以插入8位或16位槽。
6. 盖好计算机的盖子

测试Sound Blaster卡

现在卡已构造完毕并安装好，于是可以对其进行测试了。为了测试卡，你必须连接它到输入和输出设备，然后，运行测试程序，以确证卡已正确安装并工作于你的系统。

连接输入和输出设备 你必须作的第一件事是连接一输入设备。Sound Blaster只有一个音频输入插口(见图1.2)，但是Sound Blaster Pro有两个(见图1.3)。你可以接入放大器、扬声器或耳机作为输出设备。

卡的最大功率为：

- 如果你使用4欧扬声器，每通道为4瓦。
- 如果你使用8欧扬声器，则每通道为2瓦

提示 如果你没有麦克风，可试接放大器于线入或麦克风连接器。这种声音质量虽然不是很高，但对测试来说，还是足够高的。把音量控制调节到中间位置。

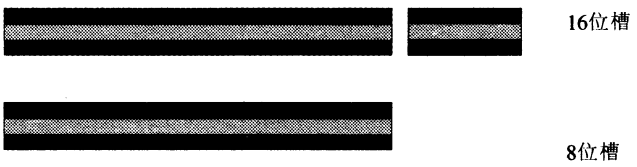


图1.4 8位和16位槽

运行测试程序

运行测试程序，考查Sound Blaster是否已正确安装：

1. 接通计算机电源。
2. 把1#盘放入一个驱动器中，并转到盘的根目录。

3. 根据你用的是Sound Blaster或Sound Blaster Pro（假定你使用的是A驱动器）

对于Sound Blaster卡，键入：

A： TEST-SBC

对Sound Blaster Pro卡，则键入：

A：TEST-SBP

测试程序会告诉你是否有硬件冲突，如果有的话，它们是什么冲突。

提示：记下测试程序在哪里发现冲突，以便你以后作端口地址和连接器的改变。

在测试过程中，如果你的程序死了,尝试用 /M选项来运行测试程序(你只能对Sound Blaster Pro这样作)。

1. 在DOS提示下，键入：

TEST-SBP /M

2. 会出现一系列的提示，于是你可以告诉程序，你是如何设置你的卡--即I/O地址、DMA通道和中断的。

若冲突出现..... 如果测试程序发现I/O地址、中断、或DMA冲突，你要小心地取出卡，并改变设置。应遵循在本章的前些部份关于改变设置的指南去做。为改变跳接器设置，你必须关掉PC电源从计算机取出卡，进行设置更改，重新装卡，再运行一次测试程序。

安装Sound Blaster软件

Sound Blaster软件以压缩文件格式存贮。Sound Blaster包括一个INST-HD(安装硬盘)程序，用来解压软件并将其拷贝到你的硬盘。INST-HD在1#盘上，同TEST-SBC在同一张盘。为运行Sound Blaster安装程序，你必须使用DOS命令，因为程序是在DOS下运行。

警告 在你安装软件之前，作一份盘的备份拷贝。使用拷贝盘代替原盘来进行软件安装。

为安装Sound Blaster软件：

1. 插1#盘在A驱动器或B驱动器
2. 键入：

INST-HD D:

并回车，此处D是你的硬盘驱动盘(如果你的硬盘是C，则键入C)。

各文件将被拷贝到新的子目录\SB(如果你用Sound Blaster Pro则为\SBPRO)。\SB目录包含一些子目录，它们是Sound Blaster的驱动程序和服务程序。驱动程序是一些存在于各自独立的文件中的程序，它们被用来控制声音卡与软件之间如何进行通讯。

修改AUTOEXEC.BAT文件

INST-HD程序对AUTOEXEC.BAT作了一些改变。特别是，它加了两行SET SOUND和SET-BLASTER：

- SET SOUND 指向你刚才用INST-HD安装软件的目录。如果你要改变目录名，你就必须相应地改变此条语句。
- SET BLASTER 指向Sound Blaster卡的设置。其参量和它们的功能如下：

SET BLASTER=Annn In Dn Tn

Annn	端口地址。例如，A220表示220H端口。
In	IRQ号。例如，I10表示IRQ号10。
Dn	DMA号。例如，D1表示通道1。
Tn	Sound Blaster类型，T1表示标准的Sound Blaster，T2表示Sound Blaster Pro。

安装驱动程序 所包含的软件使用了不同的驱动程序。如果卡被修改，你必须使用INST-DRV（安装驱动程序）程序来调整驱动程序。INST-DRV程序识别哪个驱动程序被装在目录中，而且它告诉你哪些能够改变。

键入：

INST-DRV/?

列出INST-DRV能修改的所有的驱动程序的清单。

键入：

INST-DRV

改变在现行目录中的驱动程序。

键入：

INST-DRV name

改变在子目录name中的一个或多个驱动程序。

子目录VOXKIT和FMORGAN两者都包含Sound Blaster的驱动程序。子目录DRV包含已被修改过的Sound Blaster Pro的驱动程序。最后，CMS驱动程序（CMSDRV.COM）可以改变,这是一个以标准的Sound Blaster的CMS芯片为目标的驱动程序。

Sound Blaster程序浏览

为帮助你熟悉Sound Blaster，下面是某些软件程序的简短叙述。这些程序可从\SB或\SBPRO目录启动。在Sound Blaster情况下，每个程序也能从软盘启动。

鹦鹉学舌

如你所料到的那样，学舌的鹦鹉重复你对你系统的麦克风所说的话。如果你没有麦克风，你也可以用你的键盘教鹦鹉学话。

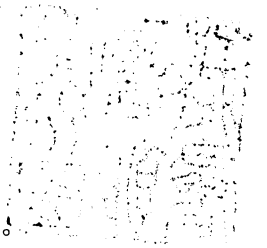
试验一下鹦鹉学舌：

1. 从SB或SBPRO目录键入

PARROT

并按回车键。

一个示波器出现，它显示出多少环境噪音正被你的麦克风所拾取。



2. 对麦克风讲话，看看你必须用多大声音说话。
3. 你被要求送入一个值。选择一值，它大约是在环境噪声之上为10的数。
   一个鹦鹉出现在屏幕上欢迎你。
4. 对麦克风讲话，鹦鹉将重复你所说的。如果你没有麦克风，你可以用敲击键盘上的键来“逗乐”鹦鹉。
5. 按ESC以退出程序。

FM琴

FM琴类似于简单的键盘合成器，使用它你可以演奏、播放一支歌曲，并能把歌曲存在磁盘上。Sound Blaster Pro FM琴具有立体声FM能力。

为了演奏FM琴：

1. 转到SB或SBPRO目录(如需要的话)。
2. 对Sound Blaster键入：

   FMORGAN

   对Sound Blaster Pro则键入：

   PRO-ORG

对如何使用FM琴的完整叙述，见有关文档。而下面则对演奏FM琴的一些有用的键组合：

键	用途
F1	在合成时，提供对所用键的帮助信息。
F2	开始记录新歌曲。
F3	让你播放你刚才已记录的歌曲。
F4	退出程序。
F6	让你装入以前存入的歌曲。
F7	让你把已录入的歌曲存到盘上。

VOXKIT

VOXKIT程序让你采样声音——即是记录并播放声音。此程序以数字形式存贮声音，所以它能被存入你的计算机的存贮器，以后再播放它。

注解 VOXKIT程序是随Sound Blaster附来的，而Sound Blaster Pro却无。但是，Sound Blaster Pro的VEDIT程序包括了VOXKIT的所有功能，它只工作在图形模式。

为了明白VOXKIT如何工作：

1. 转到SB目录(如需要的话)
2. 键入

   VOXKIT

   并按回车。

VOXKIT主菜单提供下列各选项：

记录采样　让你记录采样。当你选择这项时，VOXKIT给出采样的选项，让你选用预选的速率或改变采样率来进行采样。

注解　采样率是一个速率，它是程序记录声音的速度。速率愈高，质量越好，但是高速采样需要更多的存贮器。

播放样本　让你播放你已用麦克风记录的样本。
存贮样本　让你写入样本到盘上。
装入样本　让你装入一个在盘上的样本，该样本是以.VOC文件存贮。
用盘　让你把采样值直接记录在软盘或硬盘上，而不是记录在内存中。用此种方法存贮采样值，可以形成更大（因而更长）的样本。

PLAYCMF

PLAYCMF通过Sound Blaster的11-声部FM部份播放CMF歌曲。CMF代表Creative music file。类似于MIDI文件，CMF文件是Sound Blaster制造商Creative Labs创造出来的声音文件。

为演奏PLAYCMF：

1. 转到SB或SBPRO目录。
2. 键入

   SBFMDRV

   并按回车键以装入SM合成器驱动程序。
3. 键入

   CD PLAYCMF

   并按回车键，转移到PLAYCMF目录。
4. 键入

   PLAYCMF name.CMF

   此处name是你所想听的文件的名字。

注解　PLAYCMF与FMORGAN间的区别是：PLAYCMF使用所有的11个声部，而FMORGAN一次只能演奏一个音符，且所包含的节奏和设备都有限。

DOCTOR SBAITSO

Doctor Sbaitso演示Sound Blaster的声音能力。程序实际上读着你用键盘送入的字，并响应它们。

为了启动Doctor Sbaisto：

1. 转到适当的目录，并键入

   SBAITSO　对Sound Blaster，

   或

   SBAITSO2　对Sound Blaster Pro。

开始，程序要问你的名字，并要你输入一句话。在你输入一句话之后，程序分析它,并给

出一“智能性”的回答。使用Help命令获得有关如何改变设置的信息。使用Exit命令停止程序。

Sound Blaster Pro的附加程序

除了VOXKIT是只能在Sound Blaster找到外，上面所述的各个程序,在Sound Blaster和Sound Blaster Pro都有。但是，下面的各个程序仅工作于Sound Blaster Pro。

VEDIT

VEDIT（或VOC编辑器）程序只包括在Sound Blaster Pro中。使用它，你可以组合你已用VOXKIT记录的各个样本。你也可以记录采样，并以几种不同的办法编辑它们。

为启动VEDIT，转移到SBPRO目录，并键入

VEDIT2

多媒体播放器

Sound Blaster Pro含有一个演示程序，它有显示多媒体扩充（图形和声音结合）能力。图形文件由展示程序所产生，后者能组合图形和正文。多媒体管理器确定哪一个声音同哪个演示程序一起播放。

为试验多媒体演示程序，在SBPRO目录下键入：

MMDEMO

Tetra Compositor演示程序

Tetra Compositor 演示程序通过Sound Blaster播放Noise Tracker音乐。Noise Tracker格式是Amiga的一种流行格式，并用在许多演示和游戏中。音乐是由播放乐器的样本或不同频率的其他声音来产生。由于同时播放不同的样本，并对这些样本加入各种效果，你可以产生各种各样的音乐，从古典音乐到摇滚音乐。在Amiga上，这花费很少的时间，因为Amiga有四个独立的通道用于声音输入和输出。Sound Blaster只包含一个通道，所以播放此种格式要用较多的时间。

为试验Tetra Compositor演示程序，在SBRPRO目录下，键入：

TDEMO

Windows软件

三个Sound Blaster程序能用在MS-Windows 3.0：

程序	用途
SBMIXER	控制Sound Blaster卡上的各不同部份的音量
JUKEBOX MIDI播放器	由卡上的FM部份播放MIDI文件
SETUP	可用来修改缺省设置

WINDOWS目录也包含有SOUND.DLL文件，该文件是这三个程序所需要的。这个文件也为其他的为MS-Windows开发的软件所使用。

第二章　Sound Blaster卡一览

本章内容为：

声波如何工作
CMS和FM立体声芯片
处理数字声音的数字声音处理器
用于硬设备间通讯的MIDI接口
Sound Blaster和Sound Blaster Pro能力的比较

Sound Blaster卡实际上是5个扩充卡的混合物：

- CMS Game Blaster卡，它的立体声芯片可以在Sound Blaster卡上找到。
- AdLib音乐合成器卡，它的FM芯片（AdLib卡的心脏）为Sound Blaster所使用。
- 工作于数字和模拟声音的数字声音处理（DSP）卡。
- 同合成器通讯的MIDI卡亦在其中。
- 玩游戏的游戏杆卡。

本章讨论所有这些功能。还提供了在声音物理方面的基本内容。

声音物理

声音是由称之为声波的振动所产生。当声波反复而规则的产生时，声音就得到确定的音调，正像一个音符那样。当其声波具有不规则的模式时，它就不成为音调，而是产生一不规则的声音，如砰砰声或咯咯声音。

声音的频率由在给定的期间内声波被重复的次数来确定。音调由频率确定，随着频率的升高，音调越高。测量频率的单位称为赫兹，简写为Hz。赫兹是每秒钟的声波数。健康人的耳朵能识别20Hz至20KHz间的声音（约每秒20，000次振动）。

为了更好地理解声音以及它如何工作，见图2.1。此图用正弦曲线表示着声波图。在图中，声音振动中的各个部份被画了出来。例如，该图中可用来表示由长笛所产生的声波。用示波器观察进入麦克风的哨声，你就会看见类似于图2.1的波形。

在图中，X轴度量的是时间、Y轴度量的是幅度。幅度是用以估量声波的强度；幅度越高，声音越响。

Sound Blaster的功能

以下各节详细讨论Sound Blaster的功能。因为游戏杆端口对音乐没用，所以此处各段仅叙述CMS，FM，DSP和MIDI功能。

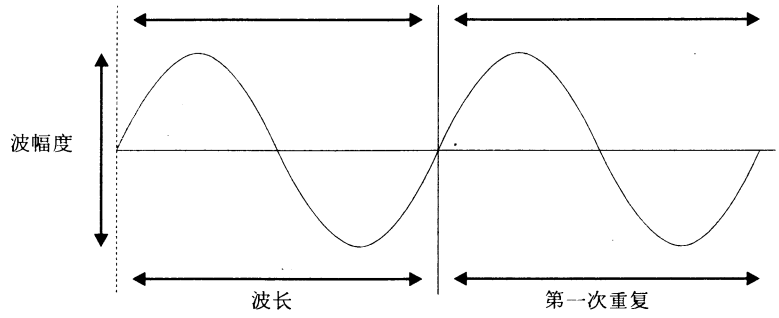


图2.1 正弦波

注解 除了对CMS立体声芯片的讨论外，关于Sound Blaster功能的以下内容也适用于Sound Blaster Pro。在本章的后面部份—“Sound Blaster Pro功能” 详细叙述Pro功能。

CMS立体声芯片

Sound Blaster并非Creative Labs开发的第一个声音卡。公司的第一个声音卡是Game Blaster，它是采用AM（调幅）合成的卡。Game Blaster卡于1987年发布，是IBM PC及其兼容机的第一批声音适配卡之一。根据销售情况，这个卡输给了AdLib卡，没有流行起来。

但是，Sound Blaster从它的Game Blaster原型继承了一个重要的部件—CMS（Creative music system）立体声芯片。CMS芯片用AM技术作声音合成。在推出Sound Blaster 1.0版时，包含了CMS芯片。1.5版就不再包含CMS芯片。

注解 第八章说明如何编程CMS芯片。

虽然CMS芯片提供了12个独立的声部和一个立体声效果，但实际上你并不能用这些芯片改变音色。你所能做的一切是调节音量和频率。你可以调节每个声音的左和右输出的音量。使声音重现期间左、右音量不等，就可以产生立体声效果。CMS芯片没有内设的时钟，所以你必须自己编程音符的宽度。尽管没有时钟，你还是可以与FM音乐结合来使用立体声芯片。

缺乏CMS支持 如果你没有CMS芯片，则在以下两情况时才有理由去购置它：

- 如果你是程序设计员并且你想借助CMS支持你的程序。
- 如果你有支撑CMS的游戏。

因为，在写本书之际，对CMS芯片的支撑很少。同时开发CMS芯片的目的只是使得Sound Blaster与它的原型—Game Blaster兼容。由于对CMS芯片的支撑缺乏，所以可能并不需要去添加它们。

用于改变声音频率的FM芯片

在FM芯片名称中的FM代表频率调制。用这种芯片，你可以采用处理由FM信号所产生的正弦波的办法，来改变声音频率。由设备所产生的声音频率可分为三个元素：音调、音色和幅度。使用FM芯片，你可以用编辑这些元素的方法产生你自己的“乐器”。

注解　有关FM芯片编程的信息见第7章。

注解　因为AdLib卡也支持FM芯片，所以Sound Blaster和AdLib完全兼容，可以一同使用。

FM芯片有两种设置：

- 11--声部设置，由FM声音的6个声部和5个打击乐器声部所组成。籍助6个声部的一个，你就可以产生你所喜欢的虚拟的任何乐器。5个打击声部表示固定的乐器，它们只能以有限的方式进行编辑。
- 9--声部FM音频设置。所有的9个声部都可以按你的爱好来使用。

FM芯片可以重现11种不同的乐器，同时具有很高的保真度。

处理数字声音的数字声音处理器

就其自然状态来说，声波是模拟信号。平滑过渡地改变频率和幅度时，就使声波升高和降低。很明显，计算机不能存贮声音本身。为了存贮声音，信息的格式需要加以改变，以使计算机能将其作为存贮器中的数据来存贮或处理它。换句话说，信息需要数字化。它需要被转换成数字量，计算机才能用它进行工作。

ADC　为了数字化声波，计算机的模拟--数字转换器（ADC）不断地采样声音，每一次得到一个声音的音量和音调的读数。ADC转换这些信息成一些数字。重复地这样作，每秒几千次，计算机就建立了模拟波形的样本--即数字表示。

采样　在采样期间，某些信息不可避免地要丢失，因为ADC不能连续地采样，所以在两次采样之间的时间内出现在音量和音调的微小变化是不会被记录下来的。这就是为什么采样速率越高所得到读数越精确的原因。采样率是声波每秒钟被读取的次数。例如，如果声波以1/8000秒来读，则采样率为8000。

DAC　为了重现样本，你需要数字--模拟转换器（DAC），它的发音读为“dack”。DAC转换数字为模拟信号--电压，并送此信号到音频端口。首先电压通过音频连接器，然后经过放大器（如果希望的话），到达终点--扬声器，扬声器转换该电压成音频振动。

采样率越高，声波被读得越精确。图2.2给出了以高采样率和低采样率所作的同一个声波的采样。如你所看到的那样，低采样率破坏了波形。低采样率的唯一优点是它们所产生的样本需用较少的存贮器来存贮。

ADC用于记录和存贮模拟声音（录音机功能），而DAC是用于重现出这些记录。它们一起就形成了DSP（数字声音处理器Digital Sound Processor）。

注解　缩写词DSP通常用来指“数字信号处理器”。Creative Labs使用字母DSP来指它的数字声音处理器。这就产生了某些混淆，你在使用Sound Blaster时，注意这一点。

用于设备间通讯的MIDI

MIDI（音乐设备数字接口），它的发音读为“middy”，它是一个串行接口标准，用于连接合成器、音乐设备、和计算机。MIDI标准用于保证音乐和声音正确地编码，以使能在设备间通讯。

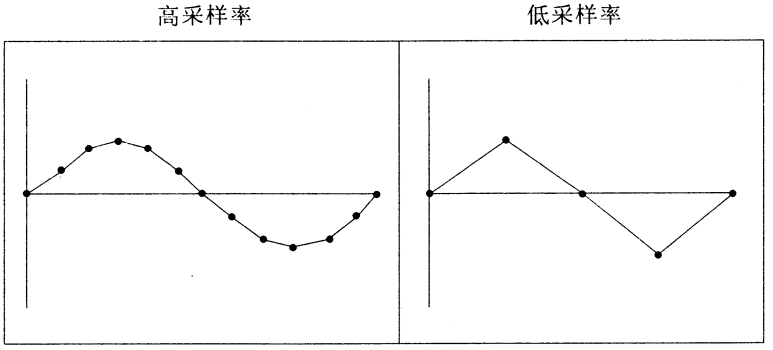


图2.2　采样率的区别

注解　有关MIDI编程的全面讨论见第10章。

现在大多数作曲家都使用合成器。事实上，为了作曲的目的，计算机和合成器已经变成了不可缺少的。音乐家和作曲家经常拥有几个合成器。在MIDI标准出现之前，制造厂为了通讯，已生产了各种连接件，因而合成器通讯是很麻烦的。现在MIDI标准已为全世界所采用，这就是为什么MIDI现在几乎成为所有的音乐和声音卡的中心部件--以及为什么Sound Blaster支持MIDI之缘故。

警告　Sound Blaster不能通过它的MIDI接口同时发送和接收数据，它只能接收或发送数据。这点它不像Sound Blaster Pro，后者是可以同时进行接收和发送的。

MIDI连接盒模块　MIDI连接盒模块能够插入游戏杆插口，它能同时演奏6种MIDI设备。这6种MIDI设备中的5种能接收所有的MIDI数据。这5种设备被称为从动的，因为它们只能重放所接收到的MIDI数据。第6种MIDI设备称为主动的，是为你使用的。

当其音乐家演奏时，主动设备转换音符成MIDI码子，MIDI设备送这些MIDI码子到MIDI连接盒之后，它们到达Sound Blaster。

MIDI设备总是包含两个MIDI端口：

- MIDI入，用于接收MIDI码子。
- MIDI出，用于发送MIDI码子。

连接两个MIDI设备的MIDI电缆也是标准的：　它们为5针DIN连接器。

MIDI也能用来编程你的合成器，例如，使用MIDI你可以创造你自己的乐器，并用你的合成器演奏它们。

Sound Blaster Pro功能

Sound Blaster和Sound Blaster Pro之间的主要区别是Pro版提供了立体声。Sound Blaster Pro包含了Sound Blaster除CMS立体声芯片外的一切功能，而且还提供下列功能：

- 两个FM芯片。我们称之为FM立体声芯片，以区别在Sound Blaster中的单声道芯片。
- 已改过了的DSP芯片。此芯片可记录和播放立体声。
- 对MIDI接口的改进。
- 支持CD-ROM驱动器。
- 对线入输入和CD音频的最佳连接。这些连接是由一个混合器来支持，混合器能混合 由其卡产生的一切声音。

在下面各段，我们详细地叙述这些功能。

立体声FM

为了产生立体声，Creative Labs在Pro上包含有两个完全相同的FM芯片。一个芯片用于左通道，一个芯片用于右通道。因为Pro卡同标准的Sound Blaster卡完全兼容，所以Pro也能工作在单声道模式。在单声道模式中，完全相同的命令和数据被发送到左、右通道。但是，你可以改变每个通道的音量以产生伪立体声。

使用两个FM立体声芯片，9种不同的设备就能够定义在左通道和右通道，这样总共作成18种设备。在11--声部FM音频设置中，每一个声部能具有6种乐器以及5种打击乐器，因此，左通道和右通道总共能产生22种设备。

使用立体声，你就具有了许多单声道不具有的能力。例如，你可以产生出赛车的声音，或造出不一般的“砰”声和“叮”声以表示迷宫游戏中的定向移动。

Pro的立体声数字声音处理器

Sound Blaster Pro较之Sound Blaster使用更为先进的DSP芯片。Pro的芯片能够记录立体声和重现立体声样本。虽然此种改进是受欢迎的，但却加重了存贮器资源的负担。在采样期间，Pro读或重现一个字节对左通道，一个字节对右通道，所以需要2倍的存贮器。

标准的Sound Blaster具有13KHz的有限采样率用于记录，和23KHz用于重现声音。在单声道模式中，用于记录和重现声音的Pro的采样率都增至44KHz。立体声模式具有22KHz的最大采样率；为单声道采用率的一半。

Pro的MIDI能力

使用它的MIDI接口，Sound Blaster可以读和发送来自外部MIDI设备的信号。但是，不像Sound Blaster Pro，Sound Blaster不能同时读和发送。就这方面而言，Sound Blaster可以说成是半双工设备（这个名词经常用来描述调制解调器，用以指不能同时发送和接收数据的设备），如果接口能同时发送和接收数据，就被称为全双工。

全双工设备 在Sound Blaster Pro中的MIDI接口是全双工的。你不必在MIDI接口的读模式和写模式之间转换。对程序设计员这是一种幸福，而且也给于Pro以更专业化的能力。

更进一步，Pro提供了MIDI时间-标记协议。由于有了这个协议，使得连到MIDI网络的外部设备能够同步。换句话说，程序能告诉所连接的MIDI设备什么时候它必须执行某个指令。例如，在一个音乐会上，时标协议可以在键盘演奏者演奏第三支歌曲的第一音符的同时接通点光源。这是个非常有力的功能，特别当它应用到多媒体时，更是如此。

CD-ROM和Pro

CD-ROM驱动器用于个人计算机已经有一段时间了。但是，只是在最近两、三年，CD-ROM的价格才大大下降到足以使普通的用户能考虑买一个。

越来越多的制造厂正在生产CD-ROM驱动器，这使其价格下降。此外，诸如Sierra和LucasArts等游戏制造厂正开始发布基于CD-ROM的游戏。如果你没有CD-ROM，而且你是一个游戏爱好者，则无论从哪种意义上来说都需要买一个。

CD-ROM（此名字是Compact Disk-Read Only Memory 的缩写）工作于紧缩盘（CD）。这种盘可以存贮多达552MB数据。CD是不能写的，CD的内容是只读存贮。

随同Pro使用的CD-ROM的技术要求 你连到Sound Blaster Pro的CD-ROM驱动器必须具有下列硬件特性：

- 它必须能够播放音频CD。
- 它必须满足CD-ROM驱动器的High Sierra标准。（此标准的名字是取自开发它 的人员所住的旅馆的名字，而不是来自超级影星Humphrey Bogart。）
- 它需要一能插入Pro的CD-ROM接口的连接器。不幸的是这是一个专为Creative Labs CD-ROM驱动器开发的专利接口，它不是SCSI接口。要了解更多的信息，请 打电话给Creative Labs。
- 它需要一个能插入Pro CD音频接口的4针音频连接器。
- 同时还必须有驱动程序，使能读CD-ROM。

CD-ROM肯定会在将来主要的学习和娱乐珍品--多媒体--中扮演非常重要的角色。多媒体组合图象、声音、信息和交互作用程序以产生对用户的全面信息冲击。多媒体一定会产生一些大的文件，这些文件必须要存贮在CD-ROM中。

第三章　用Sound Blaster制作音乐

本章内容为：

CDMS作曲器
视见作曲器
声音编辑器（Vedit）
音序器Sequencer Plus Junior

除了玩游戏之外，制作音乐是Sound Blaster带来的最流行的娱乐之一。事实上，Sound Blaster之所以非常流行，很大程度上是因为在所有的声音卡中，Sound Blaster是最适合于制作音乐的。Sound Blaster突破了小的PC扬声器所固有的障碍。它为你的游戏加入了一些真实感，而且使你能听到你只能从影片中才能听到的多声道音乐。当你在玩游戏时，你能听到爆炸声，如司法长官在大街上放枪一样。你能听到奏哀乐，如像他们抬着死者到殡仪馆一样。

但是，Sound Blaster远不是一个大的玩具，感谢FM和MIDI的支持，你能在家里作出真正的乐曲。Sound Blaster成功的一个结果是越来越多支持Sound Blaster卡的音乐程序正在问世，Sound Blaster在这里站住了脚。

本章说明用Sound Blaster创作音乐的某些最重要的方面，它还给出了几个用来产生音乐的程序。制作你自己的歌曲比听预先录好的音乐要有趣得多。幸运的是，因为Sound Blaster包含有CMS芯片、FM合成器、DSP芯片和MIDI连接器，你具有用声音来作实验的很多可能性。对这些音乐源--CMS，FM，DSP和MIDI--的每一个有很多你能用的软件程序。本章所包含的四个软件是：

- 对CMS的CDMS作曲器
- 对FM的视见作曲器
- 对DSP的Vedit
- 对MIDI的Sequencer Plus Junior

下面各段逐个看看这些作曲器，并告诉你它们能作什么。

CDMS作曲器

对大多数软件作曲器，你是用鼠标或菜单命令送入音符。但是CDMS却是让你用文本编辑器送入音符。你可以使用任何输出ASCII文件的编辑器或字处理器。使用由Creative Labs开发的CDMS，你可以使用你自己选择的文本编辑器。所有能用作标准音乐五线谱的音符和符号—包括定调，降半音，升半音，1/4音符，和休止符—都能用文本编辑器送入，而直接翻译成CDMS语言。

注解　如果你想在Sound Blaster使用CDMS作曲器，需要确保CMS立体声芯片安装在Sound Blaster上，而且CMSDRV.COM已装入。本程序可在随Sound Blaster带来的主盘上找到。

CDMS作曲器原来是由Creative Labs为它的Game Blaster卡开发的。CDMS作曲器使用了CMS立体声芯片。这个芯片用在Game Blaster和早期的Sound Blaster上。

注解　有关这些芯片的更进一步的信息见第二章的“CMS立体声芯片”。

CDMS作曲器符号

CDMS作曲器有它自己确定的符号系统。CDMS符号以实际音乐符号为模型。

点命令用于一般信息　你可以把点命令放在文件的头部，以提供一般信息。跟在点命令后面的信息，程序不读它为音乐符号。CDMS使用下列点命令：

点命令	用途
· TITLE	用以指明歌曲的标题，歌曲的标题可以长达32个字符。
· COMPOSER	指明作曲者的名字。
· MESSAGE	让你送入给读者的信息。
· LINES	改变模式。使你能用字母(传统的方法)代替数字送入音符。通常，你是以数字送入音符。如你所预想的那样，8个字母是c，d，e，f，g，a，b和O。O表示休止。

编码例子　为了让你对歌曲是什么样子有一个概念，我们考虑下面两段码例。

第一个是通常模式（送入数字作为音符），而第二个是LINE5模式，它送入字母作为音符。

通常模式：

```
m1|[1=c]   1 2 3 4 | 5 6 7 +1 |
m2|--      7 1 2 3 | 4 5 6 7  |
```

LINE5模式：

```
m1|[1=c]   c d e f | g a b +c |
m2|--      b c d e | f g a b  |
```

音符　CDMS识别8个音符和一个休止符。在通常模式，音符是数字，但是在LINE5模式，你可以用字母表示通常音乐符号。下表示出音乐符号和它们的相应数字：

音符	数字
C	1
D	2
E	3
F	4

G	5
A	6
B	7
休止	0

升半音和降半音分别以符号(#)和符号(@)送入。这些符号被放于音符之前，如:

#G @A @B

声部 CMS芯片提供12个声部。你可以为每个声部定义一种不同的设备。在CDMS语言中，声部为m1到m12，如上面的例子编码中所示的声部一个接一个地在竖直方向放于各行上，而每行所含不多于6个小节。

定调符 在歌曲开始，有一个定调符，它用1=[ ]表示。在上面的例子中，使用C调，符号读作[1=C]，其他可能的定调符有:

A，B，C，D，E，F，G

#A，#B，#C，#D，#E，#F，#G

@A，@B，@C，@D，@E，@F，@G

键盘被分成7个音阶，以字母A到G表示，A是最低的音阶，G是最高的音阶。

小节 在CDMS中，小节由竖直符(|)隔开。小节必须都是一样长短。在上面的例子中，每一小节有四拍，每一行可以包括6个小节。

表3.1示出在CDMS作曲器中使用的各种符号。

表3.1 在CDMS作曲器中所用的符号

符号	用途
+	指音符应以高一个音阶演奏*。
-	指音符应以低一个音阶演奏*。
/	如果音符短于一拍，就分音符成两个。例如，在4/4拍时的8分音符将以1/ 送入，而16分音符以1//送入。(4分音符简写为1)。斜杠也用作把一组符号弄在一起。例如，在4/4拍中，1 2 3 4//指四个四分音符组成一小节。
--	送入一个全音符，例如，在4/4拍中，全音符包括四拍，并写成1--。
^	指明圆滑的连唱。^应放在最后一个音符的前面。
%	当放在小节中时，指明小节不使用时间特征，而是只包含参量。当其很多参量必须一次加入时，使用本符号。如果你使用本参量，则所有的声部都必须以此小节为前导。
T	表示三个一组。例如，在4/4拍中，1 2 3 T//是三个成一组，由每个为半拍的三个音符组成。
[S=..]	指明乐器选择。
[t=..]	指明节拍标记。缺省为4/4，即每小节四拍，而四分音符为一拍。另外的可用节拍标记如1/4，2/4，3/4，4/3，3/8，6/8和9/8。
[V=..]	确定音量级。
[X=..]	选择立体声效果输出。如=L为对左边，=R为对右边，=B是对左右两边。

* 如果在升半音或降半音之前，你要送入这些符号，就放+或-在#和@符号的前面

为了让你对CDMS作曲器文件的形式以及它们如何工作有一个印象，下面给出两个示例文件。清单3.1说明转换一个.ABC文件成.CMS文件。例子中包含了DANCE.ABC文件、伴音TANG01.BCD文件，和这两个文件编辑成的.THM文件--DANCE.THM文件，以及此.THM文件最后编辑成的.CMS文件。

清单3.2示出歌曲SPRING.THM。之所以引入此例，是因为它包含有许多不同的技巧。

清单3.1：编辑.CMS文件。

```
{dance.abc}
;
.copy m1-m3
m1 | [ t=2/4, s=120, 1=C, r=tango1] D 5+111 / | _76/ 5 | 5+222/ | 32/1 |
AA |
m1 | _5+111/ | _76/ 5 | 5567/ | +11/ 1|
AA | C | G | Gs | C |
m1 | 1555/ | 65/ 4 | 4566/ | 54/ 5 |
AA | | F | | Gs |
m1 | 2555/ | 43/ 2| 5. /4// 32/ | 1 1|
AA | G | Gs | G | C |
{tango1.bcd}
; Tango Rhythm - variation #1
B | [p=3] C 1 5 1 5 | 11/ 05/ 1 5 |
C | [p=4, v=180] z0z0/ z0zz/ | zz0z/ z0z0/ |
R | [v=180] xxyx/ xx/ xy/ | x0/ x0/ x0/ x0/ |
{compile dance.abc}
Creative CMS Auto Bass Chord Generator Version 3.00
Copyright (c) Creative Music Lab., 1987. All rights reserved.
Reading DANCE.ABC...Line # 123456789101112131415
    Auto bass/chord TANGO1.BCD
Job done.
{dance.thm}
;
M1 | [t=2/4, 2=120, i=C, r=tngo1] D 5+111/ | _76/ 5 | 5+222/ | 32/ 1|
M2 |                               D 5+111/ | _76/ 5 | 5+222/ | 32/ 1|
M3 |                               D 5+111/ | _76/ 5 | 5+222/ | 32/ 1|
M6 | [p=4, v=180] % | 1010 | 5505/ | 5050/ | 1101/ |
M7 | [p=4, v=180] % | 3030 | 7707/ | 7070/ | 3303/ |
M8 | [p=4, v=180] % | 5050/ | +2202/ | 2020/ | _5505 |
M9 | [p=4, v=180] % | +1010/ | 5505/ | 5050/ | 1101/ |
M10 | [v=180] % | xxyx/ | x0x0/ | xxyx/ | x0x0/ |
M11 | [p=3] % | C_15 | 550+2/ | C_5+2 | _1101/ |
M12 | [p=3] % | C15 | 550+2/ | C_5+2 | _1105/ |
M1 | _5+111/ | -76/ 5 | 5567/ | +11/ 1 |
M2 | _5+111/ | -76/ 5 | 5567/ | +11/ 1 |
M3 | _5+111/ | -76/ 5 | 5567/ | +11/ 1 |
M6 | 1010/ | 5505/ | 5050/ | 1101/ |
M7 | 3030/ | 7707/ | 7070/ | 3303/ |
M8 | 5050/ | +2202/ | 2020/ | _5505 |
```

```
M9 | 1010/ | 5505/ | 4040/ | 1101/ |
M10 | xxyx/ | x0x0/ | xxyx/ | x0x0/ |
M11 | C_15 | 550+2/ | C_5+2 | _1105/ |
M12 | C15 | 550+2/ | C5+2 | _1105/ |
M1 | 1555/ | 65/ 4 | 4566/ | 54/ 5 |
M2 | 1555/ | 65/ 4 | 4566/ | 54/ 5 |
M3 | 1555/ | 65/ 4 | 4566/ | 54/ 5 |
M6 | 1010/ | 4404/ | 4040/ | 5505/ |
M7 | 3030/ | 6606/ | 6060/ | 7707/ |
M8 | 5050/ | +1101/ | 1010/ | 2202 |
M9 | 1010/ | 4404/ | 4040/ | 4404/ |
M10 | xxyx/ | x0x0/ | xxyx/ | x0x0 |
M11 | C_15 | 440+1/ | C_4+1 | _550+2/ |
M12 | C15 | 440+1/ | C4+1 | _550+2/ |
M1 | 2555/ | 43/ 2 | 5. /4 // 32/ | 1 1 |
M2 | 2555/ | 43/ 2 | 5. /4 // 32/ | 1 1 |
M3 | 2555/ | 43/ 2 | 5. /4 // 32/ | 1 1 |
M6 | 5050/ | 5505/ | 5050/ | 1101/ |
M7 | 7070/ | 7707/ | 7070/ | 3303/ |
M8 | 2020/ | 2202/ | 2020/ | _5505/ |
M9 | 5050/ | 4404/ | 5050/ | 1101/ |
M10 | xxyx/ | x0x0/ | xxyx/ | x0x0/ |
M11 | C_5+2 | _550+2/ | C_5+2 | _1105/ |
M12 | C5+2 | _550+2/ | C5+2 | _1105/ |
{ compile dance. thm }
Creative CMS Music Composer Version 3.00
Copyright (c) Creative Music Lab., 1987. All rights reserved.
Reading source file DACE.THM -46 lines read.
Master Bar: 123456789101112131415161 7
Writing CMS file DANCE. CMS...
```

清单3.2：春之歌

```
{spring. thm }
. title SONG OF SPRING
; Enter by Jessie Tan 21st August 1987
. message edited by WH Sim
m1 | % [1-A, s=120, p=13, t=2/14] |
m2 | % [p=4] |
m3 | % [p=4] |
m4 | % [p=4] |
m5 | % [p=3] |
m1 | F 3 ^ 34#45/ / | +1_5/43/ | 2. 4/ | 6. 4/ |
m2 | E 3 ^ 34#45/ / | +1_5/43/ | 2. 4/ | 6. 4/ |
m3 | D 05/5 | 01/1 | 02/2 | 06/6 |
m4 | D 03/3 | 0_5/5 | 06/6 | 0+4/4 |
m5 | C 1 0 | _3 0 | 4 0 | +2 0 |
m1 | 2 ^ 2#12#2 / / | 35/43/ | 21/_7+1/ | 3 2_5/ |
```

m2 | 2 ^ 2#12#2 / / | 35/43/ | 21/_7+1/ | 3 2_5/ |
m3 | 05/5 | 05/5 | 0#4/4 | 05/5 |
m4 | 04/4 | 03/3 | 02/2 | 01/_7 |
m5 | _7 0 | +1 0 | _6 0 | 5 0 |
m1 | +3 ^ 34#45 / / | +1_5/43/ | 2. 4 / | 6. 5/ |
m2 | +3 ^ 34#45 / / | +1_5/43/ | 2. 4 / | 6. 5/ |
m3 | 05/5 | 01/1 | 06/6 | 06/6 |
m4 | +03/3 | _05/5 | +02/2 | 03/3 |
m5 | +1 0 | _3 0 | 4 0 | #1 0 |
m1 | #4~4/32/ | 1_7/+32/ | 2 1_5/ | +5 ^ 5#4~43/ / |
m2 | #4~4/32/ | 1_7/+32/ | 2 1_5/ | +5 ^ 5#4~43/ / |
m3 | 06/6 | 04/4 | 05/5 | 01/+1 |
m4 | 04/4 | 02/_7 | +04/3 | _05/5 |
m5 | 2 0 | 5 0 | +1 0 | _3 0 |
m1 | 2_7/65/ | +5 ^ 5#4~43/ / | 2_7/65 | 7+1/51/ |
m2 | 2_7/65/ | +5 ^ 5#4~43/ / | 2_7/65 | 7+1/51/ |
m3 | 0_2/2 | 01/1 | 02/2 | 01/1 |
m4 | 05/7 | 05/5 | 05/7 | 05/5 |
m5 | 4 0 | 3 0 | 4 0 | 3 0 |
m1 | _7.7/ | 6@7/57/ | 6. 6/ | 76/7+1/ |
m2 | _7.7/ | 6@7/57/ | 6. 6/ | 76/7+1/ |
m3 | 02/2 | 05/@7/ | 05/5 | 0#4/4 |
m4 | 05/5 | +05/5 | 03/3 | 02/2 |
m5 | 2 0 | +2 0 | #1 0 | 1 0 |
m1 | 2. 2/ | 5432/ | 2 #13/ | 6#432/ |
m2 | 2. 2/ | 5432/ | 2 #13/ | 6#432/ |
m3 | 05/5 | 05/5 | 05/5 | 02/2 |
m4 | 02/2 | 02/2 | 02/2 | 01/1 |
m5 | _7 0 | @7 0 | 6 0 | #4 0 |
m1 | 75#43/ | 2_6/ 7/+2///1.// | _7. +2/ | 6#432/ |
m2 | 75#43/ | 2_6/ 7/+2///1.// | _7. +2/ | 6#432/ |
m3 | 02/2 | 01/_6 | 0+2/2 | 02/2 |
m4 | _07/7 | 04/4 | 07/7 | +01/1 |
m5 | 5 0 | 2 0 | 5 0 | #4 0 |
m1 | 75#43/ | 2_6/ +1_#4/ | 5. #5/ | 6. 7/ |
m2 | 75#43/ | 2_6/ +1_#4/ | 5. #5/ | 6. 7/ |
m3 | 020#1/ | 0101/ | 0204/ | 0304/ |
m4 | 070@7/ | 0#406/ | 070+2/ 0102/ |
m5 | 5050/ | 3030/ | 5050/ | 5050/ |
m1 | +1. #1/ | 321_6/ 5. #5/ | 6. 7/ |
m2 | +1. #1/ | 321_6/ 5. #5/ | 6. 7/ |
m3 | 0503/ | 0101/ | _57+24/ | 0 4@6/ |
m4 | 030_@7/ | 0#404/ | 57+24/ | 0 4@6/ |
m5 | 5050/ | 2020/ | 5+2/0 | 13/0 |
m1 | +1. #1/ | 2. 3/ | 4 ^ 4345// | 6 ^ 6345// |
m2 | +1. #1/ | 2. 3/ | 4 ^ 4345// | 6 ^ 6345// |
m3 | 0 5@7/ | 0 @7+@2/ | 01/_6 | 0+1/6 |
m4 | 0 5@7/ | 0 @7+@2/ | 01/_6 | 06/4 |
m5 | 35/0 | 46/0 | 6 0 | 4 0 |

m1 | 6 ^ 6#123// | 4 ^ 4#123// | 4#123// 4123// | 4#45#5//7623// |
m2 | 6 ^ 6#123// | 4 ^ 4#123// | 4#123// 4123// | 4#45#5//7623// |
m3 | 06/4 | 0@6/4 | 05/0 | 0 0 |
m4 | 04/2 | 04/2 | 04/0 | 0 0 |
m5 | 2 0 | +1 0 | _7 0 | 0 0 |
m1 | 54_7+1..32_75// | +3 ^ 34#45// | +1_543/ | 2. 4/ |
m2 | 54_7+1..32_75// | +3 ^ 34#45// | +1_543/ | 2. 4/ |
m3 | 0 0 | 05/5 | 01/1 | 02/2 |
m4 | 0 0 | 03/3 | _05/5 | 06/6 |
m5 | 0 0 | 1 0 | _3 0 | 4 0 |
m1 | 6. 4/ | 2 ^ 2#12#2// | 3543/ | 21_7+1/ |
m2 | 6. 4/ | 2 ^ 2#12#2// | 3543/ | 21_7+1/ |
m3 | 06/6 | 05/5 | 05/5 | 0#4/4 |
m4 | +04/4 | 04/4 | 03/3 | 02/2 |
m5 | +2 0 | _7 0 | +1 0 | _6 0 |
m1 | 3 2_5/ | +3 ^ 34#45// | +1_543/ 2. 4/ |
m2 | 3 2_5/ | +3 ^ 34#45// | +1_543/ 2. 4/ |
m3 | 05/5 | 05/5 | 01/1 | 02/2 |
m4 | 01/_7 | +03/3 | _05/5 | 06/6 |
m5 | 5 0 | +1 0 | _3 0 | 4 0 |
m1 | 6. 4/ | 2. #4/ | 6. 2/ | +1_765/ |
m2 | 6. 4/ | 2. #4/ | 6. 2/ | +1_765/ |
m3 | 02/2 | 01/1 | 02/2 | 04/4 |
m4 | 06/6 | 06/6 | +01/1 | 02/_7 |
m5 | 4 0 | #4 0 | 2 0 | 5 0 |
m1 | +31_76/ | 52/3/5///4.// | 3 05/ | +1_765 |
m2 | +31_76/ | 52/3/5///4. // | 3 05/ | +1_765 |
m3 | 03/3 | 04/2 | 01/1 | 04/4 |
m4 | +01/1 | _07/7 | 05/5 | +02/2 |
m5 | 5 0 | 5 0 | 1 0 | 5 0 |
m1 | +31_76/ | 524_7/ | +1. #1/ | 2. 3/ |
m2 | +31_76/ | 524_7/ | +1. #1/ | 2. 3/ |
m3 | 0101@3/ | 0204/ | 030@7/ | 060@7/ |
m4 | _050+1/ | 0_70+2/ | 0305/ | 0405/ |
m5 | 3 #4/ | 5 5 | +1 1 | 1 1 |
m1 | 4. #4/ 6542/ | 1. #1 | 2. 3/ |
m2 | 4. #4/ 6542/ | 1. #1 | 2. 3/ |
m3 | 0606/ | 0204/ | 0503/ | 0205/ |
m4 | 040@3/ | 0_70+2/ | 030_@7/ | 060+@2/ |
m5 | 1 1 | _5 5 | +1 _5 | 4 @7 |
m1 | 4. #4/ | 6542/ | 135+1/ | _7542/ |
m2 | 4. #4/ | 6542/ | 135+1/ | _7542/ |
m3 | 0406/ | 0204/ | 05/5 | 04/2 |
m4 | 010@3/ | 0_707/ +03/3 | _07/7 |
m5 | 6 +1 | _5 5 | +1 0 | _5 0 |
m1 | 135+1/ | _7542/ | 3 ^ 34#45// | +1 _02#23// |
m2 | 135+1/ | _7542/ | 3 ^ 34#45// | +1 _02#23// |
m3 | 05/5 | 04/2 | 05/5 | 05/5 |
m4 | +03/3 | _07/7 | +03/3 | 03/3 |

```
m5 | +1 0 | _5 0 | +1 0 | 1 0 |
m1 | 5 0_67+1// | 3 0_4#45// | +1_35+1/ | 35+13/ |
m2 | 5 0_67+1// | 3 0_4#45// | +1_35+1/ | 35+13/ |
m3 | 05/5 | 05/3 | _0535/ | +0535/ |
m4 | 03/3 | 03/3 | _0535/ | +0535/ |
m5 | 1 0 | 1 0 | _1 0 | +1 0 |
m1 | 5010/ | 10/ 0 |
m2 | 5010/ | 10/ 0 |
m3 | 3050/ | 50/0 |
m4 | 3050/ | 30/0 |
m5 | 0 10/ | 10/0|
```

视见作曲器

在本章中所描述的每个作曲器都使用Sound Blaster卡中的特殊硬件。视见作曲器是由AdLib卡的创造者写的，它作成了FM芯片的使用。程序已比较老，但是Sound Blaster的爱好者看来对它非常满意。新的版本没有发布，且类似于视见作曲器的程序也没有在市场上出现。

程序只有一屏，以及少量的下拉菜单。就个人来说，我们喜欢它，因为这意味着不必转换屏幕。因为视见作曲器是面向图形的，所以鼠标是必不可少的。

视见作曲器屏幕

如图3.1所示，屏幕由一张大的“创作纸”组成，这个“创作纸”类似于自动演奏的钢琴中的钢琴辊筒。在创作纸的左边是一垂直键盘。创作纸的各水平行相应于键盘的黑键；每个连续的竖直行标志一小节的开始。在小节中的点式水平线标记拍。

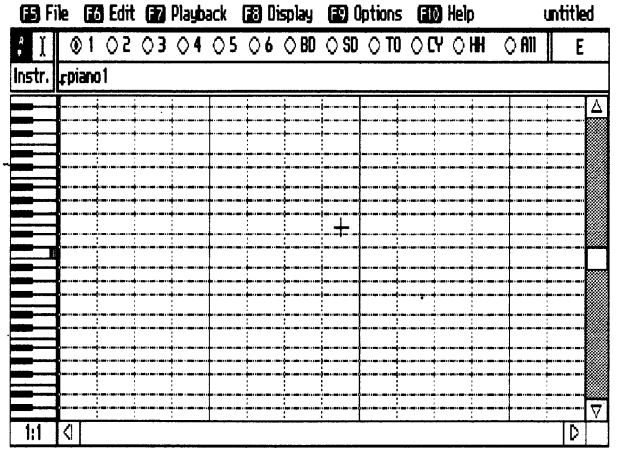


图3.1　视见作曲器屏幕

比起在屏幕上示出的键而言，在键盘上有更多的键。为了看见这些键，使用在创作纸右边的垂直滚动条来进行上、下滚动。使用在屏幕底部的水平滚动条，你可以左、右滚动创作纸。

图3.2示出了在一音乐作品已送入后的视见作曲器的创作屏。现在屏幕看起来真像钢琴辊筒！在屏幕上的各矩形标记音符要演奏。要注意，每个矩形与一键盘位置对齐成一行。在水平方向，音符表示成节拍：每个音符“占据”一个或多个拍或小节。

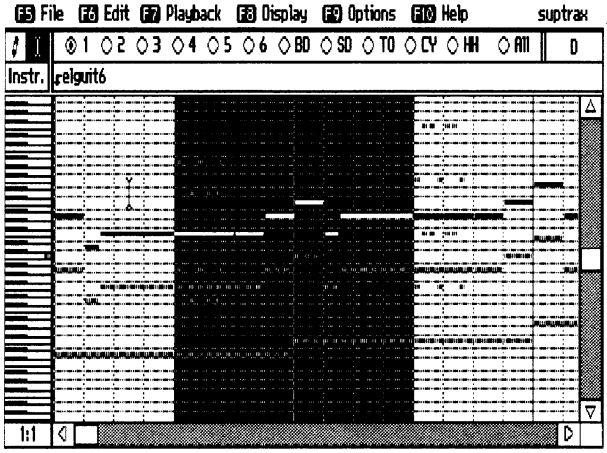


图3.2　具有送入音乐作品的视见作曲器屏幕

创作乐曲

因为视见作曲器具有FM芯片的优点，所以你可以在创作纸上送入几种不同的乐器。屏上的第二行--就在主菜单选项的下边--列出了用一个数字或两个字母码表示的所有乐器。

表示打击乐器的两个字母码为:

BD	低音鼓
DB	小鼓
TD	印度长鼓
CY	铙钹
HH	高帽

你可一次选择一种乐器，或者用第二行右边的ALL选项选择所有的乐器。

在创作纸上你总是一次编辑一种乐器，一当你已选择了一种乐器，在创作纸上的其他乐器的音符就成灰色，这告诉你它们不能够被改变。使用在一个灰色块上放黑色矩形，你可以使多种乐器演奏同一音符。事实上，如果你需要的话，你能使11种乐器演奏相同的音符。

选择乐器和送入音符　注意屏幕上第二行左边的钢笔和钩子。选择钢笔在创作纸上送入音符。当你选用钢笔时，在创作纸上的鼠标指示器变成一个十字线。通过按鼠标左按钮的方法，你放一个音符或一串音符在创作纸上。

选择钩子以改变裁切和粘贴模式。当其鼠标指示器变成钩子时，按下并按住鼠标按钮，向左或向右移动鼠标以选择块，黑色音符将被选取。如果你还想选择在块中的灰色音符，在完成剪切和粘贴之前选择ALL(所有乐器)。

菜单选项

下面是有关下拉菜单各选项的简述。

File菜单 File菜单提供下列选项：

选项	用途
New	从存贮器中删去现行歌曲，使你能开始一新的作曲。
Open	让你选择一个已装入存贮器中文件。
Save	让你以现行歌曲名字存贮一个歌曲到盘上。
Save As	让你以你选用的名字存贮文件。你第一次存贮文件时选用此选项。
Revert	恢复歌曲成为你装入它以前的样式。当你不喜欢你对一首歌曲所作的改变时，选用此选项。
Instrument	让你选择包含各种乐器的文件，这种文件通常有扩展名 .BNK。
Bank File	
Quit	退出视见作曲器，并返回到DOS。

Edit菜单 Edit菜单有下述选项：

选项	用途
Copy	拷贝所选择的块到缓冲区。
Cut	移走所选择的块，并放它到缓冲区。块右边的音符往左移。
Paste	在鼠标指示器的位置插入缓冲区的内容。
Clear	删去选择块的内容。
Semitone Up	升高所选择块的各音符半音。
Semitone Down	降低所选择块的各音符半音。
Octave Up	升高所选择块的各音符一个音阶。
Octave Down	降低所选择块的各音符一个音阶。

Playback菜单 下面为Playback菜单所提供的选项

选项	用途
Play	用所有乐器演奏歌曲。
Playvoice	以一种所选择的乐器演奏歌曲。
Interrupt	停止演奏歌曲。
Basic Tempo	设置每分钟的拍数。在Disply菜单中的BPM（每小节拍数）确定此速度。

Display菜单 下面为Display菜单选项：

选项	用途
Tempo	让你设置歌曲的速度。
Instrument	让你改变乐器。
Volume	设置歌曲的音量。
Pitch Accuracy	设置歌曲的音调。
Large Grid	改变屏幕为大号创作纸，此时使用大的音符。
Medium Grid	改变屏幕为中号创作纸。
Small Grid	改变屏幕为小号创作纸，使在一页中有许多音符。
Ticks Per Beat	设置每拍的时间节拍数。
Beat Per Measure	设置每小节的拍数。

Options菜单 Options菜单给你如下选择：

选项	用途
With Percussion	选择11--声部模式。
Without Percussion	选择9--声部模式。
Audio Feedback	当起作用时，让你听到你送入创作纸上的每个音符。
MIDI Input	从音符的普通输入转到音符的MIDI输入。

视见编辑器(Vedit)

Creative Labs的Vedit，也称为声音编辑器，是一个通用的记录和编辑程序。程序使用了DSP芯片。用Vedit，你能数字化记录模拟信号并存贮它们。对数字形式的样本，你可以裁切并粘贴它，以用在其他任何地方。你也能填充和重复数字样本。Vedit提供很多选项以增强样本。

注解 Vedit的Sound Blaster Pro版称为Vedit2，下面讨论的Vedit大部份内容也适用于Vedit2。

程序是基于图形的。为进行选项选择，你使用鼠标和下拉菜单。也可用键盘键的组合。一当你已下拉一菜单时，按下ALL键和在菜单中带下划的字母就激活该选项。

记录、播放、和编辑样本

只要运行Vedit，你就能看见主菜单。主菜单由5个下拉菜单组成：File、Record、Play、Pack、和Edit，如图3.3所示。

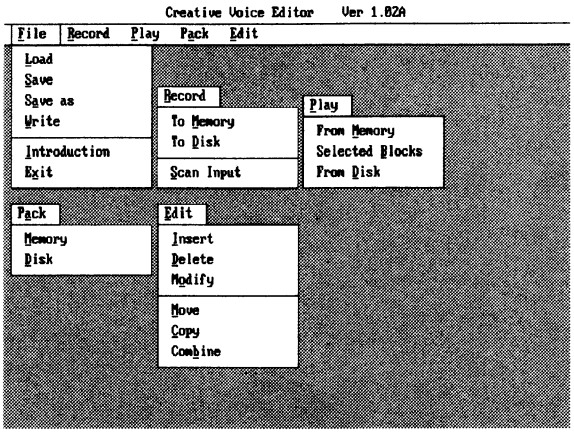


图3.3 Vedit主菜单

装入和存贮样本

使用File菜单，你可以装入和存贮样本。File菜单提供这些选项：

选项	用途
Load	从盘中取出一样本。在你选择此选项后，你将会看到File Directory屏，它显示出.VOC文件清单。当你选取一文件时，出现Block Information屏，它显示出文件的各个块。在此处，你可以删去，移动，播放和修改它们。

注解： .VOC格式在第9章进行详细地讨论。

Save	写前面装入的样本到相同文件。
Save As	完成与Save相同的功能，只是此时你可以为文件送入一不同的名字。
Write	从样本中选取一个块到盘中。
Introduction	显示有关作者的信息。
Exit	返回DOS。

记录样本

Record菜单包括三个功能：To Memory (到存贮器)、To Disk (到盘) 和Scan Input (扫描输入)。如菜单名所暗示的，你可以记录声音到存贮器或盘上。

在你开始记录之前，你可以选择Scan Input来检验一下Sound Blaster的麦克风输入是否实际上接收到了信号。如果没有信号被接受到，你将看不见任何东西，只有一条直线在输入窗中，如果信号正在被接收到，窗口就显示一波形，如图3.4所示。

一当你已确信Sound Blaster正在接受信号时，选择To Memory或To Disk以开始记录信号。在记录能开始之前，Sound Blaster将要求你送入一采样率。如果你选择To Disk，你还必须送入文件名。

提示　对音乐使用高采样率。对声音使用低采样率。记住，采样率用得越高，存贮器也用得越多。

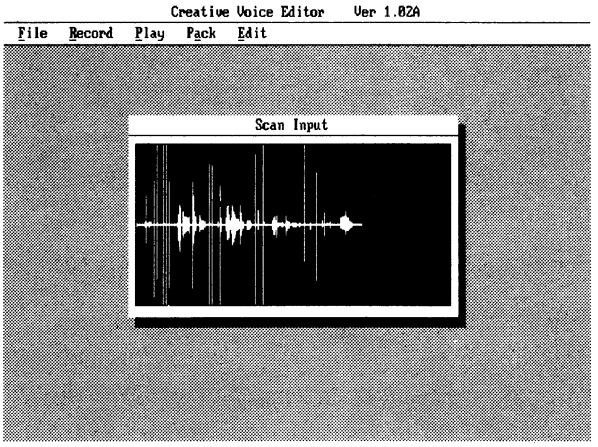


图3.4 Scan Input屏幕

如果一切都工作正确，而且你能装入样本，则你应该能看到表示出样本的各个块的窗口。这个窗口示于图3.5。如图所示，样本总是以一terminator(终止) 块结束。

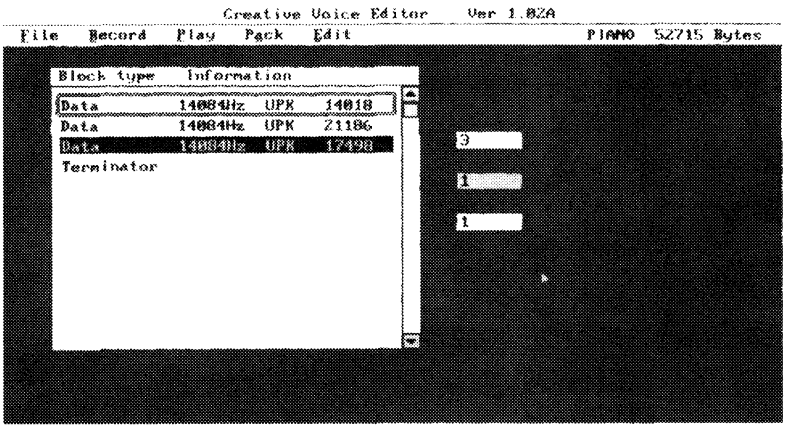


图3.5　装入样本的块信息

你可以用鼠标选择各个块，并完成对块的操作。例如，用通过选择各块，并选用Play菜单中的Selected Blocks选项的办法，你就能只播放所选择的各块。

- 为选取活动块，按鼠标左边按钮一次。活动块总是显高亮度。
- 为了一次选择几个块，按下并按住鼠标按钮，则选中的各块表示出一个轮廓。

播放样本

你可以播放你已选择的块。例如，Play菜单的Selected Blocks选项就允许你只播放被选的各块。从Play菜单，你也可以播放存贮器中的整个样本（使用From Memory选项）或播放盘上的样本（使用From Disk选项）。

使用Pack选项节省磁盘空间

样本可能要消耗掉很多盘空间，所以，Vedit含有一Pack菜单以“打包”各块在一起，而节省盘空间。当其两个、三个或四个连续的样本值被压缩成一个值时，就是打包。打包可以减小文件的大小，可以多达2/3。Vedit在它的Pack菜单中提供了两种选择：

- Memory　打包存在存贮器中的数字音频。
- Disk　打包存在硬盘文件中的数字音频。

有两类打包：无声块打包和数据块打包。使用无声块打包，则字或音符之间的　无声区被压缩；在数据块打包中，文件简单地被压缩。

警告　在打包音频文件方面，有一些商品出售，声音质量可能遭受损失，而且你不能编辑被打包的文件。

编辑所记录的样本

Edit是主菜单条上的最后一个菜单。实际的裁切和粘贴在此菜单产生。Edit菜单提供下述功能：

选项	用途
Insert	让你在Block Information窗口(见图3.5)中的活动块的前面放无声块、标记块、ASCII块，或重复块。这些块在下面被说明。
Delete	从样本中去掉活动块。
Modify	打开Modify菜单，使你能改变采样的波形。Modify菜单的选项在下面说明。
Move	将所选择的块放在活动块的前面。
Copy	复制所选择的块到活动块的后面。
Combine	归并所有选择的块到一个块。各个块被粘贴在一起，但并不混合。

块的类型　有四类块能插入在块信息窗口中的活动块的前面：

- 无声块　是一无声的时间段。用无声块替代低声平声音的办法，你可以节省磁盘 空间。
- ASCII块　是简单的ASCII正文。为制作文档而标记一声音块时，就插入一ASCII块。
- 重复块　重复你已选择的一个样本块或几个样本块。
- 标志块　用于多媒体节目中的动画处理。多媒体播放器能读这些块，并且使用它们来

同步声音和动画。

选择数据块 选择Edit菜单的Modify就产生一个类似于图3.6的编辑屏。使用此屏，你可以选择一数据块并编辑它。

为了选择和编辑一数据块，需要：

1. 移动鼠标箭头到波形中所希望的位置。
2. 按下鼠标左按钮，一条垂直线出现。
3. 为选择一数据块，按下并按住鼠标按钮，在你想放置块尾的位置释放。

编辑数据块 既然你已选择了数据块，你就可以编辑它了。注意在编辑屏上的Play按钮和Zoom条。(见图3.6)

- Play 播放波形。
- Zoom 当Zoom条移到左边时，放大波形；当Zoom条移到右边时，缩小波形。

当你在编辑波形时，使用这些工具来观察和播放波形。

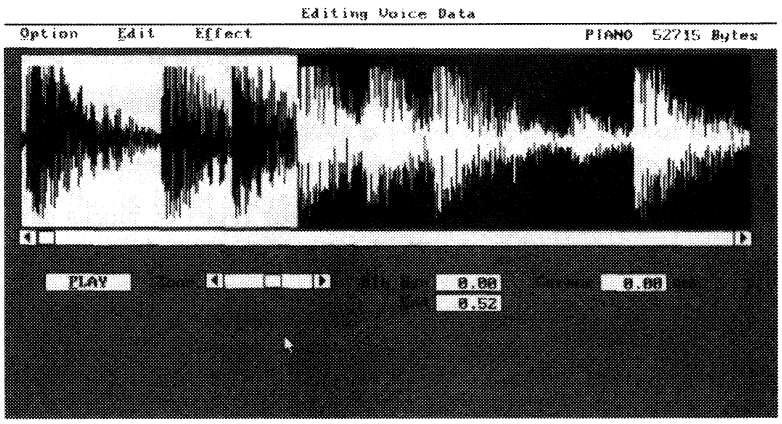


图3.6 编辑数据块

为了帮助你编辑波形，编辑屏幕在它的菜单条上有三个菜单：Option，Edit，和Effect。Option菜单提供三种选择：

选项	用途
Split block	在指示器位置分波形成两块。
Edit Sampling Rate	允许你送入一不同的采样率。
Exit	返回到主菜单。

Edit菜单提供下述选择：

选项	用途
Save	写所选择的数据块到一个文件。

Cut	从波形中移去选中的块，并放于缓冲区中，以使你能在其他地方粘贴它。
Paste	在指示器位置插入缓冲区内容。
Fill	用一个值填充所选择的数据块。
Insert	工作同Fill一样，只是用指示器在波形中指定的值来填充块。

Effect菜单，是一个有趣的菜单，提供下列选择：

选项	用途
Amplify	让你扩大一个块，因此使得它更响和更强有力。当你选择此选项时，出现一个小的对话框，用来送入你想放大块的百分比。百分之五十降低幅度一半，百分之200就放大幅度二倍。

警告　放大声音记录太多，会引起失真。

Echo	允许你产生回声。当你选择此选项时，一个小的对话框出现，用来输入两个参量：百分比和延迟，如图3.7所示。第一个参量是回声的音量，而第二个参量是在一新的回声开始之前必须经过的时间。

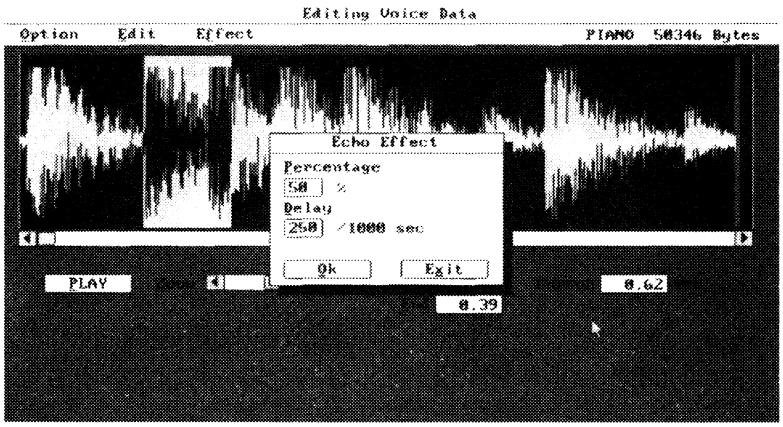


图3.7　产生回声效果

音序器Sequencer Plus Junior

Sequencer Plus Junior（SpJr）由Voyetra生产，它是一软件的MIDI音序器。该程序能记录、安排、处理和重现MIDI数据。SpJr的较易理解的版本被称为Sequencer Plus和Sequencer Plus Gold。Sequencer Plus Junior之所以在这里说明，是因为它是随Sound Blaster MIDI Kit配备

的。

注解 MIDI的详细讨论，见第10章。

如果你想在一个键盘上同时演奏出低音鼓和打击乐器，你就会遇到麻烦。使用一些技巧可以解决这个问题。但是如果你还想演奏钢琴，此时技巧也就不顶用了。使用音序器，一次演奏几种乐器就不再成其为问题。SpJr含有64个音轨。这些音轨类似于在多声道磁带上的磁道，在音带上的每一磁道都能记录一种乐器，音序器的音轨也如此一样。使用MIDI设备，你可以在每个音轨上记录不同的声音。

但是，MIDI设备只有16个通道，它不能同时送出和接收多于16种声音的内容。这就是为什么在Sequencer Plus Junior中为64音轨：作为例子，1到4音轨分配给MIDI通道1。音轨1包含低音鼓、音轨2为小鼓、音轨3为印度长鼓、而音轨4为铙钹。由于一秒钟被分成四个（或更多个）段，使用分配第一段于音轨1，第二段于音轨2，等等的办法，你就能演奏具有四个音轨的一个MIDI通道。如果你对每一个MIDI通道都这样做，你将使用全部的64个音轨。

主屏幕

在SpJr启动后，你会看到一个类似于图3.8的屏幕。注意在屏幕底部的菜单选项。注意，在右边是音轨号和乐器名。音轨之后是你可以赋值的各个选项。

在顶部的状态行示出歌曲的名字和其他的一般信息：

- BPM(每分钟的拍数)是歌曲的速度。速度范围能从16～255 BPM。
- MIDI IN或OUT是指MIDI状态。
- 当其正在进行记录时，REC显现。

为了记录，按R，状态条应显示字母REC。为启动记录，MIDI必须设为IN。当这些条件满足时，你可以演奏MIDI设备，并用MIDI数据填充所选择的音轨。除了主菜单外，程序还包含有四个别的菜单：Edit、Files、Options、View。

Option菜单 示于图3.9的Option菜单提供下列选项：

选项	用途
Metronome	让你通过你的PC机内部扬声器听速度的拍子。当你在演奏MIDI 设备时，Metronome（节拍声）可能是有用的伴音。
Lead-In	指示在演奏或记录之前SpJr记下的小节数。
TS (Time Signal)	指各音轨应如何同步，以使它们同时演奏和记录。
MIDI	指出可能连到MIDI设备的附属设备是否必须被处理。
MIDI THRU	送MIDI端口上接收到的数据到FM芯片。使用MIDI THRU允许，你可以就在记录歌曲时通过Sound Blaster FM听到该歌声。
VELOCITY Filter	赋于所有的音符以最高的速度。某些MIDI键盘不是

速度——敏感的。这意味着它们不能登录按键的速度。用此选项来赋于所有的音符以最高的速度。

Song BRANDIAD STOP Mem 221648
Tk 2 - - - - - - - - - - - - - - - - - - BPM 90 MIDI : IN 1 : 0 THRU : OFF

Trk	Name	Port	Chan	Prg	Transpose	Quantize	Loop	Mute
1	TRUMPET	3	5	37	0.5↑	- - -	- - -	- - -
2	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	MUTE
3	FLUTE	2	5	4	- - - -	- - -	- - -	- - -
4	OBOE	3	4	21	- - - -	- - -	- - -	- - -
5	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	MUTE
6	VIOLIN SOLO	2	4	40	- - - -	- - -	- - -	- - -
7	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	MUTE
8	VIOLIN I	3	3	10	- - - -	- - -	- - -	- - -
9	VIOLIN II	2	3	2	- - - -	- - -	- - -	- - -
10	VIOLA	3	2	11	- - - -	- - -	- - -	- - -
11	VIOLONE	2	2	3	- - - -	- - -	- - -	- - -
12	CELLO	3	1	3	1: 0↓	- - -	- - -	- - -
13	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	MUTE
14	HARPSICHORD	2	1	17	- - - -	- - -	- - -	- - -
15	harp (double)	3	6	17	- - - -	- - -	- - -	- - -
16	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -
17	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -
18	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -
19	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -
20	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -
21	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -
22	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -
23	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -
24	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -
25	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -
26	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -
27	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -
28	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -
29	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -
30	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -
31	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -
32	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -
33	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -
34	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -
35	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -
36	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -
37	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -
38	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -
39	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -
40	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -
41	- - - - - - - - - - - - - - - - - -	1	1	2	- - - -	- - -	- - -	- - -

Main Menu
Delete Loop Mute Name Quit Record Solo Tempo EDIT FILES OPTIONS

VIEW

图3.8 Sequencer Plus Junior的主菜单

Files菜单 示于图3.10的Files菜单是用于存贮和装入歌曲。从这个菜单你也几乎能执行有关文件管理的所有功能而不必返回到DOS。在此菜单上的各选项的意义是显而易见的。

注意SpJr能够处理三种歌曲格式：SNG，MID和ROL：

- SNG是Voyetra自己的歌曲格式。
- MID是标准的MIDI文件格式。
- ROL是由AdLib卡生产者使用的格式。

提示 SNG和MID文件能装入和存贮。ROL文件只能被装入。

View菜单 View菜单示于图3.11中，它可让你获得各个音轨的总貌。在音轨的右边是小

节(称为bar)。空的小节用负号标出，非空的小节以一矩形标出。点指示还没有被记录的小节。

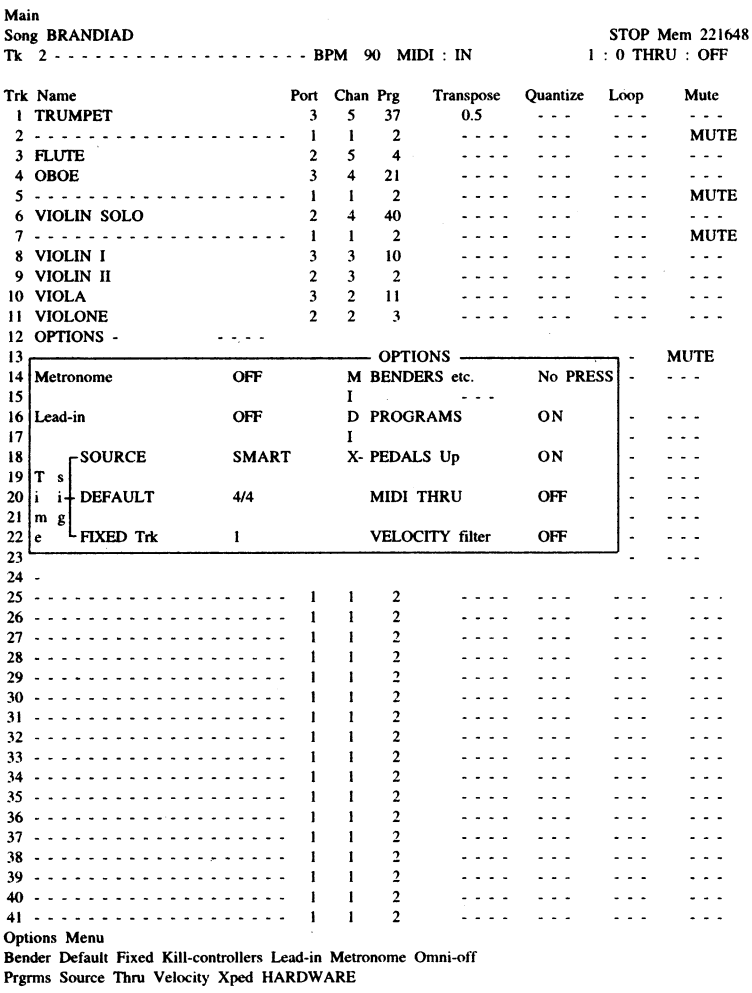


图3.9 Option菜单

View菜单提供了许多有用的能力。除了主菜单上可用的选项之外，本菜单还提供了这些选项：

选项	用途
Goto-bar	让你移到你指定的音轨小节。
Width	仅用小节填充屏幕的总宽度。
Copy	复制你所指定的小节到三个可用缓冲区之一。这些缓冲区称为0，1，和temp。
Zap	提供如Copy同样的服务，只是被拷贝的小节要抹掉。
Delete	其工作同于Zap，但在此情况下，被抹去区域右边的小节

向左移。

Insert	在光标位置插入缓冲区的内容。
Replace	以缓冲区的内容替换小节的指定部份。
Add	加一空的小节。

```
Files
BRANDIAD C : \VOYETRA\SONGS\
Mem 221648  Ext. SNG      BPM  90  MIDI : IN          1 : 0      THRU : OFF

SONG          Size      Date       Time      SONG    Size     Date     Time
[ A: ]                                       TEST    19610    8/16/91  16:58
[ B: ]
[ C: ]
[ D: ]
.
7/25/91 22 : 28
. .
7/25/91 22 : 28
2GTITARS      15819     12/24/90    0 : 49
ALLTIME       16907     12/15/90   21 : 55
BRAND-AD      17531      5/29/90   16 : 53
BRAND1AD      80618      5/30/90   12 : 14
CANCAN        21259     12/08/90   21 : 30
CHEERS3       22091      2/24/91   22 : 37
CMLCHEER      15163      7/17/90   12 : 16
CMDISCO       15467      2/19/91   14 : 00
CMLJING       18075      7/17/90   12 : 22
CMLNUTS       13096      7/17/90   13 : 19
CMLSATIN      27400      7/71/90   13 : 34
CMLSWING      18059      5/31/90   13 : 58
CMLTECH       13355      5/30/90   17 : 46
DAARGAAT      51531      2/24/91   16 : 37
DEMO           9707      3/09/91   12 : 08
DEMOSONG      22379     10/16/90   14 : 04
FANTASIA       7707     12/08/90   16 : 14
HAPPY#42      21611      3/13/91   12 : 25
HAVANAG       13387     12/02/90   18 : 32
HOLDHAND      14333      1/25/90   15 : 51
HOLYCITY      16619     12/28/90   23 : 05
HORN          22763      5/27/91   18 : 13
HORNSBY       22333      6/04/90   17 : 54
INVITE        23499      3/09/91   12 : 08
LARGO         10203     12/05/90    0 : 47
LAVIE         24155     12/18/90    0 : 20
LDYMDONA      15869      1/21/90   18 : 04
LEFREAKC      56315      2/24/91   16 : 32
LETITBE       29965      8/15/89   20 : 48
LIT_HELP      15677      8/15/89   22 : 21
LONESOME      19229      1/20/90   17 : 32
LUCY_SKY      16285      8/19/89   14 : 03
MAPLERAG      17883     11/29/90   11 : 09
MINUETG       13003     11/29/90   11 : 10
SPTUTOR        5915     11/29/90    9 : 07
Files Menu
Buffers-clear  Create-dir  Delete  Free  Load  Mode  New  Path  Quick-find
```

图3.10 File菜单

在View菜单中的记录和播放从光标的位置开始。如果你存贮一SNG文件，缓冲区0，1和temp同文件一起存贮。

编辑声音

如果你需要的话，从View菜单可以直接到Edit菜单。例如，如果你在View菜单，则你可以按E(Edit)来放大某个音轨中的指定小节。

在Edit菜单中，音符的音调在左边垂直地示出。在Edit屏中，你能够加入和移去音符。正

如在View菜单中一样，你可以选择音符进行复制、移去和替代。

音符的宽度以时间节拍（滴嗒）来度量。时间节拍是你能用以工作的最小的时间单位，等于4分音符的1/192。你可以连续地送入音符的长度、音调、和开始。长度和开始以时间节拍计量。

```
Song BRANDIAD                                              STOP Mem 221648
Tk  2 - - - - - - - - - - - - - - - - - - - BPM  90  MIDI : IN      1 : 0 THRU : OFF

Trk Name                          Port  Chan Prg BARS*     ↓8     ↓16     ↓24     ↓32
 1  TRUMPET                         3     5   37   1 - - *
 4  OBOE                            3     4   21   4 - - - - - - - - - - - - - - - *
 9  VIOLIN II                       2     3    2   9 - - *
10  VIOLA                           3     2   11  10 - - - - - *
12  CELLO                           3     1    3  12 - - - - - - - - - - *
13 - - - - - - - - - - - - - - - - - 1     1    2  13
18 - - - - - - - - - - - - - - - - - 1     1    2  18
21 - - - - - - - - - - - - - - - - - 1     1    2  21
24 - - - - - - - - - - - - - - - - - 1     1    2  24
27 - - - - - - - - - - - - - - - - - 1     1    2  27
30 - - - - - - - - - - - - - - - - - 1     1    2  30
33 - - - - - - - - - - - - - - - - - 1     1    2  33
36 - - - - - - - - - - - - - - - - - 1     1    2  36
39 - - - - - - - - - - - - - - - - - 1     1    2  39
View Menu
Add Copy Delete Goto-bar Insert Loop Mute Name Replace Solo Width
Zap EDIT FILES OPTIONS
```

图3.11　View菜单

因为要表示出在各时间节拍中的所有音符，则屏幕太小，你可以使用Unit选项。使用此选项告知Sequencer Plus Junior在光标移动一个位置时，必须加入的时间节拍数。Track功能允许你送入要编辑的另一音轨号。

Note Edit菜单　不像Edit菜单，它在一个小节中显示几个音符，示于图3.12的Note Edit菜单只示出了一个被选择的音符。Note Edit菜单提供下列选项：

选项	用途
Length and Pitch	这些选项同于Edit菜单中的同名项。
Start	让你送入音符和时间节拍的组合。例如，10:+6意味着音符在第10个单位音符之后开始6个时间节拍(此处的单位音符的值在Units中示出)。
Velocity	指示音符的音量。使用"Velocity"一词是因为一音符以该速度按键时也确定了音符的音量。
Off Velocity	它是一个速度值，所按的键以该速度释放。
Accidentals	让你改变在左边的竖直音调。你可以在显示钢琴键、数字和低音之间进行选择。
Freeze	阻止你上、下滚动。使用此选项，你停在一个音阶中。
Note-trig	保证每一个被选择的音符自动地被播放。

```
dit
Song BRANDIAD
                                                          STOP      Mem 59228
Tk   1 TRUMPET              BPM 90 MIDI : IN          2 : 0   THRU : OFF

Environment                          CURRENT NOTE          Units :    16th  Fine
Time Sig :    4/4      Sharps        Pitch :   G   5       Start :       3     0
Time Units :  16th                   Velocity : 127        Length :      1    -8
Freeze :      OFF      Note-trig : ON   Off Vel :   64
! ! % % ( ( . . / / 1 1
BAR  2    OCTAVE   5
b
a#
a
g#
g
f#
f
e
d#
d
c#
c
```

图3.12 Note Edit菜单

QWERTY合成器

你可以从Sequencer Plus Junior中的每一个菜单启动QWERTY合成器。如要启动合成器，你只需简单地按Shift-F1。QWERTY合成器是这样一种合成器，你可以用你的计算机键盘来进行演奏--你可以操作你的键盘就像是真正的MIDI设备一样。你可以记录你所演奏的音符，甚至在演奏时还伴以歌声。

设置各音符的速度、持续时间、和音阶也都是可能的。

第四章 Sound Blaster扩充件

本章内容为：

安装MIDI连接器盒
安装CMS芯片以获得更好的立体声
用Sound Blaster使用CD-ROM驱动器以及如何安装CD-ROM
一次使用两个Sound Blaster

你可以采用安装扩充硬件的办法来提高Sound Blaster和Sound Blaster Pro的能力。但是你能够增加哪种扩充件，是与你用Sound Blaster或Sound Blaster Pro有关。

本章讨论下述扩充件：

- 用于连接其他MIDI设备的MIDI连接器盒。此盒已包含在Sound Blaster Pro中，你可以连接一个到Sound Blaster。
- CMS芯片，使你能在Sound Blaster中具有12个立体声部。你可以对Sound Blaster增加CMS芯片，但对Sound Blaster Pro却不行。在Sound Blaster Pro中已使用立体声FM芯片代替。
- CD-ROM驱动器。此扩充件只适用于Sound Blaster Pro。
- 同时使用两个Sound Blaster。

MIDI连接器盒

用MIDI连接器盒是为了把标准的MIDI电缆连到Sound Blaster。它含有一个MIDI In和5个MIDI Out连接器。一个紧缩版已包含在Sound Blaster Pro中；它只有一个MIDI In和一个MIDI Out电缆。

将MIDI连接器盒接到卡的游戏杆连接器，而将游戏杆插入连接器盒内，它仍还有正常的功能。

注解 **关于MIDI连接器的更多信息，见第二章中的"MIDI用于设备间通讯",关于MIDI编程的信息，见第十章。**

CMS芯片

如果你需要Sound Blaster的12个立体声部，就需要用CMS芯片。这些芯片在Sound Blaster早期的版本中包含得有；但是在Sound Blaster卡的最新版中就不再有CMS芯片，你必须另购买它们。安装CMS芯片涉及四个步骤：

1. 从PC机中取出Sound Blaster卡。
2. 放置芯片在适当的插座中。
3. 把Sound Blaster卡放回PC机。
4. 使用INST-DRV程序安装CMS驱动程序。

因为CMS只是在一定程度上受到支持（通常与AdLib=FM选件一起用，虽然后者只产生高质量的单声道音乐），我们建议在安装任何CMS芯片之前，先听一听已有CMS芯片的Sound Blaster，以使在购买CMS芯片之前，确知你是否值得购买它。

注解　关于这些芯片的更多信息见第2章的“CMS立体声芯片”。

使用SAA 1099代替　你可以使用Phillips的SAA 1099芯片代替CMS芯片。像CMS芯片一样，你需要两个1099。这些芯片功能与CMS芯片等同，但是一般它们较便宜。

CD-ROM驱动器

你可以同Sound Blaster Pro一起使用CD-ROM。在写本书时，只有一种类型的驱动器能同Pro一起使用：　Matsushita CT-521 CD-ROM驱动器。除了提供CD-ROM所具有的一般优点——即可以存贮音乐文件，而不使你的硬盘过载——外，你也能使用Sound Blaster Pro来播放市面上的CD，以及记录它们。Sound Blaster Pro产生高质量的记录。

一片CD可以包含几百个程序以及数据。在一片CD上能存贮约560MB的数据，或大约6百万个字符。存贮整部百科全书和字典的CD正在面市。CD也能存贮音乐。

安装CD-ROM驱动器

CD-ROM驱动器是很容易连接的，首先你放存有CD-ROM驱动程序的5.25软盘到计算机软驱，其次，CD-ROM必须接通电源，当然还要将它接到Sound Blaster Pro。为连接CD-ROM到Pro，将其小型白色的4-针连接头插到小的白色插座。而将大的连接器插到Sound Blaster Pro的背面。

注解　有关CD-ROM的更多信息，见第二章的“CD-ROMS和Pro”。

安装驱动程序

其次，你必须安装两个驱动程序：　SBPCD.SYS和MSCDEX.EXE。

加SBPCD到CONFIG.SYS　加SBPCD.SYS驱动程序到CONFIG.SYS，即在CONFIG.SYS加入下行：

```
DEVICE=SBPCD.SYS /D：MSCD0001 /P：200 或 /P：240
```

下面解释这行的意义：

SBPCD.SYS	驱动程序文件。本文件一定在启动盘的根目录下。
/D	指出能找到SBPCD的设备名（在此情况下为MSCD0001）

/P　　指出I/O地址

SPBCD.SYS设备驱动程序使得CD-ROM硬件能与Microsoft的CD-ROM接口程序MSCDEX.EXE通讯。MSCDEX.EXE驱动程序必须放在AUTOEXEC.BAT文件里。

加MSCDEX.EXE到AUTOEXEC.BAT　MSCDEX驱动程序处理DOS和SBPCD.SYS驱动程序之间的接口，而且含有可由其他程序使用的标准功能。

下面是在AUTOEXEC.BAT中的MSCDEX.EXE行的例子：

MSCDEX.EXE /D：MSCD0001 /L：e /M：10 /E /V

以下是该行意思的说明：

/D：xxx　　这是你放设备名的地方。此名字必须同CONFIG.SYS /D行中的驱动器名一致。

/L：x　　驱动器字母（在上例中为E）。它就像DOS的驱动器字母一样工作。你可以转到此驱动器字母，并从你的CD-ROM运行文件。如果你已用驱动器E，则改变L开关到一未用驱动器字母（例如用/L：F使用驱动器F）。

/M：x　　在存贮器中的缓冲区的扇区数。大量的缓冲区使得处理更快而且错误更少，但是也需要更多的存贮器。MSCDEX将使用十个缓冲区，如果可能的话，并将这些缓冲区放在扩充存贮器中。如果没有扩充存贮器可用，驱动程序将使用标准存贮器。

/E　　此开关告诉MSCDEX使用扩充存贮器来存贮缓冲区。如果你没有扩充存贮器，你可以去掉选项/E。

/V　　此开关告知MSCDEX当它装入时，给出有关它使用多少存贮器的详细信息。

注解　**当你使用/L选用更高的驱动器时，要选用MS-DOS允许使用的驱动器。使用在CONFIG.SYS中的LASTDRIVE=d，你可以设置MS-DOS支持的驱动器数，在=d的情况下,D是允许的最高的驱动器，但是如果你需要的话，你可以改变它成LASTDRIVE=Q或更高。**

播放CD的CDPLYR.EXE　Sound Blaster Pro软件包含CDPLYR.EXE程序。此程序使用上述的各驱动程序来播放普通的CD。如果你使用CDPLYR开始播放CD，它就会一直播放到结尾。如果你不用停止CD来退出CDPLYR，它就会继续播放。这样你在使用另外的软件时，可以听CD。

注解　CDPLYR不常驻内存。

安装第二个Sound Blaster卡

你可以在你的计算机中安装多于一个的Sound Blaster卡。两个Sound Blaster卡就给你多两倍的机会。例如，使用连接一个Sound Blaster到左通道，并连一个到右通道，你就能够产生立体声效果：立体声DSP和立体声FM。你也可以同时用Sound Blaster及用Sound Blaster Pro

来完成此任务。

在写本书时，就我们迄今所知道的还没有人打算使用两个Sound Blaster。如果两个Sound Blaster同时工作，就会造成一个问题：你不能使用DMA来控制两个卡（DMA只能控制一个卡）。在这种情况下，其中的一个卡必须直接编程。这需要处理器时间并将明显地减慢程序执行。

安装两个Sound Blaster 如果你需要在你的PC机中安装两个Sound Blaster，注意下述事项：

- 对每个Sound Blaster设置不同的基本端口。如果你不这样作，两个卡就不能被独立地编程。
- 为安全起见，去掉一块Sound Blaster的DMA Enable跳接器。最好你从卡上移去最高基本端口的跳接器。大多数Sound Blaster检测程序都不检验是否有第二个Sound Blaster存在，所以当DMA不正常工作时，它们并不报告错误信息。在搜索Sound Blaster卡时，程序一般从低到高传送端口。

安装第三个、第四个、第五个和第六个卡都是可能的，只要你为每个卡选择不同的端口，并且去掉DMA Enable跳接器。但是，只有在你需要几个独立的通道而且你能够写你自己的软件时，才考虑同时使用几个Sound Blaster。（迄今为止就我们所知，没有用于运行几个Sound Blaster的软件）。

如果你已经具有一个Sound Blaster卡，而你想购买Sound Blaster Pro的话，你可以考虑保留老的Sound Blaster卡，而与Sound Blaster Pro一道运行，不要卖掉它。你这样的处置将还有CMS芯片留下，这就是要你保留Sound Blaster卡的充足理由。如果你运行两个Sound Blaster，这样就是两个卡用DMA来完成声音输入／输出。对于此种情况，你可以对两个卡使用不同的采样率。

使用连接Blaster到左通道，和连接Sound Blaster Pro到右通道，你就可以播放具有不同采样率的立体声样本。使用在Sound Blaster Pro上的立体声DSP这是不可能的，它对左和右两个通道是使用相同的采样率。

安装Sound Blaster和Sound Blaster Pro 如果你安装了Sound Blaster和Sound Blaster Pro两者，记住下述建议：

- Sound Blaster和Sound Blaster Pro必须被安装在不同的端口。记住Sound Blaster Pro使用所选基本端口之后的17个口，所以最后两个端口（这用于CD-ROM）可能重叠Sound Blaster上所设置的端口。即使不为相同端口，下列设置也是不可能的：

  Sound Blaster Pro上设为220H和Sound Blaster上设为230H

  Sound Blaster Pro上设为240H和Sound Blaster上设为250H

  除去这些设置之外，每种选择都是可能的。
- 因为Sound Blaster的缺省设置是DMA通道1，为了使Pro不用该DMA通道，Sound Blaster Pro的设置必须改变（DMA0用于VOICE，而DMA3用于CD-ROM，或反之）。

你不能在一个PC中安装两个Sound Blaster Pro。两个DMA选择要重叠，而且也不能断开DMA。此外，这要花更多的钱，而且此种组合可能不会为任何程序所支持。

第五章　了解MIDI设备接口

本章内容为：

连接MIDI设备
了解MIDI Thru端口
观察音序器
使用MIDI和Sound Blaster能够作什么?

MIDI代表音乐设备数字接口，其发音读为“middy”。MIDI标准允许不同的设备和合成器相互连接并互相交换音乐信息。为演奏MIDI设备，你不必熟悉MIDI语言，你只需要正确地连接电缆。

本章说明什么是MIDI，并仔细讨论MIDI设备连接、MIDI设备和MIDI软件。

注解　关于MIDI语言的详细信息，见第十章。

什么是MIDI?

MIDI在某种意义上来说是一种“国际语言”，它允许不同国家和不同的乐器制造厂所制造的乐器能相互进行交流。撇开合成器不说，由于有了MIDI标准并能进行交流，因此各种乐器都配置MIDI。近来，诸如吉它、萨克斯管、电钢琴和鼓等一些设备都有内建的MIDI接口。

MIDI硬件规范　MIDI硬件规范是指标准连接器应如何设计以及它们应是如何工作。MIDI硬件标准规定了输入和输出通道（称为MIDI端口）的类型、电缆型式以及插座类型。三种类型的端口为：MIDI In、MIDI Out以及MIDI Thru。

MIDI信息　音乐信息使用MIDI信息的办法互相交换。MIDI信息是一种编码，它包含有关音乐成份的信息，如音量、音调、音符长,自然还包含音符本身。例如，一个MIDI信息可以告诉设备“以速度74播放音符C4”。

连接MIDI设备

每个MIDI设备——无论是合成器、键盘或吉它都有MIDI In，MIDI Out,有时还有MIDI Thru端口：

- MIDI In端口接受MIDI数据。事实上，此连接是设备的“耳朵”。合成器、键盘或其他的MIDI设备用它的MIDI In端口接受MIDI信息。Sound Blaster MIDI接口的Out电缆连到合成器、键盘或其他MIDI设备的In端口。
- MIDI Out端口送出MIDI数据。若还用上面的比喻来说的话，它就是设备的“嘴巴”。合成器、键盘或其他的MIDI设备通过MIDI Out端口送出MIDI信息。因此，Sound

Blaster MIDI接口的In电缆连到合成器、键盘或其他MIDI设备的Out端口。

- MIDI Thru连接允许你菊花链式地链接设备。当一设备被连到Thru连接时，它就无阻挡地送它的信息进入设备，虽然它们是In信息。

MIDI Thru端口

为了了解MIDI Thru如何工作，考虑下面的例子。假定你是在演奏合成器，而你又想演奏两种鼓乐设备，合成器只有一个MIDI Out端口连到鼓。因此你可以连接第一个鼓到MIDI Out端口，而将第二个鼓连到第一个鼓的MIDI Thru端口。MIDI Thru端口不改变数据地再送出在MIDI In 端口接收到的所有MIDI数据。

从理论上来说，能够连到Thru端口的MIDI设备的数量是无限的。但实际上，你不能作无限地连接，因为，信号通过电缆时要减弱。例如，由MIDI Thru端口连接的第8个设备就不再能够正确地读取数据和传递数据。又信号在电缆中从A点到B点需要若干（但很少）毫秒，当其大量的设备被连接时，就会出现小的延迟。

解决这些问题的一种轻便设备是Output Selector（输出选择器）或MIDI Thru盒。此盒子具有一个MIDI IN端口，以及多至16个的MIDI Thru端口。使用一个MIDI Thru盒,则最多可有16种MIDI设备，能同时从主控设备（正在演奏的乐器）接收数据。

提示 如果你需要用长电缆连接几个MIDI设备，应考虑加入放大器。此种设备在其转发输入MIDI信号之前将信号予以放大。

MIDI设备

已经有许多声音设备的MIDI版本，包括手风琴、萨克斯管、吉它、钢琴和鼓。这些设备用拾音设备（麦克风）接收声音来产生MIDI数据，并用某些类型的智能电路来转换声音信息成MIDI信息。除了声音设备之外，还有许多数字设备可用,其中合成器和键盘是最重要的。

除了产生声音之外，你也可以将MIDI用于表达，例如，如果你需要对节日电视加上伴音，你就可以使用一MIDI同步器来使某个音乐作品在某个场景的同一时刻开始播放。

音序器 关于字处理的一件好事是当你进行校正时，你不必再重新键入整个文档。同样的事情发生在用MIDI作曲。以前，各音符必须要写下来，如果要进行大的修改，则每件事情都必须要从头重作。使用了MIDI音序器，则情形就不再是这样了。

音序器是这样一种设备，它贮存而且有时让你编辑由其他MIDI设备产生的MIDI数据。软件顺序器是计算机顺序器。它们给于你在编辑MIDI数据中以极大的灵活性。MIDI音序器被连到主控器和从动器，MIDI音序器贮存由主控器所供给的所有MIDI数据。一个MIDI音序器要配置一个硬盘驱动器，以存储MIDI数据，这样就总有数据供你使用。也可以使用音序器播放接收到的数据。MIDI数据甚至能作某种程度的处理。例如，你可以改变不正确的音符。

MIDI软件

对程序员来说，MIDI最大的优点在于MIDI设备能直接连到计算机。由于把MIDI和计算

机连接起来了，我们就可以借助于MIDI语言来访问MIDI系统。使用计算机编程，你可以用软件来替代诸如MIDI音序器之类的设备。而且软件音序器更灵活的多，价格也较“盒式音序器”便宜。

同MIDI有关的大多数软件程序都是音序器。软件音序器的最大优点是它总是反映最新的MIDI发展。软件MIDI音序器的其他优点是：

- 数据能可视见的表示在屏幕上。
- 裁切和粘贴功能允许你直接在屏幕上编辑数据。
- 软件随时都可以扩展。
- 软件音序器可以包含更多的功能，因为计算机是一开放的、交互作用的系统。
- 软件音序器是更为用户友好。
- 用户对音序器具有较多的控制。

使用Sound Blast和MIDI你能作什么？

由Roland制造的MPV-401插板是一个标准卡，它已经开发了MIDI设备和个人计算机的MIDI连接。

很多音序器都使用此卡。但遗憾的是Sound Blaster MIDI接口与MPV-401不兼容。之所以不兼容是因为MPV-401不可能复制——或者严格地说MPV-401含有一个内含专用软件的芯片。使用MPV-401软件的音序器绝对禁止使用同Sound Blaster有关的软件。

第二个问题是MIDI数据必须要送到的端口在Sound Blaster 和MPV-401上是不同的。然而，Sound Blaster是完全符合由MIDI协会所规定的MIDI协议的。麻烦的事是软件必需要重写，以使得同Sound Blaster兼容。

已做此事的第一个公司是Voyetra。此公司已发布了三个音序器，从简单的到非常全面的，它们都吸取了Sound Blaster 所提供的MIDI可能的优点。为了给你一个音序器如何工作的概念，第3章包含了Voyetra Sequencer Junior的描述。

有另一个问题,它同Sound Blaster 以及其对MIDI标准支持有关 ：Sound Blaster MIDI是半双工的。这意味着你可以送出或接收数据，但两者不能同时进行。因此，当Sound Blaster送MIDI数据到主控器时，Sound Blaster 不能同时接收你演奏主控器所产生的MIDI数据。这个缺点已由于 Sound Blaster pro 的引入而得的补偿。Sound Blaster pro是全双工的。

Sound Blaster pro配置有MIDI时间-标记协议。此协议用来同步MIDI设备。每一个MIDI信号被赋予一个时码。用这种办法，你可以精确地确定是在什么时刻接收的MIDI码。

所有的Sound Blaster都有一个64字节的缓冲器，用来存储MIDI数据。缓冲器按照FIFO（先入，先出）系统进行填充和推出，这意味着进入缓冲器的第一个字节也就是离开缓冲器的第一个字节。

Sound Blaster MIDI接口满足国际MIDI协会的要求。你可连接各种MIDI设备到Sound Blaster的MIDI接口。迄今为此，只涉及到了 Sound Blaster 1.0 和 1.5，你一定要确使各个设备与全双工MIDI连接无关。

第二部分　编程Sound Blaster

第六章　编程定时器芯片

本章内容为：

定时器芯片是如何工作的
通过通道端口编程定时器芯片
计数器和定时器芯片如何一起工作
为定时器通道设置频率

定时器芯片就像你计算机中的时钟，各种软件程序和驱动程序都使用此芯片。定时器芯片的频率可以为各种目的而进行编程。因为一定的声音类型要求其特定的定时，所以你必须有一个时钟，以确使音乐以平稳的、恒定的速度被播放。这就是定时器芯片的用途。

PC-XT使用定时器芯片8253，而PC-AT使用8254。所有的定时器芯片都具有相同的功能模式，并以相同方式工作。现代的各种芯片都重复了8254芯片。所以此处所提供的信息适用于486和386机以及XT和AT机。

定时器芯片如何工作？

定时器工作在1.19318MHz频率，并占用三个独立的定时器通道。每个通道可工作在6种不同的模式。事实上，8253和8254芯片是将三个16位定时器作在一个芯片内。你通过读和写寄存器I/O端口地址040H到043H来同定时器交互作用。端口043H用来编程定时器。每个定时器能工作在六种不同的模式。地址040H到042H用来读和写16位的值三次（编号为0，1和2）。

对每个定时器通道，设定了一个16位计数器，计数器以1.19318的固定频率加1,在恒定的时间区间产生中断。16位计数器值的范围为0到65535，值0等效于65536。

定时器通道

每个定时器通道0通过中断控制器8259产生IRQ0——即是INT8。通道0检测软驱的马达是否正在运转。

通道1用于DMA存贮器刷新操作。这些操作的目的是定时地刷新在存贮器中的所有信息。计数器的初始值为18，所以存贮器刷新的频率为：

1193180/18=66287.78Hz

通道2留给一般用途用。此通道经常同扬声器一起使用，以产生特殊音调的声音。

就本书的目的而言，只有通道0是重要的，因为通道0 对处理器产生中断。这就是为什么程序员只需要对INT8写他或她自己的中断子程序来处理音乐。使用此种办法,音乐能以恒定速度播放，而主程序不必以任何形式来监视或处理它。当然，主程序必须启动播放音乐。

通常，INT8每秒调用18.2次。这个速率远远低于播放音乐和采样的速度。在本章中，你

将会知道如何设置此值以得到更高的频率。

定时器芯片的端口

定时器芯片可通过40H到43H端口来编程。40H到42H端口能写和读，但43H端口却只能写。40H、41H和42H端口分别被分配给定时器通道0，1和 2。通过这些端口，计数器的初始值可以被设置。

43H端口是通用的设置端口。你可以通过各通道端口之一或43H端口来编程定时器通道。表6.1示出43H端口各个位的功能。

定时器通道可通过43H端口来编程。你用7和6位来选择此通道，如表6.2所示。

表6.1 43H端口各位的功能

位	功能
7，6	选择三个计数器之一
5，4	设置由7，6位选择的计数器
3，2，1	设置计数器的工作模式
0	选择普通或BCD编码

表6.2 定时器通道选择

7和6位	通道
0 0 b	0
0 1 b	1
1 0 b	2

通过40H，41H，或42H端口，一个定时器通道被用16位的初值设置。用于这些端口本身只有8位，要放入16位值到定时器则出现了问题。为了放16位值到定时器，使用5和4位，表6.3示出控制读—装载格式的各位。

表6.3 读-装载位设置

5 和 4 位	模式
0 0 b	直接读现行计数器值
0 1 b	只读和写高字节（ MSB ）
1 0 b	只读和写低字节（ LSB ）
1 1 b	首先读或写低字节（ LSB ），然后读和写高字节（MSB）

最后的三种模式——3到1位——用于选择定时器工作模式。如果第一种模式被选择，计数器的现行值可被读取，计数器就不能被设置。

3到1位确定定时器通道的工作模式，示于表6.4。

通道0工作于模式3，这意味着该定时器通道工作于方波。最后一位确定计数器的内部操作，示于表6.5。

表6.4 各种模式

3，2，1 位	模式
0 0 0 b	0
0 0 1 b	1
0 1 0 b	2
0 1 1 b	3
1 0 0 b	4
1 0 1 b	5

表6.5 计数器模式

0 位	计数器
0 b	2进制计数器模式
1 b	BCD计数器模式

计数器和定时器芯片

如果计数器要设置初始值，那么十进制值必须转换成16位的二进制数。每一次，二进制计数器减一个二进制数 1 （111b，110b，101b,...）。但是，按照BCD编 码来传送初值和使用BCD计数器也是可能的。对此种技术，使用BCD格式计数，而计数器从BCD数10到BCD数9。值00010000b （=10 BCD = 16 十进制） 的下一个值不是00001111b （=15 十进制），而是00001001b （=9 BCD = 9 十进制）。

如果使用BCD计数器，16位计数器能以最大值10000来设置。使用这种办法，范围要小得多，但是设置BCD数较设置普通的二进制数要容易得多。

对通道0，使用普通的二进制计数器。

注解 **当它们生成执行码时，编译器和汇编器转换十进制数为它们等值的二进制数。因此，在本书中将不使用BCD计数器，而总是使用普通的二进制计数器。**

提示 **在设置43H端口后，一个初值必须要赋给所选择的定时器通道。定时器芯片在基值被设置之前，将不处理任何事。换句话说，只要你是只写了43H端口，而计数器的初值没有被设置，就没有中断产生。**

为对定时器通道0设置一特定频率，应遵循下列步骤：

1. 用频率去除1,193,180来计算出初值。
2. 写值00110110b到43H端口（通道0+第一个LSB; 然后MSB+模式3+二进制计数器）。
3. 写8个低位到40H端口。
4. 写8个高位到40H端口。

现在INT8将以指定的频率被调用。你可以建立你自己的中断子程序。此子程序必须以固定的时间间隔，每秒钟调用老中断子程序18.2次。调用老中断子程序这件事是需要的，特别是当你的程序是常驻内存，或工作于时间——敏感部分时。

因为中断是通过8259芯片传递，所以你应该告诉此芯片它可能对处理器产生另外的中断。如果你没有这样作，就没有另外的IRQ产生。

用写值20H到20H端口的办法，你指明8259能处理下一个中断并送它到处理器。如果老的子程序被调用，则你不必这样作，因为此子程序自身就传送中断。

如果你的子程序慢于被调用的速度，则你必须保留一个标志，用来指明子程序是否已经为忙。如果你不这样作，此子程序将重复地被处理，即使当它还在忙时也如此。但是，不要忘记以固定的时间间隔调用老的中断子程序。

组成本章其余部分的三个清单，包含了一些编码的例子,它们表示一个跑表。清单6.1是一用Pascal写的例子，清单6.2是一C写的例子,而清单6.3是一汇编写的例子。在这些例子中，定时器通道0被置成频率100Hz，一个计数器以百分之一秒的精度严格跟踪时刻。

在 Pascal 和C清单中（清单6.1和6.2）中，老的中断子程序被设置在另一个自由中断里，所以老的中断以一种简单的方式被调用。在汇编清单中,使用一个 FAR JMP,你也可以使用一 FAR CALL，但是不要忘记，在 FAR CALL进行之前，用 PUSHF 放一标志到堆栈，因为子程序用一个 IRET 结尾。

注解 关于读和复制清单的建议，见本书序言中的“关于程序清单的说明”。

程序清单6.1：定时器使用的Pascal例子

```
/* Program TimerTest */

/* Uses Dos, Crt */

Const
    OldTimerInt=103; {old timer interrupt}

Var
    Counter : Word;          {counter for the old interrupt}
    HSec    : LongInt;       {number of hundredth of a second}
    Ch      : Char;          {for key press}

Procedure SetTimer (Rout : Pointer; Freq : Word) ;
{Set timer interrupt, Rout is called Freq times a second.}

Var
    ICnt  : Word;
    OldV  : Pointer;

Begin
    Inline ($FA) ;                    {CLI, interrupts off}
    ICnt: = 119380 Div Freq ;         {calculate basic counter}
    Port [$43] : = $36 ;              {set mode}
    Port [$40] : = Lo (ICnt) ;        {write LSB}
    Port [$40] : = Hi (ICnt) ;        {write MSB}
```

```
    GetIntVec (8, OldV);                { old int vector }
    SetIntVec (OldTimerInt, OldV) ;     { int 8 now Int OldT }
    SetIntVec (8, Rout) ;               { new int handler }
    Inline ($FB) ;                      { STI, interrupts on }
End;

Procedure RestoreTimer;

Var
    OldV : Pointer;

Begin
    Inline ($FA) ;                      { CLI, interrupts off }
    Port [$43] : =$36                   { set frequency }
    Port [$40] : =0;                    { of 18.2 Hz }
    Port [$40] : =0 ;
    GetIntVec (OldTimerInt, OldV) ;     { restore INT vector}
    SetIntVec (8, OldV) ;
        Inline ($FB) ;                  { STI, interrupts on }
End ;

Procedure NewTimer; Interrupt ;
{ The new timer interrupt }

Var
R : Registers ;             { Dummy nedded for Intr }

Begin
    Dec (Counter) ;                     { decrease counter }
    If Counter = o Then Begin           { call old INT ? }
        Intr (OldTimerInt, R) ;         { yes }
        Counter : = 100 DIV 18 ;        { restore counter }
      End
    Else { no }
        Port [$20] : =$20 ;             { int. is handled }
    Inc (HSec) ;                        { increase interrupt }
End ;

Begin
    Counter : =1 ; {Initialize counter}
    SetTimer (@ NewTimer, 100) ; {set int at 100 Hz }
    WriteLn ;
    WriteLn ('Stop watch, press a key ... ') ;
    WriteLn ;
    Ch : =ReadKey ; { wait for key press }
    HSec : =0 ; { set Hund. Sec to 0 }
    Repeat
        GotoXY(1, WhereY) ; {show time }
        Write ((HSec Div 360000) : 1, ' : '
```

```
            (HSec Div 6000 Mod 60) : 2, ' : '
            (HSec Div 100 mod 60) : 2, ' . '
            (HSec Mod 100) : 2) ;
    Until KeyPressed ; {stop at key press }
    Ch : =ReadKey;  { read key }
    RestoreTimer ; { restore timer }
    WriteLn;
End.
```

程序清单6.2：定时器使用的C例子

```
/* Timer test program */
#pragma inline                  /* use assembler */
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#define OldTimerInt 103         /* old timer interrupt */

unsigned Counter ;          /* cpimter for old interrupt */
long unsigned HSec ;        /* number of hundredth of a secound */

void SetTimer (void interrupt (*Rout) ( ) , unsigned Freq)
/* Set timer interrupt, Rout is called Freq times a second. */
{
    int ICnt ;
    asm cli ;                               /* CLI, interrupts off */
    ICnt = 1193180 / Freq ;                 /* calculate basic counter */
    outportb (0x43, 0x36) ;                 /* set mode */
    outportb (0x40, ICnt & 255) ;           /* write LSB */
    outportb (0x40, ICnt >8) ;              /* write MSB */
    setvect (OldTimerInt, getvect (8) ) ;   /* copy int 8 */
    setvect (8, Rout) ;                     /* new int handler */
    asm sti ;                               /* STI, interrupts on */
}

void RestoreTimer ( )
{
    asm cli ;                               /* CLI, interrupts off */
    outportb (0x43, 0x36) ;                 /* set frequency */
    outportb (0x40, 0) ;                    /* of 18.2 Hz */
    outportb (0x40, 0) ;
    setvect (8, getvect (OldTimerInt ) ) ;  /* restore int 8 */
    asm stil;                               /* STI, interrupts on */
}

void interrupt NewTimer ( )
/* The new timer interrupt */
{
```

```
    struct REGPACK R ;                  /* dummy nedded for intr */
    -- Counter ;                        /* decrease counter */
    ++HSec;
    if (Counter == 0) {                 /* call old INT */
        intr (OldTimerInt, &R) ;        /* yes */
        Counter = 100 / 18 ;            /* restore counter */
    }
    else /* no */

    outportb (0x20, 0x20) ;             /* int. is handled */
}

void main ()
{
    char Ch ;
    int U,M,S,H ;

    Counter =1 ;                        /* Initialize counter */
    SetTimer (NewTimer, 100) ;          /* int at 100 Hz */
    printf ("\n") ;
    printf ( " Stop watch, press a key ... \n ") ;
    Printf ( " \n " ) ;
    Ch = getch () ;                     /* wait for key press */
    HSec =0 ;                           /* set Hund. Sec at 0 */
    do {
        gotoxy (1, wherey () ) ;        /* show time */
        H = HSec % 100 ;                /* hundredth of a second */
        S = (HSec / 100)  % 60 ;        /* secounds */
        M = (HSec / 6000) % 60;         /* minutes */
        U = HSec / 360000 ;             /* hour */
        Printf ( " %2d : %2d : %2d. %2d ", U, M, S, H) ;
    } while (!kbhit () ) ;              /* stop at key press */
    Ch = getch () ;                     /* read key */
    RestoreTimer () ;                   /* restore timer */
    printf ( " \n " ) ;
}
```

注解 两个机器码指令由ASM命令使用，但是，要指出的是程序也可以不用这些指令来工作。它们包含在这里只是出于安全的理由。如果希望的话你可以省去具有ASM命令的各行，使得程序能在 Turbo C 编辑器中测试。不要忘记去掉包含 #progma inline的第一行。

程序清单6.3 定时器使用的汇编例子

```
        .MODEL TINY
        .CODE
        ORG 100H

cr=13
```

```
lf=10

Start PROC NEAR
        mov dx, offset IntroText            ; welcome
        mov ah, 9                           ; message
        int 1h
        mov dx, offset NewTimer             ; routine
        mov ax, 100                         ; frequency
        call SetTimer
        mov ah, 0                           ; wait for
        int 16h                             ; key
        mov [WORD PTR Counter +0] , 0       ; start at 0
        mov [WORD PTR Counter +2] , 0
ShowCounter :
        mov ax,  [WORD PTR Counter+0]       ; retrieve value
        mov dx, [WORD PTR Counter+2]
        mov di, offset HexText
        call ConvertHex                     ; to hexad.
        call ShowHex                        ; print
        mov ah, 1                           ; test for
        int 16H                             ; key press
        jz ShowCounter                      ; a key ?
        mov ah, 0                           ; yes, read
        int 16H                             ; form buffer
        call RestoreTimer
        mov dx, offset EndofText            ; end
        mov ah, 9                           ; message
        int 21H
        mov ax, 4c00H                       ; return to
        int 21H                             ; DOS
ENDP

IntroText       DB cr, lf, "Press a key ... ", cr, lf
                DB cr, lf, "$"
InfoText        DB "Counter : "
HexText         DB "00000000H. $"
EndText         DB cr, lf, cr, lf, "End ... ", cr, lf, "$"
OldVector       DD 0            ; old timer handler
Counter         DD 0            ; the ticks counter

HelpCounter     DW 1            ; old handler call

; Show InfoText at same line.
ShowHex PROC NEAR
        mov ah, 3               ; determine cursor position
        mov bh, 0               ; at page 0
        int 10H
        xor dl, dl              ; go to beginning of
        mov ah, 2               ; line
```

```
        int 10H
        mov dx, offset InfoText                 ; print info text
        mov ah, 9
        int 21H
        ret
ENDP

; Set timer with the new frequency AX and set
; interrupt 8 with the new interrupt routine CS : DX.
SetTimer PROC NEAR
        cli                                     ; interrupts off
        push es
        push ax                                 ; store frequency
        mov ax, 0                               ; set vector segm.
        mov es, ax
        mov ax, cs
        xchg dx, [WORD PTR es : 8*4+0]          ; change offset
        xchg ax, [WORD PTR es : 8*4+2]          ; change segment
        mov [WORD PTR OldVector+0] , dx         ; and store
        mov [WORD PTR OldVector+2], ax          ; them
        pop bx                                  ; restore frequency
        pop es                                  ; restore ES
        mov al, 00110110b                       ; chan. 0+L/M+mode 3+Bi
        out 43H, al
        mov dx, 1193180 SHR 16                  ; determine counter
        mov ax, 1193180 AND 65535
        div bx
        out 40H, al                             ; first LSB
        mov al, ah
        out 40H, al                             ; and then MSB
        sti                                     ; interr. on
        ret
ENDP

; Restore timer interrupt, so the old handler
; is called 18.2 times a second.
RestoreTimer PROC NEAR
        cli
        mov al, 00110110b                       ; chan. 0+L/M+mode 3+Bi
        out 43H, al
        mov al, 0                               ; set frequency
        out 40H, al                             ; at 18.2 Hz
        out 40H, al
        push ds                                 ; reset int. vector
        lds dx, [OldVector]
        mov ax, 2508H
        int 21H
        pop ds
        sti                                     ; interr. on
```

```
        ret
ENDP

; The new timer interrupt, increase counter and call
; the old handler, if necessary.
NewTimer PROC FAR
        add [WORD PTR cs : Counter+0] , 1        ; increase
        adc [WORD PTR cs : Counter+2], 0         ; counter
        dec [cs: HelpCounter]                    ; decrease help counter
        jz OldInterrupt                          ; old call ?
        push ax                                  ; no, store AX
        mov al, 20H                              ; pass on that int.
        out 20H, al                              ; is received
        pop ax                                   ; restore AX
        iret
OldInterrupt :
        mov [cs : HelpCounter] , 100 / 18        ; restore help c.
        jmp dword ptr [cs : OldVector]           ; old handler
ENDP

; Counvert DX : AX to a hexadecimal representation
; starting at ES : DI.
ConvertHex PROC NEAR
        push ax                                  ; store AX
        mov ax, dx                               ; first process DX
        call HexWord
        pop ax                                   : restore AX
HexWord:
        push ax                                  ; restore AXC (AL)
        mov al, ah                               ; first process AH
        call HexByte
        pop ax                                   ; restore AL
HexByte:
        push ax                                  ; store AX (AL)
        shr al, 1                                ; first process the
        shr al, 1                                ; most significant
        shr al, 1                                ; nibble (4 bits)
        shr al, 1
        call HexNibble
        pop ax                                   ; restore AX (AL)
        and al, 15                               ; least sig. nibble
HexNibble :
        add al,  "0"                             ; add '0' to AL
        cmp al, "9"                              ; AL>9
        jbe NoAsciiConversion
        add al, "A"-"9"-1                        ; AL between 'A' and 'F'
NoAsciiConversion:
        stosb                                    ; store AL
        ret
ENDP

END Start
```

第七章　编程FM芯片

本章内容为：

相加和频率调制合成
设备和组文件如何工作
以CMS格式播放音乐
以CMS文件、以Adlib、以及用Sound驱动程序播放音乐
设置与改变操作器的端口
FM芯片的寄存器设置

在本章中，你会学到如何编程FM芯片来产生声音和播放音乐。为了产生声音和播放音乐，你可以使用驱动程序和编程FM芯片。FM芯片与AdLib卡完全兼容。因此，AdLib软件都能工作于Sound Blaster卡。但是，并不是所有的Sound Blaster软件都能工作于AdLib。这不是由于FM芯片，而是因为Sound Blaster的结构的原因（在本章的后面加以说明）。

FM芯片如何产生声音

你可以用FM芯片的各种设置来产生声音。在本段，我们说明这些设置，并告诉你如何用它们作实验。

操作器　FM芯片的心脏是操作器。FM芯片具有18个操作器。一个操作器由3部分组成：

- 振荡器（oscillator）
- 包络产生器（envelope generator）
- 声平控制器（level controller）

如图7.1所示，振荡器产生一（变换过的）正弦波，此正弦波随后由包络产生器进行编辑。其次，声平控制器确定波形的强度。最后的输出是产生声音的信号。

注解　在本段的后面，我们说明操作器的每个部分如何被设置和改变。

产生声音的FM芯片模式　FM芯片提供三种模式用于产生声音：

- 旋律模式。在此模式中，你能使用9种设备。
- 节奏模式。在此模式中，可以使用六种普通设备和五种节奏设备。此五种节奏设备是：低音鼓、高帽、印度长鼓、小鼓以及铙钹。迄今为止，在本模式中，所涉及到的节奏设备，你只能改变低音鼓和印度长鼓的高、低音。
- 组合声部合成模式。在此模式中，几个声部可同时打开或关闭。使用不同的音调和音量，并同时演奏它们，你就可以产生讲话。虽然，产生讲话是一种有趣的选择，但无

论AdLib或Sound Blaster手册都没有详细地加以说明。

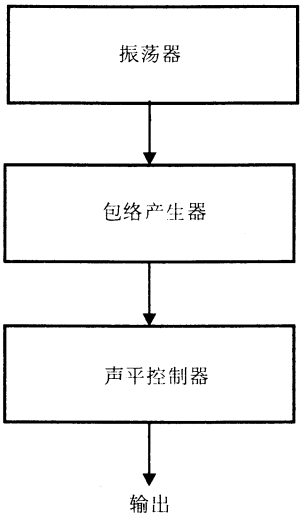


图7.1 操作器如何工作

相加合成和频率调制合成 FM芯片提供了用于普通设备和低音鼓产生声音的两种方法：相加合成和频率调制（FM）合成。相加合成和频率调制合成这两种合成都需要两个操作器。

注解 低音鼓是唯一使用两个操作器的节奏设备。高帽、印度长鼓、小鼓和铙钹只使用一个操作器。

如图7.2中所示，相加合成以并行方式使用两个操作器： 输出由操作之和组成。考虑下述简单的公式：

$$F(t)=Op1(t)+Op2(t)$$

$$Op(t)=E(t)*\sin(Wt+\Omega)$$

此处

- F(t)是输出，
- Op1(t)和Op2(t)表示操作器，
- E(t)和Ω对每个操作器不同。

此种方法产生的声音是非常有限的；只能产生风琴之类的声音。

如图7.3 中所示，对第二种方法--频率调制合成，让你产生更多变化的声音。使用此种方法，两个操作器以串行方式使用。第一个操作器--调制器，作用于第二个操作器--载波。考虑下述公式：

$$Fm(t)=Em(t)*\sin(Wt+\beta*Fm(t))$$

$$Fc(t)=Ec(t)*\sin(Wt+Fm(t))$$

此处，在第一个公式中的参数W可不同于第二个公式中的W，而调制器可以使用它自己的输出再作为输入。这就是众所周知的反馈原理。β值可以改变，以修改反馈强度。

由两种合成方法的任一种所产生的声音皆可以某个频率进行一定时间。音调（以某频率形成的声音）能按时打开和关闭。声音按照操作器的设置来继续或停止。

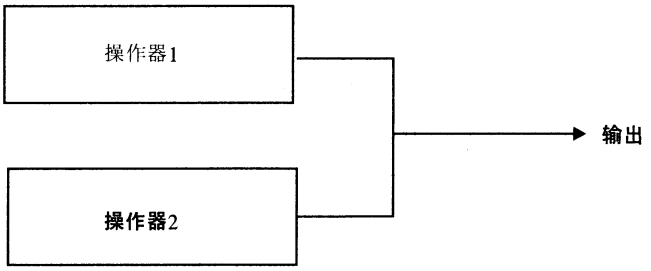


图7.2　相加合成

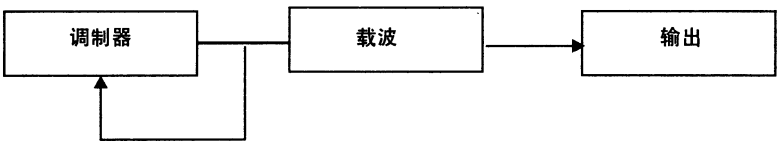


图7.3　频率调制合成

操作器的三个部分

如你所知道的，操作器的三个部分是：振荡器、包络生成器以及声平控制器。以下将详细地叙述每一个部分。

振荡器

振荡器包括四个可调的设置：

- 频率乘法器(MULTI)，对所产生信号的频率乘以某一个数。因为最后的信号由 两个操作器确定，所以此设置允许你产生由两个和声信号组成的信号。
- 频率颤音(VIB)，当选用它时，在信号的频率上产生一个小的波动。换句话说，它产生了颤音的效果。
- 调制反馈(FB)允许你设置输入信号的强度。如前所述，可以用它自己的输出信号再作为输入。
- 波形信号(WS)让你使用正弦波，以及使用由正弦波派生出来的波形。图7.4示出四种可能的波形。

包络产生器

如图 7.5 所示，包络产生器有六种设置。这六种设置（以及保持声平和最高声平）是：

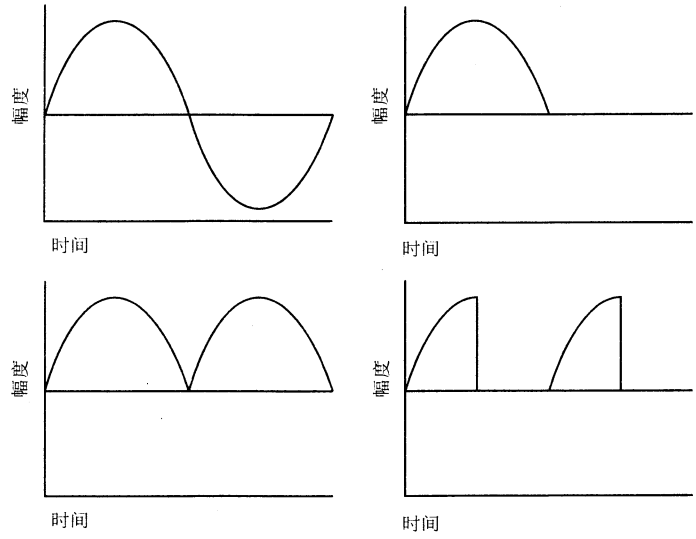


图7.4　四种可能的波形

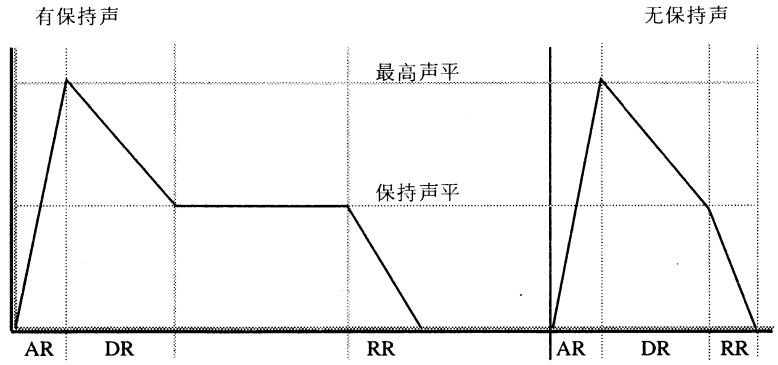


图7.5 具有和不具有保持声音的包络产生器

设置	说　明
AR	增高率。输入信号从0 声平（完全无声）开始，以后，信号以某种速率上升至它的最高声平。这是你用来设置速率的选项。
DR	衰减率。在信号到达最高声平后，你用此选项所设置的速率降至保持声平。
SL	保持声平。在信号到达最高声平后，它以所设置的衰减率降至保持声平。此声平可以等于最高声平，在此种情况下你所建立的衰减率影响很小。在音符被选用期间，信号保持在此声平。当其音符不再使用时，信号以某个速度降至0。为了达到此种情况，保持声平必须要选择。如果不是这样的话，信号在到达保持声平后就以该速率降至0。
RR	释放速率。这是上面说的信号从保持声平降到0的速度。
EG-TYP	保持声音。此选项确定在音符被选择期间，声音是否必须停留在保持声平。
KSR	长度尺。当其本选项被选用时，音调的持续期变得同音调的高低有关。例

如，钢琴较高的音调不能维持较低音调那么长、该选项允许你模仿此种效果（音调越高，它的长度越短）。

声平控制器

声平控制器包含三种设置：

设置	说　明
TL	音量级。这是操作器信号的最终音量。相加合成，两个操作器的TL设置确定最终信号的音量。频率调制合成，最终信号的音量由载波确定。
KSL	强度尺。此选项同包络产生器的长度尺有同种影响。例如，在钢琴上的低音调不仅更长，而且它们也较高音调更响。你可以用该选项获得此效果（音调越高，声音越静）。
AM	颤音幅度。正如颤音频率一样，此选项引起小的波动，只是在这时是信号强度的小波动。

注解　对这些设置的最大和最小值，以及如何作成这些设置本身的讨论，见本章后面的“在FM芯片中寄存器如何工作”。

首先，我们讨论这些设置如何被存储，以及它们如何能够被使用。

设备格式

与乐器的普通或节奏模式有关，它有一个或两个操作器。对每种设备的设置存在于设备文件中，也称为组文件 (bank file)。 有很多设备，因此也就产生了许多小的设备文件。这些文件管理起来是较困难的。

为使设备文件易于管理，Sound Blaster让你工作于设备组。一个设备组包含有几种设备的数据，这样你只必须对一个文件工作，而代替对几个文件工作。(大多数AdLib软件只对设备组工作)。对AdLib 以及Sound Blaster，设备和设备组都存在。但是，这些文件的结构是不同的。在下面分四段讨论格式。

SBI格式

SBI是为Sound Blaster所提供的设备格式。设备文件具有扩展名.SBI。文件具有下列结构：

00H-03H	文件ID。在此处，你能找到正文SBI，其后为EOF符号1AH。使用这个部分，你检查文件是否为设备文件。
04H-23H	设备名。此名字以一零字节结尾。例如： piano，0。由于它以后要清除，所以此域只有小的作用，而且在组文件中你会找不到它。

其次16个字节包含调制器以及对载波的相同信息。偶数字节（24H，26H，28H以及2CH）包含调制器的信息，而奇数字节（25H，27H，29H，2BH和2DH）包含载波信息，往后为一

附加字节，为调制器的数据。

这16个字节的结构也用在由 Sound Blaster提供的其他的文件格式中。表7.1示出SBI格式的结构。

INS格式

INS是由AdLib卡提供的设备格式。此设备文件具有扩展名 .INS。如表7.2所示，此种格式的结构很不同于SBI格式。

表7.1 SBI格式构造

字节24H和25H：音频设置

位	设 置
7	频率颤音 (VIB)
6	幅度颤音 (AM)
5	保持声音 (EG-TYP)
4	长度尺 (KSR)
3-0	频率乘法器 (MULT)

字节26H和27H：信号的强度尺和音量

位	设 置
7-6	强度尺设置 (KSL)
5-0	信号 (TL)

字节28H和29H：增高率和衰减率

位	设 置
7-4	增高率（AR）
3-0	衰减率（DR）

字节2AH和2BH：保持声平和释放率

位	设 置
7-4	保持声平 (SL)
3-0	释放率 (RR)

字节2CH和2DH：声波类型

位	设 置
7-2	0
1-0	声波类型 (WS)

字节2EH：有关反馈和所用合成的附加信息

位	设 置
7-4	0
3-1	反馈 (FB)
0	综合类型 (FM)

字节2FH-33H：留待将来扩充用

表7.2 INS格式的结构

字节	设 置
00H	设备类型。如果此字节包含0值，为普通设备。此值为1，指节奏设备。
01H	若涉及的是节奏设备，此字节包含该设备类型的设备号。对普通设备，此域无作用。

下面各字节为调制器用

字节	设 置
02H-03H	强度尺 (KSL)
04H-05H	频率乘法器 (MULT)
06H-07H	反馈设置 (FB)
08H-09H	增高率 (AR)
OAH-0BH	保持声平 (SL)
0CH-0DH	保持声音设置 (EG-TYP)
0EH-0FH	衰减率 (DR)
10H-11H	释放率 (RR)
12H-13H	音量级 (TL)
14H-15H	频率颤音设置 (VIB)
16H-17H	幅度颤音设置 (AM)
18H-19H	长度尺设置 (KSR)
1AH-1BH	合成类型。如果必须使用频率调制合成，则此字节含1；如果使用加法合成，字节为0。注：对FM芯片和SBI所用的设置这是相反的。

载波设置

字节	设 置
1CH-1DH	强度尺设置 (KSL)
1EH-1FH	频率乘法器 (MULT)
20H-21H	未用
22H-23H	增高率 (AR)
24H-25H	保持声平 (SL)
26H-27H	保持声音 (EG-TYP)
28H-29H	衰减率 (DR)
2AH-2BH	释放率 (RR)
2CH-2DH	音量级 (TL)
2EH-2FH	频率颤音 (VIB)
30H-31H	幅度颤音 (AM)
32H-33H	长度尺 (KSR)
34H-35H	未用

INS与SBI的比较 在SBI和INS设备文件之间存在明显的不同。虽然SBI格式的数据占用较少的字节，但很多信息已加到此格式，所以文件几乎是同样大小。在 INS格式中，你可以加入某些有关设备类型的信息，而在SBI格式中，这是不可能的。另一方面，在SBI格式中，你可以指定波形。对于INS格式中的设备组文件，这也是可能的。

IBK格式

IBK为Sound Blaster提供的设备组文件。文件具有扩展名 .IBK。一个IBK文件包含128种设备，其结构示于表7.3。

表7.3 IBK格式的结构

字节	设 置
000H-003H	文件ID，后跟EOF字节1AH。
004H-803H	设备参量。对每一种设备，保留了16个字节。此16个字节的结构，与在SBI格式中偏移量24H处开始的相同（见表7.1）。
804H-C83H	128种设备对应的名字。对每一个名字，保留了9个字节。名字必须以0字节结尾，所以名字的最大长度为8个字节。例如：Harp，0。

BNK 格式

BNK是AdLib用的格式。一个BNK文件可以包含高达65，535种设备。如已说过的，用于再生和存储设备数据的BNK格式为大多数新的AdLib软件所支持。如果没有组文件被指定，则STANDARD.BNK文件被使用。

组文件由三个部份组成：

- 第一部分为文件头，它为一般信息。
- 第二部分为一个或多个记录组成，为设备名。
- 第三部分为一个或多个记录组成，为设备设置。

后两部分的大小是可变的。

文件头 文件头有下列结构

字节	设 置
00H-01H	文件版本。第一字节是高位字节（点的前面部分）。第二个字节是低位字节（点的后面部分）。
02H-07H	文件ID。包含正文 AdLib，允许你检验文件是否确实为一设备组。
08H-09H	所用的设备数据的编号。
0AH-0BH	设备数据的总数。此值等于或大于所用设备数据的编号。
0CH-0FH	设备名表的绝对起始点。用此值设置文件指针，它将指向第一个设备名。
10H-13H	设备数据表的绝对起始点。上一个域的说明同样适用于此。
14H-1BH	未用，都为0值。

设备名表由两个或多个连续记录组成。记录的数量在文件头中指出。

记录 记录有下列结构

字节	设 置
00H-01H	数据偏移量索引。按下列计算，用此值确定数据的起点： Start=abs_data_start + size instrument setting * this value (起点=绝对数据起点+设备设置大小*此值)

02H	指明此记录使用（值1）或不用（值0）。
03H-0BH	设备名，用0字节结尾。名字的最大长度（不包括0字节）为8个字节。

数据记录 设备数据表也由一个或多个连续的记录所组成。有多少个设备名就有多少数据记录。数据记录具有下述结构：

字节	设　置
00H	设备类型：0=普通；1=节奏。
01H	设备号，如果节奏设备被涉及的话。
02H-1EH	调制器设置。
1FH-2BH	载波设置。
2CH	调制器的正弦波类型。
2DH	载波的正弦波类型。

调制器和载波设置同于INS格式中设置，只是此时每个设置只为一个字节，而不是两个。因为最大的可能值为63，所以这点没有什么关系。

名字表必须要按字母顺序，而且它可能不包含未用的名字。未用的名字必须在已用的最后名字之后。由于设备名表受这些规则的限制，你可以使用比线性方法要快的搜索方法，(线性方法是从第一个名字开始，向下搜索，直至找到匹配的名字)。因为各名字都有一索引值，所以数据表就不必与名字表相同顺序，而且设备数据不必相互直接跟随。

因为总数和所用数据编号是分别提供的，你可以为任何新的设备在组文件中保留空间。这就是为什么不必读和写整个文件的道理。修改可以直接放在文件中，这比读整个文件要花少得多的时间。

BNK和IBK比较 这是很明显的，BNK格式较之IBK格式更为灵活，虽然在一个组包含大量的设备时，IBK更为紧凑。IBK格式最多可包含128种设备。如果你具有更多的设备，则你必须使用几个IBK文件。如你所看到的，每种格式都有其自己的优点和缺点。

SBI、INS、IBK和BNK的程序

本部分提供一些程序，它们使得管理不同格式更容易。程序清单7.1是一通用库，它包含有读、编辑和写四种格式的各种函数。也包含了几个转换子例程。INS和SBI格式被内部使用(此种作法的理由，以后会清楚)，以及一个组类型被创建。组类型是几种设备的链接表。库没有包含任何扩展检验子例程；为证实每项事情都做得正确，这必须由你来作。当然，你可以加入你自己的检验子例程到库中。

注解　这个库也可供演奏程序和其它程序使用。

程序清单7.1 读和写四种格式的Pascal程序。

```
Unit Instr;
{ Unit for keeping up instruments and banks }
Interface

Const
    SBIPt =0;
    INSPt =1;

Type
    { Define used types }
    INSOp       = Record
                    KSL, MULTI, FB,
                    AR, SL, EG_TYPE,
                    DR, RR, TL, AM,
                    VIB, KSR, FB : Word;
                  End ;
    INSFormat   = Record
                    Mode,
                    Number    : Byte ;
                    Modulator,
                    Carrier   : INSOp;
                    MWafeSel,
                    CWafeSel  : Word;
                  End;
    SBIFormat   = Record
                    Snd, KSLTL,
                    ARDR, SLRR,
                    WS, FBFM  : Array [0..1] of Byte ;
                    Dummy     : Array [0..3] of Byte ;
                  End ;
    SBIFormatp  = ^SBIFormat ;
    INSFormatp  = ^INSFormat ;
    InsName     = Array [0..8] of Char ;
    InsTp       = Record
                    Sort : Byte ;
                    Case Boolean of
                          True : (INS : INSFormatP) ;
                          False : (SBI : SBIFormatP) ;
                  End ;
    BnkTp       = ^BnkRc ;
    BnkRc       = Record
                    Name : InsName ;
                    Ins  : InsTp ;
                    Next : BnkTp ;
                  End ;
```

```
{ Load instrument }
Procedure LoadINS (N : String; Var I : InsTp) ;
Procedure LoadSBI (N : String ; Var I : InsTp) ;
{ Load bank file }
Procedure LoadBNK (N : String ; Var B : BnkTp) ;
Procedure LoadIBK (N : String ; Var B : BnkTp) ;
{Save on instrument }
Procedure SaveINS (N : String ; I : InsTp) ;
Procedure SaveSBI (N : String ; I : InsTp) ;
{ Save bank }
Procedure SaveBNK (N : String ; B : BnkTp) ;
Procedure SaveIBK (N : String ; B : BnkTp) ;
{ Remove one instrument or whole bank from memory. }
Procedure RemoveIns (Var I : InsTp) ;
Procedure RemoveBnk (Var B : BnkTp) ;
{ Some routines for maintaining a bank }
Procedure FindIns (N : InsName; B : BnkTp ; Var F : BnkTp) ;
Procedure DelIns (N : InsName ; Var B : BnkTp) ;
Procedure AddIns (N : InsName : Var B : BnkTp ; I : InsTp) ;
Function NumberOfIns (B : BnkTp) : Word ;
Procedure CopyIns (S : InsTp ; Var T : InsTp) ;
Procedure GoodInsName (Var N : InsName) ;
{ Conversion routines }
Procedure SBIToINS (Var I : InsTp) ;
Procedure INSToSBI (Var I : InsTp) ;
Procedure StrToName (S : String ; Var N : InsName) ;

Implementation

Type
    ByteArray = Array [0..52] Of Byte ; { help types }
    WordArray = Array [0..52] Of Word ; { at conversion }
    { AdLib file formats }
    BNKHeader  = Record
                    Version :  Word ;
                    ID :  Array [0..5] Of Char ;
                    NoUsed,
                    NoTotal :  Word ;
                    NameStarrt,
                    DataStart :  LongInt ;
                    Dummy :  Array [0..7] Of Byte ;
                 End ;
    BNKInsName= Record
                    IndexNo :  Word ;
                    Used :  Byte ;
                    Name :  InsName ;
                 End ;
    BNKOp      = Record
                    KSL, MULTI, FB,
```

```
                    AR, SL, EG_TYP,
                    DR, RR, TL, AM,
                    VIB, KSR, FM : Byte ;
                End ;
BNKInsData = Record
                    Mode,
                    Number    : Byte ;
                    Modulator,
                    Carrier   : BNKOp ;
                    MWS,
                    CWS       : Byte;
                End ;
{ Sound Blaster file formats }
SBIFile    = Record
                    ID        : Array [0..3] Of Char ;
                    Name      : Array [0..31] Of Char ;
                    Operators : SBIFormat ;
                End ;
IBKFile    = Record
                    ID    : Array [0..3] Of Char ;
                    Data  : Array [0..127] Of SBIFormat ;
                    Name  : Array [0..127] Of InsName ;
                End ;

Procedure SBI2INSOp (Var O : INSOp ; H : SBIFormat ; T : Byte) ;
{ Converts SBI operator to INS operator. }
Begin
    With H Do Begin
        O.AM : = (Snd [T] AND 128) SHR 7;
        O.VIB : = (Snd [T] AND 64) SHR 6;
        O.EG_TYP : = (Snd [T] AND 32) SHR 5 ;
        O.KSR : = (Snd [T] AND 16) SHR 4 ;
        O.MULTI : = (Snd [T] AND 15 ;
        O.KSL : = (KSLTL [T] AND 192 ) SHR 6 ;
        O.TL : =KSLTL [T] AND 63 ;
        O.AR : = (ARDR [T] AND 240) SHR 4 ;
        O.DR : = ARDR [T] AND 15 ;
        O.SL : = (SLRR [T] AND 240 ) SHR 4 ;
        O.RR : = SLRR [T] AND 15 ;
        O.FB : = (FBFM [T] AND 14) SHR 1 ;
        O.FM : = (FBFM [T] AND 1) XOR 1;
    End;
End;

Procedure INS2SBIOp (Var H : SBIFormat; O : INSOp ; T : Byte) ;
{ Converts INS operator to SBI operator. }
Begin
    With O Do Begin
        H.Snd [T] : = (AM SHL 7) OR (VIB SHL 6) OR
```

```
                    (EG_TYPE SHL 5) OR (KSR SHL 4) OR
                    MULTI;
        H. KSLTL [T] :  = (KSL SHL 6) OR TL;
        H.ARDR [T] :  = (SL SHL 4) OR DR;
        H.SLRR [T] :  = (SL SHL 4) OR RR ;
        H.FBFM [T] :  = (FB SHL 1) OR (FM XOR 1) ;
    End
End;

Procedure ByteToWord (Var I :  INSOp ; O:  BNKOP) ;
{ Converts 8 bits to 16 bits }
Var
    P :  ^ByteArr4ay ;
    Q :  ^WordArray ;
    T :  Byte ;
Begin
    Q :  = Addr (I) ; P :  = Addr (0) ;
    For T :  = 0 To 13 Do
        Q ^ [T] :  = P ^ [T] ;
End;

Procedure WordToByte (Var 0 :  BNKOp ; I :  INSOp) ;
{ Converts 16 bits settings to 8 bits }
Var
    P :  ^ByteArray ;
    Q :  ^WordArray ;
    T :  Byte ;
Begin
    Q :  = Addr (I) ; P :  = Addr (O) ; {copy from Q to P }
    For T :  = 0 to 13 Do
    P ^ [T] :  = Lo (Q ^ [T] ) ; {uses the LSB}
End;

Procedure SBIToIns (Var I :  InsTp) ;
{ Converts bit settings of SBI to bytes for INS. }
Var
    S :  INSFormatP ;
Begin
    If I. Sort < > INSPt Then Begin
        New (S) ;
        SBI2INSOp (S^. Modulator, I.SBI^, 0) ;
        SBI2INSOp (S^. Carrier, I.SBI^, 1) ;
        S^. MWafeSel :  = I.SBI^. WS [0] AND 3 ;
        S^. CWafeSel :  = I.SBI^. WS [1] AND 3;
        S^. Mode :  =0;
        S^. Number :  =0 ;
        Dispose (I.SBI) ;
        I.INS :  = S ;
        I.Sort :  = INSPt ;
```

```
    End ;
End ;

Procedure INSToSBI (Var I :  InsTp) ;
{ Same story but this time the other way round. }
Var
    S :  SBIFormatP ;
Begin
    If I. Sort < > SBIPt Then Begin
        New (S) ;
        INS2SBIOp (S^, I. INS^. Modulator, 0) ;
        INS2SBIOp (S^, I.INS^. Carrier, 1) ;
        S^. WS [0] :  = I.INS^/ MWafeSel ;
        S^. WS [1] :  = I.INS^. CWafeSel ;
        Dispose (I.INS) ;
        I,SBI : = S ;
        I.Sort :  = SBIPt ;
    End;
End;

Procedure CopyIns (S :  InsTp ; Var T :  InsTp) ;
Begin
    If S.Sort = Inspt Then Begin
        New (T.INS) ;
        T.INS ^ :  = S. INS^ ;
        T.Sort :  = InsPt ;
    End
    Else Begin
        New (T.SBI) ;
        T.SVI ^ :  = S.SBI ^ ;
        T.Sort :  =SBIPt ;
    End
End ;

Procedure LoadINS (N :  String ; Var I :  InsTp) ;
Var
    F :  File ;
Begin
    I.Sort :  = INSPt ;
    New (I.INS) ;
    Assihn (F, N) ; Reset (F, 1) ;
    BlockRead (F, I.INS^, SizeOf (I.INS^) ) ;
    Close (F) ;
    I.INS^. MWafeSel :  = 0; I.INS^. CWAFESEL :  =0;
END;

PROCEDURE LOADSBI (N :  String ; Var I :  InsTp) ;
Var
    H :  SBIFile;
```

```
    F :  File ;
Begin
    Assign (F, N) ; Reset (F, 1) ;
    BlockRead (F, H, SizeOf (H) ) ;
    If H > Id = 'SBI' + #$1A Then Begin
        New (I.SBI) ;
        I.SBI ^ :  = H. Operators ;
        I.Sort :  = SBIpt ;
    End;
    Close (F) ;
End ;

Procedure LoadBNK (N :  String ; Var B :  BnkTp) ;
Var
    Header  :  BNKHeader ;
    Name    :  BNKInsName ;
    Data    :  BNKInsData ;
    NamePos :  LongInt ;
    I       :  InsTp ;
    F       :  File ;
Begin
    B + Nil ;
    I. Sort :  = INSPt ;
    New (I.INS) ;
    Assign (F, N) ; Reset (F , 1) ;
    BlockRead (F, Header, SizeOf (Header) ) ;
    If Header. ID = "ADLIB-' Then Begin
        NamePos :  = Heaer.NameStart ;                    {first name}
        While Header. NoUsed>0 Do Begin
            Seek (F, NamePos) ;                           { go to name }
            BlockRead (F, Name, SizeOf (Name) ) ;
            NamePos :  = FilePos (F) ;                    { next name }
            If Name. Used < > 0 Then Begin
                Seek (F, Header. DataStart +              { go to data }
                      Name. IndexNo * SizeOf (Data) ) ;
                BlockRead (F, Data, SizeOf (Data) ) ;
                I. INS^. Mode :  = Data Mode ;
                I.INS^. Number :  = Data. Number ;
                ByteToWord (I.INS^. Modulator, Data. Modulator) ;
                ByteRoWord (I.INS^. Carrier, Data. Carrier) ;
                I.INS^. MWafeSel :  = Data. MWS ;
                I.INS^. CWafeSel : = Data. CWS ;
                AddIns (Name. Name, B, I) ;
            End ;
            Dec (Header. NoUsed)                          { number of instr. }
        End
    End ;
    Close (F) ;
    Dispose (I.INS) ;
```

```
End ;

Procedure LoadIBK (N :  String ; Var B :  VnkTp) ;
Var
    F :  File ;
    IBK ; IBKFile ;
    I :  InsTp ;
    T :  Word ;
Begin
    B :  = Nil ; I. Sort :  = SBIPt ;
    New (I.SBI) ;
    Assign (F, N) ; Reset (F, 1) ;
    BlockRead (F, IBK, SizeOf (IBK) ) ;                    {reads all }
    Close (F) ;
    If IBK. ID = 'IBK'+#$1A Then
        For T :  = 0 To 127 Do                             { 128 instr. }
            If IBK. Name [T] [0] < > #0 Then Begin
                I.SBI^ :  = IBK. Data [T] ;
                AddIns (IBK.Name [T],  B, I) ;
        End ;
    Dispose (I.SBI) ;
End ;

Procedure SaveINS (N :  String ; I :  InsTp) ;
Var
    F :  File ;
    C :  InsTp ;
Begin
    CopyIns (I, C) ;
    SBIToINS (I) ;
    Assign (F, N) ; ReWrite (F, 1) ;
    BlockWrite (F, I.INS^, SizeOf (I.INS^) -2) ;
    Close (F) ;
    RemoveIns (C) ;
End;

Procedure SaveSBI (N :  String ; I :  InsTp) ;
Var
    D :  SBIFile ;
    F :  File ;
    C :  InsTp ;
Begin
    D.Id [0] ; = 'S' ; D.Id [1] :  ='B' ; D.Id [2] :  = 'I' ;
    D.Id [3] :  = #$1A ;
    CopyIns (I, C) ;
    INSToSBI (C) ;
    Assign (F, N) ; ReWrite (F, 1) ;
    D.Operators :  = C.SBI ^ ;
    BlockWrite (F, C.INS^, SizeOf (C.INS^) -2) ;
```

```
    Close (F) ;
    RemoveIns (C) ;
End ;

Procedure SaveBNK (N :  String ; B :  BnkTp) ;
Var
    F          :  File ;
    Header     :  BNKHeader ;
    Name       :  BNKInsName ;
    Data       :  BNKInsData ;
    NumberOf   :  Word ;
    H          :  BnkTp ;
    C          :  InsTpl
Begin
    Assign (F, N) ; ReWrite (F, 1) ;
    NumberOf :  = NumberOfIns (B) ; {determine number of instr. }
    Header. Version :  =1 ;
    Header. ID :  = 'ADLIB-' ;
    Header. NoUsed :  = numberOf;
    Header.NoTotal :  = NumberOf;
    Header. NameStart :  = SizeOf (Header) ;
    Header.DataStart :  = SizeOf (Header ) +NumberOf 8SizeOf  (Name) ;
    BlockWrite (F, Header, SizeOf (Header) ) ;
    H :  = B ;
    Name .IndexNo :  =0 ;                          { start at index 0 }
    Name . Used :  =1 ;                            {all samples are used }
    While H < > Nil Do Begin                       { Process names }
        Name. Name :  = H ^. Name ;
        BlockWrite (F, Name, SizeOf (Name) ) ;
        Inc (Name. IndexNo) ;
        H :  = H^. Next ;
    End;
    H:  =B ;
    While H < > Nil Do Begin                       { process data }
        CopyIns (H^. Ins, C) ;
        SBIToINS (C) ;
        Data.Mode :  =C.INS^. Mode ;
        Data.Number :  = C.INS^. Number ;
        WordToByte (Data.Modulator, C.INS^. Modulator) ;
        WordToByte (Data. Carrier, C.INS^. Carrier) ;
        Data.MWS :  = C.INS^. MWafeSel ;
        Data.CWS :  = C.INS^.CWafeSel ;
        BlockWrite (F, Data, SizeOf (Data) ) ;
        H :  = H^.Next ;
        RemoveIns (C) ;
    End ;
    Close (F) ;
End ;
```

```
Procedure SaveIBK (N : String ; B : BnkTp) ;
Var
    F  : File ;
    IBK : IBKFile ;
    Ins : InsTp ;
    T  : Word ;
    C  : InsTp ;
Begin
    IBK.Id [0] : = 'I' ; IBVK.Id [1] : = 'B' ; IBK. Id [2] : = 'K' ;
    IBK Id [3] : = #$1A ;
    T : =0 ;
    While (T<128) AND (B< > Nil) Do Begin                    { 128 instr }
        CopyIns (B^. Ins, C) ;
        INSToSBI (C) ;
        IBK. Name [T] : = B^. Name ;
        IBK. Data [T] : = C.SBI^ ;
        Inc (T) ;
        B : = B^.Next ;
        RemoveIns (C) ;
    End ;
    While (T <128) Do Begin                                  { file remaining instr. }
        IBK. Name [T] [0] : = #0 ;                           { with #0 }
        Inc (T) ;
    End ;
    Assign (F, N) ; ReWrite (F, 1) ;                         {save IBK }
    BlockWrite (F, IBK, SizeOf (IBKFile) ) ;
    Close (F) ;
End ;

Procedure RemoveIns (Var I : InsTp) ;
Begin
    If I.Sort = InsPt Then
        Dispose (I.INS)
    Else
        Dispose (I, SBI)
    End ;

Procedure RemoveBnk (Var B : BnkTp) ;
Var
    H : BnkTp ;
Begin
    While B < > Nil Do Begin
        H : = B^. Next ;
        RemoveIns (B^. Ins) ;                                {process instr.}
        Dispose (B) ;                                        {remove bank record }
        B : =H ;
    End ;
End ;
```

```
Procedure FindIns (N : InsName ; B : BnkTp ; Var F : BnkTp) ;
Begin
    GoodInsName (N) ;                          { all capitals }
    F : = B ;
    While (F < > Nil) And (N < > F ^. Name) Do
        F : =F^. Next ;
End ;

Procedure DelIns (N : InsName ; Var B : BnkTp) ;
Var
    H,
    V : BnkTp ;
Begin
    V : = Nil;                                 { previous bank record }
    H : = B ;                                  { works with H instead of B }
    GoodInsName (N) ;
    While ( H< > Nil ) And ( N < > H ^. Name ) Do Begin
        V : = H;                               { save previous record }
        H : =H^. Next;
    End ;
    If H < > Nil Then Begin                    { H < > Nil then found }
        RemoveIns (H^. Ins) ;
        If V < > Nil Then
            V^. Next : = H^. Next              { link previous record }
        Else
            B : = H^. Next ;                   { V=Nil then first record }
        Dispose (H) ;                          { remove found record }
    End
End ;

Procedure Addins ( N : InsName ; Var B : BnkTp ; I : InsTp) ;
Var
    H,
    Q,
    V : BnkTp ;
Begin
    V : = Nil;
    H : = B ;
    GoodInsName (N) ;
    New (Q) ;
    CopyIns (I, Q^. Ins ) ;
    Q^. Name : = N ;
    While (H < > Nil) And (N>H^. Name ) Do Begin
        V : =H ;                               { maintain alphabetic }
        H : = H^. Next ;                       { order }
    End ;
    If (H=Nil) Or (N < > H^. Name) Then Begin
        Q^. Next : =H ;                        { insert new record }
        If V < > Nil Then
```

```
                V^. Next :  =Q
            Else
                B:  = Q
        End
    End ;

    Function NumberOfIns (B :  BnkTp) :  Word ;
    Var
        I :  Word ;
    Begin
        I :  = 0 ;
        While B < > Nil Do Begin
            Inc (I) ;
            B :  =B^.Next ;
        End ;
        NumberOfIns :  = I ;
    End ;

    Procedure GoodInsName (Var N :  InsName) ;
    Var
        I :  Byte ;
    Begin
        I :  = 0;
        While (I<9) And (N [I] < > #0 Do Begin
            N [I] :  = UpCase (N [I] ) ;
            Inc (I)
        End ;
        While (I<9) Do Begin
            N [I] :  = #0 ;
            Inc (I)
        End
    End ;

    Procedure StrToName (S :  String ; Var N :  InsName) ;
    Var
        T :  Byte ;
    Begin
        T :  = 0;
        While (T<8) AND (T<Length (S) ) Do Begin
            N [T] :  = UpCase (S [T+1] ) ;
            Inc (T)
        End ;
        While (T<9) Do Begin
        N [T] :  = #0 ;
        Ind (T) ;
    End ;
End ;
End.
```

程序清单7.2 转换BNK文件为IBK文件的 Pascal 程序

下一个清单使用库函数来转换BNK文件为IBK文件。这仅是一个示例--它可使用许多扩展的选项来加以扩充，这与你需要作什么事有关。使用这个程序，你可以建立你自己的设备组管理器。

```
Program BNKtoIBK;
Uses Instr ;

Var
    Bank :  BNKTp ;

Begin
    WriteLn ;
    If ParamCount < > 2 Then Begin
        WriteLn ('Use BNK2IBK name.BNK name.IBK , ') ;
        WriteLn ;
        Halt (1) ;
    End ;
    WriteLn ('Reading :  ', ParamStr (1), '.' ) ;
    {reads one format }
    LoadBNK (ParamStr (1), Bank) ;
    WriteLn ('Writing :  ', ParamStr (2), '.' ) ;
    { writes other format }
    SaveIBK (ParamStr (2), Bank) ;
    WriteLn ;
    RemoveBnk (Bank) ;
End.
```

播放音乐

本部分讨论Sound Blaster和AdLib的音乐格式。驱动程序SBFMDRV.COM (Sound Blaster)和SOUND.COM（AdLib）用来播放音乐。由于两个驱动程序都在使用定时器中断，所以你不必关注音乐播放本身，而运行你自己的程序。

播放CMF格式音乐

CMF格式是由 Sound Blaster 提供的音乐格式。音乐文件具有扩展名 .CMF。在本段中，首先讨论CMF文件的结构，其次研究驱动程序的作用，最后是如何使驱动程序来播放音乐。

一个CMF文件由三部分组成：

- 文件头
- 设备定义块
- 实际的音乐数据

注解 **下面所述的CMF格式是1.10版本。**

文件头 文件头包含常用的信息，它们有一些在播放时使用。文件头具有下列结构：

字节	设 置
00H-03H	文件ID。包含正文CTMF (没有EOF字节1AH)。
04H-05H	格式版号。第一字节为点前面的数，第二个字节为点后面的数。对1.10版，这两个字节的值为 01H 和 0AH。
06H-07H	设备定义的绝对起点，它是从文件的开头开始。
08H-09H	音乐数据的绝对起点。
0AH-0BH	对四分之一音符的时钟节拍数。时钟节拍数的说明，参见下一域。假定有每秒96节拍和拍子是120 (每分钟的四分之一音符)。在此情况下，此域有48的值，这意味着四分之一音符持续半秒。顺便说一句这是缺省值。
0CH-0DH	每秒的时钟节拍数。此值确定定时器中断产生的频度。如果在此域中包含值96，这就意味着一秒产生96次中断。此值越高，则播放时精度越好。但是，你必须自己调整音乐段的新的速率。
0EH-0FH	音乐标题的绝对起点。必须以0字节结尾。如果此字为0值，则无标题。
10H-11H	作曲器名的绝对起点。此处，标题的规则同样适用 (见上面)。
12H-13H	注解的绝对起点。标题的规则同样也适用于此。

注解 **我们建议对标题、名字和注解不要使用多于32个字母 (包括0字节)。这样，你就避免了支持软件的问题。**

14H-23H	声部数据表。SBFMDRV驱动程序支持16种声部。在此表中，你可以指定一种声部是否使用。若使用，则该值为1；否则，其值为0。
24H-25H	所使用的设备数。CMF文件包含了有关使用各设备的信息。此域指出使用设备的数量。
26H-27H	基本拍： 这是通用的拍子。
28H-???	各种正文 (标题、作曲器、注解) 可放于此处。

设备块 在文件头之后为设备块，它是对每个设备的标准数据。此数据由16个字节组成，具有同IBK和SBI格式中的16个字节 (开始于偏移值24H)相同的结构。文件头包含所定义的设备的数量和块的起点。块的大小为16倍所用设备的数量。

音乐块 最后是音乐块，它为实际的歌曲数据。此块具有MIDI格式 (第十章叙述)。在

此格式中，三类事件被定义：MIDI事件、系统专有事件和转移（meta）事件。事件是在某个时刻所发生的一个动作。CMF格式只支持头两种事件，即MIDI和系统专有事件。

定义的MIDI事件

下列其他的MIDI事件也被定义：

事件	用　途
66H，M	参量M包含1到127间的值。M的值被赋给状态字节 (状态字节的使用在本节后面讨论)。
76H，M	参量M确定播放模式。如果此参量为0值，使用旋律模式。在此模式中，有几种设备可用，且分布于16个MIDI通道。如果M为值1，则节奏模式被使用。在此模式中，六种普通设备和5种节奏设备可用。这5种节奏设备的分配示于表7.4中的MIDI通道。

表7.4 5种节奏设备

通道	设备类型
12	低音鼓
13	小鼓
14	印度长鼓
15	铙钹
16	高帽

在第2.0版，SBFMDRV驱动程序也支持下面两种事件：

事件	用　途
68H，M	引起所有随后的音调升高半音的M/128，在此事件后，所有的音符的音稍高一点。
69H，M	同前面事件的作用相同，但是这次所有音符降低M/128。

提示　此块的大小等于CMF文件的尺寸减去音乐块的起点。

SBFMDRV驱动程序

SBFMDRV驱动程序使用定时器播放上面叙述过的CMF格式。驱动程序通过中断来编程，而且必须事先由运行SBFMDRV.COM程序来装入。此程序安装驱动程序在未用中断矢量128和191间。正文FMDRV被放于所选中断矢量段中偏移259或103H处。这可用来寻找驱动程序的正确中断。

注解　关于FMDRV如何用来找到正确中断的例子见清单7.3。

下面的子例程用 Pascal 定义：

Procedure CallFMDRV (BX: Word; Var Ax, Cx, Dx, DI: Word)

此处

- 变量 AX，BX，CX，DX和DI表示前述的各相应寄存器。
- 变量BX确定所选的功能，据此功能而使用一个或多个寄存器。

注解　只有功能14（1.21版支持）使用寄存器DI。在你不使用此功能的情况下，你可以调整上面所述各功能，使得它们只用AX，BX，CX和DX作为输入。

所有的寄存器和标志以汇编程序存贮。寄存器AX和DX是唯一能被改变的寄存器，而且通常包含某种输出结果。

Sound Blaster pro驱动程序　Sound Blaster pro随附一驱动程序的新版。此版含有两个特别的功能，并通过FM芯片播放音乐，所以某些设备能在左边听到，而另一些能在右边听到。驱动程序也能工作在标准的 Sound Blaster。随Sound Blaster提供的驱动程序是1.02版。随Sound Blaster Pro提供的是1.21版。

注解　在下面的功能描述中，输入和输出用寄存器来指定。这些寄存器对应于在Pascal所定义的变量。

功能0，　驱动程序版本

输入　　BX=0

输出　　AX=版本

此功能在AX中返回驱动程序版本。高字节（AH）包含点前面的值；低字节（AL）包含点后面的值。

功能1，　设置状态字节地址

输入：　　BX=1

DX：　AX=段：　状态字节偏移

输出：　　无

状态字节指明驱动程序正在作什么。如果此字节包含值0，驱动程序未做什么事，所以也就没有东西被播放。在一音乐段被播放时，状态字节包含值255或FFH，即使当音乐暂停时，也是这样。使用标记事件，此字节能装入它自己的值。这样，你可以保持跟踪播放的过程，而且校正动作能在正确时刻完成。此功能也允许你设置状态字节的地址。

功能2，　设置设备表

输入：　　BX=2

CX=设备的数量

DX：　AX=段：设备表偏移

输出：　　无

你使用此功能来为设备数据设置地址。这些设备数据用来编程FM芯片。

此表的结构等同于在CMF格式中的设备表： 每种设备16个字节。若此功能被调用，则最后一个设备后跟的是用于设置下一个MIDI通道的第一个设备的数据。驱动程序对设备数据具有它自己的内部表，如果本功能没有被调用，就使用此表。设备的数量不能超过128。

功能3， 设置时钟频率

输入： BX=3
AX=1193180/频率(Hz)

输出： 无

如果驱动程序不是正在播放音乐，则定时器中断被调用，每秒的次数由此功能设置。通常此数为18.2。为设置此值，你可以赋值65535或FFFFH于AX。假定你需要定时器中断每秒被调用36次，此时AX的值为1193180/36=33144。

警告 我们建议你不要使用功能3，因为它能改变时钟的功能，而引起不可预料的结果。

功能4 设置播放时的频率

输入： BX=4
AX=1193180/频率 (Hz)

输出： 无

在驱动程序播放一段音乐时，调用定时器中断以得到确定的时间段，而程序员不必考虑此事务。使用此功能，你可以确定定时器每秒必须被调用多少次。在此处你必须设置的值可以用在CMF文件中偏移0CH处的信息来确定。

例如：如果值96H在偏移0CH-0DH处，那么AX具有值1193180/12428或308CH。你可以使用此功能，用调整频率的办法来改变播放速率。

功能5 变调音乐

输入： BX=5
AX=半音数

输出： 无

你可以使用此功能来变调音乐高或低若干半音。AX含变调的半音数。使用负值变调为较低的音谱，或使用正值变调为较高的音谱。但是，在Pascal中，AX为.Word类型，如设为负值，你可以使用公式65535-x，结果表示负x。

功能6 播放音乐

输入： BX=6
DX：AX=段：音乐数据偏移

输出： AX=0，音乐被播放;1，此时另外的乐段被播放。

在此功能被使用前，你可以用前面的功能改变几个设置。音乐数据由CMF格式音乐块组成。如果此功能执行成功，AX具有值0，且状态字节具有值255。

功能7　　停止播放

输入：　　BX=7

输出：　　AX=0，音乐已停止；1，没有音乐被播放。

此功能允许你停止被播放的音乐。AX包含返回码，在此功能之后，状态字节有值0。

功能8　　复位驱动程序

输入：　　BX=8

输出：　　AX=0，无错误；1，音乐被播放。

此功能设置FM芯片的音量为0，并设置内部设备表。如果一段音乐正在被播放，前面叙述的功能必须被先调用。在你退出程序并返回DOS之前，必须调用此功能。

功能9　　暂停播放

输入：　　BX=9

输出：　　AX=0，音乐被暂停；1，没有音乐被播放。

此功能允许你暂停播放音乐。它不改变状态字节。在此功能之后，你可以使用下一功能来继续播放或使用功能7来停止播放。

功能10　　继续播放

输入：　　BX=10

输出：　　AX=0，音乐被继续；1，无暂停。

音乐段继续。

功能11　　设置用户程序

输入：　　BX=11

DX：AX=段：子例程偏移

输出：　　无

当其驱动程序遇到一系统专有事件时，用户程序被调用。在调用时，ES：BX指向紧跟此事件的字节。程序本身必须保存所有使用的寄存器，并以RETF返回驱动程序。此后，驱动程序将跳过此事件并继续播放。如果你要用户程序不起作用，则DX和AX必须含0值。

功能12　　确定MIDI通道设置的地址

输入：　　BX=12

输出：　　DX：AX=段：MIDI通道设置偏移

此功能为1.10版所支持。它返回MIDI通道设置的起始地址。在此功能被调用后DX：AX指向一16字节的表，其每个字节属于相应的MIDI通道。字节指示MIDI是否必须是可听的。如果此字节具有值0，此MIDI通道是哑的。如果字节包含值1，则所有的音符被播放。你可以用写此表来打开或关闭某个通道。例如：　用此法你可以只听节奏部分。如果驱动程序被复位，所有的字节重新装入值1，所以所有的MIDI通道被重新生成。

功能13　询问音乐块中的现行地址

输入：　BX=13

输出：　DX：AX=段：在音乐块中的偏移现行地址。

此功能为1.10版所支持。如果音乐块被处理，此功能允许你跟踪驱动程序，因为它是返回现行演奏程序所在的音乐块中的地址。DX：AX指向下一个MIDI事件。例如：此功能指示演奏的进度。

功能14　设置淡化效果

输入：　BX=14
AX=起始音量（百分率）
DX=结束音量（百分率）
CX=改变强度
DI=步距

输出：　无

此功能为1.21版所支持。使用此功能，你可以设置淡化。所谓淡化就是音量以恒定速度变大或变小。例如，用此法你可以产生淡出：在乐曲结尾，音乐慢慢减弱，但并不终止。在日常的音乐中（无线电广播或CD），你经常会发现此类效果。相反的一种效果称为淡入，能用在一首音乐的开始。使用淡入，声音逐渐地变得能听见。

变量（寄存器）AX和DX两者包含0和100间的值。AX和DX表示确定最后音量的百分率。0表示最小音量（无声）设置，而100%为最大音量设置。变量CX所包含的百分数，是AX用它来增加或减少音量，直至等于DX中的百分率。变量DI的值指示时　间区段，它以时钟节拍（滴嗒）来计算。例如：假定播放频率被设置为110Hz（每秒110次）。如果DI有值110，每秒钟百分率按CX增加或减少（直至它等于DX）。

例如：为产生一淡出，你可设置变量如下：

AX=100，　开始于现行标准设置
DX=0，　淡出到无声
CX=5，　减少百分率为5个百分点
DI=10，　每10次时钟节拍做一次，所以音乐是安静的，且在（100/5=20)*10/播放频率之后停止。

为获得淡入，交换AX和DX的值。

功能15　设置重复次数

输入：　BX=15
AX=音乐的重复次数

输出：　AX=老的设置

此功能为1.21版所支持。它允许连续地播放一段音乐几次。AX含重复次数(0=不重复)。若AX有值65535或0FFFFH，音乐则无限地重复，直至由程序停止。在调用功能8后，重复次数设置为0。

播放CMF音乐

本节叙述你如何能播放一CMF文件。此处是包含播放程序单元的Pascal清单。

为播放一CMF文件，首先必须找到驱动程序。为寻找驱动程序，在中断128和191间进行搜索。如果正文FMDRV位于偏移259（103H）处，则你已找到正确的中断。

往后，CMF文件可以被处理和播放。按下列顺序读文件：

1. 打开文件;
2. 读文件头和提取正确的数据;
3. 移动文件指针到设备块;
4. 读设备块;
5. 移动文件指针到音乐块;
6. 读音乐块;
7. 关闭文件。

现在你可以用下列步骤开始播放音乐：

1. 复位驱动程序;
2. 设置状态字节地址;
3. 设置设备表;
4. 设置播放频率;
5. 设置音乐表并开始播放。

如果需要，你可以使用淡化功能(14)和重复功能(15)。

播放一CMF文件是舒服的。在播放时，所有的设备都在表中找得到。如果你改变在表中的设备，你就会在音乐中听到其结果。但是，要听到设备的改变结果是要花一些时间的，因为FM芯片首先必须以设备数据编程，而这在驱动程序遇到正确的MIDI事件之前是不会发生的。

提示　如果在开始播放之前你修改设备表，则所改变的设备会立即被使用。

清单7.3含有一库，它搜索驱动程序，然后读和播放一CMF文件。这个库含有用早些时候给出的INSTR库工作的函数。CallFMDRV函数已经定义了几个驱动程序函数。你可以加你自己的函数到这些函数。例如：　你可以加入暂停和继续播放的函数。

注解　有关使用本书中的程序清单的建议，见绪言中的“关于程序清单的说明”。

清单7.3　播放CMF文件和改变设备的Pascal程序单元。

```
Unit PlayCmf ;
{ Unit supporting the CMF format }

Interface
Uses Instr ;
Type
    CMFTp = Record { contains the required information }
```

```
                NumberOfIns,
                MusSize,
                ClockFreq : Word ;
                InsTable,
                MusTable : Pointer ;
            End ;
Var
    StatusByte : Byte ;

{ find the driver and set the status word address }
Function InitFMDriver : Boolean ;
{ interface routine }
Procedure CallFMDRV (BX : Word ; Var AX, CX, DX, DI : Word) ;
{ load .CMF file }
Procedure LoadCMF (C : String ; Var C : CMFTp) ;
{ remove .CMF type from memory }
Procedure RemoveCMF (Var C : CMFTp) ;
{ play a loaded .CMF file }
Procedure PlayCMF (C : CMFTp) ;
{ stop playing }
Procedure StopCMF ;
{ Replace instrument data with I, T is the instrument number }
Procedure ReplaceIns (T : Word ; I : InsTp; C : CMFTp) ;
{ Copy all instrument data to a bank }
Procedure InsTabToBnk (C : CMFTp ; Var B : BnkTp) ;

Implementation
Uses Dos ;

Type
    CMFHeader = Record { for the file }
                    ID : Array [0..3] Of Char ;
                    Version,
                    InsStart,
                    MusStart,
                    QuarterNote,
                    Frequency,
                    TitStart,
                    CompStart,
                    CommStart : Word ;
                    ChannelG : Array [1..16] Of Byte ;
                    NumberOfIns,
                    BasicTempo : Word ;
                End ;
    SBIList = Array [1..128] Of SBIFormat;

Var
    FMIntr : Byte ; { the interrupt found }
```

```
Function InitFMDriver : Boolean ;
Type
    HelpRc = Record
                Filling : Array [0..2] Of Byte ;
                ID : Array [0..4] Of Char ;
             End ;
Var
    I                : Byte ;
    V                : ^HelpRc ;
    H                : Pointer ;
    F                : Boolean ;
    AX, CX, DX, DI: Word ;
Begin
    I : =128 ; F : =False ;                          { start at int 128 }
    Repeat
        GetIntVec (I,H) ;                            { H contains vector }
        V : = Ptr (Seg (H^) , $100) ;                { V points at COM }
        If V^ . ID = 'FMDRV' Then                    { testing for text }
            F : = True                               { found }
        Else
            Inc (I) ;                                { next interrupt }
    Until (I>191) OR F ;
    FMIntr : = I ;                                   { store interrupt }
    If F Then Begin                                  { if found, set }
        AX : = Ofs (StatusByte) ;                    { Status byte }
        DX : = Seg (StatusByte) ;
        CallFMDRV (1, AX, CX, DX, DI) ;
    End ;
    InitFMDriver : = F ;                             { return flag }
End ;

Procedure CallFMDRV (BX : Word ; Var AX, CX, DX, DI : Word) ;
Var
    R : Registers ;
Begin
    R.Ax : = AX ; R.BX : =BX ; R.CX : =CX ;          { set reg.}
    R.DX : = DX ; R.DI : = DI ;
    Intr ( FMIntr, R) ;                              { call driver }
    AX : = R.AX ; CX: = R.CX ;                       { set reg. with new }
    DX : = R.DX ; DI : =R.DI ;                       { values }
End ;

Procedure LoadCMF (N : String ; Var C : CMFTp) ;
Var
    Header : CMFHeader ;
    F : File ;
Begin
    Assign (F,N) ; Reset (F,1) ;
    BlockRead (F, Header, SizeOf (Header) ) ;
```

```
    If Header. ID = 'CTMF' Then Begin
        C.NumberOfIns : = Header.NumberOfins ;  {only the }
        C.ClockFreq : = Header.Frequency ;  { data required }
        C.MusSize : = FileSize (F) Header.MusStart ;
        GetMem (C.InsTable, C.NumberOfIns*16) ;
        GetMem (C.MusTable, C.MusSize) ;
        Seek (F, Header.InsStart) ;  { read tables }
        BlockRead (F,C.InsTable^, C.NumberOfIns*16) ;
        Seek (F, Header. MusStart) ;
        BlockRead (F, C.MusTable^, C.MusSize) ;
    End;
    Close (F) ;
End;

Procedure RemoveCMF (Var C : CMFTp) ;
Begin
    FreeMem (C.InsTable, C.NumberOfIns *16) ;
    FreeMem (C.MusTable, C.MusSize) ;
End ;

Procedure PlayCMF (C : CMFTp) ;
Var
    AX, CX, DX, DI : Word ;
Begin
    CX : = C.NumberOfIns ;                    { set ins table }
    AX : = Ofs (C.InsTable^) ;
    DX : = Seg (C.InsTable^) ;
    CallFMDRV (2, AX, CX, DX, DI) ;
    AX : = Trunc (1193180/C.ClockFreq ) ;     { frequency }
    CallFMDRV (4, AX, CX, DX, DI) ;
    AX : = Ofs (C.MusTable^) ;                { start music }
    DX : = Seg (C.MusTable^) ;
    CallFMDRV (6, AX, CX, DX, DI) ;
End ;

Procedure StopCMF ;
Var
    AX, CX, DX, DI : Word ;
Begin
    CallFMDRV (7, AX, CX, DX, DI) ;           {stop music }
End ;

Procedure ReplaceIns (T : Word ; I : InsTp ; C : CMFTP) ;
Var
    K : InsTp ;
    L : ^SBIList ;
Begin
    CopyIns (I, K) ;                          { make safety copy }
    INSToSBI (K) ;
```

```
    L : = C. InsTable ; { copy data }
    L^ [T] : = K. SBI^ ; { to table }
    RemoveIns (K) ; { remove copy }
End ;

Procedure InsTabToBnk (C : CMFTp; Var B : BnkTp) ;
Var
    I  : InsTp ;
    T  : Word ;
    S  : String ;
    N  : InsName ;
    L  : ^SBIList ;
Begin
    B : = Nil ;
    I. Sort : = SBIPt ; New ( I.SBI) ;                    { create instrument }
    L : = C. InsTable ;
    For T : = 1 To C. NumberOfIns Do Begin
        Str (T, S) ;                                      { use dummy name }
        StrToName ('INS'+S, N) ;
        I.SBI^ : = L^ [T] ;
        AddIns (N, B, I) ;
    End ;
End ;
End.
```

程序清单7.4　使用PLAYCM程序单元的Pascal例

下面的清单包含了一个程序，它完成装入和播放一CMF文件。如果驱动程序允许的话，则程序指出播放前进了好远。

```
Program CMFTest ;
Uses PlayCmf, CRT ;

Var
    Song            : CMFTp ;                         {CMF music }
    Version,
    AX, CX, DX, DI  : Word ;
    Ch              : Char ;                          {key press }
    StartAddr ,                                       { for function 13 }
    RelatAddr       : LongInt ;
    Done            : Real ;

Begin
    WriteLn ;
    If ParamCount < 1 Then Begin                      { no name }
        WriteLn ('Error : use CMFTest file.ext !') ;
        WriteLn ;
```

```
    Hale (1)
End ;
If Not InitFMDriver Then Begin                    { no driver }
    WriteLn ('Error : CMF driver not found !') ;
    WriteLn ;
    Halt (1)
End ;
CallFMDRV (O, Version, CX, DX, DI) ;              { determine version }
LoadCMF (ParamStr (1) , Song) ;                   { load file }
WriteLn ('Playing', ParamStr (1), '..') ;
If Version > 9 Then
    WriteLn ;
StartAddr : = LongInt (16*Seg (Song.MusTable^) ) +
            Ofs (Song.MusTable^) ;                { start music }
PlayCMF (Song) ;
    Repeat
        If Lo (Version) >9 Then Begin             { function 13 }
            CallFMDRV (13, AX, CX, DX, DI) ;      { allowed }
            RelatAddr : = LongInt (DX*16) +Ax-StartAddr ;
            Done : = 100*RelatAddr/Song. MusSize ;
            GotoXY (1, WhereY) ;
            Write ('Done : ' , Done : 7:2, '%.') ;
        End ;
    Until (StatusByte =0) Or KeyPressed ;
    If KeyPressed Then                            { key pressed }
        Ch : = ReadKey ;
    WriteLn ; WriteLn ;
    StopCMF ;                                     { stop playing and }
    CallFMDRV (8, AX, CX, DX, DI) ;               { reset driver }
End.
```

播放ROL格式音乐

ROL格式是由AdLib提供的音乐格式。在AdLib中，音乐文件具有扩展名 .ROL。本节叙述ROL文件的结构，揭示SOUND.COM驱动程序的功能，并说明用此驱动程序如何产生音乐。

ROL格式的结构

ROL文件的结构较CMF文件为复杂。ROL文件以包含一般信息的文件头开始，在此处你可以找到由视见作曲器（由AdLib提供的音乐编辑器）使用的信息。文件头后面是有关音乐的信息。在本书中，我们叙述0.4版。ROL文件也包含有几个按IEEE格式编码的浮点数。此种格式相应于 Pascal 中的单精度类型。

文件头 文件头有下列结构：

字节	设　置
00H-01H	格式版本号。对0.4版，点号前的数为0。

02H-03H	格式版本号。对0.4版，点号后的数为4。
04H-2BH	未用。
2CH-2DH	每秒时钟节拍数。
2EH-2FH	每小节的四分之一音符数；此功能用于视见作曲器。
30H-31H	Y轴比尺；由视见作曲器使用。
32H-33H	X轴比尺；由视见作曲器使用。
34H	未用；必须为0。
35H	播放模式。如果此字节具有值0，则音乐以节奏模式写。如果此字节包含值1，则音乐以旋律模式写。对第一种模式，定义了11种声部；对第二种模式，定义了9种声部。
36H-C4H	未用；开始90个字节必须是0。
C5H-C8H	基本速度。四个字节包含一浮点数。

音乐数据 其后是音乐的数据。这些数据也由含有关于事件（在某个时刻出现的变更）的信息的几个部分组成。首先是速度事件：

字节	设　置
00H-01H	此值指示事件数和两个随后的域的重复数。
02H-03H	速度事件的时间，以时钟节拍数计。
04H-07H	速度乘法器。用于计算新的速度。新的速度等于此数乘以基本速度。在此四个字节中的数为一浮点数，其值在0.01和10.00之间。
08H-...	速度事件的任何数量。

设备数据 其后是关于FM芯片的每种设备声部的相同信息。同播放模式有关，有9个或11个声部。对每一个声部，ROL格式包含了下面所述的域。首先是对第一种声部的全部域，其次是第二种声部的域，等等。第一部分包含有关于要播放的音符的信息。

字节	设　置
00H-0EH	未用。
0FH-10H	所有音符的持续时间的和，为时钟节拍数加1。为读后面两个域，此域必须被使用。若此域有值0，后面两个域不存在而此声部将不被播放。
11H-12H	音符值。如果它含值0，则意味着无声。否则，此域必须包含2和107间的值。值60表示中央C。更高或高低的值表示高半调（升C调）或低半调（B）。
13H-14H	音符持续时间，以时钟节拍计数。
15H-...	前面两个域的任意重复，只要音符的持续时间之和小于总和（这在偏移0FH处给出）。

在此之后，是关于在此声部中所使用的设备的信息：

字节	设　置
00H-0EH	未用。
0FH-10H	定调事件数。此16位值包含下两个域的重复数量。
11H-12H	定调事件时间，以时钟节拍计数。
13H-16H	定调变化。这四个字节包含一0.0和2.0间的浮点数。此值设置变调。此功能以下列公式来说明： New Pitch = Pitch + 0.5 * (Value - 1) （注：新调 = 音调 + 0.5 *（值 - 1）） 定调我们是指五线谱上的音符。此音符大多可升高或降低半调(D变成升D调或升C调)。当这四个字节的值大于1.0时，定调升高，如此值小于1.0，则定调降低。
17H-...	前两个域的任意重复。

ROL和CMF的比较　很清楚，ROL格式较CMF格式要复杂（虽然我们没有叙述CMF格式音乐块的结构）。主要的不同在于CMF文件含有关各种设备的信息，而且CMF也同时使用两种播放模式（节奏和旋律）。在ROL文件中，只有设备名可以利用，而播放模式事先是不知道的。

使用设备名的优点是可以调整设备而无需变动ROL文件。进一步，设备数据的格式可以改变而不影响ROL格式。使用名字的缺点是你想播放ROL文件时，除了ROL文件外，还必须有设备组或几个设备文件。特别是如你使用自己定义的设备时，这可导致相当大的设备组文件。

用Sound驱动程序播放音乐

为了使用Sound驱动程序，你必须事先把它放于存贮器中，为放 Sound 驱动程序到存贮器，运行命令SOUND.COM。SOUND.COM装入它自己，并使之常驻内存。你可以使用中断编程驱动程序--即：使用中断101和56H。正如SBFMDRV驱动程序那样，你可以检验驱动程序是否存在。中断程序的偏移由正文SOUND_DRIVWR_ADLIB为前导，后跟没有意义的三个字节。该正文的前面是包含驱动程序版本的两个字节。两个数按BCD码构造。

注解　检验文本和返回驱动程序版本号的函数功能举例见清单7.5。

驱动程序需要两块数据作为输入：功能号和数据其余部分地址的指针，这在不同调用可能是不同的。功能号是放在寄存器SI中，而ES：BX指向剩下的输入数据。这些数据由一个或多个整数组成，所以ES：BX是指向一个或多个整数的数组。最大的有五个整数，所以下面的Pascal函数可以使用：

```
procedure  CallSound (F, P0, P1, P2, P3, P4:  Integer)
```

在调用此函数后，往往返回一结果。此结果在寄存器AX中，并被放入SNDIOResult变量中。

驱动程序有一缓冲区（它的大小在装入时可变），各个运算量都在其中，直到它们必需被处理。这种方法的好处在于整个音乐段能以一整体传送给驱动程序，而后无须你的干预，驱动程序就播放此音乐。

下面是对Sound驱动程序的23个功能的讨论。

注解　Sound驱动程序有几个版本可用。在此书中，我们讨论1.51版。

功能0，初始化驱动程序和FM芯片

输入：　无

输出：　无

此功能初始化驱动程序。相对音量被置为100%，所以凡是声音皆为最大音量。而且，FM芯片被设置为旋律模式，而所有的9种设备皆以电子钢琴的数据设置。缓冲区为空，而速度设为每分种90个四分之一音符。波形类型的使用被关掉，且定调的差值设为一个半调（这两个功能以后说明）。

功能1，未知

AdLib文档没有提供关于此功能的信息，它也许是内部功能。

功能2，设置音乐的起始点

输入：　P0=起始点分子

P1=起始点分母

输出：　无

分数P0/P1确定音乐的起始点，对其他功能所指的时间，均以此时间为基准进行。换句话说，即是分数P0/P1被加到所给定的时刻。这样，播放可在某个时刻开始。如立即开始播放，P0和P1必须分别为0和1。

注解　很明显，分子P0不能含0值。这也适用于其他的分子。

功能3，启动或停止播放

输入：　P0=动作

输出：　无

你调用此功能以启动驱动程序。如果P0有值1，驱动程序开始处理在缓冲区中的动作。如果P0有值0，则处理停止。因此，此功能也可用来暂停和继续播放。在播放期间，被处理过的动作从缓冲区去掉，所以新功能可以被加到缓冲区中去。

功能4，询问当前的驱动程序状态

输入：　无

输出：　SNDIOResult=当前动作

此功能确定驱动程序是否正在处理缓冲区。如果是，在变量SNDIOResult包含值1。否则，它包含值0。此功能也允许你确定驱动程序是否已播放了所有的内容且已到达音乐的结尾。

功能5，清除缓冲区

输入： 无

输出： 无

此功能指令驱动程序断掉所有的声部，且清除缓冲区。此后，你可以重新开始播放。

功能6，设置播放模式

输入： P0=播放模式

输出： 无

此功能选择旋律或节奏模式。当P0为值0时，旋律模式被选择；当此值为1时，节奏模式被选择。在此功能被调用后，相对音量被重新设置为100%，而所有声部被设在缺省钢琴。定调（见下）被重新设在一个半音调。因此，此功能必须在其他功能之前必调用。

注解 P0值同ROL格式中的值相反！

功能7，询问现行模式

输入： 无

输出： SNDIOResult=现行播放模式

此功能返回由前述功能设置的模式。在调用此功能后，SNDIOResult包含现行设置：O为旋律模式，或1为节奏模式。

功能8，设置相对音量事件

输入： P0=相对音量的分子

P1=相对音量的分母

P2=时间的分子

P3=时间的分母

输出： SNDIOResult=缓冲区结果

使用此功能，你对现行选择的声部设置一音量事件。时间由除数P2/P3确定，而相对音量以值P0/P1设置。值P0/P1不可能超过1。设备所建立的强度被乘以此音量。由此功能确定的值作为一种通用音量设置。正如对另外的设置事件的功能那样，如果事件已成功地放于缓冲区的话，则SNDIOResult具有不等于0的值。但是，如果SNDIOResult包含0值，则缓冲区是满的。如果驱动程序不是正在处理 缓冲区，则你就必须给它一指令来进行处理（用功能3）。在缓冲区中所存动作处理期间，新动作的单元 ，将成可用的，而调用就能成功地重复。

功能9，设置速度事件

输入： P0=速度

P1=时间的分子

P2=时间的分母

输出： SNDIROesult=缓冲区结果

此功能允许你设置一速度事件。这是影响所有声部的一个一般事件。P1/P2再一次构成时间，而值SNDIOResult具有同前一功能相同的意义。

功能10，变调音乐

输入： P0=变调值

输出： 无

正如CMF驱动程序一样，音乐能够升高或降低若干半调。如果P0为正值，音乐升高变调，如果他含负值，则降低变调。此功能可以为下述公式表示：

播放音符=实际值+P0 (Note to play = real value + po)

功能11，询问变调值

输入： 无

输出： SNDIOResult=变调值

此功能允许你询问现行值。

功能12，选择活动声部

输入： P0=活动声部

输出： 无

大多数事件都能分别对每个声部设置。你使用此功能来选择一个声部。如果驱动程序被设置在旋律模式，则P0的值必须在0和8之间。对节奏模式，此值必须在0和10之间。在此功能之后，对声部的事件可被设置。对节奏设备的各数定义在表7.5中。在旋律模式中，对普通设备，声部0到8可用。

表7.5 节奏设备的各数值

声部	设备类型
0...5	普通设备
6	低音鼓
7	小鼓
8	印度长鼓
9	铙钹
10	高帽

功能13，返回活动声部

输入： 无

输出： SNDIOResult=活动声部

此功能允许返回活动声部。

功能14，以播放时间设置音符事件

输入： P0=定调
P1=播放时间的分子
P2=播放时间的分母
P3=总时间的分子
P4=总时间的分母

输出： SNDIOResult=缓冲区结果

此功能为加一个活动声部的音符事件到缓冲区。音符以P0定调播放。播放时间由除数P1/P2确定，而总的时间由除数P3/P4结果设置。如果总时间大于播放时间，则当播放停止时，就为无声。当前一音符的总时间已过去时，音符被播放。变量SNDIOResult包含上述结果。P2和P4的值可在1和255之间。定调（P0）包含-48和47之间的值，此处0是钢琴的中部C。表7.6示出各音符值。

表7.6 音符值

音符	值							
C	-48	-36	-24	-12	0	12	24	36
C#	-47	-35	-23	-11	1	13	25	37
D	-46	-34	-22	-10	2	14	26	38
D#	-45	-33	-21	-9	3	15	27	39
E	-44	-32	-20	-8	4	16	28	40
F	-43	-31	-19	-7	5	17	29	41
F#	-42	-30	-18	-6	6	18	30	42
G	-41	-29	-17	-5	7	19	31	43
G#	-40	-28	-16	-4	8	20	32	44
A	-39	-27	-15	-3	9	21	33	45
A#	-38	-26	-14	-2	10	22	34	46
B	-37	-25	-13	-1	11	23	35	47

注解 对ROL文件中的各音符值，在它们能送给驱动程序之前，必须减60。

功能15，设置音符事件

输入： P0=定调
P1=持续时间的分子
P2=持续时间的分母

输出： SNDIOResult=缓冲区结果

当其总的时间等于播放时间时，你可以使用此功能。它如前面的功能一样工作。当你用它时，总的时间和播放时间被设置为相同值（持续时间）。

功能16，设置设备事件

输入： P0=设备数据的偏移

P1=设备数据的段
P2=时间的分子
P3=时间的分母

输出： SNDIOResult=缓冲区结果

用此功能你设置一事件，此事件为活动声部设置指定设备。P0：P1指向一整数缓冲区，该缓冲区包含有调制器和载波的信息。缓冲区包含26或28个整数，这同功能23有关。它的结构与在INS格式中所用的结构相同。26个整数之后可跟以另外2个整数，它们分别是关于调制器和载波的波形的信息。除数P2/P3确定事件时间。包含设备数据的缓冲区必须保持该正确数据直到事件之后；然后缓冲区才可以被修改。

功能17，设置音调改变事件

输入： P0=八分音差 (必须是0)
P1=改变的分子
P2=改变的分母
P3=时间的分子
P4=时间的分母

输出： SNDIOResult=缓冲区结果

此功能允许你调整音符的频率。它在P3/P4时刻为活动音符设置一事件。在此事件被处理之后，P1/P2半调部分被加到音符，或从音符减去 (由P1的符号决定)。此后频率被计算，这样每个音调发音就稍高或稍低。使用功能22你可以设置更大的范围和使用两个音调的指定部分(举例说) 代替半调的指定部分。这样音符就发出一个较低的音符来代替低四分之一音符。P1的值必须在-100和100之间；P2的值必须在1和100之间。

功能18，设置每四分之一音符的时钟节拍数

输入： P0=时钟节拍数

输出： 无

你使用此功能设置四分之一音符的最小部分。所有的音符事件必须设置为1/P0的倍数。定时器速率也随此功能改变，因此，P0的值必须满足下述条件：

18.2 <= (P0*速度/60)　即　P0 >= 1092/速度

每秒时钟中断 (时钟节拍)数为：

时钟节拍=60 或 P0(如果此值大于60)*速度/60

功能19，立即播放一音符

输入： P0=声部
P1=定调

输出： 无

此函数以选定声部立即（以现行设置）播放音符。此函数开启音符（你可以用功能20关掉此音符）。在某种动作突然出现时（如爆炸），这是有用的。使用事先为某个声部设置的数据，此指定声部就可以在任何时刻播放。使用这个功能和下面的两个功能，你也可以直接播

放一段音乐。但要仔细才能得到定时准确。

功能20，停止播放

输入：　　P0=声部

输出：　　无

使用此功能，你可以关掉用前面功能开启的音符。

功能21，直接设置一设备

输入：　　P0=声部

P1=数据的偏移

P2=设备数据的段

输出：　　无

此功能如功能16一样工作，但此处设备被直接设置。在此功能之后，你可以立即改变设备数据，因为FM芯片是用此数据直接编程。

警告　功能22与23只为1.03版所支持。

功能22，设置音调改变范围

输入：　　P0=半调数

输出：　　无

此功能允许你改变功能17的范围，而且能以半调数来设置。在调用功能0后，此值被置为1，以使之同较老的版本相容。如果需要用的话，此功能必须在功能0之后被调用。

功能23，设置波形标志

输入：　　P0=波形信息的使用

输出：　　无

功能16和21两者都设置设备，但是较老的版本缺乏有关被使用正弦波的数据。此功能指明此信息存在并可被使用。如果你设置P0为值1，功能16和21将接受两个附加信息，它位于包含有关波形类型信息的26个整数之后。如果P0为值0，缺省正弦波被使用。此选项也在调用功能0后被设置，以再次保证同较老版本相容　。如果需要的话，此功能必须在功能0后调用。

Sound驱动程序和CMF比较　Sound驱动程序较之CMF驱动程序提供了更为广泛的功能选择，CMF驱动程序只播放CMF音乐。Sound驱动程序允许你播放各种格式的音乐。

播放ROL音乐

播放ROL文件比播放CMF文件较困难一些。由于各种地址必须更新，所以读ROL文件要花较长的时间。而且因为Sound驱动程序的缓冲区大小可变，使得播放文件本身也更为复杂，所以在整个音乐被送到Sound驱动程序之前，要花一点时间，这可导致各种延迟。

LoadROL函数　在一ROL文件被播放之前，它必须被读入。LoadROL函数提供了一个如何读ROL文件的例子。LoadROL函数按以下顺序工作：

1. 打开文件。
2. 读文件头和提取所需信息。
3. 读速度事件。
4. 读第一个声部的音符总持续时间。
5. 确定音符事件数量。
6. 读这些事件。
7. 读第一声部的设备事件。
8. 读音量和定调事件。
9. LoadROL对余下各声部重复4到8步。
10. 关闭文件。

在文件可被播放之前，必需要装入一个组文件。播放文件按以下顺序依次来完成：

1. 驱动程序被复位。
2. 各正确值--速度、提取的信息等--被设置。
3. 确定是否所有事件指针都指向第一个事件。
4. 必定产生的下一事件被寻找。
5. 此下一事件送到驱动程序。
6. 如果上述各事已作到，则搜索下一事件，且第5步被重复，直至所有各事件被处理。但是，如果下一事件找不到，缓冲区是满的，而必须被处理。为此，驱动程序开始处理缓冲区。
7. 搜索下一事件。
8. 此下一事件送给驱动程序。
9. 第8步重复，直至事件已成功地加到缓冲区。
10. 下一事件被找到，第8步被重复，直至所有事件被处理。

注解　在第5和第6步中，可以处理一定数量的事件，以代替填充缓冲区，使得音乐能较早地开始。此方法也用在下一个清单中。

清单7.5提供了用于装入和播放ROL文件的库。如同CMF库那样，一些函数使用函数CallSound来定义，而且你可以加你自己的函数到库 (例如： 暂停和继续)。此库设计来播放ROL格式，不要去改变它。如果你需要能更改、移去和加入事件，各种类型必须被调整。用此法，你可以写你自己的ROL编辑器。

注解　在清单7.5中，定义了一种类型，它使用包含一个元素的数组。但是往后在清单中，具有更高标号的元素被使用。这是可能的，因为对此数组的一定数量 的存贮空间是事先被保留的，所以几个元素能被访问。为了访问这些元素而不出现错误，范围检验必须被关掉($R-)。但是，如果处理器选项是使用的($N+)的话，则 Pascal 只支持 Single 类型。由于绝大多数用户都没有协处理器，所以仿真程序必须使用 ($E+)。这些选项引起例子中的程序在编译之后相当大。

程序清单7.5: 装入和播放ROL文件的Pascal程序单元

```
Unit PlayRol ;
{ Unit to load and play .ROL files. }
{ $N+E+R }

Interface
Uses Instr ;

Const
    ProcessEvents =100

Type
    { the different events }
    InsEvRc    = Record
                     Time : Word ;
                     Ins : InsName ;
                     Dummy : Array [0..2] Of Byte ;
                 End ;
    MultEvRc   = Record
                     Time : Word ;
                     Mult : Single ;
                 End ;
    NoteEvRc   = Record
                     Tone : Integer ;
                     Time : Word ;
                 End ;
    NoteEvs    = Record
                     Number : Word ;
                     E : Array [1..1] Of NoteEvRc ;
                 End ;
    InsEvs     = Record
                     Number : Word ;
                     E : Array [1..1] Of InsEvRc ;
                 End ;
    MultEvs    = Record
                     Number : Word ;
                     E : Array [1..1] Of MulteEvRc ;
                 End ;
    NoteEvRt   = ^NoteEvs ;
    MultEvPt   = ^MultEvs ;
    InsEvPt    = ^InsEvs ;
    { each voice has the same events }
    VoiceRc    = Record
                     Used : Bollean ;
                     NoteT ,        { are used }
                     NoteO ,        { while playing }
                     InsO,          { idem }
```

```
                    VolO,                { idem }
                    PitchO : Word ;      { idem }
                    NoteEv : NoteEvPt ;
                    InsEv : InsEvPt ;
                    VolEv,
                    PitchEv : MultEvPt ;
                End ;
    RolTp     = Record
                    BasicTempo : Single ;
                    Mode,
                    NoOfVoices,
                    TickBeat,
                    TempoO : Word ;    { to play }
                    TempoEv : MultEvPt ;
                    Voice : Array [0..10] Of ^VoiceRc ;
                End ;
Var
    SNDIOResult : Word ;

{ Test whether the Sound driver is present, SNDIOResult then contains the version.}
Function TestForSNDDriver : Boolean ;
{ Load a .ROL file }
Procedure LoadRol (N : String ; Var R : RolTp) ;
{ Remove ROL data from memory }
Procedure RemoveRol (Var R : RolTp) ;
{ Play music, use bank B }
Procedure PlayRol (Var R : RolTp ; B : BnkTp) ;
{ Reset driver }
Procedure ResetSoundDrv ;
{ General interface routine }
Procedure CallSound (F, P0, P1, P2, P3, P4 : Integer) ;

Implementation
Uses Dos ;

Const
    SNDIntr =101 ;

Type
    EventList = (TempoEve, VolEve, InsEve, NoteEve, PitchEve, NoEve) ;
    RolHeader = Record
                    MajVersion,
                    MinVersion : Word ;
                    Dummy0     : Array [0..39] Of Char ;
                    TikBeat,
                    BeatSecond,
                    YScale,
                    XScale     : Word ;
                    Dummy1     : Byte ;
```

```
                PlayMode    : Byte ;
                Dummy2      : Array [0..142] Of Byte ;
                BasicTempo  : Single ;
            End ;
    TestHeader = Record
                Version  : Word ;
                ID       : Array [0..18] Of Char ;
                Dummy    : Array [0..2] Of Byte ;
            End ;
Var
    SDiv : Word ;
    Bank : BnkTp ;

Function TestForSNDDriver : Boolean ;
Var
    P   : ^TestHeader ;
    Q   : Pointer ;
Begin
    GetIntVec (SNDIntr, Q) ;                              { driver address }
    P : = Ptr (Seg (Q^), Ofs (Q^) -SizeOf (TestHeader) ) ;
    SNDIOResult : = P^. Version ;
    TestForSNDDriver : = P^. ID = 'SOUND-DRIVER-AD-LIB' ;
End ;

Procedure CallSound (F, P0, P1, P2, P3, P4 : INTEGER) ;
Var
    R : Registers ;
    A : Array [0..4] Of Integer ;
Begin
    A [0] : = P0 ; A [1] : = P1 ; A [2] : = P2 ;        { copy data }
    A [3] : = P3 ; A [4] : = P4 ;
    R.SI : =F ;                                          { SI is function }
    R.ES : = Seg (A) ;                                   { ES : BX points at }
    R.BX : = Ofs (A) ;                                   {data list }
    Intr (SNDIntr, R) ;
    SNDIOResult : = R.AL ;
End ;

Procedure LoadRol (N : String ; Var R : RolTp) ;
Var
    F : File ;
    Procedure ReadMultEvents (Var N : MultEvPt) ;
Var
    S ,
    Number : Word ;
Begin
    S : = SizeOf (MultEvRc) ;
    BlockRead (F, Number, 2) ;
    GetMem (N, 2+Number *S) ;
```

```
    N^. Number : = Number ;
    BlockRead (F, N^. E, Number *S) ;
End ;

Procedure ReadInsEvents (Var N : InsEvPt) ;
Var
    S,
    Number : Word ;
Begin
    S : = SizeOf (InsEvRc) ;
    BlockRead (F, Number, 2) ;
    GetMem (N, 2+Number *S) ;
    N^. Number : = Number ;
    BlockRead (F, N^. E, Number *S) ;
End ;

Procedure ReadNoteEvents (Var N : NoteEvPt) ;
Var
    S,
    Total,
    Sum,
    Duration,
    Number.: Word ;
    Start : LongInt ;
Begin
    S : = SizeOf (NoteEvRc) ;
    BloclRead (F, Total, 2) ;
    Start : = FilePos (F) ;
    Sum : =0 ;                                  { determine number }
    Number : =0 ;                               { of notes }
    While Sum < Total Do Begin
        Seek (F, FilePos (F) +2) ;
        BlockRead (F, Duration, 2) ;
        Sum : = Sum+Duration ;
        Inc (Number) ;
    End ;
    Seek (F, Start ) ;                          { read notes }
    GetMem (N, 2+Number *S) ;
    N^. Number : = Number ;
    BlockRead (F, N^. E, Number *S) ;
End ;

Var
    Header : RolHeader ;
    Number : Word ;
    MaxVoice : Byte ;
    V : Byte ;
Begin
    Assign (F, N) ; Reset (F, 1) ;
```

```
    BlockRead (F, Header, SizeOf (Header) ) ;
    R. BasicTempo : = Header. BasicTempo ;          { process header }
    R. Mode : = Header. PlayMode ;
    R. TickBeat : = Header. TickBeat ;
    If R. Mode =1 Then
        R. NoOfVoices : =8
    Else
        R. NoOfVoices : =10 ;
    ReadMultEvents (R. TempoEv) ;                   { read tempo ev. }
    For V : =0 To R. NoOfVoices Do Begin
        New (R. Voice [V] ) ;
        With R. Voice [V]^ Do Begin
            Seek (F, FilePos (F) +15) ;
            ReadNoteEvents (NoteEv) ;
            Seek (F, FilePos (F) +15) ;
            ReadInsEvents (InsEv) ;
            Seek (F, FilePos (F) +15) ;
            ReadMult+Events (VolEv) ;               {read volume ev. }
            Seek (F, FilePos (F) +15) ;
            ReadMultEvents (PitchEv) ;              { read pitch ev. }
            Used : = NoteEv^. Number > 0 ;
        End ;
    End ;
    Close (F) ;
End ;

Procedure RemoveRol (Var R : RolTp) ;
Var
    V : Byte ;
Begin
    For V : = 0 To R. NoOfVoices Do Begin
        With R. Voice [V]^ Do Begin
            FreeMem (NoteEv, 2+NoteEv^. Number *SizeOf (NoteEvRc) ) ;
            FreeMem (InsEv, 2+InsEv^. Number*SizeOf (InsEvRc) ) ;
            FreeMem (PitchEv, 2+PitchEv^. Number*SizeOf (MultEvRc) ) ;
            FreeMem (Volev, 2+VolEv^. Number*SizeOf (Mul+EvRc) ) ;
        End ;
        Dispose (R.Voice [V] ) ;
    End ;
    FreeMem (R.TempoEv, 2+R. TempoEv^.Number*SizeOf (MultEvRc) ) ;
End ;

Procedure ToneEvent (V, T, S, H : Integer) ;
Begin
    CallSound (12, V, O, O, O, O, O) ;                  { select voice }
    If H < > 0 Then
        If S < > 0 Then                                 { silence after onte ? }
            CallSound (14, H60, T, SDiv, T+S, SDiv)     { yes }
        Else
```

```
        CallSound    (15, H60, T, SDiv, 0, 0)                    { no }
    Else
        CallSound (14, H60, O, SDiv, T, SDiv) ;                  { no note }
End ;

Procedure VolEvent (V, T : Word ; F : Single) ;
Begin
    CallSound (12, V, 0, 0, 0, 0) ;                              { select voice }
    CallSound (8, Trunc (F*255), 255, T, SDiv, 0)
End ;

Procedure PitchEvent (V, T : Word ; F: Single) ;
Begin
    CallSound (12, V, 0, 0, 0, 0) ;                              { select voice }
    CallSound (17, 0, Trunc (F*255), 255, T, SDiv) ;
End ;

Procedure InsEvent (V, T : Word ; I : InsName) ;
Var
    B : BnkTp ;
Begin
    GoodInsName (I) ;                                            { find data }
    FindIns (I, Bank, B) ;                                       { found ? }
    If B < > Nil Then Begin                                      { for safety reasons }
        SBIToIns (B^. Ins) ;
        CallSound (12, V, 0, 0, 0, 0) ;                          { select voice }
        CallSound (16, Ofs (B^. Ins. INS^. Modulator) ,
                    Seg (B^. Ins. INS^. Modulator), T, SDiv, O) ;
    End
End ;

Procedure TempoEvent (T : Word ; B, F : Single) ;
Begin
    CallSound (9, Trunc (B*F), T, SDiv, 0,0) ;
End ;

Procedure ResetRol (Var R : RolTp) ;
Var
    V : Byte ;
Begin
    For V : =O To R . NoOfVoices Do                              { go along all voices }
        With R.Voice [V]^ Do Begin
            NoteT : =0 ;                                         { total duration 0 }
            NoteO : =1 ;                                         { first note event }
            InsO : =1 ;                                          { idem for instr.}
            VolO : =1 ;                                          { idem for volume }
            PitchO : =1 ;                                        { idem for pitch }
        End ;
```

```
    R.TempoO : =1;                                    { iodem for tempo }
End ;

Function GiveEv (Var R : RolTp ; V : Byte ; E : EventList)  : Word ;
Var
    H : Word ;

Begin
    H : = $FFFF;
    If E < > TempoEve Then
        With R. Voice [V]^ Do
            Case E Of
                VolEve : If VolO <=VolEv^. Number Then
                            H : = VolEv^. E [VolO]. Time ;
                InsEve : If InsO ,=InsEv^. Number Then
                            H : = InsEv^. E [InsO]. Time ;
                PitchEve : If PitchO <=PitchEv^. Number Then
                            H : = PitchEv^. E [PitchO]. Time ;
                NoteEve : if NoteO <= NoteEv^. Number Then
                            H : =NoteT ;
            End
        Else
        If R. TempoO <= R. TempoEv^. Number Then
            H : = R. TempoEv^. E [R. TempoO] . Time ;
    GiveEv : =H ;          { give time for next event }
End ;

Procedure FindNextEv (Var R : RolTp ; Var V : Byte ; Var E : EventList) ;
Var
    T : Byte ;
    H : EventList ;
    G : Word ;
Begin
    E : = NoEve ;                                  { nothing found }
    G : GiveEv (R, O, TempoEve) ;                  { next tempo event }
    If G < > $ffff Then                            { does event exist? }
        E : TempoEve ;                             { yes, choose this one }
    For T : =O To R. NoOfVoices Do                 { go along all voices }
        If R. Voice [T]^. Used Then
            For H : = VolEve To PitchEve Do        { and all events }
                If G > GiveEv (R, T, H)   Then Begin
                    G : = GivEv (R, T, H) ;        { copy lowest time }
                    E : = H ;                      { accompanying event }
                    V : = T ;                      { and voice }
                End ;
End ;

Procedure SetEvent (Var R : RolTp ; V : Byte ; E : EventList) ;
Begin
```

```
    If E < > TempoEve Then      { does voice count ? }
        With R. Boice [V]^ Do     {yes }
            Case E of
                VolEve  : With VolEv^. E [VolO] Do
                            VolEvent (V, Time, Mult) ;
                InsEve  : With InsEv^. E [InsO] Do
                            InsEvent (V, Time, Ins) ;
                NoteEve : With NoteEv^. Doi
                            If E [NoteO+1]. Tone < > O Then
                                With E [NoteO] Do
                                    ToneEvent (V, Time, O, Tone)
                                Else
                                    With NoteEv^ Do
                                        ToneEvent (V, E [Note)]. Time,
                                                   E [NoteO+1]. Time,
                                                   E [NoteO]. Tone) ;
            PitchEve : With PitchEv^. E [PitchO] Do
                         PitchEvent (V, Time, Mult) ;
        End
    Else
    WithR. TempoEv^. E [R.TempoO] Do
        TempoEvent (Time, R. BasicTempo, Mult) ;
End ;

Procedure IncreaseEv (Var R : RolTp ; V : Byte ; E : EventList) ;
Begin
    If E <> TempoEve Then              { does voice count ? }
        With R. Voice [V]^ Do          { yes, choose voice }
            Case E Of                  { increase event }
                VolEve  : Inc (VolO) ;
                InsEve  : Inc (InsO) ;
                NoteEve : With NoteEv^ Do Begin
                            Inc (NoteT, E [NoteO]. Time) ;
                            Inc (NoteO) ;
                            If E [NoteO].Tone =0 Then
                            Begin                    { process silence }
                                Inc (NoteT, E [NoteO]. Time) ;
                                Inc (NoteO) ;
                            End
                          End ;
            PitchEve : Inc (PitchO);
        End
    Else
        Inc (R. TempoO) ;                            { increase event }
End ;

Procedure PlayRol (Var R : RolTp ; B : BnkTp) ;
Var
    Event : EventList ;
```

```
    Voice : Byte ;
    Counter : Word ;
Begin
    Bank : =B ;                                    { set instr. bank }
    CallSound (2, 0, 1, 0, 0, 0) ;                 { start at begin }
    CallSound (5, 0, 0, 0, 0, 0) ;                 { delete all buffers }
    CallSound (6, R.Mode XOR 1, 0, 0, 0, 0) ;      { set mode }
    CallSound (18, R.TickBeat, 0, 0, 0, 0) ;       { set ticks }
    SDiv : = R.TickBeat ;                          { for timing }
    ResetRol (R) ;                                 { restore data }
    Counter : =0 ;
    Repeat                                         { fill buffer }
        FindNextEv (R, Voice, Event ) ;
        SetEvent (R, Voice, Event) ;
        If SNDIOResult <>0 Then
            IncreaseEv (R, Voice, Event) ;
        Inc (Counter) ;
    Until (SNDIOResult=0) Or (Event=NoEve) OR
            (Counter>ProcessEvents) ;
    CallSound (3, 1, 0, 0, 0, 0) ;                 { start playing }
    While Event < > NoEve Do Begin
        Repeat                                     { place in buffer }
            SetEvent (R, Voice, Event
        Until SNDIOResult <> 0 ;                   { successful ? }
        IncreaseEv (R, Voice, Event) ;             { yes, next event }
        FindNextEv (R, Voice, Event) ;
    End ;
End ;

Procedure ResetSoundDrv ;
Begin
    CallSound (0, 0, 0, 0, 0, 0) ;
End ;
End
```

程序清单7.6: PlayROL程序单元使用的Pascal例

```
Program RolTest ;
Uses PlayRol, Crt, Instr ;

Var
    Song    : RolTp ;       { Rol music }
    Ch      : Char ;        { key }
    Bank    : BnkTp ;       { instrument bank }
    BnkFile : String ;      { file used }

Function BCD (I : Byte) : Byte ;
{ Convert BCD to decimal }
```

```
Begin
    BCD : = 10 * (I div 16) + I mod 16 ;
End ;

Begin
WriteLn ;
If ParamCount <1 Then Begin                    { 1 or 2 parameters ? }
    WriteLn ('Use ROLTEST name.rol [name.bnk] ! ') ;
    WriteLn;
    Halt (1)
End ;
If Not TestForSNDDriver Then Begin             { Sound drv. ? }
    WriteLn ('Error : Sounds driver not found !') ;
    WriteLn ;
    Halt (1)
End ;
If ParamCount=1 Then                           { bank file ? }
    BnkFile : = 'Standard.Bnk'                 { no, 'STANDARD.BNK' }
Else
    BnkFile : = ParamStr (2) ;                 { yes, use it }
WriteLn ('Sound version : ', BCD (Hi ( SNDIoResult) ) , '.',
        BCD (Lo (SNDIOResult) ) ) ;
ResetSoundDrv ;
WriteLn ('Loading : ' , ParamStr (1) ) ;
LoadRol (ParamStr (1), Song) ;
WriteLn ('Loading : ', BnkFile) ;
LoadBnk (BnkFile, Bank) ;
WriteLn ; WriteLn ('Playing ...' ) ;
PlayRol (Song, Bank) ;
Repeat
    CallSound (4, 0, 0, 0, 0, 0) ;             { playing stopped ? }
Until (SNDIOResult =0) Or KeyPressed ;         { yes or key }
If KeyPressed Then Begin
        Ch : = ReadKey ;                       { read key }
        ResetSoundDrv ;                        { and stop playing }
    End ;
    WriteLn ;
End.
```

编程FM芯片

虽然你可以使用迄今为止我们所叙述过的驱动程序来用FM芯片播放音乐，但是如果你需要直接加以作用，则它们就不行了。如果你使用你自己的音乐格式，则CMF驱动程序就毫无用处。因为此驱动程序只支持CMF格式。Sound驱动程序则较为灵活。但是，Sound驱动程序有一个缺点：当其播放大的音乐段时，程序必须等候缓冲区的自由空间--自由空间经常不能够快的提供。对这种情况，具备你自己的播放程序是有用的。

需要具备你自己的播放程序的另一个理由是你可以加入声音到你的程序中。例如：如果你想用声音来标志出错信息，则你就可能要这样作。在这种情况下，在程序中组合一些简单的子例程，较之使用Sound驱动程序要方便得多。使用你自己的子例程，存贮器使用得更好，而更多的自由存贮器可供程序本身使用。本段叙述FM芯片部分，并告诉你，你自己如何编程FM芯片。

前面各段叙述了运算量的设置，但是它们没有说明可能的值。这些值在本段中讨论，即使你不打算你自己编程FM芯片，但只要你想对FM芯片的功能有更好的了解，也应考虑阅读本段。

在说明FM芯片能作什么之前，本段首先叙述如何编程 FM芯片。

FM芯片如何工作

在FM芯片中，所有的设置都内部存贮在变量中。这些变量称为寄存器。FM芯片使用索引和数据端口进行编程。一个特别的数被送到索引端口，这个数就表示了某个寄存器的选择。其次，寄存器可通过数据端口以新的数据设置。

下面是一“伪码”例子：

```
[1] OUT  Index，11  （寄存器11被选择）
[2] OUT  Data，123  （值123赋给寄存器11）
[3] OUT  Index，13  （寄存器13被选择）
[4] OUT  Data，24   （值24赋给寄存器13）
[5] OUT  Data，210  （值210被加于寄存器13，而前值（24）被重写掉）
```

在我们的例子编码中，行4用处很小，因为寄存器13在第5行获得新值。

AdLib端口

AdLib卡具有下列各端口：

388H　索引和状态端口（读和写）
389H　数据端口（只写）

Sound Blaster端口

Sound Blaster 支持下列各端口 (注意：头两个同AdLib所支持的端口相同--据此，Sound Blaster同AdLib相容)：

388H　索引和状态端口（读和写）
389H　数据端口（只写）
2X8H　索引和状态端口（读和写）
2X9H　数据端口（只写）

对Sound Blaster，X是被选用的基端口。因此，并不是工作在Sound Blater的所有的软件（对FM芯片）都自动地工作在AdLib。很明显，我们建议你使用端　口388H和389H，因为这样你不必检验Sound Blaster的基端口。

新端口　在上述四个端口之外，Sound Blaster 有四个新的端口，两个用于左通道（第一个芯片）。两个用于右通道（第二个芯片）。对左通道（第一个芯片）的端口是：

2X0H　索引和状态端口（读和写）

2X1H　数据端口（只写）

对右通道（第二个芯片）的端口是：

2X2H　索引和状态端口（读和写）

2X3H　数据端口（只写）

如果你使用端口388H和389H或2X8H和2X9H，则左、右两通道皆被编辑(端口2X0H和2X1H以及端口2X2H和2X3H被写以数据)。

为在两个芯片设置设备(举例说)，你可以通过端口2X8H和2X9H来编程两个FM芯片。以后，通过端口2X0和2X1重设设备左通道，和经端口2X2、2X3重设右通道。

注解　重设设备的例子，见本章末的清单7.10。

如前所述，数据端口是只写的。状态寄存器也可通过索引端口来读取。此寄存器以后讨论。

在设置索引端口之后，在寄存器能被写之前，程序必须等待3.3毫秒，而在通过索引端口能选择一新的寄存器之前，程序必须等待23.3毫秒。

FM芯片寄存器设置

FM芯片并未使用所有256个寄存器，表7.7示出了FM芯片寄存器设置。这种分布对每一寄存器组是相同的。在不同的寄存器组（即是操作器的设置）中，20个寄存器（0...15H）是对18个操作器的。寄存器6、7、EH和FH未用。表7.8示出在各寄存器中的操作器的分布。此种分布对每个寄存器组是相同的。

表7.7　FM芯片寄存器设置

寄存器	位7	位6	位5	位4	位3	位2	位1	位0
01H	测　试							
02H	快计数器							
03H	慢计数器							
04H	IRQ	Mask st Cnt	Mask SI Cnt				Start/Stop SI Cnt	Start/Stop Fst Cnt
08H	CSM	SEL						
20H-35H	AM	VIB	EG-TYP	KSR	MULTI			
40H-55H	KSL		Total Level (TL)					
60H-75H	ATIACK RATE (AR)			DECAY RATE (DR)				
80H-95H	SUSTAIN LEVEL (SL)			RELEASE RATE (RR)				
A0H-A8H	F-NUMBER							
B0H-B8H			KEY	BLOCK			F-NUMBER	
BDH	Lntensity AM VIB		Rhythm	BASS	SNARE	TOM	TOP CYMBAL	HIHAT
C0H-C8H					FEEDBACK			FM
E0H-F5H							WS	

FM芯片有两个标准的播放模式，即旋律模式和节奏模式。在旋律模式中，每种设备(或声部）有两个操作器(如表7.8中所示的)。在节奏模式中，有四种设备，它们仅由一个操作器组成。

在头六种设备，在旋律模式中，操作器的分配是相等的，表7.9示出对5种节奏设备的分配。低音鼓被认为是普通设备。

表7.8 FM芯片寄存器设置

操作器	1	2	3	4	5	6	7	8	9
声　部	1	2	3	1	2	3	4	5	6
类　型	调制器			载波			调制器		
寄存器	00H	01H	02H	03H	04H	05H	08H	09H	0AH
操作器	10	11	12	13	14	15	16	17	18
声　部	4	5	6	7	8	9	7	8	9
类　型	调制器			载波			调制器		
寄存器	0BH	0CH	0DH	10H	11H	12H	13H	14H	15H

表7.9　五种节奏设备的分配

设备	操作器
低音鼓	13和16
小鼓	17
印度长鼓	15
铙钹	18
高帽	14

在FM芯片中各寄存器如何工作

你可以使用各寄存器来编程FM芯片。你可以设置振荡器、包络产生器、以及声平控制器。在寄存器的讨论中，经常为多于一位或一些位被处理，此处b7是最高位（值128）而b0是最低位（值1）。没有读到的各个位没有值，而必须被设置为0。

下面看看每一个寄存器：

寄存器1，测试寄存器

在你能编程FM芯片之前，此寄存器必须具有0值。

寄存器2，快计数器

FM芯片有两个计数器，它们为8位长，计数从0～255。快寄存器的缺省值可以被写到寄存器2。如果计数器被启动，则FM芯片以此处指示的值来设置计数器。其次，计数器每80微秒(0.00008秒)加1。如果溢出出现(从255到0，其进位=1)，则计数器重新以指示值设置。你以下面公式来计算每个溢出的时间：

T=(255-缺省值)*0.08毫秒

假定寄存器装入值140，在此种情况下，在溢出产生之前将要花9.28(256-140=116，

116*0.08=9.28)毫秒。状态寄存器允许你检验是否已出现一溢出，这在本章后面讨论。

此计数器也被用在复合声部合成模式中。如果一溢出发生，则所有的设备声部被打开，而又立即关闭，所以所有的声部在同一时刻重新产生一个声音。

寄存器3，慢计数器

使用此计数器，你为另外的计数器设置缺省值。区别在于，此时计数器为每320微秒加1。所以此计数器是慢四倍。下面的公式适用于此：

T=(256-缺省值)*0.32毫秒

如果此寄存器装入值140，则在一溢出产生之前要花37.12毫秒。用状态寄存器，你可以检验是否一溢出已出现。

寄存器4，计数器控制器

此寄存器检验前面的两个计数器：

位7　如果此位获得值1，状态寄存器的各标志位装入值0，此后，此位又重新为值0。

位6　如果此位为值1，快计数器的溢出不在状态寄存器中指出。

位5　如果此位为值1，慢计数器的溢出不在状态寄存器中指出。

位1　如果此位为值1，慢计数器以指定的缺省值装入，而且计数器加1。如果此位为值0，慢计数器停止，而不工作。

位0　如果此位为值1，则快计数器以指定的缺省值装入，而且计数器加1，如果此位为0值，则快计数器停止，而不工作。

状态寄存器

在我们继续讨论余下的各寄存器之前，我们需要看一看状态寄存器。此寄存器的值可以使用索引寄存器（288H，2X8H，2X0H或2X2H）来读。只是三个最高位才有功能：

位7　如果此位有值1，则另两位之一也有值1。

位6　这是快计数器的标志；如果溢出已经产生，则此位包含值1。

位5　此位有和位6相同的功能，但是是对慢计数器。

在溢出发生后，对应的位保持原设置。用以1装入寄存器4的位7的办法，所有的位重新设置为0。使用此寄存器和寄存器2、3 和4则一恒定的定时可以应用，而不须使用定时器。首先，计数器必须以缺省值装入；其次，寄存器4必须启动此计数器。同时，状态寄存器可设以0值。往后一等待周期出现，直至溢出发生，这在状态寄存器中指出。状态寄存器在一个指定动作能执行之前，必须再设置为0值。使用再次检验状态寄存器的办法，则恒定的时间周期就被产生。如果你不打算这样作，你可以同前面三个寄存器相结合来使用此寄存器以测试FM芯片是否存在，如清单7.7所示。当然，对Sound Blaster有另外的方法来检查此芯片，但是对AdLib这是唯一的办法。

程序清单7.7：　测试FM芯片存在的Pascal例子

```
Function TestFMChip : Boolean ;
```

```
Var
    S1, S2, T : Byte ;
Begin
    WriteFM (1, 0) ;                    { delete test register }
    WriteFM (4, $60) ;                  { disable connection/stop counters }
    WriteFM (4, $80) ;                  { delete status register }
    S1 : = ReadFM ;                     { read status }
    WriteFM (2, $f0) ;                  { set counter 1 at 255 }
    WriteFM (4, $21) ;                  { connection on + start counter 1 }
    For T : =0 To 200 Do                { wait a while }
        S2 : = ReadFM ;
    S2 : = ReadFM                       { read status }
    WriteFM (4, $60) ;                  { disable connection/stop counter }
    WriteFM (4, $80) ;
    TestFMChip : = ( (S1 AND $E0)=0) AND ( (S2 AND $E0) = $C0) ;
End ;
```

寄存器8，通用控制寄存器

位7　使用此位你选择正常合成模式或复合声部合成模式。如果此位被复位（=0）正常合成被选择。但是，如果此位被置位（=1），则第二种模式被选择。如果此位被置位，所有的设备必须被关闭(后面我们讨论如何作到这点)。

位6　此位确定被播放的音符的分割点。如果此位被复位，频率设置的位8确定分割点。如果此位被置位，频率设置的位9确定分割点。见表7.10

表7.10　音符选择

SEL=0																
音阶	0		1		2		3		4		5		6		7	
F-number b9	1		1		1		1		1		1		1		1	
F-number b8	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
分割点	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
SEL=1																
音阶	0		1		2		3		4		5		6		7	
F-number b9	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
F-number b8	X		X		X		X		X		X		X		X	
分割点	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

寄存器20H-35H，对操作器的AM/VIB/EG-TYP/MULT设置

位7：AM　此位确定是否在操作器信号的强度中有一波动。只要此位一被置位，此项就被选择。当其此位被复位时，就没有波动产生。此波动的频率为3.7Hz。对设置波动的强度有两种可能性，此选项在寄存器BDH中描述。

位6：VIB　此位有与位7相同的功能，只是此时是在音调中的波动被选择。这以6.4Hz的频率发生。对设置差别的大小，有两种可能性；此选项也在

寄存器BDH讨论。

位5：EG-TYP　此位指示只要声部为可用（置位）时，包络产生器必须保持在保持声水平呢，或者它必须以释放速率直接（复位）到0（见本章开始处的图7.5）。

位4：KSR　你使用此位来作成音调越高，声音越短。这由增加冲击、衰减和释放速率以及保持声平来产生。由此法，音符被更快处理，而声音更短。当其此位被置位时，此种情形产生。当复位时，此选项被关掉。

位3...0：MULTI　此处的值组选择所建立频率被乘的因子。由于此因子可分别地对每个操作器设置，所以和声信号可以产生。表8.6示出所有可能的MULTI设置，以及相应的乘数因子。

表7.11 MULTI设置

MULTI	乘数因子
0	0.5
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15

寄存器40H-55H，操作器的KSL/TL设置

位7和6：KSL　使用这些位，你确使音调越高，不会使声音越响。表7.12示出强度减少的程度。

位5...0：TL　这六位设置由操作器产生的信号的声平。零是最大声平而63为最小声。信号的声平由下列公式表示：

声平=(63-TL)*0.7dB

表7.12　声音降低KSL

KSL	每一音阶的音量减少
0	0dB
1	3dB
2	1.5dB
3	6dB

寄存器60H-57H，操作器AR/DR的设置

位7...4： AR　　增高率的值。0是最低率(因为你没有音调可听，所以实际上它完全不是一个速率)，而15是最高率。

位3...0： DR　　衰减率的值。如增高率同样的值用于衰减率。

如果此处设置为0值，则保持声平就绝不会达到，而信号继续在最大声平，直至此声部被关掉，而音调以释放率降到0声平。

寄存器80H-95H，操作器SL/RR设置

位7...4： SL　　对保持声平的值。此寄存器组也可以被置于0和15间的值。在此情况下，0表示最大声平，而15表示最小声平(无声)。

位3...0： RR　　释放率的值(在0和15间，0是最低值，而15是最高值)。如果RR为值0，音调永远继续，即使在该声部被关掉后也一样。停止它的唯一办法是改变释放率或设置TL到63。

寄存器E0H-15H，操作器WS设置

位1和0：WS　　这两位的值选择（可调的）正弦波之一。

寄存器C0H-C8H，对调制器操作器FB/FM设置

此寄存器组对九个调制器的每一种设置反馈以及合成种类。你必须使用属于调制器的寄存器。寄存器的号对应于属于调制器的声部。

位3...1：FB　　此处此值组确定调制器的反馈强度。表7.13指出在调制器信号再被用作输入之前，调制器信号被乘的因子。如果所建立的值为0，就没有反馈(因为以0乘是0)。

位0： FM　　本位指示设备由加法合成（置位）还是由频率调制合成（复位）。参阅本章开始部分有关这些合成之间的区别。

表7.13　各种反馈放大

反馈	0	1	2	3	4	5	6	7
因子	0	π/16	π/8	π/4	π/2	π	π*2	π*4

下面两个寄存器组为每个声部的声音设置频率。在旋律模式中，每个声部相应于相同的端口。在节奏模式中，这也用于六种普通设备。低音鼓通过第七个端口来编程，而铙钹通过第9个端口来编程。三个剩下的设备不能改变变调。所以它们不能被设置。因此，在节奏模式被选用时，第八个端口没有功能。

寄存器A0H-A8H，九个声部的频率设置

位7...0 ：F-NUNBER　　这些位包含对某个声部的频率设置的10位值的8个低位。这在下面的寄存器叙述中仔细讨论。

寄存器B0H-B8H，声部选择和频率设置

位5： KEY　当声部被接通时，用该声部所建立的操作器来产生声音。如果位EG-TYP为置位，声音将保持在保持声平，直到此KEY位被复位，而声部被关闭。这并不意味着声音立即停止。在KEY位复后，声音将以释放率降至0。如果EG-TYP被复位，这将在保持声平达到后立即发生，而不管KEY位被复位与否。如果FM芯片此时被置位，它再重新开始产生声音，即使前面的音符还在发声，它也必须被置位。在节奏模式中，当低音鼓被演奏时，此位必须被置位，而对印度长鼓则保持在复位(我们将进一步叙述这些设备如何打开和关闭)。

位4...2：BLOCK　由这三个位设置的值确定频率翻倍的次数。换句话说，你使用此值确定音符被播放的音阶。

位1和0：F-NUMBER　这是频率设置的两个高位，它同前面的寄存器组一起组成10个位。在F-NUMBER，BLOCK和实际频率之间的关系用下述公式表示：

频率=50,000*F-NUMBER*2^(BLOCK-20)

表7.14示出在某个音阶中各音符中的频率。在此音阶的BLOCK值为4。如果使用F-NUMBER为所计算的这些值，当你想使用另外的音阶时，你就只须改变BLOCK的值。

表7.14　在某个音阶中的频率和F-NUMBER

音符	频率	F-NUMBER
C	261.63	343
C#	277.18	363
D	293.66	385
D#	311.13	408
E	329.63	432
F	349.23	458
F#	369.99	485
G	329.00	514
G#	415.30	544
A	440.00	577
A#	466.16	611
B	493.88	647

寄存器BDH，颤音强度和节奏设置

位7　确定操作器的信号强度中波动的强度。如果此位被置位，波动的最大值是4.8dB；如果此位被复位，最大值为1dB。

位6　此位有同于位7的功能，但是现在是确定在频率中波动的强度。如果此位被置位，最大波动为14%；如果此位被复位，它为7%。

上面所述的两位，对所有的操作器设置同样的波动。因此，要设置不同的波动是不可能的。这些位只影响使用一个或两个设置的操作器。

位5　此位设置模式。如果此位被复位，FM芯片对九种设备使用旋律模式。如果此位被置位，FM芯片对11种设备使用节奏模式。

位4..0　这些位具有与KEY位相同的功能。它们打开或关闭节奏设备，而且这是在位5被置位时，它们才有的功能。当其相应的位被置位时，设备就打开，使用复位此位的方法，设备就被关闭。至于对释放率和保持声平的影响，关于KEY位的说明同样适用这些位。表7.15示出对应于每一位的设备。

表7.15　在寄存器BDH中的节奏设备

位	设　备
4	低音鼓
3	小鼓
2	印度长鼓
1	铙钹
0	高帽

注解　为播放低音鼓，你不仅必须设置对低音鼓的位，而且也要设置第十一声部（A6H和B6H）的KEY位。这不适用于印度长鼓。

FM芯片编程库

本节包含了一个库，你可以直接用它来编程FM芯片。在库中，编程被分成三部分：

1. 设置设备数据。
2. 设置或改变音量。
3. 以某种频率播放一声部。

如果音量不必调整的话，前两点可组合在一起。在库中你可以容易地改变这一点。当然，如果你只需要产生某个指定声音一次，则所有三点都可组合在一起。

当你在设置设备时，端口组20H，40H，60H，80H，E0H，有时还有C0H（对调制器）必需被编程。最适合的格式为SBI格式，因为16位格式相应于各寄存器的设置，而且库也作成使用此种格式。(使用前面的INSTR库)。

各寄存器只能写数据，要读某个寄存器的现行设置是不可能的(状态寄存器除外)。如果程序想改变寄存器中的某个数据而留下其他的设置不动，则它就必须存贮寄存器的现行设置在一变量中。此种情况有一个好的例子为寄存器BDH。程序本身必须记住哪个节奏设备是打开的，哪个是关闭的，因为当某个节奏设备被打开时，另外的就不能打开，而关闭也是一样。

当在设置音量时，库保存有此音量的付本。如果新的设备被置位，它就使用付本的数据。但是，不要忘记，除了TL设置外，KSL设置也位于寄存器组40H中。后者首先必须以AND和OR命令去掉，之后，它又必须被再加入(实现见库清单)。

提示　在FM合成中，调制器用作载波的输入。只有载波的音量必须被改变，因为，这控制着最后信号。在加法合成中，两个操作器相互并联使用，而两个操作器的音量都必须被改变。

当然每次你设置音量时，你也可以指定设备。由于使用内部变量，所用的结果是较为抽象的：设置音量同所选择的设备设置无关。

在一个设备被置位之后，它就能够进行播放。为作此，你首先必须确定播放模式（旋律模式，节奏模式中，设备9是不同于在旋律模式中的设备）。普通设备通过寄存器组A0H和B0H来[illegible]。对节奏设备，寄存器BDH也必须被使用。如我们前面已讨论的，另外两个寄存器组用于低音鼓和印度长鼓设置。由于每种节奏设备必须分别地打开和关闭，现行设置的付本在此外被使用。

程序清单7.8是直接编程FM芯片的Pascal程序单元。

程序清单7.8：　直接编程FM芯片的Pascal程序单元。

```
Unit FMDirect ;

Interface
Uses Instr ;

Const
    Melody  = True ;                    { the two play modes }
    Rhythmic    = False ;
    { the frequencies for the var. notes }
    FrLst : Array [0...11] Of Word = ( 343, 363, 385, 408, 432, 458, 485, 514, 544, 577, 611, 647) ;
    BassDrum    = 7 ;
    SnareDrum   = 8 ;
    TomTom      = 9 ;
    TopCymbal   = 10 ;
    HiHat       = 11 ;

Var
    Port : Word ;
{ set register }
Procedure WriteFM ( I, D : Byte) ;
{ read status register }
Function ReadFM : Byte ;
{ execute timer test }
Function TestFMChip : Booleam ;
{ set play mode }
Procedure PlayMode ( M : Boolean) ;
{ set volume for a particular voice }
Procedure Volume ( N : Byte ; V : Real) ;
{ ask voice volume }
Function GiveVolume (N : Byte) : Real ;
{ play a note on a particular voice }
Procedure PlayNote (N : Byte ; T : Word) ;
{ switch off voice }
Procedure SwitchOffNote (N : Byte) ;
{ set instrument }
Procedure SetIns (N : Byte ; I : InsTP) ;
```

```
Implementation
Const
    { the 18 operator offsets }

    OpAd : Array [1..18] Of Byte = (00, 01, 02, 03, 04, 05, 08, 09, 10,
                                    11, 12, 13, 16, 17, 18, 19, 20, 21) ;
    { the modulator operator offsets }
    Ins9 : Array [1..9] Of Byte = (01, 02, 03, 07, 08, 09, 13, 14, 15) ;
    { the modulator + rhythm. ins operator offsets }
    Ins11 : Array [1..11] Of Byte = (01, 02, 03, 07, 08, 09, 13, 17, 15, 18, 14) ;
Type
    VolumeRc = Record { for the volume setting }
                   KSLTL : Array [0..1] Of Byte ;
                   Sort  : Byte ;
                   Value : Real ;
               End;
    VolumeAr = Array [1..11] Of VolumeRc ;
    Ar       = Array [1..11] Of Byte ;  { help array }
    ArPt     = ^Ar ;
Var
    Volumes     : VolumeAr ;     { volume settings }
    MelodyMode  : Boolean ;      { play mode }
    BasicIns    : ArPt ;         { ins. table to be used }
    PortBD      : Byte :         { for the rhythm. ins. }

Procedure WriteFM (I, D : Byte) ;
Var
    K, S : Word;
Begin
    Port [Port] : = I ;                  { select register }
    For K : =0 To 28 Do S : =K ;         { wait 3.3 microsec }
    Port [Port+1] : =D ;                 { set register }
    For K : =0 To 68 Do S : =K ;         { wait 23.3 microsec }
End ;

Function ReadFM : Byte ;
Begin
    ReadFM : = Port [Port] ;             {read data }
End ;

Function TestFMChip : Boolean ;
Var
    S1, S2, T : Byte ;
Begin
    WriteFM (1, 0) ;                     { delete test register }
    WriteFM (4, $60) ;                   { disable connection/stop counters }
    WriteFM (4, $80) ;                   { delete status register }
    S1 : = ReadFM ;                      { read status }
    WriteFM (2, $ff) ;                   { set counter 1 at 255}
```

```
    WriteFM (4, $21) ;                  { connection on + start counter 1 }
    For T : =0 To 200 Do                { wait a while }
        S2 : = ReadFM ;
    S2 : = ReadFM ;                                  { read status }
    WriteFM (4, $60) ;                               { disable connection/stop counters }
    WriteFM (4, $80) ;
    TestFMChip : = ( (S1 AND $E0) =0) And ( (S2 AND $E0) = $C0) ;
End ;

Procedure PlayMode (M : Boolean) ;
Begin
    MelodyMode : =M ;                                { set mode }
    WriteFM (8,0) ;                                  { set CSM & SEL at 0 }
    If MelodyMode Then Begin
        BasicIns : = Addr (Ins9) ;                   { use 9 ins. }
        PortBD : = PortBD And (Not 32) ;             { rhythm. mode off }
    End
    Else Begin
        BasicIns : = Addr (Ins11) ;                  { use 11 ins. }
        PortBD : = PortBD Or 32 ;                    { rhythm. mode on }
    End ;
End ;

Function GiveRBit (N : Byte ) : Byte ;
{ give the bit belonging to the rhythmic instrument }
Begin
    Case N Of
        BassDrum : GiveRBit : =16 ;     { bit 4 }
        SnareDrum : GiveRBit : =8 ;     { bit 3 }
        TomTom : GiveRBit : =4 ;        { bit 2 }
        TopCymbal ; GiveRBit : =2 ;     { bit 1 }
        HiHat : GiveRBit : =1 ;         { bit 0 }
    Else
        GiveRBit : =0 ;                 { no bit }
    End ;
End ;

Procedure SetOperator (N, M : Byte ; I : SBIFormat) ;
{ set modulator or carrier at voice N }
Var
    H : Byte ;
Begin
    H : = OpAd [BasicIns^ [N]+3*m] ;                 { determine oper. port }
    WriteFM (H+$20, I. Snd [M] ) ;                   { set Am/VIB/... }
    WriteFM (H+$60, I.ARDR [M] ) ;                   { set AR + DR }
    WriteFM (H+$80, I.SLRR [M] ) ;                   { set SL + RR }
    WriteFM (H+$E0, I.WS [M] ) ;                     { set WS }
    Volumes [N]. KSLTL [M] : = I.KSLTL [M] ;         { copy volume }
    If (M=0) And (MelodyMode OR (N<8) ) Then Begin
```

```
        WriteFM (N-1+$CO, I.FBFM[0] ) ;                    { set FM + FB }
        Volumes [N] . Sort : = I . FBFM[0] AND 1 ;         { synth. type }
    End ;
End ;

Procedure Volume (N : Byte ; V : Real) ;
Var
    KSL,
    TL,
    B : Byte ;
Begin
    If MelodyMode Or (N<8) Then Begin { carrier }
        B : = BasicIns^ [N] ;       { determine oper. offset }
        KSL : = Volumes [N]. KSLTL [1] And 192 ;  { copy KSL }
        TL : = (Volumes [N]. KSLTL [1] And 63) Xor 63 ;
        TL : = Round (V*TL) ;      { new volume }
        If TL>63 Then TL : =63 ;  { too large ?}
        WriteFM ($40+OpAd [B+3], KSL OR ( TL XOR 63) ) ;
    End ;
    If (Volumes [N]. Sort =1) Or       { addit. synthesis? }
        (Not MelodyMode And (N>7) ) Then Begin     { rhythm. ? }
        KSL : = Volumes [N] . KSL TL [0] And 192;          { yes }
        TL : = (Volumes [N] . KSL TL [0] And 63) Xor 63 ;
        TL : = Trunc (V*TL) ;    { set modulator or rhythm. }
        If TL>63 Then TL : =63 ;
        WriteFM ($40+OpAd [B], KSL OR (TL XOR 63) ) ;
    End
    Else
        WriteFM ($40+OpAd [B]. Volumes [N] . KSL TL [0]) ;
    Volumes [N] . Value : = V ;
End ;

Function GiveVolume (N : Byte) : Real ;
Begin
    GiveVolume : = Volumes [N] . Value ;
End ;

Procedure SetIns (N : Byte ; I : InsTp) ;
Var
    C : InsTp ;
    P : ArPt ;
    V : Real ;
Begin
    CopyIns (I, C) ;                    { use copy }
    INSToSBI (V) ;                      { use SBI format }
    V : = GiveVolume (N) ;              { set volume temporarily }
    Volume (N, 0.0) ;                   { at 0 }
    SetOperator (N, O, C.SBI^)          { process modulator }
    If MelodyMode Or (N<8) Then
```

```
        SetOperator (N, 1, C.SBI^) ;        { and if needed carrier }
    Volume (N, V) ;                         { restore volume }
    RemoveIns (C) ;          { remove copy }
End ;

Procedure PlayNote (N : Byte ; T : Word ) ;
Var
    FNr : Word ;
    Blk : Byte ;
Begin
    FNr : = FrLst [T Mod 12] ;                          { determine frequency }
    Blk : = ( (T Div 12) And 7) Sh1 2 ;                 { and octave }
    If MelodyMode Or (N<8) Then Begin                   { set + switch }
        WriteFM ($A0+N-1, FNr and 255) ;                { on note }
        WriteFM ($B0+N-1, (FNr Shr 8) Or Blk Or 32) ;
    End ;
    If Not MelodyMode And (N=TomTom) Then Begin
        WriteFM ($A8, FNr And 255) ;                    { only setting }
        WriteFM ($B8 (FNr 8) Or Blk) ;
    End ;
    If Not MelodyMode And (N>6) Then Begin
        ProtBD : = PortBD Or GiveRBit (N) ;             { process rhythm. }
        WriteFM ($BD, ProtBD)                           { instrument }
    End ;
End ;

Procedure SwitchOffNote (N : Byte) ;
Begin
    If MelodyMode Or (N<8) Then
        WriteFM ($B0+N-1, 0) ;
    If Not MelodyMode And (N>6) Then Begin
        ProtBD : = ProtBD And (Not GiveRBit (N) ) ;
        WriteFM ($BD, ProtBD) ;
    End ;
End ;

Var
    K : Byte ;
Begin
    Port : = $388 ;
    PortBD : =0 ;
    For K : =1 To 11 D0
        Volumes [K]. Value : =0 ;
End.
```

清单7.9示出库的各种函数的使用。首先设备组STANDARD.BNK被读，其次，设备PIANOI被置位。各个设置也被表示出(此处TL和SL的设置是实际值的相反值)。你现在可以自己使用两个底部键行来播放音符。使用方括号([和])，你可以减少或增加音阶，而使用花括号

({和})，你可以改变音量。一当你按下空格键，则音调就停止。你可以用I键设置新的设备，而用ESC键退出程序。

例如：你可以用对各种设备设置的编辑器来扩充此程序，并产生你自己的设备编辑器。关于进一步的信息，参阅清单中的注解。

程序清单7.9：键盘例的Pascal清单。

```
Program KeyBoard ;

Uses FMDirect, Instr, Crt ;
{ A simple keyboard that programs the FM chip directly. }

Const
    { define piano Keys }
    Key B : String [12] = 'ZSXDCVGBHNJM' ;
    { and the corresponding notes }
    NoteT : String [12] = 'C#D#EF#G3A3B' ;
Var
    Instru : InsTp ;        { instrument to be played }
    Bank : BnkTp ;          { used instrument bank }
    Name   : InsName ;      { instrument to be found }
    Counter,                { for finding the note }
    Octave,                 { current octave }
    Voice  : Byte ;         { for rhythmic instruments }
    Note   : Word ;         { note played }
    Key    : Char ;         { read key press }
    Vol    : Real ;         { current volume }
Procedure OperatorInfo ( K : Byte ; I : InsOp) ;
{ give information about the modulator or carrier }
Var
    C : Byte ;
Begin
    If K>1 Then Begin
        For C : =3 To 13 Do Begin
            GotoXY (58, C) 1 Write ('  - ') ;
        End ;
        GotoXY (54, 14) ; Write ('    - ') ;
        GotoXY (54, 15) ; Write ( '  - ')
    End
    Else Begin
        C : = K*5+53 ;
        GotoXY (C, 3) ; Write (I. KSL : 4) ;
        GotoXY (C, 4) ; Write (I. MULTI : 4) ;
        GotoXY (C, 5) ; Write (I. AR : 4) ;
        GotoXY (C, 6) ; Write (I. Sl XOR 15) : 4) ;
        GotoXY (C, 7) ; Write (I. EG_TYP: 4) ;
        GotoXY (C, 8) ; Write (I. DR : 4) ;
        GotoXY (C, 9) ; Write (I. RR : 4) ;
```

```
        GotoXY (C, 10) ; Wirte (I. TL XOR 63) : 4) ;
        GotoXY (C, 11) ; Write (I. AM : 4) ;
        GotoXY (C, 12) ; Write (I. VIB :4) ;
        GotoXY (C, 13) ; Write (I. KSR:4) ;
        If K=0 Then Begin
            GotoXY (C, 14) ; Write (I. FM : 4) ;
            GotoXY (V, 15) ; Write (I. FB : 4)
        End ;
    End ;
End ;

Procedure NewIns (S : String ) ;
{ Find new instrument and set it }
Var
    N : InsName ;
    H : BnkTp ;
Begin
    StrToName (S, N) ;                          { convert to name }
    FindIns (N, Bank, H) ;                      { find instrument }
    If H < > Nil Then Begin                     { found ? }
        RemoveIns (Instru) ;                    { remove old inst. }
        Copy Ins (H^. Ins, Instru) ;            { copy from bank }
        SBIToINS (Instru) ;                     { use INS format }
        SwitchOffNote (Voice) ;
        Volume (Voice, 0) ;
        If Instru. Ins^. Mode = 1 Then
            Voice : = Instru. Ins^. Nummer+1    { rhythmic }
        Else
            Voice : =1 ;                        { melody }
        SetIns (Voice, Instru) ;
        Volume (Voice, Vol) ;
        Name : =N ;                             { copy name }
        OperatorInfo (O, Instru. Ins^. Modulator) 1
        If (Instru. Ins^. Mode=1) And
                (Instru.Ins^. Number>6) Then    { no carrier ? }
            OperatorInfo (2, Instru. Ins^. Carrier) ;   { yes }
        Else
            OperatorInfo (1, Instru. Ins^. Carrier) ;   { no }
    End;
End ;

Procedure Initialization ;
{ set screen and the variables needed }
Begin
    If Not TestFMChip Then Begin
        WriteLn ;
        WriteLn ('Error, no FM chip present ! ') ;
        WriteLn ;
        Halt (1) ;
```

```
    End ;
    WriteLn ;
    WriteLn (' Reading bank file ... ') ;
    LoadBnk ('Default. Bnk', Bank) ;
    CopyIns (Bank^. Ins, Instru) ;  { Instru must be an INS }
    ClrScr ;
    Octave : =3 ;                   { begin in octave 3 }
    Vol : =1.0 ;                    { no note }
    Note : = $ffff ;                { for rhythm. ins. }
    GotoXY (1, 1) ; Write (' Instrument : ') ;
    GotoXY (1, 3) ; Write (' Volume : ') ;
    GotoXY (1, 5) ; Write (' Octave :') ;
    GotoXY (1, 7) ; Write (' Voice : ') ;
    GotoXY (1, 9) ; Write (' Note : ') ;
    GotoXY (1, 11) ; Write (' Keyboard') ;
    GotoXY (1,12) ; Write (' S D    G H J  ') ;
    GotoXY (1,13) ; Write ('Z X C V B N M') ;
    GotoXY (40, 1) ; Write ('Setting  Mod. Car.') ;
    GotoXY (40, 2) ; Write ('- - - - - - - - - - - - - - - - ') ;
    GotoXY (40, 3) ; Write ('KSL                          ') ;
    GotoXY (40, 4) ; Write ' Freq mult ') ;
    GotoXY (40, 5) ; Write ('Attack rate') ;
    GotoXY (40, 6) ; Write ('Sustain level ') ;
    GotoXY (40, 7) ; Write ('Sustaining ') ;
    GotoXY (40, 8) ; Write ('Decay rate ') ;
    GotoXY (40, 9) ; Write ('Release rate ') ;
    GotoXY (40, 10) ; Write ('Total level ') ;
    GotoXY (40, 11) ; Write ('AM ') ;
    GotoXY (40, 12) ; Write ('VIB ') ;
    GotoXY (40, 13) ; Write ('KSR') ;
    GotoXY (40, 14) ; Write ('Freq synthesis ') ;
    GotoXY (40, 15) ; Write ('Feedback ') ;
    NewIns ('Piano1') ;
End ;

Procedure PrintAll ;
{ printe information about note, volume, voice,
    octave en ins. name }
Var
    C : Char ;
Begin
    GotoXY (14, 1) ; Write (Name : 2) ;
    GotoXY (14, 3) ; Write (Vol : 4 : 2) ;
    GotoXY (14, 5) ; Write (Octave : 2) ;
    GotoXY (14, 7) ; Write (Voice : 2) ;
    GotoXY (14, 9) ;
    If Note < > $ffff Then Begin
        C : NoteT [1+ (Note Mod 12) ] ;
        If C = '#' Then     { black key ? }
```

```
            Write (NoteT [Note Mod 12], C)    { yes }
        Else
            Write (C, '    ' ) ;
    End ;                                                    { no }
End ;

Procedure AskNewIns ;
{ ask for a new ins. name and set it }
Var
    S : String ;
Begin
    GotoXY (1, 17) ; Write ('New name (max 9 char.) : ') ;
    ReadLn (S) ;
    If S < > '  ' Then
        NewIns (S) ;
    GotoXY (1, 17) ;    { delete question }
    Write ( '                                          ' ) ;
End ;

Begin { main }
    Initialization ;
    Repeat
        PrintAll ;
        Note : = $ffff;                    { no note }
        Key : = UpCase (ReadKey) ;         { determine key press }
        Case Key Of
            ' [ ' : If Octave>0 Then Dec (Octave) ;
            ' ] ' : If Octave<7 Then Inc (Octave) ;
            ' { ' : If Vol>0 Then Vol : = Vol-0.01 ;
            ' } ' : If Vol<1 Then Vol : =Vol+0.01 ;
            ' I ' : AskNewIns ;
            '   ' : SwitchOffNote (Voice) ;
        Else
            For Counter : =1 To 12 Do                { one of the }
                If KeyB [Counter] = Key Then         { piano keys ? }
                    Note : = Octave*12+Counter-1 ;   { yes }
        End ;  { Case }
        If Note < > $ffff Then Begin       { play note ? }
            SwitchOffNote (Voice) ;        { yes, switch off previous }
            PlayNote (Voice, Note) ;       { play new }
        End ;
        Volume (Voice, Vol) ;      { set volume }
    Until Key =#27 ;               { ESC is end }
    SwitchOffNote (Voice) ;        { end of the note }
    Volume (Voice, 0) ;            { set volume at 0 }
    GotoXY (1, 17) ;               { give end message }
    WriteLn (' See you ... ') ;
    WriteLn ;
```

End.

Sound Blaster Pro的两个FM芯片可分别地加以设置。清单7.10用写端口2X8H和2X9H的方法设置的设备。其次，从左到右使用端口2X0H，2X1H，2X2H和2X3H，而设备被重建，反过来也一样，直到一键被按。当然，有很多另外的可能性。你可以用对Pro的很多函数来扩充上面的库。例如：使用第11章中所提供的信息，你可以通过左或右通道来重现一切，用设置一个FM芯片为旋律模式，而另一个FM芯片为节奏模式的办法你就可以使用15种普通设备和5种节奏设备。

程序清单7.10：sound Blaster Pro例的Pascal清单。

```
Program SBProTest;

Uses Instr, FMDirect, CRT;

Const
    Rate = 0.001 ;          { size of the decrease }
    I : InsFormat = ( Mode : 0 ; Number : 0 ;                    { a hum }
                      Modulator :
                      (KSL : 1 ; MULTI : 1 ; FB : 3 ; AR : 15 ; SL : 0 ;
                      EG_TYP : 1 ; DR : 0 ; RR : 2 ; TL : 0 ; AM : 1;
                      VIB : 1 ; SRK : 0 ; FM : 1) ;
                      Carier :
                      (KSL : 0 ; MULTI : 1  FB : 0 ; AR : 15 ; SL : 0 ;
                      EG_TYP : 1 ; DR : 0 ; RR : 2 ; TL : 0 ; AM : 1 ;
                      VIB : 1 ; KSR : 0 FM : 0) ;
                      MWafeSel : 0 ; CWafeSel : ) ;
    Ins : InsTp = (Sort : INSPt ; INS : @I) ;

Var
    Ch      : Char        { key press }
    Cht ,                 { counter }
    Vol     : Real ;      { volume }
    PCopy   : Word ;      {SBPro Port }

Begin
    Prot : =$220 ;                    {test left chip}
    If Not TestFMChip Then            { test OK? }
        Prot : = $240;                {no, other base port }
    PCopy : = Port ;
    Port : = PCopy+8 ;                { directly through SBPro }
    If TestFMChip Then Begin
        PlayMode (Melody) ;
        SetIns (1, Ins) ;             { set to both chips }
        Volume (1, 0) ;               { reproduce nothing }
        PlayNote (1, 48) ;            { switch on tone }
        Vol : = 1 ;                   { left at maximum }
```

```
        Cnt : -Rate ;                    { decrease VOL each time }
        Repeat
            Port : = PCopy ;             { left port at 0 }
            Volume (1, Vol) ;            { set volume VOL }
            Port : = PCopy +2 ;                  { right port at 2 }
            Volume (1, 1.0-Vol) ;                { set opposite VOL }
            Vol : = Vol+Cnt ;                    { decrease/increase VOL }
            If (Vol>1.0) Or (Vol<0.0) Then Begin
                Vol : = Vol-Cnt ;                { one step back }
                Cnt : -Cnt ;                     { opposite sign }
            End ;
        Until KeyPressed ;                       { repeat until key pressed }
        Ch : = ReadKey ;                         { read }
        Port : = PCopy+8 ;                       { both FM chips }
        SwitchOffNote (1) ;
    End ;
End.
```

第八章　编程CMS芯片

本章内容为：

CMS驱动程序的工作及其安装
CMS芯片的功能
用CMS芯片产生声音
用CMS芯片形成噪声
编程音调，定调和音符持续时间

CMS芯片总是Sound Blaster中最少收到注意的部分。只有Sound Blaster的版本1.0到2.0支持CMS芯片。本章所述内容适合于已拥有Sound Blaster V1.0至V2.0的用户。如果尚无CMS立体声芯片，可以用菲利普（phillips）公司的SAA1099芯片替代。

CMS芯片有其自身的功能性。它们只是标准的Sound Blaster的一部分，用以产生立体声。单这一点就使其CMS芯片具有分析研究价值。

注解　如果需要关于这些芯片能做什么和不能做什么的背景信息，请参看本书第二章“CMS立体声芯片”部分。有关安装CMS芯片的信息和与CMS芯片等同SAA1099芯片的信息，可在第四章查找。

CMS驱动程序

CMS芯片在很多方面都与FM芯片不同。为了送数据所必须写入的那些地址是不同的，是那些地址本身的函数。幸亏为了播放CMS歌曲所需的全部功能是靠CMS驱动程序CMSDRV.COM供给。

CMS歌曲具有.CMS扩展名。用户可用Creative Labs的创作软件CDMS来创作歌曲。CDMS作曲器曾随Game Blaster卡提供，Game Blaster是Soubd Blaster的前身。如果你没有Game Blaster卡，总可设法从公开盘装入CDMS作曲器。

注解　第三章中的“CDMS作曲器”详细地论述了CDMS的作曲能力。

安装CMS驱动程序　为了安装CMSDRV.COM，只需简单地运行程序，驱动程序将驻留在存贮器中。驱动程序可以通过一个中断来访问。在执行时，该程序试图接收中断80H，但如果中断80H已被其它程序所占用，那么驱动程序查找次高级的未用中断。为找到是否有一个未用中断，驱动程序检查中断指针是否在0：0。

CMSDRV中断　驱动程序接收这个未用的中断，并把中断向量指向自身（在存储器中CMSDRV的拷贝）。现在的问题是怎样得知驱动程序接收了哪一个中断？

为了找出，采用了下述的在存贮器中寻找文本“CMSDRV”的方法： 即在起始地址为80H处（文本CMSDRV在存储器中的起始地址）读出中断向量，并与0104H相加。查看80H开始的中断号并与伴随的CMSDRV的向量（加上了0104H）相比较，这样将可找到正确的中断号。

将CMS歌曲文件装入存贮器 一旦驱动程序装好了，就可以装入CMS歌曲文件了。可以将CMS文件放在存贮器的任何空区域，其位置由0段的偏移量提供。这样就可将CMS文件存入一个由段和偏移量所表示的存贮器地址中。

- 段由能寻址1024KB（1MB）的两个字节组成。
- 偏移量包含两个字节，共能访问64KB。

对这样一个系统，地址可以用不同的方式描述。例如地址66096可以写成1010H：0130H，也可以写成1023H：0000H。由于可以把偏移量的前面三位数和段的后三位数进行相加，使得系统能以每16位装入一个CMS文件。

提示 能被装入的CMS文件的最大尺寸是64KB。

CMS驱动程序的若干功能

存于存贮器中的驱动程序可完成许多功能。当然每种功能完成与CMS歌曲有很多关系。下面对CMSDRV.COM驱动程序的功能进行分析。

功能0

功能0实现回送驱动程序的版本号，版本号用一个点号分开的两个数来表示。

输入	ah	= 0
输出	ah	= 点号前部分的数字
	al	= 点号后部分的数字

功能1

功能1完成播放CMS音乐。驱动程序应该知道哪个段包含了这音乐（偏移量不需要，因它总为0）。

驱动程序用了一个包含驱动程序状态的播放标志，该标志是由用户选择的一个存贮器字节。假如标志中的内容等于255，表示驱动程序正在播放CMS音乐，如果内容为0，则表示驱动程序未用。

输入	ah	= 1
	al	= CMS乐曲需要播放的遍数，在1到255间变化；若其值为0，表示要播放的音乐不停止地播放。
	es	= “播放标志”的段地址。
	bx	= “播放标志”的偏移地址。

	cx	= CMS音乐的段地址。
输出	ax	= 0，意味着一切正常，音乐正在播放。
	ax	= 1，意味着所指示的CMS音乐有错误或者它不是一CMS音乐。
	ax	= 2，表示其CMS音乐是用驱动程序不支持的创作版本制作的。

功能2

功能2使音乐暂停

输入	ah	= 2
输出	ax	= 0，意味着操作已完成，音乐已暂停。
	ax	= 1，表示没有音乐在被播放，所以不能被暂停。

功能3

功能3的作用是结束暂停，并确保音乐继续播放。这种功能仅仅对音乐被暂停才有效。

输入	ah	= 3
输出	ax	= 0 意味着操作已经完成，音乐继续播放。
	ax	= 1 表示无音乐被暂停过。

功能4

功能4是停止CMS歌曲

输入	ah	= 4
输出	无	

功能5

功能5作用是取消控制键（Ctrl）+键5实现功能的能力。驱动程序还可用计算机键盘，使用下列按键进行控制（所有的数字是在数字键区）：

击键	**作用**
Ctrl+5三次	停止乐曲。
Ctrl+5二次	暂停乐曲。
Ctrl+5一次	启动乐曲。

使用功能5则取消了键盘的这些能力。

输入	ah	= 5
输出	无	

播放CMS歌曲

当你学习了一个例子，对怎样使用驱动程序就会变得一清二楚。本节介绍的样板程序是通过完成下面的一些步骤来实现播放CMS音乐的，其步骤是：

1. 检查CMSDRV.COM是否存在，并寻找中断号。
2. 把CMS文件装入一个偏移量为0的段中。
3. 保存一个播放标志。
4. 通知驱动程序播放CMS音乐。
5. 如果输出不等于0，离开程序，显示一个出错信息。
6. 等待用户输入，由四个指定的键执行播放、暂停、继续和停止的功能。
7. 如果用户按了停止键，则退出程序。

注意，在未撤消Ctrl+5功能的情况下，也可以用Ctrl+5键来替代四个指定键实现停止、启动和暂停乐曲的操作。

注解　有关阅读和使用本书的程序清单的建议，见序言中“有关程序清单的说明”。

程序清单8.1 播放CMS歌曲的Pascal程序单元

```
{ ********************************************
** Unit CMS supporting PLAYCMS.PSA          **
** Turbo Pascal 4.0                         **
******************************************** }

Unit CMS ;

INTERFACE

Uses Crt, Dos ;
var Play_flag : byte ;
    songseg : word ;
    intno : byte ;

Function CheckDrv : byte ;
Function PlaySong : boolean ;
Function Pause : boolean ;
Function Continue : boolean ;
Procedure stop ;
Procedure unlock_key ;

IMPLEMENTATION

Function CheckDrv : byte ;                { see if the driver }
```

```
const header : string [6] = 'CMSDRV' ;     { is present is memory }

var vect : byte ;
    found : boolean ;
    regs : Registers ;
    temp : string [6] ;

begin
    vect : =$80 ;
    found : =false ;
    while not found and (vect<>0) do
    begin
       regs.ah : =53 ;          { function 53 of int 21H }
       regs.al : =vect ;        { reads a vector }
       intr ($21, regs) ;

       Move (mem[regs.es : $104], temp[1], 6) ; temp[0] : =#6 ;
                                  { copy to a buffer }
       if temp=header then        { set length at 6 }

          found : =true
       else
          Inc(vect) ;
  end ;
  checkdrv : =vect ;
end ;

Function PlaySong : boolean ;      { start playing }
var regs : registers ;

begin
   regs. ah :=1 ;
   regs. al :=1 ;
   regs. es :=Seg(Play_flag) ;
   regs. bx :=Ofs(Play_flag) ;
   regs. cx :=songseg ;
   intr(intno, regs) ;
   if (regs. zx<>0)
   then
      playsong :=true
   else
      playsong :=false ;
end ;

Function Pause : boolean ;   { pause the song }
var regs : registers ;
begin
   regs. ah :=2 ;
```

```
   intr (intno, regs) ;
   if (regs. ax<>0)
   then
      pause :=true
   else
      pause :=false ;
end ;

Function Continue : boolean ;        { end pause }
var regs : registers ;
begin
   regs. ah :=3 ;
   intr (intno, regs) ;
   if regs. ax <>0
   then
      continue :=true
   else
      continue :=false ;
end ;

Procedure stop ;              { stop playing }
var regs : registers ;

begin
   regs. ah :=4 ;
   intr(intno, regs) ;
end;

Procedure unlock_key ;        { unlock numeric keypad }
var regs : registers ;
begin
   regs. ah :=5 ;
   intr(intno, regs) ;
end ;

begin
end.
```

程序清单8.2 播放CMS歌曲的Pascal实例

```
{ ********************************************
** PLAYCMS. PAS                              **
** Turbo Pascal 4.0                          **
******************************************** }

Program PlayCMS ;

uses Crt, Fos, Cms ;
```

```
var mempos : pointer ;
    size : word ;
    key : char ;
    result : integer ;

Function loadfile(filename : string) : byte ;  { load a }
var handle : file ;                            { file at the beginning }
    regs : registers ;                         { of a segment }

begin
    {$I-}
    Assign(handle, filename) ;
    Reset(handle, 1) ;
    size := filesize(handle) ;
    GetMem(mempos, (size+15) AND $FFF0) ;
    songseg := Seg(mempos^) ;
    BlockRead(handle, mem[songseg : 0], size) ;
    close(handle) ;
    {$I+}
    If ioresult<>0 then loadfile := 1 else loadfile := 0 ;
end ;

Procedure choice ;
begin
    Write(' Press P-pause, U-unpause, S-stop ') ;
    Writeln(' or M-music maestro!') ;
end ;
                                { main program }
begin
    if paramcount<>1 then
    begin
        Writeln('Use : Playcms < cms-file >') ;
        halt(1) ;
    end ;

    intno := checkdrv ;
    if intno =0 then
    begin
        Writeln('CMSDRV.COM not loaded.') ;

        halt(2) ;
    end
    else
        Writeln('CMSDRV.COM found, interrupt : ' , intno) ;

    result :=loadfile(paramstr(1)) ;
    if result<>0 then
```

```
    begin
      if result=2 then
        rwiteln(' Not enough memory available.')
      else
        writeln('Loading error ', paramstr(1)) ;
      halt(3) ;
    end ;

    choice ;

    Repeat
      If Keypressed then
        key :=Upcase(readkey)
      else
        key :=#0 ;
      case key of              { process choice }

        'M' : if playsong then
                writeln('Song starting error..')
              else
                choice ;

        'P' : if pause then
                writeln('No music is playing.')
              else
                choice ;

        'U' : if continue then
                writeln(' Nothing paused.')
              else
                choice ;
      end;
    Until (key='S') ;
    stop ;
    unlock_key ;
    freemem(mempos, (size+15) AND $FFF0) ;
end.
```

程序清单8.3 播放CMS歌曲的C程序单元

```
/* ********************************************
** Include file CMS.H in PLAYCMS.C            **
** Turbo C++ 2.0                              **
******************************************** /

typedef unsigned char byte ;
typedef unsigned int word ;
word Play_flag ;
```

```
word segm ;          /* song segment */
word intno ;         /* interrupt number of the driver */
byte intno ;
checkdrv()
{
  byte vect=0x80 ;              /* start searching at int. 80H */
  byte found=0 ;

int number=0 ;
char chr[6]="CMSDRV" ;          /* the identification */
union REGS inregs, outregs ;
struct SREGS segregs ;
byte far *oldvect ;   /* pointer to old vector */

while (!found && vect !=0)
{
  inregs.h.ah=53 ;          /* function 53 of int 21H */
  inregs.h.al=vect ;        /* asks for the vector of an int.*/
  int86x(0x21, &inregs, &outregs, &segregs) ;
  outregs.x.bx=0x0104 ;             /* es : bx returns the vector */
                                    /* set bx at 0104H */

  oldvect=MK_FP(segregs.es, outregs.x.bx) ;
  for(number=0 ; oldvect[number]==chr[number]; number++) ;

  if (number==7)            /* compare this vector to */
    found=1 ;                       /* the identification */
  else
    vect++ ; /* if unequal-> next vector */
  }

  return(vect) ;       /* return number found */
}

playsong()
{
  union REGS inregs , outregs ;
  struct SREGS segregs ;

  inregs.h.ah=1 ;           /* function 1 of the CMSDRV */
  inregs.h.al=1 ;
  segregs.es=FP_SEG(Play_flag) ;
  inregs.x.bx=FP_OFF(Play_flag) ;
  inregs.x.cx=segm ;
  int86x(intno, &inregs, &outregs, &segregs) ;
  if (outregs.x.ax!=0) return(1) ;
  return(o) ;
}
```

```
pause()

{
   union REGS inregs, outregs ;
   inregs.h.ah=2 ;              /* function 2 of the CMSDRV */
   int86(intno, &inregs, &outregs) ;
   if (outregs.x.ax!=0) return(1) ;
   return(0) ;
}

continue()
{
   union REGS inregs, outregs ;

   inregs.h.ah=3 ;              /* function 3 of the CMSDRC */
   int86(intno, &inregs, &outregs) ;
   if (outregs.x.ax!=0) return(1) ;
   return(0) ;
}

void stop()
{
   union REGS inregs, outregs ;
   inregs.h.ah=4;               /* function 4 of the CMSDRV */
   int86(intno, &inregs, &outregs) ;
}

void unlock_key()
{
   union REGS inregs, outregs ;
   inregs.h.ah=5 ;              /* function 5 of the CMSDRV */
   int86(intno, &inregs, &outregs) ;
}
```

程序清单8.4 播放CMS歌曲的C实例

```
/* ************************************
** PLAYCMS.C                          **
** Turbo C++ 2.0                      **
**************************************** /

#include<dos.h>
#include<fcnt1.h>
#include"cms.h"

extern word Play_flag ;
extern word sgem ;
extern byte intno ;
```

```
loadfile(char *filename)        /* read file at the */
{                               /* beginning of a segment */
   int handle ;
   long filesize ;
   struct REGPACK reg ;

   if ( (handle=open(filename, O_RDONLY | O_BINARY ))==1)
      return(1) ;
   filesize = filelength(handle) ;
   if (allocmem(((filesize+16)>4), &segm) !=1)
      return(2) ;
   reg.r_ax=Ox3f00 ;
   reg.r_bx=handle ;
   reg.r_cx=filesize ;          /* function 3th of int 21H */
   reg.r_dx=0 ;                 /* reads a file at an */
   reg.r_ds=segm ;              /* absolute address */
   intr(Ox21, &reg) ;
   if (reg.r_flags & 1)
   {
      freemem(segm) ;
      return(1) ;
   }
   close(handle) ;
   return(0) ;
}

void choice()
{
   printf("/nPress p-pause, u-unpause, s-stop ") ;
   printf("of m-music maestro!\n") ;
}

main (int argc, char *argv[ ])
{
   chr key ;
   int result ;

   if ( !(argc==2) )
   {
      printf("Use: playcms < cms-file > \n") ;
      return(-1) ;
   }

   if ( (intno=checkdrv())==0 )
   {
      printf("CMSDRV.COM not loaded.. \n") ;
      return(-2) ;
```

```
}

if (!((result=loadfile(argv[1]))==0 ))
{

if (result==2)
  printf("Not enough memory available.\n") ;
else
  printf("Loading error %s.\n", argv[1]) ;
return(-3) ;
}
choice() ;
while ((key=toupper((key=getch())))!='S')
{
  switch(key)                /* process choice */
  {
    case 'M':
      if (playsong()==1)
      {
        printf("Song starting error..\n") ;
        break ;
      }
      choice() ;
      break ;
    case 'P' :
      if (pause()==1)
      {
        printf("No music is playing.\n") ;
        break ;
      }
      choice() ;
      break ;
    case "U" :
      if (continue()==1)
      {
        printf("Nothing paused.\n") ;
        break ;
      }
      choice() ;
      break ;
    default :
      break ;
  }
}

stop() ;
unlock_key() ;
freemem(segm) ;
```

```
    return(0) ;
}
```

怎样用CMS芯片产生声音

由于有两个立体声芯片，所以可以认为用一个芯片控制左声道的发声，而另一芯片控制右声道。但并非一定是这样，甚至去掉一个芯片，仍可以在某些声部控制立体声。

每一CMS芯片中配置有六个单独的声部，因为每一芯片有它自己的控制端口，所以两块CMS芯片之间并不发生冲突。

- 第一块CMS芯片控制声部0到5，端口地址为2X1H和2X0H。
- 第二块CMS芯片控制声部6到11，所用端口地址为2X3H和2X2H。其中X是一个十六进制数，系I/O跳接器设定的。Sound Blaster I/O跳接器的缺省设置是220H，故此时X用2替代。

变址寄存器组 为了控制六个声部，每块CMS芯片中有32个内部寄存器用来控制六个声部的升高和降低。这些寄存器是只写入的寄存器，仅仅能被写。

由于卡上的每个寄存器有自己的端口，要占用太多的存贮器地址，所以必须采用变址方式。为此，先将欲改变的寄存器的编号送到一个端口，然后把想写入该寄存器的内容送到另一端口。这里，一个端口告诉插板放置信息的地方，另一个端中则放置信息本身。

为了方便起见，我们假定I/O跳接器设置为220H，那么第一块CMS芯片的端口地址221H和第二块CMS芯片的端口地址223H指示寄存器的地址，送到这些端口的编号必须是在0到31之间的数。第一块与第二块CMS芯片的端口地址220H和222H中分别包含了需要写到寄存器地址中的值。

假定要将数值32写到第二块CMS芯片的寄存器3中（为了控制6到11之间的一个声部），而第二块芯片的地址端口为223H，数据口为222H，那么首先应将数值3送入地址口223H，这样芯片就知道送到数据端口的数值一定是写到3号寄存器，然后把数值32送入数据端口222H。

如前所述，这32个寄存器包含了产生六个声部的声音所需要的信息。每个声部音调的音色取决于该音调的频率、音阶、幅度和包络等参量。

控制音调的音阶、幅度和包络

下面这一节将阐述怎样控制音调的频率，还讨论怎样控制音调的音阶、幅度以及包络等问题。

控制音调的频率

每块CMS芯片中有六个频率发生器供使用。

在一块CMS芯片中，用六个寄存器保存每个频率发生器的音调值，音调值寄存器为八位，所以可包含的音调值在0到255之间。音调的高低和音调的频率是同一回事，频率通常用赫兹为单位测量。现给出计算音调频率的公式如下：

频率（HZ） = （15625*（2^音阶数））/（511-音调值）

控制音调的音阶

因为用0-255的音调值来表示很多现存的音调高低是不够的，所以音调被划分成八个音阶。每个音阶有自己的频率范围，表8.1示出了八个音阶及相应的频率范围。

表8.1　八个音阶和它们的频率范围

音阶	范围
0	31-61HZ
1	61-122HZ
2	122-244HZ
3	244-488HZ
4	489-977HZ
5	978-1.95KHZ
6	1.96-3.91KHZ
7	3.910-7.81KHZ

用三个二进制位正好表示八个音阶，所以两个音阶值可装在一个字节中，这就是为什么只用三个寄存器就能存放六个频率发生器的音阶值的原因。

控制音调的幅度

幅度对应于音调的声音强度，对每种音阶，声音的强度可以分别设置。当其在建立幅度时，由于可以为左、右两个声道设定幅度，这就包含了立体声部分。

有六个寄存器，每种音阶一个。其中字节的高四位给右声道作为确定幅度用，低四位供左声道确定幅度用，这样，左右声道音量输出皆可以有16级调节。

控制音调的包络

用了两个包络产生器来指示音调的幅度方向。包络产生器能形成以下波形：

波形格式	说　明
幅度为0	无音量。
最大幅度值	最大音量。
单个衰减	音量单个线性减小。
重复衰减	音量反复线性递减，其波形如图8.1所示。
单个三角形	音量线性增加，随后线性减小。
重复三角形	音量重复线性增加，随后线性减小。
单个冲击	音量单个线性增高。
重复冲击	音量重复线性增高。

对冲击和衰减，当音量达到最大时，音量立即降低到零。在用了包络产生器后，最大音

量回复到幅度寄存器中最大值的7/8。

有两个包络寄存器，一个用于频率产生器0-2，另一个用于频率产生器3-5。对于8种包络每一种需要用寄存器的3位。表8.2示出了一个完整的包络字节的含义。

当音阶达到5，6和7时，频率的级距变大，所以需用4比特数来表示。

表8.2　包络产生器

位的编号	状态（0=截止） （1=接通）	说明
0	0	同一个包络应用于左、右二者输出。
	1	对于左和右输出的包络是相反的。
3.2.1	000	无幅度。
	001	最大幅度。
	010	单个衰减。
	011	重复衰减。
	100	单个冲击+衰减（=三角形）。
	101	重复三角形。
	110	单个冲击。
	111	重复冲击。
4	0	4位包络分辨率。
	1	3位包络分辨率。
5	0	内部包络时钟;频率产生器1或4。
	1	外部包络时钟;靠写入该寄存器字节中。
6	0，1	无功能。
7	0	包络产生器关。
	1	包络产生器开。

外部和内部时钟　为了确定重复包络何时应开始冲击或衰减，包络产生器采用了一个内部或外部时钟。当采用内时钟时，使用了频率产生器1（对于第一个包络产生器）或者频率产生器4（对于第二个包络产生器），这时不能再使用这些频率产生器。

事实上，你本身就是个外部时钟，通过写入相应的包络产生器的寄存器，你就按照已建立的包络开始一个新的冲击或衰减。

产生噪声

一个声音芯片必须要能做比再现音符更多的事，因为只有各个音符不足以产生现在很多游戏中要求的大范围的声响效果。因此，噪声产生器用来产生各种各样的声音效果。

噪声产生器使用的是随机数，相反，频率产生器使用正弦波形成音乐。这些随机数构成了各种噪声，噪声产生器产生的声音通常认为是白噪声。

每一CMS芯片内有两个噪声产生器，其噪声的频段——即挑选那些新随机数的速度——取决于噪声产生器寄存器中的两位。噪声产生器寄存器分别用两位给噪声产生器0和噪声产生器2。这些位的值决定了噪声的频率，其值是：

位0	位1	频率值
0	0	31.25KHZ
0	1	15.6KHZ
1	0	7.8KHZ
1	1	61HZ-15.6KHZ

当位0和位1皆为1时，分别由频率产生器寄存器0和3中的值来决定噪声产生器0和噪声产生器1的频率值。

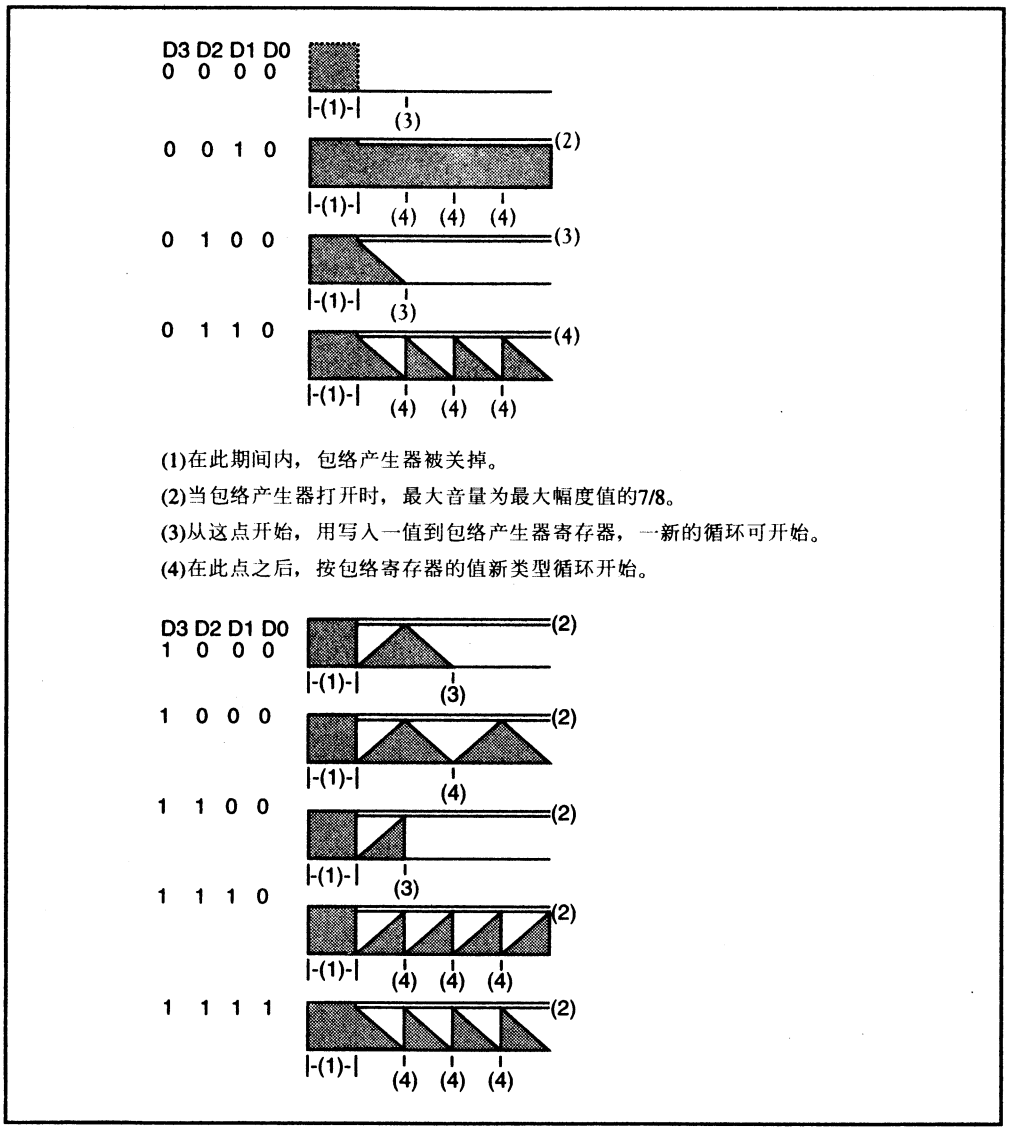


图8.1　不同波形

混合音调和噪声

CMS芯片中有六个混合器，每种声部一个。

用混合器你可以混合曲调和噪声。两个内部寄存器留作告知混合器，它们必须让曲调或噪声中的哪一个通过。在这两个寄存器中，规定用六位来指示一个对应的声部。第一个寄存器指示是否让其音调通过，当这位置为1时，表示由频率产生器产生的音调可以通过。第二个寄存器决定是否允许噪声产生器的声音通过。表8.3示出了两寄存器其他组合。

图8.2示出了CMS芯片各个部分的连接关系，它可使读者更清楚地了解CMS芯片的结构。

内部寄存器地址 在本章的前面部份对有关的寄存器进行了讨论，但未指出每个产生器的地址是什么值。表8.4和表8.5示出了所有内部寄存器和它们的地址以及各数据位。表8.4对应于第一个CMS芯片，而表8.5对应于第二块CMS芯片。在这两个表中，地址编号是以十六进制数标记的，X号表示该位未用。第一块CMS芯片中的寄存器地址端口为2X1H，数据端口为2X0H，第二块CMS芯片寄存器的地址端口是2X3H，数据端口是2X2H。其中X表示值可为1，2，3，4，5或6，由I/O跳接器的设置而具体决定。

表8.3 混合音调的各位

频率-允许寄存器	噪声-允许寄存器	说明
0	0	既不让音调也不让噪声通过
0	1	只允许噪声通过
1	0	只允许音调通过
1	1	噪声和音调两者都通过

决定音调的长度

CMS芯片存在的一个问题是它们没有用以确定或测量音符长度的内部时钟，这些芯片不能终止它们产生的音调，要由程序人员来负责完成。

有两种决定音调长度的方法：内部时钟法和定时中断法

- 如果你使用时钟来求出自音符开始所通过的时间，则要确保你的应用程序只是操纵这些CMS芯片。例如，一项从事计算分形图象或别的某些操作的应用，在时间读数期间，可能断开时钟百分之几秒。
- 用定时器中断的方法，你增加调用定时器中断的次数，并用一个计数器来精确地记录音符的持续时间。

采用定时器方法的优点是应用程序可以继续执行别的指令，由中断程序对付音调和声音效果的处理。

音符和它们的音阶的频率

表8.6给出了全部音符以及它们的音阶和频率值，通过这张表，可以确定每个音符的音阶和频率。

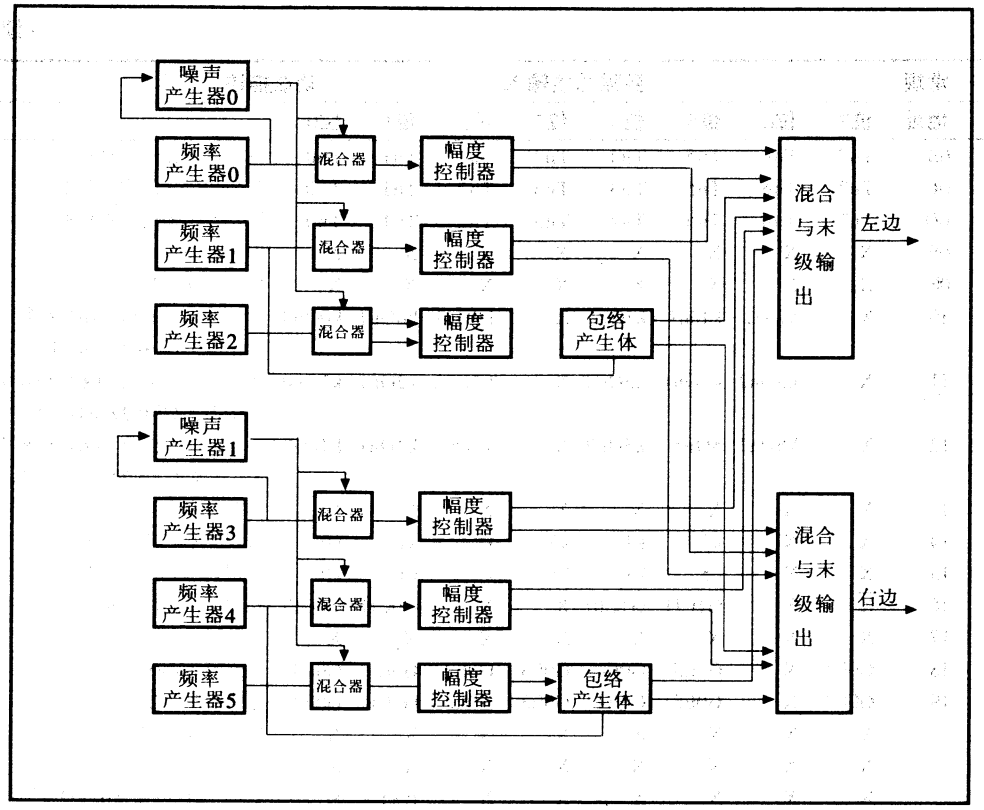


图8.2　CMS芯片的结构

表8.4　第一块CMS芯片

常规	数据总线输入								功能描述
地址	位7	位6	位5	位4	位3	位2	位1	位0	
00	Amr3	AmR2	AmR1	AmR0	AmL3	AmL2	AmL1	AmL0	左端声音和右端声音幅度值0
01	Amr3	AmR2	AmR1	AmR0	AmL3	AmL2	AmL1	AmL0	左端声音和右端声音幅度值1
02	Amr3	AmR2	AmR1	AmR0	AmL3	AmL2	AmL1	AmL0	左端声音和右端声音幅度值2
03	Amr3	AmR2	AmR1	AmR0	AmL3	AmL2	AmL1	AmL0	左端声音和右端声音幅度值3
04	Amr3	AmR2	AmR1	AmR0	AmL3	AmL2	AmL1	AmL0	左端声音和右端声音幅度值4
05	Amr3	AmR2	AmR1	AmR0	AmL3	AmL2	AmL1	AmL0	左端声音和右端声音幅度值5
06	X	X	X	X	X	X	X	X	作为扩充专用
07	X	X	X	X	X	X	X	X	作为扩充专用
08	Tn7	Tn6	Tn5	Tn4	Tn3	Tn2	Tn1	Tn0	频率产生器的音调值0
09	Tn7	Tn6	Tn5	Tn4	Tn3	Tn2	Tn1	Tn0	频率产生器的音调值1
0A	Tn7	Tn6	Tn5	Tn4	Tn3	Tn2	Tn1	Tn0	频率产生器的音调值2

（续表）

常规				数据总线输入					功能描述
地址	位7	位6	位5	位4	位3	位2	位1	位0	
0B	Tn7	Tn6	Tn5	Tn4	Tn3	Tn2	Tn1	Tn0	频率产生器的音调值3
0C	Tn7	Tn6	Tn5	Tn4	Tn3	Tn2	Tn1	Tn0	频率产生器的音调值4
0D	Tn7	Tn6	Tn5	Tn4	Tn3	Tn2	Tn1	Tn0	频率产生器的音调值5
0E	X	X	X	X	X	X	X	X	作为扩充专用
0F	X	X	X	X	X	X	X	X	作为扩充专用
10	X	k10n2	k10n1	k10n0	X	k0On2	k0On1	k0On0	频率产生器1（位4-6）和频率产生器0（位0-2）的音阶码
11	X	k3On2	k3On1	k3On0	X	k2On2	k2On1	k2On0	频率产生器3（位4-6）和频率产生器2（位0-2）的音阶码
12	X	k5On2	5On1	k5On0	X	k4On2	k4On1	k4On0	频率产生器5（位4-6）和频率产生器4（位0-2）的音阶码
13	X	X	X	X	X	X	X	X	作为扩充专用
14	X	X	k5	k4	k3	k2	k1	k0	可用作每种声音的频率
15	X	X	k5	k4	k3	k2	k1	k0	可用作每种声音的噪声
16	X	X	G1Cf1	G1Cf0	X	X	G0Cf1	G0Cf0	噪声间生器时钟频率
17	X	X	X	X	X	X	X	X	作为扩充专用
18	G0E7	X	G0E5	G0E4	G0E3	G0E2	G0E1	G0E0	包络产生器0
19	G0E7	X	G0E5	G0E4	G0E3	G0E2	G0E1	G0E0	包络产生器2
1A	X	X	X	X	X	X	X	X	作为扩充专用
1B	X	X	X	X	X	X	X	X	作为扩充专用
1C	X	X	X	X	X	X	RST	SE	复位/所有声部（0到5）的声音
1D	X	X	X	X	X	X	X	X	作为扩充专用
1E	X	X	X	X	X	X	X	X	作为扩充专用
1F	X	X	X	X	X	X	X	X	作为扩充专用

表8.5　第二块CMS芯片

常规				数据总线输入					功能描述
地址	位7	位6	位5	位4	位3	位2	位1	位0	
00	Amr3	AmR2	AmR1	AmR0	AmL3	AmL2	AmL1	AmL0	左端声音和右端声音幅度值6
01	Amr3	AmR2	AmR1	AmR0	AmL3	AmL2	AmL1	AmL0	左端声音和右端声音幅度值7
02	Amr3	AmR2	AmR1	AmR0	AmL3	AmL2	AmL1	AmL0	左端声音和右端声音幅度值8
03	Amr3	AmR2	AmR1	AmR0	AmL3	AmL2	AmL1	AmL0	左端声音和右端声音幅度值9
04	Amr3	AmR2	AmR1	AmR0	AmL3	AmL2	AmL1	AmL0	左端声音和右端声音幅度值10
05	Amr3	AmR2	AmR1	AmR0	AmL3	AmL2	AmL1	AmL0	左端声音和右端声音幅度值11
06	X	X	X	X	X	X	X	X	作为扩充专用
07	X	X	X	X	X	X	X	X	作为扩充专用

（续表）

常规	数据总线输入								功能描述
地址	位7	位6	位5	位4	位3	位2	位1	位0	
08	Tn7	Tn6	Tn5	Tn4	Tn3	Tn2	Tn1	Tn0	频率产生器的音调值6
09	Tn7	Tn6	Tn5	Tn4	Tn3	Tn2	Tn1	Tn0	频率产生器的音调值7
0A	Tn7	Tn6	Tn5	Tn4	Tn3	Tn2	Tn1	Tn0	频率产生器的音调值8
0B	Tn7	Tn6	Tn5	Tn4	Tn3	Tn2	Tn1	Tn0	频率产生器的音调值9
0C	Tn7	Tn6	Tn5	Tn4	Tn3	Tn2	Tn1	Tn0	频率产生器的音调值10
0D	Tn7	Tn6	Tn5	Tn4	Tn3	Tn2	Tn1	Tn0	频率产生器的音调值11
0E	X	X	X	X	X	X	X	X	作为扩充专用
0F	X	X	X	X	X	X	X	X	作为扩充专用
10	X	k70n2	k70n1	k70n0	X	k60n2	k60n1	k60n0	频率产生器7（位4-6）和频率产生器6（位0-2）的音阶码
11	X	k90n2	k90n1	k90n0	X	k80n2	k80n1	k80n0	频率产生器9（位4-6）和频率产生器8（位0-2）的音阶码
12	X	k110	n2110n1	k110n0	X	k100n2	k100n1	k100n0	频率产生器11（位4-6）和频率产生器10（位0-2）的音阶码
13	X	X	X	X	X	X	X	X	作为扩充专用
14	X	X	V11	V10	V9	V8	V7	V6	可用作每种声音的频率
15	X	X	V11	V10	V9	V8	V7	V6	可用作每种声音的噪声
16	X	X	G1Cf1	G1Cf0	X	X	G0Cf1	G0Cf0	噪声间生器时钟频率
17	X	X	X	X	X	X	X	X	作为扩充专用
18	G0E7	X	G0E5	G0E4	G0E3	G0E2	G0E1	G0E0	包络产生器2
19	G0E7	X	G0E5	G0E4	G0E3	G0E2	G0E1	G0E0	包络产生器3
1A	X	X	X	X	X	X	X	X	作为扩充专用
1B	X	X	X	X	X	X	X	X	作为扩充专用
1C	X	X	X	X	X	X	RST	SE	复位/所有声部（6到11）的声音
1D	X	X	X	X	X	X	X	X	作为扩充专用
1E	X	X	X	X	X	X	X	X	作为扩充专用
1F	X	X	X	X	X	X	X	X	作为扩充专用

抓小偷：应用举例

本节将通过一个实例介绍怎样使用CMS芯片及其寄存器。

例子是一个小游戏，CMS芯片用来确定玩游戏者的方向。在这个称为抓小偷的游戏中，有一个盗贼背着他新近偷的装有钱的包，沿着一块16*16大小的区间在走，包内除钱外还装有一台发射机，发射机每半秒钟发出一次立体声信号，譬如当左边的声音信号比右边的强时，表示坏人在你左边，当你接近盗贼时，信号频率变得较高。你的任务是在一分钟之内抓住这个盗贼。显示屏上所显示的除你——侦探而外还有一堵墙，除每10秒盗贼出现时你能看见他外，平时你是看不见这个盗贼的。

后面给出了抓盗贼游戏的程序清单

表8.6　音符和它们的音阶和频率值

音符	音阶数	频率值	音符	音阶数	频率值	音符	音阶数	频率值
A	0	3	A	3	3	A	6	3
A#	0	31	A#	3	31	A#	6	31
B	0	58	B	3	58	B	6	58
C	0	83	C	3	83	C	6	83
C#	0	107	C#	3	107	C#	6	107
D	0	130	D	3	130	D	6	151
D#	0	151	D#	3	151	D#	6	151
E	0	172	E	3	172	E	6	172
F	0	191	F	3	209	F	6	172
F#	0	209	F#	3	209	F#	6	209
G	0	226	G	3	226	G	6	226
G#	0	242	G#	3	242	G#	6	242
A	1	3	A	4	3	A	7	3
A#	1	31	A#	4	31	A#	7	31
B	1	58	B	4	58	B	7	58
C	1	83	C	4	83	C	7	83
C#	1	107	C#	4	107	C#	7	107
D	1	130	D	4	130	D	7	130
E	1	172	E	4	172	E	7	172
F	1	191	F	4	191	F	7	191
F#	1	209	F#	4	209	F#	7	209
G	1	226	G	4	226	G	7	226
G#	1	242	G#	4	42	G#	7	242
A	2	3	A	5	3			
A#	2	31	A#	5	31			
B	2	58	B	5	58			
C	2	83	C	5	83			
C#	2	107	C#	5	107			
D	2	130	D	5	130			
D#	2	151	D#	5	151			
E	2	172	E	5	172			
F	2	191	F	5	191			
F#	2	209	F#	5	209			
G	2	226	G	5	226			
G#	2	242	G#	5	242			

程序清单8.5：支持抓小偷游戏的Pascal程序单元

```
{ ********************************************
** Unit CMSNOTE supporting THIEF.PAS        **
** Turbo Pascal 4.0
******************************************** }

Unit CmsNote ;
```

```
INTERFACE

Uses Crt, Dos ;

const baseport=$220 ;          { selected port }
      OldTimV=103              { old timer interrupt }
var newtimepointer : pointer ;

Procedure settimer(Freq : word ; Rout : pointer) ;
Procedure resettimer ;
Procedure PlayNote(voice, ampl, tone octave,
                   sort, noiseclock, envelope : byte ;
                   duration : integer) ;

Procedure ResetCms ;
Procedure InitCms ;

IMPLEMENTATION

var timevoice : array[0..11] of integer ;      { buffers }
    comsonfreq : array[0..11] of byte ;
    cmsonnoise : array[0..11] of byte ;
    counter : byte ;
    freqtot : byte ;
    noisetot : byte ;

procedure settimspeed(Freq : word) ; { calculate the }
var             { number of clock ticks and }
   ICnt : longint ;          { pass this to the timer chip }
begin
          inline(#FA) ;
       ICnt := 1193180 div Freq ;
       port[$43] :=$36 ;
       port[$40] :=lo(ICnt) ;
       port[$40] :=hi(ICnt) ;
          inline($FB) ;
end ;

procedure settimer(Freq : word; Rout : pointer) ;
var
   OldV : pointer ;                { take over timer }
begin
          inline($FA) ;
          getintvec(8, OldV) ;
          setintvec(OldtimV, Oldv) ;
       setintvec(8, Rout) ;
       settimspeed(Freq) ;
          inline($FB) ;
```

```
end ;

procedure resettimer ;
Var
   OldV : pointer ;  { reset vector and speed }
begin
        inline($FA) ;
      port[$43] :=$36 ;
      port[$40} :=$0 ;
      port[$40] :=$0 ;
        getintvec(OldtimV, Oldv) ;
      setintvec(8,Oldv) ;
        inline($FB) ;
end ;

Pricedure PlayNote(voice, ampl, tone, octave ,
                   sort, noiseclock, envelope : byte ;
                   duration : integer) ;

var cmsvoice, cmsport : integer ;     { initialize all }
   i, j, k, l : byte ;  { registers and set flags }

begin
   cmsport :=baseport ;
   cmsvoice :=voice ;
   if(voice>=6) then  { bigger than 5 ; other chip }
   begin
      Dec(voice, 6) ;
      Inc(cmsport, 2) ;
end ;
port[cmsport+1] :=voice ;
port[cmsport] :=ampl ;          { amplitude }
port[cmsport+1] :=8+voice ;
port[cmsport] :=tone ;          { frequency }
port[cmsport+1] :=$10+(voice DIV 2) ;

j :=octave SHL ((voice AND 1) *4) ;
port[cmsport] :=j ;             { octave }
port[cmsport+1] :=$16 ;

k :=noiseclock SHL (((voice DIV 3) AND 1)*4) ;
port[cmsport] :=k ;                    { noise frequency }
port[cmsport+1] :=$18+(voice DIV 3) ;
port[cmsport] :=envelop ;              { envelope }

if (sort AND 2)=2 then
   cmsonfreq[cmsvoice] :=1 ;           { frequency voice }
if (sort AND 1)=1 then
```

```
    cmsonnoise[cmsvoice] :=1 ;          { noise voice }

    timevoice[cmsvoice] :=duration ;  { and the length of the note }
end ;

Procedure NewTimer ; INTERRUPT ;          { decrease the duration of }
var freq, noise : byte ;                    { all voices and pass }
    i, j : integer ;                      { the total of playing }
    cmsport : word ;                  { voices on to the chips }
    Regs : registers ;
begin
  For i :=0 to 11 do
    begin
      if ((cmsonfreq[i]=1) OR (cmsonnoise[i]=1)) then
      begin
        if (timevoice[i]>0) then
          Dec(timevoice[i])
        else
          begin
            cmsonniser[i] :=0 ;
            cmsonfreq[i] :=0 ;
          end ;
      end ;
  end ;
  freqtot := 0 ;
  noisetot := 0 ;
  cmsport :=baseport ;

For i :=0 to 5 do
  begin
    if (cmsonfreq[i]=1) then Inc (freqtot, (1 SHL i)) ;
    if (cmsonnoise[i]=1) then Inc (noisetot, (1 SHL i)) ;
  end ;
  port[cmsport+1] :=$14 ;
  port[cmsport] :=freqtot ;
  port[cmsport+1] :=$15 ;
  port[cmsport] :=noisetot ;

  freqtot :=0 ;
  noisetot :=0 ;

   for i :=6 to 11 do
   begin
      if (cmsonfreq[i]=1) then Inc (freqtot, (1 SHL (i-6))) ;
      if (cmsonnoise[i]=1) then Inc (noisetot, (1 SHL (i-6))) ;
   end ;
   port[cmsport+3] :=$14 ;
   port[cmsport+2] :=freqtot ;
```

```
    port[cmsport+3] :=$15 ;
    port[cmsport+2] :=noisetot ;

    Inc(counter) ;
    Intr(oldtimV, Regs) ;          { old timer interrupt }

end ;

Procedure ResetCms ;               { set the reset register }
begin
    port[baseport+1] :=$1c ;
    port[baseport] :=2 ;
    port[baseport+3] :=$1c ;
    port[baseport+2] :=2 ;
end ;

Procedure InitCms ;
var cnt : unteger ;

begin
    resetcms ;
    port[baseport+1] :=$1c ;       { enable sound }
    port[baseport] :=1 ;
    port[baseport+3] :=$1c ;
    port[baseport+2] :=1 ;
    For cnt :=0 to 11 do
    begin
        timevoice[cnt] :=0 ;
        cmsonfreq[cnt] :=0 ;
        cmsonnoise[cnt] :=0 ;
        playnote(cnt, 0,0,0,0,0,0,0) ;
end ;

port[baseport+1] :=$14 ;
port[baseport] :=255 ;
port[baseport+3] := $14 ;
port[baseport+2] :=255 ;
port[baseport+1] :=$15 ;
port[baseport] :=0 ;
port[baseport+3] :=$15 ;
port[baseport+2] :=0 ;
end ;

begin                  { initialize timer pointer }
    newtimepointer :=@newtimer ;
end.
```

程序清单8.6：抓小偷游戏的Pascal程序清单

```
{ ********************************************
** THIEF.PAS                                 **
** Turbo Pascal 4.0                          **
******************************************** }

Program Thief ;

Uses Crt, Dos, Cmsnote ;

type screentype=array[0..1999] of byte ;

var screen : screentype absolute $b800 : 0 ;
    cur_screen : word ;
    xthief, ythief, xagent, yagent : byte ;
    beepcount : integer ;
    banditcount : integer ;
    oldtimer : pointer ;
    oldnum : byte ;
    xmin : byte ;
    ymin : byte ;
    amplit : byte ;
    oct : byte ;
    stereo : byte ;
    stop : byte ;
    a,i,j : integer ;
    ch : char ;

Procedure SetScreen ;          { build a square on the screen }
var i ; integer ;
    regs : registers ;

begin
   clrscr ;
   cur_screen :=22 ;
   screen[0] :=218 ;
   i :=2 ;
   Repeat
       screen[i] :=196 ;
       inc(i, 2) ;
   Until (i>=34) ;
   screen[34] :=191 ;
   i :=80 ;
   Repeat
       screen[i] :=179 ;
       screen[i+34] := 179 ;
```

```
        inc(i, 80) ;
    Until (i>=80*17) ;
    screen[80*17] :=192 ;
    i :=80*17+2 ;
    Repeat
        screen[i] :=196 ;
        inc(i, 2) ;
    Until (i>=80*17+34)) ;
    screen[80*17+34] :=217 ;

    regs.ax :=$0100 ;
    regs.cx :=$2000 ;
    Intr(16, regs) ;              { put the cursor far sway }
end ;

Procedure Putdoll(x,y,chr : byte) ;    { put a character }
var total : word ;                     { in the square }
begin
    total :=0 ;
    Inc (total, ((x+1)*2)+((y+1)*80)) ;
    screen[total] :=chr ;
end ;

Procedure intelthief ;          { determine the new }
var xr, yr : byte ;           { coordinates of the thief }

begin
    xr :=rendom(2) ;
    yr :=random(2) ;

    if ((xr=1) AND (xthief<15))
    then
        Inc(xthief)
    else
        if (xthief>0) then Dec(xthief) ;

    if ((yr=1) AND (ythief<15))
    then
        Inc(ythief)
    else
        if (ythief>0) then Dec(ythief) ;
end ;

Procedure Newtimerint ; INTERRUPT ;     { give alternately }
var regs : registers ;                    { left and right a }
                                  { beep using the coordinates }
                                    { of agent and thief }
```

```
begin
   Inc(beepcount) ;

   if (beepcount=9) then
   begin
      if (stereo=0) then
      begin
        if (xagent<=xthief) then
        begin
          xmin :=(xthief-xagent) ;
          amplit :=0 ;
          amplit :=((15-xmin) SHL 4) ;
        end
        else
          amplit :=0 ;
        port[baseport+1] :=0 ;
        port[baseport] :=amplit ;
      end
      else
      begin
        if (xagent>=xthief) then
        begin
          xmin :=(xagentxthief) ;
          amplit :=15-xmin ;
        end
        else
          amplit :=0 ;
        port[baseport+1] :=0 ;
        port[baseport] :=amplit ;
      end ;
      sterep :=stereo XOR 1 ;

      if (yagent<ythief) then
      begin
        ymin :=(7-(ythief-yagent) div 2) ;
        oct :=ymin ;
        port[baseport+1] :=$10 ;
        port[baseport] :=oct ;
      end
        else
        begin
          ymin :=(7-(yagent-ythief) div 2) ;
          oct :=ymin ;
          port[baseport+1] :=$10 ;
          port[baseport] :=oct ;
    end ;
    beepcount :=0 ;
    port[baseport+1] :=$14 ;
```

```
    port[baseport] :=1 ;
  end ;
  if (beepcount=5) then
  begin
    Port[baseport+1] :=$14 ;
    Port[baseport] :=0 ;
  end ;

  if (banditcount=0) then putdoll(xthief, ythief, 32) ;

  Inc(banditcount) ;
  if (banditcount=182) then
  begin
    putdoll(xthief, ythief, 2) ;
    banditcount :=0 ;
  end ;

  intr(oldtimv, Regs) ;
end ;

begin
  beepcount :=0 ;
  banditcount :=0 ;
  xmin :=0 ;
  ymin :=0 ;
  oct :=0 ;
  amplit :=0 ;
  stereo :=0 ;
  stop :=0 ;

  clrscr ;
  Writeln(' The black agent has to catch') ;
  Writeln(' the white thief.') ;
  Writeln(' Use cursor keys.') ;
  Writeln(' High tones mean that the thief is') ;
  Writeln(' very close, low tones mean the contrary.') ;
  Writeln(' The stereo effect determines thether the thief') ;
  Writeln(' is to your letf or to your right.') ;
  Writeln(' Press a key to stary.') ;

  ch :=Readkey ;

  textmode(C40) ;
  port[baseport+1] :=0 ;              { set note }
  port[baseport] :=$ff ;

  port[baseport+1] :=$14 ;
  port[baseport] :=0 ;
```

```
    port[baseport+1] :=$8 ;
    port[baseport] :=64 ;

    port[baseport+1] :=$10 ;
    port[baseport] :=4 ;
    port[baseport+1] :=$15;
    port[baseport] :=0 ;

    port[baseport+1] :=$1c ;
    port[baseport] :=1 ;

    setscreen ;
    randomize ;
    xthief :=random(16) ;
    ythief :=random(16) ;
    xagent :=8 ;
    yagent :=8;

    getintvec(8,oldtimer) ;              { take over vectors }
    setintvec(oldtimv, oldtimer) ;
    setintvec(8, @newtimerint) ;

    while (stop<>1) do
    begin
      putdoll(xagent, yagent, 1) ;
      ch :=readkey ;
      if ch=#0 then ch :=readkey ;
      putdoll(xagent, yagent, 32) ;
      putdoll(xthief, ythief, 32) ;      { delete }
      case ch of                         { cursor keys }
        #77 : if (xagent<15) then Inc(xagent) ; { determine }
        #75 : i (xagent>0) then Dec(xagent) ; { new }
        #72 : if (yagent>0) then Dec(yagent) ; {coordinates }
        #80 : if (yagent<15) then Inc(yagent) ;
        #27 : stop :=1 ;
    end ;

    intelthief ;
    if ((xthief=xagent) AND (ythief=yagent)) then stop :=1 ;
end ;

Port[baseoprt+1] :=$1c ;
Port[baseport] :=0 ;

setintvec(8,oldtimer) ;

TextMode(c80) ;
Writeln('Gotcha !! ') ;
```

```
Writeln ;

initcms ;
settimer(100, newtimpointer) ;
If Keypressed then ch :=readkey ;

while not Keypressed do      { and an effect afterward }
begin
   for i :=255 downto 0 do
     playnote(0,100,i,5,2,0,0,50) ;
   for i :=0 to 255 do
     playnote(0,100,i,5,2,0,0,50) ;
end ;

ch :=readkey ;
resettimer ;
resetcms ;

end .
```

程序清单8.7：支持抓小偷游戏的程序单元

```
/* ******************************************
** Include file CMSNote.H in THIEF.C          **
** Turbo C++ 2.0                              **
****************************************** */

#define baseport 0x220        /* default port */

typedef unsigned char byte ;
unsigned timevoice[12] ;      /* time for each voice */
byte cmsonfreq[12];   /* which freq voice is on? */
byte cmsonnoise[12];  /* and which noise voice ? */
byte counter ;
byte freqtot, noisetot;    /* totals */
void playnote(byte voice, byte ampl, byte tone,
    byte octave, byte sort, byte noiseclock,
    byte envelope, unsigned duration)

{
  unsigned int cmsvoice, cmsport ;
byte i, j, k, l ;

  cmsport=baseport ;
  cmsvoice=voice ;
  if (voice>=6)               /* if the voice > 6, then */
  {                             /* the other CMS chip has to take over */
    voice-=6 ;
```

```
    cmsport+=2 ;
  }
  outp(cmsport+1, voice) ;
  outp(cmsport, ampl) ;              /* set amplitede */
  outp(cmsport+1, 8+voice) ;
  outp(cmsport, tone) ;              /* set pitch */
  outp(cmsport+1, 0x10+(voice/2)) ;
  j=(octave<<((voice&1)*4)) ;
  outp(cmsport, j) ;                 /* set octave */
  outp(cmsport+1, 0*16) ;
  k=(noiseclock < (((voice/3)&1)*4)) ;
  outp(cmsport,k) ;
  outp(cmsport+1, 0x18+(voice/3)) ;
  outp(cmsport, envelope) ;          /* and the envelope */
  if (sort&2)
    cmsonfreq[cmsvoice]=1 ;   /* a frequency sound */
  if (sort&1)
    cmsonnoise[cmsvoice]=1 ;  /* or a noise sound */
  tim,evoice[cmsvoice]=duration ;  /* and the duration of the note
*/
}

void interrupt (*oldtim)(void) ;  /* the old timer */

void interrupt newtim()       /* the new timer */
{
  byte freq, noise ;
  int i,j ;
  unsigned int cmsport ;

  for (i=0 ; i<12 ; i++)             /* go along all voices */
    {
      if (cmsonfreq[i] || cmsonnoise[i] )
                              /* is the voice enabled? */
      {
      if (timevoice[i]>0)  /* decrease the duration of the */
        timevoice[i]>0)  /* decrease the duration of the */
      else
          {                   /* if the duration is zero */
        cmsonnoise[i]=0 ;  /* disable voice */
        cmsonfreq[i]=0 ;
        }
          }
    }
  freqtot=0 ;
  noisetot=0 ;

  cmsport=baseport ;
```

```
  for(i=0 ; i<6 ; i++)          /* calculate which voices */
  {                             /* have to be switched on */
    if (cmsonfreq[i]) freqtot+=(1<i) ;
    if (cmsonnoise[i]) noisetot+=(1<i) ;
  }
  outp(cmsport+1, 0x14) ;  /* set frequency */
  outp(cmsport, freqtot) ;              /* register */

  outp(cmsport+1, 0x15) ;  /* and noise register */
  outp(cmsport, noisetot) ;

  freqtot=0 ;
  noisetot=0 ;

  for (i=6; i<12; i++)  /* same for the voices */
  {                                     /* 6 through 11 */
    if (cmsonfreq[i]) freqtot+=(1<<(i6)) ;
    if (cmsonnoise[i]) noisetot+=(1<<--(i6)) ;
  }
  outp(cmsport+3, 0x14) ;
  outp(cmsport+2, freqtot) ;

  outp(cmsport+3, 0x15) ;
  outp(cmsoprt+2, noisetot) ;

  counter++ ;           /* increment counter */
  oldtim() ;            /* call old timer */
}

void settimer(unsigned Freq, void interrupt (*newIRQ)())
{
  int ICnt ;
  asm cli ;
  ICnt = 1193180/Freq ;             /* determine the number of clock */
  outp(0x43, 0x36) ;                /* ticks of the timer and */
  outp(0x40, ICnt &255) ;           /* pass it on to the */
  outp(0x40, ICnt >8) ;             /* timer chip */
  oldtim=getvect(8) ;
  setvect(8,newIRQ) ;           /* take over timer interrupt */
  asm sti ;
}

void resettimer()
{
  asm cli ;
  outp(0x43, 0x36) ;            /* reset clock ticks at */
  outp(0x40, 0x0) ;             /* default position */
  outp(0x40, 0x0) ;
```

```
  setvect*8, oldtim) ;          /* restore old timer */
  asm sti ;
}

void resetcms()
{
  outp(baseport+1, 0x1c) ;  /* give reset command */
  outp(baseport, 2) ;
  outp(baseport+3, 0x1c) ;          /* for both chips */
  outp(baseport+2, 2) ;
}

void initcms()
{
  int cnt ;
  resetcms() ;
  outp(baseport+1, 0x1c) ;  /* enable all sound */
  outp(baseport, 1) ;
  outp(baseport+3, 0x1c) ;
  outp(baseport+2, 1) ;
  for (cnt=0 ; cnt<12; cnt++)
  {
    timevoice[cnt]=0 ;              /* initialize all */
    cmsonfreq[cnt]=0 ;            /* registers */
    cmsonnoise[cnt]=0 ;
    playnote(cnt, 0,0,0,0,0,0,0) ;
  }

}
```

程序清单8.8：抓小偷的游戏的程序清单

```
/* ******************************************
** THIEF.C                                  **
** Turbo C++ 2.0                            **
****************************************** /

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <dos.h>
#include "cmsnote.h"

typedef unsigned int word ;
byte far *screen ;      /* pointer to the screen   */
byte xthief, ythief, xagent, yagent ;  /* x/y coordinates */
                              /* of thief and agent */
void setscreen()
{
```

```
  int i ;
  clrscr() ;
  screen=MK_FP(0xb800, 22) ;  /* create a pointer to */
                              /* the screen */
  screen[0]=218 ;             /* draw a square */
  for (i=2; i<34; i+=2) screen[i]=196 ;
  screen[34]=191 ;
  fpr (i=80 ; i<80*17 ; i+=80)
  {
    screen[i]=179 ;
    screen[i+34]=179 ;
  }
  screen[80*17]=192 ;
  for (i=80*17)+2 ; i<80*17)+34 ; i+=2) screen[i]=196 ;
  screen[(80*17)+34]=217 ;
  _setcursortype(_NOCURSOR) ;    /* remove cursor */
}

void putdoll( byte x, byte y, byte chr)
{
  word total=0 ;                 /* calculate location */
  total+=((++x)*2)+((++y)*80) ;   /* in screen */
  screen[total]=chr ;            /* place the character */
                                 /* on the screen */
}
void intelthief()
{
  byte xr, yr ;              /* determine new x and y */
  xr=random(2) ;             /* coordinates of the thief */
  yr=random(2) ;
  if ((xr==1) && (xthief<15)) xthief++ ;
  else if (xthief>0) xthief-- ;
  if ((yr==1)&&(ythief<15)) ythief++ ;
  else if (ythief>0) ythief-- ;

}

void interrupt (*oldtim)(void) ;     /* old timer int. */

  int beepcount=0 ;
  int banditcount=0 ;
  byte xmin=0, ymin=0 ;
  byte amplit=0, oct=0 ;
  byte stereo=0 ;

void interrupt newtimerint(void)  /* new timer */
{                                 /* takes care of the increment */
  beepcount++ ;          /* of the counters and sends a */
```

```
if (beepcount==9) ;          /* beep based on the difference */
{                            /* between the coordinates */
  if (stereo==0)
  {
    if (xagent<=xthief)
    {
      xmin=(xthief-xagent) ;
      amplot=0 ;
      amplit=((15-xmin)<<4) ;
    }
    else
      amplit=0 ;
    outp(baseport+1, 0) ;
    outp(baseport, amplit) ;
  }
  else
  {
    if (xagent>=xthief)
    {
      xmin=(xagent-xthief) ;
      amplit=15x-min ;
    }
    else
      amplit=0 ;
    outp(baseport+1, 0) ;
    outp(baseport, amplit) ;
  }
  sterep=stereo^1 ;  /* next time other side */
  if (yagent<ythief)
  {
    ymin=(7(ythiefyagent)/2) ;
    oct=ymin ;
    outp(baseport+1, 0x10) ;
    outp(baseport, oct) ;
  }
  else
  {
    ymin=(7-yagent-ythief)/2) ;
    oct=ymin ;
    outp(baseport+1, 0x10) ;
    outp(baseport, oct) ;
  }
beepcount=0 ;
outp(baseport+1, 0x14) ;
outp(baseport, 1) ;
}
if (beenpcount==5)
{
```

```
        outp(baseport+1, 0x14) ;
        outp(baseport, 0) ;
    }
    if (banditcout==0) putdoll (xthief, ythief, 32) ;

    banditcout++ ;
    if(banditcount==182)
    {
      putdoll(xthief, ythief, 2) ;
      banditcount=0 ;
    }

    oldtim() ;
}

void main()
{
    int a, i, j ;
    int stop=0 ;
    clrscr() ;
    printf("\n\n\n The black agent has to catch") ;
    printf(" the white thief. \n" ) ;
    printf(" Use cursor keys. \n") ;
    printf(" High tones mean that the thief is") ;
    prinrf(" very near, low tones mean the contrary.\n") ;
    printf(" The stereo effect determines whether the thief") ;
    printf(" is to your left or to your right.\n") ;
    printf("\n Press a key to stary.\n") ;
    getch() ;

    textmode(C40) ;
    outp(baseport+1, 0) ;        /* set registers */
    outp(baseport, 0xff) ;
    outp(baseport+1, 0x14) ;
    outp(baseport, 0) ;
    outp(baseport+1, 0x8) ;
    outp(baseport, 64) ;
    outp(baseport+1, 0x10) ;
    outp(baseport, 4) ;
    outp(baseport+1, 0x15) ;
    outp(baseport, 0) ;
    outp(baseport+1, 0x1c) ;
    outp(baseport, 1) ;

    setscreen() ;
    randomize() ;
    xthief=random(16) ;
    ythief=random(16) ;
```

```
  xagent=8 ;
  yagent=8 ;
  oldtim=getvect(8) ;
  setvect(8, newtimerint) ;
  while (!(stop))
  {
  putdoll(xagent, yagent, 1) ;
  a=getch() ;
if (!a) a=getch() ;
putdoll(xagent, yagent, 32) ;
putdoll(xthief, ythief, 32) ;
switch (a)
{
  case 77:                          /* cursor keys */
    if (xagent<15) xagent++ ;       /* determine new */
    break ;
  case 75:
    if (xagent>0) xagent-- ;
    break ;
  case 72:
    if (yagent>0) yagent-- ;
    break ;
  case 80:
    if (yagent<15) yagent++ ;
    break ;
  case 27:
    stop=1 ;
    break ;
  default:
    break;
 }
 intelthief() ;
 if ((xthief==xagent) && (ythief==yagent)) stop=1 ;
}
outp(baseport+1, 0x1c) ;
outp(baseport, 0) ;
setvect(8, oldtim) ;
clrscr() ;
_setcursortype(_NORMALCURSOR) ;
textmode(C80) ;
printf("\n Gotcha !! \n\n") ;

initcms() ;
settimer(100, newtim) ;
while (!kbhit())                /* an effect afterward */
{
    for (i=255; i>0 ; i--)
    playnote(0,100,i,5,2,0,0,50) ;
```

```
    for (i=o ; i<255 ; i++)
    playnote(0,100,i,5,2,0,0,50) ;
  }
  getch() ;
  resettimer() ;
  resetcms() ;

}
```

第九章　编程数字声音处理器

本章内容为：

样本结构
记录和播放样本
CT-VOICE驱动器的功能
CT-TIMER，另一个CT-VOICE的驱动程序
创建声音和声音效果;包括回声、声音的淡入和淡出

第二章介绍了Sound Blaster的数字声音处理器（DSP）和怎样记录声音、播放声音以及把它存入存储器等内容。本章将进行更详细的讨论，叙述怎样构成一些样本，怎样为DSP编程。

注解　如果读者对什么是采样和什么是采样率不熟悉的话，可再参看第三章“处理数字式声音的数字声音处理器”的内容。

样本结构

一个样本包括几个字节，在八位样本中的各字节其范围为0到255。

- 128表示完全无声
- 0和255分别表示送到扬声器的负电压和正电压的最大值。

作为立体声样本，对左声道和右声道，数据是交替地给出的。在这种安排下，偶数地址的数据被指定送给左声道，而奇数地址数据则送到右声道。

Sound Blaster让你压缩样本中的数据是为了保存存贮器。例如，包含8位数据的样本可以转换成4位、2.6位或者2位的样本（为得到2.6位样本，两个值被转换成三位，而一个值转换成二位）。通过数据转换这种方法，可以使样本中的数据成2倍、3倍或4倍的减少。当再现时，Sound Blaster把数据转回成8位的样本。

压缩样本数据虽然节省了存贮器但也降低了采样的质量。在压缩之前，要考虑到样本中数据的丢失。相比而言，有几种数据类型是比较好的压缩选择。举例来说，对声音，不需要高精度的采样值就可懂得。VEDIT和VOXKIT程序可以将8位的样本转换成4位，2.6位或2位的样本。

记录和播放样本

可以用直接的和DMA两种方法来记录和播放样本。

- 使用DMA（直接存储器存取）只占用很少量的处理器时间，通过用DMA方式，可确保其它程序以全速运转。

- 直接地记录和播放样本给了一种机会，以实现各种各样的编辑功能，诸如使用回声、滤波、改变音调和音量等。不过，直接的方法要消耗较多的处理器时间。当使用直接方法时，别的程序将运行得十分缓慢。

通过DMA播放时，可使用Sound Blaster提供的CT-VOICE.DRV驱动程序，该程序是放在VOXKIT目录或者DRV目录里，使你能以VOC格式记录和播放样本。在描述这种驱动程序以前，我们先看一下VOC格式。其格式的版本号是1.10，它可同时用于Sound Blaster和Sound Blaster Pro。

VOC格式怎样管理样本

一个VOC文件由文件头和数据块两部分组成，数据块包括了实际的VOC格式。

文件头

文件头包含了一般用信息，CT-VOICE程序不用这些信息。当你只打算使用数据时，必须跳过文件头。文件头的结构如下：

字　节	设　置
00H-13H	文件类型描述，这19个字节包含下面的正文： Creative Voice File，后跟以一EOF字节1AH。
14H-15H	文件数据块的开始。因为在将来的版本中，文件头的长度可能改变，所以这个字（两个字节）被规定。通常这两个字节包含的值是001AH。
16H-17H	文件的版本号。第一个字节放的是点号后的值，第二个字节装的是点号前的值。例如，对版本1.10来说，第一个字节中的值是0AH（=10），而第二个字节中的值为1。
18H-19H	一个识别码。由这个代码可以检验其文件是否是真正的VOC文件。代码值是16H和17H单元中所存文件版本号的补码再加上1234H的结果。现举例说明，若版本号为1.10，则010AH的补码是FEF5H（010AHXORFFFFH）；FEF5H+1234H=1129H。

数据块

数据块由一个或几个子块组成。每个子块可以包含一个样本或者别的信息。这样，在一个文件中，可以存放几个由不同采样率构成的样本。

所有的子模块均由一个小的头和相应的数据部分组成。这个头包含了有关块的类型、块的长度、以及一些附加信息。对每一个块，最前四个字节的是相同的：

- 第一个字节指定块的类型。
- 后面的三个字节指示块的长度（这四个字节之后数据的尺寸大小）。

用户可以不必知道有关块的类型的任何情况就能够对块进行操作。在最前四个字节之后，可能存在一些各不相同的块的类型，现分述如下。

块类型0，数据块的结束

类型0是唯一没有指明其长度的数据的一种块类型。

这种块标志子块的结束。如果CT-VOICE驱动程序碰到这种类型的块，它就停止处理其它子块。

提示　检查这种块类型是否位于要播放部分的末尾。如果不是，添加这个块类型，否则CT-VOICE驱动程序将解释上个块之后的字节为块类型，这样就可能导致不可预见和不一定愉快的情形，从噪声和爆裂声到程序或者计算机挂起。如果这种块在数据块的末尾不存在，则驱动程序继续读出存贮器中的下个字节作为块的类型。

块类型1，新的样本数据

用这种块，可以播放一新的样本。除了样本数据以外，这种块还包含了速度、所用的压缩（或打包）方法等有关信息。

字段　块的长度包括SR字段和PACK字段，因此其值等于DATA+2。SR代表采样率，此值由下式决定：

SR=256-1，000，000/采样率

若采样率为10，000HZ，则SR=156，因为1000，000/10，000=100，256-100=156。

对于Sound Blaster，最高采样率是23，000HZ，而作为Sound Blaster Pro最高采样率为44，100HZ。

PACK包含打包的方法，以下是一些可能的值：

0：　普通的8位未打包的样本

1：　4位打包

2：　2，6位打包

2：　2位打包

DATA包含样本的数据，数据尺寸大小等于块的长度-2。

块类型2，样本数据

如果想用同样的速率和相同的打包方法播放几个样本时，就可以采用这种块类型。这个块的样本数据以上一个样本数据块的设置播放（见块类型1的描述）。用这种方式，只需改变一个块的采样率，它将会影响其他的值。

字段　现在，块的长度等于数据的尺寸大小，DATA包括样本数据。

块类型3，无声块

这个块将做到使Sound Blaster在一段时间中停止发声，它插入了一段持续的无声期。用这个无声块替代样本数据里的所有暂停，数据就占据较少的存贮器。VEDIT程序提供了这种操作的机会。

字段　这种块的长度值为3。

PERIOD包含一个16位的值，这个值加1指明不发声的持续期，它是采样率的单位的数量。

SR是采样率（请见块类型1有关采样率的解释）。

举例说明之：假定采样率是4000HZ，即值一秒钟被处理4000次。如果不发声的持续期的值是8000，则表明了静止不发声时间是精确的两秒（8000/4000）。换句话说，按照推算是一个值被操作了8000次。不过PERIOD的值是8000-1，或者7999，所以采样率越高，静止不发声时间愈准确，但最大静止时间愈小。

块类型4，标志

用这个块可以给状态字赋一个新的值（见下节）。利用这种方法，能让你的程序一直等到一个特定的样本被放送，而你只需检查其状态字。

字段 块的长度值为2。

MARKER包含了1到65534（FFFEH）之间的值。数值0和65535被分配给CT-VOICE驱动程序自身了，所以是不可使用的。

块类型5，ASCII文本

用这个块可以在VOC块中加入一个注释，文本以0字节结束。

字段 块的长度包括了文本计入0的长度。

TEXT包括文本，作为本例：Drum.0。

块类型6，重复开始

块类型6代表重复开始。CT-VOICE驱动程序将在此块与“重复结束”块类型的全部块之间重复指定的次数。

字段 块长度值为2。

COUNTER指明重复的次数，下面的块将被重复COUNTER+1次。如果COUNTER中的值是65535（FFFFH），那么这个块将无限制地一直重复下去。在下一段将叙述终止这种重复的功能。

块类型7，重复的终止

块类型7确定了重复块的终止。所有在这个块与前述的块类型6之间的子块将被重复，其重复次数表示在前述块类型中。进一步的解释请参考块类型6的说明。

字段 块长具有零值。

警告 不支持和不允许嵌套重复块。

块类型8，附加信息

块类型8仅只适宜于新的CT-VOICE驱动程序，这种驱动程序由Sound Blaster Pro提供。当这个块被处理时，下一个新的样本块（块类型1）用这个块的设置去取代它自己的设置。这种工作仅只一次，如果需要，就必须对每个新的样本块附加子块。这种类型8的块允许你传递有关Sound Blaster Pro的扩展DSP附加信息。

字段 块长的值为4。SR再次包括了采样率，这次它是一个16位的值，更精确。SR的值

可由下面公式给出：

SR=65536-256000000/采样率

最大采样率仍是44100HZ。如果涉及的是一个立体声样本，则采样率还应该加倍，这时公式变成

SR=65536-256000000/采样率*2

举例来说，一个采样率是20000HZ的单声道样本的SR值等于52736（65536-256000000/20000），而同样20000HZ采样率的立体声样本的SR将等于59136（65536-256000000/40000）。PACK包含有数据块类型1所提及的压缩方法，通常PACK的值为0。MODE则指出样　本是该用单声道或立体声播放，若MODE中字节值为0，表示单声道播放;字节值为1则为立体声。样本用立体声播放时，PACK的值必须为0。

CT-VOICE驱动程序

CT-VOICE驱动程序包含了初始化Sound Blaster卡、以VOC格式记录和播放样本等功能。为了这些功能，必须先将CT-VOICE.DRV程序装入偏移值为0的一个段中，然后用FAR调用偏移值为0的这个段，使其驱动程序被调用，所用的全部寄存器是靠驱动程序自身存放的，所以它们不改变。程序清单9.2和9.4（在功能描述后面可找到）包含了和Pascal以及C语言的接口，程序9.1和9.3包括了一个短的汇编程序，是用于Pascal和C的。

Sound Blaster Pro提供了一个比较复杂的CT-VOICE驱动程序，它包含了14种附加功能以支持Sound Blaster Pro所有可能性的实现。标准的CT-VOICE是这里所描述的2.10版。然而，扩展的CT-VOICE与标准CT-VOICE驱动程序是完全兼容的，而且也能够和标准的Sound Blaster配套使用。

这一节将讨论这些功能。首先分别列出了作为Pascal，C和汇编调用这些功能的规范，但是，有一些功能只是用汇编语言列出的，因为它们用到Pascal和C时不相同。当一些变量在Pascal和C中被作为输入和输出使用时，这些变量名在汇编中被赋给一寄存器中。这样就使得解释这些不同寄存器的功能较为容易。不过这些寄存器必须装载标准的16位的值。当其在汇编中一个结果返回时，把它放在变量SBIOResult中，这类变量是字或者是无符号整数。

注解　在能使用较高的一些功能之前，首先应该用功能3（初始化）调用驱动程序，但是可以用功能0到功能2，以及功能10（扩展版本）。

功能0，获取版本号

Function CTVerdion Word
int CTVersion ()

输入：	BX	=0
输出：	AH	=点号前部分的数字
	AL	=点号后部分的数字

功能0为送回驱动程序的版本号，使用这个功能，一个程序能够检查它是否支持这个版本

功能1，设置I/O端口

Procedure UsePort（Port：Word）
void UsePort （int Port）

输入： BX =1
AX =端口号
输出： 无

这个功能是为Sound Blaster设置入/出端口，可选择的端口地址有210H、220H、230H、240H、250H或260H。但用户应考虑这个事实，即Sound Blaster只支持端口220H和240H。在Sound Blaster卡用功能3初始化之前，应该先调用功能1。生产厂商的缺省端口值是220H。

功能2，设置中断请求（IRQ）号

procedure UseIRQ（IRQ： Work）
void UseIRQ（int IRQ）

输入： BX =2
AR =IRQ
输出： 无

这一功能是设置个正确的中断请求，在DMA传送结束后调用这个中断。可选择的中断请求号有2，3，5或7。对于Sound Blaster Pro，可选择的请求号是2，5，7或10。就像功能1一样，调用功能2应该在初始化之前。

注解 关于各种IRQ中断的信息，请看第一章的有关介绍。

功能3，初始化驱动器

Procedure Initialize Driver
int InitializeDriver（）

输入： BX =3
输出： AX =结果

调用功能3实现对驱动器和DSP的初始化（预置），只有功能0，1，2和功能10可以在用这个功能之前先被调用，以改变缺省的设置。别的一些功能不能够在功能3成功调用之先而被使用。功能3返回一代码值以指示相关的情况，一些可能的代码值是：

代码值	含义
0	一切正常，驱动器已准备好。
1	所插入的不是Sound Blaster （Pro）卡或是不认识的卡。
2	I/O读写失败（错误的端口）。
3	DMA中断失败（中断请求号或者DMA通道设置不对）。

变量SBIOResult中包含了用Pascal或C时的返回代码。

功能4，接通或断开与扬声器的连接

Procedure Speaker（W：Word）
void Speaker（int W）

输入： BX =4
AX =W
输出： 无

下面是适于W的值：

- W=0 信号不传给扬声器
- W≠0 信号传至扬声器

初始化功能3接通此连接。

提示 假如程序回到DOS或者采样被记录，程序人员必须断开此连接，被记录的声音可以通过扬声器听到并干扰录音。这对Sound Blaster Pro不适用，因为这种卡的采样已经改进了。

功能5，设置状态字偏量

输入：BX = 5
ES：DI = 段：字偏量
输出：无

状态字用于指示驱动器是否正忙于播放或者记录样本，如果是，状态字有值65535（=FFFFH）或者用标志块的值来指示（见前节）。一旦这个过程结束，状态字再恢复为静止值零。

这一功能只适于汇编设计者用;在Pascal和C语言中，变量StatusWord被定义，变量的类型是字或是无符号整数。

功能6，起动播放VOC块

Procedure PlayBlock（Start：VocTp）

void PlayBlock（VocTp Start）

输入： BX = 6
ES：DI = VOC子块的开始
输出： 无

功能6将使用DMA播放一个满足VOC格式的块。这时只花费很少的处理器时间，而且允许别的程序继续正常执行。其状态字保持为65535或FFFFH，直至这个值被一个标志块改变或者到达了块的末尾（块类型0），这时使状态字为0。对Pascal和C，一个记录或一个结构VocTp，分别被定义以包含附加信息。

要在同一时间播放另外的样本块是不可能的，为了想做到这一点，必须先用功能8停止输出，通过用状态字和标志块，确定播放哪个样本块，以及再现是否已结束，有了这些信息，才能执行正确的动作。

功能7，记录一个样本

Procedure RecordSample（length：longInt; Rate：Word，Var V： VocTp）
void RecordSimple（long Length，Unsigned Rate，VocTp &V）

输入： BX =7
AX =Rate（速率）
DX：CS =Length（长度）
ES：DI = Start（起始地址）
输出： 无

使用功能7记录声音。样本由起始地址开始，具有Length长度和Rate HZ的采样率。对于标准的Sound Blaster卡，速率值在4000HZ到12，000HZ之间，而Sound Blaster Pro的速率值则在4000HZ与44100HZ之间。如果记录的是立体声样本，采样率应选在4000HZ和22050HZ之间;不过Rate的实际值是两倍的采样率，例如采样率是13000HZ，那么Rate是26000而不是13000。驱动程序本身把Rate转换成一个8位的值。

在记录期间，状态字的值为65535;然后再为0。可以用功能8来中断录音。

注解　为要记录立体声采样可参看功能16。

这样本有一个满足VOC格式计算的文件头。对立体声样本，附加信息块（类型8）是放在文件头的前面，Length是块的总长，记录声音的实际长度大小是8个字节或16个字节。

功能8，停止记录或播放一个样本块

Procedure StopVProcess
Void StopVProcess（）

输入： BX　　=8
输出： 无

这个功能是停止正在进行的记录或播放。状态字又为0。记录时，所建立的VOC文件头以正确数据进行修改。

功能9，终止驱动器

Procedure DriverOff
Void DriverOff

输入： BX　　=9
输出： 无

当驱动器在忙于放音或录音时，本功能停止此过程，并且断开与扬声器的连接。在退出程序和返回到DOS以前功能9总是必须被调用。

功能10，暂停播放

Procedure Pause
Void Pause（）

输入： BX　　=10
输出： AX　　=0，暂停播放
　　=1，无样本在播放

驱动程序暂停样本的再现。如果这是不可能的话，返回输出数值1。对于Pascal和C，其结果再次被放在变量SBIOResult中。使用功能11可再继续播放。

功能11，继续播放

Procedure ContinuePlaying
Void ContinuePlaying（）

输入： BX　　=11
输出： AX　　=0，播放继续
　　=1，播放未曾暂停

这一功能是驱动程序继续播放一个样本块。假若这个驱动程序未曾被暂停过，本功能使输出值为1；否则返回到0。

功能12，停止重复播放和重复后继续

Procedure StopRepetition（I：word）
Void StopRepetition（int I）

输入： BX =11
AX =I
输出： AX =0，一切正常
=1，没有样本曾被播放

功能12和VOC格式有关，它使其一个或几个样本能重复地放若干遍（如果你需要的话，可无限次）。在重复之后，这个功能跳到下个子块，该子块允许你重复这样本 的特别部分，相同效果产生（举例来说，如像脚步声、直升机声、汽车声等）多久就重复多久。一旦你想改变它，可跳到下一个数据块。变量I规定了方法：如果I值为0，这些块被做最后一次重复;如果I值为1，则重复被立即停止而直接在重复末尾后的子块被处理。

功能13，设置用户程序

Procedure UserRoutine（P：Procedure）
Porcedure NoUserRoutine
Void UserRoutine（void *P（））
Void NoUserRoutine（）

输入： BX =13
DX：AX =用户子例程偏量
输出： 无

使用功能13，程序人员可以在一个数据块中每个子块开始处调用他们自己的子例程。不过，对汇编程序人员所要作的工作与对Pascal和C的编程人员所要做的工作是不相同的。

对于汇编程序人员....对汇编程序人员，用户子例程应当满足三个条件：

- 由于FAR调用不可缺少，子例程必须用RETF作结束。
- 除了进位标志位外，必须保存所有用到的寄存器。
- 如果必须跳过下个子块，进位标志应置成1；为了处理下个块，应将进位标志位置成0。

在每次调用，ES：BX指向被处理的数据块，ES：BX中的字节表示块的类型。

不言而喻，在一个结束块进位标志总应该抹去，否则，这个块被跳过（虽然这个块没有长度），以后各字节被认为是一新的子块。

为了断开用户子例程序，AX和DX必须为0。

对于Pascal和C的编程人员.... 对于Pascal和C的程序设计者，已经规定了许多变量。

Var

```
  BlockStart :  Pointer;
  Blocklength :  LongInt;
  BlockType :  Byte;
  Continue :  Boolean;
void          *Blockstart;
Long          BlockLength;
Unsigned char BlcokType;
Char          Continue;
```

当该程序被调用时，这些变量包含即将到来的块的正确值。这个变量Continue允许你指明下个块是一定要播放（True，1）或者一定不被播放（False，0）。剩下的变量可以改变为自己所希望的，它们不影响程序的进程。

不言而喻，在一个结束块之后，子例程序总一定是先处理紧接的块，此外相邻的一些字节将被作为样本块来考虑。

为了断开用户子程序，可调用NoUserRoutine。

注解　只有Sound Blaster Pro的新的CT-VOICE驱动程序才支持功能14到27。

功能14，起动播送存放在扩充存贮器块中的VOC块

```
Procedure PlayEMBBlock （Handle：Word;Start：LongInt）
Void PlayEMBBLock（unsigned Handle，long start）
```

输入：	BX	=14
	DX	=Handle
	DI：SI	=在扩展存贮器中的起点
输出：	AX	=0，一切正常
		=1，已出现一个错误

这一功能是播放若干个子块，它们是位于一个扩展存贮器块（EMB）里面。你必须自己安装HI-MEM.SYS驱动程序并用此驱动程序分配一EMB，并装入正确的VOC数据。变量Handle是属于所分配块的柄。

现在你可以播放大的VOC块，不需占据DOS所用的存贮器部分。当然，要你的计算机有真正的扩展存贮器，这个功能才能工作。在仿真的EMB（例如在硬盘上）条件下，该功能不工作，因为重现要通过DMA进行。

功能15，使用EMB记录一个采样

```
Function RecordEMBSample（Handle，SR：Word;Start：LongInt）：
                                    BOOLean ;
int RecordEMBSample（unsigned Handle，unsigned SR，long Start）；
```

输入：	BX	=15
	AX	=SR，采样率
	DX	=Handle
	DI：SI	=扩展存贮器中的起点
输出：	AX	=0，一切正常
		=1，已出现一个错误

功能15是记录一个采样在EMB中。变量Handle指出对应的EMB。EMB的大小也就是要记录的采样的尺寸。与常规的录音功能7同样的规则适合于这个功能。

功能16，设置记录模式

Procedure StereoRecording （V：Word）
Void StereoRecording（int V）

输入：	BX	=16
	AX	=V
输出：	AX	=旧的值

用功能16以决定下一个记录必须是单声道的还是立体声的样本。如果V包含值0，记录的是单声道样本，而且不建立附加信息块。如果V值为1，则记录的是立体声样本，并为新的样本块创建一个附加信息块。

功能17，确定输入

Procedure Input（V：Word）
Void Input（int V）

输入：	BX	=17
	AX	=V
输出：	AX	=旧的设置

V值确定了记录采样的输入选择，可将麦克风、线入或者CD输入分别使用数值0，1和3来代表。如果用功能7或者功能15记录一个样本，那么所选的这个输入就作输入使用。记录时每次不能多于一个输入，虽然几个输入可以发送出，而且可通过输出听见。

然而，你可以把输出连到麦克风的输入或者线路的输入，这样连接了，你就可以录音了，举例来说，可把CD输入和线路输入结合在一个样本中。

功能18，设置记录滤波

Procedure RecordingFilter（V：Word）
Void RecordingFilter（int V）

输入：BX =18
AX =V
输出：AX =旧的值

如果V包含数值0，表示使用了低通滤波；而V值是1，则用的是高通滤波。

功能19，设置DMA通道

Procedure UseChannel（C：Work）
void UseChannel（int C）

输入：BX =19
AX =C
输出：AX =0，一切正常
=FFFFH，无适当的值

变量C用于选择DMA通道，该通道是在Sound Blaster Pro上为DSP设置的。变量C的值可以是0，1和3。就像功能1和2那样，要调用这个功能，必须是在驱动器使用功能3进行初始化以前。

功能20，确定Sound Blaster卡的型式

Var CardType：Word
int CardType

输入：BX =20
输出：AX =1，标准的Sound Blaster
=2，Sound Blaster Pro
=3，Sound Blaster 2.0

这一功能送回关于插在计算机中的卡的型式信息。

注释　功能21到25涉及到混合器芯片，这些设置被写入混合器芯片，并通过CT-VOICE驱动程序保存在内部变量中。第十一章将详细讨论这芯片的功能。

功能21，设置音量

Procedure Volume（S，C，V：Word）
Void Volume（int S，int C，int V）

输入：BX =21
AX =S

	CX	=V
	DX	=C
输出：	AX	<>FFFFH，AX包含老的值
		=FFFFH，已发生一个错误

使用功能21可以调整混合器芯片各部分之一的音量。变量S决定涉及的是哪个部分。作为S可能的选择是：

数值	意　义
0	输出信号的音量
1	VOICE部分的音量
2	MIC输入的音量
3	CD-ROM的音量
4	立体声LINE输入的音量

在前面已介绍过，选择0，1，3，4是立体声，由C值可以决定是左声道还是右声道需要改变。若C=1，表示只需要修改左声道的音量;为了修改右声道的音量，C必须要为2。但是这种做法不适用于CD-ROM选项。

想要通过左声道来设置左、右侧音量，这时C必须总含有1。

变量C不影响MIC的音量设置，因为MIC用的是单声道，C值一定有值1。对于选项0，1，3和4音量V的最大值是15，对于选项2，其最大音量值是7。如果V值大于此值，驱动程序把它定在最大值。送回来的值是音量的旧值。假如是为两个声道设定音量，那么返回的值是右声道音量的旧值。

功能22，设置滤波器

Procedure Filter（S，V：Word）
Void Filter（int S，int V）

输入：	BX	=22
	AX	=S
	CX	=V
输出：	AX	=（见前述功能）

由变量S决定哪个滤波被接通或断开。如果S有值0，表示输出滤波器被编辑;若S为1，表示编辑输入滤波器。

如果变量V为0，所选用的滤波器被接通；V值为1，表示断开所选用的滤波器。

返回值是滤波器的旧的设值。

功能23，复位混合器芯片

Procedure ResetMixer
Void ResetMixer ()

输入： BX =23
输出： 无

功能23将所有的音量和滤波器设置成它们的缺省值，所有以前的设置都失败。第十一章列表并解释了这些缺省值。只有扩充了驱动程序的老版本1.21才支持功能23。

功能24，规定整个音量的设置

Procedure ReadAllVolumes
Void ReadAllVolumes ()

输入： BX = 24
输出： 无

当直接对混合器芯片编程时（第十一章讨论的主题），功能24使CT-VOICE包括新的内容。混合器芯片的所有设置被读出且放在驱动程序的内部变量中。那些老的音量设置被重写了。然后，那些内部变量又等于混合器芯片中的当前设值。

功能25，询问音量设置

Function ReturnVolume (S, C: Word)
int ReturnVolume (int S, int C)

输入： BX = 25
AX = S
DX = C
输出： AX = 询问值
= FFFFH，已出现一个错误

在功能25中，变量S和C具有功能21中所完成的相同功能。变量S用作选择芯片的哪部分，用变量C决定音量值的通道，C还可能包含值0。返回值由CT-VOICE驱动程序所用的内部变量中发出。此值不一定是混合器芯片的当前的设置，这个设置可以用别的程序重新编程。假如你对这种用法不是很有把握的话，我们建议你采用功能24。

功能26，确定采样率

Procedure AskRate (Var Max, Min: Word; M, R: Word)

int AskRate（int *Max，int *Min，int M，int R）

输入：	BX	= 26
	AX	= R
	DX	= M
输出：	AX	= 最小采样率（Min）
	DX	= 最大采样率（Max）

使用功能26，能够回送指定模式下的最小和最大采样率。变量R指出是与记录（0）有关还是与播放（1）有关。如果使用的是Sound Blaster Pro声音卡，那么变量M指示采样率是适合于单声道（值为0）还是立体声。对于别的声音卡，M总是必须为0。扩充的CT-VOICE驱动程序1.21版不支持功能26，已被用功能27代替。

功能27，询问滤波器设置

Function ReturnFilter（S：Word）：Word
int ReturnFilter（int S）

输入：	BX	= 27
	AX	= S
输出：	AX	= 旧的设置（0或1）
		= FFFFH，已出现一个错误

功能27中，变量S与功能22中的可能性是相同的。按照这种方式工作，除了这次要回送滤波器设置而外，其功能与前面所述的功能22相同。返回值来自存贮器，并且不一定和混合器芯片中的现行设定值相同。

用于Pascal和C的一些接口程序清单

下面给出了Pascal和C接口的列表，使用了少量的机器码程序。清单是用编号和标题列出的：

编号	标　题
9.1	Turbo Pascal的汇编接口
9.2	支持CT-VOICE/CT-TIMER驱动程序的Pascal程序单元
9.3	Turbo C的汇编接口
9.4	CT-INTER.OBJ与CTVOICE.C接口的头文件
9.5	支持CT-VOICE/CT-TIMER驱动程序的C函数库的头文件
9.6	支持CT-VOICE/CT-TIMER驱动程序的C函数库

程序清单9.1　Turbo Pascal的汇编接口

```
                . MODEL TPASCAL
                . CODE
; File name = CT-INTER.ASM
; Assembler file taking care of the interface
; between the CTVOICE driver and Turbo Pascal.
;
; Run TASM CTINTER to create CTINTER.OBJ.
; This object file is used in CTVOICE.PAS
;
; Defined procedures
;  Procedure CallCTVoice(Var R : Registers) ;
;  Procedure CTVoiceAddress(P : Pointer) ;
;  Procedure Variables(StPt, BsPt, BlPt, BtPt,
;                      Copt : Pointer) ;
;  Procedure UserRoutine(P : Pointer) ;
;   P points at an interrupt routine in Pascal.

PUBLIC CTVoiceAddress, CallCTVoice
PUBLIC Variables, UserRoutine

; Define a structure for the type of Registers
; that can serve for the offset calculation.

Registers STRUC
rAX    dw     0
rBX    dw     0
rCX    dw     0
rDX    dw     0
rBP    dw     0
rSI    dw     0
rDI    dw     0
rDS    dw     0
rES    dw     0
rFlags dw     0
ENDS
; Define a frequently used routine :
; copying a DWORD

@CopyDword MACRO Dest, Source
        mov ax, WORD PTR Source+0
        mov WORD PTR Dest+0, ax
        mov ax, WORD PTR Source+2
        mov WORD PTR Dest+2, ax
ENDM

; The global variables
```

```
CTVoicePt          DD 0        ; CT-VOICE address
UserRoutAddress    DD ?        ; user routine
BlockStart         DD ?        ; Pointer address
BlockLength        DD ?        ; LongInt address
BlockType          DD ?        ; Byte address
Continue           DD ?        ; Boolean address

; Procedure CTVoieAddress(P : Pointer) ;
; Set the start address of the CT-VOICE driver.

CTVoiceAddress PROC FAR P : DWORD
               @CopyDword cs : CTVoicePt, P
               ret
ENDP

; Procedure CallCTVoice(Var R : Registers) ;
; Calls the CTVOICE driver.

CallCTVoice PROC FAR R : DWORD ;
        push ds              ; has to be saved
        lds si, [R]          ; DS : SI points at R
        mov ax, [si+rSI]     ; save used SI
        push ax
        mov ax, [si+rAX]     ; set all registers
        mov bx, [si+rBX]
        mov cx, [si+rCX]
        mov dx, [si+rDX]
        mov di, [si+rDI]
        mov es, [si+ES]
        mov ds, [si+rDS]
        pop si               ; set SI
        call [DWORD PTR cs : CtVoicePt]
        push ds                   ; save DS
        push si              ; save SI
        lds si, [R]          ; DS : SI points at R
        mov [si+rAX], ax     ; save all regs.
        mov [si+rBX], bx
        mov [si+rCX], cx
        mov [si+rDX], dx
        mov [si+rDI], di
        mov [si+rES], es
        pop ax               ; AX=SI after call
        mov [si+rSI], ax
        pop ax               ; AX=DS after call
        mov [si+rDS], ax
        pop ds               ; restore used DS
        ret
        ENDP
```

```
; Procedure Variables(StPt, BsPt, BlPt, BtPt,
;                     CoPt : Pointer) ;
; Set pointers to StatusWord, BlockStart,
; BlockLength, BlockType & Continue.

Variables PROC FAR
Stp : DWORD, Bsp : DWORD, BlP : DWORD, BtP : DWORD, CoP : DWORD
            @CopyDword cs : BlockStart, BsP
            @CopyDword cs : BlockLength, BlP
            @CopyDword cs : BlockType, BtP
            @CopyDword cs : Continue, CoP
            mov bx, 5                   ; set pointer to
            les di, StP                 ; Status Word
            call [DWORD PTR cs : CTVoicePt]
            ret
ENDP

; Procedure UserRoutine(P : Pointer) ;
; Set user interrupt.

UserRoutine PROC FAR P : DWORD
            mov ax, [WORD PTR P+0]
            mov dx, [WORD PTR P+2]
            mov dx, [WORD PTR P+2]
            mov [WORD PTR cs : UserRoutAddress+2], dx
            or ax, dx                   ; P=0 : 0 ?
            jnz NewUserRoutine
            mov bx, 13                  ; yes, turn off routine
            call [DWORD PTR cs : CTVoicePt]
            ret
NewUserRoutine :
            mov ax, offset UserInterFace
            mov dx, cs
            mov bx, 13                  ; set interface
            call [dword PTR cs : CTVoicePt]
            ret
ENDP

; This routine takes care of the interfave for the
; interrupt routine in Pascal. The routine
; sets the required variables and calls the Pascal
; routine., After the call, it checks thether
; the next block has to be played.

UserInterFace PROC FAR
            push ax                     ; save used
            push si                     ; registers
            push ds
            lds si, [cs : BlockStart]   ; set BlockStart
```

```
            mov [si], bx                    ; starting from
            mov [si+2], es                  ; block to be played
            lds si, [cs : BlockLength]      ; set Blocklength
            mov ax, [es : bx+1]             ; with a 24-bit
            mov [si], ax                    ; length
            mov ax, [es: bx+3]
            xor ah, ah                      ; set BlockType
            mov [si+2], ax
            lds si, [cs : Block Type        ; set BlockType
            mov al, [es : bx]
            mov [si], al
            lds si, [cs : Continue]     ; set boolean
            mov byte ptr [si], 1        ; at default True
            pushf                           ; simulate Int
            call [DWORD PTR cs : UserRoutAddress]
            cmp byte ptr [si], 0        ; Continue = False ?
            jne ClearCarryFlag              ; no, process block
            cmp byte ptr [es : bx], 0       ; last block ?
            je ClearCarryFlag               ; yes, process this block
            pop ds                      ; restore used
            pop si                          ; registers
            pop ax
            stc                             ; don't process block
            retf                            ; trturn to CT-VOICE
ClearCarryFlag :
            pop ds                      ; restore used
            pop si                          ; registers
            pop ax
            clc                             ; process block
            retf                            ; return to CT-VOICE
ENDP
            END                             ; end of assembler file
```

程序清单9.2　支持CT-VOICE/CT-TIMER驱动程序的Pascal程序单元

```
Unit CTVOICE ;

{ Unit supporting the CT-VOICE driver.      }

Interface

Const
{ Errors occurring during the initialization }
  CallOk            = 0 ;
  CardError         = 1 ;
  IOPortError       = 2 ;
  DMAError          = 3 ;
{ For the speaker control }
  SpOn              = 1 ;
```

```
  SpOff              = 0 ;
{ Various block (Bl) types in the VOC format }
  EndBl              = 0 ;
  NwSampleBl         = 1 ;
  SampleBl           = 2 ;
  Silencebl          = 3 ;
  MarkerBl           = 4 ;
  TextBl             = 5 ;
  RepeatBl           = 6 ;
  RepeatEndBl        = 7 ;
  ExtraInfoBl        = 8 ; { Only for the Pro }
{ Possible Sound Blaster cards }
  SoundBlaster       = 0 ;
  SoundBlPro         = 1 ;
  SoundBl20          = 2 ;
{ The three parts that can serve as input }
  MIC                = 0 ;
  CD_ROM             = 1 ;
  Line               = 3 ;
{ Volume choices }
  MainVl             = 0 ;
  VoiceVl            = 1 ;
  MICVl              = 2 ;
  CD_ROMVBl          = 3 ;
  LineVl             = 4 ;
  Both               = 0 ;
  Left               = 1 ;
  Right              = 2 ;
{ Filter settings }
  LowFilter          = 0 ;
  HightFilter        = 1 ;
  OutputFilter       = 0 ;
  InputFilter        = 1 ;
{ Record mode }
  Sterep             = 1 ;
  Mono               = 0 ;
{ Normal on and off }
  On         = 0 ;
  Off        = 1 ;

Type
{ Dummy array of almost 64K }
  IntArType = Array [0..$fff0] of Byte ;
{ Record to keep info about VOC }
  VocTp      = Record
    Start    : Pointer ;
    Length   : LongInt ;
    Block    : ^IntArType ;
  End ;
```

```
{ Procedure for the user routine }
  Proc      = Procedure ;
  ProcTp    = ^Proc ;

Var
{ The status word indicates whether the CT-VOICE is busy }
  StatusWord          : Word ;
{ Points at the beginning of the sub-block }
  BlockStare          : ^IntArType ;
{ The length of the sub-block }
  BlockLength         : LongInt ;
{ The block type of the sub-block }
  BlockType           Byte ;
{ Indicates whether the next block has to be played }
  Continue            : Boolean ;
{ Type of card }
  CardType            : Byte ;
{ Result of procedures }
  SBIOResult          : Word ;

{** Disk I/O ** }
{ Load CT-VOICE driver }
Procedure LoadCTDriver(N : String) ;
{ Check whether a file meets the VOC format }
Function CheckVOCFile(N : String) : Boolean ;
{ Load VOC file }
Procedure LoadVOCFile(N : String; Var V : VocTp) ;
{ Writ VOC block }
Procedure SaveVOCFile(N : String; V : VocTp) ;
{ Remove the VOC block from memory }
Procedure DisposeVOC(Var V : VocTp) ;

{** Initialization ** }
{ Determine version }
Function CTVersion : Word ;
{ Set base port }
Procedure UsePort(P : Word) ;
{ Set base Port }
Procedure UseIRQ(I : Word) ;
{ Set DMA channel (for the PRO !) }
Procedure UseChannel(D : Word) ;
{ Initialize driver }
Procedure InitializeDriver ;

{** Utilities ** }
{ Turn speaker on or off }
Procedure Speaker(W : Word) ;
{ Stop recording or playing }
Procedure StopVProcess ;
{ Terminate the driver }
```

```
Procedure DriverOff ;
{ Set user routine }
Procedure UserRoutine(P : ProcTp) ;
{ Ask for minimum and maximum sampling rate }
Procedure AskRate(Var Max, Min : Word; M, K : Word) ;

{** Mixer chip for de PRO **}
{ Set stereo or mono record mode }
Procedure RecordMode(B : Word) ;
{ Set input }
Procedure Input(S : Word) ;
{ Turn the recording filter on or off }
Procedure RecordingFilter(B : Word) ;
{ Set volume }
Procedure Volume(S,C,V : Word) ;
{ Turn input or output filter on or off }
Procedure Filter(S,V : Word) ;
{ Reset mixer chip }
Procedure ResetMixer ;
{ Read all volume settings }
Procedure ReadAllVolumes ;
{ Ask the various volume and filter settings }
Function ReturnVolume(S,V : Word) : Word ;
Function ReturnFilter(S : Word) : Word ;

{** Playing ** }
{ Pause playing }
Procedure Pause ;
{ Continue playing }
Procedure ContinuePlaying :
{ Stop repetition }
Procedure StopRepetition (I : VocTp) ;
{ Play a VOC block }
Procedure StopRepetition(I : Word) ;
{ Play a VOC block in an EMB (for PRO !) }
Procedure PlayEMBBlock(H : Word; S : LongInt) ;

{** Recording ** }
{ Record a sample }
Procedure RecordSample(L : LongInt; SR : Word ;
                       Var V : VocTp) ;
{ Record a sample in an EMB (for PRO !) }
Procedure RecordEmBSample(SR : Word; H : Word; S : LongInt) ;

{** VOC utilities ** }
{ Set VOC Block at the first sub-block }
Procedure FirstSubBlock(Var V : VocTp) ;
{ Set VOC Block at the next sub-block }
Procedure NextSubBlock(Var V : VocTp) ;
```

```
{** Functions for de CT-TIMER.DRV ** }
{ Set volume table }
Procedure VolumeTable(V : Pointer; S : Byte) ;
{ Set echo buffer }
Procedure EchoBuffer(E : Pointer; S : Word) ;
{ Execute a command string }
Procedure Command(S : String) ;
{ Ask current volume }
Function CVolume : Byte ;

Implementation

Uses DOS ;

Var
  CTAddress : Pointer ;
  CTSize    : LongInt ;
  R         : Registers ;
{ Load and define machine code routines }
{ $L CTINTERF.OBJ }
{ $F+ }
Procedure CTVoiceAddress(P : Pointer) ; External ;
Procedure CallCTVoice(Var R : Registers) ; External ;
Procedure Variables (StPt, BsPt, BlPt, BtPt,
                     CoPt : Pointer) ; External ;
Procedure UserRoutine(P : ProcTp) ; External ;
{$F-}

{** Disk I/O ** }

Procedure LoadCTDriver(N : String) ;
Var
  F         : File ;
  S         : String[8] ;
  Off, Sg   : Word ;

Begin
  if N=' ' then N:= 'CT-VOICE.DRV' ;    { check for name }
  Assign(F, N) ; Reset (F, 1) ;
  Seek(F, 2) ; BlockRead(F, S, 9) ;     { read ID }
  Seek(F, 0) ;
  S[0] :=#8 ;                           { set length }
  If S=' CT-VOICE ' then Begin
    CTSize :=FileSize(F) ;              { determine size }
    GetMem(CTAddress, CTSize+16) ;
    Off :=Ofs(CTAddress^) ;             { Make sure }
    Sg :=Seg(CTAddress^) ;              { the CT-VOICE }
    Off :=Off and 15 ;                  { is loaded at an offset 0 }
```

```
    If Off <> 0 Then Begin              { offset = 0 ? }
      Inc(Sg) ;                           { next segment }
      Off :=0 ;                           { offset now is 0 }
    End ;
    CTAddress := Ptr(Sg, Off) ;
    BlockRead(F, CTAddress^, CTSize) ;
    Close(F) ;
    CTVoiceAddress(CTAddress) ;
    SBIOResult :=0 ;
  End
  Else Begin
    Close(F) ;                { no driver }
    SBIOResult := 1 ;
  End
End ;

Function CheckVOCFile(N : String) : Boolean ;

Var
  F          : File ;
  S          : String[19] ;
  T1, T2     : Word ;
  R          : Boolean ;
Begin
  R :=False ;                           { set result }
  Assign(F, N) ;Reset(F, 1) ;
  BlockRead(F, S, 19) ;                 { read header ID }
  S[0] :=#18 ;
  If S='Creative Voice File' Then Begin
    Seek(F,$16) ;                       { point at version }
    BlockRead(F, T1, 2) ;
    BlockRead(F, T2, 2) ;
    T1 :=(t1 XOR $FFFF) + $1234 ;       { check calcul. }
    R :=T1=T2 ;                         { equal ? }
  End ;
  Close(F) ;
  CheckVOCFile :=R ;                    { return result }
End ;
Procedure LoadVOCFile(N : String; Var V : VocTp) ;
Var
  F          : File ;
  T          : Word ;
  SgH, OfH   : Word ;
  SgE, OfE   : Word ;
  HS         : LongInt ;

Begin
  Assign(F, N) ; Reset(F, 1) ;
  Seek(F, $14) ;                    { read header size }
```

```
  BlockRead(F, T, 2) ;
  Seek(F, T) ;                          { skip it }
  V.Length :=FileSize(F)-T ;
  GetMem(V.Start, V.Length) ;
  BlockRead(F, V.Start^, V.Length) ;
  Close(F) ;
  V.Block :=V.Start ;
End ;

Procedure SaveVOCFile (N : String; V : VocTp) ;
Var
  F          : File ;
  S          : String ;
  T          : Word ;

Begin
  Assign(F, N) ; ReWrite(F, 1) ;
  S :='Creative Voice File'+#$1A ;          { ID Text }
  BlockWrite(F, S, 20) ;
  T :=$001A; BlockWrite(F, T, 2) ;          { header size }
  T :=$010A; BlockWrite(F, T, 2) ;          { Version 1.10 }
  T :=(T-$1234) XOR $FFFF ;                 { check }
  BlockWrite(F, T, 2) ;
  BlockWrite(F, V.Start^, V.Length) ;       { sample data }
  Close(F) ;
End ;
Procedure DisposeVOC(Var V : VocTp) ;
Begin
  FreeMem(V.Start, V.Length) ;
  V.Start :=Nil ;
  V.Length :=0 ;
  V.Block :=Nil ;
End ;

{** Initialization ** }

{ The next procedures and functions take care of the }
{ interface between the CT-VOICE driver and Pascal. }
{ The functionong is quite logical and simple : }
{ no further comment is provided .}

Function CTVersion : Word ;

Begin
  R.BX :=0 ;
  CallCTVoice(R) ;
  CTVersion :=R.AX
End ;
```

```
Procedure UsePort(P : Word) ;
Begin
  R.BX :=1 ;
  R.AX :=P ;
  CallCTVoice(R) ;
  SBIOResult :=R.AX ;
End ;

Procedure UseIRQ(I : Word) ;
Begin
  R.BX :=2 ;
  R.AX :=I ;
  CallCTVoice(R) ;
  SBIOResult :=R.AX ;
End ;

Procedure UseChannel(D : Word) ;
Begin
  R.BX :=19 ;
  R.AX :=D ;
  CallCTVoice(R) ;
  SBIOResult :=R.AX ;
End ;

Procedure InitializeDriver ;
Begin
  R.BX :=3 ;
  CallCTVoice(R) ;
  If R.AX=0 Then
    Variables(@StatusWord, @BlockStart, @BlockLength,
              @BlockType, @Continue) ;
  SBIOResult :=R.AX ;
End ;

{** Utilities ** }

Procedure Speaker(W : Word) ;
Begin
  R.BX :=4 ;
  R.AX :=W ;
  CallCTVoice(R) ;
End ;

Procedure StopVProcess ;
Begin
  R.BX :=8 ;
  CallCTVoice(R) ;
End ;
```

```
Procedure Driveroff ;
Begin
  R.BX :=9 ;
  CallCTVoice(R) ;
End ;

Procedure UserRoutine(P : ProcTp) ;
Begin
  If P=Nil Then
    UserRoutine(Ptr(0,0))
  Else
    UseRoutine(P) ;
End ;

Procedure AskRate(Var Max, Min : Word; M, K : Word) ;
Begin
  R.BX :=26 ;
  R.AX :=K ;
  R.DX :=M ;
  CallCTVoice(R) ;
  Max :=R.DX ;
  Min :=R.AX ;
End ;

{** Mixer chip ** }

Procedure RecordMode(B : Word) ;
Begin
  R.AX :=B ;
  R.BX :=16 ;
  CallCtVoice(R) ;
  SBIOResult :=R.AX ;
End ;

Procedure Input(S : Word) ;
Begin
  R.AX :=S ;
  R.BX :=17 ;
  CallCtVoice(R) ;
  SBIOResult :=R.AX ;
End ;

Procedure RecordingFilter(B : Word) ;
Begin
  R.AX :=B ;
  R.BX :=18 ;
  CallCtVoice(R) ;
  SBIOResult :=R.AX ;
End ;
```

```
Procedure Volume(S,C,V : Word) ;
Begin
  If C=Both Then Begin
    Volume(S, Left, V) ;
    C :=Right ;
  End ;
  R.AX :=S ;
  R.BX :=21 ;
  R.CX :=V ;
  R.DX :=C ;
  CallCtVoice(R) ;
  SBIOResult :=R.AX ;
End ;

Procedure Filter(S, V : Word) ;
Begin
  R.AX :=S ;
  R.BX :=22 ;
  R.CX :=V ;
  CallCtVoice(R) ;
  SBIOResult :=R.AX ;
End ;
Procedure ResetMixer ;
Begin
  R.BX :=23 ;
  CallCtVoice(R) ;
End ;

Procedure ReadAllVolumes ;
Begin
  R.BX :=24 ;
  CallCtVoice(R) ;
End ;

Function ReturnVolume(S,V : Word) : Word ;

Begin
  R.AX :=S ;
  R.BX :=25 ;
  R.DX :=V ;
  CallCtVoice(R) ;
  ReturnVolume :=R.AX ;
End ;

Function ReturnFilter(S : Word) : Word ;

Begin
  If CTVersion=$20A Then
```

```
    R.BX :=27
  Else
    R.BX :=26 ;
  R.AX :=S ;
  CallCtVoice(R) ;
  ReturnFilter :=R.AX ;
End ;

{** Playing ** }

Procedure Pause ;
Begin
  R.BX :=10 ;
  CallCTVoice(R) ;
  SBIOResult :=R.AX ;
End ;

Procedure ContinuePlaying ;
Begin
  R.BX :=11 ;
  CallCTVoice(R) ;
  SBIOResult :=R.AX ;
End ;

Procedure StopRepetition(I : Word) ;
Begin
  R.BX :=12 ;
  R.AX :=I ;
  CallCTVoice(R) ;
  SBIOResult :=R.AX ;
End ;

Procedure PlayBlock(V : VocTp) ;
Begin
  R.Bx :=6 ;
  R.DI :=Ofs(v.Start^) ;
  R.ES :=Seg(V.Start^) ;
  CallCTVoice(R) ;
End ;

Procedure PlayEMBBlock(H : Word; S : LongInt) ;
Begin
  R.BX :=14 ;
  R.DX :=H ;
  R.DI :=S Shr 16 ;
  R.SI :=S And $FFFF ;
  CallCTVoice(R) ;
  SBIOResult :=R.AX ;
End ;
```

```
{** Recording ** }

Procedure RecordSample(L : LongInt ; SR : Word ;
                       Var V : VocTp) ;
Begin
  GetMem(V.Start, L) ;
  V.Length :=L ;
  R.BX :=7 ;
  R.AX :=SR ;
  R.DX :=L shr 16 ;
  R.CX :=L and 65535 ;
  R.ES :=Seg(V.Start^) ;
  R.Di :=Ofs(V.Start^) ;
  CallCTVoice(R) ;
End ;

Procedure RecordEMBSample(SR : Word; H : Word; S : LongInt) ;
Begin
  R.AX :=SR ;
  R.BX :=15 ;
  R.DX :=H ;
  R.DI :=S Shr  16 ;
  R.SI :=S And $FFFF ;
  CallCTVoice(R) ;
  SBIOResult :=R.AX ;
End ;

{** VOC utilities ** }

Procedure FirstSubBlock(Var V : VocTp) ;
Begin
  V.Block :=V.Start ;
End ;

Procedure NextSubBlock(Var V : VocTp) ;
Var
  A : LongInt ;

Begin
{ Calculate 24-bit address }
  A:=16*LongInt(Seg(V.Block^))+LongInt(Ofs(V.Block^)) ;
{ Add the following : 4 + block length }
  A:=A+4+V.Block^[1]+256*V.Block^[2]+256*256*V.Block^[3] ;
{ Set V.Block }
  V.Block :=Ptr(A shr 4, A and 15) ;
End ;

{** Functions for the CT-TIMER.DRV ** }
```

```
Procedure VolumeTable(V : Pointer ; S : Byte) ;
Begin
  R.BX :=128 ;
  R.AX :=0 ;
  R.CL :=S ;
  R.DI :=Ofs(V^) ;
  R.ES :=Seg(V^) ;
  CallCtVoice(R) ;
End ;

Procedure EchoBuffer(E : Pointer ; S : Word) ;
Begin
  R.BX :=128 ;
  R.AX :=1 ;
  R.CX :=S ;
  R.DI :=Ofs(E^) ;
  CallCtVoice(R) ;
End ;

Procedure Command(S : String) ;
Begin
  S :=S+#0 ;
  R.BX :=129 ;
  R.DI :=Ofs(S[1]) ;
  R.ES :=Seg(S[1]) ;
  CallCtVoice(R) ;
End ;

Function CVolume : Byte ;

Begin
  R.BX :=130 ;
  CallCtVoice(R) ;
  CVolume :=R.AL ;
End ;

End.
```

注解 程序清单9.2使用常规的存贮器地址分配程序，在Pascal中，只可使用具有最大尺寸为65520字节的VOC文件，因为GetMem不能保存大的存贮器块。

程序清单9.3 Turbo C的汇编接口

```
                .MODEL SMALL
                .CODE
; File : CT-INTER.ASM
; Assembler file taking care of the interface
; between the CT-VOICE driver and Turbo C.
;
```

```
; Run TASM /MX CT-INTER to create CTINTER.OBJ.
; Next, link CT-INTER.OBJ to the remaining files.
;
; Defined functions :
; - void CallCTVoice(struct REGPACK far *R) ;
; - void CTVoiceAddress(char far *A, unsigned G) ;
; - void Variables(void far *StPt, void far *BsPt,
;                  void far *BlPt, void far *BtPt,
;                  Void far *CoPt)
; - void UserRoutine(void interrupt *P) ;
PUBLIC _CTVoiceAddress, _CallCTVoice
PUBIC _Variables, _UserRoutine

; Define a structure for the type REGPACK
; that can serve for the offset calculation.
Registers STRUC
rAX    dw 0
rBX    dw 0
rCX    dw 0
rDX    dw 0
rBP    dw 0
rSI    dw 0
rDI    dw 0
rDS    dw 0
rES    dw 0
rFlags dw 0
ENDS

; Define a frequently used routine :
; copying a SWORD

@CopyDword MACRO Dest, Source
       mov ax, WORD PTR Source+0
       mov WORD PTR Dest+0, ax
       mov ax, WORD PTR Source+2
       mov WORD PTR Dest+2, ax
ENDM

; The global variables

CTVoicePt        DD 0        ; CT-VOICE address
UserRoutAddress  DD 0        ; user routine
BlockStart       DD 0        ; Pointer address
BlockLength      DD 0        ; LongInt address
BlockType        DD 0        ; Byte address
Continue         DD 0        ; Boolean address

; void CTVoiceAddress(char far *A, unsigned P)
; Set the start address of the CT-VOICE driver.
```

```
_CTVoiceAddress PROC FAR
        ARG A : DWORD, G : WORD
        push bp
        mov bp, sp
        push ds
        lds si, [A]                     ; DS : SI points
        mov dx, ds                      ; at CT-VOICE
        mov ax, si                      ; at CT-VOICE
        add ax, 15                      ; DX : AX=DS : SI
        shr ax, 1                       ; minimize and
        shr ax, 1                       ; round up
        shr ax, 1
        shr ax, 1
        add dx, ax
        mov es, dx                      ; ES : 0 starts CT-VOICE
        mov cx, [G]                     ; CX = driver size
        mov di, cx                      ; start at the end
        add si, cx                      ; of the driver
        inc cx                          ; one byte more
        std
        rep movsb                       ; copy driver data
        cld
        mov [WORD PTR CTVoicePt+0], 0
        mov [WORD PTR CTVoicePt+2], es
        pop ds
        pop bp
        ret
ENDP

; void CallCTVoice(struct REGPACK far *R) ;
; Calls the CT-VOICE driver.

_CallCTVoice PROC FAR
        ARG R : DWORD
        push bp
        mov bp, sp
        push ds                         ; has to be saved
        lds si, [R]                     ; DS : SI points at R
        mov, ax, [si+rSI]               ; save used SI
        push ax
        mov ax, [si+rAX]                ; set all registers
        mov bx, [si+rBX]
        mov cx, [si+rCX]
        mov dx, [si+rDX]
        mov di, [si+rDI]
        mov es, [si+rES]
        mov ds, [si+rDS]
        pop si                          ; set SI
        call [DWORD PTR cs : CtVoicePt]
```

```
        push ds                     ; save DS
        push si                     ; save SI
        lds si, [R]                 ; DS : SI points at R
        mov [si+rAX], ax            ; save all regs.
        mov [si+rBX], bx
        mov [si+rCX], cx
        mov [si+rDX], dx
        mov [si+rDI], di
        mov [si+rES], es
        pop ax                      ; AX=SI after call
        mov [si+rSI], ax
        pop ax                      ; AX=DX after call
        mov [si+rDS], ax
        pop ds
        pop bp
        ret
ENDP

; void Variables(void far *StPt, void far *BsPt,
;                void far *BlPt, void far *BtPt,
;                void far *CoPt)
; Set the pointers to StatusWord, BlockStart,
; BlockLength, BlockType & Continue.

_Variables PROC FAR
        ARG Stp : DWORD, Bsp : DWORD, BlP : DWORD, BtP : DWORD, CoP : DWORD
        push bp
        mov bp, sp
        @CopyDword cs : BlockStart, Bsp
        @CopyDword cs : BlockLength, Blp
        @CopyDword cs : VlockType, Btp
        @CopyDword cs : Continue, Cop
        mov bx, 5                   ; set pointer to
        les di, StP                 l StatusWord
        call [DWORD PTR cs : CTVoicePt]
        pop bp
        ret
ENDP
; void UserRoutine(void interrupt *P) ;
; Set user interrupt.

_UserRoutine PROC FAR
        ARG P : DWORD
        push bp
        mov bp, sp
        mov ax, [WORD PTR P+0]
        mov [WORD PTR cs : UserRoutAddress+0], ax
        mov dx, [WORD PTR P+2]
        mov [WORD PTR cs : UserRoutAddress+2], dx
```

```
        or ax, dx                       ; P=0 : 0 ?
        jnz NewUserRoutine
        mov bx, 13                      ; yes, turn off routine
        call [DWORD PTR cs : CTVoicePt]
        pop bp
        ret
NewUserRoutine :
        mov ax, offset UserInterFace
        mov dx, cs
        mov bx, 13              ;  set interface
        call [DWORD PTR cs : CTVoicePt]
        pop bp
        ret
ENDP
; This routine takes care of the interface for the
; interrupt routine in C. The routine
; sets the required variables and calls the C
; routine. After the call, it checks whether
; the next block has to be played.

UserInterFace PROC FAR
        push ax                 ; save used
        push si                 ; registers
        push ds
        lds si, [cs : BlockStart]       ; set BlockStart
        mov [si], bx            ; starting from
        mov [si+2], es          ; block to be played
        lds si, [cs : BlockLength]      ; set BlockLength
        mov ax, [es : bx+1]     ; with the de 24-bit
        mov [si], ax            ; length
        mov ax, [es : bx+3]
        xor ah, ah              ; delete 25..32 bits
        mov [si+2], ax
        lds si, [cs : BlockType]        ; set BlockType
        mov al, [es : bx]
        mov [si], al
        lds si, [cs : Continue]         ; set boolean
        mov byte ptr [si], -1           ; at default True
        pushf
        call [DWORD PTR cs : UserRoutAddress]
        cmp byte ptr [si], 0            ; Continue = False ?
        jne ClearCarryFlag              ; no, process block
        cmp byte ptr [es : bx], 0       ; last block ?
        je ClearCarryFlag               ; yes, process this block
        pop ds                          ; restore used
        pop si                          ; registers
        pop ax
        stc                             ; don't process block
        retf                            ; return to CT-VOICE
```

```
ClearCarryFlag :
        pop ds                  ; restore used
        pop si                  ; registers
        pop ax
        clc                     ; process block
        retf                    ; return to CT-VOICE
ENDP
        END                     ; end assembler file
```

程序清单9.4　Turbo C的汇编接口

```
/* File : CT-INTER.H */
/* Header for the interface with the machine code */

/* Define machine code interface routines */
extern void far CallCTVoice(struct REGPACK far *R) ;
extern void far UserRoutine(void interrupt far *P) ;
extern void far CTVoiceAddress(char far *P, unsigned S) ;
extern void far Variables(void far *StPt,
                  void far *BsPt, void far *BlPt,
                  void far *BtPt, void far *CoPt) ;
```

注解　程序清单9.4使用常规的存贮器地址分配程序，在Pascal中，只可使用具有最大尺寸为65520字节的VOC文件，因为GetMem不能保存大的存贮器块。

程序清单9.5　CT-INTER.OBJ与CTVOICE.C接口的头文件

```
/* File : CTVOICE.H                                    */
/* Header for CTVOICE library.                         */

/* Errors occurring during initialization      */
#define CallOk          0
#define CardError       1
#define IOPortError     2
#define DMAError        3
/* For speaker control */
#define SpOn            1
#define SpOff           0

/* Various block (Bl) types in VOC format */
#define EndBl           0
#define NwSampleBl      1
#define SampleBl        2
#define SilenceBl       3
#define MarkerBl        4
#define TextBl          5
#define RepeatBl        6
#define RepeatEndBl     7
#define ExtraInfoBl     8       /* Only for the Pro */
```

```
/* Possible Sound Blaster cards */
#define SoundBlaster  0
#define OldSBPro      1
#define NewSBPro      2

/* The three part that can serve as input */
#define MIC           0
#define CD_ROM        1
#define Line          3

/* Volume choices */
#define MainVl        0
#define VoiceVl       1
#define MICVl         2
#define CD_ROMVl      3
#define LineVl        4
#define Both          0
#define Left          1
#define Right         2

/* Filter settings */
#define LowFilter     0
#define HighFilter    1
#define OutputFilter  0
#define InputFilter   1

/* Different record modes */
#define Stereo        1
#define Mono          0

/* Normal on and off */
#define On            0
#define Off           1

/* Dummy array of almost 64K */
typedef unsigned char IntArType[Oxfff0] ;
/* Record to keep info about VOC */
typedef struct {
  void *Start ;
  long Length ;
  IntArType far *Block ;
} VocTp ;

/* Global variables */
extern unsigned StatusWord ;
extern IntArType *BlockStart ;
extern long BlockLength ;
extern unsigned char BlockType ;
extern unsigned char Continue ;
```

```
extern unsigned char CardType ;
extern unsigned SBIOResult ;

/* Procedure for the user routine */

/*** Disk I/O ***/
/* Load CT-VIOCE driver */
extern void LoadCTDriver(char *N) ;
/* Dies a file meet the VOC format ? */
extern char CheckVOCFile(char *N) ;
/* Load VOC file */
extern void LoadVOCFile(char *N, VocTp *V) ;
/* Write a VOC block */
extern void SaveVOCFile(char *N, VocTp V) ;
/* Remove the VOC block from memory */
extern void DisposeVOC(VocTp *V) ;

/*** Initialization ***/
/* determine version */
extern unsigned CTVersion() ;
/* Set base port */
extern void UsePort(unsigned P) ;
/* Set IRQ number */
extern void UseIRQ(unsigned I) ;
/* Set DMA channel(for the PRO !) */
extern void UseChannel(unsigned D) ;
/* Initialize driver */
extern void InitializeDriver() ;

/*** Utilities ***/
/* Turn speaker on or off */
extern void Speaker(unsigned W) ;
/* Stop recording or playing */
extern void StopVProcess() ;
/* Terminate driver */
extern void DriverOff() ;
/* Set user routine */
extern void UserRoutine(void interrupt far *P) ;
/* Ask maximum and minimum sampling rate */
void AskRate(unsigned *Max, unsigned *Min,
             unsigned M, unsigned K) ;

/*** Mixer chip, for the PRO ***/
/* Set stereo or mono mode for recording */
extern void RecordMode(unsigned B) ;
/* Determine input */
extern void Input(unsigned S) ;
/* Turn recording filter on or off */
extern void RecordingFIlter(unsigned B) ;
```

```
/* Set volume */
extern void Volume(unsigned S, unsigned C, unsigned V) ;
/* Turn input or output filter on or off */
extern void Filter(unsigned S, unsigned V) ;
/*Reset mixer chip */
extern void ResetMixer() ;
/* Read all volume settings */
extern void ReadAllVolumes() ;
/* Ask various volume and filter settings */
extern unsigned ReturnVolume(unsigned S, unsigned V) ;
extern unsigned ReturnFilter(unsigned S) ;

/*** Playing ***/
/* Pause playing */
extern void Pause() ;
/* Continue playing */
extern void ContinuePlaying() ;
/* Stop repetition */
extern void StopRepetition(unsigned I) ;
/* Play VOC block */
extern void PlayBlock(VocTp V) ;
/* Play VOC block in an EMB (for PRO !) */
extern void PlayEMBBlock(unsigned H, long S) ;
/*** Recording ***/
/* Record a sample */
extern void RecordSample(long L, unsigned SR,
                         VocTp *V) ;
/* Record a sample in an EMB(for PRO !) */
extern void RecordEMBSample(unsigned SR, unsigned H,
                            long S) ;

/*** VOC utilities ***/
/* Set VOC.Block at first sub-block */
extern void FirstSubBlock(VocTp *V) ;
/* Set VOC.Block at next sub-block in */
extern void NextSubBlock(VocTp *V) ;

/*** Functions for CT-TIMER.DRV ***/
/* Set volume table */
extern void VolumeTable(void *V, unsigned char S) ;
/* Set echo buffer */
extern void EchoBuffer(void *E, unsigned S) ;
/* Execute command string */
extern void Command(char * S) ;
/* Ask current volume */
extern unsigned char CVolume() ;
```

程序清单9.6 支持CT-VOICE/CT-TIMER驱动程序的C函数库

```
/* File : CTVOICE.C                          */
/* Library supporting the CT-VOICE driver */

#include <stdio.h>
#include <dos.h>
#include <alloc.h>
#include "CT-INTER.H"
#define extern          /* define global */
#include "CTVOICE.H"
#undef extern
struct REGPACK R ;
char *CTAddress ;

/*** Disk I/O ***/

void LoadCTDriver(char *N)
{
  FILE *F ;
  char S[9] ;
  unsigned CTSize ;

  if (strcmp(N, "") == 0)
    F=fopen("CT_VOICE.DRV", "rb") ;
  else
    F=fopen(N, "rb") ;
  fseek(F, 3, SEEK_SET) ; fread(&S, 9, 1, F) ;     /* read ID */
  fseek(F, OL, SEEK_END) ;
  S[9] = 0 ;                                  /* set length */
  if (strcmp(S, "CT-VOICE") == 0) {
    CTSize = ftell(F) ;               /* determine size */
    fseek(F, OL, SEEK_SET) ;
    CTAddress = malloc(CTSize + 16) ;
    fread(CTAdres, CTSize, 1, F) ;
    fclose(F) ;
    CTVoiceAddress((char far *) CTAddress, CTSize) ;
    SBIOResult = 0 ;
  }
  else {
    fclose(F) ;                               /* no driver */
    SBIOResult =1 ;
  }
}

char CheckVOCFile(char *N)
{
  FILE *F ;
```

```
  char S{20] ;
  unsigned T1, T2 ;
  char R ;
  R = 0 ;                          /* set result */
  F=fopen(N,"rb") ;
  fread(S, 19, 1, F) ;             /* read header ID */
  S[20] = 0 ;
  if (strcmp(S, "Creative Voice File") == 0) {
    fseek(F, )x16, SEEK_SET) ;     /* point at version */
    fread(&T1, 2, 1, F) ;
    dread(&T2, 2, 1, F) ;
    T1 = (T1 ^ OxFFFF) + Ox1234 ;  /* verify calc. */
    R = T1 ==T2 ;                  /* equal ? */
  }
  fclose(F) ;
  return R;                        /* return result */
}

void LoadVOCFile (char *N, VocTP *V)
{
  FILE *F ;
  unsigned T ;
  unsigned SgH, OfH ;
  unsigned SgE, OfE;
  long HS ;

  F=fopen(N, "rb") ;
  fseek(F, Ox14, SEEK_SET) ;                  /* read header size */
  fread(&T, 2, 1, F) ;
  fseek(F, OL, SEEK_END) ;                    /* skip it */
  V->Length = ftell(F) - T ;
  fseek(F,T,SEEK_SET) ;
  V->Start = malloc(V->Length) ;
  fread(V->Start, V->Length, 1, F) ;
  fclose(F) ;
  V->Block = V->Start ;
}

void SaveVOCFile(char *N, VocTp V)
{
  FILE *F ;
  char S[21]= "Creative Voice File\x1A" ;
  unsigned T ;

  F = fopen(N, "wb") ;
  fwrite(S, 20, 1, F) ;
  T = Ox001A ; fwrite(&T, 2, 1, F) ;     /* header size */
  T = Ox010A ; fwrite(&T, 2, 1, F) ;     /* check */
  T = (T－0x1234) ^0xFFFF ;              /* check */
```

```
  fwrite(&T, 2, 1, F) ;
  fwrite(V.Start, V.Length, 1, F) ;        /* sample data */
  fclose(F) ;
}

void DisposeVOC(VocTp * V)
{
  free(V->Start) ;
  V->Start = NULL ;
  V->Length = 0 ;
  V->Block = NULL ;
}

/*** Initialization ***/

/* The next procedures and functions take care of          */
/* the interface between the CT-VOICE driver and C.        */
/* Their functioning is quite logical and simple :         */
/* no further comment is provided.                         */

unsigned CTVersion(void)
{
  R.R_bx = 0 ;
  CallCTVoice(&R) ;
  return R.r_ax ;
}

void UsePort(unsigned P)
{
  R.r_bx = 1 ;
  R.r_ax = P ;
  CallCTVoice(&R) ;
  SBIOResult = R.r_ax ;
}

void UseIRQ(unsigned I)
{
  R.r_bx = 2 ;
  R.r_ax = I ;
  CallCTVoice(&R) ;
  SBIOResult = R.r_ax ;
}

void UseChannel(unsigned D)
{
  R.r_ax = 19 ;
  R.r_ax = D ;
  CallCTVoice(&R) ;
  SBIOResult = R.r_ax ;
```

```
}

void InitializeDriver(void)
{
  R.r_bx =3 ;
  CallCTVoice(&R) ;
  Variables(&StatusWord, &BlockStart, &BlockLength,
            &BlockType, &Continue) ;
  SBIOResult = R.r_ax ;
}
/*** Utilities ***/

void Speaker(unsigned W)
{
  R.r_bx = 4 ;
  R.r_ax = W ;
  CallCTVoice(&R) ;
}

void StopVProcess(void)
{
  R.r_bx = 8 ;
  CallCTVoice(&R) ;
}

void DriverOff(void)
{
  R.r_bx = 9 ;
  CallCTVoice(&R) ;
}

void UserRoutine(void interrupt far *P)
{
  if (P == NULL)
    UserRoutine(MK_FP(0, 0)) ;
  else
    UseRoutine(P) ;
  }

void AskRate(unsigned *Max, unsigned *Min,
             unsigned M, unsigned K)
{
  R.r_bx = 26 ;
  R.r_ax = k ;
  R.r_dx = M ;
  CallCTVoice(&R) ;
  *Max = R.r_dx ;
  *Min = R.r_ax ;
}
```

```
void Input(unsigned S)
{
  R.r_ax = S ;
  R.r_bx = 17 ;
  CallCTVoice(&R) ;
  SBIOResult = R.r_ax ;
}

void RecordingFilter(unsigned B)
{
  R.r_ax = B ;
  R.r_bx = 18 ;
  CallCTVoice(&R) ;
  SBIOResult = R.r_ax ;
}

void Volume(unsigned S,
            unsigned C,
            unsigned V)
{
  if (C == Both) {
    Volume(S, Left, V) ;     /* first left */
    C = Right ;              /* then right */
  }
  R.r_ax = S ;
  R.r_bx = 21 ;
  R.r_cx = V ;
  R.r_dx = C ;
  CallCTVoice(&R) ;
  SBIOResult = R.r_ax ;
}

void Filter(unsigned S,
            unsigned V)
{
  R.r_ax = S ;
  R.r_bx = 22 ;
  R.r_cx = V ;
  CallCTVoice(&R) ;
  SBIOResult = R.r_ax ;
}

void ResetMixer(void)
{
  R.r_bx = 23 ;
  CallCTVoice(&R) ;
}

void ReadAllVolumes(void)
```

```
{
  R.r_bx = 24 ;
  CallCTVoice (&R) ;
}

unsigned ReturnVolume(unsigned S, unsigned V)
{
  R.r_ax = S ;
  R.r_bx = 25 ;
  R.r_dx = V ;
  CallCTVoice(&R) ;
  return R.r_ax ;
}
unsigned ReturnFilter(unsigned S)
{
  if (CTVersion() >= Ox20A)       /* as from 2.10 other */
    R.r_bx = 27 ;                 /* function number */
  else
    R.r_bx = 26 ;
  extern unsigned
  R.r_ax = S ;
  CallCTVoice(&R) ;
  return R.r_ax ;
}

/*** Playing ***/

void Pause(void)
{
  R.r_bx = 10 ;
  CallCTVoice(&R) ;
  SBIOResult = R.r_ax ;
}

void ContinuePlaying(void)
{
  R.r_bx = 11 ;
  CallCTVoice(&R) ;
  SBIOResult = R.r_ax ;
}
void StopRepetition(unsigned I)
{
  R.r_bx = 12 ;
  R.r_ax = I ;
  CallCTVoice(&R) ;
  SBIOResult = R.r_ax ;
}

void PlayBlock(VocTP V)
```

```
{
  R.r_bx = 6 ;
  R.r_di = FP_OFF(V.Start) ;
  R.r_es = FP_SEG(V.Start) ;
  CallCTVoice(&R) ;
}

void PlayEMBBlock(unsigned H, long S)
{
  R.r_bx = 14 ;
  R.r_dx = H ;
  R.r_di = S >> 16 ;
  R.r_si = S & Oxffff ;
  CallCTVoice(&R) ;
  SBIOResult = R.r_ax ;
}

/*-** Recording ***/
void RecordSample(long L, unsigned SR, VocTp *V)
{
  V->Start = (char far *) malloc(L) ;
  V->Length = L ;
  R.r_bx = 7 ;
  R.r_ax = SR ;
  R.r_dx = : > 16 ;
  R.r_cx = : & 65535 ;
  R.r_es = FP_SEG(V-Start) ;
  R.r_di = FP_OFF(V->Start) ;
  CallCTVoice(&R) ;
}

void RecordEMBSample(unsigned SR, unsigned H, long S)
{
  R.r_ax = SR ;
  R.r_bx = 15 ;
  R.r_dx = H ;
  R.r_di = S > 16 ;
  R.r_si + S & Oxffff ;
  CallCTVoice(&R) ;
  SBIOResult = R.r_ax ;
}

/*** VOC utilities ***/
void FirstSubBlock (VocTP * V)
{
  V->Block = V->Start ;
}

void NextSubBlock(VocTP * V)
```

```
{
  V-> Block = V-> Start ;
}

void NextSubBlock(VocTp * V)
{
  long A ;

  /* Calculate 24-bit address */
  A = 16 * (long) FP_SEG(*V->Block) + (long) FP_OFF(*V->Block) ;
  /* Add : 4 + block length */
  A = A + 4 + (*V->Block[1]) + (*V->Block[2]) <8 +
(*V->Block[3]) < 16 ;
  /* Set V.Block */
  V->Block = MK_FP(A>4 , A & 15) ;
}

/*** Functions for the CT-TIMER.DRV ***/

void VolumeTable(void *V, unsigned char S)
{
  void far *H = (void far *) V ;

  R.r_bx = 128 ;
  R.r_ax = 0 ;
  R.r_cx = S ;
  R.r_di = FP_OFF(H) ;
  R.r_es = FP_SEG(H) ;
  CallCTVoice(&R) ;
}

void EchoBuffer(void *E, unsigned S)
{
  void far *H = (void far *) E ;

  R.r_bx = 128 ;
  R.r_ax = 1 ;
  R.r_cx = S ;
  R.r_di = FP_OFF(H) ;
  R.r_es = FP_SEG(H) ;
  CallCTVoice(&R) ;
}

void Command(char * S)
{
  strcat(S, 0) ;
  R.r_bx = 129 ;
  R.r_di = FP_OFF(S) ;
  R.r_es = FP_SEG(S) ;
```

```
/*** Mixer chip ***/

void RecordMode(unsigned B)
{
  R.r_ax = B ;
  R.r_bx = 16 ;
  CallCTVoice(&R) ;
  SBIOResult = R.r_ax ;
}
  CallCTVoice(&R) ;
}

unsigned char CVolume(void)
{
  R.r_bx = 130 ;
  CallCTVoice(&R) ;
  return R.r_ax & Oxff ;
}
```

CT-VOICE测试程序实例

程序清单9.7到9.10是一些例子，这些程序示范了用函数库进行记录和播放样本。这四个程序用Pascal和C以同样的方式进行。程序7和8是供标准的Sound Blastere和，程序9和10是针对Sound Blaster Pro并示范了卡的扩充能力。

程序列表如下：

编号	标　　题
9.7	测试标准CT-VOICE驱动程序Pascal程序
9.8	测试标准CT-VOICE驱动程序的程序
9.9	扩充的CT-VOICE驱动程序的Pascal测试程序
9.10	填充的CT-VOICE驱动程序的C测试程序

程序清单9.7　标准CT-VOICE驱动程序的Pascal测试程序

```
Program CTTest ;

{ Test program fror the standard CT-VOICE driver. }

Uses CTVoice, CRT ;

Var
  Sample : VocTp ; { sample }

Begin
  ClrScr ;
  LoadCTDriver(' ') ;                    { load driver }
  WriteLn(Hi (CTVersion), '.', Lo(CTVersion)) ;
```

```
  UsePort($220) ;                    { set port }
  UseIRQ(7) ;                        { set IRQ }
  UseChannel(1) ;
  InitializeDriver ;
  If SBIOResult = Callok Then Begin
    Speaker(SpOff) ;                 { turn off connection }
    RecordSample(60000, 9000, Sample ) ;
    GotoXY(1,1) ; Write('Recording ...') ;
    Repeat                           { wait until the }
    Until StatusWord=0 ;             { sample is recorded }
    FirstSubBlock(Sample) ;
    NextSubBlock(Sample) ;
    Sample.Block^[0] :=0 ;           { add end block }
    Inc(Sample.Length) ;             { incl. end block }
    SaveVOCFile('Test.Voc', Sample) ; { write }
    DisposeVoc(Sample) ;             { from memory }
    LoadVOCFile('Test.Vov', Sample) ; { load sample }
    GotoXY(1, 2) ;
    Write('Playing ...') ;
    Speaker(SpOn) ;                  { turn on passage }
    PlayBlock(Sample) ;              { play back sample }
    Repeat
    Until StatusWord=0 ;             { wait till the end }
    DriverOff ;                      { everything off }
    WriteLn ;
  End
  Else
    WriteLn('Error code : ', SBIOResult) ;
End.
```

程序清单9.8　标准CT-VOICE驱动程序的测试程序

```
/* Test program for the standard CT-VOICE driver.        */

#include <conio.h>
#include "ctvoice.h"

VocTp Sample ;  /* sample */
void main()
{
  clrscr() ;
  LoadCTDriver("") ;                 /* load de driver */
  printf("%d.%d\n", CTVersion() > 8, CTVersion() & Oxff) ;
  UsePort(Ox240) ;                   /* set port */
  UseIRQ(10) ;                       /* set IRQ */
  UseChannel(0) ;
  InitializeDriver() ;
  if (SBIOResult == Callok) {
```

```
    Input(CD_ROM) ;
    Speaker(SpOff) ;                /* turn off connection */
    RecordSample(40000, 9000, &Sample) ;
    gotoxy(1, 1) ; printf("Recording ...") ;
    do {                            /* wait until the sample is recorded */
    } while (!(StatusWord == 0)) ;
    FirstSubBlock(&Sample) ;
    NextSubBlock(&Sample) ;
    *Sample.Block[0] = 0 ;          /* add end block */
    ++Sample.Length ;               /* incl. end block */
    SaveVOCFile("Test. Voc", Sample) ;  /* write */
    DisposeVOC(&Sample) ;           /* remove from memory */
    LoadVOCFile("Test.Voc", &Sample) ;  /* load sample */
    gotoxy(1, 2) ;
    printf("Playing ...") ;
    Speaker(SpOn) ;                 /* turn on connection */
    PlayBlock(Sample) ;             /* play back sample */
    do {                            /* wait until the sample is played */
    } while (!(StatusWord == 0)) ;
    DriverOff() ;
    printf("\n") ;
  }
  else
    printf("Error code : %d\n", SBIOResult) ;
}
```

程序清单9.9　扩充的CT-VOICE驱动程序的Pascal测试程序

```
Program CTProTest ;

{ Test program for the extended CT-VOICE driver }

Uses CTVoice, CRT ;
Var
  Ch        : Char ;      { key }
  Vol       : Byte ;      { volume }
  Sample    : VocTp ;     { sample }

Begin
  ClrScr ;
  LoadCTDriver(' ') ;                   { load driver }
  WriteLn(Hi(CTVersion), '.', Lo(CTVersion)) ;
  UsePort($220) ;                       { set base port }
  UseIRQ(7) ;                           { set IRQ number }
  UseChannel(1) ;                       { set DMA channel }
  InitializeDriver ;
  If SBIOResult = CallOk Then Begin
    Input(CD_ROM) ;                     { Line sample }
```

```
Volume(CD_ROMVl, Both, 15) ;        { set volume }
Volume(MainVl, Both, 15) ;
Filter(InputFilter, On) ;
Speaker(SpOff) ;                    { turn off connection }
RecordMode(Stereo) ;
RecordSample(64000, 20000, Sample) ;
GotoXY(1, 1) ; Write('Recording ...') ;
Repeat                              { Wait until the }
Until StatusWord=0 ;                { sample is recorded }
Volume(CD_ROMVl, Both, 0) ;{ set volume }
Volume(VoiceVl, Both, 15) ;
FirstSubBlock(Sample) ;
If Sample.Block^[0]=8 Then
  NextSubBlock(Sample) ;
NextSubBlock(Sample) ;
Sample.Block^[0] :=EndBl ;      { add end block }
Inc(Sample.Length) ;            { incl. end block }
SaveVOCFile('Test.Voc', Sample) ;     { write }
DisposeVoc(Sample) ;                  { remove from memory }
LoadVOCFile('Test.Voc', Sample) ;     { load sample }
GotoXY(1, 2) ; Write('Playing ...') ;
Speaker(SpOn) ;                 { turn on connection }
Ch :=#0 ;                       { for safety reasons }
Vol :=15 ;                      { start volume }
Repeat
  If StatusWord=0 Then PlayBlock(Sample) ;
  GotoXY(1, 3) ;                { Print volume }
  Write('Volume = ', ReturnVolume(MainVl, Left), ' ') ;
  If KeyPressed Then Begin      { key pressed }
    Ch :=ReadKey ;
    If Ch=#0 Then Ch :=ReadKey ;
    Case Ch of
      #72 : If Vol<15 Then Begin  { cursor up }
                Inc(Vol) ;
                Volume(MainVl, Both, Vol) ;
            End ;
      #80 ; If Vol>0 Then Begin   { cursor down }
                Dec(Vol) ;
                Volume(MainVl, Both, Vol) ;
            End ;
      #27 : { Dummy} ;              { escape }
    Else    Begin                   { pause }
                Pause ;
                Ch :=ReadKey ;      { wait for key }
                ContinuePlaying ;
            End
    End { Case }
  End ; { If }
```

```
    Until Ch=#27 ;                    { escape stops }
    Volume(CD_ROMVl, Both, 15) ;
    DriverOff ;
    WriteLn ;
  End
  Else
    WriteLn('Error code : ', SBIOResult) ;
End.
```

程序清单9.10　扩充的CT-VOICE驱动程序的C测试程序

```
/* Test program for the extended CT-VOICE driver */

#include <conio.h>
#include "ctvoice.h"
char Ch = 0 ;              /* key */
char Vol = 15 ;            /* volume */
VocTp Sample ;             /* sample */

void main()
{
  clrscr() ;
  LoadCTDriver("") ;                    /* load driver */
  printf("%d.%d\n", CTVersion() > 8, CTVersion() & Oxff) ;
  UsePort(Ox240) ;                      /* set base port */
  UseIRQ(10) ;                          /* set IRQ number */
  UseChannel(0) ;                       /* set DMA channel */
  InitializeDriver() ;
  if (SBIOResult == CallOk) {
    Input(CD_ROM) ;                     /* Line sample */
    Volume(CD_ROMVl, Both, 15) ;        /* set volume */
    Volume(MainVl, Both, 15) ;          /* listen in */
    Filter(InputFilter, On) ;
    Speaker(SpOff) ;                    /* turn off connection */
    RecordMode(Stereo) ;                /* record stereo */
    RecordSample(40000, 20000, &Sample) ;
    gotoxy(1, 1) ; Printf("Recording ...") ;
    do {                    /* Wait until the sample is recorded */
    }  while (!(StatusWord == 0)) ;
    Volume(CD_ROMVl, Both, 0) ;         /* close line */
    Volume(VoiceVl, Both, 15) ;
    FirstSubBlock(&Sample) ;
    if (*Sample.Block[0] == ExtraInfoBl)    /* skip extra */
      NextSubBlock(&Sample) ;               /* block */
    NextSubBlock(&Sample) ;
    *Sample.Block[0] = EndBl ;      /* add end block */
    ++Sample.Length ;               /* incl. end block */
    SaveVOCFile("Test. Voc", Sample) ;    /* write */
```

```
    DisposeVOC(&Sample) ;          /* remove from memory */
    LoadVOCFile("Test. Voc", &Sample) ; /* load sample */
    gotoxy(1, 2) ; printf("Playing ...") ;
    Speaker(SpOn) ;                /* turn on connection */
    do {
      if (StatusWord == 0) PlayBlock(Sample) ;
      gotoxy(1, 3) ;               /* print volume */
      printf("Volume = %d ", ReturnVolume(MainVl, Left)) ;
      if (kbhit() ) {              /* key pressed */
        Ch = getch() ;
        if ( Ch == '\x00') Ch = getch() ;
        switch (Ch) {

          case 'H'   : if (Vol < 15) { /* cursor up */
                         ++Vol ;
                         Volume(MainVl, Both, Vol) ;
                       }
                       break ;
          case 'P'   : if (Vol > 0) { /* cursor down */
                       --Vol ;
                        Volume(MainVl, Both, Vol) ;
                       }
                       break ;
          case '\x1B': break ;       /* escape */
          default    :  Pause() ;    /* other key */
                        Ch = getch() ;
                        ContinuePlaying() ;
        }
      }
    }  while (!(Ch == '\x1B')) ;   /* escape stops */
    Volume(CD_ROMVl, Both, 15) ;
    DriverOff() ;
    printf ("\n") ;
  }
  else
    printf("Error code : %d\n", SBIOResult) ;
}
```

编程DSP

本节描述怎样能够不用CT-VOICE和DMA，而用编程DSP（数据声音处理器）来录音和播放声音。但是，在详细描述这个题目以前，你应该了解为什么直接编程DSP是经常有用的。

简单地说，直接编程DSP的优点是可以同时播放声音效果和音乐，还可以把一些声音效果事先加到样本里去（虽然不是所有的效果都能适用，包括在播放一记录时设置音量）。

预先编辑样本的一个缺点是如果你想保持原始数据，就必须在编辑它之前把样本数据复制下来。通常，不能取消已做在一个样本中的操作，因此，预先编辑要求大量的存贮器空间

和编辑时间，这用于一些大的样本可能引起一些问题。

直接播放的主要缺点是它需大量的处理器时间，并将别的程序减速得很慢。在你决定是用直接播放样本还是通过DMA播放时，应该作好前后的考虑。

I/O端口

DSP用了四个I/O端口，位7是最高位。以下是I/O端口地址表，其中X表示可选择的基本端口。

I/O端口	用　途
2X6H	复位端口，该端口仅用作写入。
2XAH	数据输入端口，是只读端口。
2XCH	命令/数据端口（写入）和缓冲器状态端口（读出）。能够把数据或命令　写入该端口，也能通过这个端口读出缓冲器的状态。就缓冲器的状态而论，若位7被复位为0，能写数据或者命令到这个端口。
2XEH	指示是否有数据送出，再次用到位7，如数据被送出，则位7被置位为1。

复位端口被用作初始化DSP。和端口22AH配合，可以检查其端口是否正确端口。对所有可能的端口进行选择，将可以决定那个端口是正确的（参见程序9.11和9.12清单）。

复位DSP

为了复位DSP，需作以下的事情：

1. 写一个1到2X6H端口。
2. 等待3微秒时间。
3. 写入0到2X6H端口。
4. 读端口2XEH并检验位7是否被复位（=0），重复这种检查若干次（例如100次），若这位还是处于复位（=0），则复位DSP失败了。看下面的步骤7，以了解此种失败可能的原因。在DSP已经失败时，就不必执行步骤5和步骤6了。
5. 从端口2XAH读出一个字节
6. 读出的字节必须是AAH。如果不是的话，就必须重复第4步和第5步做一定的次数（例如说10次）。假若这个字节是AAH，表示复位已经成功，DSP可以被使用了。
   如果这个字节还不等于AAH，则复位DSP失败;看步骤7查找可能出现的原因。
7. 如果复位失败，大概有两个原因：一个是基本端口不对，或者是已经出现了一个错误。可以再试作一下复位或者是选其它的基本端口。

通常，复位DSP要花大约100微秒时间。

程序清单9.11　用作自动检测Sound Blaster端口的Pascal程序清单

```
Program AutoDetect ;
```

```
Const
  NumberOfTimes1 = 10 ;          { reset }
  NumberOfTimes2 = 50 ;          { data present test }

Var
  Port       : Word ;        { current port }
  Found      : Boolean ;     { port is SB port }
  Counter1   : Word ;        { number of times resetting }
  Counter2   : Word ;        { number of times reading }
Begin
  Port :=$210 ;              { start at port 210H }
  Found :=False ;            { nothing found yet }
  Counter1 :=NumberOfTimes1 ; { initialize counter }
  While (Port<=$260) And Not Found Do Begin
    Port[Port+$6] :=1 ;      { reset Sound Blaster }
    Port[Port+$6] :=0 ;
    Counter2 :=NumberOfTimes2 ; { data test }
    While (Counter2>0) And (Port[Port+$E] < 128 ) Do
      Dec(Counter2) ;
    If (Counter2=0) Or (Port[Port+$A]<>$AA) Then Begin
      Dec(Counter1) ;              { port 2XAH <> AAH }
      If Counter1=0 Then Begin
        Counter1 :=NumberOfTimes1 ; { resetting failed }
        Port :=Port+$10            { try next port }
      End { If }
    End { If }
    Else
      Found :=True           { resetting successful }
  End ; { While }
  If Found Then              { show result }
    WriteLn('Port found at ', Port)
  Else
    WriteLn('No port found !')
End. { program }
```

程序清单9.12　用于Sound Blaster端口自动检测的C程序清单

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

#define NumberOfTimes1 10 /* resetting */
#define NumberOfTimes2 50  /* data present test */

unsigned Port = Ox210 ;    /* current port */
char Found = 0 ;           /* port is SB port */
unsigned Cnt1, Cnt2 ;      /* reset & wait */

void main()
```

```
{
  Cnt1 = NumberOfTimes1 ;          /* initialize counter */
  while ((Port <= Ox260) && !Found) {
    outportb(Port + Ox6, 1) ;      /* reset Sound Blaster */
    outportb(Port + Ox6, 0) ;
    Cnt2 = NumberOfTimes2 ;        /* data test */
    while ((Cnt2 > 0) && (inportb(Port + OxE) < 128))
    --Cnt2 ;
    if((Cnt2==0) ||(inportb(Port + OxA) != OxAA)) {
    --Cnt1 ;                       /* port 2XAH <> AAH */
      if (Cnt1 == 0) {
        Cnt1 = NumberOfTimes1 ;    /* resetting failed */
        Port = Port + Ox10 ;       /* try next */
      }
    }
    else                           /* resetting successful */
      Found = 1 ;                  /* port found */
  }
  if (Found)                       /* show result */
    printf("Base port found at %d\n", Port) ;
  else
    printf("No base port found !\n") ;
}
```

当DSP被复位时，你可以记录或播放声音，为此，首先给DSP发一个指令，然后读或写一个字节。

记录声音

为了录音，需要：

1. 向命令端口2XCH送一个命令20H。
2. 如果你用的是一种快速语言（如像汇编）或者一台很快速的计算机，你应该检查数据有效端口2XEH的位7的值，看看DSP是否能回送一个字节，若位7的值为1，那么DSP能回送一个字节。
3. 从数据输入端口2XAH读出一个字节。
4. 在此等待以一个恒定的采样率来记录样本（这在后面说明），直到记录末尾，即缓冲区的末尾达到为止，返回到步骤1。

如果你采样是用的CT-VOICE驱动程序，则你可以用上述方法进行“消声测试”。为此，要从输入（例如麦克风）读取一字节。假如这个字节的值是在128+A与128-A之间，那么就表示通过麦克风输入没有声音录入;如果其字节的值不是在这个范围以内，则麦克风接收了声音，并能够用CT-VOICE驱动程序来记录一个样本。由于输入容易受到干扰，而使噪声决不会是准确的128（128表示完全无声），所以应给出一个无声范围，故常数A是必要的。以下的程序清单9.13和9.14是无声测试的实例。

程序清单9.13： 无声测试的Pascal例

```
Const
  Threshold = 10 ;              { silence area }

Procedure WaitForSound(Port : Word) ;

Var
  H : Byte ;                    {sample  byte }

Begin
  Repeat
    Port[Port+$C] :=$20 ;       { read byte }
    Repeat                      { data present ? }
    Until Port[Port+$E]>=128 ;
    H :=Port[Port+$A] ;         { yes, read these data }
  Until (H<128-Threshold) OR (H>128+Threshold) ;
End ;                           { H is not in silence area }
```

程序清单9.14： 无声测试的C例

```
#define Threshold 10            /* Limit : 128-Thr. en 128+Thr. */

void WaitForSound(unsigned Port)
{
  /* Wait until input is no longer between 128-Threshold
     and 128+Threshold. */
   unsigned short H ;           /* sample byte */

   do {
     outportb(Port + OxC, Ox20) ;          /* read byte */
     do {                                  /* data present ? */
     } while (!(inportb(Port + OxE) >= 128)) ;
     H = inportb(Port+ OxA) ;              /* yes, read these data */
   } while (!((H < 128 - Threshold) |
            (H > 128 + Threshold))) ;
}              /* H is not in silence area */
```

用DSP放音

为了使用DSP放音，需要：

1. 送入一个命令10H到命令端口2XCH。
2. 如果用的是一种快速语言（像汇编）或一台快速的计算机工作，那应该检查数据状态端口2XCH的第7位，看DSP是否可以接收一个字节。如果位7的值为0，则DSP就能够接收一个字节（被复位）。
3. 送数据字节到数据端口2XCH。

4. 等待、确信样本在以恒定的速率被播放，直到到达样本结束，返回到步骤1。记录和采样两者皆以一恒定的速度出现，这是通过在一些特有的瞬时和相等的时间间隔记录和播放所完成的。为了实现一恒定速度，需要用到定时器。尤其在一些高级语言里，作为记录和放送的代码可能有相当的差异，所以这些代码的宽度也是变化的。如果使用等待循环来代替定时器，那播放能够比记录要快很多或慢很多。

当然，可以通过实验使记录和播放的速率相同。但是，如果在一些机器里，特定代码部分被处理得比较快，那记录和播放的速度就可能不一样。此外，在记录期间，可能产生一个中断，它肯定要占据一定的时间。例如，考虑使用定时器的硬盘停车程序，这个中断可能出现在播放不同的瞬间，也完全可能不出现。在这种情况下，记录和重现将是不同的。

下一节将讨论一个可供选择采用的驱动程序，它使用定时器来记录和播放声音。

CT-TIMER，另一个CT-VOICE驱动程序

CT-TIMER，这个可供选用的驱动程序，与常规的CT-VOICE驱动程序兼容。此外，它拥有许多附加功能。用它可以以别的来替代一个驱动程序且还使用VOC格式。为了避免混淆，我们把第二个驱动程序称为CT-TIMER。

我们选用汇编语言来写这个驱动程序，因这种语言汇编时能得到紧凑和快速的代码。速度是非常希望的东西，由此可加快定时器的中断。

假定采样速率是10，000HZ，以这种速率，定时器中断每秒被调用10，000次。如果定时中断调用的Pascal或者C的程序，将花费较多的时间来处理其程序代码。假使这种调用仅为一次左右，那也倒无所谓，但要每秒10，000次，你会立即注意到程序将放慢下来。即使选用快速的汇编代码并设法优化它，你将注意到，计算机工作总是相当地慢。不幸的是，这是不可避免的。但是，较低的采样率可确保程序有较多的时间可用，计算机工作比它在较高采样率的情况下要快些。

虽然驱动程序把定时器速度增高了，但它仍以每秒18.2次的速率调用其老的定时器程序，所以，所有与时间敏感的功能都工作得很正常。如果在播放的时候定时器受你本身的程序控制，那么这个新的程序将以现行采样率的速度被调用，所以你最好避免此种情况。

CT-TIMER的功能

我们已为这个驱动程序规定了三个新的功能，它们从编号128开始。

功能128，建立缓冲区

```
Procedure VolumeTable（V：Pointer;C：Byte）
Procedure EchoBuffer（E：Pointer;C：Word）
Void VolumeTable（Void *V，unsigned char C）
Void EchoBuffer（Void *E，unsigned C）

输入：BX      = 128
      AX      = 0，音量表
```

= 1，回声缓冲器
CX = C
ES：DI = V或E
输出： 无

功能128是建立两个要用到的缓冲区，下一节将要说明这些缓冲器的使用。当设置音量表时，由变量C规定该音量可被调节的级数。变量E和V应当指向一个所分配的存贮器块。对于音量表，存贮器所要求的容量是256*（C+1），音量可设置为0至C。通常，回声缓冲区的大小等于C。驱动程序自己创建在所分配存贮器中的这张音量表。所以，通过指针V所指示的该存贮器不可以修改。

如果你想利用其回声效果和音量效果，可以在初始化以后调用功能128。

功能129，处理命令串

Procedure Command（S：String）
Void Command（Char *S）

输入： BX = 129
ES：DI = 段：串的偏移量
输出： 无

功能129允许送一“命令串”到驱动程序，其用法也将在下一节介绍。在汇编里，串必须是以一个零字节为终止;在Pascal和C中这个是自动完成的。

功能130，询问现有音量

Function CVolume：Byte
int CVolume（）

输入： BX = 130
输出： AX = 现行音量设置

对大多数效果，音量要被修改。使用功能130，能够读取驱动程序中的音量设置。

创建声音效果

通过把各种各样的操作（大多数是数字运算）施加到样本数据，将可以得到各种各样的声音效果，本章将就其中的一部分进行讨论。并非所有的效果全都符合物理和数学的原理，但是，用这种方式所实现的这些效果是简单而短小的。

因为大多数的操作具有数学的特性，所以用－128到+127之间的值来代替0到255。这意味着首先值128从样本字节减去，接着执行运算，再把128和样本值相加，像这样：

S = F（S-128）+128

在这一式子中使用了下面的变量：

S　　老的样本值（字节）
F（X）　“数学的”运算

音量效果

作为音量，是样本值按一定比例的增加或减少，像下式：

F（S） =S*V/VO

式中，变量V代表现行音量值，VO表示最大音量值。当V小于VO时，样本有一个较低的值，如果V大于VO，样本则被放大。这对于非常柔和的那些样本是会有用的。用这样的音量效果，可以建立声音的淡入和淡出。

- 淡入，样本从音量值为0开始，接着音量以一固有速率增加直至达到某个定值。
- 淡出，效果也是以同样的方式产生，但这次，音量以一固定的速率减小直至0，声音慢慢地逐渐消失。淡出效果是人所共知的，经常用在音乐、歌曲的结束。

CT-TIMER使用一张音量表，这个音量表计算出了所有的音量值供某个音量的选择.之所以要造表，是因为在这张表上查寻音量要比做除法、乘法省去较多时间。用功能128，你可以指定一个空的存贮器块的起点，把这些信息保存在该块里。

混合样本

混合两个样本是十分容易的。将各个样本值加起来，然后用所加的样本数去除这个加值即得。作为混合声音0和1的公式如下：

F（S0，S1） = （S0+S1）/2

当样本数为2的方次时，可以使用快速移位程序。这时你不需要执行除法，但一定要考虑其最大和最小值。如果这两个值中的一个被重写了，那么样本值一定等于这个值。

创建回声

回声是通常听到的声音效果，通过改变一些参数就能得到各种各样的回声效果，从长的回声到自动操纵的失真。计算回声效果要花费大量的时间，但是你可为回声设置两个参数：

- 再听到声音之前的时间。
- 平静下来的速率。

举例，假设在时间I样本值P被重现，经X个字节后，这个样本值同新的样本值一起　又被重现，但是这时样本值P不是那么响亮。

在CT-TIMER中，回声效果是借助缓冲器实现的，在此缓冲器内，其播放的值每次被存

起来。下次这个值能按照一个百分比增加，并且与该新值相混合，这个结果代换原有老的样本值。

因为这种方法是使用整数计算，有时，一个样本值不能达到精确的值128而且引入了噪声。所以当你预定用Pascal和C创建回声样本时，最好采用浮点数。另一种可能性是经过几步以后取出原始样本值，这个样本值已经按一定的百分比减小，然后用现行样本值减去它。

为了创建一个“自动操纵”回声效果，应取很小的步距;而对于响亮强烈的回声效果，则应取较大的步距。

接通和断开声音效果

现在剩下的问题是：怎样在CT-TIMER驱动程序中接通和断开声音效果?可以通过使用一个新的块类型来完成这任务。但是，由于别的VOC编辑器几乎不支持这种类型，所以利用新的块类型不是一种最佳的解决办法。为了调节效果，应该使用一个现成的块类型，可以用标志块或者文本块。对CT-TIMER来说，用文本块是最好的办法，因为文本块不影响其它的程序，而标志块则要影响。在文本中，命令能被处理，如果第一个字符是一个#号，CT-TIMER则检验下面的字符是否组成一个命令。若它们是，则调用相对应的程序，这些命令也能通过使用功能129被执行。

命令可以用大写字母、小写字母或者大小写字母的结合来写，每个命令必须跟一个或几个十进制数，如像：#Volume 123。下面我们看一些命令：

#Volume V 命令#Volume V是设置音量为V值，如果V大于所允许的最大音量值时，它被设置为最大值。这一功能自动接通音量效果。

#Volume Off #Volume Off是关闭音量效果的命令

#Fade s e p 用#Fade s e p命令，可以设置淡化：

- s 表示开始音量
- e 表示结束音量
- p 表示一定的字节数，它代表了每次音量被修改前所需处理的字节。

例如，可用#Fade O x 200设置一个淡入，用#Fade x 0 200设置一个淡出，这里x是最大音量值。这个功能还能接通音量效果。

#FadeOff #FadeOff命令用以停止淡化和关闭其声音效果，但音量效果不改变。

#Echo s p 在#Echo s p命令中：

- s 是在值被重复前以字节表示的步数。这个值必须是小于或等于回声缓冲器的大小！
- p 表示减小的百分数，其值等于p/256。

因为使用了缓冲器，所以当另一个样本或者无声块被播放时，回声将继续。

也可以用下一个命令或者靠再调用#Echo s p来制止这个命令。在开始这命令时，缓冲器应被清除。

#EchoOff #EchoOff命令停止回声效果。

#EffectOff #EffectOff命令将关闭所有的声音效果。

当无效果被应用时，播放只花费比较少的时间。

这些命令可以在能加入和调整文本的VOC编辑器（像VEDIT）中被编程。程序清单9.15和9.16示出了使用这些附加功能的实例。

CT-TIMER使用了两个分别放在文件CONVTABL.INC和WAITTABL.INC中的表，这些文件通过GW-BASIC程序生成该程序，在清单9.17和9.18中可找到。没有这些文件，在汇编时出现一个“fatal”错误！

程序清单9.15： CT-TIMER功能的Pascal程序例

```
Program CTTimerTest ;

{ Test program for the CT-TIMER driver. }

Uses CTVoice, CRT ;

Var
  Ch          : Char ;              { key }
  Vol         : Byte ;              { volume }
  Sample      : VocTp ;             { sample }
  EBuffer,
  VBuffer     : Pounter ;           { echo + volume buffer }

{ Creates the function St, that has a number as
  input and converts it to a string. }
Function St(N : Word) : String ;
Var
  S : String ;
Begin
  Str(N, S) ; St :=S ;
End ;

Begin
  ClrScr ;
  LoadCTDriver('CT-TIMER.DRV') ;        { load driver }
  UsePort($220) ;                       { set port }
  InitializeDriver ;
  If SBIOResult = Callok Then Begin
    GetMem(VBuffer, 129*256) ;          { reserve memory }
    VolumeTable(VBuffer, 128) ;         { for the buffers }
    GetMem(EBuffer, 4096) ;
    EchoBuffer(EBuffer, 4096) ;
    Speaker(SPOff) ;                    { turn off connection }
    RecordSample(60000, 8000, Sample) ;
    GotoXY(1, 1) ; Write('Recording ...') ;
    Repeat                              { Wait until the }
    Until StatusWord=0 ;                { sample is recorded }
    GotoXY(1, 2) ; Write('Playing ...') ;
    Speaker(SpOn) ;                     { turn on connection }
    Ch :=#0 ;                   { for safety reasons }
    Vol :=128 ;                         { start volume }
    Command('Fade 0 128 100') ;
    Command('Echo 2000 180') ;
```

```
    Repeat
      If StatusWord=0 Then                { play back sample }
        PlayBlock(Sample) ;               { again }
      GotoXY(1, 3) ;                      { print volume }
      Write('Volume = ' , CVolume, ' ') ;
      If KeyPressed Then Bagin            { key pressed }
      Case Ch of
        #72 : if Vol<128 Then Bagin       { cursor up }
                Inc(Vol) ;                { increase volume }
                Command('Volume '+St(Vol)) ;
              End ;
        #80 : if Vol>0 Then Begin         { cursor down }
                Dec(Vol) ;                { decrease volume }
                Commando ('Volume "+St(Vol)) ;
              End ;
        #27 : { Dummy } ;                 { escape }
      Else    Begin          { other key pauses }
                Pause ;
                Repeat Until KeyPressed ;
                Ch :=ReadKey ;            { wait for key }
                ContinuePlaying ;
              End
      End {Case}
    End; {if}
Until Ch=#27 ;
    Command('FADE 128 0 100') ;
    Repeat                              { Wait until }
      GotoXy(1, 3) ;                    { volume is o. }
      Write('Volume = ', CVolume, ' ') ;
      If StatusWord=0 Then PlayBlock(Sample) ;
    Until CVolume=0 ;
    DriverOff ;
    WriteLn ;
  End
  Else
    WriteLn('Error code : ', SBIOResult) ;
End.
```

程序清单9.16： CT-TIMER功能的C程序例

```
/* Test program for the CT-TIMER driver.
#include <conio.h>
#include "ctvoice.h"

char Ch = 0 ;                 /* key */
unsigned char Vol = 32 ;      /* volume */
VocTp Sample ;                /* sample */
void *EBuffer ;               /* echo buffer */
void *VBuffer ;               /* volume buffer */
char S[30] ;
```

```
void main()
{
  clrscr() ;
  LoadCTDriver("CT-TIMER.DRV") ;        /* load driver */
  UsePort(Ox240) ;                      /* set port */
  InitializeDriver() ;
  if (SBIOResult == Callok) {
    VBuffer = (void *) malloc(33*256) ;  /* reserve */
    VolumeTable(VBuffer, 32) ;
    EBuffer = (void *) malloc (4096) ;
    EchoBuffer(EBuffer, 4096) ;
    Speaker(SpOff) ;                    /* turn off connection */
    RecordSample(32000, 8000, &Sample) ;
    gotoXY(1, 1) ; printf("Recording ...") ;
    do {      /* Wait until the sample is recorded */
    } while (!(StatusWord == 0)) ;
    printf("Playing ...") ;
    Speaker(SpOn) ;                     /* turn on connection */
    Command("Fade 0 32 400:) ;          /* fade in sample */
    Command("Echo 100 180");            /* use echo */
    do {
      if (StatusWord == 0)              /* play back sample */
        PlayBlock(Sample) ;             /* again */
      gotoxy(1, 3) ;                    /* print volume */
      printf("Volume = %d", CVolume()) ;
      if (kbhit()) {                    /* key pressed */
        Ch = getch() ;
        if (Ch == '\x00' ) Ch = getch() ;
        switch (Ch) {
            case 'H' : if (Vol < 32) {    /* cursor up */
                       ++Vol ;            /* increase volume */
                       sprintf(S, "Volume %d", Vol) ;
                       Commando(S) ;
                     }
                     break ;
            case 'P' : if (Vol > 0) {     /* cursor down */
                       --Vol ;            /* decrease volume */
                         sprintf(S, "Volume %d", Vol) ;
                         Command(S) ;
                     }
                     break ;
          case '\x1B' : break ;

          default     : Pause() ;         /* other key */
                        Ch = getch() ;    /* pauses */
                        ContinuePlaying() ;
        }
      }
```

```
    } while (!(Ch == '\x1B')) ;
    Command("FADE 32 0 400") ; /* fade out sample */
    do {                       /* Wait until volume is 0 */
      gotoxy(1, 3) ;
      printf("Volume = %d ", CVolume()) ;
      if (StatusWord == 0) PlayBlock(Sample) ;
    } while (!(CVolume() == 0)) ;
    DriverOff() ;
    printf("\n") ;
  }
  else
    printf("Error code : %d\n", SBIOResult) ;
}
```

程序清单9.17： CT-TIMER的汇编源码

```
                .MODEL TINY
                .CODE
                ORG Oh

; File name : CT-TIMER.ASM
; CT-TIMER.DRV, a CT-VOICE compatible driver,
; that, however, uses the timer instead of the
; DMA. This way all kinds of effects are possible.
;
; RUN : TASM CT-TIMER.ASM
        TLINK /T CT-TIMER.OBJ, CT-TIMER.DRV
;
;  Possible situations.
Nothing      = 0
Playing      = 1
Recording    = 2
Pausing      = 3
AnotherBlock = 4

; Block types
EndBl        = 0
NewSampleBl  = 1
SampleBl     = 2
SilenceBl    = 3
MarkerBl     = 4
TextBl       = 5
RepeatStartBl = 6
RepeatEndBl  = 7

Start : jmp COntrol

IDText       DB "CT-VOICE"
;-------------------------------------------------------------------------------
```

```
; Driver variables
;-------------------------------------------------------------------------------

                ORG 30h             ; for compatibility
SBIOPort        DW 220h             ; base port
SBIRQNumber     DB 7                ; not used
InitiationFlag  DB 0                ; driver initialized ?
StatusPointer   DD 0                ; address (Status word)
OldTimerCounter DW 0                ; for the correct functioning
OldTimerICounter DW 0               ; of the old timer
OldTimerPointer DD 0                ; routine
SilenceFlag     DB 0                ; silence of sample
BlockStart      DD 0                ; pointer at sub-block
RepeatStart     DD 0                ; to repeat
RepeatEnd       DD 0                ; one or more sub-blocks
RepeatCounter   DW 0
RepeatFlag      DB 0
PlayFlag        DB 0                ; busy playing ?
PlayBackup      DB 0                ; copy of PlayFlag
SamplePointer   DD 0                ; for the playing and
SampleLength    DD 0                ; recording routines
EffctsFlag      DB 0                ; effects of application
ActionFlag      DB Nothing          ; what is the current action
ActionBackup    DB 0                ; copy of ActionFlag
UserRoutePtr    DD 0                ; user routine
UserRouteFlag   DW 0                ; user routine ?
CommandString   DD 0                ; address(effect string)
VolumeTable     DD 0                ; address(Volume Table)
CurrentVolume   DB 0                ; goes without saying
MaximumVolume   DB 0                ; idem
EchoMulti       DB 0                ; settings for the
EchoBuffer      DD 0                ; echo effect
EchoEnd         DW 0                ; end of echo
BufferStart     DW 0                ; buffer start
BufferEnd       DW 0                ; actual buffer end
MultiTable      DB 256 dup (0)      ; help table for echo
FadeCounter     DW 0                ; settings for the
FadeInit        DW 0                ; fade effect
FadeVolume      DB 0
FadeAdd         DB 0

; The functions of the driver, effects, sub-blocks
FunctionTable DW DepermineVersion, SetIOBase
              DW SetIRQ, InitDriver, SetSpeaker
              DW SetStatusAddr, PlayVOCBlock
              DW RecordVOC, StopVOCProcess
              DW DeInitDriver, Pause, Continue
              DW QuitRepeat, UserRoute
ExtraFunctions DW SetBuffer, ProcessCommandString
```

```
                DW CurrentVolumeQ
EffectFunctions DW Volume                ; last applied
                DW Fade
                DW Echo
                DW 4 Dup (NoEffect)
                DW NoEffect              ; first applied
BlockBoutines   DW REndBl, RNewSampleBl, RSampleBl
                DW RSilenceBl, RMarkerBl, RTextBl
                DW RRepeatStartBl, RRepeatEndBl

; Effect strings :
; DB comparison string, DW function.
CommandStrings      DB "ECHO", 0
                    DW EchoEffect
                    DB "FADE", 0
                    DW FadeEffect
                    DB "VOLUME", 0
                    DW VolumeEffect
                    DB "ECHOOFF", 0
                    DW EchoEffectOff
                    DB "FADEOFF", 0
                    DW FadeEffectOff
                    DB "VOLUMEOFF", 0
                    DW VolumeEffectOff
                    DB "EFFECTOFF", 0
                    DW EffectsOff
                    DB 0

; The two tables calculated by CALCTABL.BAS
SRConvTable •       LABEL WORD
INCLUDR CONVTABL.INC
WaitTable           LABEL WORD
INCLUDE WAITABL.INC

;-------------------------------------------------------------------------
; Varioues effects
;-------------------------------------------------------------------------
; At the call of each effect the sample value is in
; AL. After the call AL has to contain the
; processed value.
; The values of ES & DX are already saved, the
; other registers have to be saved.
; When DS is called, it has the same value as CS.

; Volume : process current volume, use
; previously calculated table.
Volume PROC NEAR
        push bx                  ; save BX
        les bx, [VolumeTable]    ; ES : BX = @VolmeTable
```

```
        add bh, [CurrentVolume]
        sub al, 128
        xlat es : [bx]              ; replace AL
        add al, 128
        pop bx                      ; restore BX
        ret
ENDP

; Fade effect : both the fade-in and the fade-out
; effect.
Fade PROC NEAR
                dec [FadeCounter]           ; volume change ?
                jnz EndFade
                mov dx, [FadeInit]          ; yes, restore counter
                mov [FadeCounter], dx
                mov dl, [CurrentVolume]     ; edit volume
                add dl, [FadeAdd]
                mov [CurrentVolume] , dl
                cmp dl, [FadeVolume]        ; correct value reached
                jne EndFade
                and [EffectsFlag], not 2    ; yes, turn off effect
EndFade :
                ret
ENDP

; Echo effect, mix the samples but do not divide by two.
Echo PROC NEAR
                push bx             ; save BX
                sub al, 128
                cbw                 ; 8 bits -> 16 bits
                mov dx, ax          ; dx = 16-bit value
                les bx, EchoBuffer  ; ES : BX = @EchoBuffer
                mov al, es : [bx]   ; AL = previous value
                cbw                 ; 8 bits -> 16 bits
                add ax, 128
                add ax, dx
                or ah, ah           ; 0 <= AX <= 255 ?
                jz NoOverflow
                mov al, ah          ; no, AL = 0 or 255
                cbw                 ; determine on the basis of
                not ah              ; sign
                mov al, ah
NoOverFlow :
                push ax             ; save for later use
                mov dx, bx
                mov bx, offset MultiTable
                xlat
                mov bx, dx
                mov es : [bx], al   ; store value
```

```
                inc bx                  ; next byte
                cmp bx, [EchoEnd]       ; end ?
                jb EchoOk
                mov bx, [BufferStart]   ; yes, go to beginning
EchoOk :
                mov [Word ptr EchoBuffer], bx
                pop ax                  ; restore value
                pop bx                  ; and BX
                ret
ENDP

; No effect, dummy routine.
NoEffect PROC NEAR
                ret
ENDP

;-------------------------------------------------------------------------
; Various routines to set effects
;-------------------------------------------------------------------------
; These routines are called by RTextBl.
; The routines can use the routine
; ReturnParameter that returns the value of the parameter
; in AX. CF contains the Error Flag.

; Set volume, #VOLUME value.
VolumeEffect PROC NEAR
                call ReturnParameter        ; get value
                jc VolumeEError
                cmp al, [MaximumVolume]     ; bigger than maximum
                jb NoVolumeMax
                mov al, [MaximumVolume]     ; yes, then becomes maximum
NoVolumeMax :
                cmp [WORD PTR VolumeTable+2], 0
                je VolumeEError             ; volume table ?
                mov [CurrentVolume], al     ; yes, change volume
                or [EffectsFlag], 1         ; and turn on effect
VolumeEError :
                ret
ENDP

; Set fade, #FADE start, end, rate.
; Depending on start and end a fade-in
; or a fade-out effect is created.
FadeEffect PROC NEAR
                call ReturnParameter        ; get start
                jc FadeEError
                push ax
                call ReturnParameter        ; get end
                pop bx
```

```
                jc FadeEError
                push bx
                push ax
                call ReturnParameter        ; get rate
                pop cx
                pop bx
                jc FadeEError
                mov [FadeCounter], ax       ; set whatever is necessary
                mov [FadeInit], ax
                mov [FadeVolume], cl
                mov [CurrentVolume], bl
                mov al, 1
                cmp bl, cl                  ; start < end ?
                jb PositiveFade
                neg al                      ; yes, AL=-AL
PositiveFade :
                mov [FadeAdd], al           ; store 1 or -1
                or [EffectsFlag], 3         ; turn on effect
FadeEError :
                ret
ENDP

; Set echo, #ECHO step size, multiplier.
; Delete buffer and draq up a fade table.
EchoEffect PROC NEAR
                call ReturnParameter        ; get step size
                jc EchoEError
                push bx
                call ReturnParameter        ; multiplier
                pop bx
                jc EchoEError
                cmp [WORD PTR EchoBuffer+2], 0
                je EchoEError               ; buffer set ?
                mov ch, al                  ; ch = multi value
                mov ax, cs                  ; yes, ES=CS
                mov es, ax
                mov di, offset MultiTable
                xor cl, cl                  ; calculate MultiTable
CalculateMult :
                mov al, cl
                sub al, 128                 ; decrease old value
                cbw                         ; xEchoMulti/256
                mov dl, ch
                mov dh, 0
                imul dx
                mov al, ah
                stosb                       ; store value
                inc cl                      ; after 255 comes 0
                jnz CalculateMult
```

```
                mov ax, [BufferStare]
                mov [WORD PTR EchoBuffer], ax
                add bx, ax
                cmp bx, [BufferEnd]          ; buffer overflow ?
                jb BufferSmaller
                mov bx, [BufferEnd]          ; yes
BufferSmaller :
                mov cx, bx                   ; clear buffer
                sbu cx, ax
                mov [EchoEnd], bx
                les di, EchoBuffer
                xor al, al
                rep stosb
                or [EffectsFlag], 4                  ; turn on effect
EchoEError :
                ret
ENDP

; Turn off volume effect.
VolumeEffectOff PROC NEAR
                and [EffectsFlag], not 4
                ret
ENDP
; Turn off fade effect.
FadeEffectOff PROC NEAR
                and [EffectsFlag], not 4
                ret
ENDP

; Turn off fade effect.
FadeEffectOff PROC NEAR
                and [EffectsFlag], not 4
                ret
ENDP
; Turn off all effects.
EffectsOff PROC NEAR
                mov [EffectsFlag], 0
                ret
ENDP
;-------------------------------------------------------------------------
; Various routines used by the driver functions
;-------------------------------------------------------------------------

; Write a command, try this 200H times.
TestWriteSV PROC NEAR
                push cx              ; save used CX
                mov cx, 200h         ; try 200H times
                mov ah, al           ; AH=copy command
                mov dx, [SBIOPort]   ; DX=2XO
```

```
                add dx, OCh             ; DX=2XC
TestWriteWait :
                in al, dx
                or al, al               ; test bit ?
                jns TestWriteOk         ; =0 ?
                loop TestWriteWait      ; no, once again
                stc                     ; failed
                pop cx
                ret
TestWriteOk :
                mov al, ah              ; successful
                out dx, al              ; write command
                clc
                pop cx
                ret
ENDP

; Read data. Try this 200H times.
TestReadSB PROC NEAR
                push cx                 ; save used CX
                mov dx, [SBIOPort]
                add dl, OEh             ; DX=2XEH
                mov cx, 200h
TestReadWait :
                in al, dx
                or al, al               ; test bit 7
                js TestReadOk           ; =1 ?
                loop TestReadWait       ; no, once again
                stc                     ; failed
                pop cx
                ret
TestReadOk :
                sub dl, 04h             ; DX=2XAH
                in al, dx               ; read data
                clc                     ; successful
                pop cx
                ret
ENDP

; Write a command or data to the SB, wait
; if the SB can receive no command.
WriteSB PROC NEAR
                mov dx, [SBIOPort]      ; dx=2XCH
                add dx, OCh
                mov ah, al              ; AH=copy AL
WriteWait :
                in al, dx
                or al, al
                js WriteWait            ; bit 7=1 ?
```

```
                mov al, ah              ; no, write data
                out dx, al              ; or command
                ret
ENDP

; Read SB data, check whether data is present.
; Wait as long as this is not the case.
ReadSB PROC NEAR
                mov dx, [SBIOPort]      ; DX=2XEH
                add dl, OEh
                sub al, al
ReadWait :
                in al, dx
                or al, al
                jns ReadWait            ; bit 7=0 ?
                sub dl, 4               ; no, read data from
                in al, dx               ; port 2XAH
                ret
ENDP

; Reset SB with the established IO port.
ResetSoundBl PROC NEAR
                mov dx, [SBIOPort]
                add dl, 6               ; DX=2X6H
                mov al, 1
                out dx, al              ; write 1
                in al, dx               ; wait a moment
                in al, dx
                in al, dx
                in al, dx
                sub al, al
                out dx, al              ; write 0
                mov bl, 10h             ; try 10H times
ResetTestLoop :
                call TestReadSB
                cmp al, OAAh            ; AL=AAH ??
                je ResetTestOk
                dec bl                  ; no, once again
                jnz ResetTestLoop
                stc                     ; reset failed
                ret
ResetTestOk :
                clc                     ; reset successful
                ret
ENDP

; Set timer interrupt.
SetTimerInt PROC NEAR
                pushf                   ; save flags
```

```
                push es
                cli                     ; interrupts off
                mov dx, 0               ; set interrupt
                mov es, dx
                mov dx, cs
                xchg ax, [WORD PTR es : 8*4+0]
                xchg dx, [WORD PTR es : 8*4+2]
                mov [WORD PTR OldTimerPointer+0], ax
                mov [WORD PTR OldTimerPointer+2], dx
                mov [OldTimerICounter], 1   ; call the timer
                mov [OldTimerCounter], 1    ; each time
                pop es
                popf                        ; restore I-flag
                ret
ENDP
; Restore timer interrupt and reset timer
; rate at 18.2 times a second.
RestoreTimerInt PROC NEAR
                pushf                       ; save flags
                push es
                cli                         ; interrupts off
                mov al, 36h                 ; reset timer
                out 43h, al                 ; so it is called 18.2
                xor al, al                  ; times a second
                out 40h, al
                xor al, al
                out 40h, al
                xor ax, ax                  ; restore interrupt
                mov es, ax                  ; vector
                mov ax, [WORD PTR OldTimerPointer+0]
                mov dx, [WORD PTR OldTimerPointer+2]
                mov [WORD PTR es : 8*4+0], ax
                mov [WORD PTR es : 8*4+2], dx
                pop es
                popf                        ; restore I-flag
                ret
ENDP

; Set status word at a certain value.
SetStatusWord PROC NEAR
                push ds
                push si
                lds si, [StatusPointer]
                mov [si], ax                ; set value
                pop si
                pop ds
                ret
ENDP
```

```
; Minimize segment : offset so offset is
; between 0 and 15.
Minimize PROC NEAR
            push ax                 ; save
            shr ax, 1               ; AX=AX/2
            shr ax, 1               ; idem
            shr ax, 1               ; idem
            shr ax, 1               ; AX=offset/16
            add dx, ax              ; increase segment
            pop ax                  ; restore offset AX
            and ax, 15              ; AX between 0 and 15
            ret
ENDP
; Convert a segment : offset to a 20-bit address.
Calculate20Bits PROC NEAR
            push bx                 ; save BX
            xor bx, bx
            shl dx, 1               ; DX=DX*2
            rcl bl, 1               ; BL=bit of DX
            shl  dx, 1
            rch bl, 1
            shl dx, 1
            rcl bl, 1
            shl dx, 1               ; DX=DX*2*2*2*2=DX*16
            rcl bl, 1               ; BL=DX/4096
            add ax, dx              ; AX=offest+segment*16
            adc bl, 0               ; process overflow
            mov dx, bx              ; DX=bits 20 .. 16
            pop bx
          ret
ENDP
; Go to next block.
NextBlock PROC NEAR
            les di, [BlockStart]    ; ES : DI=current start
            mov ax, [es : di+1]     ; BL, AH, AL = 20 bits
            mov bl, [es : di+3]     ; size
            xor bh, bh
            add di, 4
            add ax, di              ; add offset DI to
            adc bx, 0               ; BL, AH, AL
            and bx, Ofh
            mov dx, es              ; calculate new ES
            ror bx, 1
            ror bx, 1
            ror bx, 1
            ror bx, 1               ; BX=BX*16*16*16
            add dx, bx              ; DX=new segment
            call Minimize
            mov [WORD PTR BlockStart+0], ax
```

```
                mov [WORD PTR BlockStart+2], dx
                ret
ENDP

; Skip a number of blocks until the indicated
; block is found.
FindBlock PROC NEAR
                les di, [BlockStart]          ; ES : DI=current start
                cmp al, [es : di]             ; compare block types
                je BlockFound                 ; equal ?
                push ax                       ; no, save block type
                call NextBlock
                pop ax                        ; restore block type
                jmp FindBlock                 ; start again
BlockFound :
                ret                           ; block type found
ENDP

; Convert ASCII character to uppercase.
UpCase PROC NEAR
                cmp al, 'a'
                jb NoLetter
                cmp al, 'z'
                ja NoLetter
                sub al, 'a'-'A'
NoLetter :
                ret
ENDP

; Determine the next number in a command string.
ReturnParameter PROC NEAR
                les di, [CommandString]       ; ES : SI
                cmp BYTE PTR [es : di], 0     ; zero character ?
                je ReturnParameterError
                xor bh, bh
SkipRest :
                mov bl, [es : di]             ; BL = character
                cmp bl, 0                     ; end string ?
                je ReturnParameterError
                inc di                        ; no, increase DI
                cmp bl, '9'                   ; is BL a number ?
                ja SkipRest
                cmp bl, '0'
                jb SkipRest
                mov ax, bx                    ; yes, AX=BX-'0'
                sbu ax, '0'
DetermineNumber :
                mov bl, [es : di]             ; BL = character
                cmp bl, '9'                   ; BL again a number ?
```

```
                ja EndNumber
                cmp bl, '0'
                jb EndNumber
                sub bx, '0'                ; yes, AX=AX*10+BX-'0'
                mov cx, 10
                mul cx
                add ax, bx
                inc di                     ; next character
                jmp DetermineNumber
EndNumber :
                mov [WORD PTR CommandString], di ; store current
                clc                        ; pointer
                ret
ReturnParameterError :
                mov [WORD PTR CommandString], di
                stc
                ret
ENDP

; Verify whether a string contains a command. If necessary
; call the routine that corresponds to command.
CommandTest PROC NEAR
                les di, [CommandString]    ; ES : DI = start text
                mov si, offest CommandStrings
                xor bx, bx                 ; start at DI+0
CompareCommand :
                lodsb                      ; character from table
                cmp al, 0                  ; end of string ?
                je EndCommand
                call Upcase                ; no
                mov ah, al                 ; AH = Upcase(AL)
                mov al, [es : di+bx]       ; character from string
                call Upcase
                inc bx                     ; next character
                cmp al, ah                 ; AL= AH ?
                je CompareCommand
                xor ah, ah                 ; no, skip rest of the
SkipRestCom :                              ; command string
                lodsb
                or al, al                  ; AL=0 ?
                jnz SkipRestCom            ; yes -> Z-flag = 1
SkipCall :
                add si, 2                  ; skip offset call
                xor bx, bx                 ; start of text
                cmp [si], bl               ; end of table ?
                jne CompareCommand
                ret                        ; yes, no command !
EndCommand :
                mov al, [es : di+bx]       ; next character
```

```
                call UpCase
                cmp al, 'A'                     ; is this a letter ?
                jb ProcessCommand
                cmp al, 'Z'
                jb SkipCall
ProcessCommand :
                add di, bx                      ; no, text the same
                mov WORD PTR [CommandString+0], di
                call WORD PTR [si]
                ret
ENDP

; Process the recorded sample. Bring
; VOC block header up to date.
ProcessRecording PROC NEAR
                mov ax, [WORD PTR SamplePointer+0]
                mov dx, [WORD PTR SamplePointer+2]
                call Calculate20Bits
                mov bx, ax
                mov cx, dx
                mov ax, [WORD PTR BlockStart+0]
                mov dx, [WORD PTR BlockStart+2]
                call Calculate20Bits
                sub bx, ax                      ; calculate sample
                sbb cx, dx                      ; size
                sub bx, 4                       ; skip header and
                sbb cx, 0                       ; the last byte
                les di, [BlockStart]            ; stort length in
                mov [es : di+1], bx             ; VOC header
                mov [es : di+3], cl
                les di, [SamplePointer]         ; put an EndBlock type
                mov BYTE PYR [es : di], 0       ; at the end
                ret
ENDP

; ---------------------------------------------------------------------------
; VOC sub-block routines
; ---------------------------------------------------------------------------

; Process a VOC sub-block.
ProcessVOCBlock PROC NEAR
                cmp [UserRouteFlag], 0          ; user routine ?
                je ProcessBlockType
                les bx, [BlockStart]            ; yes, call this routine
                call dword ptr [UserRoutePtr]
                jnc ProcessBlockType            ; process this block ?
                call NextBlock                  ; no, next block
                jmp ProcessVOCBlock             ; start again
```

```
        ProcessBlockType :
                les di, [BlockStart]            ; ES : DI=start block
                mov bl, [es : di]               ; BL=block type
                cmp bl, 7                       ; known type ?
                jbe ProcessThisBlock
                call NextBlock                  ; no, next block
                jmp ProcessVOCBlock             ; start again
ProcessThisBlock :
                shl bl, 1                       ; call corresponding
                xor bh, bh                      ; routine
                call [bx+offset BlockRoutines]
                call NextBlock                  ; for next time
                cmp [ActionFlag], AnotherBlock
                je ProcessVOCBlock              ; or this time ?
                ret                             ; no, ready
ENDP

; The end block, stop playing.
REndBl PROC NEAR
                mov [PlayFlag], 0               ; playing stopped
                call RestoreTimerInt            ; restore timer
                mov ax, 0
                call SetStatusWord              ; StatusWord=0
                mov [ActionFlag], Nothing       ; doing nothing
                ret
ENDP

; New sample + settings.
RNewSampleBl PROC NEAR
                mov [SilenceFlag], 0        ; no silence
                mov [PlayFlag], 0           ; stop playing
                mov bl, [es : di+4]         ; set the right values
                xor bh, bh                  ; based on SR
                shl bx, 1
                mov ax, [bx+WaitTable]
                mov [OldTimerICounter], ax
                mov [OldTimerCounter], ax
                mov ax, di                  ; sample starts at
                add ax, 6                   ; off set 6
                mov [WORD PTR SamplePointer+0], ax
                mov [WORD PTR SamplePointer+2], es
                mov ax, [es : di+1]         ; copy length
                mov dl, [es : di+3]
                xor dh, dh                  ; after 2 bytes
                sub ax, 2                   ; are skipped
                sbb dx, 0
                mov [WORD PTR SampleLength+0], ax
                mov [WORD PTR SampleLength+2], dx
                mov al, OB6h                ; setsampling rate
```

```
                out 43h, al                     ; at timer
                mov ax, [bx+SRConvTable]
                out 40h, al
                mov al, ah
                out 40h, al
                mov [PlayFlag], 1               ; start playing
                mov [ActionFlag], Playing
                ret
ENDP

; New sample.
RSampleBl PROC NEAR
                mov [SilenceFlag], 0            ; no silence
                mov [PlayFlag], 0               ; stop playing
                mov ax, di                      ; set start
                add ax, 6                       ; of sample
                mov [WORD PTR SamplePointer+0], ax
                mov [WORD PTR SamplePointer+2], es
                mov ax, [es : di+1]             ; and copy its length
                mov [WORD PTR SampleLength+0], ax
                mov al, [es : di+3]
                xor ah, ah
                mov [WORD PTR SampleLength+2], ax
                mov [ActionFlag], Playing
                mov [PlayFlag], 1               ; start playing
                ret
ENDP

; Silence block.
RSilenceBl PROC NEAR
                mov [PlayFlag], 0               ; stop playing
                mov bl, [es : di+6]             ; set values based
                xor bh, bh                      ; on SR
                shl, bx, 1
                mov ax, [bx+WaitTable]
                mov [OldTimerICounter], ax
                mov [OldTimerCounter], ax
                mov ax, [es : di+4]             ; copy length
                mov [WORD PTR SampleLength+0], ax
                xor ax, ax
                mov [WORD PTR SampleLength+2], ax
                mov al, OB6h                    ; set sampling rate
                out 43h, al                     ; at the timer
                mov ax, [bx+SRConvTable]
                out 40h, al
                mov al, ah
                out 40h, al
                mov [SilenceFlag], 1            ; play abs. silence
                mov [PlayFlag], 1               ; start playing
                mov [ActionFlag], Playing
                ret
ENDP
```

```
; Set status word with new value.
RMarkerBl PROC NEAR
            mov ax, [es : di+4]          ; get value and
            call SetStatusWord           ; set it
            mov [ActionFlag], AnotherBlock
            ret
ENDP

; Process a text block. If the first character os #,
; check whether it is a command.
RTextBl PROC NEAR
            add di, 4
            cmp BYTE PTR [es : di], '#' ; character = # ?
            jne NoCommand
            inc di                       ; yes, store start
            mov [WORD PTR CommandString+0], di
            mov [WORD PTR CommandString+2], es
            call CommandTest             ; process command
NoCommand :
            mov [ActionFlag], AnotherBlock
            ret
ENDP

; Start the repetition. Set all right values.
RRepeatStartBl PROC NEAR
            mov [WORD PTR RepeatStart+0], di
            mov [WORD PTR RepeatStart+2], es
            mov ax, [es : di+4]
            inc ax
            mov [RepeatCounter], ax
            mov [RepeatFlag], 1
            mov [ActionFlag], AnotherBlock
            ret
ENDP

; End of the repetition, start again if necessary.
RRepeatEndBl PROC NEAR
            xor ax, ax                   ; AX=0
            cmp [RepeatFlag], 1          ; repetition busy ?
            jne EndRepetition
            mov ax, [RepeatCounter]      ; yes, AX=number of times
            sub ax, 1                    ; to come
            jc Infinite                  ; AX was 0 ?
            jz EndRepetition             ; no, ax is 0 ?
            mov [RepeatCounter], ax      ; no, store AX
Infinite :
            mov ax, [WORD PTR RepeatStart+0]      ; copy
            mov [WORD PTR BlockStart+0], ax       ; start
            mov ax, [WORD PTR RepeatStart+2]      ; repetition
```

```
                mov [WORD PTR BlockStart+2], ax
                mov [ActionFlag], AnotherBlock
                ret
EndRepetition :
                mov [RepeatCounter], ax              ; no further repetition
                mov [RepeatFlag], 0
                mov [Actionflag], AnotherBlock
                ret
ENDP

; The interrupt handler for playing.
PlayingHandler PROC FAR
                push ax                      ; save AX
                cmp [cs : PlayFlag], 1       ; play back sample ?
                je YesPlaying
PlOldTimer :
                dec [cs : OldTimerCounter]           ; no
                jnz NoOldPlCall                      ; to old timer ?
                mov ax, [cs : OldTimerICunter]       ; yes, restore
                mov [cs : OldTimerCounter], ax       ; counter
                pop ax                               ; restore AX
                jmp DWORD PTR [cs : OldTimerPointer]
NoOldPlCall :
                mov al, 20h                  ; pass on that
                out 20h, al                  ; the interrupt is
                pop ax                       ; received
                iret
YesPlaying :
                push dx                      ; play back a sample
                push ds
                mov ax, cs                   ; DS=CS
                mov ds, ax
                mov al, 10h                  ; command 10H
                call WriteSB
                cld                          ; decrease length
                sub [WORD PTR SampleLength+0], 1
                jnz NoPlEnd
                sub [WORD PTR SampleLength+2], 1
                jns NoPlEnd
                mov [PlayFlag], 0            ; end of sample
                mov al, 20h                  ; timer has to be
                out 20h, al                  ; processed each time
                sti                          ; turn on interrupts
                push di                      ; save registers
                push es
                push bx
                push cx
                push si
                mov al, 128                  ; AL=absolute silence
```

```
                call WriteSB
                call ProcessVOCBlock
                pop si                  ; restore registers
                pop cx
                pop bx
                pop es
                pop di
                pop ds
                pop dx
                pop ax
                iret
NoPlEnd :
                push es
                push di
                mov al, 128             ; AL=absolute silence
                cmp [SilenceFlag], 1    ; play silence ?
                je ProcessEffects
PlayNoSilence :
                les di, [SamplePointer]     ; no
                mov al, [es : di]           ; AL=sample value
                add [WORD PTR SamplePointer+0], 1
                jnc ProcessEffects
                add [WORD PTR SamplePointer+2], 1000h
ProcessEffects :
                cmp [EffectsFlag], 0        ; effect present ?
                jne YesEffects
                jmp NoEffects               ; no, skip it
YesEffects :
                mov dh, [EffectsFlag]       ; DI=effects flag
                mov di, dx
                xx=7*2                      ; TASM variable xx
                REPT 8                      ; repeat block 16 times
                LOCAL NoEditing
                rol di, 1                   ; apply effect ?
                jnc NoEditing
                call [EffectFunctions+xx]   ; yes, call it
NoEditing :
                xx=xx-2
                ENDM                        ; end repeat block
NoEffects :
                pop di                      ; play back value
                pop es
                call WriteSB
                pop ds
                pop dx
                jmp PlOldTimer
ENDP

; The interrupt handler for recording.
```

```
RecordingHandler PROC FAR
                push ax                         ; save AX
                cmp [cs : PlayFlag], 1          ; to old timer ?
                je YesRecording
RecOldTimer :
                dec [cs : OldTimerCounter]      ; mo
                jnz NoOldRecCall                ; to old timer ?
                mov ax, [cs : OldTimerICounter]     ; yes, restore
                mov [cs : OldTimerCounter], ax      ; counter
                pop ax                              ; restore AX
                jmp DWORD PTR [ cs : OldTimerPounter ]
NoOldRecCall :
                mov al, 20h                     ; pass on that
                out 20h, al                     ; the interrupt is
                pop ax                          ; received
                iret
YesRecording :
                push dx                         ; record a sample
                push ds
                mov ax, cs                      ; DS=CS
                mov ds, ax
                mov al, 20h                     ; command 20H
                call WriteSB
                cld                             ; decrease length
                sub [WORD PTR SampleLength+0], 1
                jnz NoRecEnd
                sub [WORD PTR SampleLength+2], 1
                jns NoRecEnd
                mov [PlayFlag], 0               ; end of sample
                mov al, 20h                     ; time has to be
                out 20h, al                     ; processed each time
                sti                             ; turn on interrupts
                push di                         ; save registers
                push es
                push bx
                push cx
                push si
                call ReadSB
                call ProcessRecording
                call REndBl                     ; restore driver
                call REndBl                     ; restore registers
                pop si
                pop cx
                pop bx
                pop es
                pop di
                pop ds
                pop dx
                pop ax
```

```
                iret
NoRecEnd :
                push es
                push di
                les di, [SamplePointer]         ; no
                call ReadSb
                mov [es : di], al               ; AL=sample value
                add [WORD PTR SamplePointer+0], 1
                jnc NoSegmentOverFlow
                add [WORD PTR SamplePointer+2], 1000h
NoSegmentOverFlow :
                pop di                  ; play value
                pop es
                pop ds
                pop dx
                jmp RecOldTumer
ENDP

; ------------------------------------------------------------------------------
; The driver functions
; ------------------------------------------------------------------------------

; Return driver version. This driver
; corresponds with version 1.10
DetermineVersion PROC NEAR
                mov ax, 10Ah                    ; AX=version 1.10
                ret
ENDP

; Set base port, first test for
; right port selection.
SetIOBase PROC NEAR
                test ax, OFh            ; test first 4 bits
                jnz NoRightPort         ; all 0 ?
                cmp ax, 210h            ; yes, AX >= 210H ?
                jb NoRightPort
                cmp ax, 260h            ; yes, AX <= 260H ?
                ja NoRightPort
                mov [SBIOPort], ax      ; yes, new port
NoRightPort :
                ret
ENDP

; Set right IRQ; Since no use is made of
; the DMA, this is a dummy function.
SetIRQ PROC NEAR
                ret
ENDP
```

```
; Initialize driver. Test whether the base port
; is correct.
InitDriver PROC NEAR
            call ResetSoundBl
            mov ax, 2           ; wrong port
            jc NoGoodReset
            dec al              ; turn on speaker
            call SetSpeaker
            mov [InitiationFlag], 1; initiation OK.
            mov [ActionFlag], Nothing
            xor ax, ax
NoGoodReset :
            ret
ENDP
; Set speaker position.
SetSpeaker PROC NEAR
            mov ah, OD1h            ; AH=speaker on
            or al, al
            jnz SpeakerControl
            mov ah, OD3h            ; AH=speaker off
SpeakerControl :
            mov al, ah              ; set speaker
            call WriteSB
            ret
ENDP

; Set status word address.
SetStatusAddr PROC NEAR
            mov [WORD PTR StatusPointer+0], di
            mov [WORD PTR StatusPointer+2], es
            ret
ENDP

; Play VOC block.
PlayVocBlock PROC NEAR
            cmp [ActionFlag], Nothing   ; not, busy ?
            je PlayBlock
            mov ax, 1                   ; no, busy !
            ret
PlayBlock :
            mov [OldTimerCounter], 1    ; call old timer
            mov ax, di                  ; optimize
            mov dx, es                  ; sample start
            call Minimize
            mov [WORD PTR BlockStart+0], ax
            mov [WORD PTR BlockStart+2], dx
            mov [PlayFlag], 0           ; play nothing
            mov ax, offset PlayingHandler
            call SetTimerInt            ; set timer-int.
```

```
                mov ax, OFFFFh
                call SetStatusWord          ; StatusWord=OFFFFH
                mov [ActionFlag], Playing
                call ProcessVOCBlock        ; start at first block
                xor ax, ax
                ret
ENDP

; Record a VOC block.
RecordVOC PROC NEAR
                sub cx, 6
                sbb dx, 0
                jnc EnoughMemory
                mov ax, 2
                stc
                ret
EnoughMemory :
                mov [WORD PTR SampleLength+0], cx
                mov [WORD PTR SampleLength+2], dx
                mov cx, ax
                mov dx, 000Fh
                mov ax, 4240h
                div cx
                neg ax                      ; SR=256-1000000/rate
                push ax                     ; save for later use
                mov ax, di                  ; Optimize ES : DI
                mov dx, es
                call Minimize
                mov di, ax
                mov es, dx
                pop bx                      ; BX=SR
                mov BYTE PTR [es : di], 1   ; block type 1
                mov [es : di+4], bl         ; SR
                mov BYTE PTR [es:di+5], 0   ; normal 8-bit sample
                mov [WORD PTR BlockStart+0], di
                mov [WORD PTR BlockStart+2], es
                add di, 6                   ; start of sample data
                mov [WORD PTR SamplePointer+0], di
                mov [WORD PTR SamplePointer+2], es
                xor bh, bh                  ; set pointer
                shl bx, 1
                push bx
                mov ax offset RecordingHandler
                call SetTimerInt
                pop bx
                mov ax, [bx+WaitTable]      ; for the old timer
                mov [OldTimerICounter], ax  ; call
                mov [OldTimerCounter], ax
                mov al, OB6h                ; set sampling rate
```

```
                out 43h, al                     ; at timer
                mov ax, [bx+SRConvTable]
                out 40h, al
                mov al, ah
                out 40h, al
                mov [PlayFlag], 1               ; start recording
                mov ax, OFFFFh
                call SetStatusWord              ; StatisWord=OFFFFH
                mov [ActionFlag], Recording
                ret
ENDP

        ; Stop recording or playing. Restore timer and if
; necessary process block length when recording.
StopVOCProcess PROC NEAR
                cmp [ActionFlag], Playing ; busy ?
                je StopPlaying
                cmp [ActionFlag], Pausing
                je StopPlaying
                cmp [ActionFlag], Recording
                je StopRecording
                mov ax, 1                       ; no !
                ret
StopPlaying :
                mov [PlayFlag], 0               ; stop playing
                call RestoreTimerInt            ; restore timer
                mov ax, 0
                call SetStatusWord              ; delete StatusWord
                mov [ActionFlag], Nothing
                xor ax, ax
                ret
StopRecording :
                mov [PlayFlag], 0
                call RestoreTimerInt
                call ProcessRecording
                mov ax, 0
                call SetStatusWord
                mov [ActionFlag], Nothing
                xor ax, ax
                ret
ENDP

; Turn off driver. Stop playing and turn off speaker.
DeInitDriver PROC NEAR
                mov al, 0                       ; turn off speaker
                call SetSpeaker
                mov [InitiationFlag], 0
                jmp StopVOCProcess
ENDP
```

```
; Pause playing.
Pause PROC NEAR
                cmp [ActionFlag], Playing      ; busy playing
                je PauseOk
                mov ax, 1                      ; no !
                ret
PauseOk :
                mov [ActionFlag], Pausing
                mov al, [PlayFlag]             ; store old flag
                mov [PlayFlag], 0              ; stop timer
                mov [PlayBackup], al
                xor ax, ax
                ret
ENDP

; Continue previuusly paused block.
Continue PROC NEAR
                cmp [ActionFlag], Pausing      ; busy pausing ?
                je ContinueOk
                mov ax, 1                      ; no !
                ret
ContinueOk :
                mov [ActionFlag], Playing      ; busy playing
                mov al, [PlayBackup]           ; set play flag
                mov [PlayFlag], al
                xor al, al
                ret
ENDP
; Leave a repeat loop.
QuitRepeat PROC NEAR
                cmp [RepeatFlag], 1            ; busy repeating ?
                je QuitRepeatOk
                mov ax, 1                      ; no !
                ret
QuitRepeatOk :
                mov [RepeatCounter], 1         ; last repetition
                cmp ax, 0                      ; immediately ?
                jne Immediately
                mov ax, 0                      ; no
                ret
Immediately :
                mov al, RepeatEndBl            ; yes, continue immediately
                call FindBlock
                mov [RepeatCounter], 1         ; at block after end
                mov [PlayFlag], 0              ; repeat block
                call ProcessVOCBlock
                ret
ENDP
```

```
; Set user routine.
UserRoute PROC NEAR
            mov [WORD PTR UserRoutePtr+0], ax
            mov [WORD PTR UserRoutePtr+2], dx
            or ax, dx
            mov [UserRouteFlag], ax
            ret
ENDP

; Set buffer for echo effect and volume effect.
SetBuffer PROC NEAR
            cmp ax, 0
            je [WORD PTR EchoBuffer+0]. di
            mov [WORD PTR EchoBuffer+2], es
            mov [BufferStart], di
            add cx, di
            mov [BufferEnd], cx
            mov [EchoEnd], cx
            ret
VolumeBuffer :
            mov [WORD PTR VolumeTable+0], di
            mov [WORD PTR VolumeTable+2], es
            mov [MaximumVolume], cl
            mov dl, cl
            mov cl, 0
CalcNextVolume :
            mov ch, 0

CalcSampleValue :
            mov al, ch
            imul cl
            idiv dl
            stosb
            inc ch
            jnz CalcSampleValue
            inc cl
            cmp cl, dl
            jbe CalcNextVolume
            ret
ENDP

; Set an effect by processing a command string.
ProcessCommandString PROC BEAR
            mov [WORD PTR CommandString+0], di
            mov [WORD PTR CommandString+2], es
            call CommandTest
            ret
ENDP
```

```
; Ask current volume.
CurrentVolumeQ PROC NRAR
                mov al, CurrentVolume
                xor ah, ah
                ret
ENDP

; ---------------------------------------------------------------------
; The interface
; ---------------------------------------------------------------------
; This is where the call prart starts.
control PROC FAR
                push bx                 ; save registers
                push cx
                push dx
                push bp
                push di
                push si
                push ds
                push es
                push cs
                pop ds                  ; DS=CS
                cld
                cmp bx, 3               ; BX= <= 3 ?
                jbe CallAllowed
                cmp [InitiationFlag], 1 ; no
                je CallAllowed          ; driver init. ?
                mov ax, -1              ; no, wrong function
                jmp short NoCall
CallAllowed :
                cmp bx, 13              ; function <= 13 ?
                jbe ProcessFunction
                cmp bx, 128             ; no, function  < 128 ?
                jae ExtraFunction
                mov ax, -2              ; yes, wrong function
                jmp NoCall
ExtraFunction :
                cmp bx, 128+(EffectFunctionsExtraFunctions}/2
                jae NoCall
                sub bx, 128-(ExtraFunctions-FunctionTable)/2
ProcessFunction :
                shl bx, 1               ; call function
                call [bx+OFFSET FunctionTable]
BoCall :
                pop es
                pop ds
                pop si
                pop di
                pop bp
```

```
        pop dx
        pop cx
        pop bx
        retf
ENDP

END Start
```

程序清单9.18　产生用于CT-TIMER中的两个表的ASIC程序

```
100 ' Table creator for CT-TIMER.ASM
110 ' Open used files
120 POEN "convtabl.ine" FOR OUTPUT AS #1
130 OPEN "waitabl.inc" FOR OUTPUT AS #2
140 ' Calculate for every SR the corresponding values
150 FOR SR !=0! TO 255 !
160 ' Calculate the actual sampling rate
170 SAMPLINGRATE ! = 1000000 !/(256!-SR!)
180 ' Calculate the value for the timer
190 TIMERRATE = INT(1193280!/SAMPLINGRATE!)
200 ' Calculate the number of times to wait
210 WAITRATE = INT (SAMPLINGRATE !/18.2)
220 ' Write the values to the files
230 IF SR! MOD 6 = 0 THEN GOSUB 300 ELSE GOSUB 340
240 NEXT
250 ' Cloes files
260 CLOSE #1
270 CLOSE #2
280 END
290 ' Start at a new line
300 PRINT #1, : PRINT #1, CHR$(9); "dw"; TIMERRATE ;
310 PRINT #2, : PRINT #2, CHR$(9); "dw" ; WAITRTE ;
320 RETURN
330 ' Write the data to the files
340 PRINT #1, "," ; TIMERRATE ;
350 PRINT #2, "," ; WAITRATE ;
360 PETURN
```

第十章　用MIDI编程

本章内容为：

按MIDI协议传输数据的设置和模式
阅读MIDI工具图表
在MIDI中状态字节如何工作
MIDI中的声部、模式、系统和实时信息
MIDI文件格式
MIDI同Sound Blaster如何工作

这一章阐述了如何对MIDI进行编程。也说明了MIDI的各种设置和模式及它们如何发声。本章告诉你如何阅读工具图表。其次，本章还介绍了数据字节和状态字节在MIDI中如何工作，MIDI文件格式，及同Sound Blaster一起工作时MIDI的用法。

注解　第四章说明了什么是MIDI，并详尽地说明连接MIDI以及MIDI设备，MIDI软件。

MIDI概述

MIDI（音乐设备数字接口）协议包含16个通道，每个通道能传送不同的数据信息，也可以说是不同的音符。由于有16个通道，故可允许多到16个设备在MIDI协议下同时工作，一个设备分配一个通道。

许多MIDI设备允许你自行设置MIDI通道。主、从式设备可用同一个MIDI通道。然而，对采用菊花链式连接的八个MIDI设备，在其不同通道上同时演奏不同音符，则需一个音序器。也仅需一个音序器，就可通过16个通道同时传送不同的数据。

传输数据的设置和模式

在MIDI协议中，有三种设置用于确定MIDI设备如何进行数据的传送和接收，它们分别是Omni、Poly、Mono。

Omni开和Omni关　如果MIDI设备只能传送和接收一路数据，这种设置称为Omni关。这种设置主要用在MIDI的配置中有音序器和许多的MIDI设备互连中。

设置是Omni开时，从动设备可同时接收所有的16路数据。如果你想使两个或更多的MIDI设备同时演奏，或者在一个MIDI设备演奏一个完整乐曲，就可采用这种设置。

Mono和Poly　设置成Mono（单声的）时，每个通道只有一种声部，而在Poly（多声的）设置时，每个通道可播放多种声部。Mono主要用于MIDI吉它，每根弦有其自己的MIDI道。对键盘琴和合成器而言，常用Poly方式。

注解　在电子音乐里，术语“声部”指一种乐器所发出的声音。

四种MIDI模式

借助模式选择器，MIDI协议有四种模式供选择，使得在Omni，Poly和Mono设置之间选择更容易。这些模式见表10.1

表10.1 四种MIDI模式

模式	描　　述
模式1	Omni开，Poly。所有16个通道都被接收而且每个通道同时用几种声部演奏。
模式2	Omni开，Mono。所有16个MIDI通道都被接收，但每次只能演奏一种声部。
模式3	Omni关，Poly。在这种模式中，只能接收被选择的一个通道，但所选中的通道可演奏几种声部，这种模式最常用。
模式4	Omni关，Mono。在这种模式中，只有选中的一个通道被接收，而且每个通道上只能演奏一种声部，这种模式主要用于MIDI。

通道和系统信息

通道信息和系统信息是主从设备间可以交换的两类信息：

- 系统信息视MIDI系统为一整体。例如，它们用于同步音序器和合成器。因为每　个连接在MIDI系统上的设备都接收系统信息，所以，系统信息能通过任一MIDI　通道被送出。
- 通道信息与所演奏的音乐有关。通道信息从16个MIDI通道中的一个送出，只能被一个MIDI设备接收到，这个设备是与送出信息的MIDI通道相连接的。

通道模式和声部信息　通道信息有两种类型：模式信息和声部信息。模式信息描述不同的模式（见表10.1），声部信息可分成以下五种：

- 音符信息　音符信息包括关于音符的一些细则，例如，什么时候弹动一个键，什么时候松开它。
- 程序更换　程序更换是选择一个新的声部或设备的方法。例如，如果你送一串音符信息到合成器模块，那么所发出的声部取决于最后的一个程序更换。如最后一个程序更换所设置设备为钢琴，则声音合成钢琴。你可随时在演奏中送程序更换来变更演奏的乐器。
- 控制更换　控制更换信息改变设备发音的方式。程序更换是换接乐器，而控制更换是修改乐器本身的声音。
- 后触效果　为了重新产生一个音符经历保持时的音调改变，几乎所有的合成器都具有后触效果。在你按下并松开一个键后，一个后触信息被送出，赋予音符　以颤音或音量改变。
- 音调转折信息　音调转折器是靠近合成器键盘的一个小操纵杆，借助这个转折器，你改变音符的音调。

注解 只有少数键盘有音调转折器或后触，没有这些功能的键盘所接收的信息中若包括音调转折或后触时，这些信息是被忽略掉。

系统专用信息 系统信息通过每个相连的MIDI设备接收和再传送。例如可让你同步一音序器和一合成器，使演奏同时开始。

系统信息中最常见的一种类型是这个系统的专有信息。这些信息主要针对一些特别的厂家生产的专用MIDI设备。系统专用信息允许你修改在MIDI协议中未专门规定为缺省值的一些参数，但是不同的厂家则有所不同。例如，借助系统专有信息可以改变声音参数而产生一种非规定的声音。

系统专有信息通常还含有一个ID号，它表明了设备的厂家。如果一个从设备接收到含有ID号的系统专有信息，而此信息与它无关，那它忽略不管这些信息。

阅读MIDI的工具图表

每个MIDI设备有其自己的MIDI工具图表，从中可找出一个特定的MIDI设备能做的事情。它还能准确表明MIDI设备能够接收和传输的信息以及所不能接收和传送的信息。MIDI的这些工具图表是标准化的，也就是说，不管这个MIDI设备是哪个厂家生产的，都有同样的行、列项目，仅仅内容不同而已。

图10.1列出了MT-32的工具图表，图10.2列出了DX7，雅玛哈合成器的工具图表。MT-32是日本Roland生产的一种发声模块，因为MT-32是发生模块，所以它没有键盘，因此不能送音符数据，MT-32送出的唯一数据是系统专用信息，为了响应被送到MT-32的系统专有信息，这是必要的。DX7有一个键盘，其工具图表的内容与MT-32的明显不同。

MIDI工具图表有四列，标题是Function（功能），Tranmsitted（传输），Recongnized（识别），Remarks（注释）：

列 项	含 义
Function	表明对联接MIDI设备至为重要的不同部份。
Transmitted	表明MIDI设备可传输的功能。
Recognized	表明MIDI设备可接收的功能。
Remarks	与不同功能有关的备注。

在图表的底部列出了方式1到4的定义及代表是与否的符号，大多数表中，用0表示具有此项功能，而用X表示否定。

表中的*号表明对应的功能不适用于MIDI设备。如果一个功能项是0，它经常伴有一些数字，这些数字表明功能的范围值。

MULTI TIMBRE SOUND MODULE
MODEL MT-32 Version : 1.02

MIDIlmplementation Chart

	功能	传输	识别	注释
基本通道	Default Changed		2-10 1-8, 10	
模式	Default Messages Altered	**********	Mode 3	
音符数	True Voice	* 0-127 **********	0-127 12-108	
速度	Note ON Note OFF	* *	O v=1-127 X	
后触	Key's Ch's	* *	X X	
音调转折器		*	O 0-24 semi	
控制更换	1 7 10 11 12 : 63 64 65 : 120 121	* * * * * * * *	O O O O X O X O	Modulation Part Volume Panpot Expression Hold 1 Reset all controllers
程序更换	True #	*	O 0-127 0-127	
系统专用	O	O		
系统共用	Song Pos Song Sel Tune	X X X	X X X	
系统实时	Clock Commands	X X	X X	
辅助信息	Local ON/OFF All Notes OFF Active Sense Reset	X X X X	X O (123-127) O X	
音符			* in OVERFLOW MODE received message goes thru MIDI OUT.	

Mode 1 : OMNI ON, POLY　　Mode 2 : OMNI ON, MONO
Mode 3 : OMNI OFF, POL　　Mode 4 : OMNI OFF, MONO

图10.1 MT-32的工具图表

Yamaha digital programmable algorithm synthesizer
Model DX7

MIDIlmplementation Chart

	功能	传输	识别	注释
基本通道	Default	1	1	
	Changed	X	1-16	
模式	Default	Mode 3	Mode 3	
	Messages	X	omni on, poly, mono	
	Altered			
音符数		36-96	0-127	
	True Voice			
速度	Note ON	O	O	
	Note OFF	X (9n 0vh)	X	
后触	Key's	X	X	Older Dx7's used
	Ch's	O	O	control 3 for this message
音调转折器		O	O	
控制更换	1	O	O	Mod
	2	O	O	Breath
	4	O	O	Food Control
	6	O	O	Data entry
	64	O	O	Sustain
	65	O	O	Portamento
	96/97	O	O	+/-
程序更换		0-63	0-127	
	True #		1-32	
系统专用	O	O		Data dumps, parameters
系统共用	:Song Pos	X	X	
	:Song Sel	X	X	
	:Tune	X	X	
系统实时	:Clock	X	X	
	:Commands	X	X	
辅助信息	:Local ON/OFF	X	X	
	:All Notes OFF	X	O	
	:Active Sense	O	O	
	Reset	X	X	
音符				

Mode 1 : OMNI ON, POLY　　Mode 2 : OMNI ON, MONO
Mode 3 : OMNI OFF, POL　　Mode 4 : OMNI OFF, MONO
O=Yes　X=No

图10.2　DX7的工具图表

MIDI工具图表中的各项功能

为了更好地理解MIDI工具图表，下面列出了各功能的详细说明：

Basic channel（**基本通道**）

Default	当设备置为开时，被选择的各通道。
Changed	这些通道可通过在合成器上的MIDI通道按钮选择。

Mode（**模式**）

Default	当设备置为开时，这是活动模式。
Messages	能够接收和送出的那些模式信息。
Altered	当模式信息被传输时，模式是否更换。

Note Number（**音符数**）能被送出或接收的音符的总数

TrueVoice	有时演奏太高或太低的音符需升高或降低音阶，行项表明哪些音符可在没被修正时演奏出来。

Velocity（**速度**）

NoteOn	键按下的速度值。
NoteOff	键弹起的速度值。

Aftertouch（**后触**）	怎样控制颤音或音符的音量变化。
Key的	MIDI设备是否能控制多音的后触（每个正被演奏音符的后触）。
Channel的	MIDI设备是否能控制通道的后触。
Pitch Bender（**音调转折器**）	是否能够处理音调转折信息，如果能，音调转折是什么范围值。
Control Change（**控制更换**）	表明MIDI设备能送出或接收哪个控制变化信息。
Program Change（**程序更换**）	表明MIDI设备能送出或接收那一个程序变化信息。
True	表明是否太高的编号引导回到其它的号。
System Exclusive（**系统专有**）	MIDI设备是否能送出或接收系统专用信息。

System Commom（**系统共用**）

Song Position	MIDI设备工作时是否用面向MIDI环境里的MIDI歌曲位置指针，以致一支乐曲的位置能够传送。
Song Select	如果设备用位置指针工作，是否能进行歌曲选择。
Tune	是否可能选择一个伴奏的节奏。

System Real Time（**系统实时**）

Clock	MIDI设备能否控制系统实时时钟信息（如果能够，它就可与另一设备同步）。
Commands	是否系统实时命令同样让MIDI设备知道何时开始和何时停止。

Aux Message（**辅助信息**）

Local ON/OFF	当本地置为通时，所演奏的音符在MIDI设备中也能听到（与本地置为断开时相反）
All Notes OFF	是否所有的音符能同时被关掉。
Active Sense	是否MIDI设备能接收信息或它是否自己送出信息，以检验MIDI设备是否仍在活动。
Reset	MIDI设备能否被复位（指设备的状态为它开始工作时的状态）。

Notes（**音符**）　　通常此列项包括各种规则的补充和例外。

MIDI规范

MIDI使用的缺省电缆是5针的DIN插头，将双绞线电缆的屏蔽层两端头固定接在插头的2脚上，双绞线分别连接到针脚4和针脚5，1脚和3脚不连。建议所用电缆长度最好不超过50英尺（15米）。

MIDI接口是一个异步串行接口，波特率为31.25K，换句话说，即每秒传送31，250位，包括八个数据位，它具有一个起始位和一个停止位。

状态字节和数据字节

被送出的字节可细分为两组：状态字节和数据字节

- 状态字节是实际命令，它们很容易被识别，其值总是在128-255之间（因为状态字节中最高位的值总为1）。
- 数据字节包含相应的信息。因其数据字节的最高位始终为0，所以值在0-127之间。一般很容易看出一个新的命令的开始，只要等到出现其值大于或等于128，那就是新命令开始了。

MIDI协议是基于16个独立的MIDI通道，用状态字节的低四位来选择通道，因为4位已足以表示0-15。

一个状态字节只对一个MIDI通道起作用。如果要对多个通道起作用，就需在对每个通道传送数据字节时传送同样数目的状态字节。但是值为240-255的状态字节适用于所有通道，因此有16种可能的功能。

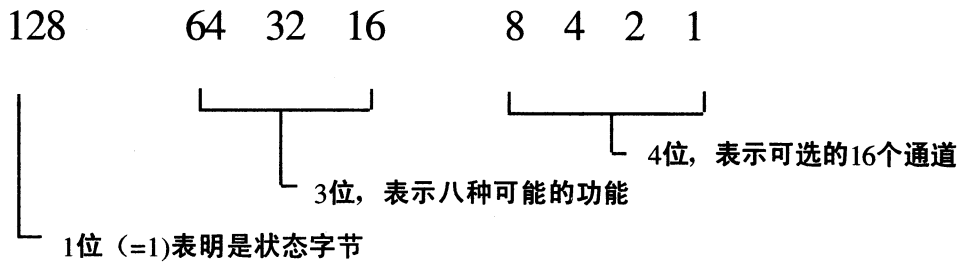


图10.3　一个状态字节的结构

值为128-239的状态字节称为声部信息，因为这些信息只供一种声部用。250-255范围的状态字节是系统信息，因为系统信息与MIDI系统有关，所以每个通道都得送系统信息。

注解　见本章前面的“通道与系统信息”，以便对系统信息有一个了解。

状态字节可分为以下几类：

状态字节	用　途
128-143	表明“Note OFF”数据。也就是说放开按下的音符。状态字节后跟随两个数据字节，第一个表示定调，第二个表示键弹起的速度。
144-159	表明“Note ON”数据。当按下一个音键时就建立此信息，状态字节后跟两个字节，第一个表明音符值，第二个表明击键速度。表10.2示出了音符数字与对应的音阶和音符。
160-175	描述了音键压力，它与多音后触相同，在演奏每个音符时，后触代码就建立了。状态字节后也跟两个字节，第一个表示定调，第二个表明压力值。
176-191	为指明控制器号和值的参数字节。第一个字节表明控制器号，第二个字节表明其值。表10.3列出了控制器号、功能、值和使用。
192-207	表明程序更换。状态字节后跟一个字节，指示一个新的声音的号。
208-223	描述通道压力。与音键压力相比，这时每个通道的后触码被通过。状态字节只附带一个字节，它表明通道压力值。
224-239	包括定调轮数据。定调轮可使你轻易地改变演奏音符的定调。定调轮类似一个小的模拟游戏杆，能检测非常小的变化。这就是增加两个字节构成一个14位数的原因。定调轮信息的结构如下：1110CCCC0LLLLLLL0HHHHHHH，其中C表示通道，L代表低7位值，H代表高7位值。

表10.2　音符号与其音阶数及音符

音阶数	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
0	0	1	2	3	4	5	6	7	8	9	10	11
1	12	13	14	15	16	17	18	19	20	21	22	23
2	24	25	26	27	28	29	30	31	32	33	34	35
3	36	37	38	39	40	41	42	43	44	45	46	47
4	48	49	50	51	52	53	54	55	56	57	58	59
5	60	61	62	63	64	65	66	67	68	69	70	71
6	72	73	74	75	76	77	78	79	80	81	82	83
7	84	85	86	87	88	89	90	91	92	93	94	95
8	96	97	98	99	100	101	102	103	104	105	106	107
9	108	109	110	111	112	113	114	115	116	117	118	119
10	120	121	122	123	124	125	126	127				

声部、模式和系统信息

以下是关于声部、模式和系统信息的描述，并介绍在MIDI中它们如何传送。

声部信息

“运行状态字节”的规则适用于所有的声部信息，换句话说，在每个数据字节后你用不着再传送状态字节，因为当两个或更多的声部信息依次不断地在同一个通道上建立时，状态字节可以忽略。数据字节后紧随的是同种类型的另一组信息的数据字节。因为所有数据字节的值小于128，所以当字节的值大于127则表明是另一种类型的信息。

速度数据字节用其相应的对数刻度表示。MIDI设备不能产生速度，所以总是固定送一个为64的速度值。

表10.3 控制器编号

编号	功能	数值	使用
0	Continuous controller #0	0-127	MSB
1	Modulation Wheel	0-127	MSB
2	Breath Control	0-127	MSB
3	Continuous controller #3	0-127	MSB
4	Foot Controller	0-127	MSB
5	Portamento time	0-127	MSB
6	Data Entry	0-127	MSB
7	Main Volume	0-127	MSB
8	Continuous controller #8	0-127	MSB
9	Continuous controller #9	0-127	MSB
"	"	"	"
31	Continuous controller #31	0-127	MSB
32	Continuous controller #0	0-127	LSB
33	Modulation Wheel	0-127	LSB
34	Breath Control	0-127	LSB
35	Continuous controller #3	0-127	LSB
36	Foot controller	0-127	LSB
37	Portamento time	0-127	LSB
38	Data Entry	0-127	LSB
39	Main Volume	0-127	LSB
40	Continuous controller #8	0-127	LSB
41	Continuous controller #9	0-127	LSB

（续表）

编号	功能	数值	使用
"	"	"	"
63	Continuous controller #31	0-127	LSB
64	Damper pedal on/off(sustain)	0=off	127=on
65	Portamento on/of	0=off	127=on
66	Sostenuto on/off	0=off	127=on
67	Soft pedal on/off	0=off	127=on
68	Undefined on/off	0=off	127=on
69	Hold 2	0=off	127=on
70	Undefined on/off	0=off	127=on
"	"	"	"
80	General purpose controller #5	0-127	MSB
81	General purpose controller #6	0-127	MSB
82	General purpose controller #7	0-127	MSB
83	General purpose controller #8	0-127	MSB
84	Undefined on/off	0=off	127=on
85	Undefined on/off	0=off	127=on
86	General purpose controller #5	0-127	LSB
87	General purpose controller #6	0-127	LSB
88	General purpose controller #7	0-127	LSB
89	General purpose controller #8	0-127	LSB
90	Undefined on/off	0=off	127=on
91	Undefined on/off	0=off	127=on
92	Tremolo Depth	0-127	
93	Chorus Depth	0-127	
94	Detune	0-127	
95	Phaser Depth	0-127	
96	Data Entry + 1	127	
97	Data Entry - 1	127	
98	Nonregistered parameter MSB		
99	Nonregistered parameter LSB		
100	Registered parameter MSB		
101	Registered parameter LSB		

（续表）

编号	功能	数值	使用
102	Undefined	?	
"	"	"	"
121	Undefined	?	
122	Local Control on/off	0=off	127=on
123	All notes off	0	
124	Omni mode off (incl. all notes off)	0	
125	Omni mode on (incl. all notes off)	0	
126	Mono mode on (incl. all notes off)	*	
127	Poly mode on (incl. mono=off and all notes off)	0	

* 此值等于通道号，或当通道号等于接收设备的各通道的号时，此值等于0。

定调轮字节的值是从0-16383，它在静止位置时的值为8192。

模式信息

模式信息的状态字节范围是176-191，第一个数据字节的值在122-127之间。模式信息有下述几种：

- 本地控制（Local Control）第一个数据字节值为122，如果本地控制关，那第二个字节值为0;相反，本地控制开，则第二个字节值为127。当本地控制是开时，所有在键盘上演奏的音符实际上在MIDI设备上演奏。如果本地控制为关，则MIDI设备什么也不能演奏。在这两种情况下，所演奏音符的数据被建立，并送到MIDI的输出端口。
- 全部音符关（All Notes Off）第一个数据字节的值为23，第二个字节值为0。这一功能禁止所有的音符在由状态字节指出的MIDI通道演奏。
- Omni模式关（Omni Mode Off）第一个数据字节值为24，第二个为0。这一功能表明声部信息只为通道接收。
- Omni模式开（Omni Mode On）这一功能表示所有的MIDI通道能接收声部信息。第一个数据字节值为125，第二个为0。
- 单音模式（Monophonic Mode）这里，第一个数据字节值为126，第二个字节指明各单音的通道号。如果第二个字节值为0，那通道号等于正在接收的设备的声部号。这种模式严格地把一个声部分给一个通道。
- 多音模式（Polyphonic Mode）此功能表示每个通道能接收几种声部，第一个字　节值为127，第二个字节为0。

最后四种模式信息的组合结果形成了模式1到4（见表10.1），如果一在接收的MIDI设备不支持某一模式，通常将其转接到模式1。

系统信息

具有状态字节值从240-247的信息是系统信息。如本章前面所述，系统信息是针对设备，而且总是被表示在MIDI工具图表中。系统信息包括以下的一些状态字节：

状态字节	含　义
240	指明是一个系统专有命令。这里厂商可以放置MIDI协议中没有规定的MIDI设备所产生的所有功能。一个系统专用命令包含了不同数量的数据字节，其命令的结尾用状态字节值247（这是一个结束符信息）。
241	备用。
242	决定歌曲位置。这是特别为MIDI音序器预备的。歌曲位置允许音序器在一支乐曲的任一点开始。状态字节跟随的两个字节构成一个14位的数字;低7位先来。歌曲位置采用节拍来表示，一个节拍等于6个MIDI的时钟节拍。
243	执行歌曲选择功能。用后随的一个数据字节来指明所选歌曲的编号。
244	备用。
245	备用。
246	指出一个曲调请求。曲调请求不包含任何数据字节，这个命令让模拟合成器调整它们的振荡器频率。
247	这是系统专用信息末尾（EOX）或结束符命令的状态字节的值，它不含数据字节。该结束符用于表明一个系统专用信息块的结束。

系统专用块　一个系统专用块可能只包含数据字节，在专用块的结尾使用一终止信息是任选的，可以不用。系统专用块的第一个字节一定是MIDI的ID号，ID号是厂商给予MIDI设备的一个特有号，由MIDI协会制定。号码1-31分配给美国的制造厂商，32-63给欧州厂商，而64-95给日本的制造厂商。

迄今为止，就Sound Blaster而言，最重要的厂商是：

1 Sequential Circuits
2 Big Briar
3 Octave/Plateau
4 Moog
5 Passport Designs
32 Bon Tempi
33 S.I.E.L.
34 Ircam
35 SynthAxe
36 Hohner

64 Kawai
65 Roland
66 Korg
66 Yamaha
67 Casio

注解 附录F列出了所有MIDI设备厂商的代码。

实时信息

最后一组信息是实时信息。实时信息的状态字节范围从248-255，实时信息可插在其它信息之间并与它们自身的状态字节有同样尺寸；它们不包含任何数据字节。实时信息同设备有关，所以它们总是在MIDI工具图表中指明。实时信息如下所述：

信　息	解　释
定时时钟	状态字节值为248。在1/4音符中，每24个时钟节拍传送一个定时时钟信号，通常用作同步MIDI设备和鼓设备。
起动	状态字节值250。作为音序器和鼓设备的起动命令。
继续	状态字节值251。它使设备在下一个时钟节拍时继续演奏。
停止	状态字节值252。它使音序器或鼓设备停止演奏。
有效读出	状态字节值254。有效读出字节每隔300毫秒送出一次，它允许接收设备检测任何干扰，使设备转换为另一模式。
系统复位	状态字节值255。当系统复位送出时，每个接收设备回到开机时的状态。此信息看作是一种紧急恢复装置。

研究以下的例子，可了解这些信息是如何工作的。一个键盘通过一个MIDI连接盒与Sound Blaster相连。在键盘上选择MIDI通道4，从工具图表可知，键盘为音符开（Note ON）建立速度64，而对音符关（Note Off）为速度0。你弹奏一个音符，然后你选另一个声音。再弹奏另一个音符，并且这个键按下不放，再弹奏另一个音符，并把刚才按下的那个键放开。这一过程中，Sound Blaster接收到以下字节信息：

147 50 64 50 0 195 4 157 55 64 55 0 59 64 62 64 62 0 59 0

状态字节147表明音符开，选通道4，后面的两个数据字节表明定调和音符速度，以速度64演奏音阶4里的D。下一个字节不是状态字节，这意味另一个音符开信息送到同一个通道中，现在，同一个音符速度为0，表明键被放开。下一个状态字节表明通道4程序更换，选择声音为4，下一个来的是音符开状态字节，表明音阶4里的G，按下这个音符，而后放开两个字节，字节59和64为音阶4里的B，形成音符开信息。在这个音符被按下时，音阶5中的D也被按下（62 64）。

此时唯一的漏失信息是接收信息的时间，这意味着我们不知道不同的音符各演奏多长时间。不过，计算机有解决这一问题的方法，下一节我们将介绍。

MIDI文件格式

正如上面所说那样，不能只存贮MIDI数据。对于一个文本文件只需存构成文本的那些字符和代码。但是，在MIDI格式中，播放MIDI数据的时间是非常重要的，在MIDI信息之前简单地附上时间并将它写到文件里是不够的。

Delta（△）时间 因为状态字节值总是大于或等于80H，而数据字节的值又小于80H，所以很容易检测信息的开始。处理信息的开始时间是十分重要的。MIDI协议规定第一个信息始终在时刻0开始，接下来各信息的时刻是基于在一个信息和前一个信息之间的时间之差来计算的。在协议中，这被称之为△（delta）时间，你需要确保准确地规定这个时间，以使信息的开始能找到。

△时间的最后一个字节的值总是小于128（数据字节，位7被复位0）而前面各字节值总是大于或等于128（状态字节，位7被置位1），这样，你能很容易确定信息的开始位置，然而计算△时间会复杂一些。

存储数据采用的方法称为可变长存储。下表显示了转换成可变长存贮可能产生的某些混乱，左边列举出可变长的值，右边列是正常的等效值。两组数值都是采用16进制的。

变长数	正常等效数
7FH	7FH
81FH	80H
C000H	2000H
7FFFH	3FFFH
8180H	4000H
C08000H	100000H
FFFF7FH	1FFFFFH
81808000H	200000H
C0808000H	8000000H
FFFFFF7FH	FFFFFFFH

MIDI的文件头和音轨块结构

MIDI文件格式包括许多块，这些块为两种类型：文件头块和音轨块。每个文件有一个文件头块和几个音轨块：

- 文件头块包含了MIDI文件所必须的信息。
- 音轨块包括MIDI数据流，至多含有有关16个MIDI通道的信息。

文件头块

文件头块具有以下结构：

MThd<文件头数据长度><文件头数据>

文件头数据可分成以下几部分：

<格式><音轨号><时间分割>

详细说，文件头块可如下所示：

- 字母MThd占四个字节
- 用四个字节表明文件头数据长度
- 用两个字节表示格式
- 用两个字节表示文件里的音轨号
- 用两字节表示时间分割

高位字节先出现。一般说来，文件头数据的长度等于00000006H。

两个格式字节 两个格式字节表明了MIDI文件的文件类型，有三种不同的文件类型：类型0、类型1和类型2：

文件类型	说　明
0	MIDI文件由包含所有16个MIDI通道数据的一个音轨组成
1	MIDI文件由一个或多个同步音轨组成
2	MIDI文件由一个或多个独立的音轨组成

其次的两个字节包含了组成MIDI文件的音轨号，最后的两个字节形成了1/4音符的时间分割，△时间取决于这个时间分割。时间分割1的意思是△时间2等于半个音符的时间。有时这两个字节的值为负，在这种情况下，时间分割是以秒为单位测量。把这个时间转换成更一般的时间测量的标准，SMPTE和MTC格式。最高字节的值可为-24，-25，-29或-30，这些数字来自电视，表示了每秒的帧数。第二个字节指出在一帧里再细分，这一定是个正数，最常用的值是4，8，10，80和100。

音轨块

音轨块被定义成以下格式

MTrK<音轨数据长度><音轨数据>

式中：

<音轨数据> = <MTrK事件>*
（*=可能重复）
<MTrK事件> = <△时间><事件>
<事件> = <MIDI事件>或
<SysEx事件>或
<meta事件>

使用可变长度的方法来存贮△时间。一个MIDI事件包括所有的MIDI通道信息。运行状态的方法在这里也适用，但是，在音轨数据中的第一个事件必须包含一个状态字节。

MIDI文件格式和系统专用信息

系统专用信息在MIDI文件格式里存在一个问题，系统专用信息可以用值F7H表示结束，而大于128的值可能很轻易被看成是可变长度值的第一个字节，为了解决这个问题，建立了系

统专用事件，它有两种形式：

F0H<长度><跟随F0H的各字节>

F7H<长度><需送出的全部字节>

之所以有两种系统专用事件的原因，是因为一些制造厂商传送他们的系统专用信息是以打包方式传送的。

存储长度使用可变长度的方法来存储，并指明其后随的字节数。第一种系统专用事件常用来存储整个系统专用信息，而第二种系统专用事件是处理包的。运行状态可以不用。

系统专用事件在一个F0H事件的末尾或者F7事件的最后一个包的末尾总是用F7H为结束。当主设备不送F7H，程序自己必须加上这个字节。

当包被送出时，第一个包总是和一个F0H事件一起传送，但不是用F7H结束，以下的包都和F7H事件一起送，最后一个包必须用F7H结束。F7H事件还可用来传送实际上并不重要的信息。像实时、歌曲指针、MIDI时间和代码等信息都可以存储在这些信息中，但它们必须被忽略不计，这些F7H事件不一定用F7H作结束。

Meta事件和MIDI文件格式

Meta事件是包括非MIDI信息的事件，这些信息对文件格式有用。Meta事件有以下的语法格式：

FFH<类型><长度><数据字节>

类型指事件号，用一个小于128的字节值表示，以变长度的方法计算即将到来的数据字节的长度。这些数据字节包括应用于Meta事件的数据。如同系统专用事件一样，可以不使用运行状态方法。

信息总需包括状态字节，程序不能识别的Meta事件应该被跳过。下面是对必须识别的Meta事件的描述。

FFH 00H 02H SS SS序列号 当使用这个事件时，必须将它放置在音轨的开始，甚至放在△时间和MIDI事件的前面。序列号分配某个值给音轨，这样，你能够用此提示信息来正走或反走这些序列号数。

Meta事件编号1-0FH安排给文本，那个文本可放置在歌曲中。

FFH 01H 长度 文本事件 文本事件被放在一个音轨的开始，它能包括各种各样的文本。

FFH 02H 长度 文本版权注解 这个事件必须是第一个音轨块时刻0处的第一个事件，应包含字符（C），版权年代和著作者。

FFH 03H 长度 文本 序列/音轨名 当这个事件出现在文件格式0的音轨或文件格式1的第一个音轨时，它的名字指的是序列名。在所有其它情况指的是音轨名。

FFH 04H 长度 文本设备名 这个事件给出了音轨里所用设备的说明。

FFH 05H 长度 文本 词句 你可以用它指明歌曲的词句。通常你分解这个词句，在△时间将它们作为词句事件放置，在这里这些词句必须是要唱的。

FFH 06H 长度 文本 标志 表明在序列里位置的名字，此标志事件在文件格式0或格式1的第一个音轨中出现。

FFH 07H **长度 文本 Cue点** Cue点事件运载适用于特定时刻的文本，例如，一幢正在倒塌的建筑物伴随着音乐和文本“House Knocked Down”。

FFH 2FH 00H **音轨的结尾** 音轨事件的结尾不像所有其它中间事件那样是可选用的，它必须被放在音轨的末尾，以表明其音轨的精确结尾处。

FFH 51H 03H tt tt tt **设置速度** 设置速度事件是以每个MIDI四分之一音符的微秒数计量，该事件导致速度的改变。

FFH 54H 05H hr min sec **帧 分数帧 SMPTE偏移** 这个事件表明一音轨必须开始的SMPTE时间，其格式为SMPTE时间;分数帧表明在一帧里面的百分之一秒的数量。

在文件格式1中，这个事件必须和设置速度事件一起定义，SMPTE事件总是放在一个音轨的开始处、在△时间之前并在任何其它MIDI事件之前。

FFH 58H 04H nn dd cc bb **时间特征** 由时间特征事件设置时间单位，它用在MIDI格式中。式中nn代表时间特征的分子，dd代表时间特征的分母，分母值表示成2的负的方次数，cc表明在一个节拍里MIDI时钟节拍的数量，bb表示在一个MIDI的四分之一音符里能有多少个30秒音符能发生。

举例来说，一个事件为FFH 58H 04H 06H 03H 24H 08H，则表明时间描述成6/8（因为2 =1/8），所以，在一个节拍里有32个MIDI时钟节拍出现，在一个MIDI四分之一音符里有8个30秒音符出现。

FFH 59H 02H sf mi**音调特征** 这一事件规定了降半音、升半音和音调的数字。例如sf=-5表示5个降半音，sf=5意味着升5个半音，sf=-1为一个降半音，sf=0是c调，mi=0是大音阶的音调，mi=1是小音阶的音调。

FFH 7FH **长度数据音序器特有的Meta事件** 你在音序器手册的附录上常可见到meta事件，因为每个厂商对这个事件有他自己的定义，总之，数据值的第一个字节就是厂商的ID号。

现在，我们已经解释了整个MIDI文件格式，恰好可给你一个实际的例子，后面列出了来自流行Wing commander游戏中的一首歌。

MIDI和Sound Blaster的DSP芯片

Sound Blaster的DSP芯片处理MDI数据。这个芯片以64KB的缓冲区存放输入的MIDI数据，送出向外输出的MIDI数据到MIDI输出。

由于MIDI系统的波特率是31，250波特，MIDI数据必须仔细而快速地处理。假定每秒接收大约3906个字节的话，缓冲区在几乎1/50秒内将被装满，旧的MIDI数据被覆盖。

MIDI读出功能具有两种从缓冲区读出MIDI数据的模式：直接模式和中断模式（MIDI写模式只能直接控制）：

- 对于直接模式而言，一个运行中的程序必须通过MIDI端口取回和送出数据，程序需要连续地检查MIDI端口是否准备好，以及MIDI数据是否可用。
- 对中断模式，一旦要接收新的MIDI数据时，将发出一个硬件中断，这个中断中止正在运行的程序，而允许从MIDI端口读取MIDI数据，然后再回到运行的程序。你自己必须写一个中断控制程序，以便当MIDI数据被接收时调用。

Wing Commander游戏中的一首歌

```
4D 54 68 64 00 00 00 06-00 01 00 0E 01 E0 4D 54    MThd.....MT
72 6B 00 00 05 DB 00 FF-03 1F 42 61 72 72 61 63    rk.....Barrac
6B 73 2D 47 6F 20 54 6F-20 53 6C 65 65 70 20 59    ks-Go To Sleep Y
6F 75 20 50 69 6C 6F 74-73 00 FF 54 05 60 00 00    ou Pilots..T.`..
00 00 00 FF 58 04 04 02-18 08 00 FF 51 03 09 62    ....X.....Q..b
58 83 60 FF 51 03 08 E3-64 83 60 FF 51 03 08 95    X.`.Q...d.`.Q...
44 83 60 FF 51 03 08 50-E8 83 60 FF 51 03 08 50    D.`.Q..P..`.Q..P
E8 83 60 FF 51 03 08 A8-CC 83 60 FF 51 03 08 8B    ..`.Q.....`.Q...
80 83 60 FF 51 03 08 95-44 83 60 FF 51 03 08 F6    ..`.Q...D.`.Q...
EC 83 60 FF 51 03 08 B2-90 83 60 FF 51 03 09 3B    ..`.Q.....`.Q..;
48 83 60 FF 51 03 08 B2-90 83 60 FF 51 03 08 9F    H.`.Q.....`.Q...
08 83 60 FF 51 03 08 C6-18 83 60 FF 51 03 09 B0    ..`.Q.....`.Q...
78 83 60 FF 51 03 08 CF-DC 83 60 FF 51 03 09 58    x.`.Q.....`.Q..X
94 83 60 FF 51 03 07 D1-F4 83 60 FF 51 03 08 E3    ..`.Q.....`.Q...
64 83 60 FF 51 03 08 81-BC 83 60 FF 51 03 08 81    d.`.Q.....`.Q...
BC 83 60 FF 51 03 08 8B-80 83 60 FF 51 03 08 64    ..`.Q.....`.Q..d
70 83 60 FF 51 03 08 ED-28 83 60 FF 51 03 08 64    p.`.Q...(.`.Q..d
70 83 60 FF 51 03 09 14-38 83 60 FF 51 03 08 A8    p.`.Q...8.`.Q...
CC 83 60 FF 51 03 08 BC-54 83 60 FF 51 03 08 6E    ..`.Q...T.`.Q..n
34 83 60 FF 51 03 08 64-70 83 60 FF 51 03 08 A8    4.`.Q..dp.`.Q...
CC 83 60 FF 51 03 08 8B-80 83 60 FF 51 03 08 A8    ..`.Q.....`.Q...
CC 83 60 FF 51 03 08 47-24 83 60 FF 51 03 08 3D    ..`.Q..G$.`.Q..=
60 83 60 FF 51 03 08 D9-A0 83 60 FF 51 03 08 C6    `.`.Q.....`.Q...
18 83 60 FF 51 03 08 64-70 83 60 FF 51 03 08 ED    ..`.Q..dp.`.Q...
28 83 60 FF 51 03 07 EF-40 83 60 FF 51 03 08 C6    (.`.Q...@.`.Q...
18 83 60 FF 51 03 08 95-44 83 60 FF 51 03 08 ED    ..`.Q...D.`.Q...
28 83 60 FF 51 03 08 B2-90 83 60 FF 51 03 08 CF    (.`.Q.....`.Q...
DC 83 60 FF 51 03 08 BC-54 83 60 FF 51 03 09 9C    ..`.Q...T.`.Q...
F0 83 60 FF 51 03 08 5A-AC 83 60 FF 51 03 09 4E    ..`.Q..Z..`.Q..N
D0 83 60 FF 51 03 08 E3-64 83 60 FF 51 03 08 C6    ..`.Q...d.`.Q...
18 83 60 FF 51 03 08 ED-28 83 60 FF 51 03 09 27    ..`.Q...(.`.Q..'
C0 83 60 FF 51 03 08 C6-18 83 60 FF 51 03 08 CF    ..`.Q.....`.Q...
DC 83 60 FF 51 03 08 95-44 83 60 FF 51 03 08 B2    ..`.Q...D.`.Q...
90 83 60 FF 51 03 09 14-38 83 60 FF 51 03 08 CF    ..`.Q...8.`.Q...
DC 83 60 FF 51 03 08 D9-A0 83 60 FF 51 03 08 CF    ..`.Q.....`.Q...
DC 83 60 FF 51 03 09 14-38 83 60 FF 51 03 08 C6    ..`.Q...8.`.Q...
18 83 60 FF 51 03 08 D9-A0 83 60 FF 51 03 08 33    ..`.Q.....`.Q..3
9C 83 60 FF 51 03 07 0E-A4 83 60 FF 51 03 07 EF    ..`.Q.....`.Q...
40 83 60 FF 51 03 08 0C-8C 83 60 FF 51 03 07 AA    @.`.Q.....`.Q...
E4 83 60 FF 51 03 08 0C-8C 83 60 FF 51 03 07 E5    ..`.Q.....`.Q...
7C 83 60 FF 51 03 07 D1-F4 83 60 FF 51 03 07 53    |.`.Q.....`.Q..S
00 83 60 FF 51 03 08 6E-34 83 60 FF 51 03 08 D9    ..`.Q..n4.`.Q...
A0 83 60 FF 51 03 07 D1-F4 83 60 FF 51 03 08 3D    ..`.Q.....`.Q..=
60 83 60 FF 51 03 07 F9-04 83 60 FF 51 03 08 C6    `.`.Q.....`.Q...
18 83 60 FF 51 03 08 A8-CC 83 60 FF 51 03 07 E5    ..`.Q.....`.Q...
7C 83 60 FF 51 03 07 BE-6C 83 60 FF 51 03 08 81    |.`.Q...l.`.Q...
BC 83 60 FF 51 03 07 C8-30 83 60 FF 51 03 08 D9    ..`.Q...0.`.Q...
A0 83 60 FF 51 03 08 64-70 83 60 FF 51 03 08 5A    ..`.Q..dp.`.Q..Z
AC 83 60 FF 51 03 08 6E-34 83 60 FF 51 03 07 E5    ..`.Q..n4.`.Q...
```

```
7C 83 60 FF 51 03 08 29-D8 83 60 FF 51 03 07 F9      |.`.Q..)..`.Q...
 04 83 60 FF 51 03 08  0C-8C  83 60 FF 51 03 08 02     ..`.Q.....`.Q...
 C8 83 60 FF 51 03 08  95-44  83 60 FF 51 03 09 0A     ..`.Q...D.`.Q...
 74 83 60 FF 51 03 09  C4-00  83 60 FF 51 03 08 B2      t.`.Q.....`.Q...
 90 00 FF 58 04 04 02  18-08  83 60 FF 51 03 07 70     ...X.....`.Q..p
 4C 83 60 FF 51 03 08  0C-8C  83 60 FF 51 03 07 83     L.`.Q.....`.Q...
 D4 83 60 FF 51 03 08  95-44  83 60 FF 51 03 07 E5     ..`.Q...D.`.Q...
 7C 83 60 FF 51 03 08  02-C8  83 60 FF 51 03 08 20     |.`.Q.....`.Q..
 14 83 60 FF 51 03 08  5A-AC  83 60 FF 51 03 08 16     ..`.Q..Z..`.Q...
 50 83 60 FF 51 03 08  02-C8  83 60 FF 51 03 07 DB     P.`.Q.....`.Q...
 B8 83 60 FF 51 03 08  3D-60  83 60 FF 51 03 08 ED     ..`.Q..=`.`.Q...
 28 83 60 FF 51 03 09  45-0C  83 60 FF 51 03 08 20     (.`.Q..E..`.Q..
 14 83 60 FF 51 03 08  5A-AC  83 60 FF 51 03 08 33     ..`.Q..Z..`.Q..3
 9C 83 60 FF 51 03 08  A8-CC  83 60 FF 51 03 08 8B     ..`.Q.....`.Q...
 80 83 60 FF 51 03 08  5A-AC  83 60 FF 51 03 08 5A     ..`.Q..Z..`.Q..Z
 AC 83 60 FF 51 03 08  95-44  83 60 FF 51 03 08 64     ..`.Q...D.`.Q..d
 70 83 60 FF 51 03 08  77-F8  83 60 FF 51 03 08 E3     p.`.Q..w..`.Q...
 64 83 60 FF 51 03 08  BC-54  83 60 FF 51 03 09 45     d.`.Q...T.`.Q..E
 0C 83 60 FF 51 03 08  E3-64  83 60 FF 51 03 08 A8     ..`.Q...d.`.Q...
 CC 83 60 FF 51 03 08  ED-28  83 60 FF 51 03 08 E3     ..`.Q...(.`.Q...
 64 83 60 FF 51 03 09  45-0C  F8 00 FF 51 03 0A F2     d.`.Q..E....Q...
 BC 83 60 FF 51 03 0A  E8-F8  83 60 FF 51 03 0B 19     ..`.Q.....`.Q...
 CC 83 60 FF 51 03 0B  54-64  83 60 FF 51 03 0B 23     ..`.Q..Td.`.Q..#
 90 83 60 FF 51 03 0B  FA-68  83 60 FF 51 03 0B 54     ..`.Q...h.`.Q..T
 64 83 60 FF 51 03 0B  FA-68  83 60 FF 51 03 0A E8     d.`.Q...h.`.Q...
 F8 83 60 FF 51 03 0B  85-38  83 60 FF 51 03 0A C1     ..`.Q...8.`.Q...
 E8 83 60 FF 51 03 0C  04-2C  83 60 FF 51 03 0A C1     ..`.Q...,.`.Q...
 E8 83 60 FF 51 03 0B  40-DC  83 60 FF 51 03 0B 4A     ..`.Q..@..`.Q..J
 A0 83 60 FF 51 03 0B  B6-0C  83 60 FF 51 03 0A A4     ..`.Q.....`.Q...
 9C 83 60 FF 51 03 0B  54-64  83 60 FF 51 03 0A E8     ..`.Q..Td.`.Q...
 F8 83 60 FF 51 03 0B  06-44  83 60 FF 51 03 0B 8E     ..`.Q...D.`.Q...
 FC 83 60 FF 51 03 0C  79-5C  83 60 FF 51 03 0B 7B     ..`.Q..y\.`.Q..{
 74 83 60 FF 51 03 0B  7B-74  83 60 FF 51 03 0A 87     t.`.Q..{t.`.Q...
 50 83 60 FF 51 03 0A  7D-8C  83 60 FF 51 03 0A B8     P.`.Q..}..`.Q...
 24 83 60 FF 51 03 0B  71-B0  83 60 FF 51 03 0A D5     $.`.Q..q..`.Q...
 70 83 60 FF 51 03 0A  D5-70  83 60 FF 51 03 0B BF     p.`.Q...p.`.Q...
 D0 83 60 FF 51 03 0B  10-08  83 60 FF 51 03 0A 7D     ..`.Q.....`.Q..}
 8C 83 60 FF 51 03 0B  D3-58  83 60 FF 51 03 0A AE     ..`.Q...X.`.Q...
 60 83 60 FF 51 03 0B  85-38  83 60 FF 51 03 0B 71     `.`.Q...8.`.Q..q
 B0 83 60 FF 51 03 0B  B6-0C  83 60 FF 51 03 0B 40     ..`.Q.....`.Q..@
 DC 83 60 FF 51 03 0B  B6-0C  83 60 FF 51 03 0B BF     ..`.Q.....`.Q...
 D0 83 60 FF 51 03 0A  91-14  83 60 FF 51 03 0B A2     ..`.Q.....`.Q...
 84 83 60 FF 51 03 0B  23-90  83 60 FF 51 03 0A DF     ..`.Q..#..`.Q...
 34 83 60 FF 51 03 0A  AE-60  83 60 FF 51 03 0B 37     4.`.Q...`.`.Q..7
 18 83 60 FF 51 03 0B  B6-0C  83 60 FE 00 02 C8 05     ..`.Q.....`.....
 00 FE 00 02 CE 05 00  FF-51  03 0A DF 34 00 FF 2F     .........Q...4../
 00 4D 54 72 6B 00 00  00-B6  00 FF 03 05 43 72 61     .MTrk.........Cra
 73 68 00 B9 07 66 81  B7-60  99 31 08 3C 89 31 40     sh...f..`.1..1@
 00 99 31 0C 3C 89 31  40-00  99 31 10 3C 89 31 40     ..1..1@..1..1@
 00 99 31 18 3C 89 31  40-00  99 31 20 3C 89 31 40     ..1..1@..1 .1@
 00 99 31 30 3C 89 31  40-00  99 31 50 3C 89 31 40     ..10.1@.:1P.1@
```

00 99 31 70 3C 89 31	40-8B	20 99 31 0C 3C 89 31	..1p.1@. .1..1
40 00 99 31 10 3C 89	31-40	00 99 31 18 3C 89 31	@..1..1@..1..1
40 00 99 31 20 3C 89	31-40	00 99 31 2C 3C 89 31	@..1 .1@..1,.1
40 00 99 31 38 3C 89	31-40	00 99 31 48 3C 89 31	@..18.1@..1H.1
40 00 99 31 78 3C 89	31-40	82 83 06 99 31 7C 82	@..1x.1@....1l.
33 89 31 40 8C 30 99	31-7F	71 89 31 40 82 AE66	3.1@.0.1.q.1@..f
FE 00 02 AA00 00 FE	00-02	B0 00 00 FF 2F 00 4D	...../.M
54 72 6B 00 00 00 2F	00-FF	03 04 46 6F 6F 74 83	Trk.../....Foot.
CD4B 99 24 6C 81 25	89-24	40 8D 15 99 24 66 81	.K.$l.%.$@...$f.
15 89 24 40 82 AE66	FE-00	02 23 00 00 FE 00 02	..$@..f...#.....
29 00 00 FF 2F 00 4D	54-72	6B 00 00 01 B1 00 FF	).../.MTrk.....
03 05 53 6E 61 72 65	83-A4	00 99 26 10 81 70 89	..Snare....&..p.
26 40 00 99 26 10 50	89-26	40 00 99 26 10 50 89	&@..&.P.&@..&.P.
26 40 00 99 26 16 50	89-26	40 00 99 26 16 81 70	&@..&.P.&@..&..p
89 26 40 00 99 26 16	81-70	89 26 40 00 99 26 16	.&@..&..p.&@..&.
81 70 89 26 40 00 99	26-16	81 70 89 26 40 00 99	.p.&@..&..p.&@..
26 18 78 89 26 40 00	99-26	18 78 89 26 40 00 99	&.x.&@..&.x.&@..
26 18 78 89 26 40 00	99-26	18 78 89 26 40 00 99	&.x.&@..&.x.&@..
26 18 81 70 89 26 40	00-99	26 18 78 89 26 40 00	&..p.&@..&.x.&@.
99 26 18 78 89 26 40	00-99	26 18 81 70 89 26 40	.&.x.&@..&..p.&@
00 99 26 1C 81 70 89	26-40	00 99 26 1E 81 70 89	..&..p.&@..&..p.
26 40 00 99 26 20 78	89-26	40 00 99 26 22 78 89	&@..& x.&@..&"x.
26 40 00 99 26 22 81	70-89	26 40 00 99 26 2A 50	&@..&".p.&@..&*P
89 26 40 00 99 26 2A	50-89	26 40 00 99 26 32 50	.&@..&*P.&@..&2P
89 26 40 00 99 26 33	81-70	89 26 40 00 99 26 33	.&@..&3.p.&@..&3
50 89 26 40 00 99 26	3A-50	89 26 40 00 99 26 30	P.&@..&:P.&@..&0
50 89 26 40 00 99 26	42-81	70 89 26 40 00 99 26	P.&@..&B.p.&@..&
2B 81 70 89 26 40 00	99-26	54 81 70 89 26 40 00	+.p.&@..&T.p.&@.

读写DSP

使用DSP功能30H-3FH可以读写MIDI端口。为了实现读或写DSP，下面的端口是一些重要端口：

2x6H　DSP复位（只能写入）

2xAH　DSP读数据（只能读出）

2xCH　DSP写数据或命令（只写）

2xCH　DSP写缓冲区状态（只读）

2xEH　DSP数据有效的状态（只读）

其中x表示16进制数中的1-6的值，根据I/O跳接器位置而定。出厂时跳接器位置是220H，如果你保留了220H位置，x就为2。

MIDI DSP的功能　MIDI DSP的功能如下：

30H　MIDI读出模式（直接）

31H　MIDI读出模式（中断）

38H　MIDI写入模式（直接）

如前所述，MIDI数据只能被直接写到MIDI端口，为此，你写一个38H命令到DSP命令端口（2XCH），然后写必须要写到DSP数据端口的MIDI字节（它是与DSP命令端口一样的）。

但是，这些指令被送到DSP以前，你必须先检查DSP是否能处理数据或命令。为进行检查，应该读出DSP写缓冲状态（端口也是2XCH），如果所读出字节的位7被置位（=1），DSP就不能接收新的数据或命令。

直接读MIDI数据 为了直接读MIDI数据，应该把功能30H命令送到DSP命令端口（2XCH），为了检验MIDI数据是否可用，需要读DSP数据有效端口（2XEH）。如果读出的这个字节的第7位被置成1，表示在缓冲器里的MIDI数据有效，这时你可以通过读DSP读数据端口（2XAH）而读出一个MIDI字节。

使用中断程序读MIDI数据

推荐的读MIDI数据的方法是使用中断程序，Sound Blaster卡包含一个跳接器，用来指明那一个中断请求（IRQ）分配给Sound Blaster，可在2，3，5和7中间选择，程序应该根据这个中断号来写。

注解 有关在Sound Blaster卡中设置跳接器的说明请见第一章。

为使DSP一定在中断方式下工作，必须把31H命令送到DSP。当接收MIDI数据时，处理器被中断，而运行中断程序。为了撤消这个中断请求，应该读一次DSP状态端口（2XEH）。完成以后，可以在DSP读数据端口（2XAH）恢复MIDI数据字节。

如果由于你不再需要中断程序，比如说想返回到DOS，就可送一个31H命令停止中断服务。

送出和读出命令及数据字节

为送出一字节到DSP或从DSP读出一字节都需通过一个过程，但在送入和读出这些字节之前，你应检验DSP是否已经准备好处理命令或数据字节。

为送一个命令或数据字节到DSP，可按下列步骤执行：

1. 读DSP写缓冲器状态端口2XCH。
2. 检查读出字节的第7位是否为复位（=0），假若第7位被置位（=1），必须重复步骤1。
3. 写命令或数据字节到DSP写数据或命令端口2XCH。

为了从DSP读数据端口读出一数据字节，需要执行下列步骤：

1. 读DSP数据有效状态端口2XEH。
2. 检查读出的字节第7位是否置位（=1），如果不为1，则重复步骤1。
3. 从DSP读数据端口2XAH读数据字节。

MIDI和Sound Blaster Pro

随着Sound Blaster Pro的开发，添加了许多功能，使MIDI有更多的能力。Sound Blaster Pro能在UART模式下运行，在这种模式中，可以同时读写MIDI数据。UART模式具有处理Pro的全双工能力。对Sound Blaster来说，增加了以下DSP MIDI功能：

34H： MIDI UART模式（登记）

35H： MIDI UART模式（中断）

UART模式假设你正在读MIDI数据，如果你打算写MIDI数据，则把该MIDI数据写到DSP的写数据端口（2XCH）。在UART方式，DSP认为所有送到DSP数据端口的数据都是被传输的MIDI数据。

MIDI时间标记的使用 MIDI时间标记同Sound Blaster Pro一起使用，在每个进入MIDI字节之前，有三个时间标记字节。三个时间标记字节表明自最后一个DSP复位以来所经过的时间。增加了两种新的模式以告诉DSP，它应该传送时间标记码：

33H： MIDI时间标记代码（中断 非UART）

37H： MIDI时间标记代码（中断 UART）

由于与Sound Blaster Pro一起工作，使用时间标记代码有许多后果：

- 大大减少了MIDI数据缓冲区的大小，在通常方式，缓冲区能容纳64MIDI代码，但现在它只能保存16MIDI代码，所以要及时从MIDI缓冲区移走数据。
- 建立一个音序器非常容易，总之，你无须注意时间或者时间中的差异，DSP芯片只简单地同MIDI字节一起存贮它们。
- 为了支持原有老的Sound Blaster，需检查它的版本

检查DSP版本 为了检查Sound Blaster版本，得知道DSP芯片的版本号。为此，创设了这一功能，通过下面的DSP命令调用该功能：

E1H： 版本号DSP

DSP将在MIDI数据缓冲区中放置两个字节：

- 在读了读数据端口后，接收的第一个字节是版本号小数点前部分的数字。
- 第二个字节表示的是小数点后部分的数字。

注解 **Sound Blaster1.0，1.5，1.6和MCV都有版本号在1.5和2.0之间的DSP，Sound blaster Pro的DSP版本号是2.0或2.0以上。**

程序实例： 音序器

在本节，用了不同的MIDI写和读模式来写一个简单的音序器。对音序器来说最重要的事是时间。如果在X时刻接收到MIDI数据，那下一个MIDI数据到达将是在X+31时刻，当MIDI数据被播放时，这31的差值也必须被表现出来。简而言之，这些连续的MIDI数据流之间在时间上的不同必须被准确地确定。

这就是定时器芯片起作用之处，这个定时器芯片可以在每秒钟被激活很多次。这种定时器芯片是基于中断的，就是说芯片有自己的中断线（中断号），所以它可在每秒钟内自动地被调用一定的次数（一般为18.2），这是由程序人员来决定的，他们确定这定时器中断多少时间需被调用一次。）

以下描述的简单音序器具有这样些功能：

- 记录进入的MIDI数据直至一个键被按下。
- 播放进入的MIDI数据直至一个键被按下。

在本章里的前四个程序列表如下：

编号	标题
10.1	MIDILIB
10.2	SEQ.PAS
10.3	MIDI.H
10.4	SEQ.C

注解　参看绪言中“有关程序清单的说明”，可了解有关阅读和使用这些程序清单的建议。

程序清单10.1　MIDILIB

```
{ ********************************************
** Unit MIDILIB supporting SEQ.PAS          **
** and MIDICMF.PAS                          **
** Turbo Pascal 4.0/6.0                     **
******************************************** }

Unit midilib ;

Interface

Const
port          = $220 ;
reset         = port+$06 ;  { registers DSP }
readdata      = port+$0a ;
writeom       = port+$0c ;
writebuf      = port+$0c ;
dataavail     = port+$0e ;
OldtimV       = 103 ;

type
version=record
        high    : byte ;
        low     : byte ;
  end ;

procedure resetDSP ;
procedure writedat (n : byte) ;
function readdat : byte ;
function polreaddat : byte ;
procedure settimspeed (Freq : word) ;
procedure settimer (Ferq : word ; Rout : pointer) ;
procedure resettimer ;
procedure get_version_number (var number : version) ;

implementation
uses dos, crt ;
```

```
procedure resetDSP ;          { initializing DSP chip }
var
  b : word ;
begin
  port [reset] :=1 ;
  for b :=3 downto 0 do ;
  port [reset] :=0 ;
  while (port[dataavaol] and 128)=0 do ;
  while not (port[readdata]=$aa) do ;
  writeln('Reset DSP OK. ') ;
end ;

procedure writedat(n : byte) ;        { write a data byte }
begin                                 { or command to the DSP }
  while (port[writebuf] and 128)<>0 do ;
  port[writecom] :=n ;
end ;

function readdat : byte ;     { read a byte from the DSP }
begin                                              { buffer }
  while (port[dataavail] and 128)=0 do ;
  readdat :=port[readdata] ;
end ;

function polreaddat : byte ;  { in polling mode also pay }
begin                         { attention to key press }
  while (((port[dataavail] and 128)=0) and not
        keypressed) do ;
  polreaddat :=port[readdata] ;
end ;

procedure settimspeed(Freq : word) ; { calculate and pass }
var                           { timer speed }
  ICnt : longint ;            { on to the timer chip }
begin
  inline($FA) ;
  ICnt := 1193180 div Freq :
  port[$43] :=$36 ;
  port[$40] :=lo(ICnt) ;
  port[$40] :=hi(ICnt) ;
  inline($FB) ;
  writeln('Clock tick = ', Freq) ;
end ;

procedure settimer(Freq : word ; Rout : pointer) ;
var
  OldV : pointer ;
begin
            inline($FA) ;
```

```
        getintvec(8, OldV) ;          { take over timer interrupt }
        setintvec(OldtimV, Oldv) ;
    setintvec(8, Rout) ;
    settimspeed(Freq) ;
        inline($FB) ;
end ;

procedure resettimer ;          { restore all changes }
Var                                     { to the timer }
  OldV : pointer ;
begin
  inline($FA) ;
  port[$43] :=$36 ;
  port[$40] :=$0 ;
  port[$40] :=$0 ;
  getintvec(OldtimV, Oldv) ;
  setintvec(8, Oldv) ;
  inline($FB) ;
end ;

procedure get_version_number( var number : version ) ;
begin                   { return DSP version number }

  writedat($E1) ;
  number.high := readdat ;
  number.low := readdat ;
end ;

end .
```

程序清单10.2： SEQ.PAS

```
{ ********************************************
** SEQ.PAS                                    **
** Turbo Pascal 6.0                           **
******************************************** }

program sequencer ;

Uses Dos, Crt, midilib ;

Const
port = $220 ;          { jumper values }
IRQ = $5 ;

var
  read_write : boolean ;
  counter : longint ;
  i : word ;
```

```
  midi_time : array [0..1000] of longint ;
  midi_note : array [0..1000] of byte ;

procedure newtim ; interrupt; { new timer interrupt }
var R : registers ;
begin
  if read_write then inc(counter)
  else
    if (counter>0) then dec (counter) ;
  intr (OldtimV, R) ;
end ;

procedure newIRQ ; interrupt ;        { interrupt for }
var dummy : byte ;                    { interrupt mode }
begin
  if (i<1000) then
    begin
      midi_note[i] :=readdat ;
      midi_time[i] :=counter ;
      inc(i) ;
      counter :=0 ;
    end ;
  dummy :=port[dataavail] ;
  port[$20] :=$20 ;
end ;

procedure uartIRQ ; interrupt;        { interrupt for }
var dummy : byte ;
begin
  if (i<1000) then
  begin
    midi_time[i] :=readdat ;
    midi_time[i] :=midi_time[i]+readdat*256 ;
    midi_time[i] :=midi_time[i]+readdat*65536 ;
    midi_note[i] :=readdat ;
      inc(i) ;
    end ;
  dummy :=port[dataavail] ;
  port[$20] :=$20 ;
end ;

var
  k : word ;
  oldport : byte ;
  choice, allowed, dummy : char ;
  number : version ;

begin                          { main program }
  resetDSP ;
```

```
  clrscr ;
  allowed :='2' ;
  choice :='0' ;
  writeln('1. MIDI polling mode. ') ;
  writeln('2. MIDI interrupt mode.') ;
  get_version_number(number) ;
  if (number.high>=2) then
    begin
    writeln('3. MIDI UART with Time Stamp.') ;
    allowed :='3' ;
    end ;
  write('DSP version number ') ;
  writeln (number.high, '.', number.low) ;
  writeln ('Select a mode : ') ;
  while (( cholce<'1') or (choice>allowed)) do
  choice :=readkey ;
  writeln('Press a key to start.') ;
  dummy :=readkey ;
  writeln('Go !') ;
  case choice of
  '1' : begin
    writedat($30) ;                    { polling mode }
    settimer(100, @newtim) ;           { use timer }
    read_write := true ;
    while ((i<1000) and not keypressed) do
    begin
      midi_mote[i] :=polreaddat ;
      midi_time[i] :=counter ;
    inc(i) ;
    counter :=0 ;
  end ;
end ;

'2' : begin
  setintvec(*+IRQ, @newIRQ) ;      { set new IRQ }
  oldport :=port[$21] ;
  port[$21] :=oldport and ($ff xor (1 shl IRQ)) ;
  writedat($31) ;
  read_write := true ;
  settimer(100, @newtim) ;
  while not keypressed do ;
  writedat($31) ;
  port[$21] :=oldport ;
  end ;
'3' : begin
  setintvec(*+IRQ, @uartIRQ) ;     { set UART IRQ }
  oldport :=port[$21] ;
  port[$21] :=oldport and ($ff xor  (1 shl IRQ )) ;
  writedat($37) ;                  { UART mode with MTC }
```

```
  while not keypressed do ;
  port[$21] :=oldport ;
  end ;
end ;
if keypressed then dummy :=readkey ;
if (choice<'3') then
  begin
    resetDSP ;
    read_write :=false ;
  end
else
  begin
    settimer(1000, @newtim) ;        { timer in millisecond }
    read_write :=true ;
    counter :=0 ;
  end ;
if (midi_note[0]<$80) then    { started with a }
begin                         { running status byte ? }
  if (choice<'3') then writedat($38) ;
  writedat($90) ;
end ;
k :=0 ;
while ((k<i) and not keypressed) do
begin
  if (choice<'3') then
  begin                       { and send the MIDI }
   counter := midi_time[k] ; { data back again }
   while (counter>0) do ;
   writedat($38) ;
   writedat(midi_note[k]) ;
   inc(k) ;
  end
  else
  begin                       { for the UATR mode }
   while (counter<midi_time[k]) do ;
   writedat(midi_note[k]) ;
   inc(k) ;
   end ;
  end ;
  resettimer ;
  writeln('Ready.') ;
  if (choice='3') then resetDSP ;
end .
```

程序清单10.3： MIDI.H

```
/* ********************************************
**  Include file MIDI.H for MIDICMF.C and      **
** SEQ.C                                       **
```

```
** Turbo C++ 2.0                                    **
********************************************** */

#include <dos.h>
#pragma inline
#define reset          port+0x06          /* definition of */
#define readdata       port+ox0a          /* all DSP ports */
#define writecom       port+0x0c          /* conc. MIDI */
#define writebuf       port+0x0c
#define dataavail      port+0x0e

struct version                            /* record for */
  {                                       /* version number */
    unsigned char high ;
    unsigned char low ;
  } ;

void resetDSP ()                  /* reset DSP chip */
{
  int b ;
  outp(reset, 1) ;
  for (b=3; b>0; b--) ;
  outp(reset, 0) ;
  while (!(inp(dataavail) &128)) ;
  while (inp(readdata) !=0xaa ) ;
  printf('\nReset DSP OK. \n') ;
}

void writedat(n)              /* write a byte to */
int n ;                       /* the DSP */
{
  while (inp(writebuf) &128) ;
  outp(writecom, n) ;
}

readdat()             /* read a byte from the DSP */
{
  short int n ;
  while (!(inp(dataavail) &128)) ;
  n=(inp(readdata)) ;
  return(n) ;
}

polreaddat()          /* in polling mode you stay in */
{                     /* a loop without kbhit() */
  short int n ;
  while (!(inp(dataavail)&128)&& !kbhit()) ;
  n=(inp(readdata)) ;
  return(n) ;
```

```
}

void interrupt (*oldtim) (void) ;      /* calculate the number */

void settimspeed (unsigned Freg)
{
  int ICnt ;
  asm cli ;
  ICnt = 1193180/Freq ;                /* calculate the number */
  outp(0x43, 0x36) ;                   /* off clock ticks and pass */
  outp(0x40, ICnt &255) ;              /* on this number */
  outp(0x40, ICnt >>8) ;
  asm sti ;
  printf('\nClock tick = %d\n', Freq) ;
}

void settimer(unsigned Freq, void interrupt (*newIRQ)())
{
  oldtim = getvect(8) ;        /* take over timer and */
  setvect(8, newIRQ) ;
  settimspeed(Freq) ;          /* set speed */
}

void resettimer()
{
  asm cli ;
  outp(ox43, ox36) ;           /* reset all timer */
  outp(0x40, 0x0) ;            /* occupations */
  outp(0x40, 0x0) ;
  setvect(8, oldtim) ;
  asm sti ;
}

struct version get_version_number()
{
  struct version number ;

  writedat(0xE1) ;     /* give DSP version num. command */
  number.high= readdat() ;  /* read the values from the */
  number.low= readdat() ;   /* DSP buffer */
  return(number) ;          /* return values */
}
```

程序清单10.4： SEQ.C

```
/* ********************************************
** SEQ.C                                      **
** Turbo C++ 2.0                              **
******************************************** */
```

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

#define port 0x220    /* selected port */
#define IRQ 0x5       /* selected IRQ */

#include "midi.h"
unsigned char read_write=0 ;
long counter=0 ;
int i=0 ;

long midi_time[1000] ;              /* buffers for */
unsigned char midi_note[1000] ;     /* time and note */

void interrupt newtim()             /* counters */
{
  if (read_write)
    counter++ ;
  else
    if (counter>0) counter-- ;
  oldtim() ;
}

void interrupt newIRQ()      /* for interrupt mode */
{
  if (i<1000)
    {
      midi_note[i]=readdat() ;      /* read midi byte */
      midi_time[i++]=counter ;      /* write time */
      counter=0 ;
    }
  inp(dataavail) ;
  outp(0x20, 0x20) ;
}

void interrupt uartIRQ()     /* for UART mode */
{
  if (i<1000)
    {
      midi_time[i]=readdat() ;              /* time stamp */
      midi_time[i]=readdat()*256 ;
      midi_time[i]+=readdat()*65536 ;
      midi_note[i++]=readdat() ;            /* midi byte */
    }
  inp(dataavail) ;
  outp(0x20, 0x20) ;
}
```

```
void main(void)
{
int k=0 ;
unsigned char oldport ;
char keuze='0', allowed ;
struct version number ;

resetDSP() ;
clrscr() ;
allowed='2' ;
printf("\n\n\n\n\n\t\t1. MIDI polling mode.\n\n") ;
printf("\t\t2. MIDI interrupt mode.\n\n") ;
number=get_version_number() ;
if (number.high>=2)
  {
   printf("\t\t3. MIDI UART with Time Stamp.\n\n") ;
   allowed='3' ;
  }
printf("\t\tDSP version number ") ;
printf("%d.%d\n\n", number. high, number.low) ;
printf("\t\tSelect a mode : ") ;
while (cholce<'1' || choice>allowed)
choice=getch() ;
printf("\nPress a key to start.\n") ;
getch() ;
printf ("GO ! \n") ;
switch (choice) {
case '1' :                    /* polling mode */
   writedat(0x30) ;
   settimer(100, newtim) ;
   read_write=1 ;
   while ((i<1000) && !kbhit())
   {
      midi_note[i]=polreaddat() ;       /* reading each time */
      midi_time[i++]=counter ;
      counter=0 ;
   }
   break ;
case '2' :                    /* interrupt mode */
   setvect(8+IRQ, newIRQ) ;
   oldport=inp(0x21) ;
   outp(0x21, oldport&(0xff^(1<<IRQ))) ;   /* mask IRQ */
   writedat(0x31) ;
   settimer(100, newtim) ;
   read_write=1 ;
   while(!kbhit()) ;                  /* wait for a key */
   writedat(0x31) ;                   /* return to normal */
   outp(0x21, oldport) ;
   break ;
```

```
case '3' :                      /* UART mode */
   setvect(8+IRQ, uartIRQ) ;
   oldport=inp(0x21) ;
   outp(0x21, oldport&(0xff^(1<<IRQ))) ;
   writedat(0x37) ;             /* command UART MTC */
   while(!kbhit()) ;
   outp(0x21, oldport) ;
   break ;
default : break ;
  }
if (kbhit()) getch() ;
if (choice<'3')                 /* play all */
  {
    resetDSP() ;
    read_write=0 ;
  }
else
  {
  settimer(1000, newtim) ;
  read_write=1 ;
  counter=0 ;
  }
if (midi_note[0]<0x80)          /* started with running */
{                                       /* status byte ? */
  if (choice<'3')writedat(0x38) ;
  writedat(0x90) ;
}
while (k<i && !kbhit())
 {
 if (choice<'3')
 {
  counter=midi_time[k] ;        /* if counter is zero, */
  while (counter>0) ;           /* send the MIDI byte */
  writedat(0x38) ;
  writedat(midi_note[k++]) ;
 }
 else
 {
  while (counter<midi_time[k]) ;
  writedat(midi_note[k++]) ;
 }
}
resettimer() ;
  printf("Ready. \n") ;
  if (choice=='3') resetDSP() ;
}
```

通过MIDI播放CMF歌曲

CMF格式由一部分为Creative Labs公司创建的许多参数，另一部分为实际的MIDI文件所组成。具体地说，在CMF格式里，文件头块和设备块之后接着是音乐块，这个与MIDI文件格式0相同。

一个装入和播放CMF文件的程序清单如清单10.5和10.6所示，程序使用了用Pascal写成的MIDI.LIB程序单元（参见清单10.1）和用C写成的MIDI.H文件（见清单10.3），这两个程序同音序器程序一起使用。

程序清单10.5： 用Pascal编写的CMFMIDI播放程序

```
{ ************************************************
** MIDICMF.PAS                                   **
** Turbo Pascal 4.0                              ))
************************************************ }

Program Midicmf ;

Uses Dos, Crt midilib ;

Const  port = $220 ;          { selected jumpers }
       IRQ = $7 ;

var   nexttoken : byte ;
      clockick : word ;
      stop : byte ;
      counter : longint ;
      mempos : pounter ;
      size ; word ;
      number_of_bytes : integer ;
      result : integer ;
      formattype : integer ;
      cont : integer ;
      curmempos : word ;
      ch : char ;

Function ReadPos (index : word) : byte ;
                        { reads an indexed byte }
begin
  readpos :=mem[Seg(mempos^) :Ofs(mempos^)+index]
end ;

Function LoadFile(Filename : string) : byte ;
var fp : FILE ;
begin                   { loads a file in memory }
       { $I- }
```

```
        Assign(fp, filename) ;
        if (ioresult <> 0) then
        begin
          loadfile :=1 ;
          exit ;
        end ;
        System.Reset (fp, 1) ;
        { $I+ }
        if (ioresult < > 0) then
        begin
          loadfile : = 2 ;
          exit
        end ;
        If (filesize(fp)>$FFFF) then
        begin
          Writeln('File too large for buffer. ') ;
          loadfile :=3 ;
        end ;
        size := filesize(fp) ;
        getmem(mempos, size) ;
        if (mempos=nil) then
        begin
          writeln('no memory') ;
          loadfile :=4 ;
          exit
        end ;
        blockread(fp, mempos^, size, number_of_bytes) ;
        if ((ioresult<>0) or (mumber_of_bytes<>size)) then
        begin
          loadfile :=3 ;
          exit ;
        end ;
        close (fp) ;
        loadfile :=0 ;
end ;

Function Check : byte ;                    { is the loaded }
const midiheader : string[4]='MThe' ;      { file a MIDI }
      cmfheader : string[4]='CTMF' ;       { or a CMF file ? }

var j : byte ;
  temp : string [4]

begin
  j := Readpos [4]
   Move(mempos^, temp[1], 4) ; temp[0] :=#4 ;    { size of 4 bytes }

   if (temp=cmfheader) then
     check :=0
```

```
    else
    if (tem=midiheader) then
      check :=1
    else
      check :=255 ;

end ;

Procedure Error ;
begin
  writeln('What you say is nonsense...') ;
  writeln(nexttoken) ;
  halt(1) ;
end ;

Procedure Send (token : byte) ;      { send a byte to }
begin                                        { the DSP }
  writedat($38) ;
  writedat(token) ;
end ;

Function Next : byte ;               { read the next byte }
begin
  next :=ReadPos(curmempos) ;
  Inc(curmempos) ;
end ;

Procedure SysExmsg ;
begin
  send(nexttoken) ;
  nexttoken :=next ;

  while (nexttoken>=#f8) do          { real-time messages }
  begin
    send (nexttoken) ;
    nexttoken :=next ;
  end ;
  if (nexttoken<$80) then            { data byte }
  begin
    send (nexttoken) ;
    nexttoken :=next ;
  end
  else
    error ;

  while not (nexttoken=$f7) do       { EOX }
  begin
    while (nexttoken>=$f8) do        { real-time messages }
    begin
```

```
        send(nexttoken) ;
        nexttoken :=next ;
      end ;

      while (nexttoken<$80) do        { data bytes }
      begin
        send(nexttoken) ;
        nexttoken :=next ;
      end ;

    end ;
    send(nexttoken) ;
    nexttoken :=next ;
end ;

Procedure SysSelect ;
begin
    send(nexttoken) ;
    nexttoken :=next ;
    while (nexttoken>=$f8) do         { real-time messages }
    begin
      send(nexttoken) ;
      nexttoken :=next ;
    end ;

    if (nexttoken < $80) then         { data byte }
    begin
      send(nexttoken) ;
      nexttoken :=next ;
    end
    else
      error ;
end ;

Procedure SysSongPos ;
begin
    send(nexttoken) ;
    nexttoken :=next ;
    while (nexttoken>=$f8) do         { real-time messages }
    begin
      send (nexttoken) ;
      nexttoken :=next ;
    end ;

    if (nexttoken <$80) then          { data byte }
    begin
      send(nexttoken) ;
      nexttoken :=next ;
    while (nexttoken >= $f8) do       { real-time messages }
```

```
  begin
    send (nexttoken) ;
    nexttoken := next ;
  end
  if (nexttoken <$ 80) then          { data byte }
  begin
    send (nexttoken) ;
    nexttoken :=next ;
  end
  else
      error ;
  end
  else
    error ;

end ;

procedure SysCommonMsg ;
begin
  if (nexttoken=$f6) then
  begin
    send(nexttoken) ;
    nexttoken :=next ;
  end
  else
    if (nexttoken=$f2) then
        syssongpos
    else
        if (nexttoken=$f3) then
          sysselect
        else
          error ;
end ;

Procedure Metaevent ;          { handle the different }
var length : byte ;            { meta events }
    i : integer ;
    tempo : longint ;
begin
  nexttoken :=next ;
  Case nexttoken of
                                   { title }
  3 : begin
      length :=next ;
      for i ;=0 to length-1 do Write(Chr(next)) ;
      Writeln ;
      nexttoken :=next ;
    end ;
$2f : begin                        { end of track }
```

```
        nexttoken :=next ;
        If nexttoken=0 then stop :=1 ;
      end ;
$51 : begin                          { tempo }
        nexttoken :=next ;
        If nexttoken=3 then
        begin
          Writeln('Tempo change') ;
          nexttoken :=next ;
          tempo :=nexttoken*65536 ;
          nexttoken :=next ;
          Inc(tempo, nexttoken*256) ;
          nexttoken :=next ;
          Inc(tempo, nexttoken) ;

          tempo :=tempo div clocktick ;
          tempo :=tempo div 12 ;
          settimspeed(tempo) ;
          nexttoken :=next ;
         end ;
        end ;
$54 : begin                          { time stamp }
        nexttoken :=next ;
        if nexttoken=5 then
        begin
          writeln('SMPTR") ;
          for i :=0 to 5 do nexttoken :=next ;
        end ;
     end ;
$58 : begin                          { time signature }
        nexttoken :=next ;
        if nexttoken=4 then
        begin
          writeln("Time sign.") ;
          for i :=0 to 4 do nexttoken :=next ;
        end ;
      end ;

else
  begin
    writeln('Unknown Meta event', next) ;
    Repeat
      nexttoken :=next ;
    Until nexttoken<$80 ;
  end ;
 end ;
end ;

Procedure Sysmsg ;
```

```
begin
  if (nexttoken >= $f8) then
  begin
    if (nexttoken = $ff) then MetaEvent ;
  end
  else
   if (nexttoken = $f0)
   then
    sysexmsg
  else
    SysCommonMsg ;
end ;

Procedure Chan1ByteMsg ;
begin
  send (nexttoken) ;
  nexttoken :=next ;
  while (nexttoken>=$f8) do          { real-time messages }
  begin
    send(nexttoken) ;
    nexttoken :=next ;
  end ;
  if (nexttoken < $80) then          { data byte }
  begin
    send (nexttoken) ;
    nexttoken :=next ;
  end
  else
   Error ;
end ;

Procedure Chan2ByteMsg ;
begin
  send(nexttoken) ;
  nexttoken :=next ;
  while (nexttoken>=$f8) do          { real-time messages }
  begin
    send(nexttoken) ;
    nexttoken :=next ;
  end ;

if (nexttoken<$80) then        { data byte }
begin
  send(nexttoken) ;
  nexttoker :=next ;
  while (nexttoken>=$ft8) do         { real-time messages }
  begin
    send(nexttoken) ;
    nexttoken :=next ;
```

```
  end ;
  if (nexttoken <$80) then  { data byte }
    begin
      send(nexttoken) ;
      nexttoken :=next ;
    end
   else
    Error ;
  end
  else
    Error ;
end ;

Procedure Chanmsg ;
begin
  if ((nexttoken>=$CO) AND (nexttoken<$E0))
  then
    chan1ByteMsg
  else
    chan2ByteMsg ;
end ;

procedure MIDImsg ;
begin
  if (nexttoken >= $80) then
    if (nexttoken >= $F) then
      sysmsg
    else
      chanmsg
  else
  begin                    { running status byte }
    send(nexttoken) ;
    nexttoken :=next ;
    if (nexttoken < $80) then          { data bytes }
    begin
      send(nexttoken) ;
      nexttoken :=next ;
    end
    else
      error ;
  end ;
end ;

Procedure ReadVarLength ;          { calculate number of }
var value : longint ;              { clock ticks }
begin
  value :=nexttoken ;
  if (value and $80) =$80 then
  begin
```

```
    value :=value AND $7f ;
    Repeat
      nexttoken :=next ;
      value :=(value sh1 7)+(nexttoken AND $7f) ;
    Until((nexttoken AND $80)<>128) ;
  end ;
  nexttoken :=next ;
  counter ;=value ;
  while (counter>0) do ;
end ;

procedure MIDIstream ;
begin
  readvarlength ;
  MIDImsg ;
end ;

procedure VmidiHeader ;              { determine beginning of the reack }
var i : integer ;
begin
  if Readpos(9)>0 then
    writeln('This is No format O MIDI file !') ;

  curmempos :=Readpos(4)*65536 ;
  Inc(curmempos, ReadPos(5)*4096) ;
  Inc(curmempos, swap(ReadPos(6))+ReadPos(7)+8) ;
  Inc(curmempos, 8) ;
end ;

Procedure NewTim; interrupt ;
var R : registers ;
begin
  if (counter>0) then dec(counter) ;

  Intr(oldtimV, R) ;
end ;

begin                    { main program }
  checkbreak :=false ;
  stop :=0 ;

  if paramcount<>1 then
  begin
    writeln('Use : playmidi < CMF-/MIDI-file >') ;
    halt(1) ;
end ;

result :=loadfile(paramstr(1)) ;
```

```
  if result<>0 then
  begin
    if(result=2) then
      writeln('Not enough memory available.')
    else
      writeln('Loading error ', paramstr(1), '.') ;
  halt(2) ;
end ;

formattype :=check ;
  if formattype=255 then
  begin
    writeln('No MIDI or CMF format .') ;
    halt(3) ;
  end ;

  if formattype=1 then                { read speed }
  begin
    clocktick :=ReadPos(13) + swap(ReadPos(12)) ;

  end
  else
  begin
    clockick :=ReadPos($Oc) + swap(ReadPos($Od)) ;
  end ;

  settimer(clocktick, @newtim) ;

  resetDSP ;

  if formattype=1 then                { read offset }
    Vmidiheader
  else
    curmempos :=Readpos(8)+swap(Readpos(9)) ;

  Nexttoken :=next ;

  While (stop<>1) and not keypressed
   do MIDIstream ;                    { play on }

  if keypressed then ch :=readkey ;


  resettimer ;
  Writeln('That's all...') ;
  freemem(mempos, size) ;
end .
```

程序清单10.6： 用C实现的CMFMIDI播放程序

```
/* ********************************************
**  MIDICMF.C                                 **
** Turbo C++ 2.0                              **
******************************************** */

#pragma inline
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <dos.h>
#include <stdlib.h>
#include <string.h>

#define port 0x220    /* base port */

#include "midi.h"
void *here ;
unsigned char *header ;
unsigned char *pointer ;
unsigned char nexttoken ;
unsigned int clocktick ;
int stop=0 ;
long counter=0 ;


loadfile (char *filename)      /* loads file */
{                              ?* into memory */
  int handle ;
  long filesize ;
  int number_of_bytes ;

  if ( (handle=open(filename, O_RDONLY | O_BINARY))==-1)
    return(1) ;
  filesize = filelength(handle) ;
  if ((here=malloc(filesize))==NULL)
    return(2) ;
  if ((read(handle, here, filesize))==-1)
    return(1) ;

  close(handle) ;
  return(0) ;
}

check()        /* determine the song format */
{
  int j ;
  char *type1="MThd", *type2="CTMF" ;
```

```
  header=(unsigned char *)here ;
  j=*(header+4) ;
  *(header+4)=0 ;
  if ( (strcmp(header, type2)==0) ) /* CMF ? */
  {
    *(header+4)=j ;
    return(0) ;
  }
  else
    if (strcmp(header, type1)==0)   /* MIDI ? */
    {
      *(header+4)=j ;
      return(1) ;
    }
    else
      return(-1) ;
}

void error()
{
  printf("\nWhat you say is nonsense...\n") ;
  printf("%\n", nexttoken) ;
  exit(1) ;
}

void send(unsigned char token)
{
  writedat(ox38) ;    /* command MIDI write */
  writedat(token) ;   /* data byte */
}

unsigned char next() /* read next byte */
{
  unsigned char k ;
  k=*pointer ;
  pointer++ ;
  return(k) ;
}


void sysexmsg()       /* a system exclusive message */
{
  send(nexttoken) ;           /* consists of : */
  nexttoken=next() ;
  while (nexttoken>=0xf8)   /* possible real-time mesg. */
  {
    send(nexttoken) ;
    nexttoken=next() ;
  }
```

```
  if (nexttoken < Ox80)     /* 1 data byte */
  {
     send(nexttoken) ;
     nexttoken=next() ;
  }
  else
     error() ;
  while(nexttoken !=0xf7)   /* terminated with EOX */
  {
     while (nexttoken>=0xf8) /* real-time m,essages */
     {
        send(nexttoken) ;
        nexttoken=next() ;
     }
     while(nexttoken < 0x80)        /* data bytes */
     {
        send(nexttoken) ;
        nexttoken=next() ;
     }
  }
  send(nexttoken) ;
  nexttoken=next() ;
}

void sysselect()
{
  send(nexttoken) ;
  nexttoken=next() ;
  while (nexttoken>=0xf8)   /* real-time messages */
  {
     send(nexttoken) ;
     nexttoken=next() ;
  }
  if ( nexttoken < 0x80)    /* data byte */
  {
     send(nexttoken) ;
     nexttoken=next() ;
  }
  else
     error() ;
}

void syssongpos()
{
  send(nexttoken) ;
  nexttoken=next() ;
  while(nexttoken>=0xf8)    /* real-time messages */
  {
     send(nexttoken) ;
```

```
    nexttoken=next() ;
  }
  if (nexttoken < 0x80)      /* data byte */
  {
  send(nexttoken) ;
  nexttoken=next() ;
  while (nexttoken>=0xf8)   /* real-time messages */
  {
    send(nexttoken) ;
    nexttoken=next() ;
  }
  if (nexttoken < 0x80)      /* with data byte */
  {
    send(nexttoken) ;
    nexttoken=next() ;
  }
  else
    error() ;
}

void SysCommonMsg()
{
  if (nexttoken==0xf6)
  {
    send(nexttoken) ;
    nexttoken=next() ;
  }
  else
    if (nexttoken==0xf2)
   sysselect() ;
    else
      error() ;
}

void metaevent()      /* handles the different */
{                             /* meta events */
  unsigned char length ;
  int i ;
  long tempo ;

  nexttoken=next() ;
  switch(nexttoken) {

  case 3:        length=next() ;          /* title */
              dor (i=0 ; i<length; i++)
              putc(next(), stdout) ;
              nexttoken=next() ;
              break ;
```

```
case 0x2f :     if ((nexttoken=next())==0)
                stop=1 ;                /* end of track */
                break ;

case 0x51 :     if ((nexttoken=next())==3)          /* tempo */
                printf("\nTempo change") ;
                tempo+=((nexttoken=next())*65536) ;
                tempo+=((nexttoken=next())*256) ;
                tempo+=(nexttoken=next()) ;
                tempo/=clocktick ;
                tempo/=12 ;
                settimspeed(tempo) ;
                  nexttoken=next() ;
                break ;

case 0x54 :     if ((nexttoken=next())==5)
                printf("\nSMPTE") ;  /* time stamp */
                for (i=0 ; i<6; i++)
                nexttoken = next() ;
                break ;

case 0x58 :     if ((nexttoken=next()==4)
                printf("\nTime sign.") ;
                for (i=0 ; i<5 ; i++)          /* time signature */
                nexttoken=next() ;
                break ;

default :       printf("\nUnknown Meta event %d", next()) ;
                while((nexttoken=next())<0x80) ;
                break ;
        }
}

void sysmsg()
{
  if (nexttoken >= 0xf8)
  {
    if (nexttoken == 0xff)
    metaevent() ;
  }
  else
   if (nexttoken == 0xf0)
       syssongpos() ;
   else
       if (nexttoken) == 0xf3
       sysexmsg() ;
   else
      SysCommonMsg() ;
}
```

```
void chan1ByteMsg()
{
  send(nexttoken) ;
  nexttoken=next() ;
  while(nexttoken>=0xf8)          /* real-time messages */
  {
    send(nexttoken) ;
    nexttoken=next() ;
  }
  if (nexttoken < 0x80)      /* with data byte */
  {
    send(nexttoken) ;
    nexttoken=next() ;
  }
  else
    error() ;
}

void chan2ByteMsg()
{
  send(nexttoken) ;
  nexttoken=next() ;
  while(nexttoken>=0xf8)          /* real-time messages */
  {
    send(nexttoken) ;
    nexttoken=next() ;
  }
  if (nexttoken < 0x80)           /* with data byte */
  {
    send(nexttoken) ;
    nexttoken=next() ;
    while(nexttoken>=0xf8)        /* real-time messages */
    {
      send(nexttoken) ;
      nexttoken=next() ;
    }
    if (nexttoken <0x80)          /* with data byte */
    {
      send(nexttoken) ;
      nexttoken=next() ;
    }
    else
      error() ;
  }
  else
    error() ;
}
```

```
void chanmsg()
{
  if (nexttoken >= 0xCO && nexttoken < 0xE0)
      chan1ByteMsg() ;
  else
    chan2ByteMsg() ;
}

void MIDImsg()
{
  if (nexttoken >= 0x80)
    if (nexttoken >= 0xF0)
      sysmsg() ;
    else
      chanmsg() ;
  ekse                          /* running status */
  {
    send(nexttoken) ;           /* data byte */
    nexttoken=next() ;
    if (nexttoken < 0x80)       /* followed by 0 or 1 */
    {                           /* data bytes */
      send (nexttoken) ;
      nexttoken=next() ;
    }
    else
      error() ;
  }
}

void readvarlength()  /* read number of clock ticks */
{
  long value ;

  if ((value = nexttoken) & 0x80)
  {
    value &= 0x7f ;
    do
    {
      value=(value << 7)+((nexttoken=next()) & 0x7f) ;
    } while (nexttoken & 0x80) ;
  }
  nexttoken=next() ;
  counter=value ;
  while(counter>0) ;
}

void MIDIstream()
{
```

```
  readvarlength() ;
  MIDImsg() ;
}

void Vmidiheader()   /* determine offset to the data */
{
  if ((*(header+9))>0)
      printf("\nThis is NO format 0 MIDI file !\n") ;
  pointer=header+((*(header+4))&65536) ;
  pointer+=((*(header+5))*4096) ;
  pointer=pointer+((*(header+6))&256)+(*(header+7))+8 ;
  pointer+=8 ;
}

void interrupt newtim()       /* decrease counter */
{
  if (counter>0)
  counter-- ;
  oldtim() ;
}

main (int argc, char *argv[])
{
  int result, type, tel ;
  if ( !(argc==2) )
  {
    printf("\nUse : playmidi < CMF-/MIDI-file > \n") ;
    return(-1) ;
  }

  if (!((result=loadfile(argv[1]))==0 ))
  {
    if (result==2)
      printf("\nNot enough memory available.\n") ;
    else
      printf("nLoading error %S.\n", argv[1]) ;
    return(-2) ;
}

  if ((type=check())==-1)
  {
    printf("\nNo MIDI or CMF format.") ;
    return(-3) ;
  }
  if (type )   /* determine speed */
    clocktick=( *(header+13) ) + ( (*(header+12))*256) ;
  else
    clocktick=( *(header+0x0c) )+( (*(header+0x0d))*256) ;
  settimer(clocktick, newtim) ;
```

```
    resetDSP() ;

    if (type)                    /* offset to the start */
      Vmidiheader() ;
    else
      pointer=header+(*(header+8))+((*(header+9))*256) ;
    nexttoken=next() ;

    while (!stop && !kbhit())
    MIDIstream() ;

    if (kbhit())getch() ;

    resettimer() ;
    printf("\nThat's all...") ;
    free(here) ;
    return(0) ;
}
```

第十一章　Sound Blaster Pro的混合器芯片

本章内容为：

编程端口2X4H和2X5H
定义混合器芯片的寄存器
为声道设置音量
滤波器和其它设置

至此你已知道，Sound Blaster Pro的功能要比Sound Blaster强许多，大部分功能在前面章节已经讨论过了。这一章将讨论一些新的功能，特别是涉及混合器芯片所支持的一些功能。

混合器芯片的作用

混合器芯片处理进入的和出去的信号，以及混合所有信号成为一个输出信号。它处理从CD-ROM、线入、麦克风来的三种输入信号。这样，你能直接听到正在记录的内容。

由于有了混合器芯片，你可以对每部分的音量进行调节。而且，这个芯片还含有影响记录和播放的一些设置。可以通过端口2X4H和2X5H对其编程（X=2或4，这取决于基本端口）。

编程端口

端口2X4H和2X5H的编程与它们在FM部分的编程方式相同。把选定的某个寄存器的字节（混合器芯片所用的变量）写到2X4H，将这个寄存器的值作为一种索引，然后，该寄存器的内容可通过端口2X5H读出，也可将一个新的值赋给该寄存器，如像下面伪码所示：

```
[1]    OUT  2X4H，3
[2]    IN   D，2X5H
[3]    OUT  2X5H，9
[4]    OUT  2X4H，7
[5]    OUT  2X5H，D
```

以上寄存器完成了下述动作：

寄存器	用　途
1	选择寄存器3
2	如果寄存器1通过端口2X5H读，该寄存器的现行值被赋给变量D
3	此寄存器装入新值9

4	选择寄存器7
5	用寄存器3的值装入寄存器7

执行完该程序后，寄存器3中的值为9，寄存器7中为寄存器3的老值。

注解　下面的“设置音量”讨论为混合器芯片所定义的各寄存器。

你可用缺省值对所用寄存器进行初始化，混合器芯片被复位就发生于此时。为此，首先你需对端口2X4H写入0，程序必须等待一短时间（大约0.5微秒），然后你必须写个0到2X5H端口。芯片被复位以后，所有寄存器中都为其缺省值，原来旧的设置均被覆盖。

建立音量设置

每个音量设置存放在一独立的寄存器里，你可以改变每个部分左右两个声道的音量。麦克风输入是唯一没有音量设置的输入，因为麦克风没有立体声能力，所以它没有音量设置。对三种输入信号的音量设置要影响记录，若设置为0，那记录的只是无声。

音量设置寄存器　在表11.1中的所有寄存器值有同样的结构，麦克风部分的值是唯一一个不同于其它的值。

音量寄存器结构　音量能在0（最小值、无声）到7（最大值）之间设置。但是CT-VOICE驱动程序和大多数别的Sound Blaster Pro软件允许你在0-15的范围内设置音量。　之所以能这样做，是因为这些程序用了0位到4位，所以有4位值可被设置。在混合器芯片中，这些位不作音量设置用（它们总是1），大概0位到4位将用在Sound Blaster Pro的下一个版本上。在这种情况下，音量可能在0到15内设置。表11.2示出了音量寄存器的结构。

表11.1　用于音量设置的寄存器

部分	寄存器索引	缺省值
通用的	22H	44H
Voice	04H	44H
FM	26H	44H
CD	28H	00H
Line	2EH	00H
Mic	0AH	00H

表11.2　音量寄存器的结构

位	功能
7、6、5	左声道音量（0...7）
3、2、1	右声道音量（0...7）

麦克风的音量寄存器结构 对麦克风音量的设置采取相同的做法：程序用位2、位1和位0来设置音量，其值从0到7，因位0总是1，所以它不影响音量。麦克风音量寄存器的结构如表11.3所示。

表11.3 麦克风音量寄存器的结构

位	功能
2、1	麦克风声道音量（0...3）

在不改变另一声道情况下设置一个声道的音量

如果想在不改变另一声道的音量情况下设置左或右声道的音量，很显然，应该使用“5”或“6”和“移位”功能。当你在设置麦克风的音量时，使用“或”和“移位”也是很可取的，因为Sound Blaster Pro将来的版本可能用寄存器0AH的其它各位，而且不能装入一个固定值。

设置音量的一般算法是：

1. 选择寄存器。
2. 读旧的值。
3. 使用“与”清除旧值。
4. 如果需要，把新值移位。
5. 用“或”设置新的值。
6. 把新值写到寄存器。

程序11.1，11.2和11.3是一短小程序的例子，程序是在不改变右声道音量设置的情况下，把左声道音量设置为通常的值6。对2X4H和2X5H，X是所选的基端口；你需用正确的设置作替换。

程序清单11.1： 汇编举例

```
MOV     AH, 6
MOV     DX, 2X4H    ; step 1
MOV     AL, 22H
OUT     DX, AL
INC     DX          ; step 2
IN      AL, DX
AND     AL, OFH     ; step 3
MOV     CL, 4       ; step 4
SHL     AH, CL
OR      AL, AH      ; step 5
OUT     DX, AL      ; step 6
```

程序清单11.2： Pascal举例

```
Var
  OldV, NewV, Help : Byte ;
Begin
  Port[$2X4] :=$22 ;          { step 1 }
  OldV :=Port[$2X5] ;         { step 2 }
  NewV :=OldV And $OF ;{ step 3 }
  Help :=6 shl 4 ;            { step 4 }
  NewV :=NewV Or Help ; { step 5 }
  Port[$2X5] :=NewV ;
End ;
```

程序清单11.3： C的举例

```
{
  int OldV, NewV, Help ;
  outp(0x2X4, 0x@@) ;       /* step 1 */
  OldV = inp(0x2X5) ;       /* step 2 */
  NewV = OldV & 0xOF ; /* step 3 */
  Help = 6 <<<< 4 ;         /* step 4 */
  NewV = NewV | Help ; /* step 5 */
  outp(0x2X5, NewV)         /* step 6 */
}
```

滤波器及其它设置

除了各种音量设置以外，在混合器芯片里，其它一些设置也是做得到的。它们用三个寄存器：06H，0CH，0EH来设置。

寄存器06H 寄存器06H确定如何重现FM通道，其设置如表11.4所示。

表11.4 确定FM通道

位	功能	设置
6	右通道	0=通（开）
		1=断（关）
5	左通道	0=开
		1=关

如果位6、位5皆被复位（=0），则重现常规的立体声。假如两位中之一位为1，所有FM通道都通过另一声道重现。例如，当设置位6=1、位5=0时，每样都通过左声道被重现。而当两位都为1时，则无FM重现发生。这样，你可以在不改变音量的情况下，关掉FM通道。

寄存器0CH 寄存器0CH包括三种和采样记录有关的功能，这些功能和设置示于表11.5中。

表11.5 用于采样记录的寄存器

位	功能	设置
5	声音输入滤波	0=开
		1=关
3	高低通滤波	0=高
		1=低
2，1	声音输入选择	0=麦克风输入
		1=CD-ROM
		2=未用
		3=线入

由位5决定是否用位3。位3选择滤波器型式，位2和位1决定采样记录的输入。虽然能同时听到几个输入信号，但三个信号中只有一个能被内部记录下来。所以记录线入和CD-ROM输入的混合信号的采样是不可能的。

寄存器0EH 寄存器0EH包含两个功能，如表11.6所示。

位1同时用于记录和播放。位5决定是否使用输出滤波器，你不必使用“与”或“移位”功能来改变各种设置。Sound Blaster Pro将来的版本可能用某些还未被用的位，所以建议不要改变这些位的值。

表11.6 具有两个功能的寄存器

位	功能	设置
5	输出滤波器	0=通
		1=断
1	立体声/单声选择	0=单声
		1=立体声

上一节描述的算法也适用于各位的设置。

最后，想必你已注意到，对混合器芯片编程相当简单，这是由于每种设置是以同样方式编程。

附录A Sound Blaster 16的硬件和软件

附录A将使你了解声霸更多的能力，将向你介绍如何安装这块卡，看一下随Sound Blaster 16带来的软件和怎样安装该软件，并对Sound Blaster 16用的第三方的软件和硬件作一简洁的描述。本附录还将介绍你的系统如何配置Sound Blaster 16。

关于Sound Blaster 16的概述

Sound Blaster 16（SB16）是Creative Labs公司出品中的最好的声音卡，这个16指的是Sound Blaster 16能记录和播放16位以及8位的数字声音。

ASP芯片 声霸自称为具有先进的信号处理（Advanced signal Processing-ASP），Creative Labs公司就将其专有的数字信号处理芯片命名为ASP。一块ASP芯片能实现许多算法，例如，它能创建各种声音效果，这是由硬件的信号处理部件创建的，包括混响、延时、均衡、动态限制、压缩和展开。不过，产生如此先进的声音效果现在只是一种“潜能”，因为提供这些先进声音效果算法的软件还没写成，迄今，ASP芯片仅被用于实现数据压缩。换句话说，该芯片可用比正常情况下小的空间来存贮数字的声音。

注释 不要将数据压缩和动态声音压缩相混淆，动态声音压缩指控制声音的动态范围的一种技术。

怎样制作声音样本 Sound Blaster 16能够采样声音，即把声音转换成数字数据，并且以4KHZ到44.1KHZ的速率记录在硬盘上。SB16采样声音是通过一个1/8英寸小型插头麦克风输入（这个卡配有一只兼容的麦克风），或者通过它的1/8英寸立体声连线输入。它还能从其FM合成器或者配接的CD-ROM驱动器采样声音。

板上合成器 Sound Blaster 16有一板上合成器，它是以雅玛哈OPL-3四操作器的合成器芯片为基础构成的。虽然此芯片提供20种声部的百音盒和20个多音色部件，但它有一个很受限制的声音调节板，除了简单的游戏声音外，用起来比较困难。

Wave Blaster Sound Blaster 16具有一个出色的合成器特性：它能和Creative Labs公司的Wave Blaster子板相连。实际上，Wave Blaster给了SB16 E-mu Proteus样本-播放合成器所具有的所有能力。这块卡价值差不多比Sound Balster本身要贵200美元。不过，Wave Blaster提供255种出自Creative Labs公司声音库的采样声音。用Wave Blaster，可将SB16变成一个灵活、出色的声音合成器。

Sound Blaster16的MIDI接口 Sound Blaster16有个MIDI接口，该接口支持MPU-401哑UART模式和Sound Blaster标准模式。其MIDI的功能是要允许PC机应用程序用于与外部的MIDI设备以及其它的设备通信。例如，使用一正在运行的音序器应用程序，你能够弹奏MIDI的键盘并具有音序器记录性能，尔后，你可以用SB16的FM合成器，Wave Blaster、外部键盘或者这三种的任何组合来播放它。

但是，在你使用MIDI接口与外接MIDI设备或者其它设备通信以前，需要把选用的MIDI套件接到SB的游戏杆端口上。

音频输出 为了音频输出，Sound Blaster16 ASP有一小型插孔输出，它可接耳机或者立体声放大器插头。其输出可配置成可变的，或者每声道4瓦，或者线一级的输出。SB能驱动一功率放大器或无源的4欧姆的扬声器，可用在卡后背板上的拨盘来设置最后的输出音量，板上还有音频和数字连接器，使其能接CD-ROM驱动器。

音频混合器 最后，SB16有一音频混合器，可使用成组的Windows applet，SB16混合器，或者SB16SET和SB16MIX DOS应用程序来控制此混合器。该混合器可以接收来自卡的所有输入源一线入、麦克风、合成器，CD-ROM驱动器以及PC扬声器的声音，并把这些声音组合成单一的立体声输出。当音频混合器在组合这些声音时，可以改变所有声源的相对音量级别，以及升高或降低它们的基音、高音或输出增益。

Sound Blaster16的软件 SB16卡的软件有用于DOS和Windows的两种。大多数DOS软件是为有兴趣于设计自己使用SB16的软件的开发者而提供的，但有一些DOS软件在一定程度上是面向用户的。Windows的SB16软件一般有更多的灵活性和易于使用。它应用的重点更侧重于娱乐、多媒体、音乐和OLE等方面。

安装Sound Blaster16卡

这部分讲述如何安装Sound Blaster16硬件，其步骤如下：

- 确信硬件正确无误
- 打开计算机
- 正确地操作这块卡
- 把Wave Blaster接到SB上（如果有Wave Blaster并且决定要装它）
- 改变缺省设置为所希望的设置
- 把SB16卡插入槽内
- 连接游戏杆，立体声/音频，麦克风，线路输入和CD-ROM驱动器

现将每一步骤详细描述如下。

Sound Blaster16的硬件需求

在你安装Sound Blaster16之前——实际上是在买它之前，确信你的PC机和用于卡的外设是正确的。

- 对PC的需求 Sound Blaster16可用在现今大部分型号的PC机上，诸如IBM AT，286，386，486；PS/2（型号25/30）；Tandy AT；或者这些型号的任何一种100%兼容的机器。
- 监视器 有VGA模式监视器是最好，EGA的也行。
- 磁盘空间 至少需要5MB磁盘空间供SB软件所用。
- 存储器 至少要配置2MB的RAM。
- 运行环境 如果具有Windows 3.1是很好的事，但并不绝对需要Windows 3.1，因为如果你只打算玩游戏则它会建立它自己的环境，用Windows 3.0就行了。但如果想用SB

作多媒体或创建音乐，则用Windows 3.1合适得多。

假如PC机符合这些要求，就可准备安装Sound Blaster16卡了。

警告　不推荐SB用Windows 3.0。

打开你的计算机

为了插卡，首先是要打开计算机：

1. 关闭PC机电源，但不必拆卸其电源线—你需要PC保持接地。
2. 旋去机盖的螺丝。如有需要，可参阅机器的手册。
3. 慢慢地取出PC机的外盖，小心勿弄断任何电缆。
4. 在PC机后面板的内面，拧开螺丝，取下任一未用的16位插槽的盖片，这即是SB16卡的背条的开口。注意不要弄丢了螺丝，以后还需用它把SB卡固定在PC机中。

提示　16位插槽是比较长的那种，8位插槽差不多是16位插槽的一半。

操作SB16卡的忠告

现在，你随时可以安装SB16卡了，但在安装前请阅读下列的注意点，以避免此卡和计算机遭损坏：

- 在你拿和移动此卡时，保持它在它们保护袋中。
- 拿卡时总是拿其边缘，这样不致于弄弯元器件。
- 在接触此卡前，或者触及PC机内的电源或者接触一些未上漆的金属表面，以释放掉你人体可能带的静电。
- 避免你的脚在地毯上摩擦，摩擦衣服，或作任何可能产生静电的动作。

遵守了这些预防措施，SB卡和PC机就安全了。

警告　避免人体静电，就减少了将静电传输到SB的可能性，如果当卡接触PC插槽时有静电放电，那么卡、甚至计算机的部件都可能会被损坏。

将Wave Blaster与SB相连

如果有一Creative Labs公司的Wave Blaster，现在是将它连到Sound Blaster16的时候了。假若没有Wave Blaster，则跳到下一节“SB16的缺省设定及如何改变”。本节是讲述如何将Wave Blaster连接到SB上。

注解　假如你已安装好了SB卡，现想将Wave Blaster与它相连，在连接之前，卸下所有连在SB上的外接电缆。当你从PC插槽中取出SB卡时，一定遵循前面的“操作SP16卡的忠告”。然后，按下述操作将Wave Blaster连到SB上。

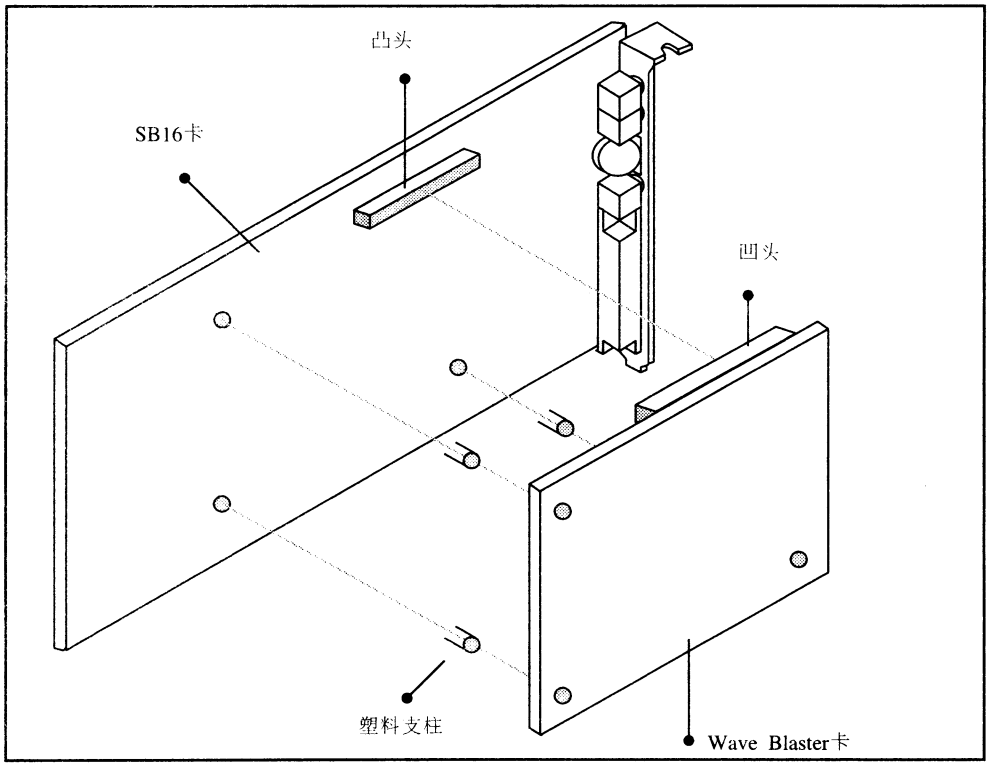
为了实现Wave Blaster与SB相连，需要：

1. 把SB16上的三个塑料支柱插入Wave Blaster卡的孔内，如图A.1所示。
2. 把Wave Blaster安放在SB卡上。做法是将这三个塑料柱压进Wave Blaster卡上的孔内，

然后，将Wave Blaster卡上的凹座压入SB的凸头。

提示　在将两块卡连接时，压入的力量比所想象的可能要大些，但如果仔细地将塑料柱和插头严格对位，再施以平稳的压力就能将两块板可靠在结合在一起了。

当Wave Blaster卡安装在SB16上了后，就可准备进入到下一步骤，考虑是否使用SB的缺省设置。



图A.1　Wave Blaster卡与Sound Blaster16卡的连接

SB16的缺省设置及其如何改变

就象所有的扩展卡一样，SB有某些缺省设置使其与PC以一定方式工作。当然，你可以通过重新组合卡上的硬件来改变这些设置，使其适合你的需要。此外，SB16也允许你用软件方法改变很多缺省设置。

下表列出了SB16可设置的一些参量。除了最后两条只能用硬件进行修改外，其它所有各项均可通过软件或硬件改变。

- IRQ（缺省5）
- 8位DMA（缺省1）
- 16位DMA（缺省5）
- I/O基地址（缺省220H到233H）
- MIDI端口I/O基地址（缺省330H到331H）

- 游戏杆端口（允许/禁止）
- 音频输出（放大或者线电平）

只有三种理由才需改变SB16的出厂缺省值：

- 当你想用已安装在PC机上的游戏杆端口去替换在SB16上的那个时。
- 当你计划把SB16的音频输出与任一类放大器相连——包括有源扬声器——这与连到普通的无源的扬声器相反。
- 如果一个无关设备连到PC机上，例如打印机或调制解调器，已经用了SB16的IRQ，DMA，或I/O地址设置中的一种，该设备将与SB16相冲突。如你认为这样会发生冲突的话，可以在安装SB16之前改动跳接器以改变它的设置。

下面讨论这三种缺省设置。假如你打算保持SB16的缺省设置而不计划改变它们的话，可以跳到后面的“将卡板安置入插槽”。

改变游戏杆和放大器设置

如果你想用别的游戏杆控制器卡，必须使SB16的游戏杆断开，不用这个游戏杆将不影响游戏杆端口的MIDI操作。假设你想用一个外接的放大器，就得旁路SB16的内部放大器，旁路内部放大器将减少失真，给MIDI和音频以更大的听域空间。

注解　关于IRQ、DMA、I/O地址的设置在附录B有介绍，如果有对这些设置不清楚之处，请参见该附录。

为了使游戏杆断开和旁路内部放大器，就必须改接SB16卡上跳接器位置。跳接器是连接在卡上的一对针上的小的塑料和金属帽，很容易将它们从一对针移到另一对针。对游戏杆来说，得改变一个跳接器，而对放大器，则需改变两个跳接器。对于这些的缺省值，不能用软件来改变。

如何移动跳接器　一般来说，移动跳接器最好的方法是：

1. 将卡平放在一软的绝缘物面上（在桌上的毛巾就好）。
2. 用一把好的尖嘴钳，轻轻地直拉每个跳接器，使其从针上脱开。
3. 如果需要，则将这跳接器插到你所希望插的一对针上，用钳子或手指垂直压入。

警告　在移动跳接器时，小心别弄弯任何针或者卡上的元器件，一定要触及某些接地的金属物体以便释放静电。

断开游戏杆端口　为了使SB16的游戏杆端口失效，可这样实现：

- 在所示图A.2中找出跳接器JYEN，将它拆下来。

不需改接此跳接器，仅仅不用它即可，但将其保存，以备后用。

旁路内部放大器　为了旁路内部放大器，用跳接器的作法：

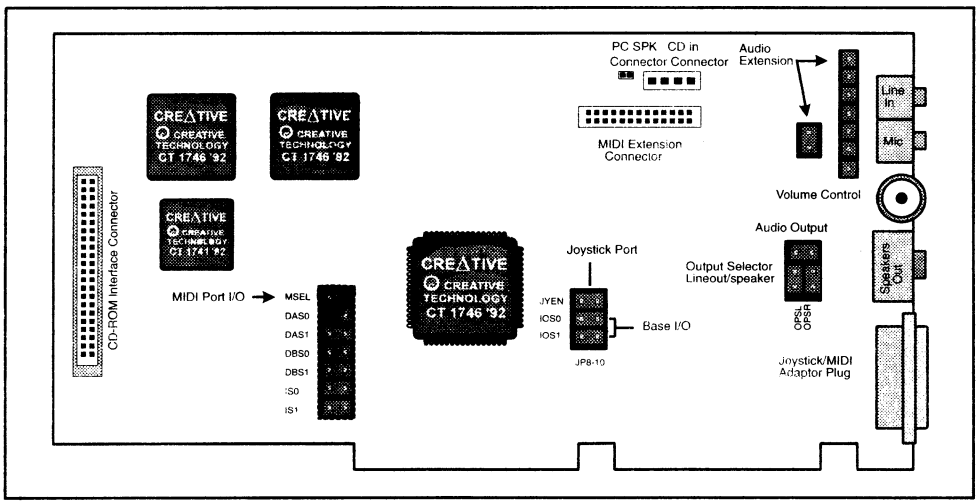
1. 在所示图A.2中靠近音频输出旁找到跳接器OPSL和OPSR。
2. 将这两个跳接器从针2和针3（缺省位置）移插到针1和针2上。

改变游戏杆和放大器缺省值，在安装步骤里是一件十分简单的操作。但是切记，如果在将来你想改变这些设置，一定要拆下所有连线，从PC机上取下这块卡，重走一遍其安装步骤。

这个麻烦是因为上述改变设置的第三个理由——冲突带来的，为避免冲突还是应该不厌其烦。

改变IRQ，DMA和I/O地址的设置

除了游戏杆和放大器的设置以外，还可改变SB16的IRQ、DMA和I/O地址的设置。认识到



图A.2　在SB16卡上游戏杆和音频放大器的跳接器布置

这点是重要的，因为大多数情况下，你不知道SB16是否会和别的一些设备冲突，是否今后需要改变这些缺省值，一直到你安装并使用了SB16后才清楚。多数人甚至不知道IRQ，DMA是什么，即或知道，谁又记得他们的调制解调器的IRQ设置是什么方式呢？

注解　如需要有关IRQ，DMA和I/O地址设置的信息，可参见附录B。

经验告诉我们，最好的对策是用其缺省设置安装该卡，并予试用，观其结果。如果发现了冲突，先不去动跳接器，而通过安装或者设置软件来控制SB16的IRQ，DMA和I/O地址设置。甚至你确认将有冲突，你也可不管跳接器，而在你安装SB16软件时改变它们的缺省值。

通过软件改变DMA，IRQ和I/O地址设置是又容易又快且好查证的办法，用软件设置不需将SB16从PC机中取出后又装入。综上理由，建议你保留硬件在出厂时的配置，待卡安装好后借助软件完成所需的任何改变。

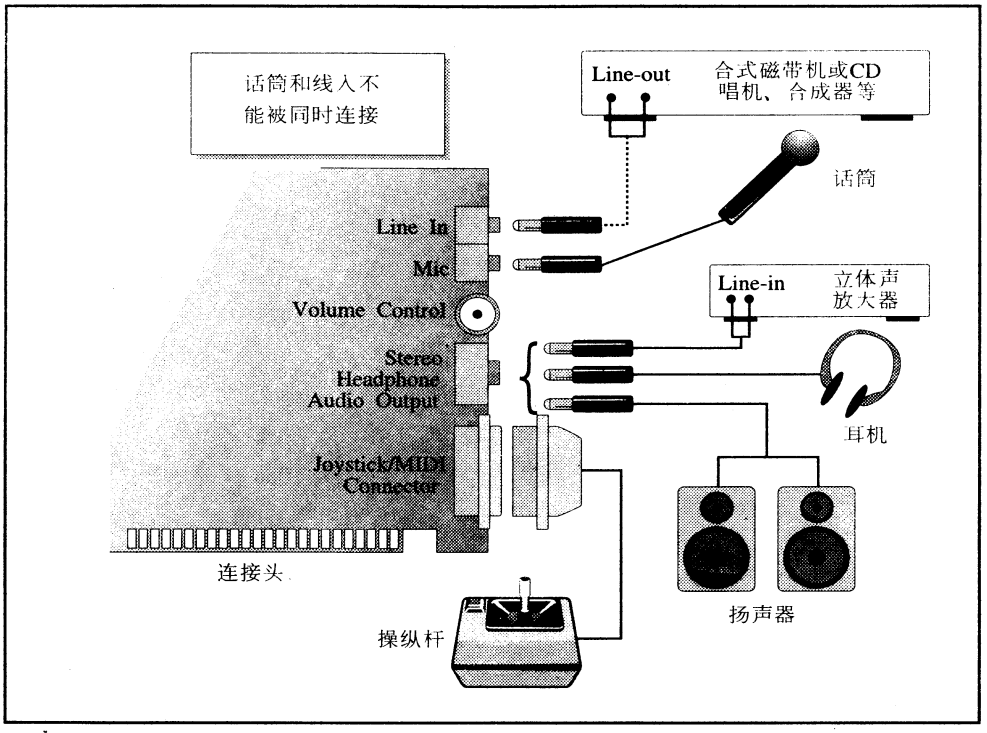
但是，假如你想移改跳接器，希阅读SB卡手册。也阅读附录B，以使你知道你在作什么。

将SB卡插入插槽

当你的游戏杆和放大器已被设成你所需要的值，这时就可将SB16插入插槽内。在卡的一端有一金属片，称为背板条，上有各种插孔。在卡长边的一侧，是连接插头，如图A.3所示，插头为一系列细窄的、金色的、金属矩形条。连接插头是插入PC机扩展插槽中的卡的一部分，通过连接插头上的金色矩形条将卡与计算机在电气上相连。

在PC中安装Sound Blaster16：

1. 触及PC机电源或别的未上漆的金属表面，以释放掉你可能带的静电。
2. 拿起卡，再次触碰电源，确信静电已释放。
3. 将SB16连接插头直接对准你欲插入的插槽，卡的背条应面对PC的后部。
4. 平缓但稳固地将SB的连接插头插入PC插槽。



图A.3　Sound Blaster 16背条连接和连接插头

警告　不要猛力推卡，以免损坏物件。如果卡不易插入，取出来并检查插槽，背板或卡的其它部分，确保没有障碍物，然后再慢慢地试插一次。在这里损坏的东西可能是很昂贵的。

背条应该依靠卡牢固地定位于你先前在PC背面打开的地方。

5. 移动背条的上部，以使它同你先前移去螺钉的孔对准。当你去掉PC盖条时移去此螺钉。
6. 用螺丝把Sound Blaster卡的背条固定在PC上。

硬件安装的最后部分应该是完成把SB16和外部连起来。

游戏杆、立体声/音频、麦克风、线入和CD-ROM的连接

图A.3示出了SB16的背条，从底部到顶部依次是：

- 15针游戏杆D型连接插口
- 立体声输出，1/8英寸小型插孔
- 音量调节拨盘
- 音频输入，先是接麦克风后是线入，均为1/8英寸小型插孔

以下是关于如何连接这些端口的简介

连接游戏杆到Sound Blaster

游戏杆端口是一标准端口，可连接任何模拟游戏杆，这些游戏杆均用一个15针的D型插头座连接。Sound Blaster支持所有的用于任何模拟游戏杆的标准PC软件。如果在你的PC机上已经准备好了游戏杆端口，要么就拆下老的一个，要么断开SB16卡上的。

注解　要了解有关使Sound Blaster的游戏杆断开的信息，参看本附录前面已讲述过的“改变游戏杆和放大器设置”。

为了将游戏杆接到Sound Blaster：

- 将游戏杆的D型连接器插入SB16的端口即可。

MIDI配套件　游戏杆端口还可与MIDI配套件相连，这样就能在SB16上增加MIDI IN和MIDI OUT端口。为了使用MIDI和游戏杆：

- 装好MIDI配套件，然后将游戏杆插入MIDI配套件的直通连接器中。

用两个游戏杆　为了在SB16中使用两个游戏杆，需要一条游戏杆分叉电缆。Creative Labs公司销售这种电缆。它的电缆是唯一能保证正常地工作的产品。大概只需额外花很少一笔钱就可得到Creative Labs的样品并免去了和许多不可靠的计算机生意商打交道，这是值得的。

将立体声输出与SB相连

立体声输出是通过一1/8英寸的小型插孔实现与SB相连的，见图A.3所示。用这个插孔，SB能将立体声输出到一对带功放或无功放的扬声器，或立体声放大器，或立体声耳机中。如果你打算使用放大器，一定要正确地设置音频跳接器。否则就只需将无功放扬声器或耳机插入这个插孔中就行了。

注解　若要了解有关设置音频跳接器的信息，可参看本附录前述的“改变游戏杆和放大器设置”。

连接麦克风

你可将随SB16带来的麦克风直接插入卡的麦克风输入插孔，但是，使用SB自带的麦克风的录音质量比较差，几乎所买的任何麦克风都比SB自带的好，这种情况还将持续很长时间（我是这种看法）。你需准备一个适配插头或者电缆，可将除开最劣质的麦克风以外的任何型号的麦克风插入SB16的小插孔。

实践第一，SB麦克风用于语音是完全适合的，如果你只打算用来记录语音，完全可以。但如果用其记录有很宽音域的声音，例如音乐，则声音失真很厉害。

线入

连接线入插孔到任何你想用SB记录的线级声源，包括磁带或CD唱机，收音机，电视机和合成器之类的电子乐器。对SB来说，需要一分叉电缆，以便将来自声源的两个单音插头信号混合成为一个立体声插头输出。

记住，家用设备，如立体声收音机和CD唱机要求RCA插座，而专业用设备，如合成器具有1/4英寸插座，而SB使用一单插脚立体声小插头。当你去商店购置分叉电缆时记住这些变换。

与CD-ROM驱动器相连

为了与一内置的或外挂的CD-ROM驱动器相连，遵循用户手册里的指导，一个CD-ROM驱动器典型地要求有两种连接：

- 一种是连接到SB16的SCSI端口，以便将计算机文件从CD-ROM传送到PC
- 一种是连接到SB16的音频混合器，为使CD的声音（红书音频）从CD-ROM送到SB16

安装SB16的软件

Sound Blaster 16最得意的一件事就是安装和设置Creative Labs公司所包含的用于卡的软件。仅需要用户最小的输入，四个程序完成所有技术繁琐工作，包括重构缺省值、安装DOS和Windows驱动程序、甚至卡的测试。另外，当你完成卡的设置以后，还可以改变IRQ，DMA及I/O地址。如果今后你安装一些新设备导致同SB相冲突时，将体会到可以改变IRQ，DMA和I/O设置的好处。对早期的SB版本，你必须卸下卡重新调节这些设置。

四个软件安装程序是：

INSTALL.EXE	安装程序，这个程序将SB16的软件安装在其硬盘上，并且相应地变更CONFIG.SYS和AUTOEXEC.BAT文件。
SBCONFIG.EXE	构造程序，此程序建立卡的IRQ，DMA和I/O设置。
TESTSB16.EXE	测试程序，此程序测试卡的声音能力。
WINSETUP.EXE	Windows建立程序，该程序安装Windows驱动程序，并修改SYSTEM.INI文件，使SB16在Windows下工作。

你应该已将卡安装在一个扩展槽里，下面讨论如何安装SB16软件和使卡配套。

程序INSTALL，安装SB用

为了运行INSTALL.EXE，遵循下列步骤：

1. 用MS-DOS引导PC机。
2. 将SB16 1#软盘插入软盘驱动器（假定使用B驱动器）。
3. 在DOS的提示符处键入B：INSTALL，然后回车。
4. 遵照程序给你的提示进行。

提示　在你开始安装前，先阅读README文件，以知道新的变动（如果没有，也别失望）。

在INSTALL程序运行时，你应花大部分时间注视其写硬盘过程，如果你遵照了我们的建议且没有改变过任何IRQ、DMA或I/O地址设置，那你可对所有问题都回答“yes”。当你看到屏幕上出现“Software Installation Completed”（软件安装完成）时：

5. 按任意键。
6. 退出程序。

现在，软件已在硬盘里，目录为SB16。你的CONFIG.SYS和AUTOEXEC.BAT文件已经被修改为适合SB16。

程序SBCONFIG，实现IRQ、DMA和I/O地址设置用

构造程序建立这个卡的IRQ、DMA和I/O地址设置。为运行该程序，按照以下步骤：

1. 在DOS提示符处键入CD SB16以转到SB16目录。
2. 在SB16提示符处键入SBCONFIG，并回车。
3. 按程序给出的提示继续进行。

如果你采用缺省值，如程序所说明的，只需按回车或者S再回车，你可以通过按箭头键来改变设置。当程序询问重新启动系统时，回答yes。

现在SB16已建好，可在DOS下工作。如果计算机支持的话，可以测试本块卡。

程序TEST，测试卡的声音能力用

测试程序让你发现你的SB卡能产生什么种类的声音。为了运行这一测试程序，应按照以下步骤：

1. 在DOS的提示符处键入CDSB16，转移到SB16目录。
2. 在SB16的提示符处键入TESTSB16，然后回车。
3. 根据程序给出的提示继续执行。

在它做了一些自身测试之后，程序将会让你知道卡是否真正工作，它通过给你显示一些选项，你可以播放测试卡全部的声音能力的短声音。你可以播放磁盘上四个文件中的任何一个。

- 包括两个MIDI文件。一个MIDI文件使用2-操作器FM合成器，另一个使用4-操作器FM合成器。
- 包括两个数字音频文件。其中一个音频文件播放8位声音，另一个播放16位声音。

选择一项并按回车键来播放一个2-3秒的测试声音。如果你听不到任何声音，检查你所有的接线。

如果你听不到任何声音... 如果你仍然听不到任何声音，那么可能是你的SB16设置与PC机上别的设备的设置相冲突。很不幸运，你得做一些麻烦的工作：

- 在有关手册上查看其它设备的设置，把它们抄写下来，然后看看是否它们的设置与SB16的设置相匹配。如果设置是匹配的，则回到SBCONFIG程序，并开始变更设置。如果你不能确信设置是匹配的，则开始变更SB16设置。首先检查那些IRQ设置，然后是DMA，最后是地址设置。你得改变一种设置，然后检查此卡，变更另一种设置，再次测试此卡，就这样直到SB16能产生声音，或者直到你用尽各选项。
- 如果上述的技术不能使其工作，开始拆下你的PC其它卡，一次一种。首先关掉电源，拔下一卡，重新启动，且再试Test程序，最好的情况是你拔出一个卡之后，SB16就工作了。然后你所必须做的一切是画出卡的设置并改变它们，或者改变SB16的设置，以使两个卡不再冲突。最糟糕的情况是你拔下所有的卡，但发现SB仍不工作，如该情况发生，则在别的计算机上检查该卡，或通知Creative Labs公司，你的这块卡可能有问

题，尽管这种情况不常发生。

提示　一些计算机在播放16位音频时有问题，SB16入门手册很好地写入了解决此问题的过程。

程序WINSETUP，设置Windows驱动程序用

WINSETUP程序安装Windows驱动程序并且修改SYSTEM.INI文件，以使SB16在Windows下工作。

正如我们前面提到的，如果你的PC机上安装了Windows 3.1，你就可以最有效地利用SB16和它的软件。如果你已经使用Windows 3.1，并且在DOS中已经成功地安装和测试了SB，你就可以按下列步骤安装SB16的Windows驱动程序。步骤如下：

1. 启动Windows。
2. 打开Program Manager。
3. 从File菜单中，选择Run。
4. 在会话窗中键入C：\SB16\WINAPPL\WINSETUP，然后按OK。
5. 当程序完成时，重新启动Windows。

安装Wave Blaster软件

如果在你的SB16上安装了Creative Labs公司的Wave Blaster子板插件，你也必须安装它的软件。步骤如下：

1. 在DOS下启动PC。
2. 把Wave Blaster1#盘放入软盘驱动器（我们假定驱动器是B）。
3. 键入B：进B驱动器目录。
4. 键入INSTALL并回车。

安装程序产生一个叫WAVEBLST的目录，并在这个目录里面安装所有运行Wave Blaster所必需的文件。

提示　通过在SB16上安装Wave Blaster子板，你就可以非常有效地改善SB16MIDI的音质。可在本附录的后面参看“The Wave Blaster”，有Wave Blaster的详细讨论。

测试Wave Blaster：　为测试Wave Blaster，运行DEMO.BAT程序：

1. 键入CD WAVEBLST，进入Wave Blaster目录。
2. 在DOS提示符下，键入DEMO并回车。

程序DEMO.BAT将在Wave Blaster上播放一首例曲。

发送MIDI数据到Wave Blaster　为在Windows下使用Wave Blaster，你必须使用窗口MIDI Mapper以从音序器或其它的Wave Blaster（不是OPL-3的）MIDI应用程序以得到MIDI数据。为使Wave Blaster得到MIDI数据，要建立MIDI Mappers：

1. 打开Windows的控制面板。
2. 打开MIDI Mapper图标。

注意MIDI Mapper缺省是对SB16 Ext FM，这意味着设置为使用OPL-3 FM芯片。你需要

改变此：

3. 按下箭头键，移到SB16 Ext FM设置的右边。
4. 从下拉表中，选择Extended MIDI。

现在，由Windows应用程序产生的任何MIDI数据将被发送到Wave Blaster。代替你播放MIDI文件或任何MIDI插曲时听到的OPL-3，你将听到具有高的保真度、动态范围以及其它悦耳音质的被采样乐器的声音。

安装CD-ROM软件

将CD-ROM驱动器安装在你的PC机上与SB16一起工作，开创出许多可能性，所以我们衷心地推荐它。为使你的PC机读CD-ROM数据，你的CD-ROM驱动器所带来的设置程序安装MSCDEX.EXE，并修改CONFIG.SYS文件，这样可使得在启动时将程序MSCDEX.EXE装入。驻留在具有MSCDEX.EXE目录的DOS设备驱动程序告知DOS，CD-ROM驱动器存在。

为了播放CD-ROM音频，你的PC机需要Windows 3.1带来的MCICDA.DRV驱动程序。你能用控制面板和驱动程序图标安装它，这样你将看见它被列出为[MCI]CD Audio。阅读你的CD-ROM和Windows手册，学习该如何安装MCICDA.DRV。

Sound Blaster 16增强

Sound Blaster 16拥有不平常的功能集，这在几年前，无论是任何价格的任何单卡上都是不可能有的。举例来说，SB16的记录和播放16位数字音频的能力为所有各类应用提供了巨大的机会，从产生你自己的记录到享受商品化的多媒体节目。

为利用SB16硬件的优点，当然，你需要软件操作它。SB16配有许多DOS和Windows的程序为卡的潜在创造力服务。然而大部分的DOS程序对开发创造能力是相当有限的和不直观的。至于Windows，一些Windows程序是有用的，但是即使是最好的也不如所想象的好，或缺少能够让你利用卡的全部创造力的充分的功能性。

令人高兴的是已有一些优秀的第三方软件已可为SB16所用。好的MIDI音序器或数字音频记录/编辑应用程序，能帮助用户利用SB16建立音乐和记录数字音频。很多种游戏，多媒体、教育软件及其它程序，也使用了SB16的声音来达到最佳效果。

这就是说，让我们来看看成组的SB应用程序的更多的能力。

Wave Blaster

最糟糕的是SB16使用了讨厌的Yamaha OPL-3合成器芯片。由这种芯片产生的那些声音曾因新奇而被接受，且用于计算机游戏。但是，这些声音在动态、音色、变化和声音复合性等等方面还相当缺乏，使得听起来十分烦人。有鉴于OPL-3的限制，SB16最卓越的方面在于它用Creative Lab的Wave Blaster替代OPL-3。

Wave Blaster（WB） 是一块可选用的SB16子板，它装有相同于用在豪华音响E-mu Proteus的数字采样的4MB存贮器。Proteus是一个专业的样本回放合成器，它的安排很好的声学合成乐器的声音一旋律和打击乐一被应用到各处播音室中。Wave Blaster的板上ROM包含383种不同的声音，提供高达32音符的多音。它能单独地和同时地在所有的16个MIDI通道上响应，

而缺省为General MIDI设备和打击映射。

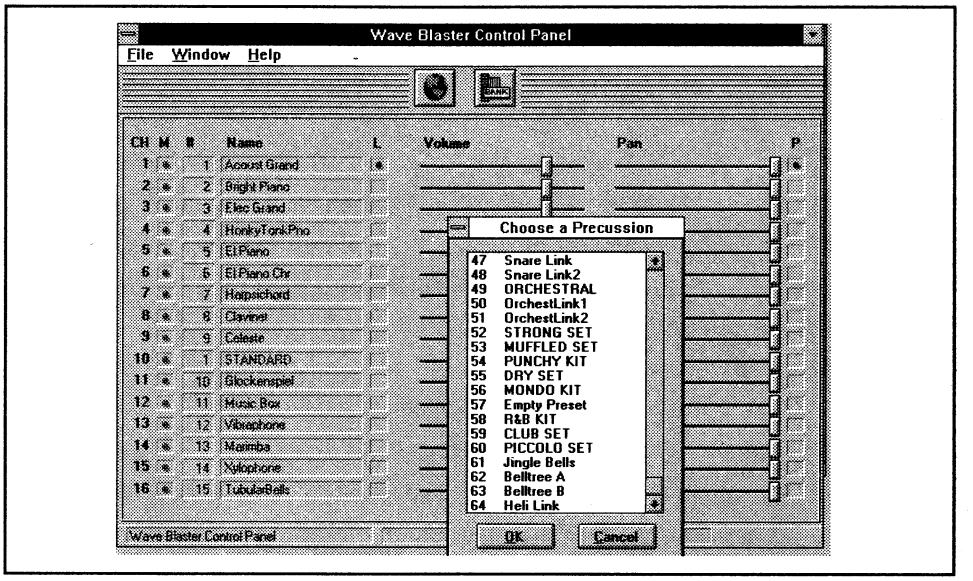
通过在SB16上安装Wave Blaster子板，可以很大地改善你的SB16的MIDI的声音质量。OPL-3芯片还能起作用，但是在你听了Wave Blaster之后，你将决不再想听那些Yamaha声音。

控制面板

在Wave Blaster中，声音是按照General MIDI设备映射来安排的，使它们对于商业应用和电子布告板系统的几千个MIDI文件都有效兼容。另外，你能通过使用Wave Blaster控制板，使Wave Blaster在许多方面符合客户要求，如图A.4所示。在Sound Blaster 16程序组中可找到Wave Blaster控制面板。

下列各项是使用Wave Blaster控制面板的一些基本指南：

- 为了将16个Wave Blaster MIDI通道中的任何一个开或关，在通道左边的盒内触发按钮。
- 为琐定任一给定的声音到指定的MIDI通道，选M列（M：哑的）触发按钮。
- 为设置通道音量和显示位置，方法就是简单地触发按钮并拖动Volume或Pan滑条向左或右。



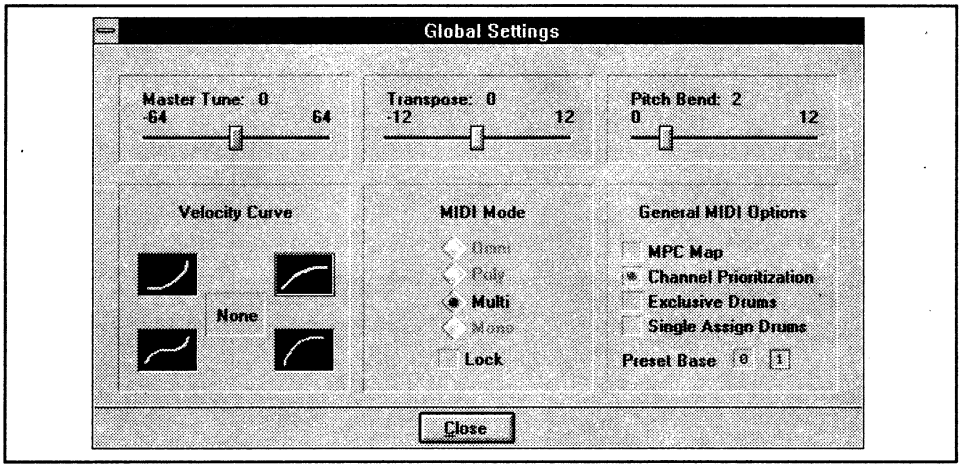
图A.4 控制声音和其它参数的Wave Blaster面板

建立全局的设置

触发Global按钮（看起来象球的一个）产生如图A.5所示的全局设置窗口。在这里，你能设定许多的综合参数。

- 为设置主控调谐、变调、或定调转换范围，向左或右移动各自的滑动块。
- 为了要改变Wave Blaster的速度曲线（它给定当你按连在SB16上的MIDI键盘上的键时，对不同压力的不同响应声音），触发五个速度按钮中的一个。
- 为了选择四个MIDI模式或四个General MIDI选项中的一个，触发这些面板上的任一个

按钮，这些在Wave Blaster中建立了许多复杂的映射和优先级项目。然而，设置这些模式和选项需要MIDI和合成器的声音知识。



图A.5 Wave Blaster 的全局设置窗口让你布局Wave Blaster响应MIDI

为建立新声音的组管理器（Bank Arranger）

触发Bank按钮来访问WB的组管理器。组管理器为你示出——并让你装载和处理——设备产生的声音，以产生全新的定调组。

为使用组管理器：

1. 打开任一既在Instrument Bank窗口中又在Available Instrument Preset窗口中的设备组。
2. 由Available窗口的一个组替换Instrument Bank窗口中的任一组。为做到此，在每个窗口里选择一个组，并触发箭头按钮，这将给Wave Blaster一个完全不同的声音组。

Creative WaveStudio

从狭义上来说，Creative WaveStudio是一具有惊人能力的数字音频编辑器。如图A.6所示，它用图形显示了各WAV文件。由所显示的WAV文件，你可通过在概观窗口上触动和拖动来迅速移到文件里的不同点。除了它缺少一条Undo命令—— 一个较为严重的疏忽之外，Creative WaveStudio是一个出色的工具。通过它，你可建立短的音频剪辑，甚至混合两个剪辑。

在它的主窗口中，程序显示出一波形，一个菜单和一组按钮。你可以触发按钮来开始、停止或暂停Creative WaveStudio的记录和播放功能。你能通过观看显示你的选择的开始和结束的三个读出数据，得知你处在文件的何处。滚动按钮让你改变选择的大小和位置，而且你也可以在读出数据上拖动，并键入不同的数值来改变大小和位置。读出数据的变化在选择中被反映出来。如图A.7所示，显示窗口能保持许多文件。

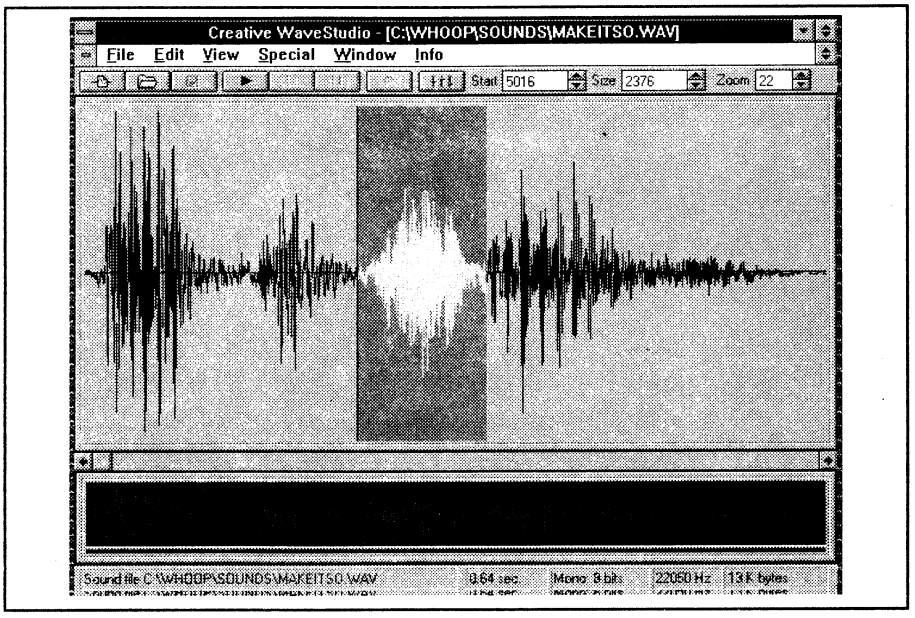
移动镜头（Zoom）

在屏幕底部的信息框中示出了文件的持续时间、采样率、大小和格式，你可移近或移远

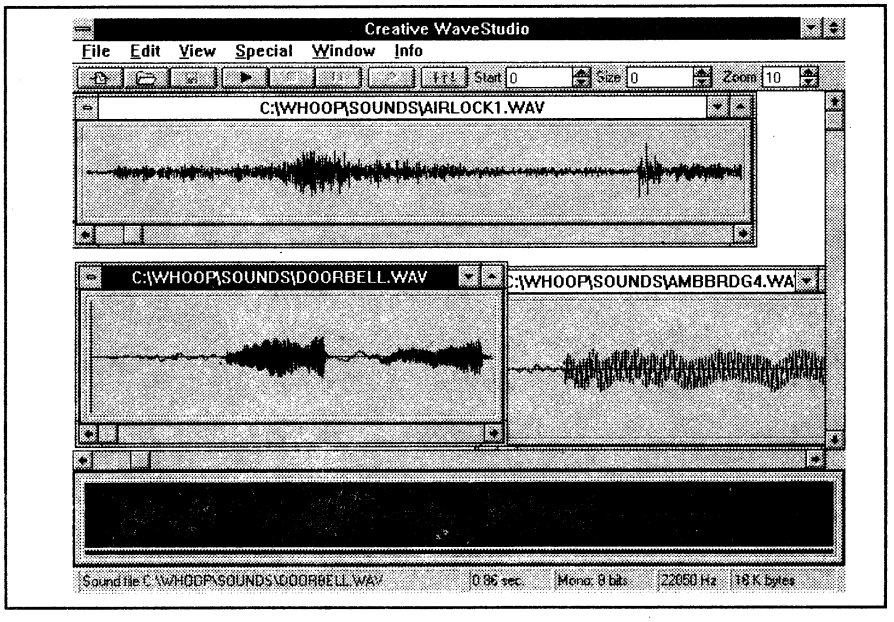
镜头，以看几个周期或整个样本：

• 为移远镜头并同时选择整个文件，那么在总视屏上触发两次。

• 为了选择样本的一部分，在屏幕底部的Preview窗口中拖动，你所拖过的部分将显示在窗口里。遗憾的是，当你作此时，镜头移动因子不变，为看见你的文件，你必须手动改变镜头移动因子。



图A.6 Creative WaveStudio让你记录和编辑数字音频



图A.7 Creative WaveStudio让你一次同很多文件工作

编辑文件

触发并拖动鼠标以选择文件的一部分。Edit菜单提供以下选项用来处理文件的部分：

选 项	用 途
Cut	取出一选项部分并把它放在剪取板（Clipboard）中，以使你能在另一处粘贴它。
Copy	将选择的部分复制到剪取板中，使你能将它放在任意处。
Paste	把剪取板中的选择放在文件中新的位置上。
PasteMix	融合两种声音。该菜单也让你删除一个选择或修剪一个文件——也就是删掉除选择以外的任何东西。

special菜单 special菜单让你在文件中插入无声的段，将选择转换为无声，淡入或淡出，或使选择倒向播放。例如，Rap！命令将一条选择的一个拷贝直接放在原来的之前，以造成一个口吃效果。用Rap！命令你能产生回声和放大声音音量至500%（但不降低它）。

混合器按钮 触发混合器按钮启动Sound Blaster 16混合器控制程序。Sound Blaster16混合器在下面部分介绍。

用SB16混合器混合声源

Sound Blaster 16混合器让你通过简单地移动屏幕上的滑块来设置卡的麦克风、线入、CD和其它输入的音量。你也可以通过触发窗口右边的按钮来调整增益和主控输出音量，以及每个声源的接通和断开。你在此窗口所作的各设置，将对你使用的每个应用都有影响。

提示 能关掉输入声源是有帮助的。例如，如果你不是正在使用麦克风，则你可关掉该输入，以减少电路上的杂音。设置两个增益控制到*4，可在信号音量和噪音之间得到最好的平衡。

Soundo'LE

目标链接和嵌入（OLE）是一种新技术，它能让你把一种应用中建立的数据“目标”复制和粘贴到另一个应用程序创建的文件中去。例如，你能用Creative Sound'LE来将你的声音数字化，然后剪取并粘贴此声音到创作纸上去，在创作纸中的一个图标表示你的声音。你记录声音的文件叫源文件，你粘贴目标的文件称为目的文件。

Creative Soundo'LE被设计成让你使用OLE技术。要用Soundo'LE记录声音就象你在Creative WaveStudio中记录声音一样：触发记录按钮，通过麦克风讲话，然后当你结束讲话时，触发Stop按钮。然而，由于它没有编辑功能，比较好的办法是在WaveStudio中录音，编辑、记录直到你满意为止，然后在Soundo'LE中打开结果文件。

为了往文件中插入一个Soundo'LE的记录：

1. 打开你要插入声音的文件。
2. 选择菜单提供的InsertObject。
3. 选择Soundo'LE，Soundo'LE窗口将打开。

4. 从Soundo'LE的File菜单中选择目标并触发OK。

5. 从Soundo'LE的File菜单选择Update（目的应用）。

6. 回到你的目的应用。

7. 在你的目的文件里的Soundo'LE图标上触发两下，它将被播放。

如果你想了解OLE技术更多的信息，可阅读Microsoft Windows手册。

附录B 直接存储器存取（DMA），中断请求（IRQ）和输入/输出地址（I/O地址）

本附录解释DMA通道，IRQ号和I/O地址的用法，同时将解释它们和Sound Blaster卡如何配合。

了解DMA通道

当Sound Blaster（SB16）记录或播放数字音频时，它用到PC上的直接存储器存取（DMA）通道，在其本身和PC的RAM之间进行音频的传送。SB16这样做法是为了旁路PC的主处理器，而按数字音频需要的速度来传送数据。Sound Blaster用到PC AT机6个DMA通道中的两个：其中，一个用于8位音频（缺省通道1），另一个用于16位音频（缺省通道5）。其它的一些设备也可与PC机相连，也可以占用DMA通道。两个软、硬盘驱动器一般占用通道中的2或3，以下一些设备通常用通道1：

- SCSI控制器卡
- 网络卡
- 备份磁带机控制器卡
- 扫描器卡
- PostScript和其它打印卡

由于数字音频有时要求传输大量数据，所以当你记录或播放声音时，将Sound Blaster和一个别的设备不用的DMA通道相连是很重要的。如果你在演奏或记录时，不用SCSI设备或网络、打印机或其它控制器卡，那么，在任何情况下这种缺省应是对的。

如果你在放音时听到嘶嘶声或根本不能放音，这是告诉你有一个DMA冲突。如果你在使用SB16且有DMA冲突时，要重新运行一次设置程序，并且使各DMA参数复位。设置程序将修改你的系统文件以供给新的设置。如果你在使用Sound Blaster Pro，就要拆下卡并且改动DMA的设置。如使用其它版本的Sound Blaster，你必须转到其它设备并改变它们的DMA设置。

了解IRQ级

当你的PC所连的一台设备需要计算机的CPU完成一些功能时，它要求CPU中断它正在做的工作而代之以马上来干你的工作。你的这项工作可能是接收一个MIDI音符、写软盘、解释键盘指令或者其它一些事。为跟踪这些请求，与PC相连的设备有各自的中断号，这样，当处理器同时从两个或更多的设备获得中断请求（IRQ）时，它知道它首先应注意较低的中断号。

表B.1 列出了80X86 PC外设最常用的IRQ设置

表B.1 80X86 PC外设常用的IRQ设置

中断请求	设 备
0	定时器
1	键盘
2	和IRQ9合用
3	串口2： COM2和4（调制器）
4	串口1： COM1和3（串行鼠标器）
5	并行打印机2
6	软盘驱动器
7	并行打印机1
8	时钟
9	对80X86计算机为IRQ2
10-12	未分配
13	80X87协处理器
14	硬盘
15	未分配

IRQ的级别是在出厂时定好的，制造厂商相信这是避免卡的互相冲突的最好机会。Sounb Blaster 16缺省为IRQ5，Sound Blaster缺省为IRQ7。

了解I/O地址

PC机所接的每一台设备都有一个输入/输出（I/O）地址，使其与计算机通信。I/O地址是一个号码，将一设备所有连接插头都接到PC机的插槽，并让PC机区分安装在其上的各种设备。当你的PC机的CPU想送出或接收数据到或来自Sound Blaster或其它设备时，它“广播”设备的输入/输出地址，设备听到自己的地址被“广播”时，就知道CPU有一信息要给自己，并听候着。如果设备被置成其它的地址，就简单地不管此广播。显然，一个地址不能为多个设备所用。

有些设备，包括Sound Blaster有一个地址范围。例如，一个设备可在10个地址位置上使用，假如说其“基”I/O地址为330，那么所有从330到339的地址都可以用。另外，Sound Blaster和其它完成多个功能的各设备，对应每一个功能，具有不同的基地址（如必要的话，有一个地址范围）。

提示 **I/O地址用16进制表示。这是一种广泛用在计算机中并以16为基的数字系统（普通是以10为基的十进制系统）。16进制数用字母A到F表示10到15，这样，我们将看到如23FH或30DH这些数字。在每个数后的H为表示16进制的符号，你也可见在16进制数的前面，放一美元符号（$220）。这里H和$只是告诉你是16进制，没有什么别的作用。**

Sound Blaster有一个基地址用于一般的功能，而另外一个用于MIDI端口。普通基地址缺省是220H范围到233H；MIDI端口基地址为330H，范围到331H。

Creative Labs告诫Sound Blaster用户不要改变卡的地址，因为这些地址很少用于别的设

备，而且在大量的游戏和一些支持Sound Blaster的软件中都已用到这些地址进行编程。我们也建议你不要改变I/O地址，只是因为不改变他们降低了所存在的混乱，这是你必须要做的。

附录C Sound Blaster 端口地址

表C.1示出了控制Sound Blaster各个部分的寄存器的端口地址。表C.2表示了Sound Blaster Pro的同样内容。Sound Blaster允许的基端口为210H，220H，230H，240H，250H，260H；Sound Blaster Pro允许的基端口是220H和240H。

表C.1： Sound Blaster的端口图表

端口*	说明	访问
200H-207H	模拟游戏杆端口	读/写
端口+00H	C/MS 1...6寄存器数据	写
端口+01H	C/MS 1...6寄存器选择	写
端口+02H	C/MS 7...12寄存器数据	写
端口+03H	C/MS 7...12寄存器	写
端口+06H	DSP复位	写
端口+08HFM	寄存器选择和状态	写和读
端口+09H	FM寄存器数据	写
端口+0AH	DSP读数据	读
端口+0CH	DSP给出命令或数据和缓冲器状态	写和读
端口+0EH	DSP数据可用的状态	读
388H	FM寄存器选择和状态	写和读
389H	FM寄存器数据	写

*端口 在此表中指的是基端口

表C.2 Sound Blaster Pro的端口图表

端口*	说明	访问
200H-207H	模拟游戏杆端口	读/写
端口+00H	左FM寄存器选择和状态	写和读
端口+01H	左FM寄存器数据	写
端口+02H	右FM寄存器选择和状态	写和读
端口+03H	右FM寄存器数据	写
端口+04H	混合器芯片寄存器选择	写
端口=05H	混合器芯片寄存器数据	写和读
端口+06H	DSP复位	写
端口+08H	左右FM寄存器选择和状态	写和读
端口+09H	左右FM寄存器数据	写
端口+0AH	DSP读数据	读
端口+0CH	DSP给出命令或数据及缓冲器状态	写和读
端口+0EH	DSP数据可用的状态	读

（续表）

端口*	说明	访问
端口+10H	CD-ROM命令和数据寄存器	写和读
端口+11H	CD-ROM状态寄存器	读
端口+12H	CD-ROM复位寄存器	写
端口+13H	CD-ROM存取寄存器	写
388H	FM寄存器选择和状态	写和读

*端口　在此表中指的是基端口

附录D　混合器芯片寄存器

本附录示出了用于控制混合器芯片操作的寄存器。表D.1表示通用寄存器。表D.2表示输入选择。下列是表D.1中所用缩写的说明

编写	意义
xx	未定义
1.	左
r.	右
vol.	音量
FINP	输入滤波器
TFIL	输入滤波器类型
FOUT	输出滤波器
ST	立体声/单声模式

表D.1：　通用寄存器

寄存器	位7	位6	位5	位4	位3	位2	位1	位0
00H	Data reset							
02H	DSP volume l.			XX	DSP volume r.		XX	
0AH	XX	XX	XX	XX	XX	Mic vol.	XX	
0CH	XX	XX	FINP	XX	TFIL	Select	XX	
0EH	XX	XX	FOUT	XX	XX	XX	ST	XX
22H	General vol.			XX	General vol.			XX
26H	FM volume l.			XX	FM volume r.			XX
28H	CD volume l.			XX	CD volume r.			XX
2EX	Line volume l.			XX	Line volume r.			XX

表D.2　输入选择

选择	输入项
0	麦克风
1	CD-ROM
3	线入

附录E DSP命令

本附录讨论所有的DSP命令。一些命令超出了本书的范围，但仍以某种方式叙述，从而使你获得一个它们用于什么的概念。端口2XCH用来写DSP；端口2XAH用来读DSP。

播放命令

以下是DSP播放命令

10H，8位直接播放

1. 给出命令10H
2. 送数据字节。
3. 以一恒定速度重复步骤1和2。

14H，8位通过DMA播放

1. 设置DMA控制器并接收IRQ中断申请。
2. 设置采样率（参看命令40H）。
3. 给出命令14H。
4. 送样本块1长度的低位字节。
5. 送样本块1长度的高位字节。
6. 第5步以后，样本输出立即开始，结束时，DSP通过设置的IRQ中断请求产生一个中断申请。通过DMA可读出最大长度64KB的样本块。这样，一些较大的样本必须分割成一些较小的块。

91H，高速的8位通过DMA播放（Pro）

1. 设置DMA控制器并接收IRQ中断申请。
2. 设置采样率（参看命令40H）。
3. 给出命令48H。
4. 传送样本块1长度的低位字节。
5. 传送样本块1长度的高位字节。
6. 给出命令91H。
7. 第6步之后样本输出开始，结束时，DSP通过设置的IRQ产生一中断申请，通过DMA可传送最大长度64KB的样本块。这样，一些较大的样本必须分割成一些较小的块。

注解 Sound Blaster Pro只支持命令48H和91H，这些命令必须用来播放以较高的采样率记录的样本。

用以播放压缩样本的命令

以下是播放压缩样本的命令表。

16H，2位压缩通过DMA播放

这个命令采用命令14H相同的方法。

17H，带基准字节的2位压缩通过DMA播放

这个命令采用和命令14H相同的方法，样本数据的第一字节为基准字节；该字节包括通常的8位样本数据。如播放不同的样本块，命令16H一定被用来播放下面的各块，因为这些样本块不包括基准字节。

74H，4位压缩通过DMA播放

参看命令16H。

75H，带基准字节的4位压缩通过DMA播放

参看命令17H。

76H，2.6位压缩通过DMA播放

参看命令16H。

77H，带基准字节的2.6位压缩通过DMA播放

参看命令17H。

记录命令

以下是DSP的记录命令。

20H，直接记录

1. 给出命令20H。
2. 读数据字节并存贮。
3. 以恒定速度重复步骤1和2。

24H，通过DMA记录

1. 设置DMA控制器并接收IRQ中断请求。
2. 设置采样率（参看命令40H）。
3. 给出命令24H。
4. 送样本块1长度的低位字节。

5. 送样本块1长度的高位字节。
6. 第5步后样本输入立即开始，结束时DSP通过设置的IRQ产生中断申请，通过DMA可读出最大长度为64KB的样本块。因此较大的样本必须记录为较小的一些块。

99H，高速8位通过DMA记录（Pro）

1. 设置DMA控制器并接收IRQ中断请求。
2. 设置采样率（参看命令40H）。
3. 给出命令48H。
4. 送样本块1长度的低位字节。
5. 送样本块1长度的高位字节。
6. 给出命令99H。
7. 第6步之后样本输入开始，结束时DSP通过设置的IRQ产生一中断申请，通过DMA 可读出最大长度64KB的样本块。因此，对于较大样本必须记录为较小的一些块。

注解 Sound Blaster Pro只支持命令48H和99H，这些命令必须用在以较高速采样率来记录样本。

扬声器命令

下列是扬声器命令。

D1H，接通扬声器的连接

此命令后，DSP信号被送至卡上所装有的放大器。此命令约需112毫秒时间。

D3H，断开扬声器连接

此命令后，DSP不送任何信号给放大器。DSP执行命令至多需要220毫秒。

D8H，询问扬声器的设置

1. 给出命令D8H。
2. 从DSP读一个字节，如该字节值为0，表示扬声器不接通，如其字节值为255，表示扬声器接通。

注解 此命令只被Sound Blaster Pro支持。

其它的DSP命令

以下是DSP命令余留部分。

40H，设置采样率

1. 给出命令40H。

2. 将按下列公式计算得出的结果字节送出。

256-1，000，000/采样率

48H，设置样本块长度

为使用此命令，参看命令91H和99H。

80H，设置无声块

1. 接收IRQ中断申请。
2. 设置采样率（参看命令40H）。
3. 给出命令80H。
4. 送无声块1长度的低位字节。
5. 送无声块1长度的高位字节。
6. 结束时，DSP通过设置IRQ产生一中断请求。

DOH，停止DMA

停止DSP与存储器之间的任何DMA输入或输出。

D4H，继续DMA

在DMA操作已停止的情况下，可通过给出此命令继续DMA操作。

E1H，查询版本号

1. 给出命令E1H。
2. 读出高位部分的版本号。
3. 读出低位部分的版本号。

DSP MIDI命令

以下是DSP MIDI命令的描述。

30H，直接的MIDI输入

1. 给出命令30H。
2. 检查端口2XEH的位7是否被置位（=1）。
3. 读MIDI码。
4. 重复步骤2和3。

31H，通过中断的MIDI输入

1. 接收IRQ中断申请。
2. 给出命令31H。
3. 如果一MIDI码被传送，DSP产生一个中断申请。

4. 中断程序必需读出并且必须保存这MIDI码。
5. 读一次2XEH端口，结束中断。

32H，直接带时间标记的MIDI输入

1. 给出命令32H。
2. 检查2XEH端口的位7是否被置位（=1）。
3. 读时间标记的最低字节。
4. 读时间标记的中间字节。
5. 读时间标记的最高字节。
6. 读MIDI码。
7. 重复步骤2到6。

经步骤3到5产生一个以毫秒来表示的24位时间值，这就是命令32H被送出后所送的毫秒数。

33H，通过中断的带时间标记MIDI输入

该命令的工作方式和命令31H相同，只是这次中断程序需读四个字节，前三个字节包含时间标记，最后一个字节包含实际的MIDI码。

34H，直接MIDI UART模式

此命令之后，MIDI代码可以被送出和读出。你可直接将MIDI码写到DSP，而不需先发送命令38H。

35H，通过中断的MIDI UART模式

除了MIDI输入通过中断产生外（见命令31H），该命令和前述命令作用相同。

36H，直接的带时间标记MIDI UART模式

此命令和命令34H的工作方式相同，但读MIDI码的条件与命令32H相同。

37H，通过中断的带时间标记MIDI UART模式

此命令通过中断进行读出。就象命令33H一样，中断应读出MIDI码和时间标记。

对UART模式，DSP用了一个64字节的缓冲器，因此，当MIDI输入太快时，DSP可以把MIDI码存在缓冲器中。用此方法，DSP可以存贮不多于64个无时间标记的代码和16个有时间标记的代码。

38H，送出一个MIDI码

1. 给出命令38H。
2. 读端口22CH。
3. 重复步骤2，直至第7位被复位（=0）。
4. 写一个MIDI码。

注解：MIDI UART和时间标记只有Sound Blaster Pro才支持。如果想不同UART模式，DSP应通过端口2×6H来复位。

附录F MIDI设备制造厂商的标识码（ID）

表F.1列出了代表各个MIDI设备制造厂商的标识（ID）码。这些代码作为系统专有的信息，即仅适用于由该制造厂商生产的系统的MIDI信息。

注解 有关系统专有信息更详细的信息，参见第十章。

表F.1： MIDI设备制造厂商的代码

十进制	十六进制	厂商标识
0	00	For general use
1	01	Sequential Circuits
2	02	Big Briar
3	03	Octave Plateau
4	04	Moog
5	05	Passport Designs
6	06	Lexicon
7	07	Kurzweil
8	08	CBS/Fender
9	09	Steinway & Sons
10	0A	Delta Lab Research
11	0B	Sound Composition System
12	0C	General Electro Music
13	0D	Techmar
15	0F	Ensoniq
16	10	Oberheim
17	11	Paia Electronics
18	12	Simmons
19	13	Gentle Electric
20	14	Fairlight
22	16	Lowery
32	20	Bon Tempi
33	21	SIEL
34	22	Ircam
35	23	Synthaxe
36	24	Hohner
37	25	Crumar
38	26	Solton
39	27	Jellinghaus
44	2C	Peter Struven
57	39	Soundcraft Electronics
62	3E	Waldorf
64	40	Kawai

(续表)

十进制	十六进制	厂商标识
65	41	Roland
66	42	Korg
67	43	Yamaha
68	44	Casio
70	46	Kamiya Studio
71	47	Akai
72	48	Japan Victor
73	49	Meishosa
78	4E	TEAC
80	50	Matsushita
81	51	Fostex

附录G　MIDI的状态和数据字节

表G.1　.列出了各个MIDI信息字节的值

表G.1：　MIDI的状态和数据字节

状态字节			数据字节	
第一字节值	功能		第二字节值	第三字节值
80 128	Channel 1	Note off	Note Number	Note Velocity
81 129	Channel 2	"	(0-127)	(0-127)
82 130	Channel 3	"	"	"
83 131	Channel 4	"	"	"
84 132	Channel 5	"	"	"
85 133	Channel 6	"	"	"
86 134	Channel 7	"	"	"
87 135	Channel 8	"	"	"
88 136	Channel 9	"	"	"
89 137	Channel 10	"	"	"
8A 138	Channel 11	"	"	"
8B 139	Channel 12	"	"	"
8C 140	Channel 13	"	"	"
8D 141	Channel 14	"	"	"
8E 142	Channel 15	"	"	"
8F 143	Channel 16	"	"	"
90 144	Channel 3	"	"	"
83 131	Channel 1	Note on	Note Number	Note Velocity
91 145	Channel 2	"	(0-127)	(0-127)
92 146	Channel 3	"	"	"
93 147	Channel 4	"	"	"
94 148	Channel 5	"	"	"
95 149	Channel 6	"	"	"
96 150	Channel 7	"	"	"
97 151	Channel 8	"	"	"
98 152	Channel 9	"	"	"
99 153	Channel 10	"	"	"
9A 154	Channel 11	"	"	"
9B 155	Channel 12	"	"	"
9C 156	Channel 13	"	"	"

(续表)

状态字节			数据字节	
第一字节值	功能		第二字节值	第三字节值
9D 157	Channel 14	"	"	"
9E 158	Channel 15	"	"	"
9F 159	Channel 16	"	"	"
A0 160	Channel 1	Polyphonic	Note number	Aftertouch
A1 161	Channel 2	Aftertouch	(0-127)	(0-127)
A2 162	Channel 3	"	"	"
A3 163	Channel 4	"	"	"
A4 164	Channel 5	"	"	"
A5 165	Channel 6	"	"	"
A6 166	Channel 7	"	"	"
A7 167	Channel 8	"	"	"
A8 168	Channel 9	"	"	"
A9 169	Channel 10	"	"	"
AA 170	Channel 11	"	"	"
AB 171	Channel 12	"	"	"
AC 172	Channel 13	"	"	"
AD 173	Channel 14	"	"	"
AE 174	Channel 15	"	"	"
AF 175	Channel 16	"	"	"
B0 176	Channel 1	Control/	"	"
B1 177	Channel 2	Mode	"	"
		Change		
B2 178	Channel 3	"	"	"
B3 179	Channel 4	"	"	"
B4 180	Channel 5	"	"	"
B5 181	Channel 6	"	"	"
B6 182	Channel 7	"	"	"
B7 183	Channel 8	"	"	"
B8 184	Channel 9	"	"	"
B9 185	Channel 10	"	"	"
BA 186	Channel 11	"	"	"
BB 187	Channel 12	"	"	"
BC 188	Channel 13	"	"	"
BD 189	Chennel 14	"	"	"
BE 190	Chennel 15	"	"	"
BF 191	Chennel 16	"	"	"
C0 192	Channel 1	Program	Program	None
C1 193	Channel 2	Change	Number	"
C2 194	Channel 3	"	(0-127)	"
C3 195	Channel 4	"	"	"
C4 196	Channel 5	"	"	"
C5 197	Channel 6	"	"	"

(续表)

状态字节			数据字节	
第一字节值	功能		第二字节值	第三字节值
CC 204	Channel 13	"	"	"
CD 205	Channel 14	"	"	"
CE 206	Channel 15	"	"	"
CF 207	Channel 16	"	"	"
D0 208	Channel 1	Channel	Aftertouch	None
D1 209	Channel 2	Aftertouch	Pressure	"
D2 210	Channel 3	"	(0-127)	"
D3 211	Channel 4	"	"	"
D4 212	Channel 5	"	"	"
D5 213	Channel 6	"	"	"
D6 214	Channel 7	"	"	"
D7 215	Channel 8	"	"	"
D8 216	Channel 9	"	"	"
D9 217	Channel 10	"	"	"
DA 218	Channel 11	"	"	"
DB 219	Channel 12	"	"	"
DC 220	Channel 13	"	"	"
DD 221	Channel 14	"	"	"
DE 222	Channel 15	"	"	"
DF 223	Channel 16	"	"	"
E0 224	Channel 1	Pitch Wheel	Pitch Wheel	Pitch Wheel
E1 225	Channel 2	"	LSB	MSB
E2 226	Channel 3	"	(0-127)	(0-127)
E3 227	Channel 4	"	"	"
E4 228	Channel 5	"	"	"
E5 229	Channel 6	"	"	"
E6 230	Channel 7	"	"	"
E7 231	Channel 8	"	"	"
E8 232	Channel 9	"	"	"
E9 233	Channel 10	"	"	"
EA 234	Channel 11	"	"	"
EB 235	Channel 12	"	"	"
EC 236	Channel 13	"	"	"
ED 237	Channel 14	"	"	"
EE 238	Channel 15	"	"	"
EF 239	Channel 16	"	"	"
F0 240	System Exclusive	**	**	
F1 241	System Common	Not defined?	?	
F2 242	System Common	Song Position Pointer	LSB	MSB
F3 243	System Common	Song Select	Song number (0-127)	None
F4 244	System Common	Not defined?	?	
F5 245	System Common	Not defined?	?	

(续表)

状态字节			数据字节	
第一字节值	功能		第二字节值	第三字节值
F6 246	System Common	Tune Request	None	None
F7 247	System Common	End of SysEx (EOX)	"	"
F8 248	System Real Time	Timing Clock	"	"
F9 249	System Real Time	Not defined	"	"
FA 250	System Real Time	Start	"	"
FB 251	System real Time	Continue	"	"
FC 252	System Real Time	Stop	"	"
FD 253	System Real Time	Not defined	"	"
FE 254	System Real Time	Active Sensing	"	"
FF 255	System Real Time	System Reset	"	"

*见附录F

** 系统专有的第二个字节为制造厂的ID。它后跟几个数据字节并以EOX（F7）结尾。