高级DOS程序设计

严红岩

北京希望电脑公司

一九九一年九月

目　录

第一部分　磁盘分析

这部分，将对磁盘及其引导记录，FAT 表、目录、子目录、文件以及分区进行研究。从中你可了解到它们的功能，怎样去使用它们，及各种不同类型的磁盘的特点。通过设计一个对磁盘进行检测和修改的实用程序，你可以了解一些对磁盘编程的技巧。

这一部分的结尾是一个修改 DOS 内部命令的程序。

了解磁盘的工作原理在许多方面对你会很有帮助。如果你操作磁盘时犯了某种失误，比如不小心删除了一个文件，重新格式化了硬盘，或毁坏了一个目录，你就能知道怎样去恢复这些数据。这些知识可帮助你开发或利用那些使用磁盘的应用程序，比如数据库或加密系统。

第一章　磁盘简介

你每次使用计算机，其实都是在使用磁盘。从磁盘引导，从磁盘装载应用程序，再把结果存到磁盘上。本章将讲述信息在磁盘上是怎样存贮和组织的，这将有助你编写那些充分利用磁盘资源的功能强大的应用程序。

通常，把磁盘看成是文件的集合。你可以运行程序文件，编辑数据文件，还可以拷贝、删除文件，但磁盘能做的远不止这些。

实际上，磁盘上包含有更多的信息，磁盘必须识别它自己是什么类型的磁盘，包含什么文件，这些文件存在什么位置，哪里有空间来存贮更多的文件。这些信息是以四种基本结构存贮的：分区表，引导记录，文件分配表和目录。用户基本上是看不到这四种结构的。磁盘的物理结构为面、磁道和扇区。首先，看看磁盘的物理结构。

磁盘是由磁介质组成的圆盘，分为软盘和硬盘。硬盘是一类这样的圆盘：每一个圆盘有顶和底两面。正在使用的磁盘上的那一面被称作面。磁头指的是做读工作的电子机械设备，它是一个在磁盘上来回移动的小电磁体。软磁盘有两个面。硬盘的每一圆盘通常都有两个面。

磁盘的每一面进一步分成许多同心环，叫作**磁道**或**柱面**。磁头可在磁盘上的任何磁道上移动。由于磁盘的旋转，磁头就可以读到存贮在磁盘上的不同信息。

每一道又分成许多**扇区**。扇区是能从磁盘上读或写的最小信息块。一旦读扇区，就可以检查单独的字节。一个扇区一般是 512 字节长。

在一个磁盘上有许多扇区需要管理，因此，扇区又组成**簇**。簇是相邻扇区的集合。根据磁盘的类型，每簇有 2－8 个扇区不等。图 1.1 是磁盘的示意图。

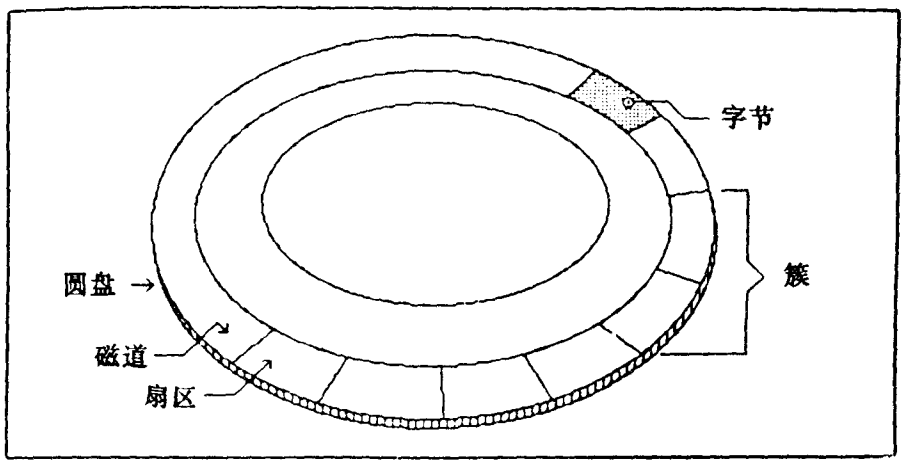

图 1.1　磁盘

上面这些解释听起来很复杂，但你却勿需担心。因为 DOS 已为你做了绝大部分工作，你一般不用考虑簇或扇区，几乎可以不用过问道和面。

绝大多数时间里，你是使用一个操作系统(DOS)，但有一些 PC 用户还需使用 CP/M 或 XENIX 系统。如果这些用户有一个硬盘，他们肯定想用它存贮所有这些操作系统的文件，而这些操作系统操作磁盘的方式与 DOS 不兼容。为使一个硬盘能被一个以上的操作系统所使用，硬盘最多可分成四个区，每一区可以使用一个不同的操作系统。

分区表包含的信息有：每一个分区的位置和大小；如果计算机是以硬盘引导的，那么将使用哪一个区(即**活动区**)；哪一个区用 DOS 作为其操作系统。这最后一点很重要。你在后面几章将了解到，DOS(和实用程序)要求在磁盘的一些地方找到某种类型的信息。这些预先定义的位置可能在硬盘的任何地方，这就取决于分区是如何建立的了。DOS 利用分区表来确定所使用的分区从哪开始，并把此开始位置作为访问磁盘时的偏移地址。这样，就总能找到那些特别信息，并且对 DOS 和用户双方，分区都是透明的。

当打开一台计算机时，ROM 里的一个程序首先尝试从 A 驱动器(不管 A 驱动器中是否有盘)引导系统。如果 A 驱动器中没有磁盘，那么这个 ROM 里的程序就把控制权转给分区表之前的一个程序。(在一个无硬盘的系统里，就启动 ROM BASIC)。该程序读分区表，如果找到的信息有效，就寻找活动分区引导记录的位置，并移交控制权。

记住，分区表只能在硬盘里找到。

引导记录是磁盘中的第二个重要部分。它包含装载和启动操作系统的程序，还包含一张描述磁盘的表：大小、类型、操作系统、格式以及有关文件分配表和目录的重要信息。DOS 靠这些信息确定怎样对磁盘进行读写。

数据存放在簇中的文件里。簇经常不是顺序出现的，一个文件的第一部分可能在靠近磁盘前面某个地方，而其余部分则可能在磁盘的中间或末尾部分。**文件分配表**(FAT)确定每个文件使用的簇，磁盘的哪一部分是未用的，磁盘的哪一部分因为物理故障而不能使用。文件分配表非常重要，所以在磁盘上常有两个拷贝。没有文件分配表，磁盘上的所有数据将被随意组织而毫无意义。

目录是最后一个组织结构。它包含文件的名字、大小、生成时间，以及 DOS 怎样才能找到文件。

程序设计要点：

- 磁盘包括面、磁道和扇区。
- 扇区组成叫做簇的组织单元。
- 磁盘的五个基本部分是分区表、引导记录、文件分配表、目录和数据区。

第二章　EXPLORER 概貌

本章将详细介绍磁盘的工作原理。你不仅可以了解磁盘的结构和使用方法，还可实际测试一下。为此，将用 Pascal 编写一个磁盘检测程序，由于将讨论磁盘的不同特征，所以要在程序里增加些模块来检测这些特征。

图 2.1 显示了 EXPLORER 的主菜单。

```
----------------------MAIN MENU-------------------
F1:      Dump and Modify Sectors
F2:      Examine Boot Record
F3:      Cluster by Cluster FAT Dump
F4:      Examine Root Directory
F5:      Examine File
F6:      Get Back Erased File
F7:      Examine Partition

F10:     Exit
```

图 2.1　EXPLORER 的主菜单

该程序的全部代码都以文本形式给出。它是用 Turbo Pascal 3.0 和 Turbo Pascal 5.0 写成的。从代码可看出程序在 Turbo Pascal 5.0 下运行，去掉一些注释，就可在 3.0 版本下运行。有关两种版本的区别，已在文本中给出了。如果你使用的是另一种版本的 Pascal，则程序需稍微做些修改。

因为 EXPLORER 程序很长，所以一次只能写一部分，然后用 include($I)命令和主模块一起编译。主模块叫做 explorer.pas。

Turbo Pascal 5.0 像 C 一样有几种不同的整数格式。这些格式为带符号和不带符号，2 个或 4 个字节长。Turbo Pascal 3.0 只使用两字节的整数，但有时不免要处理 4 字节的长整数。为使程序与 Turbo Pascal 3.0 或 Pascal 的其它版本兼容，本书中的程序假定整数是两字节的，带符号的值，也就是大于 32K 的整数是负数，大于或等于 64K 的值返转。但这样造成了一些混乱，为解决这个问题，有时你得把整数值作为浮点数值，执行算术运算，然后再把浮点数回整数。如果你使用的是 Turbo Pascal 5.0，可把这些部分改成使用长整数，避免混乱的选型。

本书将用到二进制、十进制、十六进制编码。例如，二进制数将写成 1001111b，十进制是 14，十六进制数将写成 $15 或 15h。

程序设计要点：

- 编写一个叫做 EXPLORER 的磁盘检测程序。
- 把 EXPLORER 程序分成一系列的模块来编写。对应每一模块，都在 explorer. pas 中增加一个 include($I)命令，然后编译 explorer. pas。

第三章　读命令及磁盘I/O

在这一章将从 explorer. pas 开始编写 EXPLORER。explorer. pas 是 EXPLORER 的基础。在 explorer. pas 里，要为整个程序设置常量和类型说明，以及主菜单的进程命令。对磁盘进行读写的基本程序也要放在 explorer. pas 里。

为了获得你需要的灵活性和控制权，你编写的程序要从键盘读命令并执行磁盘 I/O。磁盘 I/O 程序可能比后面四章的程序更复杂，因为它要用汇编语言编写。

如果不熟悉汇编语言程序设计，你可能会觉得这些程序相当混乱。为使程序功能更强，必须使用 BIOS 和 DOS 中断调用。如果这些你也不熟悉，那么你就略过并使用本章末尾的程序。你也许不能完全理解它，但至少你知道它的功能。

读命令

为使 EXPLORER 易于使用，看起来也专业化，必须做到能处理功能键和光标键。但 readln 不能处理这些键，你必须要用一个 BIOS 中断调用(将在第二部分详细讨论)才行。

BIOS 中断是存贮在 ROM 里的子程序，执行最基本的与计算机硬件的交互。例如，有向屏幕上打印字符，检测光笔位置和往打印机送信息的 BIOS 中断，也有从键盘上读击键的中断。每个中断都使用一个特定的数值和参数组。

Pascal 中调用中断的方法是，用 intr 命令传送中断号和参数。参数用下列数据结构来设置：

```
result = record

    ax,bx,cx,dx,bp,si,di,ds,es,flags: integer;
end;
```

每个整数变量代表一个 8086 寄存器(若你对 8086 寄存器不熟悉，不要着急，这个概念在这并不至关重要，书中会告诉你怎样去设置参数)，有时不使用 ax，bx，cx 或 dx，而使用 ah，al，hl，ch，cl，dh 或 dl。这就涉及到相应寄存器的高位或低位字节。例如：ah＝ax/256；al＝ax％256。

读键盘用中断 16h，用 AH＝0 调用。中断程序等待一个击键键入，有一击键后返回，AX 是击键代码。若 AL 不为 0，则该键是一个标准字母键或数字键，AL 中是它的 ASCII 码。若 AL 为 0，AH 中是一个功能键或光标键的码。

中断 16h 和其返回码将在第 15 章详细讨论。现在只检测一下调用中断 16h 后，其返回值的类型。例如，可从主过程看到，假如键入 F10 键，中断将返回 AX＝4400h。

读键盘的 Pascal 程序如下：

```
regs.ax := 0;              {regs is of type result.  AX = 0 means
                            read keyboard}
intr($16,regs);  {wait for a key and return the code}
if regs.ax and $ff = 0 then ... {key is a function or cursor
                                 key}
```

你可以用下列语句检查 AX 的值：

```
case (regs.ax) of .....
```

explorer. pas 程序列在本章的末尾。可从中找到读键盘的程序。

磁盘读写

对磁盘进行读写也要使用中断。这次使用 DOS 中断。DOS 中断同 BIOS 中断非常相似，只是 DOS 中断程序不是 ROM 里的一部分。它们是在计算机被引导时，才装载上的，就程序而言，没有什么区别。

用中断 25h 和 26h 进行扇区读写，用逻辑扇区号访问扇区。磁盘的第一个扇区编号为 0，第二个为 1，以此下去，直到没有扇区剩下为止。

有时用物理扇区号访问扇区。在物理编号系统里，必须说明面和磁道。每个磁道包含本磁道上从 1 到最后一个扇区的所有扇区。DOS 中断则使之变得非常容易，DOS 只考虑什么时候改变磁道或面。只需告诉其一个扇区号，DOS 就会为你找到那个扇区。(相反，BIOS 磁盘中断则使用面、磁道、扇区参考系统。)

进行磁盘读写操作时，几乎都使用逻辑扇区号。

调用中断 25h 时，需要给出四个参数：

AL＝驱动器号(0＝A，1＝B，2＝C，以此类推)
CX＝要读的扇区号
DS:BX＝放置读出数据的缓冲区地址(DS 为缓冲区的段地址，BX 是偏移量)。

中断 26h 使用相同的参数，只是 CX 为要写的扇区号，DS:BX 指向能找到的要写数据的位置。

如果你能象读键盘时所做的一样用 intr 命令调用这些参数，那将很方便。但这两个中断有些麻烦，不象其它的中断，这两个中断在返回调用程序前不能复位，而是留下了一些数据——一个栈状态标记。如果你使用 intr 命令，在读写过程结束之前一切运行良好。但，当它想返回调用它的过程时，就将发生冲突。

为避免这个问题，你必须用 inline 功能在 explorer. pas 里增加一个小汇编语言程序(如果你使用的是 Turbo Pascal，那你一定要用一个相等同的命令。变量进栈的格式可能会有些不同。查阅你的编译员手册)。仔细观察从磁盘读的过程。往磁盘写的过程基本上是相同的。

你要生成一个叫做 DiskRead 的函数来读磁盘。该函数要使用驱动器号，开始扇区号，要读的扇区号和将用来存贮信息的缓冲区的指针。如果磁盘读失败，它就返回一个错误码。用 inline 功能调用 DOS 中断，检查错误，复位并返回。

仔细注意 Turbo Pascal 和汇编语言之间的接口处理过程。在 Turbo Pascal 手册中(2.0 和 3.0 版本)有关这点的叙述是错误的，各 Turbo Pascal 版本不同，与汇编语言的接口也不同。

当调用一个函数时，该函数把它的参数送到栈。数值参数作为数据传送，变量参数和数组作为指针传送。Turbo Pascal 3.0 中数组也是作为指针传送的，而不管是用数值还是变量传送的。Turbo Pascal 5.0 不是这样。所以当你转换程序使其在 Pascal 5.0 版本下运行时，要注意这种不同。不同类型的数据用不同的格式传送。字节和整数作为字传送，指针作为偏移量和段传

送，其它类型的传送情况，详见 Turbo Pasacl 手册。

函数说明之后，是用于存放函数结果的栈。整数型、布尔型、字节型、字符型其结果是一个字。然后，参数从说明的左边压入右边，接着压入三个以上的字(注：在 Turbo Pascal 手册中没有记载这些)。最后调用 inline 程序。

读参数首先要保存 bp 寄存器，把其压入栈。然后，把 sp 赋给 bp 并加 8 以指向参数区(刚压入的 bp 用了 2 个字节，刚才压入栈的三个字用了 6 个字节)。与 bp 相加得到下一个参数。记住，因为参数是从左向右压入栈的，所以当你与 bp 相加时要从右到左计数。

这个函数需要传送一个字节、两个整数值和一个指针。功能结果是一个整数。因此，压入 bp 后，将 sp 赋给 bp，再加 8，结果如下：

bp　指向指针偏移量值

bp＋2　指向指针段值

bp＋4　指向一整数

bp＋6　指向一整数

bp＋8　指向一个字节(以字来存贮的)

bp＋10　指向函数结果

记住在你的 inline 程序中要说明你压入栈的任何值。加 8 是因为你压入了 bp，Turbo Pascal 压入了三个附加字。

为了返回函数结果要把你想返回的值放入与返回值相对应的栈单元。

Turbo Pascal 5.0 里，栈调用略有不同。返回值存在 bp－6 里，而不是 bp＋10 里。

可在 inline 程序中修改任何 8086 寄存器。退出之前，你必须恢复 bp，sp，ds 和 ss。

下面看看从磁盘读一个扇区的汇编语言程序。该程序读参数，调用 DOS 磁盘读中断，检查错误，清栈，返回错误码(若没有错误返回 0)。注：程序结尾不需要 ret 指令，在 Turbo Pascal 函数中用 end 代替。为了在 Pascal 程序中使用汇编语言程序，需汇编该程序，并从 .LST 文件中读汇编值。下面这段 explorer.pas 完成这个功能。

```
; Reads sectors from disk using DOS interrupt 25h.  Assumes used
; within a function xxx(drive: byte; sector,no_sects: integer;
;      var buffer:array): integer
; Drive is a number indicating the disk drive, where 0=A, 1=B, ...
; Sector is the starting sector, no_sects tells how many sectors to
; read, and buffer is the disk buffer where the data will be stored.
;
; Returns a zero if the read was successful, otherwise returns a
; number telling an error code.

        push    bp              ;save bp register
        mov     bp,sp           ;get pointer to stack
        add     bp,8            ;point to parameters
        push    ds              ;save ds register
        mov     bx,[bp]         ;offset of disk buffer
        mov     ax,[bp+2]       ;segment of disk buffer
        mov     ds,ax           ;point to buffer for interrupt
        mov     cx,[bp+4]       ;number of sectors to read
        mov     dx,[bp+6]       ;first sector to read
        mov     al,[bp+8]       ;drive
        push    bp              ;save bp because it is destroyed
                                ;by the interrupt
        int     25h             ;issue the DOS disk read
                                ;interrupt
        jnc     ok              ;if there is no carry, the read
                                ;was successful
```

```
                popf                    ;Was no good--pop disk flags
                pop     bp              ;restore bp
                                        ;Use error code for return val
                mov     [bp-6], ax      ; for Turbo 5.0
;               mov     [bp+10],ax      ; for Turbo 3.0
                jmp     cont
ok:             popf                    ;Was good read--pop disk flags
                pop     bp              ;restore bp
                xor     ax,ax           ;Use 0 for error code and use
                mov     [bp-6],ax       ;it as the function return
                                        ;value
;               mov     [bp+10],ax      ; for Turbo 3.0
cont:           pop     ds              ;restore ds
                pop     bp              ;restore bp
                                        ;and continue with the code in
                                        ;the function
```

DOS 缓冲区

为了加速访问时间，DOS 在内存缓冲区中存贮一些磁盘数据。这样，DOS 可直接从内存而不是从磁盘读数据。当用 DOS 写扇区中断写磁盘时，直接修改了磁盘，但不必修改内存缓冲区。因此，在磁盘发生此变化时，观察内部缓冲区的程序将"看"不到这个变化，这样非常危险，特别是对那些从非移动介质中恢复删除文件的程序。

你可在调用中断 26h 后，复位内部文件缓冲区，这样可以很容易地避免这个问题。用 DOS 中断 21h，功能 13 完成这个任务：

操作：复位内部磁盘缓冲区
版本：DOS 1.0—
中断号：21h
功能号：13
调用值：AH=13

DiskWrite 功能在进行了磁盘写操作之后调用该中断。

EXPLORER.PAS

你已能读键盘命令，进行磁盘读写操作，现在来看看整个 explorer.pas 程序。

注意，磁盘读写程序是怎样用 inline 命令插入的。还要注意键盘读线。

大多数常量用来处理窗口位置。在后面的模块中将更多地用到窗口。注意，磁盘缓冲区已足够大，一次可读入九个扇区。其原因将在第 5 章讨论。

还要注意全局变量用来指定被检测的驱动器和扇区，这样你就可以在一个模块中指定一个扇区，然后在另一个模块中对它进行修改。

这段程序在 Turbo Pascal 5.0 下运行。有些部分用{$IFDEF Turbo 3}语句隔开。要在 Turbo Pascal 3.0 下运行该程序，需删除{$IFDEF}行，和所有和 Turbo Pascal 5.0 程序，这部分程序在一个{$ELSE}和一个{$ENDIF}之间。

```
{.....................DISK EXPLORER.....................
.
. This is a program to explore disks.  It has commands for
. decoding the partition, boot record, FAT, and directory,
```

```
. for examining and modifying data sectors and directories,
. for displaying files, and for getting back erased
. files.
.
. This is the main body of code -- the one to be compiled.
. Because of the length of the code, each section is kept in a
. separate disk file which is included into the listing.
.
.............................................................)

program DiskExplorer;

{$IFNDEF Turbo3}
uses Dos, Crt;
{$ENDIF}


{
.................................
.
. Constant definitions
.
.................................
}


const
   BUFLEN = 4608;                {size in bytes in disk buffer}
   BUFSECT = 9;                  {number of sectors in the disk buffer}
   COMMANDX = 1;                 {these four constants define the location}
   COMMANDY = 20;                {of the window for displaying command menus}
   COMMAND_RT = 79;
   COMMAND_BOT = 24;
   STATUSX = 1;                  {these four constants define the location}
   STATUSY = 18;                 {of the window for displaying the status}
   STATUS_RT = 79;
   STATUS_BOT = 19;
   LFTSCRN = 1;                  {these four constants define the location}
   TOPSCRN = 1;                  {of the main display window}
   RTSCRN = 80;
   BOTSCRN = 17;
   ERROR = -1;                   {these four constants are for parameter passing}


{
.................................
.
. Type declarations
.
.................................
}


type

   {for DOS interrupts}


{$IFDEF Turbo3}
   result = record
       ax,bx,cx,dx,bp,si,di,ds,es,flags: integer;
     end;
{$ELSE}
        result = Registers;
{$ENDIF}

   {buffer for reading in disk sectors}
   buf = array[0..BUFLEN] of byte;
```

```
{
.................................
.
. Global variables
.
.................................
}


var
   finished: boolean;
   glbl_drive: byte;                    {drive in use}
   glbl_sector: integer;                {sector read}
   buffer: buf;                         {buffer for disk info}
   regs: result;                        {for DOS interrupts}


{
.................................
.
. General subroutines
.
.................................
}


{
: DiskRead
:
: This function reads sectors from disk using DOS interrupt 25h.
: Drive is a number indicating the disk drive, where 0 = A,
: 1 = B, ...  Sector is the starting sector, no_sects tells how
: many sectors to read, and buffer is the disk buffer where the
: data will be stored.
:
: Returns a zero if the read was successful, otherwise returns a
: number telling an error code.
}

function DiskRead(drive:byte; sector,no_sects:integer;
                  var buffer:buf):integer;
  begin

{$IFDEF Turbo3}
    inline ($55/          {     push    bp}
            $88/$EC/      {     mov     bp,sp}
            $83/$C5/$08/  {     add     bp,8            ;pt to vars}
            $1E/          {     push    ds}
            $8B/$5E/$00/  {     mov     bx,[bp]         ;offset of buff}
            $8B/$46/$02/  {     mov     ax,[bp+2]       ;segment of buf}
            $8E/$D8/      {     mov     ds,ax}
            $8B/$4E/$04/  {     mov     cx,[bp+4]       ;number of sect}
            $8B/$56/$06/  {     mov     dx,[bp+6]       ;first sector}
            $8A/$46/$08/  {     mov     al,[bp+8]       ;drive}
            $55/          {     push    bp              ;save bp}
            $CD/$25/      {     int     25h             ;read it}
            $73/$08/      {     jnc     ok}
            $9D/          {     popf                    ;pop disk flags}
            $5D/          {     pop     bp              ;restore bp}
            $89/$46/$0A/  {     mov     [bp+10],ax      ;return error code}
            $EB/$08/$90/  {     jmp     cont}
            $9D/          {ok:  popf                    ;pop disk flags}
            $5D/          {     pop     bp              ;restore bp}
            $33/$C0/      {     xor     ax,ax           ;0}
            $89/$46/$0A/  {     mov     [bp+10],ax      ;return no erro}
            $1F/          {cont: pop    ds              ;restore ds}
            $5D);         {     pop     bp              ;restore bp}
```

```
{$ELSE}
     inline (
            $55/          {     push    bp}
            $8B/$EC/      {     mov     bp,sp   }
            $83/$C5/$08/  {     add     bp,8            ;pt to vars}
            $1E/          {     push    ds}
            $8B/$5E/$00/  {     mov     bx,[bp]         ;offset of buff}
            $8B/$46/$02/  {     mov     ax,[bp+2]       ;segment of buf}
            $8E/$D8/      {     mov     ds,ax}
            $8B/$4E/$04/  {     mov     cx,[bp+4]       ;number of sect}
            $8B/$56/$06/  {     mov     dx,[bp+6]       ;first sector}
            $8A/$46/$08/  {     mov     al,[bp+8]       ;drive}
            $55/          {     push    bp              ;save bp}
            $CD/$25/      {     int     25h             ;read it}
            $73/$08/      {     jnc     ok}
            $9D/          {     popf                    ;pop disk flags}
            $5D/          {     pop     bp              ;restore bp}
            $89/$46/$FA/  {     mov     [bp-6],ax       ;return error code}
            $EB/$08/$90/  {     jmp     cont}
            $9D/          {ok:  popf                    ;pop disk flags}
            $5D/          {     pop     bp              ;restore bp}
            $33/$C0/      {     xor     ax,ax           ;0}
            $89/$46/$FA/  {     mov     [bp-6],ax       ;return no error}
            $1F/          {cont: pop    ds              ;restore ds}
            $5D);         {     pop     bp              ;restore bp}
{$ENDIF}
   end;


{
: DiskWrite
:
: This function writes sectors to disk using DOS interrupt 26h.  The
: parameters and return value are the same as for DiskRead.
}

function DiskWrite(drive:byte; sector,no_sects:integer;
                   var buffer:buf):integer;
   begin
{$IFDEF Turbo3}
     inline ($55/          {     push    bp}
            $8B/$EC/      {     mov     bp,sp}
            $83/$C5/$08/  {     add     bp,8            ;pt to vars}
            $1E/          {     push    ds}
            $8B/$5E/$00/  {     mov     bx,[bp]         ;offset of buff}
            $8B/$46/$02/  {     mov     ax,[bp+2]       ;segment of buf}
            $8E/$D8/      {     mov     ds,ax}
            $8B/$4E/$04/  {     mov     cx,[bp+4]       ;number of sect}
            $8B/$56/$06/  {     mov     dx,[bp+6]       ;first sector}
            $8A/$46/$08/  {     mov     al,[bp+8]       ;drive}
            $55/          {     push    bp              ;save bp}
            $CD/$26/      {     int     26h             ;write it}
            $73/$08/      {     jnc     ok}
            $9D/          {     popf                    ;pop disk flags}
            $5D/          {     pop     bp              ;restore bp}
            $89/$46/$0A/  {     mov     [bp+10],ax      ;return error code}
            $EB/$08/$90/  {     jmp     cont}
            $9D/          {ok:  popf                    ;pop disk flags}
            $5D/          {     pop     bp              ;restore bp}
            $33/$C0/      {     xor     ax,ax           ;0}
            $89/$46/$0A/  {     mov     [bp+10],ax      ;return no erro}
            $1F/          {cont: pop    ds              ;restore ds}
            $5D);          {    pop     bp              ;restore bp}
{$ELSE}
     inline ($55/          {     push    bp}
            $8B/$EC/      {     mov     bp,sp}
            $83/$C5/$08/  {     add     bp,8            ;pt to vars}
            $1E/          {     push    ds}
            $8B/$5E/$00/  {     mov     bx,[bp]         ;offset of buff}
```

```
            $8B/$46/$02/  {     mov     ax,[bp+2]       ;segment of buf}
            $8E/$D8/      {     mov     ds,ax}
            $8B/$4E/$04/  {     mov     cx,[bp+4]       ;number of sect}
            $8B/$56/$06/  {     mov     dx,[bp+6]       ;first sector}
            $8A/$46/$08/  {     mov     al,[bp+8]       ;drive}
            $55/          {     push    bp              ;save bp}
            $CD/$26/      {     int     26h             ;write it}
            $73/$08/      {     jnc     ok}
            $9D/          {     popf                    ;pop disk flags}
            $5D/          {     pop     bp              ;restore bp}
            $89/$46/$FA/  {     mov     [bp-6],ax       ;return error code}
            $EB/$08/$90/  {     jmp     cont}
            $9D/          {ok:  popf                    ;pop disk flags}
            $5D/          {     pop     bp              ;restore bp}
            $33/$C0/      {     xor     ax,ax           ;0}
            $89/$46/$FA/  {     mov     [bp-6],ax       ;return no erro}
            $1F/          {cont: pop    ds              ;restore ds}
            $5D);          {    pop     bp              ;restore bp}
{$ENDIF}

        {Under some conditions, DOS may not update its internal disk
         buffers, even though the disk itself has been changed with
         a DOS call.  Adding the flush buffer call here corrects for
         this condition.}

        regs.ax := $D00;
        intr($21,regs);

    end;



{
: beep
:
: Makes an error tone.
}


procedure beep;
   begin
      sound(400);
      delay(500);
      sound(300);
      delay(500);
      nosound;
   end;



{
: display_start_screen
:
: Displays the main menu.
}

procedure display_start_screen;
   begin
      window(1,1,80,25);
      clrscr;
      writeln('====================Disk Examiner=======================');
      writeln;
      writeln;
      writeln;
      writeln;
      writeln('---------------------MAIN MENU----------------------');

      {add command information for later modules here}
```

```
      writeln;
      writeln('F10:  Exit');
   end;



{
.................................
.
. Load in other DiskExplorer files
.
.................................
}

{add $I statements for later modules here}



{
.................................
.
. Main
.
.................................
}


begin
   glbl_drive := 0;        {initialize global drive and sector}
   glbl_sector := 0;
   finished := false;

   while not finished do begin
      display_start_screen;                  {display menu}
      regs.ax := 0;                          {wait for key}
      intr($16,regs);
      if regs.ax and $ff = 0 then begin      {is it a function key?}
         clrscr;
         case (regs.ax) of

            {add commands for calling other modules here}

            $4400:  {F10}
                    finished := true;
          end;
      end;
   end;
end.
```

输入这段程序，以 explorer.pas 名存盘。还可对其进行编译和运行。

程序设计要点：

- 使用中断读键盘和进行磁盘扇区读写。
- Turbo Pascal 和汇编语言的接口用 inline 和 intr 命令。
- DOS 用逻辑扇区号访问扇区。

第四章　扇区分析

前面你已学会怎样读写磁盘扇区，这章将增加一个模块，能让你选择一个扇区，显示它包含的数据，对其进行修改。这样你对磁盘会更加了解。

每一扇区由 512 个字节组成。这些字节可以是文本数据，一个程序或某种编码信息。你将编写一程序打印出每个字节的十六进制和 ASCII 码值，这将使你更清楚地知道磁盘上有什么类型的信息，然后还要增加另一个显示数据的方式。

一次可显示 256 字节，每屏有 16 行，每行 16 个字节。在屏幕的左边显示十六进制值，在右边显示 ASCII 值。为便于读十六进制值，每个十六进制值间有一空格。（见图 4.1）

```
0F 03 00 00 00 00 FF FF 05 02 06 02 07 02 FF FF  ..............
FF FF 0A 02 0B 02 0C 02 0D 02 0E 02 FF FF FF FF  ..........
11 02 12 02 13 02 14 02 15 02 16 02 FF FF 18 02  ..............
19 02 1A 02 FF FF FF FF 1D 02 1E 02 1F 02 20 02  .......... .
FF FF 22 02 23 02 24 02 FF FF FF FF FF FF 28 02  ".#.$.(.
29 02 2A 02 2B 02 2C 02 2D 02 2E 02 2F 02 30 02  ).*.+.,.-.../.0.
31 02 32 02 33 02 34 02 35 02 36 02 37 02 38 02  1.2.3.4.5.6.7.8.
39 02 3A 02 3B 02 3C 02 3D 02 3E 02 3F 02 40 02  9.:.;.<.=.>.?.@.
41 02 42 02 43 02 44 02 FF FF 46 02 47 02 48 02  A.B.C.D.F.G.H.
49 02 4A 02 4B 02 4C 02 4D 02 4E 02 4F 02 50 02  I.J.K.L.M.N.O.P.
51 02 52 02 53 02 54 02 55 02 56 02 FF FF 58 02  Q.R.S.T.U.V.X.
59 02 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  Y.
FF FF FF FF FF FF FF FF FF FF 66 02 FF FF 68 02  f.h.
69 02 6A 02 6B 02 6C 02 6D 02 FF FF 6F 02 FF FF  i.j.k.l.m.o.
71 02 72 02 73 02 74 02 75 02 76 02 77 02 FF FF  q.r.s.t.u.v.w.
FF FF FF FF FF FF 7C 02 7D 02 7E 02 FF FF 6C 02  ¦.}.~.l.

Drive: 2   Sector: 3

--------------------SECTOR DISPLAY--------------------
F1: Change Drive   F2: Read Sector    F3: Read Previous
F4: Read Next      sh-F5: Write       F6: Toggle Style
F10: Exit
```

图 4.1　一个扇区转储

因为一次只能显示 256 个字节，因此需要一个命令来切换扇区的前后 256 字节。使用 PgUp 和 PgDn 键，还需要能改变驱动器号和扇区号的命令。

最后，想对磁盘上的数据进行修改，这就需要一个能在屏幕上移动的光标，一种输入新数据的方法，显示在扇区中应当修改哪一字节的方法，及往磁盘写数据的方法。

为显示数据，把屏幕分成两部分——十六进制部分和 ASCII 部分。ASCII 码显示从第 50 列开始，对显示的每一字节，在左边打印其两字节的十六进制码值，跳到右边对应的列，显示其 ASCII 码值。若该字节是一个控制码（例它的 ASCII 码值小于 32），则只显示一个句点。因此，当你想显示一个有特别意义的字符时，如制表格或换行键，屏幕就不会被破坏。

用变量存贮光标的 x 和 y 值，用 gotoxy 命令定位光标。用四个箭头键移动光标。如果键入一个普通键——即不是光标或功能键时，则认为用户想把当前光标上的字节的值改为键入的键的值，如果光标是在显示屏幕的十六进制一边，就可看到光标处的十六进制数，看到键入的值是否是个有效的十六进制数字，并可修改这个数的高或低位数字。然后把光标移动右边，以

易于输入几个数字。如果光标在屏幕的右边，你就能看到光标处的字符，及该字符被改为刚刚键入的字符。

如你所见，你需要用程序来决定光标在屏幕的哪一边，完成从十六进制字符到数字、从数字到十六进制字符以及从数字到 ASCII 字符的转换，还需要一个程序循环跳过磁盘缓冲区中的 256 个字节。

一开始是读 0 盘的 0 扇区，接着就要有命令来输入新的扇区号，读上一个扇区或读下一个扇区。注意你用的是磁盘号而不是字母(0=A，1=B，C=2 等等)。

现在，试试下面这段程序。把它放到一个叫做 sector. pas 的程序模块中，然后加到 explorer. pas 程序里。

```
{.....................SECTOR.PAS.....................
.
. This file contains routines for reading, decoding,and
. writing disk sectors.
.
.....................................................}



{
.................................
.
. General subroutines
.
.................................
}



{
: hex
:
: Returns the hex character corresponding to num.
: Returns ERROR if num >= 16.
:
}

function hex(num:byte): char;
   begin
      if (num <= 0) or (num > 15) then
         hex := '0'
      else if (num > 0) and (num <= 9) then
         hex := chr(ord('1') + num - 1)
      else
         hex := chr(ord('A') + num - 10);
   end;



{
: unhex
:
: Returns the number corresponding to a hex character.
: Returns ERROR if not a hex character.
:
}

function unhex(letter:char): integer;
   begin
      if letter in ['0'..'9'] then
         unhex := ord(letter) - ord('0')
      else if letter in ['A'..'F'] then
         unhex := ord(letter) - ord('A') + 10
```

```
      else if letter in ['a'..'f'] then
         unhex := ord(letter) - ord('a') + 10
      else
         unhex := ERROR;
   end;



{
: get_new_drive
:
: Reads in the number for a new drive.  Tries to read in a sector to
: see if it is a valid drive.  Continues prompting for a drive number
: until the read is good.
:
: Sets glbl_drive to the new drive number.
:
}

procedure get_new_drive(var data:buf);
   var
      fin: boolean;
   begin
      window(COMMANDX,COMMANDY,COMMAND_RT,COMMAND_BOT);
      clrscr;
      fin := false;
      {  note - when switching from hard drive to
         floppy, will have to first move sector no. to
         a decent sector number, or else will give errors}
      while not fin do begin
         write('Drive: ');
         readln(glbl_drive);
         if diskread(glbl_drive,glbl_sector,1,data) <> 0 then
            writeln('ERROR -- try again')
         else
            fin := true;
      end;
   end;



{
---------------------------------
.
. Routines to figure cursor location
.
---------------------------------
}


{
: in_hex_side
:
: Returns true if the cursor is in the hex display side and is over
: a hex digit during a hex dump.
:
}

function in_hex_side(curs_x,curs_y: integer): boolean;
   begin
      if (curs_x >= LFTSCRN) and (curs_x div 3 < 16) {on ASCIISIDE}
            and (curs_y <= BOTSCRN) and (curs_y >= TOPSCRN)
            and (curs_x mod 3 <> 0) then              {not blank space}
         in_hex_side := true
      else
         in_hex_side := false;
   end;
```

```
{
: in_ascii_side
:
: Returns true if the cursor is in the ASCII display side during
: a hex dump.
:
}

function in_ascii_side(curs_x,curs_y: integer): boolean;
   begin
      if (curs_x >= ASCIISIDE) and (curs_x <= ASCIISIDE + 15)
           and (curs_y <= BOTSCRN) and (curs_y >= TOPSCRN) then
         in_ascii_side := true
      else
         in_ascii_side := false;
   end;



{
.................................
.
. Routines to display data
.
.................................
}


{
: display_status
:
: Prints the drive and sector being examined.
:
}

procedure display_status;
   begin
      window(STATUSX,STATUSY,STATUS_RT,STATUS_BOT);
      clrscr;
      writeln('Drive: ',glbl_drive,'   Sector: ',glbl_sector);
   end;


{
: prt_data
:
: Prints a byte in hex and the corresponding ASCII value.  Puts the
: hex number in the left side of the screen, and the ASCII value in
: the right.
:
}

procedure prt_data(row,column:integer;value:byte);
   begin
      gotoxy(column,row);
      write(hex(value shr 4));       {high hexit}
      write(hex(value and $f));      {low hexit}
      gotoxy((column div 3) + ASCIISIDE,row);        {to right side of screen}
      if value < 32 then
         write('.')
      else
         write(chr(value));
   end;


{
: display_page
:
```

```
: Displays the hex and ASCII values of 256 bytes from the data buffer.
: The values are displayed in 16 rows, each containing 16 values.  The
: hex numbers are displayed in the left side of the screen, each as two
: digits followed by a space.  The ASCII characters are displayed in the
: right side of the screen, with no separator.  That is, a display may
: look like:
:
: 41 42 43  ... ...  ... ... ...   ABC ... ...
: 70 6f  ..   ..   ..   ..    .    po ... ..
:
: The values are taken starting with the offset-th byte.
:
}

procedure display_page(var data:buf;offset:integer);
   var
      row,column:integer;
      value:byte;
   begin
      window(LFTSCRN,TOPSCRN,RTSCRN,BOTSCRN);
      clrscr;
      for row := 0 to 15 do
         for column := 0 to 15 do begin
            value := data[offset + row*16 + column];
            prt_data(row+1,3*column+1,value);
         end;
   end;


{
: display_data
:
: This procedure checks the style value to see whether to do a
: hex dump or to do a directory decoding of the data.
:
}

procedure display_data(var data: buf;offset,style: integer);
   begin
      if style = HEX_DUMP then
         display_page(data,offset*256)
   end;



{
.................................
.
. Routines to change data in the buffer
.
.................................
}



{
: adjust_hex_data
:
: This routine is called to change the data in the buffer during a
: hex dump.  It determines whether the cursor is over a hex byte or
: an ASCII character, adjusts the old value according to the input_data,
: and then writes that into the proper location in the data buffer.
}

procedure adjust_hex_data(var curs_x,curs_y,input_data: integer;var data:buf;
            offset: integer);
   var
      d_loc,d_val: integer;
   begin
      if in_hex_side(curs_x,curs_y) then begin    {cursor is in hex side}
```

```
      d_loc := (curs_x - 1) div 3 + (curs_y - 1)*16;
      {.....what was the old value?.....}
      d_val := unhex(chr(input_data and $ff));
      if d_val <> ERROR then begin
         if curs_x mod 3 = 1 then    {high hexit}
            data[d_loc + offset] := (data[d_loc + offset] and $f)
                 + d_val*16
         else   {low hexit}
            data[d_loc + offset] := (data[d_loc + offset] and $f0) + d_val;
         prt_data(1 + d_loc div 16,(d_loc mod 16)*3 + 1,
                  data[d_loc + offset]);   {update the display}
         curs_x := curs_x + 1;   {advance the cursor}
       end
    end
    else if in_ascii_side(curs_x,curs_y) then begin   {ascii side}
       d_loc := curs_x - 50 + (curs_y - 1)*16;
       data[d_loc + offset] := input_data and $ff;
       prt_data(1 + d_loc div 16,(d_loc mod 16)*3 + 1,data[d_loc + offset]);
       curs_x := curs_x + 1;
    end;
  end;




{
.....................................
.
. Main sector examining routines
.
.....................................
}


{
: display_dump_commands
:
: This procedure displays the sector examining menu.
:
}

procedure display_dump_commands;
   begin
      window(COMMANDX,COMMANDY,COMMAND_RT,COMMAND_BOT);
      clrscr;
      writeln('--------------------SECTOR DISPLAY--------------------');
      writeln('F1: Change Drive   F2: Read Sector   F3: Read Previous');
      writeln('F4: Read Next      sh-F5: Write');
      writeln('F10: Exit');
   end;


{
: sector_dump
:
: This procedure is the main sector examining procedure.  It initializes
: some variables, then reads and acts upon the sector examining commands.
:
}

procedure sector_dump(var data: buf);
   var
      fin,finished: boolean;
      offset,curs_x,curs_y,style: integer;
      regs: result;
   begin
      finished := false;
      curs_x := ASCIISIDE;          {start cursor in the ASCII section}
```

```
curs_y := 1;
offset := 0;
style := HEX_DUMP;              {initially, do hex dumps}
if diskread(glbl_drive,glbl_sector,1,data) <> 0 then     {read sector}
   beep;
display_dump_commands;
display_status;
display_page(data,offset);
while not finished do begin
   gotoxy(curs_x,curs_y);
   regs.ax := 0;
   intr($16,regs);          {get command}
   if regs.ax and $ff = 0 then    {is it a function key?}
      case (regs.ax) of
         $4800:  {up arrow -- move cursor up one row}
                 if curs_y > TOPSCRN then
                    curs_y := curs_y - 1;
         $4b00:  {left arrow -- move cursor left}
                 if curs_x > LFTSCRN then
                    curs_x := curs_x - 1;
         $4d00:  {right arrow -- move cursor right}
                 if curs_x < RTSCRN then
                    curs_x := curs_x + 1;
         $5000:  {down arrow -- move cursor down one row}
                 if curs_y < BOTSCRN then
                    curs_y := curs_y + 1;
         $4900,$5100:  {page up,down}
                 begin
                    offset := (offset + 1) mod 2;
                    display_data(data,offset,style);
                 end;
         $3b00:  {F1  change drive}
                 begin
                    get_new_drive(data);
                    display_dump_commands;
                    display_status;
                    display_data(data,offset,style);
                 end;
         $3c00:  {F2  read sector}
                 begin
                    window(COMMANDX,COMMANDY,COMMAND_RT,COMMAND_BOT);
                    clrscr;
                    fin := false;
                    while not fin do begin       {read in new number}
                       write('Sector: ');
                       readln(glbl_sector);
                       if diskread(glbl_drive,glbl_sector,1,data)
                             <> 0 then          {check for validity}
                          writeln('ERROR -- try again')
                       else
                          fin := true;
                    end;
                    display_dump_commands;
                    display_status;
                    display_data(data,offset,style);
                 end;
         $3d00:  {F3  read previous sector}
                 if glbl_sector > 0 then begin
                    glbl_sector := glbl_sector - 1;
                    if diskread(glbl_drive,glbl_sector,1,data) <> 0 then
                       beep;
                    display_status;
                    display_data(data,offset,style);
                 end;
         $3e00:  {F4  read next sector}
                 begin
                    glbl_sector := glbl_sector + 1;
                    if diskread(glbl_drive,glbl_sector,1,data) <> 0 then
                       {invalid or bad sector}
                       beep;
                    display_status;
```

```
                    display_data(data,offset,style);
                  end;
          $5800:  {Shift-F5  write sector to disk}
                  if diskwrite(glbl_drive,glbl_sector,1,data) <> 0 then
                    beep;
          $4400:  {F10 exit}
                    finished := true;
        end
      else if style = HEX_DUMP then    {not a command}
{$IFDEF Turbo3}
        adjust_hex_data(curs_x,curs_y,regs.ax,data,offset*256)
{$ELSE}
        adjust_hex_data(curs_x,curs_y,integer(regs.ax),data,offset*256)
{$ENDIF}
    end;
  end;
```

注意 display—status,display—page,display—dump—command 是怎样使用窗口的。还要注意是怎样把功能键和箭头键做为命令来使用的,全局值 glbl—drive 和 glbl—sector 的使用方法及是怎样把磁盘信息送入缓冲区的,DiskRead 和 DiskWrite 的使用方法和显示过程的操作方式。运行全部过程,更好地了解它们的操作方法。

在你使用这个新模块之前,要对 explorer.pas 作些改动。把该模块作为 sector.pas 存上,装入 explorer.pas 并做如下改动:

在常量说明部分,增加:

```
ASCII SIDE = 50;
HEX_DUMP = 2;
```

在 display—start—screen 中增加:

```
writeln('F1:   Dump and Modify Sectors');
```

在包括其它 DiskExplorer 文件的部分增加:

```
{$I b:sector.pas}
```

若想把你的源盘放在另外的驱动器里,例如 A 或 C,要适当地修改这一行。在主程序部分的 case 语句中,增加:

```
$3b00:  {F1}
        sector_dump(buffer);
```

使用这个程序

做了这些改动之后,存盘,编译,改正输入错误并运行。记住,即使只是修改 sector.pas 里的错误,每次也都要编译 explorer.pas。

运行 EXPLORER 时,可看到屏幕上的启动菜单。按 F1 键转储并修改扇区。接着探查一下你的磁盘。先看看 0 扇区,5 扇区,再看 23 和 35 扇区。再查看另外一个磁盘或驱动器。每个扇区的前、后 256 个字节都要看。你很可能看到各种有趣的信息。例如看 0 扇区时,会看到格式化磁盘所用的 DOS 版本。如果看一个软盘的第 7 扇区,你能看到磁盘上的文件名。下面几章将详细讨论这些问题。

程序设计要点：

- 扇区是磁盘的最小组织单元，它包括 512 字节。
- 可通过编辑扇区看到感兴趣的信息和修改程序。
- EXPLORER 使用箭头键移动光标。

第五章　引导记录

现在准备观察引导记录——磁盘的四种组织结构之一。引导记录在 0 扇区。它包含描述磁盘的一组参数和一个引导计算机的程序，在这章里你将用 EXPLORER 观察引导记录。

最开始的三个字节是一条跳到引导程序的机器语言命令。该程序使磁盘复位，检查引导记录参数，装入部分根目录以检查 IBMBIO. COM 和 IBMDOS. COM 是第一个文件，接着读并激活这些程序。这个程序之后是文本信息，引导记录的结束标志为 55h 和 AAh。

跳指令之后是一组描述磁盘的参数：

引导记录参数

字　节	含　　义
(0)	(Jump 指令)
3—10	格式化磁盘的 DOS 版本的 8 字母名字
11—12	每扇区字节数
13	每簇扇区数
14—15	引导记录的扇区数
16	FAT 的个数
17—18	根目录表项数
19—20	每个磁盘的扇区数
21	磁盘类型
22—23	每个 FAT 的扇区数
24—25	每磁道的扇区数
26—27	每磁盘的面数
28—29	保留扇区数

磁盘类型字节的描述如下：

值	磁盘类型
F8h	硬盘
F9h	倍倍密度或 3.5"
FCh	单面，每道 9 扇区
FDh	双面，每道 9 扇区
FEh	单面，每道 8 扇区
FFh	双面，每道 8 扇区

利用 EXPLORER 观察几个磁盘的引导记录。尽可能观察多些不同类型的磁盘。一般有以下几种类型：

类　型	扇　区	扇区/簇	根目录表项	FAT 扇区数	扇区/道
ss,8/道	320	1	64	1	8
ds,8/道	640	2	112	1	8
ss,9/道	360	1	64	2	9
ds,9/道	720	2	112	2	9
倍倍密度	2400	1	224	7	15
3.5"	1440	2	112	3	9
3.5"高密盘	2880	1	224	9	18
10MB 硬盘	20740	8	512	8	17
20MB 硬盘	41752	4	512	41	17
30MB 硬盘	62203	4	512	61	17
RAM	1984	1	128	6	9

硬盘的参数根据分区建立和硬盘的类型而有所不同。磁盘的每扇区有 512 字节，两个 FAT 备份。软磁盘都有两面，没有保留扇区。RAM 磁盘参数变化较大。

实用程序通过读引导记录，可以知道有关磁盘格式的重要信息。例如，引导记录的第一个扇区的地址能从引导记录的大小，FAT 表的大小，及 FAT 表的个数计算出来。引导记录和保留扇区之后是第一个数据扇区。扇区数能告诉你磁盘的大小或分区情况。如果一个程序要改变 FAT，它可以通过查看引导记录来确定需要改变的个数。在编写 EXPLORER 时，要经常用到引导记录内的信息。

引导记录和 IBM PC 兼容机

引导记录的格式在一些 IBM PC 兼容机中有稍许不同。一些 Leading Edge 计算机使用的 DOS 2.11－1.03 版本没有 DOS 版本的 8 字母名字。磁盘信息的其余部分是相同的，只是它是从字节 3 开始的。DOS 2.11－1.04 版本就不是这样。如果你在兼容机上用 EXPLORER 读引导记录时得到一些奇怪的结果，要检查引导记录格式，看看是否没有 DOS 版本的 8 字母名字，若是这样，适当修改 EXPLORER。你还可以修改 EXPLORER，利用中断 21h 的 50 功能调用（见 21 章）得到磁盘信息而不读引导记录。

BOOT.PAS

现在给 EXPLORER 增加一个译码引导记录里的信息的模块，这个 boot.pas 模块功能是读磁盘的 0 扇区，并打印出它在那儿找到的一张信息表。

```
{.....................BOOT.PAS.....................
.
. This file contains routines for decoding the boot record
.
.............................................. ........}


{
: display_boot_data
:
: This procedure decodes the data in the boot record.
```

```
: See the text for an explanation of the boot record format.
:
}

procedure display_boot_data(var data: buf);
   var
      i: integer;
   begin
      window(LFTSCRN,TOPSCRN,RTSCRN,BOTSCRN);
      clrscr;
      write('Formatted with ');
      for i := 3 to 10 do
         write(chr(data[i]));
      writeln;
      writeln('There are ',data[11] + 256*data[12],' bytes per ',
              'sector');
      writeln('There are ',data[13],' sectors per cluster');
      writeln('There are ',data[14] + data[15]*256,' sectors in ',
              'the boot record');
      writeln('There are ',data[16],' copies of the FAT');
      writeln('Can be up to ',data[17] + data[18]*256,' entries in ',
              'the root directory');
      writeln('There are ',data[19] + int(data[20])*256:6:0,
              'sectors');
      write('The disk is ');
      case data[21] of
         $f8:  writeln('a hard disk');
         $f9:  writeln('a quad density or 3.5 inch');
         $fc:  writeln('single-sided, 9 sectors/track');
         $fd:  writeln('double-sided, 9 sectors/track');
         $fe:  writeln('single-sided, 8 sectors/track');
         $ff:  writeln('double-sided, 8 sectors/track');
         else writeln;
      end;
      writeln('There are ',data[22] + data[23]*256,' sectors in ',
              'the FAT');
      writeln('There are ',data[24] + data[25]*256,' sectors/ ',
              'track');
      writeln('There are ',data[26] + data[27]*256,' sides');
      writeln('There are ',data[28] + data[29]*256,' reserved ',
              'sectors');
   end;


{
: display_boot_commands
:
: This procedure displays the boot record examining commands.
:
}

procedure display_boot_commands;
   begin
      window(COMMANDX,COMMANDY,COMMAND_RT,COMMAND_BOT);
      clrscr;
      writeln('-----------------BOOT DISPLAY-----------------');
      writeln('F1: Change Drive                    F10: Exit');
   end;


{
: boot_dump
:
: This is the main routine.  It displays the menus and reads and
: acts upon commands.
:
}
```

```
procedure boot_dump(var data: buf);
  var
     finished:boolean;
     regs: result;
  begin
     finished := false;
     glbl_sector := 0;                    {boot record starts here}
     if diskread(glbl_drive,glbl_sector,1,data) <> 0 then {read it}
        beep;
     display_boot_commands;
     while not finished do begin
        display_boot_data(data);          {display data}
        display_status;
        regs.ax := 0;                     {get command}
        intr($16,regs);
        if regs.ax and $ff = 0 then
           case (regs.ax) of
              $3b00:  {F1 new drive}
                      begin
                         get_new_drive(data);
                         display_boot_commands;
                         display_status;
                      end;
              $4400:  {F10 exit}
                      finished := true;
            end;
     end;
  end;
```

输入这段程序后，存盘，还需在 explorer. pas 中增加如下内容：

在 display—start—screen 中，增加

```
writeln('F2:   Examine Boot Record');
```

然后加上

```
{$I b:boot.pas}
```

若需要可改变驱动器名。在主程序中加上

```
$3c00:  {F2}
       boot_dump(buffer);
```

做完这些增加后，重新存 explorer. pas，编译，并改正输入错误，然后运行它。用 F2 命令观察你以前看过的相同磁盘的引导记录数据。

程序设计要点：

- 引导记录包含有关磁盘大小和磁盘组织结构大小的重要信息。
- 引导记录包含一个启动 DOS 的短程序。
- 实用程序通过读引导记录，可知道有关磁盘的重要信息。
- 与 IBM 兼容的计算机 DOS 版本，如 DOS 2. 11—1. 03 引导记录格式有所不同。

第六章　文件分配表

这章你将观察文件分配表——DOS 用此表来决定每个文件使用哪部分磁盘，哪部分是空的，哪部分是坏的。但首先，需要看看 DOS 在磁盘上存贮文件的方式。

磁盘的开始扇区是为引导记录、文件分配表 FAT 和目录保留的。磁盘的其余部分叫做**数据区**，用来存贮文件。DOS 把数据区分成相邻的编号的扇区组，叫做**簇**。DOS 知道哪些簇没有使用，当生成一个文件时，DOS 寻找一个空簇，并把它分配给该文件。这个簇就是该文件开始存贮数据的地方。如果文件长度大于一个簇，DOS 就寻找下一个空簇，也把它分配给该文件，此过程继续下去，直到该文件的数据全部存完。

簇是最小的分配单元，即使一个文件只有三字节长，它也要占用磁盘的一个整簇。如你在第 5 章看到的一样，簇的大小从 1－8 个扇区不等。DOS 在其需要保持的簇数和被部分充满的簇所浪费的空间大小之间做一权衡折衷，选择一簇的大小。

FAT 表的每一项都对应于一个簇。每项告诉你该簇是否是空的，是否是文件的一部分，是否是保留的或是否是坏的。若该簇是一个文件的一部分，该项就会告诉你文件下一簇的簇号或文件结束值。表示空的，保留的或坏的扇区的值则在一个文件的有效簇号范围之外，也就是，它们或是大于磁盘的簇数或是小于第一簇号，这样就不会混乱。

例如，假定你生成了一个新文件，分配的第一个簇的簇号为 10。当你往文件写数据时，你就是往第 10 簇里的扇区中写。同时，若 10 簇是该文件的最后一簇，10 簇的 FAT 表项中将包含一个文件结束标记。假设你的文件需要不止一个簇：DOS 从第 10 个表项开始搜索 FAT 表，寻找下一个空簇。假设找到的这个空簇是 15 簇，你的文件就将开始往 15 簇的扇区中继续写数据。FAT 10 簇的表项就变成 15，而 15 簇的表项就变成文件结束标志。

为了确定一个文件所使用的簇数，只需跟踪 FAT 表，从簇表项跳到簇表项，这样生成的表叫做**文件链**。

数据区的第一组扇区叫做簇 2，下一组叫簇 3，以此类推。例如，双面软盘每道有 9 个扇区，数据区从 12 扇区开始，每簇有两个扇区，因此扇区 12－13 是簇 2，扇区 14－15 是簇 3，等等。簇从 2 开始编号，是为了更容易地访问文件分配表，这个你一会儿将看到。

查看引导记录可确定数据区的起始位置。数据区的第一扇区是：

引导记录所用的扇区数＋FAT 个数×FAT 所用的扇区数＋
引导目录中的表项数×32/每扇区字节数＋保留扇区。

12 位和 16 位 FAT 表

现在更仔细地观察 FAT 表。有两种类型的 FAT 表——12 位和 16 位的。12 位的 FAT 用于“小磁盘”，象软盘和 10MB 硬盘。16 位 FAT 用于“大磁盘”，象 20MB 和 30MB 硬盘。12 位 FAT 用 12 位存贮每个簇代码，因此它能寻址到 4096 个簇（减去一些状态代码簇）。16 位 FAT 用一个字存贮每个簇代码，因此它能寻址到 65,536 个扇区。开发 16 位 FAT 是为了能使用与

20MB 硬盘的扇区一样多的磁盘，而不用增加簇的大小。

DOS 在格式化磁盘时要决定该磁盘是使用 12 位还是 16 位的 FAT。DOS 有两种决定方法。第一种，若磁盘的扇区数多于 31,110，就用 16 位 FAT，这对应于 15MB 的磁盘（注意在 DOS 手册中关于这点的叙述是不正确的。实际上能寻址 16MB 数。因为选择每个分区的扇区数是不可能的，所以实际扇区的结束差别很小。DOS 技术手册给出了 FDISK 结束值，用这种方法计算 FORMAT 结束值与在 DOS 手册中计算 FDISK 结束值得到的结果一致）。若磁盘扇区数小于 31,110，就用 12 位 FAT。第二种方法见第 10 章“分区表”。

有趣的是，在硬盘上建立分区的程序 FDISK 也要决定 DOS 应当使用的 FAT 类型，但 FDISK 使用一个不同的标准。对有 20,740 个以上扇区的磁盘，它都选择 16 位 FAT。当格式化分区表时，DOS 又根据 FORMAT 标准改变 FAT 类型。

在 12 位 FAT 中每三个字节存贮两个簇的信息。FAT 的第一个字节包含磁盘类型码——与引导记录中的相同。下两个字节都是 ffh，再下一个字节是文件信息开始的第一簇，编号为 2。注意，如果你认为前三个字节包含簇 0 和簇 1 的信息，那么簇 2 的信息应当从簇 2 表项开始。这就是为什么簇从 2 编号开始的原因。下一簇的信息顺序其后。

因为每簇表项占用一个半字节，所以簇是三字节一组来编码的，为译码该值：

1）用 3 乘以簇值

2）结果除以 2 取整

这样得到 FAT 表中的一个偏移量。

3）如果簇号是偶数，装入 FAT 表中此偏移量指向的字，并和 ffh 相与。

如果簇号是奇数，装入 FAT 表中此偏移量指向的字，并右移四位。

结果 FAT 就象下面这样：

```
xx ff ff    yy yy yy   yy yy yy   yy yy yy  .....
Disk type   group 1    group 2    group 3
```

每一组为：

```
yy yy yy
23 x1 xx        even cluster
xx 3x 12        odd cluster
```

在此，1 是最有意义的十六进制数，3 则是最没意义的。

下表列出簇表项值以及它们的含义：

值（十六进制）	含　义
000	空簇
001	正被使用的簇。因为簇是从 2 开始，所以其链号无效
002－fef	正被使用的簇。表项指向链中的下一簇
ff0－ff6	保留簇（不是空的，也不是文件链的一部分）
ff7	坏簇
ff8－fff	文件链中的最后一簇（文件结束标记）

16 位 FAT 格式较简单，每个簇表项用一个字。与 12 位 FAT 格式一样，前两个表项标识磁盘类型。第一个字节是磁盘类型码，与在引导记录中用的一样，后三个字节都是 fffh，随后的

每一个字都包含一个簇表项代码。

为译码该值，用簇号乘以 2。读出 FAT 表中此偏移量中的字，这就是译码后的值。

16 位 FAT 簇表项值的含义与 12 位 FAT 相似：

值(十六进制)	含　义
0000	空簇
0001	正在使用，其链号无效，因为簇从 2 开始编号。
0002—ffef	正在使用，表项指向链中的下一个簇。
fff0—fff6	保留簇(非空也非文件链的一部分)
fff7	坏簇
fff8—ffff	文件链中的最后一簇(文件结束标志)

12 位和 16 位的 FAT，一个簇的第一个扇区都是这样计算：

(簇号－2)×每簇的扇区数＋数据区的第一个扇区号

FAT 紧跟在引导记录之后，可通过读引导记录数据表找到它的位置，引导记录数据表决定了引导记录的大小。通常，FAT 从扇区 1 开始，因为 FAT 非常重要，所以 DOS 在 FAT 生成之后立即为其做拷贝。拷贝数可在引导记录中找到。一般有两个拷贝：原始的和备份的。

双面每道 9 扇区的磁盘(从现在起，只称做双面盘)的 FAT 从扇区 1 开始，两个扇区长，后接一个备份。用 EXPLORER 观察双面盘上的文件分配表。然后，看看它的备份，试着译码一些簇表项。注意 FAT 表开始处的磁盘类型代码。

FAT. PAS

现在看一个观察 FAT 的程序。它将确定 FAT 从哪开始，FAT 的类型及通过引导记录确定 FAT 的大小。然后将扫遍 FAT 表，一次一个扇区，以译码簇表项，将用字符代表每个簇的状态。

因为文件分配表可能非常大，所以一次将读一组，一组有九个扇区。这就是 BUFSECT 常量的含义。12 位和 16 位 FAT 都用九扇区，这样的组的最后一个字节将是一个完整的表项(9×512 能被 1.5 和 2 除，结果是偶数)。

复杂的部分是译码 12 位 FAT。FAT 表项不能整除扇区数，所以需要计算出一个调节系数以便在一簇表项组的起点开始扫描。跳过磁盘类型字节(簇表项 0 和 1)。下面这段程序是 fat. pas。输入并以文件名 fat. pas 存盘。

```
{.........................FAT.PAS.........................
.
. This file contains routines for reading and decoding the
. FAT.  It will work with both the 12 bit and 16 bit formats.
.
.........................................................}
```

```
{
.....................................
.
. General subroutines
.
.....................................
}

{
: get_disk_info
:
: This procedure reads the boot record to get information about
: the disk's format
:
}

procedure get_disk_info(var info:disk_info;var data:buf);
   begin
      if diskread(glbl_drive,0,1,data) <> 0 then  {read boot record}
         beep;
      info.fat_start := data[14] + data[15]*256;
      info.fat_copies := data[16];
      info.root_entries := data[17] + data[18]*256;
      info.sect_in_fat := data[22] + data[23]*256;
      info.fat_type := 12;
      if (data[21] = $f8) and (data[19] + int(data[20])*256 > 31110)
           then info.fat_type := 16;
      info.sect_per_clust := data[13];
    end;



{
.....................................
.
. Main fat examining routines
.
.....................................
}



{
: display_fat_data
:
: This routine graphically displays the usage of one sector from the
: FAT.  For 12 bit FATS, it computes an offset so that the data
: examined will not start in the middle of a cluster number.  It then
: reads through the sector, and displays a symbol to represent used,
: free, bad, or reserved clusters.  It skips over the disk id code.
:
: 12 bit FATs are displayed in 10 rows of up to 35 columns.
:
: 16-bit FATs are displayed as 8 rows of up to 32 columns.  The
: disk id bytes are skipped.
:
}


procedure display_fat_data(var data:buf;offset,fat_type:integer);
   var
      row,column,off_adj,fat_off,clust_val: integer;
   begin
      window(LFTSCRN,TOPSCRN,RTSCRN,BOTSCRN);
      clrscr;
      if fat_type = 12 then begin      {12 bit}
         off_adj := -(512*offset mod 3);        {don't break a clust num}
         for row := 0 to 9 do begin
```

```
            for column := 0 to 34 do
               if (row*35 + column < 512/(3/2)) and     {just display 1 sect}
                not ((row = 0) and (column < 2) and (offset = 0)) {not id}
                then begin
                  fat_off := 3*(row*35 + column) div 2 + off_adj +
                           (offset mod BUFSECT)*512;
                  if not odd(row*35 + column) then      {even cluster}
                     clust_val := data[fat_off]*16 + data[fat_off + 1] mod 16
                  else          {odd cluster}
                     clust_val := (data[fat_off] div 16)*256 +
                         data[fat_off+1];
                  case clust_val of     {print code}
                     0:           write('.');
                     1:           write('1');
                     2..4079:     write('*');
                     4080:        write('B');
                     4081..4087:  write('R');
                     4088..4095:  write('*');
                   end;
               end;
            writeln;
         end;
      end
      else begin {16 bit fat}
         for row := 0 to 7 do begin
            for column := 0 to 31 do
               if not ((row = 0) and (column < 2) and (offset = 0)) then begin
                  fat_off := row*8 + column + (offset mod BUFSECT)*512;
                  clust_val := data[fat_off] + data[fat_off+1]*256;
{$IFDEF Turbo3}
                  case clust_val of
                     0:             write('.');
                     1:             write('1');
                     2..$7fff:      write('*');
                     $8000..$ffef:  write('*');
                     $fff0..$fff6:  write('R');
                     $fff7:         write('B');
                     $fff8..$ffff:  write('*');
                   end;
{$ELSE}
                  {can no longer trick turbo into doing a case on a
                   word value}
                  if (clust_val < 0) then
                     case (clust_val and $7fff) of
                        0..$7fef:          write('*');
                        $7ff0..$7ff6:      write('R');
                        $7ff7:             write('B');
                        $7ff8..$7fff:      write('*');
                     end
                  else
                     case clust_val of
                        0:                 write('.');
                        1:                 write('1');
                        2..$7fff:          write('*');
                     end;
{$ENDIF}
               end;
            writeln;
         end;
      end;
   end;


{
: display_fat_exm_commands
:
```

```
: Display the FAT examination menu.
:
}

procedure display_fat_exm_commands;
   begin
      window(COMMANDX,COMMANDY,COMMAND_RT,COMMAND_BOT);
      clrscr;
      writeln('--------------------FAT DISPLAY---------------------');
      writeln('F1: Change Drive                    F10: Exit');
      writeln;
      writeln('. FREE   * IN USE   B BAD   R RESERVED   1 SPECIAL');
   end;

{
: fat_dump
:
: This is the main procedure.  It displays menus and data, and reads
: and acts upon commands.
:
: Because the FAT can be very large on hard disks, only a portion of the
: FAT (BUFSECT sectors) is read at a time.  The code to move through the
: FAT checks when a new section of the FAT needs to be read in.
:
}

procedure fat_dump(var data:buf);
  var
     finished:boolean;
     regs: result;
     offset_sector: integer;    {how many sectors into FAT}
     info: disk_info;
  begin
     finished := false;
     offset_sector := 0;        {start at beginning of FAT}
     get_disk_info(info,data);
     display_fat_exm_commands;
     glbl_sector := info.fat_start;
     if diskread(glbl_drive,glbl_sector,BUFSECT,data) <> 0 then {read FAT}
        beep;
     display_status;
     while not finished do begin
        display_fat_data(data,offset_sector,info.fat_type);
        display_status;
        regs.ax := 0;
        intr($16,regs);         {read command}
        if regs.ax and $ff = 0 then
           case (regs.ax) of
              $4900:  {page up -- go to previous sector}
                      if offset_sector <> 0 then begin
                         offset_sector := offset_sector - 1;
                         {check to see if need to read in another set
                          of sectors from disk.}
                         if offset_sector mod BUFSECT = BUFSECT - 1 then
                            if diskread(glbl_drive,info.fat_start +
                                offset_sector - (BUFSECT - 1),BUFSECT,data)
                                <> 0 then
                               beep;
                         {set glbl_sector to the sector being examined}
                         glbl_sector := info.fat_start + offset_sector;
                      end;
              $5100:  {page down -- go to next sector}
                      if offset_sector <> info.sect_in_fat - 1 then begin
                         offset_sector := offset_sector + 1;
                         {see if need to read in another set of sectors
                          from disk}
                          if offset_sector mod BUFSECT = 0 then
                             if diskread(glbl_drive,info.fat_start +
                                 offset_sector,BUFSECT,data)
```

```
                    <> 0 then
                   beep;
              glbl_sector := info.fat_start + offset_sector;
            end;
      $3b00: {F1  new drive}
            begin
              glbl_sector := 0;
              get_new_drive(data);
              offset_sector := 0;
              get_disk_info(info,data);
              glbl_sector := info.fat_start;
              if diskread(glbl_drive,glbl_sector,    {read FAT}
                BUFSECT,data) <> 0 then
                 beep;
              display_fat_exm_commands;
              display_status;
            end;
      $4400: {F10  exit}
            finished := true;
    end;
  end;
end;
```

还需对 explorer. pas 增加如下内容：

在类型说明中，增加：

```
{stores important facts about the disk format}
disk_info = record
    fat_start: integer;
    fat_copies: byte;
    root_entries: integer;
    sect_in_fat: integer;
    fat_type: byte;
    sect_per_clust: byte;
  end;
```

在 display—start—screen 部分增加：

```
writeln('F3:   Cluster by Cluster FAT Dump');
```

还增加：

```
{$I b:fat.pas}
```

如必要修改驱动器的位置。在主程序的 case 语句中，增加：

```
$3d00:  {F3}
      fat_dump(buffer);
```

存盘，编译，改正输入错误。然后运行 explorer。用这个程序观察几个磁盘的 FAT。翻页观察一会儿 FAT，然后返回到主菜单，再选择 Sector Dump。可看到你刚刚观察的 FAT 扇区的十六进制信息。

程序设计要点：

- 扇区组成簇。文件由簇组组成。一个文件的簇可以不相邻
- 文件分配表(FAT)告诉文件所使用的簇和未用的簇。
- 有两种 FAT：12 位和 16 位的，16 位 FAT 用于 20 和 30MB 磁盘。
- FAT 存在引导记录之后。通常，有两个 FAT。

第七章　根目录

接下来介绍的组织结构是根目录。根目录包含所有根级文件的信息，包括卷名和根级子目录。利用根目录信息可跟踪文件，恢复删除的文件。调整根目录，可以改变文件的属性，增加卷名，保护文件不被访问。

根目录紧跟在FAT之后。可通过引导记录找到它的位置，计算如下：

引导记录的大小＋FAT的扇区数×FAT个数

根目录的表项数也存在引导记录中。每个表项占用32个字节，因此，根目录的扇区数为：

根目录的表项数×32/每扇区的字节数

一般的DOS磁盘，每扇区有512字节，所以计算公式简化为：

根目录的表项数/16

用EXPLORER看看根目录的第一个扇区。双面磁盘的根目录从扇区5开始。在根目录里可以看到你所有文件的名字，还有一些无用信息。

一个目录表项中的32字节的定义如下：

字节	内　容
0—10	文件名和扩展名，为ASCII字符。三个扩展名字母前由空格填充。不包括分隔文件名和扩展名的句点。 若第一个字节是0，那么该目录表项和后面所有的目录表项都没有用到。 若第一个字节是一个句点(2eh)，第二个字节是一空格，这表项就是当前目录的表项，用来作为访问子目录的指针。 若第一、二字节都是句点，第三个字节为一空格，则该表项是父目录的表项，用来作为从子目录返回到父目录的指针。 若第一个字节是e5h，那么对应于该表项的文件已被删除。剩下的文件名字符和信息没有改变。增加新文件时首先要使用已删除的目录位置。
11	文件属性。该表项指明文件的特性，具体含义如下：

位置	含义
01	只读
02	隐含

04　系统
08　卷标号
10　子目录
20　档案
40　未用
80　未用

普通的目录搜寻看不到隐含、系统、卷标号、子目录表项。一个文件被修改时，要设档案位。硬盘在做备份拷贝时，要用到档案位，卷标号只在根目录中有意义。

12－21　当前未用，留给以后用。

22－23　文件生成时间或最后一次修改时间。编码为：

15 14 13 12 11　10 9 8 7 6 5　4 3 2 1 0

<时>　<分>　<秒/2>

译码后的信息为：

时 = (字节 23)SHR3

分 = ((字节 23)AND111b)SHL3+((字节 22)SHR5)

秒 = 2×((字节 22)AND11111b)

24－25　文件生成时或最后一次修改的日期。编码为：

15 14 13 12 11 10 9　8 7 6 5　4 3 2 1 0

<年－1980>　<月>　<日>

译码后的信息为：

年 = 1980+((字节 25)SHR1)

月 = 8×((字节 25)AND1)+((字节 24)SHR5)

日 = (字节 24)AND 11111b

26－27　分配给文件的第一个簇，是文件链的起始簇。

28－31　文件的字节数，为双字。第一个字是最小有效字。注意此长度由最后一个修改文件的程序设置。这两个字既不必表示文件的真正长度，也不必表示不同程序用相同方法计算出来的长度。有一些程序利用这个长度决定程序的结束位置。另一些程序则寻找文件结束字符或两者兼做。

ROOT. PAS

现在，将写一段译码根目录信息的程序。它的功能是通过读引导记录确定根目录的位置和大小，一次译码一个扇区的目录信息。输出见图 7.1。

```
IBMBIO   .COM    RHS...    12: 0: 0   12/30/1985        2       16369
IBMDOS   .COM    RHS...    12: 0: 0   12/30/1985       10       28477
BOOKS    .       ....D.    13:21:44    7/ 4/1986      280           0
COMM     .       ....D.    13:22:10    7/ 4/1986      539           0
CONFIG   .       ....D.    13:22:16    7/ 4/1986      611           0
DOS      .       ....D.    13:22:18    7/ 4/1986      633           0
ED       .       ....D.    13:22:32    7/ 4/1986      750           0
FASTBACK.        ....D.    13:18:30    7/ 4/1986      192           0
LANGUAGE.        ....D.    13:22:46    7/ 4/1986      870           0
MOUSE1   .       ....D.    15:21:46    8/ 1/1986       32           0
PCPAINT  .       ....D.    18:33:20    8/ 1/1986      361           0
PSG      .       ....D.    10: 8:10   7/18/1986       67           0
UTILITY  .       ....D.    13:24:20    7/ 4/1986     1842           0
WINDOWS  .       ....D.    20:42:38   8/11/1986     4397           0
WP       .       ....D.    13:24:28    7/ 4/1986     1921           0
AUTOEXEC.BAT     ......    10: 0:18    9/ 4/1986       47         207

Drive: 2    Sector: 123

---------------ROOT DIRECTORY DISPLAY-----------------
F1: Change Drive                          F10: Exit
```

图 7.1 根目录

```
{.....................ROOT.PAS.....................
.
. This file contains routines for reading and decoding
. the disk directory.
.
.............................................................}

{
: display_dir_data
:
: This procedure decodes directory data.  Offset tells the number
: of sectors into the directory to start decoding.
:
}

procedure display_dir_data(var data:buf;offset:integer);
   var
      entry,i,data_start: integer;
   begin
      window(LFTSCRN,TOPSCRN,RTSCRN,BOTSCRN);
      clrscr;
      for entry := 0 to 15 do begin   {16 entries per sector}
         data_start := entry*32 + (offset mod BUFSECT)*512;  {find start}
         if data[data_start] = 0 then
            writeln('...UNUSED...')
         else begin  {entry is used}
            case data[data_start] of     {file name}
               $e5:  begin   {erased}
                        write('*');
                        for i := 1 to 7 do
                           write(chr(data[data_start + i]));
                        write('.');
                        for i := 8 to 10 do
                           write(chr(data[data_start + i]));
                        write(' ');
                     end;
```

```
            $2e:  if data[data_start + 1] = $2e then
                     {data for parent}
                     write('...PARENT... ')
                  else
                     write('...CURRENT.. ');
               else begin    {normal file}
                     for i := 0 to 7 do
                        write(chr(data[data_start + i]));
                     if (data[data_start + 11] and 8) = 0 then
                        write('.');
                     for i := 8 to 10 do
                        write(chr(data[data_start + i]));
                     if (data[data_start + 11] and 8) <> 0 then
                        write(' ');
                     write(' ');
                  end;
            end;
            write('  ');
            {file attribute}
            if (data[data_start + 11] and 1) <> 0 then  {read only}
               write('R')
            else
               write('.');
            if (data[data_start + 11] and 2) <> 0 then  {hidden}
               write('H')
            else
               write('.');
            if (data[data_start + 11] and 4) <> 0 then  {system}
               write('S')
            else
               write('.');
            if (data[data_start + 11] and 8) <> 0 then  {volume}
               write('V')
            else
               write('.');
            if (data[data_start + 11] and $10) <> 0 then  {directory}
               write('D')
            else
               write('.');
            if (data[data_start + 11] and $20) <> 0 then  {archive}
               write('A')
            else
               write('.');
            write('   ');
            {time created}
            write((data[data_start + 23] shr 3):2);
            write(':');
            write(((data[data_start + 23] and 7) shl 3 +
                  (data[data_start + 22] shr 5)):2);
            write(':');
            write((data[data_start + 22] and 31)*2:2);
            write('  ');
            {date created}
            write((((data[data_start + 25] and 1) shl 3) +
                  (data[data_start + 24] shr 5)):2);
            write('/');
            write((data[data_start + 24] and 31):2);
            write('/');
            write((data[data_start + 25] shr 1) + 1980);
            write('  ');
            {first cluster}
            write(data[data_start + 26] + int(data[data_start + 27])*256:6:0);
            write('  ');
            {size}
            writeln(data[data_start + 28] + int(data[data_start + 29])*256 +
                  int(data[data_start + 30])*256*256 +
                  int(data[data_start + 31])*256*256*256:8:0);
         end;
      end;
   end;
```

```
{
: display_root_dir_commands
:
: This procedure displays the root directory examining menu.
:
}

procedure display_root_dir_commands;
   begin
      window(COMMANDX,COMMANDY,COMMAND_RT,COMMAND_BOT);
      clrscr;
      writeln('---------------ROOT DIRECTORY DISPLAY-----------------');
      writeln('F1: Change Drive                    F10: Exit');
   end;


{
: root_dir_dump
:
: This is the main procedure.  It displays menus and data, and reads
: and acts upon commands.  Because the directory files can be very
: large, only a portion (BUFSECT sectors) is read in at a time.  The
: commands for moving through the directory check to see when another
: section needs to be read in.
:
}

procedure root_dir_dump(var data:buf);
   var
      finished:boolean;
      regs: result;
      offset_sector,direct_start,direct_size: integer;
      info: disk_info;
   begin
      finished := false;
      offset_sector := 0;   {start at beginning}
      get_disk_info(info,data);
      display_root_dir_commands;
      {find start and size}
      direct_start := info.fat_start + info.fat_copies*info.sect_in_fat;
      direct_size := info.root_entries * 32 div 512;
      {read in first chunk of info}
      glbl_sector := direct_start;
      if diskread(glbl_drive,glbl_sector,BUFSECT,data) <> 0 then
         beep;
      display_status;
      while not finished do begin
         display_dir_data(data,offset_sector);
         display_status;
         regs.ax := 0;
         intr($16,regs);   {get command}
         if regs.ax and $ff = 0 then
            case (regs.ax) of
               $4900:  {page up -- look at previous sector}
                       if offset_sector <> 0 then begin
                          offset_sector := offset_sector - 1;
                          {see if need to read in new section of directory}
                          if offset_sector mod BUFSECT = BUFSECT - 1 then
                             if diskread(glbl_drive,direct_start +
                                 offset_sector - (BUFSECT - 1),BUFSECT,data)
                                 <> 0 then
                                beep;
                          glbl_sector := direct_start + offset_sector;
                       end;
               $5100:  {page down -- look at next sector}
                       if offset_sector <> direct_size - 1 then begin
                          offset_sector := offset_sector + 1;
                          {see if need to read in new section of directory}
                          if offset_sector mod BUFSECT = 0 then
                             if diskread(glbl_drive,direct_start +
                                 offset_sector,BUFSECT,data)
```

```
                    <> 0 then
                   beep;
              glbl_sector := direct_start + offset_sector;
            end;
      $3b00: {F1  new drive}
            begin
              glbl_sector := 0;
              get_new_drive(data);
              offset_sector := 0;
              get_disk_info(info,data);
              direct_start := info.fat_start +
                    info.fat_copies*info.sect_in_fat;
              direct_size := info.root_entries * 32 div 512;
              glbl_sector := direct_start;
              if diskread(glbl_drive,glbl_sector,  {read in direc}
                BUFSECT,data) <> 0 then
                 beep;
              display_root_dir_commands;
              display_status;
              end;
       $4400: {F10  exit}
              finished := true;
      end;
  end;
end;
```

输入这段程序，并用文件名 root. pas 存盘。然后对 explorer. pas 做如下改动：

在 display—start—screen 增加：

```
writeln('F4:   Examine Root Directory');
```

然后增加：

```
{$I b:root.pas}
```

若必要改变驱动器字母。在主程序的 case 语句中增加：

```
$3e00:  {F4}
      root_dir_dump(buffer);
```

存盘，编译，改正输入错误，然后运行。观察几个磁盘的目录。

改变目录信息

比译码目录信息更有用的是修改目录信息的功能。利用这个功能可隐藏文件，把一个子目录改成一个常规文件，这样你就能用编辑程序和 DOS 文件命令处理它，还可改变文件生成时间和日期。你还可改变卷名或使文件不能被访问。在第 11 章，你可看到通过修改目录信息而得到的几个技巧。

利用目前的程序，通过使用其根目录检查功能，然后切换到扇区转储的方法来修改目录表项。首先，必须修改编码信息。

现在，为使目录表项修改起来更容易，再增加一些程序。给 Sector Dump 过程增加译码数据的功能，好象数据是目录扇区而不是十六进制数。还要增加改变目录译码信息的程序。若光标在文件名字上，键入一个字符就会改变文件名。若光标在文件属性部分，键入任何键都会触发文件属性状态，还可增加程序以易于修改时间和日期。不要改变文件的长度和起始簇。

为做这些改变，需要稍微修改一下 root. pas，对 sector. pas 也要做些增加和改动。

在 root. pas 中，把 displsy—dir—data 的过程说明改为：

```
procedure display_dir_data;
```

在 sector. pas 中如下改动。在 display—data 增加：

```
else
   display_dir_data(data,0);
```

在 display—dump—commands 中改

```
writeln('F4: Read Next    sh-F5: Write');
```

为

```
writeln('F4: Read Next    sh-F5: Write    F6: Toggle Style');
```

在 sector—dump 的 case statement 中增加：

```
$4000:  {F6 toggle display style}
        begin
           style := (style mod 2) + 1;
           display_data(data,offset,style);
        end;
```

将

```
         else if style = HEX_DUMP then    {not a command}
{$IFDEF Turbo3}
           adjust_hex_data(curs_x,curs_y,regs.ax,data,
                           offset*256);
{$ELSE}
           adjust_hex_data(curs_x, curs_y, integer(regs.ax), data,offset*256);
{$ENDIF}
```

改为：

```
        else if style = HEX_DUMP then    {not a command}
{$IFDEF Turbo3}
           adjust_hex_data(curs_x,curs_y,regs.ax,data,
                           offset*256)
{$ELSE}
           adjust_hex_data(curs_x,curs_y,integer(regs.ax),
                           data,offset*256)
{$ENDIF}
        else  {style = DIRECT_DUMP}
{$IFDEF Turbo3}
           adjust_direct_data(curs_x,curs_y,regs.ax,data);
{$ELSE}
           adjust_direct_data(curs_x,curs_y,integer(regs.ax),
                           data);
{$ENDIF}
```

在 display—data 过程前增加：

```
{
: Forward declare this procedure from root.pas so that you can
: decode sectors as portions of directory files.
}

procedure display_dir_data(var data:buf;offset:integer); forward;
```

在包含改变缓冲区数据的程序部分，增加：

```
{
: adjust_direct_data
:
: This procedure is for adjusting data during a directory dump.
: Only the name and attributes can be modified.  Modification of
: other parameters is left to the reader.  To change a character
: in the name, type in the new character over that to be changed.
: To change the setting of an attribute, move to the position for
: that attribute and hit any key to toggle it.
:
}


procedure adjust_direct_data(var curs_x,curs_y,input_data:
          integer; var data: buf);
  begin
    input_data := input_data and $ff;
    case (curs_x) of
      1..8:  {name}
             begin
               data[(curs_y - 1)*32 + curs_x - 1] := input_data;
               curs_x := curs_x + 1;
             end;
     10..12: {ext}
             begin
               data[(curs_y - 1)*32 + curs_x - 2] := input_data;
               curs_x := curs_x + 1;
             end;
     16..21: {attributes}
             begin
               data[(curs_y - 1)*32 + 11] := data[(curs_y - 1)*32
                  + 11] xor (1 shl (curs_x - 16));
               curs_x := curs_x + 1;
             end;
    end;
   display_dir_data(data,0);
  end;
```

做如上改动，存盘，重新编译 EXPLORER。修改输入错误，运行。用 Sector Dump 看看目录。触发显示译码目录格式。把光标移到一个文件名上，输入字符，也可输入小写字符。再试着改变一些属性。用 shift—F5 命令把这些改变动写到磁盘上。一定要记住你所做的改动。对重要文件别做改动，除非你有备份。退出 EXPLORER，用 dir 命令看你刚修改过的磁盘。

在第 11 章，可通过修改目录，实际操作一些有趣的技巧。在第 18 章(第二部分)，将通过修改文件属性移动子目录。

程序设计要点：

- 根目录包含文件的名字，大小，位置和根级目录。
- 每个文件的信息存在一个 32 位的表项中。
- 根目录在 FAT 之后。
- 通过修改目录可以隐藏文件。

第八章　文件

对于文件你已知道很多：文件数据存在数据区；目录告诉文件使用的第一个簇；文件链存在FAT里。本章将更进一步讨论文件。

观察一个文件，首先扫描目录直到你找到文件的名字。然后读起始簇。从起始簇开始，在FAT里跟踪文件。把文件链存到一个数组里，看一个簇，要看对应于该簇的扇区。看文件的上一个或下一个簇，只需看看上一个或下一个数组位置，这样可看到所装入的簇。这样，每次想转换簇时就不用读FAT的扇区。

读文件链时不可漫不经心，因为有可能文件的簇数超过了数组的界限，虽然可把数组扩大，以避免这种情况发生。虽然这种情况不一定发生，但还是有可能文件分配表已经遭到破坏，并产生了一个循环链。发生这种情况时，文件中的一个簇指向文件的上一个簇，导致死循环。为防止这种情况，只要检查一下不要超过数组的界限就行。

一旦知道一个文件的簇数，把簇数转换成扇区数。簇2是数据区的第一个扇区，数据区的起始部分可从引导记录计算出来，这在第6章已讨论过。

FILE. PAS

现在再给EXPLORER增加一个观察文件的模块。先观察根目录中的文件，然后再观察子目录中的文件。

你需要用到很多程序，需要读搜索的文件名，要把它转换成目录格式，把它变成大写，将扩展句点换成空格。然后读根目录部分，把它们的文件名表项与你正在寻找的文件相比较。还得跟踪FAT读文件链。为此，要记录刚才读过的FAT部分以及哪部分包含下一个文件链的信息。还需要一个译码FAT给定簇信息的程序。

要做到以上这些，必须有下面这[illegible]：

```
{.....................FILE.PAS.....  ..........
.
. This file contains routines for reading and displaying files
.
.....................................................}

{
.................................
.
. General name subroutines
.
.................................
}

{
: buf_strcmp
:
: This procedure compares the name in fn to that in the data
: buffer pointed to by offset.  Returns true if there is a match.
:
}
```

```
function buf_strcmp(fn:file_name;offset:integer;var data:buf): boolean;
   var
      i: integer;
      match: boolean;
   begin
      match := true;
      i := 0;
      while match and (i < 11) do       (11 chars in a file name)
         if ord(fn[i + 1]) <> data[offset + i] then
            match := false
         else
            i := i + 1;
      if match = true then
         buf_strcmp := true
      else
         buf_strcmp := false;
   end;


{
: get_file_name
:
: This procedure reads in a file name.
:
}



procedure get_file_name(var fn: file_name);
   begin
      window(COMMANDX,COMMANDY,COMMAND_RT,COMMAND_BOT);
      clrscr;
      write('Name: ');
      readln(fn);
   end;



{
: convert_file_name
:
: This procedure takes a filename in xxx.yyy format, and converts it
: to the form names are stored in the directory.  That is, it pads out
: spaces and removes the '.'.  For example, MIRNA.COM is converted to
: MIRNA   COM
:
}

procedure convert_file_name(var fn:file_name);
   var
      i,fn2_ptr: integer;
      fn2: file_name;
   begin
      fn2 := '
      fn2_ptr := 1;             (pts to position in new string)
      for i := 1 to length(fn) do
         if fn[i] = '.' then
            fn2_ptr := 9
         else begin
            fn2[fn2_ptr] := upcase(fn[i]);
            fn2_ptr := fn2_ptr + 1;
         end;
      fn := fn2;
   end;
```

```
{
.................................
.
. Information printing subroutines
.
.................................
}

{
: display_status2
:
: This procedure prints information about the current sector and file
: being examined.
:
}

procedure display_status2(fn: file_name; f_length: integer);
   begin
      window(STATUSX,STATUSY,STATUS_RT,STATUS_BOT);
      clrscr;
      writeln('Drive: ',glbl_drive,'  Sector: ',glbl_sector,'  Name: ',
              fn,'  Length: ',f_length);
   end;


{
: display_file_use
:
: This procedure prints out a list of the clusters used by a file, in
: the order that they are used (the file chain).  This procedure is called
: after an attempt to move before the beginning of a file.  (That is,
: after a PREV CLUSTER command is entered while the beginning of the file
: is being examined.)
:
}

procedure display_file_use(ft:file_trace);
   var
      i: integer;
   begin
      window(LFTSCRN,TOPSCRN,RTSCRN,BOTSCRN);
      clrscr;
      writeln('File uses clusters: ');
      for i := 1 to ft.length  do
         write(ft.chain[i],' ');
      writeln;
   end;


{
: print_end_of_file
:
: This procedure prints a message to indicate that the end of the
: file being examined has been reached.
:
}


procedure print_end_of_file;
   var
      i: integer;
   begin
      window(LFTSCRN,TOPSCRN,RTSCRN,BOTSCRN);
      clrscr;
      gotoxy(20,8);
      writeln('End Of File');
   end;
```

```
{
.................................
.
. Routines for converting between sectors and clusters,
. and for determining FAT sectors to use
.
.................................
}

{
: fat_sector_for
:
: This procedure determine which section, i.e., which set of BUFSECT
: sectors, in the FAT contains the information for a particular cluster.
: Returns the sector number of the first sector in that section.
:
: For example, if the cluster information is contained in the first
: BUFSECT set of FAT sectors, will return 0; in the second set, BUFSECT;
: and so forth.
:
}

function fat_sector_for(clust,fat_type: integer): integer;
   begin
      if fat_type = 12 then    {12 bit format}
         fat_sector_for := (trunc(clust * 3 / (2 * 512)) div BUFSECT) *
                BUFSECT
      else   {16 bit format}
         fat_sector_for := (trunc(clust * 2 / 512) div BUFSECT) * BUFSECT;
   end;


{
: cluster_to_sector
:
: This procedure returns the number of the sector at which a cluster
: begins.
:
}

function cluster_to_sector(cur_cluster: integer;info: disk_info): integer;
   var
      data_area: integer;
   begin
      data_area := info.fat_start + info.fat_copies*info.sect_in_fat +
                   info.root_entries * 32 div 512;
      cluster_to_sector := (cur_cluster - 2) * info.sect_per_clust +
                           data_area;
   end;

{
: next_cluster
:
: This function returns the number of the next cluster in a file
: chain.  It does so by reading the value stored in the FAT for the
: cur_cluster location.  If the value is an end of file code, it
: returns ERROR.
:
}

function next_cluster(cur_cluster: integer;info: disk_info;
                      var data:buf;var cur_FAT_clust:integer): integer;
   var
      new_clust_off,new_clust,next_FAT_clust: integer;
   begin
      {both 1.5 and 2 byte clusters divide evenly across 9 sectors}
      next_FAT_clust :=  fat_sector_for(cur_cluster,info.fat_type);
      if next_FAT_clust <> cur_FAT_clust then begin  {read in proper}
         cur_FAT_clust := next_FAT_clust;            {section of FAT}
         if diskread(glbl_drive,info.fat_start+cur_FAT_clust,BUFSECT,data)
```

```
                <> 0 then
            beep;
        end;
        if info.fat_type = 12 then begin
           new_clust_off := (cur_cluster * 3 div 2) mod (BUFSECT*512);   {loc}
           new_clust := data[new_clust_off] + 256*data[new_clust_off + 1];  {val}
           if odd(cur_cluster) then       {decode value}
              new_clust := new_clust shr 4
        else
           new_clust := new_clust and $fff;
        if (new_clust >= $ff8) and (new_clust <= $fff) then  {eof marker}
           next_cluster := ERROR
        else
           next_cluster := new_clust;
    end
    else begin {16 bit fat}
       new_clust_off := (cur_cluster * 2) mod (BUFSECT*512);
       new_clust := data[new_clust_off] + 256*data[new_clust_off + 1];
       if (new_clust >= $fff8) and (new_clust <= $ffff) then  {eof marker}
          next_cluster := ERROR
       else
          next_cluster := new_clust;
    end;
end;


{
.................................
.
. Routines for finding and tracing files
.
.................................
}


{
: trace_file
:
: Traces the file chain through the FAT, saving the results in the file_trace
: array.
:
: The size of the array imposes an arbitrary limit on the maximum file
: size which can be examined.  This can be changed by changing the
: CHAINSIZE constant.  Using an array with a length pointer is much
: more efficient, in this case, than a more flexible list structure.
:
}

procedure trace_file(var ft:file_trace; first_cluster:integer; var data:buf);
   var
      info: disk_info;
      cur_loc,next_loc: integer;
  begin
      get_disk_info(info,data);
      ft.length := 1;
      ft.chain[ft.length] := first_cluster;      {starts at first_cluster}
      cur_loc := -1;    {indicates FAT sector.  -1 => none was read}
      repeat    {traverse chain through FAT}
         ft.chain[ft.length + 1] := next_cluster(ft.chain[ft.length],info,
                                    data,cur_loc);
         ft.length := ft.length + 1;
      until (ft.chain[ft.length] = ERROR) or (ft.length = CHAINSIZE);
      ft.length := ft.length - 1;
  end;


{
: search_for_sub_file
:
: Searches through the data buffer, which contains a section of a
: directory, for an entry which matches the filename.  Num entries
```

```
: imposes a limit on the number of entries searched.  If a matching
: name is found, returns an offset into the buffer.  If not, returns
: ERROR.
:
}


function search_for_sub_file(fn:file_name;num_entries:integer;
        var data:buf): integer;
   var
      i: integer;
      finished: boolean;
   begin
      finished := false;
      i := 0;
      while not finished and (i < num_entries) do       {search through data}
         if buf_strcmp(fn,i*32,data) then       {look for matching name}
            finished := true
         else
            i := i + 1;
      if finished then
         search_for_sub_file := i*32
      else
         search_for_sub_file := ERROR;
   end;


{
: search_for_file
:
: Searches through the directory structure to find the information
: for a file.
:
: Will find the information for files with any attributes, including
: hidden or subdirectory.
:
: Assumes that it is called with the first BUFSECT sectors of the
: root directory in the data buffer.
:
: Returns an offset into the data buffer for the file information, if
: the file is found, else returns ERROR.
:
}

function search_for_file(full_name: file_name; info: disk_info;
           var data: buf): integer;
   var
      found_it: boolean;
      first_cluster,i,root_size,search_size,direct_start: integer;
      ft: file_trace;
   begin
      convert_file_name(full_name);
      root_size := (info.root_entries * 32 div 512) div BUFSECT;
      direct_start := info.fat_start + info.fat_copies*info.sect_in_fat;
      i := 0;
      found_it := false;
      if root_size > 0 then     {number of entries to examine}
         search_size := BUFSECT * 512 div 32
      else
         search_size := info.root_entries;
      while (i <= root_size) and not found_it do begin   {search through root}
         first_cluster := search_for_sub_file(full_name,search_size,data);
         if first_cluster <> ERROR then
            found_it := true
         else begin
            i := i + 1;   {not found, so read next section of the root}
            if diskread(glbl_drive,direct_start + i*BUFSECT,BUFSECT,data) <> 0
                  then
               beep;
         end;
```

```
   end;
   {root search is finished}
   search_for_file := first_cluster;
end;


{
.................................
.
. Main routines
.
.................................
}


{
: display_file_commands
:
: This procedure displays the menu for examining files.
:
}

procedure display_file_commands;
   begin
      window(COMMANDX,COMMANDY,COMMAND_RT,COMMAND_BOT);
      clrscr;
      writeln('----------------ROOT FILE DISPLAY-----------------');
      writeln('F1: Change Drive   F2: Change File   F3: Prev Cluster');
      writeln('F4: Next Cluster   F5: Toggle Style  F10: Exit');
   end;


{
: file_dump
:
: This is the main procedure.  It displays menus and data, and reads and
: processes commands.  File data can be displayed as hex data or as directory
: data.
:
}

procedure file_dump(var data:buf);
  var
     finished:boolean;
     regs: result;
     offset_sector,direct_start,style,cur_sector,
          new_sector,file_ptr,first_cluster: integer;
     info: disk_info;
     fn: file_name;
     ft: file_trace;
begin
   finished := false;
   display_file_commands;
   offset_sector := 0;
   cur_sector := 0;
   style := HEX_DUMP;
   fn := '';
   ft.length := 0;
   get_disk_info(info,data);
   direct_start := info.fat_start + info.fat_copies*info.sect_in_fat;
   while not finished do begin
      if cur_sector <> 0 then         {display data}
         if style = DIRECT_DUMP then
            display_dir_data(data,offset_sector)
         else
            display_page(data,offset_sector*256);
      display_status2(fn,ft.length);
      regs.ax := 0;
      intr($16,regs);          {get command}
      if regs.ax and $ff = 0 then
         case (regs.ax) of
```

```
$4900:  {page up -- display previous sector in cluster}
        begin
           if offset_sector = 0 then
              offset_sector := style * info.sect_per_clust
                     - 1   {move two sectors at a time if
                            displaying directory info.}
           else
              offset_sector := offset_sector - 1;
           glbl_sector := cur_sector + (offset_sector div style);
        end;
$5100:  {page down -- display next sector in cluster}
        begin
           offset_sector := (offset_sector + 1) mod
                  (style * info.sect_per_clust);
           glbl_sector := cur_sector +
                  (offset_sector div style);
        end;
$3b00:  {F1  new drive}
        begin
           glbl_sector := 0;
           get_new_drive(data);
           offset_sector := 0;
           get_disk_info(info,data);
           direct_start := info.fat_start +
                 info.fat_copies*info.sect_in_fat;
           display_file_commands;
           fn := '';
           ft.length := 0;
           display_status2(fn,ft.length);
        end;
$3c00:  {F2  new file}
        begin
           get_file_name(fn);
           display_file_commands;
           {read in start of directory info}
           if diskread(glbl_drive,direct_start,BUFSECT,data)
                <> 0 then
              beep
           else begin
              {find start of file info}
              first_cluster := search_for_file(fn,info,data);
              if first_cluster = ERROR then
                 beep
              else begin  {file found}
                 {find where it begins}
                 first_cluster := data[first_cluster + 26] +
                       256*data[first_cluster + 27];
                 {trace the chain through the fat}
                 trace_file(ft,first_cluster,data);
                 {start displaying it from the beginning}
                 file_ptr := 0;
                 cur_sector := 0;
                 display_file_use(ft);
              end;
           end;
        end;
$3d00:  {F3  previous cluster}
        begin
           if file_ptr > 0 then begin
              file_ptr := file_ptr - 1;  {move back in chain}
              if file_ptr = 0 then begin {if move before
                                          start, display file
                                          use}
                 cur_sector := 0;
                 display_file_use(ft);
              end
              else begin
                 {find start of cluster}
                 cur_sector := cluster_to_sector(ft.chain
                        [file_ptr],info);
```

```
                        glbl_sector := cur_sector;
                        offset_sector := 0;
                        if diskread(glbl_drive,glbl_sector,
                                info.sect_per_clust,data) <> 0 then
                           beep;
                     end;
                  end;
               end;
         $3e00: {F4  next cluster}
               begin
                  if file_ptr <= ft.length then begin
                     file_ptr := file_ptr + 1;  {move forward in chain}
                     if file_ptr > ft.length then begin
                        cur_sector := 0;
                        print_end_of_file;
                     end
                     else begin
                        {find where cluster starts}
                        cur_sector := cluster_to_sector(ft.chain
                              [file_ptr],info);
                        glbl_sector := cur_sector;
                        offset_sector := 0;
                        if diskread(glbl_drive,glbl_sector,
                                info.sect_per_clust,data) <> 0 then
                           beep;
                     end;
                  end;
               end;

         $3f00: {F5  switch style}
               style := (style mod 2) + 1;

         $4400: {F10  exit}
               finished := true;
      end;
  end;
end;
```

输入这段程序，以 file. pas 的名字存盘。

还有一些要注意的事情。指定文件名时，不包括驱动器名。也就是用 FAILSAFE. COM，而不是 A:FAILSAFE. COM。用改变驱动器的命令改变驱动器。若找不到这个文件，该程序就将发出嘟嘟的声音。

然后，增加

```
{$I b:file.pas}
```

若有必要，改变驱动器名，在主程序的 case 语句中增加：

```
$3f00:  {F5}
      file_dump(buffer);
```

存盘，编译，改正输入错误，运行。观察根目录中的一些文件。在簇间和簇的扇区间移动。再试一些程序文件。如果正在看的是一张系统盘，那么再看看 IBMBIO. COM——它是一个隐含的系统文件。观察一些文本文件，象信或批文件。

子目录

接下来，看看子目录。假设你有一个叫做 DOS 的子目录，把 DOS 作为要研究的文件名用 EXPLORER 来观察。注意，这样将会找到一个文件。看该找到的文件的第一个扇区。现在触发目录译码的显示方式。它包含和根目录一样的目录信息类型。事实上，子目录就是文件和根目录的一个混合物。它们是任意长度的文件，包含许多目录表项。同根目录一样，它们也可包含指向另外子目录的表项。

若想观察子目录里的一个文件，首先在根目录中搜索，找到文件路径中第一个子目录表项。然后搜索子目录文件(该文件链中的每一簇)，找到文件路径中的下一个子目录表项。继续搜索，直到找到这个文件自己的目录表项。从目录表项找到起始簇，在 FAT 里跟踪文件，进行观察。

若在显示文件第一个簇时，使用显示上一簇的命令，屏幕将列出该文件使用的所有簇。若在显示文件最后一簇时，使用显示下一簇的命令，则将显示已到文件尾的信息。你第一次观察文件时，将显示文件链。文件还以十六进制或目录格式显示。一会儿将用到这些方法。

还要对 explorer. pas 增加一些内容。

在类型说明中，增加

```
{a list for tracing the clusters that a file uses}
file_trace = record
    chain: array[1..CHAINSIZE] of integer;
    length: integer;
  end;

{string for filename.ext}
file_name = string[12];
```

在常量部分，增加：

```
CHAINSIZE = 200;
DIRECT_DUMP = 1;
```

在 display—start—screen，增加：

```
writeln('F5:   Examine File');
```

使 FILE. PAS 能处理子目录

对 file. pas 做些改动，使其能观察包含在子目录里的文件。

在 get—file—name，把

```
var fn: file_name
```

改为：

```
var fn: long_name
```

在 display—status2，把

```
fn: file_name
```

改为：

```
fn: long_name
```

在 file—dump，把

```
fn: file_name
```

改为：

```
fn: long_name
```

然后，将 search—for—file 功能改变如下：

```
{
: search_for_file
:
: Searches through the directory structure to find the information
: for a file.  Starts by looking through the root directory for
: the first subdirectory, searches through that subdirectory for
: the next subdirectory entry, and so on, until the desired file
: is found.
:
: Subdirectories are searched by examining them cluster by cluster,
: using their file chain.
:
: Will find the information for files with any attributes,
: including hidden or subdirectory.
:
: Assumes that it is called with the first BUFSECT sectors of the
: root directory in the data buffer.
:
: Returns an offset into the data buffer for the file information,
: if the file is found, else returns ERROR.
:
}

function search_for_file(full_name: long_name; info: disk_info;
           var data: buf): integer;
  var
     an_error,found_it: boolean;
     first_cluster,i,root_size,search_size,direct_start: integer;
     sub_name: file_name;
     ft: file_trace;
  begin
     an_error := false;
     parse_name(full_name,sub_name);{find first file to look for}
     convert_file_name(sub_name);
     root_size := (info.root_entries * 32 div 512) div BUFSECT;
     direct_start := info.fat_start + info.fat_copies*info.
     sect_in_fat; i := 0;
     i := 0;
     found_it := false;
     if root_size > 0 then     {number of entries to examine}
       search_size := BUFSECT * 512 div 32
     else
       search_size := info.root_entries;
     while (i <= root_size) and not found_it do begin
       {search through root}
       first_cluster := search_for_sub_file(sub_name,search_size,
        data);
       if first_cluster <> ERROR then
          found_it := true
```

```
    else begin
       i := i + 1;{not found, so read next section of the root}
       if diskread(glbl_drive,direct_start + i*BUFSECT,BUFSECT,
        data) <> 0 then
             beep;
       end;
   end;
   {root search is finished}
   if first_cluster = ERROR then  {file not found}
      an_error := true;
   while not an_error and (full_name <> '') do begin
      first_cluster := data[first_cluster + 26] + 256*data
         [first_cluster + 27];
         {find where file (next directory) begins}
      trace_file(ft,first_cluster,data);    {trace it}
      parse_name(full_name,sub_name);
         {begin looking for next file}
      convert_file_name(sub_name);
      found_it := false;
      i := 1;
      while not found_it and not an_error and (i <= ft.length)
        do begin
           glbl_sector := cluster_to_sector(ft.chain[i],info);
           if diskread(glbl_drive,glbl_sector,info.sect_per_clust,
              data) <> 0 then
              an_error := true
           else begin   {read through subdirectory chain}
              first_cluster := search_for_sub_file(sub_name,
                   info.sect_per_clust * 16, data);  {16=512/32}
              if first_cluster <> ERROR then
                 found_it := true;
           end;
           i := i + 1;
        end;
      if not found_it then {file (or subdir) not there, so quit}
           an_error := true;
      end;
      if not an_error then
         search_for_file := first_cluster
      else begin
         beep;
         search_for_file := ERROR;
      end;
   end;
```

在 general name 子程序里，在 get—file—name 过程后，增加如下内容：

```
{
: parse_name
:
: This procedure removes the first name from a file name path,
: given by full_name, and returns it in sub_name.  If full_name
: is only a file name (not a path), then returns that name and
: clears full_name.
:
: For example, if called with:   full_name = sub1\sub2\file
: would return with:             full_name = sub2\file
:                                sub_name = sub1
:
}

procedure parse_name(var full_name: long_name; var sub_name:
   file_name);
   var
```

```
    slash_pos: integer;
begin
    slash_pos := pos('\',full_name);
    if slash_pos = 0 then begin
        sub_name := full_name;
        full_name := '';
    end
    else begin
        sub_name := copy(full_name,1,slash_pos - 1);
        delete(full_name,1,slash_pos);
    end;
end;
```

对 file. pas 做以上改动，存盘。在 explorer. pas 的类型定义中增加：

```
{string for pathname\filename.ext}
long_name = string[64];
```

存 explorer. pas，编译，改正输入错误，运行。观察一些子目录下的文件。记住要写全路径名字，但不要写驱动器名。例如，不用 C：UTILITIES\FAILSAFE. COM，而用 UTILITIES\FAILSAFE. COM。还有文件名不能以反斜线(\)开始。EXPLORER 总是从根目录开始搜寻文件。

程序设计要点：

- 文件是簇的集合。第 1 簇号存在目录中，FAT 包含一个剩余簇的表。
- 看文件数据，需要找到文件链中的簇，然后看看那些簇里的扇区内容。
- 子目录是包含目录格式信息的文件。寻找子目录里的一个文件，要扫描路径里的每一个子目录，寻找路径里下一个子目录的表项。

第九章　删除文件

删除一个文件时，其目录表项的第一个字符置为 e5h，文件链中的所有簇都置空(0)。数据簇的信息没有变，目录表项的其它信息也没有变。如果另一文件表项没有使用此删除文件的表项位置，就能找到被删除文件的起始簇和长度。如果磁盘没有增加新文件，那么被删除文件的所有数据还是完整的。只需解决怎样把它们组织起来的问题。根据磁盘存储残片和大小，即使磁盘增加了一些新文件，被删除文件的全部或一部分数据也有很大可能是完整的。这章，你将学会怎样用 ERASED. PAS 恢复被删除文件。

首先找到删除文件的目录表项，接着找到其原来的大小和第一个簇。通过查看存在簇里的标志为空的数据，从文件第一簇开始试着拼合文件链。因为 DOS 存文件时一般用相邻的簇，所以很有可能文件的下一个簇在第一个簇后面。如果发现有一个簇里面有部分文件，把该簇加到文件链上。当然，该簇标志必须为空，并且以前没有加到过链上。在找到尽可能多的文件数据后，把文件链写到 FAT 和 FAT 的拷贝上，恢复目录表项，还需更新目录表项里的起始簇指针和文件大小。

如果是包含文件的子目录遭到破坏，那么先恢复子目录，再恢复文件。如果这样做不行，但是还能找到文件表项——也就是部分子目录没有被覆盖——那么拷贝该文件表项信息。然后把它作为根目录(或某子目录)的最后一个表项输入。注意，第一个字符要保持 e5h。然后用上面的过程恢复该文件。

如果该文件的信息都已被破坏，那么用其文件名在根目录增加一个表项，第一个字符为 e5h，不要管文件时间和日期。把属性字节置 0，第一簇置 2，大小置 0。然后再用上面的过程。为了有助于找到包含丢失信息的簇，需要写一个程序，搜索包含特定文本或特定字符串的磁盘扇区。

还可用这种方法恢复一个被重新格式化的硬盘上的信息，确定文件碎片非常费时间，除非有一个搜索程序。

ERASED. PAS

恢复被删除文件的程序有几个特点。它必须显示旧文件包含多少个簇和有多少个文件需要恢复。能在 FAT 中向前或向后搜寻空簇。还能跳到包含特定扇区的簇并显示已恢复的内容。大多数特点是用于薄记。当尽可能多的文件恢复后，该程序还要把恢复的信息存盘。

看这个恢复文件的程序。此模块需要输入要恢复的文件名。在文件名字部分找到第一个字符，记下，把它改为 e5h。这样，目录搜寻程序就能找到它。用 find—pos 确定字符串里的最后一个反斜线(\)，其后是文件名的第一个字符。

例行程序 next—avail—cluster 和 prev—avail—cluster 的功能是搜寻 FAT 直到发现空簇。

由于这些例行程序经常用到 FAT，所以将一部分 FAT 存在一缓冲区中。因为 FAT 可能会比缓冲区大，所以要记录缓冲区中的 FAT 部分。

ERASED. PAS 模块中的一个过程选择驱动器和文件，然后用恢复命令的主菜单调用这个程序。这样可使用户在恢复过程中避免不小心地重新选择了不同的文件或驱动器，同时也简化

了恢复命令部分。

```
{.....................ERASED.PAS.....................
.
. This file contains routines for recovering erased files.
.
.....................................................}



{
.................................
.
. Routines to display information
.
.................................
}


{
: display_status3
:
: This procedure prints information about the current information
: being displayed and about the file that is being recovered.
:
}

procedure display_status3(fn:long_name;f_length,old_length: integer);
   begin
      window(STATUSX,STATUSY,STATUS_RT,STATUS_BOT);
      clrscr;
      writeln('Drive: ',glbl_drive,'  Sector: ',glbl_sector,' Name: ',
              fn,'  Length: ',f_length,' Old length: ',old_length);
   end;


{
: display_in_use_error
:
: This procedure displays a message that the cluster cannot be added
: to a file because it is already in use.
:
}

procedure display_in_use_error;
   var
      i: integer;
   begin
      window(LFTSCRN,TOPSCRN,RTSCRN,BOTSCRN);
      clrscr;
      writeln('Cluster in use, cannot chain');
      delay(2000);
   end;



{
.................................
.
. Name processing routines
.
.................................
}
```

```
{
: last_pos
:
: This procedure returns the last position where the character obj
: occurs in the string.  It is used to find the first character in
: the name of the file that was erased.
:
}

function last_pos(obj:char;str:long_name): integer;
   var
      final_pos,next_obj: integer;
      new_str: long_name;
   begin
      final_pos := 0;
      new_str := str;
      repeat   {search until there are no more obj's}
         next_obj := pos(obj,new_str);
         final_pos := final_pos + next_obj;
         delete(new_str,1,next_obj);
      until next_obj = 0;
      last_pos := final_pos;
   end;



{
.....................................
.
. Conversion routines
.
.....................................
}



{
: sector_to_cluster
:
: This function returns the number of the cluster containing a
: particular sector.
:
}

function sector_to_cluster(cur_sector: integer;info: disk_info): integer;
   var
      data_area: integer;
   begin
      data_area := info.fat_start + info.fat_copies*info.sect_in_fat +
                   info.root_entries * 32 div 512;
      sector_to_cluster := (cur_sector - data_area) div info.sect_per_clust
                   + 2;
   end;

{
.....................................
.
. FAT processing routines
.
.....................................
}

{
: next_avail_cluster
:
: This function returns the cluster number of the first free cluster
: following the cluster passed in variable cluster.  It examines each
```

```
    : cluster entry in the FAT following the passed cluster, until one is
    : found that is unused (cluster value will be 0).  If there are no more
    : free clusters on the disk, returns ERROR.  (Note: will search through
    : FAT until comes back to original cluster.)  There is no need to worry
    : about which section of the FAT to examine;  next_cluster takes care
    : of that.
    :
    }

    function next_avail_cluster(cluster:integer;info: disk_info;var FAT:buf;
                var cur_FAT:integer):integer;
      var
         i,max_search: integer;
         finished: boolean;
      begin
         i := 0;
         finished := false;
         if info.fat_type = 12 then  {12 bit format}
            max_search := (info.sect_in_FAT * 512 * 2 div 3) - 2
         else   {16 bit format}
            max_search := (info.sect_in_FAT * 512 div 2) - 2;

        {Note: max_search tells the number of cluster entries in the
         FAT.  i indicates how many clusters have been examined, so
         after i > max_search, have examined the whole FAT.  cluster
         tells the current cluster being examined.  So when cluster
         > max_search, have wrapped around from the end of the FAT
         to the beginning.}

      while (i < max_search) and not finished do begin
         i := i + 1;
         cluster := cluster + 1;     {look at next cluster}
         if cluster > max_search + 2 then  {wrap around FAT}
            cluster := 2;
         if next_cluster(cluster,info,FAT,cur_FAT) = 0 then  {check value}
            finished := true;
      end;
      if finished then
         next_avail_cluster := cluster
      else
         next_avail_cluster := ERROR;
   end;



{
: prev_avail_cluster
:
: This procedure is like next_avail_cluster, except that it
: returns the cluster number for the first free cluster which is
: before the passed cluster.
:
}

function prev_avail_cluster(cluster:integer;info: disk_info;var FAT:buf;
            var cur_FAT: integer):integer;
   var
      i,max_search: integer;
      finished: boolean;
   begin
      i := 0;
      finished := false;
      if info.fat_type = 12 then
         max_search := (info.sect_in_FAT * 512 * 2 div 3) - 2
      else {16-bit FAT}
         max_search := (info.sect_in_FAT * 512 div 2) - 2;
      while (i < max_search) and not finished do begin
         i := i + 1;
         cluster := cluster - 1;
         if cluster < 2 then
            cluster := max_search + 2;
```

```
            if next_cluster(cluster,info,FAT,cur_FAT) = 0 then
               finished := true;
         end;
         if finished then
            prev_avail_cluster := cluster
         else
            prev_avail_cluster := ERROR;
      end;



   {
   : change_cluster_num
   :
   : This procedure changes the FAT to update a file chain.  new_val contains
   : the number of the cluster which is to follow cluster.  The offset in the
   : FAT which contains cluster's information is computed and then changed.
   : Note: nothing is written to disk.
   :
   : This procedure is used when writing a new chain to disk after an
   : erased file has been recovered, or when a new cluster has been added
   : to the file chain as a file is being recovered.
   :
   }

   procedure change_cluster_num(cluster,new_val: integer;info: disk_info;
                  var FAT:buf);
      var
         clust_off: integer;
      begin
         if info.fat_type = 12 then begin
            clust_off := (cluster * 3 div 2) mod (BUFSECT * 512); {compute loc}
            if new_val = EOCHAIN then      {use this value to indicate file end}
               new_val := $fff;
            if odd(cluster) then begin
               FAT[clust_off] := (FAT[clust_off] and $f) + ((new_val and $f)
                        shl 4);
               FAT[clust_off+1] := new_val shr 4;
            end
            else begin
               FAT[clust_off] := new_val and $ff;
               FAT[clust_off + 1] := (FAT[clust_off + 1] and $f0) +
                     new_val shr 8;
            end;
         end
         else begin {12-bit FAT}
            clust_off := (cluster * 2) mod (BUFSECT * 512);
            if new_val = EOCHAIN then

{$IFDEF Turbo3}
               new_val := $ffff;
{$ELSE}
               new_val := integer($ffff);
{$ENDIF}
            FAT[clust_off] := new_val and $ff;
            FAT[clust_off+1] := new_val shr 8;
         end;
      end;



   {
   : write_chain_to_FAT
   :
   : This procedure takes a file chain, passed in ft, writes it into the
   : FAT, and writes the FAT to disk.  Because the FAT may be very large,
   : it is updated in groups of BUFSECT sectors.  Any duplicate copies
   : of the FAT are also updated.
```

```
  :
  }

  procedure write_chain_to_FAT(ft:file_trace;info: disk_info;var FAT:buf);
     var
        i,fat_part,fat_size,save_size: integer;
     begin
        fat_size := info.sect_in_fat div BUFSECT; {sections in FAT}
        for fat_part := 0 to fat_size do begin    {for each section ..}
           if diskread(glbl_drive,info.fat_start + fat_part*BUFSECT,  {read it}
                    BUFSECT,FAT) <> 0 then
              beep;
           for i := 1 to ft.length - 1 do   {add all file chain elements which
                                             occur in this section}
              if fat_sector_for(ft.chain[i],info.fat_type) =
                    fat_part*BUFSECT then
                 change_cluster_num(ft.chain[i],ft.chain[i+1],info,FAT);
              {now for the end of chain marker}
              if fat_sector_for(ft.chain[ft.length],info.fat_type) =
                       fat_part*BUFSECT then
                 change_cluster_num(ft.chain[ft.length],EOCHAIN,info,FAT);
              {write this updated section to disk for each FAT copy}
              for i := 1 to info.fat_copies do begin
                 {if the part of the FAT to be saved is the very last section,
                  then all BUFSECT sectors need not be written.  (In fact,
                  doing so would overwrite part of the directory and part of
                  the duplicate FAT.)  save_size is the number of sectors
                  to save.}
              if fat_part <> fat_size then
                 save_size := BUFSECT
              else
                 save_size := info.sect_in_fat - BUFSECT*fat_part;
              if diskwrite(glbl_drive,info.fat_start + (i - 1)*info.sect_in_fat
                    + fat_part*BUFSECT,save_size,FAT) <> 0 then
                 beep;
           end;
        end;
     end;



  {
  ..................................
  .
  . Procedures for getting back a particular file
  .
  . Note:  there are two large procedures, recover_erased_file and
  . get_back files.  get_back_files is used to choose the drive and
  . file to be recovered.  If the name entered was in fact erased, then
  . recover_erased_file is called.  It contains the routines to find
  . free clusters, display them, add them to the chain, and to review
  . the clusters added to the file being erased.
  .
  ..................................
  }



  {
  : display_recover_commands
  :
  : This procedure displays the menu for getting back a particular
  : erased file.
  :
  }

  procedure display_recover_commands;
     begin
```

```
      window(COMMANDX,COMMANDY,COMMAND_RT,COMMAND_BOT);
      clrscr;
      writeln('----------------RECOVERING ERASED FILE--------------------');
      writeln('F1: Add to File    F2: Prev Free Clust   F3: Next Free Clust');
      writeln('F4: Toggle Style   F5: Prev in File      F6: Next in File');
      writeln('F7: Goto Sector    sh-F8: Save File      F10: Exit');
    end;


{
: recover_erased_file
:
: This procedure is the most important one.  It processes commands
: to find free clusters, to display the data they contain in either
: hex/ASCII format or in directory format, to add clusters to the file
: being recovered, to review the progress so far, to move to any part of
: the disk (to either add that cluster, or continue searching from there),
: and to save the recovered information from the file that was erased.
:
}

procedure recover_erased_file(fn:long_name;first_clust: integer;
          first_let:char;num_clust,direct_sect,entry_offset: integer;
          var data:buf;var cur_FAT_clust:integer);
  var
     ft: file_trace;
     FAT: buf;
     fin,finished: boolean;
     new_size: real;
     style,cur_sector,offset_sector,cur_cluster,file_ptr,i: integer;
     sect_to_read:integer;
     info: disk_info;
  begin
     FAT := data;       {this buffer just stores FAT info}
     ft.length := 0;    {file length is initially 0}
     finished := false;
     display_recover_commands;
     offset_sector := 0;
     file_ptr := 0;
     style := HEX_DUMP;
     get_disk_info(info,data);
     cur_sector := cluster_to_sector(first_clust,info);  {read in info from}
     glbl_sector := cur_sector;                          {the first cluster}
     if diskread(glbl_drive,glbl_sector,info.sect_per_clust,data) <> 0
         then
       beep;
     while not finished do begin
       if cur_sector <> 0 then     {display info}
          if style = DIRECT_DUMP then
             display_dir_data(data,offset_sector)
          else
             display_page(data,offset_sector*256);
       display_status3(fn,ft.length,num_clust);
       regs.ax := 0;       {get command}
       intr($16,regs);
       if regs.ax and $ff = 0 then
          case (regs.ax) of
           $4900:  {page up -- display prev sector in cluster}
                   begin
                      if offset_sector = 0 then
                         offset_sector := style * info.sect_per_clust
                               - 1
                      else
                         offset_sector := offset_sector - 1;
                      glbl_sector := cur_sector + (offset_sector div style);
                   end;
           $5100:  {page down -- display next cluster in sector}
                   begin
                      offset_sector := (offset_sector + 1) mod
                             (style * info.sect_per_clust);
```

```
            glbl_sector := cur_sector +
                    (offset_sector div style);
         end;
$3b00:   {F1  Add cluster to file}
         begin
            {compute the cluster number}
            cur_cluster := sector_to_cluster(cur_sector,info);
            {is it used by another file?}
            if next_cluster(cur_cluster,info,FAT,cur_FAT_clust)
                  <> 0 then
               display_in_use_error

            {Now check if the cluster has already been added
             to the file.  Note: even though we mark
             clusters in the FAT as used after they are
             added (see code which follows), these changes
             are only temporary.  If the FAT is very large,
             then these changes may have been lost when a
             new FAT section was read in.  Therefore, need
             to double check by looking through the file
             chain.}

            else begin
               fin := false;
               for i := 1 to ft.length do
                  if ft.chain[i] = cur_cluster then
                     fin := true;
               if fin then
                  display_in_use_error
               {cluster unused, so add it}
               else begin
                  ft.length := ft.length + 1;
                  ft.chain[ft.length] := cur_cluster;
                  change_cluster_num(cur_cluster,EOCHAIN,info,
                        FAT);   {mark in FAT that cluster is}
                              {in use.  This is temporary}
               end;
            end;
         end;

$3c00:   {F2 Find previous available cluster}
         begin
            {find it}
            cur_cluster := prev_avail_cluster(sector_to_cluster(
                  cur_sector,info),info,FAT,cur_FAT_clust);
            if cur_cluster <> ERROR then begin
               cur_sector := cluster_to_sector(cur_cluster,
                     info);
               offset_sector := 0;
               glbl_sector := cur_sector;
               if diskread(glbl_drive,glbl_sector, {read it}
                     info.sect_per_clust,data) <> 0 then
                  beep;
            end;
         end;

$3d00:   {F3 Find next available cluster}
         begin
            {find it}
            cur_cluster := next_avail_cluster(sector_to_cluster(
                  cur_sector,info),info,FAT,cur_FAT_clust);
            if cur_cluster <> ERROR then begin
               cur_sector := cluster_to_sector(cur_cluster,
                     info);
               offset_sector := 0;
               glbl_sector := cur_sector;
               if diskread(glbl_drive,glbl_sector, {read it}
                     info.sect_per_clust,data) <> 0 then
                  beep;
            end;
         end;
```

```
$3e00:  {F4 Toggle display style}
        style := (style mod 2) + 1;

$3f00:  {F5 Display previous cluster in file}
        begin
           {Note: the file pointer is initially set to 0.
            Thus, if this command is used before an F6
            command, nothing will happen.  Once F6 has been
            pressed, it will operate normally.  It is the
            same in function as the corresponding function
            in the display files procedure in file.pas}
           if file_ptr > 0 then begin
              file_ptr := file_ptr - 1;
              if file_ptr = 0 then begin
                 cur_sector := 0;
                 display_file_use(ft);
              end
              else begin
                 cur_sector := cluster_to_sector(ft.chain
                       [file_ptr],info);
                 glbl_sector := cur_sector;
                 offset_sector := 0;
                 if diskread(glbl_drive,glbl_sector,
                          info.sect_per_clust,data) <> 0 then
                    beep;
              end;
           end;
        end;

$4000:  {F6 Display next cluster in file}
        begin
           {Note: file_ptr is initially set to 0, not
            to the last cluster added to a file.  So the
            first time this command is used, will display
            the very first sector added to the file.  Works
            just as the corresponding feature in file.pas}

           if file_ptr <= ft.length then begin
              file_ptr := file_ptr + 1;
              if file_ptr > ft.length then begin
                 cur_sector := 0;
                 print_end_of_file;
              end
              else begin
                 cur_sector := cluster_to_sector(ft.chain
                       [file_ptr],info);
                 glbl_sector := cur_sector;
                 offset_sector := 0;
                 if diskread(glbl_drive,glbl_sector,
                          info.sect_per_clust,data) <> 0 then
                    beep;
              end;
           end;
        end;
$4100:  {F7  input a sector number and go to that cluster}
        {This is especially useful when the file may be
         very fragmented, and it is known in advance where
         certain sections of the file are.}
         begin
           window(COMMANDX,COMMANDY,COMMAND_RT,COMMAND_BOT);
           clrscr;
           fin := false;
           while not fin do begin
              write('Sector: ');
              readln(cur_sector);   {read sector number}
              {find the sector which begins the cluster
               containing cur_sector}
              cur_sector := cluster_to_sector(
                   sector_to_cluster(cur_sector,info),info);
              glbl_sector := cur_sector;
```

```
                        offset_sector := 0;
                        {read it}
                        if diskread(glbl_drive,glbl_sector,
                                info.sect_per_clust,data) <> 0 then
                           writeln('ERROR -- try again')
                        else
                           fin := true;
                     end;
                     display_recover_commands;
                  end;

         $5b00:   {shft-F8  save file}
                  if ft.length <> 0 then begin
                     write_chain_to_FAT(ft,info,FAT);  {update FAT}
                     sect_to_read := entry_offset div 512;
                     if diskread(glbl_drive,direct_sect + sect_to_read,1,
                             data) <> 0 then
                        beep;
                     {update directory}
                     entry_offset := entry_offset mod 512;
                     {first, fix the first letter in the file name}
                     data[entry_offset] := ord(upcase(first_let));
                     {now, put in the (possibly changed) first
                      cluster entry}
                     data[entry_offset+26] := ft.chain[1] and $ff;
                     data[entry_offset+27] := ft.chain[1] shr 8;

                     {if the file size was changed, update it
                      to the approximate new size.  Calculate
                      the new size as the number of bytes per cluster
                      times the number of clusters added.  (may be off
                      by several bytes.)}

                     if ft.length <> num_clust then begin
                        new_size := ft.length * 512 *
                              info.sect_per_clust;
                        data[entry_offset+31] := trunc(new_size /
                              (256.0*256*256));
                        data[entry_offset+30] := trunc
                                    ((new_size - 256.0*256*256*data
                                    [entry_offset+31]) / (256.0*256));
                        data[entry_offset+29] := trunc((new_size - 256.0*256*
                            (data[entry_offset+30] + 256.0*
                             data[entry_offset+31])) / 256.0);
                        data[entry_offset+28] := trunc(new_size - 256.0*
                            (data[entry_offset+29] +
                             256.0*data[entry_offset+30] +
                             256.0*256*data[entry_offset+31]));
                     end;
                     if diskwrite(glbl_drive,direct_sect + sect_to_read,1,
                           data) <> 0 then
                        beep;
                  end;

         $4400:   {F10  exit}
                  finished := true;
            end;
      end;
   end;


{
.....................................
.
. Main routine
.
.....................................
}
```

```
{
: display_get_back_files_commands
:
: This procedure displays the menu for choosing erased files
: to recover.
:
}


procedure display_get_back_files_commands;
   begin
      window(COMMANDX,COMMANDY,COMMAND_RT,COMMAND_BOT);
      clrscr;
      writeln('-------------GET BACK AN ERASED FILE---------------------');
      writeln('F1: Change Drive   F2: Choose File    F10: Exit');
   end;



{
: get_back_files
:
: This procedure processes commands to choose an erased file.  If the
: directory entry is found for the erased file, then this procedure
: calls recover_erased_file.
:
}

procedure get_back_files(var data:buf);
  var
     finished:boolean;
     regs: result;
     direct_start,first_let_pos,file_loc,direct_sect,old_size,
         first_clust: integer;
     old_size_rl: real;
     info: disk_info;
     fn: long_name;
     first_let: char;
     FAT_clust: integer;
  begin
     finished := false;
     display_get_back_files_commands;
     fn := '';
     get_disk_info(info,data);
     direct_start := info.fat_start + info.fat_copies*info.sect_in_fat;
     while not finished do begin
        display_status;
        regs.ax := 0;
        intr($16,regs);    {get commands}
        if regs.ax and $ff = 0 then
           case (regs.ax) of
              $3b00:  {F1 new drive}
                      begin
                         glbl_sector := 0;
                         get_new_drive(data);
                         get_disk_info(info,data);
                         direct_start := info.fat_start +
                               info.fat_copies*info.sect_in_fat;
                        display_get_back_files_commands;
                        display_status;
                      end;
              $3c00:  {F2 new file}
                      begin
                         get_file_name(fn);
                         display_get_back_files_commands;
                         {find first letter in file name}
                         first_let_pos := last_pos('\',fn) + 1;
                         first_let := fn[first_let_pos];
                         {change it to the erased indicator}
```

```
                fn[first_let_pos] := chr($e5);
                {search for the directory entry}
                glbl_sector := direct_start;
                if diskread(glbl_drive,glbl_sector,BUFSECT,data)
                     <> 0 then
                   beep;
                file_loc := search_for_file(fn,info,data);
                if file_loc = ERROR then  {not found}
                   beep
                else begin
                   {directory entry found.  Find its old size
                    and old first cluster}
                   FAT_clust := -1;
                   clrscr;
                   direct_sect := glbl_sector;
                   first_clust := data[file_loc+26] + 256*
                          data[file_loc + 27];
                   old_size_rl := ((data[file_loc+28] +
                               data[file_loc+29]*256.0 +
                               data[file_loc+30]*256.0*256 +
                               data[file_loc+31]*256.0*256.0*256) /
                               512.0) ;
                   {bytes divided by 512 = sectors}
                   old_size := round(0.49 + (old_size_rl /
                         info.sect_per_clust));
                   {= number of clusters}
                   writeln('Need to get ',old_size,' clusters');
                   {if first cluster is in use, then the file
                    is damaged}
                   if next_cluster(first_clust,info,data,FAT_clust)
                        <> 0 then
                      writeln('File is Partly Damaged');
                   delay(2000);
                   recover_erased_file(fn,first_clust,first_let,
                            old_size,direct_sect,file_loc,data
                            ,FAT_clust);
                             display_get_back_files_commands;
                             display_status;
                           end;
                       end;

             $4400:  {F10  exit}
                     finished := true;
           end;
    end;
end;
```

输入这个模块，以 erased. pas 的名字存盘，然后对 explorer. pas 做如下改动：
在常量部分，增加：

P121(2)

在 display—start—screen，增加：

P121(3)

接着，增加：

P121(4)

如有必要改变驱动器名，在主程序的 case 语句中增加：

```
$4000:  (F6)
        get_back_files(buffer);
```

存盘，编译，修改输入错误，运行。格式化一张盘，并拷贝一些文本和程序文件。删除一个文本文件，用 EXPLORER 恢复它。再试一个程序文件，再试试子目录。一次删除几个文本文件，再试着恢复它们。

注：这个程序改变了 FAT。在你真正用它恢复文件时，先在一张试验盘上试试。这样，若你的输入有错，在你真正修改一张重要盘上的 FAT 之前，就能把错误找出来。

程序设计要点：

- 当一个文件被删除时，其目录表项的第一个字节变为 e5h，文件链中的所有表项都置为 0。
- 可通过手工观察所有的空簇并拼合一个文件链，来恢复一个被删除文件。

第十章　分区表

只有硬盘有一个分区表。分区表告诉你，每个分区从哪开始，它的大小，是否是一个 DOS 分区，如果是，则告之它的 FAT 类型及是否是活动分区，有一个检查分区表的小程序，功能是证明分区表是有效的，然后装入和执行活动分区的引导记录。这章，你将使用 PART. PAS 观察分区表。

分区表硬盘的第一个扇区：0 头、0 道、1 扇区。注意，现在使用的是头、道、扇区编号体系，不是 DOS 编号体系。DOS 磁盘调用把活动分区的第一扇区当作 0 扇区。分区表位于所有分区之前，所以用 DOS 磁盘调用读不到分区表的信息。因此要用 BIOS 磁盘调用。BIOS 调用可访问磁盘的任一扇区，而不管它所在的分区。

首先看看分区信息是怎样存贮的，及怎样使用 BIOS 磁盘调用，怎样写一个观察分区表的模块。

分区表从第一扇区的 446th(1beh)字节开始，有 64 字节长。后面的字节是 55h aah。它包含四个相邻的 16 字节块。第一块是给分区 1 的，第二块是给分区 2 的，如此下去。

每块的格式如下：

字节	含　义
0	引导指示符，活动分区为 80h，其它分区为 0，一次只能有一活动分区。
1	分区开始的面。
2	低 6 位是分区开始的扇区。高 2 位是分区起始道的高 2 位。这是 BIOS 调用使用的格式。
3	分区起始道的低八位。
4	系统指示符，若是使用 16 位 FAT 的 DOS 分区，为 4；若是使用 12 位 FAT 的 DOS 分区，则为 1，其它为 0。
5	分区的结束面。
6	低 6 位是分区的结束扇区，高 2 位是结束道号的高 2 位。
7	分区结束道号的低 8 位。
8－11	双字，包含此分区前面的扇区数，先存低级字。
12－16	包含分区扇区数的双字，先存低级字。

用 FDISK 建立分区表。有时用 FORMAT 来修改它。

用 BIOS 读分区表

读分区表要用到 BIOS 磁盘读命令。上面提到，BIOS 用驱动器、面、道和扇区格式指定扇区。软盘驱动器和硬盘驱动器编号不同。软盘驱动器是 0－3，硬盘驱动器是 80h－87h，因此 B 驱动器是驱动器 1，而 C 驱动器(假设其为硬盘驱动器)是驱动器 80h，面的范围一般在 0－7 之间，道是一个 10 位数，扇区号是 6 位。

用 BIOS 中断 13h 读磁盘。可用 intr 指令调用。用下列方式设置参数：

AH	=	2
AL	=	要读的扇区数(不为 0)
CL	=	低六位是扇区号
	=	高二位是磁道的高二位
CH	=	磁道的低八位
DL	=	驱动器
DH	=	面
ES:BX	=	将要存储读信息的缓冲区指针

BIOS 中断 13h 将在第二部分详细讨论。

PART.PAS

下面是一段观察分区表的模块程序。注意该程序需要知道需观察的分区表的硬磁盘号。0 是第一个硬盘，1 是第二个硬盘，以此类推。这段程序把硬盘号加上 80h，是为了把它转换成 BIOS 磁盘编号体系。

```
{.......................PART.PAS.......................
.
. This file contains routines for reading and decoding
. the partition.
.
.....................................................}



{
: get_new_hard_drive
:
: This procedure reads in the number of the hard disk from
: which to read the partition information.  0 is for the first
: hard drive, 1 the second, and so on.
:
}

procedure get_new_hard_drive(var disk_num:integer);
   begin
      window(COMMANDX,COMMANDY,COMMAND_RT,COMMAND_BOT);
      clrscr;
      write('Number of hard drive: ');
      readln(disk_num);
   end;


{
: decode_partition
:
: This procedure decodes the partition information.  See the text
: for details.
:
}
```

```
procedure decode_partition(disk:integer;var data: buf);
   const
      TABLE_OFF = $1be;
   var
      i: integer;
   begin
      window(LFTSCRN,TOPSCRN,RTSCRN,BOTSCRN+1);
      clrscr;
      writeln('Hard disk: ',disk,' Sector: 0  Track: 0  Side: 0');
      for i := 0 to 3 do begin
         write('Partition ',i+1,' is ');
         if data[TABLE_OFF + i*16] <> $80 then
            write('not ');
         writeln('active');
         write('It is ');
         if data[TABLE_OFF + i*16 + 4] = 1 then
            write('a 12-bit FAT ')
         else if data[TABLE_OFF + i*16 + 4] = 4 then
            write('a 16-bit FAT ')
         else
            write('not a ');
         writeln('DOS partition');
         writeln('It starts at side ',data[TABLE_OFF + i*16 + 1],' sector ',
              data[TABLE_OFF + i*16 + 2],' track ',
              data[TABLE_OFF + i*16 + 3]);
         writeln('It ends at side ',data[TABLE_OFF + i*16 + 5],' sector ',
              data[TABLE_OFF + i*16 + 6],' track '
              data[TABLE_OFF + i*16 + 7]);
      end;
   end;



{
: display_part_commands
:
: This procedure displays the partion examining menu.
:
}

procedure display_part_commands;
   begin
      window(COMMANDX,COMMANDY,COMMAND_RT,COMMAND_BOT);
      clrscr;
      writeln('-----------------EXAMINE PARTITION TABLE---------------');
      writeln('F1: Choose Drive                          F10: Exit');
   end;


{
: partition_dump
:
: This is the main procedure.  It reads and displays data, and
: processes commands.
:
: Because the partition record can not be accessed through DOS,
: this procedure uses a BIOS disk read call.
:
}

procedure partition_dump(var data:buf);
   var
      finished: boolean;
      regs: result;
      disk: integer;
   begin
      finished := false;
      clrscr;
      display_part_commands;
      while not finished do begin
```

```
        regs.ax := 0;
        intr($16,regs);          {get command}
        if regs.ax and $ff = 0 then
           case (regs.ax) of
              $3b00:   {F1 new drive}
                       begin
                          get_new_hard_drive(disk);
                          display_part_commands;
                          {set up registers to read the disk using
                           a BIOS call.  Note that hard disks are
                           numbered starting with $80}
                          regs.ax := $201;
                          regs.dx := $80 + disk;
                          regs.cx := 1;
                          regs.es := seg(data);
                          regs.bx := ofs(data);
                          intr($13,regs);   {read it}
                          decode_partition(disk,data);
                       end;
              $4400:   {F10  exit}
                       finished := true;
           end;
     end;
  end;
```

输入这段程序，以 part. pas 的名字存盘，然后对 explorer. pas 做如下改动：

在 display—start—screen，增加：

```
writeln('F7:   Examine Partition');
```

然后，增加：

```
{$I b:part.pas}
```

改变驱动器(如有必要的话)。

在主程序的 case 语句中增加：

```
$4100:  {F7}
        partition_dump(buffer);
```

存盘，编译 EXPLORER，改正输入错误。然后，观察你硬盘上的分区表。

程序设计要点：

- 只有硬盘才有分区表。
- 分区表告诉每个分区的位置。对 DOS 分区，还告诉其 FAT 类型。该表还指出用来引导的分区。
- 分区表位于 0 头、0 道、1 扇区只能用 BIOS 中断读分区表。

第十一章　磁盘问题及技巧

FAT 会出现一些问题，但是很少。当不正确地更新 FAT 时，就会发现这些问题。如果在 FAT 写的过程中移动磁盘，如果介质减少，或是出现一般的程序故障，例如只重写了 FAT 的一部分，FAT 都会有问题。这些问题在本章里指的是交叉链接，循环链接及部分链接的文件和孤簇。文件存储碎片是另一个 FAT 问题。

当一个文件的文件链指向另一个文件的文件链时，就会出现**交叉链接的文件**。两个文件共享同一簇组。编写检查交叉链的程序时，要设置一个叫做 COUNT 的缓冲区，与 FAT 一样大，置 0。然后读磁盘上每个文件的文件链。这需要几种树检索，例如一个深度优先的检索，来读所有的目录结构。每次使用文件链中的一个簇时，就把 COUNT 中的簇值加 1，做完之后，任何 COUNT 大于 1 的表项都为交叉链接的文件。为了修正交叉链，要确定哪个文件包含有重叠数据区的簇，并在那点打断了其它的文件链。还需检索磁盘，把文件链的结尾放到其它簇里，可对 EXPLORER 进行一些改动，增强其功能，然后用 EXPLORER 来修复交叉链。

循环链接的文件是与自己交叉链接的文件，也就是文件链中的一个簇指向该链中前面的一簇，所以文件没有结尾。COUNT 计数相当大的表项就表明这是一个循环链接的文件。为修正循环链，要找到循环开始的地方并打破此链接。再有，为修复文件还要把链的结尾放到其它簇中。

现在试试一个循环链接的文件。拷贝一个长度超过一个簇的文本文件。用 EXPLORER 找到它的第一个簇。改变这个簇的 FAT 表项，使它指向它自己。然后退出 EXPLORER，TYPE 该文件，那么第一簇将一直循环重复直到字节数等于文件长度。现在用 EXPLORER 改变目录长度表项，让文件更长，再 TYPE 一下。

部分链接的文件是它们的文件链在文件结束标志之前就断了，最后一簇或是一空簇或是一坏簇。修改这种文件的方法是把最后一个表项或倒数第二个表项改为文件结束标志。完全修复文件，还要链接更多的簇。

孤簇是那些标志为正在使用但又不是任何文件部分的簇。因此它们从来不被读。但是它们影响了磁盘的使用空间。把 COUNT 与 FAT 相比，可检查孤簇。任何 COUNT 计数为 0 但在 FAT 中又非空非坏的簇就是孤簇。解决孤簇，可以把孤簇改为空簇也可生成一个新文件，把孤簇分配给该文件。CHKDSK 执行这个功能。

磁盘存储碎片引起执行问题。当一个文件需要一个新簇时，就使用下一个空簇。如果在第一个文件生成之后，在这第一个文件需要新簇之前写了另外一个文件，那么在第一个文件需要一个新簇时，DOS 分配给它的新簇就不再和这第一个文件的其它簇相邻了。实际上，在磁盘使用过多后，文件可能会被分配到磁盘的各个部分。这是硬盘的一个特殊问题。当一个文件被分成碎片后，访问它就需要更多的时间，因为磁头要移动很多次。通过文件链可以看到存储碎片。通过 CHKDSK 也能看到。为拼接文件碎片，要读文件链直到发现不相邻的簇，然后把文件数据从这个不相邻的簇移到相邻的簇中。例如，假设一相邻部分的最后一个簇是 7，接下来的簇是 20，把 8 簇的数据移到别处，修改指向它的文件链。然后把 20 簇的数据移到 8 簇并修改文件链。每个文件都这样修改使它们的簇全部相邻。这项工作要花费很多时间。

上面给出的这个方法有些简单，稍微修改一下可提高其过程速度。例如，如果磁盘没有全用，可以先查出分片文件的长度，然后在磁盘上找到一组相邻的空簇，大小要大于分片文件的长度，然后把整个文件都移到该部分。拼接软盘上分片文件的最好方法是用 XCOPY 把它拷贝到一张空盘上。

磁盘技巧

了解了磁盘及其所能产生的问题后，现在试试一些解决技巧。首先，把目录里的日期和时间表项改成很大的值，看 DOS 怎样显示。接着，隐藏一个子目录，并 DIR 磁盘。用 alt 键输入扩展字符(如£符号)到一个文件名中。在文件名和卷名中输入小写字母。这些在目录中都能显示出来，但是就不能访问这些文件了。用这些特点，可以编写一个非常简单的加密程序。

给磁盘起一个名字，该名字是一块正文，也许有几行长，它将给出比卷名更多的信息。用 EXPLORER 的带有扇区转储特点的十六进制显示，在目录中设置几个虚拟子目录表项。在文件名部分输入信息，一定要用小写字母，这样，这些伪文件就不可能被访问到。可用制表符、回车和换行清屏。例如在文件扩展名部分输入制表符号，在最后一行键入一些换行键，结果如下：

```
Volume in drive B has no label
Directory of  B:\

COMMAND  COM    23210   3-07-85   1:43p
********                          <DIR>      15-31-0107  19:63p
This is                           <DIR>       7-22-0103  18:55p
a disk                            <DIR>       7-22-0103  18:55p
title                             <DIR>       7-22-0103  18:55p



                                  <DIR>       7-22-0103  18:55p
PART     COM    30549   6-05-86   1:46p
       7 File(s)    308224 bytes free
```

为去除磁盘名字，把每个名字表项的第一个字符置 e5h 即可。如果把目录表项的所有时间和日期字节都置 0，那么 DIR 时，DOS 就不会打印出文件的时间和日期。

在第 18 章，你将学会通过把整个目录从一个目录移到另一个目录重新组织你的磁盘。还将学会给一个目录设置几个目录名字。例如目录\BOOKPROG 和\PROGRAMS\DOSBOOK\UTILITY 都指向相同文件。

另一个你会用到的方法是改变引导记录的前几个字节，使 IBM DOS 4.0 可辩认出非 IBM DOS 或 OS/2 分区。IBM DOS 4.0 检索引导记录的 3—6 字节，判断是否是“IBM”(3—10 字节是 DOS 或 OS/2 版本的 ASCII 名字)。如果磁盘不是用 IBM 版本的操作系统格式化的，那么 IBM DOS 4.0 将拒绝辩认该盘。

利用 EXPLORER 把引导记录的前几个字节设置成“IBM”是很容易的。

程序设计要点：

- 磁盘会有一些问题发生，最严重的是 FAT 的破坏。
- 通过修改存储在磁盘组织结构里的数据，可得到几个磁盘使用技巧。

第十二章　改变DOS内部命令

这章将用 EXPLORER 改变 DOS 的内部命令，观察 COMMAND. COM。在前几簇之后，是 DOS 的错误信息和一张 DOS 内部命令表。每个命令都以 ASCII 文本显示，其后是一个四字节 jump 指令，指向该命令的代码地址。若想改变 DOS 的内部命令名，就在这儿改，并把改动存盘。键入 DOS 及 COMMAND 重新装载命令层。一定不要改动四字节指针。若想废用一个命令，把它变为小写即可。例如把 DEL 改为 del。

当然，这样改动有一点冒险，有可能不小心改动了指针部分或者在想恢复它时却忘了是怎样改的了。编写个改命令程序就可避免这些问题。该程序的功能是记住旧命令并读 COMMAND. COM，看该命令是如何被改动的。把想改变的命令告诉程序，并输入你的新命令。程序截断该命令并更新文件，还可让程序把命令回复到 DOS 的设置。

有一点要注意，DOS 版本不同，COMMAND. COM 也不同。DOS 2. 0—3. 2 版本的命令表是一样的——只是位置和四字节指针变了。程序必须能知道它所观察的 COMMAND. COM 的 DOS 版本，在该版本中命令表的偏移量及每个命令在命令表中的偏移量。如果观察的是 DOS 3. 3 或 4. 0 的 COMMAND. COM，该程序还要能够处理两个新命令，在 DIR 之后的 CHCP 和 CALL，方法是把所有命令的偏移量加 16。

COMMAND. COM 的每个版本都用一个文本字符串来标识其版本号。有一简单程序专为扫描 COMMAND. COM 的这个信息，打印出它的位置，及在命令表中的偏移量。3. 30 之前的版本，命令在命令表中的偏移量是一样的。

DOS 版本	标识符	偏移量	命令表偏移量
2. 0	2. 00	3839	14794
2. 1	2. 10	3839	14813
3. 0	3. 00	3993	18794
3. 1	3. 10	4283	19473
3. 2	3. 20	4571	20017
3. 3	3. 30	5003	21348
4. 0	4. 00	4864	32965

标识符是一 ASCII 字符串，用 NEWCMMDS 列表看命令表中的命令偏移量表。

NEWCMMDS

现在看看改变 DOS 内部命令的程序。该程序需要知道命令文件的名字——假使你已把它重新命名，这样程序就不会改变原来的 COMMAND. COM——并扫描版本标识符。用标识符信息确定命令表的偏移量，并把此作为要读的文件长度。然后读入当前命令名，同原来的 DOS 命令名一起显示。程序需要知道要改变的表项号和要改成的新命令的正文，作为可选项把原来的命令名存起来。新命令名转换成大写。改动可存盘也可放弃。

```
{modifies the names of DOS internal commands}

program make_new_commands;

{$IFNDEF Turbo3}
uses dos, crt;
{$ENDIF}

const
   NUM_CMDS = 20;
type
   {stores a command name}
   name_str = string[6];

   {stores offset, original, and new command names.  Original name
    is never changed.}
   command_info = record
      offset: integer;
      old_name: name_str;
      new_name: name_str;
    end;

   {stores info for each command}
   commands = array[1..NUM_CMDS] of command_info;

   {for checking DOS version}
   ver_str = string[4];

  {for reading from disk}
  buf = array[0..5119] of byte; {40 blocks of 128 bytes}

var
   cmd_file: file;
   cmd_name: string[14];
   cnt: integer;
   cmd_list: commands;
   buffer: buf;
   cmd_offset,old_offset: integer;
   ver_num: ver_str;


{
: initialize_cmd_list
:
: This procedure sets up the offsets and original command names for the
: command list.
:
: Just uses the major commands.  Could be changed to modify all of the
: commands or error messages as well.
:
: Note that there are two new commands, CALL and CHCP, for DOS 3.30.
: Their offset must be adjusted for.
}

procedure initialize_cmd_list(var cl:commands;ver_num: ver_str);
   var
      i: integer;
   begin
      cl[1].offset := 0;
      cl[1].old_name := 'DIR';
      cl[2].offset := 7;
      cl[2].old_name := 'RENAME';
      cl[3].offset := 17;
      cl[3].old_name := 'REN';
      cl[4].offset := 24;
      cl[4].old_name := 'ERASE';
      cl[5].offset := 33;
      cl[5].old_name := 'DEL';
```

```
      cl[6].offset := 40;
      cl[6].old_name := 'TYPE';
      cl[7].offset := 55;
      cl[7].old_name := 'COPY';
      cl[8].offset := 72;
      cl[8].old_name := 'DATE';
      cl[9].offset := 80;
      cl[9].old_name := 'TIME';
      cl[10].offset := 88;
      cl[10].old_name := 'VER';
      cl[11].offset := 95;
      cl[11].old_name := 'VOL';
      cl[12].offset := 102;
      cl[12].old_name := 'CD';
      cl[13].offset := 108;
      cl[13].old_name := 'CHDIR';
      cl[14].offset := 117;
      cl[14].old_name := 'MD';
      cl[15].offset := 123;
      cl[15].old_name := 'MKDIR';
      cl[16].offset := 132;
      cl[16].old_name := 'RD';
      cl[17].offset := 138;
      cl[17].old_name := 'RMDIR';
      cl[18].offset := 173;
      cl[18].old_name := 'PROMPT';
      cl[19].offset := 183;
      cl[19].old_name := 'PATH';
      cl[20].offset := 245;
      cl[20].old_name := 'CLS';

      {Adjust offsets for DOS 3.30.}
      if (ver_num = '3.30') or (ver_num = '4.00') then
         for i := 2 to 20 do
            cl[i].offset := cl[i].offset + 16;

   end;



{
: make_string
:
: This procedure reads four bytes from the buffer and converts them to
: string format.
:
}

procedure make_string(var st: ver_str;var buffer: buf;offset: integer);
   var
      i: integer;
      n: ver_str;
   begin
      for i := 1 to 4 do
         n[i] := chr(buffer[offset + i - 1]);
      st := copy(n,1,4);
   end;



{
: get_ver_num
:
: This procedure reads the command file and checks for the version number
: identifier.  The identifier is returned in ver_num.  Returns ERR! if no
: identifier is found.
:
}
```

```
procedure get_ver_num(var ver_num:ver_str;var buffer:buf;var cmd_file:file);
   begin
      blockread(cmd_file,buffer,40);
      make_string(ver_num,buffer,3839);
      if (ver_num <> '2.00') and (ver_num <> '2.10') then begin
         make_string(ver_num,buffer,3993);
         if ver_num <> '3.00' then begin
            make_string(ver_num,buffer,4283);
            if ver_num <> '3.10' then begin
               make_string(ver_num,buffer,4571);
               if ver_num <> '3.20' then begin
                  make_string(ver_num,buffer,5003);
                  if ver_num <> '3.30' then
                     make_string(ver_num,buffer,4864);
                     if ver_num <> '4.00' then
                        ver_num := 'ERR!';
               end;
            end;
         end;
      end;
   end;



{
: get_offset
:
: This function takes the version identifier as its argument and returns
: the location of the command table.  5120 bytes are subtracted because
: they have already been read in to determine the version number.
:
}


function get_offset(ver_num: ver_str): integer;
   begin
      if ver_num = '2.00' then
         get_offset := 14794 - 5120
      else if ver_num = '2.10' then
         get_offset := 14813 - 5120
      else if ver_num = '3.00' then
         get_offset := 18794 - 5120
      else if ver_num = '3.10' then
         get_offset := 19473 - 5120
      else if ver_num = '3.20' then
         get_offset := 20017 - 5120
      else if ver_num = '3.30' then
         get_offset := 21348 - 5120
      else
         get_offset := 32965 - 5120;
   end;



{
: make_string2
:
: This procedure is exactly like make_string, except that it returns a
: string of type name_str.
:
}

procedure make_string2(var st: name_str;var buffer: buf;offset,
      len: integer);
   var
      i: integer;
      n: name_str;
   begin
```

```
      for i := 1 to len do
         n[i] := chr(buffer[offset + i - 1]);
      st := copy(n,1,len);
   end;


{
: read_commands
:
: This procedure reads through the command file until it comes to the
: portion containing the command table.  It then reads the names stored
: in the command table.
:
}

procedure read_commands(var offset:integer; var cmd_list:commands;
      var buffer:buf; var cmd_file: file);
   var
      i: integer;
   begin
      while offset > 5120 do begin
         blockread(cmd_file,buffer,40);
         offset := offset - 5120;
      end;
      blockread(cmd_file,buffer,(offset + cmd_list[NUM_CMDS].offset) div
            128 + 1);

      for i := 1 to NUM_CMDS do
         make_string2(cmd_list[i].new_name,buffer,offset +
               cmd_list[i].offset,length(cmd_list[i].old_name));
   end;

{
: display_commands
:
: This procedure prints out the original and new command names.
:
}

procedure display_commands(cmd_list: commands);
   var
      i: integer;
   begin
      clrscr;
      writeln('#      Old Command      New Command');
      writeln;
      for i := 1 to NUM_CMDS do begin
         write(i);
         gotoxy(8,i+2);
         write(cmd_list[i].old_name);
         gotoxy(25,i+2);
         writeln(cmd_list[i].new_name);
      end;
   end;

{
: prepare_to_write
:
: This procedure repositions the file pointer to the beginning of the
: command table section.
:
}

procedure prepare_to_write(offset: integer;var cmd_file: file);
   var
      buffer: buf;
   begin
      offset := offset + 5120;
      close(cmd_file);
      reset(cmd_file);
      while offset > 5120 do begin
```

```
        blockread(cmd_file,buffer,40);
        offset := offset - 5120;
      end;
   end;



{
: modify_commands
:
: This procedure prompts for changes to command names.  It changes all
: new names to uppercase commands.  You may want to remove this feature.
:
}

procedure modify_commands(cmd_list:commands;var buffer:buf;
      var cmd_file: file; offset:integer);
   var
      i,j: integer;
      changed_name: string[6];
   begin
      repeat
         display_commands(cmd_list);
         write('Enter # to change, or 100 to write, or 200 to exit: ');
         readln(i);
         if i = 100 then
            blockwrite(cmd_file,buffer,(offset + cmd_list[NUM_CMDS].offset)
                              div 128 + 1)
         else if (i > 0) and (i <= NUM_CMDS) then begin
            write('New name (<Return> to restore original): ');
            readln(changed_name);
            if length(changed_name) = 0 then
               changed_name := cmd_list[i].old_name;
            for j := 1 to length(cmd_list[i].old_name) do begin
               cmd_list[i].new_name[j] := upcase(changed_name[j]);
               buffer[offset + cmd_list[i].offset + j - 1] :=
                  ord(upcase(changed_name[j]));
            end;
         end;
      until i = 200;
   end;

   begin
      clrscr;
      writeln('===============DOS Command Changer================');
      writeln;
      writeln;
      writeln;
      write('Enter the name of the COMMAND.COM file to change: ');
      readln(cmd_name);
      assign(cmd_file,cmd_name);
      reset(cmd_file);
      get_ver_num(ver_num,buffer,cmd_file);
      writeln('Version is DOS ',ver_num);
      initialize_cmd_list(cmd_list,ver_num);
      delay(2000);
      if ver_num <> 'ERR!' then begin
         cmd_offset := get_offset(ver_num);
         old_offset := cmd_offset;
         read_commands(cmd_offset,cmd_list,buffer,cmd_file);
         prepare_to_write(old_offset,cmd_file);
         modify_commands(cmd_list,buffer,cmd_file,cmd_offset);
      end;
      close(cmd_file);
   end.
```

输入这段程序，以 newcmmds. pas 名存盘，编译，改正输入错误，运行。第一次运行时，最好把 COMMAND. COM 做一次拷贝，例如拷到 COMMAND2. COM，这样运行中若有输入错误，

COMMAND2.COM 也不致遭到破坏。改变几个命令名字，存盘，退出，键入 COMMAND（或 COMMAND2）重新装载改过的命令。试试用原来的命令名和新命令名。

该程序的一个应用实例是重新命名 DEL 和 ERASE 命令，使文件不会由于误操作而遭到破坏。

程序设计要点：

- COMMAND.COM 包含 DOS 内部命令名，它们的位置因 DOS 版本不同而不同。
- 通过编辑 COMMAND.COM 可改变 DOS 命令。

第二部分　BIOS 和 DOS 中断及其实用程序

这部分，你将学会怎样使用 BIOS 和 DOS 中断。你将知道怎样使用屏幕，键盘，磁盘和存贮器，怎样编写实用程序，还将知道其它有用的中断——包括 Microsoft 没有告诉你的在内。

在你编写的绝大多数汇编语言程序和高级语言程序中都要用到中断。例如，你可在汇编语言程序中用中断访问磁盘文件，或在 C 语言程序中用中断生成图形。

第十三章　中断和汇编程序设计简介

中断是执行计算机和外设间基本交互的子程序集。本章中，中断指的是软中断。硬中断将在第十四章讨论。与你所编程序中的子程序不同，中断是常驻在计算机里的。中断基本上有两种：BIOS 中断和 DOS 中断。BIOS 中断存在 ROM 里，执行与硬件的最基本交互。例如，有读磁盘扇区的中断，有往屏幕写字符的中断，还有读光笔位置的中断。

大多数 DOS 中断也处理计算机和外设之间的交互，只是这些中断处在一个更高的级别。DOS 中断处理文件和目录或者输入行以及内存分配，这些都是在 BIOS 中断基础上实现的。每次引导 DOS 的同时，装入 DOS 中断。DOS 中断随 DOS 版本的不同而不同。DOS 的后期版本对先期版本的中断进行了修改和提高，并增加了一新中断。

调用中断用一个特殊的汇编语言命令——int 命令。每个中断有一与其相关的号，并使用一组送入 8086 寄存器里的参数。

许多中断都有子特征。每个子特征叫做功能。当调用一个有几个功能的中断时，要将功能号放在 ah 寄存器里。例如，中断 21h 是主要的 DOS 中断，有将近 100 个功能，调用中断 21h 的功能 1 时，要这样：

```
(set parameters)
mov ah,1;select function
int 21h ;do interrupt
```

使用中断很简单。当你需要中断服务时，只需设置寄存器并调用中断即可。

本书中，中断号采用十六进制记数法，功能值采用十进制记数法(以免混乱)。

为什么使用中断?

中断非常有用，因为它们能提供可移植性，加速软件设计和调试，并提供一个简单而有力的访问所有硬盘的方法。中断根据新硬件的类型而更新，以支持它们。这使程序具有可移植性——即从一个机器环境移到另一个机器环境的能力。

由于中断能处理硬件的所有关键细节，所以可节省大量时间。例如，BIOS 要考虑怎样为磁盘驱动器控制器设计程序，处理出错。而你只需读一些扇区，这一点至关重要，因为有这么多的硬件设备和直接处理硬件的程序，而这些程序又很难调试。

更进一步，中断还节省程序空间。中断调用很短，而子程序相对就要长一些。中断不需要你去专门开发在程序间传送的标准程序库。

用 DOS 中断，不用考虑任何用在处理文件和设备的内部结构。你只需告诉 DOS“寻找这个文件”或“在那个文件里增加这一行”，而无需译码文件链和目录。

中断和实用程序

中断是大多数实用程序的中心，它们对计算机所有部分提供几乎完全的控制。下面几章，你将观察不同的中断，看看怎样利用它们增强你程序的功能。你将学会处理屏幕，磁盘和内存，还将学到处理其它中断、程序和重要程序设计结构的中断。

注：中断执行的功能有几百个。许多后期开发的中断对前期的中断作了改进；前期的中断保留了向上兼容性。如果执行同一功能时后期中断比早期中断执行得更好，并在程序设计上有一些混乱时，那么就以后期中断为主，对前期中断不再加以讨论。

汇编语言程序的结构

从这本书的其余部分，你将学到许多处理汇编语言程序的技巧。汇编语言是最基本的计算机语言（这里不考虑微程序，因为它不是 RAM 可执行的）。它由短助记符组成，可直接翻译成命令计算机执行非常基本的指令的数字，如"add two bytes"或"read a byte in memory"。

汇编语言程序是汇编语文指令的集合。任何数字集合都能被装入计算机内存的任何地方，并被当作是一个汇编语言程序。这样的程序可挂起计算机。因此当你编写汇编语言程序时，必须有一个结构，这样才能正确执行。

汇编语言程序的结构有三部分：初始化、操作和结束。**初始化部分**设置所有寄存器和数据区以为程序操作使用，检查计算机和正在使用的外设的类型。**操作部分**包含程序的主体部分——所有的输入、数据处理和输出指令。**结束部分**包括修改计算机可能产生的任何混乱的指令，和安全返回 DOS 的指令。

还有一些概念必须知道。BIOS 和 DOS 在内存地址的顶部和尾部保留了一些空间。DOS 要记录空的内存地址。当一个程序运行时，该程序通常是装在最低的可能内存位置。运行结束，释放占用的内存空间。程序可以是**内存驻留**的。这样程序运行结束后，它们使用的内存空间予以保留。下一个程序就装在它们后面的内存里。因此内存驻留程序总是装在内存里，任何时候只要选择了该程序，它的指令就能执行。详见第三部分。

程序还能产生子程序。父程序告诉 DOS 执行子程序，子程序装在父程序之后的内存里。子程序执行结束，控制返回到父程序。在某种意义上说，你编写的所有程序都是 DOS 的子程序。

产生子程序是一个非常有用的手段；它允许你在运行你自己的程序时，使用其它程序。例如，假设你写了一个 LISP 计算机语言的解释程序。LISP 解释程序可同时读入几组定义（LISP 程序）。假设你必须对其中的一个文件做一个改动。LISP 解释程序就能产生一个字处理程序。当你退出这个字处理程序后，不用再启动就可返回 LISP。在你的程序运行期间，还可执行一个 DOS 命令，例如若在程序运行中间用完了磁盘空间，可使用 FORMAT。

程序也可以覆盖方式装入，它替换了目前不用的一部分程序码。若程序太大，不能一次全部装入时，可采用这种方法。

汇编和编译程序的扩展名为 EXE 或 COM。COM 程序一定要小于 64K，且不能有栈段，它们占用较少的磁盘空间，装入时比 EXE 文件快。EXE 文件可大于 64K，可随心使用不同类型的段。COM 和 EXE 程序执行时，段寄存器设置方式稍有不同。COM 文件最好小一些。最简单的方法是只使用一个段，程序以 org 100h 命令开始。连接后，使用 EXE2BIN 命令生成 COM 文件。

DOS 使用标准输入和输出设备。一般输入设备是键盘，输出设备是屏幕。可用＞和＜DOS 命令指定。程序不能确定输入或输出是否做重定向了。用 DOS 命令写的读写标准输入输出设备的程序也一样，不考虑重定向。

编程须知：

汇编语言程序可能会非常难理解，除非你采取一些预防措施。尽可能用结构化设计方法。

程序块中只包含一个入口和出口，使用子程序。给变量起有意义的。程序中加注释。

只要可能就使用中断——而不直接对硬件进行编程——以增强程序的可读性和可移植性。记住，中断经常在寄存器和标志中返回结果，若需要一个被改变的寄存器，那么在调用中断前，一定要保留寄存器中的内容。

如果写 EXE 程序，设置一个至少 800 字节的栈，要比你程序所需的大，越大越好。程序也许用不了这么多，但内存驻留程序有可能会用到。不要假设一特定的外设是连着的。如果需要一个设备，例如图形板，在用之前先检查一下。

程序设计要点：

- 中断在计算机和其外设间提供一易于使用的连接。
- 使用中断可使程序更短，移植性更好，更容易开发和更新。
- 汇编语言程序有开始，操作，结束三部分。
- 使用结构化程序设计方式，避免假定。
- 产生子程序是一个有用的手段。
- 如可能把实用程序变为 COM 文件。

第十四章　输出，屏幕控制，正文和图形

计算机可以几种方式显示信息。配有彩色监视器和适配器，能显示 40 或 80 列的正文或图形。配有单色适配器，能显示 80 列的正文。显示方法叫做**屏幕方式**。计算机启动时，总是使用 80 列正文显示格式。要使用不同的格式，必须改变屏幕方式。在本章，你将学会控制屏幕，正文和图形。

用中断 10h，功能 0 设置屏幕方式。设置屏幕方式的同时也清屏了。

操作：设置屏幕方式
版本：BIOS
中断号：10h
功能号：0
调用值：
　AH＝0
　AL＝屏幕方式
返回值：AX 被破坏

屏幕方式号如下：

号	显示方式	适配器
0	B&W40 列正文	CGA，EGA，3270
1	彩色 40 列正文	CGA，EGA，3270
2	B&W80 列正文	CGA，EGA，3270
3	彩色 80 列正文	CGA，EGA，3270
4	彩色中分辨率图形	CGA，EGA，3270
5	B&W 中分辨率图形	CGA，EGA，3270
6	高分辨率图形	CGA，EGA，3270
7	单色 80 列正文	MONO，EGA，3270
13	16 色中分辨率图形	EGA
14	16 色高分辨率图形	EGA
15	4 色调单色图形	EGA
16	4 色或 16 色超高分辨率图形	EGA
48	3270 图形	3270

选择视频页

配有 CGA 和 EGA 板，正文方式可包括几页，每页包含一数据屏。开关页确定显示哪一个数据屏，被确定的屏就成为活动页。页从 0 开始编号。

在显示一页的同时，可往另一页写数据，这就使屏幕改变非常方便。在不显示的页上做所有的更新工作。在完成这些工作之后，把这页变为活动页。还可在一页上存储 help 信息，需要时再翻阅它。

设置屏幕方式时，活动页置为 0。除非你必须使用一页以上的显示页(大多数程序不需要)，否则你不用考虑设置活动页——总是使用 0 页。

40 列正文屏幕包含 8 页；80 列正文屏幕包含 4 页。使用单色适配器时，单色 80 列正文屏幕只包含一页，而使用 EGA 适配器的则包含 8 页。

操作：设置活动页
版本：BIOS
中断号：10h
功能号：5
调用值：
　　AH＝5
　　AL＝页号
返回值：AX 被破坏

确定屏幕方式和活动页

当一个程序启动时，最好重新设置屏幕方式。这样，屏幕方式和活动页就可得到保证。一些应用程序，象弹出内存驻留程序，在改变屏幕方式之前必须知道原来的屏幕方式和活动页。

操作：确定屏幕方式和活动页
版本：BIOS
中断号：10h
功能号：15
调用值：AH＝15
返回值：
　　AL＝屏幕方式
　　AH＝屏幕上的字符列
　　BH＝活动页

清屏

使用单色适配器时，清除所有页或清屏，要重新设置屏幕方式。清除一单个的显示页，打印 50 行空行和一个回车。

打印字符

有几个中断可打印字符，它们在处理控制字符，属性和光标时稍有不同。

当打印原始十六进制数据时，选择一个不处理控制码的中断，这样屏幕上的数据是排列成行的。如果打印输出正文数据又不需记录行或处理控制码，就使用一个处理控制码的中断。它将自动处理回车，制表格和换行。

移动光标的中断把光标放在最后一个打印的字符的后面。如果使用非光标调节的中断，则在打印每个字符之前必须先设置光标。

属性是描述字符颜色和亮度的一个字节。在彩色正文屏幕中，低位控制字符的颜色，高位控制背景的颜色。通常把背景设置成亮色以使字符闪烁。

值	颜色
0	黑色
1	蓝色
2	绿色
3	青蓝色
4	红色
5	紫色
6	棕色
7	白色
8	灰色
9	深蓝色
10	深绿色
11	淡青色
12	淡红色
13	淡紫色
14	黄色
15	亮白色

在一单色屏幕中属性含义更复杂。低位控制前背景，高位控制后背景。如果一个属性字节的低三位是 0，字符(或背景)就是黑色的，否则，就是白色的。设置低属性字节的高位使字符更亮；设置高属性字节的高位使字符闪烁。低三位为 001 使字符有下划线。下表为属性实例。用亮度和闪烁位增强属性。

值(十六进制)	结果
00	无显示
01	下划线
07	正常显示
70	反向显示
77	显示白色块

在图形方式里，属性选择字符的颜色，如果设置属性的高位，字符就将与屏幕异或。用异或方式恢复字符下面原来存在的内容。第一次异或一个字符，该字符显示；第二次异或该字符，它就消失。

一些中断需要知道它们要写的显示页。对没有显示的一些方式——象图形方式——将忽略该参数。

操作：打印一个字符
　　属性设置
　　光标不移动
　　不处理控制字符
版本：BIOS
中断号：10h
功能号：9
调用值：
　AH＝9
　AL＝字符
　BL＝属性
　BH＝页
　CX＝重复字符计数。图形方式不回行。文本方式回行
返回值：AX 被破坏

操作：打印一个字符
　　属性不改变
　　光标不移动
　　不处理控制字符
版本：BIOS
中断号：10h
功能号：10
调用值：
　AH＝10
　AL＝字符
　BL＝图形方式中为属性，否则未用
　BH＝页
　CX＝重复字符计数。图形方式中不回行。文本方式中回行。
返回值：AX 被破坏

操作：打印一个字符
　　属性不改变
　　处理控制码

光标可调节
往活动页写

版本:BIOS
中断号:10h
功能号:14
调用值:
AH=14
AL=字符
BL=图形方式中为属性,否则未用。
返回值:AX 被破坏

操作:打印一个字符
属性不改变
不处理控制码
往活动页写
光标可调节
输出重定向到标准输出设备
版本:DOS 1.0—
中断号:21h
功能号:2
调用值:
AH=2
DL=字符
返回值:AX 被破坏

显示字符串

一些打印字符串的中断,但它们不使用标准的字符串格式。首先它们要求知道长度,或用一个 $ 作为结束符。这都是人为约定。普通的字符串结束符(在 C 和 UNIX 里)是一个 0,这使程序员在计算正文长度时遇到了麻烦。所以必须有一个程序,接受一个正文字符串的指针,重复调用中断 10h,功能 14 直到结束符 0。如果 想改变输出方向,用中断 21h,功能 2 的指针取代该中断。

```
;
;disp_msg
;
;     Displays an ASCIIZ string.
;     Called with:
;         DS:DX pointing to string
;   Returns with:
;          all registers preserved
;
;     The color for graphics modes is set up as a constant.  You
;     may wish to make this a passed parameter.
;
;     The characters will always be displayed on the screen.  To
;     allow for output redirection, change the interrupt 10h
```

```
;     call to an interrupt 21h, function 2 call, and set up the
;     registers appropriately.
;
disp_msg        proc    near
                push    ax
                push    bx
                push    si
                mov     bl,FORE_COLOR   ;color for graphics modes
                mov     si,dx           ;point to string
d_m_loop:       mov     al,[si]         ;load character
                cmp     al,0            ;check for end
                je      d_m_end
                mov     ah,0eh          ;print char service
                int     10h             ;print it
                inc     si
                jmp     d_m_loop
d_m_end:        pop     si
                pop     bx
                pop     ax
                ret
disp_msg        endp
```

控制光标

如果你用不调节光标的字符打印服务或不在屏幕当前光标位置打印字符，那么你必须能读并且设置光标位置。每一个显示页有不同的光标。光标调用适用于正文和图形屏幕。

操作：设置光标位置
版本：BIOS
中断号：10h
功能号：2
调用值：
AH＝2
DH＝行
DL＝列
BH＝页(图形方式设置成 0)
返回值：AX 被破坏

操作：读光标位置
版本：BIOS
中断号：10h
功能号：3
调用值：
AH＝3
BH＝页号(图形方式设置成 0)
返回值：
AX 被破坏
CH＝光标起始行(见下面中断)
CL＝光标结束行(见下面中断)
DH＝行

DL＝列

操作：改变光标形状
版本：BIOS
中断号：10h
功能号：1
调用值：
AH＝1
CH＝光标起始行
CL＝光标结束行
光标行在一字符块的顶部，从 0 开始编号。彩色图形板，有 8 个光标行；单显和 EGA，有 14 个光标行。若开始行在结束行之后，光标将返转到字符块的顶部。不要使用 15 以上的号。
返回值：AX 被破坏

从屏幕读字符

一个输出例程或内存驻留程序可能需要读在屏幕某一位置上显示的字符。屏幕读中断从图形和正文屏幕读字符。如果想读一大块屏幕，或者要求速度，可直接读屏幕内存（见第 28 章“编写弹出程序”）。

操作：读光标位置的字符
从图形屏幕中读
版本：BIOS
中断号：10h
功能号：8
调用值：
AH＝8
BH＝显示页。不要用在图形方式中。
返回值：
AL＝字符
AH＝属性

图形

几个中断处理图形。显示图形，首先要用设置方式中断选择一个图形方式。

操作：选择调色板或背景颜色
用于中等分辨率图形
版本：BIOS
中断号：10h

功能号:11
调用值:
AH=11
BH=0(选择背景颜色)
=1(选择调色板)
BL=背景颜色或调色板号
返回值:AX 被破坏

此中断用于中分辨率图形。背景色可是 16 种颜色值的一种。有四种不同的调色板。在选择调色板之前要设置背景色。用从 0—15 的背景色选择调色板 0 或 1,把背景色加上 16,就可使用调色板 2 或 3。

调色板号	颜色 1	颜色 2	颜色 3
0	绿	红	棕
1	青	紫	白
2	亮绿	亮红	黄
3	亮青	亮紫	亮白

配有 EGA 卡,一次能使用 16 种颜色。每种颜色是通过挑选红、绿和蓝的亮度来选择的。由于亮度范围是从 0—3,所以一共有 64 种颜色可用,颜色由下列格式的一个字节描述:

EGA 颜色值

位	含义
0	蓝色亮度值的低位
1	绿色亮度值的低位
2	红色亮度值的低位
3	蓝色亮度值的高位
4	绿色亮度值的高位
5	红色亮度值的高位
6,7	未用

例如亮蓝色可用 11h,暗蓝色可用 1。可写一个程序试试所有可能的颜色。

操作:设置 EGA 颜色值
版本:EGA BIOS
中断号:10h
功能:16
调用值:
AH=16
AL=0

BL＝要设置的颜色号(0－15)
BH＝要使用的颜色(0－63)
返回值:AX 被破坏

下面两个中断是用来设置和读图形象素的。坐标位置没有范围限制。如果它们超过了屏幕的界限,它们就会绕回。

操作:设置图形象素
版本:BIOS
中断号:10h
功能:12
调用值:
AH＝12
AL＝象素颜色。设置要异或的高位
BH＝显示页(CGA 设置成 0)
CX＝X 坐标
DX＝Y 坐标
返回值:AX 被破坏

操作:读图形象素
版本:BIOS
中断号:10h
功能:13
调用值:
AH＝13
BH＝显示页(CGA 置成 0)
CX＝X 坐标
DX＝Y 坐标
返回值:AL＝象素的颜色值

程序设计要点:

- 有几种正文和图形屏幕方式。一些方式不止一页,在屏幕上显示的页叫做活动页。
- 可在正文和图形屏幕上读、写字符。
- 字符显示中断在控制码的处理、颜色的使用和光标的移动上有所不同。
- 几个与图形有关的中断。

第十五章　输入：键盘、光笔和鼠标器

许多中断的功能是读键盘。这些中断在报告击键，等待击键，回显击键和处理 Ctrl－Break 的方式上有所不同。本章讨论两种类型的击键：ASCII 键和扩充键。ASCII 键是字母数字键和控制键，扩充键包括功能键和光标键。一些中断报告扩充键结果需要两次调用。若程序中需要输入功能键和光标键，应使用一个立即返回扩充字符代码的中断。本章还讨论光笔和鼠标器输入。

一些中断在返回之前要等待一次击键，另一些中断则报告一次击键就绪并立即返回。如果你使用的程序重复执行一个操作直到有一个命令修改该操作，那么就使用一个不等待键盘输入的中断。例如，若你的程序正在做一循环直到有一个命令改变循环方向为止或你正在编制一个电子游戏时，可能就不想等待输入。

一些中断把输入字符回显到屏幕上，这在逐字符地读正文的程序中很有用。如果不想显示击键，象前面的两个例子一样，那么就使用一个不回显中断。不回显中断常用来读口令。回显中断不显示控制码。

如果没有把 BREAK ON 命令送给 DOS，一些键盘中断在已按了 Ctrl－Break 之后还将跳到 Ctrl－Break 处理程序。若你想在输入 Ctrl－Break 时退出程序，但不能确定 Break 开关是置 on 还是 off 时，可使用这些处理程序。

DOS 中断以标准输入设备读，一般是键盘，但也可定向。BIOS 中断从键盘读。

键入的键由一个包含一正常字节和一扩充字节的两字节返回码表示。如果键入的键是一个扩充键，例如一个功能键，那么其正常字节是 0，扩充字节描述键入的键。对所的的其它键来说，正常字节包含 ASCII 码，扩充字节包含一个位置码。位置码用来区别具有相同 ASCII 码的键，例如两个加号键。

DOS 中断对普通键只返回其 ASCII 码，扩充键需要两次调用。第一次调用返回一个 0 的键码（正常字节），第二次调用返回扩充码。0 字节表示后面是一个扩充码。

BIOS 中断立即返回两个字节码。若正常字节是 0，则扩充码描述键入的键。若正常字节非 0，扩充字节则包含位置代码。

正常码 键	码（十六进制）	码（十进制）
^A	01	1
^B	02	2
^C	03	3
^D	04	4
^E	05	5
^F	06	6
^G	07	7
^H, backspace	08	8
^I, tab	09	9
^J	0A	10

^K	0B	11
^L	0C	12
^M, enter	0D	13
^N	0E	14
^O	0F	15
^P	10	16
^Q	11	17
^R	12	18
^S	13	19
^T	14	20
^U	15	21
^V	16	22
^W	17	23
^X	18	24
^Y	19	25
^Z	1A	26
^[, escape	1B	27
^/	1C	28
^]	1D	29
^^	1E	30
^_	1F	31
(space)	20	32
!	21	33
"	22	34
#	23	35
$	24	36
%	25	37
&	26	38
'	27	39
(	28	40
)	29	41
*	2A	42
+	2B	43
,	2C	44
-	2D	45
.	2E	46
/	2F	47
0	30	48
1	31	49
2	32	50
3	33	51
4	34	52
5	35	53
6	36	54
7	37	55
8	38	56
9	39	57
:	3A	58
;	3B	59
<	3C	60
=	3D	61
>	3E	62
?	3F	63
	43	67

D	44	68
E	45	69
F	46	70
G	47	71
H	48	72
I	49	73
J	4A	74
K	4B	75
L	4C	76
M	4D	77
N	4E	78
O	4F	79
P	50	80
Q	51	81
R	52	82
S	53	83
T	54	84
U	55	85
V	56	86
W	57	87
X	58	88
Y	59	89
Z	5A	90
[	5B	91
\	5C	92
]	5D	93
^	5E	94
	5F	95
`	60	96
a	61	97
b	62	98
c	63	99
d	64	100
e	65	101
f	66	102
g	67	103
h	68	104
i	69	105
j	6A	106
k	6B	107
l	6C	108
m	6D	109
n	6E	110
o	6F	111
p	70	112
q	71	113
r	72	114
s	73	115
t	74	116
u	75	117
v	76	118
w	77	119
x	78	120
y	79	121

z	7A	122
{	7B	123
|	7C	124
}	7D	125
~	7E	126

(码 127～25 由 ALT127～255 产生)。

扩充码

键	码(十六进制)	码(十进制)
^@	03	3
shift-tab	0F	15
alt-Q	10	16
alt-W	11	17
alt-E	12	18
alt-R	13	19
alt-T	14	20
alt-Y	15	21
alt-U	16	22
alt-I	17	23
alt-O	18	24
alt-P	19	25
ctrl-enter	1C	28
alt-enter	1C	28
alt-A	1E	30
alt-S	1F	31
alt-D	20	32
alt-F	21	33
alt-G	22	34
alt-H	23	35
alt-J	24	36
alt-K	25	37
alt-L	26	38
alt-Z	2C	44
alt-X	2D	45
alt-C	2E	46
alt-V	2F	47
alt-B	30	48
alt-N	31	49
alt-M	32	50
F1	3B	59
F2	3C	60
F3	3D	61
F4	3E	62
F5	3F	63
F6	40	64
F7	41	65
F8	42	66
F9	43	67
F10	44	68
home	47	71
up arrow	48	72
PgUp	49	73

left arrow	4B	75
right arrow	4D	77
end	4F	79
down arrow	50	80
PgDn	51	81
insert	52	82
delete	53	83
shift-F1	54	84
shift-F2	55	85
shift-F3	56	86
shift-F4	57	87
shift-F5	58	88
shift-F6	59	89
shift-F7	5A	90
shift-F8	5B	91
shift-F9	5C	92
shift-F10	5D	93
ctrl-F1	5E	94
ctrl-F2	5F	95
ctrl-F3	60	96
ctrl-F4	61	97
ctrl-F5	62	98
ctrl-F6	63	99
ctrl-F7	64	100
ctrl-F8	65	101
ctrl-F9	66	102
ctrl-F10	67	103
alt-F1	68	104
alt-F2	69	105
alt-F3	6A	106
alt-F4	6B	107
alt-F5	6C	108
alt-F6	6D	109
alt-F7	6E	110
alt-F8	6F	111
alt-F9	70	112
alt-F10	71	113
ctrl-PrtSc	72	114
ctrl-left arrow	73	115
ctrl-right arrow	74	116
ctrl-end	75	117
ctrl-PgDn	76	118
ctrl-home	77	119
alt-1	78	120
alt-2	79	121
alt-3	7A	122
alt-4	7B	123
alt-5	7C	124
alt-6	7D	125
alt-7	7E	126
alt-8	7F	127
alt-9	80	128
alt-0	81	129
alt- -	82	130
alt- =	83	131

ctrl-PgUp	84	132
F11	85	133
F12	86	134
shift-F11	87	135
shift-F12	88	136
ctrl-F11	89	137
ctrl-F12	8A	138
alt-F11	8B	139
alt-F12	8C	140
ctrl-up arrow	8D	141
ctrl-pad -	8E	142
ctrl-pad 5	8F	143
ctrl-pad +	90	144
ctrl-down arrow	91	145
ctrl-insert	92	146
ctrl-delete	93	147
ctrl-tab	94	148
ctrl-pad /	95	149
ctrl-pad *	96	150
alt-sep home	97	151
alt-sep up arrow	98	152
alt-sep PgUp	99	153
alt-sep left arrow	9B	155
alt-sep right arrow	9D	157
alt-sep end	9	159
alt-sep down arrow	A0	160
alt-sep PgDn	A1	161
alt-sep insert	A2	162
alt-sep delete	A3	163
alt-pad /	A4	164
alt-tab	A5	165
alt-enter	A6	166
ctrl-pad enter	E0	224

正常码

键	码(十六进制)	码(十进制)
esc	01	1
1, !	02	2
2, @	03	3
3, #	04	4
4, $	05	5
5, %	06	6
6, ^	07	7
7, &	08	8
8, *	09	9
9, (	0A	10
0,)	0B	11
-, _	0C	12
=, +	0D	13
backspace	0E	14
tab	0F	15
Q, q	10	16
W, w	11	17
E, e	12	18
R, r	13	19

T, t	14	20
Y, y	15	21
U, u	16	22
I, i	17	23
O, o	18	24
P, p	19	25
[, {	1A	26
], }	1B	27
enter, nmpd ent.	1C	28
A, a	1E	30
S, s	1F	31
D, d	20	32
F, f	21	33
G, g	22	34
H, h	23	35
J, j	24	36
K, k	25	37
L, l	26	38
;, :	27	39
', "	28	40
`, ~	29	41
\, |	2B	43
Z, z	2C	44
X, x	2D	45
C, c	2E	46
V, v	2F	47
B, b	30	48
N, n	31	49
M, m	32	50
, <	33	51
. >	34	52
/ ? ,numpad /	35	53
numpad *	37	55
(space)	39	57

数字键盘

键	码(十六进制)	码(十进制)
7, Home	47	71
8, up arrow	48	72
9, PgUp	49	73
-	4A	74
4, left arrow	4B	75
5	4C	76
6, right arrow	4D	77
+	4E	78
1, End	4F	79
2, down arrow	50	80
3, PgDn	51	81
0, Insert	52	82
., Delete	53	83

AT BIOS的6/10/85版本，大于132的扩充码由中断16h，功能0和1返回。11/15/85版本，中断16功能0和1则不识别任何大于132的代码。11/15/85版本中，中断16的功能16和17返回大于132的扩充码和其它码。

中断16h，功能16和17，新键盘中的各个光标键(箭头键，Insert，PgDn等)在AL中返回224而不是0，而且Ctrl－Enter(正常的和数字键盘的)在AL中返回10。

操作：从键盘读字符
　　等待击键
　　立即报告正常和扩展码
　　无回显
　　不处理Ctrl－Break
版本：BIOS
中断号：16h
功能号：0
调用值：AH＝0
返回值：
　AL＝正常码
　AH＝扩展码(若是正常键为扫描码)

操作：从键盘读字符
　　不等待击键
　　立即报告正常和扩展码
　　无回显
　　不处理Ctrl－Break
版本：BIOS
中断号：16h
功能号：1
调用值：AH＝1
返回值：
　Zero标志＝0(若字符准备好)
　Zero标志＝1(若字符没准备好)
　AL＝正常码(若字符准备好)
　AH＝扩展码(若字符准备好)
　若字符准备好，则报告它的码，但并不从键盘缓冲区中删除。也就是说下一个键盘读中断将返回相同的键码。用中断16h，功能0删除该键码。

操作：从键盘读字符
　　等待击键
　　立即报告正常和扩展码

报告 132 以上的扩展码
无回显
不处理 Ctrl－Break

版本：BIOS－－11/15/85 和较新版本
中断号：16h
功能号：16
调用值：AH＝16
返回值：
AL＝正常码
AH＝扩展码(若是正常键则为扫描码)
该功能返回 132 以上的扫描码。因此，可用于检测 F11 和 F12 功能键，以及几个其它新的键组合。注意：对扩展键 AL 寄存器不总是设置成 0。

操作：从键盘读字符
不等待击键
立即报告正常和扩展码
报告 132 以上的扩展码
无回显
不处理 Ctrl－Break

版本：BIOS－－11/15/85 和近期版本
中断号：16h
功能号：17
调用值：AH＝17
返回值：
Zero 标志＝0(若字符准备好)
Zero 标志＝1(若字符没准备好)
AL＝正常码(若字符准备好)
AH＝扩展码(若字符准备好)
如果一个字符准备好，则报告它的码，但它并没有从键盘缓冲区中删除，也就是说下一个键盘读中断将返回相同的键 码。用中断 16h，功能 16 删除该键码。报告 132 以上的扩展码。注意，对所有的扩展码 AL 不等于 0。

操作：从标准输入设备读字符
等待字符
分别报告正常和扩展码
有回显
处理 Ctrl－Break－

版本：DOS 1.0－

中断号:21h
功能号:1
调用值:AH=1
返回值:
AL=键码。扩展键其键 码为 0。这时,重复该中断,AL 将包含扩展码。

操作:从标准输入设备读字符
等待字符
分别报告正常和扩展码
无回显
若是 BREAK ON 则处理 Ctrl—Break
版本:DOS 1.0—
中断号:21h
功能号:7
调用值:AH=7
返回值:
AL=键码。扩展键其键码为 0。这时,重复该中断,AL 将包含扩展码。

操作:从标准输入设备读字符
等待字符
分别返回正常和扩展码
无回显
处理 Ctrl—Break
版本:DOS 1.0—
中断号:21h
功能号:8
调用值:AH=8
返回值:
AL=键码。若是扩展键其键码为 0。这时,重复该中断,并且 AL 将不包含扩展码。

操作:清除键盘缓冲区然后读字符
版本:DOS 1.0—
中断号:21h
功能号:12
调用值:
AH=12
AL=1,7,8 或 10

清除了缓冲区，然后调用功能号在 AL 中的功能。如果距离上次读键很长时间，键盘缓冲区则被充满。该功能清除掉所有键。如果在读过程中，必需键入一些要读的键，那么使用该功能。如果在系统正在处理时，想能够处理已进入的命令，就不要使用这个功能。

返回值：AL 中的功能号

键入 Insert，Caps Lock，Num Lock 或 Scroll Lock 键会触发一个标志。用下面这个中断可检查这些标志，也可检查是不是键入了 Shift，Ctrl 或 Alt 键。

这些标志存在由该中断读的一个 BIOS 变量里，格式如下：

键盘状态标记：

位	十六进制	含　义
0	1	键入了右 Shift 键
1	2	键入了左 Shift 键
2	4	键入了 Control 键
3	8	键入了 Alt 键
4	10	Scroll Lock 键为 on
5	20	Num Lock 键为 on
6	40	Caps Lock 键为 on
7	80	Insert Lock 键为 on

操作：报告 Insert，Caps Lock，Num Lock 和 scroll Lock 的状态
　　如果键入了 shift，control 或 alt 键则报告

版本：BIOS

中断号：16h

功能号：2

调用值：AH＝2

返回值：AL＝键盘状态字节

BIOS 11/15/85 以上的版本，还可以检查其它几个键的状态。中断 16h，功能 18 返回键盘状态标志和扩充状态标志。

扩充键盘状态标志：

位	十六进制	含　义
0	1	键入左 Control 键
1	2	键入左 Alt 键
2	4	键入右 Control 键
3	8	键入右 Alt 键
4	16	键入右 Scroll Lock 键
5	32	键入右 Num Lock 键

操作:报告 Insert,Caps Lock,num Lock 和 Scroll Lock 的状态
如果键入了 shift,control,alt,Caps Lock,Num Lock 和 Scroll Lock 则报告
区分左、右 control 和 alt 键
版本:BIOS——11/15/85 和较新版本
中断号:16h
功能号:18
调用值:AH=18
返回值:
AL=键盘状态字节
AH=扩展键盘状态字节

给键盘缓冲区增加键

除了从键盘缓冲区读以外,还可以直接往键盘缓冲区增加键,这时宏扩展程序或用特定键码传送信息的程序非常有用。

操作:放一个字符到键盘缓冲区
版本:BIOS——11/15/85 和较新版本
中断号:16h
功能号:5
调用值:
AH=5
CL=要放入缓冲区的字符的字符码
CH=要放入缓冲区的字符的扩展码或位置码
返回值:AL=0(若调用成功),AL=01(若调用失败)

读字符串

有一个读字符串的中断,除了在内存驻留程序中外,其功能很强。可以用一个读字符串中断写你自己的读字符串程序。
还可以用在第 17 章讨论的文件访问程序读字符串。

操作:从标准输入设备读字符串
等待字符
回显
处理控制字符和 DOS 编辑键
只报告正常键
读字符直到一个回车为止
版本:DOS 1.0—

中断号:21h
功能号:10
调用值:
AH=10
DS:DX=输入缓冲区的指针
DS:DX[0]=缓冲区的大小
返回值:
DS:DX[1]=输入字符串的长度
DS:DX[2……]=输入字符串,以一个回车结束
该中断从输入设备读,直到有一个回车为止。键入编辑键例如backspace。在读字符串时,结果字符串存在缓冲区中。字节0设置成缓冲区大小减1。输入字符串,包括回车符,要比这个长度短。例如,读一个254字节长的字符串,分配缓冲区中的256字节并在字节0中存贮255。该中断在字节1返回字符串的长度,不包括回车。若字符串比缓冲区的长度长,DOS将发出嗡嗡声并删除回车前的字符。

光笔

光笔是一个有用的输入设备。可用下面这个中断读与图形卡直接相连的光笔。其分辨率一般不高,高级些的光笔分辨率较高。

操作:读光笔的位置和状态
中断号:10h
功能号:4
调用值:AH=4
返回值:
AH=0(若没有按下光笔开关)
AH=1(若按下光笔开关)
若AH=1,那么
DH=行(0—24)
DL=列(0—79)
CH=行(0—199)
BX=水平位置(0—319或0—639)
若AH=0,那么
BX,CX和DX被破坏

鼠标器

鼠标器是一个有用的指向输入设备,使用鼠标器中断必须安装鼠标器驱动程序软件,在使用鼠标器中断之前要检查鼠标器是否与计算机相连。

鼠标器初始化时，假设它是在屏幕的中间。鼠标器移动时，由它的驱动程序记录其位置，该位置可由一个中断读出。而且鼠标器驱动程序在鼠标器移动时可自动在屏幕上显示一个光标，还可报告鼠标器按钮的状态。

操作：检查鼠标器是否安装
　　确定按纽数
　　复位鼠标器
版本：鼠标器
中断号：33h
功能号：0
调用值：AX＝0
返回值：
　AX＝－1(若鼠标器安装)
　　＝0(若鼠标器没安装)
　BX＝按纽数
　另外，鼠标器位置被复位到屏幕中心，不显示鼠标器光标，使用缺省鼠标器光标，能够模拟光笔，使用缺省移动比率。

鼠标器可用于任何屏幕方式。屏幕位置的报告方法和使用的光标类型依赖于屏幕类型。在正文方式和中、高分辨率图形方式中屏幕位置 y 方向从 0 到 199，x 方向从 0 到 639。坐标值弥补了屏幕分辨率的不足。

屏幕方式	鼠标器坐标
0－1	x 坐标＝16×屏幕列
	y 坐标＝8×屏幕行
2－3	x 坐标＝8×屏幕列
	y 坐标＝8×屏幕行
4－5	x 坐标＝2×屏幕 x 坐标
	y 坐标＝屏幕 y 坐标
6	X 坐标＝屏幕 x 坐标
	y 坐标＝屏幕 y 坐标
7	x 坐标＝8×屏幕列
	y 坐标＝8×屏幕行
14－16，48	x 坐标＝屏幕 x 坐标
	y 坐标＝屏幕 y 坐标

操作：获取鼠标器位置
　　获取鼠标器按纽状态
版本：鼠标器

中断号：33h
功能号：3
调用值：AX＝3
返回值：

BX＝按纽状态

位	含　义
0	若左按纽按下则设置
1	若右按纽按下则设置

CX＝x 坐标
DX＝y 坐标

操作：设置鼠标器位置
版本：鼠标器
中断号：33h
功能号：4
调用值：

AX＝4
CX＝新 x 坐标
DX＝新 y 坐标

通常鼠标器可在屏幕上到处移动。下面两个中断用来设置鼠标器移动的界限。

操作：设置鼠标器 x 边界
版本：鼠标器
中断号：33h
功能号：7
调用值：

AX＝7
CX＝最小 x 边界
DX＝最大 x 边界

操作：设置鼠标器 y 边界
版本：鼠标器
中断号：33h
功能号：8
调用值：

AX＝8
CX＝最小 y 边界
DX＝最大 y 边界

鼠标器驱动程序能自动地在鼠标器位置显示一个光标。在正文方式里，这个光标或是改变属性或是移动硬件光标。发生光标由**屏幕屏蔽码**和**光标屏蔽码**定义。屏幕码保留字符某些初始属性。光标码选择改变哪一个属性码，屏幕码和光标码都是字值。屏幕码的低字节应当是ffh；光标码的低字节应当是0h。逻辑上，光标下字符位置的属性和字符字节先与屏幕码相与，然后与光标码异或。

硬件光标与中断10h设置的一样(见14章)。

在图形方式里，光标也是由屏幕和光标码组成。屏幕码与屏幕相与，光标码与屏幕异或。对图形来说，屏蔽码是位映象块。每个字包含一行的象素信息；表从第一行开始。

方式	光标大小	每个光标象素的位数
4,5	8×16	2
6,14—16,48	16×16	1

每个图形光标还有一个热点。这部分光标用来确定鼠标器所在的坐标。热点的值在—16和16之间。

操作：设置图形光标
版本：鼠标器
中断号：33h
功能号：9
调用值：
AX＝9
BX＝过热点x位置
CX＝过热点y位置
ES:DX＝屏幕和光标屏蔽码的指针。相邻放置，但屏幕屏蔽码在先。

操作：设置文本光标
版本：鼠标器
中断号：33h
功能号：10
调用值：
AX＝10
BX＝0(选择属性光标)
＝1(选择硬件光标)
CX＝屏幕屏蔽码或硬件光标扫描起始行
DX＝光标屏蔽码或硬件光标扫描结束行

鼠标器初始化时，不显示鼠标器光标。但可以触发此状态。

操作：显示鼠标器光标

版本：鼠标器
中断号：33h
功能号：1
调用值：AX＝1

操作：不显示鼠标器光标
版本：鼠标器
中断号：33h
功能号：2
调用值：AX＝2

鼠标器软件监视鼠标器移动的物理位置数。每个物理位置为 1/200 英寸。最初，二个物理位置变化将导致一个鼠标器屏幕位置变化。

操作：确定移动的实际位置值
版本：鼠标器
中断号：33h
功能号：11
调用值：AX＝11
返回值：
CX＝自上次功能 11 调用后移动的 x 位置值。正数代表右边。
DX＝自上次功能 11 调用后移动的 y 位置值。正数代表下面。

操作：设置屏幕移动的实际比率
版本：鼠标器
中断号：33h
功能号：15
调用值：
AX＝15
CX＝指明在 8 x 坐标变化范围内的实际位置值。初始设置为 8。
DX＝指明在 8 y 坐标变化范围内的实际位置值。初始设置为 16。
CX 和 DX 的高位必须是 0。最小值为 1。

鼠标器还可模拟光笔。

操作：设置光笔模拟
版本：鼠标器
中断号：33h
功能号：13
调用值：AX＝13

操作:停止光笔模拟
版本:鼠标器
中断号:33h
功能号:14
调用值:AX=14

程序设计要点:

- 键盘读中断在它们报告击键,等待击键,回显击键和处理 Ctrl—Break 的方式上各不相同。
- 键盘有两种类型:新键盘有两个新的功能键和分开的光标键以及页翻动键。配有新键盘的计算机有其特定的读新键值中断。这些中断不能处理配有旧键盘的计算机,所以使用时要加以小心。
- DOS 中断从标准的输入设备读,BIOS 中断从键盘读。
- 有许多与鼠标器有关的中断。包括决定鼠标器是否与计算机相连的中断;控制鼠标器光标的中断,还有控制和检查鼠标器位置及调节鼠标器移动灵敏度的中断。

第十六章　PSP 和参数传送

程序段前缀(PSP)是 DOS 放在每个程序开始之前的一个 256 字节的块。COM 文件的 PSP 位于 CS:0。写一个 COM 文件的汇编代码时，总 是以 org 100h 语句开始以给 PSP 留下空间。在 EXE 文件里，汇编代码从 CS:0 开始，DS 和 ES 初始时指向 PSP。

PSP 包含程序操作必需的信息，还包含命令行参数和 DOS 的工作空间，DOS 用它来存储有关该程序使用磁盘情况的信息。命令行参数是在一个程序名后输入的参数。例如，如果你输入 **format b:/v**，**b:**和**/v** 就是命令行参数。

DOS 的早期版本通过**文件控制块**(FCB)访问文件。从 DOS 2.0 开始，DOS 使用一种更简单的利用**文件手柄**的方法。但所有的旧 FCB 中断都保留下来，以保证向上兼容性，PSP 里的更多空间留给了 FCB 型的文件访问。

下面是 PSP 的格式：

字节	内　容

0—1	INT 20h 指令(程序结果)。该指令有两个用途。一个汇编语言程序可用一个中断来结束，或者也可以
	`push ds`
	`xor ax,ax`
	`push ax`
	开始，以
	`retf`
	结束。
	程序开始时，ds 指向 PSP。远返回把 ds:0 读出栈。这条指令指向 PSP 的开始，程序结束指令也放在这。这个单元也可用作有非分辨外部单元程序的安全特征。连接程序把其设置为 0；因此，转移到该处程序就结束了。

2—3	可用内存尾的段地址。
5—9	DOS 功能调度程序的长调用(INT 21h)。单元 6 里的字包含代码段的有效字节数。

10—13	终止程序的例行程序地址的拷贝，当一个程序结束时，该地址指向被执行的例行程序，通常，结束例行程序是 COMMAND.COM 的一部分，但一个程序可改变所用的例行程序。PSP 里的指针是在程序开始前建立的结束地址的拷贝；程序结束时，从 PSP 拷贝中恢复结束单元里的地址，跳过未恢复的结束地址。在一个程序中 EXEC 另外一个程序时，要使用这个方法。被 EXEC 的程序的结束地址置为返回到 EXEC 中断后面的那个单元，把 PSP 拷贝设置成调用程序的结束地址。被 EXEC 的程序结束时，恢复调用程序的结束地址。改变这个单元就改变了调用程序的结束地址。
14—17	在程序执行之前 Ctrl—Break 退出地址的拷贝。Ctrl—Break 的实际地址放在执行程序的调用之后的单元。程序结束时，恢复原始的 Ctrl—Break 地址。
18—21	重要错误退出地址的拷贝。程序结束时，恢复实际重要错误退出地址。
22—43	DOS 工作区。
44—45	环境段。环境是一组包含环境设置情况的 ASCIIZ 字符串，象 PROMPT 和 PATH 信息，它在这个段的开始处，偏移量为 0。信息可通过该环境传送给程序。例如，可在环境里设置一个变量指明存贮临时文件的驱动器。
46—79	DOS 工作区

80—82	`int 21h`
	`retf`
	的汇编代码。

85—127	文件控制块区
128	命令行参数的字节长度
129—255	命令行参数
128—255	缺省磁盘传送区域

在命令行输入参数后，参数被拷贝到 PSP 的单元 129，除去了 I/O 重定向信息。单元 128 存贮在命令行输入的字符数，第一个字符为空格。

读参数要从单元 129 开始扫描存贮的数据。注意要去掉开头和中间的空白。要检查所有已输入的参数。若输入的参数不够，或建立缺省或终止。检查长度字节看看命令行何时结束。回车出表明命令行结束。在 MOVE.ASM 和 DIR2.ASM 里有读命令行参数的例子。

单元 128—255 是缺省磁盘传送区(DTA)，磁盘传送区时由一些处理磁盘文件的中断，象目录搜索中断使用的内存部分。过去的 FCB 型文件访问方法把 DTA 作为存贮磁盘信息的缓冲区。如果你的程序用到使用 DTA 的中断，在使用这些中断之前先处理命令行参数。还可把参数拷到另一区或重定位 DTA。

PSP 的几个部分用作 DOS 工作区。DOS 用以记录其正在做什么的数据存在里面。内存驻留程序用 DOS 中断时，必须复位 PSP(象第三部分讨论的一样)。

取 PSP 中断时读 EXE 文件的 PSP 信息很有用。

操作：获得 PSP 的地址
版本：功能 81：无文献记载。DOS 2.0—
　　功能 98：DOS 3.0—
中断号：21h
功能号：81 或 88
调用值：
　AH＝81 或 88
　这两个功能做相同的事情
返回值：BX＝PSP 的段

操作：设置 PSP 的地址
版本：无文献记载。DOS 2.0
中断号：21h
功能号：80
调用值：
　AH＝80
　BX＝要改变的 PSP 的段
　注：该功能最初用于内存驻留程序。在改变前，一定要保存旧 PSP 的地址。

操作：获得 DTA 的地址
版本：DOS 2.0—
中断号：21h
功能号：47
调用值：AH＝47
返回值：ES：BX＝DTA 的指针

操作：设置 DTA 的地址
版本：DOS 1.0—
中断号：21h
功能号：26
调用值：
　AH＝26
　DS：DX＝DTA 新单元的指针。不要距离一个段边界太近，以免引起返转。

程序设计要点：

• PSP 在所有程序之前，包含 DOS 变量和命令行参数。

第十七章　磁盘文件

用汇编语言访问文件很简单。许多操作，象删除或重命名文件，只需把文件名送给 DOS，调用一个中断，就完成了。本章讨论如何操作磁盘文件——从打开、关闭一个文件到解释错误码；移动、重命名和删除文件；改变文件属性；控制输入/输出。

读文件或是写文件时要从打开文件开始。把文件名送给 DOS，DOS 返回一个文件手柄——标识该文件的唯一一个两字节数。然后调用读、写或定位中断，用它的手柄指定文件。操作结束后，关闭文件。关闭过程清除该文件的所有内部缓冲区，全面地更新目录，并释放文件手柄。

一次可有几个活动手柄。还可定义和打开下面的手柄。

手柄	设　备
0	标准输入设备(正常为键盘)
1	标准输出设备(正常为屏幕)
2	标准错误设备(屏幕)
3	标准辅助设备
4	标准打印机(LPT1 和 PRN)

在需要这些手柄的地方，可使用任何带有这些手柄的中断。例如，利用手柄 1，用写中断把一正文块送到屏幕。如果是用手柄 4，则把正文送到打印机。同样，用手柄 0 从键盘读字符串。注意，I/O 重定向会改变标准的输入、输出设备，这样打开文件时，手柄从 5 开始编号。

把文件名送给 DOS 时，同时也把一个指向一 ASCIIZ 字符串的指针送给了 DOS。这个字符串为驱动器，路径，文件名，扩展名，都是 ASCII 字符，后面是十六进制的 0。例如，可用下面的方法建立文件名：

```
failsafe        db        'c:\utilities\failsafe.com', 0
```

字符的大小写无关紧要，若省略了驱动器或路径，那么就使用缺省值。

读文件或写文件时，要指定一个用来装或读数据的缓冲区。这个单元可在任何地方。如果可能，直接往将用到的缓冲区装数据。一次最多能读或写 64K。如果一次要读大量数据，那么用内存分配调用(见第 20 章)建立缓冲区，而不用在程序里设置空间；这样可大大减少程序使用的磁盘空间。该缓冲区由　段:偏移量　定义。如果要读、写的数据大于段中保留的空间，那么读、写操作就越过此段到下一个内存单元，而不绕回到段的开始处。例如，假设你的缓冲区从 1000:FFF0h 开始，你要从磁盘读 100 个字节。前 16 个字节将存在 1000:FFF0h－1000:FFFFh，后 84(54h)个字节将存在 1100:0000h－1100:0054h. 如果数据绕回，那么前 16 个字节存在 1000:FFF0h－1000:FFFFh，后 84 个字节存在 1000:0000h－1000:0054h。

通过设置进位标志，文件访问中断发出出现错误的信号。错误代码返回到 AX 中。对 DOS 2.1，直接用功能 38h－57h 检查错误码，靠进位判断是不可靠的。

错误代码	含　义
1	无效功能号
2	未找到文件
3	未找到路径
4	没有可用的手柄(文件打开太多)
5	拒绝访问
6	无效手柄
7	内存控制块遭到破坏
8	内存不够
9	无效的内存块地址
10	无效环境
11	无效格式
12	无效访问代码
13	无效数据
14	未用
15	无效驱动器
16	不能移动当前目录
17	不相同的设备
18	未找到更多的文件

DOS 3.0 包含更高级的错误信号。DOS 调用出错之后，设置进位标志，错误代码返回到 AX。所以也可以只检查进位，看看错误代码。更多的错误信息可用中断 21h，功能 89 检查，其错误代码可能与 AX 中的不一样。

错误代码	含　义
1	无效功能号
2	未找到文件
3	未找到路径
4	无可用的手柄
5	拒绝访问
6	无效手柄
7	内存控制块遭到破坏
8	无足够的内存
9	无效内存块地址
10	无效环境
11	无效格式
12	无效访问代码
13	无效数据
14	保留

15	无效驱动器
16	不能移动当前目录
17	不相同的驱动器
18	未找到更多的文件
19	有写保护
20	未知单元
21	驱动器未就绪
22	无效命令
23	CRC 错误
24	错误请求
25	查找错误
26	未知介质类型
27	未找到扇区
28	打印纸用尽
29	写错
30	读错
31	一般错误
32	共享违约
33	擅锁
34	无效磁盘更改
35	不可用的文件控制块
36	共享缓冲区溢出
37—49	保留
50	不支持网络请求
51	远程计算机没在倾听
52	网络上名字重复
53	未找到网络名字
54	网络忙
55	网络设备不存在
56	超过 BIOS 命令限制
57	网络适配器硬件错误
58	不正确的网络响应
59	不可预见的网络错误
60	不兼容的远程适配器
61	打印队列满
62	给打印文件用的空间不够
63	删除的打印文件
64	删除的网络名字
65	拒绝访问
66	不正确的网络设备类型

67	未找到的网络名字
68	网络名字太多
69	越过 BIOS 会话限制
70	暂停
71	不接收网络请求
72	暂停重定向
73—79	保留
80	文件存在
81	保留
84	重定向太多
85	重定向重复
86	无效口令
87	无效参数
88	网络设备故障

错误类型	含 义
1	资源用完
2	临时问题
3	权限问题
4	内部软件错误
5	硬件失败
6	系统软件错误
7	应用软件错误
8	未找到项
9	无效格式
10	项锁住
11	介质失败或错误磁盘
12	项存在
13	未知

建议动作	含 义
1	重试
2	过一会重试
3	要求用户重新输入请求
4	放弃并清除
5	立即放弃，不清除
6	忽略
7	要求用户执行补救性动作，然后重试

原因	含 义
1	未知或非专用
2	块设备
3	网络设备
4	串行设备
5	内存

操作:获取错误信息
版本:DOS 3.0
中断号:21h
功能号:89
调用值:

AH=89
BX=0

返回值:

AX=错误码
BL=建议动作
BH=错误种类
CH=原因
CL,DX,DI,SI,DS,ES 被破坏
注:DS 和 ES 被破坏

打开文件

有五个 DOS 功能用来打开文件。**功能 60** 打开一个文件并把其长度设为 0。如果此文件不存在,则生成一个新文件。可以用这个功能生成一个全新的文件或在一个旧文件中存储全新的信息。

功能 61 打开一个读或写的文件。文件必须是已存在的,旧数据保留不动,要改变旧数据,则须在旧数据上重写。可用这个功能读文件或修改部分旧文件。

功能 91,DOS 3.0 以上版本才有,与功能 60 相似,不同之处是如果文件已存在,功能 91 将返回一个错误信息。如果你不想冒险重写一个已存在的文件,也就是如果你只想生成一个目前不存在的文件,可以使用这个功能。

功能 90 生成一个新文件并且选择文件名。同功能 91 一样,它也是 DOS 3.0 以上版本才有。选择名字是为了不与其要进入的目录中的其它文件同名。可用这个功能生成一个包含临时信息的文件,不用担心重写现有文件。如果打算在程序中删除这个文件,一定要保留 DOS 选择的名字。

功能 108,DOS 4.0 以上版本才有。它合并了其它打开文件功能,提供数据的自动记录。如果你的程序将在 DOS 4.0 以上版本运行,这个功能很有用。

没有必要打开手柄 0—4。这些中断都不生成子目录。

操作:打开 一个文件

若文件不存在则建立一个新文件

长度设置成 0(清除旧文件)

版本:2.0—

中断号:21h

功能号:60

调用值:

AH=60

CH=0

CL=文件属性。与目录属性字节使用相同格式。不设置子目录或卷位。

位	含义
0	只读
1	隐含
2	系统

正常文件 CL 设置成 0

DS:DX=ASCIIZ 文件名的指针

返回值:

AX=文件手柄或错误码

若有错进位位置 1。错误码为 3(路径没找到),4(没有空手柄),5(拒绝访问)。

操作:打开一个文件

文件必须是存在的

长度不能改变(除非由追加数据加长)

版本:DOS 2.0

DOS 3.0 用存取码位 3—7

中断号:21h

功能号:61

调用值:

AH=61

AL=存取码。该码决定怎样使用文件。对于 DOS 2.0 或不联网或无多任务的系统只需设置位 0—2。

位	含义
0—2	存取方式
	000 只读
	001 只写
	010 读和写访问
3	保留
	总设置成 0

4—6　共享方式。如果想多次打开文件则设置这些位。(也就是,由多个并行操作程序设置)。这用来保证数据的完整性。

000　兼容方式。其它进程没有存取。可由当前进程多次打开。

001　拒绝读/写。其它进程没有存取。不能由当前进程多次打开。

010　拒绝写.想打开文件进行写操作的进程没有存取。

011　拒绝读.想打开文件进行读操作的进程没有存取。

100　允许全部存取。所有其它进程都能存取。

7　继承模式

0　如果子进程自动继承对文件的存取.所有共享和存取约束也被继承。

1　如果子进程不自动继承对文件的存取。

注:安排这些设置是为了使用 DOS 2.0 写的程序(只设置低三位)可有缺省的最一般的存取。

DS:DX=ACSIIZ 文件名的指针

返回值:

AX=文件手柄或错误码

若有错误进位位设为 1。错误码为 2(文件没找到),4(没有空 4 手柄),5(访问被拒绝),12(无效访问码)。

操作:建立一个具有唯一名字的文件

版本:DOS 3.0—

操作号:21h

功能号:90

调用值:

AH=90

CH=0

CL=文件属性。与目录中的格式一样。

DS:DX=包含新文件路径名的 ASCII 字符串的指针.若字符串为空,新文件将放在当前目录中。若路径非空,用后跟 13 个 0 字节的 a \结束字符串,DOS 在\后将填入一个 12 字节的文件名。若你想从程序中删除该文件,一定要使文件名区不被破坏,因为删除文件中断要用到它。

返回值:

AX=文件手柄或错误码

DS:DX=ASCIIZ 文件名的指针

若有错进位位置 1。

操作:打开一个文件
　　建立一个新文件
　　若文件已存在则返回错误码
版本:DOS 3.0—
中断号:21h
功能号:91
调用值:
　AH=91
　CH=0
　CL=文件属性。与目录属性字节格式一样。
　DS:DX=ASCIIZ 字符串的指针
返回值:
　AX=文件手柄或错误码
　若有错进位位置 1

操作:打开一个文件
　　返回文件现有设置表
　　可设置自动记录标志
　　文件必须已存在
版本:DOS 4.0—
中断号:21h
功能号:108
调用值:
　AH=108
　AL=0
　BX=访问码

位	含义
0—2	访问码
	000　只读
	001　只写
	010　读和写访问
3	保留
	总设置成 0

4—6　共享方式。若想多次打开文件则设置这些位。(也就是,由多个并行操作程序设置)。用于保证数据的完整性。

000　兼容方式。其它进程没有访问,可由当前进程多次打开。

001　拒绝读/写。其它进程没有访问,不能由当前进程多次打开。

010　拒绝写。想打开文件进行写操作的进程没有访问。

011　拒绝读。想打开文件进行读操作的进程没有访问。

100　全部允许访问。所有其它进程都可访问。

7　继承方式

0　若子进程自动继承对文件的访问。所有共享和访问约束都是可继承的。

1　若子进程不自动继承对文件的访问。

8—12　设置成 0

13　设置错误级别。若为 0,若有致命错误,使用正常的致命错误处理程序(int 24h)。若为 1,对与该文件有关的 I/O 停用 int 24h,只返回扩展错误码。

14　记录码。若为 0,该文件是正常文件。若为 1,无论何时往该文件写数据,数据都被自动记录(不往磁盘写)。自动记录保证如果挂起或掉电有 RAM 缓冲区中的数据不会丢失。

CX=建立属性。与目录表项中的格式一样。

DX=文件现有控制

位　含义

0—3　若文件已存在要执行的动作

0000　返回一个错误码

0001　打开文件

0010　打开文件并设置长度为 0

4—7　若文件不存在要执行的动作

0000　返回一个错误码

0001　生成文件

8—15　设置成 0

DS:DI=ASCIIZ 文件名的指针

返回值:

若有错进位位置 1。否则,AX 包含文件手柄,CX 指明已采取的动作:1=文件是打开的,2=文件是生成的(以前不存在),3=文件是打开的,长度为 0。

读、写以及定位

读、写以及定位都需要打开文件手柄。文件打开后，**文件指针置 0**。文件指针指明下一步将写或读文件的哪一部分。每次读或写文件，文件指针都移到刚刚访问部分的末尾。若是顺序写文件，就重复使用写中断。不必记录或修改文件位置。若是随机读或写文件，或向一个文件追加数据，那么就使用定位中断改变文件指针。

读和写中断返回被读/写的字节数。如果返回的字节数不符，那么或是遇到了文件结束标志，或是磁盘已满，或是从标准输入设备读时到了行尾(回车)。

可用读中断从键盘读字符串。把 CX 设置为你所想读的字符串的最大长度。读中断读键盘或标准输入设备，直到键入一个回车或文件结束(Ctrl－Z)或读够字符串长度为止。检查 AX 以确定字符串的实际长度。

操作：读文件
　　文件指针移到被读区域的结尾
版本：DOS 2.0－
中断号：21h
功能号：63
调用值：
　AH＝63
　BX＝文件手柄
　CX＝要读的字节数
　DS：DX＝缓冲区位置
返回值：
　AX＝实际读的字节数或错误码
　若有错进位位置 1。错误码为 5(拒绝访问)，为 6(无效手柄)。

操作：写文件
　　文件指针移到被写区域的结尾
版本：DOS 2.0－
中断号：21h
功能号：64
调用值：
　AH＝64
　BX＝文件手柄
　CX＝要写的字节数
　DS：DX＝缓冲区位置
返回值：
　AX＝实际写的字节数或错误码
　若有错进位位置 1。错误码为 5(拒绝访问)，为 6(无效手柄)。

操作：定位文件指针
版本：DOS 2.0—
中断号：21h
功能号：66
调用值：

AH＝66

AL＝移动方法

0　从文件开始移动的偏移字节数

1　从当前文件指针位置移动的偏移字节数

2　在文件结尾之后移动的偏移字节数

可用方法 2，设偏移量为 0 不确定文件的大小。文件大小返回在 DS：AX 中。若指针在文件结尾之后移动并写数据，那么新写的数据将在文件指针处开始。在文件结尾和文件指针之间的磁盘上的数据将成为该文件我一部分。可以清除这部分数据。

BX＝要移动的偏移字节数。CX 是高字。

返回值：

DX：AX＝指针的新偏移量，或 AX＝错误码。

若错进位位置 1。错误码为 1(无效的移动方法)，6(无效手柄)。

记录文件数据

写文件时，信息存在 DOS 的磁盘缓冲区里。这些缓冲区在内存里。缓冲区写满之后，就写往磁盘。大多数时间里这种情况是正常的。但是如果有掉电或系统性事故发生，存在缓冲区里的数据就会丢失。而且，如果你正在网络上工作，缓冲区里的数据在存到磁盘之前别的机器是看不到的。为保证大多数的最新数据写到磁盘上，有三种办法。第一，关闭文件，再重新打开，定位到离开时的位置。关闭一个文件就清除了所有缓冲区。第二，对该文件生成一个重复的文件手柄，然后关闭重复的文件手柄。这种方法不要求重新打开文件及重新定位手柄。第三，可以记录文件数据(DOS 3.3 以上版本)，这样保证了所有缓冲区里的数据都存到磁盘上。

操作：生成一个重复的文件手柄
版本：DOS 2.0—
中断号：21h
功能号：69
调用值：

AH＝69

BX＝文件手柄

返回值：

AX＝新文件手柄或错误码。若有错进位位置 1。新文件手柄指的是送入 BX 中的那个相同文件，但是号不同。两个手柄的文件指针总指向相同的位置。如果移动一个文件手柄的文件指针，另一个文件手柄的文件指针也移动。

操作：送 DOS 文件缓冲区到磁盘上
版本：DOS 3.3—
中断号：21h
功能号：104
调用值：
AH＝104
BX＝文件手柄
返回值：AX＝错误码(若进位位为 1)

关闭文件

完成文件处理后，应当关闭文件。用中断 21h，功能 62 完成此任务：

操作：关闭文件
版本：DOS 2.0—
中断号：21h
功能号：62
调用值：
AH＝62
BX＝文件手柄
返回值：
AX＝错误码。若有错进位位置 1。错误码为 6(无效手柄)。

传送和重命名文件

重命名一个文件，要把其原始名字和新名字都传送给该文件。如果原始名字以一个驱动器名字开始，新名字也一定要以一个驱动器名字开始，而且驱动器名字一定要相同。新文件名还可指定文件的新路径，也就是，重命名功能可以把一个文件从一个子目录传送到另一个子目录。子目录也可重命名，但不能传送。

一次只能重命名一个文件。文件名中不能使用通配符＊和?。

在第 18 章，将用下面这个中断生成一个重命名文件并在目录间传送文件的实用程序——MOVE。

操作：移动和重命名文件
版本：DOS 2.0—

中断号:21h
功能号:86
调用值:
AH=86
DS:DX=ASCIIZ 原始名的指针
ES:DI=ASCIIZ 新名的指针
返回值:
AX=错误码
通过把第一个字符设置成 e5h,属性字节设置为 0 清除了原始目录表项。若有错进位位置 1。错误码为 3(路径没找到),5(拒绝访问),17(不相同的驱动器)。

删除文件

删除文件要用文件名调用下列中断。只读文件、子目录文件或卷名不能删除。删除只读文件首先要把只读文件的属性改为正常文件属性(见下章的中断讨论)。一次只能删除一个文件。文件名中不能使用通配符*或?。

操作:删除文件
不删除只读文件
版本:DOS 2.0—
中断号:21h
功能号:65
调用值:
AH=65
DS:DX=ASCIIZ 名字的指针
返回值:
AX=错误码
若有错进位位置 1。错误码为 2(文件没找到),5(拒绝访问)。

改变文件的属性、日期和时间

属性改变中断可使你灵活地操作文件的属性。为保护文件,该中断不改变子目录和卷位。改变子目录和卷位,要使用一个象 EXPLORER 一样的实用程序。

如果打开文件进行写操作,那么关闭手柄中断将自动更新文件日期、时间和归档位。

操作:获取或改变文件属性
不改变子目录或卷位
版本:DOS 2.0—
中断号:21h
功能号:67

调用值：

AH＝67

AL＝操作

值	含义
0	读属性
1	设置属性

CL＝属性(若设置属性)。属性码与存在目录中的一样。子目录和卷位应当为 0

DS:DX＝ASCIIZ 文件名的指针

返回值：

AX＝错误码

CL＝属性(若读属性)。属性码与在目录中的一样。

若有错进位位置 1。错误码 2(文件没找到),3(路径没找到),5(拒绝访问)。

操作：获取或设置文件日期和时间

文件必须是打开的

文件关闭时可改动

版本：DOS 2.0—

中断号：21h

功能号：87

调用值：

AH＝87

AL＝命令码

值	含义
0	读日期和时间
1	设置日期和时间

BX＝文件手柄

CX＝时间(若设置日期和时间)。格式与在目录中的一样。设置成：(小时 SHL 11)＋(分 SHL 5)＋(秒 SHR 1)。

DX＝日期(若设置时间和日期)。格式和目录中的一样。设置成：((年－1980) SHL 9)＋(月 SHL 5)＋日。

返回值：

AX＝错误码

CX＝时间(若读日期和时间)。格式和用在目录中的一样。

DX＝日期(若读日期和时间)。格式和用在目录中的一样。

若有错进位位置 1。错误码为 1(AL 中命令无效),6(无效手柄)。

设备的输入/输出控制

有一专门处理设备输入/输出控制的中断(IOCTL)。该中断用以与设备驱动程序接口并能处理大多数设备 I/O,访问逻辑设备(如驱动器 A 或 C)和处理手柄(如文件),把控制数据和实际信息送给设备。每个 DOS 版本都有 IOCTL 中断。

IOCTL 的功能是查询或访问设备,包含基本的查询和访问功能。更高级的查询功能将在第 24 章讨论。

基本查询功能以下列格式返回信息。下面一个表是设备的,一个表是文件的。

文件的询问结果

位 7=0

位	含 义
0—3	设备号的低 4 位(0=A,1=B,等等)
5	设备号的高位
6	写文件则清 0

设备的询问结果

位 7=1

位	含 义
0	若是控制台输入设备则设置
1	若是控制台输出设备则设置
2	若是空设备则设置
3	若是时钟设备则设置
4	保留
5	若是二进制方式则设置
6	若是输入的文件尾则设置
8—13	保留
14	若能处理控制字符串则设置
15	保留

若是二进制方式,数据不用修改就送给文件;若是 ASCII 方式,Control—Z 表示文件结束,输入时把回车/换行一起转换成输入换行,输出时把换行转换成回车/换行。

操作:获取设备信息
版本:DOS 2.0—
中断号:21h
功能号:68
调用值:
AH=68
AL=0

BX＝手柄

返回值:DX＝设备信息

操作:设置设备信息

版本:DOS 2.0—

中断号:21h

功能号:68

调用值:

AH＝68

AL＝1

BX＝手柄

DH＝0

DL＝询问结果的低字节要设置的设备信息

注意不能改变高字节

操作:获取输入状态

版本:DOS 2.0—

中断号:21h

功能号:68

调用值:

AH＝68

AL＝6

BX＝手柄

返回值:

AL＝对文件:

＝0(在到达的文件结尾之后)

＝ffh(在到达的文件结尾之前)

对设备:

＝0　未准备好

＝ffh　准备好

操作:获取输出状态

版本:DOS 2.0—

中断号:21h

功能号:68

调用值:

AH＝68

AL＝7

BX＝手柄

返回值:

AL＝对文件：
＝0(在到达的文件结尾之后)
＝ffh(在到达的文件结尾之前)
对设备：
＝0　未准备好
＝ffh　准备好

设备有两种类型——**字符设备**和**块设备**。字符设备是象屏幕和磁盘文件那样逐字符接受信息的设备，块设备是象磁盘驱动器那样以块为单位接受信息的设备。

操作：从字符设备读
版本：DOS 2.0—
中断号：21h
功能号：68
调用值：
AH＝68
AL＝2
BX＝手柄
CX＝要读的字节数
DS：DX＝要存贮数据的缓冲区的指针
返回值：AX＝读的字节数

操作：往字符设备写
版本：DOS 2.0—
中断号：21h
功能号：68
调用值：
AH＝68
AL＝3
BX＝手柄
CX＝要写的字节数
DS：DX＝要写的数据的指针
返回值：AX＝写的字节数

操作：从块设备读
版本：DOS 2.0—
中断号：21h
功能号：68
调用值：
AH＝68

AL＝4

BL＝驱动器号，0＝缺省，1＝A 等

CX＝要读的字节数

DS：DX＝要存贮数据的缓冲区的指针

返回值：AX＝读的字节数

操作：往块设备写

版本：DOS 2.0—

中断号：21h

调用值：

AH＝68

AL＝5

BL＝驱动器，0＝缺省，1＝A 等。

CX＝要写的字节数

DS：DX＝要写的数据的指针

返回值：AX＝写的字节数

操作：从一个设备读一个磁道

版本：DOS 3.2—

中断号：21h

功能号：68

调用值：

AH＝68

AL＝13

BL＝驱动器号，0＝缺省，1＝A 等

CL＝97

CH＝8

DS：DX＝一个命令块的指针，在此：

字节	含义
0	设为 0
1—2	面
3—4	磁道
5—6	要读的第一个扇区
7—8	要读的扇区数
9—10	放读数据的缓冲区的偏移量
11—12	放读数据的缓冲区的段

操作：往设备写一个磁道

版本：DOS 3.2—

中断号：21h

功能号：68

调用值：

AH＝68

AL＝13

BL＝驱动器号，0＝缺省，1＝A 等

CL＝65

CH＝8

DS：DX＝命令块的指针，在此：

字节	含义
0	设为 0
1—2	面
3—4	磁道
5—6	要写的第一个扇区
7—8	要写的扇区数
9—10	要写的数据的偏移量
11—12	要写的数据的段

程序设计要点：

- 在执行中断功能时，若出错，则 DOS 3.0 以上版本返回详细的信息。
- 使用文件之前要打开。文件打开后，将得到一个唯一的手柄号。
- 标准的输入/输出设备总是打开的。
- 文件一般是顺序访问的。随机访问文件要移动文件指针。
- 文件可在目录间移动。
- IOCTL 功能提供对 I/O 设备的完全控制。

第十八章　终止和一个程序实例

程序执行完成之后，应当把计算机恢复到程序开始时它的状态，当然要除了程序的目的就是改变计算机的状态这种情况。例如，如果切换到了图形屏幕，那么还要再恢复到正文屏幕。一个更明显的例子是，停用键盘而用扬声器启动音调，就更应该恢复使用键盘而关闭扬声器。不必恢复寄存器的起始状态，也不必清栈。本章将编写一个 MOVE 程序，该程序可以通过重命名文件，在子目录间传送文件，甚至传送整个子目录来清理磁盘。

完成恢复工作之后，调用一个终止中断。终止中断能从 PSP 中恢复某些中断值并能清除文件缓冲区。终止中断还返回一个**返回代码**，这个返回代码是一个用来标志父程序的号。返回代码可由一个有 ERRORLEVEL 命令的批处理文件来检查或从一个有中断 21h，功能 77 的父程序检查出来，本章后面将讨论这个问题。

约定情况是，若程序运行成功则返回一个 0，若有错则返回一个不同的数。返回代码还可用来指明程序已经做了些什么工作。

终止功能有两个。功能 49 用于内存驻留程序(见第三部分)。

操作：终止一个程序并驻留
　　程序驻留在内存中
　　分配的内存不释放
中断号：21h
功能号：49
调用值：
　　AH＝49
　　AL＝返回码
　　DX＝要驻留的节数
返回值：终止。不返回到使用该中断的程序。

操作：终止一个程序
中断号：21h
功能号：76
调用值：
　　AH＝76
　　AL＝返回码
返回值：终止。不返回到使用该中断的程序。

一个程序实例：MOVE

MOVE 主要用中断 21h，功能 86——传送和重命名中断。MOVE 的功能是重命名文件，在

子目录间传送文件。有了 MOVE 不用拷贝和删除文件就可以清理磁盘，清理磁盘是个很长的过程，并且导致磁盘碎片，在满盘上不能做清理。

还可用 MOVE 传送子目录——这是重新组织磁盘特别是硬盘的一个重要特征。首先用 EXPLORER 触发你想传送的子目录的目录属性。把这个变动存起来。接着，用 MOVE 把子目录送到新单元。最后，用 EXPLORER 把它们重新标识为子目录。

通过输入

```
move old_name new_name
```

使用 MOVE。

new—name 的路径可以和 old—name 的不同，但如果 old—name 以一个驱动器名开始，new—name 也必须以相同的驱动器名开始。例如：

```
move b:\utilities\failsafe.com \f.com
```

是无效的，下列形式才有效

```
move b:\utilities\failsafe.com b:\f.com
```

MOVE 首先检查 PSP 的命令行参数——新、旧文件名。若没找到足够的参数，就打印一个出错信息并结束。若找到足够的参数，就把参数变成 ASCIIZ 字符串，设置传送和重命名文件功能的寄存器，执行中断。然后打印一个信息。MOVE 以一个返回代码结束。若 MOVE 运行成功返回代码为 0，反之为 1。

MOVE 是一个 COM 文件。因此为了给 PSP 留出空间，程序以 **org 100h** 开始。因为它只使用代码段，所以没有数据段。注意变量是放在代码的前面，变量前面是一个跳到程序主体的 jump 指令。这样变量就很容易找到，而且由于汇编程序不需猜变量段的大小，也使汇编过程效率更高。程序从建立段寄存器开始(这倒不必一定要这样)。COM 文件执行时，四个段寄存器都指向 PSP，而 EXE 文件执行时，DS 和 ES 指向 PSP，CS、IP、SS 和 SP 都置为连接程序给出的值。因为程序中段的建立很明确，所以程序操作也更清楚。

注意，每个子程序都只有一个 **ret** 指令，主程序只能有一个结束点，这样使得程序更容易跟踪和调试。

回忆一下第 14 章的 disp—msg 例行程序。注意正文信息如何包括回车和换行，如何以结束符 0 结束。find—next—space 和 find—next—char 用来读参数。

下面是程序：

```
;MOVE.ASM
;
;This program moves files from one directory to another, renaming
;them in the process.
;
;The format is:   MOVE  old_name new_name
;
;If a drive name is specified in old_name, new_name must have the
;same drive_name.
;
;This is set up to be a .COM file.
;
```

```
;-------------CONSTANTS--------------

PARAM_START     equ     81h
PARAM_LENGTH    equ     80h
SPACE           equ     20h
RETURN          equ     0dh
FORE_COLOR      equ     3


;-------------BEGINNING--------------

code            segment
                assume  cs:code,ds:code
                org     100h            ;make this a .COM file
move:           jmp     main_code


;-------------DATA-------------------
;
;Note: data is placed here to make assembling more efficient
;

old_name        dw      ?
new_name        dw      ?
old_attr        dw      ?
improper_name_msg db    'Improper file name', 0ah, 0dh
                db      '0 files moved', 0ah, 0dh, 0
missing_msg     db      'Requires source and test file names', 0ah, 0dh
                db      '0 files moved', 0ah, 0dh, 0
ok_msg          db      '1 file moved', 0ah, 0dh, 0



;-------------MAIN CODE--------------

main_code:      mov     ax,cs     ;set up segment registers
                mov     ds,ax
                mov     es,ax
;................................
;
; Get the parameters
;
;................................
                mov     cl,ds:[PARAM_LENGTH]
                mov     ch,0
                mov     bx,PARAM_START
                jcxz    missing_filename    ;no params
                call    find_next_char      ;find old_name
                jcxz    missing_filename    ;no params
                mov     old_name,bx
                call    find_next_space     ;find end, to make ASCIIZ
                jcxz    missing_filename    ;no new_name
                mov     [bx],byte ptr 0     ;make ASCIIZ
                call    find_next_char
                jcxz    missing_filename    ;no new_name
                mov     new_name,bx
                call    find_next_space
                mov     [bx],byte ptr 0
;................................
;
; Do the move
;
;................................
move_it:        mov     dx,old_name         ;perform DOS interrupt
                mov     di,new_name
                mov     ah,56h
                int     21h
                jnc     move_ok             ;check for error
```

```
;..............................
;
; Display messages and return with error code
;
;..............................
improper_names:  lea     dx,improper_name_msg ;display message
                 mov     al,1
                 jmp     move_end
missing_filename: lea    dx,missing_msg
                 mov     al,1
                 jmp     move_end
move_ok:         lea     dx,ok_msg           ;print message
                 mov     al,0
move_end:        call    disp_msg
                 mov     ah,4ch              ;terminate
                 int     21h



;-------------SUBROUTINES------------

;find_next_space
;   returns offset of next space or return or 0
;      starting offset in BX
;      count in CX
;      result returned in BX
;      CX decremented by characters examined
;
find_next_space  proc    near
                 push    ax
f_n_s_loop:      mov     al,[bx]             ;load character
                 cmp     al,SPACE            ;is it a space?
                 je      found_space
                 cmp     al,RETURN           ;is it a return?
                 je      found_space
                 cmp     al,0                ;is it a 0?
                 je      found_space
                 je      found_space
                 inc     bx                  ;keep on looking
                 loop    f_n_s_loop
found_space:     pop     ax
                 ret
find_next_space  endp



;find_next_char
;   returns offset of next non-space, non-return, non-0 character
;      starting offset in BX
;      count in CX
;      result returned in BX
;      CX decremented by characters examined
;
find_next_char   proc    near
                 push    ax
f_n_c_loop:      mov     al,[bx]             ;load character
                 cmp     al,SPACE            ;is it a space?
                 je      cont_loop
                 cmp     al,RETURN           ;is it a return?
                 je      cont_loop
                 cmp     al,0                ;is it a 0?
                 je      cont_loop
                 jmp     found_char
cont_loop:       inc     bx                  ;keep on looking
                 loop    f_n_c_loop
found_char:      pop     ax
                 ret
find_next_char   endp
```

```
;disp_msg
;     displays an ASCIIZ message
;     offset of message in DS:DX
;
disp_msg        proc    near
                push    ax
                push    bx
                push    si
                mov     bl,FORE_COLOR
                mov     si,dx           ;point to string
d_m_loop:       mov     al,[si]         ;load character
                cmp     al,0            ;check for end
                je      d_m_end
                mov     ah,0eh          ;print char service
                int     10h             ;print it
                inc     si
                jmp     d_m_loop
d_m_end:        pop     si
                pop     bx
                pop     ax
                ret
disp_msg        endp


code            ends
                end       move
```

输入这段程序并以 move. asm 名存盘，编译，改错，连接。忽略“no stack”的警告。用 EXE2BIN 把它变成一个 COM 文件，全过程如下。

```
masm move;
link move;
exe2bin move
rename move.bin move.com
```

试一下程序。重命名文件和目录，在目录之间传送文件。用 MOVE 和 EXPLORER 传送一个目录。首先，用 EXPLORER 触发目录属性。然后，用 MOVE 传送目录，这时的目录如同一个文件一样。最后用 EXPLORER 恢复其目录属性。

文件传送时，由于把目录表项的第一个字符置为 e5h(删除文件指示符)，并清除了属性字节，所以原始的文件目录表项也被清除了。其它数据没有改变，文件分配表也没有动。可利用这个特征为同一目录建立多个路径。传送目录后，恢复其原始表项，即以原来的字符取代 e5h，并在属性字节里设置目录位。原始目录包含其原有的文件，被传送的目录也一样。

例如，假设你在目录\BUSINESS\FINANCES\BALSHEET 里保留了你公司的账目结果。为使存取更容易，把整个目录传送到根级，直接调用\BALSHEET 即可。恢复\BUSINESS\FINACES\BALSHEET 的表项，就有两个都指向你公司账目的目录。

程序设计要点：

- 若必要，结束程序时要恢复系统状态。然后使用一个终止中断。
- 程序应当核实输入的格式是否正确。
- COM 文件以 org 100h 开始。
- 把变量放在程序的开始使汇编效率更高。

第十九章　目　　录

DOS 磁盘中断把目录当作分隔实体对待，防止那些在正常文件上运行的程序，如字处理程序，不小心破坏了存储在目录中的重要信息。有一组中断专门处理目录。在本章，你将学会怎样建立和删除目录，怎样确定和改变当前目录及怎样搜索目录。最后，是一个实用程序 DIR2。

建立和删除目录

建立和删除目录的中断与 DOS 的 MKDIR 和 RMDIR 命令相似。只有空目录可以被删除。当前目录和根目录不能被删除。

操作：建立子目录
版本：DOS 2.0—
中断号：21h
功能号：57
调用值：
　AH＝57
　DS：DX＝要建立的子目录的 ASCIIZ 名字的指针。路径名可来自根目录或当前目录。
返回值：
　AX＝错误码
　若有错进位位置 1。错误码为 3(路径没找到)，5(拒绝访问)。

操作：删除子目录
　不能是根目录或当前目录
　必须是一个空目录
版本：DOS 2.0—
中断号：21h
功能号：58
调用值：
　AH＝58
　DS：DX＝要删除的目录的 ASCIIZ 名字的指针。路径名可来自根目录或当前目录。
返回值：
　AX＝错误码
　若有错进位位置 1。错误码为 3(路径没找到)，5(拒绝访问)，16(不能删当前目录)。

当前目录

当前目录是搜索及文件和目录建立时用的缺省目录。例如，若想打开 b:worms，DOS 就在当前 B 驱动器目录中搜索名为“worms”的文件。若想打开 b:\worms，DOS 则会在 B 驱动器根目录中搜索名为“worms”的文件。

操作：获取当前目录
版本：DOS 2.0—
中断号：21h
功能号：71
调用值：
AH＝71
DL＝驱动器号，0＝缺省，1＝A 等
DS:SI＝用户内存的 64 字节区的指针。在你的数据空间里一定要为该区另外设置空间。不要填入当前目录的 ASCIIZ 名字。
返回值：
AX＝错误码
DS:SI＝当前目录的 ASCIIZ 名字的指针。注：寄存器的值没有改变，但是 64 字节数据区被填满了。当前目录名以根目录名开始。不包括驱动器和第一个\。
若有错进位位置 1。错误码为 15(AL 中的驱动器号无效)。

操作：改变当前目录
版本：DOS 2.0—
中断号：21h
功能号：59
调用值：
AH＝59
DS:DX＝建立当前目录的 ASCIIZ 名字的指针。路径名可来自根目录或当前目录。
返回值：
AX＝错误码
若有错进位位置 1。错误码为 3(路径没找到)。

搜索目录中的文件

用来搜索目录中文件的中断有两个。用你要搜索的名字模式调用第一个中断。可使用通配字符。例如，可以使用象 b:utilites\failsafe.com 或 *.e? e 这样的模式。该中断找到要搜索的目录，扩充模式名的文件部分(例，*.e? e 扩充成???????? e? e)，并搜索第一个匹配表项。使用这个中断之后，用第二个中断继续搜索。若找不到匹配文件，两个中断都返回一个错误信

息。

可以选择搜索文件的类型，正常文件、隐含文件、系统文件和目录文件均可。也可搜索卷名。

文件找到后，返回数据，报告文件的各字、大小、建立日期及其它信息。这些数据存在 DTA 中。若一定要使用命令行参数，在开始目录搜索之前必须进行处理或拷贝。

下面是在 DTA 的信息格式。如果在目录搜索期间改变格式，搜索过程也将被修改。例如，可以将匹配模式改成开始时用一个模式来搜索而结束时则用另一个模式。注意一点，前 21 个字节的用法没有文献记载，改变这些值时要小心，调整驱动器或文件指针时更要小心。

字节	含　义
0	驱动器 1＝A，2＝B……
1－12	匹配模式。不包括驱动器或路径信息。将＊模式扩充成？模式。必要时可插入空格。不包括分隔文件名和扩展名的句点。匹配模式用于与目录中文件名表项的直接比较，不能用于删除文件。
14－15	文件名在目录中的位置。第一个目录表项是 0，第二个是 1，以此类推。这个计数包括删除文件和卷位置。例如，假设搜索所有文件，并且第一个匹配文件是卷，那么其位置是 0，然后是一个删除文件，它不予以匹配。再就是一个正常文件，其位置是 2。
16－17	目录(路径)位置
18－20	保留
21	找到的文件的属性。格式与目录中用的一样。
22－23	文件时间。格式与目录中用的一样。
24－25	文件日期。格式与目录中用的一样。
26－29	文件字节数。先低字后高字。
30－43	匹配文件的 ASCIIZ 名字。若有扩展名，则包括文件名和扩展名间的句点。不能用空格填充——即，是 DIR2. COM，不是 DIR2 . COM。

记住，使用目录搜索中断时，要先使用寻找第一个匹配文件的中断，然后再使用寻找下一个匹配文件的中断。

操作：寻找目录中的第一个匹配文件
版本：DOS 2. 0－
中断号：21h
功能号：78
调用值：
　AH＝78
　CH＝0

CL＝文件属性码。格式与在目录中的格式一样。为 0，只搜寻正常文件。设置隐含、系统或子目录位以包括那些搜寻中的文件。如果设置了卷位，只能找到卷表项。

DS:DX＝ASCIIZ 文件模式的指针。可包括驱动器和路径名字。路径可来自当前或根目录。必须给出全文件名模式。例如，当 b:*有效时，b:是无效的。

返回值：

AX＝错误码

文件信息存在 DTA 中

若有错进位位置 1。错误码为 2(文件没找到)和 18(无更多的匹配文件)。

操作：寻找目录中下一个匹配文件

必须已经调用过中断 21h，功能 78

版本：DOS 2.0—

中断号：21h

功能号：79

调用值：AH＝79

返回值：

AX＝错误码

文件信息存贮在 DTA 中

若有错进位位置 1。错误码为 18(无更多的匹配文件)。

DIR2：一个目录搜索实用程序

DIR2 使用目录搜索中断，这个中断功能与 DIR 相似，但它可以搜索隐含文件、系统文件和目录文件。此程序有两种选择：只打印隐含文件和目录文件或逐屏显示，换屏时暂停。

通过输入

```
dir2  file_pattern -options
```

来使用 DIR2。

文件模式和选择都是可选择项的，文件模式不仅包括路径，还必须包含文件名模式。选择项为 h,d,p，选择项前必须有一减号(－)。h 和 d 不能一起使用。选择 h 只打印隐含文件，d 只打印目录文件，p 则在显示区满之后暂停屏幕，按任一键后继续搜索。只打印文件名，目录表项用＜DIR＞表示。

例如，搜索 B 驱动器中名字以 m 开始的所有目录。如下操作：

```
dir2 b:m*.* -d
```

搜索当前目录的 NAMES 子目录中的所有文件，如下操作：

```
dir2 names\*.*
```

DIR2 是基于寻找第一个目录表项和寻找下一个目录表项这两个中断的。首先，DIR2 搜索命令行参数，若没有文件模式，则使用缺省 *.*，若没有选择项，则假设所有文件都不用停屏打印。选择项次序可任意指定。h 和 d 不能一起使用，但改变这个限制也很容易。

DIR2 检查参数时，将文件模式转换成 ASCIIZ 字符串。然后用寻找 DTA 的中断找到 DTA 位置。其在 PSP 中的缺省位置是 80。倒不一定要用寻找 DTA 的中断，但如果是许多人共用一个程序设计项目或是 DTA 改变了，那么寻找 DTA 位置就是很重要的了。然后，调用寻找第一个表项中断，若调用不成功，则打印出错信息；调用成功，则打印匹配文件名，并接着使用寻找下一个目录表项中断直到找不到匹配文件为止。将搜索属性字节设置成与所有属性匹配。如果设置了 h 或 d 标志，则要检查文件属性。

用一变量记录打印的行数。若设置了暂停选择项，则每隔 24 行就显示一个“Press any key to continue”的信息，例行程序暂停等待击键。这个由 BIOS 读键盘中断来完成。

打印文件名后，若该文件是个子目录，则显示“<DIR>”，然后打印回车和换行移到下一行。

注意，省略的参数使用缺省值，对选择项来说，缺省倒不重要。

print—it 例行程序放在 main—loop 例行程序的外面。根据流程它该在 main—loop 程序的里面，因为它不是一个子程序。它只有一个出口点。把它放在 main—loop 外是为了使循环结构更清楚。

若有错，返回码为 1；否则为 0。

```
;DIR2.ASM
;
;This program examines the disk directory and, like DIR, prints out
;names matching a pattern.  As an option, it will print out only
;hidden or directory files.  It also has an option to pause after
;every 24 lines.
;
;The format is:   DIR2 [file pattern] [- or / [h or d] [p]]
;
;For example, DIR2 C:*.COM -hp will list all hidden .COM files
;on the C drive, pausing every 24 lines.  Only one flag (- or /)
;should be used, even if there are several options.
;
;This is to be set up as a .COM file.
;


;--------------CONSTANTS--------------

PARAM_START     equ     81h
PARAM_LENGTH    equ     80h
SPACE           equ     20h
RETURN          equ     0dh
FORE_COLOR      equ     3
FLAG1           equ     '/'
FLAG2           equ     '-'
LINES_PER_PAGE  equ     24
ON              equ     1
OFF             equ     0
ALL             equ     0ffh
DIR_ONLY        equ     10h
HIDDEN_ONLY     equ     2
```

```
;--------------BEGINNING--------------

code            segment
                assume cs:code,ds:code
                org    100h                ;make this a .COM file
dir2:           jmp    main_code           ;go to main code


;--------------DATA--------------------
;
;Note: data is placed here to make assembling more efficient
;

pattern         dw     ?
attr_mask       db     ALL
pause_option    db     OFF
default_name    db     '*.*', 0
error_msg       db     '0 files found', 0ah, 0dh, 0
pause_for_key   db     '----Hit any key to continue----', 0
dir_msg         db     '     <DIR>',0
new_line        db     0ah, 0dh, 0

;--------------MAIN CODE---------------

main_code:      mov    ax,cs           ;set up segment registers
                mov    ds,ax
                mov    es,ax
;.................................
;
; Get the parameters
;
;.................................
                mov    cl,ds:[PARAM_LENGTH]
                mov    ch,0
                mov    bx,PARAM_START
                jcxz   set_defaults    ;no params
                call   find_next_char  ;find pattern or option flag
                jcxz   set_defaults    ;no params
                mov    al,[bx]         ;what is char?
                cmp    al,FLAG1        ;is it a flag?
                je     options_def_name;yes => read options, and
                cmp    al,FLAG2        ;get the default pattern
                je     options_def_name
                mov    pattern,bx      ;pattern starts here
                call   find_next_space ;find end, to make ASCIIZ
                mov    [bx],byte ptr 0 ;make ASCIIZ
                jcxz   start_loop      ;no options, use defaults
                call   find_next_char
                jcxz   start_loop      ;no options, use defaults
                mov    al,[bx]         ;is the character an
                cmp    al,FLAG1        ;option flag?
                je     rd_option       ;read the options
                cmp    al,FLAG2
                je     rd_option
                jmp    start_loop
;.................................
;
; Call routines to read in options or defaults
;
;.................................
set_defaults:   call   get_default_name
                jmp    start_loop
options_def_name: call read_options
                call   get_default_name
                jmp    start_loop
rd_option:      call   read_options
;..... ...........................
;
; Search through directory as long as possible, printing file names
; if they match the pattern and options.
```

```
;
;-------------------------------
start_loop:     mov     si,0            ;si = line count
                mov     ah,2fh          ;get DTA address, which will
                int     21h             ;be 80.  Check it anyway.
                                        ;result in ES:BX
                mov     ax,4e00h        ;find first matching file
                mov     dx,pattern
                mov     cx,1ah          ;match to all
                int     21h
                cmp     al,0            ;was there an error?
                je      main_loop       ;no.
                lea     dx,error_msg    ;an error.
                call    disp_msg
                mov     al,1            ;error return code
                jmp     dir2_end

main_loop:      cmp     attr_mask,ALL   ;display all files?
                je      print_it        ;yes, so print name
                mov     al,es:[bx]+21   ;look at file attribute
                and     al,attr_mask    ;see if it matches option
                jne     print_it        ;yes, so print name
cont_match:     mov     ax,4f00h        ;look for next matching file
                int     21h
                cmp     al,0            ;no more files?
                je      main_loop
                mov     al,0            ;successful finish
dir2_end:       mov     ah,4ch          ;terminate
                int     21h

;-------------------------------
;
; Print file name, pause for keystroke if necessary,
; then return to loop.
;
;-------------------------------

print_it:       cmp     pause_option,ON     ;check for pause?
                jne     p_cont
                cmp     si,LINES_PER_PAGE   ;time to pause?
                jne     p_cont
                lea     dx,pause_for_key    ;yes, display message
                call    disp_msg
                mov     ah,0                ;wait for keystroke
                int     16h
                lea     dx,new_line         ;advance to a new line
                call    disp_msg
                mov     si,0                ;reset count
p_cont:         mov     dx,bx               ;offset of file name
                add     dx,30
                call    disp_msg            ;print it
                test    es:[bx]+21,byte ptr DIR_ONLY
                                                    ;is it a subdirectory?
                jz      p_new_line
                lea     dx,dir_msg
                call    disp_msg
p_new_line:     lea     dx,new_line         ;advance to a new line
                call    disp_msg
                inc     si                  ;increment line count
                jmp     cont_match          ;continue with search



;--------------SUBROUTINES------------
;read_options
;   examines command line characters following BX to set options.
;
```

```
read_options    proc   near
                push   ax
                inc    bx        ;look at char following flag
                mov    al,[bx]
                call   check_options
                inc    bx        ;look at next char
                mov    al,[bx]
                call   check_options
                pop    ax
                ret
read_options    endp


;check_options
;   given a character in AL, sets pause_option or attr_mask
;   accordingly
;

check_options   proc   near
                cmp    al,'p'
                je     set_pause
                cmp    al,'P'
                je     set_pause
                cmp    al,'h'
                je     set_hidden
                cmp    al,'H'
                je     set_hidden
                cmp    al,'d'
                je     set_dir
                cmp    al,'D'
                je     set_dir
                jmp    c_o_end
set_pause:      mov    pause_option,ON
                jmp    c_o_end
set_hidden:     mov    attr_mask,HIDDEN_ONLY
                jmp    c_o_end
set_dir:        mov    attr_mask,DIR_ONLY
c_o_end:        ret
check_options   endp

;get_default_name
;   Because no pattern was given, just use current directory
;   in default drive. (*.*)
;

get_default_name proc  near
                 push  ax
                 lea   ax,default_name
                 mov   pattern,ax
                 pop   ax
                 ret
get_default_name endp



;**
;**
;**  Place find_next_space, find_next_char, and disp_msg here.
;**  Their code is in MOVE.ASM  (Chapter 18).
;**
;**


code             ends
                 end   dir2
```

输入这段程序，以 dir2. asm 名存盘。汇编，改正输入错误，连接。把它变成一 COM 文件。然后试一遍。一定要试试开关。

程序设计要点：

- 有专门处理目录的 DOS 功能。
- 文件建立和目录搜索时把当前目录作为缺省目录。
- 用寻找第一个目录表项中断开始目录搜索，然后重复使用寻找下一个目录表项中断。这些功能将信息存储在 DTA 中。

第二十章　存储器

本章讨论三种类型的 RAM(随机存取存储器):常规存储器、扩充存储器、扩展存储器。**常规存储器**为通常处理的存储器——即最前面的一兆字节。这其中的前 640K 是不保留的。**扩充存储器**是接在常规存储器之后的 15 兆字节。AT 以上的机器才有这种存储器,而且 DOS 1.0—3.3 不能对其寻址。**扩展存储器**是用一特定的存储器扩展板(如 Intel Above Board)增加的页式存储器。

常规存储器

当一个程序装入后,所有的空存储器都分配给该程序,也就是只要是在单用户环境里,你就可以放心地在前 640K 里使用任何一个不保留的存储器来存储数据。但这样做阻止了子程序的使用。如果数据要占用很大的存储区或是你计划产生子程序,则首先必须释放该程序不用的所有存储器。在必须使用大块存储器时,要发出一个存储器请求。DOS 监视空的存储器,并分配一部分给程序。这部分存储器用完之后要释放。

控制存储器的中断有三个,都属于中断 21h。这些功能以 16 字节(一段)一组为单位处理存储器。功能 72 分配存储器。当需要存储器存储数据时就调用它。它与 C 的 alloc 功能相似。功能 73 释放由功能 72 分配的存储器。在程序结束前要释放所有已分配的存储器。功能 74 修改一个存储块的大小。在使用其它存储器中断之前,必须用这个功能使分配给程序的存储器空间与程序的大小一致。为此,必须确定程序在哪结束。单段程序,在结束尾设一个标志,多段程序在结尾设一伪段。如果使用扩号方法,把偏移量除以 16,加上段值,结果再加 1,就是结束段址。

DOS 用一连接表记录存储器(该连接表保存存储块的第一节),并监视其大小及是否为空。在 DOS 里,节是一个 16 字节的块。节是顺序编号的,从常规存储器的 0 字节开始,0 字节的段即为 0 节。节号与段号是一回事,只是节指的是 16 字节的块,而段指的是 64K 字节的块。使用存储器功能调用时,DOS 扫描连接表找到你想释放或想修改或它可分配的存储器块。

操作:分配内存
版本:DOS 2.0—
中断号:21h
功能号:72
调用值:
　AH=72
　BX=要分配的节数
返回值:
　AX=分配块的起始节或错误码
　BX=最大可用块的节数(若有错)

若有错进位位置 1。错误码为 7(内存控制块被破坏),为 8(内存不够)。

操作:释放分配的内存块
内存块必须是已分配给的
版本:DOS 2.0—
中断号:21h
功能号:73
调用值:
AH=73
ES=要释放的内存块的起始节
返回值:
AX=错误码
若有错进位位置 1。错误码为 7(内存控制块被破坏),9(无效内存块地址)。

操作:修改内存块的大小
可减少或扩大
内存块必须是已分配给的
版本:DOS 2.0—
中断号:21h
功能号:74
调用值:
AH=74
ES=要修改的内存块的起始节
BX=新的大小,以节为单位。
返回值:
AX=错误码
BX=可用的最大内存块的大小(若想扩大内存块大小而且有错)
若有错进位位置 1。错误码为 7(内存控制块被破坏),为 8(内存不够),为 9(无效内存块地址)。

用这些调用对已分配的存储器寻址的最简单方法是在一个段寄存器里装入起始节号。该段的偏移量 0 就是块中的第一个字节。

如果已分配给你一个块,但不符合你想要的大小,就再分配一个符合你要求的新块。如果分配成功,把旧块中的信息拷贝到新块并且释放旧块。

为确定可用存储器的总数,把 ES 指向 PSP ,用 ffffh 个节扩展存储块。这样做会引起出错,但空存储器的数量将返回到 BX 中。记住,如果想得到一个有意义的答复就要释放程序不用的存储器(反之,BX 将返回 0)。

用下列中断确定常规存储器的总数。该中断检查硬件配置开关。

操作:报告在计算机里安装的常规存储器的数量
版本:BIOS
中断号:12h
调用值:NA
返回值:AX=已安装的存储器的千字节

扩充存储器

只有 AT 以上的机器才有扩充存储器。

操作:确定扩充存储器的数量
版本:AT BIOS
中断号:15h
功能号:136
调用值:AH=136
返回值:AX=扩充存储器的千字节

扩展存储器

扩展存储器是用一个扩展存储器板来增加的,可增加几兆字节,但一次只能访问其中的一小部分。在旁边设置一部分常规存储器作为扩展存储器窗口。可通过这个窗口寻址部分扩展存储器。所有处理此窗口存储器地址内的存储器单元的程序指令,都由扩展存储器板硬件改为对映象到该窗口的扩展存储器部分进行寻址。用中断控制被寻址的存储器部分。

例如,假设你有一兆字节的扩展存储器,窗口大小为 16K。为搜索所有扩展存储器,要使用一个中断先查看前 16K,进行处理,再看下一个 16K,再进行处理,以此下去。一次只能看 16K 的扩展存储器。假设窗口从 C000:0000 开始,扩展存储器的第二个 16K 被映象到此窗口。如果一个程序要看存储器的单元 C000:0001h,那么它将会看到存储在扩展存储器 4001h(第二个 16K 的第一个字节)单元的值。若一程序改变存储器单元 0000:0017h 的值,那么将改变存储在扩展存储器 4017h 单元里的值。

几家扩展存储器板制造商采用不同的使用存储器的标准。本书讨论 Lotus/Intel/Microsoft 扩展存储器规范,简称 EMS,这是目前最广为接受和使用的标准。

采用 EMS,可通过位于高存储器(640K 以上)的四个相邻 16K 窗口寻址到至多 8 兆字节的存储器,这些窗口的位置可由一个中断调用确定。所有扩展存储器功能调用都在中断 67h 中。

使用扩展存储器和使用文件及常规存储器相似。需要扩展存储器时,就发出一以 16K 为单位,分配一定数量存储器的中断请求。该中断返回一个手柄号,通过这个手柄号你可处理那个存储块。扩展存储器管理软件(EMM)控制哪一个物理扩展存储器页为存储块所用。你可把存储块看成是从 0 开始编号的相邻页组。观察存储器要指定手柄和页号,还要指定你想在哪一个窗口中存取信息。例如,你想使用扩展存储器存储动画的图形屏幕的图象。你想把五张图

片排序，假设每张图片占 16K，因此你需要 5 个 16K 的存储块。假设手柄号为 7。看第二张图片，要看手柄 7 的第二页，当然该页的物理位置可能在扩展存储器中的任意位置。设窗口在段 C000h 开始，你把该页映象到第三个窗口，然后你就可以通过存储器单元 C800:0000h—C800:3fffh 寻址图形信息。

再例如，假设你想在扩展存储器里存储一个 128K 的数组。发出分配请求，并得到一个手柄号。看一个阵列元素，要指出它在存储块里的偏移量，用偏移量除以 16K，得到其所在页，然后再请求看这一页。同时还能将 4 个数组页映象到窗口。这样，你可立刻寻址到 64K 的数组。

扩展存储器一旦用完就应释放。

应用程序必须记录它所用的窗口以防重写重要数据。

程序使用扩展存储器之前，应当检查扩展存储器管理设备驱动程序是否安装，扩展存储器板是否存在并可操作。然后还应当找到窗口的起始位置。

执行下列步骤，检查软件驱动程序是否安装：

1)找到中断 67h 指向的地址。第 23 章和第三部分将详细讨论这点。用中断 2h，功能 53：

```
mov  ah,53      ;read value for interrupt
mov  al,67h     ;67h and return location in
int  21h        ;es:bx
```

2)把程序里(ES:000Ah)在偏移 10 开始的 8 个字节与 ACSII 字符串 EMMXXXX0 比较。

3)若比较结果相同，则安装了扩展存储管理软件。这时你就可以使用任何一个扩展存储器板中断。

前面提到，所有的存储器中断都在中断 67h 里，功能号放在 AH 里。错误码返回到 AH 中。下面列出错误码及其含义。

错误码	含　义
0	没错
128	EMM 软件故障
129	EMM 硬件故障
130	未用
131	无效手柄
132	无效功能代码
133	无可用的 EMM 手柄(所有手柄都在使用)
134	页映象上下页错误
135	无足够的扩展存储器页
136	无足够的空存储器页
137	不能分配 0 页
138	请求比已分配给该手柄的页数多的页
139	手柄页映象到的物理存储页无效
140	无空间存储扩展存储器板状态
141	扩展存储器板状态信息已与手柄号有关

142　　无状态信息与手柄号有关
143　　无效子功能

接下来，看看主要的扩展存储器中断。记住，在使用这些中断之前先要检查扩展存储器软件驱动程序是否安装。如果没有安装驱动程序而调用中断，你的计算机将挂起。还有，DOS 不要使用扩展存储器手柄扩展存储器，也不要使用 DOS 手柄。两者之间不存在相关性，如果这样做了，你会得到出乎意料的结果。

操作：检查扩展存储器板状态
版本：EMS
中断号：67h
功能号：64
调用值：AH＝64
返回值：
　AH＝错误码。错误码为 128，129，132。若扩展板已安装并正在工作，则没有错误(0)。

操作：获取窗口位置
版本：EMS
中断号：67h
功能号：65
调用值：AH＝65
返回值：
　AH＝错误码。错误码为 128，129，132。
　BX＝窗口开始的段。第一个窗口在偏移量 0，第二个窗口在偏移量 4000h；第三个窗口在偏移量 8000h；第四个窗口在偏移量 c000h。

操作：确定空扩展存储器页和全部扩展存储页的数量
　　一页为 16K
版本：EMS
中断号：67h
功能号：66
调用值：AH＝66
返回值：
　AH＝错误码。错误码为 128，129，132。
　BX＝空存储器页的数量
　DX＝存储器页的总数

操作：分配扩展存储器页
　　页由返回的手柄引用

版本:EMS
中断号:67h
功能号:67
调用值:
AH=67
BX=要分配的存储器页的数量
返回值:
AH=错误码。错误码为 128,129,132,133,135,136,137。
DX=手柄

操作:获取扩展存储器页
把属于一个手柄的一页映象到一个窗口
通过窗口的内存地址寻址数据
版本:EMS
中断号:67h
功能号:68
调用值:
AH=68
AL=窗口号(0-3)
BX=要获取的页。手柄的第一页是 0,第二页是 1 等。
DX=手柄
返回值:
通过窗口位置可寻址页数据
AH=错误码。错误码为 128,129,131,132,138,139。

操作:释放扩展存储器块
版本:EMS
中断号:67h
功能号:69
调用值:
AH=69
DX=要释放的存储块的手柄。释放分配给手柄的所有页。
返回值:
AH=错误码。错误码为 128,129,131,132,134。

还有几个实用程序功能,你可用其编写一个观察扩展存储器的实用程序(类似EXPLORER)。显示分配给一个手柄的存储块,要找到所有的手柄及其大小。然后,通过它们的数据开窗口。包括译码目录信息的能力,看空存储器,先要把它分配给存储块。从你所能分配的最大的存储块开始。如果不能分配那么大的存储块,就一次减少一页直到你能分配的最大存储块为止。这样就可以使用尽可能少的手柄,还可观察所有的空存储器。之后,一定要释放所

分配的存储块。

操作：获取 EM 软件驱动程序的版本号
版本：EMS
中断号：67h
功能号：70
调用值：AH＝70
返回值：
AH＝错误码。错误码为 128,129,132。
AL＝版本号。高位是主要版本号(也就是 3.1 里的 3),低位是次要版本号(也就是 3.1 中的 1)。

操作：确定活动手柄数
版本：EMS
中断号：67h
功能号：75
调用值：AH＝75
返回值：
AH＝错误码。错误码为 128,129,131,132。
BX＝手柄数
BH 总为 0

操作：确定存储块大小
版本：EMS
中断号：67h
功能号：76
调用值：

AH＝76
DX＝手柄
返回值：
AH＝错误码。错误码为 128,129,131,132。
BX＝分配给手柄的存储页数。大于 0,小于或等于 512。

操作：获取所有手柄大小的数组
版本：EMS
中断号：67h
功能号：77
调用值：

AH＝77

ES:DI＝要存贮结果的数据区的指针。该区大小至少为 4×(活动手柄数)字节。不能超过 1K。也不能超过 ES 段边界。例如,若数组是 10 个字节长,就不要放在 ES:FFF8h。

返回值:

AH＝错误码。错误码为 128,129,132。

BX＝活动手柄数。BH 总是 0。

ES:DI＝包含大小信息的数组的指针。寄存器值不改变,但将填充数据,每个 BX 表项 2 字长,在 DI＋0 开始,表项的第一个字是手柄号,第二字为手柄大小。

在内存驻留程序和设备驱动程序里使用扩展存储器时要特别小心。这些例行程序和驱动程序一定会保存它们分配的手柄,必要时还要解除分配手柄。在它们访问扩展存储器时,必须保存扩展存储器板的当前状态——即有关通过每个窗口寻址什么数据的信息。它们要装载扩展存储器板最后一次使用时的状态信息。结束之前,它们一定要恢复其被调用之前的状态。

保存和恢复状态由两个中断来完成,每个中断都需要传送给一个手柄号。为防止冲突,这个手柄号应当为驻留程序所有(即由驻留程序初始化)。

驻留程序装入时,还要分配所有必需的扩展存储器并在其被调用时使用这个存储块。若是这样,在保存和恢复扩展存储器板状态时使用该存储块手柄。

一台计算机可使用几个扩展存储器板,状态保存和恢复中断保存并恢复所有这些板的状态。

操作:保存扩展存储器板状态

版本:EMS

中断号:67h

功能号:71

调用值:

AH＝71

DX＝与状态相联的手柄。手柄应当由保存状态的程序拥有。

返回值:

AH＝错误码,错误码为 128,129,131,132,140,141

操作:恢复扩展存储器板状态

版本:EMS

中断号:67h

功能号:72

调用值:

AH＝72

DX＝与要恢复的状态相联的手柄。状态应用功能 71 已与此手柄相联。

返回值：

恢复旧状态(窗口映象)

AH＝错误码。错误码 128,129,131,132,142。

EMS 标准也支持多任务环境。有关这方面的全面讨论和一些汇编语言的讨论，请参看 LIM EMS 手册。

程序设计要点：

- 存储器有三种类型：**常规、扩充、扩展**。
- 程序装入后，所有空存储器都分配给它。释放程序不用的所有存储器。
- DOS 3.3 以上版本(包括 3.3)才使用扩充存储器。
- 扩展存储器是页式的，并映象到高存储器窗口。
- EMS 包括许多分配、使用、释放扩展存储器的功能。

第二十一章　磁盘扇区和驱动信息

本章讨论如何访问磁盘扇区和驱动信息。正如你在 EXPLORER 看到的一样，访问磁盘扇区有两个方法——BIOS 调用和 DOS 调用。BIOS 调用使用面、道、扇区，DOS 调用只使用一个扇区号。BIOS 调用处理绝对的磁盘单元，忽略分区界限。DOS 调用处理相对单元，扇区 0 指的是当前分区中的第一个扇区，而且不能读当前分区外的扇区。

一般 DOS 调用更易于使用因为它们处理相对扇区，所以你不必决定什么时候切换面和道。用 BIOS 调用访问当前分区外的扇区。

注意，DOS 读和写中断返回时，初始标志却留在栈中。因此调用中断后，必须把这些标志清除出栈。或者检查错误标识符的标志，然后 popf 或者把标志弹出送入一个伪单元。如果没有把这些标志清除出栈，子程序的返回地址将是不正确的。

操作：读磁盘扇区
　　DOS 扇区编号体系
　　原始标志返回栈中
版本：DOS 1.0—
中断号：25h
调用值：
　AL＝驱动器，0＝A，1＝B，2＝C 等
　DS：DX＝要存贮读信息的区域的指针
　CX＝要读的扇区数
　DX＝要读的第一个扇区
返回值：
　AX＝错误码（见下个中断后的表）
　若有错进位位置 1
　除了段寄存器所有寄存器都可能遭到破坏
　原始标志返回到栈中

操作：写磁盘扇区
　　DOS 扇区编号体系
　　在栈中返回原始标志
版本：DOS 1.0—
中断号：26h
调用值：
　AL＝驱动器，0＝A，1＝B，2＝C 等。
　DS：BX＝要从此读信息往磁盘写的区域的指针
　CX＝要写的扇区数

DX＝要写的第一个扇区

返回值：

AX＝错误码

若有错进位位置 1

除了段寄存器所有寄存器都可能遭到破坏

在栈中返回原始标志

中断 25h，26h 的错误码如下。DOS 2.0 技术参考手册中叙述的一些硬件错误的 AH 代码与这不同。但 AL 值是正确的。

AH 值	含　义
2	不知道的错误
3	磁盘写保护
4	坏扇区号
8	坏 CRC
64	不能访问磁道
128	驱动器不响应

AL 值	含　义
0	磁盘写保护
1	无效驱动器号
2	驱动器没准备好
4	坏 CRC
6	不能访问磁道
7	不能识别的磁盘格式
8	未找到扇区
10	写错
11	读错
12	一般的未知错误

下面两个中断是 BIOS 磁盘读和写中断。如果读或写一个以上的扇区，这些扇区必须在同一面和磁道上。

操作：读磁盘扇区

BIOS 磁盘编号体系

扇区必须在相同的面和磁道上

版本：BIOS

中断号：13h

功能号：2

调用值：

AH＝2

AL＝要读的扇区数。不应为 0。

ES:BX＝要存贮读信息的区域的指针

CL＝低六位是扇区号，高二位是磁道号的高二位。

CH＝磁道号的低八位

DL＝驱动器号。0－3 为软盘，80h－87h 为硬盘。

DH＝面

返回值：

AH＝错误码(见下表)

若有错进位位置 1

操作：写磁盘扇区

BIOS 扇区编号体系

扇区必须在同一面和磁道上

版本：BIOS

中断号：13h

功能号：3

调用值：

AH＝3

AL＝要写的扇区数。不应为 0。

ES:BX＝要从此读信息往磁盘写的区域的指针

CL＝低六位是扇区号，高二位是磁道号的高二位。

CH＝磁道号的低八位

DL＝驱动器号。0－3 为软盘，80h－87h 为硬盘。

DH＝面

返回值：

AH＝错误码(见下表)

若有错进位位置 1

下面是 BIOS 读和写中断的错误代码。

值	含　义
1	坏命令
2	未找到扇区地址标志
4	未找到扇区
9	DMA 失败
10	坏扇区
11	坏道
16	坏 ECC(奇偶错)

17	已检测和改正的数据错 ——数据极大可能是好的
32	磁盘控制器失败
64	不能访问磁道
128	驱动器不响应
170	驱动器没准备好
187	未知错
204	写故障

磁盘信息

一些驱动器，像 AT 高密度软盘驱动器，可以检测磁盘是否被改变了。有两个 AT 中断完成此功能，一个确定驱动器的类型，一个报告磁盘是否被改变了。

操作：获取磁盘类型
版本：AT BIOS
中断号：13h
功能号：21
调用值：
AH＝21
DL＝驱动器号
返回值：
AH＝驱动器类型

值	含义
0	无驱动器
1	软盘驱动器，不能检测磁盘变化
2	软盘驱动器，可检测磁盘变化
3	硬盘

CS:DX＝硬盘上的扇区数(若为硬盘)

操作：检查改动的磁盘
版本：AT BIOS
中断号：13h
功能号：22
调用值：
AH＝22
DL＝驱动器号(0 或 1)
返回值：
AH＝6(若磁盘改动了)，否则为 0。
若磁盘改动了，进位位置 1。

获取磁盘信息调用提供有关磁盘的有用信息。由功能50返回的信息格式如下。

字　节	含　义
0	驱动器号－0＝A，1＝B，等等
2－3	每扇区的字节数
4	每簇的扇区数－1
5	log2(每簇扇区数)
6－7	引导记录的扇区数
8	FAT的个数
9－10	根目录中的表项数
11－12	数据区的第一个扇区
13－14	最后的簇号
15	FAT中的扇区数
16－17	根目录的第一个扇区
18－21	内部
22	磁盘类型字节

操作：获取磁盘信息
版本：DOS 2.0－
中断号：21h
功能号：54
调用值：
　AH＝54
　DL＝驱动器，0＝缺省，1＝A等
返回值：
　AX＝每簇扇区数。若无效驱动器为ffffh。
　BX＝空簇数
　CX＝每扇区字节数
　DX＝总簇数

操作：获取磁盘信息
版本：无文献记载。DOS 2.0－
中断号：21h
功能号：50
调用值：
　AH＝50
　DL＝驱动器号，0＝缺省，1＝A等
返回值：
　A＝0(若有驱动器)

=ffh(若有错)

DS:BX=磁盘信息块的指针

注:改变了 DS 段寄存器

程序设计要点:

- BIOS 调用使用面、道和扇区号确定绝对磁盘位置。DOS 功能使用与分 区开始有关的扇区,并使用逻辑扇区号。
- 高密驱动器能检测磁盘是什么时候改变的。
- 有几个中断能提供磁盘信息。一般,使用这些中断比读引导记录更好。

第二十二章　子程序和覆盖

子程序是被另一程序调用的程序。子程序在父程序后装入存储器并执行。执行结束后，控制返回到父程序。覆盖程序就象一个子程序一样，但是其不执行。父程序可以调用位于覆盖程序里的任何程序。

在制造集成软件包时子程序很有用。前面讨论过，父程序在操作期间可调用任何其它程序——如字处理程序。子程序结束后，父程序仍然返回到离开时的位置继续运行。还可在某个程序运行中用子程序执行 DOS 命令。把 COMMAND. COM 作为子程序。输入 DOS 命令 exit 返回到父程序。

大程序使用覆盖。若可能，把一个大程序分成一个普通控制程序和一组自含模块。例如，一个字处理程序，就有一个具有 cut 和 paste 功能的模块和一个具有搜索功能的模块。当需要一特定模块时，如果该模块不在存储器里，普通控制程序把其作为一个覆盖装入。这样可使用该模块里的任何程序。

执行子程序之前，父程序必须释放它不用的所有存储器，象在第 20 章里讲的一样。子程序将破坏所有寄存器，包括栈寄存器，只有 cs 和 ip 可以恢复。因此要 push 所有必须保存的寄存器，包括 ds 和 es。把 ss 和 sp 存在位于代码段的变量里。子程序返回时，恢复 ss 和 sp 并 pop 其它寄存器。子程序装入时，自动生成一个 PSP。在生成子程序的中断后的母体语句设置成结束和 Ctrl－Break 向量。PSP 结束和 Ctrl－Break 地址拷贝包含父程序使用的向量。子程序结束后，恢复父程序的结束和 Ctrl－Break 向量。子程序用中断 21h，功能 76 结束时返回一个返回代码，如第 18 章中讨论的一样。用中断 21h，功能 77 读该返回码。

父程序使用的所有手柄子程序也能使用，除非它们是以不继承的方式打开的（见中断 21h，功能 61，AL 位 7）。

生成子程序的中断使用下列格式的参数表：

偏移量	含　义
0－1	环境字符串开始的段。环境须在该段内的偏移 0 开始。环境为一组 ASCIIZ 字符串，如 VERIFY＝ON。环境的最后一个 ASCIIZ 后必须跟一 0 字节。使用父环境时，要把此表项填 0（即字节 1 和 2 设置为 0）。
2－3	命令行参数的偏移量

4－5	命令行参数的段，这些参数与那些在命令行上及 PSP 里的参数格式一致。所指向的第一个字节应当包含命令行参数的字符数。参数由一返回字符(ASCII 13)结束。若没有命令行参数，2－5 字节填 0。
6－7	FCB1 的偏移量
8－9	FCB1 的段
10－11	FCB2 的偏移量
12－13	FCB2 的段。以前的文件访问方法需要 FCB。只要你使用本书讨论的方法(手柄方法)，就不需要 FCB 而且应当把 6－13 字节填 0。覆盖装入时，覆盖程序不执行，也不生成 PSP。使用覆盖之前，父程序一定要释放掉它不用的所有存储器，就象调用子程序时一样。给最大的覆盖分配足够大的存储块。把覆盖作为一远过程调用。

覆盖功能需要如下格式的参数表：

偏移量	含　义
0－1	覆盖将被装入的段。覆盖将在该段的偏移 0 开始装入。
1－2	重定位调节系数。与字节 0－1 的值相同。

覆盖可包含任意段，包括它自己的栈段。段寄存器一定要由覆盖改变以指向它自己的段。

操作：装载并运行子程序或装载覆盖程序
版本：DOS 2.0－
中断号：21h
功能号：75
调用值：
AH＝75
AL＝命令码

值	含　义
0	装入并运行子程序
3	装覆盖程序

ES：DX＝参数表的指针。若命令码为 0，则使用子程序类型参数表。若命令码为 3，则使用覆盖程序类型参数表。
返回值：
AX＝错误码
若有错为进位位置 1。错误码为 1(AL 中的命令码无效)，2(文件没找到)，5(拒绝访问)，8(内存不够)，10(无效环境)，11(无效表格式)。
若装入并运行了一个子程序，除了 CS 和 IP 所有寄存器都将遭到破坏。

操作：恢复子程序的返回代码

代码只能恢复一次

版本：DOS 2.0—

中断号：21h

功能号：77

调用值：AH＝77

返回值：

AL＝返回码

AH＝指明子程序终止方式的码

值	含义
0	正常终止
1	由 control break 终止
2	由于致命错误而终止
3	终止并驻留

程序设计要点：

- 程序可调用子程序。
- 子程序在返回父程序之前破坏所有的寄存器。
- 子程序可使用以继承方式打开的父程序手柄。
- 覆盖程序不执行。可由远调用访问。必要时覆盖进程负责调节 DS 和 ES 寄存器。

第二十三章　处理中断的中断

执行 int 指令时，中断例行程序的地址从一个存在存储器里的表装入，该表在段 0，偏移 0 (0000:0000h)开始。每个表项由对应中断程序的偏移量和段组成。该表能容纳 256 个表项。详见第 26 章的“中断”部分。

可使用中断检查或改变存在中断表里的地址。一般只在建立或检查内存驻留程序或设备驱动程序时才这样做。例如，要看看扩展存储器板软件驱动程序是否安装，就一定要找到中断 67h 的程序地址。

操作：确定中断例程的地址
版本：DOS 2.0—
中断号：21h
功能号：53
调用值：
　AH＝53
　AL＝要检查的中断号
返回值：
　ES＝中断例程的段
　BX＝中断例程的偏移

操作：设置中断例程的地址
版本：DOS 1.0—
中断号：21h
功能号：37
调用值：
　AH＝37
　AL＝要设置的中断号
　DS＝新中断例程的段
　DX＝新中断例程的偏移

程序设计要点：

- 中断表包含所有中断程序的地址。
- 可重定向中断向量使之指向新的中断例行程序。这将在第三部分讨论。

第二十四章　系统和设备信息

已经讨论过几个确定硬件类型及有关磁盘信息的中断。本章将介绍更多的提供系统和设备信息的中断。

操作：确定系统建立
版本：BIOS
中断号：11h
调用值：NA
返回值：

AX＝设备表

位	含　义
0	若连接软盘驱动器则设置
1	未用(PC)
	安装了算术协处理器(AT)
4－5	初始视频方式

值	含　义
1	40×25 文本，彩色卡
2	80×25 文本，彩色卡
3	80×25 文本，单显卡

6－7	磁盘驱动器数。加 1。若位 0 为 1 则无效。
9－11	RS－232 端口数
12	若安装了游戏适配器则设置(PC)
	未用(AT)
13	未用(PC)
	若安装了内部调制解调器则设置(AT)
14－15	打印机数

从单元 ffff：000eh 字节，可确定计算机类型。

值	计算机
252	AT
253	PCjr
254	XT
255	PC

操作：确定 DOS 版本

版本:DOS 2.0—
中断号:21h
功能号:48
调用值:AH=48
返回值:
AL=主要版本号(3.10 中的 3)
AH=次要版本号(3.10 中是 10)。用数字格式。
BX,CX 被破坏

操作:确定当前驱动器
版本:DOS 1.0—
中断号:21h
功能号:25
调用值:AH=25
返回值:
AL=当前驱动器号。0=A,1=B,2=C 等

切换字符用来指定 DOS 命令行里的选择项。一般是斜杠(/),例如:FORMAT B:/V。

操作:确定或设置 DOS 切换字符
版本:无文献记载。DOS 2.0—
中断号:21h
功能号:55
调用值:
AH=55
AL=命令码

值	含 义
0	确定切换字符
1	设置切换字符

DL=新切换字符(若设置切换字符)
返回值:DL=切换字符(若确定切换字符)

操作:确定日期
版本:DOS 1.0—
中断号:21h
功能号:42
调用值:AH=42
返回值:
AL=星期,0=星期日,1=星期一,等等
CX=年(1980—2099)

DL＝日(1－31)
DH＝月,1＝一月,2＝二月,等等

操作:设置日期
版本:DOS 1.0－
中断号:21h
功能号:43
调用值:
AH＝43
CX＝年
DL＝日
DH＝月,1＝一月,2＝二月,等等。
返回值:
AL＝0(若操作成功)
＝ffh(若无效),不变化。

操作:确定时间
版本:DOS 1.0－
中断号:21h
功能号:44
调用值:AH＝44
返回值:
CL＝分(0－59)
CH＝时(0－23)
DL＝百分秒(0－99)
DH＝秒(0－59)

操作:设置时间
版本:DOS 1.0－
中断号:21h
功能号:45
调用值:
AH＝45
CL＝分
CH＝时
DL＝百分秒
DH＝秒
返回值:
AL＝0(若操作成功)
＝ffh(若无效),不变化。

设备信息

IOCTL 中断比第 17 章讨论的基本询问提供更多的设备信息。

操作:检查可移动的介质
版本:DOS 3.0—
中断号:21h
功能号:68
调用值:
AH=68
AL=8
BL=驱动器号。0=缺省,1=A 等。
返回值:
AX=0(若为可移动的)
=1(若为固定的)
=15(若是无效驱动器号)

操作:检查设备是局部的还是远程的
版本:DOS 3.1—
中断号:21h
功能号:68
调用值:
AH=68
AL=9
BL=驱动器号。0=缺省,1=A 等。
返回值:DX=位 12(若是远程的)

操作:检查手柄是局部的还是远程的
版本:DOS 3.1—
中断号:21h
功能号:68
调用值:
AH=68
AL=10
BX=手柄
返回值:DX=位 15(若是远程的)

操作:检查分配给设备的驱动器字母数
版本:DOS 3.2—

中断号：21h

功能号：68

调用值：

AH＝68

AL＝14

BL＝驱动器号。0＝缺省，1＝A 等。

返回值：

AL＝0(若设备只有一个驱动器字母)

＝其它(用于存取设备的最后驱动器的驱动器号)

若有错进位位置 1，错误码返回到 AX 中。

下列中断返回扩展设备信息：

字节	含　义
0	为 0 则返回一个建立 BPB 参数块 为 1 则返回一个缺省的 BPB 参数块
1	设备字节

值	含　义
0	5.25"双密磁盘
1	5.25"高密磁盘
2	3.5"磁盘
3	8"单密磁盘
4	8"双密磁盘
5	硬盘
6	磁带驱动器
7	其它

2　设备属性：

位	含　义
0	若是固定介质则置 1
1	若能检测软盘改变则置 1
3－4	磁道数
5	驱动器介质类型。 0 指缺省类型。
6－7	每扇区字节数
8	每簇扇区数
9－10	保留扇区(引导记录)
11	FAT 的个数
12－13	根目录的表项数
14－15	扇区数
16	磁盘类型字节

17—18　每 FAT 扇区数
19—20　每磁道扇区数
21—22　面数
23—26　隐含扇区

操作：获取设备信息
版本：DOS 3.2—
中断号：21h
功能号：68
调用值：
AH＝68
AL＝13
BL＝驱动器号。0＝缺省，1＝A 等。
CL＝96
CH＝8
返回值：
DS：DX＝参数块的指针
注：不改变 DS

程序设计要点：

·有几个中断能提供重要的系统和设备信息。

第二十五章　其它中断

这章列了各种类型的中断——寻找活动字节、打印屏幕、确定 Control—break 状态和改变文件手柄号——它们不能归到其它种类中。这类中断非常有用。

活动字节计算当前正在进行的 DOS 功能调用的数目。它对内存驻留程序非常有用(见第 28 章的"使用 DOS 和多任务程序"部分)。

操作:寻找活动字节
版本:无文献记载。DOS 2.0—
中断号:21h
功能号:52
调用值:
　ES=活动字节的段
　BX=活动字节的偏移量
　注:ES 改变了。ES:BX 指向一个字节值。

操作:打印屏幕
　　与按下 shift—print screen 作用相同
版本:BIOS
中断号:5h
调用值:NA
返回值:屏幕被打印

DOS 的 BREAK ON 命令强迫 DOS 在执行任何 DOS 中断之前检查 Control—Break。这对断开程序很有用。在 DOS 4.0 里,这个中断增加了一个附加参数返回引导驱动器号。

操作:确定或设置 control—break 状态,检查引导驱动器。
版本:DOS 2.0—
　　检查引导驱动器为 DOS 4.0
中断号:21h
功能号:51
调用值:
　AH=51
　AL=命令码

值	含义
0	确定状态
1	设置状态

5　检查引导驱动器

DL＝状态(若设置状态)。OFF 为 0,ON 为 1。

返回值:

DL＝状态(若确定状态)。OFF 为 0,ON 为 1。

若检查引导驱动器,DL 返回引导驱动器。A＝1,B＝2,C＝3 等。

DOS 3.3,一次允许 20 个以上的文件手柄同时活动。若增加文件手柄,在程序启动时一定要释放所有不用的存储器,象在第 20 章"常规存储器"部分讨论的一样。这样就给了 DOS 新手柄空间。可让 DOS 同时访问到 65,535 个手柄。若把手柄数降低到值 n,那么在这些手柄关闭之前,所有 n 以上的手柄都是无效的。

注:到 DOS3.3 为止,此中断有点不可靠,使用时要格外小心。

操作:改变允许的文件手柄数

版本:DOS 3.3—

中断号:21h

功能号:103

调用值:

AH＝103

AL＝0

BX＝允许打开的手柄数。若小于 20,DOS 就使用 20。

返回值:若有错进位位置 1。错误码返回到 AX 中。

程序设计要点:

- 活动字节对使用 DOS 调用的内存驻留实用程序非常有用。
- 设置 BREAK ON 使程序断开更容易。
- 允许 DOS 最多使用 65,535 个文件手柄。

第三部分
内存驻留实用程序

这部分将讨论内存驻留实用程序。从中你能详细了解中断是怎样工作的及如何重定向中断；观察驻留和不驻留部分及写键盘激活例行程序和弹出例行程序时的特殊细节；看到怎样使 DOS 和你的应用程序共存；还将观察检测、通讯、潜伏和卸载。整个第三部分都有共用的、性能良好的例行程序。

用内存驻留程序可大大增强计算机的功能。键盘增强程序、打印机假脱机程序、弹出记事簿、联机拼写检查程序都属于内存驻留程序。

第二十六章　内存驻留程序的组成

本章讨论内存驻留程序的基本组成。内存驻留程序也许是实用程序中最有意思和最重要的类型。它们允许计算机完全定制化，提供高功能的后台特征。不象一般程序，内存驻留程序一旦装入，就总存在计算机里。任何时候都可调用，甚至在另一程序正在运行时也可调用内存驻留程序。因此，如果你在电子数据报表运行中想记录点东西，就可用 Sidekick 草草记录在计算机的记事簿上。如果你正在运行的程序出错了，可用 FAILSAFE 检查，甚至如果该程序挂起了还可断开。在第 27 章的 PROTECT 程序，使计算机定制化，一旦装入，可使硬盘免遭重新格式化的破坏。

内存驻留程序通过重定向中断而起作用。当一个中断出现时，它们就取代被调用的中断服务程序，有时它们重定向一个中断——象每次按键时调用的中断——只是为了看看自己何时行动。例如，FAILSAFE 重定向键盘是为了看看是否已键入 Ctrl－Alt－Left Shift 键。有时内存驻留程序重定向中断以修改中断操作的方式。例，PROTECT 修改了 BIOS 和 DOS 磁盘访问中断，这样它们就不会格式化硬盘了。

内存驻留程序有很强的功能，但其在编程响应性上要求很多。多数应用程序和操作系统在设计时没有考虑到内存驻留程序，所以在设计内存驻留程序时要格外小心，别让其操作破坏其它程序的操作。还要注意，你的内存驻留程序要与其它内存驻留程序协调工作而又不会破坏它们。

程序设计时，若不注意与其它程序共存，其结果是可怕的：栈溢出、程序挂起、屏幕错乱、磁盘崩溃。本章和第 27、28 章，将告诉你怎样避免这点。

中断

每次激活一个中断——不管是通过硬件动作如击键还是通过软件 int 调用——都要调用一个特殊的机器语言程序。该程序的位置可在中断表里找到。中断表从偏移量 0000:0000h 开始。共有 256 个表项，每个表项包含与之对应的中断程序的偏移量和段。计算机引导后，中断表初始化成缺省值。如果想改变一个中断表项使之指向你写的程序，那么每次该中断激活时就调用你的程序。

在更详细地讨论中断之前，先看看中断表。中断表将使你对中断范围有更全面的了解。

计算中断表项地址的方法是用 4 乘以中断号。记住表项的第一个字是偏移量，第二个字是中断程序的段。

中断号 (D)	中断号 (H)	机器	类型	中　断
0	00		H	除法溢出
1	01		S	单步
2	02		H	不可屏蔽中断
		PC	H	8087 动作

3	03		S	断点
4	04		S	算术溢出
5	05		IG	打印屏幕(由 9 调用)
		AT	H	边界异常(BIOS 不支持这种情况)
8	08		H	时钟滴答
		AT	H	lidt 异常(这种情况无 BIOS 支持)
9	09		H	击键
		AT	H	80287 段越限(这种情况无 BIOS 支持)
10	0A		H	I/O 通道动作
		AT	IG	由 71H 调用
11	0B		H	COM1 动作
		AT	H	COM2 动作
12	0C		H	COM2 动作
		jr	H	调制解调器动作
		AT	H	COM1 动作
13	0D	XT	H	硬盘动作
		jr	H	视频垂直回扫
		AT	H	LPT2 动作
		AT	H	80286 段越限(这种情况无 BIOS 支持)
14	0E		H	软盘动作
15	0F		H	LPT1 动作
16	10		S	BIOS 视频服务
17	11		S	BIOS 设备表
18	12		S	BIOS 存储器大小
19	13		S	BIOS 磁盘服务
20	14		S	BIOS 通讯服务
21	15		S	BIOS 盒式磁带服务
		AT		BIOS 控制杆,计时器,系统频率,其它 AT 扩展设备
22	16	AT	S	BIOS 键盘服务
23	17		S	BIOS 打印机服务
24	18		S	启动 ROM BASIC
25	19		IG	热引导(用 9 调用)
26	1A		S	BIOS 时间和日期
27	1B		IG	断开(由 9 调用)
28	1C		IG	时钟滴答(由 8 调用)
29	1D		T	视频控制参数
30	1E		T	磁盘控制参数
31	1F		T	ASCII128－255 的图形字符
32	20		S	DOS 程序结束

33	21		S	DOS 服务
34	22		S	跳到程序结束之后的程序
35	23		S	DOS 的 control－break 处理程序(由 21H 调用)
36	24		S	DOS 的致命错误处理程序
37	25		S	DOS 磁盘读
38	26		S	DOS 磁盘写
39	27		S	结束,驻留
40	28		H	DOS 空时调用
47	2F		S	多用途程序接口中断
51	33	鼠标器	S	鼠标器中断
64	40	XT,AT	S	安装硬盘时指向磁盘中断程序
65	41	XT,AT	T	硬盘参数
68	48	jr	T	ASCII0－127 的图形字符
72	48	jr	S	翻译键盘代码
73	49	jr	T	键盘翻译表
74	4A	AT	IG	报警(由 70H 调用)
96	60		S	可用
.	.		.	
.	.		.	
103	67	AT	S	
*		EMS	S	EMS 标准的扩展存储器管理中断
112	70	AT	H	实时时钟
113	71	AT	H	重定向的 IRQ2
117	75	AT	H	算术协处理器动作
118	76	AT	H	硬盘动作
128	80			保留给 BASIC
.	.		.	
.	.		.	
.	.		.	
240	F0			

H＝硬件　　IG＝由其它中断产生的中断

S＝软件　　T＝表

注:IG 中断也可由 int 指令直接调用。

中断的类型

从中断表中已经看到,中断有几种不同的类型。最重要的是**硬件中断**、**软件中断**和**表中断**。

硬件中断由一特定硬件动作初始化:键的击入、时钟的滴答或一些硬件问题。硬件中断有

两种：与外设有关的(8－fh；AT 机上还有 70h－77h)和与微处理器或非外设硬件问题有关的(0 和 2；AT 机上还有 5,8,9 和 dh)。与外设有关的中断叫做可屏蔽中断。非外设中断叫做非屏蔽中断。

硬件用屏蔽中断向微处理器发出中断请求(IRQ)。接到请求后，处理器停止其当前任务，执行中断表里该请求中断所指向的程序。由于同时可出现几个硬件中断(经常这样)，所以硬件中断有优先级。如果处理器当前正处理一个中断请求，那么它只能再接受比执行中的中断优先级更高的中断请求。

非屏蔽中断具有最高优先级。非屏蔽中断之后才是屏蔽中断(中断 8～fh)。AT 机上，屏蔽中断优先顺序是 8,9,70－77h,bh－fh(ah 用来向 70－77h 传送控制)。

屏蔽中断允许外设与处理器异步操作，处理器可向外设传送一条指令，然后继续其处理，而由外设处理该条指令。若需要中断，则初始化中断；因此，处理器不必不断地轮询外设看看它们是否需要某些类型的处理(例，看看是否击键了)，外围也仅在需要处理器信息时，向处理器发送一个信号。这样做节省了 CPU 的时间。

IBM PC 系列使用一个特殊芯片——8259——来处理外设中断。该芯片使用一位标志，屏蔽比当前处理中的中断优先级低的中断请求。中断请求被接受时要设置该标志，而中断处理结束后，该标志必须由软件复位。

软件中断没有优先级。它们由机器语言指令 int 调用。int 指令执行时，标志入栈；返回地址(在 int 语句的下一行)的段和偏移量也入栈；清除中断标志(屏蔽中断不能这样)，从中断表装入中断程序地址，并跳到该地址。这些动作相当于下列语句：

```
PUSHF
CLI
CALL DWORD PTR 0000:4*interrupt number
```

用 iret 指令退出中断程序，返回到调用程序，并把标志弹出栈。用 ret2 代替 iret 可修改标志。如果用 ret2，还可以做 sti。

表中断绝不能被软件或硬件调用，它们只指向重要信息表。

使用中断表

改变处理硬件和软件中断的方式可使功能大大增强。利用中断表确定你想改变的中断，如果你想增强一个中断的功能，使其不仅仅检查触发，还可查阅适当的参考信息。对硬件中断，看看 BIOS 源程序，可以清楚 IBM 是怎样处理中断的。然后，加到你的程序中。对软件中断，要知道它们的调用参数，返回值，及它们改变了什么，这样你的取代才会与之兼容。

若可能，用 DOS 功能调用 37 和 53 重定向中断(见第 23 章)。若做不到——例如，你是在一个内存驻留程序里重定向一个中断——用 cli，并接着直接设置该中断。每个中断地址占用 4 个字节，所以一个中断的表项在 4 乘以中断号处开始。

在本章和第 27、28 章，你将学到怎样改变和利用其中几个中断编写内存驻留程序。

内存驻留程序如何工作

内存驻留程序通过接管一个中断向量来工作。每次使用这个中断，内存驻留程序就检查特定的激活条件。如果条件符合，驻留程序就转为活动。若条件不符，中断就象正常情况时一样

活动。

内存驻留程序一旦装入就总驻留在内存里。它使用的内存其它任何程序都不能用。因此只要它所在的中断被激活，即使这时候有应用程序或其它内存驻留程序正在运行，此内存驻留程序也照样运行。

内存驻留程序主要有两部分：**不驻留部分和驻留部分**。不驻留部分包含所有装入和建立实用程序的代码。它检查实用程序是否已经安装，检查和建立系统配置信息，显示安装信息，重定向中断，然后退出，而代码驻留在机器里。

驻留部分完成的功能有：检查激活条件，条件符合，执行其特定程序，然后返回。通常它还需要调用它所取代的中断程序。对执行某些活动的实用程序，它需要检查或改变机器设置，检查系统和 DOS 状态，与其它中断交互。

为防止系统崩溃，驻留部分必须完全恢复它被调用之前的计算机状态。许多情况下，必须使用尽可能小的栈，活动状态保持尽可能短的时间。而且，代码必须是可再入的；也就是，它必须假定例行程序运行时激活中断被多次调用。

写内存驻留程序时要注意许多事情。例如，调试内存驻留程序非常困难，象 DEBUG 和 SYMDEB 这样的程序起不了什么使用。因此，最好有一个内存驻留调试程序或 FAILSAFE。为使调试过程更容易，每个变量之前要包括正文变量名。如下列：

```
x_position      db      'x_position'
                dw      ?
```

有了 FAILSAFE，很容易搜索这些正文信息，检查变量值。搜索可以在程序安装之后，而且处于活动状态时执行，或者也可以从一些其它应用程序中搜索。若变量不正确，可改变其以调试程序的其它部分。

有时可能打印出下列信息：

```
Now attempting to save file
```

或者

```
DOS active, can't save file
```

设计驻留程序时，要清晰，要使之具有良好的共存性。不要假设特定的硬件配置或改变 BIOS 变量。不要假定这一例行程序是唯一的驻留程序。也不能假设其为处理中断的第一个或最后一个程序，并且改变和检查系统状态时要特别小心。如果写一个键盘扩展程序，那么一定要考虑与热键的冲突，写弹出程序时，还需考虑多任务系统。

还有，设计程序时要注意到灵活性。

如果可能，把实用程序变成一个 COM 文件。实现这点的最简单方法是只使用一个段(组)，并且程序以 org 100h 开始。EXE 文件也可驻留，但是把不驻留部分放在最后和选择驻留程序大小很难。还有，记住，EXE 文件的 CS:0 指向程序开始，COM 文件中 CS:0 指向 PSP。

不驻留部分

不驻留部分在程序装入后不留在内存里。其功能是保证程序能被装入，装入实用程序，建立任何类型的配置变量。

把不驻留部分放在程序尾，从程序的第一行跳到不驻留部分。程序驻留在内存后，只有前面部分留在内存。因为不驻留部分在尾部，所以它不保留在内存里，这样就为其它实用程序和应用程序节省更多的内存空间。应用程序驻留后就不需要不驻留部分的内容了。

不驻留部分或安装部分的开始部分应当检查实用程序是否已经装入。一个实用程序不允许装入两次。每次装入，都会占用更多的内存。如果实用程序设计的好，装入两次后虽然中断调用将经过一不必要的长链，将使用更多的栈空间，但不会引起任何冲突。设计不好的程序如果装入两次就会出故障。

看实用程序是否已经装入要在驻留数据部分设置一个标识名。把名字放在跳到安装部分和通讯例行程序的语句后面(后面还要讨论)。名字后面放一个 0。如下列：

```
start:          jmp     install
communicate:    jmp     comm_routine
ame             db      'FAILSAFE',0
```

在结束安装例行程序之前，用一个用户定义中断(60h－67h)指向 communicate。后面是名字字段。看实用程序是否安装，把正在装入的实用程序的名字字段与由任何非 0、用户定义中断指向的语句之后的三个字节开始的数据进行比较。若匹配，那么实用程序已装入；否则，打印出错信息并停止。

在寻找匹配名字时，保留找到的最后一个未用的用户定义中断号。在建立名字指针时，将用到该中断。若没有未用的中断，就没有办法标志实用程序已经装入或与之通讯。打印一个警告的信息。

然后，打印程序正在装入的信息。这个信息有两个目的。第一，保证你知道装入了什么。第二，使你在参数有所改变之前 Ctrl－Break 程序。

接下来，检查一下你程序需要的没有改变的系统状态信息。例如，你的程序必须接受某些依赖机器类型——象算术协处理器或 DOS 版本——的例行程序，以确定安装部分的状态，并把结果存在一个驻留部分的变量里。如果你的程序只能在某些安装的硬件环境下运行——象扩展存储器板，那么你就要检查现有的硬件情况。若找不到这些信息，就打印出错信息并停止。

若有用户安装输入，象口令字，就在此部分处理。可包括读命令行参数或从磁盘或键盘接收输入。

读并保存你将重定向的中断的旧向量。链接和卸装时需要这些向量。

然后，释放环境拷贝，通过观察 PSP 和使用中断 21h 的释放存储器功能找到其位置。不论程序何时装入，都要做一次环境拷贝并送到程序中。这个环境拷贝对内存驻留程序没用，因为在程序装入后，环境有可能改变。删除环境拷贝为其它程序释放更多的内存。

把中断重定向到你的中断例行程序。用一个未用的用户定义中断指向 communicate jump (和名字字段)。然后，重定向软件中断。接着重定向硬件中断。如果要使用时钟中断，那么最后重定向它。以这样的次序重定向向量，就使中断例行程序，如链到时钟中断中的一个例行程序，在它支持的例行程序放进去之前激活的机会降低到最低。

最后，结束并驻留。通过结束中断传送一个返回代码，表明实用程序是否已成功装入。

例子：VIDEOTBL

下面将看一个简单的内存驻留程序。该程序置换了视频参数表，因为它只建立一个表，因此也就不需要什么复杂的驻留代码——只这张表本身而已。

这个程序很有用。许多非 IBM 图形板在图形方式或单色显示方式下，屏幕位置存在问题。屏幕或左或右或滚动没有头尾。通过改变表中适当的数字，可自动地调节你所有的特定图形板。不像 MODE 命令，该程序调节的参数总是有效的——不仅在 MODE 命令之后有效。

VIDEOTBL 有一简单的安装部分。没有必要保存旧表的位置，也不必看其是否已经装。

在看完 VIDEOTBL 之后，一定要看第 27 章的 PROTECT 的安装部分。

```
;VIDEOTBL.ASM
;
;This program replaces the video parameter table.  The new set of
;values is used to adjust for a particular graphics board or
;monitor. For example, the values used here are to adjust for a
;Tecmar Graphics Master Board.
;
;The table stays resident in memory.  It does not check to see if
;another table was previously entered.
;
;This program is set up to be a .COM file.
;


;--------------CONSTANTS--------------

TABLE_LEN       equ     64


;--------------BEGINNING--------------
code            segment
                assume cs:code,ds:code
                org     100h
video_table:
                jmp     set_int

;---------NEW VIDEO TABLE DATA--------

table:
                db      38h,28h,2dh,0ah,1fh,  6,19h,1ch   ;40 x 25
                db        2,  7,  6,  7,  0,  0,  0,  0
                db      71h,50h,5ah,0ah,1fh,  6,19h,1ch   ;80 x 25
                db        2,  7,  6,  7,  0,  0,  0,  0
                db      38h,28h,2ah,0ah,7fh,  6,64h,70h   ;graphics
                db        2,  1,  6,  7,  0,  0,  0,  0
                db      61h,50h,52h,0fh,19h,  6,19h,19h   ;mono
                db        2,0dh,0bh,0ch,  0,  0,  0,  0

;--------------MAIN CODE-------------

set_int:
                mov     ax,cs           ;set up segment registers
                mov     ds,ax
                lea     dx,table        ;set up interrupt to point
                mov     ah,37           ;to new video table
                mov     al,1dh
                int     21h
                mov     ah,49           ;terminate and stay resident
                mov     al,0            ;no error
                lea     dx,set_int      ;Keep everything up to
                mov     cl,4            ;the paragraph containing
                shr     dx,cl           ;the main code.
                inc     dx
                int     21h
code            ends
                end     video_table
```

驻留部分

驻留部分包含变量和中断处理程序部分。每个中断处理程序包含激活检查部分，与其它中断的链接部分，例行程序激活后的特别处理部分。

前面提到，驻留部分应当以一个跳到安装部分的 jump 指令开始，后面跟着 jump 到通讯例行程序的指令和变量部分。变量部分以一个正文标识符开始，标识符后跟着你想包括的任何版本信息。之后要建立内存空间来保存你想改变的中断的旧向量。随后是内部变量，最后是文本信息。这个次序并不很重要，但在调试时可使你较容易地找到信息。为使汇编更有效，变量部分必须在两个 jump 之后，或者在一分开的目标模块里，该模块里所有变量都描述为外部变量。

中断处理程序

中断向量重定向到叫做中断处理程序的例行程序。无论何时只要有一个重定向的中断激活，就调用一个这样的例行程序。

中断处理程序的第一部分是激活和链接部分。它们出现的次序依赖于应用程序。激活是确定内存驻留程序是否应当成为活动状态的过程。例如，一个弹出程序要检查一个特定的键组合。链接是调用被处理程序取代的中断的过程。链接很重要，因为它给那些链到此中断的其它中断处理程序一个激活的机会。另外，硬件中断经常要执行一些特殊硬件处理和 8259 的复位。如果有可能不用完全取代原始硬件操作就能增强原始中断功能，那么最好是扩充原中断处理程序功能并且链接到原处理程序上。

根据中断和应用程序的重定向，可在激活检查之前或之后进行链接。如果激活检查观察的是一个可能被原始中断修改的系统状态指示器，那么先链接。例如，重定向键盘中断让它检查，同时按下左和右 Shift 键。在使用中断 16h 检查此状态之前(15 章)，必须由键盘中断处理程序更新变量，因此必须在检查其状态之前与旧中断链接。

在其它情况下，激活不依赖于由重定向中断设置的一些状态而依赖于一些其它状态。这种情况下，就要在链接旧中断之前处理。先检查激活然后链接。例如，一个修改 DOS 的实用程序，在 DOS 每次删除一个文件时，把这个文件移到一个隐含目录下。重定向 21h，检查删除文件功能调用的寄存器。若找到，改变寄存器，指定一重命名调用，再与旧的中断 21h 链接。

还有新的处理程序总处于活动状态的情况。你必须确定是在处理之前还是处理之后链接或者就不链接。这常取决于是应用程序。例如，假设你已写了一个从几种非标准输入设备——象 homemade，三维游戏杆——接受输入的实用程序。把该设备的输入转化成光标击键，并放在一个特殊缓冲区里。接着重定向 16h，这样每次程序寻找一个键时，你都可以从特殊缓冲区中取一个光标击键送给它。

如果想使三维游戏杆的光标击键具有优先级，那么每次 int 16h 调用时，先看看在特殊缓冲区是否有任何击键等待。若有，则从中断返回特殊缓冲区中的下一个击键；若没有，则与旧中断 16h 调用链接。假设想使键盘具有较高优先级。这种情况，就先与中断 16h 链接(从功能 0 调用转到功能 1 调用)，若没有有效击键，则检查特殊缓冲区。

一般原则是，如果激活依赖一些由重定向的中断设置的硬件状态，就先链接后检查激活。如果激活依赖一些其它信息或寄存器，若应用程序需要就最后链接；若应用程序不需要最后链接，就先链接，因为这样共用性更好。

一般，重定向硬件中断，是为了检查一些类型的触发指示器的系统状态。在第 28 章，用键盘作为一个触发器。一般重定向软件中断，是为了修改中断操作的方式，这里，用寄存器来检查

激活。这两种中断重定向都可检测激活标志。

如果不符合触发条件，把控制传送给链接。

链接

链接时，一定要把所有寄存器和标志按照你正在应用的例行程序的要求设置。因为你正在模仿一个 int 调用，所以中断标志必须清除，并压入栈。调用的例行程序以一个 iret 结束。如果你想在与旧中断处理程序链接后要回控制，那么使用一个 call 调用：

```
pushf
cli
call    dword ptr cs:old_interrupt
```

反之，使用一个 jmp 指令：

```
restore system status
pop off any registers you have used
cli
jmp     dword ptr cs:old_interrupt
```

如果你的中断把一个软件功能调用与另一个连接，就象在删除文件的例子里一样，你可以重新发出中断而不链接。这就使其它所有实用程序有机会修改你的请求。如果重新发出中断，要保证例行程序是可重入的。

cli 和 sti

调用一个中断处理程序时，就清除了中断标志，这就意味着不会有其它的屏蔽中断出现。在不能遭到破坏的操作中间，例如中断向量的转换，不能动中断标志应当勿打扰。但是中断屏蔽这样长时间很危险，特别是在后台中正在使用一个外设——如调制解调器——时。一有可能就发出一个 sti 重新允许中断。

可重入

如果在处理程序结束之前，允许中断重新出现，则例行程序必须是可重入的。也就是例行程序可从开始部分再次调用，即使该例行程序当前正处于活动状态。

例如，假设你重定向键盘中断。并设定新的中断程序激活后，还可以从键盘读信息。这种情况下，在第一个键盘中断仍在处理时，键盘中断可被再次调用。甚至你的例行程序没有键盘中断调用，而另一个不同的中断处理程序，象一个基于时钟中断的例行程序，可进入活动状态并请求键盘输入。还有，在第一个键盘中断处理中，你可以按下一个键。只检查激活顺序是不够的，还要按下激活键，引起处理程序自动重复执行。

另一个例子，假设你扩充了磁盘写中断。在处理一个磁盘写中断时，可能就会有另一个写中断出现，该写中断也许来自一个内存驻留程序。因此，两个磁盘写中断协调工作至关重要。

总是假定你的例行程序是可重入的，即使它所取代的例行程序并不可重入，或者应用程序并不需要可重入地调用中断，DOS 的生产商没有假定可重入，这使得内存驻留程序的设计很难。

注意一下如果你的例行程序不可重入的情况下的一些例子。一个简单的例子，假设你重定向磁盘写中断，并临时性地在一个内部变量里存储扇区号。假设在第一个例行程序调用结束之前，该例行程序被再次调用。那么第一个调用的磁盘扇区号就会丢失，其中内容将设置成第二个调用的扇区号。这样当第二个调用结束，控制返回第一个调用时，磁盘的完整性就受到了破

坏。

另一个例子是，假设你写了一个弹出记事簿程序，启动条件是左、右 Shift 键同时按下。因为这是一个弹出程序，所以要保存背景屏幕才能在例行程序结束时恢复屏幕(见第 28 章)。假设你正往记事簿上打印信息，不小心同时按了两个 Shift 键。这样弹出例行程序被再次调用，这时应保存屏幕以便此次调用结束后恢复屏幕。但是这次保存的是正有记事簿的屏幕。而原始屏幕丢失了。

最后一个例子，假设你重定向了时钟中断。每次进入时钟中断，把所有寄存器存到栈中后开始处理。假设在某种条件下处理要用很长时间，并且在下次时钟中断之前没有结束。这样就会有更多的寄存器压入栈。最后，将导致栈溢出。

在第一、二个例子中，程序出错是因为存储有重要信息的变量被覆盖。第二、三个例子，程序应当检查优先级高的激活。

使例行程序具有可再入性有两种途径。速度很快的短程序采用第一种方法最好——即绝不用变量存储信息，只使用寄存器。例行程序开始时，一定要先保存寄存器，结束后再恢复。PROTECT 和 BIOS 就使用这种方法。

长一些的程序，特别是硬件中断处理程序，象弹出例行程序，采用第二种方法最好——即设一个内部状态变量，指明中断程序何时是不活动态。如果程序是活动的，立即用一个 jmp 指令跳出中断程序，这不占用任何栈空间。如果例行程序是不活动的，把状态设置为 CHECKING 以防激活检查完成以前又发出中断。然后检查激活。适当的地方用一个 call 链接。再把状态改变成 ACTIVE 或 INACTIVE。

通过特别检查不活动性，可有许多指明不同活动类型的状态设置。下面几节将讨论这些状态设置。

下面是 FAILSAFE 程序中的一个例子：

```
new_kybd_9_int  proc   far
                cmp    cs:status,INACTIVE          ;can FAILSAFE be called?
                je     nk9_1                       ;yes
;................................
;
; FAILSAFE is already active.  Chain immediately.  Do not use
; up stack space with a pushf and call.  Do not return.
;
;................................
                jmp    dword ptr cs:old_kybd_9_int

;................................
;
; FAILSAFE can be activated.  Chain, then check for activation.
;
;................................
nk9_1:          mov    cs:status,CHECKING          ;indicate checking
                pushf                              ;prepare to chain
                call   dword ptr cs:old_kybd_9_int ;chain
                sti                                ;allow interrupts

;
; Check for activation here
;

;
; IF NOT ACTIVE:
;

                cli                                ;not active
                mov    cs:status,INACTIVE          ;reset status
                iret                               ;return
```

```
;
; IF ACTIVE:  set status to ACTIVE, begin processing
;
```

准备处理

一旦确定实用程序已被触发，则必须准备处理。首先要压入你想改变的寄存器。若你的例程将在一长时间内处于活动状态，那么暂时停用 Ctrl—Break 处理向量(16h 和 23h)，以免由于 Ctrl—Break 例行程序而引起想不到的挂起。

不要使用 DOS 功能停用这些中断。DOS 是不可重入的，可能使系统崩溃(见第 28 章"使用 DOS 和多任务程序")。必须直接访问中断表，但一定要先执行 cli，防止在新中断地址只进入一部分时出现外设中断。

```
;
; This segment is for accessing the interrupt table
;

int_table       segment     at 0h
                org         1bh*4
BIOS_CTRL_C     dw          ?,?
                org         23h*4
DOS_CTRL_C      dw          ?,?
int_table                   ends


;this part is in the normal code segment

old_bios_ctrl_c  dw     ?,?
old_dos_ctrl_c   dw     ?,?


;redirect to this routine to disable an interrupt

disable_int:    iret            ;don't do any processing if disabled


;this routine disables the control-break vectors

disable_breaks  proc    near
                push    ds      ;temporarily save these registers
                push    ax
                xor     ax,ax   ;ax <- 0
                assume  ds:int_table   ;point to int table
                mov     ds,ax
                cli             ;prevent interrupts from occurring
                mov     ax,BIOS_CTRL_C ;save old vectors
                mov     cs:old_bios_ctrl_c,ax
                mov     ax,BIOS_CTRL_C[2]
                mov     cs:old_bios_ctrl_c[2],ax
                mov     ax,DOS_CTRL_C
                mov     cs:old_dos_ctrl_c,ax
                mov     ax,DOS_CTRL_C[2]
                mov     cs:old_dos_ctrl_c[2],ax
                lea     ax,disable_int ;now disable them
                mov     BIOS_CTRL_C,ax
                mov     DOS_CTRL_C,ax
                mov     ax,cs
                mov     BIOS_CTRL_C[2],ax
                mov     DOS_CTRL_C[2],ax
                sti             ;allow interrupts
                pop     ax
                assume  ds:code ;or what it used to point to.
                                ;If unknown, assume nothing
```

```
                pop     ds
                ret
disable_breaks  endp
```

离开中断程序之前要恢复这些向量。

如果你的例行程序用了许多栈空间，那么你可能会想重定向栈。重定向栈要特别小心，而且尽可能不这样做，记住你的程序活动时，可能会有许多其它的中断程序处于激活状态。这样你必须生成一个足够大的栈。为安全起见，你程序中的栈至少要多开出 800 字节。要记录旧的 SS 和 SP 值，这些值必须存在变量里，而不是在栈里。如果需要多次进入切换栈的程序部分，那么必须有一个进入和退出次数的计数器。例行程序一进入，计数器加 1，离开之前，计数器减 1。入栈时，若计数器为 1(第一次入栈)，该栈变为新栈。出栈时，若计数器为 1(最后一次出栈)，该栈变为旧栈。记住切换栈前要先执行 cli。

转换栈是危险的，因为其它例行程序可能已把数据放入你的栈中。转换成原始栈时一定不要删除这些信息。进一步讲，其它例行程序也可能重定向该栈。不能把它们的栈复位成原始栈。另一方面，如果它们转换回你的栈，而你的栈是不活动的，那么一定要恢复成原始栈。如果这些做不到，就不要转换栈。

应用程序应当保留一个大栈，但有许多应用程序做不到，若可能，尽量减少栈的使用。如果写应用程序，就生成一个大栈。

最后，你的例行程序可能要修改和检查系统状态。例如，从屏幕读写的实用程序需要检查当前活动的屏幕的类型或页号(第 13 和 28 章)。若改变这个状态，那么一定要保存旧的状态，并在结束前恢复。只要可能就使用 BIOS 调用，避免直接对端口写。如果真的直接对端口写，那么一定要更新适当的 BIOS 变量。

解决热引导

重新引导系统有两种方式。一个是同时按下 Ctrl－Alt－Del 三键。这个引导过程要经过系统建立检查，重置低存储器，从磁盘重新装入引导记录，重新装入所有的隐含系统文件和 DOS 文件等等。这与关机再打开几乎一样的。有另一种方法是发布中断 19h，这与第一种方式有些相似，只是它跳过了系统检查步骤，从装入引导记录开始。因此不恢复低存储器变量。

一些实用程序重新配置系统信息标志，然后重新引导系统。这样 DOS 就装入了新的配置信息。这使你能够改变 DOS 看见的存储器的数目，并使实用程序驻留。

当系统以这种方式重新引导时，认为所有 DOS 可寻址存储器都是空的。DOS 重新装入和重新初始化中断表部分。DOS 版 2.0－3.0 版本重新预置中断 20h－27h，3.1 和 3.2 版本重新预置向量 0fh－3fh。IBMBIO.COM 重新预置中断 0fh。

因此，你的例行程序所在的存储器被释放，但是指向那里的中断向量并没有改变。为安全起见，必须重定向中断 19h——即重新引导中断。该中断出现时，必须恢复已改变的任何中断向量，包括重新引导中断，然后 jump 到旧的重新引导中断，这就保证在你的例行程序和其它重定向的中断例行程序，已从存储器清除之后，当其中的一个中断出现时将不会被调用。

由于还要重定向 Ctrl－Break 向量，因此必须直接访问中断表，恢复你保存的旧中断值，包

括重引导中断和通讯/名字标志中断。

通讯中断

在这章的前面部分，在 comm—routine 的例行程序里在名字标志符之前放了一个 jump，然后重定向了一个用户定义的中断指向该 jump。现在可用这个中断与一个外部程序的内存驻留程序通讯。

这个驻留程序建立不同的通讯功能，并且在通讯中断调用时，检查、处理这些功能。外部程序调用此通讯中断，从而使用这些功能。

首先，外部程序搜索指向该程序的用户定义中断。方法与检查实用程序是否安装时一样。然后，外部程序发出中断，在 AH 寄存器里为所需的功能编码。

例如，FAILSAFE 的通讯例行程序有报告状态，改变触发键，去活或重新激活 FAILSAFE 的功能。功能 0 报告状态特征。假设 FAILSAFE 用中断 62h 来通讯，则用下列语句就可得到状态：

```
mov    ah,0          ;get status of FAILSAFE
int    62h           ;communicate interrupt
```

为使你的例行程序能处理这些外部命令，要建立适当的 comm—routine，其应当检查有效功能请求，并做处理。例如，下面是 FAILSAFE 程序中的一段：

```
;--------------------COMMUNICATION INTERRUPT-----------------
;
; Redirected user definable interrupt for communicating with
; FAILSAFE.
;
; Called with:
;  AH = 0  Get status
;    Returns:
;    AH = status flag
;    AL = keyboard mask
;    ES:BX = pointer to version number
;
;  AH = 1  Set keyboard mask
;    AL = new keyboard mask
;
;  AH = 2  Deactivate
;
;  AH = 3  Activate
;

comm_routine   proc   near
               sti                     ;re-enable interrupts
               cmp    ah,0             ;get status?
               jne    c_r_1            ;nope
;....AH = 0 Get status
               mov    ah,cs:status     ;status
               mov    al,cs:key_mask   ;keyboard mask
               push   cs               ;segment of version number
               pop    es
               lea    bx,version       ;offset of version number
               jmp    c_r_iret         ;return
```

若不想要通讯功能，只需写一条 iret 指令：

```
comm_routine proc  near
             iret
comm_routine endp
```

处理部分

写处理部分要注意下面一些方面。首先，最大限度地减少栈的使用，以防栈溢出问题。还有，程序尽量短。最大限度地减少停用中断的时间。记住，即使中断可用，低优先级的中断在8259接受中断结束命令之前都是被拒绝的，所以如果在处理以前没有与硬件中断键接或打开了一后台通讯线打开，要特别注意时间。

避免使用只写端口。因为该端口的状态可能已由应用程序或一些其它中断服务程序设置了。若不能保证其先前状态，对其进行改变是危险的。

还要避免写BIOS变量，如键盘Shift——状态字节。因为其它程序可能会读它们或希望它们为一特殊格式。某些中断服务程序可能会改变标准BIOS的位置。例如，许多键盘扩充程序就使用了内部键盘缓冲区。直接往BIOS缓冲区写可能会无效或破坏性极大。只要有可能，就使用BIOS中断。

例如，许多键盘增强程序就使用了内部键盘缓冲区。直接往BIOS缓冲区写或无效或破坏性极大。只要有可能，就使用BIOS中断。

如果直接往一端口写，象6845芯片，一定要更新所有适当的BIOS变量。退出该中断时仅仅恢复状态是不够的。因为当你的中断服务程序活动时，其它已触发的中断处理程序可能会用到这些BIOS变量。

避免使用DOS功能，至少在你读28章之前不要使用。一般来说，DOS功能是不可重入的，而且会重写重要的DOS变量。其后果将会使从屏幕上出现的奇怪字符或系统完全挂起。

若正在执行重要的处理，象改变状态字节、BIOS变量或重定向中断，一定要先执行cli。这样做将防止其它中断打断这些处理。一可能就执行sti，使正常的中断处理能继续进行。

绝不要假定任何情况。不要假定当前正用显示屏幕的0页或能用其它页做暂时缓冲区；不要假定DOS给程序员使用的区域是空的。若需要一个暂时缓冲区，另外设置一个；如果往屏幕写，要先用一个BIOS调用检查活动页。

记住，同一时间内总可能有其它活动的实用程序存在，应用程序可能正在使用未标志的或正常为空的存储区。更进一步说，程序可能正在一台PC兼容机上运行，所以硬件结果和BIOS变量会有很大的不同；因此只要可能，就使用BIOS例行程序校验或改变系统状态。如果例行程序与机器紧密相关，那么在装入程序之前先检查机器。

退出中断处理程序

退出中断处理程序时，要注意以下几点。首先退出时，系统状态必须与进入时完全一样。如果改变了屏面方式或活动页，那么一定要复位。还有调色板，break状态，以及你所做的任何改变都要恢复原状。

然后，若改变了栈，则恢复它。还要恢复暂时重定向的中断。若使用了一个变量来指明活动或不活动状态，那么也要适当调整它。然后弹出你压入栈的寄存器并且iret退出。

如果重定向一个硬件中断，要保证8259芯片接收到了中断结束指令。若与旧中断链接，则假定8259已接收到了中断结束指令。若没有链接，一定要清除8259以备将来使用。这个步骤靠把20h发送到端口20h来完成。

若需要由标志返回信息，例如，伪造一个BIOS错误，如下做：

```
set the flags

sti
ret 2
```

若不需要，则用一个 iret 退出。

程序设计要点：

- 内存驻留程序装入后，留在存储器里，任何时候都可被激活。
- 内存驻留程序重定向中断。
- 中断调用由中断表指向的例行程序。
- 中断分为硬件、软件和表中断。硬件中断又分为可屏蔽中断和不可屏蔽中断。
- 可屏蔽中断具有优先级。
- 通过修改中断表允许定制中断特征。
- 内存驻留程序分为驻留部分和不驻留部分。
- 若可能，使驻留程序为 COM 文件。
- 不驻留部分放在程序尾。
- 要确认实用程序还没有装入。
- 打印安装信息。
- 检查未改变的系统信息。
- 保存旧向量。
- 释放环境。
- 设置通讯向量和重定向向量。
- 结束并驻留。
- 变量放在程序的开始。
- 中断处理程序包含激活检查，链接和处理部分。链接及激活检查的次序依赖于应用程序。
- 只要可能就使中断可用。
- 让中断处理程序可重入。或把所有数据存在寄存器中或保留一个状态字节。
- 若例行程序将活动很长一段时间，那么停用 Ctrl—Break 中断。直接处理表。
- 若计算机重引导则恢复所有向量。
- 使用通讯中断与驻留程序通讯。
- 尽量减少栈的使用。
- 尽量缩短中断停用或屏蔽的时间。
- 避免使用只写端口。
- 避免改变 BIOS 变量。
- 避免使用 DOS 功能。
- 不要假定一特定的系统配置或状态。
- 退出前恢复系统状态。
- 恢复所有的寄存器并清栈。

- 恢复任何暂时重定向的中断。
- 调整状态字节。

第二十七章　实例:PROTECT 程序

在这一章里,你将了解如何建立一个内存驻留实用程序 PROTECT,该程序保护硬盘以免被意外地重新格式化。如果你是一个硬盘用户,尤其是在处理办公室业务时,你应该输入并汇编 PROTECT 程序,它将使你免遭损失。

一旦安装了 PROTECT,你就不用担心你的硬盘被重新格式化。若你想重新格式化你的硬盘,用 FORMAT 命令时一开始硬盘会发出一阵噗噗声,然后将停止损害硬盘,显示出一个错误信息,并返回 DOS。若一个破坏性程序企图用 BIOS 调用做低级重新格式化,也将被阻止。格式化你的硬盘的唯一方法是对磁盘控制器进行直接编程。

PROTECT 将说明在第三部分中讨论的大部分问题。PROTECT 检查其是否装入,显示安装信息,并释放环境。如系统重新引导,它可重新进入并卸装,还将重定向几个中断。

如何进行格式化

在用使该程序前,你应首先对如何进行格式化有所了解。格式化有两种类型——高级和低级的。低级格式化是用软盘或破坏性损坏硬盘的程序所做的格式化。也就是除了一些扇区标志外将磁盘上每一个扇区的所有数据清掉。该过程是用 BIOS 调用中断 13h,功能 5 实现的。一旦用这种方法对磁盘进行了格式化,数据将无法恢复。

高级格式化是通过 DOS FORMAT 命令对硬盘所做的格式化。首先,检测整个磁盘搜寻坏扇区。这个过程需耗费时间并可听到磁盘的转动声。在这一过程的任何地方,你都可用 Ctrl—Break 或 Ctrl—Alt—Del 中断,磁盘将不会改变。但当 FORMAT 命令完成扫描磁盘后,就会重写引导记录、FAT、根目录。该过程将所有数据删除。使用 DOS 版本 3.2 和 3.3 时,当 DOS 调用 DOS 磁盘写中断写引导记录时,就开始破坏信息。

数据区中的扇区不会改变。若你很熟练,你就可以使用 EXPLORER 恢复大多数的文本文件。首先在根目录建立一个伪表项,表项的第一个字符为 e。然后,通过磁盘搜寻,用恢复删除文件功能将旧文件段加到一起。你也可以写一个程序,自动搜寻磁盘并打印出数据段位置。

PROTECT 如何保护磁盘

PROTECT 用一系列的方法保护磁盘不被格式化。首先,它重定向 BIOS 磁盘中断,中断 13h。若调用该中断要对硬盘做低级格式化,则请求将被拒绝并返回一个错误信息,这将阻止控制之外的程序对硬盘做低级格式化。

其次,PROTECT 将记录成功调用验证扇区中断的次数,若该数据大于一个常数,程序中设置为 10,那么 PROTECT 假定 FORMAT 命令正在对磁盘进行处理。PROTECT 就会通过发出 DOS 结束程序中断以阻止格式化进行。若执行了一个非检验命令,则清除计数器。

不止 FORMAT 命令,还有一些程序可能产生一连串成功的检验请求,本书作者还没有发现这种情况。COPY 和 XCOPY 都是单一地写和检验扇区。

最后,PROTECT 重定向 DOS 磁盘写中断,若调用写中断写引导记录时,请求将被拒绝并

返回一个错误信息。

PROTECT(代码)源程序

请注意定义中断列表的段以及环境指针的 PSP 定义。注意怎样使用常数以使代码更容易读。也请注意 org 100h,跳到安装部分,以及跳到通讯区部分的指令。审查 PROTECT 是如何检查是否已安装或如何打印出适当的信息。看看安装程序中的处理次序——打印信息,释放环境,设置标志,设置重定向中断,以及转换中断和结束。审查实际的重定向中断,了解如何检查激活、链接和结束。也请注意重定向的重新引导中断。

审查如何设置驻留和非驻留部分。注意怎样在非驻留部分显示过程信息以及错误信息,因为它们只用于程序安装过程中。

```
;PROTECT.ASM
;
;This program prevents fixed disks from being reformatted.  It replaces
;the BIOS and DOS disk access interrupt.
;
;There are two types of formats:  low level and high level.  PROTECT
;prevents low level formatting by redirecting any calls to the BIOS
;format command.  If it is for a hard disk, the routine returns an
;error message.  If it is for a floppy, the format can proceed.
;
;High level formatting is done by the FORMAT command.  The FORMAT command
;starts by verifying all disk sectors, then writes to the boot record,
;FAT, and directory structure.  With DOS 3.2 and 3.3, the first write
;is to the boot record using the DOS write interrupt.  With earlier
;versions of DOS, BIOS and DOS are circumvented and the disk is
;directly written.  PROTECT works against high level formatting in
;two ways.  First, it returns an error if an attempt is made to
;write to the boot record with the DOS write interrupt.  Also,
;PROTECT keeps a count of the number of successive verify calls.
;If verify (INT 13h, 4) is repeatedly called with no other disk
;calls in between, PROTECT assumes a format is taking place,
;and terminates the FORMAT command.
;
;This method is not necessarily foolproof, but I have not found
;any other commands the execute a successive set of verify calls.
;
;This program deinstalls when the system is rebooted.
;
;This program stays resident.  It is set up to be a .COM file.
;
;Interrupts 13h, 19h, and 26h are redirected.
;
;
;----------------CONSTANTS----------------

DISK_INTR           equ     13h
REBOOT_INTR         equ     19h
DOS_WRITE           equ     26h
FREE_INT1           equ     60h
FREE_INT_LAST       equ     67h
DISK_ENTRY          equ     DISK_INTR*4
REBOOT_ENTRY        equ     REBOOT_INTR*4
DOS_WRITE_ENTRY     equ     DOS_WRITE*4
FORMAT_CMD          equ     5
VERIFY_CMD          equ     4
WRITE_CMD           equ     3
C_DRIVE             equ     2
BAD_TRACK           equ     40h
BAD_TRACK_LO        equ     6
BAD_SECTOR          equ     10
FORE_COLOR          equ     3
```

```
RETURN          equ     0dh
LINEFEED        equ     0ah
TRUE            equ     1
FALSE           equ     0
VERIFY_TERM     equ     10



;-------------DUMMY SEGMENT---------------

;..............................
;
; Set up a dummy segment heading pointing to the interrupt
; table.
;
;..............................

int_table       segment at 0
int_table       ends



;---------------BEGINNING----------------

code            segment
                assume  cs:code
                org     44              ;here is the environment
environ_addr    dw      ?               ;pointer
                org     100h            ;make this a .COM file
protect:        jmp     install         ;install interrupt


;----------------DATA--------------------

communicate:    jmp     comm_routine
program_name    db      'PROTECT',0
copyright       db      '(C) Michael I. Hyman 1986 '
version         db      'V2.0',0
old_disk_intr   dw      ?,?
old_reboot_intr dw      ?,?
old_dos_write   dw      ?,?
ident_intr      db      DOS_WRITE
verify_count    db      0

IDENT_LENGTH    equ     OFFSET copyright - OFFSET program_name


;--------------NEW INTERRUPT ROUTINES-----


;..............................
;
; This routine replaces the BIOS disk interrupt.  If the interrupt call
; is to format a hard disk, it returns the track not found error.
;
; If the interrupt call is a verify command, then a counter
; is incremented.  If this counter is greater than a constant
; then a terminate program interrupt is generated, to stop
; the FORMAT command.
;
; If the call is not a verify call, the counter is cleared.
;
; Note that this routine is called with the command in AH and
; the dri  in DL.  DL will be greater than 80h for hard drives.
;
;..............................
```

```
new_disk_intr   proc    far
                cmp     ah,VERIFY_CMD       ;is it a verify command?
                je      is_verify           ;yes
                mov     cs:verify_count,0   ;no, clear counter
                cmp     ah,FORMAT_CMD       ;is it a format command?
                jne     do_old_int          ;nope, pass control
                test    dl,80h              ;is it a hard disk?
                jz      do_old_int          ;nope, pass control
form:           mov     ah,BAD_TRACK        ;yes, fake a bad track error
                stc                         ;indicate error
                sti                         ;allow interrupts
                ret     2                   ;return, clearing old flags

is_verify:      cmp     cs:verify_count,VERIFY_TERM   ;it was a verify
                                            ;call.  See if have done
                                            ;several successive
                                            ;verifies
                ja      kill_format         ;yes!
                inc     cs:verify_count     ;no, update counter
                jmp     do_old_int          ;pass control
kill_format:    mov     ah,76               ;looks like a FORMAT
                mov     al,4                ;better kill it.
                int     21h

do_old_int:     jmp     dword ptr cs:old_disk_intr
                                            ;pass control to old int

new_disk_intr   endp

;................................
;
; This routine replaces the DOS disk write interrupt.
; If it is used to change sector 0 (the boot record) then
; a bad track error is returned.  Otherwise, the control is
; passed along the interrupt chain.
;
; Note that this routine is called with the sector in DX and
; the drive in AL.  Flags must be left on the stack.
;
;................................

new_dos_write   proc    far
                cmp     dx,0                ;write to boot record?
                jne     do_old_write        ;no, do old interrupt
                cmp     al,C_DRIVE          ;Writing to the hard disk?
                jne     do_old_write        ;no, do old interrupt
                sti                         ;allow interrupts
                mov     ah,BAD_TRACK        ;fake a bad track error
                mov     al,BAD_TRACK_LO
                stc                         ;signal error
                ret                         ;exit, keep stack

do_old_write:   jmp     dword ptr cs:old_dos_write
                                            ;pass control to old int

new_dos_write   endp
```

```
;------------------------------
;
; The following uninstalls the redirected interrupts
; if there is a soft reboot.  This keeps the vectors from
; pointing to an area of memory that no longer contains
; program code.
;
;------------------------------

new_reboot_intr   proc    far
                  cli                           ;don't allow interrupts
                  push    ds                    ;save old values
                  push    bx
                  assume  ds:int_table          ;point to interrupt table
                  xor     bx,bx
                  mov     ds,bx
                  mov     bl,ident_intr             ;find marker interrupt
                  xor     bh,bh
                  shl     bh,1
                  shl     bh,1
                  mov     [bx],word ptr 0           ;clear it
                  mov     [bx+2],word ptr 0
                  mov     bx,cs:old_disk_intr       ;replace disk
                  mov     ds:DISK_ENTRY,bx          ;interrupt
                  mov     bx,cs:old_disk_intr+2
                  mov     ds:DISK_ENTRY+2,bx
                  mov     bx,cs:old_reboot_intr     ;replace reboot
                  mov     ds:REBOOT_ENTRY,bx        ;interrupt
                  mov     bx,cs:old_reboot_intr+2
                  mov     ds:REBOOT_ENTRY+2,bx
                  mov     bx,cs:old_dos_write       ;replace DOS write
                  mov     ds:DOS_WRITE_ENTRY,bx     ;interrupt
                  mov     bx,cs:old_dos_write+2
                  mov     ds:DOS_WRITE_ENTRY+2,bx
                  pop     bx                        ;restore registers
                  assume  ds:nothing
                  pop     ds
                  jmp     dword ptr cs:old_reboot_intr ;continue
new_reboot_intr   endp


;------------------------------------
;
; The following allows for communication between PROTECT
; and outside programs.
;
; No features are currently implemented.
;
;------------------------------------


comm_routine      proc    near
                  iret
comm_routine      endp
;---------------INSTALLATION SECTION-------------


install:          assume  ds:code
                  push    cs                ;set up segments
                  pop     ds
;------------------------------
;
;See if PROTECT is installed already
```

```
;
;.............................
                mov     al,FREE_INT1        ;scan the interrupt table
check_loop:     mov     ah,35h              ;free range for PROTECT
                int     21h                 ;marker
                cmp     bx,0                ;see if interrupt is in use
                jne     compare_strings     ;yes, see if it is marker
                mov     cx,es               ;now check segment
                cmp     cx,0
                jne     compare_strings     ;else, this vector is free
                mov     ident_intr,al       ;remember it
                jmp     cont_loop

compare_strings: mov    di,bx               ;es:di points to target
                add     di,3                ;don't look at jump
                lea     si,program_name     ;ds:si points to name
                mov     cx,IDENT_LENGTH     ;see if target points to
                repe    cmpsb               ;same string as name
                jcxz    already_installed   ;matched => installed
cont_loop:      inc     al                  ;look at next interrupt
                cmp     al,FREE_INT_LAST    ;time to stop?
                jne     check_loop          ;no, keep checking
                cmp     ident_intr,DOS_WRITE  ;a free interrupt?
                jne     not_installed       ;yes, so install it
;.............................
;
; No free room to install communications and identifier interrupt
; vector.  Print a message and install anyway.
;
; ident_intr is initialized to the DOS write interrupt.  Thus, if no
; marker is installed, the sections which redirect during installation
; and upon rebooting, will change the DOS write interrupt when modifying
; the identifier interrupt, then correct it when modifying the DOS write
; interrupt.
;
;.............................
                lea     dx,no_ident_warning
                call    disp_msg
                jmp     not_installed
;.............................
;
; Already installed, so print a message
; and terminate.
;
;.............................
already_installed: lea  dx,already_msg      ;display message
                call    disp_msg
                mov     al,1                ;indicate error
                mov     ah,4ch              ;terminate
                int     21h
;.............................
;
; Not installed, so continue
;
;.............................


not_installed:  lea     dx,start_up_msg     ;print message that
                call    disp_msg            ;protect is installed
;.............................
;
; Read old interrupts
;
;.............................
                mov     al,DISK_INTR        ;read old disk interrupt
                mov     ah,35h
                int     21h
                mov     old_disk_intr,bx    ;save it
```

```
                mov     old_disk_intr+2,es
                mov     al,REBOOT_INTR      ;read old reboot interrupt
                mov     ah,35h
                int     21h
                mov     old_reboot_intr,bx  ;save it
                mov     old_reboot_intr+2,es
                mov     al,DOS_WRITE
                mov     ah,35h
                int     21h
                mov     old_dos_write,bx
                mov     old_dos_write+2,es
;...............................
;
; Free environment
;
;...............................
                mov     ax,environ_addr     ;find environment
                mov     es,ax               ;free it
                mov     ah,73
                int     21h
;...............................
;
; Redirect interrupts
;
;...............................
                lea     dx,communicate      ;set up communication /
                mov     al,ident_intr       ;identifier interrupt
                mov     ah,25h
                int     21h
                lea     dx,new_disk_intr    ;redirect disk interrupt
                mov     al,DISK_INTR
                mov     ah,25h
                int     21h
                lea     dx,new_reboot_intr  ;redirect reboot interrupt
                mov     al,REBOOT_INTR
                mov     ah,25h
                int     21h
                lea     dx,new_dos_write    ;redirect DOS disk
                mov     al,DOS_WRITE        ;write interrupt
                mov     ah,25h
                int     21h

;...............................
;
; Now terminate and stay resident
;
;...............................
                lea     dx,install          ;keep everything up to
                mov     cl,4                ;install
                shr     dx,cl
                inc     dx
                mov     al,0                ;no error
                mov     ah,49               ;terminate
                int     21h

;-----------UTILITIES  (non-resident) ----------


;disp_msg
;     displays an ASCIIZ message
;     offset of message in DS:DX
;
disp_msg        proc    near
                push    ax
                push    bx
                push    si
                mov     bl,FORE_COLOR
                mov     si,dx               ;point to string
d_m_loop:       mov     al,[si]             ;load character
                cmp     al,0                ;check for end
                je      d_m_end
                mov     ah,0eh              ;print char service
```

```
                int     10h             ;print it
                inc     si
                jmp     d_m_loop
d_m_end:        pop     si
                pop     bx
                pop     ax
                ret
disp_msg        endp


;-----------------MESSAGES (non-resident)-------------

start_up_msg    db      LINEFEED,RETURN
                db      '----------------------------------------'
                db      LINEFEED,RETURN
                db      'PROTECT is now installed.',LINEFEED
                db      LINEFEED,RETURN
                db      'Your hard disks are safe from reformatting'
                db      LINEFEED,LINEFEED,RETURN
                db      'PROTECT  V2.0    ',LINEFEED,RETURN
                db      'Copyright (C) 1987, Michael I. Hyman'
                db      LINEFEED,RETURN
                db      '----------------------------------------'
                db      LINEFEED,LINEFEED,RETURN,0
already_msg     db      LINEFEED,RETURN
                db      '----------------------------------------'
                db      LINEFEED,RETURN
                db      'Sorry -- PROTECT was already installed.',LINEFEED
                db      LINEFEED,RETURN
                db      '----------------------------------------'
                db      LINEFEED,LINEFEED,RETURN,0
no_ident_warning db     LINEFEED,RETURN
                db      '----------------------------------------'
                db      LINEFEED,RETURN
                db      'Warning -- Many resident utilities are already'
                db      LINEFEED,RETURN
                db      'installed.  PROTECT will not warn you if you'
                db      LINEFEED,RETURN
                db      'install it twice.  Communications with PROTECT'
                db      LINEFEED,RETURN
                db      'will be limited.
                db      LINEFEED,RETURN
                db      '----------------------------------------'
                db      LINEFEED,LINEFEED,RETURN,0

code            ends
                end     protect
```

输入并汇编这段程序，修改输入错误，用 EXE2BIN 将其转变成 COM 文件。运行这个程序，试试再次装入。用 FAILSAFE 或 DEBUG 填充未用中断 60－67h。重新装入 PROTECT。

你可以试着格式化你的硬盘来测试 PTOTECT。应首先做备份拷贝以防输入错误(本书作者在硬盘上对该程序做过多次测试，一直运行良好)。若检验了 10 个扇区(硬盘转 10 圈)格式化没有停止，那么就中断运行并检查你的源程序。

程序设计要点：

- 软盘是低级格式化，硬盘是高级格式化。
- 高级格式化不破坏数据区中的信息。
- FORMAT 命令随不同的 DOS 版本而不同。总之，格式化硬盘时总是先执行一连串的检验调用。

第二十八章　程序协同操作及其它问题

在设计内存驻留实用程序中最重要的概念是协同操作。本章通过考虑其它程序及内存驻留实用程序的蕴含，提供帮助你开发可共存的应用程序软件包的信息。本章内容包括把键盘作为触发器使用；使用时钟中断的程序；写弹出程序；使用 DOS 和多任务程序，去活，卸载，以及 AT 机上存在的问题。最后是程序设计要点和一个小结。

把键盘作触发器使用

把键盘作为内存驻留程序的激活信号，是一个非常方便的办法。许多流行的内存驻留程序都用键盘来激活。例如，FAILSAFE 就是通过按 Ctrl－Left Shift－Alt 来激活的。键盘增强程序把许多键组合作为宏指令处理。

每击一次键，无论是字符，Shift 还是箭头键，就激活中断 9。正常(BIOS)程序检查键盘控制器，译码扫描码，若适当，则把一个字符码放到键盘缓冲区中。该中断还说明 Shift 状态，若缓冲区已满则发出嘟嘟声，检查 Control－Break，Control－Num Lock，暂停，打印屏幕，以及 Control－Alt－Delete。此外，还设置 BIOS 变量来指明某些键如 Shift 和 Control 是否已键入。中断 9 在退出前，复位键盘和中断控制器。

每次有一个键请求，就调用中断 16h。它返回字符的 ASCII 码和扫描码。

重定向键盘中断有两个原因。第一是为了检查一个特定的触发键或键组合。绝大多数弹出实用程序，如 FAILSAFE，都采用这种方法。触发键被称作热键。第二个原因是为了用不同的键盘解释程序代替键盘中断。你还可以扩展键盘缓冲区或重新组织键盘布局。另一个方法是(不常用)，只检查由中断处理程序的不同部分设置的内部变量，看看是否要激活。

检查触发键有两种方法。第一种是每次击键时都检查键入的键。这种方法依赖于中断 9，使实用程序能立即检查激活。如果已限定了激活键组合，用这种方法相当简单。

对于不是很紧急或需要检查许多触发器的程序，使用中断 16h 更好。可在每次应用程序或其它实用程序请求输入时，检查触发键。

检查使用中断 9 的触发键

按键时，键盘控制器产生一个扫描码，并把它放入一个可读的 I/O 端口。BIOS 中断 9 处理程序读该端口并把扫描码及对应的 ASCII 码放入键盘缓冲区。同时还设置指明 Shift，Control 和其它键的状态的位。

在此介绍三种测试触发键击键的方法。最简单的一种是限制触发器为 Insert，Caps Lock，Num Lock，Scroll Lock，Alt，Control，Ledt，Right Shift 键的组合。例如，Control－Alt，Left－Right，Alt－Scroll Lock 等等组合都是有效的触发器。

每击一个键，就检查状态字节看看是否符合触发组合。有两个字节指示状态，它们位于存储单元 0000:0417h 和 0000:0418h。

字节　　0000:0417h

位	十六进制值	含 义
0	01	键入右 Shift 键
1	02	键入左 Shift 键
2	04	键入 Control 键
3	08	键入 Alt 键
4	10	Scroll Lock 为 on
5	20	Num Lock 为 on
6	40	Caps Lock 为 on
7	80	Insert 为 on

字节 0000:0418h

位	十六进制值	含 义
3	08	Control Num Lock 为 on
4	10	键入 Scroll Lock
5	20	键入 Num Lock
6	40	键入 Caps Lock
7	80	键入 Insert

若你有 FAILSAFE,检查这些字节。激活 FAILSAFE 并用 Snoop 命令检查从 0000:0417h 开始的页。试验一下按下 Shift,Insert,Control 和其它键。与此同时,按下一箭头键更新屏幕。你将会看到状态字节的改变。

若你打算用这些状态位触发,首先链接旧的中断 9 向量,这样会保证状态位得到更新,还会顾及到任何键盘硬件处理。然后,用 BIOS 中断 16h,功能 2 检查状态(见第 15 章)。结果是将字节 0000:0417h 的内容返回到 AL 寄存器中。

但是这个过程使你局限于 Shift,Control 和 Alt 键的组合。若你需要处理第二个键盘状态字节,那么从内存读出这两个字节。

```
bios_variables      segment     at 0h
                    org         417h
keyboard_status     dw          ?
bios_variables      ends

                    xor         ax,ax    ;set to 0
                    push        ds
                    assume      ds:bios_variables
                    mov         ds,ax
                    mov         ax,keyboard_status
                    assume      ds:code
                    pop         ds
```

若 BIOS 版本很新,你就可以使用中断 16h,功能 18。返回信息的格式请参看第 15 章。

BIOS 方法比较好。因为 BIOS 变量单元随不同的 BIOS 版本而改变,并且一个与其同驻留的键盘中断处理程序可能置换或不更新键盘状态字节。无论哪一种情况,为了准确地返回键盘状态都要调节 BIOS 中断。

不要简单"与"掉与你无关的位,检查触发条件的剩余部分。这会引起许多键组合触发你的

程序。例如，如果你正寻找 Control－Right Shift，并且"与"掉了所有的其它位，那么 Control－Right Shift－Left Shift 键也将触发你的程序。只需屏蔽字节 0000:0417h 的位 4－7 和字节 0000:0418h 的位 4。

若两个共驻留程序使用相同的触发组合，并且是使用这种方法，那么先装入的程序先被激活。如果你的程序是后装入的，不要改变状态字节。让每个程序有其自己的激活次序。改变 BIOS 字节会有很大影响。另一不同的驻留程序在你改变它们之前可能已检查了这些字节，并且认为在你的程序操作后(或期间)，它们没有改变。

若你想用非 Shift 组合的热键或你需要几个热键，可使用另外一种方法。最好的方法是重定向中断 16h 并且观察中断流。

为了更快速的检查，则必须重定向中断 9。这样做有两种方法：检查中断 9 放入键盘缓冲区的键代码或直接读键盘控制器。这两种方法都存在问题。为了完满，这里将讨论读缓冲区，尽管与此相比重定向中断 16h 更安全。直接检查输入流更麻烦并依赖于机器相关。本书不推荐这种方法。

键盘缓冲区作为字的循环队列存在低内存中，一般从 0000:041eh 开始。循环队列意味着把缓冲区内存当作一个循环处理。字符不断地增加直到到达缓冲区内存的底部。然后，字符返转到开始。为了要记录在什么地方增加字符，确定缓冲区是否已满，设有两个指针。头指向要从缓冲区读出的下一个字符。这是字符信息的开始位置。尾指向下一个字符要增加到缓冲区中的位置。这是字符信息的结束位置。若头和尾指向同一个位置，则缓冲区为空。若尾正好指向头的前一个位置，则缓冲区已满。当尾超过缓冲的结尾时，就绕回到缓冲区的开始。

有四个指向键盘缓冲区的 BIOS 变量。它们均为偏移量与段 0040h 有关的字长度指针。你可以在段 0，起始偏移量 400h(0000:400h)处或在段 40h，起始偏移量为 0(0040:0000h)处访问这些变量。

键盘缓冲区指针

位置	含　义
0040:0080	指向键盘缓冲区内存区的起始地址
0040:0082	指向键盘缓冲区内存区的结束地址
0040:001ah	指向缓冲区头
0040:001ch	指向缓冲区尾

读刚刚键入的字符，需比较头和尾指针看看在缓冲区中是否一个字符在等待(若缓冲区中没有字符在等待，头和尾指针是相等的)。若缓冲区中有一个字符在等待，则读最近的一个字符，该字符存放在尾指针指向的字符前面的存储单元。记住，缓冲区是一个循环。若尾指针指向缓冲区的起始位置，要读的字符就是缓冲区结束前的一个字。字符以字方式存储，扫描码之后是 ASCII 值。

不幸的是，这种方法有三个不足之处。第一，它依赖一些 BIOS 变量位置。更重要的是它假定所使用的是由 BIOS 的指针指向的键盘缓冲区。因为这个缓冲区很小，所以大多数键盘增强程序取代了它，因此，存放在 BIOS 区中的数据就不一定是最近键入的字符。而且，在读字符的同时，其它驻留键盘处理程序可能正在修改 BIOS 数据。

你也可直接检查中断 9h 送去的扫描码。在 PC 机上，这个过程要求直接访问键盘控制器

芯片。在 AT 机上，该过程就很容易了。每次中断 9h 从键盘控制器接收一个扫描码，就将扫描码放在 AL 寄存器中，设置进位标志，并调用中断 15h，功能 79。若返回时清除了进位标志，就删除了扫描码。

通常，中断 15h，功能 79 只是简单地返回。可重定向该功能，当它往键盘控制器送扫描码时，查看扫描码。若符合触发组合，或清除进位标志以删除扫描码或改变 AL 以修改返回的击键。

检查使用中断 16h 的触发键

若你需使用许多不同的触发键，最好的办法就是重定向中断 16h。如果你正写一个键盘增强程序，想用其它文本字符串取代某些键组合，例如 Alt－A，则这种方法特别有用。若你有许多不同类型的例行程序要运行并且每个程序都有不同的触发器，用这种方法也比较好。例如，用 Alt－C 弹出一个计算器，用 Alt－W 弹出一个表。

这种方法并不是说一链接上中断 9h 就立即起作用。在一些其它程序正在请求输入时，如果键入了你程序的触发键，那么你的程序将被激活，这种事情是相当普遍的。你的程序将与 DOS，字处理程序和电子表格一起协调工作，但是，若一个程序处于一些紧急运算中时，就不能调用你的程序。

若你有某些需要立即访问的功能，就把它们与状态字节相连（象前面讨论的那样）。若你有需要许多触发器的功能，特别是如果它们包含增强的输入功能，就将它们同中断 16h 相连。

与中断 16h 相连的程序在多任务操作系统下会运行得更好。

若调用的是中断 16 的功能 2，则只跳到旧中断 16h。若调用功能 0 和 1，则先调用旧中断，然后，检查其在 AX 寄存器中返回的字符码。若是你的热键，则开始处理，若不是，则忽略这个值（从中断返回）。

若你的程序请求输入，切记 要标记一个状态变量，使你的程序可重新装入。

记住，若触发了你的程序，正在调用的程序依然等待一个要返回的字符。不能将无用的信息返回放在 AX 寄存器中。或是等待另一次输入，或是按照你的应用程序适当调节 AX。

用这种方法，所有字符在到达调用程序之前都得到检查。此外，也使在你之前装入的程序有被激活的机会。通过记录以前键入的字符，还可以使用复杂激活码，例如所有的字。记住，调用程序也会看到以前的字符。

用中断 16h 设计一个宏程序是一个很好的办法。对每一个字符请求，都要检查一下看看来自一个扩展宏功能部分的字符（也即是替换触发键的文本信息）是否在你的内部宏功能缓冲区中。若有一个字符在等待，则通过 ax 将其返回到调用程序。若没有字符在等待，则链接旧的中断 16h 命令。这种方法唯一的缺点就是直接搜索 BIOS 键盘缓冲区的运行不正常的程序将不检测你的字符。

通过使用中断 16h，功能 5 可以解决这个问题。遗憾的是，并不是所有的 BIOS 版本都有这个功能。

下面是一个中断 16h 处理程序。

```
new_interrupt_16h       proc            far
                        pushf
                        cli
```

```
                call    dword ptr cs:old_interrupt_16h
                cmp     ax,TRIGGER
                jne     no_trigger

        start processing section here
        then return

no_trigger:     iret
new_interrupt_16h   endp
```

与其它键盘触发程序共存

对单一的键组合数有一定的限制，但使用它们却有许多可能性。不要假定你的程序是唯一的键盘触发程序。要容许用一些方法来改变触发命令。若你有几种触发特性，你就可以生成一个单独的程序来选择触发器命令并用通讯向量(comm—routine)来设置它们。若你只有几个触发器，则应允许从命令行中选择它们。

注意，如果装入了几个使用相同触发器的实用程序，每个程序的优先级依赖于每个程序使用的方法。

取代中断 9

为完成某些功能，如键盘的新格式，就需要取代中断 9。从键盘端口截取所有的扫描码，进行处理，并将它们放在 BIOS 键盘缓冲区里或放在内部键盘缓冲区中。

基本的处理就是截取扫描码，检查它们是中断代码还是释放代码，是字符码还是 Shift 码，并且进行适当的处理。要把字符码翻译成 ASCII 码并放在缓冲区里。若你使用一个内部缓冲区，你还必须重定向中断 16h，让它从该缓冲区中读字符。

记住，要清除键盘和中断控制器。若你打算设计一个复杂的键盘取代程序，首先要查阅 IBM BIOS 源程序。

使用时钟中断的程序

时钟中断是一个功能强大的中断。表现在三个方面：首先，它具有最高屏蔽优先级；其次，它被有规律地经常地调用；第三，它告诉你绝对的和已经过的时间。

时钟中断对于协调各中断非常有用。例如，你想让你的实用程序，特别是弹出程序在处理之前一直等待直到没有其它的中断处于活动。为此，在符合触发条件时，把一个变量设置为 TRIGGERED。让时钟中断检查这个变量。如果时钟中断检测到一个 TRIGGERED 变量，则检查系统是否准备运行实用程序。若是，把状态改为 ACTIVE 并开始处理。

你也可用时钟中断在一规定的时间之后再执行程序。例如，你可以在几分钟停用之后使屏幕变为空白或在某一特定时间弹出一个报警信息。

时钟中断的另一个用途是在后台运行处理程序。例如，你可以设计一个在后台奏出音乐的程序。每次时钟滴答，该程序就检查是否应当演奏下一个音符。若是，则建立声音发生电路图演奏下一个音符。

有两个时钟中断。每次时钟滴答就调用中断 8。一般每秒钟 18.2 次。中断 8 更新系统时钟，检查磁盘电机是否运行太久。若是，中断 8 就关掉电机。然后发出中断 1ch。在重新获得控制权之后，向中断控制器发出一个中断结束命令。

一秒钟调用中断 1ch 18.2 次。如果你的实用程序也想以这样的频率被调用，那么与中断 1ch 链接。记住，在你返回到中断 8 以前不要发送 EOI。

如果你的程序不以这样的频率被调用，就使用中断 8。先链接，后处理。

与时钟中断链接时，让例行程序检查激活的时间尽可能短。花在时钟中断的每一个周期也就是应用程序不接受的一个周期。

写弹出程序

激活时，弹出实用程序在屏幕上弹出窗口。结束时，弹出实用程序把屏幕恢复到其原始内容。它们是内存驻留的下拉式窗口。若你已装入 FAILSAFE，按下 Control－Shift－Alt，一个窗口就会弹出在屏幕上。写弹出程序时要小心。与其它中断程序相比，弹出程序控制计算机的时间比较长。在激活弹出程序前，一定不要破坏任何信息。一旦弹出程序激活，检查屏幕状态和类型，保存旧屏幕的信息。弹出程序结束后，要恢复原始的屏幕状态和内容。在多任务操作系统下操作时，还要注意另外一些事情。

何时弹出

在弹出一个窗口之前，要确认一下当前没有低优先级的中断处于活动态。这样可避免破坏重要的硬件服务，例如磁盘调用。让检查激活的例 行程序把状态标志设置成 TRIGGERED。用中断 8 检查此标志。找到后，检查服务寄存器(ISR)里的中断控制器。若没有其它中断正在处理就立即激活并把状态标志设置成 ACTIVE。

用下列语句检查中断控制器：

```
cli
mov             al,     11              ;check status of
out             20h,    al              ;interrupt controller
jmp short       $ + 2                   ;pause for AT
in              al,20h                  ;get result
sti
```

在 al 中返回的字节有一位是为活动的中断设置的：

位	含 义
0	中断 8 活动
1	中断 9 活动
2	中断 0ah 活动
3	中断 bh 活动
4	中断 0ch 活动
5	中断 0dh 活动
6	中断 0eh 活动
7	中断 0fh 活动

检查激活的处理程序在退出前应当把状态设置成 TRIGGERED。这样可防止中断 8 处理程序在其它处理程序结束之前就开始运行，从而节省栈的使用。

```
; end of interrupt handler that checks for activation
; pop registers
                cmp             cs:status,WAS_TRIGGERED
                je              new_int_x_1
```

```
                cli                                     ;shut int's off
                mov             cs:status,INACTIVE      ;clear status
                jmp             new_int_x_end
new_int_x_1:    cli                                     ;shut int's off
                mov             cs:status,TRIGGERED     ;triggered
new_int_x_end: iret
```

检查屏幕状态并保存屏幕

激活一个弹出窗口时，要用中断 10h，功能 15 确定屏幕方式和活动页。然后，保存旧的屏幕信息。使用 BIOS 调用，或直接访问视频存储器。对比较小的弹出窗口使用 BIOS。把光标在窗口将出现的屏幕上的每一个位置上移动，并读入字符和属性。

对大一些的屏幕，直接读视频存储器。这种方法的优点是速度更快，缺点是直接读视频内存需要额外的内存以便在图形屏幕下工作，而且还因为直接读显示内存不使用 BIOS 调用，所以这种方法的协作性不好。

内存中有一特殊区域用来存储屏幕上的字符。显示内存中的每个字对应于屏幕上的一个字符位置。第一个字节是字符码，第二个字节是属性(见第 14 章)。

字 0 存储屏幕左上角位置的信息。字 1 存贮在行 0，列 1 的字符信息。然后是行 0，列 2 的信息。第一列信息结束后，开始第二列的信息，以此下去直到第 80 列的信息。行 1，列 0 的信息在行 0，列 79 的信息之后；因此，特定字符位置信息的偏移量，可从下列公式得出：

(位置行×每行字符数+位置列)×2

乘以 2 是因为每字符位置有两个字节(一个字)。

单色适配器，正文信息从存储单元 b000:0000h 开始。彩色图形和增强图形适配器，存储单元依赖于显示页。40 列的显示页每页使用 1000 字，80 列显示页每页使用 2000 字。第一页从存储单元 b800:0000h 开始，因此特定字符的信息，可按照下列方法得出。

单色屏幕，把上面计算的偏移量加上

b000:0000h;

图形正文屏幕，在上面计算的偏移量加上

b800:0000h+(页号)×50

因为正文信息是顺序存放的，所以用一个 movsw 命令或一个 movsw 命令的循环就可很容易读窗口之后的所有正文。

例如，读整个屏幕，可以这样：

```
;assumes that es points to data segment
;assumes that ds points to beginning location of screen buffer
;note--this is the reverse of the usual situation
          lea           di,our_buffer_location          ;point to where you
```

```
                                        ;will store info
mov         si,text_offset              ;as computed above
mov         cx,text_length              ;depends on # of
                                        ;columns on the
                                        ;screen

cld
rep         movsw
```

为保存图形屏幕的信息，必须读窗口后每一个象素的数据。使用的方法依赖于屏幕和方式，而且需要很多内存。还可做到转换成正文屏幕，而不考虑保存图形屏幕。

转换屏幕

你可能想把你的屏幕设计成 80 列屏幕或就是 80 列正文屏幕。若屏幕不是这些方式之一，在弹出任何窗口之前用中断 10h，功能 0 进行转换。单色屏幕不需这样做。若是彩色屏幕，则选择方式 2，即 80 列黑白正文屏幕。在 RGB 监视器上将出现彩色 80 列屏幕；在综合监视器上，则屏幕清晰。

注意，如果转换屏幕方式，将丢失旧屏幕的全部信息。这种损失有些可惜。如果在基于图形的窗口环境下写字符，可使用另外一种不同的方法。

往屏幕写

一旦已经保存了旧屏幕信息，并且转换了方式，就弹出你的窗口。使用 BIOS 调用或直接往内存写。BIOS 调用很慢，因为 BIOS 调用在写每个字符时有一个暂停，所以使用 BIOS 调用会引起一点闪烁。但是它们可移植性较好，并可在图形屏幕下工作。无论何时产生一个命令，如果这时你没有对屏幕做太多更新，使用 BIOS 调用可能是最简单的。

如果你经常更新屏幕的一大部分，那么直接往内存写直观效果更好。在读屏幕时要计算内存存储单元，然后往该存储单元写新字符信息。

为使显示更好，把所有信息先往内部缓冲区写。信息要显示时，立刻把所有信息都送到屏幕。

例如，假设在 80 列正文屏幕的下面 10 行弹出一个窗口。设置一个 10×80×2 字节的缓冲区存贮正文信息。要往窗口行 2，列 7(对应于屏幕行 18，列 7)写一个字符，就先写在缓冲区的〔(2×80)+7〕×2 字节中。在全部写操作结束时，用一个 rep movsw 命令把整个缓冲区拷贝到屏幕。记住，如果你的窗口不是全页宽的，必须用一系列的 rep movsw 命令拷贝并读正文。

若可能，包括一个移动窗口位置的命令。如果你使用一个内部屏幕缓冲区或使用 BIOS 调用，往屏幕写信息是最简单的。

准备退出

弹出程序结束时，一定要恢复原始方式和页，并拷回原始屏幕信息。恢复屏幕信息的过程与往屏幕写的过程是相同的。直接恢复或用 BIOS 调用恢复。

弹出程序与多任务处理环境

在多任务环境工作时必须特别注意弹出程序。主要问题是上下文转换。弹出程序开始执行时，屏幕上可能会有一个特别的应用程序。弹出程序保存此屏幕信息，但到弹出程序结束时，多任务程序可能已转换到一个不同的应用程序，这时屏幕可能就完全改变了。

每个多任务程序都提供一个不同的与弹出程序协调屏幕的方法。例如，一些多任务程序提供一个可供选择的的可寻址显示区，它们将处理直接写到这个区的任何信息。其它多任务程序则需要一个更复杂的处理过程。

使用 DOS 和多任务程序

DOS 调用提供了一个与计算机交互的有力方法。但是，它们是不可重入的，因此在内存驻留程序中使用时要格外小心。当只是 DOS 不忙时就触发程序是不行的。即使在 DOS 显示命令行提示符时，它还在用 DOS 调用等待字符串输入。如果你在不正确的时间调用了一个 DOS 的功能调用，DOS 将丢失应用程序的栈的记录或重写一个自己的栈，其后果是不堪设想的。

DOS 使用三个内部栈，PSP，DTA 和 FCB 栈，来存贮信息。DOS 功能 51，80，81，98 和 100 不使用内部栈。功能 89 使用第一个栈；功能 1－12 使用另一个栈，所有其它功能使用第三个栈。

所有 DOS 调用都通过中断 21h。当该中断被调用时，DOS 首先检查功能号是否是有效的，并在 PSP 和一个 DOS 变量里保存 SS 和 SP 寄存器；然后，DOS 保存 DOS 变量里的其它信息，决定使用哪一个栈。通过把 SP 设置成一个内部栈的开始地址来选择一个新栈。中断结束时，DOS 把栈恢复为应用程序的栈。当调用中断 21h 时 DOS 把 DOS 活动字节加 1(见第 25 章)，调用结束后，DOS 活动字节减 1。

在内部驻留程序中执行 DOS 调用时，要保证栈信息没有丢失，应用程序的 PSP，DTA 或 FCB 栈里的数据没被重写。

若可能，要完全避免 DOS 调用。把 DOS 屏幕或键盘调用改为 BIOS 调用。但是用 BIOS 调用进行磁盘访问是非常困难的。

为解决 DOS 重进入问题，需重定向中断 8h 和 28h。中断 28h 在 DOS 等待键盘输入时被 DOS 反复调用，此时，DOS 使用第二个栈，而用于 12 以上的功能调用的第三个栈是空的，因此，你可以执行任何 12 以上的 DOS 调用，包括磁盘调用。

重定向中断 8 以防应用偶尔使用 DOS 输入功能。设置一个状态变量告诉中断 8 你想执行 DOS 调用。当中断 8 检测到该标志时，它应当检查 DOS 活动字节。若该字节为 0，就表明没有未完成的 DOS 调用，这时就可以放心地执行任何 DOS 调用。在安装程序部分要找到活动字节的存储单元，并保存起来，以便在驻留部分使用。

你可能还要检查中断控制器状态寄存器，看看目前是否还有其它中断正在处理。因为 DOS 文件访问调用处理时间很长，所以如果有其它中断处于活动，要推迟 DOS 文件访问调用。

在做 DOS 调用之前，必须转换 PSP。首先，读入当前 PSP，然后把 PSP 转换成实用程序的 PSP。在退出之前恢复 PSP。如果使用了任何使用 DTA 的调用，象目录搜索调用，还必须转换 DTA。

多任务执行系统

多任务执行系统在程序间来回转换。在这段时间，它们几乎只是在中断向量表和 BIOS 变量之间换入、换出。

与多任务程序一起操作的主要问题是硬件中断不依附于一特定的上下文；因此，如果你从一个中断，象计时器或键盘中激活，你所在的上下文可能要转换。这将改变你的程序可能已检查过或计划去恢复的向量状态和系统变量。为解决这个问题，只能从硬件中断中检查触发条

件，然后设置一个标志，此标志可被经常调用的软件中断检查。你的程序在激活之前，必须等待下一个软件中断，但在这段时间里上下文将处于一个稳定的状态。

如果在一个多任务程序中检查 DOS 活动字节，那么在软件中断中也要检查该字节。

扩展存储器

如果你的程序使用扩展存储器，一定要保存和恢复页映象。参见第 20 章。

去活，卸载和 AT 机中要注意的问题

装载内存驻留程序后，你可能想装入另一个使用相似命令或重复某些特性的实用程序，或不想再使你的实用程序特性处于活动态，这时你应该暂时关掉它的操作。

为暂时关掉一个内存驻留程序，应当有去活和重新激活实用程序的方法。设置一个标志，指明实用程序当前是否是被去活的。如果你已经设置了一个状态变量，只再增加一个新的可能性即可：DORMANT。因为这样除了 INACTIVE 外还有一个值，所以你的实用程序将不再触发。让一个使用通讯中断的外部实用程序告诉实用程序应该潜伏还是重新激活。如果没有空中断可安装通讯中断，就扫描你重定向的一个中断的中断链，检查名字标识符的偏移量。从这个偏移量起，找到转向 comm—routine 的 jump。但这种方法非常乏味。

你可能想从内存中完全删除你的实用程序。这有一点难度。如果你的实用程序很小，去活是一个比较好的办法。为卸载一个实用程序，必须保存同旧中断的链接，还要释放尽可能多的空间。为释放空间，需要逐个地放置所有的中断处理程序，并让它们跳到主程序：

```
;old interrupt vectors stored here
old_int_x               dw              ?,?
old_int_y               dw              ?,?
        .
        .
        .
;interrupts redirected to this section.
;it takes up very little memory
new_int_x:      cmp     status,DEINSTALLED
                jne     new_int_x_routine
                jmp dword ptr cs:old_int_x
new_int_y:      cmp     status,DEINSTALLED
                jne     new_int_y_routine
                jmp dword ptr cs:old_int_y
        .
        .
;variables section here
        .
        .
;interrupt processing in this section
;it could take up a lot of memory
new_int_x_routine
        .
        .
        .
```

为卸载处理程序，把状态设置成 DEINSTALLED。用存储块修改中断，减小实用程序的内存空间，只留下 new—int—x 和 new—int—y 程序。

再有，这种方法只对大程序有用。如果程序很小，它所释放的存储块没必要再次利用。总之，任何在当前实用程序之后装入的实用程序都将是存在的，而且你的代码碎片仍然保留。

还可以通过恢复你已修改的所有中断向量来完全删除你的实用程序。这样做将删除所有

此卸载程序之后装入的程序。一旦你真的恢复了所有已修改的中断向量，你就能释放从 PSP 开始的所有内存。为此，跳到你的 PSP 里的偏移量 0。这个步骤非常危险。它假设如果实用程序在卸载程序之后装入，那么它们不能使用先前程序使用的任何中断。若它们使用其它中断，那些中断将指向未用的内存，而且系统将挂起。为安全起见，必须从最后装的实用程序开始卸载程序。

AT 机要注意的问题

编写用在 AT 机上的中断处理程序时，要注意那些编号小的中断，这些中断具有硬件和 IBM 分配的双重含义，如在第 26 章已列出的一样。例如，中断 5 既是打印屏幕中断，又是界限异常中断。BIOS 不支持任何界限异常的支持，只把它作为一个屏幕转储请求处理。如果写中断 5 处理程序，要注意这个问题。

还要注意其它有冲突的中断。

程序设计要点：

- 每次击键，都激活中断 9。
- 每次有一键请求就调用中断 16h。
- 键盘缓冲区是一循环队列。
- 重定向中断 9 允许立即检查热键，这对受限于 Control，Alt，和 Shift 键组合的热键是最好不过的。
- 在 AT 机上，在中断 9 处理扫描码时，你可检查、改变和删除。
- 如果想使用多种热键可重定向中断 16h。
- 用通讯程序改变热键。
- 时钟中断具有最高屏蔽优先级。
- 时钟对于协调中断和运行后台处理很有用。
- 若可能，使用中断 8，先链接。
- 时钟处理程序中的激活检查部分要简洁。
- 弹出一个窗口之前，要确认没有其它的屏蔽中断处于活动态。可把检查激活的程序放在时钟中断中。
- 在显示一个弹出窗口之前，要检查屏幕状态，并保存屏幕。
- 直接读屏幕或使用 BIOS 调用。
- 退出前恢复屏幕。
- 在一个多任务处理环境里，把处理部分与一个软件中断相连。若在屏幕上弹出窗口需要特殊处理。
- DOS 功能是不可重入的，因为 DOS 会重写内部变量，最重要的是会破坏栈信息。
- 重定向中断 8 和 28h 要使用 DOS 功能。
- 做 DOS 调用之前在中断 8 里检查 DOS 活动字节。
- 在中断 28h 被调用时，可以使用 12 以上的 DOS 功能。
- 检查 8259 确保进程中没有其它的中断。
- 暂时重定向 PSP，如果必要的话，还有 DTA。
- 如果实用程序使用扩展存储器，保存和恢复页映象。

- 用状态字节和通讯中断释放一个驻留程序。
- 用状态字节、通讯中断和减小存储块功能来卸载一个驻留程序。必须结构化程序使你能释放尽可能多的内存。
- 小心 AT 机上的冲突中断。

小结

协同操作是设计内存驻留程序最重要的部分。决不要假定任何情况。总是要考虑其它程序和内存驻留程序的隐含。遵从书中给出的指导，你能开发出介面更友好的实用程序包。

如果你的程序可以潜伏，如果你有可选项来选择热键，如果你检查何时激活，并且如果你遵从书中所有其它的指导，在绝大多数情况下，你的程序将运行良好，并具有较好的协调性。

[General Information]
书名=高级DOS程序设计
作者=
页数=1000
SS号=
DX号=
出版日期=
出版社=

书名
目录
目录