深入 DOS 编程

香港金山公司　　求伯君　主编

参与编写人员

雷　军　马贤亮　陈　波

王全国　　冯志宏

审　　校

马贤亮　雷　军

北京大学出版社

新登字（京）159号

内 容 简 介

本书全面、深入地剖析了最新MS-DOS 5.0的功能调用，包括未写入文档的功能调用；分析了各种DOS版本功能调用的差异；介绍了基于DOS的EMS、XMS、DPMI等先进技术，以及新版MOUSE驱动程序的强劲功能；给出了各种编程语言调用DOS功能的方法和丰富实例；书后还收录了程序员经常要查阅的各种图表。

全书深入浅出地论述了DOS编程的各种问题，适合于不同层次的程序员使用，以进一步了解整个DOS编程环境，突破语言的限制，充分利用机器资源，编写出高效、简洁、兼容的软件。

好的工具可以让你事半功倍；同样的，一本好的工具书，可以帮助你在知识的领域更上一层楼。

深入DOS编程
求伯君 主编
*
北京大学出版社出版
（北京大学校内）
双青印刷厂印刷
新华书店北京发行所发行 各地新华书店经售
*
787×1092毫米 16开本 27印张 648千字
1993年1月第一版 1993年1月第一次印刷
印数：0001—10000册
ISBN 7-301-02039-2/TP·168
定价：24.50元

前 言

这几年，我一直负责金山系列汉卡、金山汉字系统SPDOS和文字处理系统WPS的开发工作。在实践中我深深感到，优秀的软件必须适应于众多机型和各种运行环境，这就要求开发者掌握丰富全面的编程资料，以便设计出精巧完善、兼容性好的程序。有时，为了让一个功能能在各种版本的DOS上正确运行，我查阅了大量的参考书，却无法解决这个问题，后来只有深入分析DOS系统，才圆满解决。这样往往花费比较多的时间，得不到相应的效果。大家也许都有同感：搜寻资料最大的烦恼是，要么是资料难找，要么是资料太多、太杂，找不到合适的内容。于是我收集了大量的资料，根据自己的体验，筛选和编写成了这本《深入DOS编程》，希望它对广大的DOS程序员有所裨益。

本书奉献给大家的是有关DOS的难点、热点及常用资料。

所谓热点，就是剖析最具革命性的DOS 5.0版新增加的功能，如INT 21H功能4410H(查询IOCTL句柄)、INT 21H功能3306H(取MS-DOS版本号)等。最明显的是关于MOUSE驱动程序、XMS和DPMI的详细解说。

鼠标由于操作简单，易学好用，支持鼠标的软件越来越多，如Windows 3.0的所有图形界面均可用鼠标快速操作。国外的MOUSE驱动程序更新很快，提供的功能日趋丰富和强大。然而，对于开发者，支持MOUSE异常繁琐，而且国内至今没有周详的MOUSE资料，这就更令人头痛。

为了访问1M以上内存，Microsoft等公司联合制订了扩展内存规范EMS和扩充内存规范XMS。关于XMS，有些书略有提及，可惜都不太详细，无法指导编程。本书在说明这些规范后还给出了完整的例子程序。

另外，为了突破DOS 640K内存的限制，美国几大软件公司共同发布了DOS保护方式接口DPMI，Borland C++3.0就是通过DPMI技术实现的。DOS虽然有很多缺陷，但是一段时间内，DOS仍会占据相当大的市场，如何突破640K限制，在技术上已成为一大热点，本书详尽列出了DPMI的七十多种功能调用。

所谓难点，就是指在考虑程序兼容性的同时，常常要仔细比较DOS各版本间功能的差异。另外还有一部分未写入文档的DOS功能。

MS-DOS 5.0刚推向市场时，我们使用INT 25H（绝对磁盘读）碰到过这样的问题：硬盘是用DOS 5.0格式化的大硬盘（>33M），针对DOS 3.x编写的应用程序使用绝对磁盘读时，总是失败。后来经过分析，发现这是因为，DOS 5.0为了考虑大硬盘的读写，改动了调用接口。许多这类兼容性的问题都可以在本书中找到答案。

编写程序有时要用到DOS内部调用，这些内部调用为DOS本身所保留，有的调用随版本变动很大。比如BPB、DCB、DPB等内部数据的格式，在各版本中均有差异。调用时一定要小心。本书第二章就列出了这些内部数据结构。

至于常用资料，是指本书第四部分《DOS常用资料速查》。它并不是各种杂表的简单罗列，而是选其精要，比如BIOS功能调用、BIOS低地址含义、I/O端口功能表、CMOS数据格式等。

经过上述筛选，相信从有一些 DOS 编程经历的初学者到经验丰富的高级程序员都能从本书中获益。

由于时间仓促，书中的错误在所难免，真诚希望广大读者就本书的各个方面提出宝贵意见，以便我们再版时修正。

不少的程序员在 SPDOS 上做应用开发工作，苦于无详细的技术手册，无法利用 SPDOS 的强大功能。鉴于这个情况，不久，我们会出一本《金山汉字系统 SPDOS 程序员参考手册》。该书将详细介绍 SPDOS 新版本 1.0 具体的功能调用和各种语言编写的示范程序。新版本 1.0 版为了将来的发展，将使用全新的调用规范，会给你耳目一新的感觉。希望广大的 SPDOS 和 WPS 用户多提宝贵意见，让 SPDOS、WPS 更好地为大家服务。

求伯君

1992 年 10 月

绪论 DOS发展史

多年来，DOS一直是微机上的首选操作系统，其用户比任何其他操作系统都多。它是一个成熟完善的磁盘操作系统，有众多的工具和应用程序支持；同时，它又是一个广泛并且充满扩充性的环境，支持像80386，80486这样的处理器。将来的DOS可能还支持多任务和多用户操作。

DOS首先是由Seattle Computer Products (SCP)以86-DOS面世，开发者是Tim Paterson。当时，微机上的操作系统多是Digital Research的CP/M。为了方便CP/M上应用程序的移植，86-DOS着意作了这样的设计：其文件控制块结构和功能与CP/M相同，CP/M下的程序可方便地转换到86-DOS中。

86-DOS只在8088/8086 CPU芯片上工作，当时这种芯片刚刚问世，所以知道86-DOS的人不多；有些用户是在S-100系统中使用8086 CPU，他们发现从8位的8080/Z80标准和CP/M升级到86-DOS有很多好处，于是Seattle Computer Products为这几十个用户（其中有硬件制造商）建立了一个专门的支持小组。

恰在此时，Microsoft与SCP交涉，想为某个用户写一个专门的操作系统版本，但当时没有人知道正是IBM在寻求一个PC机上的操作系统。到1980年1月，等到Paterson知道这个客户名字的时候，Microsoft已取得了登记证，用它自己的名义在销售86-DOS。同年4月，Paterson离开了SCP，加盟Microsoft；尔后，他在Microsoft又花了几个月的时间完善此系统，以满足IBM的需要。

1981年7月，Microsoft从SCP买下86-DOS的所有版权。当IBM在1981年8月10日发表PC时，Microsoft已准备好了MS-DOS 1.0。

Paterson 1982年以后没有再直接参与DOS开发，但仍在PC舞台上活动，最近，他担任了Phoenix Technologies Inc.的顾问。

PC问世后，IBM选择CP/M-86和Softech的P-system作为PC操作系统的备用系统，DOS的地位并不突出，但CP/M-86和P-system支持的语言太少，销售乏力；相反，Microsoft因其支持的语言多而声名鹊起。

DOS有过多次正式的变化。在这一演进中，虽然有完善和更正程序错误的考虑，但主要是为了适应硬件的变化，尤其是磁盘驱动器格式和容量的变化。下表列出的是主要的DOS版本、发表时间及比较重大的变动。

版本	日期	说明
86-DOS	1980.8	SCP版本，作者是Tim Paterson
1.0	1981.8	支持早期PC，单面磁盘
1.1	1982.3	支持双面磁盘，在文件目录项中加入了日期和时间
1.25	1982.3	第一个OEM版本（ZDOS），加入了命令VERIFY
2.0	1983.3	支持PC/XT和硬盘
2.1	1983.10	支持IBM PCjr和PC便携机

续表

版本	日期	说明
3.0	1984.8	支持 PC AT 和高密磁盘
3.1	1985.3	支持网络
3.2	1985.12	增强了对新介质的支持
3.3	1987.4	支持 PS/2
4.0	1988.7	支持 32M 以上的硬盘驱动器和 EMS
5.0	1991.7	支持 XMS、上位存储块（UMBs）和 HMA

下面是 DOS 各版本的简单介绍及版本间的相互比较。

V1.0

V1.0 是原始 PC 机上的操作系统。它支持单面、8 扇区磁盘格式，提供各种基本的磁盘服务。相对于 CP/M，它有更加完善的磁盘目录结构，可管理文件属性，标记文件的准确大小。V1.0 又在 86-DOS 中加入更高级的磁盘分配和管理，更多的操作系统服务，还有启动初始化时 AUTOEXEC 批文件。此版本只由 IBM 销售。

V1.1

V1.1 更正了 V1.0 中的某些程序错误，支持双面驱动器，并可为文件标上时间和日期，后者为 DOS 最显著的特征之一。

V1.25

V1.25 是第一个由原始设备制造厂家（OEM）提供的 DOS 版本。

V1 远不是一个统一的标准。Microsoft 并未将其卖给最终用户，而只是准许 OEM 使用，并且 OEM 可修改它，甚至把它重新命名。如 Heath-Zenith 在 1982 年 3 月将其命名为 ZDOS。

V2.0

V2.0 支持可用在 PCjr 上的单面、双面 9 扇区软盘、硬盘和盒式磁带机，并大大增强了 DOS 服务，还加入了类似于 UNIX 的分级文件系统。下面是 V2.0 中比较重大的变化：

- ●文件句柄
- ●I/O 重定向
- ●管道
- ●过滤器
- ●假脱机打印
- ●卷标
- ●扩展的文件属性
- ●系统配置文件
- ●程序环境块维护
- ●程序动态内存维护
- ●支持用户制定命令处理程序
- ●支持多国语言

和 V1 一样，V2 只准许 OEM 使用。此时，大部分 OEM 意识到市场需要与 IBM 完全兼容的机器；这样，DOS 的变化范围就很窄了。有些制造商，比如生产 Model 2000 的 Tandy，为了与 IBM BIOS 兼容，干脆在其 BIOS 中提供了两套向量表。

V2.1

V2.1中，只是为了更好地使用IBM的PCjr和便携式PC，作了计时方面的改动。MS-DOS V2.11现在仍用在一部分机器上。

V3.0

DOS V3.0是为PC AT提供的最早版本。此版本支持高密（1.2M）磁盘和另外的硬盘格式，除此之外，还增加了支持网络磁盘的基本功能。下面是其中的新功能：

- ●应用程序控制打印假脱机程序
- ●扩充的错误报告
- ●建议的改正错误码
- ●支持文件和目录锁定

V3.0发表以后，OEM不再随意修改DOS结构。当然，在IBM和Microsoft的版本之间还是有些差异。如IBM将其大部分支持工具以COM文件提供，而Microsoft用的是EXE格式；另外还有很多代码上的差异。

为了支持网络操作，必须强化DOS结构标准，尤其是在DOS内部格式上。此时，OEM之间形成了默契，共同响应这一需要。至此，DOS关键部分已比较稳定。

V3.1

DOS V3.1增加了网络磁盘支持，包括文件共享，并更正了以前的一些程序的错误。

V3.2

V3.2支持3.5英寸软盘，它亦将格式控制纳入外围设备控制驱动程序中。V3.2是Microsoft以自己的名义销售给最终用户的第一个DOS版本。

V3.3

DOS V3.3是MS-DOS最为成熟的一个版本，其设备支持包括IBM的PS/2系列，它增加了两个用户命令（NLSFUNC和FASTOPEN）和两个新功能，并将DOS服务升级，特别地，它改正了V3.2中的一个严重错误。

V4.0

DOS V4.0增加了几个功能，强化了许多用户命令，并引入了图形用户Shell程序。最重大的变化是支持超过32M字节的硬盘分区，并引入了扩展驱动程序。

IBM V4.0发表后两个月，IBM又推出了一个更新版V4.01，更正了其中的一些错误，不过VER命令视其为4.0版。要区别以上两个版本，只有看文件SHARE.EXE的日期：V4.01为08/03/88，V4.0为06/17/88，Microsoft发表V4要迟一些，MS-DOS V4.00相当于IBM的V4.01，不过，Microsoft对其V4.0作过一些修正后，也发表了V4.01。

V5.0

DOS V5.0支持扩充内存，强化了许多用户命令，也引入了一些新命令，如UNDELETE，UNFORMAT，MIRROR和支持鼠标的全屏幕文本编辑，在Shell程序中加入了任务切换器

API，DOS V5.0 调整了 DOS 内核，其占用内存比 V4 少；另外，DOS 现在可从 ROM 运行，不再需要 SHARE 支持 32M 以外的 DOS 部分。

Microsoft 推出 V4.0 是为了对抗 IBM 的 DOS V4.0，而 DOS V5.0 才是自 V3.3 以来的一次真正革命。

展望未来

随着 DOS 向前创新，它将不断提供新的服务和新的选择。Microsoft Windows 和 DESQview 之类的窗口化环境来临后，程序员可得到的服务更加完善。使用这些高级服务，必须在编程方便和程序高速运行之间权衡。不过，随着处理器速度的不断加快，对低级服务的需要将逐渐减小。只有系统程序员才需深入 DOS，Windows 或 DESQview，直接访问机器。这就是进步！

1992 年底已看到了 DOS 6.0 的测试版本。由于广大用户的需求，在将来很长一段时间内，Windows 和 Unix 不会取代 DOS 的地位，DOS 仍将是 PC 机上最重要的操作系统。

目 录

绪论 DOS 发展史 …… (1)

第一部分 DOS 编程基础

第一章 DOS 功能调用 …… (3)
1.1 DOS 功能调用索引 …… (3)
1.2 DOS 功能调用使用说明 …… (9)
1.3 DOS 功能调用详解 …… (13)

第二章 如何调用 DOS 功能 …… (163)
2.1 一般的 DOS 功能调用 …… (163)
2.2 未写入文档的 DOS 功能调用 …… (174)
2.3 关于 DOS 严重错误处理 …… (183)

第二部分 DOS 编程必备

第三章 系统内存管理技术 …… (187)
3.1 内存工作原理 …… (187)
3.2 内存管理 …… (188)
3.3 EMS 技术 …… (190)
3.4 使用扩展内存 …… (192)
3.5 XMS 技术 …… (197)
3.6 XMS 使用示范 …… (204)

第四章 EMS 功能调用 …… (210)
4.1 EMS 功能调用索引 …… (210)
4.2 EMS 功能调用详解 …… (211)

第五章 XMS 功能调用 …… (247)
5.1 XMS 功能调用索引 …… (247)
5.2 XMS 功能调用详解 …… (247)

第六章 AT 机 BIOS 功能 INT 15H …… (258)
6.1 BIOS 功能 INT 15H 列表 …… (258)
6.2 INT 15H 功能 86H－89H 详解 …… (258)
6.3 INT 15H 示范程序 …… (261)

第七章 MOUSE 功能调用 …… (265)

7.1 MOUSE 功能调用索引 …… (265)
7.2 MOUSE 功能调用详解 …… (265)
7.3 EGA 寄存器接口 …… (281)
7.4 如何使用 MOUSE 功能调用 …… (286)

第八章 识别程序运行环境 …… (291)
8.1 识别 CPU 类型 …… (291)
8.2 识别协处理器类型 …… (293)

第三部分 突破 DOS 编程

第九章 DOS 保护模式接口 DPMI …… (298)
9.1 DPMI 的引入 …… (298)
9.2 DPMI 功能详解 …… (299)
第十章 虚拟控制程序接口 VCPI …… (342)
10.1 VCPI 的引入 …… (342)
10.2 VCPI 概述 …… (343)
10.3 VCPI 功能调用详解 …… (345)
第十一章 任务切换 …… (353)
11.1 数据结构 …… (353)
11.2 通知功能 …… (356)
11.3 服务功能 …… (361)
第十二章 标准 TSR 识别技术 …… (366)
12.1 用户参数块 …… (366)
12.2 功能 00H—安装检查 …… (367)
12.3 功能 01H—返回用户参数指针 …… (368)

附录 DOS 常用资料速查

速查 A 英文制表 ASCII 码表 …… (372)
速查 B 中文线框区位码表 …… (372)
速查 C 键盘扫描码表 …… (373)
速查 D BIOS 数据区 …… (383)
速查 E BIOS 功能调用表 …… (389)
速查 F 系统功能调用表 …… (400)
速查 G I/O 端口功能表 …… (405)
速查 H CMOS 数据格式 …… (409)
速查 I ROM 信息 …… (411)
速查 J 显示器标准显示方式 …… (413)
速查 K 显示器字符属性/颜色代码 …… (414)
速查 L 磁盘分区表 …… (415)
速查 M 磁盘目录项格式 …… (416)
速查 N DOS 内存控制块 (MCB) 结构表 …… (417)
速查 O EXE 文件头信息 …… (417)

速查 P　FCB 结构表 …… (418)
速查 Q　文件属性字节 …… (418)
速查 R　程序段前缀 …… (419)
速查 S　设备驱动程序属性字 …… (420)

第一部分　DOS 编程基础

• MEMO •

第一部分 DOS 编程基础

第一章 DOS 功能调用

1.1 DOS 功能调用索引

表 1.1 提供了 DOS 功能的快速参考。表中所有数字都是十六进制的。加星号的中断或功能未写入文档。

表 1.1 DOS 功能表

INT	功能	子功能	用途	页号
20			终止程序	13
21	00		终止程序	14
	01		带回显的键盘输入	14
	02		显示输出	15
	03		辅助输入	15
	04		辅助输出	16
	05		打印机输出	16
	06		直接控制台 I/O	17
	07		直接 STDIN 输入	17
	08		STDIN 输入	18
	09		显示字符串	18
	0A		带缓冲区的 STDIN 输入	19
	0B		检查 STDIN 状态	20
	0C		清缓冲区并输入	20
	0D		复位磁盘	21
	0E		选择磁盘	21
	0F		打开文件（FCB）	22
	10		关闭文件（FCB）	23
	11		查找第一个目录项（FCB）	23
	12		查找下一个目录项（FCB）	24
	13		删除文件（FCB）	25
	14		读顺序文件（FCB）	25
	15		写顺序文件（FCB）	26
	16		创建文件（FCB）	27
	17		更改文件名（FCB）	27
	18		保留	/
	19		取缺省驱动器	28

续表

INT	功能	子功能	用途	页号
21	1A		设置 DTA 地址	29
	1B		取分配表信息	29
	1C		取指定驱动器的分配表信息	30
	1D		保留	/
	1E		保留	/
	1F		取缺省磁盘参数块	30
	20		保留	/
	21		随机文件读（FCB）	32
	22		随机文件写（FCB）	33
	23		取文件大小（FCB）	33
	24		置随机记录域（FCB）	34
	25		置中断向量	34
	26		创建 PSP	35
	27		随机块读（FCB）	36
	28		随机块写（FCB）	36
	29		分析出文件名	37
	2A		取系统日期	38
	2B		置系统日期	39
	2C		取系统时间	39
	2D		置系统时间	40
	2E		设置校验标志	40
	2F		取 DTA 地址	41
	30		取 DOS 版本号	41
	31		终止且驻留	42
	32		取驱动器参数块	42
	33	00	取 Ctrl-Break 标志	43
		01	设置 Ctrl-Break 标志	43
		05	取引导驱动器代码	44
		06	取 MS-DOS 真正版本号	44
	34		返回 InDOS 标志地址	44
	35		取中断向量	46
	36		取自由磁盘空间	46
	37	00 *	取开关字符	47
		01 *	置开关字符	47
		02 *	读设备可用性	48
		03 *	置设备可用性	48
	38		取/置国别信息	48
	39		创建子目录	51
	3A		删除子目录	51
	3B		设置目录	52
	3C		创建/截断文件（句柄）	52
	3D		打开文件（句柄）	53

续表

INT	功能	子功能	用途	页号
21	3E		关闭文件（句柄）	55
	3F		读文件或设备（句柄）	56
	40		写文件或设备（句柄）	56
	41		删除文件	57
	42		移动文件指针	57
	43	00	取文件属性	58
		01	置文件属性	59
	44		设备驱动程序控制（IOCTL）	60
		00	取设备信息	61
		01	设置设备信息	63
		02	设备 IOCTL 读	64
		03	设备 IOCTL 写	64
		04	块驱动程序 IOCTL 读	65
		05	块驱动程序 IOCTL 写	66
		06	取输入状态	66
		07	取输出状态	67
		08	块设备是否可更换	68
		09	块设备是本地/远程	68
		0A	句柄是本地/远程	69
		0B	设置共享重试次数	70
		0C	句柄通用 I/O 控制	71
		0D	块设备通用 I/O 控制	73
		0E	取逻辑驱动器映象	78
		0F	置逻辑驱动器映象	79
		10	查询 IOCTL 句柄	79
		11	查询 IOCTL 设备	80
	45		复制句柄	81
	46		强行复制句柄	81
	47		取当前目录	82
	48		分配内存	82
	49		释放已分配内存	83
	4A		修改内存分配	83
	4B	00	执行程序（EXEC）	84
		01 *	装入但不执行程序	84
		03	装入覆盖	/
		05 *	进入执行状态	/
	4C		带返回码终止	87
	4D		取返回码	87
	4E		搜索第一个匹配的文件	88
	4F		搜索下一个匹配的文件	89
	50		置 PSP 段地址	90
	51		取 PSP 段地址	91

续表

INT	功能	子功能	用途	页号
21	52 *		取磁盘参数表	91
	53 *		把 BPB 转换成 DPB	93
	54		取校验标志	95
	55 *		创建 PSP	95
	56		文件改名	95
	57	00	取文件的日期和时间	96
		01	置文件的日期和时间	97
	58	00	取内存分配策略	97
		01	置内存分配策略	98
		02	取 UMB 的 Link 状态	99
		03	置 UMB 的 Link 状态	100
	59		取扩充错误信息	100
	5A		创建唯一名字的文件	104
	5B		创建一个文件	105
	5C	00	设置文件访问锁定	106
		01	清除文件访问锁定	107
	5D	00 *	复制数据到 DOS 保存区	107
		06 *	取严重错误标志地址	108
		0A	设置错误数据值	108
	5E	00	读取机器名	109
		01 *	设置机器名	109
		02	设置网络打印机设置	110
		03	取网络打印机设置	110
	5F	02	取重定向表项	111
		03	置重定向表项	112
		04	取消重定向表项	113
	60 *		扩展路径名字符串	113
	61		保留	/
	62		取 PSP 地址	114
	63	00	取系统引导字节表	114
		01	设置/清除临时控制台标志	114
		02	取临时控制台标志值	115
	64 *		设置当前国别字节	115
	65		取国别扩充信息	115
		20	字符转换	117
		21	字符串转换	118
		22	ASCIIZ 字符串转换	118
	66	01	取全局代码页	118
		02	置全局代码页	119
	67		置句柄计数	119
	68		刷新缓冲区到磁盘	120
	69		保留	/

续表

INT	功能	子功能	用途	页号
21	6A *		分配内存	120
	6B		保留	/
	6C		扩展的文件打开/建立	120
22			终止地址	122
23			Ctrl-C 中断处理程序	123
24			严重错误处理程序	123
25			绝对磁盘读	126
26			绝对磁盘写	128
27			终止并驻留内存	128
28 *			DOS 可安全使用	129
29 *			快速输出字符	129
2A *			Microsoft 网络接口	129
2B			保留	/
2C			保留	/
2D			保留	/
2E *			原 Shell 程序装入	130
2F			多路服务中断	130
	01	00	打印安装检查	131
		01	把文件交给假脱机打印	131
		02	从打印队列中去掉文件	132
		03	取消打印队列中的全部文件	132
		04	暂停打印任务	132
		05	结束打印 HOLD 状态	133
		06	取打印机设备状态	133
2F	05 *		取外部严重错误处理程序安装状态	134
	06	00	取 ASSIGN. COM/ASSIGN. EXE 安装状态	134
	08 *	00	取 DRIVER. SYS 安装状态	134
	10	00	取 SHARE. EXE 安装状态	135
	11	00	取网络重定向器	135
	12	00 *	取 DOS 安装状态	136
		01 *	刷新文件	136
		02 *	取中断向量地址	136
		03 *	取 DOS 数据段地址	137
		04 *	设置标准路径分隔符	137
		05 *	输出一个字符	137
		06 *	调用严重错误处理程序	138
		07 *	移动磁盘缓冲区	138
		08 *	用户计数减一	138
		0C *	DOS IOCTL 打开文件	139
		0D *	为关闭文件而取日期和时间	139
		0E *	搜索缓冲区链	139
		10 *	寻找修改了的缓冲区，或执行时间延迟	140

续表

INT	功能	子功能	用途	页号
		11 *	ASCIIZ 文件名规范化	140
		12 *	确定 ASCIIZ 字符串长度	141
		13 *	大小写和国别变换	141
		14 *	比较 32 位数字	141
		16 *	取 DCB 地址	142
		17 *	取 LDT 地址	142
		18 *	取用户堆栈地址	143
		19 *	置 LDT 指针	143
		1A *	从路径名中读取驱动代码	143
		1B *	闰年调整	144
		1C *	从月初计算天数	144
		1D *	计算日期	144
		1E *	字符串比较	145
		1F *	初始化 LDT	145
		20 *	取 DCB 号	146
		21 *	扩展 ASCIIZ 路径名	146
		22 *	转换扩充错误代码	147
		24 *	执行延迟	147
		25 *	取 ASCIIZ 字符串长度	147
		26 *	打开文件	148
		27 *	关闭文件	148
		28 *	定位文件指针	149
		29 *	读文件	149
		2B *	IOCTL 接口	150
		2D *	取扩充错误代码	151
		2F *	存贮 DX	151
	14	00	取 NLSFUNC 安装状态	151
	15 *		取 CDROM 安装状态	152
	16	80	MS-DOS 空调用	152
	1A	00	取 ANSI.SYS 安装状态	152
	43	00	取 XMS 驱动程序安装状态	153
		10	取 XMS 驱动程序入口地址	153
	48	00	取 DOSKEY.COM 安装状态	153
		10	读命令行	154
	4B	01	建立通知链	154
		02	检测切换器	155
		03	给切换器分配一标识（ID）	155
		04	释放切换器标识（ID）	156
		05	识别立即数	156
	AD	80	取 KEYB.COM 的版本号	157
		81	取 KEYB.COM 活动代码页	157
		82	置 KEYB.COM 国别标志	157

续表

INT	功能	子功能	用途	页号
		83	取 KEYB.COM 国别标志	158
	B0	00	取 GRAFTABL.COM 安装状态	158
	B7	00	检测 APPEND 是否安装	158
		02	读取 APPEND 版本号	158
		04	取 APPEND 路径指针	159
		06	取 APPEND 功能状态	159
		07	置 APPEND 功能状态	160
		11	设置创建或打开文件时返回文件名的状态	160

注意：未写入文档（保留）的功能注有星号（*）。在“用途”栏中标注为保留的，本书未予解释。

1.2 DOS 功能调用使用说明

DOS 服务指范围在 20H—2FH 之间的中断。在深入 DOS 服务详细说明之前，必须先说明一些事项，使读者对如何使用这些服务有充分的了解。

1.2.1 怎样调用 DOS 服务

DOS 服务与 BIOS 服务都是通过软件中断调用。这些中断如何使用依赖于编程语言的习惯。

如果某一 DOS 中断有多种用途，需在中断调用前把功能号装入 AH 寄存器，把子功能号装入 AL 寄存器。

除了中断号、功能号和子功能号之外，通常还需在 CPU 寄存器中提供专门的参数。寄存器的使用依据 DOS 服务调用接口和 DOS 版本而变化。

概括地说，调用 DOS 服务时，必须遵从以下几条基本步骤：

1. 把用于 DOS 服务的参数装入特定的寄存器中；
2. 如果需要功能号，把它装入 AH；
3. 如果需要子功能号，把它装入 AL；
4. 调用 DOS 中断；
5. 最后，检查返回参数是否正确。

1.2.2 DOS 重入问题

由于 DOS 是单用户、单任务的系统，所以 DOS 服务不能重入，在 DOS 服务中不能再次调用 DOS 服务。例如，假设用户开发了一个中断驱动的系统并将其驻留内存。当用户软件在处理一个中断，而又发生了另一个同种中断时，该怎么办呢？如果 DOS 能重入，用户就可以简单地进行下去，在中断出现时分别处理就行了。然而这不可能，因为 DOS 不能重入。图 1.1 说明了这样做的后果。

请注意步骤 D。进程中已经有了某个 DOS 命令，一个新的中断要再次进入中断处理过程。这一中断过程的处理在步骤 I 完成，控制返回到 DOS 命令第一次调用时被中断的地方（步骤

J)。可是，在这一点上，步骤 C 中用过的所有 DOS 变量和堆栈位置都被步骤 F 和 G 改变了，结果，在试图返回 DOS 的初始调用点（步骤 B）时，实际上却回到了第二次调用点（步骤 E），程序失去了对系统的控制，造成系统死锁。

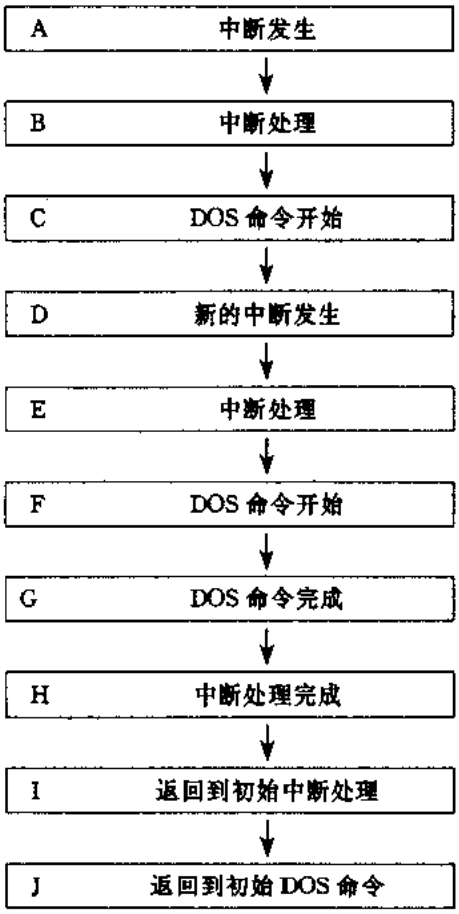


图 1.1 DOS 不允许重入的影响

1.2.3 保留功能

表 1.2 列出的功能由 IBM 和 Microsoft 保留。据 Tim Paterson 称，为了与旧的 CP/M 操作系统兼容，86-DOS 特意包括了表中的前四个功能，但 IBM 在其初始 DOS 版本中未提供它们的文档。为了与 V1 向后兼容，这些功能继续被保留。这四个功能通过 INT 21H 调用，后面的 DOS 功能调用详解中没有包括它们。功能 61H 存在的理由现在还不清楚，不过它使用和前四个功能一样的两指令代码序列。

表 1.2 保留功能

中断号	功能
21H	18H
	1DH
	1EH
	20H

续表

中断号	功能
	61H
2BH	
2CH	
2DH	

1.2.4 未写入文档的功能

IBM 和 Microsoft 未提供表 1.3 所列功能的文档，但编程者在后来总结出了它们的意义和用法。这些意义和用法是通过分析代码，并经过多次测试得到的。在 DOS 功能调用详解及后面的内容中说明了这些功能。不过要记住，由于它们未载入正式文档，IBM，Microsoft 或别的 DOS 厂商可以随时不加提示地修改它们。用户应该对其调用进行测试，并检查对于具体应用是否返回同样的结果。关于保留功能和未写入文档的功能，在后面还会有详细说明。

表 1.3 未写入文档的中断和功能

INT	功能	用途
21H	37H	取/置开关字符
	52H	取磁盘表
	53H	翻译 BPB
	55H	创建程序段前缀 PSP
	5DH	取严重错误标志地址
	60H	扩展到完整路径名
29H		快速写字符
2EH		执行命令

1.2.5 怎样介绍 DOS 服务

DOS 服务用一种标准格式介绍，所有服务都按中断及功能号增序编排。下面是描述格式的一个例子。

INT 21H	功能 66H	子功能 01H	V3.3
	取全局代码页		

每个功能表的第一行是中断号、功能号、子功能号（如果有的话）以及可用的 DOS 版本。第二行是该功能的用途。接下来是一个简洁的描述，调用寄存器说明，返回寄存器说明以及注释。下面介绍这种格式的每一项。

一、中断号

中断号用于调用某种服务。本部分涉及的中断由表 1.4 给出。

表 1.4 DOS 中断

中断号	用途
20H	程序终止
21H	多用途 DOS 中断
22H	终止地址
23H	Ctrl-C 中断向量
24H	严重错误向量
25H	绝对磁盘读
26H	绝对磁盘写
27H	结束且驻留内存
28H	键盘忙循环
29H	快速写字符
2AH	网络接口
2EH	执行命令
2FH	多路转接接口

二、功能号

功能号装在 AH 寄存器中，指定所需的 DOS 服务。功能号有多种选择，根据需要指定。例如 INT 21H 有 150 多种可用功能。除了前面介绍的 5 种“保留功能”以外，1.3 节包含了全部功能的说明。

三、子功能号

正如功能号一样，可以指定子功能号以进一步定义所需的 DOS 服务。只有个别 DOS 功能划分为若干子功能。如果所选的 DOS 功能要求说明子功能，在调用 DOS 中断前应把子功能号装入 AL 寄存器。如果没有子功能，AL 的使用可以不加限制，也可以用来向 DOS 服务传递其他参数。

四、支持该功能的 DOS 版本

用户需注意支持该功能的 DOS 版本，不要试图调用该 DOS 版本没有提供的功能。

五、用途

这一部分是 DOS 服务的一个简单描述，它简洁地概括了这种服务。大多数情况下，这些用途是从 DOS 的设计者——IBM 和 Microsoft 的技术出版物中逐渐总结出来的。个别情况下，他们列出的用途过于含糊或不准确，这里的叙述已作过一些修正，使得语义更加明确。

六、描述

描述是用一两句话对 DOS 服务所能完成的内容作一个简洁的说明。这和“用途”类似，只不过还简洁解释了服务范围。

七、调用寄存器

接下来是正确调用 DOS 功能所需设置的寄存器表。虽然它们可能是指向参数表的指针，但仍可以视其为参数。

八、返回寄存器

与“调用寄存器”部分一样，“返回寄存器”是 DOS 服务返回寄存器的列表。

九、注释

“注释”部分是服务表的主体。它叙述了这种功能做什么、如何使用、可能的应用以及应当注意的可能出现的曲解。若要用到参数表，则在此进行了讲解，或者给出了有助于确定

表中内容的信息。

1.3 DOS功能调用详解

INT 20H	终止程序	V1

终止程序的运行并把控制返回到父程序（通常是COMMAND.COM）

调用寄存器： 无

返回寄存器： 无

注释：

在DOS V1中，它是终止程序的标准方式，和功能00H完成相同的操作。在引入DOS功能4CH和31H之后，除了用于保持对DOS V1系统的兼容性，将不再推荐这种方式来终止程序。新的功能允许向高层程序和批文件返回出口代码。事实上，新版本的DOS通过把AX设置为4C00H并调用INT 21H来实现INT 20H。因此，在这些系统中使用本中断时，出口代码为00H。

Microsoft建议：除非要维护旧软件，最好不要使用该功能。该功能随时都可能不再被支持，应该用功能4CH取代。

除了终止程序并释放由程序占用的内存，该中断还进行以下工作：

1. 从程序段前缀中（偏移量0AH）恢复终止处理程序向量；
2. 从程序段前缀中（偏移量0EH）恢复Ctrl-C处理向量；
3. 在V2以上版本中，从程序段前缀中（偏移量12H）恢复严重错误处理程序向量；
4. 把DOS内部的文件缓冲区的内容刷新到磁盘上；
5. 释放程序占用的全部内存。

V2之前的DOS版本不执行第3步。上述几步完成之后，系统的控制递交到第1步中恢复的终止处理程序。

如果使用了FCB文件处理功能，以上过程还不够完整。使用FCB功能时，本功能不能关闭文件。虽然DOS文件缓冲区中的信息已被刷新到磁盘上，但目录信息并未做修改以反映文件的变化，应用程序的文件缓冲区也不能保证被刷新。只有使用FCB关闭文件功能（功能10H）才会正确地关闭文件、更新目录，并释放缓冲区给其他程序使用。由此可见，作为一种良好的编程风格，最好在使用本功能之前明确地关闭任何打开了的文件。

在EXE程序中调用该功能时一定要格外谨慎。CS寄存器必须指向程序段前缀。

完成了以上功能之后，系统控制返回到父程序，该父程序是通过EXEC（DOS功能4BH）来执行刚终止的子程序。一般情况下，父程序是COMMAND.COM，但也可以是任意其他程序。如果返回到DOS，控制就递交给COMMAND.COM的驻留内存部分，并进行内存测试以决定是否需要重新装入暂住部分。如果测试失败，暂住部分会被重新装入。最后，假如处理的是批文件，就要取批处理文件中下一行的执行。

INT 21H	功能 00H 终止程序	V1

终止程序的运行并把控制返回到父程序

调用寄存器： AH 00H

CS 程序段前缀（PSP）

返回寄存器： 无

注释：

本功能的操作与 INT 20H 一致，请参考 INT 20H 的注释。

INT 21H	功能 01H 带回显的键盘输入	V1

从键盘（从 DOS V2 开始是 STDIN）读一个字符并回显到显示器上（从 DOS V2 开始是 STDOUT）

调用寄存器： AH 01H

返回寄存器： AL 8 位的字符数据

注释：

这是一种最简单的惯用的键盘输入方法。该功能等待从键盘输入一个字符，并返回给程序。

DOS V1 的处理很简单，该功能从键盘取字符，而且仅在视屏显示器上显示。从 DOS V2 开始，这个过程随重定向的引入而变得复杂了。字符可以从标准输入设备（STDIN）中读取，可以在标准输出设备（STDOUT）上显示。一般情况下，STDIN 是键盘，STDOUT 是视屏显示器，但它们可被用户重定向。

如果 STDIN 中没有字符可取，该功能等待直到得到一个字符。如果 STDIN 已被重定向到非键盘的设备．输入是变化和分散的时候，就可能产生问题。

当得到字符并已显示时，本功能就返回其 ASCII 值。如果该字符是扩展 ASCII 字符，需调用本功能两次，第一次返回零，第二次返回所按键的扫描码。

在 STDIN 和 STDOUT 被重定向后，使用该功能时可能发生问题：

- 如果输入来自文件，可能返回与扩展键盘码不一致的零字节。
- 在 V4 以前的 DOS 版本中，该功能无法检测文件尾。

  对于 V4，输入重定向时，遇到文件尾会报告一个致命错误（Out of Data）。

当 STDIN 被重定向为文件时，这些原因可能导致严重错误。为此，用户可能要用其他 DOS 输入功能：06H、07H、08H 或 3FH（使用句柄 0，STDIN）。

使用本功能时，如果按下 Ctrl-C 或 Ctrl-Break，DOS 在返回前调用 INT 23H。

本功能可能对某些 Alt 组合键产生误解。

Microsoft 建议：除非要维护旧软件，最好不要使用该功能。该功能随时都可能不再被支持，应用功能 3FH 取代。

INT 21H	功能 02H 显示输出	V1

输出一字符到视屏显示器上（从 DOS V2 开始是 STDOUT）

调用寄存器： AH 02H

DL 8 位的字符数据

返回寄存器： 无

注释：

同大部分 INT 21H 的低编号 I/O 功能一样，该功能的使用取决于所用 DOS 的版本。在 DOS V1 下，该功能将字符直接输出到显示器上；从 DOS V2 开始，定向输出到标准设备（STDOUT），缺省时则是视屏显示器。

系统会正确地执行退格符，但不清除屏幕上的字符。在运行中如果检测到 Ctrl-C 或 Ctrl-Break，将调用 INT 23H 进行处理。

当输出被重定向时，该功能会导致问题。如果定向输出到文件，一个磁盘错误能使系统挂起，因为此功能没有检测或控制磁盘错误的内部方法（V4 以前）。V4 引入了错误控制，能迫使致命错误终止程序。请注意，此功能无返回值，因此输出字符时无法指明错误。由于这种原因，用户可能要使用其他 DOS 输出功能，诸如使用功能 40H（使用预定义句柄 1，STDOUT）。

Microsoft 建议：除非要维护旧软件，最好不要使用该功能。该功能随时都可能不再被支持，应用功能 40H 取代。

INT 21H	功能 03H 辅助输入	V1

从串行口（从 DOS V2 开始是 STDAUX）读一个字符

调用寄存器： AH 03H

返回寄存器： AL 来自 STDAUX 的 8 位输入数据

注释：

与键盘不同，串行设备不带缓冲区，一次只能处理一个字符。如果串行设备输入数据速度快于用户软件的处理速度，则会丢失字符。如果无字符可取，该功能则等待，直到获得一个字符才返回。

从 DOS V2 开始，本功能从标准辅助设备（STDAUX）中获取字符，缺省时是 COM1。在 IBM 的 DOS 版本中，COM1 的初始化为 2400bps，8 个数据位，无奇偶校验，一个停止位。可以使用 DOS MODE 命令来重定向 STDAUX，也可调用 BIOS 功能，或直接对硬件编程来设定数据格式。后一种方法已超出了本书的范围。

遗憾的是，无法通过这种 DOS 功能获得串行口状态信息。本功能既不能告知字符是处于等待状态还是已经丢失，也不能设定端口参数。要做与串口相关的任何重要事情，都至少要在 BIOS 级别（一般是在硬件一级通过专用的中断处理软件）来进行端口操作。

在本功能中允许 Ctrl-C 和 Ctrl-Break 处理。一旦检测出 Ctrl-C 或 Ctrl-Break，立即执行 INT 23H。

除此功能外，也可用功能 3FH（使用预定义句柄 3，STDAUX）来从串行口读取信息。

Microsoft 建议：除非要维护旧软件，最好不要使用该功能。该功能随时都可能不再被支持，应用功能 3FH 取代。

INT 21H	功能 04H 辅助输出	V1

输出一个字符到第一串行口（从 DOS V2 开始是 STDAUX）

调用寄存器： AH 04H

DL 输出到 STDAUX 的 8 位数据

返回寄存器： 无

注释：

本功能用来发送一个字符到串行口。从 DOS V2 开始，输出重定向到标准辅助设备（STDAUX），缺省时是 COM1。IBM 版本的 DOS 把 COM1 初始化为 2400bps，8 个数据位，无奇偶校验，1 个停止位。虽然其他 DOS 版本可能在缺省数据格式上不同，但都把 COM1 作为缺省的标准辅助设备。

如果在试图输出时 STDAUX 设备忙，本功能等待直到设备有空。因此，STDAUX 无法使用时，对此功能的调用极易使计算机挂起。本功能不能返回串行口状态信息，因此作用有限。要做任何关于串口的重要事情，必须至少在 BIOS 级别，而且一般要在硬件水平采用中断处理软件进行端口操作。

幸运的是，在该调用中，允许 Ctrl-C 和 Ctrl-Break 处理。一旦检测出 Ctrl-C 和 Ctrl-Break，就调用 INT 23H。借助截取 Ctrl-Break 处理程序，有可能使计算机从挂起状态中解脱出来，如处于等待根本无法获得的输入数据的状态，但是这很麻烦。最好（且对用户更加友好）用其他控制串口的方式编程。

用户也可以用功能 40H（使用预定义句柄 3，STDAUX）向 STDAUX 发送一个字符。

Microsoft 建议：涂非要维护旧软件，最好不要使用该功能。该功能随时都可能不再被支持，应用功能 40H 取代。

INT 21H	功能 05H 打印机输出	V1

输出一个字符到打印机（从 DOS V2 开始是 STDPRN）。

调用寄存器： AH 05H

DL 送给 STDPRN 准备打印的 8 位数据

返回寄存器： 无

注释：

本功能等到打印机准备好后发送一个字节。由于不返回打印机状态信息，在等待与系统没有相联或未准备好的打印机时，计算机可能挂起。可以采用BIOS打印功能（INT 17H）或功能40H（使用预定义句柄4）这些更好的方法。

在本功能中检测Ctrl-C和Ctrl-Break，检测到后执行INT 23H。

Microsoft建议：除非要维护旧软件，最好不要使用该功能。该功能随时都可能不再被支持，应用功能40H取代。

INT 21H	功能 06H	V1
	直接控制台I/O	

不经过DOS处理，直接读写控制台

调用寄存器： AH 06H
DL 功能请求
00H至0FEH，要输出的字符
0FFH，请求输入字符

返回寄存器： 如果输出字符，则无返回
如果输入字符：
若无字符可取，置零标志（ZF=1）
若有字符可取，清零标志（ZF=0）
AL 8位数据

注释：

本功能依据DL寄存器的设置来输入或输出字符。因为DL寄存器中的FFH表示输入，该功能显然不能用于输出FFH字符。

对用户来说，如果不能输出全部ASCII代码是一个欠缺的话，可以用功能3FH（使用预定义句柄1，STDIN）和功能40H（使用预定义句柄2，STDOUT）来完成同类的输入和输出。

有时，本功能被称作原始I/O操作。它不带回显地读字符；不对Ctrl-C和Ctrl-Break进行特殊处理，而是将其直接传递给调用程序，不转到中断处理程序。编辑器、字处理和别的程序因为需要解释所有击键，它们常使用该功能。

与功能01H一样，从键盘返回的是ASCII代码。如果该字符是扩展ASCII字符，需调用本功能两次，第一次返回零，第二次返回所按键的扫描码。这是仅有的能正确读出Alt组合键输入的DOS功能。

INT 21H	功能 07H	V1
	直接STDIN输入	

从标准输入设备STDIN读一个字符，不进行Ctrl-C中断处理

调用寄存器： AH 07H

返回寄存器： AL 8 位输入数据

注释：

本功能的输入操作与功能 01H 相似，不同的是字符不在视屏显示器上回显，而且不进行 Ctrl-C 或 Ctrl-Break 检查处理。在 DOS V1 中，字符仅能从键盘上读出，若字符未准备好，要等待到可以获取字符。在 DOS V2 及以上版本中，该功能从标准输入设备 STDIN 中读取，因而支持重定向。

本功能得到了字符后返回其 ASCII 值。如果字符是扩展 ASCII 字符，需调用本功能两次，第一次返回零，第二次返回所按键的扫描码。

该功能不显示字符。与直接 I/O（功能 06H）相似，这一功能不进行 Ctrl-C 和 Ctrl-Break 检查处理。若要求 Ctrl-C 和 Ctrl-Break 处理，应使用功能 08H。

INT 21H	功能 08H STDIN 输入	V1

从标准输入设备(STDIN)中读一个字符

调用寄存器： AH 08H

返回寄存器： AL 8 位输入数据

注释：

本功能的操作与功能 07H 最为相似，但它支持 Ctrl-C 和 Ctrl-Break 中断处理。

在 DOS V1 中，字符仅读自键盘。如果字符未准备好，该功能则要等待。在 DOS V2 及以上版本中，本功能从 STDIN 中读字符，因此支持重定向。

得到了字符后返回其 ASCII 值。如果是扩展 ASCII 字符，则返回 0 值，再次调用该功能返回所按键的扫描码。

该功能不显示字符，允许程序按需要控制该功能。若测出 Ctrl-C 或 Ctrl-Break，则执行 INT 23H。

与所有执行 Ctrl-C 检测的 DOS 键盘输入功能相似，本功能可能对某些 Alt 组合键输入产生误解。

INT 21H	功能 09H 显示字符串	V1

输出一个字符串到标准输出设备（STDOUT）

调用寄存器： AH 09H

DS：DX 指向以美元符（'$'，即 ASCII 码 24H）结尾的字符串

返回寄存器： 无

注释：

这是一经常使用的功能。本功能输出从某一给定的地址开始直到美元符的所有字符。

该功能处理的字符串与高级语言不同，它必须以美元符结束。C 字符串以 NUL 字符作终

结符，Pascal 和 BASIC 字符串有长度单元。由于无法输出美元符，这极大地限制了该功能的使用。用其他 DOS 输出功能编写一个与所用高级语言匹配的字符串输出过程，效果可能更好些。

Microsoft 建议：除非要维护旧软件，最好不要使用该功能。该功能随时都可能不再被支持，应用功能 40H 取代。

INT 21H	功能 0AH 带缓冲区的 STDIN 输入	V1

从标准输入设备 STDIN 中读取字符并放入用户指定的缓冲区中

调用寄存器： AH 0AH

DS：DX 指向输入缓冲区的指针，缓冲区结构如下：

字节 0	缓冲区能容纳的字节数
字节 1	所读的字节数
字节 2—?	返回的字符

返回寄存器： 无

注释：

带缓冲区的 STDIN 输入是一种有效而通用的功能，它提供了一般键盘输入操作的全部功能。输入取自 STDIN（缺省时是键盘）并放到用户定义的缓冲区中。键盘输入缓冲区必须由调用程序指定，其设置如表 1.5。

表 1.5 键盘输入缓冲区格式

字节偏移	内容
0	可读字节的最大数目
1	所读的字节数
2—?	来自键盘的实际字节

要使用此功能，只需简单地把允许输入的字节数存到 DS:DX 指向的缓冲区的首字节中。因为缓冲区至少要存放一回车符（ASCII 0DH），所以缓冲区最小为 1。若为 1，在使用时不会有任何键盘输入。实际的缓冲区至少为 2 字节(1 字节输入加回车)。缓冲区逻辑上限为 255，因为缓冲区长度说明仅有一个字节。

该功能从键盘上读入字节并将其放入缓冲区从第三个字节开始的地方。每个 ASCII 字符占缓冲区一个字节。当读到的实际字符数目等于缓冲区大小减一时，新的字符就被忽略而且每次击键都会响铃。当最后按下 ENTER 键时，所读的字节数被放入缓冲区的第二字节，控制返回到调用程序。

输入允许超前打字，而且支持全部键盘编辑命令。若检测到 Ctrl-C 和 Ctrl-Break，则执行 INT 23H。

请注意字符串的长度由缓冲区中的返回值确定，这个长度不包括回车结束。

本功能可能对某些 Alt 组合键产生误解。

Microsoft 建议：除非要维护旧软件，最好不要使用该功能。该功能随时都可能不再被支持，应用功能 3FH 取代。

INT 21H	功能 0BH 检查 STDIN 状态	V1

检查是否可以从标准输入设备（STDIN）读取字符

调用寄存器： AH 0BH

返回寄存器： AL FFH 可从 STDIN 获取字符

00H 不能从 STDIN 获取字符（V4 之前）

≠FFH 不能从 STDIN 获取字符（V4）

注释：

本功能检查是否可从 STDIN 获取字符。因为 STDIN 通常设为键盘，该功能一般用于确定键盘缓冲区中是否存在一个处于等待状态的击键。

调用时，本功能立即把状态返回给寄存器 AL，指出是否有一个字符等待输入。如果可以得到字符，AL 中是 FFH。请注意该功能并不返回实际字符，仅提供一种能否获取的提示。该功能连续返回同样的状态，直到一种 DOS 输入功能（01H，06H，07H，08H 或 0AH）来读取字符。

在该功能执行之中，如果检测出 Ctrl-C 或 Ctrl-Break，就调用 INT 23H。

INT 21H	功能 0CH 清缓冲区并输入	V1

清除标准设备 STDIN 缓冲区，然后执行指定的输入功能调用

调用寄存器： AH 0CH

AL 清除缓冲区后执行的功能号

01H 等待键盘输入

06H 直接控制台 I/O

DL=FFH 直接控制台输入

DL≠FFH 写到 STDOUT 的字符

07H 无回显直接控制台输入

08H 无回显控制台输入

返回寄存器： 返回由功能定义：

01H 等待键盘输入

AL，来自 STDIN 的字符

06H 直接控制台 I/O

ZF=1，从 STDIN 得不到字符

ZF=0，AL 等于来自 STDIN 的字符

07H 无回显的直接控制台输入

AL 来自 STDIN 的字符

08H 无回显的控制台输入

AL 来自 STDIN 的字符

注释：

在程序运行时经常会出现超前打字，编程者可使用该功能来防止因此而出现的错误，避免用户偶然打字超过程序输入点。格式化磁盘就是一个恰当的例子。因为这一操作要清除磁盘上的数据，所以程序要询问使用者是否确实要格式化磁盘。使用该功能来读取用户的回答，就可以阻止由于偶然超前打字读入错误的回答而引发问题。

Microsoft 目前声明 0AH 子功能被保留而不应使用。

INT 21H	功能 0DH	V1
	复位磁盘	

将磁盘缓冲区内容（如果已被修改）刷新到相应的磁盘文件中

调用寄存器： AH 0DH

返回寄存器： 无

注释：

该功能把磁盘缓冲区内容写到相应的磁盘文件中（刷新磁盘缓冲区）。此功能不更新磁盘目录，而且不关闭文件。它既不影响磁盘操作，也不复位任何其它磁盘参数。

在 3COM 网络中，当关闭所有文件时，本功能强行产生一个网络卷文件分配表（FAT）。

INT 21H	功能 0EH	V1
	选择磁盘	

改变缺省磁盘驱动器

调用寄存器： AH 0EH

DL 驱动器号（A=0 到 Z=25）

返回寄存器： AL 逻辑驱动器数

注释：

除了选择缺省驱动器之外，该功能可用于确定与系统相联的逻辑驱动器数。逻辑驱动器指系统中的块设备——RAM 磁盘、硬盘、磁盘仿真器等。

本功能通常返回最小值 2，表示存在两个逻辑驱动器（DOS 通常把单个物理软盘驱动器视为两个逻辑驱动器：A 和 B）。若需确定系统物理软盘驱动器的数目，可用 BIOS 功能 11H。

从 DOS V3 开始，该功能返回 CONFIG.SYS 中的 LASTDRIVE 值和实际逻辑驱动器数中较大的一个。如果仅有 3 个逻辑驱动器而 CONFIG.SYS 未说明 LASTDRIVE 值，就返回缺省 LASTDRIVE 值 5，所以返回值不一定对应于实际驱动器数。例如返回值是 5，并不指 A，

B，C，D 和 E 驱动器全都存在。

驱动器标志符的最大数目依据 DOS 版本而变化，如表 1.6 所示：

表 1.6 不同 DOS 版本下驱动器标志符的最大数目

DOS 版本	指定可获得数
1	16 (00—0FH)
2	63 (00—3FH)
3	26 (00—19H)

要与所有 DOS 版本兼容，使用时最多只能限定 16 个驱动器(DOS V1 所允许的最大值)。需要与 V2 及以上版本兼容时限定为 26。

请注意：AL 中的返回值以 1 为基数，代表与系统相联的磁盘驱动器数；而用于调用该功能的数值以 0 为基数，代表新的缺省磁盘驱动器。因此，若想将缺省驱动器置为最后逻辑驱动器，必须完成以下步骤：

1. 确定当前缺省驱动器（功能 19H)；
2. 把 DL 置为从步骤 1 中取得的当前缺省驱动器，调用本功能；
3. 步骤 2 的返回值减一（使之以 0 为基数)；
4. 用步骤 3 的导出值再次调用本功能。

INT 21H	功能 0FH 打开文件（FCB)	V1

在当前目录中搜索指定文件。如果找到指定文件，打开文件并填写文件控制块（FCB)

调用寄存器：	AH	0FH
	DS：DX	指向未打开 FCB 的指针
返回寄存器：	AL	00H，文件成功地打开
		FFH，文件未打开

注释：

本功能用 FCB 打开现存磁盘文件。这一功能不能创建文件，创建文件应使用功能 16H。在设置好 FCB 中的驱动器、文件名和扩展名之后，调用本功能。关于标准 FCB 和扩充 FCB 的格式参见后面部分。

应该注意，缺省驱动器的标志是 0，A 驱动器是 1，B 驱动器是 2 等。如果驱动器项置为缺省驱动器 (0)，调用后该项值被改变为正确的驱动器值。即使以后缺省驱动器发生了变化，对文件的后续使用仍然会保持正确。另外，该功能置 FCB BLOCK 项为 0，记录 RECORD 大小为 80H (128 字符的记录长度)，并从文件的目录项中得出文件长度、日期和时间。如果文件操作要使用不同的 BLOCK 或记录大小，应在该功能完成之后，其他 FCB 文件操作之前进行修改。

在网络环境中，请记住本功能在兼容模式下能自动打开文件。如果要求不同的模式，应

使用句柄功能。如果文件在不同的模式下创建，然后在兼容模式下打开（使用该功能），则会导致 DOS 严重错误并执行 INT 24H。

与其他 DOS 文件操作相似，错误由 AL 寄存器中返回的状态代码指出。如果 AL 为 0，则表示未发现错误；如果 AL 为 FFH，表示在操作时出错。

Microsoft 建议：除非要维护旧软件，最好不要使用该功能。该功能随时都可能不再被支持，应用功能 3DH 取代。

INT 21H	功能 10H	V1
	关闭文件（FCB）	

使用文件控制块（FCB）关闭已打开的文件

调用寄存器： AH 10H

DS：DX 指向已打开 FCB 的指针

返回寄存器： AL 00H 文件成功地关闭

FFH 文件未关闭

注释：

本功能使用 FCB 关闭已打开的磁盘文件。由于没有其他的方式能使 DOS 修改文件目录项，所以用 FCB 关闭文件，对程序的运行来说至关重要。不关闭文件，可能丢失数据。

要使用该功能，必须在 FCB 中提供文件名、扩展名和驱动器标志信息。已打开文件的 FCB 中，这些信息都已存在。

关闭文件时，系统要检查文件在目录中的位置。如果不一致，系统认为磁盘已发生变化，在 AL 中返回 FFH。有资料表明，该功能不像 DOS V2 所称的那样能正常工作。事实上，它将覆盖文件分配表（FAT）和目录，从而毁坏磁盘内容。

与其他 FCB 文件操作相似，AL 寄存器返回状态代码。如果 AL 为 0，则未发现错误；如果为 FFH，在磁盘操作中存在错误。对 DOS V3 及以上版本，在告知出错时，可调用取扩充错误功能（功能 59H）确定具体错误。

Microsoft 建议：除非要维护旧软件，最好不要使用该功能。该功能随时都可能不再被支持，应用功能 3EH 取代。

INT 21H	功能 11H	V1
	查找第一个目录项（FCB）	

在当前目录中查找第一个匹配的目录项

调用寄存器： AH 11H

DS：DX 指向未打开 FCB 的指针

返回寄存器： AL 00H 找到匹配的目录项

FFH 未找到匹配的目录项

注释：

本功能使用文件控制块（FCB）查找第一个出现的指定目录项。

要使用该功能，必须在 FCB 中提供文件名、扩展名和驱动器标志。从 DOS V2.1 开始，支持用问号（?）作为文件说明中的通配符。星号（*）仅在 DOS V3 及以上版本可用作通配符。

要查找带指定属性的文件，必须使用扩展 FCB。所需属性可由下表中的属性值组合得到。

表 1.7 文件属性值

属性值	文件匹配类型
00H	正常
02H	正常和隐含
04H	正常和系统
06H	正常、隐含和系统
08H	卷标
10H	目录

成功地完成本功能后，磁盘传输区（DTA）包含被找到的文件的 FCB。若使用扩展 FCB 查找，DTA 就包含扩展 FCB，否则只有普通 FCB。要得到 DTA 的更多的信息，请参考功能 1AH。

与其他 FCB 文件操作相似，返回的状态代码在 AL 寄存器中。如果 AL 为 0，则未发现错误；如果为 FFH，在磁盘操作中存在错误。对 V3 及以上版本，在出错后调用取扩充错误功能（功能 59H），可确定具体错误。

如果使用通配符查找文件，该功能不返回错误，可用功能 12H 继续查找下一个匹配的文件。

Microsoft 建议：除非要维护旧软件，最好不要使用该功能。该功能随时都可能不再被支持，应用功能 4EH 取代。

INT 21H	功能 12H 查找下一个目录项（FCB）	V1

在当前目录中查找下一个匹配的目录项

调用寄存器：	AH	12H	
	DS：DX	由功能 11H 或 12H 返回的指向 FCB 的指针	
返回寄存器：	AL	00H	找到匹配的目录项
		FFH	未找到匹配的目录项

注释：

调用功能 11H 之后，可用本功能继续查找，并可以根据查找文件的需要多次调用，但寻找的是下一个匹配的目录项，而不是第一个。进一步的信息可参见功能 11H。

显然，该功能仅在使用通配符进行目录项查找时才有价值。由 DS：DX 指向的文件控制块（FCB）应与功能 11H 调用时指向的 FCB 相同。

成功地完成该功能后，磁盘传输区包含所找到文件的 FCB。如果起初查找是用扩展 FCB 初始化的，DTA 就包含扩展 FCB；否则是普通 FCB。更多的信息请参考功能 1AH。

与其他 FCB 文件操作相似，返回的状态代码在 AL 寄存器中。如果 AL 为 0，则未发现错误；如果为 FFH，表示在磁盘操作中存在错误。对 V3 及以上版本，在告知出错时可调用取扩充错误功能（功能 59H）确定具体错误。

如果使用通配符查找文件，该功能不返回错误，可用本功能继续查找下一个匹配的文件。

Microsoft 建议：除非要维护旧软件，最好不要使用该功能。该功能随时都可能不再被支持，应用功能 4FH 取代。

INT 21H	功能 13H	V1
	删除文件（FCB）	

删除所有与提供文件说明匹配的目录项

调用寄存器： AH 13H

DS：DX 指向未打开的 FCB 的指针

返回寄存器： AL 00H 文件已删除

FFH 文件未删除

注释：

本功能通过文件控制块（FCB）删除文件。本功能仅删除普通文件。只读文件、系统文件、隐含文件、卷标或目录都不能用此功能删除。

要使用该功能，必须在相应的 FCB 项中提供文件名、扩展名和驱动器标志。从 DOS V2.1 开始，支持用问号（?）作为文件说明中的通配符。星号（*）仅在 DOS V3 及以上版本可用作通配符。

用本功能删除的文件并不从磁盘中清除。修改的目录项表示该文件已被删除而目录项可以使用了；其他文件能够使用以前由该文件使用的数据簇（Data Clusters）。文件中包含的数据没变，可以利用诸如 Norton Utilities，Mace Utilites 或 PC Tools 等特殊文件恢复程序予以恢复。

与其他 FCB 文件操作相似，返回的状态代码在 AL 寄存器中。如果 AL 为 0，则未发现错误；如果为 FFH，表示在磁盘操作中存在错误。出现错误的原因包括试图删除非法文件或找不到指定文件名。对 DOS V3 及以上版本，在告知出错时，可调用取扩充错误功能（功能 59H）确定具体错误。

在网络环境中，要删除文件，一定要拥有创建访问权。

Microsoft 建议：除非要维护旧软件，最好不要使用该功能。该功能随时都可能不再被支持，应用功能 41H 取代。

INT 21H	功能 14H	V1
	读顺序文件（FCB）	

从文件指针当前位置开始读下一个数据块并移动文件指针

调用寄存器：	AH	14H	
	DS：DX	指向打开的 FCB 的指针	
返回寄存器：	AL	00H	读成功
		01H	未读，已到 EOF
		02H	读命令，DTA 边界错误
		03H	读一部分后文件结束

注释：

本功能采用文件控制块（FCB）简化了从磁盘文件顺序读取信息，只能从以前打开了的文件（功能 0FH）中读取信息。

使用该功能时，要确保 DS：DX 指向成功地打开文件后创建的 FCB。FCB 中设置的参数控制顺序读，读取的长度在记录大小项中给定，位置由当前块号和当前记录号给定。在运行该功能之前，可以根据需要改变这些 FCB 项的值。

完成读取后，读出的信息被放入磁盘传输区（DTA），而且 FCB 中的记录地址自动加一。

因为从磁盘上读出的信息要放入 DTA，所以要保证 DTA 有足够空间接受这些信息。否则，可能覆盖其他数据。

与其他 FCB 文件操作相似，返回的状态代码在 AL 寄存器中。如果 AL 为 0，则未发现错误；如果为 FFH，表示在磁盘操作中存在错误。如果此功能读到的数据量穿越 DTA 中的内存段边界（以 000 结束的内存地址），就用 AL＝2 表示失败。若只读了部分记录（AL＝3），则要从记录尾到结尾处都续上 0 字符。对于 V3 及以上版本，在告知出错时，可调用取扩充错误功能（功能 59H）确定具体错误。

在网络环境中，使用此功能一定要拥有读取访问权。

Microsoft 建议：除非要维护旧软件，最好不要使用该功能。该功能随时都可能不再被支持，应用功能 3FH 取代。

INT 21H	功能 15H	V1
	写顺序文件（FCB）	

在文件控制块（FCB）所指定的当前位置上写入记录

调用寄存器：	AH	15H	
	DS：DX	指向已打开的 FCB 的指针	
返回寄存器：	AL	00H	写成功
		01H	未尝试写，磁盘已满或是只读文件
		02H	取消写，DTA 边界错误

注释：

本功能采用 FCB 简化了向磁盘文件顺序写数据，只能向已打开的（功能 0FH）或创建（功能 16H）的文件写数据。

使用该功能时，要确保 DS：DX 指向成功地打开文件后创建的 FCB。FCB 中设置的参数控制顺序写，写的长度在记录大小项中给定，位置由当前块号和当前记录号给定。在运行该

功能之前，可以根据需要改变这些FCB项的值。

因为信息来自磁盘传输区（DTA），注意所要写的记录大小即所要的数据量。否则，乱七八糟的数据会写入磁盘文件中。

如果所写的数据量不能填满整个DOS磁盘缓冲区（DOS内部），就简单地把数据加到磁盘缓冲区中已有数据之后，直到缓冲区满后再写入磁盘。成功地完成本功能后，它自动地修改FCB中的记录地址。

与其他FCB文件操作相似，返回的状态代码在AL寄存器中。如果AL为0，则未发现错误；如果为FFH，表示在磁盘操作中存在错误。如果磁盘已满或试图写只读文件，则AL=1；如果在写操作中DTA穿越了内存段地址(以000结尾)边界，功能失败并返回AL=2。

在网络环境中，使用此功能一定要拥有写入访问权。

Microsoft建议：除非要维护旧软件，最好不要使用该功能。该功能随时都可能不再被支持，应用功能40H取代。

INT 21H	功能16H 创建文件（FCB）	V1

根据文件控制块(FCB)提供的信息创建磁盘文件

调用寄存器：	AH	16H
	DS：DX	指向未打开的FCB的指针
返回寄存器：	AL	00H，文件已创建
		FFH，文件未创建

注释：

本功能是打开文件功能0FH的补充。它创建指定文件并提供FCB供以后使用。

由于本功能在创建文件的同时，也截去了已经存在的同名文件内容而且没有警告，所以不能在所有的场合都使用本功能。首先，该功能在当前目录中查找指定文件，如果发现文件已存在，就将其删除并修改FCB，如同新建立的一样。如果文件不存在，则创建文件并为访问新文件而设置FCB。

要使用该功能，一定要在FCB中提供驱动器、文件名和扩展名。在使用扩展FCB时，还可以通过设置属性来创建隐含文件或卷标。

与其他FCB文件操作相似，返回的状态代码在AL寄存器中。如果AL为0，则未发现错误；如果为FFH，表示在磁盘操作中存在错误。对于V3及以上版本，在告知出错时，可调用取扩充错误功能（功能59H）确定具体错误。

在网络环境中，使用此功能一定要拥有创建访问权。

Microsoft建议：除非要维护旧软件，最好不要使用该功能。该功能随时都可能不再被支持，应用功能3CH取代。

INT 21H	功能17H 文件改名（FCB）	V1

更改现存文件的文件名

调用寄存器： AH 17H

DS：DX 指向改进的 FCB 的指针

返回寄存器： AL 00H 文件被改名

FFH 文件未改名

注释：

本功能通过改进的文件控制块（FCB）改变现存文件的文件名。只可改普通文件名，不能用该功能更改只读文件、系统文件、隐含文件、卷标或目录的名字。

该功能使用改进的 FCB 格式（见表 1.8）：

表 1.8 改进的 FCB 格式

偏移量	含义
00H	驱动器标志符
01H	原文件名
09H	原文件扩展名
11H	新文件名
19H	新文件扩展名

注意，只要求三项基本信息：驱动器标志符（必须在同一驱动器中更名）和新、旧文件名。从 DOS V2.1 开始，在文件名说明中支持问号（?）通配符。星号（*）仅在 DOS V3 及以上版本中可作为通配符。在原文件名中放入通配符可以把与格式匹配的每个文件都更名。把通配符放入新文件名中可使文件名的相应字符保持不变。

任意给定目录中的文件名都必须唯一，若要求把文件更名为已有的文件名，该功能就会停止并返回错误。有效地运用通配符可以同时更名多个文件。例如，有一系列文件，文件名为 ABC01.DAT，ABC02.DAT，ABC03.DAT 等，想把它们更名为扩展名为 .OLD，如果选择原文件名为 ABC?? .DAT，新文件名为 *.OLD，就可以顺利地更名。

与其他 FCB 文件操作相似，返回的状态代码在 AL 寄存器中。如果 AL 为 0，则未发现错误；如果为 FFH，表示在磁盘操作中存在错误。对于 V3 及以上版本，在告知出错时，可调用取扩充错误功能（功能 59H）确定具体错误。

在网络环境中，使用此功能一定要拥有创建访问权。

Microsoft 建议：除非要维护旧软件，最好不要使用该功能。该功能随时都可能不再被支持，应用功能 56H 取代。

INT 21H	功能 19H	V1
	取缺省驱动器	

返回当前缺省驱动器号

调用寄存器： AH 19H

返回寄存器： AL 当前驱动器号（从 A=0 到 Z=25）

注释：

本功能可确定 DOS 正用哪一个磁盘驱动器作缺省驱动器，在 AL 中返回代表缺省驱动器的数。该号码以 0 为基数，0 代表 A，1 代表 B 等，这与把 0 规定为缺省驱动器的其他功能略有区别。本功能与功能 0EH 有关，功能 0EH 用来设置缺省驱动器。

INT 21H	功能 1AH	V1
	设置 DTA 地址	

建立 DOS 用作磁盘传输区（DTA）开始的地址

调用寄存器： AH 1AH

DS：DX 指向新的 DTA 的指针

返回寄存器： 无

注释：

本功能规定一个由 DOS 用作磁盘操作的 DTA。DTA 可由很多 DOS 功能使用，最明显的是 FCB 的文件功能；用于文件查找的处理功能（功能 4EH 和 4FH）和 INT 25H 及 INT 26H 也使用 DTA。程序开始时，128 个字节的缺省 DTA 被设置在程序段前缀（PSP）偏移量为 80H 处。该功能的逆功能是 2FH，用于取当前 DTA 地址。

程序员应当注意，DTA 对正在执行的任务应是足够充裕的。因为 DOS 只记录 DTA 的开始地址，系统在磁盘操作期间无法获知磁盘操作是否到达 DTA 的末尾。结果，如果被传送的数据量超过 DTA 的容量，则从磁盘上传来的信息很容易覆盖程序数据和代码。

INT 21H	功能 1BH	V1
	取分配表信息	

取缺省驱动器中磁盘的磁盘分配基本信息

调用寄存器： AH 1BH

返回寄存器： AL 每簇扇区数

CX 每物理扇区字节数

DX 每磁盘簇数

DS：BX 指向介质描述（Media descriptor byte）的指针

注释：

本功能返回缺省驱动器中磁盘容量的基本信息。一般所用容量比乘积 CX * AL * DX 少得多，这个乘积给出了磁盘的总容量（用字节表示）。功能 1CH 返回指定驱动器内磁盘的同一信息，而功能 36H 用于确定磁盘剩余空间的数量。

从 DOS V2 开始，DS：BX 指向 FAT 中介质描述字节，但 DOS V1 版本中，它实际指向内存中的 FAT。介质描述（或 FAT ID）字节可用于识别下表中介质的格式：

表 1.9 磁盘介质描述字节

值	含义
F0H	无法识别（1.44M3 英寸软盘使用该代码）
F8H	固定磁盘
F9H	双面，每道 15 扇区（1.2M）
F9H	双面，每道 9 扇区（720K）
FCH	单面，每道 9 扇区
FDH	双面，每道 9 扇区（360K）
FEH	单面，每道 8 扇区
FFH	双面，每道 8 扇区

请注意：F9H FAT ID 字节仅能表明磁盘已格式化为高容量，必须检查该功能返回的其他信息以确定磁盘实际容量。而且，FAT ID 并非为所有 DOS 版本支持。所给的标准来自 IBM DOS 版本的技术手册，可能不适合于特殊厂商的 DOS 版本。

Microsoft 建议：除非要维护旧软件，最好不要使用该功能。该功能随时都可能不再被支持，应用功能 36H 取代。

INT 21H 功能 1CH V2
取指定驱动器分配表信息

取指定驱动器中磁盘的磁盘分配基本信息

调用寄存器： AH 1CH
DL 驱动器号（当前驱动器＝0，A＝1 到 Z＝26）

返回寄存器： AL 每簇扇区数
CX 每物理扇区字节数
DX 每盘簇数
DS：BX 指向介质描述字节的指针

注释：

本功能返回指定驱动器中磁盘容量的基本信息，这些信息与功能 1BH 返回的一样，详见功能 1BH 的注释。

Microsoft 建议：除非要维护旧软件，最好不要使用该功能。该功能随时都可能不再被支持，应用功能 36H 取代。

INT 21H 功能 1FH V2
取缺省磁盘参数块

返回缺省驱动器磁盘参数块（DPB）的地址

调用寄存器： AH 1FH

返回寄存器： AL 00H 无错

FFH 出错

DS：BX 磁盘参数块地址

注释：

本功能在 DS：BX 中返回磁盘参数块（DPB）地址，DOS 用 DPB 来确定缺省驱动器内磁盘的具体结构信息。DPB 结构如表 1.10 所示。

表 1.10 驱动器参数块（DPB）

偏移字节	域宽	含义
【所有版本】		
00H	字节	驱动器号（0=A，1=B 等）
01H	字节	物理驱动器号
02H	字	每扇区字节数
04H	字节	每簇扇区数（以 0 为基数）
05H	字节	转换因子（扇区大小）
06H	字	包含 FAT 的第一个扇区号
08H	字节	FAT 份数
09H	字	根目录项数
0BH	字	第一个数据扇区号
0DH	字	最高簇号加 1
【仅对 V2 或 V3】		
0FH	字节	每 FAT 扇区数（0—255）
10H	字	根目录的起始扇区号
12H	双字	驱动器的设备驱动程序地址
16H	字节	介质描述字节
17H	字节	磁盘参数块访问标志（0FFH 表示需重建）
18H	双字	下一个设备参数块地址
【仅对 V2】		
1CH	字	当前目录开始簇号
1EH	64 字节	当前目录的 ASCIIZ 路径
【仅对 V3】		
1CH	字	上一次从本驱动器分配出的簇号
1EH	字	未知 一般是 FFFFH
【仅对 V4】		
0FH	字	每 FAT 扇区数（0—65535）
11H	字	根目录开始扇区号
13H	双字	驱动器的设备驱动程序地址
17H	字节	介质描述字节
18H	字节	磁盘参数块访问标志（0FFH 指需要重建）
19H	双字	下一个设备参数块地址
1DH	字	上一次从本驱动器分配出的簇号
1FH	字	未知 一般是 FFFFH
【仅对 V5】		
0FH	字	每 FAT 扇区数（0—65535）

续表

偏移字节	域宽	含义
11H	字	根目录开始扇区号
13H	双字	驱动器的设备驱动程序地址
17H	字节	介质描述字节
18H	字节	磁盘参数块访问标志（0FFH 指需要重建）
19H	双字	下一个设备参数块地址
1DH	字	上一次从本驱动器分配出的簇号
1FH	字	自由簇数

请注意：对于每个 DOS 版本，DPB 结构都不一样。表中每一项的意义必须结合特定的 DOS 版本进行理解。

因为本功能把返回值放入 DS 寄存器，在调用该功能前应该保存 DS 值。

INT 21H　　功能 21H　　V1

随机文件读（FCB）

从磁盘文件中读 FCB 当前块和当前记录项指定的记录，把信息放入磁盘传输区（DTA）

调用寄存器：　AH　　21H

　　DS：DX　指向打开的 FCB 的指针

返回寄存器：　AL　　00H　读成功

　　01H　未读，已到 EOF

　　02H　读取消，DTA 边界错误

　　03H　部分记录读，正处于 EOF

注释：

本功能用文件控制块（FCB）简化了从磁盘文件读取随机（不连续）信息。只能从已打开的文件（功能 0FH）中读取信息。

使用本功能，要确保 DS：DX 指向成功地打开文件后建立的 FCB，FCB 中设置的参数控制随机读取。要读取的记录由 FCB 中设置的随机记录项指定，数据量由记录大小项控制。在使用该功能之前，可以根据需要改变这两个 FCB 项的值，DOS 用这两个值计算开始读取文件的实际位置。

此功能完成后，读出的信息放入磁盘传输区（DTA）。与顺序写不同，此功能不修改 FCB 当前位置项。除非随机记录项发生变化，否则，对文件的顺序访问返回同样的数据。

因为从磁盘上读出的信息要放入 DTA，所以要保证 DTA 有足够空间接受这些信息。否则，磁盘来的信息可能覆盖其他数据。

与其他 FCB 文件操作相似，返回的状态代码在 AL 寄存器中。如果 AL 为 0，表示未发现错误；如果为其他值，则在磁盘操作中存在错误。如果此功能读到的数据量超过了 DTA 内存段边界（以 000 结束的内存地址，即整 64K 边界），就用 AL＝2 表示失败。若读取的是部分记录（AL＝3），则从记录尾到结尾处都补上 0 字符。

在网络环境中，使用此功能一定要拥有读取访问权。

Microsoft 建议不再使用此功能，除非要维护旧的软件。对这种功能的支持可能随时结束，应用功能 3FH 取而代之。

INT 21H	功能 22H	V1
	随机文件写（FCB）	

把文件控制块（FCB）中的当前块和记录项指定的记录写入磁盘文件，从磁盘传输区（DTA）传送信息。

调用寄存器： AH 22H

DS：DX 指向已打开的 FCB 的指针

返回寄存器： AL 00H 写成功

01H 未尝试写，磁盘已满或是只读文件

02H 取消写，DTA 边界错误

注释：

本功能采用 FCB 简化了写随机（不连续）信息到磁盘文件中。只能向已打开了的（功能 0FH）或创建（功能 16H）的文件写数据。

要使用该功能，就要确保 DS：DX 指向成功地打开或创建文件后建立的 FCB，FCB 中设置的参数控制随机写。要写的记录由 FCB 中设置的随机记录项给定，数据量由记录大小项控制。在运行该功能之前，可以改变这些 FCB 项值为具体使用的需要值，DOS 用这些值计算开始写文件的位置。

因为写入磁盘上的信息来自 DTA，请注意所要写的记录大小即所要的数据量。否则，乱七八糟的数据会被错误地写入磁盘文件中。要得到关于 DTA 的更多的信息，可参见功能 1AH。

如果所写的数据量不能填满整个 DOS 磁盘缓冲区（DOS 内部），先将这些数据加到磁盘缓冲区中已有数据之后，待磁盘满后再写入磁盘。

与顺序功能不同，该功能不修改 FCB 当前位置项，除非随机记录项已发生变化。否则，后续随机文件写入会把信息传送给同一文件记录。

与其他 FCB 文件操作相似，错误由 AL 寄存器中返回的状态代码指出。如果 AL 为 0，则未发现错误；如果为其他值，在磁盘操作中存在错误。如果磁盘已满或试图写到只读文件中，AL＝1；如果在写操作超过 DTA 内存段地址边界（以 000 结尾的内存地址，即整 64K 边界），则功能失败并返回 AL＝2。

在网络环境中，要使用此功能，一定要拥有写入访问权。

Microsoft 建议：除非要维护旧软件，最好不要使用该功能。该功能随时都可能不再被支持，应用功能 40H 取代。

INT 21H	功能 23H	V1
	取文件大小（FCB）	

查找目录中的匹配文件名，如果找到文件，则在指定的文件控制块（FCB）中填入大小信息

调用寄存器： AH 23H
DS：DX 指向未打开的 FCB 的指针

返回寄存器： AL 00H 找到匹配文件
FFH 未找到匹配文件

注释：

本功能使用 FCB 确定指定文件中的记录数。在使用本功能时，文件应是未打开的。

在 FCB 中填好驱动器、文件名、扩展名和记录大小之后，就可以使用本功能了。提供的文件名一定要完整而且唯一，不允许用通配符。以字节表示找到文件的大小，只要简单地置记录大小项为 1。

如果已找到与指定文件名匹配的文件，由 DS：DX 指向的 FCB 的随机记录项要改为该文件的记录数，记录数由文件大小（以字节表示）除以记录大小得到。记录数向上取整。如果在调用此功能前忘了设置记录大小项，或者对于文件来说，目录项的文件大小部分不正确或取整为整个扇区，该功能返回的信息就值得怀疑。

与其他 FCB 文件操作相似，错误由 AL 寄存器中返回的状态代码指出。如果 AL 为 0，则未发现错误；如果为 FFH，在磁盘操作中存在错误（一般是指定文件未找到）。

Microsoft 建议：除非要维护旧软件，最好不要使用该功能。该功能随时都可能不再被支持，应用功能 42H 取代。

INT 21H	功能 24H 置随机记录项（FCB）	V1

从顺序文件 I/O 到随机文件 I/O 切换时，用来设置基于当前文件位置的文件控制块（FCB）随机记录项

调用寄存器： AH 24H
DS：DX 指向打开的 FCB 的指针

返回寄存器： 无

注释：

本功能修改打开的 FCB，为随机访问功能做准备。填好 FCB 的记录大小、记录号和块号之后，就可以使用这种功能了。该功能以这些项的值为基础修改随机记录项。

Microsoft 建议：除非要维护旧软件，最好不要使用该功能。该功能随时都可能不再被支持，应用功能 42H 取代。

INT 21H	功能 25H 置中断向量	V1

安全地修改中断向量，使之指向指定的中断服务程序

调用寄存器： AH 25H

AL 中断号
DS：DX 指向中断服务程序的指针

返回寄存器： 无

注释：

本功能使本来很敏感的操作——改变中断向量迅速完成。因为中断向量放在内存低端的一张表中，很容易直接修改；但是，如果修改过程中发生中断，特别是正当只有一部分地址被传入表中的时候，这样可能很危险。

为了安全地管理中断设置，与其自己填写代码，不如使用这种功能。它可以保证把中断向量安全地改为用户提供的地址。这是改变中断向量的唯一可靠方法。

改变中断向量可能导致一些特殊问题。当程序以正常方式终止时，INT 22H，23H 和 24H 自动复位到初始值（见 INT 20H，INT 27H 和 INT 21H 功能 00H、31H 及 4CH）。如果程序修改其他任何中断向量，它们将保持修改值，即使程序终止以后也如此。这会出问题：在程序终止后出现了中断条件，就可能跳转到不存在的中断处理程序处。

为了防止这种情况，改变中断向量的程序应先用功能 35H 取出并保存初始向量值。程序结束时，恢复初始向量值。一个程序若改变除 22H、23H 或 24H 之外的中断向量，它必须阻止所有可能非正常终止程序的途径——这是指 Ctrl-C 和 Ctrl-Break 中断服务、DOS 严重错误服务及其他任何可能导致程序终止的潜在中断服务，如被 0 除。

如果程序不做任何截取而所有方式都能使之终止，挂起的中断向量可以保持。如果程序破坏了中断向量而不能使之恢复，唯一安全的办法就是在做任何其他事情之前复位系统。

INT 21H	功能 26H 创建 PSP	V1

从当前执行程序把程序段前缀（PSP）复制到指定段地址，然后为适合新程序的使用而更新 PSP

调用寄存器： AH 26H
DX 新 PSP 的段地址

返回寄存器： 无

注释：

本功能为准备运行的下一个程序创建 PSP。在指定的内存段地址处有当前程序的 PSP 备份，PSP 结构参见后面列表。

理论上，可以把 COM 文件直接复制到新 PSP 后的内存空间并执行它。但是它并不是一个好方法，理由很简单：这种功能已经过时。在 V2 及以后的版本中，它已为更完善、更简洁的 EXEC 功能（功能 4BH）所代替。所有 Microsoft 和 IBM 资料都推荐在生成程序时用功能 EXEC，而不用功能 26H。EXEC 能仔细控制程序的装入执行，并把程序与运行中潜在的错误隔开。本功能不装入或执行下一个程序，它只为文件准备 PSP。调用该功能后，还必须装入和执行。

如果用户程序改变了 INT 22H、23H 和 24H 的中断向量，新的向量被复制到新创建的 PSP 中。另外，内存分配信息也相应修改。

Microsoft 建议：除非要维护旧软件，最好不要使用该功能。该功能随时都可能不再被支持，应用功能 4BH 子功能 00H 取代。

INT 21H	功能 27H 随机块读（FCB）	V1

从磁盘文件读一个或多个相连的随机记录到磁盘传输区（DTA）

调用寄存器：	AH	27H	
	CX	要读的记录数	
	DS：DX	指向已打开的 FCB 的指针	
返回寄存器：	AL	00H	成功读所有记录
		01H	未读，已到 EOF
		02H	读取消，DTA 边界错误
		03H	部分记录读，正处于 EOF
	CX	读过的记录数	

注释：

本功能通过文件控制块（FCB）简化了从磁盘文件读一组相连的随机记录。它只能从已打开了的文件（功能 0FH）中读取信息。

使用该功能，要确保 CX 包括要读的记录数，而且 DS：DX 指向成功地打开文件后创建的 FCB。FCB 中设置的参数控制随机读取，要读取的起始记录在 FCB 中由设置随机记录项指定，数据量由记录大小项控制。在使用该功能之前，可以改变这些 FCB 项值为具体使用需要的值，DOS 用这两个值计算开始读取文件的实际位置。

完成读取后，从磁盘中读出的信息被放入磁盘传输区（DTA）。相关的解释见功能 21H。当此功能成功完成后，随机记录、当前块和当前记录等项都被修改。

因为从磁盘上读出的信息要放入 DTA，所以要保证 DTA 足够充裕以接受这些块信息。否则，磁盘来的信息可能覆盖其他数据或程序代码。

与其他 FCB 文件操作相似，错误由 AL 寄存器中返回的状态代码指出。如果 AL 为 0，则未发现错误；如果为其他值，在磁盘操作中存在错误。如果此功能读到的数据量超过 DTA 中的内存段边界（以 000 结束的内存地址），就用 AL＝2 表示失败。若是部分记录读（AL＝3），则从 EOF 处到记录结尾都续上 0 字符。

在网络环境中，使用此功能一定要拥有读取访问权。

Microsoft 建议：除非要维护旧软件，最好不要使用该功能。该功能随时都可能不再被支持，应用功能 3FH 和功能 42H 取代。

INT 21H	功能 28H 随机块写（FCB）	V1

从磁盘传输区（DTA）写一个或多个相连的记录到磁盘文件

调用寄存器： AH 28H
CX 要写的记录数
DS：DX 指向打开的FCB的指针
返回寄存器： AL 00H 成功地写所有记录
01H 未曾写，磁盘已满或是只读文件
02H 取消写，DTA边界错误
CX 所写的记录数

注释：

本功能采用FCB简化了写随机（不连续）信息到磁盘文件中，只能向已打开（功能0FH）或已创建（功能16H）的文件写数据。

使用该功能，要确保CX包括要读的记录数，而且DS：DX指向成功地打开文件或创建文件后建立的FCB。FCB中设置的参数控制随机写，要写的起始记录在FCB中由设置随机记录项指定，数据量由记录大小项控制。在使用该功能之前，可以改变这些FCB项值为具体使用需要的值，DOS用这两个值计算开始写文件的实际位置。

因为写入磁盘上的信息来自磁盘传输区（DTA），注意，所要写的记录大小和计数要对应于所需的数据量。否则，乱七八糟的数据会写入磁盘文件中。

如果所写的数据不能填满整个DOS磁盘缓冲区（DOS内部），则简单地将这些数据加到磁盘缓冲区中已有数据之后，待磁盘缓冲区满后再写入磁盘。

成功地完成功能后，FCB的随机记录、当前块和当前记录项即被更新。

与其他FCB文件操作相似，错误由AL寄存器中返回的状态代码指出。如果AL为0，则未发现错误；如果为其他值，则表明磁盘操作中存在错误。如果磁盘已满或试图写到只读文件中，AL=1；如果此功能读到的数据量超过DTA中的内存段边界（以000结束的内存地址），就用AL=2表示失败。

在网络环境中，使用此功能一定要拥有写访问权。

Microsoft建议：除非要维护旧软件，最好不要使用该功能。该功能随时都可能不再被支持，应用功能40H和功能42H取代。

INT 21H 功能29H V1
分析出文件名

分析出文件名字符串，把它放到文件控制块（FCB）备用

调用寄存器：AH 29H
AL 分析控制标志（见表1.11）
DS：SI 指向待分解字符串的指针
ES：DI 指向FCB的指针
返回寄存器：AL 00H 未遇通配符
01H 找到通配符
FFH 驱动器说明符无效
DS：SI 指向分析过的文件名后下一个字符

ES：DI　指向已修改而未打开的 FCB

注释：

本功能用于从命令行中抽取文件名，并将其以适当的格式放入打开的 FCB 中。这么用之前，先要用一指针指向文件名字符串，另一指针指向要用的 FCB。FCB 没有什么格式，只要是一块足够大的内存就行了。

各种版本中的分隔符是句号（.），逗号（,），分号（;），冒号（:），等号（=），加号（+），Tab 和空格。在 DOS V1 中，还有下列符号可用作分隔符：包括双引号（"），斜杠（/），左方括号（[）和右方括号（]）。

该功能返回文件的一个正确的、未打开的 FCB，还有一个指向文件名后第一个字符的指针。本功能中星号被自动变为一个或多个问号，在文件名或扩展名的末尾。

由于是 FCB 功能，与路径名不兼容，因此，文件名中不能包括目录。本功能只用于当前目录中的文件。文件名的解释由表 1.11 中的分析标志控制。

表 1.11　分析控制标志

位 7654	位 3210	含义
....	...0	如果遇到文件分隔符，停止分析
....	...1	忽略引导分隔符，继续往下解析
....	..0.	如果不含驱动器字母，置 FCB 中驱动器 ID 为 0(缺省驱动器)
....	..1.	如果不含驱动器字母，则不更改 FCB 中驱动器 ID
....	.0..	如果不含文件名，FCB 中文件名置为 8 个空格(20H)
....	.1..	如果不含文件名，则不更改 FCB 中原文件名
....	0...	如果不含扩展名，FCB 中扩展名项置为 3 个空格
....	1...	如果不含扩展名，则不更改 FCB 中原扩展名

当要用到 FCB 时，可用本功能设置 FCB，因为本功能执行后，生成了一个格式正确的 FCB，接下来就可将其打开。准备好后，要使用 FCB 打开或创建功能，ES：SI 中的指针必须移到 DS：DX。

如果无有效的文件名可分析，该功能返回指针 ES：DI，以使指针 ES：DI+1 指向空格符。

INT 21H	功能 2AH	V1
	取系统日期	

取系统日期，包括年、月、日及星期几

调用寄存器：　AH　2AH

返回寄存器：　CX　年（1980—2099）

DH　月（1—12）

DL　日（1—31）

【对 DOS V1.1 及以后版本】

AL 星期（0=星期日，1=星期一，等）

注释：

本功能返回DOS所记录的当前系统日期。这里指的是DOS内部时钟，而不是实时时钟。一般，如果系统配有实时时钟，当系统启动时AUTOEXEC.BAT文件将读取日期，或者由操作者设置。

如果让系统开几天却不用它，就可能引起影响系统时间错误。当调用一个指定功能时，DOS先检查午夜标志，如果已设置，日期就更新；而如果根本不调用指定功能，日期就没法更新。这样，可能过了多个午夜而日期只加了一天。注意：BIOS功能1AH（取时钟计数）会复位午夜标志。系统长期未用后，开始使用时要检查并改变日期。

本功能与功能2BH（置系统日期）寄存器格式相同。

INT 21H	功能2BH 置系统日期	V1

置系统日期为指定值而不影响系统时间

调用寄存器： AH 2BH
CX 年（1980—2099）
DH 月（1—12）
DL 日（1—31）

返回寄存器： AL 00H 日期设置成功
FFH 日期无效，未设置

注释：

本功能与功能2AH使用相同的寄存器。如果有实时时钟，可以访问它来获取当前日期，并使用本功能修改系统日期；若没有实时时钟，也可以让用户输入。

标记文件时用的是本功能设置的日期。

如果系统中带有CMOS时钟，本功能将设置CMOS时钟。

INT 21H	功能2CH 取系统时间	V1

以小时、分、秒和百分之一秒的方式取系统时间

调用寄存器： AH 2CH

返回寄存器： CH 小时（0—23）
CL 分钟（0—59）
DH 秒（0—59）
DL 百分之一秒（0—99）

注释：

本功能与取系统日期功能相似，报告或屏幕显示需频繁调用这一功能。但要注意以下两

点：

- ●该功能取的不是实时时钟时间，而是 DOS 内部系统时间。
- ●系统的实时时钟不能精确到百分之一秒。这时，可用该功能返回百分之几秒的不连续时间值。

如有计时程序从置 DOS 时间 0 开始，检测经过的时间，退出时，应将其设成原来的时间。否则，系统时间被错误更改，其他程序调用这个功能时，得不到正确的系统时间。

INT 21H	功能 2DH	V1
	置系统时间	

设置系统时间，包括小时、分、秒和百分之一秒，不影响系统日期

调用寄存器：

AH	2DH	
CH	小时（0—23）	
CL	分钟（0—59）	
DH	秒（0—59）	
DL	百分之一秒（0—99）	

返回寄存器：

AL	00H	时间设置成功
	FFH	时间无效，未设置

注释：

寄存器格式与功能 2CH（取系统时间）一样。与他人一起工作时，精度设置最好不要超过正负一秒。

用实时时钟芯片或计时装置工作的程序，可将时间准确地设到百分之几秒。由于实时时钟不够精确，一些计算机不能连续返回这种精度的时间。

如果系统带有 CMOS 时钟，本功能可设置其时间。

INT 21H	功能 2EH	V1
	设置校验标志	

切换 DOS 校验标志。当校验打开时，写磁盘时将进行 CRC (Cyclic Redundancy Check) 校验，这样会增加磁盘传输时间

调用寄存器：

AH	2EH	
AL	00H	关闭校验
	FFH	打开校验
DH	00H（DOS 3.0 以前的版本）	

返回寄存器：无

注释：

如果不用磁盘写校验，就不能确保正确地写入数据。但是，如果对所有数据都进行磁盘写校验，那就会成倍地增加操作时间。某些非 IBM 的 BIOS 变种根本不支持该操作。

为了节省时间，可以不进行校验。当不用保证每次磁盘写都正确时，可关闭校验，只在至关重要的地方设置校验标志。

即使打开校验标志，也不能绝对保证数据写得正确，因为校验过程并不比较要写的数据和已写入数据的每一字节。

如果在内存和磁盘控制器间发生数据错误，这个过程无法提供保护。

功能 54H 可判定当前校验标志的设定。

INT 21H	功能 2FH 取 DTA 地址	V2

返回指向 DOS 当前使用的磁盘传输区（DTA）的指针

调用寄存器： AH 2FH

返回寄存器： ES：BX 指向 DTA 的指针

注释：

缺省 DTA 是一块位于程序段前缀（PSP）偏移量 80H 处的 128 字节缓冲区。如果要使用大的记录或进行特殊的磁盘传输，可另设一个 DTA（功能 1AH）。

INT 21H	功能 30H 取 DOS 版本号	V2

返回 DOS 版本号，2.0 以前的 DOS 版本返回 0

调用寄存器： AH 30H

AL 版本标志

00H OEM 号

01H DOS 版本标志

返回寄存器： AL 主版本号（2，3）

AH 次版本号（2.1=10）

BH OEM 号或 DOS 版本标志

BL：CX 24 位序号

注释：

在 Microsoft C 和 Borland C++中，DOS 版本值是全局变量。

若返回的 DOS 版本标志的第 3 位设置为 1，表明 MS-DOS 在 ROM 中运行；否则，DOS 是在 RAM 中运行。其他位未定义。如果取的是 OEM 号，与 OEM 有关的序号在 BL：CX 中；如果取 DOS 版本标志，BL：CX 为 0。

这里的 DOS 版本号用 MS-DOS SETVER 命令设置，它与功能 3306H 返回的版本号不同。

INT 21H	功能 31H	V2
	终止且驻留	

终止并驻留一个进程，然后把控制返回到其父进程

调用寄存器： AH 31H
AL 返回代码
DX 要保留的内存大小（以节计算）

返回寄存器： 无

注释：

终止且驻留（TSR）功能用得很广泛。除了用于弹出式计算器、日历和记事本等应用程序中外，TSR 还可用在向多个程序提供公用服务的子例程中。在建立了一个通过中断调用的功能库后，就可以向多个程序提供标准的公用服务，而不用把这些功能连接在所有的程序中。这样能减小程序的大小并提高装载速度。

本功能终止程序运行（像功能 4CH 一样），但不释放程序所占用的内存。这样，如果程序接管了某个中断，一旦条件满足，即可被激活。例如，可以设计一个接管时钟中断的程序，使其在屏幕上显示一个时钟。

本功能取代了 DOS V1 提供的 INT 27H 功能。原来的功能最多允许驻留 64K 而且不提供返回代码。使用本功能时，可在父进程和批处理文件得到返回代码。本功能从程序启动时分配的内存中，把 DX 寄存器要求的那部分内存驻留。它处理不了通过功能 48H 分配给进程的内存。

本功能不关闭程序打开的任何文件。句柄功能可通过程序段前缀（PSP）中一个未写入文档的区域与当前活动进程相联。当此 TSR 功能不是当前活动进程时，它使用句柄功能操作的文件就是当前实际活动进程所打开的文件。使用文件控制块（FCB）功能时就不会出现这种情况，因为文件控制块是在进程自己的内存中分配缓冲区。

因此，任何使用句柄功能处理文件或设备的 TSR，以及需要在弹出或其他活动时使用句柄功能的 TSR，在使用句柄前，必须用未写入文档的功能 50H 和 51H 切换 DOS 当前活动进程。否则，结果不正确，而且还会对其他程序或数据产生严重破坏。

INT 21H	功能 32H	V2
	取驱动器参数块	

取指定磁盘驱动器的驱动器参数块 DPB。如果调用时 DL＝0，本功能等同于功能 1FH

调用寄存器： AH 32H
DL 驱动器号（0＝缺省，1＝A 等）

返回寄存器： AL FFH 如果驱动器号无效
DS：BX 驱动器参数块地址

注释：

本功能在 DS：BX 中返回 DL 指定的驱动器的磁盘参数块（DPB）的地址。DPB 向 DOS

提供磁盘的具体结构信息。

DPB 的结构如表 1.10 所示（见功能 1FH）。请注意，DPB 的结构在每个 DOS 版本中都不同。

本功能的返回值用到了 DS 寄存器，所以在调用前应保存 DS 的值。

INT 21H	功能 33H	子功能 00H	V2
	获取 Ctrl-Break 标志		

获取 Ctrl-Break 检查标志的状态

调用寄存器：	AH	33H	
	AL	00H	获取标志状态
返回寄存器：	DL	00H	Ctrl-Break 检查关闭
		01H	Ctrl-Break 检查打开

注释：

除了字符 I/O 功能 01H—0CH（不含 06H、07H）外，其他 INT 21H 功能一般不做 Ctrl-Break 和 Ctrl-C 检查。当打开检查时，除了少数几个设置和获取标志数据的功能外，其他的 INT 21H 功能都作该检查。该子功能在 DL 中返回 Ctrl-Break 检查标志的当前状态。

当检查打开时，如果发现 Ctrl-Break 或 Ctrl-C，那么控制转到 INT 23H 的处理过程。INT 23H 的处理过程可以替换，用户可以用自己的方法处理 Ctrl-Break 或 Ctrl-C。

注意：Ctrl-Break/Ctrl-C 是一个系统全局标志，除启用或停用该功能的进程外，它可能影响 DOS 系统上所有其他进程，必须小心处理。

INT 21H	功能 33H	子功能 01H	V2
	设置 Ctrl-Break 标志		

设置 Ctrl-Break/Ctrl-C 检查标志的状态

调用寄存器：	AH	33H	
	AL	01H	设置标志状态
	DL	00H	关闭 Ctrl-Break 检查
		01H	打开 Ctrl-Break 检查
返回寄存器：	无		

注释：

除了字符 I/O 功能 01H-0CH（不含 06H、07H）外，其他 INT 21H 功能一般不做 Ctrl-Break 和 Ctrl-C 检查。当打开检查时，除了少数几个设置和获取标志数据的功能外，其他的 INT 21H 功能都作该检查。该子功能在 DL 中设置 Ctrl-Break 检查标志的当前状态。

当检查打开时，如果发现 Ctrl-Break 或 Ctrl-C，那么控制转到 INT 23H 的处理过程。INT 23H 的处理过程可以替换，用户可以用自己的方法处理 Ctrl-Break 或 Ctrl-C。

注意：Ctrl-Break/Ctrl-C 是一个系统全局标志，除启用或停用该功能的进程外，它可能

影响 DOS 系统上所有其他进程，必须小心处理。

INT 21H	功能 33H	子功能 05H	V4
	取引导驱动器代码		

返回最近引导系统的驱动器代码

调用寄存器： AH 33H

AL 05H 获取引导驱动器代码

返回寄存器： DL 引导驱动器代码（1=A，2=B，3=C 等）

注释：

V4 中增加了该功能，程序可用它判定是哪一个驱动器引导了系统。每当系统初始化时，这一信息就被存入 DOS 内核区。

INT 21H	功能 33H	子功能 06H	V5
	取 MS-DOS 真正版本号		

返回 MS-DOS 版本号、次版本号和版本标志

调用寄存器： AH 33H

AL 06H，获取 MS-DOS 版本号

返回寄存器： BL 主版本号

BH 次版本号

DL 次版本号放在第 0—2 位；其他位保留并置零

DH 版本标志

第 3 位如果置位

表示 MS-DOS 从 ROM 运行

否则，表示 MS-DOS 从 RAM 运行

第 4 位如果置位

表示 MS-DOS 从高端内存（HMA）运行

否则，表示 MS-DOS 从常规内存开始运行

所有其他位保留并置零

注释：

SETVER 不改变版本号。

INT 21H	功能 34H	V2
	返回 InDOS 标志地址	

这是一个 DOS 内部标志，用于告知 DOS 正在处理一个 INT 21H 功能

调用寄存器： AH 34H

返回寄存器： ES：BX 指向InDOS标志的指针

注释：

当DOS进入一个INT 21H功能时，DOS将InDOS标志加1；当退出时，该标志减1。这个标志由DOS和TSR实用程序使用，用来确定DOS是否在内核中处理一个INT 21H功能。当TSR实用程序判定DOS正在执行某一INT 21H功能时，它只可能有两种选择：

- ●如果TSR实用程序不需要调用INT 21H功能，则它继续执行；
- ●如果TSR实用程序也需调用INT 21H功能，因为已有INT 21H功能被调用，则它只能拒绝处理。

由于操作系统内核不是全部可重入，因此需要InDOS标志。如果操作系统已经在DOS内核时又有一个中断发生，则服务例程不能调用INT 21H功能，因为该功能调用可能和前面的冲突。这时，系统很容易崩溃，如果再去直接跟踪中断处理程序、寻找崩溃原因就毫无必要了，因为这是重入造成的。

由于操作需要，DOS某些内核代码的确通过INT 21H接口来调用其他部分，并且用InDOS标志充当一个信号来防止代码重入。然而，它允许诸如PRINT.COM及与其类似的正式TSR程序使用INT 21H，当然，这样容易使人产生误解。

当DOS等待键盘输入时，它空循环直到有字符进入。只要DOS在该点等待，即使InDOS标志指示正处在DOS内核，也可安全使用文件操作和其他功能。为了告知用户能安全使用这些功能，DOS在其输入循环期间连续调用INT 28H（键盘忙循环）。INT 28H缺省时为一个IRET指令，TSR能截取INT 28H，并检查在TSR激活时要做的事情。

例如，用户的TSR由按一个热键开始，但因为InDOS标志置位，所以发现DOS正在执行一个INT 21H功能，这样，TSR可设置一个内部调用标志，表示“我已经被调用，但我不能做任何事情”。无论何时调用INT 28H，都会检查本内部调用标志；如果该标志置位，就立即执行被请求的TSR功能。

TSR也可截取时钟中断，并且在时钟嘀哒期间检查InDOS标志和TSR内部调用标志。这样，在首次调用TSR时，就能注意字符I/O功能以外的INT 21H功能活动的情况，这时，若时钟中断查看到InDOS标志被清、TSR调用标志被置，则执行所请求的功能。

然而，DOS V2中使用INT 28H会使系统神秘地崩溃。这是由于V2把控制调度给任何被请求的功能以前，V2切换到了它本身内部的堆栈。如果DOS已经在处理范围00H到0CH中（调用INT 28H时，大多是在该范围内）某个功能时，再调用任何功能，由此产生的第二次堆栈切换会破坏原先功能调用所需的信息。

对付这一问题的办法相当简单但很巧妙：可以通过改变其严重错误CritErr标志来欺骗DOS，使它相信它正在处理一个严重错误。在调用功能前设置CritErr标志，迫使DOS使用其内部备用堆栈。功能返回以后，必须恢复该标志的初始值。

在V3中，做任何堆栈切换之前，为了检查字符I/O功能以外的INT 21H功能活动的情况，修改了DOS调度代码（Dispatch Code）；如果调用了其中某个功能，DOS就干脆使用调用程序本身的堆栈，以避免这类问题。在V4中，不修改这个代码。

CritErr标志是一个字节，表示DOS遇到了严重错误，现在正在执行一个INT 24H（严重错误）处理过程。为了让INT 24H使用尽可能多的DOS功能，将该标志置位，这样，DOS调度代码使用的是其内部备用堆栈而不是普通堆栈空间。虽然这不是完全可重入编码，但允

许有限次重新使用它。

在 DOS 从 V2 到 V4 的所有版本中，CritErr 标志和 InDOS 标志总是相邻的。不幸的是，在 V2 和 V3 中其位置是颠倒的。在 V2 中，CritErr 是高地址字节；在 V3 和 V4 中，它是低地址字节。

注意，从 INT 28H 处理过程调用本功能时可能要出问题。正确用法是：在装入 TSR 时就从 INT 28H 处理过程调用本功能，要在 TSR 驻留之前！在此时，可以使用所有 DOS 工具来判定应当用哪一种方法修正 CritErr 地址，本功能可以用来获取和保存 InDOS 标志地址、确定 DOS 版本、按需要加 1 或减 1 以及在一个独立的指针中保存 CritErr 标志地址。

然后，当 TSR 弹出时，它能用它自己的 InDOS 指针来确定 DOS 是否在运行，并且，如果需要的话，TSR 能用它自己的 CritErr 指针来获取 CritErr 标志的当前值，保存该值，设置该标志来强行使用备用堆栈、做任何需要的处理以及在退出前恢复初始标志值。

INT 21H	功能 35H 取中断向量	V2

取指定中断的中断处理程序地址

调用寄存器： AH 35H

AL 中断号

返回寄存器： ES：BX 指向中断处理程序的指针

注释：

一个给定中断的中断处理过程地址很容易获得，但只有本功能才是获取一个中断向量当前设置的正规途径。本功能有条理地工作，并且返回一个可靠的向量值。当然，可能有另一个程序，例如 TSR 实用程序，在本功能返回以后会改变中断向量——这就是超出单用户、单任务操作系统界限时所发生的情况。

当建立一个由中断开始工作的程序时，应当用本功能来确定中断的初始设置，以便完成时能恢复它。做的时候必须小心，因为这可能与 TSR 或其他中断处理程序发生冲突。如果保存了中断值，但 TSR 启动后并改变了它，那么就可以用某种停用 TSR 的方法来重用原中断向量。

为了防止这种事件发生，最好经常比较要恢复的某个中断向量的当前内容和用户程序初始设置的值。如果值不一致，从保存中断向量以后，一定是有程序修改了它，这时，恢复或从内存中清除用户程序不安全。如果它们相匹配，就可恢复初始值而无危险。

INT 21H	功能 36H 取自由磁盘空间	V2

获取指定盘可用空间的数量以及其他有关盘的信息

调用寄存器： AH 36H

DL 磁盘驱动器（0＝缺省，1＝A 等）

返回寄存器： AX 每簇扇区数（如果该驱动器无效，则置为 FFFFH）
BX 可用簇数
CX 每扇区字节数
DX 磁盘总簇数

注释：

本功能类似于功能 1BH 和 1CH，返回的基本信息可用来确定一个磁盘可用空间的数量。指定想要检查的磁盘驱动器后，可得到以下的原始信息：

- 每簇扇区数
- 可用簇数
- 每扇区字节数
- 驱动器总簇数

用这些，可求得可用空间数量：

（可用簇数）*（扇区数/簇）*（字节数/扇区）

用下面的公式可求得一个磁盘的可用空间：

（驱动器的簇数）*（扇区数/簇）*（字节数/扇区）

功能 1BH 和 1CH 返回的信息相似。

INT 21H	功能 37H	子功能 00H	V2
	取开关字符		

获取当前开关字符

调用寄存器： AH 37H
AL 00H

返回寄存器： AL FFH AL 子功能不在 0—3 范围之内
DL 开关字符

注释：

开关字符是 DOS 在分析指定命令开关字符串语法时使用。一般开关字符设置成斜线(/)，也可把它设置成其他字符。

INT 21H	功能 37H	子功能 01H	V2
	置开关字符		

允许重新设置开关字符

调用寄存器： AH 37H
AL 01H
DL 开关字符

返回寄存器： AL FFH AL 子功能不在 0—3 范围之内

注释：

开关字符是 DOS 在分析指定命令开关语法期间使用的字符。一般开关字符设置成斜线(/)，但是如果是应用需要，那么也可把它设置成其他字符。

使用该功能时，最好先判定并保存当前开关字符（子功能 0），程序结束后予以恢复。

INT 21H	功能 37H	子功能 02H	仅 V2
	读设备可用性		

表示设备是否必须冠以伪路径名/DEV/

调用寄存器：	AH	37H
	AL	02H
返回寄存器：	AL	FFH，AL 子功能不在 0—3 范围之内
	DL	0，/DEV/必须放在设备名前面
		≠0，/DEV/不必放在设备名前面

注释：

本功能仅在 V2 中出现，设备可用性标志控制一个类似 UNIX 的特性，它迫使所有设备处在伪目录 DEV 中，这样就允许文件和设备有相同的名称。当标志为 00H 时，设备只能在 DEV 目录中打开或关闭。任何非零标志都允许设备放在文件名前，意即系统中任何地方的文件不和任何设备同名。V3 中该标志永远为非零，子功能 3 不起作用。

INT 21H	功能 37H	子功能 03H	仅 V2
	置设备可用性		

决定设备是否必须冠以伪路径名/DEV/

调用寄存器：	AH	37H
	AL	03H
	DL	0，/DEV/必须放在设备名前面
		≠0，/DEV/不必放在设备名前面
返回寄存器：	AL	FFH，AL 子功能不在 0—3 范围之内
	DL	设备可用性标志（和输入相同）

注释：

本功能仅在 V2 中出现，设备可用性标志控制一个类似 UNIX 的特性，它迫使所有设备处在伪目录 DEV 中，这样就允许文件和设备有相同的名称。当标志为 00H 时，设备只能在 DEV 目录中打开或关闭。任何非零标志都允许设备放在文件名前，意即系统中任何地方的文件不和任何设备同名。V3 中该标志永远为非零，子功能 3 不起作用。

INT 21H	功能 38H	V2
	取/置当前国别信息	

获取当前国别信息；用 DOS V3.0 及更新版本还可设置国别信息

调用寄存器： AH 38H

AL 00H 获取当前国别信息
用 DOS V3.0 及更新版本
01H 到 FEH，指定国别代码小于 255
FFH 国别代码放在 BX 寄存器中

BX 如果 AL=FFH，则放置国别代码

DS：DX 指向存放信息的缓冲区

DX FFFFH=设置国别代码（DOS V3.0 以及更新版本）

返回寄存器： 如果成功，进位标志清零

BX 国别代码（仅 DOS V3.0 有）

DS：DX 指向返回的国别信息的指针

如果失败，进位标志置位

AX 错误代码

02H 无效国别（文件未找到）

注释：

该功能告诉用户程序许多与国别有关的信息显示参数。

当该功能获取与国别有关的信息时，返回一个指向带有该信息的 32 字节缓冲区的指针。在 DOS V3 和更新版本中，该功能也可用来设置供其他程序使用的国别信息。

国别代码往往是国际电话的前缀代码（DOS V3 和更新版本中）。某些代码（例如，美洲萨摩亚是 684，葡萄牙是 351）可在任何电话簿的前面找到。这里要注意的一点是数字可以超过 255。为了适应这个要求，该功能当 AL=FFH 时，用 BX 放国别代码。

表 1.12 给出了国别信息表的格式，用 DS：DX 指向它们。

表 1.12 国别信息缓冲区

偏移量	长度	含义
		——DOS V2——
00H	字	日期和时间格式 0=美国，月 日 年，时：分：秒 1=欧洲，日 月 年，时：分：秒 2=日本，年 月 日，时：分，秒
02H	字节	货币符号
03H	字节	零
04H	字节	千分隔符
05H	字节	零
06H	字节	小数分隔符
07H	字节	零
08H	18 字节	保留
		——DOS V3 或 V4——
00H	字	日期格式

续表

偏移量	长度	含义
		0=美国，月 日 年
		1=欧洲，日 月 年
		2=日本，年 月 日
02H	5 字节	货币符号字符串（ASCIIZ）
07H	字节	千分隔符
08H	字节	零
09H	字节	小数分隔符
0AH	字节	零
0BH	字节	日期分隔符
0CH	字节	零
0DH	字节	时间分隔符
0EH	字节	零
0FH	字节	货币格式
		00H=符号放在货币前，无空格
		01H=符号放在货币后，无空格
		02H=符号放在货币前，一个空格
		03H=符号放在货币后，一个空格
		04H=符号替换小数分隔符
10H	字节	小数位位数
11H	字节	时间格式
		第 0 位=0，12 小时时钟
		第 0 位=1，24 小时时钟
12H	双字	大小写映象调用地址
16H	字节	数据表分隔符
17H	字节	零
18H	8 字节	保留
		——DOS V5——
00H	字	日期格式
		0=美国，月 日 年
		1=欧洲，日 月 年
		2=日本，年 月 日
02H	5 字节	货币符号字符串（ASCIIZ）
07H	2 字节	千分隔符（ASCIIZ）
09H	2 字节	小数分隔符（ASCIIZ）
0BH	2 字节	日期分隔符（ASCIIZ）
0DH	2 字节	时间分隔符（ASCIIZ）
0FH	字节	货币格式
		00H=符号放在货币前，无空格
		01H=符号放在货币后，无空格
		02H=符号放在货币前，一个空格
		03H=符号放在货币后，一个空格
		04H=符号替换小数分隔符

续表

偏移量	长度	含义
10H	字节	小数位位数
11H	字节	时间格式
		第0位=0，12小时时钟
		第0位=1，24小时时钟
12H	双字	大小写映象调用地址
16H	2字节	数据表分隔符（ASCIIZ）
18H	10字节	保留

表中在偏移量12H处列出了大小写映象调用地址，这是一个格式过程的远程址(段地址：偏移)，该过程完成指定国别大于7FH的值从小写到大写的映像转换。调用映象过程时应把被映像字符放在AL寄存器中，调整后的值在AL寄存器中返回。

INT 21H 功能39H V2
创建子目录

在指定驱动盘和路径位置创建一个子目录

调用寄存器： AH 39H
DS：DX 指向ASCIIZ路径说明的指针

返回寄存器： 如果成功， 进位标志清零
如果失败， 进位标志置位
AX 错误代码
03H 路径未找到
05H 拒绝访问

注释：

DOS只提供了本功能和另外两个功能（功能3AH和3BH）来对目录项操作。这个功能允许创建一个新目录，如果有必要，可以带有目录的路径名和驱动器标志符。

如果要创建的目录已经存在，或路径名的某一部分不存在，或目录从根出发而根已满，那么本功能将返回一个错误码并且不会创建所需的目录。

在网络环境中，创建一个子目录必须具有创建访问权。

INT 21H 功能3AH V2
删除子目录

如果子目录空，那么删除该子目录。

调用寄存器： AH 3AH
DS：DX 指向ASCIIZ路径说明的指针

返回寄存器： 如果成功，进位标志清零

如果失败，进位标志置位

AX 错误代码

03H 路径未找到

05H 拒绝访问

10H 试图移动当前目录

注释：

这是对其他目录中目录项操作的三个功能之一。它删除指定的目录，但条件是被删除的目录存在且是空的，并且不是缺省目录时才行。

在网络环境中，删除一个子目录必须具有创建访问权。

INT 21H	功能 3BH	V2
	设置目录	

设置与指定字符串一致的当前或缺省目录。

调用寄存器： AH 3BH

DS：DX 指向 ASCIIZ 路径字符串的指针

返回寄存器： 如果成功，进位标志清零

如果失败，进位标志置位

AX 错误代码

03H 路径未找到

注释：

本功能允许用户把程序放在目录系统中的一个指定位置。

对于工作在特定目录下的程序，可在用本功能设置一个新目录区之前，用功能 47H 判定当前目录并保存其信息。当程序完成时，返回到原先目录。很少有子程序返回时这一简单的步骤。

INT 21H	功能 3CH	V2
	创建/截断文件（句柄）	

指定文件如果不存在，就创建它；如果存在，就截断为零长度

调用寄存器： AH 3CH

CX 文件属性

DS：DX 指向 ASCIIZ 文件说明的指针

返回寄存器： 如果成功，进位标志清零

AX 文件句柄

如果失败，进位标志设置

AX 错误代码

03H 路径未找到

04H 无可用句柄
05H 拒绝访问

注释：

这是基本的文件操作功能。它是面向句柄的功能，而功能 16H 是面向文件控制块（FCB）的功能，其操作一样。命名文件不存在时，本功能创建它；存在时，把长度截断为 0，文件中所有的数据均丢失。需要的文件用 ASCIIZ 字符串命名，可以包含驱动器和路径说明符。本功能返回一个 16 位的文件句柄，可用它访问该文件。新文件属性由 CX 寄存器设置。

表 1.13 描述了不同值所对应的文件类型（属性）。

表 1.13 文件类型代码

值	相匹配的文件类型
00H	普通文件
02H	隐含文件
04H	系统文件
06H	隐含文件且为系统文件

当不让截断功能丢失文件中所有的数据时，根据正在运行的 DOS 版本有三种选择：

- DOS V2，用功能 3DH 试图打开文件，如果功能调用失败，则调用本功能创建该文件
- DOS V3，用功能 5BH 创建文件，如果功能调用失败，则调用功能 3DH 打开文件
- DOS V4，也可以使用功能 6CH，只用它就可提供所有文件打开选择

不管用以上哪种方法，都需要仔细考虑本功能的用法。许多程序员因使用错误而丢失了许多重要的数据。

如果路径名的某一部分不存在，或文件在根目录处创建而根目录已满，或者存在一个相同名称的只读文件，那么本功能调用失败。

创建的普通文件允许读/写访问。如果需要，可用功能 43H 改变该文件的属性。不能用本功能创建子目录或卷标。

在网络环境中，创建或截断一个文件必须具有创建访问权。

INT 21H	功能 3DH 打开文件（句柄）	V2

打开指定目录下的文件，返回一个用来指示该打开文件的文件句柄（16 位字）

调用寄存器： AH 3DH
AL 访问方式（DOS V2）
访问和文件共享方式（DOS V3 及以上版本）
DS：DX 指向 ASCIIZ 文件说明的指针

返回寄存器： 如果成功，进位标志清零

AX 文件句柄

如果失败，进位标志设置

AX 错误代码

01H 无效功能

02H 文件未找到

03H 路径未找到

04H 无可用句柄

05H 拒绝访问

0CH 访问代码无效

注释：

要打开的文件用 ASCIIZ 字符串来指定文件名。本功能可访问普通、隐含或系统文件。寄存器 AL 用来告诉访问文件的方式。表 1.14 表示如何在 DOS V2 和 V3 中设置 AL 寄存器。

表 1.14 访问和文件共享方式

位	含义
—— DOS V2——	
76543210	
.....000	读访问
.....001	写访问
.....010	读/写访问
—— DOS V3 或 V4——	
76543210	
.....000	读访问
.....001	写访问
.....010	读/写访问
....x...	保留
.000....	共享方式——兼容方式
.001....	共享方式——拒绝读/写访问
.010....	共享方式——拒绝写访问
.011....	共享方式——拒绝读访问
.100....	共享方式——允许各种访问
0.......	由子进程继承
1.......	只适合当前进程

在 DOS V3 及以上版本，若为请求读/写访问，还可请求网络访问（文件共享方式），并指出被本进程执行的任何子进程是否可以继承对该文件的访问。

返回时，按需要的访问方式打开文件，除非文件找不到或者所需访问方式不允许（例如，不能用读/写访问方式访问一个只读文件）。如果文件成功打开，则读/写指针处于文件的开始。

在 DOS V2 中，只有 AL 寄存器的第 0—2 位有意义，其他位应置成 0。在 DOS V3 及以后版本中装有文件共享软件，AL 寄存器中的 4 位用于设置其他进程访问许可权（第 4—6 位，

共享方式；第7位，继承位)。这样设置的缺点是：当出现一个文件共享错误时，会出现严重错误INT 24H，带有错误代码02H（驱动器没有准备好)。

若继承位置1，则文件为打开它的进程所独用；子进程不会访问该文件。若该位置0，则文件打开以后，子进程可访问该文件。如果一个文件句柄被一个子进程继承或被一个进程复制，那么，其文件共享方式中的一切设定也被继承下来。

兼容方式（第3—7位置成0)，对于DOS V3以前写的大多数DOS软件，它就是普通方式，对以后版本写的许多软件也一样。软件在单个工作站上运行，当然就没有文件访问冲突。当引入网络软件、实现文件共享后，兼容方式不再适合于文件控制。

为了和网络环境中其他程序一起工作，程序在打开调用中必须使用共享方式，以指明编程任务范围之内的程序的访问权。用FCB功能打开的文件设定为处于兼容方式，除非打开时限制为只读访问、拒绝写共享方式；由句柄功能以只读方式打开的文件也拒绝写共享方式。所有其他的兼容方式拒绝所有外部文件访问。

为了正确使用本功能，程序员必须仔细考虑文件所需的访问，随便对文件写会破坏文件数据。访问方式包括：

- 拒绝读/写：该方式下打开的文件不能由网络上任何程序（或当前程序）再打开。控制数据库操作的临界刷新（Critical Updating）就需这种方式；
- 拒绝写：该方式下打开的文件仅能被其他程序打开读；
- 拒绝读：该方式下打开的文件不能被其他文件打开读；
- 不拒绝：不拒绝其他程序的访问（读或写)。

在网络环境中，经常有多个程序访问数据文件，若多个程序试图同时修改相同文件记录，数据库可能会被破坏。在不同程序间协调文件访问的机制超出了本书的范围，读者应当查阅有关网络数据库或操作系统的书。

INT 21H	功能 3EH 关闭文件（句柄）	V2

关闭一个用句柄功能打开的文件

调用寄存器： AH 3EH
BX 文件句柄

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误代码
06H 无效句柄

注释：

本功能关闭已用DOS句柄功能打开或创建的文件。句柄返回给系统使用，并且完成文件更新。如果文件作了变动，那么记录在目录项里的文件日期亦被修改。

作为一种良好的编程风格，一个程序总是应关闭它所打开的任何文件。虽然当程序终止时，DOS会自动关闭活动的文件句柄，但这样影响了程序的可移植性。

注意：关闭文件句柄0要当心，它是标准输入设备（一般为键盘)。如果偶然关闭了文件

句柄 0，就只能立即重新打开 CON 设备，否则将无法和键盘通信。

INT 21H	功能 3FH 读文件或设备（句柄）	V2

从文件句柄参数指定的文件或设备中读取数据；这些数据被传送到指定内存位置

调用寄存器：
- AH 3FH
- BX 文件句柄
- CX 字节数
- DS：DX 指向缓冲区的指针

返回寄存器：
- 如果成功，进位标志清零
  - AX 读取的字节数
- 如果失败，进位标志设置
  - AX 错误代码
    - 05H 拒绝访问
    - 06H 无效句柄

注释：

这是一个基本的操作，它从文件中读规定数量的字节到指定缓冲区。如果读成功完成，而 AX 小于 CX，则是表示未读完 CX 所指定的字节数就读到了 EOF，即出现了部分读。如果一调用此功能就遇到 EOF，那么进位标志置位，而 AX 寄存器为 0。

如同其他的文件句柄调用，设备完全可以当作文件来对待。可用本功能从字符设备（如键盘）上读。但是当处理一台字符设备时，要受到某些特殊的限制，如果字符设备处于虚拟方式（参见功能 44H），那么此读以回车终止（即只读一行）。

在网络环境中，读一个文件或设备必须具有读访问权。

INT 21H	功能 40H 写文件或设备（句柄）	V2

数据写到句柄指定的文件

调用寄存器：
- AH 40H
- BX 文件句柄
- CX 要写的字节数
- DS：DX 指向待写的缓冲区的指针

返回寄存器：
- 如果成功，进位标志清零
  - AX 已写的字节数
- 如果失败，进位标志设置
  - AX 错误代码
    - 05H 拒绝访问

06H 无效句柄

注释：

用文件句柄功能写一个文件，只需简单地指定文件句柄、字节数并指向数据缓冲区，然后调用本功能把这些字节写到文件的当前位置。

寄存器 AX 返回已写的字节数。如果调用失败，则返回一个错误代码。一般 AX 中返回的字节数和要写的字节数（CX 寄存器中）相同。如果写成功且 AX 小于 CX，那么只有部分记录被写入。磁盘空间用完了就会出现部分写；这时，可用功能 36H、1BH 或 1CH 检查可用空间。如果文件标识为只读，则返回一个出错代码。

在网络环境中，写一个文件或设备必须具有写访问权。

INT 21H 功能 41H V2
删除文件

从系统中删除指定的文件

调用寄存器： AH 41H
DS：DX 指向 ASCIIZ 文件说明的指针

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志设置
AX 错误代码
02H 文件未找到
05H 拒绝访问

注释：

本功能删除文件的方法是在目录项中把文件名的第一个字符标识为 E5H。如果删除后没有创建或改变别的文件，就可以恢复“被删除的”文件。目录项中其他内容没有任何改变，分配给该文件的簇返回给系统重新使用，实际文件没有改写。

不像文件控制块（FCB）的删除功能（13H），这里不允许通配符。如果想删除与使用通配符的文件名对应的一组文件，须用查找功能（4EH、4FH）将文件逐个定位。该功能允许访问子目录中的文件。

如果文件存在但只可读，或文件找不到，则该功能调用失败。

为了在网络环境中删除一个文件，必须具有创建访问权。

INT 21H 功能 42H V2
移动文件指针

改变文件指针的当前位置，即文件指针指到相对于文件头、文件尾或当前位置的某个位置

调用寄存器： AH 42H
AL 方式代码
00H 偏移从文件头开始

01H 偏移从文件当前位置开始
02H 偏移从文件尾开始
BX 文件句柄
CX 偏移值的高 16 位
DX 偏移值的低 16 位

返回寄存器： 如果成功，进位标志清零
DX：AX 新文件指针的位置
如果失败，进位标志设置
AX 错误代码
01H 无效功能（文件共享）
06H 无效句柄

注释：

用本功能调节文件读/写指针，指向从文件头、尾或当前位置开始的一个新位置。偏移值可指定为一个 32 位数(范围大到 4096M)。V4 以前，实际上不能用这样尺度的文件，因为 DOS 把单个硬盘分区容量限制在 32M。V4 中，这一限制被取消了。当文件指针移动时，文件指针指向从文件读或写数据下一个点。

除了设置文件位置外，本功能还用于确定文件大小。通过置寄存器 AL 为 2（相对于文件尾）并把 CX 和 DX 寄存器置为 0（相对文件尾的偏移值）来获取文件大小。AX 和 DX 寄存器中返回用字节表示的文件实际大小。当然，这使指针留在文件尾。如果不满意，必须在读或写之前重新把位置设置到所需的地方。

该功能另外一个重要用途是添加功能，即在文件尾打开并补充。可用功能 3DH 打开文件，然后，以和确定文件大小（AL＝2，CX＝DX＝0）相同的方法，用本功能立即把读/写指针指向文件的结尾处。

用本功能可把文件指针指向文件头前面或文件尾后面的一个位置。把文件指针指向文件尾后面不会出错，除非试图从这个不存在的地方读东西；向超过文件尾的地方写数据，将导致重新给文件分配空间，以使文件足够大来适应写。把文件指针指向文件头前面，试图读/写时会出错。

如果网络系统中有一个文件，它处于拒绝读且拒绝写的共享方式时，则文件指针信息由拥有该文件的计算机来调节。如果文件处于其他任何共享方式，那么文件指针信息保存在远程计算机中。

INT 21H	功能 43H	子功能 00H	V2
	取文件属性		

获取文件的属性

调用寄存器： AH 43H
AL 00H 获取文件属性
DS：DX 指向 ASCIIZ 文件说明的指针

返回寄存器： 如果成功，进位标志清零

CX 属性字节（见表）

如果失败，进位标志设置

AX 错误代码

01H 无效功能（文件共享）

02H 文件未找到

03H 路径未找到

05H 拒绝访问

注释：

一个目录项的文件的属性位映像如表 1.15 所示：

表 1.15 目录项文件属性

位 7654 3210	含义
....1	只读
.... ..1.	隐含
.... .1..	系统
.... 1...	卷标
...1	目录
..1.	档案
xx..	未用

对于卷标或子目录，目录项的属性用本功能不能访问。

INT 21H	功能 43H	子功能 01H	V2
	置文件属性		

设置一个文件的属性

调用寄存器： AH 43H

AL 01H 设置文件属性

CX 文件新属性（见表 1. 16）

DS：DX 指向 ASCIIZ 文件说明的指针

返回寄存器： 如果成功，进位标志清零

如果失败，进位标志设置

AX 错误代码

01H 无效功能

02H 文件未找到

03H 路径未找到

05H 拒绝访问

注释：

文件的属性可以设置成表 1.16 所示的值：

表 1.16 文件属性值

位 654 3210	含义
... ...1	只读
... ..1.	隐含
... .1..	系统
.1.	档案

用本功能不能设置子目录或卷标的属性。要创建一个卷标，必须使用文件控制块（FCB）文件创建功能和一个扩充的 FCB。功能 39H 是唯一的创建目录功能。

在网络环境下，除了档案位外，要改变任何文件属性位都必须具有创建访问权。改变档案位没有任何约束。

INT 21H 功能 44H 设备驱动程序控制（IOCTL） V2

控制信息传递到设备驱动程序或从设备驱动程序取回

调用寄存器：
- AH 44H
- AL 设备子功能代码（见表 1.17）
- BX 句柄(子功能代码 00H，01H，02H，03H，06H，07H，0AH，0CH 和 10H)
- BL 驱动器代码，0=缺省，1=A，等（子功能代码 04H，05H，08H，09H，0DH 和 11H）
- CX 读或写的字节数
- CH 种类代码（子功能代码 0CH，0DH，10H 和 11H）
- CL 功能代码（子功能代码 0CH，0DH，10H 和 11H）
- DS：DX 指向缓冲区的指针（子功能代码 02H—05H）
- DS：DX 指向参数块的指针（子功能代码 0CH，0DH，10H 和 11H）
- DX 设备信息（子功能代码 01H）（见表 1.17）

返回寄存器：如果成功，进位标志清零
- AX 传送的字节数（子功能代码 02H—05H）
- AL 状态（子功能代码 06H—07H）
  - 00H 没有准备好
  - FFH 准备好
- AX 值(子功能代码 08H)
  - 00H 可更换
  - 01H 不可更换

DX 设备信息（子功能代码 00H）

如果失败，进位标志设置

AX 错误代码

01H 无效功能（文件共享）

04H 无可用句柄

05H 拒绝访问

06H 无效句柄

0DH 数据无效

0FH 无效驱动器

注释：

IOCTL 功能是 DOS 下综合性最强的可用功能之一。本功能有 18 个不同的子功能，表 1.17 给出了这些功能的总览以及这些功能正式起作用的 DOS 版本。传送的信息的意义依赖于被选中的专用设备驱动程序。

表 1.17 设备功能代码

AL	含义	DOS 版本
00H	取设备信息	2.0
01H	置设备信息	2.0
02H	字符设备读	2.0
03H	字符设备写	2.0
04H	块设备读	2.0
05H	块设备写	2.0
06H	取输入状态	2.0
07H	取输出状态	2.0
08H	块设备可变?	3.0
09H	块设备本地/远程?	3.1
0AH	句柄本地/远程?	3.1
0BH	置共享重试次数	3.0
0CH	通用句柄 I/O 控制	3.2
0DH	通用块设备 I/O 控制	3.2
0EH	取逻辑驱动器映象	3.2
0FH	置逻辑驱动器映象	3.2
10H	查询 IOCTL 句柄	5.0
11H	查询 IOCTL 设备	5.0

IOCTL 功能是一个通用设备驱动器接口程序。其目的不是传送数据而是和驱动器通信，告诉它如何工作。

INT 21H 功能 44H 子功能 00H V2

取设备信息

获取由句柄指定的设备或文件信息

调用寄存器： AH 44H
AL 00H
BX 句柄

返回寄存器： 如果成功，进位标志清零
DX 设备信息（见表 1.18）
如果失败，进位标志设置
AX 错误代码
01H 无效功能
05H 拒绝访问
06H 无效句柄

注释：

DX 寄存器从系统返回关于字符设备或文件的代码化信息，字符设备或文件由 BX 寄存器中的文件句柄指定。表 1.18 列出了这些代码和意义。句柄必须指向打开的文件或字符设备。

表 1.18 设备信息代码

位 FEDCBA98	76543210	含义
		——字符设备——
....	1	标准输入设备
....	......1.	标准输出设备
....	.....1..	空设备
....	....1...	时钟设备
....	...0....	设备不支持 INT 28H
....	...1....	设备支持 INT 28H
....	..0.....	虚拟方式
....	..1.....	原始(二进制)方式
....	.0......	文件输入结束
....	.1......	文件输入没有结束
....	1.......	字符设备
..xxxxxx	....	保留
.1......	....	设备能处理子功能 02H 和 03H 发送的控制字符串，这一位只可读，不可设置
x.......	....	保留
		——块设备(磁盘文件)——
....	..xxxxxx	块设备号(对于第一台块驱动器 0=A,1=B 等)
....	.0......	文件已被写入
....	.1......	文件没被写入
....	0.......	块设备(磁盘文件)
..xxxxxx	....	保留，调用时必须置零
.1......	....	设备能处理子功能 02H 和 03H 发送的控制字符串，该位只能读，不能设置
x.......	....	保留

对于字符设备，第5位特别有用。UNIX程序员处理终端设备时十分熟悉两个术语：虚拟方式（Cooked Mode）和原始方式（Raw Mode）。在DOS中，虚拟方式意思是处理Ctrl-C、Ctrl-P、Ctrl-Q、Ctrl-S和Ctrl-Z，并且发现回车后输入终止，而不是在找到指定字符数后终止；虚拟方式是全编辑属性方式，许多参考手册把它描述成唯一的控制台输入方式。

在原始方式中，所提供的I/O系统驱动程序忽略这些字符的特定意义，在从读操作返回之前一直等待，直到指定的字节数收满为止。收到的所有字符直接传递到应用程序，而不通过I/O系统或通过DOS做任何解释（变形）。在输出处理中存在相似的区别（TAB字符扩展，行输入前回车的自动加入等）。

BX寄存器中的句柄必须指向一个打开的文件或设备；否则，该功能返回错误代码06H（无效句柄）。

返回的DX寄存器的第8—15位对应于设备驱动程序属性字的相同位。

INT 21H	功能44H 子功能01H 设置设备信息	V2

本功能是子功能00H的补充，仅适合字符设备，用于设置设备信息代码。

调用寄存器： AH 44H
AL 01H
BX 句柄
DX 设备数据字

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误代码
01H 无效功能
05H 拒绝访问
06H 无效句柄
0DH 数据无效

注释：

只可用子功能01H对字符设备设置设备数据字的限定部分。一般，本功能只修改第5位。有关原始方式和虚拟方式的解释见子功能00H。

如果DH寄存器非零，返回错误代码01H（无效功能）。句柄须指向一个打开的设备；如果句柄指向文件，不作任何更新。

表1.19给出了设备数据字（DX）的解释。

表1.19 设备数据字

位		
FEDCBA98	76543210	含义
.....	1	标准输入设备
.....	.....1.	标准输出设备

续表

位		
FEDCBA98	76543210	含义
.....	.....1..	空设备
.....	....1...	时钟设备
.....	...1....	设备支持 INT 28H
.....	..0.....	虚拟方式
.....	..1.....	原始(二进制)方式
.....	.0.....	文件输入结束
.....	1.....	字符设备
xxxxxxxx	.....	保留

INT 21H 功能 44H 子功能 02H V2
设备 IOCTL 读

从驱动程序读取控制字符串信息供调用程序使用

调用寄存器： AH 44H
AL 02H
BX 句柄
CX 待取字节数
DS：DX 指向数据缓冲区的指针

返回寄存器： 如果成功，进位标志清零
AX 传送的字节数
如果失败，进位标志置位
AX 错误代码
01H 无效功能
05H 拒绝访问
06H 无效句柄
0DH 数据无效

注释：

关于驱动程序的任何信息都可以通过控制字符串传递到调用程序。这些信息取决于驱动程序的支持，其格式和内容没有一定标准。对请求的响应取决于驱动程序。

子功能 00H 的第 0EH 位表示驱动程序是否提供或响应控制字符串。

INT 21H 功能 44H 子功能 03H V2
设备 IOCTL 写

发送控制字符串信息到驱动程序

调用寄存器： AH 44H

AL 03H
BX 句柄
CX 发送的字符数
DS：DX 指向数据缓冲区的指针

返回寄存器： 如果成功，进位标志清零
AX 传送字节数
如果失败，进位标志置位
AX 错误代码
01H 无效功能
05H 拒绝访问
06H 无效句柄
0DH 数据无效

注释：

关于驱动程序的任意信息都可以通过控制字符串传递给驱动程序。这些信息取决于驱动程序的支持，其格式和内容没有一定标准。对请求的响应取决于驱动程序。该功能通常用来向驱动程序传递配置信息，如波特率或字长。

子功能 00H 的第 0EH 位表示驱动程序是否提供或响应控制字符串。

INT 21H	功能 44H 子功能 04H 块驱动程序 IOCTL 读	V2

从块驱动程序（磁盘类型）获取控制信息

调用寄存器： AH 44H
AL 04H
BL 驱动器号
CX 待取字符数
DS：DX 指向数据缓冲区的指针

返回寄存器： 如果成功，进位标志清零
AX 传送的字节数
如果失败，进位标志置位
AX 错误代码
01H 无效功能
05H 拒绝访问
06H 无效句柄
0DH 数据无效

注释：

关于块驱动程序的任何信息都可用本子功能通过控制字符串获取。它可以是驱动程序支持的状态信息或任何其他信息，其格式和内容没有一定标准。对请求的响应取决于驱动程序。

频繁使用该子功能时，需考虑块设备是否准备好。对 CDROM、磁带或其他的块设备可以

先进行查询。

子功能 00H 的第 0EH 位表示驱动程序是否提供或响应控制字符串。本功能对块设备并不是必须的，如果驱动程序不支持它，则返回错误代码 01H（无效功能）。

INT 21H	功能 44H 子功能 05H 块设备 IOCTL 写	V2

向块驱动程序（磁盘类型）发送控制字符信息

调用寄存器：
- AH 44H
- AL 05H
- BL 驱动器号
- CX 待发送字符数
- DS：DX 指向数据缓冲区的指针

返回寄存器：
- 如果成功，进位标志清零
  - AX 传送字节数
- 如果失败，进位标志置位
  - AX 错误代码
    - 01H 无效功能
    - 05H 拒绝访问
    - 06H 无效句柄
    - 0DH 数据无效

注释：

关于块驱动程序的任何信息都可通过控制字符串传给块设备。它可以是驱动程序支持的命令或任何其他信息，其格式和内容没有一定标准。

对请求的响应取决于驱动程序。一些非 I/O 设备功能频繁使用本子功能，如磁带倒带和磁盘弹出。

子功能 00H 的第 0EH 位表示驱动程序是否提供或响应控制字符串。本子功能对块设备并不是必须的，如果驱动程序不支持它，则返回错误代码 01H（无效功能）。

INT 21H	功能 44H 子功能 06H 取输入状态	V2

返回设备或文件的输入状态。

调用寄存器：
- AH 44H
- AL 06H
- BX 句柄

返回寄存器：
- 如果成功，进位标志清零
  - AL 输入状态代码（见表 1.20）

如果失败，进位标志置位

AX 错误代码

01H 无效功能

05H 拒绝访问

06H 无效句柄

注释：

使用本功能可以检查特定设备或文件是否已准备好进行输入操作。除了使用功能 42H 定位在 EOF 的文件外，其他文件可以用本功能来检查 EOF，或者检查字符设备是否已准备好进行操作。表 1.20 给出了输入状态代码（寄存器 AH）的解释。

表 1.20 输入状态代码

代码	文件	设备
00H	在 EOF	没有准备好
FFH	不在 EOF	准备好

特殊情况：

如果一个文件用 INT 21H 功能 42H 定位在 EOF 处，本功能不能返回 EOF。

INT 21H 功能 44H 子功能 07H V2

取输出状态

返回设备或文件输出状态

调用寄存器： AH 44H

AL 07H

BX 句柄

返回寄存器： 如果成功，进位标志清零

AL 输出状态代码（见表 1.21）

如果失败，进位标志置位

AX 错误代码

01H 无效功能

05H 拒绝访问

06H 无效句柄

注释：

使用本功能可以检查特定设备或文件是否已准备好进行输出操作。如表 1.21 所示，对输出，文件总是返回准备好；字符设备则不是这样（即：对于文件，返回 00F 或 FFH 均表示可输出）。

表 1.21 输出状态码

代码	文件	设备
00H	在 EOF	没有准备好
FFH	不在 EOF	准备好

INT 21H 功能 44H 子功能 08H V3
块设备是否可更换

用来确定块设备是否可更换

调用寄存器： AH 44H
AL 08H
BL 驱动器号（0=缺省，1=A 等）

返回寄存器： 如果成功，进位标志清零
AX 00H 可更换介质
01H 不可更换介质
如果失败，进位标志置位
AX 错误代码
01H 无效功能
0FH 无效驱动器

注释：

需要在一个特殊设备上放置数据文件或覆盖的应用程序，可用本功能确定设备是否可更换。如果所需文件不在设备上而且设备是可更换的，那么程序应提示用户放入正确的磁盘。

设备驱动程序属性字的第 0BH 位表示驱动程序是否能支持本功能。有些驱动程序不支持，这时返回错误代码 01H。

INT 21H 功能 44H 子功能 09H V3.1
块设备是本地/远程

确定块设备是本地还是远程设备

调用寄存器： AH 44H
AL 09H
BL 驱动器号（0=缺省，1=A 等）

返回寄存器： 如果成功，进位标志清零
DX 设备属性字
第 12 位=1，驱动器是远程
第 12 位=0，驱动器是本地
如果失败，进位标志置位
AX 错误代码

01H 无效功能
0FH 无效驱动器

注释：

如果网络还没有开始运行，则本功能返回错误代码 01H，即无效功能。

作为一种良好的编程风格，应避免使用本功能，以免程序依赖于设备在网络中的位置。但是，某些未写入文档功能调用仅能对本地文件正确地操作，而不能用于远程文件。如果在程序中必须使用这类功能，那么，用本功能可以避免某些错误情况。

如果 DX 的第 12 位是 0，则表示驱动器是本地的，DX 中返回的是驱动程序设备头的信息，其中各位的意义如下：

第 1 位 1＝驱动器用 32 位扇区寻址
第 6 位 1＝驱动器支持功能 44H 子功能 0DH，0EH 和 0FH
第 7 位 1＝驱动器支持功能 44H 子功能 11H
第 9 位 1＝驱动器是本地的，但被网络中的其他计算机共享
第 11 位 1＝驱动器支持功能 44H 子功能 08H
第 13 位 1＝驱动器在 FAT 中需要介质描述符
第 14 位 1＝驱动器支持功能 44H 子功能 04H 和 05H
第 15 位 1＝替换驱动器（例如，由 SUBST 命令设置）

其他所有位都为零。

INT 21H	功能 44H 子功能 0AH	V3.1
	句柄是本地/远程	

确定句柄是本地还是远程

调用寄存器： AH 44H
AL 0AH
BX 句柄

返回寄存器： 如果成功，进位标志清零
DX 设备属性
第 15 位＝1，句柄是远程
第 15 位＝0，句柄是本地
如果失败，进位标志置位
AX 错误代码
01H 无效功能
06H 无效句柄

注释：

如果网络没有开始运行，则本功能返回错误代码 01H，即无效功能。

作为一种良好的编程风格，应避免使用本功能，以免程序依赖于设备在网络中的位置。但是，某些未写入文档功能调用仅能对本地句柄正确地操作，而不能用于远程句柄。如果在程序中必须使用这类功能，那么，用本功能可以避免某些错误情况。

如果 DX 的第 7 位是 0，则句柄等同一个文件，DX 中各位的意义如下：

第 0—5 位	驱动器号（0=A，1=B 等）
第 6 位	1=没有写文件
第 12 位	1=没有继承
第 14 位	1=关闭处没有设置日期和时间

其他所有的位为零。

如果 DX 的第 7 位是 1，则句柄是一个设备句柄，DX 中各位的意义如下：

第 0 位	1=控制台输入设备
第 1 位	1=控制台输出设备
第 2 位	1=空设备
第 3 位	1=时钟设备
第 4 位	1=特殊设备
第 5 位	1=二进制方式
	0=ASCII 方式
第 6 位	0=如果设备被读，那么返回 EOF
第 11 位	1=网络假脱机系统
第 12 位	1=没有继承
第 13 位	1=命名管道

其他所有位都为零。

INT 21H　　功能 44H　　子功能 0BH　　V3.0

设置共享重试次数

改变网络文件共享重试参数

调用寄存器：　AH　　44H

AL　　0BH

CX　　重试间暂停

DX　　重试次数

返回寄存器：　如果成功，进位标志清零

如果失败，进位标志置位

AX　　错误代码

01H　无效功能

注释：

当有多个 PC 机在网络中运行时，重试参数与文件锁定机制相关。这里假定文件锁定是暂时的，并且在一个简短修改以后锁定被清除。如果第一次尝试时文件被锁定，那么这样建立的机制将自动重试对一个文件的访问。

两个参数（重试次数和重试之间的暂停）依赖于系统，CPU 和时钟速度的不同对暂停的实际长度有重大影响。CX 寄存器通过设定执行严格定时值循环的次数来控制暂停。无论何时被调用，定时循环都重复 65536 次。很显然，重试次数是报告失败以前试图访问的次数。缺

省时 PAUSE=1，RETRY=3。

这些参数可协助系统减少文件共享问题。如果希望文件被锁定时间更长，当再一次访问该文件时可扩大暂停。如果改变了任何缺省值，要注意恢复缺省值，以防对其他程序起副作用。

INT 21H	功能 44H 子功能 0CH	V3.2
	句柄通用 I/O 控制 (Generic I/O Control for Handle)	

在 DOS V3.2 中，它用以设置或获取面向字符设备的重复次数（Iteration Count）
在 DOS V3.3 及以上版本，该子功能还完成代码页切换
在 DOS V5.0 中，该功能还可获取和设置显示方式

调用寄存器：
- AH 44H
- AL 0CH
- BX 句柄
- CH 种类代码（设备类型）
  - 【DOS V3.2】
    - 05H 打印机
  - 【DOS V3.3 及以上】
    - 00H 未知
    - 01H COMx
    - 03H CON
    - 05H LPTx
- CL 辅助功能（Minor Function）代码
  - 【DOS V3.2 及以上】
    - 45H 设置重复次数
    - 65H 获取重复次数
  - 【DOS V3.3 及以上】
    - 4AH 选择代码页
    - 4CH 准备开始
    - 4DH 准备结束
    - 6AH 查询选择
    - 6BH 查询准备表
  - 【DOS V5.0】
    - 5FH 设置显示方式
    - 7FH 获取显示方式
- DS：DX 指向重复次数字的指针（辅助功能代码 45H 和 65H）
  指向参数块的指针（辅助代码 4AH，4CH，4DH，5FH，6AH，6BH 和 7FH）

返回寄存器：
- 如果成功，进位标志清零
- 如果失败，进位标志置位

AX 错误代码

01 无效功能

注释：

重复次数规定了一个操作在放弃前尝试的次数。DOS V3.2 只允许种类代码 05H（打印机）。

DOS V3.3 及以上版本，本子功能转而为设备处理代码页切换。辅助功能包括以下几个：

- ●准备开始（4CH） 告诉驱动程序准备好用子功能 03H 装入代码页字形。一个特殊的开始操作是“刷新”，它把所有代码页 ID 置成 FFFFH；
- ●准备结束（4DH） 告诉驱动程序代码页字形的装入已完成；
- ●选择代码页（4AH） 选择要使用的代码页；
- ●设置显示方式（5FH） 为设备设置显示方式；
- ●查询被选代码页（6AH） 从设备判定代码页的状态；
- ●查询准备表（6BH） 判定设备上的代码页表；
- ●取显示方式（7FH） 获取设备显示方式；

表 1.22 定义了由 DS：DX 指向的参数块。

表 1.22 参数块

字节	含义
	——辅助功能 4AH、4DH、6AH——
0—1	后面数据的长度
2—3	代码页 ID
	——辅助功能 4CH——
0—1	标志
2—3	参数块长度（从这点往后）
4—5	代码页数
.	
.	代码页标志
.	
	——辅助功能 6AH——
0—1	参数块长度（从这点往后）
2—3	硬件代码页数
.	
.	硬件代码页标志
.	
n—n+1	准备的代码页数
.	
.	准备的代码页标志
.	
	——辅助功能 5FH 和 7FH——
0	信息级别（必须是零）
1	保留

续表

字节	含义
2—3	参数块长度（从这点往后）
4—5	控制标志：
	0＝关闭亮度
	1＝打开亮度
6	显示方式
	1＝文本方式
	2＝图形方式
7	保留
8—9	颜色数
0AH—0BH	屏幕宽度（象素个数，仅对图形方式）
0CH—0DH	屏幕长度（象素个数，仅对图形方式）
0EH—0FH	列
10—11	行

INT 21H　　功能 44H　　辅助功能 0DH　　V3.2

块设备通用I/O控制

在块设备上集中了6个输入/输出功能来处理特殊功能

调用寄存器：
- AH　44H
- AL　0DH
- BL　驱动器号
- CH　种类代码
  - 08H　磁盘驱动器
- CL　辅助功能代码
  - 40H　设置块设备参数
  - 41H　在逻辑驱动器上写磁道
  - 42H　在逻辑驱动器上格式化并检验磁道
  - 46H　设置介质ID
  - 60H　获取块设备参数
  - 61H　在逻辑驱动器上读磁道
  - 62H　在逻辑驱动器上检验磁道
  - 66H　获取介质ID
  - 68H　读介质类型
- DS：DX 指向参数块的指针

返回寄存器：
- 如果成功，进位标志清零
- 如果失败，进位标志置位
  - AX　错误代码

01H 无效功能
02H 无效驱动器

注释：

本功能用以扩充控制块设备的能力。在设备无关类型中，通过该IOCTL 功能可以控制许多原语操作。下面逐个分析每个辅助功能。

对于给定设备，在其他辅助功能调用以前，必须先调用辅助功能 40H（设置设备参数）。

辅助功能 40H：设置设备参数

该辅助功能的参数块（见表 1.23）表示块设备的完整格式，包括物理特征、介质类型等。

表 1.23 参数块格式

字节偏移量	含义
00H	特殊功能代码（表 1.24）
01H	设备类型代码（表 1.25）
02—03H	设备属性代码（表 1.26）
04—05H	磁道柱面数
06H	介质类型代码（表 1.27）
07—25H	设备 BPB（表 1.28）
26H—?	磁道格式表（表 1.29）

表 1.24 特殊功能代码

位 7654 3210	含义
....0	BPB 项是一个新 BPB
....1	使用当前 BPB
.... ..0.	使用参数块中的所有项
.... ..1.	仅使用磁道格式项
.... .0..	磁道中的扇区可以是不同大小
.... .1..	磁道中的扇区大小都相同
0000 0...	保留

表 1. 25 设备类型代码

代码	含义
00H	320/360K，5.25 英寸① 磁盘
01H	1.2M，5.25 英寸磁盘
02H	720K，3.5 英寸磁盘
03H	单密度，8 英寸磁盘
04H	双密度，8 英寸磁盘
05H	固定磁盘
06H	磁带驱动器
07H	其他块设备

① 1 英寸＝0.0254m

表 1.26 设备属性代码

位 7654 3210	含义
....0	可替换存储器
....1	不可替换存储器
.... ..0.	设备不指明改变行状态(驱动器门不可锁)
.... ..1.	设备指明改变行状态(驱动器门可锁)
xxxx xx..	保留

表 1.27 介质类型代码

代码	含义
00H	1.2M，5.25 英寸磁盘
01H	320/360K，5.25 英寸磁盘

表 1.28 BIOS 参数块（BPB）格式

字节偏移量	域宽	含义
00H	字	每扇区字节数
02H	字节	每簇扇区数
03H	字	扇区 0 开始的保留扇区数
05H	字节	FAT 数
06H	字	根目录项最大数
08H	字	扇区总数
0AH	字节	介质描述符
0BH	字	每 FAT 扇区数
0DH	字	每磁道扇区数
0FH	字	磁头数
11H	双字	隐含扇区数
15H	11 字节	保留

表 1.29 磁道格式表

长度	含义
字	磁道中的扇区数
字	磁道中的第一扇区号
字	磁道中的第一扇区大小
.	
.	
.	
字	磁道中的最后一个扇区号
字	磁道中的最后一个扇区大小

辅助功能 41H： 写磁道

写磁道指定一个磁道所有重要参数（磁头、柱面、扇区、扇区数和数据位置），计算时，扇区号和柱面号从零开始。

表 1.30 参数块——写磁道

偏移量	含义
00H	特殊功能=0
01—02H	要使用的磁头号
03—04H	要使用的磁道柱面号
05—06H	要使用的第一扇区
07—08H	要传递的扇区数
09—0CH	指向数据传送缓冲区的指针

辅助功能 42H： 格式化并检验磁道

这一功能格式化和检查磁盘上的磁道。只需指明所用的磁头和柱面，所用其他工作由驱动程序完成。

表 1.31 参数块——磁道格式化

偏移量	含义
00H	特殊功能=0
01—02H	要使用的磁头号
03—04H	要使用的磁道柱面号

表 1.32 参数块——检验格式化状态

偏移量	含义
00H	特殊功能=1
01—02H	要使用的磁头号
03—04H	要使用的磁道柱面号

完成后，如果检查特殊功能项状态，则可能返回如下值：

0=ROM BIOS 支持，磁头/柱面允许

1=ROM BIOS 不支持

2=指定磁头/柱面号不允许

3=驱动器是空的

辅助功能 46H： 置介质 ID

本功能对指定驱动器设置序号、卷标和文件系统类型（参数块见表 1.33）。

表 1.33 参数块——置介质 ID

偏移量	含义
00—01H	信息级别
02—05H	序号
06—10H	ASCII 卷标
11—18H	文件系统类型
"FAT12"	12 位 FAT
"FAT16"	16 位 FAT

辅助功能 60H： 取参数

本功能和辅助功能 40H 配对。它从驱动程序检索有关设备的信息，其参数块格式和辅助功能 40H 相同。

辅助功能 61H： 读磁道

本功能把磁道读到参数块提供的内存缓冲区中。象写磁道辅助功能一样，向驱动程序提供定位信息。

表 1.34 参数块——读磁道

偏移量	含义
00H	特殊功能＝0
01—02H	要使用的磁头号
03—04H	要使用磁道柱面号
05—06H	要使用的第一扇区
07—08H	要传送的扇区数
09—0CH	指向数据传送缓冲区的指针

辅助功能 62H： 校验磁道

该功能完成辅助功能 42H（格式化和校验磁道）中的磁道校验操作。

表 1.35 参数块——校验磁道

偏移量	含义
00H	特殊功能＝0
01—02H	要使用的磁头号
03—04H	要使用的磁道柱面号

辅助功能 66H： 取介质 ID

本功能获取指定驱动器序号、卷标和文件系统类型。

表 1.36 参数块——取介质 ID

偏移量	含义
00—01H	信息级别
02—05H	序号
06—10H	ASCII 卷标
11—18H	文件系统类型
"FAT12"	12 位 FAT
"FAT16"	16 位 FAT

辅助功能 68H： 读介质类型

本功能返回指定驱动器介质类型。

表 1.37 参数块——读介质类型

偏移量	含义
00H	缺省标志
	0=介质不是缺省类型
	1=介质是缺省类型
01H	介质类型
	2=720K 磁盘
	7=1.44M 磁盘
	9=2.88M 磁盘

INT 21H 功能 44H 子功能 0EH V3.2

取逻辑驱动器映象

确定是否有多个逻辑驱动器名分配给设备

调用寄存器： AH 44H

AL 0EH

BL 驱动器号

返回寄存器： 如果成功，进位标志清零

AL 驱动器号

0=只分配了一个逻辑驱动器

1=A，2=B 等

如果失败，进位标志置位

AX 错误代码

01H 无效功能

05H 拒绝访问

0FH 无效驱动器

注释：

如果多个驱动器标志用于某设备，那么，本功能调用返回的驱动器号告知最后用来访问驱动器的驱动器标志。

INT 21H	功能 44H 子功能 0FH	V3.2
	置逻辑驱动器映象	

设置要用来访问该设备的逻辑驱动器名

调用寄存器： AH 44H
AL 0FH
BL 驱动器号

返回寄存器： 如果成功，进位标志清零
AL 驱动器号
0＝只分配了一个逻辑驱动器
1＝A，2＝B等
如果失败，进位标志置位
AX 错误代码
01H 无效功能
05H 拒绝访问
0FH 无效驱动器

注释：

当在两个磁盘之间拷贝文件，每个磁盘对应于一个不同的逻辑设备，但两者必须用相同的物理设备时，若对目前不在驱动器上的设备做I/O，系统就提示改换磁盘。本功能强行切换而不须取得操作系统的提示。

该功能设置下一个驱动器字母，该字母指向这一设备。然后DOS不会提示插入磁盘（Insert Disk）。子功能0EH确定最近用于访问该设备的逻辑驱动器名称。

INT 21H	功能 44H 子功能 10H	V5
	查询IOCTL句柄	

判定用句柄指定的设备是否支持指定的IOCTL功能

调用寄存器： AH 44H
AL 10H
BX 设备句柄
CH 种类代码
CL 功能代码
DS：DX 指向参数块的指针

返回寄存器： 如果成功，进位标志清零

AX 0

如果失败，进位标志设置

AX 错误代码

01H 无效功能

05H 拒绝访问

注释：

建立参数块以用于通用 IOCTL 调用。如果设备支持 IOCTL 查询，就可确定驱动程序是否支持指定的通用 IOCTL 调用。一般，在调用一个通用 IOCTL 功能以前应询问驱动程序。

INT 21H 功能 44H 子功能 11H V5

查询 IOCTL 设备

确定用句柄指定的设备是否支持指定的 IOCTL 功能

调用寄存器： AH 44H

AL 11H

BL 驱动器号

CH 种类代码

CL 功能代码

DS：DX 指向参数块的指针

返回寄存器： 如果成功，进位标志清零

AX 0

如果失败，进位标志设置

AX 错误代码

01H 不支持该功能

05H 拒绝访问

0FH 无效驱动器

注释：

建立参数块以用于通用 IOCTL 调用。如果设备支持 IOCTL 查询，就可判定驱动程序是否支持指定的通用 IOCTL 调用。一般，在调用一个通用 IOCTL 功能以前应询问驱动程序。

DOS 5.0 支持的功能：

40H 置设备参数

41H 写逻辑驱动器上磁道

42H 格式化逻辑驱动器上磁道

46H 置介质 ID

60H 取设备参数

61H 读逻辑驱动器上磁道

62H 校验逻辑驱动器上磁道

66H 取介质 ID

68H 读介质类型

INT 21H 功能 45H V2
复制句柄

给已打开的设备或文件提供一个新句柄

调用寄存器： AH 45H
BX 文件句柄

返回寄存器： 如果成功，进位标志清零
AX 新文件句柄
如果失败，进位标志置位
AX 错误代码
04H 无可用句柄
06H 无效句柄

注释：

本功能为同一文件提供了另一个句柄。文件指针一起移动，如果移动一个文件的文件指针，另一个文件的指针也随之移动。

使用本功能大多是为了强行修改一个文件的目录项而省去打开和关闭文件的麻烦。在DOS V3.3以前，这是强行修改的唯一途径。DOS V3.3引入了新功能68H，这种事变得更加容易了。

INT 21H 功能 46H V2
强行复制句柄

使两个文件句柄指向相同文件的相同位置。

调用寄存器： AH 46H
BX 第一文件句柄
CX 第二文件句柄

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误代码
04H 无可用句柄
06H 无效句柄

注释：

本功能的结果类似于功能45H，它使两个文件句柄指向相同的文件并一起移动。本功能最重要的用途是提供设备重定向，可以在另外一个程序内部控制重定向过程，并使设备返回正常状态。步骤如下：

1. 用功能45H复制要重定向的句柄。为了后面的恢复，保存新文件句柄；
2. 用功能46H重定向。把“要重定向的句柄”放进CX，把“要重定向到的句柄”放在

BX 中。

当要返回正常情况时，再调用功能 46H。重定向过的句柄放在 CX 中，功能 45H 返回的复制句柄放在 BX 寄存器中。

在调用寄存器中，如果 CX 中的句柄指向一个打开的文件，那么在功能开始以前文件应关闭。

INT 21H	功能 47H 取当前目录	V2

返回一个当前目录完整路径的 ASCIIZ 字符串，不含驱动器名和作为开头的反斜线（\）字符。

调用寄存器： AH 47H
DL 驱动器号（0=缺省，1=A 等）
DS：SI 指向 65 字节临时缓冲区（Scratch Buffer）

返回寄存器： 如果成功，进位标志清零
DS：SI 不变，缓冲区含当前目录路径的 ASCIIZ 字符串
如果失败，进位标志置位
AX 错误代码
0FH 无效驱动器

注释：

本功能返回不含驱动器名和开头反斜线（\）的当前目录的路径名。因为调用该功能时设置了驱动器号，所以不含驱动器名和反斜线（\）是合适的（如果想用本功能返回的字符串建立一个文件名，得给文件名加上驱动器名和反斜线）。如果目录是根目录，那么返回的字符串是 NUL（第一字节为 0）。

在改变目录以前调用本功能有一优点：当程序结束时，用户能返回到初始目录。程序员必须小心，因为无效的驱动器号将导致该功能调用失败。要设置当前目录，可参见功能 3BH。

INT 21H	功能 48H 分配内存	V2

给程序分配一块所需内存，返回一个指向该块开始的指针

调用寄存器： AH 48H
BX 所需内存的节数

返回寄存器： 如果成功，进位标志清零
AX 已分配块的初始段地址
如果失败，进位标志设置
AX 错误代码
07H 内存控制块破坏
08H 没用足够内存

BX 如果功能调用失败，它为最大可用块的大小

注释：

该指针是块基址的段地址（基址是AX：0000H）。因为COM程序总是分配所有内存，所以，在一个COM程序内调用本功能时总是失败，除非进入程序后首先释放内存。

在多任务环境中，进程查到的内存顶部可能不是内存的实际顶部。如DESQView和Windows程序只给每个程序分配程序信息文件中允许的空间。

如果获取空间的尝试失败了，该功能返回最大可用内存块的大小；请求空间小于该块的调用才会成功。EXEC装入程序用这一方法给COM程序分配所有的可用空间。首先，它在BX中装入FFFFH，用来请求1048560个RAM字节（超过可用的）；因为必定导致错误代码8，所以分配失败。分配所有可用的空间是这样的：第一次调用在BX返回实际可用数量为多少，第二次调用则分配多少。

如果需要知道可用RAM的数量（没用被保留给任何程序），可在用户程序中使用相同的技术。如果使用功能48H，其中BX=FFFFH，那么BX中返回值是可用的块大小。

INT 21H	功能49H	V2
	释放已分配内存	

释放一块已分配内存到由DOS管理的共用区（使其他程序可用该内存）

调用寄存器： AH 49H
ES 待释放块的段地址

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误代码
07H 内存控制块破坏
09H 内存块地址无效

注释：

本功能假定释放的内存块是通过功能48H获取的。如果内存块不是由功能48H获取的，那么该功能可能简单地返回错误信息（如果幸运的话），或者它可能在释放内存的程序或驻留在内存的其他程序，从而产生不可预测的错误。这是因为系统要释放内存，所以在等待一个指向某内存块的地址，以将其纳入整个内存分配计划。这样，问题就产生了。

INT 21H	功能4AH	V2
	修改内存分配	

扩大或减少前面由功能48H分配的内存块大小

调用寄存器： AH 4AH
BX 新请求块的节数
ES 要修改块的段地址

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误代码
07H 内存控制块破坏
08H 没有足够内存
09H 内存块地址无效
BX 最大可用块的大小（如果 AX＝08H）

注释：

程序可用本功能修改用功能 48H 所获得的一个内存块，或者修改它们本身的内存分配。COM 程序运行时分配了所有内存，所以，如果 COM 程序希望执行其他程序，就必须调用本功能。EXE 程序也需要调用本功能来释放内存，除非 EXE 头部的 MAXALLOC 参数已被修改，其请求内存少于所有内存。

本功能经常被称为 SETBLOCK。

INT 21H	功能 4BH 执行程序（EXEC）	V2

在程序控制下执行另一程序。

调用寄存器： AH 4BH
AL 00 装入并执行程序
01 装入程序，但不执行 【未写入文档功能】
03 装入覆盖（Overlay）
05 进入 Exec State 【DOS V5 新增功能】
ES：BX 指向参数块的指针
DS：DX 指向程序说明的指针
DS：DX 指向 Exec State 结构的指针，其定义如下（仅在 AL＝05H）：
00H 字，保留，应为零
02H 字，类型标志
0000H COM 文件
0001H EXE 文件
0002H Overlay 文件
04H 双字，指向 ASCIIZ 程序名（可包含路径说明）
08H 字， 新程序的 PSP 段地址
0AH 双字，新程序的 CS：IP
0EH 双字，程序大小（含 PSP）

返回寄存器： 如果成功，进位标志清零
除了 CS 和 IP 以外，包括堆栈指针在内的所有其他寄存器都被破坏。在调用前，SS 和 SP 必须保存在可用 CS 寻址的地方，在调用后恢复。
如果失败，进位标志置位

AX 错误代码

01H 功能无效
02H 文件未找到
05H 拒绝访问
08H 没用足够内存
0AH 环境无效
0BH 格式无效

注释：

EXEC 功能用于执行程序和管理覆盖。新程序（子进程）结束后，原来的程序（父进程）重新得到控制。如果子进程用一个带返回代码的 DOS 终止功能，那么父进程可以从子进程收到一个出口代码。

本功能也可装入覆盖，覆盖可由程序段或数据组成。程序执行和覆盖操作之间的主要区别是：程序执行时，系统中无论释放什么内存都分配给程序，而覆盖只装入调用覆盖功能的程序已经占有的内存。如果需要，程序在运行另一个程序之前应释放内存（COM 程序的需要）。

该操作是由 ES：BX 寄存器指向的参数块控制的。参数块的格式在表 1.38 给出。

表 1.38 参数块格式

字节偏移量	域宽	内容
		——EXEC 功能（AL＝00H）——
00H	字	指向环境块的段指针
02H	字	命令行尾偏移地址
04H	字	命令行尾段地址
06H	字	第一个 FCB 偏移地址（偏移 5CH）
08H	字	第一个 FCB 段地址
0AH	字	第二个 FCB 偏移地址（偏移 6CH）
0CH	字	第二个 FCB 段地址
		——Debug 功能（AL＝01H）——
00H	字	指向环境块的段指针
02H	字	命令行尾偏移值
04H	字	命令行尾段地址
06H	字	第一个 FCB 偏移地址（偏移 5CH）
08H	字	第一个 FCB 段地址
0AH	字	第二个 FCB 偏移地址（偏移 6CH）
0CH	字	第二个 FCB 段地址
0EH	字	装入程序的 SP
10H	字	装入程序的 SS
12H	字	装入程序的 IP
14H	字	装入程序的 CS
		——覆盖功能（AL＝03H）——
00H	字	指向覆盖装入点的段指针
02H	字	用于代码图象的再定位因子（仅对 EXE 文件）

环境块是一系列 ASCIIZ 字符串，用来把环境信息传递给被执行的程序。这些字符串在命令级通过 SET 功能设置，或者在程序内部创建。通常这些字符串包括 COMSPEC 变量（从中查找系统命令处理程序 COMMAND.COM），PATH 变量（从中寻找可执行程序），以及其他系统指定的变量。

一个典型的环境块类似于如下形式：

COMSPEC=C:\COMMAND.COM*PATH=C:\DOS**

星号代表 NUL 或 0 字节。如果环境块指针是 0，那么子进程将继承父进程的环境。在 DOS V3 及以后的版本中，环境块中最后的 0 后面跟着一个带有字符计数的字，字符计数后面跟着带有被执行程序文件的驱动器和路径名的 ASCIIZ 字符串。

命令行尾是一个字符串，由命令行中要执行的命令之后的字符组成。它前面为一个单字节的长度计数值，其后为字符串并以一个回车结束。总长度不超过 128 字节；它将被复制到程序段前缀（PSP）中偏移量 80H 处，在它进入程序前，仅有 128 字节的命令行尾。

典型的命令行尾形式为：

#/c CHAPT01.DOC@

#为一个单字节的数字，其值为 14（表示不含回车字符，命令行尾共 14 个字符），@代表单字节的回车字符。

这样装入的子进程将继承父进程 I/O 文件，除非是父进程在打开文件的调用中（功能 3DH）明确指明为不继承。标准文件保持打开。如果标准文件在父进程中已重定向，则子进程仍然保持该重定向。父进程可以重定向文件（见功能 46H）。

调用该功能需注意一些事项。因为该功能用于执行另一程序，在作任何 EXEC 调用时，必须假定所有的寄存器在该功能调用期间要改变。当从 EXEC 功能调用返回时，仅有 CS 和 IP 可认为是正确的。在调用前，父进程必须至少把 SS 和 SP（加上需保存的其他寄存器）存于某处，该处可用 CS 段寄存器寻址。在返回时，SS 和 SP 可恢复为它们原先的值，但在恢复过程中应禁止中断，以便恢复过程不被中途打断；否则，系统就会进入不稳定的状态。

如果没有足够的内存来装入要执行的程序，EXEC 功能就不会成功。在调用 EXEC 之前，汇编语言程序应调用功能 4AH 来释放所需的内存。当 C 语言程序开始时，不需要的内存已被释放出来。

在 DOS V2 中，该功能常常出错。其中很多是由于编程者试图用压入堆栈的方法来保存寄存器的值，而在一些版本中未能确保 CPU 的方向标志正确设置，结果，系统断续地被挂了起来。为防止这个问题，在调用 EXEC 之前，简单地使用 CLD 指令作为设置的一部分。若程序只在 V3 及以上版本中使用，这就不必要了，因为 DOS 代码中加入了这种检查。

有趣的是，在 IBM 的 DOS 2.x 中，该功能实际上不在 IBMDOS.COM 文件中，而作为 COMMAND.COM 中的一部分。IBM 这样做是为了节省空间；其他的 V2.x 系统把它作为 MSDOS.SYS 的一部分，IBM 从 3.0 开始就是这样做的（此时，两个 DOS 文件不能被分割成碎片的限制没有了）。

调试程序用子功能 01H 把一个程序加载到内存中，使其准备好执行。该功能用于 Microsoft 的 DEBUG、SYMDEB 和 CODEVIEW 调试程序以及 Borland 的 Turbo Debugger。用于调试程序的特殊之处是被加载程序的 SS、SP、CS 和 IP 寄存器值返回时存贮于 ES：[BX+0EH] 处。和写入文档的子功能 00H 和 03H 一样，子功能 01H 在返回时破坏了全部寄存器

的值；程序必须在某处保存 ES 和 BX 寄存器的值，在程序使用 SS：SP 和 CS：IP 值之前，可相对于 CS 从该处取回它们的值。

子功能 05H 是 V5 中新加入的，支持程序截取 MS-DOS EXEC 调用而由它自己完成加载。它完成一些内部处理，例如设置被加载程序可见的版本号（它由 SETVER 设置），以及完成加载程序的存入时间补片。

子功能 05H 应是把控制传给被加载程序前的最后一个调用。特别是，在该功能成功返回和把控制权传给被加载程序之间，不应该进行 MS-DOS，BIOS 调用或者使用任何中断。

如果 DOS 是在 HMA 中运行，从子功能 05H 调用返回时，A20 就被关闭。

INT 21H	功能 4CH 带返回码终止	V2

退出程序，回到其父进程

调用寄存器： AH 4CH

AL 返回码

返回寄存器： 无

注释：

退出时，DOS 要做以下事情：

- 从 PSP：000AH 处恢复终止处理程序向量；
- 从 PSP：000EH 处恢复 Ctrl-Break 处理程序向量；
- 从 PSP：0012H 处恢复严重错误处理程序向量（DOS 3.0 及以上版本）；
- 刷新文件缓冲区（句柄文件）；
- 跳转到终止处理程序地址。

推荐使用本功能来终止程序。在 DOS 2.0 以后开发的程序，通常都用这种功能而不是 INT 20H 或 INT 21H 的 00H 功能。这个功能与以前的终止程序功能相比，有两个优点：

- 它返回一个出口代码，该代码在批处理文件中可被用作 ERRORLEVEL 参数，或父程序中通过功能 4DH 来确定返回信息；
- 它不依赖于寄存器的设定而正确操作，例如，CS 寄存器不必指向 PSP 所在的段。

该功能调用自动关闭文件句柄和更新磁盘目录，所以它避免了文件处理中无意的错误。但它不会对文件控制块（FCB）文件做任何操作。

INT 21H	功能 4DH 取返回码	V2

从一个成功的 EXEC 功能调用取返回码

调用寄存器： AH 4DH

返回寄存器： AH 系统出口代码

00 正常终止

01 用 Ctrl-C 终止
02 严重设备错误引起终止
03 用功能调用 31H 终止

AL 子出口代码

注释：

调用时，该功能调用通过系统返回一次（且仅一次）子进程出口代码。该功能把出口代码复位为 0；如果想把它传给一个父进程，该码必须保持且作为用户程序的出口代码而返回。系统出口代码能标明程序是否正常终止。子出口代码可以任意设置，对它的解释依赖于所运行的程序。

INT 21H 功能 4EH V2
搜索第一个匹配的文件

定位第一个匹配的文件名，文件名由一个可包含通配符的 ASCIIZ 字符串指定

调用寄存器： AH 4EH
CX 搜索时使用的属性
DS：DX 指向 ASCIIZ 文件说明的指针

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误代码
02H 文件未找到
03H 路径无效
12H 没有更多文件

注释：

当给出一个包含了文件全名的 ASCIIZ 字符串（也可能含通配符 * 或?）后，该功能会在磁盘传送区（DTA）填上返回文件的信息。搜索受限于提供给该功能的属性，仅仅找到满足文件属性的文件。文件的属性可以包含下列值（见表 1. 39）。

表 1.39 文件属性

值	匹配的文件类型
00H	普通
02H	普通和隐含
04H	普通和系统
06H	普通、隐含和系统
08H	卷标
10H	目录

由于文件属性仅为一个字节，可以把 CL 设为合适的属性而把 CH 置为 0。此功能返回时，DTA 设置如下（见表 1. 40）。

表 1.40 DTA 格式

偏移	域宽	内容
00H	21字节	为DOS后序搜索而保留
15H	字节	匹配的文件属性
16H	字	文件的时间
18H	字	文件的日期
1AH	双字	文件的长度
1EH	13字节	ASCIIZ字符串表示的文件名及扩充名，已删除空白字符，扩展名前已放入句点（.）

时间和日期项表述如下（见表 1.41）。

表 1.41 时间和日期格式

位 FEDCBA98	76543210	含义
		时间项编码
xxxxx...	.....	小时(0—23)
.....xxx	xxx.....	分钟(0—59)
.....	...xxxxx	双秒的增量(0—29)
		日期项编码
xxxxxxx.	.....	年份——1980
.....x	xxx.....	月份(1—12)
.....	...xxxxx	天(1—31)

可以读取DTA以得到所定位文件的信息，若搜索的字符串含有通配符，所得文件就会是一个与搜索的字符串不同的文件名。在进一步搜索时，DTA应不受破坏。

INT 21H 功能 4FH V2

搜索下一个匹配的文件

成功地调用功能4EH之后，这个功能调用继续搜索满足要求的文件。DTA必须保留着4EH功能调用放在那儿的原始信息

调用寄存器： AH 4FH

返回寄存器： 如果成功，进位标志清零

如果失败，进位标志置位

AX 错误代码

12H 文件找完

注释：

如果在第一次搜索（功能调用4EH）时使用了通配符，重复使用这个功能调用可以接着

找到满足通配符说明的文件。搜索失败（返回时进位标志置位），表明不再有满足要求的文件名。

本功能的调用过程与第一次搜索相同。这个功能更新 DTA 以表示找到文件名及其他数据。在连续搜索时，DTA 一定不能修改。表 1.42 显示出该功能调用在 DTA 中返回的数据。

表 1.42 DTA 格式的返回值

偏移	域宽	内容
00H	21 字节	为 DOS 后序搜索而保留
15H	字节	匹配的文件属性
16H	字	文件的时间
18H	字	文件的日期
1AH	双字	文件的长度
1EH	13 字节	ASCIIZ 字符串表示的文件名及扩展名，已删除空白，扩展名前已放入句点（.）

时间和日期项表述如表 1.41。

可以读取 DTA 以取回所定位文件的信息。若搜索的字符串含有通配符，所得文件就会是一个与搜索的字符串不同的文件名。在进一步搜索时，DTA 应不受破坏。

INT 21H	功能 50H 置 PSP 段地址	V2

设置当前运行程序的程序段前缀（PSP）的地址

调用寄存器： AH 50H

BH 新 PSP 的段地址

返回寄存器： 无

注释：

TSR 程序在 TSR 进程与 DOS 系统中被中断的进程间的现场切换（Context Switching），可用本功能实现。现场切换使 DOS 以为 TSR 进程比被中断的进程优先级更高。为此，需要记录原先的 PSP 地址（功能 51H），并通知 DOS：TSR 的 PSP 为当前的 PSP（功能调用 50H）。当 TSR 返回被中断的程序时，必须恢复原来的 PSP。

据说该功能在 DOS V3 以前版本中不可靠，尤其是在 INT 28H 处理程序（键盘忙循环）中它不能正确设置 PSP。

这种错误的原因很简单：DOS V2 在把控制权交给任何功能调用前，先切换到自己内部堆栈；在 DOS 已准备好处理另一个输入功能时调用了这一功能，第二次堆栈切换使得输入功能所需的信息遭到破坏。

解决办法相当简单而又比较巧妙：通过改变严重错误标志欺骗 DOS，使它以为正在处理一个严重错误；在调用这个功能前设置严重错误标志，迫使 DOS 用它的备用内部堆栈。从该功能调用返回后，必须恢复原来的 CritErr 标志。关于严重错误标志的详细情况，可参见 INT

21H 功能 34H 的注释。

在 V3 中，DOS 的调度代码已被修改，以在做任何堆栈切换之前检测这个功能调用；如果要用的正是这个功能，DOS 就干脆使用调用程序的堆栈，彻底避免了这个问题。

PSP 段地址有时被称作运行进程标识（PID），这个功能常常被称为 SetPID。

INT 21H	功能 51H 取 PSP 段地址	V2

取得正在执行进程的程序段前缀的地址

调用寄存器： AH 51H

返回寄存器： BX 当前运行进程的 PSP

注释：

这个功能用于获取被 TSR 程序中断的进程的 PSP。TSR 可以保留这个地址，并通知 DOS 它自己的 PSP 是当前运行进程的 PSP。

PSP 段地址有时被称作运行进程的标识(PID)，这个功能常常被称为 GetPID。在 DOS V3 中，作为补充，增加了写入文档的功能 62H 来完成同样的任务；在所有的版本中，两个功能执行完全相同的代码。

这个功能调用详细说明请参见功能 50H。

INT 21H	功能 52H 取磁盘参数表	V2

DOS 驱动器参数块保存着有关磁盘配置的数据，这个功能可访问这样的参数块表和很多 DOS 内部表

调用寄存器： AH 52H

返回寄存器： ES：BX 指向下述的 DOS 表的指针

注释：

由于内部的需要，DOS 在以链表形式做成的驱动器参数块（DPB）中，保存着进行磁盘操作需要的关键参数。该功能返回的指针指向链表的开头。但是，DPB 表仅仅是 DOS 保存的一系列内部数据表中的一个，如表 1.43 中所示，该功能调用返回的指针也可以用来定位其他许多项。

注意，表 1.43 的结构随 DOS 的版本的不同而变化。在三个版本中只有 DPB 指针－2H 到＋0FH 一段保持不变。在 V2 之中，在 10H 处的一个字节给出了逻辑驱动器数，其后，一个字给出了以字节数表示的最大块的大小，一个双字指向第一个缓冲区控制块。

在 DOS V3 中，在表的前头又加了 6 个字节：在－08H 处一个双字指向当前的缓冲区控制块；在－04H 处，一个字表示当前缓冲区的偏移值（通常为 0000）。V2 中 10H 处的字节在 V3 中被移到 21H 处，移来了其后的 6 字节来填补这一空间。一个指向逻辑驱动器表的双字指针被加到 16H 处，接着在 1AH 处加入一个双字指针指向系统 FCB 链，在 1EH 处一个字表明

在网络交换时应保持 FCB 数目。20H 处的字节表明当前的块设备数目，在 21H（前面提到过）处的字节表示了当前的逻辑驱动器数目。

从 V3 到 V4，如表 1.43 所示，发生的变化仅有 12H 处的 BCB 指针，它变为文件缓冲区的新的 EMS 链接块指针，以及－06H 处当前 BCB 指针可直接访问的缓冲区链。

表 1.43 中有些“空洞”存在于许多数据区域之中。这些未知的字节不是无用，而是由于它们的作用还不知道。已知部分是那些致力于详解 DOS 的人偶然挖掘出的。

表 1.43 配置变量表（CVT）

偏移	域宽	含义
		——DOS V3.x 及以上版本——
－08H	双字	在 BUFFERS=链中的当前的缓冲区
－04H	字	当前缓冲区中的偏移值
		——DOS V2.x 及以上版本——
－02H	字	第一个内存控制块的段地址
00H	双字	指向第一个驱动器参数块的指针
04H	双字	指向第一个 DCB 的指针（系统文件表设备控制块）
08H	双字	指向 CLOCK $ 设备驱动程序的指针
0CH	双字	指向 CON 设备驱动程序的指针
		——仅 DOS V2.x——
10H	字节	逻辑驱动器数
11H	字	任意块设备中每扇区最大字节数
13H	双字	指向磁盘缓冲链的开头的指针
17H		NUL 设备驱动程序的头；设备驱动程序链中的第一个
		——DOS V3.x 及以上版本——
10H	字	任意块设备中每扇区的最大字节数
12H	双字	指向磁盘缓冲链开头的指针（在 V4 中，指向 EMS 链记录，该记录依次指向缓冲区链）
16H	双字	指向逻辑驱动器表的指针（参看下面的讨论）
1AH	双字	指向 DOS 的 FCB 链开头的指针
1EH	字	在交换中保持的 FCB 数目
20H	字节	块设备数目
21H	字节	逻辑驱动器数目，由 CONFIG.SYS 中的 LASTDRIVE 的值设定（不做说明时，缺省值是 5）
22H		NUL 设备驱动程序的头；设备驱动程序链的第一个设备

表 1.44 是表 1.43（DOS 3.0 以上）偏移量 16H 所指的逻辑驱动器表的内容。指针指向表的开头。系统中每个驱动器有一个表，以驱动器 A 开始。最小的数目是 5（从 A 到 E），或由 CONFIG.SYS 文件中的 LASTDRIVE 的值来确定。在 V3 中，每个表长 81 字节（在 V4 中为 88 字节），这些表在内存中一个接一个地放置。

表 1.44 逻辑驱动器表

偏移	域宽	含义
00H	2 字节	实际驱动器标识符和：
02H	65 字节	以 ASCIIZ 字符串表示的该驱动器的当前路径（包括根目录的斜线及结束符 00H 字节）
43H	字	当前驱动器的状态（位映象） 8000H＝未知 4000H＝准备就绪 2000H＝未知 1000H＝替换单元（SUBSTed Unit） 0000H＝逻辑驱动器没有映象到物理驱动器
45H	双字	指向驱动器的 DOS DPB 的指针
49H	字	当前目录的首簇（对未访问过的驱动器为 FFFFH，0 为根目录）
4BH	字	未知
4DH	字	未知
4FH	字	报告目录时应跳过的字节数，对正常驱动器为 0002，替换单元时隔得更多 ——DOS V4——
51H	7 字节	未知，缺省为 0

INT 21H 功能 53H V2

把 BPB 转换为 DPB

将 BIOS 参数块（BPB）转换为 DOS 的驱动器参数块（DPB）

调用寄存器： AH 53H

DS：SI 指向 BPB 的指针

ES：BP 指向 DPB 区域的指针

返回寄存器： 无

注释：

BIOS 和 DOS 都保存着连到系统上的磁盘和驱动器的信息。这个功能调用可以把 BIOS 的参数块（BPB）转换成 DOS 的驱动器参数块（DPB）。又见 INT 21H 的功能 1FH 和 32H，以及 INT 21H 功能 44H 子功能 0DH，辅助功能 40H。表 1.45 展示了 BPB 和 DPB 的格式。可以看出，列出的主要是磁盘信息（尤其是在 BPB 之中）。

表 1.45 BIOS 和 DOS 的参数块

偏移量	域宽	含义
		——BIOS 参数块的信息——
00H	字	每扇区的字节数
02H	字节	每簇的扇区数
03H	字	从扇区 0 开始的保留扇区数
05H	字节	FAT 的数量
06H	字	根目录最大项数

续表

偏移量	域宽	含义
08H	字	总扇区数（或为 0，在 V4 中表明应用扩充的 BPB 格式）
0AH	字节	介质描述
0BH	字	每个 FAT 扇区数
0DH	字	每磁道的扇区数
0FH	字	磁头数
11H	双字	隐含的扇区数
15H	双字	在 BPB 扩充格式中总的扇区数，仅当偏移量 08H 处为 0 时有用。仅 V4 中使用
19H	7 字节	保留
		——— DOS 参数块信息 ———
		【所有版本】
00H	字节	驱动器号（0=A，1=B 等）
01H	字节	设备驱动单元号
02H	字	每扇区的字节数
04H	字节	每簇的扇区数（以 0 为基数）
05H	字节	移位因子（Shift Factor）
06H	字	保留的引导扇区数
08H	字节	FAT 的份数
09H	字	根目录的项数
0BH	字	第一个数据扇区号
0DH	字	最高的簇数加 1
		【仅 V2，V3】
0FH	字节	每个 FAT 的扇区数（0—255）
10H	字	根目录的起始扇区号
12H	双字	驱动器的设备驱动程序地址
16H	字节	介质描述字节
17H	字节	磁盘参数块访问标志（0FFH 表示必须重建）
18H	双字	下一个设备参数块地址
		【仅 V2】
1CH	字	当前目录的开始簇号
1EH	64 字节	当前目录路径的 ASCIIZ 字符串
		【仅 V3】
1CH	字	上一次从本驱动器分配出的簇号
1EH	字	未知；一般为 FFFFH
		【仅 V4】
0FH	字	每 FAT 扇区数（0—65535）
11H	字	根目录开始的扇区号
13H	双字	驱动器的设备驱动程序地址
17H	字节	介质描述字节
18H	字节	磁盘参数块存取标志（0FFH 表示必须重建 DPB）
19H	双字	下一个设备参数块地址
1DH	字	上一次从本驱动器分配出的簇号
1FH	字	未知，一般为 FFFFH

INT 21H	功能 54H	V2
	取校验标志	

获取写后读（校验）标志的当前值

调用寄存器： AH 54H

返回寄存器： AL 00H 校验关闭

01H 校验打开

注释：

系统在写之后是否进行读盘校验是由校验标志来控制的。缺省的标志为OFF（值00）。

功能2EH用于置校验标志。设置的结果是允许校验，磁盘操作速度会降低一些，但确保了磁盘操作的成功。

由于网络系统不支持校验功能，在这种情况下，返回的代码没有意义。

INT 21H	功能 55H	V2
	创建 PSP	

在设定的段地址上创建程序段前缀（PSP）

调用寄存器： AH 55H

DX 建立PSP的段地址

返回寄存器： 无

注释：

这个功能与INT 21H功能26H相似。差别在于这个功能不是简单地复制PSP；它建立一个独立的不同的“子”PSP，以便运行另一个程序。与INT 21H功能26H一样，该功能的作用已被EXEC功能取代（INT 21H功能4BH）。

INT 21H	功能 56H	V2
	文件改名	

改文件名或将其移动到同一磁盘驱动器上的另外子目录下

调用寄存器： AH 56H

DS：DX 指向ASCIIZ当前文件名的指针

ES：DI 指向ASCIIZ新文件名的指针

返回寄存器： 如果成功，进位标志清零

如果失败，进位标志置位

AX 错误代码

02H 文件未找到

03H 路径未找到

05H 拒绝访问

11H 非同一设备

注释：

与应用文件控制块（FCB）的功能相比，两功能各有千秋。该功能允许用目录路径来指定文件，甚至可以在不同目录间移动文件；但由于它不允许用通配符，所以不可以成组地改动多个文件名。

对大多数一般的任务，这种限制不明显。对于通常应用来说，改文件名和在目录间移动文件是相当重要的；多个文件的情况可以用程序来处理。

如果路径不存在，或目标目录下新文件名已存在，这个功能就无法完成。这个操作也无法在不同磁盘设备间进行。如果被改名的文件是打开的，首先应关闭它。重新命名一个打开的文件将导致无法预料的错误。

在网络环境下，必须具备创建访问权才可以更改文件名。

INT 21H	功能 57H 子功能 00H	V2
	取文件的日期和时间	

读取目录项中文件最后修定的时间和日期

调用寄存器： AH 57H
AL 00H 读取日期和时间
BX 文件句柄

返回寄存器： 如果成功，进位标志清零
CX 获取的时间
DX 获取的日期
如果失败，进位标志置位
AX 错误代码
01H 功能无效（文件共享）
06H 句柄无效

注释：

此功能对于用功能 3CH，3DH，5AH 或 5BH（句柄打开或创建功能）打开的文件有效。表 1.16 显示了位的格式和如何解释日期和时间。

表 1.46 日期和时间格式

位		
FEDCBA98	76543210	含义
		——时间项编码——
xxxxx...	.....	小时(0—23)
.....xxx	xxx.....	分钟(0—59)
.....	...xxxxx	双秒的增量(0—29)
		——日期项编码——
xxxxxxx.	.....	年份——1980
.....x	xxx.....	月份(1—12)

INT 21H 功能 57H 子功能 01H V2
置文件的日期和时间

设置目录项中文件最后修定的时间和日期

调用寄存器： AH 57H
AL 01H 设置日期和时间
BX 文件句柄
CX 设置的时间
DX 设置的日期

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误代码
01H 功能无效（文件共享）
06H 句柄无效

注释：

本功能对于用功能 3CH，3DH，5AH 或 5BH（句柄打开或创建功能）打开的文件有效。表 1.47 展示了位格式和如何解释日期和时间。

表 1.47 日期和时间格式

位 FEDCBA98	76543210	含义
		——时间项编码——
xxxxx...	.....	小时(0—23)
.....xxx	xxx.....	分钟(0—59)
.....	...xxxxx	双秒的增量(0—29)
		——日期项编码——
xxxxxxx.	.....	年份——1980
.....x	xxx.....	月份(1—12)
.....	...xxxxx	天(1—31)

INT 21H 功能 58H 子功能 00H V2
取内存分配策略

读取决定内存分配策略的代码

调用寄存器： AH 58H
AL 00H 读取策略代码

返回寄存器： 如果成功，进位标志清零
AX 策略代码
00H 第一个适合块，低 640K 优先（缺省）
01H 最佳适合块，低 640K 优先

02H 最后一个适合块，低 640K 优先
40H 第一个适合块，仅 UMBs
41H 最佳适合块，仅 UMBs
42H 最后一个适合块，仅 UMBs
80H 第一个适合块，UMBs 优先
81H 最佳适合块，UMBs 优先
82H 最后一个适合块，UMBs 优先

如果失败，进位标志置位

AX 错误代码
01H 功能无效（文件共享）

注释：

在 V2 中，Microsoft 和 IBM 宣布该功能为“DOS 内部使用”。如果是从第三方零售商拷贝来的 V2，这个功能或许用于其他的目的。在 V3 中，在 Microsoft 的文件中，该功能成为一个已写入文档的功能，而在 IBM 的版本之中，它还未写入文档。

这个功能允许程序了解 DOS 处理内存分配的策略。通常，这类参数是没有意义的，除非有理由相信一种策略比另一种策略更好。

DOS 所知的可能策略包括第一个适合、最佳适合和最后一个适合。在 V5 中，可以修改这些策略以说明是 UMBs 优先还是低 640K 优先。如果选择第一个适合块的策略，那么它从低内存到高内存搜索与所要求的同样大或更大的第一个内存块，返回找到的第一个内存块。

最佳分配策略将检查所有的内存块，找到满足内存要求的最小内存块。尽管这种策略会最有效地利用内存，但需花费更多的时间。

最后适合块的分配策略与第一个适合块策略相似，只是它是从高内存到低内存搜索，而不是从低向高搜索，返回满足要求的最后一个内存块。

在 V5 之中，如果所检查的第一个区域的结果是可行的（不考虑在低 640K 或是在 UMBs），就不会检查第二个区域。因此，最佳适合策略也不会是最好的结果。只要在优先的区域中存在一块可分配的内存块，就不会检查第二个区域（即使第二个区域中会有更合适的内存块）。

在 V5 之前的版本中，最后适合策略（代码 02）可以是大于或等于 02 的数值，因此存贮的可能是一个 02 以外的数。做策略检查时，应该考虑到比 02 大的数。

INT 21H	功能 58H 子功能 01H	V2
	置内存分配策略	

设置所用于内存分配策略的代码

调用寄存器： AH 58H
AL 01H 读取策略代码
BX 策略代码
00H 第一个适合块，低 640K 优先（缺省）
01H 最佳适合块，低 640K 优先

02H 最后一个适合块，低 640K 优先
40H 第一个适合块，仅 UMBs
41H 最佳适合块，仅 UMBs
42H 最后一个适合块，仅 UMBs
80H 第一个适合块，UMBs 优先
81H 最佳适合块，UMBs 优先
82H 最后一个适合块，UMBs 优先

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误代码
01H 功能无效（文件共享）

注释：

在 V2 中，Microsoft 和 IBM 宣布该功能为“DOS 内部使用”。如果是从第三方零售商拷贝来的 V2，这个功能或许用于其他的目的。在 V3 中，在 Microsoft 的文件中，该功能成为一个已写入文档的功能，而在 IBM 的版本之中，它还未写入文档。

当别的程序向 DOS 请求内存时，可用这个功能选择内存分配策略。对于大多数程序来说，这类调整的参数没有什么意义，除非有理由表明一种策略胜过另一种策略。

DOS 所知的可能策略包括第一个适合、最佳适合和最后一个适合。在 V5 中，可以修改这些策略以说明是 UMBs 优先还是低 640K 优先。如果选择第一个适合块的策略，那么它从低内存到高内存搜索与所要求的同样大或更大的第一个内存块，返回找到的第一个内存块。

最佳分配策略将检查所有的内存块，找到满足内存要求的最小内存块。尽管这种策略会最有效地利用内存，但需花费更多的时间。

最后适合块的分配策略与第一个适合块策略相似，只是它是从高内存到低内存搜索，而不是从低向高搜索，返回满足要求的最后一个内存块。

在 V5 之中，如果所检查的第一个区域的结果是可行的（不考虑在低 640K 或是在 UMBs），就不会检查第二个区域。因此，最佳适合策略也不会是最好的结果。只要在优先的区域中存在一块可分配的内存块，就不会检查第二个区域（即使第二个区域中会有更合适的内存块）。

在 V5 之前的版本中，最后适合策略（代码 02）可以是大于或等于 02 的数值，因此存贮的可能是一个 02 以外的数。做策略检查时，应该考虑到比 02 大的数。

INT 21H	功能 58H 子功能 02H 取 UMB 链入状态	V5

读取 UMB 链入状态

调用寄存器： AH 58H
AL 02H

返回寄存器： 如果成功，进位标志清零
AL 结果

00H 非链入状态
01H 链入状态
如果失败，进位标志置位
AX 错误代码
07H 废弃区域（Arena Trashed）

注释：

这个子功能让程序了解 DOS 是否可在它的内存分配表中使用 UMBs。

INT 21H	功能 58H 子功能 03H	V5
	置 UMB 链入状态	

设置 UMB 链入状态

调用寄存器： AH 58H
AL 03H
BX 链状态
0000H 不链入
0001H 链入

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误代码
01H 无效功能，加载 MS-DOS 时无 DOS=UMB
07H 废弃区域（MCB 被破坏）

注释：

该子功能让程序选择 DOS 是否在它的内存分配表使用 UMBs。

MS-DOS 利用 MCB 的链表来作内存分配管理，MCB 通常被称为 Memory Arena（内存竞技场）。当 MS-DOS 启动时，若发现 CONFIG.SYS 有 DOS=UMB 命令，且有 UMB 存在，就将其纳入分配管理。注意：Upper Memory 指物理地址 0A0000H—0FFFFFH 之间的区域，UMB 则是指 Upper Memory 内可供自由分配的 RAM。MS-DOS 启动时，Upper Memory 处于未链入的状态。若程序更改了设定，结束后，MS-DOS 不会将其自动还原为未链入的状态。另外，功能 49H、4AH 不受此功能影响。

INT 21H	功能 59H	V3
	取扩充错误信息	

返回关于 INT 21H 功能调用的错误信息，包括建议的解决行动。这个调用将破坏寄存器 AX，BX，CX，DX，SI，DI，BP，DS 和 ES 的内容。

调用寄存器： AH 59H
BX 00

返回寄存器： AX 扩充错误代码
BH 错误类型
BL 建议的行动
CH 错误设备代码（Error Locus）

注释：

本功能使DOS在错误处理方面获得很大进展，使DOS成为诊断和解决运行时错误的帮手。这个功能增加了很多分析和定位DOS调用错误的能力。当调用INT 21H或INT 24H返回错误时，可以调用该功能；若没有错误，该功能将返回AX＝0000H。调用文件控制块（FCB）返回FFH时，也可用该功能来解决问题。

表中对于返回的信息进行了分类，寄存器AX内返回的是扩充错误码（参见表1.48），这些都是通常的INT 21H功能调用返回的系统错误；寄存器BH包括了错误的类型，它提供了进一步的错误信息（参见表1.49）。

寄存器BL返回的结果最有用，它是解决错误的建议行动（参见表1.50）。最后，寄存器CH返回错误轨迹，它用来识别物理错误的地址（见表1.51）。公开这些信息的意义到底有多大？因为一般的错误处理不成问题，其步骤可通过所发生错误的类型来确定。

如果一个功能返回时进位标志置位，表明发生了错误。错误处理程序可以如下编写：

1. 设置这个功能的寄存器内容；
2. 组织对INT 21H的调用；
3. 如果进位标志清零，继续正常运行；
4. 如果进位标志置位，不考虑返回的错误代码，组织对于功能59H的调用；
5. 用BL寄存器中建议的行动来确定适当的措施。

有些功能以在AL寄存器中返回代码（AL＝FFH）来表明出错。这些情况下，可以如下来调用：

1. 设置功能调用的寄存器内容；
2. 组织对INT 21H的调用；
3. 如果AL中无错误报告，继续进行正常的运行；
4. 如果AL报告错误，不考虑所报告的错误，组织对于功能59H的调用；
5. 应用BL寄存器中建议的行动来确定适当的措施。

应当小心地应用这一功能调用，返回时，寄存器AX，BX，CX，DX，SI，DI，BP，DS和ES的内容遭破坏。出错后应立即调用此功能；如果调用前执行了其他的DOS功能调用，返回的状态不是对应的错误。

表1.48 AX中返回的扩充错误代码

代码 十进制	十六进制	含义
1	01	功能号无效
2	02	文件未找到
3	03	路径未找到
4	04	没有可用的句柄

续表

代码 十进制	十六进制	含义
5	05	拒绝访问
6	06	句柄无效
7	07	内存控制块遭破坏
8	08	内存不够
9	09	内存块地址无效
10	0A	环境无效
11	0B	格式无效
12	0C	访问代码无效
13	0D	数据无效
14	0E	保留
15	0F	驱动器无效
16	10	试图删除当前目录
17	11	设备不相同
18	12	无更多文件
19	13	磁盘写保护
20	14	未知单元
21	15	驱动器未准备好
22	16	未知的命令
23	17	CRC 错误
24	18	要求的结构长度错误
25	19	搜索错误
26	1A	未知介质类型
27	1B	扇区未找到
28	1C	缺纸
29	1D	写错误
30	1E	读错误
31	1F	一般错误
32	20	共享混乱（Violation）
33	21	加锁混乱
34	22	改变磁盘无效
35	23	FCB 不可用
36	24	共享缓冲区溢出
37	25	代码页未匹配
38	26	未能完成文件操作【仅 V4】
39	27	磁盘满
40—49	28—31	保留
50	32	不支持网络请求
51	33	远程计算机不收听
52	34	网络上名字重复
53	35	网络名未找到
54	36	网络忙
55	37	网络设备不存在
56	38	超出 NetBIOS 命令限制
57	39	网络适配器错误
58	3A	网络响应不正确
59	3B	未预料的网络错误

续表

代码 十进制	十六进制	含义
60	3C	不兼容的远程适配器
61	3D	打印队列满
62	3E	打印文件空间不足
63	3F	打印文件被删除
64	40	网络名被删除
65	41	网络拒绝访问
66	42	网络设备类型不正确
67	43	网络名未找到
68	44	超出网络名的限制
69	45	超出NetBIOS会话限制
70	46	共享暂停
71	47	网络请求未被接受
72	48	重定向打印机或磁盘暂停
73—79	49—4F	保留
80	50	文件已存在
81	51	FCB重复
82	52	不能建立目录项
83	53	INT 24H失败
84	54	重定向太多
85	55	重复重定向
86	56	口令无效
87	57	参数无效
88	58	网络数据出错
89	59	功能不被网络支持【仅V4】
90	5A	要求的系统组件未装【仅V4】

表1.49 错误类型代码（BH）

类型代码 十进制	十六进制	含义
1	01	资源用光
2	02	暂时状态（不是最终定位的错误）
3	03	权限
4	04	系统软件内部错误
5	05	硬件错误
6	06	系统失败
7	07	应用程序出错
8	08	未找到
9	09	格式不对
10	0A	已加锁
11	0B	介质
12	0C	已存在
13	0D	未知原因的错误

表 1.50 建议行动代码（BH）

行动代码	含义
1	重试，不清除的话，提示用户放弃或忽略
2	等待再重试。不清除的话，提示用户放弃或忽略
3	从用户获得正确的数据（磁盘驱动器或文件名错误）
4	放弃并清除应用程序
5	放弃但不清除（清除增加了出错的可能）
6	忽略错误
7	提示用户改正错误，再重试

表 1.51 错误设备代码（CH）

设备代码	含义
1	未知
2	块设备（磁盘或磁盘仿真器）
3	网络
4	串行设备
5	相关内存

INT 21H	功能 5AH	V3
	创建唯一名字的文件	

在指定的目录中产生一个与其他文件同名的文件

调用寄存器： AH 5AH

CX 属性

DS：DX 指向以反斜线（\）结束的 ASCIIZ 路径说明的指针

返回寄存器： 如果成功，进位标志清零

AX 句柄

DS：DX 指向附加了文件名的 ASCIIZ 文件说明的指针

如果失败，进位标志置位

AX 错误代码

03H 路径未找到

04H 无句柄可用

05H 拒绝访问

注释：

唯一名文件常用于临时文件。“实际文件名是什么?”留给操作系统去完成，程序不必考虑。

为应用这一功能，需要提供要产生临时文件的子目录全路径名，全路径名以反斜线结束。还可以说明创建文件的属性。表 1.52 列出了该功能可设置的属性。

表 1.52 文件类型

值	匹配的文件类型
00H	普通
02H	隐含
04H	系统
06H	隐含、系统

仅有的出错可能性是路径不存在，已使用完所有可用句柄，在根目录下产生文件名而根目录已满。要注意：临时创建的文件与其他文件一样的存在，这类文件不会被自动删除，程序结束前应删除该功能创建的全部文件。

在网络环境中，必须有创建访问权才可应用该功能。

INT 21H	功能 5BH 创建一个文件	V3

在指定的目录中创建一个新的文件

调用寄存器： AH 5BH
CX 属性
DS：DX 指向 ASCIIZ 文件说明的指针

返回寄存器： 如果成功，进位标志清零
AX 句柄
如果失败，进位标志置位
AX 错误代码
03H 路径未找到
04H 无句柄可用
05H 拒绝访问
50H 文件已存在

注释：

这是创建非临时文件的正常方法，该功能返回一个用于访问该文件的句柄。如果路径不存在，无可用句柄，或要在根目录下创建而根目录已满，则不能创建文件，该功能失败。

与功能 3CH 不同，如果文件已存在，该功能失败；所以可以用该功能来检测指定的文件是否存在。如果创建成功，则文件原先是不存在的。

以普通属性创建的文件可以进行读写访问，还可以用功能 43H 来改变属性。但不能用它创建卷标或子目录。有效的属性如表 1.53 所列：

表 1.53 文件类型

值	匹配的文件类型
00H	普通
02H	隐藏
04H	系统
06H	隐藏、系统

本功能的一个有趣的应用是在 PC 网络间完成信号量（Semaphore）传递机制。若该功能成功地创建了一个文件，程序就有了信号量，可以进入关键代码段（Critical Code Section）。若不能创建文件，可以周期地测试这种操作。

当创建此文件的程序以关键段结束时，它将删除该文件，因而释放了信号量。

在网络环境下，必须具有创建访问权才可以应用该功能。

INT 21H	功能 5CH 子功能 00H	V3
	设置文件访问锁定	

锁定文件指定的区域

调用寄存器：
- AH 5CH
- AL 00H
- BX 文件句柄
- CX 区域偏移值的高位
- DX 区域偏移值的低位
- SI 区域长度的高位
- DI 区域长度的低位

返回寄存器：
- 如果成功，进位标志清零
- 如果失败，进位标志置位
  - AX 错误代码
    - 01H 功能无效
    - 06H 句柄无效
    - 21H 锁定混乱（Lock Violation）
    - 24H 缓冲区共享超限

注释：

对于基于数据库和交流功能的网络环境，文件锁定是一个基本的操作。如果允许多个进程对一个文件的同一段进行写入，则结果会不确定。记录加锁就使得在一个程序开始写之前，另一个程序必须完成。它不保证按顺序写，而是仅保证两者互不干扰。

锁定和开锁就像 Pascal 中的 Begin-End 或 C 语言中的 { }，应始终匹配。每锁定一个文件，在同一程序中必须有对应的文件解锁。没有解锁将导致文件的状态不确定。

应用了文件锁定的程序应设法俘获所有可能的错误出口，以便在非正常情况下也能进行解锁处理。访问已锁定或可锁定文件的程序不应当直接访问文件。不应当依赖加锁机制来防

止直接冲突，正确过程是：首先，试图锁定文件的一部分并检查返回的错误代码；如果可以锁定，文件操作可以继续进行。若不能锁定，程序应当延迟后另试。

锁定机制包含了自动重试功能。用 IOCTL 功能 44H 子功能 0BH 可以改变重试次数和重试间隔。

用功能 45H 复制的文件句柄将继承对锁定区域的访问，用功能 EXEC（4BH）产生的程序不继承文件锁定。

INT 21H	功能 5CH 子功能 01H	V3
	清除文件访问的锁定	

对于用子功能 00H 锁定的文件区域进行解锁

调用寄存器：

AH	5CH
AL	01H
BX	文件句柄
CX	区域偏移值的高位
DX	区域偏移值的低位
SI	区域长度的高位
DI	区域长度的低位

返回寄存器：如果成功，进位标志清零

如果失败，进位标志置位

AX 错误代码

01H	功能无效
06H	句柄无效
21H	锁定混乱
24H	共享缓冲区超限

注释：

对于基于数据库和交流功能的网络环境，文件锁定是一个基本的操作。如果允许多个进程对一个文件的同一段进行写入，则结果会不确定。记录加锁就使得在一个程序开始写之前，另一个程序必须完成。它不保证按顺序写，而是仅保证两者互不干扰。

锁定和开锁就像 Pascal 中的 Begin-End 或 C 语言中的 { }，应始终匹配。每锁定一个文件，在同一程序中必须有对应的文件解锁。没有解锁将导致文件的状态不确定。

更详细的情况，参见对子功能 00H 的讨论。

INT 21H	功能 5DH 子功能 00H	V3
	复制数据到 DOS 保存区	

把 DS：SI 所指区域的 18 个字节复制到 DOS 内部寄存器保存区域（Register-save Area），当 DOS 返回到调用程序时，它就成为寄存器内容

调用寄存器： AH 5DH
AL 00H
返回寄存器： DS：SI 指向要复制的数据区的指针
注释：

这个功能替代了 DOS 所保存的寄存器值，可以把控制权交给系统的另一部分。它的目的尚不清楚。

INT 21H	功能 5DH	子功能 06H	V3
	取严重错误标志的地址		

返回一个指针，它指向存贮了系统严重错误标志的地方
调用寄存器： AH 5DH
AL 06H
返回寄存器： DS：SI 指向严重错误标志的指针
注释：

这个功能返回一个指向 DOS 存贮严重错误标志的指针，用于 DOS 判定是否已发生严重错误。

INT 21H	功能 5DH	子功能 0AH	V3
	设置错误数据值		

改变每一步 DOS 操作存于内部的系统错误数据代码
调用寄存器： AH 5DH
AL 0AH
DS：SI 指向 ERROR 结构的指针。ERROR 结构如下：

字	扩充错误代码
字节	待置的行动代码
字节	待置的类别代码
字	待置的轨迹代码
字	错误发生时 DX 的值
字	错误发生时 SI 的值
字	错误发生时 DI 的值
字	错误发生时 DS 的值
字	错误发生时 ES 的值
字	保留
字	计算机的 ID（本机为 0）
字	程序 ID（本身时为 0）

返回寄存器： 无

特别说明：据台湾旗标出版有限公司《DOS 5.0 技术手册（三）系统呼叫篇》指出 DS：SI 应为 DS：DX，并指出是 Microsoft 原手册的错误。

注释：

本子功能改变了 DOS 用于 INT 21H 功能 59H（取扩充错误信息）返回信息的内部存储单元。当检测到错误时，它可以同子功能 06H（取错误标志地址）一起使用，以保存错误代码，并在初步处理（可能修改存储的代码）完成后恢复它们。

TSR 代码可能因偶然改变了一个错误代码而被弹出，从而产生不准确结果，这些功能可防止这类事情发生。其方法是，用子功能 06H 来得到地址，把数据保存到自己的区域。保存时注意 DOS 内部的结构与子功能 0AH 使用的不同：

字节	轨迹代码
字	扩充错误代码
字节	行动代码
字节	分类代码
双字	指向驱动程序地址的指针

INT 21H	功能 5EH 子功能 00H	V3.1
	读取机器名	

获取网络机器名

调用寄存器：

AH	5EH
AL	00H 取机器名
DS：DX	指向接收机器名的缓冲区的指针
DS：SI	指向设置串（Setup String）的指针

返回寄存器：

如果成功，进位标志清零

CH＝00H	未定义名称
CH＞00H	定义了名称
CL	NETBIOS 名称号（CH＞00H）
DS：DX	指向标识符的指针（CH＞00H）

如果失败，进位标志置位

AX	错误代码
	01H 功能无效

注释：

机器名是一个 15 字节（有的书上指出是 16 字节）的 ASCIIZ 字符串，用于网络识别机器。这个功能要求网络在运行。若网络不运行，则其结果不可预测。

INT 21H	功能 5EH 子功能 01H	V3.1
	设置机器名	

设置网络机器名

调用寄存器： AH 5EH
AL 01H 设置机器名
DS：DX 指向以 ASCIIZ 字符串表示机器名的缓冲区的指针

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误码
01H 功能无效

注释：

机器名是一个 15 字节的 ASCIIZ 字符串，用于网络识别机器。这个功能要求网络正在运行。若网络不运行，则其结果不可预测；因为它要修改网络操作的基本信息，故只能由网络软件所用。

INT 21H	功能 5EH 子功能 02H	V3.1
	设置网络打印机配置 (Set Network Printer Setup)	

设置网络打印机设置 (Setup)

调用寄存器： AH 5EH
AL 02H 设置网络打印机配置
BX 重定向表索引
CX 设置串长度（最大为 64 字节）
DS：SI 指向设置串的指针

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误码
01H 功能无效

注释：

在执行任何网络打印任务之前，需要先发一个打印机配置串。该功能设置这样的串。

INT 21H	功能 5EH 子功能 03H	V3.1
	取网络打印机配置	

读取网络打印机设置串

调用寄存器： AH 5EH
AL 03H 取网络打印机设置
BX 重定向表索引
ES：DI 指针指向接收参数串的缓冲区的指针

返回寄存器： 如果成功，进位标志清零

CX 准备串长度
ES：DI指向设置串的指针
如果失败，进位标志置位
AX 错误码
01H 功能无效

注释：

在任何作业访问网络打印机之前，先发出一个打印机设置串。该功能取回这样的串。

INT 21H	功能 5FH	子功能 02H	V3.1
	取重定向表项		

读取网络重定向表项

调用寄存器： AH 5FH
AL 02H
BX 重定向表索引
DS：SI 指向设备名的128字节缓冲区的指针
ES：DI 指向网络名的128字节缓冲区的指针

返回寄存器： 如果成功，进位标志清零
BH 设备状态标志
位 0=0，设备有效
位 0=1，设备无效
BL 设备类型
03H 打印机
04H 磁盘驱动器
CX 存储的参数值
DX 被破坏
BP 被破坏
DS：SI 指向 ASCIIZ 本地设备名的指针
ES：DI 指向 ASCIIZ 网络名的指针
如果失败，进位标志置位
AX 错误码
01H 无效功能
12H 不再有文件

注释：

本功能用于获取网络上设备（打印机或磁盘目录）的重定向。为支持本功能，网络必须在运行状态。例如，可用本功能（以及相关的子功能03H）将磁盘驱动器标识符与一个网络目录相联，也可以把本地打印机名指派给要访问的远程打印机。本功能支持访问远程磁盘访问的远程口令。

必须用本功能来装载文件共享程序块。所有的标识符以ASCIIZ字符串来传递，因而与C

语言兼容而不直接与 Pascal 或 BASIC 兼容。当要读取重定向项时，子功能 02H 每次返回重定向表中的一个项，这些 ASCIIZ 字符串代表了本地设备名（DS：SI 指向）和网络名（ES：DI 指向）。

接连调用子功能 02H，当返回错误代码 12H 时，意味着到表头。尽管返回值不用 DX 和 BP，但该功能调用还是破坏了它们的值。

尽管可以多重定向，但 COM 设备、STDOUT 和 STDERR 是不可重定向的。

INT 21H	功能 5FH	子功能 03H	V3.1
	置重定向表项		

读取或修改网络重定向表项

调用寄存器：
- AH　5FH
- AL　03H
- BL　设备类型
  - 03H　打印机
  - 04H　磁盘驱动器
- CX　为调用程序保存的参数
- DS：SI　指向 ASCIIZ 本地设备名的指针
- ES：DI　指向后跟 ASCIIZ 口令的 ASCIIZ 网络名的指针

返回寄存器：
- 如果成功，进位标志清零
- 如果失败，进位标志置位
  - AX　错误码
    - 01H　功能无效
    - 03H　路径未找到
    - 05H　拒绝访问
    - 08H　内存不够
    - 0FH　驱动器无效
    - 12H　不再有文件
    - 57H　参数无效

注释：

本功能用于设置网络上设备（打印机或磁盘目录）的重定向表，通过修改与网络设备、文件、或目录相联的本地设备名表而实现该功能。为支持本功能，网络必须在运行状态。例如，可用本功能把一个磁盘驱动器标识符与一个网络目录相联，也可向要访问的远程打印机配以本地打印机名。该功能支持访问远程磁盘访问的口令。

必须用本功能来装载文件共享程序块。所有的标识符以 ASCIIZ 字符串来传递，因而与 C 语言兼容而不直接与 Pascal 或 BASIC 兼容。这些 ASCIIZ 字符串代表本地设备名（DS：SI 指向）和网络名（ES：DI 指向）。

子功能 03H 可以指定重定向。如指定打印机或磁盘重定向（BL＝设备类型），并指出本地名（A，B 等用于磁盘重定向；PRN：，LPT1：等用于打印机重定向）。若重定向了一个打

印机，则打印机输出被送入缓冲区，然而向合适的设备发送假联机打印；因这种重定向发生于INT 17H水平，除硬件访问打印机自身外，其他重定向均可俘获。

尽管可以多重定向，但COM设备、STDOUT和STDERR不可重定向。

INT 21H	功能 5FH 子功能 04H 取消重定向表项	V3.1

取消网络重定向表项

调用寄存器： AH 5FH
AL 04H
DS：SI 指向ASCIIZ设备名的指针

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误码
01H 功能无效
0FH 驱动器无效

注释：

本功能用于取消网络上设备（打印机或磁盘目录）的重定向。通过修改一张表而实现本功能。此表为一与网络设备、文件或目录相联的本地设备名表。只有网络运行时才支持本功能。例如，可用它把磁盘驱动器标识符与网络目录相联，也可以把本地打印机名指派给远程打印机。该功能支持远程磁盘访问的口令。

必须用本功能装载文件共享程序块。所有的标识符以ASCIIZ字符串传递，因而与C语言兼容而不直接与Pascal或BASIC兼容。

子功能04H仅用于本地设备名。若该设备被再分配，则重定向就被破坏了。当设备名以两个反斜线开始，则本地机器和网络目录之间的联系被打断。

尽管可以多重定向，但COM设备、STDOUT和STDERR不能被重定向。

INT 21H	功能 60H 扩展路径名字符串	V3

把一个可能包含替代或指定的（SUBSTed or ASSIGNed）驱动器的相对路径名扩展为一个包括物理驱动器的全路径名

调用寄存器： AH 60H
DS：SI 指向ASCIIZ相对路径名的的指针
ES：DI 67字节工作缓冲区地址

返回寄存器： 如果成功，进位标志清零
ES：DI 不变，工作缓冲区内包含了全路径的名称
如果失败，进位标志置位

AX 错误码

02H 在输入字符串中有非法字符

注释：

V3 中有个功能允许 EXEC 功能把程序的全路径名加到环境区域的尾部，该功能似乎就是这个功能。据说它可以完成附加标记的语法分析，但我们的测试表明任何一个合法的可做路径名的字符串（包括应用通配符？和＊）都被解释成单个路径。至于该名字的文件是否存在，该功能不作检测；只有输入字符中包含了非法字符才会报告出错。

INT 21H	功能 62H	V3
	取 PSP 地址	

为当前运行的程序取程序段前缀（PSP）的段地址

调用寄存器： AH 62H

返回寄存器： BX PSP 的段地址

注释：

该功能用于程序启动后，在任何时候取回 PSP 的段地址，而无需把它明确地存到一个可访问的区域内。大多数功能都避免直接访问 PSP，仅该功能有此能力。

在所有已测试的 DOS 版本中，该功能是从保留功能 51H 中复制过来的，用的代码完全相同。

INT 21H	功能 63H 子功能 00H	V2.25
	取系统引导字节表（Get System Lead Byte）	

取系统引导字节表的地址

调用寄存器： AH 63H

AL 00H

返回寄存器： DS：SI 指向引导字节表的指针

注释：

该功能返回系统引导字节表的地址。其数据结构与处理双字节字符的显示系统相关（如 Kanji 和 Hangeul）。本功能首次出现在 DOS V2.25 中，在 DOS V3 中是不可用的。但在 V4 中又出现了（无文档说明）。

在 V4 中，功能 63H 的其他子功能不可运行，调用时无错误指示。

INT 21H	功能 63H 子功能 01H	仅 V2.25
	设置/清除临时控制台标志	

控制临时控制台标志

调用寄存器： AH 63H

AL 01H

DL 00H 设置临时控制台标志

01H 清除临时控制台标志

返回寄存器： 无

注释：

该功能控制临时控制台标志。其数据结构与处理双字节字符的显示系统相关(如Kanji和Hangeul)。本功能仅在DOS 2.25中可用，在DOS V3及以上版本中不可用。

INT 21H	功能 63H 子功能 02H	仅 V2.25
	取临时控制台标志	

读取临时控制台标志的数值

调用寄存器： AH 63H

AL 02H

返回寄存器： DL 临时控制台标志的数值

注释：

该功能控制临时控制台标志。其数据结构与处理双字节字符的显示系统相关(如Kanji和Hangeul)。本功能仅在DOS 2.25中可用，在DOS V3及以上版本中不可用。

INT 21H	功能 64H	V3
	设置当前国别字节	

设置DOS内部的当前国别字节

调用寄存器： AH 64H

AL 当前国别字节值

返回寄存器： 无

注释：

AL寄存器的值被存贮在当前内部国别代码的地方。对于合法性不做检测，也不返回错误代码。所有寄存器保持不变。

INT 21H	功能 65H	V3
	取国别扩充信息	

返回指定国别的扩充信息

调用寄存器： AH 65H

AL 感兴趣的信息标识（1，2，4，5，6或7）或子功能（20H，21H或22H）

BX 感兴趣的代码页（-1=活动的CON设备）

CX 返回的数据量
DX 国别标识（缺省为－1）
ES：DI 指向返回信息缓冲区的指针

返回寄存器： 如果成功，进位标志清零
ES：DI 指向已返回信息缓冲区的指针
如果失败，进位标志设置
AX 错误代码
01H 功能无效
02H 文件未找到

注释：

有些程序要访问某国特有的信息（例如货币符号和日期格式），功能 65H 为用户程序返回这样的信息（依赖于指明的国别）。

表 1.54 可由国别标识来检索。缺省（－1）代表美国。调用检索的数据量由 CX 指明。如果表中有附加的数据，数据被截断并无错误信息返回。

表 1.54 扩充的国别信息

偏移	域宽	含义
		——扩充的国别信息缓冲区——
		信息标识：01
00H	字节	信息标识＝01
01H	字	度（38 或以下）
03H	字	国别标识
05H	字	代码页
07H	字	日期和时间格式代码
		0＝美国 月 日 年，时：分：秒
		1＝欧洲 日 月 年，时：分：秒
		2＝日本 年 月 日，时：分：秒
09H	5 字节	货币符号（ASCIIZ）
0EH	字节	千分隔符
0FH	字节	零
10H	字节	小数分隔符
11H	字节	零
12H	字节	日期分隔符
13H	字节	零
14H	字节	时间分隔符
15H	字节	零
16H	字节	货币格式
		00H＝符号在货币前，无空格
		01H＝符号在货币后，无空格
		02H＝符号在货币前，1 个空格
		03H＝符号在货币后，1 个空格
		04H＝符号替代小数分隔符

续表

偏移	域宽	含义
17H	字节	小数点后位数
18H	字节	时间格式
		位0=0，12小时计时
		位0=1，24小时计时
19H	双字	大小写映象调用地址
1DH	字	数据表分隔符
1EH	字节	零
1FH	10字节	保留
		——扩充的国别大写字母表——
		信息标识：02
00H	字节	信息标识=02
01H	双字	指针指向大写字母表，大写字母表长130字节；2字节的长度加上128字节的大写字母值
		——扩充的国别文件名大写字母表——
		信息标识：04
00H	字节	信息标识=04
01H	双字	指针指向文件名大写字母表，大写字母表长130字节；2字节的长度加上128字节的大写字母值
		——扩充的文件名字符表——
		信息标识：05
00H	字节	信息标识=05
01H	双字	指针指向文件名字母表。表开头是16位的长度，其后是指定国别文件名中不可用的字符
		——扩充国别排序表——
		信息标识：06
00H	字节	信息标识=06
01H	双字	指针指向排序表。排序表长258字节；2字节的长度加上以排序顺序排列的256字节
		——DBCS引导字节表（仅V4）——
		信息标识：07
00H	字	其后所跟字节数
02H	2字节	第一个引导字节的起止范围
04H	2字节	第二个引导字节的起止范围
.	2字节	（必要时重复）
.	2字节	0，0标记表尾；不在计数内

INT 21H	功能65H	子功能20H	V4
	字符转换		

用当前大写字符表把指定字符转换成大写格式

调用寄存器： AH 65H
AL 20H
DL 字符
返回寄存器： 如果成功，进位标志清零
DL 大写字符
如果失败，进位标志设置

注释：

本功能提供了一个可靠的方法把当前国别字符转换成大写格式。

INT 21H	功能 65H 子功能 21H 字符串转换	V4

用当前大写字符表把指定字符串转换成大写格式

调用寄存器： AH 65H
AL 21H
CX 字符串长度
DS：DX 指向字符串的指针
返回寄存器： 如果成功，进位标志清零

注释：

本功能提供了一个可靠的方法把当前的国别字符串转换成大写格式。

INT 21H	功能 65H 子功能 22H ASCIIZ 字符串转换	V4

用当前的大写字符表将指定的串转化成大写格式

调用寄存器： AH 65H
AL 22H
DS：DX 指向 ASCIIZ 字符串的指针
返回寄存器： 如果成功，进位标志清零

注释：

本功能提供了一个可靠的方法把当前的国别的字符串转换成大写格式。

INT 21H	功能 66H 子功能 01H 取全局代码页	V3.3

读取当前国别的代码页

调用寄存器： AH 66H
AL 01H

返回寄存器： 如果成功，进位标志清零
BX 活动代码页
DX 系统代码页
如果失败，进位标志设置
AX 错误代码
02H 文件未找到

注释：

本功能表明是哪一个COUNTRY.SYS中的国别数据驻留在代码页的国别缓冲区域。如果设备支持该国别，则该设备在文件CONFIG.SYS的代码页切换中会被自动选中。

INT 21H	功能 66H 子功能 02H 置全局代码页	V3.3

对当前国别设置代码页

调用寄存器： AH 66H
AL 02H
BX 活动代码页
DX 系统代码页

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志设置
AX 错误代码
02H 文件未找到

注释：

本功能将COUNTRY.SYS中的国别数据移到代码页的驻留国别缓冲区域。如果设备支持该国别，则该设备在文件CONFIG.SYS的代码页切换中会被自动选中。

INT 21H	功能 67H 置句柄计数	V3.3

让一个进程动态地修改进程允许使用的文件句柄数（正常情况为20）

调用寄存器： AH 67H
BX 允许打开的句柄数

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志设置
AX 错误代码

注释：

本功能允许程序在运行时调节文件句柄数。这对于复杂的数据库程序特别有用，因为这类程序需管理它们打开的大量文件。其内存由功能4AH释放后获得，如果全部内存少于打开

文件的数目，只有当前文件数减少到界限以下时，内存才可利用。

在 DOS V3.3 中，最多可设置 CONFIG.SYS 中 FILES＝255 个文件句柄。本功能可把文件数提高到 64K。如果说明的数目小于 20，会默认为 20。

INT 21H	功能 68H 刷新缓冲区到磁盘	V3

把文件的全部缓冲区数据刷新到设备中

调用寄存器： AH 68H
AL 文件句柄

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志设置
AX 错误代码

注释：

刷新缓冲区到磁盘的标准方法是先关闭该文件再重新打开。一个改进的办法是用功能 45H 复制该文件句柄，然后再关闭复制的句柄，这样去关闭文件就不用再打开。

然而，若用功能 68H 不需再用以上那些手段。若想刷新缓冲区，直接用它就行了，它比复制句柄或依次关闭/打开句柄之类的方法更快、更安全。

INT 21H	功能 6AH 分配内存	V4

分配一块可用内存，返回指向块头的指针

调用寄存器： AH 6AH
BX 文件句柄

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志设置
AX 错误代码

注释：

在 IBM V4.01 中，本功能的代码同写入文档的功能 48H 完全一样。

INT 21H	功能 6CH 扩充的文件打开/建立	V4

将所提供的功能 3CH，3DH 和 5BH 结合在一起，成为一个多用途的文件打开工具

调用寄存器： AH 6CH
AL 00H（要求）
BX 打开方式（位映象）

位

FEDCBA98	76543210	含义
....	000	只读
....	001	只写
....	010	读/写
....	011	未用,0
....	.....1xx	未用,0
....	....x...	未用,0
....	.000....	兼容方式
....	.001....	拒绝任何共享
....	.010....	拒绝写共享
....	.011....	拒绝读共享
....	.100....	不作拒绝
....	.101....	未用,0
....	.110....	未用,0
....	.111....	未用,0
....	0.......	子进程继承句柄
....	1.......	子进程不继承句柄
...xxxxx	....	未用,0
..0.....	....	用 INT 24H 处理程序
..1.....	....	仅返回错误
.0......	....	写可进入缓冲
.1......	....	所有的写立即执行
x.......	....	未用,0

CX 文件属性(位映象)

位

FEDCBA98	76543210	含义
....	0	读/写
....	1	只读
....	......0.	可见
....	......1.	隐含
....	.....0..	普通用户文件
....	.....1..	系统文件
....	....0...	非卷标
....	....1...	卷标
....	...x....	未用,0
....	..0.....	未修改
....	..1.....	修改过(档案)
xxxxxxxx	xx......	未用,0

DX 行动标志(位映象)

位		
FEDCBA98	76543210	含义
....	0000	文件存在时,显示 ERROR(失败)
....	0001	文件存在时,打开文件
....	0010	文件存在时,打开文件并清除其内容(替换)
....	0011	未用,0
....	....01xx	未用,0
....	....1xxx	未用,0
....	0000....	文件不存在,显示 ERROR(失败)
....	0001....	文件不存在,建立新文件
....	001x....	未用,0
....	01xx....	未用,0
....	1xxx....	未用,0
xxxxxxxx	....	未用,0

DS：SI 指向 ASCIIZ 文件路径名的指针

返回寄存器： 如果成功，进位标志清零

AX 文件句柄

CX 采取的行动

01 文件存在，已打开

02 文件不存在，已建立

03 文件存在，已替换

如果失败，进位标志置位

AX 错误代码

注释：

与组成它的功能 3CH，3DH 和 5BH 一样，调用本功能时，DS：SI 指向待打开或建立文件的路径名，AL 中存有所希望的访问权限，若想用此调用建立一个文件，则 CX 为设定的永久的属性位映象；与功能 3CH，3DH 和 5BH 不同，本功能在 DX 中带有附加信息，可告诉系统在文件存在或不存在时如何去做。有了它，在文件有问题时，可不用 INT 24H 严重错误处理程序去处理，而让程序自己去处理各种错误，这样就不用修改 DOS 的行动以免影响别的程序的运行。

本功能是为兼容 OS/2 而加入 DOS 之中的，只可在 V4 以后 DOS 版本中使用。

如果某部分路径未找到，该功能会失败（文件名部分除外，它通过 DX 控制）。在一个网络中运行时，用户的访问权至少要等于调用该功能时所指定的那样。

INT 22H	V1
终止地址	

这并不是中断，仅是当前程序运行完后控制应转去的例程的地址

调用寄存器： 无

返回寄存器： 无

注释：

加载程序时，在这一存储单元中的内容被复制到程序段前缀（PSP）的偏移量 0AH 处的字节。当程序终止时，该值从同一单元恢复。它仅是一个存贮区，不应被直接运行。

INT 23H	V1
Ctrl-C 中断处理程序	

当检测到 Ctrl-C（或 Ctrl-Break）后，该中断接受控制权

调用寄存器： 没用

返回寄存器： 没用

注释：

一旦在进行 I/O 操作期间检测到 Ctrl-C，或其他情况下 BREAK 置为 ON，系统就转向该向量给出的地址（Ctrl-Break 服务程序把 Ctrl-C 强迫压入 DOS 输入缓冲区，然后间接地进入例程）。当加载一个程序时，该向量被复制到程序段前缀（PSP）的 0EH 处，程序结束时恢复该向量。

Ctrl-C 和 Ctrl-Break 处理程序通常是最需要写的两个程序。好的程序不把控制权交给缺省的处理程序，而是控制所有 break 操作以正确地处理各种问题（尤其是没经验的程序员所写的通信程序会碰到这类错误）。

由于这些控制问题，应用程序不能调用 INT 23H。当系统处理 Ctrl-C 和 Ctrl-Break 字符时，该功能很快就起作用。

Ctrl-C 和 Ctrl-Break 处理程序处理 break 情况有许多选择项：

- 该处理程序可以设置一个局部标志，以便主程序检测后采取进一步的行动。主程序可先直接采取某些行动，然后执行中断返回（IRET），把控制权交给 DOS；接下来 DOS 从头开始调用这个中断。这对每毫秒就调用一次中断的应用程序很有用，如高速通信程序可用该方法来编码；
- 处理程序可针对引发中断的情况采取行动，然后通过远返回（ret far）将控制权返回 DOS。可设置进位标志以表明应用程序是否须放弃或清除；
- 处理程序采取任何它需要的行动，然后直接继续执行程序而不返回 DOS。

任何的选择都有效，视需要而定。若须尽量减少正常处理以外的时间，应做第一个选择；可以放弃进程时，第二个方法就很有用；当返回原始操作不好时，第三个方法可改向操作或直接继续进行。若无可选择，从程序退出也行。

在 Ctrl-C/Ctrl-Break 处理程序中，可调用任何 DOS 功能。

INT 24H	V1
严重错误处理程序	

当检测到严重错误时，该中断接受控制。严重错误通常表示某种硬件错误，一般是由于 DOS 调用不正当的设备驱动程序引起的

调用寄存器：	AH	错误信息
	AL	驱动器号
	DI	错误代码
	Bp：SI	指向设备驱动程序首部的指针
	STACK	设置如下：
高地址	FLAGS	原调用程序的标志寄存器
	CS	原调用程序的 CS 寄存器
	IP	原调用程序的 IP 寄存器
	ES	原调用程序的 ES 寄存器
	DS	原调用程序的 DS 寄存器
	BP	原调用程序的 BP 寄存器
	DI	原调用程序的 DI 寄存器
	SI	原调用程序的 SI 寄存器
	DX	原调用程序的 DX 寄存器
	CX	原调用程序的 CX 寄存器
	BX	原调用程序的 BX 寄存器
	AX	原调用程序的 AX 寄存器
	FLAGS	DOS 的标志寄存器
	CS	DOS 的 CS 寄存器
低地址	IP	DOS 的 IP 寄存器
返回寄存器：	AL	行动代码

注释：

装载一个程序时，该向量中的内容被读到程序段前缀（PSP）的偏移量 12H 开始的地方。当程序结束时，这个向量由终止处理程序恢复。绝不能直接调用该中断。

当 DOS 调用本处理程序时，AH，AL 和 DI 中存有错误特性信息，BP：SI 指向设备驱动程序首部。SS，SP，DS，ES，BX，CX 和 DX 寄存器必须由严重错误处理程序保留。DOS 在调用 INT 24H 以前，一般要重试三次。只有 DOS 功能 00H—0CH（字符 I/O），30H（取 DOS 版本）和 59H（取扩充错误信息）可在严重错误处理程序中被调用；其他功能调用会破坏 DOS 的内部堆栈，应避免。

如果是磁盘错误，AH 寄存器的第 7 位清零；否则，位 7 被置位。出错时，AH 寄存器的各位如表 1.55 所示，AL 中含有寄存器号（0=A，1=B 等）。

表 1.55 磁盘错误时 AH 寄存器用法

位	含义
7	0，表示磁盘错误
6	保留
5	0=不允许忽略响应 1=允许忽略响应
4	0=不允许重试响应 1=允许重试响应
3	0=不允许功能调用失败响应 1=允许功能调用失败响应
1—2	出错的磁盘区域 00=MS-DOS 区域 01=文件分配表（FAT） 10=根目录 11=文件区域
0	0=读盘 1=写盘

如果不是磁盘错误，可以检测［BP：SI+4］处的字；若第 15 位置位，错误是由于字符设备调用引起的。8 字符设备名从［BP：SI+10］处开始。

DI 低字节返回的错误代码与设备驱动程序请求头中返回错误代码完全相同，见表 1.56。

表 1.56 错误代码（DI 的低字节）

代码	含义
00H	写保护错误
01H	未知的单元
02H	驱动器未准备好
03H	不认识的命令
04H	数据错误（CRC 错误）
05H	请求结构长度错误
06H	查找错误
07H	未知介质类型
08H	扇区未找到
09H	打印机缺纸
0AH	写盘错误
0BH	读盘错误
0CH	一般错误
0FH	非法更换磁盘

当严重错误处理程序准备返回时，依据表 1.57 在 AL 中设置处理代码。

表 1.57 处理代码

代码	含义
00H	忽略错误
01H	重试操作
02H	终止程序
03H	系统调用失败（V3 及以上版本）

处理代码设置后，恢复寄存器内容，再从中断返回（IRET）。

该处理程序可以直接返回到用户程序，在引用中断返回（IRET）之前，它必须清除堆栈，删去堆栈中最后三个字以外的其他内容。这样，控制权就会转到发生 I/O 错误的功能调用之后的语句中。这时，DOS 会处于一个不稳定状态（直到完成另一个 0CH 以上功能调用后才脱离不稳定状态）。尤其是 DOS 的 CritErr 标志仍为置位，在 DOS 内部堆栈中会引起潜在错误。用户处理程序可以取得该标志的地址，并明确地清除它，这样就克服了上述问题（参见 INT 21H 功能 34H）；只有在别无选择时，才推荐使用这种修改方法。在大多数情况下，INT 24H 处理过程最好是经 DOS 返回，而不要直接返回。

附带注意处理程序的行动代码：有些程序员说在一些 DOS 版本中行动代码 2 不起作用。但是，还不知道具体是哪个版本。在 V4 之前的版本，行动代码 2 是通过 INT 23H 终止。在 V4 的《IBM 技术参考手册》中，改为 22H（正常终止过程）。显然早期的版本出现了印刷错误，因为实际上从未调用过 INT 23H。

在 DOS V4 中增加了扩充文件打开功能，可指明不用严重错误处理程序。即如果打开文件时寄存器的位设置适当，严重错误会自动以行动代码 3 返回，而不会进行屏幕对话。

INT 25H　　绝对磁盘读　　V1

从指定的磁盘扇区读数据到指定的内存区域

调用寄存器：
- AL　驱动器号（0=A，1=B 等）
- CX　>0，要读的扇区数
  - −1，用扩充格式（V4）
- DX　开始的相对（逻辑）扇区号
- DS：BX　若 CX>0，指向 DTA
  - 若 CX=−1，指向参数

返回寄存器：
- 如果成功，进位标志清零
- 如果失败，进位标志置位
  - AX　错误代码
    - 0207H　所用格式错误（仅 V4）

注释：

本功能直接访问逻辑扇区，把扇区上的数据读到内存中。因为绕过了 DOS 的目录结构，

必须小心对待这类访问。

逻辑扇区从 0 磁道 0 磁头开始定位，该磁道的第一个扇区是磁盘 0 扇区。扇区编号然后到下一个磁头，然后到下一个磁道，如此下去。逻辑扇区与磁盘本身所存的扇区号相对应，可能与物理扇区不对应。通过指定对应关系，逻辑扇区在磁盘物理位置上可以是分散的。位置上分散可提高磁盘效率。

如果调用该功能后进位标志置位，AX 寄存器的解释可能相应的表格。AH 和 AL 是分离与错误代码（见表 1.58）。

绝对磁盘读破坏除段寄存器以外的任何寄存器的内容，在调用之前，应保存有关值。

有一个问题使得高级语言难以直接调用该功能：INT 25H 曾将含有 CPU Flags 的字压入堆栈中，从功能调用返回时，此字还在堆栈中。为了摆脱此字并把堆栈恢复为希望的状态，可以做一个 POPF 把该数从堆栈中去掉，或者通过 ADD SP，2 增大堆栈指针而越过这个数。因为高级语言不能这样做，不得不用汇编语言来调用该功能，以防止系统出错（如 Turbo Pascal 中，可嵌入汇编代码。）。

在 DOS V4 中，逻辑扇区号被扩充为 32 位。为了与此配套，创立了 INT 25H 扩充格式；当读大于 32M 的卷（Volume）时，即使扇区号可以用 16 位表示，也必须用扩充格式来读。

表 1.58 INT 25H 返回错误代码的解释

代码	含义
	——AH 错误代码——
80H	磁盘控制器没有响应
40H	寻找操作错误
20H	控制器出错
10H	数据错误（CRC 错误）
08H	DMA 失败
04H	未找到请求的扇区
03H	写保护错误
02H	地址标记错误
01H	命令错误
	——AL 错误代码——
00H	写保护错误
01H	未知单元
02H	驱动器未准备好
03H	不认识的命令
04H	数据错误（CRC 错误）
05H	请求结构长度不对
06H	寻找错误
07H	不认识的介质类型
08H	扇区未找到
09H	打印机缺纸
0AH	写错误
0BH	读错误
0CH	一般错误

使用扩充格式要设置 CX 为 FFFFH（−1），DS：BX 解释为参数块地址而不是数据的缓冲区地址。参数块是按表 1.59 的格式安排的：

表 1.59 参数块格式

偏移量	长度	注释
00H	双字	基于 0 的逻辑扇区号
04H	字	要传送的扇区数
06H	双字	指向数据缓冲区的指针

INT 26H 绝对磁盘写 V1

把指明的磁盘传送区（DTA）内的数据写到指定的磁盘扇区。

调用寄存器： AL 驱动器号（0=A，1=B 等）
CX >0，要写的扇区数
−1，应用扩充格式（V4）
DX 开始的相对逻辑扇区号
DS：BX 若 CX>0，指向 DTA
若 CX=−1，指向参数

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误代码
0207H 所用格式错误（仅 V4）

注释：

请参见 INT 25H 的注释。

INT 27H 终止并驻留程序 V1

终止当前运行的程序，但把它保留在内存中

调用寄存器： DX 要驻留程序最后一个字节的偏移量（相对于 PSP）再加 1
CS PSP 的段地址

返回寄存器： 无

注释：

终止并驻留（TSR）工具用得很多，比如 SideKick 就是一个看似多任务操作的工具。本中断是原始的（DOS V1）TSR 终止过程，程序在设置所需要的中断之后，该功能使程序驻留在它本身所在的内存。

终止时，程序先恢复 INT 22H，INT 23H 和 INT 24H，然后把控制权交给终止地址。它使程序留在其内存区域（DX 指示保存区域大小），以保持这个 TSR 的活性。

这个中断受到很多限制。首先，必须把 CS 设置为 PSP——通常 CS 即为 PSP，但还需确认一下；最明显的是它只能为 TSR 程序保留 64K 字节。在 DOS V2 和 V3 中，用得更多的是

INT 21H 的功能 31H，因为它允许任意大的内存、并无需设置 CS；而 INT 27H 只用于 DOS V1.x 系统。

与一般的终止过程不一样，该中断并不关闭它打开的文件。若想关闭文件，必须在应用该中断前明确地关闭它。

INT 28H	V1
DOS 处于命令行状态可安全使用	

在 DOS 控制台 I/O 查询循环时常调用本中断，以使得驻留（TSR）程序（例如 DOS 提供的 PRINT.COM）知道使用文件操作或非字符 I/O 的 INT 21H 功能是安全的

调用寄存器： 无

返回寄存器： 无

注释：

在 DOS 进入控制台 I/O 查询循环的好几个地方调用此中断，这样进行文件操作或大多数 0CH 以上功能调用就安全了。

通常，INT 28H 中断向量为一个简单的 IRET 指令，它是一个伪处理程序；用户编写的任何处理程序应连接在下一行，以使 IRET 能返回 DOS。这样，当 COMMAND.COM 在等待输入命令行时，所有依赖于该中断的程序都可以运行。

注意：在 DOS V2 中，由于 DOS 内部堆栈应用冲突，并不是所有 0CH 以上功能调用都安全。参考 INT 21H 功能 34H 注释中对此的讨论及解决方法。

INT 29H	V2
快速输出字符	

DOS 输出例程中断

调用寄存器： AL 要显示字符

返回寄存器： 无

注释：

如果 DOS 向一个设备输出，而该设备驱动程序属性字的位 3 置 1，DOS 输出例程就调用该中断。除回车、换行和响铃字符外，所有 ASCII 字符输出不做任何变动，以上三个字符当成控制字符处理。

该中断一般只由 DOS 设备驱动程序使用，除 DOS V2.0 外，没有将其载入文档。但在最近的版本中，ANSI.SYS 就是用该方法输出，因此可期望它以后会受到支持。

INT 2AH	V3
Microsoft 网络接口	

该中断有许多功能，它可避免网络系统中用户之间干扰

调用寄存器： 无

返回寄存器： 无

注释：

在网络上操作时，DOS 经常调用该中断来防止多进程间的干扰。当网络不存在时，禁止调用它；在安装上网络后，禁止码（Disabling Code）自动解除，并调用该中断以控制对关键代码区的访问。不可在应用程序中用它！只有装上网络软件后它才是 DOS 的一部分，该中断功能超出了本书的范围。

INT 2EH		V3 (?)
	初始的 Shell 程序加载器	

在初始的命令解释程序 Shell 下加载一个程序，解释执行一条 DOS 命令

调用寄存器： DS：SI 指向已计数的、以 CR（回车）结束的命令字符串，它与由 INT 21H 功能 0AH 提供给 DOS 的串相同

返回寄存器： 返回到由 INT 22H 向量指向的地址

注释：

该功能可访问主环境区，它基本已被 EXEC 功能（INT 21H 功能 4BH 子功能 00H）取代。

使用该功能而不用 EXEC 的已知的唯一原因是：它可以访问系统的主环境区并改变它。使用了本功能的程序在网络中或多用户情况下可能无法运行。

此功能是未写入文档的功能。

INT 2FH		V3
	多路服务中断	

多路服务仅部分载入文档。功能号由入口处 AH 寄存器的内容来指定，AH 的值在 01H 到 FFH 的范围内

注释：

附加服务功能只有当相应的程序安装后，才加入到服务功能中。例如，只有安装 DOS 打印假脱机程序（PRINT.COM）后，功能 01H 才有意义。

功能号 00H 到 BFH 由 DOS 保留，功能 C0H 到 FFH 已载入文档供用户使用。原先分界线为 80H 而不是 C0H，在 DOS APPEND（DOS V3.3）功能 B7H 出现之后，才改变过来。许多用户程序接管 C0H 以下功能。至少有三分之一的实用程序搜寻了 INT 2FH 链以找到可用代码。

为了保持一致，所有功能的 00H 子功能（AL 中内容）被正式保留，用于读取安装状态；如写入文档的功能 01H：返回时，AL 中为 00H 表明该功能未安装，但可以安装；01H 表明该功能未安装并不可安装；FFH 表明该功能已安装。根据这一标准，可以避免重复安装多个同样的驻留工具。不幸的是，不是所有的 INT 2FH 功能符合这一标准。

INT 2FH	功能 01H 子功能 00H	V3
	打印安装检查	

DOS打印假脱机程序（PRINT.COM）的驻留部分的接口

调用寄存器： AH 01H

AL 00H

返回寄存器： 如果成功，进位标志清零

AL 状态

00H 未安装，准备好安装

01H 未安装，未准备好安装

FFH 已安装

注释：

本功能00H判定假脱机程序是否已安装。

INT 2FH	功能 01H 子功能 01H	V3
	向打印假脱机程序提交文件	

DOS打印假脱机程序（PRINT.COM）的驻留部分的接口

调用寄存器： AH 01H

AL 01H

DS：DX 指向包（Packet）地址的指针

返回寄存器： 如果成功，进位标志清零

如果失败，进位标志置位

AX 错误代码

01H 功能无效

02H 文件未找到

03H 路径未找到

04H 打开文件太多

05H 访问被拒绝

08H 队列已满

09H 假脱机程序正忙

0CH 文件名过长

0FH 驱动器无效

注释：

本功能需提供一个5字节的包，第一个字节为优先级，其后是一个指针，指向要打印文件的ASCIIZ字符串。假脱机程序取得数据后，若不进行人为中止，将自动打印文件。

INT 2FH	功能 01H 子功能 02H	V3
	从打印队列中去掉文件	

DOS 打印假脱机程序（PRINT.COM）的驻留部分的接口

调用寄存器： AH 01H
AL 02H
DS：DX 指向 ASCIIZ 文件说明的指针

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误代码
01H 功能无效
02H 文件未找到
03H 路径未找到
04H 打开文件太多
05H 访问被拒绝
08H 队列已满
09H 假脱机程序正忙
0CH 文件名过长
0FH 驱动器无效

注释：

子功能 02H 接受文件说明中的通配符（* 和?），允许终止多个打印文件。

INT 2FH	功能 01H 子功能 03H	V3
	取消打印队列中的全部文件	

停止当前的打印任务，并删除打印队列中的全部文件

调用寄存器： AH 01H
AL 03H

返回寄存器： 无

注释：

该功能并不清除正在打印文件的最后一页；若要清除，可以让调用程序发一个换页符。

INT 2FH	功能 01H 子功能 04H	V3
	暂停打印任务	

DOS 打印假脱机程序（PRINT.COM）驻留部分的接口

调用寄存器： AH 01H
AL 04H

返回寄存器： 如果成功，进位标志清零
DX 错误计数
DS：SI 指向打印队列的指针
如果失败，进位标志置位
AX 错误代码
01H 功能无效
09H 假脱机程序忙

注释：

子功能 04H 返回一个指针，指向一系列文件名项，每项长 64 字节，包含一个指明要打印文件的 ASCIIZ 文件说明。第一个项为当前正在打印的文件；最后一项的首字节为 NUL 字符（文件名字符串长度为 0）。该功能同时将假脱机程序推入 HOLD 状态，这样在调用程序再操作它之前，信息保持不变。子功能 05H 释放 HOLD 状态。

INT 2FH	功能 01H	子功能 05H	V3
结束打印 HOLD 状态			

DOS 打印假脱机程序（PRINT.COM）驻留部分的接口

调用寄存器： AH 01H
AL 05H

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误代码
01H 功能无效
09H 假脱机程序忙

注释：

子功能 05H 取消应用子功能 04H 所建立的 HOLD 状态；在这两次功能调用之间，所有的打印假脱机操作都不能进行。

INT 2FH	功能 01H	子功能 06H	V3
取打印机设备状态			

返回当前打印机设备头的地址

调用寄存器： AH 01H
AL 06H

返回寄存器： 若队列为空，进位标志清零
AX 0000H
若队列非空，进位标志置位
DS：SI 指向设备头的指针

AX 0008H

注释：

该功能可用于判定打印机队列是否为空。

INT 2FH 功能 05H V3
取外部严重错误处理程序安装状态

外部严重错误处理程序的接口

调用寄存器： AH 05H
AL 00H

返回寄存器： 如果成功，进位标志清零

AL	状态
00H	未安装，已准备好安装
01H	未安装，未准备好安装
FFH	已安装

注释：

还不完全了解这个未写入文档的功能。它似乎允许与 DOS 严重错误处理程序直接通信。

INT 2FH 功能 06H V3
取 ASSIGN.COM 或 ASSIGN.EXE 安装状态

判定 ASSIGN.COM 或 ASSIGN.EXE 的安装状态

调用寄存器： AH 06H
AL 00H

返回寄存器： 如果成功，进位标志清零

AL	状态
00H	未安装，已准备好安装
01H	未安装，未准备好安装
FFH	已安装

注释：

该功能为与 ASSIGN.EXE 的接口，若 ASSIGN 未加载，该功能不做任何事情。

INT 2FH 功能 08H V3
取 DRIVER.SYS 安装状态

判定 DRIVER.SYS 的安装状态

调用寄存器： AH 08H
AL 00H

返回寄存器： 如果成功，进位标志清零

AL 状态

00H 未安装，已准备好安装

01H 未安装，未准备好安装

FFH 已安装

注释：

该功能为与 DRIVER.SYS 的接口，若该文件未加载，不做任何事情。

INT 2FH	功能 10H	V3
	取 SHARE.EXE 安装状态	

判定 SHARE.EXE 的安装状态

调用寄存器： AH 10H

AL 00H

返回寄存器： 如果成功，进位标志清零

AL 状态

00H 未安装，已准备好安装

01H 未安装，未准备好安装

FFH 已安装

注释：

该功能为与 SHARE.EXE 的接口，若该文件未加载，不做任何事情。

INT 2FH	功能 11H	V3
	取网络重定向器安装状态	

判定网络重定向接口是否已安装

调用寄存器： AH 11H

AL 00H

返回寄存器： 如果成功，进位标志清零

AL 状态

00H 未安装，已准备好安装

01H 未安装，未准备好安装

FFH 已安装

注释：

该功能为与标准网络重定向器例程的接口，若未加载 Microsoft Networks 或完全兼容的程序，该功能不做任何事情。

INT 2FH	功能 12H	子功能 00H	V3
	取 DOS 安装状态		

确定 DOS 是否已安装；主要是为了与其他 DOS 功能保持一致

调用寄存器： AH 12H
AL 00H

返回寄存器： AL FFH（返回 DOS 总是安装的）

注释：

功能 12H 可访问某些 DOS 内部服务（V3 中 00H—25H，V4 中 00H—2FH）。注意，其中许多只有在全部段寄存器都设为 DOS 内核段（Kernal Segment）时才可调用；若不满足这种限制，极有可能毁坏数据。但它们确实提供了其他方法难以获得的信息，开发者采取适当预防措施后方可调用它们。

子功能 00H 总是返回“已安装”，这是因为该功能已被写入 DOS 内核。它是标准的“该功能可用吗?”测试，显然它是为了保持与做安装检查的其他 DOS 功能的一致性。

INT 2FH	功能 12H	子功能 01H	V3
	刷新（Flush）文件		

访问 DOS 内部服务以刷新文件

调用寄存器： AH 12H
AL 01H
BX 文件句柄
AX 错误代码

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误代码

注释：

关于调用本子功能应注意的事项，参见 INT 2FH 功能 12H 子功能 00H 的注释。

子功能 01H 是刷新句柄的文件。即，它把关于该文件的所有积累在缓冲区中的数据写到磁盘上，它假设全部段寄存器已指向 DOS 内核区。还不清楚这一动作是否关闭该文件。

INT 2FH	功能 12H	子功能 02H	V3
	取中断向量的地址		

访问 DOS 内部服务，检索中断向量

调用寄存器： AH 12H
AL 02H
STACK 要获取的中断号

返回寄存器： ES：BX 指向中断向量的远指针

注释：

关于调用本子功能应注意的事项，参见INT 2FH功能12H子功能00H的注释。

子功能02H提供了直接访问内部例程的方法，该例程是INT 21H功能25H和35H用来确定中断向量表地址的。使用该功能，必须把中断向量号传递到位于堆栈顶端的字的低字节。在ES和BX中返回一个指向中断向量本身（不是中断服务例程）的远指针。

INT 2FH	功能12H 子功能03H	V3
	取DOS数据段地址	

访问DOS内部服务，返回DOS内核数据段地址的值

调用寄存器： AH 12H

AL 03H

返回寄存器： DS DOS内核段地址

注释：

关于调用本子功能应注意的事项，参见INT 2FH功能12H子功能00H的注释。

子功能03H获取DOS内核段地址，用该地址可以把其他段寄存器指向DOS内核区域。由于该调用破坏DS内容，调用程序的DS值应当首先保存起来以便恢复。

INT 2FH	功能12H 子功能04H	V3
	设置标准路径分隔符	

访问DOS内部服务，设置标准路径分隔符

调用寄存器： AH 12H

AL 04H

STACK 要处理的分隔符（在低字节）

返回寄存器： AL 5CH（ASCII "\"）

注释：

关于调用本子功能应注意的事项，参见INT 2FH功能12H子功能00H的注释。

子功能04H把备用的路径分隔符（/）转换成标准分隔符（\）。

INT 2FH	功能12H 子功能05H	V3
	输出一个字符	

访问DOS内部服务，用INT 29H输出一个字符

调用寄存器： AH 12H

AL 05H

STACK 要输出的字符（在低字节）

返回寄存器： 无

注释：

关于调用本子功能应注意的事项，参见 INT 2FH 功能 12H 子功能 00H 的注释。

子功能 05H 把堆栈顶端的字符用 INT 29H 发送到 CRT。如果 INT 29H 可用，该功能就是多余的。

INT 2FH	功能 12H	子功能 06	V3
	调用严重错误处理程序		

访问 DOS 内部服务，就像遇到错误时那样调用严重错误处理程序

调用寄存器： AH 12H

AL 06H

返回寄存器： AL 行动代码（详见 INT 24H）

注释：

关于调用本子功能应注意的事项，参见 INT 2FH 功能 12H 子功能 00H 的注释。

子功能 06H 就像遇到错误时那样调用严重错误处理程序，并返回行动代码。

INT 2FH	功能 12H	子功能 07	仅 V3
	移动磁盘缓冲区		

访问 DOS 内部服务（仅在 DOS V3 中），移动磁盘缓冲区

调用寄存器： AH 12H

AL 07H

返回寄存器： DS：SI 指向磁盘缓冲区的指针

注释：

关于调用本子功能应注意的事项，参见 INT 2FH 功能 12H 子功能 00H 的注释。

子功能 07H 管理由 CONFIG.SYS 中“BUFFERS=”所建立的 DOS 内部磁盘缓冲区。除非对缓冲区非常熟悉，否则，不应该应用本功能。由于文件分配表（FAT）经常驻留在这些缓冲区内，应用错误可能会破坏硬盘上的全部数据。

INT 2FH	功能 12H	子功能 08H	V3
	用户计数减一		

访问 DOS 内部服务，将设备控制块的用户计数字减一

调用寄存器： AH 12H

AL 08H

ES：DI 指向文件或设备的DCB的指针

返回寄存器： AX 新的用户计数

注释：

子功能08H将DOS内部设备控制块的用户计数字减一。调用时，ES：DI指向适当的DCB（因为用户计数是DCB中第一个字，DI自动指向计数字本身），返回时DCB用户计数已减一，新的用户计数在AX寄存器中。通常该例程是作为处理关闭句柄功能的一部分被调用，但把ES：DI指向RAM中的其他字，同样可以将该字减一。它对段寄存器设置没有要求。

INT 2FH	功能12H 子功能0CH	V3
	DOS IOCTL打开文件	

访问DOS内部服务，用IOCTL过程来打开文件或设备

调用寄存器： AH 12H

AL 0CH

返回寄存器： 如果成功，进位标志清零

注释：

关于调用本子功能应注意的事项，参见INT 2FH功能12H子功能00H的注释。

通过IOCTL的打开子功能，子功能0CH打开一个设备或文件，该设备或文件在此之前已被DOS内部指针指定为当前的对象。它仅可以被DOS调用，或者是能作为控制DOS全部内部表和标志的程序使用。

INT 2FH	功能12H 子功能0DH	V3
	为文件关闭而取日期和时间	

访问DOS内部服务，检索系统的日期和时间用以标明文件的日期和时间

调用寄存器： AH 12H

AL 0DH

返回寄存器： AX 以压缩（文件）格式表示的系统日期

DX 以压缩（文件）格式表示的系统时间

注释：

关于调用本子功能应注意的事项，参见INT 2FH功能12H子功能00H的注释。

当关闭一个修改过的文件时，DOS调用子功能0DH，以获取用文件目录格式表示的系统日期和时间。该功能假设所有的段寄存器指向DOS的内核。

INT 2FH	功能12H 子功能0EH	仅V3
	搜寻缓冲区链	

访问DOS内部服务，在DOS V3之下搜寻缓冲区链

调用寄存器： AH 12H
AL 0EH
返回寄存器： 未知
注释：

关于调用本子功能应注意的事项，参见 INT 2FH 功能 12H 子功能 00H 的注释。

子功能 0EH 的目的还不完全清楚。它似乎是寻找 DOS 所用的磁盘缓冲区。使用时应小心。参见子功能 07H 的注释。

INT 2FH	功能 12H	子功能 10H	V3
寻找修改了的缓冲区（仅 V3），或执行时间延迟（V4）			

访问 DOS 内部服务，寻找已修改的磁盘缓冲区（在 DOS V3）或执行一个时间延迟（在 DOS V4）

调用寄存器： AH 12H
AL 10H
返回寄存器： 如果成功，进位标志清零
注释：

关于调用本子功能应注意的事项，参见 INT 2FH 功能 12H 子功能 00H 的注释。

子功能 10H 在 V3 之中用于定位一个重写了的（Dirty）缓冲区（也就是，一个已被修改的、需写到磁盘上的缓冲区）。在 V4 之中引入了全面的缓冲区重写模式，该子功能就过时了，变成了时间延迟例程，提供一个无害的空操作。除 DOS 本身之外，不应该用该功能。

INT 2FH	功能 12H	子功能 11H	V3
ASCIIZ 文件名规范化			

访问 DOS 内部服务，对于路径名进行规范化转换

调用寄存器： AH 12H
AL 11H
DS：SI 指向要规范化的文件名的指针
ES：DI 指向接受输出的缓冲区的指针
返回寄存器： ES：DI 不变，指向规范化的文件名，全部字母大写并且所有“/”字符变为“\”字符

注释：

子功能 11H 把文件或路径名转换成为其他 DOS 功能可用的标准格式。所有小写字母转换成大写，所有斜线转换成反斜线。输入输出都是 ASCIIZ 字符串。该功能对于段寄存器没有做假设。

INT 2FH 功能 12H 子功能 12H V3
确定 ASCIIZ 字符串长度

访问 DOS 内部服务，确定一个由 ES：DI 指向的 ASCIIZ 字符串的长度

调用寄存器： AH 12H
AL 12H
ES：DI 指向 ASCIIZ 字符串的指针

返回寄存器： CX 以字节表示的串长度

注释：

关于调用本子功能应注意的事项，参见 INT 2FH 功能 12H 子功能 00H 的注释。

子功能 12H 统计由 ES：DI 所指的 ASCIIZ 字符串的字节数，在 CX 寄存器中返回计数值（不包括结尾的 NUL 字符）。假设 SS 寄存器指向 DOS 内核区。

INT 2FH 功能 12H 子功能 13H V3
国别和大小写转换

访问 DOS 内部服务，转换 ASCII 字符的大小写（及国别信息，如果需要的话）

调用寄存器： AH 12H
AL 13H
STACK 要转换的字符

返回寄存器： AL 大写字符

注释：

关于调用本子功能应注意的事项，参见 INT 2FH 功能 12H 子功能 00H 的注释。

子功能 13H 从栈顶取得一个字，把它的低字节转换成大写；若国别转换起作用，就把扩充的 ASCII 字符转换成为正常的 ASCII 字符。结果在 AL 中返回。该功能假定 SS 寄存器指向 DOS 内核代码，以定位国别转换表和转换标志；因此，在大多数情况下，用不着它。

INT 2FH 功能 12H 子功能 14H V3
比较 32 位数字

访问 DOS 内部服务，比较两个 32 位值（典型为指针）

调用寄存器： AH 12H
AL 14H
DS：SI 第一个指针
ES：DI 第二个指针

返回寄存器： 若指针相等，零标志（Zero Flag）位置位
若不相等，则零标志位清除

注释：

子功能 14H 可比较任何两个 32 位数。DOS 一般用它作指针比较。它对段寄存器不做任何假设。

INT 2FH	功能 12H 子功能 16H 取 DCB 地址	V3

访问 DOS 内部服务，检索一个指定句柄的设备控制块地址

调用寄存器： AH 12H
AL 16H
BX 文件句柄

返回寄存器： 如果成功，进位标志清零
ES：DI 指向与该句柄对应的 DCB 的指针
如果失败，进位标志置位
AX 错误代码

注释：

关于调用本子功能应注意的事项，参见 INT 2FH 功能 12H 子功能 00H 的注释。

子功能 16H 直接访问文件或设备句柄所使用的设备控制块。但是，该功能假定所有的段寄存器指向 DOS 内核，因此，它主要由 DOS 本身使用。

INT 2FH	功能 12H 子功能 17H 取 LDT 地址	V3

访问 DOS 内部服务，返回逻辑驱动器表的地址

调用寄存器： AH 12H
AL 17H
STACK 字，驱动器代码（0=A，1=B 等）

返回寄存器： 如果成功，进位标志清零
DS：SI 指向该驱动器的逻辑驱动器表（DOS 内部指针也置位）
如果失败，进位标志置位
AX 错误代码

注释：

关于调用本子功能应注意的事项，参见 INT 2FH 功能 12H 子功能 00H 的注释。

子功能 17H 假定全部段寄存器指向 DOS 内核代码，因此主要是对 DOS 有用。该功能置当前驱动器的指针，并返回指定驱动器的表地址。

INT 2FH	功能 12H	子功能 18H	V3
	取用户堆栈地址		

访问 DOS 内部服务，获取用户堆栈地址

调用寄存器： AH 12H

AL 18H

返回寄存器： DS：SI 指向 DOS 堆栈区域的指针，在进入 INT 21H 时，寄存器的内容被保留在该处

注释：

在进入 INT 21H 时，DOS 在压入全部寄存器的内容之后，把 SS 和 SP 存于某处，子功能 18H 从该处装载 DS 和 SI。这就可以直接控制 INT 21H 返回值（除非后面的代码又改变了这些值）。该功能用途有限，对于段寄存器并没有做任何假设。

INT 2FH	功能 12H	子功能 19H	V3
	置 LDT 指针		

访问 DOS 内部服务，设置逻辑驱动器表指针

调用寄存器： AH 12H

AL 19H

返回寄存器： 无

注释：

关于调用本子功能应注意的事项，参见 INT 2FH 功能 12H 子功能 00H 的注释。

子功能 19H 使用 DOS 内部数据建立 LDT 指针，以与当前选择的驱动器保持一致。它假设全部段寄存器指向 DOS 内核。除 DOS 本身以外，它的实用性不大。

INT 2FH	功能 12H	子功能 1AH	V3
	从路径名中读取驱动器代码		

访问 DOS 内部服务，从所给的 ASCIIZ 路径名中分析驱动器代码

调用寄存器： AH 12H

AL 1AH

DS：SI 指向 ASCIIZ 路径名的指针

返回寄存器： AL 驱动器代码（0=缺省，1=A 等）

DS：SI 若有驱动器，指针越过所说明的驱动器；否则不变

注释：

子功能 1AH 确定在给它的路径名中是否指明了驱动器，若指明了，把驱动器字母转换成数值，并使 SI 向前移动，越过驱动器说明部分。该功能对于段寄存器未做假设。

INT 2FH	功能 12H	子功能 1BH	V3
	闰年调整		

访问 DOS 内部服务，根据所提供的年份是否为闰年来设置二月份的天数

调用寄存器： AH 12H
AL 1BH
CX 年份（完整值，如 1992）

返回寄存器： AL 若为闰年，即为 29；否则为 28

注释：

关于调用本子功能应注意的事项，参见 INT 2FH 功能 12H 子功能 00H 的注释。

子功能 1BH 假设所有的段寄存器指向 DOS 内核，在 DOS 之外的应用有限。它检测 CL 中的值以判断该年是否为闰年，若是，则相应地修改 DOS 中每月天数表。在任何情况下，均在 AL 中返回指明年份二月份的天数。该功能是 DOS 的时间/日期处理过程的一部分。

INT 2FH	功能 12H	子功能 1CH	V3
	计算从月初以来的天数		

访问 DOS 内部服务，返回从月初开始到现在已过去的天数

调用寄存器： AH 12H
AL 1CH
CX 当月的数字代码
DX 以前年份的总天数（若是从今年开始，为 0）
DS：SI 指向每月天数表的指针

返回寄存器： DX 当月开始的总天数

注释：

关于调用本子功能应注意的事项，参见 INT 2FH 功能 12H 子功能 00H 的注释。

子功能 1CH 假定所有段寄存器指向 DOS 内核，在 DOS 以外应用有限。它计算出从 DOS 的“零天”（1980 年 1 月 1 日）到现在共过去的天数，或者是从当年开始（如果 DX＝0）到当月开始之间的天数。该功能是 DOS 时间/日期处理的一部分。

INT 2FH	功能 12H	子功能 1DH	V3
	计算日期		

访问 DOS 内部服务，当提供了已过去的天数后计算所对应的月和天

调用寄存器： AH 12H
AL 1DH
CX 00H
DX 今年已过去的天数

DS：SI 指针指向每月天数表

返回寄存器： CX 月份

DX 天

注释：

关于调用本子功能应注意的事项，参见INT 2FH功能12H子功能00H的注释。

子功能1DH假定所有的段寄存器指向DOS的内核，在DOS以外应用有限。它接受了今年已过去天数，计算出当前的月和日数值。该功能是DOS的时间/日期处理过程的一部分。

该功能与功能12H子功能1CH相反。它假设入口处DX小于367，但若该条件不满足也不报告出错。如果DX超过范围，结果不可预料。由于该功能从未公开，故未包括错误检测。

INT 2FH	功能 12H 子功能 1EH	V3
	字符串比较	

访问DOS内部服务，比较测试两个ASCIIZ字符串

调用寄存器： AH 12H

AL 1EH

DS：SI 指向一个串的指针

ES：DI 指向另一串的指针

返回寄存器： 若两串相匹配，零标志（Zero Flag）置位，否则，零标志清零

注释：

子功能1EH对两个ASCIIZ串作一般的相等测试，对于段寄存器没有做假定。若两串有差别，该功能只返回它们是否相同的信息。

INT 2FH	功能 12H 子功能 1FH	V3
	初始化LDT	

访问DOS内部服务，对指定驱动器建立逻辑驱动器表

调用寄存器： AH 12H

AL 1FH

STACK 以ASCII表示的驱动器字母

返回寄存器： 如果成功，进位标志清零

如果失败，进位标志设置

AX 错误代码

注释：

关于调用本子功能应注意的事项，参见INT 2FH功能12H子功能00H的注释。

子功能1FH假定所有段寄存器都指向DOS内核，在DOS以外，它的应用有限。它是SUBST(替换)功能的一部分，SUBST允许把子目录标识为一个单独的逻辑器。它修改了DOS内部指针，应用时应小心。

INT 2FH	功能 12H	子功能 20H	V3
	取 DCB 号		

访问 DOS 内部服务，返回指定文件句柄的驱动器控制块号（而不是地址）

调用寄存器： AH 12H
AL 20H
BX 文件句柄

返回寄存器： 如果成功，进位标志清零
ES：DI 指向当前进程句柄表中的适当字节（该句柄的 DCB 号的地址）
如果失败，进位标志设置
AX 错误代码

注释：

子功能 20H 假定 SS 寄存器指向 DOS 的内核，以定位当前进程的句柄表，其他段寄存器没作假设。成功返回时，ES：DI 指向句柄表中字节，句柄表包括了指定句柄的 DCB 号。应用子功能 16H，该 DCB 号可以定位出 DCB 本身的位置。

通过把 SS：SP 设定于未用驱动器的逻辑驱动器表偏移量 40H 处，SS 就进入了 DOS 区域；LDT 总是在 DOS 内核里。关于 LDT 定位，请参考本书相关章节。

INT 2FH	功能 12H	子功能 21H	V3
	扩展 ASCIIZ 路径名		

访问 DOS 内部服务，把部分路径名扩展为全路径名

调用寄存器： AH 12H
AL 21H
DS：SI 指向要扩展的路径名
ES：DI 指向接受扩展串的 65 字节的缓冲区

返回寄存器： 如果成功，进位标志清零
ES：DI 未变，指向已扩展串
如果失败，进位标志设置
AX 错误代码
02H 在输入串中有非法字符

注释：

关于调用本子功能应注意的事项，参见 INT 2FH 功能 12H 子功能 00H 的注释。

子功能 21H 访问的例程与 INT 21H 功能 60H 所访问的相同。为了能访问扩展中所需的内部表格，该功能假定 SS 寄存器指向 DOS 内核。扩展的字符串包含驱动器字母和全路径名（若输入字符串包含一替代的（SUBSTed）驱动器，输出串中以真实物理驱动器代替）。

INT 2FH 功能 12H 子功能 22H V3
转换扩充错误码

访问 DOS 内部服务，确定出一个错误的扩充错误代码

调用寄存器： AH 12H
AL 22H

返回寄存器： 无

注释：

关于调用本子功能应注意的事项，参见 INT 2FH 功能 12H 子功能 00H 的注释。

子功能 22H 仅由 DOS 本身调用。该功能用 DOS 内部表把错误转换成适当的扩充错误代码、类型代码、行动代码和轨迹（Locus）代码；然后把这些值放入 DOS 内部存储单元，INT 21H 功能 59H 从这里取回它们。别的程序不应该调用它。

INT 2FH 功能 12H 子功能 24H V3
执行延迟

访问 DOS 内部服务，执行一个基于 DOS 内部值、长度可变的延迟

调用寄存器： AH 12H
AL 24H

返回寄存器： 无

注释：

关于调用本子功能应注意的事项，参见 INT 2FH 功能 12H 子功能 00H 的注释。

应用存于 DOS 内核的延迟数值，子功能 24H 执行一个延迟循环。它假定所有段寄存器指向 DOS 内核。在 DOS 以外不应调用它。

INT 2FH 功能 12H 子功能 25H V3
取 ASCIIZ 字符串长度

访问 DOS 内部服务，确定由 DS：SI 指向的 ASCIIZ 串的长度

调用寄存器： AH 12H
AL 25H
DS：SI 指向 ASCIIZ 串的指针

返回寄存器： CX 以字节表示的串长度

注释：

关于调用本子功能应注意的事项，参见 INT 2FH 功能 12H 子功能 00H 的注释。

子功能 25H 计算出 DS：SI 指向的 ASCIIZ 字符串的字节数，在 CX 寄存器中返回计数值。SS 寄存器假定为指向 DOS 内核。

INT 2FH	功能 12H	子功能 26H	V4
	打开文件		

访问 DOS 内部服务，通过 INT 21H 功能 3DH 打开一个文件

调用寄存器： AH 12H
AL 26H
CL 访问方式（同 INT 21H 功能 3DH 的 AL）
DS：DX 指向 ASCIIZ 文件说明的指针

返回寄存器： 如果成功，进位标志清零
AX 句柄
如果失败，进位标志置位
AX 错误代码
01H 功能无效
02H 文件未找到
03H 路径未找到
04H 无可用句柄
05H 拒绝访问
0CH 无效访问代码

注释：

关于调用本子功能应注意的事项，参见 INT 2FH 功能 12H 子功能 00H 的注释。

子功能 26H 把 CL 传给 AL，然后执行 INT 21H 功能 3DH（打开文件）所执行的同一例程。它假定所有的段寄存器指向 DOS 内核。因此，大多数情况下，该功能仅为 DOS 本身所用。

INT 2FH	功能 12H	子功能 27H	V4
	关闭文件		

访问 DOS 内部服务，通过 INT 21H 功能 3EH 关闭一个文件

调用寄存器： AH 12H
AL 27H
BX 文件句柄

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误代码

注释：

关于调用本子功能应注意的事项，参见 INT 2FH 功能 12H 子功能 00H 的注释。

子功能执行 INT 21H 功能 3EH 所执行的同一例程，它假设所有段寄存器指向 DOS 内

核。因此，大多数情况下，该功能仅为DOS本身所用。

INT 2FH	功能 12H	子功能 28H	V4
	定位文件指针		

访问DOS内部服务，通过INT 21H功能42H来定位一个文件指针

调用寄存器：
- AH 12H
- AL 28H
- BX 文件句柄
- CX 偏移值的高位
- DX 偏移值的低位
- BP INT 21H功能42H中AX的正常值

返回寄存器：
- 如果成功，进位标志清零
  - DX：AX 新文件指针位置
- 如果失败，进位标志置位
  - AX 错误代码
    - 01H 无效功能（文件共享）
    - 06H 句柄无效

注释：

关于调用本子功能应注意的事项，参见INT 2FH功能12H子功能00H的注释。

子功能28H把BP传给AX，然后执行INT 21H功能42H（移动文件句柄）所执行的同一例程。它假设所有段寄存器指向DOS内核。因此，大多数情况下，该功能仅为DOS本身所用。

INT 2FH	功能 12H	子功能 29H	V4
	读文件		

访问DOS内部服务，通过INT 21H功能3FH读一个文件

调用寄存器：
- AH 12H
- AL 29H
- BX 文件句柄
- CX 字节数
- DS：DX 指向缓冲区的指针

返回寄存器：
- 如果成功，进位标志清零
  - AX 已读字节数
- 如果失败，进位标志置位
  - AX 错误代码

05H 访问被拒绝
06H 句柄无效

注释：

关于调用本子功能应注意的事项，参见 INT 2FH 功能 12H 子功能 00H 的注释。

子功能 29H 执行 INT 21H 功能 3FH（读文件或设备）所执行的同一例程。它假设所有段寄存器指向 DOS 内核。因此，大多数情况下，该功能仅为 DOS 本身所用。

INT 2FH	功能 12H	子功能 2BH	V4
	IOCTL 接口		

访问 DOS 内部服务，以便于 INT 21H 功能 44H 的 IOCTL 服务接口

调用寄存器：
AH 12H
AL 2BH
BX 句柄（子功能代码 00H，01H，02H，03H，06H，07H 和 0AH）
BL 驱动器代码：0＝缺省，1＝A 等（子功能代码 04H，05H，08H 和 09H）
CX 读或写的字节数
BP IOCTL 功能中 AX 的正常值
DS：DX 指向缓冲区的指针（子功能代码 02H—05H）
DX 设备信息（子功能代码 01H）

返回寄存器：
如果成功，进位标志清零
AX 传递的字节数（子功能代码 02H—05H）
AL 状态（子功能代码 06H—07H）
00H 未准备好
FFH 准备好
AX 值（子功能代码 08H）
00H 可更换的
01H 固定的
DX 设备信息（子功能代码 00H）
如果失败，进位标志置位
AX 错误代码
01H 无效功能（文件共享）
04H 无可用句柄
05H 访问拒绝
06H 句柄无效
0DH 数据无效
0FH 驱动器无效

注释：

关于调用本子功能应注意的事项，参见 INT 2FH 功能 12H 子功能 00H 的注释。

子功能 2BH 将 BP 传给 AX，然后执行 INT 21H 功能 44H (IOCTL) 所执行的同一例程。它假设所有的寄存器指向 DOS 内核。因此，大多数情况下，该功能仅为 DOS 本身所用。

INT 2FH	功能 12H	子功能 2DH	V4
	取扩充错误代码		

访问 DOS 内部服务，获取扩充错误代码信息

调用寄存器： AH 12H
AL 2DH

返回寄存器： AX 错误代码

注释：

关于调用本子功能应注意的事项，参见 INT 2FH 功能 12H 子功能 00H 的注释。

子功能 2DH 在 AX 中返回扩充错误代码的当前值，扩充错误代码也可由 INT 21H 功能 59H 返回。它假设 SS 和 DS 寄存器指向 DOS 内核。

INT 2FH	功能 12H	子功能 2FH	V4
	存贮 DX		

访问 DOS 内部服务，把 DX 的值存于 DOS 内核

调用寄存器： AH 12H
AL 2FH
DX 要存的新值

返回寄存器： 无

注释：

关于调用本子功能应注意的事项，参见 INT 2FH 功能 12H 子功能 00H 的注释。

子功能 2FH 把 DX 中的值存于 DOS 内部一个字中，其目的还不清楚。使用该功能时应极其小心。它假设所有段寄存器指向 DOS 内核。

INT 2FH	功能 14H	子功能 00H	V3
	取 NLSFUNC 安装状态		

取 NLSFUNC.COM 安装状态，除非该程序已加载，否则，它不做任何事情

调用寄存器： AH 14H
AL 00H

返回寄存器： 如果成功，进位标志清零

AL 状态
00H 未安装，准备好安装
01H 未安装，未准备好安装

FFH 已安装

注释：

功能 14H 子功能 00H 用来取 NLSFUNC.COM 安装状态。其他子功能的应用目前还不清楚。

INT 2FH	功能 15H 取 CDROM 安装状态	V3

取 CDROM 安装状态，除非该程序已加载，否则，它不做任何事情

调用寄存器： AX 1500H 取安装状态

返回寄存器： 如果成功，进位标志清零

AL 状态

00H 未安装，准备好安装

01H 未安装，未准备好安装

FFH 已安装

注释：

用于 CDROM 接口例程，如 Microsoft 在 CDROM extensions 中提供的例程。这些讨论超出了本书的范围。

INT 2FH	功能 16H 子功能 80H MS-DOS 空调用	V5

告诉系统本调用程序为空

调用寄存器： AH 16H

AL 80H

返回寄存器： 无

注释：

如等待用户输入时，程序就可应用本中断。在此之前，调用程序应确保 INT 2FH 向量不为 0。

该中断是非阻塞性的（Non-Blocking），若系统没有其他程序要运行，该中断就会立即返回，调用程序继续运行。

INT 2FH	功能 1AH 子功能 00H 取 ANSI.SYS 安装状态	V4

返回 ANSI.SYS 的安装状态

调用寄存器： AH 1AH

AL 00H

返回寄存器： AL FFH ANSI.SYS 已加载
00H ANSI.SYS 未加载

注释：

该功能用于确定是否加载了 ANSI.SYS。该 ANSI.SYS 驱动程序提供了一组 ANSI 屏幕处理代码。在程序应用这些代码前，应当调用本功能以确定这些代码是否受支持。

INT 2FH 功能 43H 子功能 00H V5
取 XMS 驱动程序的安装状态

返回 XMS 驱动程序的安装状态

调用寄存器： AH 43H
AL 00H

返回寄存器： AL 80H XMS 驱动程序已加载
00H XMS 驱动程序未加载

注释：

在调用功能 4310H 之前一定要调用本功能，以确保已有 XMS 驱动程序。

INT 2FH 功能 43H 子功能 10H V5
取 XMS 驱动程序的入口指针地址

返回 XMS 驱动程序功能调用地址

调用寄存器： AH 43H
AL 10H

返回寄存器： ES：BX 指向功能调用的指针

注释：

只有在调用功能 43H 子功能 00H 确定已安装 XMS 驱动程序后，该功能才可调用。若 HMA 和 UMB 已由 MS-DOS 管理，程序就不应利用 XMS 驱动器进行 HMA 和 UMB 管理。

INT 2FH 功能 48H 子功能 00H V5
取 DOSKEY.COM 的安装状态

返回 DOSKEY.COM 的安装状态

调用寄存器： AH 48H
AL 00H

返回寄存器： AL 非零 DOSKEY.COM 已加载
00H DOSKEY.COM 未加载

注释：

该功能用于确定是否安装了 DOSKEY.COM。DOSKEY.COM，它是一个增强型命令行

解释器。

INT 2FH	功能 48H	子功能 10H	V5
	读命令行		

读一个最多 126 字节的行，并把它复制到指定缓冲区

调用寄存器： AH 48H
AL 10H
DS：DX 指向缓冲区的指针
字节 0，缓冲区最大长度
字节 1，返回的行长
字节 2，输入行的首字节

返回寄存器： AX 如果成功，为 0

注释：

在调用期间，DOSKEY 的宏指令和功能键起作用。若输入一个宏功能名，AX 就为 0，但一定要做第二次调用来扩展这个宏。

该行尾已加入回车（ODH），返回的行长度不包括回车字符。

键入的行被加到 DOSKEY 的历史记录中。

INT 2FH	功能 4BH	子功能 01H	V5
	建立通知链		

建立一个通知功能处理程序的链状表

调用寄存器： AH 4BH
AL 01H
ES：BX 0
CX：DX 服务功能处理程序地址

返回寄存器： ES：BX 0 不存在通知链
非 0 SWCALLBACKINFO 结构的地址
其结构格式如下：
双字 链中下一个 SWCALLBACKINFO 结构的地址
双字 通知功能处理程序地址
双字 保留
双字 SWAPIINFO 结构地址

注释：

为支持任务切换 API，用户程序须将该调用作为它的 INT 2FH 处理中的一部分。如果不这样，用户程序必须跳转到先前装入的 INT 2FH 处理程序。

若用户程序决意处理该调用，它必须首先调用先前装入的 INT 2FH 处理程序；返回时，

填写自己的SWINSTANCEITEM结构，把先前INT 2FH处理程序返回的ES：BX值置为第一个双字的内容（即下一个SWINSTANCEITEM结构的地址），并把它自己的SWINSTANCEITEM结构地址设置到ES：BX之中。这样，最近加载的用户程序就成了链中的第一个。

INT 2FH	功能 4BH	子功能 02H	V5
	检测切换器		

检测任务切换器的是否存在

调用寄存器：

AH	4BH
AL	02H
BX	0
DI	0
ES：BX	0

返回寄存器：

ES：BX	若未安装，则为0
	若已安装，则为服务功能处理程序的地址

注释：

若程序需要阻止或控制由任务切换引起的中断，在初始化过程中应调用该功能。

INT 2FH	功能 4BH	子功能 03H	V5
	给切换器分配一标识（ID）		

返回一个唯一的切换器标识符

调用寄存器：

AH	4BH
AL	03H
BX	0
ES：DI	服务功能处理程序

返回寄存器：

BX	不能分配标识符时则为0
	01H—0FH 标识符

注释：

该功能只能由任务切换器调用，而不能由用户程序调用。

一个任务切换器首先必须知道它是不是由功能4BH子功能02H装入的第一个。若是，它必须处理别的任务切换器对该功能的调用；否则，它必须调用本功能以得到一个切换ID。如果这样做时失败，它必须自行退出或取消。

任务切换器把切换ID作为它产生的会话标识符的高4位，以保证没有两个会话标识符相同。

在检测和改变已分配的切换标识符时，任务切换器必须禁止中断。在其他时间，任务切换器可以打开中断并调用MS-DOS功能。它可以修改AX和BX寄存器，但它必须保存其他寄存器。

INT 2FH	功能 4BH 子功能 04H	V5
	释放切换器标识（ID）	

释放由功能 4BH 子功能 03H 分配的切换标识符

调用寄存器： AH 4BH
AL 04H
BX 标识符
ES：DI 服务处理程序地址

返回寄存器： BX 0 释放了有效标识符
非 0 无效标识符

注释：

当任务切换器退出时，它应调用该功能来释放标识符。

INT 2FH	功能 4BH 子功能 05H	V5
	识别立即数	

识别一个用户程序保存的立即数

调用寄存器： AH 4BH
AL 05H
ES：BX 0
CX，DX 服务功能处理程序

返回寄存器： ES：BX 0 无立即数链
非 0 SWSTARTUPINFO 结构地址
其结构格式如下：
字 版本号
双字 先前处理程序的 SWSTARTUPINFO 结构的地址
双字 忽略
双字 忽略
双字 SWINSTANCEITEM 结构的地址

注释：

为支持任务切换 API，用户程序须将该调用作为它的 INT 2FH 处理中的一部分。如果不这样，用户程序必须跳转到先前装入的 INT 2FH 处理程序。

若用户程序决意处理该调用，它必须首先调用先前装入的 INT 2FH 处理程序；返回时，填写自己的 SWCALLBACKINFO 结构，把先前 INT 2FH 处理程序返回的 ES：BX 值置为第一个双字的内容（即下一个 SWCALLBACKINFO 结构的地址），并把它自己的 SWCALLBACKINFO 结构地址设置到 ES：BX 之中。这样，最近加载的用户程序就成了链中的第一个。

INT 2FH	功能 ADH 子功能 80H 取 KEYB.COM 的版本号	V3.3

返回 KEYB.COM 的版本号

调用寄存器： AH ADH
AL 80H

返回寄存器： BH 主版本号
BL 次版本号

注释：

若 KEYB.COM 未加载，版本号为 0。

INT 2FH	功能 ADH 子功能 81H 置 KEYB.COM 活动代码页	V3.3

设置 KEYB.COM 活动代码页

调用寄存器： AH ADH
AL 81H
BX 代码页标识

返回寄存器： 如果成功，进位标志清零
如果出错，进位标志置位
AX 0001H

注释：

本功能用于选择键盘驱动程序的代码页。

INT 2FH	功能 ADH 子功能 82H 置 KEYB.COM 国别标志	V3.3

设置 KEYB.COM 的国别标志

调用寄存器： AH ADH
AL 82H
BL 国别标志
00H 本国（美国）
FFH 外国

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位

注释：

本功能为国别标志在 USA（美国）和 non-USA（非美国）之间作出选择。

INT 2FH	功能 ADH 子功能 83H 取 KEYB.COM 国别标志	V3.3

读取 KEYB.COM 的国别标志

调用寄存器： AH ADH
AL 83H

返回寄存器： BL 国别标志

注释：

该功能用于获取国别标志。值 00H 为美国，FFH 为非美国。

INT 2FH	功能 B0H 子功能 00H 取 GRAFTABL.COM 安装状态	V3.3

返回 GRAFTABL.COM 的安装状态

调用寄存器： AH B0H
AL 00H

返回寄存器： AL FFH GRAFTABL.COM 已加载
00H GRAFTABL.COM 未加载

注释：

该功能确定是否安装了 GRAFTABL.COM。GRAFTABL.COM 允许 MS-DOS 在图形方式下显示在 80H 到 FFH 之间的字符。

INT 2FH	功能 B7H 子功能 00H 检查 APPEND 是否安装	V3.3

返回在 DOS 级是否安装了 APPEND

调用寄存器： AH B7H
AL 00H

返回寄存器： AH 若已安装 APPEND ≠0（非零）

注释：

通过检查返回的 AH 值，可确定是否已安装了 APPEND。通过 APPEND 命令可把 APPEND 安装在 DOS 级。在过去的几年中，网络标准化工作很少，不是所有的网络都能正确响应本中断。

INT 2FH	功能 B7H 子功能 02H 读取 APPEND 的版本号	V4

确定安装的是哪一版本的 APPEND

调用寄存器： AH B7H
AL 02H
返回寄存器： 如果成功，进位标志清零
AX 若安装的是 V4 APPEND，则置为 FFFFH；否则，不是 MS-DOS 的 V4 APPEND
如果失败，进位标志置位
AX 错误代码

注释：

该功能用于判定所有加到 APPEND V4 的特性是否可用。无论什么版本的 APPEND 时，子功能 00H 都返回 "installed"；本功能判定版本是否是 V4。

INT 2FH	功能 B7H 子功能 04H	V4
	取 APPEND 路径指针	

若 APPEND 已安装，返回的指针指向当前活动的 APPEND 路径

调用寄存器： AH B7H
AL 04H
返回寄存器： 如果成功，进位标志清零
ES：DI 指向活动的 APPEND 路径
如果失败，进位标志置位
AX 错误代码

注释：

该功能用于确定当前活动的 APPEND 路径的地址。

INT 2FH	功能 B7H 子功能 06H	V4
	取 APPEND 功能状态	

返回当前 APPEND 功能的位映象（若 APPEND 已安装）

调用寄存器： AH B7H
AL 06H
返回寄存器： 如果成功，进位标志清零
BX APPEND 状态（位映象）

位		
FEDCBA98	76543210	含义
.....	0	APPEND 禁止使用
.....	1	APPEND 可用
...xxxxx	xxxxxxx.	未用(0)
..0.....	.....	/PATH 不活动

..1..... ..... /PATH 活动
.0..... ..... /E 开关不活动
.1..... ..... /E 开关活动
0..... ..... /X 开关不活动
1..... ..... /X 开关活动

如果失败，进位标志置位
AX 错误代码

注释：

该功能用于读取 APPEND 功能当前状态。

INT 2FH	功能 B7H 子功能 07H 置 APPEND 的功能状态	V4

通过位映象代码改变 APPEND 的功能。

调用寄存器： AH B7H
AL 07H
BX APPEND 状态（位映象）

位		
FEDCBA98	76543210	意义
.....	0	关闭 APPEND
.....	1	启用 APPEND
...xxxxx	xxxxxxx.	未用(0)
..0.....	.....	/PATH 不活动
..1.....	.....	/PATH 活动
.0.....	.....	/E 开关不活动
.1.....	.....	/E 开关活动
0.....	.....	/X 开关不活动
1.....	.....	/X 开关活动

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误代码

注释：

通过传入一个位映象描述的希望状态，该功能设置 APPEND 功能状态。

INT 2FH	功能 B7H 子功能 11H 设置创建或打开文件时返回文件名的状态	V4

本中断仅修改下次 DOS 访问时 APPEND 的动作

调用寄存器： AH B7H
AL 11H

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误代码

注释：

该功能可使 APPEND 修改 INT 21H 功能 3DH，43H 和 6CH 的动作。该功能执行过后，下次调用功能 3DH，43H 和 6CH 中的任何一个，返回的就是全文件名，返回的文件名与传入该功能的文件名在同一地方。必须确保该缓冲区足够长(67 字节)，以接受任何可能的返回值。

在返回一个扩展的文件名之后，该功能所设置的标志被清除，INT 21H 以上功能的操作返回到正常状态。因此，每调用一次该子功能，仅对一次 INT 21H 功能 3DH，43H 或 6CH 调用有效。

• MEMO •

第二章　如何调用 DOS 功能

本章介绍如何在程序中使用 DOS 功能调用，并包括了汇编、C、Pascal 以及 BASIC 语言调用 DOS 的例子。本章内容同样适用于对 BIOS 功能的调用。

2.1　一般的 DOS 功能调用

DOS 功能调用的一般步骤为：

1. 装入入口参数到规定的 CPU 寄存器中；
2. 如有必要，装入功能号到 AH 中；如有必要，装入子功能号到 AL 中；
3. 调用 DOS 相应功能的中断；
4. 如有必要，取出返回结果或出错信息。

一个简单的 DOS 调用过程如下。这个程序片段在屏幕当前光标位置显示一个字母 X。

```
mov     dl,'X'      ; 将要显示的字母 X 送入规定的寄存器 DL 中
mov     ah, 2       ; 将 DOS 功能调用的主功能号送 AH，无子功能号
int     21h         ; 执行 DOS 功能调用，显示一字母 X
                    ; 无返回结果，无状态返回
```

比较而言，在汇编语言中调用 DOS 功能显得既自然又简单。只要先设定好规定的入口参数，然后调用 DOS 中断功能，就可以得到预期的结果。在高级语言中情况就显得复杂些。不同的高级语言调用 DOS 的方法不同，同一种高级语言对 DOS 的调用方法也有所不同，这取决于所用语言的编译器或解释系统的厂家来源及版本。例如，同样是 BASIC 语言，Quick BASIC 的调用语法为：

```
CALL    INTERRUPT (interrupt_number, registers_in, registers_out)
```

而实现相同功能的 Turbo BASIC 调用为：

```
CALL    INTERRUPT (interrupt_number)
```

这种编程语言上的差异请参考有关的技术资料。本节给出的实例尽可能使其在 Microsoft 及 Borland 这两大系列的编程语言系统下顺利通过。另外，这两种语言系列都提供有丰富的库函数，DOS 的大部分功能都有相应的函数可用。示范程序旨在展示编程语言如何实现对 DOS 功能的调用。

2.1.1　使用汇编语言调用 DOS 功能

以下的汇编语言示范程序（程序 2.1）显示出今天是星期几。

用 Turbo Assember V3.00汇编这个程序的方法是：

```
TASM week
TLINK week
```

用 Microsoft Assember V6.00汇编这个程序的方法是：

```
ML week.asm
```

【程序2.1】 汇编语言示范程序，显示今天是星期几

```
;
;     WEEK.ASM – DOS Function calling sample / week display
;
;     汇编语言调用 DOS 功能的示范程序 / 本程序显示出今天是星期几
;
code        segment para public' sample'
              assume cs: code, ds: data
week_demo     proc    near
              mov     ax, seg data
              mov     ds, ax              ; 数据区的段地址-->DS
              mov     dx, offset today_msg ; 字符串的偏移地址->DX
              mov     ah, 09h             ; 9号功能，显示字符串
              int     21h                 ; 调用 DOS 功能

              mov     ah, 2ah             ; 2ah 号功能，取系统日期
              int     21h                 ; 调用 DOS 功能 AL 返回星期的代码

              xor     ah, ah
              shl     ax, 1               ; 乘以2
              mov     si, offset week_msg
              add     si, ax              ; 起始地址加上偏移

              mov     dl, [si]            ; 前半个汉字
              mov     ah, 02h             ; 2号功能显示一个字符
              int     21h                 ; 调用 DOS 功能
              mov     dl, [si+1]          ; 后半个汉字
              mov     ah, 02h             ; 2号功能显示一个字符
              int     21h                 ; 调用 DOS 功能

              mov     dx, offset lfcr     ; 回车换行
              mov     ah, 09h
              int     21h

              mov     ah, 4ch             ; 4ch 号功能，程序退出
              xor     al, al              ; 返回码为0
              int     21h                 ; DOS 功能调用，程序结束返回 DOS
week_demo     endp

code          ends

data          segment para public' sample'
today_msg     db      0dh, 0ah,' 今天是星期$'
week_msg      db      '日一二三四五六'
lfcr          db      0dh, 0ah,' $'
data          ends

              end     week_demo
```

注：程序中用到4个 DOS 功能调用，入口和返回参数简介如下：

1. int 21h，功能02h，显示一个字符
   入口：AH=02h　　出口：无
   DL=ASCII 码
2. int 21h，功能09h，显示字符串
   入口：AH=09h　　出口：无
   DS:DX 字符串指针（字符串以 $ 结束）
3. int 21h，功能2ah，取系统的日期和时间
   入口：AH=2ah　　出口：AL=星期
   CX=年
   DH=月
   DL=日
4. int 21h，功能4ch，程序终止

入口：AH＝4ch　　出口：无
　　AL＝返回码

2.1.2　使用 C 语言调用 DOS 功能

高级语言调用 DOS 的一个关键问题是入口寄存器参数的传递。在 Microsoft C 和 Borland C++中，都定义有用于传递寄存器入口参数的数据结构和调用函数。这些内容包含在头文件 dos.h 中。

```
/* 字寄存器的定义 */
struct WORDREGS {
      unsigned int ax, bx, cx, dx, si, di, cflag;
      };
/* 字节寄存器的定义 */
struct BYTEREGS {
      unsigned char al, ah, bl, bh, cl, ch, dl, dh;
      };
/* 通用寄存器的定义，字寄存器与字节寄存器的联合 */
union REGS {
      struct WORDREGS x;
      struct BYTEREGS h;
      };
/* 段寄存器的定义 */
struct SREGS {
      unsigned int es, cs, ss, ds;
      };
```

对于一般的中断调用，DOS.H 中包含以下的函数原型：

```
int int86 (int intno, union REGS *inregs,
                      union REGS *outregs);
int int86x (int intno, union REGS *inregs,
                       union REGS *outregs,
                       struct SREGS *segregs);
```

对于 DOS 中断调用，DOS.H 中包含以下的函数原型：

```
int intdos (union REGS *inregs,
            union REGS *outregs);
int intdosx (union REGS *inregs,
             union REGS *outregs,
             struct SREGS *segregs);
```

以下首先给出一个在 C 语言中调用 BIOS 功能的示范程序 prnok.c（程序2.2），该程序显示打印机的联机情况。事实上，C 语言对 BIOS 和 DOS 的调用没有区别，可以认为对 DOS 的调用只是对 BIOS 调用的一个特例。在 prnok.c 中 int86（）函数的第一个参数 PRN_INT 为 0x17，指出调用 BIOS 的打印机服务功能，同样的，如果第一个参数为0x21则指明是对 DOS 功能的调用，intdos（）函数与固定第一个参数为0x21的 int86（）函数没有区别，只是前者稍微方便些。另一个对 DOS 功能调用的函数 intdosx（）可以在调用入口传送段寄存器值，用以传递远指针参数。第二个示范程序 chmod.c（程序2.3）用到 intdosx（）以传送文件名指针 DS:DX。

在这一小节中展示的所有 C 程序，如无特别说明，用以下两种编译器的任意一个编译均可通过。

Borland C++ Version 3.00
Microsoft Quick C Version 2.01

编译连接的方法分别是：

BCC <程序名，如：prnok.c>
QCL <程序名，如：prnok.c>

【程序2.2】 C 语言示范程序，显示打印机的联机情况

```
/*
    prnok.c — DOS Function calling sample / display printer status
      Copyright (c) 1992 by Mr. Wang
    C 语言调用 DOS 功能的示范程序 / 本程序显示打印机的工作状态
*/
#include <conio.h>
#include <dos.h>
#define PRN_INT 0x17     /* BIOS 打印服务中断号          */
#define STAT_RQ 0x02     /* 取打印机工作状态的功能号     */
int prnok (void)
{
        union REGS regs;
                                /* 装入主功能号到 AH 中              */
        regs.h.ah = STAT_RQ;    /* AH=02 取打印机工作状态            */
                                /* 装入入口参数到规定的 CPU 寄存器中 */
        regs.x.dx = 0;          /* DX=0  指定第一个并行口 LPT1       */
        int86 (PRN_INT, &regs, &regs);      /* 调用 BIOS 相应功能的中断程序 */
        return ( ( (regs.h.ah & 0x80) == 0x80) ? 1 : 0); /* 取出结果 */
}
void main ()
{
        if (prnok ())
            cputs (" \n    打印机已就绪，可以打印\n");
        else
            cputs (" \n    打印机未准备好，请检查\n");
}
```

注：程序中用到的 BIOS 功能调用的入口和返回参数简介如下：
int 17h，功能0x02，取打印机工作状态
入口：AH=02h　　出口：AH=打印机接口状态，各位的意义如下
　　DX=打印机号

```
7 6 5 4 3 2 1 0
x . . . . . .    Printer not busy (0=busy)
. x . . . . .    Acknowledgement from printer
. . x . . . .    Out of paper
. . . x . . .    Printer selected
. . . . x . .    I/O error
. . . . . 0 0 .  Not used
. . . . . . . x  Time-out error
```

【程序2.3】 C 语言示范程序，改变文件的属性

```
/*
    chmod.c — DOS Function calling sample / change file attribute
              Copyright (c) 1992 by Mr. Wang

    C 语言调用 DOS 功能的示范程序 / 本程序用来改变文件的属性
```

```
*/
#include <conio.h>
#include <stdio.h>
#include <ctype.h>
#include <process.h>
#include <dos.h>

#define GS_FATTR  0x43
#define GET_FATTR 0x00
#define SET_FATTR 0x01

#define ARCHIVE_BIT  0x20
#define SYSTEM_BIT   0x04
#define HIDDEN_BIT   0x02
#define RDONLY_BIT   0x01

typedef enum {clr, set} clrorset;

void disphelp ()              /* 显示帮助信息 */
{
    cputs (" 语法：chmod <文件命>  /xy          \n\r"
           " 说明：x = A（后备）或H（隐含）或  \n\r"
           "           R（只读）或S（系统）    \n\r"
           "       y = +（设置）或 -（清除）    \n\r");
}

#define CLEAR 0
#define isclear (x, y) ( (x&y) ==CLEAR)      /* Parse around x, y? */
#define putstat (x, y) cputs (isclear (x, y) ? " 清除 "," 设置").

void showattr (int attr)      /* 显示文件属性 */
{
    cputs (" 后备  系统  隐含  只读\n\r");
    putstat (attr, ARCHIVE_BIT);
    putstat (attr, SYSTEM_BIT);
    putstat (attr, HIDDEN_BIT);
    putstat (attr, RDONLY_BIT);
    cputs (" \n\r");
}
                    /* 命令行语法分析 */
int parsearg (char *thearg, clrorset *action, char *selection)
{
    if ( * (thearg) == '/') {
       switch ( * (thearg+2)) {
           case '+':
                *action = set; break;
           case '-':
                *action = clr; break;
           default:
               puts (" 用 '+' 设置，或用 '-' 清除\n\r");
               return (0);
       }
       *selection = toupper ( * (thearg+1));
       return (1);
    } else {
       disphelp ();
       return (0);
    }
}

void main (int argc, char *argv [])        /* 主程序开始 */
{
    extern char *sys_errlist [];
    extern int errno;
```

```
union REGS regs;
struct SREGS sregs;

clrorset action;
char selection;

unsigned attrib, setting;
int goahead;

if (argc == 3) goahead = parsearg (argv [2], &action, &selection);
else goahead = 0;

if (!goahead)
   {
   if (argc==3)
        {
        cputs (" 语法错:");
        cputs (argv [2]);
        }
   else
        disphelp ();
   exit (1);
}

switch (selection) {
    case 'A': setting = ARCHIVE_BIT; break;
    case 'H': setting = HIDDEN_BIT; break;
    case 'R': setting = RDONLY_BIT; break;
    case 'S': setting = SYSTEM_BIT; break;
    default:
       cputs (" 输入有错:"); cputs (argv [2]); cputs (" \n\r");
       exit (1);
}

regs.h.ah = GS_FATTR;                  /* 装入功能号到 AH     */
regs.h.al = GET_FATTR;                 /* 装入子功能号到 AL   */
regs.x.dx = (unsigned) argv [1];       /* 装入文件名参数串的偏移   */
segread (&sregs);                      /* 装入文件名参数串的段地址 */

intdosx (&regs, &regs, &sregs);        /* 调用 DOS 功能，取文件的当前属性字 */

if (!regs.x.cflag) {                   /* 测试 CF 标志，调用是否成功       */
    attrib = regs.x.cx;                /* 装入返回结果                     */
    cputs (" ------- 原来属性 -------\n\r");
    showattr (attrib);
    if (action==clr) {
       setting = ( (~setting) &attrib);
      else {
       setting = (setting | attrib);
    }

    regs.h.ah = GS_FATTR;              /* 装入功能号到 AH */
    regs.h.al = SET_FATTR;             /* 装入子功能号到 AL */
    regs.x.cx = setting;               /* 装入新的文件属性 */
    regs.x.dx = (unsigned) argv [1]    ; /* 装入文件名参数串的偏移 */
    segread (&sregs);                  /* 装入文件名参数串的段地址 */

    intdosx (&regs, &regs, &sregs);    /* 调用 DOS 功能，设定文件属性字 */

    attrib = regs.x.cx;                /* 装入设定后的属性          */
    cputs (" ------- 改后属性 -------\n\r");

    showattr (attrib);

} else {                   /* That is, if carry is not set */
    char *msg;
    cputs (" DOS 功能0x43调用失败:");
    switch (regs. x. ax) {
       case 1: msg = " 功能号有错\n"; break;
```

```
            case 2: msg = " 文件名有错\n"; break;
            case 3: msg = " 路径名有错\n"; break;
            case 5: msg = " 属性不能改变\n"; break;
            default: msg = " 错误类型未知\n";
        }
        cputs (msg);
    }
}                              /* 主程序结束 */
```

注：程序中用到的 DOS 功能调用的入口和返回参数简介如下：

int 21h，功能0x43，取或置文件属性

入口：AH=0x43　　出口：CF=0成功，CX=属性字节

AL=0　取文件属性　　CF=1 失败，AX=错误代码

1　置文件属性　　01　功能号有错

DS:DX　路径文件名指针　　02　文件名有错

(ASCIIZ 串,以00结束)　　03　路径名有错

05　属性不能改变

2.1.3　使用 Pascal 语言调用 DOS 功能

Turbo Pascal 中为寄存器定义的数据结构在 DOS 单元中。

其定义为：

```
Type
      { Registers record used by Intr and MsDos }
     Case Integer of
        0: (AX, BX, CX, DX, BP, SI, DI, DS, ES, Flags: Word);
        1: (AL, AH, BL, BH, CL, CH, DL, DH : Byte);
     End;
```

Turbo Pascal 中为访问 BIOS 和 DOS 中断功能调用定义过程如下：

```
Intr (IntNo : Byte; vars Regs : Registers);
MsDos (vars Rega : Registers);
```

其中 Intr 可以访问所有的软中断功能，MsDos 访问 DOS 功能调用，相当于 IntNo 为21H 的 Intr 过程。类似 C 语言中 int86 () 和 intdos () 函数及其关系。

以下首先给出一个在 Turbo Pascal 语言中调用 BIOS 功能的示范程序 PrnDemo. PAS(程序2.4)，该程序显示打印机的联机情况。第二个示范程序 FileDT. PAS (程序2.5) 要求输入一个文件名，然后调用 DOS 功能取得此文件的日期和时间，显示在屏幕上。

【程序2.4】　Turbo Pascal 示范程序，显示打印机的联机情况

```
{
   prndemo.  pas - DOS Function calling sample / display printer status
   Turbo Pascal 语言调用 DOS 功能的示范程序 / 本程序显示打印机的工作状态
}
Program PrinterDemo;
Uses DOS;                          { 使用 DOS 单元 }

Function PrinterOnline : Boolean;
Const
  PrnStatusInt   : Byte = $17;     { 打印服务中断              }
  StatusRequest: Byte = $02;       { 取打印机工作状态的功能号   }
  PrinterNum     : Word = 0;       { 0对应 LPT1，1 对应 LPT2，等等 }
Var
  Regs : Registers;                { 寄存器数据类型定义在 Dos 单元中 }
Begin
                                   { 装入主功能号到 AH 中      }
```

```
  Regs. Ah := StatusRequest;          { AH=02取打印机工作状态  }
                                      { 装入入口参数到规定的 CPU 寄存器中  }
  Regs. DX := PrinterNum;             { DX=0 LPT1              }
  Intr (PrnStatusInt, Regs);          { 调用 BIOS 相应功能的中断程序 }
  PrinterOnline := (Regs. AH and $80) = $80;    { 取出结果 }
end;

Begin       { Main Program }
  If PrinterOnline Then
      WriteLn ('      打印机已就绪，可以打印')
    Else WriteLn ('      打印机未准备好，请检查');
End.
```

注：程序中用到的 BIOS 功能调用的入口和返回参数简介如下：

int 17h，功能0x02，取打印机工作状态

入口：AH=02h　　出口：AH=打印机接口状态，各位的意义如下

DX=打印机号

```
7 6 5 4 3 2 1 0
x . . . . . .   Printer not busy (0=busy)
. x . . . . . .  Acknowledgement from printer
. . x . . . . .  Out of paper
. . . x . . . .  Printer selected
. . . . x . . .  I/O error
. . . . . 0 0 .  Not used
. . . . . . . x  Time-out error
```

【程序2.5】 Turbo Pascal 示范程序，显示给定文件的最后修改日期和时间

```
{
   FileDT. PAS - DOS Function calling sample / file date and time display
   Turbo Pascal 调用 DOS 功能的示范程序 / 本程序显示文件的日期和时间
}
Program FileDateAndTime;
   Uses Dos;                  { 使用 DOS 单元 }
   Type
      PathNameType = String [64];
      String5 = String [5];
      { Max DOS path is 63; add 1 for the null. }
   Var
      Pathname : PathNameType;
      DateWord, TimeWord : Word;
      Hours, Mins, Secs, Year, Month, Day : Integer;
      ch : Char;
Procedure GetDateAndTime (Pathname : PathStr;
                          Var DateWord, TimeWord : Word);
   Const
      DosGetDateAndTime : Byte = $57;
      DosCloseFile : Byte = $3E;
   Var
      Regs : Registers;
      Handle : Word;
   Function CarryClear (Regs : Registers) : Boolean;
   Begin CarryClear := ( (Regs. Flags and 1) = 0) End;
   Function GetFileHandle (Pathname : PathStr) : Word;
      Const GetHandle : Byte = $3D; ReadAccess : Byte = 0;
      Var PathSeg, PathOfs : Word;
   Begin
      Pathname := Pathname + Chr (0);
      PathSeg  := Seg (Pathname [1]); PathOfs:=Ofs (Pathname [1]);
      Regs. Ah:= GetHandle;      { 装入主功能号到 AH 中      }
```

```
        Regs. AL:= ReadAccess;      {装入模式号到 AL 中          }
        Regs. DS:= PathSeg;         { 装入入口参数:              }
        Regs. Dx:= PathOfs;         { DS:DX 指向文件名字符串     }
        MsDos (Regs);               {调用 DOS 功能的中断程序     }
        If CarryClear (Regs)            { 测试操作成功否?        }
        Then GetFileHandle:= Regs. AX   { 取出结果               }
        Else Begin WriteLn (' 文件功能失败!'); Exit End;
    End;
Begin { procedure GetDateAndTime }
    Handle:= GetFileHandle (Pathname);  { 以句柄方式打开文件     }
    Regs. AH:= DosGetDateAndTime;       { 装入主功能号到 AH 中   }
    Regs. AL:= 0;                       { 装入子功能号到 AL 中   }
    Regs. BX:= Handle;                  { 装入入口参数, BX=句柄  }
    MsDos (Regs);                       { 调用 DOS 功能的中断程序 }
    If CarryClear (Regs)                { 测试操作成功否?        }
    Then Begin
        DateWord:= Regs. DX;            { 取出结果               }
        TimeWord:= Regs. CX
        End
    Else Begin
        DateWord:= 0;
        TimeWord:= 0
        End;
    Regs. AH:= DosCloseFile;            { 调用 DOS 功能关闭文件  }
    Regs. BX:= Handle;
    MsDos (Regs);
    If NOT CarryClear (Regs) Then
        WriteLn (' 警告: GetDateAndTime 没有关闭文件', Pathname);
End;    { procedure GetDateAndTime }
Begin { Main Program }
  Write (' 文件名?>'); ReadLn (Pathname);          { 读取文件名串     }
  GetDateAndTime (Pathname, DateWord, timeWord);   { 取文件日期和时间 }
  If (Dateword<>0) Then
  Begin                                            { 展开日期时间结果 }
      Year  := ( (DateWord AND $FE00) SHR 9) + 1980;
      Month := (DateWord AND $01E0) SHR 5;
      Day   := (DateWord AND $001F);
      Hours := (TimeWord AND $F800) SHR 11;
      Mins  := (TimeWord AND $07E0) SHR 5;
      Secs  := (TimeWord AND $001F) SHL 1;
      WriteLn (文件时间:', Hours: 2,' 点', Mins: 02,' 分', Secs: 2,' 秒');
      WriteLn (' 文件日期:', Year: 4,' 年', Month: 2,' 月', Day: 02,' 日');
  End
  Else WriteLn (' GetDateANDTime 运行失败!');
End.
```

注：程序中用到的 DOS 功能调用的入口和返回参数简介如下：

[1] int 21h，功能 $3D，打开文件

入口：AH=$3D　　出口：CF=0成功，AX=句柄

AL=存取模式　　CF=1失败，AX=错误代码

DS:DX 指向文件名字符串（ASCIIZ 串）

[2] int 21h，功能 $3E，关闭文件

入口：AH=$3E　　出口：CF=0成功

BX=句柄　　CF=1失败，AX=错误代码

[3] int 21h，功能 $57，子功能00，取文件日期和时间

入口：AH=$57　　出口：CF=0成功，CX=时间，DX=日期

AL=$00　　CF=1失败，AX=错误代码

BX=句柄

2.1.4 使用 Quick BASIC 语言调用 DOS 功能

Quick BASIC 中为存放寄存器的内容定义的数据结构在 QB.BI 文件中。其定义为：

```
'
' Define the type needed for INTERUPT
'
        TYPE RegType
            ax      AS INTEGER
            bx      AS INTEGER
            cx      AS INTEGER
            dx      AS INTEGER
            bp      AS INTEGER
            si      AS INTEGER
            di      AS INTEGER
            flags   AS INTEGER
        END TYPE
'
' Define the type needed for INTERUPTX
'
        TYPE RegTypeX
            ax      AS INTEGER
            bx      AS INTEGER
            cx      AS INTEGER
            dx      AS INTEGER
            bp      AS INTEGER
            si      AS INTEGER
            di      AS INTEGER
            flags   AS INTEGER
            ds      AS INTEGER
            es      AS INTEGER
          END TYPE
```

Quick BASIC 中为访问 BIOS 和 DOS 中断功能调用定义过程如下：

```
'
' Generate a software interrupt, loading all but the segment registers
'
DECLARE SUB INTERRUPT (intnum AS INTEGER, inreg AS RegType, outreg AS RegType)
'
' Generate a software interrupt, loading all registers
'
DECLARE SUB INTERRUPTX (intnum AS INTEGER,
                        inreg AS RegTypeX, outreg AS RegTypeX)
```

其中 INTERRUPT 可以访问所有的软中断功能，但不能传送段寄存器；INTERRUPTX 可以访问所有的软中断功能，还可以传送段寄存器。

以下首先给出一个在 Quick BASIC 语言中调用 BIOS 功能的示范程序 PRNOKQB.BAS（程序2.6），该程序显示打印机的联机情况。第二个示范程序 FILEFIND.BAS（程序2.7）要求输入一个可以带通配符的文件名，然后调用 DOS 功能搜索并显示出所有匹配的文件名。

【程序2.6】　Quick BASIC 示范程序，显示打印机的联机情况

```
'
'     PRNOKQB.BAS - DOS Function calling sample / display printer status
'
'     QuickBASIC 调用 DOS 功能的示范程序 / 本程序显示打印机的联机状态
'
'       BC PRNOKQB;
'       LINK PRNOKQB;
'
' PRNOKQB.BAS
REM $INCLUDE:' QB.BI'                          ' include header file
CONST PRN.Status.rq% = &H200                   ' 2in AH
CONST BIOS.PRN.INT% = &H17                     ' BIOS int 17h
DIM InRegs AS RegType, OutRegs AS RegType      ' define register var

InRegs.AX = PRN.Ststus.rq%                     ' set sub function no
CALL INTERRUPT (BIOS.PRN.INT%, InRegs, OutRegs)' call BIOS function
IF ( (OutRegs.AX AND &H8000) = &H8000) THEN    ' get the result
   PRINT" Printer OK"
ELSE
   PRINT" Please check the printer"
END IF
END            ' PROGRAM
```

【程序2.7】　Quick BASIC 示范程序，搜索匹配的文件

```
'
'     FILEFIND.BAS - DOS Function calling sample / find match files
'
'     QuickBASIC 调用 DOS 功能的示范程序 / 本程序显示所有匹配的文件名
'
'       BC FILEFIND;
'       LINK FILEFIND;
'
REM $INCLUDE:' QB.BI'                          ' load header file
DECLARE SUB SetDTA (TheDTA$)                   ' declare sub routine
DECLARE SUB FindFirst (FileSpec$, FileName$)
DECLARE SUB FindNext (FileName$)
DECLARE SUB BuildName (TheName$)
DTA$ = SPACE$ (43)
INPUT" Filespec?>", FileSpec$                  ' get filespace
CALL SetDTA (DTA$)
' Get the first matching file name
CALL FindFirst (FileSpec$, FileName$)          ' find first match file name
IF FileName$ <>"" THEN
   PRINT" First match:"; FileName$
   DO
      CALL FindNext (FileName$)                ' find next match file name
      IF FileName$ <>"" THEN
         PRINT" Next match:"; FileName$
      END IF
   LOOP UNTIL FileName$ = ""
ELSE
   PRINT" No files match"; FileSpec$
END IF
END            ' PROGRAM
```

```
SUB BuildName (TheName $)                          ' sub—routine get file name
SHARED DTA $                                       '    from DTA
    EndOfStr% = INSTR (31, DTA $, CHR $ (0))
    TheName $ = MID $ (DTA $, 31, EndOfStr% - 31)
END SUB

SUB FindFirst (FileSpec $, FileName $)             ' make ASCIIZ
DIM InRegs AS RegTypeX, OutRegs AS RegTypeX
InRegs. AX = &H4E00                                ' find firstmmatching file
InRegs. DX = SADD (FileSpec $)                     ' offset of FileSpec $
InRegs. DS = VARSEG (FileSpec $)                   ' seg od FileSpec $
InRegs. CX = 0

CALL INTERRUPTX (&H21, InRegs, OutRegs)            ' call DOS function
'    Got a match?Yes, if CARRY FLAG (bit 0 of FLAGS) is clear
IF (OutRegs. FLAGS AND 1) = 0 THEN                 ' get the result
    CALL BuildName (FileName $);
ELSE
    FileName $ = ""
END IF
END SUB

SUB FindNext (FileName $)                          ' sub—routine find the next
    DIM InRegs AS RegTypeX, OutRegs AS RegTypeX    '      match file name
    InRegs. AX = &H4F00                            ' DOS function 4FH find next
    CALL INTERRUPTX (&H21, InRegs, OutRegs)        ' call DOS function
    IF (OutRegs. FLAGS AND 1) = 0 THEN
        CALL BuildName (FileName $);
    ELSE
        FileName $ = ""
    END IF
END SUB

SUB SetDTA (DTA $)
DIM InRegs AS RegTypeX, OutRegs AS RegTypeX
' Set the Disk Transfer Area address
InRegs. DX = SADD (DTA $)                          ' offset of DTA
InRegs. DS = VARSEG (DTA $)                        ' segment of DTA
InRegs. AX = &H1A00                                ' DOS function for setting DTA addr
CALL INTERRUPTX (&H21, InRegs, OutRegs)            ' call DOS function
' no return value for functin &H1A
END SUB
```

2.2 未写入文档的 DOS 功能调用

在第一章中，我们已尽力描述了 INT 21H 的所有功能，其中包括那些被正式标识为保留而在大多数参考手册中未予说明的功能。

下面关于保留功能的描述，是通过非正式途径获得的。在许多情况下，需要对 DOS 代码进行反汇编，然后分析出它可能完成的功能。有些描述或许还不太准确，但我们尽了最大的努力，把已有的信息全部提供给读者。

这些功能可分成四组，在以下将分别予以描述。在分析了每组应完成的功能后，给出了用某些“有用”功能写出来的一组程序，这些程序提供了在其他情况下无法获得的信息。

设计者保留这些功能的一个重要目的，是为了能在更新版本时根据需要改变它们。对那些未写入文档的功能来说，这种情况很可能发生。这是使用这些功能编程时所必须承担的风险。

2.2.1　四类保留的DOS功能

这四类保留的DOS功能是：

- 有用的功能，比如SetPID和GetDPB；
- 没有真正使用的功能，它们执行空操作后返回；
- 衔接未来的功能（Hooks For Future），它们可能被实现，也可能不被实现；
- 那些本质属性还没有确定的功能，主要指DOS V4中新增加的功能。

下面，让我们分别看看这些功能。注意，功能号为59H和大于59H的功能在V3以前未出现；功能号为69H和大于69H的功能在V4以前未出现；在59H与68H之间的功能是随着V3过渡到V4时逐渐加上去的。

一、有用的功能

在DOS V1中，唯一未写入文档的功能是1FH(取缺省磁盘参数块)。此功能只是在IBM的DOS版本中未写入文档。在最初的SCP（Seattle Computer Products）86-DOS中不存在任何未写入文档的功能。此功能与CP/M 2.2中的相同。

随着V2的出现，又有七种功能出现了这种情况。它们分别是：功能32H（取指定驱动器参数块）、功能34H（取InDOS标志地址）、功能50H（SetPID）、功能51H（GetPID）、功能52H（GetCVT）、功能53H（GetDPB）和功能55H（MakePSP）。其中大多数，在DOS的一些实用程序（比如PRINT.COM，FORMAT.COM等）中使用过，用以获取某些必要的信息。

功能5DH（CritErr）、功能60H（ExpandPath）和功能64H（Set Country Code）在V3以前尚未出现。这些功能似乎用在存取DOS内部数据的网络软件中。

最后，还有一个功能，功能6AH（CommitFile）是在V4中出现的。设置此功能的目的还不清楚，它同有文档说明的功能68H执行一样的代码。

二、没有真正使用的功能

五个保留功能（18H，1DH，1EH，20H和61H）都执行空操作。根据设计者Tim Paterson所称，前四个功能由原SCP操作系统提供，用以与CP/M相兼容，但在MS-DOS中没有任何意义。第五个功能直到V3才出现，也不知为何设置。这五个功能都执行一样的过程：仅将AL置零，然后返回。

三、衔接未来的功能

这一类有两个保留功能（37H和63H）。前一个在V2的某些OEM版本中有文档，它使DOS更象UNIX那样运行；但是在V3中，这一功能被大大削弱了，而剩下的又没有被所有正式的DOS实用程序所接受。后一个仅在V2的一种变体版本中出现过，然后又以稍微不同的形式重现在V4中。

就一般的编程而言，这两个功能没有什么用处。

四、还不能肯定的功能

69H，6BH这两个功能都是在V4中才增加的，它们至今还没有被弄清楚。看上去它们与DOS临界区有关，但它们究竟干什么仍是个谜。

2.2.2　功能52H：取配置变量表

随着V3的出现和对网络支持的引入，DOS有必要将系统配置参数标准化，以便要求在网

络上运行的应用程序能得到所需信息。为此，Edwin Floyd 博士（他是发现 CVT 存在和用途的首批非 Microsoft 和 IBM 工作人员之一）编制了配置变量表（CVT）。实际上，表中大部分内容已在 V2中出现过。在 V3中，CVT 被明显地扩展了，而且它在 RAM 中的位置也稍有变动。

配置变量表的设置靠近 MSDOS.SYS 程序的起始部分。表中不仅含有指向内存控制块（MCB）链的指针和其他一些基本的控制信息，还有关于驱动器的一些信息。

一、CVT 的内容和格式

在 DOS V2中，CVT 只占25个字节，在 V3和 V4中则占42个字节。这三个版本中，紧跟在 CVT 后的是 NUL 设备驱动程序，它总是链中的第一个设备驱动程序（可安装的设备驱动程序被安装在 NUL 设备驱动程序和其余的设备驱动程序之间）。

为定位 CVT，须调用 INT 21H 未写入文档的功能52H，此功能不需任何参数。它在 ES：BX 中返回一个指针，指向由 DOS 维护的第一个驱动器参数块 DPB 的地址。在 V2中，指针指向的是 CVT 的第二项（在它之前是双字节的内存控制块 MCB 链起始段地址）。在 V3和 V4中，指针指向的是 CVT 的第八个字节，因为在 MCB 段地址前增加了指向当前缓冲区控制块的32位远指针和定位当前缓冲区的16位偏移值。

为了得到 CVT 的起始地址，在 V2中，该功能返回后，BX 必须减2；在 V3和 V4中，返回后 BX 必须减8。

在已经知道如何访问 CVT 后，下面看看它的内容。

二、V2中的 CVT

下面的 CVT 格式表明最初它是如何出现在 DOS V2中的（表2.1）。“*”表示由功能52H 所返回的地址，表中所有偏移量均是相对该地址而言。

表2.1 DOS V2的 CVT 格式

偏移量	域宽	含义
−02H	字	第一个 MCB 的段地址
*00H	双字	指向第一个驱动器参数块的指针
04H	双字	指向第一个 DCB（系统文件表设备控制块）的指针
08H	双字	指向 CLOCK $ 设备驱动程序的指针
0CH	双字	指向 CON 设备驱动程序的指针
10H	字节	逻辑驱动器数
11H	字	块设备上每扇区的最大字节数
13H	双字	指向磁盘缓冲区链开头的指针
17H		NUL 设备驱动程序的起始点；设备驱动程序链中的第一个

在 V2和 V3中，偏移量−02H 处的字所指向的 MCB 的格式很简单（表2.2）。

表2.2 DOS V2和 V3的 MCB 格式

偏移量	域宽	含义
00H	字节	MCB 标识符；除最后一个 MCB 之外，“M”为所有 MCB 的标识；“Z”为最后一个 MCB 的标识；标识符都位于偏移量0000H 处
01H	字	占用该 MCB 的进程的 PSP 地址；如果地址为0000H，则表明这块内存未被分配
03H	字	下面内存块的节数。不包括这一16字节的 MCB 区域
05H	11字节	此节中未用的部分

CVT中偏移量为00H处的双字指向的DPB结构，已在本书中未写入文档的功能53H的相关章节中描述。在V2和V3中，DPB没有发生变动。考虑到DPB的内容已在有关章节出现过，在此就不重复了。系统中每个物理驱动器（在单驱系统中也包括驱动器B）都有自己的DPB；上一个DPB的指向“下一块”的指针为FFFF：FFFFH，标识链结束。

系统DCB的数目由CONFIG.SYS文件中的“FILES＝”语句设定，缺省值为5。这些DCB链接在一起，每个DCB的头部有3个字的连接头（表2.3）。

表2.3　DOS V2和V3的DCB连接头格式

偏移量	域宽	含义
00H	双字	指向下一个连接头的远指针，如为FFFF：FFFFH，则表明是最后一个DCB
04H	字	该链中DCB的数量

CVT中偏移量04H处的双字是指向DCB链中第一个DCB头的指针。在DOS V2中，DCB长40个字节（表2.4），与文档中说明的FCB结构相同，但稍有变动。

表2.4　DOS V2的DCB格式

偏移量	域宽	含义
00H	字节	使用此DCB的当前用户数
01H	字节	打开DCB的访问方式
02H	字节	来自目录项的属性字节或00H表示设备
03H	字节	驱动器代码（1代表A，等等）
04H	8字节	文件或设备名
0CH	3字节	文件扩展名或为空
0FH	13字节	未知；可能类似于文件FCB
1CH	双字	指向设备驱动程序的远指针；对文件还未知
20H	8字节	未知

在偏移量11H处的字指出块设备每扇区的最大字节数。它是加电初始化时，随着块设备驱动程序的安装而设定的。在配置过程中，它用于设置缓冲区的大小，以保证每个缓冲区足够存放系统现有的最大扇区而又不致过大。

CVT中偏移量13H处的双字指向磁盘缓冲区链，格式如表2.5（V2中的缓冲区结构还不是很清楚）：

表2.5　DOS V2的磁盘缓冲区链格式

偏移量	域宽	含义
00H	双字	指向下一个缓冲区的远指针，如为FFFF：FFFFH，表明是链尾
04H	4个字	尚未完全分析清楚的控制信息，包括标志字节、逻辑扇区号和驱动器代码
0CH	双字	指向与此缓冲区有关的DPB的远指针
10H	变化量	缓冲区，常为512个字节，但实际长度存放在CVT中的偏移量11H处

三、V3中的CVT

在DOS V3中，CVT的两头都有扩充，如表2.6所示。同V2的一样，“*”表示功能52H返回的地址，表中所有偏移量都是相对该地址而言。

表2.6 DOS V3的CVT格式

偏移量	域宽	含义
−08H	双字	在“BUFFERS=”链中的当前缓冲区
−04H	字	当前缓冲区内的偏移值
−02H	字	第一个MCB的段地址
*00H	双字	指向第一个DPB的指针
04H	双字	指向第一个DCB（系统文件表设备控制块）的指针
08H	双字	指向CLOCK$设备驱动程序的指针
0CH	双字	指向CON设备驱动程序的指针
10H	字	块设备每扇区的最大字节数
12H	双字	指向磁盘缓冲区链开头的指针
16H	双字	指向逻辑驱动器表的指针
1AH	双字	指向DOS的FCB链开头的指针
1EH	字	交换时要保存的FCB数
20H	字节	块设备数
21H	字节	逻辑驱动器数，由CONFIG.SYS中LASTDRIVE设置，缺省值为5
22H		NUL设备驱动程序的起始点；设备驱动程序链中的第一个

偏移量−08H处的双字指向磁盘缓冲区链中的当前缓冲区。缓冲区控制块的格式稍后介绍。偏移量−04H处的字是当前缓冲区内相对缓冲区首字节的字节偏移量；值0000H表示缓冲区已用完，需要进入链中的下一个缓冲区。

与V2一样，偏移量−02H是MCB链中的第一个MCB的段地址，MCB的格式同V2一样。DPB链也和V2一样，由偏移量00H处的双字指向它的第一项。

为了适应网络和多进程公用同一文件或设备的要求，V3中的DCB比V2有所扩展。但与V2一样的是，DCB仍借助连接头链接起来，连接头的格式没有改变。DCB的新结构格式如表2.7。

表2.7 DOS V3的DCB格式

偏移量	域宽	含义
00H	字	此DCB的用户数
02H	字	每次打开时的访问方式
04H	字	磁盘属性字节
05H	字节	设备属性
06H	字节	设备属性第二字节
07H	双字	指向驱动程序的远指针
0BH	字	首簇号

续表

偏移量	域宽	含义
0DH	字	文件时间字
0FH	字	文件日期字
11H	双字	文件大小
15H	双字	当前字节位置
19H	字	总簇数
1BH	字	当前簇号
1DH	字	目录扇区
1FH	字节	目录项在扇区中的索引
20H	8字节	设备或文件名
28H	3字节	文件扩展名或为空
2BH	字	未知
2DH	字	未知
2FH	字	未知
31H	字	拥用者的PSP段地址
33H	字	未知

在偏移量10H处的字指出块设备每扇区的最大字节数。它是加电初始化时，随着块设备驱动程序的安装而设定的。在配置过程中，它用于设置缓冲区的大小，以保证每个缓冲区足够存放系统现有的最大扇区而又不致过大。

偏移量12H处的字指向磁盘缓冲区链的开头。链中每个缓冲区的大小等于表2.8偏移量10H处的值（通常为512字节），再加上数据区前的16字节的连接头。格式如表2.8。

表2.8　DOS V3的磁盘缓冲区链格式

偏移量	域宽	含义
00H	双字	指向下一个缓冲区的远指针，或为FFFF：FFFFH，表明是链尾
04H	字节	逻辑驱动器号
05H	字节	动作码
06H	字	逻辑扇区号
08H	字节	FAT数或为01
09H	字节	每FAT的扇区数或为00
0AH	双字	指向与该缓冲区相关的DPB的远指针
0EH	字	未知
10H	变化量	缓冲区自身，通常为512字节，实际大小存放在CVT中偏移量10H处

在V2中，“当前目录”区是DPB的一部分；在V3中，它变成一个独立的逻辑驱动器表。在CVT中偏移量16H处的双字指向LDT，每一系统逻辑驱动器在LDT中都占有一项，表的项数由CONFIG.SYS文件中的LASTDRIVE设置，缺省时为5（A：到E：）。在V3中，每个表占81个字节，在内存中首尾相邻。格式如表2.9

表2.9 DOS V3的 LDT 格式

偏移量	域宽	含义
00H	2字节	实际驱动器名和“:”
02H	65字节	以 ASCIIZ 字符串表示的当前驱动器路径（包括根目录斜线和用于终止的字节0）
43H	字	驱动器当前状态（位映象） 8000H＝未知，可能表示远程网络驱动器 4000H＝已准备好 2000H＝未知 1000H＝使用 SUBST 命令指定的驱动器
45H	双字	指向驱动器 DPB 的指针
49H	字	当前目录首簇号
4BH	字	未知
4DH	字	未知
4FH	字	报告目录时应略去的字节数。 0002H 代表普通驱动器， 大于0002H 的数代表使用 SUBST 命令指定的驱动器

偏移量1AH 处的双字和其后字节与文件 CONFIG. SYS 中的 FCBS＝语句有关，它们分别是指向系统 FCB 链的远指针和交换时要保护的 FCB 数。尽管 FCB 主要用于网络，但在单用户环境中有时也使用。FCB 的格式同 DCB 一样，由多个块组成并连接起来。

在 V3 CVT 中，最后两项分别位于偏移量20H 和21H 处。它们是块设备数和逻辑驱动器数。前者在安装块设备时由其总数决定，后者由文件 CONFIG. SYS 中的 LASTDRIVE＝语句设置。

四、V4中的 CVT

与从 V2到 V3的变动相比，CVT 从 V3到 V4的变动要小些。然而，由于 DOS 增加了对 EMS 支持，并且逻辑扇区号由16位变为32位，这些变动也必然要影响到 CVT。另外，重新改写的缓冲算法对缓冲区结构也有较大的影响。表2.10是 V4的 CVT，唯一重要的变动是在偏移量12H 处的缓冲区链头指针变成了指向 EMS 链记录的指针。

表2.10 DOS V4的 CVT 格式

偏移量	域宽	含义
－08H	双字	在“BUFFERS＝”链中的当前缓冲区
－04H	字	当前缓冲区内的偏移量
－02H	字	第一个 MCB 的段地址
＊00H	双字	指向第一个 DPB 的指针
04H	双字	指向第一个 DCB（系统文件表设备控制块）的指针
08H	双字	指向 CLOCK $ 设备驱动程序的指针
0CH	双字	指向 CON 设备驱动程序的指针
10H	字	块设备每扇区的最大字节数
12H	双字	指向链到 DOS 缓冲区链的 EMS 链记录的指针（V3和 V4之间的唯一变动）
16H	双字	指向逻辑驱动器表的指针（参见此表后的讨论，这里稍有改动）
1AH	双字	指向 DOS FCB 链开头的指针
1EH	字	交换时要保存的 FCB 数
20H	字节	块设备数
21H	字节	逻辑驱动器数，其值由 CONFIG. SYS 中的 LASTDRIVE 设置，缺省值为5
22H		NUL 设备驱动程序的起始点；设备驱动程序链中的第一个

在V4中由功能52H调用返回的地址没有改变，但是在偏移地址－08H处的双字所指向的缓冲区控制块却有了较大的变化。以前，每一缓冲区都有自己的段地址，并且所有缓冲区是靠一个远指针按向前的顺序链接起来的。在V4中，所有缓冲区在同一段内，通过某一EMS链接块访问，而且它们通过近指针双向（向前、向后）链接成循环链。

为适应新的缓冲区算法，缓冲区控制块的大小从16字节扩充为20字节。在V4中，其结构如表2.11。

表2.11　DOS V4的缓冲区控制块格式

偏移量	域宽	含义
00H	字	链中上一个缓冲区的偏移量，位于段内或EMS物理页面内
02H	字	链中下一个缓冲区的偏移量，位于段内或EMS物理页面内
04H	字节	逻辑驱动器号（同V3）
05H	字节	动作码（同V3?）
06H	双字	逻辑扇区号，已被扩展
0AH	字节	FAT数或为01；完整的含义未知
0BH	字	如FAT在缓冲区中，则表示每FAT的扇区数；否则，含义未知
0DH	双字	指向与此缓冲区相关的物理驱动器DPB的远指针
11H	字	含义未知
13H	字	含义未知
14H	512字节	缓冲区自身；与V3中一样，它的实际长度由CVT设置，但通常为512个字节

为了提高缓冲区的搜索速度，在V4的缓冲算法中加入了一个新的连接记录。这11个字节的记录存放了一个散列值，以指向相应的BCB。它的所有细节还没有完全得知。其结构如表2.12。

表2.12　DOS V4缓冲区搜索中的连接记录格式

偏移量	域宽	含义
00H	字	散列值
02H	双字	指向相应的BCB的远指针
06H	字节	使用计数器
07H	双字	含义未知

为能把缓冲区存贮在EMS而不是在常规存储器中，在算法中增加了第二个新记录。这个记录把一个EMM句柄、物理页面和一个连接记录联系在一起。由于在系统软件和非IBM EMS硬件之间还存在着兼容问题，我们还无法详细、准确地测试当EMS活动时这一记录是如何起作用的。表2.13的描述适合非EMS的环境。

表2.13 DOS V4新增的记录格式

偏移量	域宽	含义
00H	双字	指向上述连接记录的远指针
04H	字	此记录控制的页数
06H	双字	含义未知
0AH	字	含义未知
0CH	字节	标志；若没用 EMS，值为 FFH
0DH	字	EMM 句柄值
0EH	字节	EMS 物理页面号

下一个有变动的地方是 DPB 结构，它由 CVT 中偏移量00H 处的指针所指向。驱动器参数块本身唯一的变化是“每 FAT 的扇区数”从8位变到16位，从而使所有随后项增加1个字节的偏移量。

从 DPB 指针起到偏移地址12H 处的内容保持不变，但偏移量为12H 处的远指针却变成了指向磁盘缓冲区 EMS 记录的指针。这与 BCB 的变动有关，已在上面介绍了。

CVT 中的偏移量16H 处的远指针所指向的 LDT 扩充了7个字节，扩展的意图未知。新结构如表2.14。

表2.14 DOS V4的 DPB 格式

偏移量	域宽	含义
00H	2字节	实际驱动器名和“:”
02H	65字节	以 ASCIIZ 字符串表示的当前驱动器的当前路径（包括根目录斜线和用于终止的字节0）
43H	字	驱动器当前状态（位映象） 8000H＝未知 4000H＝已准备好 2000H＝未知 1000H＝使用 SUBST 命令指定的驱动器
45H	双字	指向驱动器 DPB 的指针
49H	字	当前目录的首簇号
4BH	字	未知
4DH	字	未知
4FH	字	报告目录时所略去的字节数； 0002代表普通驱动器，大于0002的数代表使用 SUBST 命令指定的驱动器
51H	字节	未知
52H	字	未知
54H	字	未知
56H	字	未知

以反汇编和测试的结果来看，CVT 中的其他项与 V3一样，没有变动。

2.2.3 小结

在本节中，已介绍了未写入文档的 DOS 功能的全貌。我们写了两个示范程序 CVT3.PAS 和 CVT4.PAS，分别适合 DOS V3.x 和 V4，V5。示范程序旨在演示如何用未写入文档的功能调用去探索 DOS 内部工作概貌。限于篇幅，在这里不列出源程序清单，有兴趣的读者可与出版社联系。

切记，如果您的程序使用了未写入文档的功能，一旦这些功能在 DOS 的新版本中有了较大的变化，您的程序就可能出现令人头痛的问题。有鉴于此，在开发商用或公用程序时，应尽量少用这些功能。

2.3 关于 DOS 严重错误处理

在中断处理程序中，最需要的可能是 DOS 严重错误处理程序，发生严重错误时，DOS 调用此处理程序，让用户决定是终止程序，还是处理错误后再继续运行。

DOS 标准的处理就是“Abort，Retry or Fail”提示。对程序员来说，这简直是一个灾难，因为它破坏了程序精心设计的屏幕，并只给出了很少的错误信息。

DOS 向错误处理程序提供的错误信息相当详细，编写大型软件系统一定要充分利用 DOS 提供的错误信息，从而制作出完善可靠的产品。

我们用 C 语言编写了一个完整的严重错误处理程序，限于篇幅，这里略去了程序清单。

·MEMO·

第二部分　DOS 编程必备

• MEMO •

第三章　系统内存管理技术

3.1 内存工作管理

PC 及其兼容机有 1M 的可寻址空间，但程序只能用其中的 640K 内存，剩下的 384K 分配给 ROM BIOS、显示适配器和卡式录音机等。但这 640K 空间也不是全供应用程序使用，其中有 1K 字节中断向量表，接下来还有 BIOS 和 DOS 内部变量表、DOS 内核、系统驱动程序，最后还有命令处理程序的驻留部分。

计算所剩的空间比较困难，因为它取决于所用系统和装入的驱动程序。一般首先要安装的是 TSR 程序，比如 SideKick。图 3.1 就是这种装入 TSR 后的情况。如果再装入像 DESQView 之类的窗口软件，640K 就只剩下 350K 左右了，而实际上还没有开始做任何事情！

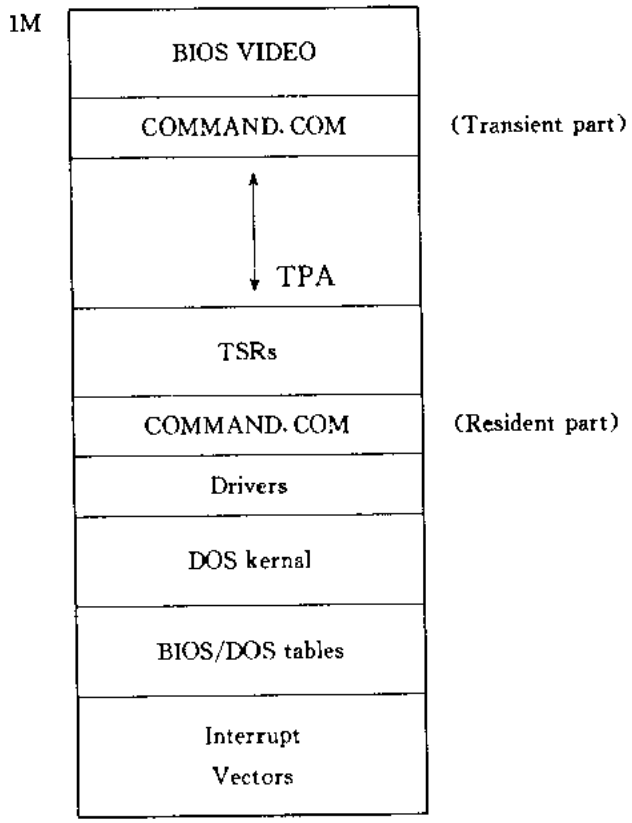


图 3.1　装入 TSR 后的内存状态

系统启动后所剩的空间（如装入 SideKick 后为约 500K－550K）称为临时程序区（TPA）。这个名称很贴切，因为用户程序在此运行是临时的。

随着需要的增加，640K 内存成了系统的限制之一。在引入 80286 后，PC 可用多达 16M

的内存空间。但 DOS 不能使用这么多空间。要访问这些内存，80286 必须工作在保护方式下，而 DOS 做不到这点。在 DOS 下，只有那些能控制 CPU 方式切换的专用程序才能使用扩充内存。

DOS 在实模式下运行所有程序。这样，一般程序不能访问 1M 以上的内存。即使系统有另外的内存，也不能有效利用。处理器在扩充内存上操作时，必须先切换到保护方式，然后切换回来取数据。

PC AT 出现后，BIOS 新增加了两个功能。INT 15H 功能 88H 用来判定可用内存有多大，INT 15H 功能 87H 用来使用 1M 以外的扩充内存交换数据块。但由于没有对扩充内存进行管理，某个程序可能改写另一个程序（或 RAM 磁盘驱动器）保存在扩充内存上的数据，并且根本不报告错误。

扩展内存管理规范 EMS（V3.0）首先由 Lotus 和 Intel 在 1985 年共同引入。使用它可以不用查询处理器模式就能访问多达 8M 的内存。扩展内存让处理器以 16K 为页面访问扩展内存。16K 内存可映象到 640K—1M 之间的没有使用的地址区域。4 个 16K 页映象到一个 64K 的页面帧，其位置由用户在装入时决定。

扩展内存管理程序（EMM）采用类似于文件句柄的方法管理扩展内存。当用户程序请求扩展内存时，EMM 为其设定某个空间，并返回一个可访问该空间的唯一的句柄。

可用 INT 67H 调用获得一个 16K 页，并把它放进页面帧。这样，用户在程序中就可直接寻址此内存。

在 CONFIG.SYS 文件中加入 DEVICE＝EMM.SYS 之后就安装了 EMS 驱动程序。

访问 EMM 的功能主要通过 INT 67H，其可用功能包括：

- 报告扩展内存状态
- 分配扩展内存页
- 释放扩展内存页
- 诊断
- 多任务支持
- 把扩展内存的物理页映象到该程序的逻辑页

扩展内存引入后不久，Microsoft 宣布支持 EMSV3.2。V3.2 就变成了现在所说的 LIM EMS。V3.2 中包括一些可用于多任务操作系统的功能，后来，Ashton－Tate，AST Research 和 Quadram 认为只把 16K 页映象到不用内存区不合理，应从 TPA 中映象更大的页，这就导致了增强型扩展内存规范 EEMS。在 EEMS 中，可以将整个程序移至扩展内存。这样，PC 就变成了真正的多任务系统。

3.2 内 存 管 理

DOS 的内存管理针对的是 TPA 中的自由区域。TPA 被纳入 Memory Arena 结构中。DOS 中保存了一个链状的内存块表，内存块是一 Arena 项，每一项都带有一个特定的叫 Arena Header 的控制块。有三个 DOS 功能（INT 21H 功能 48H、49H 和 4AH）可用来请求或释放内存块。

Arena 链把各个内存块链入内存块表。如果有两个内存块相邻，它们就被组合成为一个更

大的块，并且只带有一个 Arena Header。

当有内存请求时，DOS 搜寻 Arena 链，以找到一个足够大的块。DOS 采用最先适合策略，返回链中第一个足够大的块。此块的剩余空间作为一个单独内存块放回链表中。从 DOS V3 开始，用户还可选择最佳适合策略和最后适合策略。

3.2.1　减少程序占用内存

DOS 本身是一个单用户系统，每次只能运行一个程序。当一个 COM 程序开始运行时，它分得了全部内存。

EXE 程序一般也是分得全部内存，但用户可以减小这一分配。EXE 程序头有两个参数：MINALLOC 和 MAXALLOC。MINALLOC 是程序运行所需的最小内存。Microsoft Linker 一般设 MINALLOC 为 0，MAXALLOC 为 0FFFFH，否则由用户指定。注意，Turbo C 和 Microsoft C 编译的程序在开始运行时自动释放多余内存，但 Turbo Pascal 和 Microsoft Linker 不是这样。Turbo Pascal V4 和 V5 所产生的 EXE 文件缺省时用所有可用内存，并把高端区域作为堆管理。但它也有编译器指令可设置所用内存大小。

旧版本的 Turbo Pascal 生成 COM 文件，这样，只有在编译时才能设置程序所用最大内存。由于它分配内存时，堆栈和堆在程序之上，所以无法判定此程序在哪里终止。

BASIC 程序也会遇到同样的问题。程序可用专门的 MEMSET 功能设置内存上限，但这一功能主要用于 BASIC 程序装入汇编语言支持例程，而不是其他程序的执行。CHAIN 和 RUN（用于执行其他程序的 BASIC 语句）用一新程序取代内存中已有的程序，而不能维持当前程序的状态。

在高级语言中，C 和新版的 Turbo Pascal 适于动态内存分配。C 提供了用于内存分配和释放的良好功能。新版的 Turbo Pascal 相对于标准 Pascal 有许多扩充功能，也有与 C 相同的功能，只是名字稍有不同。

用汇编语言写的程序在开始运行时必须释放掉多余的内存。下面就是这样的一个例子：

【程序 3.1】

```
        mov   sp, offset stack      ; Move stack to safe area
        mov   ah, 4ah               ; Set block
        mov   bx, 280h              ; Retain 10K of spaceint 21h
        jc    alloc_error           ; Allocation error
;
;       ; [MORE PROGRAM CODE]
;
        dw    64 dup (?)
stack   equ   $
```

3.2.2　获得更多内存

如果一程序需要更多的内存，它可调用 INT 21H 功能 4AH 来获得。返回时，如果内存够用，进位标志清零，AX 寄存器有此内存块的段基址；如果内存不够，则进位标志置位，AX 中为错误标志（7＝内存控制块被破坏，8＝无足够内存），寄存器 BX 为最大可用块的大小。

C 和新版本的 Turbo Pascal 都提供了获得和释放附加内存的功能，老版本的 Turbo Pas-

cal (V4.0 以前) 在请求分配另外内存时，功能很弱或者不存在，除非使用特殊的编译器开关。BASIC 无法改变 BASIC 程序的内存分配，也不能决定从哪里去进入内存以限制内存使用。

有一些编程技巧有时可用于释放内存，但对运行在交互编程环境下的 Turbo BASIC，它在高内存保存了一个符号表，如果用户试图在此内存中装入其他程序时，此表将被重写。

再强调一次：只有在汇编语言中才需要直接访问 DOS 分配功能，在 C 和新版本 Turbo Pascal 中，是库函数分配例程访问这样的 DOS 功能。程序 3.2 给出了一个在第一次尝试失败后取内存及判定所剩内存的例子。

【程序 3.2】

```
getmem:
        mov     ah, 48h         ; Allocate memory
        mov     bx, bufsize     ; 16K memory
        int     21h
        jc      nomem           ; Cannot allocate
        mov     bufseg, ax      ; Save pointer
nomem:
        cmp     ax, 8
        jnz     quit            ; Major alloc error
        mov     bufsize, bx     ; Save buffer size
        mov     ah, 48h         ; Allocate memory
        int     21h
        jc      quit            ; still cannot allocate
pgm:
        ;
                ; [MAIN PART OF PROGRAM]
        ;
done:
        mov     ah, 49h         ; Deallocate memory
        mov     es, bufseg      ; Point to buffer segment
        int     21h
bufsize dw      400h
bufseg  dw      0
```

有几个分配和释放功能可为 C 或新版本 Turbo Pascal 所写的程序调用，它不用访问 DOS 内存分配调用，这样就大大简化了操作并易于控制。Turbo Pascal 4.0 以前版本拥有堆中所有可用内存，为程序分配和释放这一空间，这些 Pascal 中例程不在 ARENA 中分配或释放空间。

BASIC 和 Pascal (V4.0 以前) 程序员可用编译器开关人工控制内存，然后用 DOS 功能去请求或释放另外内存。但这样会导致混乱。有动态内存处理的程序应该使用 C、新版本的 Turbo Pascal 或汇编语言编写。

3.3 EMS 技术

使用扩展内存可以快速、有效地访问大量的数据。扩展内存也为进行多任务提供了空间。

3.3.1 判定扩展内存是否可用

欲判定是否有扩展内存，可采用以下方法：

● 试用打开文件 'EMMXXXX0'，如果打开成功，表示 EMS 驱动程序或一个同名

的文件存在。使用IOCTL功能取输出状态调用，可进一步判定是否是EMS驱动程序。如果是EMS驱动程序，则返回值为0FFH；如果只是一个文件，则返回值为00H。检测到EMS驱动程序后，关闭文件以使句柄可重用。

- 检查INT 67H向量。如果EMS存在，那么该中断向量的段地址是驱动程序的基址，偏移0AH处是驱动程序名，它是驱动程序头的一部分。虽然使用这一方法比打开一个文件要快，但它需要访问程序正常内存范围以外的内存。

程序3.3和程序3.4给出了两个独立的检测EMS驱动程序是否存在的例程。

【程序3.3】

```
/* emmtest () */
#include <dos.h>

emmtest ()
 {
        union REGS regs;
        struct SREGS sregs;
        short int result;
        unsigned int handle;

        regs.h.ah = 0x3d;                       /* Open file */
        regs.h.al = 0;                          /* Read mode only */
        regs.x.dx = FP_OFF ("EMMXXXX0");/* File name */
        sregs.ds = FP_SEG ("EMMXXXX0");/* Set the DS register */
        intdosx (&regs, &regs, &sregs);

        handle = regs.x.ax;                     /* File handle */
        /* If opened OK, then close the file */
        if (result = (regs.x.cflag == 0)) {
                regs.h.ah = 0x3e;
                regs.x.bx = handle;
                intdos (&regs, &regs);
        }
        return (result);
}
```

注意：用BORLAND C++编译程序3.3的函数时，在if（result=...）语句处会出现一个警告，用户可不去管它。

【程序3.4】

```
/* emmtest () */
#include <dos.h>
#include <malloc.h>             /* Note : See new include file */

emmchk ()
 {
        union REGS regs;
        struct SREGS sregs;
        char far *emptr, far *nameptr;
        nameptr = "EMMXXXX0";

        regs.h.ah = 0x35;       /* Get interrupt vector */
        regs.h.al = 0x67;       /* Get it for the EMM */
        intdosx (&regs, &regs, &sregs);

        /* Make a FAR pointer to access the driver */
        emptr = MK_FP (sregs.es, 0); /* Not in Microsoft C 6.0 */
        if ( (emptr = _fmalloc (sregs.es, 9)) == NULL)
             {
```

```
            printf (" Unable to allocate far pointer. \n");
            exit (0);
            }
        /* Return TRUE if they are the same for eight characters */
        return (farcmp (emptr+10, nameptr, 8));
}
#define    FALSE  0
#define    TRUE   ! FALSE
farcmp (str1, str2, n);
char    far *str1,
        far *str2;
int     n;
 {
        while ( *str2 && n>0) {
                if ( *str1 != *str2)
                        return (FALSE);
                n--;
                str1++; str2++;
        }
        return (TRUE);
}
```

程序 3.5 是一个使用了以上两种方法来检测 EMS 的简单例子。

【程序 3.5】

```
#include <stdio.h>
main ()
 {
        if (emmchk ())
            printf (" MEM: Expanded memory is present\n");
        else
            printf (" MEM: Expanded memory is NOT present\n");
        if (emmtest ())
            printf (" OPEN: Expanded memory is present\n");
        else
            printf (" OPEN: Expanded memory is NOT present\n");
}
```

3.4 使用扩展内存

程序 3.6 和程序 3.7 显示了如何使用 INT 67H 获取并分配扩展内存。

程序 3.6 所列程序完成以下工作：

- 测试扩展内存是否存在
- 测试扩展内存管理程序是否工作正常
- 试着分配 10 页扩展内存
- 每次把一个页面映象到页面帧，并在其中写入一测试字符串
- 再把页面映象回页面帧，然后读取并打印测试字符串。

【程序 3.6】

```
#include <stdio.h>
unsigned int emmalloc ();
```

```
main ()
{
        unsigned int emmhandle;
        char     teststr [80];
        int      i;
        /* Is there any expanded memory ? */
        if (! emmtest ()) {
            printf (" Expanded memory is NOT present\n");
            printf (" cannot run this program\n");
            exit (0);
        }
        /* Is the expanded memory manager functional ? */
        if (! emmok ()) {
            printf (" Expanded memory manager NOT available\n");
            printf (" cannot run this program\n");
            exit (0);
        }
        /* Get 10 pages of expanded memory for the demo */
        if ( (emmhandle = emmalloc (10)) < 0) {
            printf (" There are not enough pages available\n");
            printf (" cannot run this program\n");
            exit (0);
        }
        /* Write the test string into each of the 10 pages */
        for (i=0; i<10; i++) {
            sprintf (teststr, "%2d: Terry was here\n", i);
            emmmap (emmhandle, i, 0);
            emmmove (0, teststr, strlen (teststr) +1);
        }
        /* Now read them back in and recover the test string */
        for (i=0; i<10; i++) {
            emmmap (emmhandle, i, 0);
            emmget (0, teststr, strlen (teststr) +1);
            printf (" READING FROM BLOCK %d : %s\n", i, teststr);
        }
        /* Finally, release the expanded memory */
        emmclose (emmhandle);
}
```

程序 3.7 中的函数包括了扩展内存的几种基本操作：

emmtest，emmok，emmalloc，emmap，emmove，emmget，emmclose，其功能在此无需赘述。

【程序 3.7】

```
#include <dos.h>
#define    FALSE    0
#define    TRUE     ! FALSE
#define    EMM      0x8767
char far * emmbase;
emmtest ()
 {
        union REGS regs;
        struct SREGS sregs;
        short int result;
        unsigned int handle;
```

```
        regs.h.ah = 0x3d;                       /* Open file */
        regs.h.al = 0;                          /* Read mode only */
        regs.x.dx = (int)" EMMXXXX0";           /* File name */
        sregs.ds = _DS;                         /* Set the DS register */
        intdosx (&regs, &regs, &sregs);

        handle = regs.x.ax;                     /* File handle */
        if (result = (regs.x.cflag == 0)) {
                regs.h.ah = 0x3e;
                regs.x.bx = handle;
                intdos (&regs, &regs);
        }
        return (result);
}

emmok ()
 {
        union REGS regs;

        regs.h.ah = 0x40;                       /* Get manager status */
        int86 (EMM, &regs, &regs);
        if (regs.h.ah ! = 0)
                return (FALSE);

        regs.h.ah = 0x44;                       /* Get page frame segment */
        int86 (EMM, &regs, &regs);
        if (regs.h. ah ! = 0)
                return (FALSE);

        emmbase = MK_FP (regs.x.bx, 0);
        return (TRUE);
}

unsigned int emmalloc (n);
int n;
 {
        union REGS regs;

        regs.h.ah = 0x43;       /* Get handle and allocate memory */
        regs.x.bx = n;
        int86 (EMM, &regs, &regs);
        if (regs.h.ah ! = 0)
                return (-1);
        return (regs.x.dx);
}

emmmap (handle, phys, page)
unsigned int handle;
int       phys;
int       page;
 {
        union REGS regs;

        regs.h.ah = 0x44;                       /* Map memory */
        regs.h.al = page;
        regs.x.bx = phys;
        regs.x.dx = handle;
        int86 (EMM, &regs, &regs);
        return (regs.h.ah = 0);
}

emmmove (page, str, n)
int       page;
chat      *str;
int       n;
 {
        char far *ptr;
```

```
        ptr = emmbase + page * 16384;
        while (n-- > 0)
            *ptr++ = *str++;
}

emmget (page, str, n)
int     page;
char    *str;
int     n;
 {
        char far *ptr;

        ptr = emmbase + page * 16384;
        while (n-- > 0)
            *str++ = *ptr++;
}

emmclose (handle)
unsigned int handle;
 {
        union REGS regs;

        regs.h.ah = 0x45;       /* Release handle */
        regs.x.dx = handle;
        int86 (EMM, &regs, &regs);
        return (regs.h.ah == 0);
}
```

以下是一个使用 EMS 的 PASCAL 的示范程序。

【程序 3.8】

```
{ EMS.PAS          Compiled by Turbo Pascal 6.0          }
{$A+, B-, D+, E+, F-, G-, I+, L+, N-, O-, R-, S-, V+, X+}
{$M $2000, 0, 0}
{*********************** EMS ROUTINE ***********************}
var
  EmmErr : Byte;
  EmmBase : Word;

function EMM_test: Boolean;
var
  F : File;
begin
  assign (F, 'EMMXXXX0');
  {$I-} reset (F); {$I+}
  if IOresult = 0 then begin
    Close (F);
    EMM_test := True;
  end
  else EMM_test := False;
end;       { EMM_test }

function EMM_ok (var emmbase : word) : Boolean; assembler;
asm
  MOV     AH, 40H           { Get Manager Status }
  INT     67H
  MOV     EmmErr, AH
  OR      AH, AH
  JNZ     @1
  MOV     AH, 41H           { Get Page Frame Segment }
  INT     67H
  MOV     EmmErr, AH
```

```
  OR      AH, AH
  JNZ     @1
  LES     SI, emmbase
  MOV     WORD PTR ES: [SI], BX
  MOV     AL, 1              { True }
  JMP     @2
@1:
  MOV     AL, 0              { False }
@2:
end;

function EMM_alloc (n: word): word; assembler; asm
  MOV     AH, 43H            { Get handle and allocate memory }
  MOV     BX, N
  INT     67H
  OR      AH, AH
  JZ      @1
  MOV     DX, 0FFFFH         { False }
@1:
  MOV     AX, DX
end;

function EMM_map (Handle, Phys: word; Page: Byte): Boolean; assembler;
{ Phys: Logical page number (zero_based, FFFFh to unmap physical page) }
asm
  MOV     AH, 44H            { Map/UnMap memory }
  MOV     AL, Page
  MOV     BX, Phys
  MOV     DX, Handle
  INT     67H
  MOV     EmmErr, AH
  OR      AH, AH
  JZ      @1
  MOV     AL, 0              { False }
  JMP     @2
@1:
  MOV     AL, 1              { True }
@2:
end;

procedure EMM_move (Page: byte; P: Pointer; N: word);
var
  PP: Pointer;
begin
  PP := Ptr (EmmBase, Page * $4000);
  Move (P^, PP^, N);
end;

procedure EMM_get (Page: byte; P: Pointer; N: word);
var
  PP: Pointer;
begin
  PP := Ptr (EmmBase, Page * $4000);
  Move (PP^, P^, N);
end;

procedure EMM_close (Handle: word); assembler;
asm
  MOV     CX, 3
@3:
  PUSH    CX
  MOV     AH, 45H            { Release handle }
  MOV     DX, Handle
  INT     67H
  MOV     EmmErr, AH
```

```
  POP     CX
  OR      AH, AH
  JZ      @1
  LOOP    @3
  MOV     AH, 0              { False }
  JMP     @2
@1:
  MOV     AH, 1              { True }
@2:
end;

{ ********************************************************************** }

var
  EmmHandle: integer;
  TestStr: string;
  I: word;

begin     { Main }
  Writeln;
  { Is there any expanded memory }
  if not EMM_test then begin
    Writeln('Expanded memory is NOT present ');
    Writeln(' cannot run this program          ');
    Halt(0);
  end;

  { Is the expanded memory manager functional ?}
  if not EMM_ok(EmmBase) then begin
    Writeln('Expanded memory manager NOT available ');
    Writeln(' cannot run this program                ');
    Halt(0);
  end;

  { Get 10 pages of expanded memory for the demo }
  emmhandle := EMM_alloc(10);
  if (emmhandle = -1) then begin
    Writeln('There are not enough pages available ');
    Writeln(' cannot run this program             ');
    Halt(0);
  end;

  { Write the test string into each of the 10 pages }
  for I := 0 to 9 do begin
    TestStr := 'EMS Test Page '+Chr(I+$30);
    EMM_map(EMMhandle, I, 0);
    EMM_move(0, @TestStr, 16);
  end;

  { Now read them back in and recover the test string }
  TestStr := '';
  for I := 0 to 9 do begin
    EMM_map(EMMhandle, I, 0);
    EMM_get(0, @TestStr, 16);
    Writeln(TestStr);
  end;
  EMM_close(EMMhandle);
end.
```

3.5　XMS 技术

3.5.1　扩充内存一般介绍

扩充内存指 1M 寻址范围之外的内存。在扩充内存规范（XMS）中，扩充内存也指高端

存储区 HMA 和上位存储块 UMB。

UMB 指在 DOS 内存 640K 边界和 1M 边界之间的内存。在 DOS 5.0 以前，程序员只有通过 XMS 驱动程序才可使用这一区域。从 DOS 5.0 开始可以通过 DOS 内存服务来访问 UMB，实际上 DOS 内存服务代为访问了 XMS 驱动程序。

HMA 的存在比较特殊。当 CPU 处于实模式并且第 21 条地址线（A20 线）被激活的状态时，CPU 就可以访问一块 65520 字节的内存，这块内存就叫做 HMA。HMA 的存在与 CPU 的寻址方式有关。CPU 根据一段地址：偏移对寻址时，首先将段地址乘以 16，再加上偏移量，形成物理地址。如果此值超过 20 位，则截去其高位，使物理地址在 000000H—0FFFFFH 之间。如果 A20 线不激活，地址 0FFFF：0010H 就是物理地址 000000H；若 A20 线激活，0FFFF：0010H 就是物理地址 010000：0000H，这样就有了额外的 65520 字节的内容。也就是说地址 0FFFF：0010H—0FFFF：0FFFFH 通常映象到物理地址 000000H—00FFEFH，但当 A20 线被激活后，映象到的物理地址范围就变成 0100000H 到 010FFEFH。

XMS 驱动程序提供了五组功能：驱动程序信息、HMA 管理、A20 线管理、扩充内存管理和上位存储区管理。另外的两个功能是检查 XMS 驱动程序是否存在和 XMS 驱动程序控制功能的地址。

3.5.2 判定扩充内存是否可用

要判定 XMS 程序是否存在，可执行如下代段：

【程序 3.9】

```
;
;
mov   AX，04300H
int   02FH
cmp   AL，080H
jne   XMS__NotPresent
;
; XMS is present
;
```

如果存在，还需取 XMS 驱动程序控制功能的地址，可以用如下的代码完成此功能：

【程序 3.10】

```
code segment
;
;
;
mov   AX，04310H
int   02FH
mov   word ptr [XMS__Control]，BX
mov   word ptr [XMS__Control]，ES
;
;
;
code ends
data segment
;
;
```

```
;
XMS_Control     dd (?)
;
;
;
data ends
```

3.5.3 使用扩充内存

程序 3.10 示例了扩充内存的基本操作，此程序完成如下功能：

- 测试扩充内存是否存在
- 取 XMS 版本号，检查 HMA 是否存在，并且取扩充内存最大自由块的大小
- 分配一扩充内存块
- 向扩充内存块写数据
- 从扩充内存块读出数据
- 释放扩充内存块
- 分配上位存储块
- 释放上位存储块

以上是扩充内存的基本操作，用户可以使用这类技术把数据放到扩充内存。

【程序 3.10】

```
{ XMSTEST.PAS          Compiled by Turbo Pascal 6.0          }
{ Copyright (c) 1992.8 by Mr. Lei                            }

Program XMS_demo;
{$X+}
uses XMS;

const
  MaxErrMsg = 27;
  ErrMsg : array [1..MaxErrMsg] of string = (
    #$00'Failure',
    #$01'Success',
    #$80'Function not implemented',
    #$81'VDISK device detected',
    #$82'A20 error occurred',
    #$90'HMA not exist',
    #$91'HMA already in use',
    #$92'No enough bytes to allocate HMA',
    #$93'HMA not allocation',
    #$94'A20 line still enable',
    #$A0'All extended memory is allocated',
    #$A1'All available extended memory handles in use',
    #$A2'Invalid handle',
    #$A3'Invalid source handle',
    #$A4'Invalid source offset',
    #$A5'Invalid destination handle',
    #$A6'Invalid destination offset',
    #$A7'Length is invalid',
    #$A8'Move has invalid overlap',
    #$A9'Parity error occurred',
    #$AA'Block not locked',
    #$AB'Block is locked',
    #$AC'Lock''s lock count overflowed',
    #$AD'Lock failed',
```

```
    # $B0'Smaller UMB available',
    # $B1'No UMBS available',
    # $B2'Invalid UMB segment number');
{*****************************************************}
const
  hexChars : array [0..$F] of Char =
    '0123456789ABCDEF';
Function hexb (b : byte) : string;
Begin
  hexb := hexChars [b shr 4] + hexChars [b and $F];
End;
Function hb (b : byte) : string;
var
  h : string;
Begin
  if b shr 4 = 0 then
    h := ''
  else h := hexChars [b shr 4];
  hb := h + hexChars [b and $F];
End;
Function hexw (w : Word) : string;
Begin
  hexw := hexChars [Hi (w) shr 4] +
          hexChars [Hi (w) and $F] +
          hexChars [Lo (w) shr 4] +
          hexChars [Lo (w) and $F];
End;
Function hexl (l : longint) : string;
Begin
  hexl := hexw (l shr 16) + hexw (l);
End;
{*****************************************************}
procedure ErrorHandle;
var
  I : Byte;
  Str : String;
begin
  Write ('Error, ');
  I := 1;
  While (ErrorStatus <> ord (ErrMsg [I, 1])) and (I <= MaxErrMsg) do
    Inc (I);
  if I > MaxErrMsg then
    Writeln ('Unknown error message ! ')
  else begin
    Str := ErrMsg [I];
    Delete (Str, 1, 1);
    Writeln (Str);
  end;
end; { ErrorHandle }

procedure Init;
var
  S : XMS_status;
begin
  Write ('XMS DEMO ');
  Writeln ('Copyright (c) 1992.9 by Yellow Rose Software Co.');
  Writeln;
  if not xms_test then
```

```
  begin
    Writeln ('Sorry, XMS not found! ');
    Halt (255);
  end;
  XMS_stat (S);
  Write ('XMS version ', hb (Hi (S.version)), '.', hexb (Lo (S.version)) );
  Writeln (' (Revision ', hb (Hi (S.revision)), '.', hexb (Lo (S.revision)), ') ');
  if S.HMA_exist then
    Writeln ('High Memory Area existence ');
  Writeln;
end;

procedure Test_HMA;
var
  P : Pointer;
  A20_status, old_A20_status : integer;
begin
  Writeln ('Test HMA function ... ');
  Write ('HMA_alloc func -- ');
  if HMA_alloc ($FFF0) then
  begin
    Writeln ('HMA allocated to current process');
    Write ('A20_stat func -- ');
    A20_stat;
    if ErrorStatus = 1 then begin
      Writeln ('A20 Enable ');
      A20_status := 1              { A20 Enable }
    end
    else if ErrorStatus = 0 then begin
      Writeln ('A20 Disable ');
      A20_status := 0;             { A20 Disable }
    end
    else A20_status := -1;
    old_A20_status := A20_status;
    if A20_status = 0 then begin { Disable }
      Write ('Alter_A20 func -- ');
      if Alter_A20 (GlobalEnableA20) then
      begin
        Writeln ('Enable success');
        A20_status := 1
      end
      else ErrorHandle;
    end;
    if A20_status = 1 then begin { Enable }
      Write ('* * * * * * * * * * * * * * * ');
      P := Ptr ($FFFF, $10);
      FillChar (P^, $FFF0, ' ');
      if Char (P^) <> ' ' then
        Writeln ('Error transferring data to/from HMA')
      else Writeln ('Success transferring data to/from HMA');
    end;
    if old_A20_status = 0 then
      if not Alter_A20 (GlobalDisableA20) then begin
        Write ('', 18);
        ErrorHandle;
      end;
    Write ('HMA_free func -- ');
    if HMA_free then
      Writeln ('HMA successfully freed')
```

```
    else ErrorHandle;
  end
  else ErrorHandle;
end;

procedure Test_UMB;
var
  P : Pointer;
  Ch : Byte;
  PSize, SegAddr : word;
begin
  Writeln ('Test UMB function ... ');
  PSize := $FFFF;
  UMB_alloc (PSize, SegAddr);          { Return largest available UMB }
  Write ('UMB_Alloc func -- ');
  if not UMB_alloc (PSize, SegAddr) then
    ErrorHandle
  else begin
    Writeln (LongInt (PSize) * 16, ' bytes allocated at ',
      hexw (SegAddr), ':0000');
    Write ('* * * * * * * * * * * * * * * * ');
    P := Ptr (SegAddr, 0);
    Randomize;
    Ch := Random (255);
    if LongInt (PSize) * 16 > $FFFF then
      FillChar (P^, $FFFF, Ch)
    else FillChar (P^, LongInt (PSize) * 16, Ch);
    if Ch = Byte (P^) then
      Writeln ('Access UMB success (fill number ', ch, ') ')
    else Writeln ('Access UMB failure ! ');
  end;
  Write ('UMB_Free func -- ');
  if UMB_free (SegAddr) then
    Writeln ('UMB successfully freed. ')
  else ErrorHandle;
end;      { Test_UMB }

procedure Test_EMB;
var
  EMB : EMBstruc;
  M : XMS_mem_stat;
  MyAddr : LongInt;
  Handle, I : word;
  A, B : array [0..1023] of byte;
  LockCount, NumFreeHandle : Byte;
  function ToEMB (P : Pointer; Count : LongInt; Handle : Word) : Boolean;
  var
    EMB : EMBstruc;
    I, J : LongInt;
begin
    I := Seg (P^);
    J := Ofs (P^);
    EMB.Count := Count;
    EMB.SourceHandle := 0;
    EMB.SourceOfs := I shl 16 + J;
    EMB.DestinHandle := Handle;
    EMB.DestinOfs := 0;
    ToEMB := XMS_move (EMB)
  end;
  function FromEMB (P : Pointer; Count : LongInt; Handle : Word) : Boolean;
```

```
  var
    EMB : EMBstruc;
    I, J : LongInt;
  begin
    EMB.count := Count;
    EMB.SourceHandle := Handle;
    EMB.SourceOfs := 0;
    EMB.DestinHandle := 0;
    I := Seg (P^);
    J := Ofs (P^);
    EMB.DestinOfs := I shl 16 + J;
    FromEMB := XMS_move (EMB)
  end;
begin
  Writeln ('Test EMB function ... ');
  Write ('XMS_avail func -- ');
  if XMS_avail (M) then begin
    Writeln;
    Writeln ('  Largest block of XMS memory available = ',
      M.LargestBlock, 'K bytes');
    Writeln ('  Total free memory = ', M.TotalFreeMemory, 'K bytes');
  end
  else begin
    ErrorHandle;
    Exit;
  end;
  Write ('XMS_alloc func -- ');
  if XMS_alloc (M.LargestBlock, Handle) then
    Writeln ('Memory allocated; Handle = ', hexw (Handle), 'H')
  else ErrorHandle;
  Write ('XMS_realloc -- ');
  if XMS_realloc (M.LargestBlock div 2, Handle) then
    Writeln ('Block successfully reallocated in half')
  else ErrorHandle;
  Write ('XMS_move func -- ');
  A [0] := $55;
  if not ToEMB (@A, 1024, Handle) then
    ErrorHandle
  else begin
    Writeln ('Data successfully written to XMS');
    A [0] := $AA;
    if not FromEMB (@A, 1024, Handle) then
      ErrorHandle
    else begin
      Writeln (''#18, 'Data successfully read from XMS ');
      I := 0;
      While (I < 16) and (A [I] = B [I]) do Inc (I);
      if (I < 16) and (A [I] <> B [I]) then
        if (A [I] <> $55) then
          Writeln (''#18,
            hexb (A [I]), ' ', hexb (B [I]), ' ',
            'Data read from XMS verified error')
      else Writeln (''#18, 'Data read from XMS verified ok ');
    end;
  end;
  Write ('XMS_Lock func -- ');
  if XMS_Lock (Handle, MyAddr) then
    Writeln ('Block locked; linear address = ', hexl (MyAddr))
  else ErrorHandle;
```

```
  Write ('XMS_free func -- ');
  if XMS_free (Handle) then
    Writeln ('XMS memory freed ')
  else ErrorHandle;
  Write ('XMS_bstat func -- ');
  if XMS_bstat (Handle, LockCount, NumFreeHandle) then
    Writeln ('Lock count = ', LockCount,
      '; Number of free handle = ', NumFreeHandle)
  else ErrorHandle;
  Write ('XMS_Lock func -- ');
  if XMS_Lock (Handle, MyAddr) then
    Writeln ('Block locked; linear address = ', hexl (MyAddr))
  else ErrorHandle;
  Write ('XMS_bstat func -- ');
  if XMS_bstat (Handle, LockCount, NumFreeHandle) then
    Writeln ('Lock count = ', LockCount,
      '; Number of free handle = ', NumFreeHandle)
  else ErrorHandle;
  Write ('XMS_unlock -- ');
  if XMS_unlock (Handle) then
    Writeln ('Block unlocked')
  else ErrorHandle;
  Write ('XMS_bstat func -- ');
  if XMS_bstat (Handle, LockCount, NumFreeHandle) then
    Writeln ('Lock count = ', LockCount,
      '; Number of free handle = ', NumFreeHandle)
  else ErrorHandle;
  Write ('XMS_unlock -- ');
  if XMS_unlock (Handle) then
    Writeln ('Block unlocked')
  else ErrorHandle;
  Write ('XMS_free func -- ');
  if XMS_free (Handle) then
    Writeln ('XMS memory freed ')
  else ErrorHandle;
end;

begin { MAIN }
  Init;
  Test_HMA;
  Writeln;
  readln;
  Test_UMB;
  Writeln;
  Test_EMB;
  readln;
end.
```

3.6 XMS 使用示范

本程序请读者了解了 XMS 规范和使用方法后自行分析。

【程序3.12】

```
{XMS.PAS Compiled by Turbo Pascal 6.0}    {written by Mr. Lei    1992.10}
```

```
unit XMS;

interface

type
  XMS_status = record
    version, revision: word;
    HMA_exist          :Boolean;
  end;

  XMS_mem_stat = record
    LargestBlock, TotalFreeMemory: Word;
  end;

  EMBstruc = record
    Count:          LongInt;    { 32-bit number of bytes to be transferred      }
    SourceHandle:   word;       { Source block handle                           }
    SourceOfs:      LongInt;    { 32-bit source offset                          }
    DestinHandle:   word;       { Destination block handle                      }
    DestinOfs:      LongInt;    { 32-bit destination offset                     }
     {注意: 如果使用常规内存, 置 Handle 为零, 这里的 Offset 是带段地址的偏移    }
  end;

var
  ErrorStatus: Byte;

Const
   { Define about A20 }
  GlobalEnableA20 = 00;
  GlobalDisableA20 = 01;
  LocalEnableA20 = 02;
  LocalDisableA20 = 03;

function XMS_test: boolean;

procedure XMS_stat (var stat: XMS_status);
function XMS_avail (var MemStat: XMS_mem_stat) : Boolean;
function XMS_alloc (KSize: Word; var Handle: Word) : Boolean;
function XMS_realloc (KSize, Handle: Word) : Boolean;
function XMS_free (Handle: word) : Boolean;
function XMS_Lock (Handle: Word; var MyAddr: LongInt) : Boolean;
function XMS_unlock (Handle: word) : Boolean;
function XMS_hsta: (Handle: word;
  var LockCount, NumFreehandle: Byte) : Boolean;
function XMS_move (var EMB: EMBstruc) : Boolean;

function HMA_alloc (Size: word) : Boolean;
function HMA_free: Boolean;
function Alter_A20 (Func: Byte) : Boolean;
function A20_stat: Boolean;

function UMB_alloc (var PSize, SegAddr: word) : Boolean;
function UMB_free (SegAddr: Word) : Boolean;

implementation

uses DOS;

var
  XMS_control: Pointer;

function XMS_test: boolean;
var
  Regs: Registers;
begin
  Regs.AX := $4300;
  Intr ($2F, Regs);
  if Regs.AL = $80 then begin
    Regs.AX := $4310;
    Intr ($2F, Regs);
```

```
    XMS_Control := Ptr (Regs.ES, Regs.BX);
    XMS_test := True;
  end
  else XMS_test := False;
end;
{ ********************** Extended Memory ******************** }
{ Get EMS version number }
procedure XMS_stat (var stat: XMS_status); assembler;
asm
  MOV     AH, 0
  CALL    XMS_Control
  LES     SI, stat
  MOV     XMS_status (ES: [SI]) .version, AX
  MOV     XMS_status (ES: [SI]) .revision, BX
  MOV     XMS_status (ES: [SI]) .HMA_exist, DL
end;
{ Checks the availability of extended memory }
function XMS_avail (var MemStat: XMS_mem_stat) : Boolean; assembler;
asm
  MOV     AH, 8
  CALL    XMS_control
  LES     SI, MemStat
  MOV     XMS_mem_stat (ES: [SI]) .LargestBlock, AX
  MOV     XMS_mem_stat (ES: [SI]) .TotalFreeMemory, DX
  MOV     ErrorStatus, BL
end;
{ Allocates a block of the given size from the pool of free extended memory }
function XMS_alloc (KSize: Word; var Handle: Word) : Boolean; assembler;
asm
  MOV     AH, 9
  MOV     DX, KSize
  CALL    XMS_Control
  LES     SI, Handle
  MOV     ES: [SI], DX
  MOV     ErrorStatus, BL
end;
function XMS_realloc (KSize, Handle: Word) : Boolean; assembler;
asm
  MOV     AH, 0FH
  MOV     BX, KSize
  MOV     DX, Handle
  CALL    XMS_control
  MOV     ErrorStatus, BL
end;
function XMS_Lock (Handle: Word; var MyAddr: LongInt) : Boolean; assembler;
asm
  MOV     AH, 0CH
  MOV     DX, Handle
  CALL    XMS_control
  LES     SI, MyAddr
  MOV     ES: [SI], BX
  MOV     ES: [SI+2], DX
  MOV     ErrorStatus, BL
end;
function XMS_unlock (Handle: word) : Boolean; assembler;
asm
  MOV     AH, 0DH
  MOV     DX, Handle
  CALL    XMS_control
```

```
  MOV     ErrorStatus, BL
end;

function XMS__bstat (Handle: word;
  var LockCount, NumFreeHandle: Byte) : Boolean; assembler;
asm
  MOV     AH, 0EH
  MOV     DX, Handle
  CALL    XMS__control
  LES     SI, LockCount
  MOV     ES: [SI], BH
  LES     SI, NumFreeHandle
  MOV     ES: [SI], BL
  MOV     ErrorStatus, BL
end;

{ Move Extended Memory Block }
function XMS__move (var EMB: EMBstruc) : Boolean; assembler;
asm
  MOV     AH, 0BH
  PUSH    DS
  POP     ES
  PUSH    DS
  LDS     SI, EMB
  CALL    ES: XMS__control
  POP     DS
  MOV     ErrorStatus, BL
end;

{ Free Extended Memory Block }
function XMS__free (Handle: word) : Boolean; assembler;
asm
  MOV     AH, 0AH
  MOV     DX, Handle
  CALL    XMS__control
  MOV     ErrorStatus, BL
end;

{ ********************** HMA ********************** }

{ Assigns the 65520-byte HMA to the caller }
function HMA__alloc (Size: word) : Boolean; assembler;
asm
  MOV     AH, 1
  MOV     DX, Size
  CALL    XMS__control
  MOV     ErrorStatus, BL
end;

{ Release the HMA }
function HMA__free: Boolean; assembler;
asm
  MOV     AH, 2
  CALL    XMS__control
  MOV     ErrorStatus, BL
end;

{ Alter A20 }
function Alter__A20 (Func: Byte) : Boolean; assembler;
asm
  MOV     AH, Func
  ADD     AH, 3
  CALL    XMS__control
  MOV     ErrorStatus, BL
end;
```

```
{ Checks the status of the A20 line }
function A20__stat: Boolean; assembler;
asm
  MOV    AH, 7
  CALL   XMS__control
  MOV    ErrorStatus, BL
end;

{ * * * * * * * * * * * * * * * * * * * * * * * * * UMB * * * * * * * * * * * * * * * * * * * * * * * * * }
{ Allocates an upper memory block }
function UMB__alloc (var PSize, SegAddr: word) : Boolean; assembler;
{ PSize is size of block in paragraphs
  Return:  if succeasful
             SegAddr Segment number of UMB
             PSize:  Actual size of allocatedb lock in paragraphs
           else
             PSize:  Size of largest available UMB in paragraphs
}
asm
  MOV    AH, 10H
  LES    SI, PSize
  MOV    DX, ES: [SI]
  CALL   XMS__control
  OR     AX, AX
  JZ     @1
  LES    SI, SegAddr
  MOV    ES: [SI], BX
@1:
  LES    SI, Psize
  MOV    ES: [SI], DX
  MOV    ErrorStatus, BL
end;

{ Releases an upper memory block }
function UMB__free (SegAddr: Word) : Boolean; assemblet;
asm
  MOV    AH, 11H
  MOV    DX, SegAddr
  CALL   XMS__control
  MOV    ErrorStatus, BL
end;

end.
```

·MEMO·

第四章 EMS 功能调用

4.1 EMS 功能调用索引

INT	功能	子功能	用途	页号
67	40		取管理程序状态	212
	41		取页面帧段地址	212
	42		取页数	213
	43		取句柄且分配内存	213
	44		内存映象/映象解除	214
	45		释放句柄和内存	215
	46		取 EMM 版本号	215
	47		保存页面映象	216
	48		恢复页面映象	216
	49		保留	217
	4A		保留	217
	4B		取句柄数	218
	4C		取某句柄的页面	218
	4D		取所有句柄的页面	219
	4E	00	取页映象寄存器	219
		01	置页映象寄存器	220
		02	取并设置页映象寄存器	220
		03	取页映象数组大小	221
	4F	00	取部分页面映象	221
		01	置部分页面映象	222
		02	取部分页面映象大小	222
	50	00	多个句柄页面映象/映象撤消（物理页面号方式）	223
		01	多个句柄页面映象/映象撤消（段地址方式）	224
	51		重新分配页面	224
	52	00	取句柄属性	225
		01	置句柄属性	226
		02	取属性支持能力	226
	53	00	取句柄名	227
		01	置句柄名	227
	54	00	取句柄目录	228
		01	查找指名句柄	228
		02	取句柄数	229
	55		改变页面映象且跳转	229
	56	00	改变页面映象且调用（物理页面号方式）	230
		01	改变页面映象且调用（段地址方式）	/

续表

INT	功能	子功能	用途	页号
67	56	02	取栈空间大小	232
	57	00	移动内存区	232
		01	交换内存区	233
	58	00	取可映象物理地址数组	234
		01	取可映象物理地址数组大小	235
	59	00	取扩展内存硬件信息	235
		01	取未分配的原始页数	236
	5A	00	分配标准页面	237
		01	分配原始页面	237
	5B	00	取替换映象寄存器	238
		01	置替换映象寄存器	238
		02	取替换映象寄存器组大小	239
		03	分配替换寄存器组	[illegible]
		04	释放替换寄存器组	[illegible]
		05	分配 DMA 寄存器组	[illegible]
		06	启用 DMA 寄存器组	241
		07	停用 DMA 寄存器组	242
		08	解除 DMA 寄存器组的分配	243
	5C		为热启动作硬件准备	243
	5D	00	启用 OS/E 功能	244
		01	停用 OS/E 功能	244
		02	把 OS/E 存取码返回给 EMS	[illegible]

EMS 功能调用使用说明与 DOS 部分类似。

4.2 EMS 功能调用详解

EMS 功能：INT 67H

通过页面切换技术，扩展内存可有多达 8M 字节的可访问内存

在这一技术中，定义一小块可转化为处理器物理寻址的附加存储区（EMS 内存）为一个存储单元。页面切换技术在计算机中运用了多年，它是一种对高速临时存储器的扩充访问方法；而对 EMS 来说，在一个定义的页面帧内，16K 存储单元为一页，这样的页可被转换为与正常存储器一样。

为了让程序访问 EMS 内存，装入了一个特殊的驱动程序。这样，访问 EMS 内存就如访问文件。用 INT 67H 功能 43H 打开 EMS 内存后，先要告知将要访问哪些存储页（用 INT 67H 功能 44H）；当一存储页可访问时，就可以读或写这一页了；访问完毕后，须用 INT 67H 功能 45H 关闭处理过程。

EMS 标准版本 3.0（更早版本未见公开使用）是由 Lotus、Intel 和 Microsoft 三家公司共

同制定的（因此 EMS 内存称为 LIM 存储器）。V3.2 增加了对多任务操作系统，像 Windows 和 DESQView 的支持；之后又制定了新的版本（4.0），在此新的版本中，可从扩展内存中运行程序并将信息存入。本详解据 LIM4.0 写成。版本在每一描述的顶行的“LIM 规范版本”中提示。

扩展内存的唯一缺点是，程序员必须跟踪信息存放在扩展内存什么地方。为避开内存工作方式，已在编程语言方面做了许多努力，这样，就可不必记住一个变量在哪一内存块。甚至在汇编语言中，如果使用一标号指向一变量，汇编程序可跟踪正在处理的那个变量。

但是对扩展内存，程序员必须记住目前哪个扩展内存块在页面帧里，哪一块包含数据。如所需扩展内存块不在当前页面帧时，必须将正确的块移入当前页面帧。使用多个数据段的汇编语言程序会碰到这样的问题，但高级语言程序没有这样的问题。

INT 67H	功能 40H 取管理程序状态	LIM 规范 V3.0

测试扩展内存硬件（如已被装入）是否可操作

调用寄存器： AH 40H

返回寄存器： AH 00H 无错误
80H EMS 软件内部错误
81H EMS 硬件故障
84H 未定义的功能

注释：

当确认扩展内存板已装上后，这一功能测试它是否可操作。

INT 67H	功能 41H 取页面帧段地址	LIM 规范 V3.0

取 EMS 使用的页面帧段地址

调用寄存器： AH 41H

返回寄存器： 如果成功
AH 00H
BX 页面帧段地址
如果失败
AH 80H EMS 软件内部错误
81H EMS 硬件故障
84H 未定义的功能

注释：

当测定板已装上且可操作后，需要测定其在内存中的位置。此板映象 4 个 16K 页（共 64K）到 640K（段 A000H）至 1M（段 10000H）内存区，在机器 1M 以内的内存映象中，大

部分已被视屏显示和ROM占用，但实际上还有大量白占空间在单系统中并没用上。当扩展板装上后，用板上开关选择EMS窗口地址，但必须谨慎选择出未作它用的存储器段地址。

返回段地址是第一个16K页的基地址，所有其他页是相对此地址的偏移地址。

INT 67H	功能42H 取页数	LIM规范V3.0

取系统EMS内存总页数和可用页数

调用寄存器： AH 42H

返回寄存器： 如果成功

AH 00H

BX 没有分配的页数

DX 系统总页数

如果失败

AH 80H EMS软件内部错误

81H EMS硬件故障

84H 未定义的功能

注释：

使用这一功能测定是否有足够可用的空间，只要调用一次，就可得知共有多少内存可用。举例来说，不管DX返回的总页数有多少，假如BX等于0，所有扩展内存已被分配，已无内存可作它用。

注意：假如/X（EMS）开关在CONFIG.SYS命令中被使用，在DOS4.X下运行这一功能可能返回不精确结果。因为操作系统可能占用了EMS中的页，而并不将其标记为已分配过的，这样做是为了保护它而不至被一般程序侵占。假如被占页碰巧被磁盘缓冲区使用，且其包含硬盘部分FAT表，当一些其它程序使用它时，将发生数据丢失。因此，应避免使用DOS V4的/X开关。

INT 67H	功能43H 取句柄且分配内存	LIM规范V3.0

打开EMS句柄，且为进程分配指定的页数

调用寄存器： AH 43H

BX 将要分配的逻辑页数

返回寄存器： 如果成功

AH 00H

DX 句柄（0001H到00FEH）

如果失败

AH 80H EMS软件内部错误

81H EMS 硬件故障
84H 未定义的功能
85H 无更多的可用句柄
87H 要求分配页多于物理可使用页；无页面分配
88H 指定逻辑页多于目前可用页；无页面分配
89H 零页面请求

注释：

这一功能如文件打开一样取得 EMS 内存。虽然与普通文件打开调用功能不同，这一功能确实向内存提供了一个类似文件的接口。

这一调用返回的句柄用于所有对扩展板的访问。这一调用将指定 EMS 内存页数与句柄联接，这些页面由功能 44H 控制。

在操作过程中，单一进程可分得多个 EMS 句柄。但进程必须用功能调用 45H 正确关闭每一句柄；否则，赋予这一句柄的内存将完全消失而不能使用，除非系统重启。

0000H 句柄保留，用于操作系统。用此功能调用绝不会返回这一句柄。

INT 67H	功能 44H	LIM 规范 V3.0
	内存映象/映象解除	

将赋给了句柄的一个 EMS 页映象到调用进程页面帧的四个物理页之一，也可置其逻辑页号为 FFFFH，解除一页的映象，使其不能被访问

调用寄存器： AH 44H
AL 物理页号（0—3）
BX 逻辑页号（以 0 为基；FFFFH 是解除物理页映象）
DX 句柄

返回寄存器： 如果成功
AH 00H
如果失败
AH 80H EMS 软件内部错误
81H EMS 硬件故障
83H 无效句柄
84H 未定义的功能
8AH 逻辑页没赋予句柄
8BH 物理页号无效

注释：

逻辑页是通过功能 43H 赋予句柄的 EMS 内存页。逻辑页编号为 0—n-1，n 为请求页数。特殊逻辑页号 FFFFH 用于解除映象。

在 EMS 板上，页不能被程序寻址，因为它超出了计算机物理寻址空间。这一功能映象一逻辑页到计算机物理寻址空间，这样，可用内存指令操作页面区位信息。从一个板上可映象多达 4 个逻辑页（号 0—3）。

INT 67H	功能 45H 释放句柄和内存	LIM 规范 V3.0

关闭指定句柄并返回其他进程使用的内存

调用寄存器： AH 45H
DX EMS 句柄

返回寄存器： 如果成功
AH 00H
如果失败
AH 80H EMS 软件内部错误
81H EMS 硬件故障
83H 无效句柄
84H 未定义的功能
86H 保存或恢复映象上下文出错

注释：

释放 EMS 内存等价于文件关闭操作，这一简单功能特别重要。因为 EMS 内存管理程序作为操作系统的附加驱动程序，当程序结束时，操作系统将不代为关闭一个打开的 EMS 句柄。当程序终止前，必须关闭所有分配给它的 EMS 句柄，否则，被分配的内存将继续被占用，不能被其他程序使用。如果这样，释放内存的唯一方法是重启计算机。

柄 执行关闭操作后，分配给句 的内存返回到可公用 EMS 内存。如果这一功能没有完全成功，内存不会返回公用区，必须接着试下去。

大多数程序只是关闭文件，而没有检查功能调用是否成功。这对文件并没有什么问题（指句柄文件），因为 DOS 在程序终止后关闭文件。但是对 EMS 内存，必须校验出口代码，如果功能失败，须再试。

INT 67H	功能 46H 取 EMM 版本号	LIM 规范 V3.0

返回软件版本号

调用寄存器： AH 46H

返回寄存器： 如果成功
AH 00H
AL EMM 版本号
如果失败
AH 80H EMS 软件内部错误
81H EMS 硬件故障
84H 未定义的功能

注释：

这一功能返回（以 BCD 码）EMM（扩展内存管理）软件版本号。AL 寄存器高四位代表主版本号（十进制小数点的左部分）；低四位代表次版本号（十进制小数点的右部分）。例如，V3.2 在 AL 中返回 32H，V4.0 返回 40H。一个程序应不断准备接受比设定值大的版本号；EMS 规范要求驱动程序向上兼容。

INT 67H	功能 47H	LIM 规范 V3.0
	保存页面映象	

保存 EMS 硬件映象当前的状态

调用寄存器： AH 47H

DX 句柄

返回寄存器： 如果成功

AH 00H

如果失败

AH 80H EMS 软件内部错误

81H EMS 硬件故障

83H 无效句柄

84H 未定义的功能

8CH 页映象硬件状态保存区满

8DH 映象上下文保存失败。因为它已与指定句柄联接

注释：

假如正写一个驻留程序（TSR）、一个中断服务例程或设备驱动程序，而其中使用了 EMS 内存，在做任何 EMS 操作前，必须保存映象硬件状态，否则，别的程序会干扰 EMS 内存使用。这一功能保存映象硬件状态。INT 67H 功能 48H 用于恢复硬件映象。

这一功能只保存由 LIM 规范 V3.x 定义的 64K 页面帧的映象寄存器状态。若应用程序使用 LIM V3.x 页面帧以外的可映象内存区，应使用功能 4EH 和 4FH 来保存和恢复映象寄存器状态。

INT 67H	功能 48H	LIM 规范 V3.0
	恢复页面映象	

恢复与指定文件句柄相联的 EMS 硬件映象

调用寄存器： AH 48H

DX 句柄

返回寄存器： 如果成功

AH 00H

如果失败

AH	80H	EMS 软件内部错误
	81H	EMS 硬件故障
	83H	无效句柄
	84H	未定义的功能
	8EH	恢复失败，保存区无与句柄相应的上下文

注释：

假如正写一个驻留程序（TSR）、一个中断服务例程或设备驱动程序，当程序完成时，必须恢复映象硬件状态，否则，将导致不可预测的结果。例如，当软件运行时，一个使用 EMS 内存的应用程序可能正在运行，而且希望 EMS 映象寄存器不被改变。这一功能恢复映象硬件状态。INT 67H 功能 47H 用于保存硬件映象状态。

这一功能只恢复由 LIM 规范 V3.x 定义的 64K 页面帧的映象寄存器状态。若应用程序使用 LIM V3.x 页面帧以外的可映象内存区，应使用功能 4EH 和 4FH 来保存和恢复映象寄存器状态。

INT 67H	功能 49H 保留	LIM 规范 V3.0

这一功能在 EMS 标准 V3.2 以上没有定义

注释：

在 LIM/EMS 原先的版本中，这一功能用来从 EMS 硬件中取回页面映象寄存器 I/O 数组。这一功能现在保留，新的软件不用它。

功能 49H 和 4AH 已从规范中去掉，因为他们针对确定的 Intel EMS 板硬件，不能保证对其他公司板子可行。使用 LIM V4.0 新增功能的程序不能使用上述两个功能。使用了这些功能的程序不能与 IBM 微通道结构（IBM Micro Channel Architecture）兼容，对其他硬件可能可行。

INT 67H	功能 4AH 保留	LIM 规范 V3.0

这一功能在 EMS 标准 V3.2 以上没有定义

注释：

在 LIM/EMS 原先的版本中，这一功能用来从 EMS 硬件中取回逻辑-物理页面转换数组。这一功能现在保留，新的软件不用它。

功能 49H 和 4AH 已从规范中去掉，因为它们针对确定的 Intel EMS 板硬件，不能保证对其他公司板子可行。使用 LIM V4.0 新增功能的程序必须不使用上述两个功能的任一个。使用了这些功能的程序不能与 IBM 微通道结构兼容，对其他硬件可能可行。

INT 67H	功能 4BH 取句柄数	LIM 规范 V3.0

返回所用的 EMS 句柄数

调用寄存器： AH 4BH

返回寄存器： 如果成功

AH 00H

BX 现用的 EMS 句柄数（0－255）

如果失败

AH 80H EMS 软件内部错误

81H EMS 硬件故障

84H 未定义的功能

注释：

这一功能告知在任一给定时间有多少句柄正被使用。如 BX＝0，扩展内存没被使用。BX 可在 0－255（EMS 句柄最大数）之间。

用户程序对这个数字的解释并不像看起来那么清晰。现用的句柄不一定与目前使用扩展内存的程序数相同，因为不能排除一个程序使用多个 EMS 句柄去访问扩展内存。事实上，某些程序使用多个句柄可简化管理数据的操作。最好用这一功能测定所剩的句柄数，即 255 减去 BX。程序可用它迅速确定 EMS 剩下的句柄是否够用。

INT 67H	功能 4CH 取某句柄占有的页面	LIM 规范 V3.0

测定与指定句柄相联的页面数

调用寄存器： AH 4CH

DX 句柄

返回寄存器： 如果成功

AH 00H

BX 逻辑页数（1－2048）

如果失败

AH 80H EMS 软件内部错误

81H EMS 硬件故障

83H 无效句柄

84H 未定义的功能

注释：

任一句柄可访问 1－2048（LIM V4 前为 512）页扩展内存。因为每一页代表 16K 内存，一个句柄可寻址 16K 到 32M（LIM V4 前为 8M）。这一功能不会返回 0 页，因为扩展内存管理程序将至少一页赋予一句柄。

INT 67H	功能 4DH	LIM 规范 V3.0
	取所有句柄的页面	

返回句柄和所有句柄的逻辑页数

调用寄存器： AH 4DH

ES：DI 指向保存信息数组的指针（可能需 1020 字节大小）

返回寄存器： 如果成功

AH 00H

BX 现用的 EMS 句柄数（1—255）

如果失败

AH 80H EMS 软件内部错误

81H EMS 硬件故障

84H 未定义的功能

注释：

这一功能返回一数组，其值显示当前句柄和赋予每一句柄的页面数。EMS 句柄表由一长度不定的双字登记项组成，字的安排为：字 0 为 EMS 句柄，字 1 为页数。

决定这一表所需的内存的公式可表达为 4 * BX。因最大可能有 255 个句柄，4 * 255（或 1020 字节）是这一表的最大的地址分配，BX 是有效表项数，在 LIM V4 系统中，BX 不能小于 1，因为特定的 OS 句柄总是保留着的。在早期的版本中，如果 BX 为 0，则 EMS 管理程序处于空闲状态。

当表被传送到内存中时，注意不要由于放置表（由 ES：DI 指向）而引起段重叠，否则会出错。

INT 67H	功能 4EH 子功能 00H	LIM 规范 V3.2
	取页映象寄存器	

取 EMS 页映象寄存器到一局部数组而不使用句柄

调用寄存器： AH 4EH

AL 00H

ES：DI 指向接收信息数组的指针

返回寄存器： 如果成功

AH 00H

如果失败

AH 80H EMS 软件内部错误

81H EMS 硬件故障

84H 未定义的功能

8FH 子功能参数无效

注释：

这一功能在 EMM 软件 V3.2 中加入，以支持像 Windows 或 DESQView 这样的多任务系统。程序用功能 4EH 可直接访问被 EMS 板内部使用的页映象信息及那些极端依赖硬件的信息。上述数组保持页映象寄存器和另外的控制信息。多任务系统是唯一使用这类信息的程序——没有这些信息它们就无法有效工作，单个程序不能使用这类信息。

当表被传送到内存中时，注意不要由于放置表（由 ES：DI 指向）而引起段重叠，否则会出错。

INT 67H	功能 4EH 子功能 01H 置页映象寄存器	LIM 规范 V3.2

不使用句柄而用局部数组设置 EMS 页映象寄存器

调用寄存器： AH 4EH
AL 01H
DS：SI 指向设置信息数组的指针

返回寄存器： 如果成功
AII 00II
如果失败
AH 80H EMS 软件内部错误
81H EMS 硬件故障
84H 未定义的功能
8FH 子功能参数无效

注释：

这一功能在 EMM 软件 V3.2 中加入，以支持像 Windows 或 DESQView 这样的多任务系统。程序用功能 4EH 可直接访问被 EMS 板内部使用的页映象信息及那些极端依赖于硬件的信息。上述数组保持页映象寄存器和另外的控制信息。

INT 67H	功能 4EH 子功能 02H 取、置页映象寄存器	LIM 规范 V3.2

取 EMS 页面映象寄存器到一局部数组，同时用另一局部数组置数，而不使用句柄

调用寄存器： AH 4EH
AL 02H
DS：SI 指向设置信息数组的指针
ES：DI 指向接收信息数组的指针

返回寄存器： 如果成功
AH 00H
如果失败

AH 80H EMS软件内部错误
81H EMS硬件故障
84H 未定义的功能
8FH 子功能参数无效

注释：

这一功能在EMM软件V3.2中加入，以支持像Windows或DESQView这样的多任务系统。程序用功能4EH可直接访问被EMS板内部使用的页映象信息及那些极端依赖于硬件的信息。上述数组保持页映象寄存器和另外的控制信息。

INT 67H	功能 4EH 子功能 03H 取页映象数组大小	LIM 规范 V3.2

取存放EMS页面映象寄存器所需局部数组大小

调用寄存器： AH 4EH
AL 03H

返回寄存器： 如果成功
AH 00H
AL 页映象数组字节
如果失败
AH 80H EMS软件内部错误
81H EMS硬件故障
84H 未定义的功能
8FH 子功能参数无效

注释：

这一功能在EMM软件V3.2中加入，以支持像Windows或DESQView这样的多任务系统。程序用功能4EH可直接访问被EMS板内部使用的页映象信息及那些极端依赖于硬件的信息。上述数组保持页映象寄存器和另外的控制信息。

或者在分别使用子功能00H和01H、或者在使用子功能02H交换页面映象寄存器内容之前，用子功能03H来取所需数组大小。

INT 67H	功能 4FH 子功能 00H 取部分页面映象	LIM 规范 V4.0

保存指定的可映象内存区的部分映象上下文；比处理所有可映象内存的功能4EH快

调用寄存器： AH 4FH
AL 00H
DS：SI 指向页面列表
ES：DI 指向目的数组

返回寄存器： 如果成功

AH 00H

如果失败

AH 80H EMS 软件内部错误

81H EMS 硬件故障

84H 未定义的功能

8BH 指定的某一段不可映象

8FH 子功能参数无效

A3H 部分页映象结构不正确

注释：

这一功能需传入两个指针，一个指向被映象的段地址表，另一个指向存放映象的数组。页面表的第一个字存放表中页面数，接下来的许多字都是将被映象的页面号。

目的数组至少必须包含子功能 03H（取部分页映象大小）返回的字节数。它的格式 LIM V4 没规定，因为这一数组的唯一用途是向子功能 01H 提供输入（置部分页映象）。

INT 67H	功能 4FH 子功能 01H	LIM 规范 V4.0
	置部分页面映象	

恢复被子功能 00H 保存的部分映象上下文；比处理所有可映象内存的功能 4EH 快

调用寄存器： AH 4FH

AL 01H

DS：SI 指向源数组

返回寄存器： 如果成功

AH 00H

如果失败

AH 80H EMS 软件内部错误

81H EMS 硬件故障

84H 未定义的功能

8FH 子功能参数无效

注释：

该功能需传入一数组指针，这一数组是由子功能 00H 保存的映象。它的格式 LIMV4 没规定，因为这一数组的唯一用途是为这一功能提供输入，然后子功能恢复这一数组中保存的每页的映象。

INT 67H	功能 4FH 子功能 02H	LIM 规范 V4.0
	取部分页面映象大小	

返回用子功能 00H 保存页映象时所需数组字节数

调用寄存器： AH 4FH
AL 02H
BX 页面表中的页数

返回寄存器： 如果成功
AH 00H
AL 子功能00H和01H使用的数组所需的字节数
如果失败
AH 80H EMS软件内部错误
81H EMS硬件故障
84H 未定义的功能
8BH 指定的页面号超出了系统物理页面的范围
8FH 子功能参数无效

注释：

这一功能可得到进行部分页映象时所需数组大小。它应在使用页面映象功能前调用，以确认传给上述功能的数组大小是否合适。

INT 67H 功能50H 子功能00H LIM规范V4.0
多个句柄页面映象/映象撤消（物理页面号方式）

使用物理页面号方式映象逻辑页到物理页或撤消这一映象

调用寄存器： AH 50H
AL 00H
CX 要处理的页面数
DX EMM句柄
DS：SI 指向逻辑/物理页对应表

返回寄存器： 如果成功
AH 00H
如果失败
AH 80H EMS软件内部错误
81H EMS硬件故障
83H 不能找到指定的EMM句柄
84H 未定义的功能
8AH 逻辑页出界
8BH 某一段不可映象
8FH 子功能参数无效

注释：

作为LIM V3.2为句柄单页映象功能的扩展，本功能一次可处理多页。DS：SI指向的对应表由字对构成。字对表示要映象或映象要撤消的页面。每个字对的第一字是逻辑页号，如果此页映象要撤消，逻辑页号要设置为FFFFH；第二字是相应的物理页号。两个字都是以零为

基。

INT 67H	功能 50H 子功能 01H	LIM 规范 V4.0
	多个句柄页面映象/映象撤消（段地址方式）	

使用段地址方式映象逻辑页到物理页或撤消这一映象

调用寄存器： AH 50H
AL 01H
CX 要处理的页面数
DX EMM 句柄
DS：SI 指向逻辑页/段地址对应表

返回寄存器： 如果成功
AH 00H
如果失败
AH 80H EMS 软件内部错误
81H EMS 硬件故障
83H 不能找到指定的 EMM 句柄
84H 未定义的功能
8AH 逻辑页出界
8BH 某一段不可映象
8FH 子功能参数无效

注释：

这一功能可代替子功能 00H，其结果相同，但本功能有时容易使用些。DS：SI 指向的对应表由字对构成。字对表示要映象或映象要撤消的页面，每个字对的第一字是逻辑页号，如果此页映象要撤消，逻辑页号要设置为 FFFFH；第二字是映象页或映象要撤消页所在的段地址。逻辑页以 0 为基，而段地址则是由功能 58H（取可映象物理地址数组大小）获得的，而且必须严格对应。

INT 67H	功能 51H	LIM 规范 V4.0
	重新分配页面	

允许一应用程序增加或减少分配给 EMM 句柄的逻辑页数

调用寄存器： AH 51H
BX 期望逻辑页数
DX EMM 句柄

返回寄存器： 如果成功
AH 00H
BX 分配到的逻辑页数

如果失败

AH 80H EMS 软件内部错误
81H EMS 硬件故障
83H 不能找到指定的 EMM 句柄
84H 未定义的功能
87H 系统中无足够页
88H 无足够可用页

注释：

这一功能允许四种情况：新页数为 0、比原页数少、等于原页数、比原页数多。如为 0，句柄仍然打开，但其所有页面回到可用公共区。另外三种情况下，在此功能调用完毕后，全都是从以 0 为基的连续的页面序列中分配了页面。

句柄决定了被分配的逻辑页面类型。如果页面原来是调用功能 43H 或功能 5AH 子功能 00H 分配的，则是 16K 标准页；如果它们原来是调用功能 5AH 子功能 01H 分配的，它们是原始页面，大小由 EMS 软件定义。

INT 67H	功能 52H 子功能 00H 取句柄属性	LIM 规范 V4.0

返回句柄属性

调用寄存器： AH 52H
AL 00H
DX EMM 句柄

返回寄存器： 如果成功

AH 00H
AL 指定 EMM 句柄的属性
0 易失的（Volatile）
1 非易失的

如果失败

AH 80H EMS 软件内部错误
81H EMS 硬件故障
83H 不能找到指定的 EMM 句柄
84H 未定义的功能
8FH 子功能参数无效
91H 不支持的特征

注释：

这一功能是一可选项，在大多数系统中都没有，因为硬件在热启动操作后不保护内存内容。如果无此功能，AH 中返回 91H。

如有此功能，子功能 00H 在 AL 中返回指定句柄的当前属性。易失（AL=0）说明句柄内容在热启动操作后不能保存，非易失（AL=1）说明能保存。

INT 67H	功能 52H 子功能 01H 置句柄属性	LIM 规范 V4.0

置句柄属性

调用寄存器： AH 52H
AL 01H
BL 指定 EMM 句柄的新属性
0 易失的
1 非易失的
DX EMM 句柄

返回寄存器： 如果成功
AH 00H
如果失败
AH 80H EMS 软件内部错误
81H EMS 硬件故障
83H 不能找到指定的 EMM 句柄
84H 未定义的功能
8FH 子功能参数无效
90H 属性不为零也不为一
91H 不支持的特征

注释：

这一功能是一可选项，在大多数系统中都没有，因为硬件在热引导操作后不保护内存内容。如果无此功能，AH 中返回 91H。

如有此功能，子功能 00H 在 AL 中返回指定句柄的当前属性，易失（AL=0）说明句柄内容在热引导操作后不能保存。非易失（AL=1）说明能保存。

INT 67H	功能 52H 子功能 02H 取属性支持能力	LIM 规范 V4.0

返回系统属性支持能力

调用寄存器： AH 52H
AL 02H

返回寄存器： 如果成功
AH 00H
AL EMS 软件和硬件的属性权能
0 只支持易失性句柄
1 非易失性和易失性句柄都支持
如果失败

AH 80H EMS 软件内部错误
81H EMS 硬件故障
84H 未定义的功能
8FH 子功能参数无效

注释：

子功能 02H 在 AL 中返回 EMS 软件和硬件的属性支持能力。0 值表示支持易失性句柄，1 值表明易失性和非易失性句柄都支持。

INT 67H	功能 53H 子功能 00H	LIM 规范 V4.0
	取句柄名	

取指定的 EMM 句柄的 8 字符名

调用寄存器： AH 53H
AL 00H
DX EMM 句柄
ES：DI 指向接收句柄名的 8 字节缓冲区

返回寄存器： 如果成功
AH 00H
如果失败
AH 80H EMS 软件内部错误
81H EMS 硬件故障
83H 不能找到指定的 EMM 句柄
84H 未定义的功能
8FH 子功能参数无效

注释：

这一功能取指定的句柄名。名字长度总是 8 个字符，每个字符可以是任何 8 位值；它并不限于 ASCII 字符集。当句柄被分配或分配解除时，系统把句柄名初始化为 8 个 NUL (00H) 字符。根据定义，有上述空字符串的句柄没有名字。当一句柄被赋名时，8 个字节中至少有一个非 0，以便与无名句柄区别开来。

INT 67H	功能 53H 子功能 01H	LIM 规范 V4.0
	置句柄名	

置指定的 EMM 句柄的 8 字符名

调用寄存器： AH 53H
AL 01H
DX EMM 句柄
ES：DI 指向 8 字节句柄名

返回寄存器： 如果成功

AH 00H

如果失败

AH 80H EMS 软件内部错误

81H EMS 硬件故障

83H 不能找到指定的 EMM 句柄

84H 未定义的功能

8FH 子功能参数无效

A1H 与已存在的名重复

注释：

这一功能置指定句柄名。名字长度总是 8 个字符，每个字符可以是任何 8 位值；它并不限于 ASCII 字符集。当句柄被分配或分配解除时，系统把句柄名初始化为 8 个 NUL (00H) 字符。根据定义，有上述空字符串的句柄没有名字。当一句柄被赋名时，8 个字节中至少有一个非 0，以便与无名句柄区别开来。

INT 67H	功能 54H 子功能 00H	LIM 规范 V4.0
	取句柄目录	

局部复制所有打开的 EMM 句柄目录

调用寄存器： AH 54H

AL 00H

ES：DI 指向目的数组

返回寄存器： 如果成功

AH 00H

AL 数组中项数（同打开句柄数一样）

如果失败

AH 80H EMS 软件内部错误

81H EMS 硬件故障

84H 未定义的功能

8FH 子功能参数无效

注释：

子功能 00H 将所有现用句柄及它们的名字复制到一数组中（一个 EMM 句柄-句柄名目录复制品）。数组中每一项的格式是一包含 EMM 句柄的字，后跟包含名字的 8 个字节。数组的字节大小是功能被调用时打开的句柄数乘 10。

INT 67H	功能 54H 子功能 01H	LIM 规范 V4.0
	查找指名句柄	

对给定的句柄名返回句柄值

调用寄存器： AH 54H
AL 01H
DS：SI 指向句柄名

返回寄存器： 如果成功
AH 00H
DX 与名相联的EMM句柄
如果失败
AH 80H EMS软件内部错误
81H EMS硬件故障
84H 未定义的功能
8FH 子功能参数无效
A0H 没找到句柄
A1H 句柄无名

注释：

子功能00H查找指名的句柄-句柄名目录。如发现，返回句柄值。这一功能中的输入项DS：SI指向要查找的名字的8字节字符串。

INT 67H	功能54H 子功能02H 取句柄数	LIM规范V4.0

返回内存管理系统支持的句柄总数

调用寄存器： AH 54H
AL 02H

返回寄存器： 如果成功
AH 00H
BX 支持的句柄数
如果失败
AH 80H EMS软件内部错误
81H EMS硬件故障
84H 未定义的功能
8FH 子功能参数无效

注释：

子功能02H返回系统可分配的最大句柄数，包括保留的OS句柄0。这一功能用来决定支持子功能00H（取句柄目录）必需的最大数组大小。

INT 67H	功能55H 改变页面映象且跳转	LIM规范V4.0

永久性改变页面映象并转移控制

调用寄存器： AH 55H
AL 物理页/段选择数
0 使用物理页号
1 使用段地址
DX EMM 句柄
DS：SI 指向映象/跳转结构

返回寄存器： 如果成功
AH 00H
如果失败
AH 80H EMS 软件内部错误
81H EMS 硬件故障
83H 不能找到指定的 EMM 句柄
84H 未定义的功能
8AH 逻辑页出界
8BH 物理页出界
8FH 子功能参数无效

注释：

这一功能通过改变页面映象允许应用程序在 EMS 内存运行，然后传送控制到映象区。它与汇编语言 FAR JUMP 操作等价。使用这一功能前存在的内存映象上下文将丢失。

功能 55H 的输入项 DX 包含 EMM 句柄，而 DS：SI 指向一下列格式的结构：

字节偏移	域宽	含义
00H	双字	远指针，指向控制转移的地址
04H	字节	映象数组中字对数
05H	双字	指向映象数组的远指针

映象数组是一可变长度的字对集。字对将逻辑页映象到物理页号或段地址中，这决定于传入的 AL 值。它的结构如下：

字节偏移	域宽	含义
00H	字	被映象逻辑页数
04H	字	物理页号或段地址，取决于 AL 中的值

跳转后，所有不含所需参数的寄存器将保持原值，象标志条件一样。

INT 67H 功能 56H 改变页面映象且调用 **LIM 规范 V4.0**

暂时改变页面映象并转移控制，返回时恢复映象

调用寄存器： AH 56H
AL 物理页/段选择数
0 使用物理页号
1 使用段地址
DX EMM句柄
DS：SI 指向映象/跳转结构

返回寄存器： 如果成功
AH 00H
如果失败
AH 80H EMS软件内部错误
81H EMS硬件故障
83H 不能找到指定的EMM句柄
84H 未定义的功能
8AH 逻辑页出界
8BH 物理页出界
8FH 子功能参数无效

注释：

这一功能通过改变页面映象，允许应用程序在EMS内存运行，然后传送控制到映象区。它与汇编语言FAR CALL操作等价。使用这一功能前的内存映象上下文被保存。从调用过程返回时，通过RETF（RET FAR）操作来恢复它。

这一功能的输入项DX包含EMM句柄，DS：SI指向下列格式的结构：

字节偏移	域宽	含义
00H	双字	远指针，指向控制转移的地址
04H	字节	新映象数组中字对数
05H	双字	指向新的映象数组的远指针
09H	字节	老映象数组中的字对数
0AH	双字	指向老映象数组的远指针
0EH	4个字	保留，被EMS软件使用

映象数组是一可变长度的字对集。字对将逻辑页映象到物理页号或段地址中，这决定于传入AL中的值。新的映象数组中指定了CALL过程中将被使用的上下文，老的映象数组保存原来的上下文，以便能恢复。它们的结构如下：

字节偏移	域宽	含义
00H	字	被映象物理页数
04H	字	物理页号或段地址，取决于AL中的值

跳转后，所有不含所需参数的寄存器将保持原值，像标志条件一样。

INT 67H	功能 56H 子功能 02H 取栈空间大小	LIM 规范 V4.0

返回使用功能 56H 所需的附加栈空间字节数

调用寄存器： AH 56H
AL 02H

返回寄存器： 如果成功
AH 00H
BX 改变页面映象且调用子功能时所需栈空间字节数
如果失败
AH 80H EMS 软件内部错误
81H EMS 硬件故障
84H 未定义的功能
8FH 子功能参数无效

注释：

这一功能在 BX 中返回栈指针需增加的数，以便从栈中移走所有保存元素。这一功能要在改变页面映象且调用功能前使用，以便从后者返回时能校准 SP 的值。

INT 67H	功能 57H 子功能 00H 移动内存区	LIM 规范 V4.0

移动内存区

调用寄存器： AH 57H
AL 00H
DS：SI 指向移动参数块

返回寄存器： 如果成功
AH 00H 没遇到困难
92H 重叠（Overlay），源区被破坏
如果失败
AH 80H EMS 软件内部错误
81H EMS 硬件故障
83H 不能找到指定的 EMM 句柄
84H 未定义的功能
8AH 逻辑页出界
8FH 子功能参数无效
93H 对句柄来说内存区太大
94H 常规区和扩展区重叠
95H 逻辑页区距超出逻辑页范围

96H 长度超出1M字节
98H 源类型和目的类型未定义
A2H 1M中的环绕（Wraparound）发生，不传送任何数据

注释：

这一功能将一个内存区内容移动到另一内存区。移动可以是常规和扩展内存区的任何组合：常规到常规、常规到扩展、扩展到常规或扩展到扩展。

运用这一功能时不必修改页映象；在操作过程中当前上下文保持不变，移动区长度可从0到1M字节，但总应是16K的倍数。

如源和目的句柄同在常规或扩展内存区，源和目的区在移动前作重叠检查。如重叠存在，目的区将接收一个完整的源区拷贝，虽然源区在这一过程中可能被破坏。在这种情况下返回一状态码表明已发生重叠。

这一功能操作由DS：SI指向的参数块控制。块的格式如下：

字节偏移	域宽	含义
00H	双字	将要移动的字节数
04H	字节	源内存类型（0=常规，1=扩展，其他未定义）
05H	字	源句柄
07H	字	源初始偏移量
09H	字	源段或页： 如为常规类型，这是段地址； 如为扩展类型，这是逻辑页号
0BH	字节	目的内存类型（0=常规，1=扩展，其他未定义）
0CH	字	目的句柄
0EH	字	目的初始偏移量
10H	字	目的段或页： 如为常规类型，这是段地址； 如为扩展类型，这是逻辑页号

INT 67H	功能57H 子功能01H	LIM规范V4.0
	交换内存区	

交换两个内存区

调用寄存器： AH 57H
AL 01H
DS：SI 指向移动参数块

返回寄存器： 如果成功
AH 00H 没遇到问题
如果失败
AH 80H EMS软件内部错误
81H EMS硬件故障

83H 不能找到指定的 EMM 句柄
84H 未定义的功能
8AH 逻辑页出界
8FH 子功能参数无效
93H 对句柄来说内存区太大
94H 常规区和扩展区重叠
95H 逻辑页区距超出逻辑页范围
96H 长度超出 1M 字节
98H 源类型和目的类型未定义
A2H 1M 中的环绕发生，不传送任何数据

注释：

这一功能交换两个内存区内容。它们可以是常规和扩充内存区的任何组合：常规和常规、常规和扩展、扩展和常规或扩展和扩展。

运用这一功能时不必修改页映象；在操作过程中当前上下文将保持，移动区长度可从 0 到 1M 字节，但总应是 16K 的倍数。

如源和目的句柄同在常规或扩展内存区，源和目的区在移动前要作重叠检查。如重叠存在，则不交换且返回错误状态。

这一功能操作由 DS：SI 指向的参数块控制。块的格式如下：

字节偏移	域宽	含义
00H	双字	将要交换的字节数
04H	字节	源内存类型（0=常规，1=扩展，其他未定义）
05H	字	源句柄
07H	字	源初始偏移量
09H	字	源段或页： 如为常规类型，这是段地址； 如为扩展类型，这是逻辑页号
0BH	字节	目的内存类型（0=常规，1=扩展，其他未定义）
0CH	字	目的句柄
0EH	字	目的初始偏移量
10H	字	目的段或页： 如为常规类型，这是段地址； 如为扩展类型，这是逻辑页号

INT 67H　功能 58H　子功能 00H　LIM 规范 V4.0
取可映象物理地址数组

返回包含系统中每一个可映象物理页的段地址和物理页号的数组

调用寄存器：　AH　58H
　　AL　00H

ES：DI　指向目的数组

返回寄存器：　如果成功

AH　00H

CX　数组的项数

如果失败

AH　80H　EMS 软件内部错误

81H　EMS 硬件故障

84H　未定义的功能

8FH　子功能参数无效

注释：

这一功能用字对的方式填写用户提供的数组，这些字对是实际段地址和与之相关的物理页号。此数组按段排序，这并不意味着物理页号也是按序排列的。数组中单个字对的结构如下：

字节偏移	域宽	含义
00H	字	可映象物理的页段地址
02H	字	与页相联的逻辑页号

INT 67H　功能 58H　子功能 01H　LIM 规范 V4.0

取可映象物理地址数组大小

返回调用子功能 00H 时可返回的数组项数

调用寄存器：　AH　58H

AL　01H

返回寄存器：　如果成功

AH　00H

CX　数组的项数

如果失败

AH　80H　EMS 软件内部错误

81H　EMS 硬件故障

84H　未定义的功能

8FH　子功能参数无效

注释：

这一功能在 CX 中返回子功能 00H 能返回的数组项数，用 4 乘以返回值即可提供子功能 00H 目的数组所需字节数。

INT 67H	功能 59H 子功能 00H	LIM 规范 V4.0
	取扩展内存硬件信息	

返回硬件配置数组

调用寄存器： AH 59H
AL 00H
ES：DI 指向目的数组

返回寄存器： 如果成功
AH 00H
如果失败
AH 80H EMS 软件内部错误
81H EMS 硬件故障
84H 未定义的功能
8FH 子功能参数无效
A4H 功能调用被操作系统拒绝

注释：

这一功能向一 5 个字的数组填写系统硬件配置细节。ES：DI 指向的目的数组格式如下：

字节偏移	域宽	含义
00H	字	以节（16 字节）表示的原始可映象物理页的大小
02H	字	可用的替换映象寄存器组数
04H	字	保存映象上下文所需字节数
06H	字	能赋予 DMA 通道的寄存器组数
08H	字	如 DMA 寄存器组像功能 5BH 描述的那样工作时，为 0； 硬件只有 1 个 DMA 寄存器组时，为 1；LIM 标准板此值为 0

INT 67H	功能 59H 子功能 01H	LIM 规范 V4.0
	取未分配的原始页数	

返回未分配原始可映象页数和原始页面总数

调用寄存器： AH 59H
AL 01H

返回寄存器： 如果成功
AH 00H
BX 当前可用的原始页数
DX EMS 中原始页总数
如果失败
AH 80H EMS 软件内部错误

81H EMS 硬件故障
84H 未定义的功能
8FH 子功能参数无效

注释：

有些 EMS 硬件页的大小是标准的 16K 字节的整数倍，这种非标准页面大小称为原始页面。这一功能返回可用原始页面数。对使用标准页面大小的硬件，这一功能同功能 42H 完全相同。

INT 67H	功能 5AH 子功能 00H 分配标准页面	LIM 规范 V4.0

分配进程要求的标准页面数后，返回句柄

调用寄存器： AH 5AH
AL 00H
BX 希望分配的页面数

返回寄存器： 如果成功
AH 00H
DX EMM 句柄
如果失败
AH 80H EMS 软件内部错误
81H EMS 硬件故障
84H 未定义的功能
85H 所有的 EMM 句柄正在使用
87H 系统中没有足够页
88H 无足够页可用
8FH 子功能参数无效

注释：

这一功能分配请求的标准页（16K）数供调用进程使用，且返回被进程用于访问指定内存的唯一句柄，这类似于文件打开过程。

INT 67H	功能 5AH 子功能 01H 分配原始页面	LIM 规范 V4.0

在请求的原始页数分配给进程后，返回句柄

调用寄存器： AH 5AH
AL 01H
BX 希望分配页面数

返回寄存器： 如果成功

AH 00H

DX EMM 原始句柄

如果失败

AH 80H EMS 软件内部错误

81H EMS 硬件故障

84H 未定义的功能

85H 所有的 EMM 句柄正在使用

87H 系统中没有足够页面

88H 无足够页面可用

8FH 子功能参数无效

注释：

本功能分配指定数量的原始（非标准）页面供调用过程使用，并且返回一个由调用进程用于访问指定内存的唯一句柄。这类似于打开文件过程。应用程序不应该使用本子功能，因为原始页面的大小可能因 EMS 板的不同而变化，只使用标准页面能保持可移植性。

INT 67H	功能 5BH 子功能 00H	LIM 规范 V4.0
	取替换映象寄存器	

本功能只用于操作系统控制 EMS 硬件时使用；取替换映象寄存器组

调用寄存器： AH 5BH

AL 00H

返回寄存器： 如果成功

AH 00H

BL 非 0 则为当前替换映象寄存器组号，
而且 ES：DI 不受本功能影响。

0 ES：DI 指向操作系统提供的上下文保存区

ES：DI 指向操作系统提供的上下文保存区，
该保存区有系统中所有插件的所有映象寄存器的状态以及恢复插件的初始状态所需的附加数据

如果失败

AH 80H EMS 软件内部错误

81H EMS 硬件故障

84H 未定义的功能

8FH 子功能参数无效

A4H 操作系统拒绝访问本功能

注释：

本功能只由操作系统调用，而且操作系统可以在任何时候用 OS/E 功能（功能 5DH）来停用它。其详细操作极为复杂，使用时应参考正式的 LIM V4 说明。

INT 67H	功能 5BH 子功能 01H 置替换映象寄存器	LIM 规范 V4.0

本功能只用于操作系统控制 EMS 硬件时使用；置替换映象寄存器组

调用寄存器： AH 5BH

AL 01H

BL 新的替换映象寄存器组号：

0 ES：DI 为指向保存区的指针；

非 0 指定替换映象寄存器启动。

如果可行，本功能启动这一寄存器组。

ES：DI 的内容不受影响，被忽略

ES：DI 指向原来由子功能 00H 提供的映象寄存器上下文恢复区

返回寄存器： 如果成功

AH 00H

如果失败

AH 80H EMS 软件内部错误

81H EMS 硬件故障

84H 未定义的功能

8FH 子功能参数无效

9AH 不支持所请求的寄存器组

9CH 不支持任何替换映象寄存器组，且 BL 非零

9DH 所请求的寄存器组未定义或未分配

A3H 恢复数组，或指向该数组的指针错误

A4H 操作系统拒绝访问本功能

注释：

本功能只由操作系统调用，而且操作系统可以在任何时候用 OS/E 功能（功能 5DH）来停用它。其详细操作极为复杂，使用时应参考正式的 LIM V4 说明。

INT 67H	功能 5BH 子功能 02H 取替换映象寄存器组大小	LIM 规范 V4.0

本功能只用于操作系统控制 EMS 硬件时使用；取替换映象寄存器组保存区大小

调用寄存器： AH 5BH

AL 02H

返回寄存器： 如果成功

AH 00H

DX 请求取、置或取且置子功能时，要被传送到操作系统提供的内存

区的字节数

如果失败

AH 80H EMS 软件内部错误
81H EMS 硬件故障
84H 未定义的功能
8FH 子功能参数无效
A4H 操作系统拒绝访问本功能

注释：

本功能只由操作系统调用，而且操作系统可以在任何时候用 OS/E 功能（功能 5DH）来停用它。其详细操作极为复杂，使用时应参考正式的 LIM V4 说明。

INT 67H	功能 5BH 子功能 03H 分配替换映象寄存器组	LIM 规范 V4.0

本功能只用于操作系统控制 EMS 硬件时使用；分配替换映象寄存器组

调用寄存器： AH 5BH
AL 03H

返回寄存器： 如果成功

AH 00H
BL 替换映象寄存器组号；
如果无寄存器可用，则为零

如果失败

AH 80H EMS 软件内部错误
81H EMS 硬件故障
84H 未定义的功能
8FH 子功能参数无效
9BH 所有的替换映象寄存器组都已被分配
A4H 操作系统拒绝访问本功能

注释：

本功能只由操作系统调用，而且操作系统可以在任何时候用 OS/E 功能（功能 5DH）来停用它。其详细操作极为复杂，使用时应参考正式的 LIM V4 说明。

INT 67H	功能 5BH 子功能 04H 释放替换映象寄存器组	LIM 规范 V4.0

本功能只用于操作系统控制 EMS 硬件时使用；释放替换映象寄存器组

调用寄存器： AH 5BH
AL 04H

BL 欲释放的映象寄存器组号；不能为零

返回寄存器： 如果成功

AH 00H

如果失败

AH 80H EMS 软件内部错误
81H EMS 硬件故障
84H 未定义的功能
8FH 子功能参数无效
9CH 不支持任何替换映象寄存器组
9DH 所需寄存器组未定义或未分配
A4H 操作系统拒绝访问本功能

注释：

本功能只由操作系统调用，而且操作系统可以在任何时候用 OS/E 功能（功能 5DH）来停用它。其详细操作极为复杂，使用时应参考正式的 LIM V4 说明。

INT 67H	功能 5BH 子功能 05H 分配 DMA 寄存器组	LIM 规范 V4.0

本功能只用于操作系统控制 EMS 硬件时使用；分配 DMA 寄存器组

调用寄存器： AH 5BH

AL 05H

返回寄存器： 如果成功

AH 00H

BL 被分配的 DMA 寄存器组号；
如果无寄存器可用，则为零

如果失败

AH 80H EMS 软件内部错误
81H EMS 硬件故障
84H 未定义的功能
8FH 子功能参数无效
9BH 所有的 DMA 寄存器组均被分配
A4H 操作系统拒绝访问本功能

注释：

本功能只由操作系统调用，而且操作系统可以在任何时候用 OS/E 功能（功能 5DH）来停用它。其详细操作极为复杂，使用时应参考正式的 LIM V4 说明。

INT 67H	功能 5BH 子功能 06H 启用（Enable）DMA 寄存器组	LIM 规范 V4.0

本功能只用于操作系统控制 EMS 硬件时使用；启用 DMA 寄存器组

调用寄存器： AH 5BH
AL 06H
BL 欲启用的 DMA 寄存器组号；
若为零，则对指定的 DMA 通道无具体作用
DL 与指定组相联的 DMA 通道号

返回寄存器： 如果成功
AH 00H
如果失败
AH 80H EMS 软件内部错误
81H EMS 硬件故障
84H 未定义的功能
8FH 子功能参数无效
9AH 不支持指定的替换 DMA 寄存器组
9CH 不支持任何替换 DMA 寄存器组，且 BL 非零
9DH 所需的 DMA 寄存器组未定义或未分配
9EH 不支持专用 DMA 通道
9FH 不支持指定的 DMA 通道
A4H 操作系统拒绝访问本功能

注释：

本功能只由操作系统调用，而且操作系统可以在任何时候用 OS/E 功能（功能 5DH）来停用它。其详细操作极为复杂，使用时应参考正式的 LIM V4 说明。

INT 67H	功能 5BH 子功能 07H 停用 DMA 寄存器组	LIM 规范 V4.0

本功能只用于操作系统控制 EMS 硬件时使用；停用 DMA 替换寄存器组

调用寄存器： AH 5BH
AL 07H
BL 欲停用的 DMA 寄存器组号；若为零，则无任何作用

返回寄存器： 如果成功
AH 00H
如果失败
AH 80H EMS 软件内部错误
81H EMS 硬件故障
84H 未定义的功能
8FH 子功能参数无效
9AH 不支持指定的替换 DMA 寄存器组
9CH 不支持任何替换 DMA 寄存器组，且 BL 非零

9DH 所需的DMA寄存器组未定义或未分配
9EH 不支持专用DMA通道
9FH 不支持指定的DMA通道
A4H 操作系统拒绝访问本功能

注释：

本功能只由操作系统调用，而且操作系统可以在任何时候用OS/E功能（功能5DH）来停用它。其详细操作极为复杂，使用时应参考正式的LIM V4说明。

INT 67H	功能5BH，子功能08H	LIM规范V4.0
	释放DMA寄存器组	

本功能只用于操作系统控制EMS硬件时使用；释放DMA寄存器组

调用寄存器： AH 5BH
AL 08H
BI 欲释放的DMA寄存器组号；
若为零，则无任何作用

返回寄存器： 如果成功
AH 00H
如果失败
AH 80H EMS软件内部错误
81H EMS硬件故障
84H 未定义的功能
8FH 子功能参数无效
9CH 不支持任何DMA寄存器组，且BL非零
9DH 所需的DMA寄存器组未定义或未分配
A4H 操作系统拒绝访问本功能

注释：

本功能只由操作系统调用，而且操作系统可以在任何时候用OS/E功能（功能5DH）来停用它。其详细操作极为复杂，使用时应参考正式的LIM V4说明。

INT 67H	功能5CH	LIM规范V4.0
	为热启动作好硬件准备	

为热启动操作作EMS系统准备

调用寄存器： AH 5CH

返回寄存器： 如果成功
AH 00H
如果失败

AH 80H EMS 软件内部错误
81H EMS 硬件故障
84H 未定义的功能

注释：

本功能为即将进行的热启动操作作 EMS 硬件准备。一般情况下，当前映象上下文、使用中的替换寄存器组以及其他要初始化的 EMS 信息将受影响。任何把内存映象到 640K 以下的应用程序，一定要俘获所有可能引起热启动的条件，并且在热启动执行前调用本功能。

INT 67H	功能 5DH 子功能 00H 启用 OS/E 功能	LIM 规范 V4.0

本功能只用于操作系统启用对操作系统功能的访问

调用寄存器： AH 5DH
AL 00H
BX/CX 32 位存取码（第一次使用本功能时除外）

返回寄存器： 如果成功
AH 00H
BX/CX 32 位存取码（仅在第一次调用时使用）
如果失败
AH 80H EMS 软件内部错误
81H EMS 硬件故障
84H 未定义的功能
8FH 子功能参数无效
A4H 操作系统拒绝访问本功能

注释：

本功能只能由操作系统调用，而且可在任何时候被关闭。子功能 00H 启用功能 59H、5BH 和 5DH。

INT 67H	功能 5DH 子功能 01H 停用 OS/E 功能	LIM 规范 V4.0

本功能只用于操作系统停用对操作系统功能的访问

调用寄存器： AH 5DH
AL 01H
BX/CX 32 位存取码（第一次使用本功能时除外）

返回寄存器： 如果成功
AH 00H

BX/CX 32位存取码（仅在第一次调用时使用）

如果失败

AH 80H EMS软件内部错误
81H EMS硬件故障
84H 未定义的功能
8FH 子功能参数无效
A4H 操作系统拒绝访问本功能

注释：

本功能只能由操作系统调用，而且可在任何时候被停用。子功能01H停用功能59H、5BH和5DH。用子功能02H可再次访问它们。

INT 67H	功能5DH 子功能02H	LIM规范V4.0
	把OS/E存取码返回给EMS	

本功能只能由操作系统使用，控制对操作系统其他功能的访问

调用寄存器： AH 5DH
AL 02H
BX/CX 32位存取关键码

返回寄存器： 如果成功
AH 00H
如果失败
AH 80H EMS软件内部错误
81H EMS硬件故障
84H 未定义的功能
8FH 子功能参数无效
A4H 操作系统拒绝访问本功能

注释：

本功能只由操作系统使用，而且可在任何时候被停用。子功能02H把存取关键码返回给EMS，并将系统设置成初始状态；然后，启用OS/E功能（如果曾被停用的话），以后再次调用功能5DH任一子功能，都将返回一个新的存取码。

• MEMO •

第五章 XMS 功能调用

5.1 XMS 功能调用索引

功能号	功能	版本	页号
功能 000H	取 XMS 版本号	XMS 规范 V2.0	
功能 001H	请求高内存区 HMA	XMS 规范 V2.0	
功能 002H	释放高内存区 HMA	XMS 规范 V2.0	
功能 003H	全程启用 A20	XMS 规范 V2.0	
功能 004H	全程停用 A20	XMS 规范 V2.0	
功能 005H	局部启用 A20	XMS 规范 V2.0	
功能 006H	局部停用 A20	XMS 规范 V2.0	
功能 007H	查询 A20 状态	XMS 规范 V2.0	
功能 008H	查询自由扩充内存	XMS 规范 V2.0	
功能 009H	分配扩充内存块	XMS 规范 V2.0	
功能 00AH	释放扩充内存块	XMS 规范 V2.0	
功能 00BH	移动扩充内存块	XMS 规范 V2.0	
功能 00CH	锁住扩充内存块	XMS 规范 V2.0	
功能 00DH	扩充内存块解锁	XMS 规范 V2.0	
功能 00EH	取 EMB 句柄信息	XMS 规范 V2.0	
功能 00FH	重新分配扩充内存块	XMS 规范 V2.0	
功能 010H	请求上位存储块 UMB	XMS 规范 V2.0	
功能 011H	释放上位存储块 UMB	XMS 规范 V2.0	

5.2 XMS 功能调用详解

XMS 功能

XMS 功能由专门的设备驱动程序提供。(DOS 盘中 HIMEM. SYS) 用户可使用这些功能来访问上位存储块（UMB，在标准 640K 与 1M 之间的内存)、高内存区（HMA）和扩充内存（EMB）（见图 5.1)。

功能 000H	XMS 规范 V2.0
取 XMS 版本号	

确定正在使用的 XMS 版本号，并返回关于 HMA 是否存在的信息

调用寄存器： AH 000H

返回寄存器： AX XMS 版本号
BX XMS 驱动程序内部版本号
DX HMA 存在标志
00000H HMA 不存在
00001H HMA 存在

注释：

返回的版本号是 16 位 BCD 码值，如 AX 中的值为 01234H 意味着驱动程序用的是 XMS 12.34 版本。驱动程序内部版本号主要用于调试。HMA 的存在并不意味着可以使用。

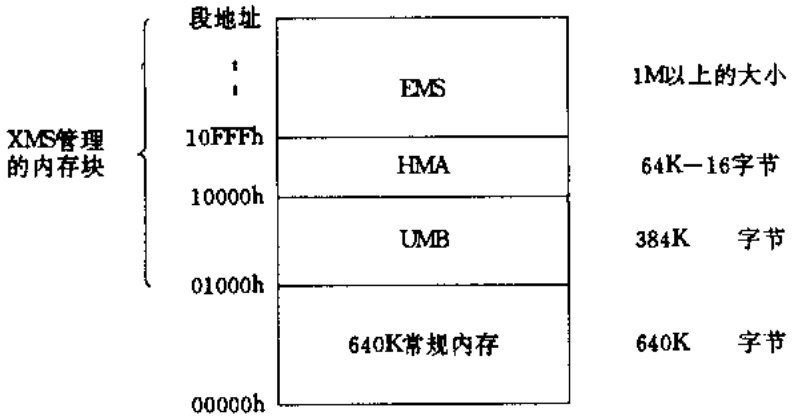


图5.1 XMS管理的内存结构

功能 001H	XMS 规范 V2.0
请求高内存区 HMA	

给调用程序分配 HMA（HMA 总共 65520 字节）

调用寄存器： AH 001H
DX 请求长度，以字节表示

返回寄存器： AX 状态
00000H 失败
BL 错误代码
080H 功能未实现
081H 检测到了 VDISK 设备
090H 不存在 HMA
091H HMA 正被占用
092H 请求字节长度小于 HMAMIN 参数

注释：

如果调用程序是一个应用程序，则 DX 中应置为 0FFFFH；否则，比如是设备驱动程序或 TSR，DX 的值应反映所需的字节数。可以通过 '/HMAMIN=' 参数设置分配 HMA 的最小值。

功能 002H	XMS 规范 V2.0
释放高内存区（HMA）	

释放 HMA

调用寄存器： AH 002H

返回寄存器： AX 状态

00000H 失败

00001H 成功

BL 错误代码

080H 功能未实现

081H 检测到了 VDISK 设备

090H 不存在 HMA

093H HMA 未被分配

注释：

分配了 HMA 的程序在退出前应释放 HMA。HMA 被释放后，其中的代码或数据不再有效，并不能再被访问。

功能 003H	XMS 规范 V2.0
全程启用（Enable）A20	

启用 A20 线

调用寄存器： AH 003H

返回寄存器： AX 状态

00000H 失败

00001H 成功

BL 错误代码

080H 功能未实现

081H 检测到了 VDISK 设备

082H A20 出现错误

注释：

本功能只能由控制 HMA 的程序调用。程序退出之前，应调用功能 004H 使 A20 线停用在许多机器上，A20 线的启用和停用的切换相当慢。

功能 004H	XMS 规范 V2.0
全程停用（Disable）A20	

停用 A20 线

调用寄存器： AH 004H
返回寄存器： AX 状态
00000H 失败
00001H 成功
BL 错误代码
080H 功能未实现
081H 检测到了 VDISK 设备
082H A20 出现错误
094H A20 线仍然可用

注释：

本功能只能由控制 HMA 的程序调用。在许多机器上，A20 线的启用和停用的切换相当慢。

功能 005H XMS 规范 V2.0
局部启用 A20（Enable）

启用 A20 线
调用寄存器： AH 005H
返回寄存器： AX 状态
00000H 失败
00001H 成功
BL 错误代码
080H 功能未实现
081H 检测到了 VDISK 设备
082H A20 出现错误

注释：

本功能只应由需直接访问扩充内存的程序使用。程序退出之前，应调用功能 006H 来停用 A20 线。在许多机器上，A20 线的启用和停用的切换相当慢。

功能 006H XMS 规范 V2.0
局部停用 A20（Disable）

停用 A20 线
调用寄存器： AH 006H
返回寄存器： AX 状态
00000H 失败
00001H 成功
BL 错误代码

080H 功能未实现
081H 检测到了 VDISK 设备
082H A20 出现错误
094H A20 线仍然可用

注释：

本功能只应由需直接访问扩充内存的程序使用。在许多机器上，A20 线的启用和停用的切换相当慢。

功能 007H	XMS 规范 V2.0
查询 A20 状态	

检查 A20 线的状态

调用寄存器： AH 007H

返回寄存器： AX 状态

00000H 失败
00001H 成功

BL 错误代码

000H 无错
080H 功能未实现
081H 检测到了 VDISK 设备

注释：

本功能由 Microsoft 的 HIMEM.SYS 驱动程序实现。实现时，用独立于硬件的方法检查内存环绕（Wrap）。

功能 008H	XMS 规范 V2.0
查询自由的扩充内存	

检查扩充内存是否可用

调用寄存器： AH 008H

返回寄存器： AX 以 KB 计的最大自由扩充内存块的大小

DX 以 KB 计的自由扩充内存的总的大小

BL 错误代码

080H 功能未实现
081H 检测到了 VDISK 设备
0A0H 所有扩充内存都已被分配

注释：

HMA 即使未被使用，返回值中也不包括它。

功能 009H	XMS 规范 V2.0
分配扩充内存块	

从自由扩充内存池中分配一给定大小的内存块

调用寄存器： AH 009H

DX 以 KB 计的内存块的大小

返回寄存器： AX 状态

00000H 失败

00001H 成功

DX 分配块的句柄

BL 错误代码

080H 功能未实现

081H 检测到了 VDISK 设备

0A0H 所有扩充内存都已分配

0A1H 所有扩充内存句柄均被占用

注释：

DX 中返回的句柄将在后继扩充内存调用功能中使用。如果未分配内存，返回的句柄为空。程序退出前，应调用功能 00AH 释放分配到的内存。

功能 00AH	XMS 规范 V2.0
释放扩充内存块	

释放扩充内存块

调用寄存器： AH 00AH

DX 欲释放的块句柄

返回寄存器： AX 状态

00000H 失败

00001H 成功

BL 错误代码

080H 功能未实现

081H 检测到了 VDISK 设备

0A2H 句柄无效

0ABH 句柄加锁

注释：

当一个块被释放后，其句柄和存储的数据将失效，不应再访问它。

功能 00BH	XMS 规范 V2.0
移动扩充内存块	

移动扩充内存块

调用寄存器： AH 00BH

DS：SI 指向扩充内存块移动结构的指针（见表 4.1）

返回寄存器： AX 状态

00000H 失败

00001H 成功

BL 错误代码

080H 功能未实现

081H 检测到了 VDISK 设备

082H A20 出现错误

0A3H 无效源句柄

0A4H 无效源偏移量

0A5H 无效目的句柄

0A6H 无效目的偏移量

0A7H 长度无效

0A8H 移动有重叠

0A9H 奇偶校验错

注释：

本功能不仅可把数据块在常规 DOS 存储区和扩充内存间移动，也可把块在常规 DOS 内存中和扩充内存中移动。

源或目的句柄都未被加锁。

长度必须是偶数。在 386 或 486 机器上，如果块按字或双字对准了，则可以提高性能。如果块重叠，只有向前移动（源基址小于目的基址）方可保证操作正确。

在调用本功能之前，不应试图控制 A20 线。A20 线的状态由本功能保存。

在大量数据的传输过程中，要保证有数量合理的中断窗口。

表 4.1 扩充内存块移动结构

偏移量	含义
000H－003H	需传送的 32 位字节数
004H－005H	源块句柄
006H－009H	32 位源偏移值
00AH－00BH	目的句柄
00CH－00FH	32 位目的偏移值

功能 00CH XMS 规范 V2.0

锁住扩充内存块

禁止移动某一内存块，并返回其物理地址

调用寄存器： AH 00CH

DX 要锁住的块句柄

返回寄存器： AX 状态

00000H 失败

00001H 成功

DX：BX 块的 32 位线性地址

BL 错误代码

080H 功能未实现

081H 检测到了 VDISK 设备

0A2H 句柄无效

0ACH 块的加锁计数溢出

0ADH 加锁失败

注释：

只有当块被成功加锁时，返回的指针才有效。被加锁的块应尽快解锁。每个扩充内存块都对应有一个加锁计数器。

功能 00DH XMS 规范 V2.0

扩充内存块解锁

解锁内存块

调用寄存器： AH 00DH

DX 要解开块的句柄

返回寄存器： AX 状态

00000H 失败

00001H 成功

BL 错误代码

080H 功能未实现

081H 检测到了 VDISK 设备

0A2H 句柄无效

0AAH 块未加锁

注释：

内存块解锁后，过去用来访问该块的任何 32 位指针都将失效。

本功能调用后，加锁计数器减一。要解开一个块，对该块的解锁调用次数必须与加锁调用次数相等。

功能 00EH XMS 规范 V2.0

取 EMB 句柄信息

取扩充内存块的附加信息

调用寄存器： AH 00EH

DX 块句柄

返回寄存器： AX 状态

00000H 失败

00001H 成功

BH 块的加锁计数值

BL 如果成功，为系统中还未用的 EMB 句柄数

如果失败，错误代码

080H 功能未实现

081H 检测到了 VDISK 设备

0A2H 句柄无效

注释：

要得到块的基址，可调用功能 00CH。

功能 00FH XMS 规范 V2.0

重新分配扩充内存块

改变扩充内存块的大小

调用寄存器： AH 00FH

BX 以 KB 计的新大小

DX 要改变大小的块的句柄

返回寄存器： AX 状态

00000H 失败

00001H 成功

BL 错误代码

080H 功能未实现

081H 检测到了 VDISK 设备

0A0H 所有扩充内存已分配

0A1H 所有可用扩充内存句柄已被占用

0A2H 句柄无效

0ABH 块加锁

注释：

如果新块大小小于旧块，在旧块高端的数据将丢失。

功能 010H	XMS 规范 V2.0

• MEMO •

第六章 AT 机 BIOS 功能 INT 15H

6.1 BIOS 功能 INT 15H 列表

在 IBM PC 系列机中，BIOS 的 INT 15H 提供了一些特殊的功能，见下表：

表 6.1 INT 15H 功能表

INT	15H,	40H	(64)	Read/Modify Profiles	Convertible
INT	15H,	41H	(65)	Wait for External Event	Convertible
INT	15H,	42H	(66)	Request System Power Off	Convertible
INT	15H,	43H	(67)	Read System Status	Convertible
INT	15H,	44H	(68)	Activate/Deactivate Modem Power	Convertible
INT	15H,	4FH	(79)	Keyboard Intercept	many
INT	15H,	80H	(128)	Device Open	many
INT	15H,	81H	(129)	Device Close	many
INT	15H,	82H	(130)	Device Program Termination	many
INT	15H,	83H	(131)	Event Wait	many

等待指定的微秒数，然后返回

调用寄存器： AH 86H

CX，DX 等待的微秒数

返回寄存器： 无

注释：

PC，PCjr，XT 不具备本功能。微秒计数由实时时钟产生，给定等待的微秒数最小为 976 微秒。

CX 是高字，DX 是低字。例如：CX＝98H，DX＝9680H 调用本功能产生 10 秒的延时。

INT 15H	功能 87H 块移动	XT-286，AT

本功能提供给实方式下运行的程序使用，以存取高于 1M 的内存空间，实现只有在保护方式下才能进行的 16MB 空间的存储块移动

调用寄存器： AH 87H

CX 字单位的块长度（最大为 8000H）

ES：SI 指向全程描述表（GDT）

返回寄存器： CF＝0 调用成功，否则 CF＝1

ZF＝1 调用成功，否则 ZF＝0

AH 返回操作状态

00H 成功

01H 内存校验错

02H 产生意外中断错误

03H 使能 A20 地址线失败

注释：

本功能只适于 AT 和 XT-286。

ES：SI 指向 GDT，GDT 由 6 个描述符组成，每个描述符长 8 个字节。描述符的格式见表 6.2。

表 6.2 描述符格式

偏移	字节数	解 释
00H	2	段限定
02H	2	24 位地址的低 16 位字
04H	1	24 位地址的高 8 位字节
05H	1	存取权限
06H	2	保留（必须置为 0）

全程描述符 GDT 的结构为：

表 6.3 全程描述符 GDT 结构

	偏移	字节数	解 释	初始化值
描述符 1	00H	8	哑描述符	用户置零
描述符 2	08H	8	本 GDT 的描述符 (由 BIOS 使用)	用户置零
描述符 3	10H	8	源块的描述符	用户指定
描述符 4	18H	8	目标块的描述符	用户指定
描述符 5	20H	8	保护方式下代码段的描述符 (由 BIOS 使用)	用户置零
描述符 6	28H	8	保护方式下堆栈段的描述符 (由 BIOS 使用)	用户置零

描述符 3 和描述符 4 必须由用户填入源块和目标块的描述，要求：

- 段限定字（偏移 00H）必须大于等于 2*(CX-1)
- 24 位地址处放入源块和目标块的地址
- 存取权限必须为 93H
- 传输大数据块时不处理中断，因为块移动时不允许中断

其余描述符全部置零。

INT 15H	功能 88H 取扩充内存的尺寸	XT-286, AT

取扩充内存的尺寸

调用寄存器： AH 88H

返回寄存器： CF=0 成功返回，否则 CF=1

AX 1M 边界以后的扩充内存尺寸，以 1K 为单位

注释：

本功能只适于 AT 和 XT-286，返回的扩充内存尺寸是存放在 CMOS RAM 的配置信息中的数。

INT 15H	功能 89H 切换到保护方式	XT-286, AT

切换到保护方式。将控制传送到用户指定的保护方式下的代码段

调用寄存器： AH 89H

BH 指向一级中断的中断描述表（IDT）

BL 指向二级中断的中断描述表（IDT）

ES：SI 指向全程描述表（GDT）

返回寄存器：　AH＝0　成功，否则 AH 不为 0

被改变的寄存器：　　AX，BP 和所有段寄存器

注释：

本功能只适于 AT 和 XT-286。

ES：SI 指向 GDT，GDT 由 8 个描述符组成，每个描述符长 8 个字节。描述符的结构与功能 87H 中的描述符结构相同。

全程描述符 GDT 的结构为：

表 6.4　全程描述符 GDT 结构

	偏移	字节数	解　　释	初始化值
描述符 1	00H	8	哑描述符	用户置零
描述符 2	08H	8	本 GDT 作数据段的描述符	用户指定
描述符 3	10H	8	中断描述表 IDT1 的描述符	用户指定
描述符 4	18H	8	中断描述表 IDT1 的描述符	用户指定
描述符 5	20H	8	附加段的描述符	用户指定
描述符 6	28H	8	堆栈段的描述符	用户指定
描述符 7	30H	8	代码段的描述符	用户指定
描述符 8	38H	8	临时 BIOS 代码段的描述符	用户置零

●由用户指定初始值的 6 个描述符，必须指定段限定、24 位地址和访问权限

●切换到保护方式后，BIOS 被关闭，用户必须处理所有的 I/O 操作

●用户必须初始化所有例外处理过程及参数表

●GDT 中的 IDT 不能与实方式 BIOS 中的 IDT 重叠

●用户必须保存中断向量表。

●用户必须重新初始化硬件中断向量表

6.3　INT 15H 示范程序

【程序 6.1】　INT 15H 示范程序

```
;
;     PutCCLIB.ASM - DOS Function calling sample
;                    / put CCLIB to extended memory
;
;          INT 15H 功能调用示范程序，用功能 87H、88H 将汉字库放入扩展内存
;
;          TASM putcclib
;          TLINK putcclib /t
;
block_len     equ       8000h                ; bytes
dosseg
              .model tiny
              .code
              org       100h
start :
              jmp       install
```

```
exm_sa_la   dw      0000h           ; 24-bit extended memory start address
exm_sa_ha   db      10h             ; start at 100000h
GDT         label   byte            ; GDT define
dum         db      8 dup (0)       ; must set to 0
desc        db      8 dup (0)       ; must set to 0
sour_sl     dw      0               ; source segment limit
sour_la     dw      0               ; 16-bit source address of 24-bit
sour_ha     db      0               ; 8-bit hi source address of 24-bit
sour_arb    db      0               ; access rights
sour_rsr    dw      0               ; reserved must be set to 0
dist_sl     dw      0               ; distance segment limit
dist_la     dw      0               ; 16-bit distance address of 24-bit
dist_ha     db      0               ; 8-bit hi distance address of 24-bit
dist_arb    db      0               ; access rights
dist_rsr    dw      0               ; reserved must be set to 0
dca         db      8 dup (0)       ; must set to 0
das         db      8 dup (0)       ; must set to 0
cclib       db      " cclib", 0     ; CCLIB file name
reset_gdt   proc                    ; sub-routine
            mov     di, offset gdt
            mov     cx, 6 * 8
            mov     al, 0
            cld
            rep     stosb
            mov     cl, 93h
            mov     sour_arb, cl
            mov     dist_arb, cl
            ret
move_to_exm proc                    ; sub-routine
            call    reset_gdt
            mov     cx, block_len
            mov     sour_sl, cx
            mov     dist_sl, cx
            mov     ax, cs
            mov     bx, offset my_bfr
            mov     cl, 4
            shr     bx, cl
            add     ax, bx
            mov     bx, ax
            mov     cl, 4
            shr     ah, cl
            mov     sour_ha, ah
            mov     cl, 4
            shl     bx, cl
            mov     dx, offset my_bfr
            and     dx, 0fh
            add     bx, dx
            mov     sour_la, bx
            mov     ax, exm_la
            mov     bl, exm_ha
            mov     dist_la, ax
            mov     dist_ha, bl
            add     ax, block_len
            adc     bl, 0
            mov     exm_la, ax
            mov     exm_ha, bl
            mov     si, offset gdt
            mov     cx, block_len / 2   ; 32kb = 16kw
            mov     ah, 87h
```

```
                int     15h
                ret
install :
                push    cs
                push    cs
                pop     ds
                pop     es
                mov     ah, 88H
                int     15h                     ; get zise of extended memory
                jc      no_exm                  ; extended memory error ?
                cmp     ax, 260                 ; above 260kb ?
                jb      no_exm

                mov     ax, 3d00h
                mov     dx, offset cclib
                int     21h                     ; open CCLIB
                jc      no_exm

                mov     bx, exm_sa_la           ; load extended memory start address
                mov     exm_la, bx
                mov     bh, exm_sa_ha
                mov     exm_ha, bh
                mov     bx, ax
reread :
                mov     ah, 3fh
                mov     cx, block_len           ; 32kb block length
                mov     dx, offset my_bfr
                int     21h                     ; read a block from CCLIB
                jc      no_exm
                cmp     ax, 0                   ; read to file end ?
                js      done
                push    bx
                call    move_to_exm             ; move a block to extended memory
                pop     bx
                jmp     reread
done :
                mov     dx, offset success_msg
                mov     al, 0                   ; success exit code
                jmp     prg_end
no_exm :
                mov     dx, offset err_msg
                mov     al, 1                   ; error exit code
prg_end :
                push    ax
                mov     ah, 9
                int     21h
                pop     ax
                mov     ah, 4ch
                int     21h

exm_la          dw      0
exm_ha          db      0

success_msg     db      " CCLIB put to extended memory successful", 0dh, 0ah, " $"
err_msg         db      " Can't load CCLIB to Extended Memory !", 7, 0dh, 0ah, " $"
my_bfr :                                ; program data buffer start here
                end     start
```

• MEMO •

第七章 MOUSE 功能调用

7.1 MOUSE 功能调用索引

MOUSE 功能通过 INT 33H 调用，索引如表 7.1，表中所列功能均为十六进制。

表 7.1 MOUSE 功能调用索引

功能	用 途	页号	功能	用 途	页号
00	鼠标初始化		14	交换用户事件处理程序	
01	显示鼠标光标		15	取保存状态存储区大小	
02	隐藏鼠标光标		16	保存鼠标驱动器状态	
03	取鼠标位置		17	恢复鼠标驱动器状态	
04	置鼠标位置		18	设置交替鼠标用户处理程序	
05	取按钮按下信息		19	取用户交替中断向量	
06	取按钮释放信息		1A	设置鼠标灵敏度	
07	置鼠标 X 界限		1B	取鼠标灵敏度	
08	置鼠标 Y 界限		1C	设置中断频率	
09	置图形光标形状		1D	设置 CRT 页号	
0A	置文本光标类型		1E	取 CRT 页号	
0B	读移动计数器		1F	禁止鼠标驱动	
0C	置用户定义事件处理程序		20	允许鼠标驱动	
0D	开始光笔模拟		21	软件复位	
0E	停止光笔模拟		22	设置消息语言	
0F	设置鼠标单位象素比		23	取消息语言	
10	有条件的光标关闭		24	取鼠标信息状态	
13	设置倍速阈值				

7.2 MOUSE 功能调用详解

DOS 鼠标功能调用是通过设备驱动程序（MOUSE.SYS 或 MOUSE.COM）实现的，中断号是 INT 33H。驱动程序根据鼠标本身的移动，随时在屏幕上更新鼠标的相对位置。程序不需要维护驱动程序的动作。

任何时候，程序都可以从驱动程序得知鼠标的当前状态。

鼠标驱动程序有好几个版本。这里讲述由 Microsoft V6 驱动程序提供的功能，它们是被广泛承认的工业标准。一些较早版本的驱动程序或一些别的鼠标制造商提供的驱动程序都缺乏某些功能（尤其是功能号高于 18H 的功能）。除了 DOS V1（它不允许装入设备驱动程序），大多数鼠标驱动程序是作为 TSR 方式来提供的。但不管它是以驱动程序还是以 TSR 来安装的，功能调用都相同。许多鼠标用户在 DOS 下使用 TSR 驱动程序。

INT 33H	功能 00H 初始化鼠标	V2

判断是否安装了鼠标；如果鼠标存在，将驱动程序复位，并返回鼠标的按钮数

调用寄存器： AX 0000H

返回寄存器： AX 0 未安装鼠标

−1 安装了鼠标

BX 按钮数（Microsoft 为 2，有些其他牌子的为 3）

注释：

使用鼠标的程序开始必须判定鼠标是否存在。该功能把鼠标复位在屏幕的中央，关闭鼠标，并设置缺省的鼠标光标和缺省的移动比例。

在 DOS V2.x 中，最好在调用该功能前检查 INT 33H 中断向量。如果中断向量地址的 4 个字节不全部为 00H，使用本功能才安全。

本功能设置的鼠标驱动器初始状态是：

显示页：	第 0 页
光标活动范围：	整个屏幕（X=0—639，Y=0—199）
非活动范围：	无
光标位置：	屏幕中央（X=300，Y=100）
光标状态：	隐藏
光标形状：	绘图方式下为箭头 文本方式下为反显示块
用户中断：	禁止
光笔模拟：	允许
Mickey/Pixel 比值：	水平方向=8∶8 垂直方向=16∶8
速度阈值：	每秒 64 个 mickey（mickey 为鼠标移动的单位， 1 个 mickey 约 1/200 英寸）

INT 33H	功能 01H 显示鼠标光标	V2

在显示器上显示鼠标光标

调用寄存器： AX 0001H

返回寄存器： 无

注释：

该功能并不一定显示鼠标，而是将内部的鼠标光标标志加 1。初始时，该标志设置为−1，

一旦标志为 0，鼠标就会显示出来。如果标志为 0 时调用功能 02H，光标标志减 1 为－1，光标消失。因此，若多次调用了功能 02H，就需多次调用本功能。标志不会超过 0。

INT 33H	功能 02H 关闭鼠标光标	V2

关闭鼠标光标的显示

调用寄存器： AX 0002H

返回寄存器： 无

注释：

该功能关闭显示功能，但并不关闭驱动程序。该功能将光标标志减 1，若该值非零，则光标关闭。因为光标标志不会比零大，仅调用一次就可把光标隐藏起来。

INT 33H	功能 03H 取鼠标位置	V2

返回当前鼠标位置和按钮状态

调用寄存器： AX 0003H

返回寄存器： BX 按钮状态

CX X 坐标（水平）

DX Y 坐标（垂直）

注释：

该功能返回鼠标所在位置。不论屏幕为什么方式，功能 03H 总是返回 X 坐标（列，0—639）和 Y 坐标（行，0—199）。表 7.2 显示了每种显示方式允许的光标位置，以屏幕坐标（象素）表示。

表 7.2 鼠标光标位置

屏幕方式	鼠标坐标	
00H，01H	x＝16＊列，	y＝8＊行
02H，03H	x＝8＊列，	y＝8＊行
04H，05H	x＝2＊屏幕 X，	y＝屏幕 Y
06H	x＝屏幕 X，	y＝屏幕 Y
07H	x＝8＊屏幕列，	y＝8＊屏幕行
0EH—10H	x＝屏幕 X，	y＝屏幕 Y

鼠标按钮的状态由 BX 返回，只有低位有效。表 7.3 说明各位的意义。由于各按钮的动作是独立的，对两按钮鼠标而言，该值在 0 到 3 之间；对三按钮鼠标而言，在 0 到 7 之间。

表 7.3 鼠标按钮状态位

位 7654 3210	含 义
....0	左边按钮放开
....1	左边按钮压下
.... ..0.	右边按钮放开
.... ..1.	右边按钮压下
.... .0..	中间按钮放开(若有)
.... .1..	中间按钮压下(若有)
xxxx x...	未定义

INT 33H	功能 04H 置鼠标位置	V2

设置鼠标光标的位置

调用寄存器： AX 0004H
CX 新 X 坐标（水平）
DX 新 Y 坐标（垂直）

返回寄存器： 无

注释：

可用该功能把鼠标光标置于屏幕上任何位置。这样，鼠标驱动程序就认为操作是从该位置开始的。例如，可把鼠标置于屏幕上菜单的某一项。

若不是对应于当前方式的合适坐标值，它会调整到最近的合适值上。如果所指定的位置处于由功能 07H 和 08H 所建立的范围以外，光标放在限制范围内尽可能靠尽指定位置的地方。若位置处于由功能 10H 定义的排除区域，将被隐藏起来。

INT 33H	功能 05H 取按钮按下信息	V2

返回关于按钮按下的信息

调用寄存器： AX 0005H
BX 按钮
0 左边
1 右边
2 中间（若有）

返回寄存器： AX 按钮状态（见表 7.3）
BX 按钮按下次数（在每次调用时复位 0）
CX 最后按下时光标的水平位置
DX 最后按下时光标的垂直位置

注释：

功能 05H 提供了从上次调用本功能以来指定按钮的信息。可以说明该按钮是否被压下过、从上次调用以来按下的次数及最后一次按下时的位置。

按钮的状态在 AX 寄存器中返回。该状态与由功能 03H 返回的 BX 相同（见表 7.3）。

INT 33H	功能 06H 取按钮释放信息	V2

返回关于按钮释放的信息

调用寄存器： AX 0006H

BX 按钮

0 左边

1 右边

2 中间（若有）

返回寄存器： AX 按钮状态（见表 7.3）

BX 上次调用以来释放的次数

CX 最后放开按钮时光标的水平位置

DX 最后放开按钮时光标的垂直位置

注释：

功能 06H 返回鼠标按钮放开的信息（功能 05H 返回按下的信息）。按钮的释放不同于按下，用功能 06H 和 07H 可以区别出这两种状态。寄存器返回的信息对应于放开操作。

INT 33H	功能 07H 置鼠标 X 界限	V2

设置鼠标在屏幕上移动的 X 界限

调用寄存器： AX 0007H

CX 最小 X 值

DX 最大 X 值

返回寄存器： 无

注释：

当希望限制鼠标在 X（水平）方向的移动范围时，可以调用功能 07H。例如，可以用于菜单内。功能 08H（Y 界限）对于限制鼠光标的移动也很有用。

鼠标驱动程序初始化 X 最小为 0，最大为 639。传入本功能的值在这两个值之间。如果 CX 寄存器中的值大于 DX 寄存器中的值，两值被交换。若该功能执行后鼠标在界限以外，它自动移到边界上。

INT 33H	功能 08H	V2
	置鼠标 Y 界限	

设置鼠标在屏幕上移动的 Y 界限

调用寄存器： AX 0008H
CX 最小 Y 值
DX 最大 Y 值

返回寄存器： 无

注释：

当希望限制鼠标在 Y（垂直）方向的移动范围时，可以调用功能 08H。例如，可以用于菜单内。功能 07H（X 界限）对于限制鼠标光标的移动也很有用。

鼠标驱动程序初始化 Y 最小为 0，最大为 199。传入本功能的值在这两个值之间。如果 CX 寄存器中的值大于 DX 寄存器中的值，两值被交换。若该功能执行后鼠标在界限以外，它自动移到边界上。

INT 33H	功能 09H	V2
	置图形鼠标形状	

设置图形方式下所用的鼠标形状

调用寄存器： AX 0009H
BX 热点 X 位置（−16 到 16）
CX 热点 Y 位置（−16 到 16）
ES：DX 指向视屏掩码和光标掩码的指针

返回寄存器： 无

注释：

在图形方式下，光标是由视屏掩码、光标掩码和热点来定义的。功能 09H 全部作了定义。

可见鼠标总是定义为一个 16 * 16 方块，热点是此方块左上角对应的单一象素地址，这正是光标位置。在方块内，可见的指示字形状是由视屏掩码和光标掩码确定的。

当鼠标在图形方式下产生时，视屏掩码同屏幕做与运算，而光标掩码与结果做异或运算。这样，对图形方式的实际效果见如下每位：

图形方式 1—6

光标掩码位 \ 视屏掩码位	0	1
0	0	1
1	变化	反视

对视屏方式 7 以上，效果如下：

光标掩码位 \ 视屏掩码位	0	1
0	黑	白
1	无变化	无变化

在 ES：DX 所指位置存储的掩码是 16 位字的位映象块，位映象的每一个字对应于 16 行

光标的一行（从最上面的行开始，视屏掩码在先）。字 0—15 是视屏掩码，字 16—31 是光标掩码。

在图形方式 4—6 和 14—16 下，每一光标象素有两位，即使光标大小不同也如此。

视屏掩码暂时擦去光标区域，然后光标掩码将指示字形状画到这一空区，这样，鼠标指示字好像覆盖了屏幕数据。如提供几套掩码，且使用这一功能在其间切换，就可以在程序的不同部分使用不同的指示字形状。

热点坐标总是相对于 16＊16 象素区的左上角。热点坐标定义了一个特定的坐标，每当任何功能要求鼠标信息时，它将返回。

设置热点为（0，0），图像的左上角成为鼠标位置（缺省的箭头图形的尖点）；设置为（8，8），则区域中心为正式位置；设置为（16，16），则位置移到右下角。（—16，—16）是一个合法热点设置，正式位置可以移到实际图像外边，当然，很难设想出好的理由去这样做。

INT 33H	功能 0AH	V2
	设置文本光标类型	

设置文本方式光标

调用寄存器： AX 000AH

BX 0 选择属性光标

1 选择硬件光标

CX 如 BX＝0，为视屏掩码（AND 值）

BX＝1，为开始扫描行

DX 如 BX＝0，为光标掩码（XOR 值）

BX＝1，为结束扫描行

返回寄存器： 无

注释：

在文本方式下，光标可为属性光标或硬件光标，这取决于 BX 的设置。对属性光标，CX 和 DX 寄存器分别为视屏和光标掩码，它们将对 CRT 属性字符字节对进行操作，就如图形方式掩码对象素作用一样。视屏掩码保存大多数的原字符属性，光标掩码决定哪个属性将被改变。为保持字符不变，视屏掩码的低字节应为 FFH，而光标掩码的低字节应为 00H。

在鼠标光标处的属性和字符字节与视屏掩码“与”，再与光标掩码“异或”。假如选择硬件操作方式，CX 和 DX 寄存器选择鼠标的起止扫描行，与 INT 10 功能 01H 中 CH 和 CL 的作用相同。用这一选择将鼠标设置为与正常文本光标不同的形状，这样，在屏幕上容易识别。因为可用形状数有限，通常用属性光标代替。

INT 33H	功能 0BH	V2
	读移动量	

测定从上次功能调用以来的实际鼠标移动量

调用寄存器： AX 000BH
返回寄存器： CX 水平移动的 mickey 数
（－32768 到 32768 负数表示向左移动）
DX 垂直移动的 mickey 数
（－32768 到 32768 负数表示向上移动）

注释：

功能 0BH 可返回两次调用间鼠标的相对移动。驱动程序一直跟踪鼠标位置并在每次调用这一功能时返回其位置。

CX 和 DX 寄存器返回相对移动量（正值对应向右移动和向下移动），返回数以 mickey 计量。mickey 为鼠标移动单位，1 个 mickey 约等于 0.5 毫米（0.02 英寸）。

INT 33H	功能 0CH	V2
	设置用户定义事件处理程序	

建立一个指向由设备驱动器程序调用的功能的向量，一旦所设调用掩码条件出现，就产生该调用

调用寄存器： AX 000CH
CX 调用掩码（见注释部分）
ES：DX 指向用户中断例程的指针
返回寄存器： 无

注释：

这一功能设置一个能被鼠标驱动程序认识的特殊处理程序。处理功能可对应于按下按钮、释放按钮、光标位置变化或这些事件的组合。一般是在程序运行过程中调用该功能，其操作基本上与中断处理程序相同。但所有细节都在鼠标驱动程序中，用户处理程序不必关心它们。

处理程序的操作是由 CX 中的调用掩码控制的。它指明哪种条件将引发处理程序。每当调用掩码中某位为 1 且条件发生时，由 ES：DX 指向的功能将被执行。在同样位置为 0 时，将取消这一功能。

表 7.4 是调用掩码字节中的位分配。

表 7.5 是用户事件处理例程入口处 CPU 寄存器的状态。

表 7.4 调用掩码位

位 76543210	含 义
.......1	鼠标移动了
......1.	左按钮按下
.....1..	左按钮释放
....1...	右按钮按下
...1....	右按钮释放
..1.....	中间按钮按下(如存在)
.1......	中间按钮释放(如存在)

表 7.5 调用用户功能时寄存器内容

寄存器	内 容
AX	事件引发位（与表 7.4 提供的条件掩码相同，但只有引发这一调用的位存在）
BX	按钮状态（同功能 05H 和 06H）
CX	光标水平坐标（同功能 03H）
DX	光标垂直坐标（同功能 03H）
DI	水平方向计数（同功能 0BH 中 CX）
SI	垂直方向计数（同功能 0BH 中 DX）

当调用功能 00H 时，整个调用掩码复位为 0。若一程序为以上条件建立了一个特殊处理程序，则该程序结束前应调用功能 00H，0CH 或 14H 来复位调用掩码。程序结束前记住恢复调用掩码的初始值和子程序的地址。

INT 33H 功能 0DH V2
打开光笔模拟

打开光笔模拟方式
调用寄存器： AX 000DH
返回寄存器： 无
注释：

在光笔模拟方式下，鼠标位置即是光笔位置。按下鼠标的两个按钮即相应于光笔指向屏幕。并非所有鼠标驱动程序都提供这一接口。

INT 33H 功能 0EH V2
关闭光笔模拟

关闭光笔模拟方式
调用寄存器： AX 000EH
返回寄存器： 无
注释：

这一功能用来通知设备驱动程序停止处理模拟光笔输入。功能 0EH 使得鼠标以其正式方式工作，不再用驱动程序转换。并非所有鼠标驱动程序都提供这一接口。

INT 33H 功能 0FH V2
设置鼠标移动单位与象素的比值

设置光标物理移动（以 mickey 计算，为 1 英寸的 1/200）和坐标变化（象素）的比值

调用寄存器： AX 000FH
CX 水平位置变化 8 个象素时所需 mickey 数（缺省为 8）
DX 纵向位置变化 8 个象素时所需 mickey 数（缺省为 16）

返回寄存器： 无

注释：

在 CX 和 DX 两个寄存器中，高位必须为 0，最小比值为 1。比值愈高，鼠标在屏幕上移动愈慢。

INT 33H	功能 10H 条件光标关闭	V2

定义一屏幕区域（排斥区域），在这一区域光标将隐藏

调用寄存器： AX 0010H
CX 区域左上 X 坐标
DX 区域左上 Y 坐标
SI 区域右下 X 坐标
DI 区域右下 Y 坐标

返回寄存器： 无

注释：

功能 10H 定义一屏幕区域。在这一区域内光标将消失。使用这一功能后，当光标移出隐藏区外时，一定要再调用功能 01H 恢复光标的可见性。

INT 33H	功能 13H 设置倍速阈值	V2

设置阈值速度。超过这一阈值时光标将以正常速率的两倍移动

调用寄存器： AX 0013H
DX 阈值速度（mickey/秒）

返回寄存器： 无

注释：

当光标以阈值或高于阈值速度运动时（mickey/秒），屏幕上光标移动因子乘 2。

INT 33H	功能 14H 交换用户事件处理程序	V2

设置用户中断功能和调用掩码，同时返回以前的值

调用寄存器： AX 0014H
CX 新的用户中断调用掩码
ES：DX 指向新的用户中断例程的指针

返回寄存器： CX 以前的用户中断掩码
ES：DX 以前的用户中断向量

注释：

像功能 0CH，这一功能设置对应特殊事件的用户定义处理功能。这些特殊事件是鼠标设备驱动程序所接受到的。调用掩码字与功能 0CH 所用的相同（见表 7.6），当调用掩码位置 1 时，允许功能 14H。当置 0 时，禁止该功能。

表 7.6 调用掩码位

位 76543210	含 义
.......1	鼠标移动了
......1.	左按钮按下
.....1..	左按钮释放
....1...	右按钮按下
...1....	右按钮释放
..1.....	中间按钮按下(如存在)
.1......	中间按钮释放(如存在)

不像功能 0CH，用这一功能可返回以前的值，允许保存恢复，除非两个功能完全相同。

当调用处理程序时，用表 7.5 所示信息设置 CPU 寄存器。DS 寄存器指向鼠标驱动数据段。如用户功能需取自己的数据，则须把 DS 指向自己的数据段。

这一功能与功能 0CH 的主要差别是可保存原来的调用掩码和用户中断向量。

记住：在结束程序前要恢复它们。

INT 33H 功能 15H V2
取保存状态存储区大小

返回用于保存鼠标驱动器当前状态的缓冲区的大小

调用寄存器： AX 0015H

返回寄存器： BX 存放当前鼠标状态所需缓冲区的大小（以字节计）

注释：

这一功能为功能 16H 和 17H 做准备。为了使用另一也需要鼠标的程序，必须保存鼠标当前状态。该功能测定保存状态所需内存大小。返回值正好是存放驱动程序状态所需的字节数。

INT 33H	功能 16H 保存鼠标驱动器状态	V2

拷贝鼠标驱动器状态到 ES：DX 指向的缓冲区

调用寄存器： AX 0016H
ES：DX 指向保存鼠标状态缓冲区的指针

返回寄存器 无

注释：

这一功能将鼠标当前状态拷贝到 ES：DX 指向的缓冲区。所需缓冲区的大小由功能 15H 返回。

当你挂起一使用鼠标的程序而运行另一程序时，使用这一功能。当第二个程序结束时，在结束前可恢复到原来的鼠标状态。

INT 33H	功能 17H 恢复鼠标驱动器状态	V2

恢复由功能 16H 保存的鼠标驱动状态

调用寄存器： AX 0017H
ES：DX 指向保存鼠标状态缓冲区的指针

返回寄存器： 无

注释：

在执行另一使用鼠标的程序后恢复到原来的程序时，用这一功能将鼠标状态恢复为第二个程序运行前的状态（假设已使用功能 16H 保存了这一状态）。

INT 33H	功能 18H 设置额外的鼠标用户处理程序	V2

允许设置多达 3 个的特殊事件处理程序（类似于在功能 0CH 或 14H 中定义的）

调用寄存器： AX 0018H
CX 调用掩码（见注释部分）
ES：DX 功能的地址偏移量

返回寄存器：AX 0018H 如果调用成功
FFFFH 如果调用失败

注释：

这一功能生成一个其条件能被鼠标驱动程序识别的特殊处理程序。一个功能可对应于按钮按下、按钮释放或光标位置改变。可分别调用功能 18H 定义多达 3 个这样的处理程序。由这一功能生成的处理程序与功能 0CH 或 14H 生成的处理程序的不同之处在于这一功能在 Shift，Alt 或 Ctrl 键按下时将作出响应。驱动程序可同时允许 4 个处理程序；这一功能允许 3

个，功能 0CH 允许 1 个。该功能不支持象 Shift-Alt 和 Ctrl-Alt 的组合。这一点与别处所作的描述不同。

中断操作是由调用掩码控制的。它规定了哪种条件将引发特殊处理程序。例如调用掩码某位为 1，在条件发生时，将执行由 DX 寄存器指向的功能。如该位为 0，则取消这一功能。表 7.7 显示了调用掩码位的用途。注意，因为定义三按钮鼠标器中间按钮的调用掩码位已用作 Shift 和 Ctrl 键指示器，这一功能不会对三键鼠标的中间键作出响应。

表 7.7 调用掩码位

位 76543210	含 义
.......1	鼠标移动
......1.	左按钮按下事件
.....1..	左按钮释放事件
....1...	右按钮按下事件
...1....	右按钮释放事件
..1.....	事件过程中 Shift 键按下
.1.....	事件过程中 Ctrl 键按下
1.....	事件过程中 Alt 键按下

当调用 00H 功能后，整个调用掩码复位为 0。在一个为上述条件设置了特殊处理的程序结束前，应调用功能 18H 或功能 00H 来复位调用掩码。

在调用处理程序时，CPU 寄存器如表 7.5 设置。

INT 33H	功能 19H 取用户额外中断向量	V2

返回一功能指针。此功能由调用 18H 定义

调用寄存器： AX 0019H
CX 调用掩码

返回寄存器： BX：DX 用户中断向量
CX 调用掩码（无匹配时为 0）

注释：

这一功能搜索由功能 18H 定义的事件处理程序，以找到那些调用掩码与 CX 寄存器值一致的处理程序。当发现时，BX 中返回段地址，DX 中返回偏移量。

INT 33H	功能 1AH 设置鼠标灵敏度	V2

设置鼠标速度和倍速阈值

调用寄存器： AX 001AH

BX 水平 mickey 数（可为 1—100，缺省为 8）
CX 垂直 mickey 数（可为 1—100，缺省为 16）
DX 倍速阈值（缺省为 64）

返回寄存器： 无

注释：

这一功能将功能 0FH 和 13H 合为一个调用，其值不被调用功能 00H 复位。BX 和 CX 的最大值为 100。

INT 33H	功能 1BH	V2
	取鼠标灵敏度	

返回由功能 1AH 设置的灵敏度值

调用寄存器： AX 001BH

返回寄存器： BX 每象素水平 mickey 数
CX 每象素垂直 mickey 数
DX 倍速阈值

注释：

这一功能返回鼠标灵敏度，以光标在屏幕上移动一个象素时鼠标移动的 mickey 数表示（水平和垂直）。

INT 33H	功能 1CH	V2
	设置中断频率	

设置鼠标驱动程序查询鼠标状态的频率

调用寄存器： AX 1CH
BX 中断频率代码
1 无查询
2 每秒查询 30 次
4 每秒查询 50 次
8 每秒查询 100 次
16 每秒查询 200 次

返回寄存器： 无

注释：

本功能只适合于 Microsoft InPort 鼠标。如 BX 不是上述所列值，其最低位将起作用（例如，3=无查询，因为它包含位 1）

INT 33H	功能 1DH	V2
	设置 CRT 页号	

设置鼠标显示所在的页

调用寄存器： AX 001DH

BX CRT 页号

返回寄存器： 无

注释：

这一功能只用来设置鼠标显示所在的视屏页。有效页号既取决于安装的视屏硬件，又取决于显示方式；参见中断 10H 功能 05H 对合法组合的讨论。

INT 33H	功能 1EH	V2
	取 CRT 页号	

取鼠标显示所在的 CRT 页

调用寄存器： AX 001EH

返回寄存器： BX CRT 页号

注释：

这一功能用于测定鼠标显示所在的视屏页。有效页号既取决于安装的视屏硬件，又取决于显示方式。参见中断 10H 功能 05H 对合法组合的讨论。

INT 33H	功能 1FH	V2
	关闭鼠标驱动	

通过恢复被鼠标驱动程序使用的中断向量来关闭鼠标驱动

调用寄存器： AX 001FH

返回寄存器： AX 001FH 成功

FFFFH 失败

ES：BX 原来的向量 INT 33H（只在成功时返回）

注释：

这一功能用恢复 INT 10H 和 INT 71H（8086 处理器系统）或 INT 74H（80286 或 80386 系统）的中断向量来禁止鼠标操作。INT 33H（鼠标中断）不直接受这一调用的影响，但 INT 33 原向量返回在 ES：BX 寄存器对中。这一向量可将 INT 33H 恢复为其原来值，完全禁止鼠标处理程序。

禁止任何中断处理程序都可能靠不住，因为无法得知在鼠标处理程序安装成功后，是否还有其他的处理程序链在这些中断链上。恢复的中断向量是鼠标第一次安装时的值。如果另外的处理程序此后将自己连在上面，调用这一功能也将禁止其他处理程序，这有可能导致系统崩溃。

INT 33H	功能 20H 打开鼠标驱动	V2

重装鼠标驱动程序占用的中断向量

调用寄存器： AX 0020H

返回寄存器： 无

注释：

如果 33H 中断向量没有使用，这一功能将恢复被调用功能 1FH 取消的 INT 10H 和 INT 71H 或 INT 74H 的中断向量。

INT 33H	功能 21H 软件复位	V2

复位鼠标驱动程序（并非鼠标器本身）

调用寄存器： AX 0021H

返回寄存器： AX FFFFH 鼠标驱动程序已安装

0021H 鼠标驱动程序还没安装

BX 鼠标按钮数（Microsoft 鼠标为 2，其他可能为 3）

注释：

除了它不复位鼠标硬件外，这一功能同功能 00H 完全相同。

INT 33H	功能 22H 设置提示信息语言	V2

选择鼠标驱动程序所用的提示和错误消息的语言（并非所有版本支持）

调用寄存器： AX 0022H

BX 语言号

0 英语
1 法语
2 荷兰语
3 德语
4 瑞士语
5 芬兰语
6 西班牙语
7 葡萄牙语
8 意大利语

返回寄存器： 无

注释：

只有鼠标驱动程序国际版有此功能。

INT 33H	功能 23H 取提示消息语言	V2

返回鼠标驱动程序所用的提示和错误消息的语言（并非所有版本支持）

调用寄存器： AX 0023H

返回寄存器： BX 语言号

0 英语
1 法语
2 荷兰语
3 德语
4 瑞士语
5 芬兰语
6 西班牙语
7 葡萄牙语
8 意大利语

注释：

只有鼠标驱动程序国际版有此功能。

INT 33H	功能 24H 取鼠标信息	V2

返回驱动程序版本号、鼠标类型和使用的 IRQ 号

调用寄存器： AX 0024H

返回寄存器： BH 主版本号（如 6.11 版为 6）

BL 次版本号（如 6.11 版为 11）

CH 鼠标类型

1 总线鼠标
2 串行鼠标
3 InPort 鼠标
4 PS/2 鼠标
5 HP 鼠标

CL 使用的 IRQ（中断请求）

0 PS/2 系统
2—7 IRQ 号

注释：

这一功能返回标准 Microsoft 驱动程序使用的驱动程序版本号、鼠标类型和中断请求线，其他公司的驱动程序可能不支持这一功能或返回不同结果。

7.3 EGA 寄存器接口

与 CGA 和 VGA，TVGA 不同，多数 EGA 硬件是通过只写寄存器控制的，这就使得显示控制异乎寻常地困难。为了简化跟踪 EGA 控制寄存器，鼠标驱动程序生成了一个虚拟 EGA 控制器，通过它控制物理 EGA 硬件。鼠标驱动程序用的是视频 BIOS 功能 INT 10H。假如实际运用中使用鼠标而且想操作 EGA 或 VGA 寄存器，就应该通过这个接口控制 EGA 寄存器。

INT 10H　　功能 F0H
读寄存器

从指定的 EGA 寄存器返回数据

调用寄存器：	AH	F0H
	BH	0
	BL	寄存器索引号（对单寄存器，不用这一寄存器）
	DX	端口号（见表 7.8）
返回寄存器：	BL	数据

注释：

非 Microsoft 的驱动程序可能不支持这一功能。

表 7.8 EGA 寄存器值

端口号	EGA 寄存器
00H	CRT 控制器
08H	定序器
10H	图形控制器
18H	属性控制器
20H	杂用输出寄存器
28H	特征控制寄存器
30H	图形 1 控制寄存器
38H	图形 2 控制寄存器

INT 10H　　功能 F1H
写寄存器

写数据到一指定寄存器

调用寄存器：	AH	F1H
	BL	指针/数据芯片索引或单寄存器数据
	BH	指针/数据芯片数据（单寄存器时不用）
	DX	端口号（见表 7.8）

返回寄存器： BH 修改后的内容
DX 修改后的内容

注释：

非 Microsoft 的驱动程序可能不支持这一功能。这一功能不检查使用的值；使用这一功能调用前，务必确认输入值，因为某些寄存器值的组合可能造成 EGA 适配器或监视器的物理损坏。本功能不保存 BH 和 DX 寄存器的内容

INT 10H 功能 F2H
读寄存器范围

从指定范围的 EGA 寄存器中返回数据（一个片上索引连续的几个寄存器）

调用寄存器： AH F2H
CH 起始索引
CL 寄存器数（必须大于 1）
DX 端口号
00H CRT 控制器
08H 定序器
10H 图形控制器
18H 属性控制器
ES：BX 一字节入口表指针，其长度为 CL 的值

返回寄存器： CX 修改后的内容

注释：

非 Microsoft 的驱动程序可能不支持这一功能。这一功能调用不保存 CX 寄存器值。

INT 10H 功能 F3H
写寄存器范围

对指定范围的 EGA 寄存器写数据（一个芯片上索引连续的几个寄存器）

调用寄存器： AH F3H
CH 起始索引
CL 寄存器数（必须大于 1）
DX 端口号
00H CRT 控制器
08H 定序器
10H 图形控制器
18H 属性控制器
ES：BX 一字节入口表指针，其长度为 CL 的值

返回寄存器： BX 修改后的内容

CX 修改后的内容
DX 修改后的内容

注释：

非 Microsoft 的驱动程序可能不支持这一功能。这一功能不检查使用的值。使用这一功能调用前，务必确认输入值，因为某些寄存器值的组合可能造成 EGA 适配器或监视器的物理损坏。这一功能调用不保存 BX，CX 和 DX 寄存器的值。

INT 10H 功能 F4H
读寄存器组

从指定范围的 EGA 寄存器组返回数据（几个寄存器可能不在同一个芯片上，即使在同一芯片上，索引也可能不连续）

调用寄存器： AH F4H
CX 寄存器数（必须大于 1）
ES：BX 记录表指针，其格式如下：
字节 0 端口号（见表 7.8）
字节 1 必须为 0
字节 2 指针寄存器索引值，单寄存器为 0
字节 3 这里是数据缓冲区
返回寄存器： CX 修改后的内容

注释：

非 Microsoft 的驱动程序可能不支持这一功能。这一功能调用不保存 CX 寄存器值。

INT 10H 功能 F5H
写寄存器组

调用前，务必确认输入值，因为某些寄存器值的组合可能造成 EGA 适配器或监视器的物理损坏。这一功能调用不保存 CX 寄存器的值。

INT 10H 功能 F6H
恢复为缺省寄存器

恢复所有被 EGA 寄存器接口改变的寄存器的缺省设置。缺省值是由功能调用 F7H 设定的

调用寄存器： AH F6H

返回寄存器： 无

注释：

非 Microsoft 的驱动程序可能不支持这一功能。如果用 INT 10H 功能 00H 用来设置显示方式，缺省寄存器值为所选方式的 BIOS 值。

INT 10H 功能 F7H
定义缺省寄存器表

定义单寄存器或指针/数据芯片上所有寄存器的缺省设置

调用寄存器：	AH	F7H
	CX	VGA 颜色选择标志 5448H
	ES：BX	所指表中字节偏移 14H 处为 VGA 颜色选择寄存器
	DX	端口号（见表 7.8）
	ES：BX	一字节入口地址表指针；表中必须包括指针/数据芯片上所有寄存器的值。
返回寄存器：	BX	修改后的内容
	DX	修改后的内容

注释：

非 Microsoft 的驱动程序可能不支持这一功能。这一功能不检查使用的值。使用这一功能调用前，务必确认输入值，因为一些寄存器值的组合可能造成 EGA 适配器或监视器的物理损坏。这一功能调用不保存 BH 和 DX 寄存器的值。

INT 10H 功能 FAH
查询驱动程序

查询鼠标驱动程序 EGA 寄存器接口是否存在

调用寄存器：	AH	FAH
	BX	0
返回寄存器：	BX	EGA 寄存器接口不存在时为 0

ES：BX　EGA 接口版本号指针。格式为：

字节 0　主版本号

字节 1　次版本号（以百分之几表示）

注释：

非 Microsoft 的驱动程序可能不支持这一功能。

7.4　如何使用 MOUSE 功能调用

【程序 7.1】　鼠标功能调用示范程序

```
{
    mdemo.pas- MOUSE Function calling sample
                Copyright (c) 1992 by Mr. Chen

    Turbo PASCAL 语言调用鼠标功能的示范程序
}
program TestMouse;
uses Dos, Crt, Graph;
type
  GraphCursMaskType = Record          { 定义图形鼠标的数据结构 }
    Mask:Array [0..1, 0..15] of Word;                { 鼠标图形掩码 }
    HorzHotSpot, VertHotSpot:Integer;                { 图形鼠标热点 }
  end;
const
  BgiPath = '';
  MouseDelay = 250;
  StandardShapeCurs:GraphCursMaskType = (           { 标准箭头鼠标 }
    Mask: ( ($3fff, $1fff, $0ff, $07ff, $03ff, $01ff, $00ff, $007f,
             $003f, $001f, $01ff, $10ff, $30ff, $f87f, $f8ff, $fc3f),
            ($0000, $4000, $6000, $7000, $7800, $7c00, $7e00, $7f00,
             $7f80, $7fc0, $7c00, $4600, $0600, $0300, $0180, $00c0));
    HorzHotSpot:-1;
    VertHotSpot:-1
  );
  PointingHandCurs:GraphCursMaskType = (            { 手形鼠标 }
    Mask:( ($e1ff, $e1ff, $e1ff, $e1ff, $e1ff, $e000, $e000, $e000,
            $0000, $0000, $0000, $0000, $0000, $0000, $0000, $0000),
           ($1e00, $1200, $1200, $1200, $1200, $13ff, $1249, $1249,
            $1249, $9001, $9001, $9001, $8001, $8001, $8001, $ffff));
    HorzHotSpot: 5;
    VertHotSpot:0
  );
var
  Regs:Registers;
  NumMouseKeys:Byte;
  MousePresent, MKP:Boolean;
  MKey: (None, Left, Right, Both);
  MouseX, MouseY, CellSizeX, CellSizeY:Word;
procedure SetKeyStatus (MStatus : Word);
begin
  Case MStatus Of
    0:MKey:= None;
    1:MKey:= Left;
    2: MKey:= Right;
    3:MKey:= Both;
```

```
  end;
end;

procedure ResetMouse;
var
  MouseInt:Pointer;
begin
  MKP:= False;
  NumMouseKeys:= 0;
  MousePresent:= False;
  GetIntVec ($33, MouseInt);
  if MouseInt <> Nil then begin
    Regs.AX := 0;
    Intr ($33, Regs);
    if Regs.AX <> 0 then begin
      MousePresent := True;
      NumMouseKeys := Regs.BX
    end;
  end;
end;

procedure VirtualScreenSize;
begin
  Regs.AH := $0F;
  Intr ($10, Regs);
  CellSizeX := 1;
  CellSizeY := 1;
  Case Regs.AL Of
      0, 1 : begin
               CellSizeX := 16;
               CellSizeY := 8;
             end;
      2, 3 : begin
               CellSizeX := 8;
               CellSizeY := 8;
             end;
        7 : begin
              CellSizeX := 9;
              CellSizeY := 14;
            end;
   4, 5, 13 : CellSizeX := 2;
  end;
end;

procedure ShowMouseCursor;                    {打开鼠标}
begin
  Regs.AX := 1;
  Intr ($33, Regs);
end;

procedure HideMouseCursor;                    {关闭鼠标}
begin
  Regs.AX := 2;
  Intr ($33, Regs);
end;

procedure GetButtonStatus;
begin
  Regs.AX := 3;
  Intr ($33, Regs);
  With Regs Do begin
    SetKeyStatus (BX);
    MouseX := CX;
    MouseY := DX;
  end;
```

```
end;

procedure SetGraphicsCursor (Var Mask : GraphCursMaskType);        { 设置图形鼠标 }
begin
  With Regs Do begin
    AX := 9;
    BX := Word (Mask.HorzHotSpot);
    CX := Word (Mask.VertHotSpot);
    DX := Ofs (Mask);
    ES := Seg (Mask);
  end;
  Intr ($33, Regs);
end;

Function LeftMouseKeyPressed : Boolean;
begin
  If MKP Then Delay (MouseDelay);
  GetButtonStatus;
  MKP := MKey = Left;
  LeftMouseKeyPressed := MKP;
end;

Function RightMouseKeyPressed : Boolean;
begin
  If MKP Then Delay (MouseDelay);
  GetButtonStatus;
  MKP := MKey = Right;
  RightMouseKeyPressed := MKP;
end;

Function BothMouseKeysPressed : Boolean;
begin
  If MKP Then Delay (MouseDelay);
  GetButtonStatus;
  MKP := MKey = Both;
  BothMouseKeysPressed := MKP;
end;

procedure DetectMouse;
begin
  ResetMouse;
  If Not MousePresent Then begin
    Writeln ('Mouse or mouse driver not present');
    Halt;
  end;
  VirtualScreenSize;
  Writeln ('Mouse present');
  Writeln ('Mouse has ', NumMouseKeys:1, ' keys');
  Write ('Press RETURN to continue demo ...');
  Readln;
end;

procedure HideAndShowDemo;
begin
  ClrScr;
  Writeln ('This demostrates hiding and showing the cursor');
  Writeln ('Press <Left Mouse Key> to hide cursor ...');
  Writeln ('Press <Right Mouse Key> to show cursoe ...');
  Write ('Press < Both Mouse Kev> to exit ...');
  ShowMouseCursor;
  While Not BothMouseKeysPressed Do begin
    GetButtonStatus;
    Case MKey of
      Left : HideMouseCursor;
      Right : ShowMouseCursor;
```

```
    end;
    GotoXY (1, 23);
    Write ('X: ', (MouseX div CellSizeX) :3,
        ' Y: ', (MouseY div CellSizeY) :3);
  end;
  HideMouseCursor;
end;

procedure GraphicsMouseDemo;
var
  S, S1, S2: String;
  StandardMouse: Boolean;
  GrBkCol, GrFgCol: Word;
  GraphDriver, GraphMode, GraphErrorCode: integer;
begin
  GraphDriver := Detect;
  InitGraph (GraphDriver, GraphMode, BgiPath);
  GraphErrorCode := GraphResult;
  If GraphErrorCode <> grOK Then begin
    ClrScr;
    Writeln ('Graphics error: ', GraphErrorMsg (GraphErrorCode));
    Halt;
  end;
  GrBkCol := GetBKColor;
  GrFgCol := GetColor;
  OutTextXY (0, 0, 'Press left mouse button to display position');
  OutTextXY (0, 10, 'Press right mouse button to change cursor shape');
  OutTextXY (0, 20, 'Press both mouse button to exit ...');

  ShowMouseCursor;
  GetButtonStatus;
  Str (MouseX:3, S1);
  Str (MouseY:3, S2);
  S := 'X: ' + S1 + ' Y: ' + S2;
  SetColor (GrFgCol);
  OutTextXY (0, 100, S);

  StandardMouse := True;
  While Not BothMouseKeysPressed Do begin
    If LeftMouseKeyPressed Then begin
      SetColor (GrBkCol);
      OutTextXY (0, 100, S);
      Str (MouseX:3, S1);
      Str (MouseY:3, S2);
      S := 'X: ' + S1 + ' Y: ' + S2;
      SetColor (GrFgCol);
      OutTextXY (0, 100, S);
    end;
    If RightMouseKeyPressed Then begin
      StandardMouse := Not StandardMouse;
      if StandardMouse
        then SetGraphicsCursor (StandardShapeCurs)
        else SetGraphicsCursor (PointingHandCurs);
    end;
  end;
  HideMouseCursor;
  CloseGraph;
end;
{----------------------主程序----------------------}
begin
  DetectMouse;                    { 鼠标检测并初始化 }
  HideAndShowDemo;                { 示范如何打开和关闭鼠标 }
  GraphicsMouseDemo;              { 示范如何使用和定义图形鼠标 }
end.
```

·MEMO·

第八章　识别程序运行环境

程序设计者为使软件更好地适应各种机器，发挥不同配置机器的特长，必须考虑对不同的硬件和软件资源采取相适应的软件设计策略。这些支持程序运行的硬件和软件资源构成程序运行环境。如：CPU、协处理器、显示卡、硬盘机、软盘机、通讯口、网络卡、BIOS、操作系统等等。不同的运行环境需要不同的软件处理方法，第一步是要识别程序运行的环境。本章介绍最典型最重要的程序运行环境的识别方法。

8.1　识别 CPU 类型

本节的汇编语言示例给出一个识别 INTEL 80x86系列 CPU 的完整程序。它可以独立运行，用以显示 CPU 的类型；或者用于批处理命令序列通过 ERRORLEVEL 由随后的程序检测而得知 CPU 类型。程序员也可以将标号行 SAYSO：以下的6条指令改为一条 RET 指令，将 IDx86过程直接应用到自己的程序中。程序清单中包含了每一步操作的详细说明以及汇编连接的方法。

【程序8.1】

```
;
;     IDX86.ASM — Identifying the 80x86 CPU chip
;
;     识别程序运行环境的示范程序之一
;     该程序显示检测到的系统 CPU 型号
;     对应8088/8086，80286，80386，80486分别置程序返回码0，2，3，4
;
;     汇编方法：
;     1、用 Turbo Asambler Ver3.00
;       TASM IDX86
;       TLINK IDX86 /t
;     2、用 MicroSoft MASM Ver6.00
;       ML IDX86
;             EXE2BIN IDX86.EXE IDX86.COM
;
comseg          segment
                assume cs：comseg，ds：comseg，es：comseg，ss：comseg
                org   100h
check           proc  near
          ;第一步首先检测是否为8088/8086 CPU.其关键不同点在 PUSH 指令.
          ;8088/8086 CPU 先减 SP，然后再送数到堆栈.
          ;80286，80386，80486 CPU 先送数到堆栈，然后再减 SP.
              push    sp              ;压入 SP
              pop     ax              ;弹出到 AX
              cmp     ax，sp          ;如果相同，则是80286或更高级的 CPU
              jne     short is_86     ;AX 与 SP 不同，8088/8086 CPU
          ;第二步检测是否为80286 CPU.其关键不同点在标志字的 IOPL 位.
          ;80286 CPU 没有 IOPL 位，对应位置恒为零
          ;80386 80486 CPU 的 IOPL 位可以设置
```

```
        pushf                   ; 压入标志字
        pop     ax              ; 弹出到 AX
        or      ax, 3000h       ; IOPL 位置1
        push    ax              ; 传送到标志字
        popf
        pushf                   ; 再从标志字取回
        pop     ax
        test    ax, 3000h       ; 如果 IOPL 位为1，则是80386 80486 CPU
        jz      short is_286    ; 如果 IOPL 位为0，80286 CPU
; 第三步检测是否为80386 CPU.其关键不同点在标志字的调整检测位.
; 80386 CPU 没有此位，对应位置恒为零
; 80486 CPU 可以设置
        db      66h             ; 32位指令前缀
        pushf                   ; 压入扩展标志寄存器 (32-bit)
        pop     ax              ; 读出低字
        and     ax, 0FFFh       ; 清除 IOPL 位
        pop     dx              ; 读出高字
        or      dx, 0004h       ; 设置调整检测位
        push    dx              ; 传送到扩展标志寄存器 (32-bit)
        push    ax
        db      66h
        popf
        db      66h             ; 再从扩展标志字取回
        pushf
        pop     ax
        pop     dx
        test    dx, 4           ; 如果为0，则是80386 CPU
        jnz     short is_486    ; 否则是80486 CPU
        mov     dx, offset say386
        mov     al, 3
sayso:
        push    ax
        mov     ah, 9
        int     21h             ; 显示 CPU 型号
        pop     ax              ; 返回 CPU 型号在 errorlevel
        mov     ah, 04Ch
        int     21h             ; 程序结束
is_486:
        mov     dx, offset say486
        mov     al, 4
        jmp     sayso
is_86:
        mov     dx, offset say86
        mov     al, 0
        jmp     sayso
is_286:
        mov     dx, offset say286
        mov     al, 2
        jmp     sayso
check   endp
say86   db      " 8088 or 8086 $ "
say286  db      " 80286 $ "
say386  db      " 80386 $ "
say486  db      " 80486 $ "
comseg  ends
        end     check
```

8.2　识别协处理器类型

本节的汇编语言示例给出了一个识别 INTEL 80x87系列协处理器的完整程序。它可以独立运行，用以显示协处理器的类型；或者用于批处理命令序列通过 ERRORLEVEL 由随后的程序检测而得知 CPU 类型。程序员也可以将 _math _ident 子程序单独摘出，嵌入自己的程序中。

【程序8.2】

```
;
;     IDX87.ASM - Identifying the 80x87 Math Coprocessor chip;
;     识别程序运行环境的示范程序之二
;     该程序显示检测到的数学协处理器型号
;     对应无协处理器，8087，80287，80387分别在程序返回码中返回0，1，2，3
;
;     汇编方法：
;     1.用 Turbo Assmbler Ver3.00
;       TASM IDX87
;       TLINK IDX87
;     2.用 MicroSoft MASM Ver6.00
;       ML IDX87
;
m _data         segment para 'M _DATA'
math _type      db      'no math $'
                db      '8087    $'
                db      '80287   $'
                db      '80387   $'
scratch         dw      (?)         ; have the math chip store data here
m _data         ends
m _code         segment para 'M _CODE'
                assume cs:m _code, ds:m _data
                ; 识别协处理器类型的子程序，返回型号在 AX 中

_math _ident    proc    far
                fninit                      ; 初始化协处理器
                mov     scratch, 055AAh
                fnstsw scratch              ; 存入协处理器状态字
                cmp     byte ptr scratch, 0 ; 有协处理器为0
                jne     no _math            ; 否则无协处理器
                fnstcw scratch              ; 存入协处理器控制字
                mov     ax, scratch         ; 进一步检测协处理器是否存在
                and     ax, 0103Fh
                cmp     ax, 0003Fh
                jne     no _math            ; 无协处理器
                and     scratch , 0FF7Fh    ; 有协处理器，判断其型号
                fldcw   scratch             ; 装入控制字
                fdisi                       ; 关中断
                fstcw   scratch             ; 存入控制字
                test    scratch, 00080h
                jnz     found _8087         ; 比较8087
                finit                       ; 重新初始化
                fld1
                fldz
```

```
                fdiv
                fld     st
                fcompp
                fstsw   scratch
                mov     ax, scratch
                ahf
                je      found_80287         ; 比较80287
                mov     ax, 3
return_result:
                ret
no_math:
                xor     ax, ax
                jmp     return_result
found_8087:
                mov     ax, 1
                jmp     return_result
found_80287:
                mov     ax, 2
                jmp     return_result
_math_ident     endp

start:                                      ; 主程序
                mov     ax, seg m_data
                mov     ds, ax
                call    _math_ident         ; 识别协处理器型号
                mov     bx, seg m_data
                mov     ds, bx
                push    ax
                mov     cl, 3
                shl     ax, cl
                mov     dx, offset math_type
                add     dx, ax
                mov     ah, 9
                int     21h
                pop     ax                  ; 协处理器类型在 AL 中返回 errorlevel
                mov     ah, 4ch
                int     21h
m_code          ends
                end     start
```

• MEMO •

第三部分　突破 DOS 编程

• MEMO •

第九章 DOS 保护模式接口 DPMI

9.1 DPMI 的引入

虽然 INTEL 公司的微处理器功能越来越强大，但它们受制于 DOS 的 1M 内存和 16 位代码，这越来越令人不满。现在，市场上已出现了大量克服这种局限的系统。这些系统充分利用 80286、80386 或 80486 的结构，提供了访问 1M 以上内存的途径（通过 EMS 或 XMS 内存管理程序）、多任务环境（windows、OS/2、DESQView）和用于运行保护模式程序的 DOS Extender 环境。DOS Extender 提供了可以调用 16 位 DOS 功能和 BIOS 功能的保护方式，还允许访问超过 1M 的内存。

DOS 保护方式接口（DPMI）是一种常用的 DOS 扩充规范，由一些著名的公司共同制定，它们分别是：

Borland International，
Ergo Computer Inc.，
Intelligent Graphics Corporation，
IBM Corporation，
Intel Corporation，
Locus，
Lotus Development Corporation，
Microsoft Corporation，
Phar Lap Software Inc.，
Phoenix Technologies Ltd.，
Quarterdeck Office Systems，
Rational Systems Inc.

有一点请用户注意：DPMI 只是一种供 DOS 扩展器使用的接口，通常情况下，请不要在应用程序中直接调用 DPMI。要想正确使用 DPMI 的功能，需要对保护模式下的机器资源有充分的了解，而这些知识已超出了普通应用程序设计的范围。

DPMI 可以在支持保护方式的 80286 上实现。但要想充分运用 DPMI 的功能，需要 80386 或 80486。

DPMI 0.9 版本是 Windows 3.0 的一部分。1.0 版本的功能有所增强，能更好地支持多任务。

在 DPMI 下，当一个实模式程序调用 DPMI 接口，执行了进入保护方式的初始化切换后，就产生了一个 DPMI 用户。一个 DOS 环境和从中产生的一个或多个保护方式 DPMI 用户总称为 DPMI 虚拟机。一个多任务主机可同时运行几个 DPMI 虚拟机，因而，一个 DPMI 用户不能假设它拥有机器的所有资源。在 80386 或 80486 上运行的 DPMI 可能会使用硬件保护方式来虚拟化 I/O 和中断，以及强制使用“超级用户/普通用户”权限机制，所以，DPMI 用户不能对它的权限级别作任何假定。

9.2 DPMI 功能详解

INT 2FH	DPMI 版本 1.0
释放当前虚拟机时间片	

允许操作系统将控制切换给另一个用户或对膝上型及笔记本型计算机采取节电措施

调用寄存器： AX 1680H

返回寄存器： AL 00H 得到主机支持

80H 未得到主机支持

注释：

每当程序空闲时，就应调用本功能，例如等待键盘输入时、程序再次获控制时，只要还是处于空闲状态，它就应不断调用本功能。

INT 2FH	DPMI 版本 0.9
取 CPU 模式	

返回当前 CPU 模式

调用寄存器： AX 1686H

返回寄存器： AX 00H 如果 CPU 处于保护方式

非零 如果 CPU 处于实模式或虚拟 86 方式

注释：

某些环境提供了既可在实模式下运行，也可在保护方式下运行的程序或库函数（双模式代码）。本调用允许这样的程序在运行时确定 CPU，以便相应地使用系统设备。只能在保护方式下运行的程序或库函数不需要调用本功能。

此调用必须在确定了 DPMI 主机已存在的前提下调用，否则结果无效。

INT 2FH	DPMI 版本 0.9
获取实模式到保护模式切换入口地址	

测试 DPMI 主机是否存在，并获得模式切换例程的地址，调用该例程可以切换到保护方式，此调用只能在实模式下进行

调用寄存器： AX 1687H

返回寄存器： AX 状态

00H 成功

非零 失败

如果成功 BX 标志

位 0=1 表示支持 32 位程序

位 0=0 表示不支持 32 位程序

位 1—15，未用

CL 处理器类型

02H 80286

03H 80386

04H 80486

05H—FFH，为将来的 Intel 处理器保留

DH DPMI 主版本号

DL DPMI 次版本号

SI DPMI 主机专用数据需要的节数（可能为零）

ES：DI 保护方式入口地址的段地址：偏移量

注释：

本功能成功的返回后，应用程序就可以通过调用模式切换例程而进入保护方式。进行此调用的步骤如下：

1. 如果 SI 的返回值不为零，调用程序必须按 SI 指定的节数分配一块供 DPMI 主机专用的内存，并把其段地址放在 ES 中；如果 SI 的返回值为零，忽略对 ES 的设置；
2. 在 AX 中设置指示应用程序是 16 位（位 0 为 0）或 32 位（位 0 为 1）的标志；
3. 对模式切换例程进行远调用。

如果此调用返回时进位标志被设置，表示模式切换失败，用户仍然处于实模式。AX 指出失败原因：8011H 表示描述符不能用，或者不能为 CS、DS、ES、SS、PSP 和环境指针分配描述符；8021H 表示用户指定了 32 位程序，但 DPMI 不支持。

如果此调用返回时进位标志被清零，表示模式切换成功。CS、DS 和 SS 将成为基于实模式的以 64K 为限的 16 位选择器。ES 为基于程序 PSP 的以 100H 为限的选择器。FS 和 GS 若存在，则为空选择器。程序 PSP 的环境指针也转变为选择器；如果用户想释放环境空间，它必须在进入保护方式之前进行，而且必须将 PSP：2CH 处的字（环境段址）清零。程序在进入保护方式以后，可以释放或修改 CS、DS、SS 描述符；也可以改变 PSP 中的环境指针，但应在退出之前予以恢复。程序在任何时候都不能释放或修改 PSP 和环境描述符。

INT 2FH	DPMI 版本 1.0
取专用 API 入口点	

返回用于访问专用 DPMI 扩充功能的入口地址

调用寄存器： AX 168AH

DS：(E)SI ASCIIZ 字符串的选择器:偏移量，用来标识专用 DPMI

返回寄存器： AL 00H 成功

8AH 失败

ES：(ES)DI 扩充 API 入口地址

注释：

本功能只能在保护方式下调用。DPMI 0.9 版本也支持本功能，但这一点未写入文档。用户必须对入口地址使用远调用。传递的字符串区分大小写。

INT 31H	功能 0000H 分配 LDT 描述符	DPMI 版本 0.9

在局部描述符表（LDT）中分配一个或多个描述符

调用寄存器： AX 0000H
CX 要分配的描述符数

返回寄存器： 如果成功，进位标志清零
AX 基选择器
如果失败，进位标志置位
AX 8011H 描述符不可用

注释：

被分配的描述符初始化为数据描述符，该描述符已设置了 Present 位、基地址和零限制，其优先级与调用程序的代码段优先级相同。

如果分配了多个选择器，描述符连续存放，形成一数组，AX 为第一个描述符。数组中后继描述符的选择器可通过把 INT 31H 功能 0003H 的返回值加上某一值计算出来。

INT 31H	功能 0001H 释放 LDT 描述符	DPMI 版本 0.9

释放一个 LDT 描述符

调用寄存器： AX 0001H
CX 要释放的描述符选择器

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 8022H 无效选择器

注释：

由 INT 31H 功能 0000H 分配的描述符必须单个释放。对于多个描述符，即使它们连续存放并当作一个描述符数组同时分配，也应单个释放。

对低于 1.0 的版本，任何含有要释放的选择器的段寄存器，将在调用后被清零。

INT 31H	功能 0002H 把段地址转换为描述符	DPMI 版本 0.9

将一个实模式段地址转换成一个能被保护方式的程序用来访问同一内存的 LDT 描述符

调用寄存器： AX 0002H
BX 实模式段地址
返回寄存器： 如果成功，进位标志清零
AX 访问实模式段的选择器
如果失败，进位标志置位
AX 8011H 描述符不可用

注释：

本功能提供了一种访问共用实模式段的简单方法，例如在段地址 0040H 处的 BIOS 数据区以及在段地址 A000H，B000H 或 B800H 的视屏缓冲区。由本功能得到的描述符不能被修改或释放。如果一个用户想用同一个选择器访问不同的实模式地址，可先调用 INT 31H 功能 0000H 来分配描述符，再调用 INT 31H 功能 0007H 来改变基址。

描述符大小限制为 64K，用同一段地址多次调用本功能将返回同一选择器。

INT 31H	功能 0003H	DPMI 版本 0.9
	取选择器的增量值	

返回选择器的增量值。该值用来访问描述符邻接数组的后续描述符，例如通过调用 INT 31H 功能 0000H 或 INT 31H 功能 0100H 所分配的描述符

调用寄存器： AX 0003H
返回寄存器： 进位标志清零
AX 选择器增量值

注释：

返回值总是一个 2 的幂。

INT 31H	功能 0006H	DPMI 版本 0.9
	取段基址	

返回 32 位线性基址，该地址来自指定段的 LDT 描述符

调用寄存器： AX 0006H
BX 选择器
返回寄存器： 如果成功，进位标志清零
CX：DX 32 位段线性基址
如果失败，进位标志置位
AX 8022H 无效选择器

注释：

用户程序必须使用 LSL 指令来查询描述符限界。在 80386 或 80486 机器上，如果段大小超过 64K，则大小限制为 32 位形式。

INT 31H	功能 0007H	DPMI 版本 0.9
	设置段基址	

把 LDT 描述符的 32 位线性基址项设置成指定的段地址

调用寄存器： AX 0007H
BX 选择器
CS：DX 32 位线性段基址

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误代码
8022H 无效选择器
8025H 无效线性地址

注释：

DPMI 版本 1.0 自动重装入任何包含 BX 所指选择器的段寄存器。有的 DPMI 0.9 主机有这种功能，有的没有。

INT 31H	功能 0008H	DPMI 版本 0.9
	设置段限界	

设置 LDT 描述符指定段的限界值

调用寄存器： AX 0008H
BX 选择器
CS：DX 32 位段限界

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误代码
8021H 无效值
8022H 无效选择器
8025H 无效线性地址

注释：

CX：DX 中的值必须是段的字节长度减 1。如果限界大于或等于 1M，那么必须是按页对准的（也就是说，最低的 12 个有效位必须为零）

DPMI 主机将根据请求，清除或设置描述符的 Granularity 位。

DPMI 1.0 将重新装入任何具有 BX 寄存器中的选择器的段寄存器。

INT 31H	功能 0009H	DPMI 版本 0.9
	设置描述符访问权限	

修改 LDT 描述符中的访问权限和类型域

调用寄存器： AX 0009H
BX 选择器
CL 访问权限字节
CH 80386 扩充的访问权限字节

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误代码
8021H 无效值
8022H 无效选择器
8025H 无效线性地址

注释：

CL 寄存器中访问权限字节格式如下：

表 9.1 访问权限字节

位	含 义
0	0=未被访问，1=被访问过
1	数据：0=可读，1=可读/可写 代码：必须为 1（可读）
2	数据：0=向上扩充，1=向下扩充 代码：必须为零
3	0=数据，1=代码
4	必须为 1
5—6	必须等于调用程序的 CPL
7	0=不存在，1=当前存在

在 CH 寄存器中的扩充访问权限字节格式如下：

表 9.2 扩充访问权限字节

位	含 义
0—3	忽略
4	可为零或 1
5	必须为零
6	0=16 位，1=32 位
7	0=字节 Granular，1=页 Granular

程序应使用 LAR 指令来检查描述符的访问权限

DPMI 1.0 将重载入任何引用受影响的描述符的段寄存器。

INT 31H	功能 000AH 产生别名描述符	DPMI 版本 0.9

产生一个新的 LDT 描述符，它同指定的描述符有相同的基和限界

调用寄存器： AX 000AH
BX 选择器

返回寄存器： 如果成功，进位标志清零
AX 数据选择器（别名）
如果失败，进位标志置位
AX 错误代码
8011H 描述符不可用
8022H 无效选择器

注释：

选择器可以是数据或代码选择器。已出版的 DPMI 0.9 文档指出如果为数据选择器，将产生错误，这种说法是错误的。

返回的描述符别名不随原描述符的变化而变化。这样，如果原段的基和限界在产生一个别名之后改变，那么此段和它的别名不再映象同一个内存。

INT 31H	功能 000BH 取描述符	DPMI 版本 0.9

将指定选择器的 LDT 项拷贝到一个 8 字节缓冲区

调用寄存器： AX 000BH
BX 选择器
ES：(E)DI 8 字节缓冲区的选择器：偏移量

返回寄存器： 如果成功，进位标志清零
由 ES：(E)DI 指向的的缓冲区将含有描述符
如果失败，进位标志置位
AX 8022H 无效选择器

注释：

16 位程序应使用 ES：DI；32 位程序应使用 ES：EDI。

INT 31H	功能 000CH 设置描述符	DPMI 版本 0.9

将一个 8 字节缓冲区内容拷贝到指定选择器的 LDT 描述符中

调用寄存器： AX 000CH
BX 选择器

ES：(E)DI 包含描述符的 8 字节缓冲区的选择器：偏移量

返回寄存器： 如果成功，进位标志清零

如果失败，进位标志置位

AX 错误代码

8021H 无效值

8022H 无效选择器

8025H 无效线性地址

注释：

32 位程序必须使用 ES：EDI；16 位程序应使用 ES：DI。DPMI 1.0 将重载入包含指定选择器的段。

INT 31H	功能 000DH 分配专用 LDT 描述符	DPMI 版本 0.9

分配一个专用的 LDT 描述符

调用寄存器： AX 000DH

BX 选择器

返回寄存器： 如果成功，进位标志清零

如果失败，进位标志置位

AX 错误代码

8021H 描述符不可用

8022H 无效选择器

注释：

最前面的 16 个描述符（值为 04H—7CH）由本功能保留。在 DPMI 0.9 下，也许别的应用程序已被装入并被分配了一些描述符，使之不可利用。而在 DPMI 1.0 下，每一个用户除了有它自己的 LDT 外，还可以访问所有的 16 个描述符。

常驻内存服务程序（保护方式 TSR）不能调用本功能。

INT 31H	功能 000EH 取多个描述符	DPMI 版本 1.0

把一个或多个 LDT 描述符项拷贝到用户缓冲区

调用寄存器： AX 000EH

CX 要拷贝的描述符数

ES：(E)DI 指向表 9.3 格式的缓冲区的选择器：偏移量

表 9.3 缓冲区格式

偏移量	长度	内 容
0000H	2	选择器#1（由用户设置）
0002H	8	描述符#1（由主机返回）
000AH	2	选择器#2（由用户设置）
000CH	8	描述符#2（由主机返回）
.		
.		
.		

返回寄存器： 如果成功，进位标志清零

缓冲区中将包含特定选择器描述符的拷贝

如果失败，进位标志置位

AX 8022H 无效选择器

CX 已成功拷贝的描述符数

注释：

如果检测到一个无效选择器或描述符，所有在此之前的描述符拷贝依然有效。32 位程序必须使用 ES：EDI；16 位程序应使用 ES：DI。

INT 31H	功能 000FH	DPMI 版本 1.0
	设置多个描述符	

从一个用户缓冲区中把一个或多个 LDT 描述符项拷贝到 LDT

调用寄存器： AX 000FH

CX 要拷贝的描述符数

ES：(E)DI 指向表 9.4 格式的缓冲区的选择器:偏移量

表 9.4 缓冲区格式

偏移量	长度	内 容
0000H	2	选择器#1
0002H	8	描述符#1
000AH	2	选择器#2
000CH	8	描述符#2
.		
.		
.		

返回寄存器： 如果成功，进位标志清零

如果失败，进位标志置位

AX 错误代码

8021H 无效值
8022H 无效选择器
8025H 无效线性地址
CX 已成功拷贝的描述符数

注释：

如果因一个无效选择器或描述符而导致错误，则在 CX 中返回已成功拷贝的描述符数。在失败的描述符之前，所拷贝的均有效，剩余的描述符没更新。

32 位程序必须使用 ES：EDI；16 位程序应使用 ES：DI。

任何包含在数据结构中指定的选择器的段寄存器将被再载入。

INT 31H	功能 0100H 分配 DOS 内存块	DPMI 版本 0.9

从低于 1M 的内存区中分配内存块

调用寄存器： AX 0100H
BX 需要的节数

返回寄存器： 如果成功，进位标志清零
AX 已分配块的实模式段基址
DX 已分配块的选择器
如果失败，进位标志置位
AX 错误代码
0007H 内存控制块损坏
0008H 内存不足
8011H 描述符不可用
BX 以节计的最大块的大小

注释：

如果块大于 64K，用户为一个 16 位程序，那么将分配相连的描述符并返回基本选择器。后继的选择器可利用 INT 31H 的功能 0003H 的返回值计算获得。除了最后一个块有大小模 64K 的限界外，其余的描述符都是 64K 的限界。如果主机是 32 位的，第一个描述符限界为整个块的大小；如果主机是 16 位的，即使在 80386 上运行，第一个描述符的限界也是 64K。

如果用户是 32 位的，只分配一个描述符。

用户程序不能修改或释放任何由本功能分配的描述符。描述符可调用 INT 31H 功能 0101H 来释放。

本功能使用了 DOS 功能 48H，用来分配 DOS 内存。

INT 31H	功能 0101H 释放 DOS 内存块	DPMI 版本 0.9

释放由 INT 31H 功能 0100H 分配的内存块

调用寄存器： AX 0101H
DX 要释放块的选择器

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误代码
0007H 内存控制块损坏
0009H 内存不足
8022H 无效选择器

注释：

所有为内存块分配的描述符自动释放，不再有效。

在 DPMI 1.0 下，任何包含被释放选择器的段寄存器将被置零。

INT 31H	功能 0102H 改变 DOS 内存块大小	DPMI 版本 0.9

改变由 INT 31H 功能 0100H 分配的内存块大小

调用寄存器： AX 0102H
BX 新块大小（以节表示）
DX 要修改块的选择器

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误代码
0007H 内存控制块损坏
0008H 指定的内存不足
8011H 描述符不可用
8022H 无效选择器
BX 最大可能块大小

注释：

如果 DOS 内存是碎块或内存不够，或块超过 64K 界限且下一个 LDT 中描述符不可用，那么增大块操作将失败。

减小块大小可能释放一些描述符并减小新的最后一个描述符的限界。

在 DPMI 1.0 下，将重载入任何包括一个要修改的选择器的段寄存器，任何包括一个要释放的选择器的段寄存器将置为零。

用户程序不应修改或释放任何由本功能分配的描述符。INT 31H 功能 0101H 将自动释放它们。

INT 31H	功能 0200H 取实模式中断向量	DPMI 版本 0.9

返回当前虚拟机的实模式中断向量

调用寄存器： AX 0200H

BL 中断号

返回寄存器： 进位标志清零

CX：DX 实模式中断处理程序的段地址：偏移量

注释：

CX 中的返回值是一个实模式段地址，而不是一个选择器。若试图把此值置入一个段寄存器可能导致一般性保护错误。

INT 31H	功能 0201H 设置实模式中断向量	DPMI 版本 0.9

设置当前虚拟机的实模式中断向量

调用寄存器： AX 0201H

BL 中断号

CX：DX 实模式中断处理程序的段地址：偏移量

返回寄存器： 进位标志清零

注释：

CX 中地址必须是一个实模式段地址，而不是一个选择器。处理程序必须驻留在 DOS 内存或用户必须分配一个实模式的调用返回地址。

如果此中断为硬件中断，由处理程序使用的内存必须加锁。

INT 31H	功能 0202H 取处理器异常处理程序向量	DPMI 版本 0.9

对于给定的异常号，返回当前用户的保护方式异常处理程序地址

调用寄存器： AX 0202H

BL 异常号 (00H—1FH)

返回寄存器： 如果成功，进位标志清零

CX：(E) DX 异常处理程序的选择器：偏移量

如果失败，进位标志置位

AX 8021H 无效值

注释：

本功能由向后兼容的 DPMI 1.0 支持，DPMI 1.0 的用户可以使用 INT 31H 功能 0210H 和功能 0211H。CX 中的值是一个选择器，不是一个段地址。对于 32 位用户，在 EDX 中返回

值是一个 32 位的偏移量。

INT 31H	功能 0203H	DPMI 版本 0.9
	设置处理器异常处理程序向量	

设置 CPU 异常情况或错误的处理程序地址

调用寄存器： AX 0203H
BL 异常号（00H—1FH）
CX：(E)DX 处理程序的选择器：偏移量

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误代码
8021H 无效值
8022H 无效选择器

注释：

CX 中的值是一个有效的保护方式代码选择器，而不是一个段地址。32 位用户必须在 EDX 中提供一个 32 位的偏移量。

DPMI 1.0 的用户不要使用本功能，因为它是后向兼容的，可以用 INT 31H 功能 0212H 和 0213H 代替。

INT 31H	功能 0204H	DPMI 版本 0.9
	取保护方式中断向量	

返回当前保护方式下指定中断的中断处理程序地址

调用寄存器： AX 0204H
BL 中断号

返回寄存器： 进位标志清零
CX：(E)DX 中断处理程序的选择器：偏移量

注释：

CX 中的值是一个有效的保护方式选择器，而不是一个实模式的段地址。EDX 中 32 位偏移量是为 32 位用户的返回值。

INT 31H	功能 0205H	DPMI 版本 0.9
	设置保护方式中断向量	

设置指定的中断的保护方式中断处理程序地址

调用寄存器： AX 0205H
BL 中断号

CX：(E)DX 中断处理程序的选择器:偏移量

返回寄存器： 如果成功,进位标志清零

如果失败,进位标志置位

AX 8022H 无效选择器

注释：

CX 中的值是一个有效的保护方式选择器,而不是一个实模式的段地址。32 位用户必须在 EDX 中传递一个有效的 32 位的偏移量。

INT 31H	功能 0210H	DPMI 版本 1.0
	取扩充处理器异常处理程序向量(保护方式)	

返回用户保护方式处理程序地址,该处理程序用于处理保护方式异常

调用寄存器： AX 0210H

BL 异常号(00H—1FH)

返回寄存器： 如果成功,进位标志清零

CX：(E)DX 异常处理程序的选择器：偏移量

如果失败，进位标志置位

AX 8021H 无效值

注释：

DPMI 1.0 的用户应使用本功能，而不是 INT 31H 功能 0202H。

INT 31H	功能 0211H	DPMI 版本 1.0
	取扩充处理器异常处理程序向量（实模式）	

返回用户实模式处理程序地址，该处理程序用于处理实模式异常

调用寄存器： AX 0211H

BL 异常号 (00H—1FH)

返回寄存器： 如果成功，进位标志清零

CX：(E)DX 异常处理程序的选择器:偏移量

如果失败,进位标志置位

AX 8021H 无效值

注释：

CX：(E)DX 是一个选择器:偏移量对,而不是段地址:偏移量。

INT 31H	功能 0212H	DPMI 版本 1.0
	设置扩充处理器异常处理程序向量(保护方式)	

设置用户保护方式处理程序的地址,该处理程序用于处理指定保护方式异常

调用寄存器： AX 0212H
BL 异常号(00H—1FH)
CX：(E)DX 异常处理程序的选择器：偏移量

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误代码
8021H 无效值
8022H 无效选择器

注释：

DPMI 1.0 的用户应使用本功能，而不是 INT 31H 功能 0203H。

INT 31H	功能 0213H	DPMI 版本 1.0
	设置扩充处理器异常处理程序向量（实模式）	

设置用户保护方式处理程序的地址，该处理程序用于处理指定实模式异常

调用寄存器： AX 0213H
BL 异常号（00H—1FH）
CX：(E)DX 异常处理程序的选择器：偏移量

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误代码
8021H 无效值
8022H 无效选择器

注释：

指定地址为保护方式处理程序地址。实模式异常处理程序在实模式切换后传递控制给保护方式处理程序。

INT 31H	功能 0300H	DPMI 版本 0.9
	模拟实模式中断	

模拟一个实模式下的中断

调用寄存器： AX 0300H
BL 中断号
BH 标志（全为零）
CX 从保护方式栈拷贝到实模式栈的字数
ES：(E)DI 实模式寄存器数据结构的选择器：偏移量

返回寄存器： 如果成功，进位标志清零
ES：(E)DI 已修改的实模式寄存器数据结构的选择器：偏移量

如果失败，进位标志置位

AX 错误代码

8012H 线性内存不可用

8013H 物理内存不可用

8014H 后备存储器不可用

8021H 无效值

注释：

本功能将控制传递给由实模式中断向量指定的实模式中断处理程序。CS：IP 值被忽略。SS：SP 值被实模式处理程序使用，除非它为零；在此情况下，DPMI 主机提供一个实模式栈。指定标志压入实模式中断处理程序的 IRET 帧，中断和追踪标志被清除。段寄存器值必须为实模式段地址，而不是选择器。即使在 80386 上运行，16 位主机也不能要求传递 32 位寄存器的高位字或 FS 和 GS 寄存器。从中断处理程序返回后，数据结构包括由实模式中断处理程序返回的值。

实模式寄存器结构如表 9.5 设置：

表 9.5 实模式寄存器结构

偏移量	内 容
0000H	(E)DI
0004H	(E)SI
0008H	(E)BP
000CH	保留，应为零
0010H	(E)BX
0014H	(E)DX
0018H	(E)CX
001CH	(E)AX
0020H	CPU 标志
0022H	ES
0024H	DS
0026H	FS
0028H	GS
002AH	IP(未使用)
002CH	CS(未使用)
002EH	SP
0030H	SS

INT 31H	功能 0301H	DPMI 版本 0.9
	调用带段间返回帧的实模式过程	

模拟一个实模式过程的段间调用

调用寄存器： AX 0301H

BH　　标志（全为零）
CX　　从保护方式栈拷贝到实模式栈的字数
ES：(E)DI　实模式寄存器数据结构的选择器:偏移量

返回寄存器：　如果成功，进位标志清零
ES：(E)DI 已修改的实模式寄存器数据结构的选择器:偏移量
如果失败，进位标志置位
AX　　错误代码
8012H　线性内存不可用
8013H　物理内存不可用
8014H　后备存储器不可用
8021H　无效值

注释：

本功能将控制传递给由 CS：IP 值指定的实模式过程。SS：SP 值被实模式处理程序使用，除非它为零；在此情况下，DPMI 主机提供一个实模式栈。段寄存器值必须为实模式段地址，而不是选择器。即使在 80386 上运行，16 位主机也不能要求传递 32 位寄存器的高位字或 FS 和 GS 寄存器。从此过程返回后，数据结构包括由实址方式过程返回的值。

实模式寄存器结构如表 9.6 设置：

表 9.6　实模式寄存器结构

偏移量	内　容
0000H	(E)DI
0004H	(E)SI
0008H	(E)BP
000CH	保留；应为零
0010H	(E)BX
0014H	(E)DX
0018H	(E)CX
001CH	(E)AX
0020H	CPU 标志
0022H	ES
0024H	DS
0026H	FS
0028H	GS
002AH	IP
002CH	CS
002EH	SP
0030H	SS

指定过程必须执行 RETF 指令返回。

INT 31H	功能 0302H 调用带 IRET 帧的实模式过程	DPMI 版本 0.9

模拟一个实模式过程的段间调用，该段间调用标志被压入栈中

调用寄存器： AX 0302H
BH 标志（全为零）
CX 从保护方式栈拷贝到实模式栈的字数
ES：(E)DI 实模式寄存器数据结构的选择器：偏移量

返回寄存器： 如果成功，进位标志清零
ES：(E)DI 已修改的实模式寄存器数据结构的选择器：偏移量
如果失败，进位标志置位
AX 错误代码
8012H 线性内存不可用
8013H 物理内存不可用
8014H 后备存储器不可用
8021H 无效值

注释：

本功能将控制传递给由 CS：IP 值指定的实模式过程。SS：SP 值被实模式处理程序使用，除非它为零；在此情况下，DPMI 主机提供一个实模式栈。标志值压入实模式中断处理程序的 IRET 帧；中断和追踪标志被清。段寄存器值必须为实模式段地址，而不是选择器。即使在 80386 上运行，16 位主机也不能要求传递 32 位寄存器的高位字或 FS 和 GS 寄存器。从过程返回后，数据结构包括由实模式过程返回的值。

实模式寄存器结构如表 9.7 设置：

表 9.7 实模式寄存器结构

偏移	内 容
0000H	(E)DI
0004H	(E)SI
0008H	(E)BP
000CH	保留，应为零
0010H	(E)BX
0014H	(E)DX
0018H	(E)CX
001CH	(E)AX
0020H	CPU 标志
0022H	ES
0024H	DS
0026H	FS
0028H	GS
002AH	IP
002CH	CS
002EH	SP
0030H	SS

指定过程必须执行 IRET 或 RETF（2）指令返回

INT 31H	功能 0303H 分配实模式回调地址	DPMI 版本 0.9

返回一唯一实模式段地址：偏移量作为一个实模式回调。它将把控制从实模式传递到一个保护方式过程

调用寄存器： AX 0303H
DS：(E)SI 要调用的保护方式过程的选择器:偏移量
ES：(E)DI 当调用回调例程时，由实模式寄存器数据结构使用的 32H 字节缓冲区的选择器：偏移量

返回寄存器： 如果成功，进位标志清零
CX：DX 实模式回调的段地址：偏移量
如果失败，进位标志置位
AX 8015H 回调不可用

注释：

要求 DPMI 主机给每一用户提供最少 16 个回调地址。然而，回调地址是有限资源，不必要时应释放。

INT 31H	功能 0304H 释放实模式回调地址	DPMI 版本 0.9

释放由 INT 31H 功能 0303H 分配的回调地址

调用寄存器： AX 0304H
CX：DX 要释放的实模式回调的地址

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 8024H 无效回调地址

注释：

实模式回调是一种有限资源，不必要时，用户应将其释放。

INT 31H	功能 0305H 取状态保存/恢复地址	DPMI 版本 0.9

返回用来保存当前任务的寄存器的两个例程的地址

调用寄存器： AX 0305H

返回寄存器： 进位标志清零
AX 缓冲区字节大小
BX：CX 保存/恢复功能的实模式地址
SI：(E)DI 保存/恢复功能的保护方式地址

注释：

实模式的地址是一个例程的地址，此例程由实模式调用，用来保存或恢复保护方式寄存器的状态；保护方式地址是一个由保护方式调用的例程地址，用来保存或恢复实模式寄存器的状态。

这些例程不需要调用，除非用户正在使用原始切换服务。功能 0300H，0301H 和 0302H 不需要这些例程。

在调用例程时，可设置 AL 为零来保存状态；为 1 来恢复状态。设置 ES：（E）DI 为状态保存缓冲区的地址，可做一个远调用来执行适当例程。缓冲区必须至少同 AX 中返回值一样大。一个 DPMI 主机不可能要求保存寄存器，那么本功能可能返回大小为零。对保存/恢复例程的调用依然安全，但什么也没做。

INT 31H	功能 0306H 取原始方式的切换地址	DPMI 版本 0.9

返回例程地址，此例程可被调用来执行低级方式切换

调用寄存器： AX 0306H

返回寄存器： 进位标志清零

BX：CX 实模式/保护方式切换例程地址

SI：(E)DI 保护方式/实模式切换例程地址

注释：

实模式到保护方式切换例程只能在实模式调用；保护方式到实模式切换例程只能在保护方式调用。用下列参数通过远跳转进入方式切换例程：

AX	新的 DS 值
CX	新的 ES 值
DX	新的 SS 值
（E）BX	新的（E）SP 值
SI	新的 CS 值
（E）DI	新的（E）IP 值

方式切换后，（E）AX、（E）BX、（E）CX、（E）DX、（E）DI 和（E）SI 的值没定义。（E）BP 在方式切换后保存。在 80386 和 80486 的 CPU 寄存器，FS 和 GS 设置为零。当请求方式切换时，如果中断被停用，那么 DPMI 主机不能再使用它们；用户必须在方式切换后做启用中断。

当切换到保护方式时，如果指定选择器值无效，则发生异常情况。

如果必须保存寄存器状态，那么一定要使用通过功能 0305H 可访问的功能。

INT 31H	功能 0400H 取版本号	DPMI 版本 0.9

返回由主机支持的 DPMI 规范的版本号和关于主机的性能的附加信息

调用寄存器： AX 0400H

返回寄存器： 进位标志清零

AH DPMI 主版本号

AL DPMI 次版本号

BX 标志 0 位 0=16 位设备

1=32 位设备

1 位 0=对于反射中断 CPU 返回到虚拟 86 方式

1=对于反射中断 CPU 返回到实模式

2 位 0=不支持虚拟内存

1=支持虚拟内存

3-15 位 保留

CL 处理机类型

02H 80286

03H 80386

04H 80486

DH 虚拟主 PIC 基本中断当前值

DL 虚拟次 PIC 基本中断当前值

注释：

版本 0.9 显示 AH=0，AL=3AH（十进制 90）；版本 1.0 显示 AH=1，AL=0。

INT 31H	功能 0401H 取 DPMI 功能	DPMI 版本 1.0

返回有关 DPMI 主机的信息

调用寄存器： AX 0401H

返回寄存器： 如果成功，进位标志清零

AX 功能标志

0 位 0=不支持页面访问/重写

1=支持页面访问/重写

1 位 0=不支持异常再启动能力

1=支持异常再启动能力

2 位 0=不支持装置映象

1=支持装置映象

3 位 0=不支持常规内存映象

1＝支持常规内存映象

4 位 0＝不支持请求填零

1＝支持请求填零

5 位 0＝不支持写保护用户

1＝支持写保护用户

6 位 0＝不支持写保护主机

1＝支持写保护主机

CX 0

DX 0

ES：(E)DI 128 字节主机信息缓冲区的选择器：偏移量

如果失败，进位标志置位

注释：

在版本 0.9 下，本功能调用常失败。

页面访问/重写是主机维持访问页面和页面重写标志的能力，用功能 0506H 读，功能 0507H 写。如果 DPMI 主机不能提供请求分页式虚拟存储器，这个功能必须得到支持。否则，它是可选的，用户有能力读或写由主机维持的虚拟访问页和页面重写标志写。

异常再启动是指如果用户的异常处理程序改正引起异常的原因并返回后，重新启动主机内核中的一条故障指令的能力。

装置映象表明是否支持功能 0508H。常规内存映象表明是否支持功能 0509H。

请求填零如果得到支持，意味着所有提交页在创建时均初始为零；如果不支持，页的内容不定。

写保护主机是用来保护页、防止主机写操作产生页错误的能力。主机信息缓冲区定义如表 9.8。

表 9.8 主机信息缓冲区格式

偏移量	长 度	内 容
0	字节	主机主版本号
1	字节	主机次版本号
2	126 字节	标识主机的以空结束字符串

主机的主次版本是 OEM 指定，不是 DPMI 版本。

INT 31H	功能 0500H 取自由内存信息	DPMI 版本 0.9

返回可用内存信息

调用寄存器： AX 0500H

ES：(E)DI 指向 48 字节缓冲区的选择器：偏移量

返回寄存器： 进位标志清零

注释：

本功能在版本 1.0 中被功能 050BH 代替。

缓冲区如表 9.9 定义。

表 9.9 缓冲区格式

偏移量	长度	内 容
00H	双字	最大可用自由内存块字节数（如果内存将要分配且剩余的未加锁，则为可分配的邻片线性内存的最大块）
04H	双字	最大未加锁可分配页的字节数，为偏移 0 的值除以页大小所得。如果不支持，此值为 FFFFFFFFH
08H	双字	最大加锁分配页的页数（可分配和加锁的最大块）。如果不支持，此值为 FFFFFFFFH
0CH	双字	线性地址空间字节数，包括已分配的线性空间。如果不支持，此值为 FFFFFFFFH
10H	双字	未加锁的页的总数。这些页可翻阅且可包括任何自由页。如果不支持，此值为 FFFFFFFFH
14H	双字	自由页的总数（当前未用物理页数）。如果不支持，此值为 FFFFFFFFH
18H	双字	物理页总数。包括自由、加锁、未加锁页。如果不支持，此值为 FFFFFFFFH
1CH	双字	自由线性空间页数。如果不支持，此值为 FFFFFFFFH
20H	双字	页面调度/文件分区页数。如果不支持，此值为 FFFFFFFFH
24H	12 字节	保留，所有字节设置为 FFH

页大小可调用功能 0604H 得到。

INT 31H	功能 0501H 分配内存块	DPMI 版本 0.9

分配一线性内存块

调用寄存器： AX 0501H

BX：CX 块大小字节数，不能为零

返回寄存器： 如果成功，进位标志清零

BX：CX 已分配内存块的线性地址

SI：DI 内存块句柄

如果失败，进位标志置位

AX 错误代码

8012H 线性内存不可用

8013H 物理内存不可用

8014H 后备存储器不可用

8016H 句柄不可用

8021H 无效值

注释：

块要保证至少是节对准的。如果支持虚拟内存，提交的已创建的页是未加锁的。未加锁

页可用功能 0600H 加锁。未分配描述符，分配和初始化需要访问块的描述符是用户的责任。

在许多 DPMI 主机下，分配是页 Granular；如果请求 n 页加上 1 个字节，就分配 n+1 页。因此内存最好以 Granular 单元分配。可调用功能 0604H 来获得 Granular 单元；如果失败，推荐值为 4K。

INT 31H	功能 0502H 释放内存块	DPMI 版本 0.9

释放由功能 0501H 或 0504H 分配的内存块

调用寄存器： AX 0502H
SI：DI 内存块句柄

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 8023H 无效句柄

注释：

本功能正确释放提交页、未提交页和映象页。它不能释放映象到块中的描述符。在释放线性内存块之前应先释放描述符。

INT 31H	功能 0503H 改变内存块的大小	DPMI 版本 0.9

改变由功能 0501H 或 0504H 分配的内存块大小

调用寄存器： AX 0503H
BX：CX 块的新大小字节数（必须非零）
SI：DI 内存块句柄

返回寄存器： 如果成功，进位标志清零
BX：CX 新的块线性地址
SI：DI 新的内存块句柄
如果失败，进位标志置位
AX 错误代码
8012H 线性内存不可用
8013H 物理内存不可用
8014H 后备存储器不可用
8016H 句柄不可用
8021H 无效值
8023H 无效句柄

注释：

本功能使旧句柄无效，旧句柄不再使用。当增大块大小时，提交创建的页。当减小块大

小时，不用页被正确释放。更新映象内存块的描述符是用户的责任。

INT 31H	功能 0504H	DPMI 版本 1.0
	分配线性内存块	

分配一按页对准的线性地址空间

调用寄存器：
- AX 0504H
- EBX 需要的按页对准的线性地址。若调用程序不要求，设为零。
- ECX 块大小字节数（不能为零）
- EDX 标志
  - 0 位 0 建立未提交的页
  - 1 建立已提交的页
  - 1—31 位 保留

返回寄存器：
- 如果成功 进位标志清零
  - EBX 内存块的线性地址
  - ESI 内存块句柄
- 如果失败，进位标志置位
  - AX 错误代码
    - 8011H 不支持本功能
    - 8012H 线性内存不可用
    - 8013H 物理内存不可用
    - 8014H 后备存储器不可用
    - 8016H 句柄不可用
    - 8021H 无效值
    - 8025H 线性地址不是页对准

注释：

16 位主机不支持本功能，但 32 位主机的 16 位用户能用本功能。

功能 0501H 也分配线性内存，但不能保证是页对准、建立未提交页或在一指定线性地址分配一个块。

INT 31H	功能 0505H	DPMI 版本 1.0
	改变线性内存块大小	

改变以前由功能 0504H 分配的内存块大小

调用寄存器：
- AX 0505H
- ESI 内存块句柄
- ECX 新的块大小字节数（不能为零）
- EDX 标志

位 0 0 建立未提交的页
1 建立已提交的页
位 1 0 不更新段描述符
1 更新段描述符
位 2—31 保留

如果 EDX 位 1 设置
ES：EBX 包含一选择器数组的缓冲区选择器偏移地址，每选择器为一个字长（16 Bits）
EDI 数组中选择器的个数

返回寄存器： 如果成功，进位标志清零
EBX 包括新内存块线性地址
ESI 包括新内存块的句柄
如果失败，进位标志置位
AX 错误码
8001 不支持的功能
8012 线性内存不可用
8013 物理内存不可用
8014 后备存储器不可用
8016 句柄不可用
8021 大小无效
8023 句柄无效

注释：

16 位设备不支持该功能，但 32 位主机的 16 位用户可以使用该功能。返回后，旧句柄失效，不能再使用。

如果操作失败，块大小和基址不会改变。

如果块增大，基地址有可能改变。如果块变小，块尾部的页面被释放，基址不改变。

如果基地址改变且 EDX 的位 1 置位，当段全在内存块中时，更新表中选择器描述符会被修改。如果基址在内存块中，则上扩展段也在其中；如果（基址 +极限－1）在内存块中，则下扩展段也在其中。若必须移动块，则在块移动过程中不向用户传硬件中断。功能 0503H 也用于改变内存块大小，但它无需页对准块，不产生未提交页，不更新描述符。

INT 31H	功能 0506H 取页面属性	DPMI 版本 1.0

返回由功能 0504H 分配的线性内存地址的一页或多页的属性

调用寄存器： AX 0506H
ESI 内存块句柄
EBX 页面的内存块或多页面的第一页的基偏移量
ECX 页面数

ES：EDX 接收页面属性的缓冲区的选择器：偏移量

返回寄存器： 如果成功，进位标志清零；缓冲区更新。

如果失败，进位标志置位

AX 错误码，

8001H 不支持的功能

8023H 无效句柄

8025H 偏移量不在指定块内

注释：

16 位主机不支持该功能，但 32 位主机的 16 位用户可以使用。

如果 EBX 不是页对准，它将被下舍入到下一个页面的下界。

页面属性缓冲区是每页一个字的数组，格式如下：

位 0—2	页类型
	0—未提交页
	1—提交页
	2—映象页
位 3	0—只读页
	1—可读/写页
位 4—6	0—访问过的页，页面重写标志位不可用
	1—页面未被访问，页面未重写
	3—页面已被访问，页面未重写
	5—页面未被访问，页面已重写
	7—页面已被访问，页面已重写
位 7—15	保 留

INT 31H	功能 0507H	DPMI 版本 1.0
	设置页面属性	

可以设置由功能 0504H 分配的线性内存块中一个或多个页面的属性

调用寄存器： AX 0507H

ESI 内存块句柄

EBX 属性要改变的页在内存块中的偏移量

ECX 页数

ES：EDX 包含页面属性的缓冲区的选择器偏移地址

返回寄存器： 如果成功，进位标志清零

如果失败，进位标志置位

AX 错误码

8001H 不支持的功能

8002H 页状态错误

8013H 物理存贮器不可用

8014H 后备存贮器不可用
8021H 一页或多页的属性字的位 0-2 值非法
8023H 无效句柄
8025H 线性地址不在内存块内

注释：

16 位主机不支持该功能，但 32 位主机的 16 位用户可以使用。

可以从提交页或映射页内创建未提交页。提交页也可以从未提交页或映射页创建。

属性字的格式如下：

位 0—2	页类型
	0—将页类型改为非提交页
	1—将页类型改为提交页
	2—无效
	3—不改变页类型而修改属性
	4—7 无效
位 3	0—设置页只读
	1—设置页可读/写
位 4—6	0—不修改访问过的页/页面重写标志位
	1—将页面标记为未被访问，未重写
	3—将页面标记为已被访问，未重写
	5—将页面标记为未被访问，已重写
	7—将页面标记为已被访问，已重写
位 7—15	保留，应为零

当且仅当主机支持已访问页面/页面重写时，已访问页面/页面重写标志位才能修改。

INT 31H 功能 0508H DPMI 版本 1.0

内存块设备映象

把分配给设备的物理地址映象到由功能 0504H 分配的内存块的线性地址

调用寄存器： AX 0508H
ESI 内存块句柄
EBX 待映象页面（必须是页对准）在内存块内的偏移量
ECX 待映射的页面数量
EDX 设备的物理地址（必须是页对准）

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误码
8001H 不支持的功能
8003H 无效的设备地址

8023H 无效句柄

8025H 线性地址不在内存块内或不是页对准

注释：

16 位主机不支持这功能，但 32 位主机的 16 位用户可以使用该功能。

这个功能是可选的；需要这一功能的应用程序或 DOS Extenders 并不依从 DPMI。

提交或映象页面将由主机变为非提交或非映象页面。

与 0800H 不同的是，这个功能支持在现存内存块内的物理设备映象（这样，32 位程序可以用段内指针访问设备内），并且可以映射 1M 以下的地址，如显示适配器刷新缓冲区。

INT 31H	功能 0509H	DPMI 版本 1.0
	内存块常规内存映象	

将 1M 以下的物理地址映象到由功能 0504H 分配的内存块的线性地址

调用寄存器： AX 0509H

ESI 内存块句柄

EBX 待映象页面（必须是页对准）在内存块中的偏移量

ECX 要映象的页数

EDX 常规存贮器的线性地址（必须是页对准）

返回寄存器： 如果成功，进位标志清零

如果失败，进位标志置位

AX 错误码

8001H 不支持的功能

8003H 无效的常规内存的地址

8023H 无效句柄

8025H 线性地址不在常规内存块或不是页对准

注释：

16 位主机不支持这功能，但 32 位主机的 16 位用户可以使用该功能。这一功能是可选的；需要这一功能的应用程序或 DOS Extenders 并不依从提交或映象页面将由主机变为非提交或非映象页面。

用功能 0100H 或通过翻译服务调用 DOS INT 21H 功能 48H，用户只可映象已经拥有的常规存贮器。

INT 31H	功能 050AH	DPMI 版本 1.0
	取内存块的大小和基地址	

返回由功能 0501H 或 0504H 分配到的内存块的大小

调用寄存器： AX 050AH

SI：DI 内存块句柄

返回寄存器： 如果成功，进位标志清零

SI：DI 内存块的大小（字节）

BX：CX 内存块的基地址

如果失败，进位标志置位

AX 8023H 无效句柄

注释：

这个功能在 DPMI 版本 0.9 中失败。

INT 31H	功能 050BH	DPMI 版本 1.0
	取内存信息	

返回可用的物理和虚拟内存的信息

调用寄存器： AX 050BH

ES：(E)DI 128 字节缓冲区的选择器偏移量

返回寄存器： 如果成功，进位标志清零，缓冲区内容更新

如果失败，进位标志置位

注释：

这个功能在 DPMI 版本 0.9 中失败。

缓冲区被格式化为如表 9.10 的格式。

表 9.10 缓冲区格式

偏移量	长度	含 义
00H	双字	主机控制的分得的物理内存总字节量
04H	双字	主机控制的分得的虚拟内存总字节量
08H	双字	主机控制的可用的虚拟内存总字节量
0CH	双字	本虚拟机分配到的虚拟内存总字节量
10H	双字	本虚拟机可用的虚拟内存字节量
14H	双字	本用户分配到的虚拟内存字节量
18H	双字	本用户可用的虚拟内存字节量
1CH	双字	本用户锁定的内存总字节量
20H	双字	本用户锁定的内存的最大字节量
24H	双字	本用户可利用的最高线性地址
28H	双字	最大可用自由内存块字节数
2CH	双字	最小分配单元字节数
30H	双字	分配对准单元字节数
34H	76 字节	保 留

INT 31H	功能 0600H	DPMI 版本 0.9
	锁定线性域	

锁定指定线性地址范围

调用寄存器：　AX　　0600H
　　BX：CX　线性地址起始位置
　　SI：DI　域的字节大小

返回寄存器：　如果成功，进位标志清零
　　如果失败，进位标志置位
　　AX　错误码
　　8013H　物理存贮器不可用
　　8017H　锁定计数越界
　　8025H　无效的线性地址

注释：

如果本功能返回失败，将不锁定任何域。如果此域两头各和某页的一部分重叠，此页面将被锁定。此功能可被多次调用；它保存着一个锁定计数。

INT 31H	功能 0601H 对线性域解锁	DPMI 版本 0.9

开启由功能 0600H 锁定的域

调用寄存器：　AX　　0601H
　　BX：CX　线性域的起始地址
　　SI：DI　域的大小（字节）

返回寄存器：　如果成功，进位标志清零
　　如果失败，进位标志置位
　　AX　错误码
　　8002H　页未锁定
　　8025H　地址无效

注释：

如果操作失败，不开启任何域。直到锁定计数为 0 时，域才能被解锁。例如，调用本功能的次数必须与调用功能 0600H 的次数相同。

INT 31H	功能 0602H 标记实址方式域为可分页区域	DPMI 版本 0.9

标记一个 1M 以下的内存域为磁盘可分页区域

调用寄存器：　AX　　0602H
　　BX：CX　起始地址
　　SI：DI　域的字节大小

返回寄存器：　如果成功，进位标志清零

如果失败，进位标志置位

AX 错误码

8002H 域已被标记为可分页区域

8025H 地址超过 1M 边界

注释：

如果操作失败，未标志任何内存为可分页区域。这功能不同于锁定操作，如果此域两头各和某页的一部分重叠，重叠页面不被标记为可分页，且多次调用没有结果。

在终止时，用户程序应调用功能 0603H 重新锁定此内存区。DPMI 1.0 主机在用户程序终止时会自动执行这一操作，而 DPMI 0.9 不会。若未成功地执行这一操作，当其他程序在这空间运行时，将导致致命的页错误。

用户只应标记自己拥有的空间，也即通过功能 0100H 或直接调用 DOS 内存分配服务程序而得到的空间。值得注意的是，由 DOS 或 DPMI 主机拥有的区域不能标记为可分页区。如果这样做，将会导致致命的页错误。

INT 31H	功能 0603H 重新加锁实模式域	DPMI 版本 0.9

重新锁定由功能 0602H 标记为可分页的内存区

调用寄存器： AX 0603H

BX：CX 重新锁定的开始地址

SI：DI 加锁区域字节大小

返回寄存器： 如果成功，进位标志清零

如果失败，进位标志置位

AX 错误码

8002H 区域未被标记为可分页

8013H 物理内存不可用

8025H 区域超过 1M 边界

注释：

如果出现错误，则未重新分配任何区域。与区域两端重叠的页不会重新锁定。

INT 31H	功能 0604H 取页大小	DPMI 版本 0.9

返回单个存储器页字节数

调用寄存器： AX 0604H

返回寄存器： 如果成功，进位标志清零

BX：CX 页字节大小

如果失败，进位标志置位

AX 8001H 不支持的功能

注释：

这功能返回基本内存分配块的大小。

INT 31H	功能 0702H 将页标记为请求调页候选者	DPMI 版本 0.9

标记一个或一组要被放置在调页候选者表首部的页面。

调用寄存器： AX 0702H
BX：CX 线性地址开始位置
SI：DI 区域字节大小

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 8025H 域未被分配

注释：

本功能并不一定将所指页面调出内存，这由主机决定。非完整页面不作标记。

如果用户已经知道在一段时间内，不会访问某一区域的内容，且区域占用的内存可被更好地使用，就可用这一功能。

INT 31H	功能 0703H 删除页内容	DPMI 版本 0.9

删除指定线性地址区域内的内容，这样，就可免于不必要的调页

调用寄存器： AX 0703H
BX：CX 线性地址开始位置
SI：DI 删除区域的字节大小

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位。
AX 8025H 区域未被分配

注释：

非完整页和锁定页的内容不可删除。删除后的页内容未予定义。

INT 31H	功能 0800H 物理地址映象	DPMI 版本 0.9

将物理地址转换成线性地址

调用寄存器： AX 0800H
BX：CX 物理地址

SI：DI　区域字节大小

返回寄存器：　如果成功，进位标志清零

BX：CX　线性地址

如果失败，进位标志置位

AX　错误码

8003H　地址落于 DPMI 主机内存区域

8021H　地址小于 1M 边界

注释：

本功能只由需直接访问 1M 以上内存映象设备的用户使用。用户要对访问区域分配和初始化描述符。

INT 31H	功能 0801H 释放物理地址映射	DPMI 版本 0.9

把功能 0800H 获得的物理地址释放到线性地址

调用寄存器：　AX　0801H

BX：CX　待释放的线性地址

返回寄存器：　如果成功，进位标志清零

如果失败，进位标志置位

AX　8025H　无效地址

注释：

当用户结束使用功能 0800H 映象的设备时应使用本功能。

INT 31H	功能 0900H 取并禁止虚拟中断标志	DPMI 版本 0.9

禁止虚拟中断标志，返回原来的标志状态。

调用寄存器：　AX　0900H

返回寄存器：　如果成功，进位标志清零

AL　原来状态

0　虚拟中断原来被禁止

1　虚拟中断原来被允许

注释：

这个功能调用不修改 AH 的内容，用户可以再次调用 INT 31H 来恢复中断标志的原来状态，而不必知道它。

用户可使用 CLI 关闭中断，但 DPMI 主机可能俘获了该指令。用户须假定 CLI 指令执行很慢。

INT 31H	功能 0901H 取且允许虚拟中断状态	DPMI 版本 0.9

允许虚拟中断标志，并返回原来的标志状态

调用寄存器： AX 0901H

返回寄存器： 如果成功，进位标志清零

AL 原来状态

0 虚拟中断原来被禁止

1 虚拟中断原来被允许

注释：

这个功能调用不修改 AH 的内容，用户可以再次调用 INT 31H 恢复中断标志的原来状态，而不必知道它。

用户可用 STI 打开中断，但 DPMI 主机可能俘获了该指令。用户必须假定 CLI 指令执行很慢。

INT 31H	功能 0902H 取虚拟中断状态	DPMI 版本 0.9

返回当前虚拟中断标志的状态

调用寄存器： AX 0902H

返回寄存器： 进位标志清零

AL 虚拟中断状态

0 虚拟中断原来被禁止

1 虚拟中断原来被允许

注释：

这个功能可代替 PUSHF 指令，因为 PUSHF 返回物理中断标志，而不是虚拟化的各用户的中断标志。

INT 31H	功能 0A00H 取专用 API 入口点	DPMI 版本 0.9

返回用于访问专用 DPMI 扩充功能的入口地址

调用寄存器： AX 0A00H

DS：(E)SI ASCIIZ 字符串的选择器：偏移量，用来标识专用 DPMI

返回寄存器： 如果成功，进位标志清零

ES：(E)DI 扩充的 API 入口点的选择器：偏移量

其他寄存器也可能被修改

如果失败，进位标志置位

AX 8001H 扩充功能未找到

注释：

用户必须对入口地址使用远调用。传递的字符串区分大小写。

DPMI 1.0的用户应使用 INT 2FH 功能168AH 来代替本功能。

INT 31H	功能0B00H 设置调试观察点	DPMI 版本0.9

在指定位置设置调试观察点

调用寄存器： AX 0B00H

BX:CX 线性地址

DL 观察点的大小（1，2或4字节）

DH 观察点类型

0 可执行

1 写

2 读/写

返回寄存器： 如果成功，进位标志清零

BX 观察点句柄

如果失败，进位标志置位

AX 错误码

8016H 观察点太多

8021H 观察点大小或类型无效

8025H 线性地址未被映象或界对齐错误

注释：

在 DPMI 1.0中返回的句柄在0到14之间，在 DPMI 0.9中，此范围无限制。

INT 31H	功能0B01H 清除调试观察点	DPMI 版本0.9

清除由功能0B00H 设置的观察点并释放句柄

调用寄存器： AX 0B01H

BX 观察点句柄

返回寄存器： 如果成功，进位标志清零

如果失败，进位标志置位

AX 8023H 句柄无效

注释：

这个功能在程序终止之前调用。

INT 31H	功能0B02H 取调试观察点的状态	DPMI版本0.9

返回以前由功能0B00H设置的调试观察点的状态

调用寄存器： AX 0B02H
BX 观察点的句柄

返回寄存器： 如果成功，进位标志清零
AX 状态
位0 = 0 没有遇到观察点
= 1 遇到了观察点
位1-15 保留
如果失败，进位标志置位
AX 8023H 句柄无效

注释：

使用功能0B03H清除观察点状态，但不释放观察点。

INT 31H	功能0B03H 调试观察点复位	DPMI版本0.9

观察点复位，以使以后调用功能0B02H时不会遇到观察点

调用寄存器： AX 0B03H
BX 观察点句柄

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 8023H 句柄无效

注释：

这个功能应在遇到观察点后调用

INT 31H	功能 0C00H 给常驻服务提供程序装入回叫功能	DPMI版本1.0

本功能用于，当同一虚拟机上的用户装入或终止时，常驻服务提供程序随时能从主机获得通知

调用寄存器： AX 0C00H
ES:(E)DI 指向40字节缓冲区的选择器：偏移量
缓冲区的结构如表9.11的格式。

表9.11 缓冲区格式

偏移量	长 度	内 容
00H	8字节	16位数据段的描述符
08H	8字节	16位代码段的描述符
10H	2字节	16位回叫过程的偏移量
12H	2字节	保 留
14H	8字节	32位数据段的描述符
1CH	8字节	16位代码段的描述符
24H	4字节	32位回叫过程的偏移量

返回寄存器： 如果成功，进位标志清零

如果失败，进位标志置位

AX 错误码

8015H 回叫不可用

8021H 访问权限或类型字节无效，偏移量超出段界限

8025H 描述符标记了非法的线性地址范围

注释：

只有在 DPMI 用户想要提供保护方式驻留服务时，才应使用本功能，用户接下来应调用功能0C01H 终止并驻留。只提供实模式方式驻留服务的 DPMI 用户不需要本功能。

可以使用功能000BH 方便地获取代码和数据描述符的拷贝。如果用户只提供了16位服务和32位服务中的一种，那么，未被支持的方式在缓冲区中对应的代码段描述符应设置为零。

INT 31H	功能0C01H 终止并驻留服务	DPMI 版本1.0

退出 DPMI 用户的运行，用户仍处于保护模式，不释放用户所分配的保护模式内存和实模式内存

调用寄存器： AX 0C01H

BL 返回代码

DX 需保留的 DOS 内存节数

返回寄存器： 无

注释：

只有当 DPMI 用户要对另外 DPMI 保护方式用户提供驻留服务时，使用本功能。

如用户只想为实模式程序提供驻留服务，应使用0300H 直接调用 DOS 的 INT 21H 功能31H。

DX 的值是 DOS 内存节数；DPMI 主机不释放用户所分配的保护模式内存。DX 必须为0或最小为6的数。如果 DX 为0，将使用 DOS 的 INT 21H 功能4CH 代替功能31H。

如果用户先前没有调用过功能0C00H，用户将被简单地终止。

INT 31H	功能0D00H	DPMI版本1.0
	分配共享内存	

分配一块可供DPMI用户共享的内存

调用寄存器： AX 0D00H

ES:(E)DI 指向共享内存分配请求结构的选择器：偏移量格式如表9.12所示。

表9.12 共享内存分配请求结构

偏移量	长 度	内 容
00H	4字节	需要分配的内存块长度（可以为0）
		（由用户设置）
04H	4字节	实际分配的内存块长度
		（由主机设置）
08H	4字节	内存块句柄
		（由主机设置）
0CH	4字节	内存块的线性地址
		（由主机设置）
10H	6字节	指向以0结尾的块的ASCII名字的32位选择器：偏移量
		（由用户设置）
16H	2字节	保 留
18H	4字节	保留；必须为0

返回寄存器： 如果成功，进位标志清零

偏移为04H，08H和0CH处的项被设置

如果失败，进位标志置位

AX 错误码

8012H 线性内存不可用

8013H 物理内存不可用

8014H 后备存贮器不可用

8016H 句柄不可用

8021H 名字太长

注释：

16位用户必须将32位偏移量的高位字设为零。包括末尾的字节0在内，内存块的名字最长为128个字节，

主机将保证向同一虚拟机上使用同名共享内存块的用户，提供相同的线性地址。用户必须使用自己分配并初始化的描述符来寻址此内存块。

对于返回的句柄值不要做任何假定；当同一用户对同一内存块再次进行分配时，返回的句柄值将不同。

第一个分配共享内存块的用户决定了此共享内存块的大小。其他用户可以使用不同的指

针分配此共享内存块，但返回的实际分配到的块大小与第一个用户的一样。

共享内存块大小为零是合法的。返回的句柄可以被功能0D02H 和0D03H 使用，但其线性地址未定义，引用它会引起页面错误。

在第一次分配时，块的前面16个字节被清零，用户以此作为“域初始化”指示器。

INT 31H	功能0D01H	DPMI 版本1.0
	释放共享内存块	

释放由功能0D00H 分配的共享内存块

调用寄存器： AX 0D01H

SI:DI 共享内存块句柄

返回寄存器： 如果成功，进位标志清零

如果失败，进位标志置位

AX 8023H 句柄无效

注释：

成功返回后，此句柄不再有效。

所有用于映象此共享内存块的描述符应由用户释放。主机为此内存块设置了两个计数器：虚拟机使用计数器和全局使用计数器。前者是在此虚拟机上成功地调用分配此共享块的次数；后者为访问此块的虚拟机计数。当虚拟机计数器为零时，此虚拟机上的用户就不能访问寻址该块。当全局计数器为零时，此块已被主机破坏。

INT 31H	功能0D02H	DPMI 版本1.0
	共享内存块加锁	

请求对共享内存块加锁，以便对它进行访问

调用寄存器： AX 0D02H

SI:DI 共享内存块的句柄

DX 标志

位0＝0 用户挂起，直到共享内存可用

＝1 如果共享内存不可用，立即返回错误

位1＝0，排它性访问请求

＝1 共用性访问请求

位2—15 保留；必须为0

返回寄存器： 如果成功，进位标志清零

如果失败，进位标志置位

AX 错误码

8004H 主机检测到死锁

8005H 请求已被功能0D03H 取消

8017H 锁定计数越界
8018H 另一用户正在对共享内存进行排它性访问
8019H 另一用户正在对共享内存进行共用性访问
8023H 句柄无效

注释：

成功的加锁表示用户拥有了访问共享内存块的权力。DPMI用户使用这种机制来同步对共享内存块的读取和修改。

排它性访问请求等价于写访问权；共用性访问请求等价于只读访问。成功的排它性访问请求将禁止以后的任何其他访问请求，包括共用性的和排它性的；成功的共用性访问请求将禁止以后的排它性访问请求。

当被挂起等待的用户仍可响应中断，如果用户中断服务例程，调用了功能0D03H，主机将结束对用户的挂起并返回。错误码8005H就是在这种情况下返回的。

不要求主机检查死锁。

INT 31H	功能0D03H 释放对共享内存块的加锁	DPMI版本1.0

释放由成功地调用功能0D02H获得的对共享内存的加锁

调用寄存器： AX 0D03H
SI:DI 共享内存块的句柄
DX 标志
位0=0 释放排它性访问请求
=1 释放共用性访问请求
位1=0 不释放挂起的访问请求
=1 释放挂起的访问请求
位2—15 保留；必须为0

返回寄存器： 如果成功，进位标志清零
如果失败，进位标志置位
AX 错误码
8002H 用户不拥有指定的加锁
8023H 无效的句柄

注释：

当调用程序结束访问共享内存时，应调用本功能，以便其他用户访问共享内存。

INT 31H	功能0E00H 取协处理器状态	DPMI版本1.0

返回关于协处理器是否存在，协处理器类型，以及主机是否支持协处理器仿真的信息

调用寄存器： AX 0E00H

返回寄存器： 如果成功，进位标志清零

AX 协处理器状态

位0＝0 用户被禁止使用数字协处理器

＝1 用户被允许使用数字协处理器

位1＝0 用户没提供协处理器仿真

＝1 用户提供了协处理器仿真

位2＝0 有数字协处理器

＝1 无数字协处理器

位3＝0 主机没提供协处理器仿真

＝1 主机提供了协处理器仿真

位4—7 处理器类型

00H 无协处理器

02H 80286

03H 80386

04H 80486

位8—15 保留

如果失败（DPMI 0.9版本），进位标志置位

注释：

如果位2与位4—7有冲突，忽略位2。

INT 31H	功能0E01H 设置处理机仿真	DPMI 版本1.0

允许或禁止虚拟机数字协助理器和向用户报告协助理器异常

调用寄存器： AX 0E01H

BX 标志

位0＝0 禁止用户使用数值协处理器

＝1 允许用户使用数值协处理器

位1＝0 用户不提供协处理器仿真

＝1 用户将提供协处理器仿真

位2—15 保留

返回寄存器：如果成功，进位标志清零

如果失败，进位标志置位

AX 8026H 无效请求

注释：

如果调用时要将 BX 的位1设为1，那么用户应该在调用前设置一个协处理器不存在错误的异常处理过程。

·MEMO·

第十章 虚拟控制程序接口 VCPI

VCPI (Virtual Control Program Interface) 是一种比较高级的程序接口，是对 EMS 4.0 规范的一个扩充，只能用在 80386 和 80486 上。VCPI 接口通常由 EMS 模拟器 (EMS emulators) 提供。DOS 扩展器 (DOS extenders) 运行时，如果检测到了 VCPI，就使用 VCPI 功能进行保护模式切换和 EMS 内存分配。现在，大多数 EMS 模拟器和 DOS 扩展器都支持 VCPI。

用户先安装 EMS 模拟器，然后运行 DOS 扩展应用程序。只有比较高层次的软件开发者才需要了解 VCPI 的存在。标准的 PC 用户没必要研究 VCPI。设计这一接口的目的，就是为了在不让用户知道和干预的情况下，协调程序的运行。但是，对保护模式应用程序开发者来说，至少应该意识到 VCPI 的存在，因为他们必需在两种环境下测试程序 (DOS，以及 DOS 与提供了 VCPI 功能的 EMS 模拟器)。对希望在其程序中直接使用 VCPI 功能的开发者来说，就需要了解 VCPI。而对那些开发 EMS 模拟器和 DOS 扩展器的人来说，就更应该掌握 VCPI，因为 VCPI 主要是为这两类程序设计的。

本章中，我们首先将说明设计 VCPI 的原因，然后总体介绍一下 VCPI 的功能，最后详细讲解 VCPI 的各项功能。

10.1 VCPI 的引入

多个程序同时使用未经 DOS 管理的资源时，就会产生冲突。在 EMS 模拟器和 DOS 扩展器之间就存在这种冲突，因为它们都使用了扩充内存和保护模式。为了解决使用这两种资源时的冲突，引入了 VCPI。

下面，让我们看看 EMS 模拟器和 DOS 扩展器是怎样工作的。

DOS 扩展器能提供保护模式，允许在 DOS 下运行大的应用程序。它在标准的 DOS 环境之上提供了一个保护模式环境，而且能直接分配扩充内存供应用程序使用。在 286，386 或 486 机器上，DOS 扩展器的具体实现细节虽然有所不同，但其处理方法基本上还是一样的。

EMS 模拟器一种用于把扩充内存模拟成扩展内存的纯软件产品。EMS 模拟器直接分配一大块扩充内存，使其在调用 EMS 的程序看来像扩展内存。在 286 机器上，EMS 模拟器调用 BIOS 的块移动功能 (INT 15H 功能 87H)，来把扩充内存中的数据拷贝到被映象的 EMS 页。在 386/486 机器上，EMS 模拟器使用处理器保护模式操作，利用它们的硬件页映象功能，避免了费时的内存拷贝。

无论是哪种机器，都存在使用扩充内存的冲突。如果没有 VCPI，用户就不能同时使用 DOS 扩充应用程序和需要 EMS 的应用程序。用户要么为了安装和拆掉 EMS 模拟器频繁地重启动机器，要么对 EMS 模拟器进行配置，使其只使用一部分扩充内存，让出一些自由空间。对用户来说，这两种方法都不方便，都不是我们所希望的。

在 386/486 机器上，由于 EMS 模拟器和 DOS 扩展器都使用保护模式，这就造成了它们之间的另一个冲突。如果用户已安装 EMS 模拟器，然后又需要运行 DOS 扩展应用程序，就

得关闭EMS模拟器或拆掉它。如果采用关闭EMS的方法，就不能使用EMS占用的内存；如果采用拆掉EMS的方法，就不得不重启动机器。

设计VCPI就是为了同时解决以上两种冲突。VCPI是386/486特用的，不能用于286。其原因是：1）在286上不存在保护模式冲突；2）由于在286上模拟EMS需进行费时的数据拷贝操作，因此用于286的EMS模拟器并不多，大多数用户使用的是EMS硬件板。

10.2　VCPI概述

VCPI由一套由EMS模拟器（称为服务器Server）提供的服务组成，并被DOS扩展器（称为客户Client）使用。从技术上讲，VCPI服务器可指同时提供了EMS 4.0和VCPI接口的任何程序；VCPI客户可指任何进行VCPI功能调用的程序。在以下内容中，我们用服务器一词来代替EMS模拟器，用客户一词来代替DOS扩展器。

VCPI功能调用有些只能在V86模式下使用；有些既可在V86模式下使用，也可在保护模式下使用；还有一个（模式切换调用）只能在保护模下使用。在V86模式下，VCPI通过标准的EMS INT 67H接口调用，在AH寄存器中放入0DEH，AL寄存器中放入VCPI功能号。在保护模式下，客户通过对服务器的一个入口进行远调用来使用VCPI功能。客户可在接口初始化过程中得到服务器的保护模式入口地址。在保护模式下调用VCPI与在V86模式下一样，需将AH设为0DEH，将AL设为VCPI功能号。

表10.1　VCPI功能汇总

功能号	可用模式	功能名
【初始化功能】		
00H	V86	VCPI存在测试
01H	V86	取保护模式接口
02H	V86	取最大物理内存地址
【内存分配功能】		
03H	V86，保护	取自由页面数（4K大小）
04H	V86，保护	分配一页面（4K大小）
05H	V86，保护	释放一页面（4K大小）
06H	V86	取页面的物理地址
【系统寄存器存取功能】		
07H	V86	读CR0寄存器
08H	V86	读调试寄存器
09H	V86	置调试寄存器
【中断控制管理功能】		
0AH	V86	取8259A中断向量映射
0BH	V86	置8259A中断向量映射
【模式切换功能】		
0CH	V86	从V86模式切换到保护模式
0CH	V86	从保护模式切换到V86模式

VCPI功能可分五类。在保护模式下，只有内存分配调用和从保护模式切换到V86模式调

用有效。VCPI 功能 0CH 同时用于两个方向的模式切换，因为 VCPI 只支持两种处理器模式——V86 模式和保护模式。下面，我们将稍微详细地介绍这五类功能。

10.2.1 接口初始化功能

DOS 扩展器开始执行时，它通过以下步骤来测试 VCPI 服务器的存在：1）测试 EMS 驱动程序的存在；2）分配一 EMS 页面以保证 EMS 模拟器处于打开状态；3）进行 VCPI 存在测试调用（VCPI 功能 00H）。为了使用 VCPI 接口必须先打开 EMS 模拟器。

客户在初始化时进行取保护模式接口调用（VCPI 功能 01H）。此调用为服务器在它自己的环境和客户的环境之间进行切换，以及客户在保护模式下对服务器进行调用，设置了必要的条件。此调用也返回了在保护模式下 VCPI 调用的服务器入口地址。

10.2.2 内存分配功能

在 VCPI 下，内存以 4K 页面为单位进行分配。注意，这是 386/486 上硬件分页的单位，而不是 EMS 接口所使用的 16K 页面单位。EMS 内存空间是一块由模拟器在安装时所分配的连续的扩充内存。这一空间可看作是由 16K 的 EMS 页面构成的一个数组（参见下图）。每一 16K EMS 页被进一步分成四个 4K 的页面（由于处理器硬件分页的需要，它们对齐在 4K 物理地址边界上）。EMS 内存空间可以两种方式进行分配：1）通过 EMS 调用以 16K 为单位分配；2）通过 VCPI 调用以 4K 为单位分配。

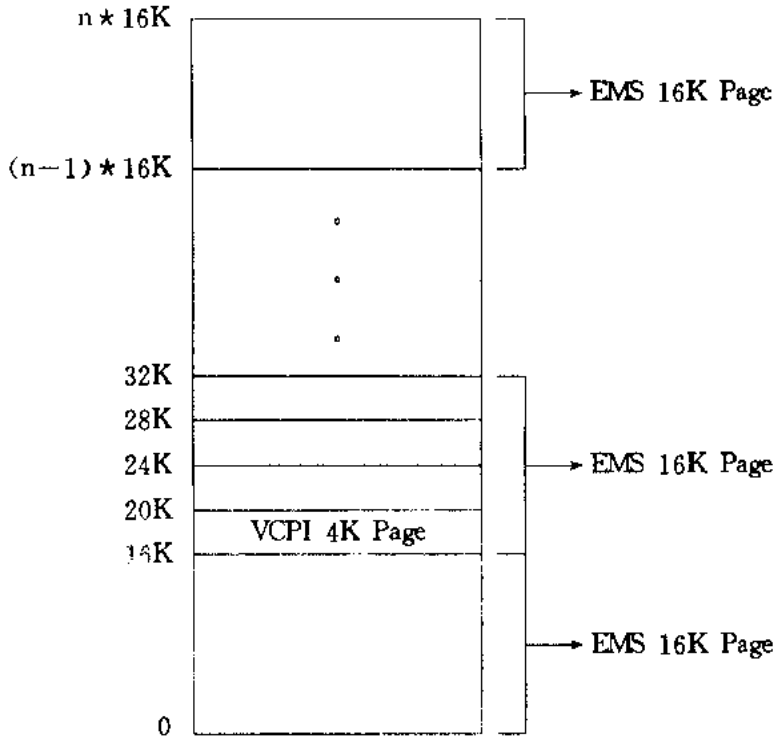


图10.1 EMS内存空间

VCPI 客户可使用其中一种方式或同时使用这两种方式，从 EMS 模拟器获得内存。被分配的页面通过其物理地址而不是句柄加以区分。由于客户建立了自己的保护模式环境，所以要处理物理地址。

取页面物理地址功能（VCPI 功能 06H）是为那些使用 EMS 接口分配内存的客户设立的。客户可以像通常那样，分配 EMS 页面并将其映射到 EMS 页面帧。然后调用功能 06H 取得构成 EMS 页面的四个 4K VCPI 页面的每一个的物理地址。

10.2.3　系统寄存器存取功能

在 V86 模式下，客户不能直接存取 386/486 的系统寄存器（CR0，CR2，CR3，GDTR，LDTR，IDTR，TR，测试寄存器和调试寄存器），否则会导致处理器异常。在 V86 模式下，其中的大多数也用不着存取。有几个寄存器在保护模式下用得着，并在模式切换时作为环境切换的一部分已进行了装载。

功能 08H 和 09H 允许客户在 V86 模式下读写处理器调试寄存器。使用调试寄存器，可在应用程序中设置多达四个的代码断点或数据观察点，并能确定调试异常的来源。多数基于 386 的调试器用到了调试寄存器。

10.2.4　中断控制管理功能

DOS 扩展器必须处理保护模式下的所有中断。由于处理硬件中断和软件中断的方式不同，DOS 扩展器需要知道哪些中断向量用于硬件中断。如果这些硬件中断仍使用 DOS 标准的中断向量，DOS 扩展器就把 IRQ0—IRQ7 重定向，以便更容易区分硬件中断和处理器异常。

客户可使用功能 0AH 和 0BH 决定哪些中断向量用于硬件中断。在实模式下，向量 08H—0FH 用于 IRQ0—IRQ7，向量 70H—77H 用于 IRQ8—IRQ15。由于中断向量 08H—0FH 也用于处理器异常，所以有些保护模式应用程序对中断控制器重新编程，使硬件中断使用不同的中断向量。

如果系统仍使用的是常规的向量映射，就允许客户对中断控制器重新编程并调用功能 0AH 告诉服务器使用了新的中断映射。如果中断控制器已被重新编程，就不允许客户修改映射。

10.2.5　模式切换功能

每当中断发生时，不管是硬件中断、处理器异常，还是比如 DOS 或 BISO 系统调用的软件中断，DOS 扩展器都要把模式切换到 V86，然后切换回保护模式。

功能 0CH 不仅是只进行模试切换，它还在服务器和客户的环境之间进行切换。

10.3　VCPI 功能调用详解

VCPI 功能 00H

VCPI 存在测试

调用寄存器：　AX　　DE00H

返回寄存器： 如果 VCPI 存在

AH＝0

BL＝VCPI 主版本号

BH＝VCPI 次版本号

如果 VCPI 不存在

AH＝非零

注释：

本功能要在 EMS 模拟器存在测试和分配一 EMS 页面后进行，以保证 EMS 驱动器被启动并且处理器切换在 V86 模式下。

VCPI 功能 01H

取保护模式接口

调用寄存器：	AX	DE01H
	ES：DI	指向客户第 0 个页表的指针
	DS：SI	指向客户 GDT 中三个描述符的指针
返回寄存器：	AH	0
	DI	指向客户第 1 个未初始化页表的指针
	EBX	保护模式代码段中，服务器入口的偏移值

注释：

调用本功能后，服务器将客户页表中映射线性地址 0 至 4MB 的那部分进行初始化。服务器并把线性地址 0 至 1MB 的那部分映射成与服务器的一样。服务器还将客户 GDT 中的三个描述符初始化。其中第一个描述符必须是代码段，服务器返回的入口地址就在此代码段内。通过这些初始化后，服务器和客户之间的切换才有可能。

VCPI 功能 02H

取最大物理内存地址

调用寄存器：	AX	DE02H
返回寄存器：	AH	0
	EDX	服务器所能分配的最高页面（4K）的物理地址

注释：

客户可在初始化时调用本功能取服务器所能分配的最大物理页地址。有些客户在进行内存管理时需要这一信息。

VCPI 功能 03H

取自由页面数（4K 大小）

调用寄存器：　AX　　DE03H
返回寄存器：　AH　　0
　　EDX　　自由页面数（4K 大小）

注释：

本功能返回服务器内存空间中未分配的页面数。本功能既可在 V86 模式下调用，也可在保护模式下调用。

VCPI 功能 04H
分配一页面（4K 大小）

调用寄存器：　AX　　DE04H
返回寄存器：　如果成功
　　AH＝0
　　EDX＝页面的物理地址
　　如果失败
　　AH＝非零
　　EDX＝未知

注释：

客户可使用本功能分配一页面。本功能既可在 V86 模式下调用，也可在保护模式下调用。

VCPI 功能 05H
释放一页面（4K 大小）

调用寄存器：　AX　　DE05H
　　EDX　　页面的物理地址
返回寄存器：　如果成功
　　AH＝0
　　如果失败
　　AH＝非零

注释：

本功能用于释放通过功能 04H 分配的内存。客户必须在退出前释放掉所分配的全部内存。本功能不能释放通过 EMS 接口分配的内存。本功能既可在 V86 模式下调用，也可在保护模式下调用。

VCPI 功能 06H
取页面的物理地址

调用寄存器： AX DE06H
CX 页号（由页线性地址右移 12 位得到）
返回寄存器： 如果成功
AH＝0
EDX＝页面的物理地址
如果页号无效
AH＝非零

注释：

本功能用于取通过 EMS 接口分配的页面的物理地址。在保护模式下，客户可将这些页面映射到自己的页表中。

VCPI 功能 07H
读 CR0 寄存器

调用寄存器： AX DE07H
返回寄存器： AH 0
EBX CR0 的值

注释：

本功能返回 CR0 系统寄存器的当前值。

VCPI 功能 08H
读调试寄存器

调用寄存器： AX DE08H
ES：DI 指向由 8 个双字构成的数组的指针；
DR0 为第一个双字，DR4 和 DR5 未用
返回寄存器： AH 0

注释：

客户使用本功能可得到调试寄存器的当前值。

VCPI 功能 09H
置调试寄存器

调用寄存器： AX DE09H
ES：DI 指向由 8 个双字构成的数组的指针；
DR0 为第一个双字，DR4 和 DR5 未用
返回寄存器： AH 0

注释：

客户使用本功能可将调试寄存器设置为指定的值。

VCPI 功能 0AH

取 8259A 中断向量映射

调用寄存器：　AX　DE0AH

返回寄存器：　AH　0

BX　IRQ0 的中断向量

CX　IRQ8 的中断向量

注释：

本功能返回当硬件中断发生时，由中断控制器产生的中断向量。

VCPI 功能 0BH

置 8259A 中断向量映射

调用寄存器：　AX　DE0BH

BX　IRQ0 的中断向量

CX　IRQ8 的中断向量

中断被关闭

返回寄存器：　AH　0

注释：

客户使用本功能来告知服务器中断控制器已被重新编程，使用了与以前不同的中断向量。在对中断控制器重新编程和调用本功能之前，客户必须关闭中断。

只有在系统仍使用 DOS 标准的硬件中断向量时（08H—0FH 用于 IRQ0—IRQ7，70H—77H 用于 IRQ8—IRQ15），才允许客户进行这一操作。

如果客户对中断控制器进行了重新编程，那么在退出前必须恢复以前的向量映射并调用本功能通知服务器。

VCPI 功能 0CH

从 V86 模式切换到保护模式

调用寄存器：　AX　DE0CH

中断被关闭

ESI　指向 1MB 地址以下的如表 10.2 结构的线性地址

表 10.2 切换到保护模式所用数据结构

偏移	大小	描述
00H	双字	客户 CR3 寄存器值
04H	双字	含有客户 GDTR 寄存器值的在 1MB 以下 6 个字节的线性地址
08H	双字	含有客户 IDTR 寄存器值的在 1MB 以下 6 个字节的线性地址
0CH	字	客户 LDTR 寄存器值
0EH	字	客户 TR 寄存器值
10H	PWORD	客户保护模式入口地址

返回寄存器： GDTR，IDTR，LDTR 和 TR 被装入
中断被关闭
控制被传送到指定的入口处
EAX，ESI，DS，ES，FS 和 GS 未知

注释：

服务器通过装入系统寄存器切换到客户的环境，并将控制转移到客户指定的入口。

控制被传送到指定的入口处

EAX 未知

注释：

服务器首先装入自己的系统寄存器，切换到自己的保护模式环境下。然后通过一条 IRETD 指令切换到 V86 模式，装入客户的寄存器，把控制转移到客户。客户必须自己设置 EFLAGS，以控制 V86 模式下的 IOPL。本功能只能在保护模式下调用。

• MEMO •

第十一章 任 务 切 换

任务切换是由 DOS 5.0 引入的功能。在 DOS 5.0 的命令解释器(Shell)中,用户可以暂停正在运行的程序并开始另一程序;在它结束之后,又可以继续运行被暂停的程序。在进行任务切换时,系统把被暂停的程序从内存中移出并将其映象存到磁盘,然后再调入新的程序。这时,系统中只有新调入的程序在独自运行。由此可见,这种任务切换并不同于真正的多任务系统。

在任务切换环境中,有两类程序:一是任务切换器,用以管理被切换的任务;另一类是用户程序,它是在任务切换器管理下运行的程序。Microsoft 公司已提供了有关任务切换的三类功能的文档。这三类功能分别是通知功能、服务功能和任务切换器功能。

通知功能由用户程序提供给任务切换器使用,以向用户程序提供有关任务切换和任务会话事件的信息。

服务功能由任务切换器提供给用户程序使用,以控制任务切换、获得关于任务切换器和其它用户程序的有关信息。

任务切换器功能由任务切换器提供给其它任务切换器使用。

11.1 数 据 结 构

在任务切换中使用了一些数据结构。许多功能都涉及到了指向一个或多个这些数据结构的指针。这一节将以 C 语言和汇编语言格式描述这些数据结构。

11.1.1 SWAPIINFO 结构

SWAPIINFO 结构描述了用户程序对特定类型的异步 API 提供的支持级别。此结构由在任务切换器下运行的网络管理器使用。

```
SWAPIINFO struc
        aisLength       dw      10
        aisAPI          dw      ?
        aisMajor        dw      ?
        aisMinor        dw      ?
        aisSupport      dw      ?
SWAPIINFO ends

struct SWAPIINFO
      {
        unsigned        int     aisLength;
        unsigned        int     aisAPI;
        unsigned        int     aisMajor;
        unsigned        int     aisMinor;
        unsigned        int     aisSupport;
      };
```

aisLength 是以字节计算的结构体的大小。它应该为 10。

aisAPI 用于标识由用户程序所支持的网络接口类型。其有效值及含义解释如下:

表 11.1 aisAPI 标识含义

有效值	含义
0001H	NETBIOS 接口
0002H	802.2 接口
0003H	TCP/IP 接口
0004H	LAN 管理器命名管道接口
0005H	NetWare IPX 接口
其他值被保留	

aisMajor 用于标识应用程序所支持的网络接口的最高主版本号。

aisMinor 用于标识应用程序所支持的网络接口的最高次版本号。

aisSupport 用于标识用户程序所支持的网络接口的级别。其有效值及含义如表 11.2：

表 11.2 aisSupport 标识含义

有效值	含义
0001H	最低程度支持。在一个应用程序调用了由 API 支持的功能后，即使请求已完成，用户程序仍然禁止任务切换
0002H	API 层次支持。用户程序跟踪异步请求。当请求没完成时，禁止任务切换；当所有的请求都已完成时，允许任务切换
0003H	切换器兼容级别。即使异步请求没完成，API 提供者也允许切换发生。由于资源有限，有些请求可能失败
0004H	完全兼容级别。API 提供者始终允许会话切换发生

11.1.2 SWCALLBACKINFO 结构

SWCALLBACKINFO 结构包含有关用户程序的信息。

```
SWCALLBACKINFO struc
    scbiNext            dd      ?
    scbiEntryPoint      dd      ?
    scbiReserved        dd      ?
    scbiAPI             dd      ?
SWCALLBACKINFO ends

struct SWCALLBACKINFO
   {
    struct SWCALLBACKINFO far  * scbiNext;
    void far                    * scbiEntryPoint;
    long                       scbiReserved;
    struct SWAPIINFO far        * scbiAPI;
  };
```

ScbiNext 是一指向通知链表中下一 SWCALLBACKINFO 结构的 32 位指针（段地址：偏移量）。

scbiEntryPoint 是一指向用户通知功能处理过程的 32 位指针（段地址：偏移量）。任务切换器使用此地址调用用户程序的通知功能。

scbiReserved 是保留区，别使用它。

scbiAPI 是一指向以零结束的 SWAPIINFO 表的 32 位指针（段地址：偏移量），它用来指明用户程序所支持的各种异步 API 的级别。

11.1.3　SWINSTANCEITEM 结构

SWINSTANCEITEM 结构包含关于一块事例数据的信息。块的内容不确定，可以包含任何需要的内容。

```
SWINSTANCEITEMstruc
    iisPtr      dd    ?
    iisSize     dw    ?
SWINSTANCEITEM ends

struct SWINSTANCEITEM
   {
    void far          *iisPtr;
    unsigned int      iisSize;
  };
```

iisPtr 是一指向事例数据块的 32 位指针（段地址：偏移量）。

iisSize 是以字节计算的事例数据块的大小。

11.1.4　SWSTARTUPINFO 结构

SWSTARTUPINFO 结构包含关于用户程序事例数据的信息。

```
SWSTARTUPINFO struc
    sisVersion          dw      ?
    sisNextDev          dd      ?
    sisVirtDevFile      dd      0
    sisReferenceData    dd      ?
    sisInstanceData     dd      ?
SWSTARTUPINFOenda

struct SWSTARTUPINFO
   {
    int                              sisVersion;
    struct SWSTARTUPINFO far         *sisNextDev;
    long                             sisVirtDevFile;
    long                             sisReferenceData;
    struct SWINSTANCEITEM far        *sisInstanceData;
  };
```

sisVersion 未使用。

sisNextDev 是指向在通知键表中下一个 SWSTARTUPINFO 结构的 32 位指针（段地址：偏移量）。

sisVirtDevFile 未使用。

sisReferenceData 未使用。

sisInstanceData 是一指向以零结束的 SWINSTANCEITEM 表的 32 指针（段地址：偏移量）。

11.1.5　SWVERSION 结构

SWVERSION 结构包含有关任务切换器的信息。

```
SWVERSION struc
    svsAPIMajor         dw      ?
    svsAPIMinor         dw      ?
    svsProductMajor     dw      ?
    svsProductMinor     dw      ?
    svsSwitcherID       dw      ?
    svsFlags            dw      ?
    svsName             dw      ?
    svsPrevSwitcher     dw      ?
SWVERSION ends
```

```
struct SWVERSION
   {
    unsigned    int     svsAPIMajor;
    unsigned    int     svsAPIMinor;
    unsigned    int     svsProductMajor;
    unsigned    int     svsProductMinor;
    unsigned    int     svsSwitcherID;
    unsigned    int     svsFlags;
    char        far      * svsName;
    void        far      * svsPrevSwitcher;
   };
```

svsAPIMajor 是由任务切换器所支持的任务切换协议的最高主版本号。

svsAPIMinor 是由任务切换器所支持的任务切换协议的最高次版本号。

svsProductMajor 是任务切换器的主版本号。

svsProductMinor 是任务切换器的次版本号。

svsSwitcherID 是任务切换器标识符。每个任务切换器都具有唯一的 4 比特标识符；此标识符在低 4 位中。

svsFlags 是任务切换器的操作标志。目前，只有位 0 已定义，其余位被保留并应为零。如果位 0 置位，任务切换器当前被关闭；如果位 0 是零，任务切换器是活动的。

svsName 是一指向表示任务切换器名字的以零结束的 ASCII 串的 32 位指针（段地址：偏移量）。

svsPrevSwitcher 是一指向上次装入的任务切换器的 32 位指针（段地址：偏移量）。这个地址能用来调用上次装入的任务切换器的服务功能。

11.2 通 知 功 能

这一节将以类似于本书别处描述中断功能的方式来描述通知功能。

用户程序通过 INT 2FH 功能 4BH 子功能 01H（建立通知链表），来将其通知功能处理过程地址，提供给任务切换器和调用、控制任务切换器的程序。任务切换器，或者调用、控制任务切换器的程序，通过把功能号赋给 AX 并同时使用其它寄存器，来调用通知功能。

通知功能 0000H　　V5

初始化

通知用户程序，一个新的任务切换器在初始化中

调用寄存器：　AX　　0000H

ES：DI 任务切换器服务功能入口地址

返回寄存器： AX 状态

0000H，任务切换器能被安全装入

非零，任务切换器不能被安全装入

注释：

在任务切换器、其控制程序或这两者同时初始化时，必须调用本功能。如果任务切换器提供了服务功能的话，必须支持服务功能 0000H（取版本）。服务功能入口地址可能是 NULL 指针，特别是当任务切换器的控制程序调用通知功能时。

如果某一用户程序作了否定回答（即返回时置 AX 为非零值），任务切换器应当关闭自己。这时，任务切换器可能要调用别的用户程序的 0007H 号通知功能。如果在此之前没有调用初始化切换器，用户程序忽略此调用。

调用时，任务切换器已打开中断。用户程序在进行通知服务时，可以使用任何 MS-DOS 系统功能。返回时，用户程序必须设置 AX 寄存器并保护其他寄存器。

通知功能 0001H **V5**

查询暂停

通知用户程序，任务切换器准备进行任务切换

调用寄存器： AX 0001H

BX 当前任务标识符

ES：DI 任务切换器服务功能入口地址

返回寄存器： AX 状态

0000 允许切换

0001 禁止切换

其他值被保留

注释：

当切换发生时，在全局内存中的用户程序通过任务识别符能得知哪一个任务将要被暂停。

用户程序通过调用服务功能 0001H（测试内存区）可以得知，内存中的特定代码或数据是否将受任务切换影响，从此决定是否允许进行切换。

在当异步 API 的状态不允许从而禁止任务切换之前，用户程序应调用服务功能 0006H（查询 API 支持）以保证另一个用户程序没在使用 API。

如何某一用户程序返回非零，禁止任务切换，任务切换器可能对所有用户程序作一通知功能 0004H（任务已激活）调用。在没有调用查询暂停或暂停任务之前，用户程序应忽略此调用。

调用时，任务切换器已打开中断。用户程序在进行通知服务时，可以使用任何 MS-DOS 系统功能。返回时，用户程序必须设置 AX 寄存器并保护其他寄存器。

通知功能 0002H V5

暂停任务

通知用户程序一个会话切换将要发生

调用寄存器： AX 0002H

BX 会话标识符

ES：DI 任务切换器服务功能入口地址

返回寄存器： AX 状态

0000H 允许切换

0001H 不允许切换

所有其他值保留

注释：

如果所有的用户程序对通知功能 0001H（查询暂停）都返回 0000H，任务切换器就关闭中断，并调用暂停任务功能。这是用户程序禁止任务切换的最后的机会。用户程序绝对不能发出软中断或进行可能打开中断的调用。

如果所有的用户程序都返回 0000H，在打开中断之前，任务切换器用保存的拷贝取代当前中断向量表，以确保在调用暂停任务和激活任务之间，没有局部的中断处理过程被调用。

全局内存中的用户程序可能会收到中断，但它一定不能使用非全局内存。

在当异步 API 的状态不允许从而禁止任务切换之前，用户程序应调用服务功能 0006H（查询 API 支持）以保证另一个用户程序没在使用 API。

如何某一用户程序返回非零，禁止任务切换，任务切换器可能对所有用户程序作一通知功能 0004H（任务已激活）调用。在没有调用查询暂停或暂停任务之前，用户程序应忽略此调用。

调用时，任务切换器已关闭中断。用户程序在进行通知服务时，不能使用任何 MS-DOS 系统功能。返回时，用户程序必须设置 AX 寄存器并保护其它寄存器。

通知功能 0003H V5

准备激活任务

通知用户程序，一个任务将被激活

调用寄存器： AX 0003H

BX 任务标识符

CX 任务状态标志

位 0

1 此任务是首次被激活；

0 此任务以前曾被暂停过，现在被重新激活

其他位被保留且必须为零

ES：DI 任务切换器服务功能入口地址

返回寄存器： AX 0000H

注释：

如果是一个暂停过的任务现在被重新激活，那么局部内存和向量表已经被重新装入。

如果会话是被首次激活，此后还会进行建立任务调用。

调用时，任务切换器已关闭中断。用户程序在进行通知服务时，不能使用任何 MS-DOS 系统功能。返回时，用户程序必须清除 AX 寄存器并保护其他寄存器。

通知功能 0004H V5

任务已激活

通知用户程序，一个任务已被激活

调用寄存器： AX 0004H

BX 任务标识符

CX 任务状态标志

位 0

1 此任务是首次被激活；

0 此任务以前曾被暂停过，现在被重新激活

其他位被保留且必须为零

ES：D 任务切换器服务功能入口地址

返回寄存器： AX 0000H

注释：

如果某个用户程序在服务查询暂停或暂停任务功能时，返回了失败，那么，任务切换器可能会对所有的用户程序调用本功能，而不管这些程序是否已接受过查询暂停或暂停任务调用。用户程序应忽略这次调用。

调用时，任务切换器已打开中断，用户程序可使用任何 MS-DOS 系统功能。返回时，用户程序必须清除 AX 寄存器并保护其他寄存器。

通知功能 0005H V5

建立任务

通知用户程序，任务切换器将要建立一个新的任务

调用寄存器： AX 0005H

BX 新任务的标识符

ES：DI 任务切换器服务功能入口地址

返回寄存器： AX 状态

0000H 新任务能安全建立

0001H 新任务不能安全建立

其他值被保留

注释：

本功能允许用户程序禁止建立新任务。

如果某个用户程序返回了 0001H，任务切换器可能对所有用户程序进行废除任务调用。如果用户程序收到了废除任务调用而事先没有收到建立任务调用，则应忽略此废除任务调用。

调用时，任务切换器已打开中断，用户程序可使用任何 MS-DOS 系统功能。返回时，用户程序必须设置 AX 寄存器并保护其他寄存器。

通知功能 0006H　　废除任务　　V5

通知用户程序，任务切换器将要废除一个任务

调用寄存器：　AX　　0006H
　　BX　　将被废除的任务的标识符
　　ES：DI　　任务切换器服务功能入口地址

返回寄存器：　AX　　0000H

注释：

当前任务终止时，任务切换器调用本功能。不过，在会话管理器的命令下，也可能对正在运行的当前任务或一个暂停任务调用本功能。

如果某个用户程序在服务建立任务功能时，返回了失败，那么，任务切换器可能会对所有的用户程序调用本功能，而不管这些程序是否已接受过建立任务调用。用户程序应忽略这次调用。

调用时，任务切换器已打开中断，用户程序可使用任何 MS-DOS 系统功能。返回时，用户程序必须清除 AX 寄存器并保护其他寄存器。

通知功能 0007H　　切换器退出　　V5

通知用户程序，任务切换器不再活动

调用寄存器：　AX　　0007H
　　BX　　标志
　　位 0
　　1　此任务切换器是已存在的唯一一个
　　0　至少还有一个别的任务切换器在活动
　　其他位被保留且必须为零
　　ES：DI　　任务切换器服务功能入口地址

返回寄存器：　AX　　0000H

注释：

全局用户程序应停止它们正在进行的必须同任务切换器共存的处理过程。

本功能可由任务切换器来调用，但更有可能由控制任务切换器的程序来调用。所以，服务功能入口地址可能不同于调用其他通知功能时所传递的地址，而且有可能是空（NULL）指针。

调用时，任务切换器已打开中断，用户程序可使用任何 MS-DOS 系统功能。返回时，用户程序必须清除 AX 寄存器并保护其他寄存器。

11.3　服 务 功 能

本节以类似于书中别处描述中断功能的方式将描述服务功能。

当任务切换器使用通知功能或任务切换器功能时，它同时向用户程序提供了其服务功能入口地址。因为通知调用可能由不同的任务切换器或控制任务切换器的程序发出，所以用户程序不能认为服务功能入口地址是相同的。就用户程序来说，只有在处理任务切换器对它的调用时，所给服务功能入口地址是有效的。在调用服务功能前，用户程序要在 AX 寄存器中放入功能号并设置好其他寄存器。

服务功能 0000H　　V5

取版本

返回当前任务切换器的 SWVERSION 结构

调用寄存器：　AX　　0000H

返回寄存器：　如果成功，进位标志清零

AX　　000H

ES：BX指向 SWVERSION 结构的指针

如果任务切换器不支持本功能，进位标志置位

注释：

任务切换器可能打开中断，并调用任何 MS-DOS 系统功能。本功能必须保护除 AX、ES 和 BX 寄存器以外的其他寄存器。

服务功能 0001H　　V5

测试内存区

确定给定内存对当前任务是全局的还是局部的。当任务切换发生时，局部内存块被交换到磁盘上

调用寄存器：　AX　　0001H

CX　　以字节计算的缓冲区大小

ES：DI　缓冲区地址

返回寄存器：　如果成功，进位标志清零

AX　　内存状态

0000H 缓冲区在全局内存中
0001H 缓冲区既在全局内存又在局部内存中
0002H 缓冲区在局部内存中
其他值被保留

如果不支持此功能，进位标志置位

注释：

当用户程序收到查询暂停调用或任务已激活调用时，可以使用本功能来确定其内存是全局的还是局部的。象网络管理器之类的为其他程序提供服务的用户程序，应把那些程序标识为全局的或局部的。

在服务此功能时，任务切换器一定不能打开中断或调用任何 MS-DOS 系统功能。任务切换器应设置 AX 寄存器并保护其他寄存器。

服务功能 0002H V5

暂停切换器

通知任务切换器暂停操作

调用寄存器： AX 0002H
ES：DI 新任务切换器的服务功能入口地址

返回寄存器： 如果成功，进位标志清零
AX 状态
0000H 当前任务切换器暂停了操作
0001H 当前任务没有暂停操作；
新任务切换器还能开始
0002H 当前任务没有暂停操作；
新任务切换器可以开始
其他值被保留

如果不支持本功能，进位标志置位

注释：

本功能只能由另一任务切换器调用。

当任务切换器收到这个调用时，它应该继续进行服务，但在收到恢复任务切换器调用之前，它不能响应键盘中断或进行任务切换。

本功能可以被嵌套调用。在收到相同次数的恢复任务切换器调用之前，应停正常操作。有一种情况是例外。如果运行在另一任务切换器的子程序悬挂了它的任务管理器的任务切换器，在把控制返回到任务管理器之前，没能成功地恢复任务切换器，这时，任务管理器可以安全地重新激活其任务切换器。

在服务本功能时，任务切换器可能会打开中断，并可能调用 MS-DOS 系统功能。

新的任务切换器可通过调用检测切换器功能（INT 2FH 功能 4B02H）来获得前一个任务切换器的服务功能入口地址。

服务功能 0003H V5

恢复任务切换器

通知一个被暂停的任务切换器重新开始工作

调用寄存器： AX 0003H

ES：DI 调用者的服务功能入口地址

返回寄存器： 如果成功，进位标志清零

AX 0000H

如果不支持本功能，进位标志置位

注释：

如果一个任务切换器通过暂停切换器调用暂停了另一个任务切换器，应使用本功能来恢复任务切换器。调用本功能时所给服务功能入口地址，应同调用暂停切换器功能时所给服务功能入口地址相同。

在服务本功能时，任务切换器可能会打开中断，并可能调用 MS-DOS 系统功能。

服务功能 0004H V5

连接通知链表

告知任务切换器，把指定 SWCALLBACKINFO 结构加到切换器的通知链表中

调用寄存器： AX 0004H

ES：DI 指向 SWCALLBACKINFO 结构的指针

返回寄存器： 如果成功，进位标志清零

AX 0000H

如果不支持本功能，进位标志置位

注释：

用户程序可以调用检测切换器功能（INT2 FH 功能 4B02H）来检测任务切换器并得到其服务功能入口地址。如果切换器存在，用户程序就能调用本功能。用户程序必须在调用之前就设置好了 SWCALLBACKINFO 结构。

某些任务切换器可能会在每次进行任务切换前，特意调用建立通知链表功能（INT 2FH 功能 4B01H）。在这种情形下，任务切换将简单地从本功能返回，不做任何事情。

如果任务切换器不能自己建立通知链表，就必须靠用户程序通过本功能来建立链表。

用户程序终止之前，必须通过调用脱离通知链表功能（服务功能 0005H）使自己脱离此通知链表。

在服务本功能时，任务切换器可能会打开中断，并可能调用 MS-DOS 系统功能。

服务功能 0005H V5

脱离通知链表

告知任务切换器，从其通知链表中取下用户程序的 SWCALLBACKINFO 结构

调用寄存器： AX 0005H

ES：DI 指向 SWCALLBACKINFO 结构的指针

返回寄存器： 如果成功，进位标志清零

AX 0000H

如果不支持本功能，进位标志置位

注释：

用户程序在终止前必须调用本功能，用来从任务切换器的通知链表中取下用户程序的 SWCALLBACKINFO 结构。

如果任务切换器是通过在每次任务切换前，调用建立通知链表功能来重建其通知链表，它可能直接从本功能返回，不做任何事情。

在服务本功能时，任务切换器可能会打开中断，并可能调用 MS-DOS 系统功能。

服务功能 0006H **V5**

询问 API 支持

返回某用户程序 SWAPIINFO 结构的地址，此用户程序为指定的异步 API 提供了最高级别的支持

调用寄存器： AX 0006H

BX API 标识符

0001H NETBIOS 接口

0002H 802.2 接口

0003H TCP/IP 接口

0004H LAN 管理器命名管道接口

0005H NetWare IPX 接口

其他值被保留

返回寄存器： 如果成功，进位标志清零

AX 0000H

ES：BX指向 SWAPIINFO 结构的指针

如果失败，进位标志置位

注释：

在决定是否进行任务切换之前，提供 API 支持的用户程序应调用此功能，以发现是否另一个用户程序为同一个 API 提供了更高层次的支持。如果返回了指向它自己的 SWAPIINFO 结构的指针，它必须决定是否进行任务切换。如果指针指向另一个用户程序 SWAPIINFO 的结构，则由它判定是否进行任务切换。

如果调用时中断已被关闭，任务切换器一定不能打开中断或进行任何 MS-DOS 系统功能调用。

• MEMO •

第十二章 标准 TSR 识别技术

同时使用多个 TSR 程序时，一个常见的问题是缺少管理程序的标准。

早在 1986 年，一些 TSR 开发者就组织了一个组织以建立这样的标准。许多著名的 TSR 开发者参加了这个组织。他们后来建立了一个称作 TesSeRact 的标准。

TesSeRact 标准确定了一组链入 DOS 多路中断 (INT 2FH) 的功能。因为 DOS 使用中断 2FH 同它自己的 TSR (诸如 ASSIGN，PRINT 和 SHARE) 通信，所以 TesSeRact 开发者认为用相同的接口为各自开发的 TSR 程序服务是可行的。这些功能可通过先在 AX 寄存器放一特殊码 (5453H 或 ASCII 字符 "TS")，然后发出 INT 2FH 调用来访问。

这与 DOS 文档中 INT 2FH 的标准用法不一致。DOS 规定，识别码应在 AH 寄存器中，(限制在 80H—FFH 范围内)，并用 AL=0 作为 "安装与否" 功能。

这个组织在 DOS 的规定发表之前就已开发了 TesSeRact 标准。后来，由于此标准已被广泛采用，以至于在某些基本方面不能再作改变。当前，还没有一个已知的功能使用了代码 54H，因此不会导致任何问题 (即使功能号相同，但同时还使用 53H 子功能码的机率很小)。

值得注意的是，DOS V3.3 在引入 APPEND 时就没有遵循它自己的规定。APPEND 使用了功能 B7H。而按规定，这个功能代码是供非 DOS 应用程序使用的。

为减少 TSR 之间的冲突，这个组织鼓励所有的开发者在自己的代码中支持 TesSeRact 标准。为此只需提供一个支持用户参数块和两个子功能的 INT 2FH 处理程序，这两个功能在章中给予了描述。

12.1 用户参数块

TesSeRact 标准之一是用户参数块。它描述如下 (必须放在 TSR 的 INT 2FH 处理过程起始处)：

```
New_2F:     jmp     OverParams      ; INT 2F 向量指向此处
UserParams  db      8 dup (")       ; 8 字节程序识别串
IdNum       dw      0               ; TSR 识别号
FuncFlag    dd      0ffffffffh      ; 支持功能位映象
HotKey      db      0               ; 使用的热键扫描码
ShiftSt     db      0               ; 弹出时的 Shift 状态
HotFlag     db      0               ; 热键标识
ExtCnt      db      0               ; 额外的热键数
ExtHot      dd      0               ; 指向额外热键的指针
Status      dw      0               ; TSR 状态标志
OurPSP      dw      0               ; 自己的 PSP 段地址
OurDTA      dd      0               ; 自己的 DTA 地址
DSeg        dw      0               ; 缺省的数据段段地址
```

FuncFlag 是 4 字节的位映象变量，它指出了这个 TSR 支持的所有多路功能。调用这些功能时，功能号必须通过 BX 寄存器传递给处理过程，而不是在 AL 中。此变量的映象如下：

第 0 位　　功能 00H (安装检查 必须提供)

第 1 位	功能 01H　(返回用户参数块　必须提供)
第 2 位	功能 02H　(检查热键)
第 3 位	功能 03H　(替换 INT 24H)
第 4 位	功能 04H　(返回数据指针)
第 5 位	功能 05H　(设置额外热键)
第 6－7 位	未定义　保留给将来使用
第 8 位	功能 10H　(打开 TSR)
第 9 位	功能 11H　(关闭 TSR)
第 10 位	功能 12H　(从 RAM 中释放 TSR)
第 11 位	功能 13H　(重启动 TSR)
第 12 位	功能 14H　(获取当前状态)
第 13 位	功能 15H　(设置 TSR 状态)
第 14 位	功能 16H　(获取弹出类型)
第 15 位	未定义　保留给将来使用
第 16 位	功能 20H　(调用用户过程)
第 17 位	功能 21H　(填充键盘)
第 18－31 位	未定义　保留给将来使用

如果 TSR 支持某个功能，此功能相应的位应被设置为 1，否则应被设置为 0。

12.2　功能 00H　安装检查

该功能用于确定 TSR 程序是否已装入。它使用下面的调用方式：

```
mov     ax,  5453h          ;  'TS'
mov     si,  offset IDStr   ;  参见下面的说明
mov     ds,  seg IDstr
xor     cx,  cx             ;  处理程序计数器
xor     bx,  bx             ;  功能 0000H
int     2Fh
```

IDStr 长八个字节，含有用于标识 TSR 的字符串。用户的 INT 2FH 处理程序把以参数形式传送的字符串（见下例）与它自身的标识串相比较。如果匹配，TSR 返回时，将自身的句柄置入 CX，并设置 AX 为 0FFFFH。

如果不匹配，则恢复所有寄存器，将 CX 加 1，然后将中断顺着链往下传递。如果系统中每一个与 TesSeRact 标准兼容的 TSR 在链到下一个之前将 CX 加 1，那么，当中断过程最后返回到调用程序时，CX 寄存器的值要么为正被搜索的 TSR 的句柄，要么是下一个可用句柄(没发现与 IdStr 匹配的 TSR 时)。

如果不匹配，AL 的返回值不会等于 FFH，TSR 安装程序就能利用 CX 中的句柄作为它自身的标识，并将句柄保存在参数块的 IdNum 中。

下面的代码，是在参考 TesSeRact 库中的 INT 2FH 处理过程的基础上编辑而成的。它表明了怎样才能保证程序的一致性。

```
overparms:
    cmp   ax, 5453h          ; ax=5453HTesSeRact 功能
    jne   not_our_2F         ; 其他多路中断
    push  ds                 ; 保存 DS
    push  cs
    pop   ds                 ; 在此设置 DS
    push  ax
    push  bx
    or    bx, bx             ; 首先进行安装检查
    jnz   test_for_1         ; 进行其他功能检查
```

```
;
; 以下是安装检查 (CHECK INSTALL) 的代码
; DS:SI 指向 IDStr
; CX 是链的当前号码
;
    push cx
    push si                 ; 保存 SI
    lea  di, UserParms      ; 程序自身的 IDStr
    push cs
    pop  es
    mov  cx  8              ; 检查是否匹配
    rep  cmpsb
    pop  si
    pop  cx
    jnz  next_one           ; 不匹配，不归我们处理
    pop  bx                 ; 匹配，清堆栈
    pop  ax
    mov  cx, es:[di]        ; 在 CX 中返回 IdNum
    xor  ax, ax
    dec  ax                 ; AX=-1 表示已驻留
    jmp  shortdone_2F
next_one:                   ; 检查下一 ID 号
    inc  cx
    pop  bx                 ; 恢复寄存器
    pop  ax
    pop  ds
not_our_2F:                 ; 通过保存的指针
    jmp  dword ptr [oldint2F]; 链到下一 INT 2FH 处理程序
done_2F:                    ; 返回调用者
    pop  ds
    iret
```

12.3 功能 01H—返回用户参数指针

作为对 TesSeRact 标准的最小支持还必需的另一个功能就是功能 01H。此功能返回一个指向用户参数块区的远指针。调用方法如下：

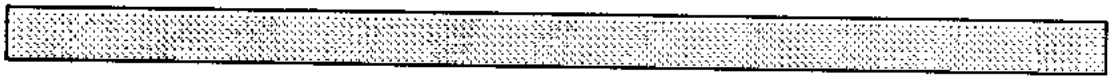
```
    mov  ax, 5453H          ; 'TS'
    mov  bx, 01H            ; 功能号
    mov  cx, TsrIdNum       ; 标识号
    int  2Fh
```

如果 CX 中的标识号与 TSR 的标识号相等(TSR 标识号由功能 00H 返回)，则返回时，ES:BX 指向用户参数区并且 AX 为零。下面的代码片断表示了一种实现方法：

```
test_for_1:                 ; 如果不是功能 00H
    cmp  cx, IdNum          ; 与自己的进行比较
    jne  not_our_2F
    push cs                 ; 对了，设置返回值
    pop  es
    lea  bx, UserParms
    xor  ax, ax             ; 调用成功状态码
    pop  ds
    jmp  done_2F
```

· MEMO ·

附录　DOS 常用资料速查



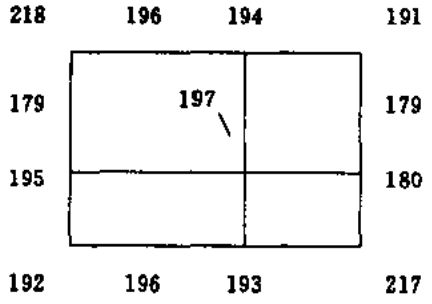
香港金山公司

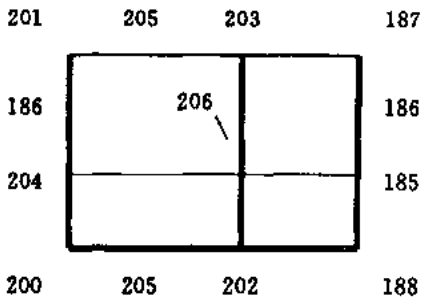
KINGSUN

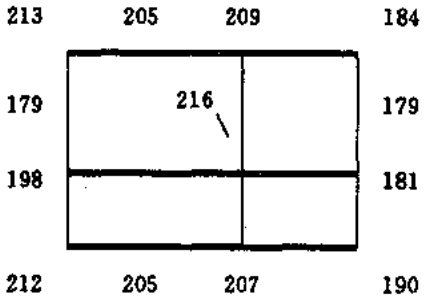
大家风范　承诺永恒

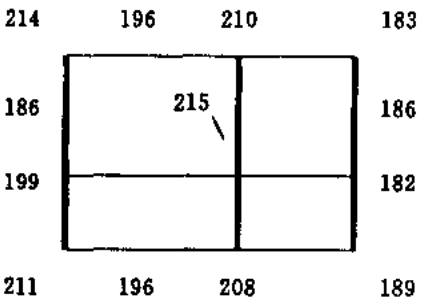
• MEMO •

速查A 英文制表ASCII码表



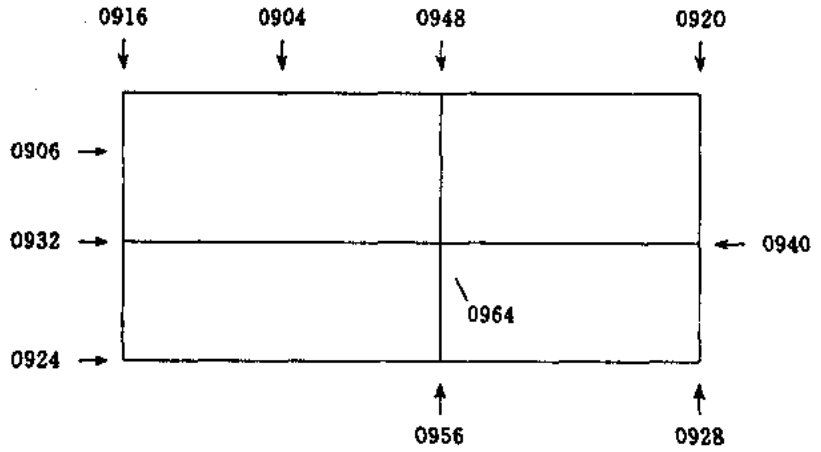




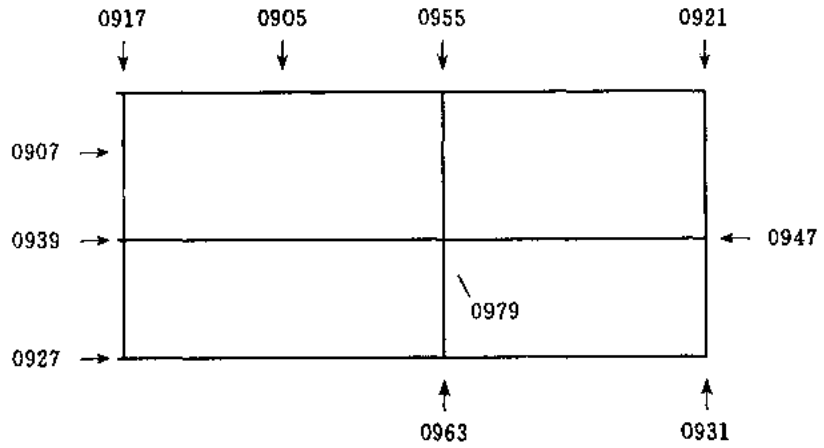


速查B 中文线框区位码表

(1)细线



(2) 粗线



速查 C 键盘扫描码表

击　　键	83/84—键盘 标准功能调用	101/102—键盘 标准功能调用	101/102—键盘 扩展功能调用
Esc	01/1B	01/1B	01/1B
1	02/31	02/31	02/31
2	03/32	03/32	03/32
3	04/33	04/33	04/33
4	05/34	05/34	05/34
5	06/35	06/35	06/35
6	07/36	07/36	07/36
7	08/37	08/37	08/37
8	09/38	09/38	09/38
9	0A/39	0A/39	0A/39
0	0B/30	0B/30	0B/30
—	0C/2D	0C/2D	0C/2D
=	0D/3D	0D/3D	0D/3D
Backspace	0E/08	0E/08	0E/08
Tab	0F/09	0F/09	0F/09
q	10/71	10/71	10/71
w	11/77	11/77	11/77
e	12/65	12/65	12/65
r	13/72	13/72	13/72
t	14/74	14/74	14/74
y	15/79	15/79	15/79
u	16/75	16/75	16/75
i	17/69	17/69	17/69
o	18/6F	18/6F	18/6F
p	19/70	19/70	19/70

续表

击　　键	83/84－键盘 标准功能调用	101/102－键盘 标准功能调用	101/102－键盘 扩展功能调用
[	1A/5B	1A/5B	1A/5B
]	1B/5D	1B/5D	1B/5D
Enter	1C/0D	1C/0D	1C/0D
Ctrl	* *	* *	* *
a	1E/61	1E/61	1E/61
s	1F/73	1F/73	1F/73
d	20/64	20/64	20/64
f	21/66	21/66	21/66
g	22/67	22/67	22/67
h	23/68	23/68	23/68
j	24/6A	24/6A	24/6A
k	25/6B	25/6B	25/6B
l	26/6C	26/6C	26/6C
;	27/3B	27/3B	27/3B
'	28/27	28/27	28/27
`	29/60	29/60	29/60
Shift	* *	* *	* *
\	2B/5C	2B/5C	2B/5C
z	2C/7A	2C/7A	2C/7A
x	2D/78	2D/78	2D/78
c	2E/63	2E/63	2E/63
v	2F/76	2F/76	2F/76
b	30/62	30/62	30/62
n	31/6E	31/6E	31/6E
m	32/6D	32/6D	32/6D
,	33/2C	33/2C	33/2C
.	34/2E	34/2E	34/2E
/	35/2F	35/2F	35/2F
Gray *	37/2A	37/2A	37/2A
Alt	* *	* *	* *
Space	39/20	39/20	39/20
Caps Lock	* *	* *	* *
F1	3B/00	3B/00	3B/00
F2	3C/00	3C/00	3C/00
F3	3D/00	3D/00	3D/00
F4	3E/00	3E/00	3E/00
F5	3F/00	3F/00	3F/00
F6	40/00	40/00	40/00
F7	41/00	41/00	41/00
F8	42/00	42/00	42/00
F9	43/00	43/00	43/00
F10	44/00	44/00	44/00

续表

击　　键	83/84－键盘 标准功能调用	101/102－键盘 标准功能调用	101/102－键盘 扩展功能调用
F11	－－	－－	85/00
F12	－－	－－	86/00
Num Lock	* *	* *	* *
Scroll Lock	* *	* *	* *
White Home	47/00	47/00	47/00
White Up Arrow	48/00	48/00	48/00
White PgUp	49/00	49/00	49/00
Gray －	4A/2D	4A/2D	4A/2D
White Left Arrow	4B/00	4B/00	4B/00
Center Key	－－	－－	4C/00
White Right Arrow	4D/00	4D/00	4D/00
Gray ＋	4E/2B	4E/2B	4E/2B
White End	4F/00	4F/00	4F/00
White Down Arrow	50/00	50/00	50/00
White PgDn	51/00	51/00	51/00
White Ins	52/00	52/00	52/00
White Del	53/00	53/00	53/00
SysReq	－－	* *	* *
Key 45 [5]	－－	56/5C	same
Enter（数字键盘）	－－	1C/0D	E0/0D
Gray /	－－	35/2F	E0/2F
PrtSc	－－	* *	* *
Pause	－－	* *	* *
Gray Home	－－	47/00	47/E0
Gray Up Arrow	－－	48/00	48/E0
Gray Page Up	－－	49/00	49/E0
Gray Left Arrow	－－	4B/00	4B/E0
Gray Right Arrow	－－	4D/00	4D/E0
Gray End	－－	4F/00	4F/E0
Gray Down Arrow	－－	50/00	50/E0
Gray Page Down	－－	51/00	51/E0
Gray Insert	－－	52/00	52/E0
Gray Delete	－－	53/00	53/E0
Shift Esc	01/1B	01/1B	01/1B
!	02/21	02/21	02/21
@	03/40	03/40	03/40
#	04/23	04/23	04/23
$	05/24	05/24	05/24
%	06/25	06/25	06/25
^	07/5E	07/5E	07/5E
&	08/26	08/26	08/26
* (white)	09/2A	09/2A	09/2A

续表

击　　键	83/84－键盘 标准功能调用	101/102－键盘 标准功能调用	101/102－键盘 扩展功能调用
(	0A/28	0A/28	0A/28
)	0B/29	0B/29	0B/29
—	0C/5F	0C/5F	0C/5F
+ (white)	0D/2B	0D/2B	0D/2B
Shift Backspace	0E/08	0E/08	0E/08
Shift Tab (Backtab)	0F/00	0F/00	0F/00
Q	10/51	10/51	10/51
W	11/57	11/57	11/57
E	12/45	12/45	12/45
R	13/52	13/52	13/52
T	14/54	14/54	14/54
Y	15/59	15/59	15/59
U	16/55	16/55	16/55
I	17/49	17/49	17/49
O	18/4F	18/4F	18/4F
P	19/50	19/50	19/50
{	1A/7B	1A/7B	1A/7B
}	1B/7D	1B/7D	1B/7D
Shift Enter	1C/0D	1C/0D	1C/0D
Shift Ctrl	* *	* *	* *
A	1E/41	1E/41	1E/41
S	1F/53	1F/53	1F/53
D	20/44	20/44	20/44
F	21/46	21/46	21/46
G	22/47	22/47	22/47
H	23/48	23/48	23/48
J	24/4A	24/4A	24/4A
K	25/4B	25/4B	25/4B
L	26/4C	26/4C	26/4C
:	27/3A	27/3A	27/3A
"	28/22	28/22	28/22
~	29/7E	29/7E	29/7E
|	2B/7C	2B/7C	2B/7C
Z	2C/5A	2C/5A	2C/5A
X	2D/58	2D/58	2D/58
C	2E/43	2E/43	2E/43
V	2F/56	2F/56	2F/56
B	30/42	30/42	30/42
N	31/4E	31/4E	31/4E
M	32/4D	32/4D	32/4D
<	33/3C	33/3C	33/3C
>	34/3E	34/3E	34/3E

续表

击　　键	83/84－键盘 标准功能调用	101/102－键盘 标准功能调用	101/102－键盘 扩展功能调用
?	35/3F	35/3F	35/3F
Shift Gray *	* *	* *	37/2A
Shift Alt	* *	* *	* *
Shift Space	39/20	39/20	39/20
Shift Caps Lock	* *	* *	* *
Shift F1	54/00	54/00	54/00
Shift F2	55/00	55/00	55/00
Shift F3	56/00	56/00	56/00
Shift F4	57/00	57/00	57/00
Shift F5	58/00	58/00	58/00
Shift F6	59/00	59/00	59/00
Shift F7	5A/00	5A/00	5A/00
Shift F8	5B/00	5B/00	5B/00
Shift F9	5C/00	5C/00	5C/00
Shift F10	5D/00	5D/00	5D/00
Shift F11	－－	－－	87/00
Shift F12	－－	－－	88/00
Shift Num Lock	* *	* *	* *
Shift Scroll Lock	* *	* *	* *
Shift 7（数字键盘）	47/37	47/37	47/37
Shift 8（数字键盘）	48/38	48/38	48/38
Shift 9（数字键盘）	49/39	49/39	49/39
Shift Gray －	4A/2D	4A/2D	4A/2D
Shift 4（数字键盘）	4B/34	4B/34	4B/34
Shift 5（数字键盘）	4C/35	4C/35	4C/35
Shift 6（数字键盘）	4D/36	4D/36	4D/36
Shift Gray ＋	4E/2B	4E/2B	4E/2B
Shift 1（数字键盘）	4F/31	4F/31	4F/31
Shift 2（数字键盘）	50/32	50/32	50/32
Shift 3（数字键盘）	51/33	51/33	51/33
Shift 0（数字键盘）	52/30	52/30	52/30
Shift .（数字键盘）	53/2E	53/2E	53/2E
Shift SysReq	－－	* *	* *
Shift Key 45 [5]	－－	56/7C	56/7C
Shift Enter－number pad	－－	1C/0D	E0/0 D
Shift Gray /	－－	35/2F	E0/2F
Shift PrtSc	－－	* *	* *
Shift Pause	－－	* *	* *
Shift Gray Home	－－	47/00	47/E0
Shift Gray Up Arrow	－－	48/00	48/E0
Shift Gray Page Up	－－	49/00	49/E0
Shift Gray Left Arrow	－－	4B/00	4B/E0

续表

击　　键	83/84－键盘 标准功能调用	101/102－键盘 标准功能调用	101/102－键盘 扩展功能调用
Shift Gray Right Arrow	－－	4D/00	4D/E0
Shift Gray End	－－	4F/00	4F/E0
Shift Gray Down Arrow	－－	50/00	50/E0
Shift Gray Page Down	－－	51/00	51/E0
Shift Gray Insert	－－	52/00	52/E0
Shift Gray Delete	－－	53/00	53/E0
Ctrl Esc	01/1B	01/1B	01/1B
Ctrl 1	－－	－－	－－
Ctrl 2 (NUL)	03/00	03/00	03/00
Ctrl 3	－－	－－	－－
Ctrl 4	－－	－－	－－
Ctrl 5	－－	－－	－－
Ctrl 6 (RS)	07/1E	07/1E	07/1E
Ctrl 7	－－	－－	－－
Ctrl 8	－－	－－	－－
Ctrl 9	－－	－－	－－
Ctrl 0	－－	－－	－－
Ctrl －	0C/1F	0C/1F	0C/1F
Ctrl ＝	－－	－－	－－
Ctrl Backspace (DEL)	0E/7F	0E/7F	0E/7F
Ctrl Tab	－－	－－	94/00
Ctrl q (DC1)	10/11	10/11	10/11
Ctrl w (ETB)	11/17	11/17	11/17
Ctrl e (ENQ)	12/05	12/05	12/05
Ctrl r (DC2)	13/12	13/12	13/12
Ctrl t (DC4)	14/14	14/14	14/14
Ctrl y (EM)	15/19	15/19	15/19
Ctrl u (NAK)	16/15	16/15	16/15
Ctrl i (HT)	17/09	17/09	17/09
Ctrl o (SI)	18/0F	18/0F	18/0F
Ctrl p (DEL)	19/10	19/10	19/10
Ctrl [(ESC)	1A/1B	1A/1B	1A/1B
Ctrl] (GS)	1B/1D	1B/1D	1B/1D
Ctrl Enter (LF)	1C/0A	1C/0A	1C/0A
Ctrl a (SOH)	1E/01	1E/01	1E/01
Ctrl s (DC3)	1F/13	1F/13	1F/13
Ctrl d (EOT)	20/04	20/04	20/04
Ctrl f (ACK)	21/06	21/06	21/06
Ctrl g (BEL)	22/07	22/07	22/07
Ctrl h (Backspace)	23/08	23/08	23/08
Ctrl j (LF)	24/0A	24/0A	24/0A
Ctrl k (VT)	25/0B	25/0B	25/0B

续表

击　　键	83/84－键盘 标准功能调用	101/102－键盘 标准功能调用	101/102－键盘 扩展功能调用
Ctrl l (FF)	26/0C	26/0C	26/0C
Ctrl ,	－－	－－	－－
Ctrl '	－－	－－	－－
Ctrl '	－－	－－	－－
Ctrl Shift	* *	* *	* *
Ctrl　(FS)	2B/1C	2B/1C	2B/1C
Ctrl z (SUB)	2C/1A	2C/1A	2C/1A
Ctrl x (CAN)	2D/18	2D/18	2D/18
Ctrl c (ETX)	2E/03	2E/03	2E/03
Ctrl v (SYN)	2F/16	2F/16	2F/16
Ctrl b (STX)	30/02	30/02	30/02
Ctrl n (SO)	31/0E	31/0E	31/0E
Ctrl m (CR)	32/0D	32/0D	32/0D
Ctrl ,	－－	－－	－－
Ctrl .	－－	－－	－－
Ctrl /	－－	－－	－－
Ctrl Gray *	－－	－－	96/00
Ctrl Alt	* *	* *	* *
Ctrl Space	39/20	39/20	39/20
Ctrl Caps Lock	－－	－－	－－
Ctrl F1	5E/00	5E/00	5E/00
Ctrl F2	5F/00	5F/00	5F/00
Ctrl F3	60/00	60/00	60/00
Ctrl F4	61/00	61/00	61/00
Ctrl F5	62/00	62/00	62/00
Ctrl F6	63/00	63/00	63/00
Ctrl F7	64/00	64/00	64/00
Ctrl F8	65/00	65/00	65/00
Ctrl F9	66/00	66/00	66/00
Ctrl F10	67/00	67/00	67/00
Ctrl F11	－－	－－	89/00
Ctrl F12	－－	－－	8A/00
Ctrl Num Lock	－－	－－	－－
Ctrl Scroll Lock	－－	－－	－－
Ctrl White Home	77/00	77/00	77/00
Ctrl White Up Arrow	－－	－－	8D/00
Ctrl White PgUp	84/00	84/00	84/00
Ctrl Gray －	－－	－－	8E/00
Ctrl White Left Arrow	73/00	73/00	73/00
Ctrl 5 (数字键盘)	－－	－－	8F/00
Ctrl White Right Arrow	74/00	74/00	74/00
Ctrl Gray ＋	－－	－－	90/00

续表

击　　键	83/84一键盘 标准功能调用	101/102一键盘 标准功能调用	101/102一键盘 扩展功能调用
Ctrl White End	75/00	75/00	75/00
Ctrl White Down Arrow	— —	— —	91/00
Ctrl White PgDn	76/00	76/00	76/00
Ctrl White Ins	— —	— —	92/00
Ctrl White Del	— —	— —	93/00
Ctrl SysReq	— —	* *	* *
Ctrl Key 45 [5]	— —	— —	— —
Ctrl Enter(数字键盘)	— —	1C/0A	E0/0A
Ctrl /（数字键盘）	— —	— —	95/00
Ctrl PrtSc	— —	72/00	72/00
Ctrl Break	— —	00/00	00/00
Ctrl Gray Home	— —	77/00	77/E0
Ctrl Gray Up Arrow	— —	— —	8D/E0
Ctrl Gray Page Up	— —	84/00	84/E0
Ctrl Gray Left Arrow	— —	73/00	73/E0
Ctrl Gray Right Arrow	— —	74/00	74/E0
Ctrl Gray End	— —	75/00	75/E0
Ctrl Gray Down Arrow	— —	— —	91/E0
Ctrl Gray Page Down	— —	76/00	76/E0
Ctrl Gray Insert	— —	— —	92/E0
Ctrl Gray Delete	— —	— —	93/E0
Alt Esc	— —	— —	01/00
Alt 1	78/00	78/00	78/00
Alt 2	79/00	79/00	79/00
Alt 3	7A/00	7A/00	7A/00
Alt 4	7B/00	7B/00	7B/00
Alt 5	7C/00	7C/00	7C/00
Alt 6	7D/00	7D/00	7D/00
Alt 7	7E/00	7E/00	7E/00
Alt 8	7F/00	7F/00	7F/00
Alt 9	80/00	80/00	80/00
Alt 0	81/00	81/00	81/00
Alt —	82/00	82/00	82/00
Alt =	83/00	83/00	83/00
Alt Backspace	— —	— —	0E/00
Alt Tab	— —	— —	A5/00
Alt q	10/00	10/00	10/00
Alt w	11/00	11/00	11/00
Alt e	12/00	12/00	12/00
Alt r	13/00	13/00	13/00
Alt t	14/00	14/00	14/00
Alt y	15/00	15/00	15/00

续表

击　　键	83/84－键盘 标准功能调用	101/102－键盘 标准功能调用	101/102－键盘 扩展功能调用
Alt u	16/00	16/00	16/00
Alt i	17/00	17/00	17/00
Alt o	18/00	18/00	18/00
Alt p	19/00	19/00	19/00
Alt [	－－	－－	1A/00
Alt]	－－	－－	1B/00
Alt Enter	－－	－－	1C/00
Alt Ctrl	* *	* *	* *
Alt a	1E/00	1E/00	1E/00
Alt s	1F/00	1F/00	1F/00
Alt d	20/00	20/00	20/00
Alt f	21/00	21/00	21/00
Alt g	22/00	22/00	22/00
Alt h	23/00	23/00	23/00
Alt j	24/00	24/00	24/00
Alt k	25/00	25/00	25/00
Alt l	26/00	26/00	26/00
Alt ;	－－	－－	27/00
Alt '	－－	－－	28/00
Alt `	－－	－－	29/00
Alt Shift	* *	* *	* *
Alt \	－－	－－	2B/00
Alt z	2C/00	2C/00	2C/00
Alt x	2D/00	2D/00	2D/00
Alt c	2E/00	2E/00	2E/00
Alt v	2F/00	2F/00	2F/00
Alt b	30/00	30/00	30/00
Alt n	31/00	31/00	31/00
Alt m	32/00	32/00	32/00
Alt ,	－－	－－	33/00
Alt .	－－	－－	34/00
Alt /	－－	－－	35/00
Alt Gray *	－－	－－	37/00
Alt Space	39/20	39/20	39/20
Alt Caps Lock	* *	* *	* *
Alt F1	68/00	68/00	68/00
Alt F2	69/00	69/00	69/00
Alt F3	6A/00	6A/00	6A/00
Alt F4	6B/00	6B/00	6B/00
Alt F5	6C/00	6C/00	6C/00
Alt F6	6D/00	6D/00	6D/00
Alt F7	6E/00	6E/00	6E/00

续表

击　　键	83/84－键盘 标准功能调用	101/102－键盘 标准功能调用	101/102－键盘 扩展功能调用
Alt F8	6F/00	6F/00	6F/00
Alt F9	70/00	70/00	70/00
Alt F10	71/00	71/00	71/00
Alt F11	－－	－－	8B/00
Alt F12	－－	－－	8C/00
Alt Num Lock	＊＊	＊＊	＊＊
Alt Scroll Lock	＊＊	＊＊	＊＊
Alt Gray －	－－	－－	4A/00
Alt Gray ＋	－－	－－	4E/00
Alt 7（数字键盘）	#	#	#
Alt 8（数字键盘）	#	#	#
Alt 9（数字键盘）	#	#	#
Alt 4（数字键盘）	#	#	#
Alt 5（数字键盘）	#	#	#
Alt 6（数字键盘）	#	#	#
Alt 1（数字键盘）	#	#	#
Alt 2（数字键盘）	#	#	#
Alt 3（数字键盘）	#	#	#
Alt Del	－－	－－	－－
Alt SysReq	－－	＊＊	＊＊
Alt Key 45［5］	－－	－－	－－
Alt Enter（数字键盘）	－－	－－	A6/00
Alt /（数字键盘）	－－	－－	A4/00
Alt PrtSc	－－	＊＊	＊＊
Alt Pause	－－	＊＊	＊＊
Alt Gray Home	－－	－－	97/00
Alt Gray Up Arrow	－－	－－	98/00
Alt Gray Page Up	－－	－－	99/00
Alt Gray Left Arrow	－－	－－	9B/00
Alt Gray Right Arrow	－－	－－	9D/00
Alt Gray End	－－	－－	9F/00
Alt Gray Down Arrow	－－	－－	A0/00
Alt Gray Page Down	－－	－－	A1/00
Alt Gray Insert	－－	－－	A2/00
Alt Gray Delete	－－	－－	A3/00

注：标准功能调用　　INT 16H 的 0，1，2 号功能
　　扩展功能调用　　INT 16H 的 10H，11H，12H 号功能
　　［＊＊］　　不能进入键盘缓冲区
　　［－－］　　无效功能键
　　［#］　　转换键，ALT＋小键盘输出 ASCII 字符

速查D　BIOS数据区

偏移址	长度	位信息 7654 3210	意　　义
0h	字		RS－232适配器COM1之基地址
2h	字		RS－232适配器COM2之基地址
4h	字		RS－232适配器COM3之基地址
6h	字		RS－232适配器COM4之基地址
8h	字		并行打印机适配器LPT1端口地址
0Ah	字		并行打印机适配器LPT2端口地址
0Ch	字		并行打印机适配器LPT3端口地址
0Eh	字		并行打印机适配器LPT4端口地址
010h	字		设备编码表，调用BIOS中断11H可返回此信息
010h		xx..	软盘驱动器数
		..xx	显示方式：
			00：保留
			01：40X25彩色
			10：80X25彩色
			11：80X25单色
		.... xx..	PC：内存已经安装
			AT：保留
		.... .x..	PC/AT：保留
			PS/2：装有点设备
		.... ..x.	装有8087/287/387协处理器
		x	装有软盘驱动器
011h		xx..	打印机适配器号
		..x.	便携机：内部调制解调器
		...x	装有游戏棒
		.... xxx.	RS－232适配器号
		x	保留
012h	字节		PC/AT：初始化状态
			便携机：电源自检状态
013h	字		内存容量，以K为单位
015h	字		PC/AT：保留
			便携机：电池状态
017h	字		键盘状态
017h		x...	Inset状态
		.x..	Caps Lock状态
		..x.	Num Lock状态
		...x	Scroll Lock状态
		.... x...	Alt键按下
		.... .x..	Ctrl键按下
		.... ..x.	Left Shift键按下
		x	Right Shift键按下
018h		x...	Inset键按下

续表

偏移址	长度	位信息 7654 3210	意义
018h		.x..	Caps Lock 键按下
		..x.	Num Lock 键按下
		...x	Scroll Lock 键按下
		.... x...	Pause 状态
		.... .x..	仅在 AT:SysRq 键按下
		.... ..x.	仅在 AT:Left Alt 键按下
		x	仅在 AT:Left Ctrl 键按下
019h	字节		用 ALT+小键盘键入工作区
01Ah	字		键盘缓冲区首址
01Ch	字		键盘缓冲区尾址
01Eh	32 字节		键盘缓冲区
03Eh	字节		软盘驱动器重校状态
		x...	中断标志
		.xxx	保留
		.... x...	重新调整软盘驱动器 3
		.... .x..	重新调整软盘驱动器 2
		.... ..x.	重新调整软盘驱动器 1
		x	重新调整软盘驱动器 0
03Fh	字节		软盘驱动器电机状态
		x...	读/写操作
		.x..	保留
		..xx	选择操作驱动器
		.... x...	软盘驱动器 3 电机开
		.... .x..	软盘驱动器 2 电机开
		.... ..x.	软盘驱动器 1 电机开
		x	软盘驱动器 0 电机开

续表

偏移址	长度	位信息 7654 3210	意　　义
060h	字		光标类型
062h	字节		当前显示页
063h	字		显示控制器基地址
065h	字		显示方式寄存器口地址(03x8h)
066h	字		颜色设置寄存器口地址(03x9h)
067h	5字节		PC:磁带控制的定时计数器(字),CRC 寄存器(字)和最后输入数值字节 AT:双字指针,指向BIOS开关使80286 由保护虚地址方式转换到实地址方式 时控制返回的位置 PS/2:复位码指针
06Ch	双字		时钟计数器的双字单元
070h	字节		时钟翻转字节
071h	字节		BREAK键标志,如果键入CTRL-Break序 列,则该字节第七位为1
072h	字		复位标志 00064H:不变方式 01234H:跳过内存测试(热启动) 04321H:保护内存(用于PS/2) 05678H:系统挂起(用于便携机) 09ABCH:环境(MFG)测试(用于便携机) 0ABCDH:电源自检循环(用于便携机)
074h	字节		最近一次硬盘操作状态,参见INT 13H 1号功能
075h	字节		硬盘驱动器数
076h	字节		硬盘驱动器控制(仅用于XT)
077h	字节		硬盘控制器端口,一般为0320H　(仅用于XT)
078h	字节		并行打印机适配器LPT1超时值
079h	字节		并行打印机适配器LPT2超时值
07Ah	字节		并行打印机适配器LPT3超时值
07Bh	字节		并行打印机适配器LPT4超时值
07Ch	字节		RS-232适配器COM1超时值
07Dh	字节		RS-232适配器COM2超时值
07Eh	字节		RS-232适配器COM3超时值
07Fh	字节		RS-232适配器COM4超时值
080h	字		键盘缓冲区首址偏移指针
082h	字		键盘缓冲区尾址偏移指针
084h	字节		EGA/PGA,屏幕显示行数,为真正显示行数减一
085h	字节		EGA/PGA:字符高度
087h	字节		VGA显示控制状态
		x...	清除RAM
		.xx.	显示适配器上存储容量 00:64K　　01,128K

续表

偏移址	长度	位信息 7654 3210	意　　义
087h			10:192K　　11:256K
		.... 0...	EGA/VGA 显示适配器
		.... .x..	等待显示有效
		.... ..0.	彩色/ECD 显示器
		.... ..1.	单色显示器
		0	在 350 线方式下使用 ECD 显示器变换光标
		1	禁止光标变换
088h	字节		EGA/VGA 交换数据
		xxxx	特征连接位
		 xxxx	选项交换位
089h	字		EGA/VGA 控制位
		x...	200 线
		...x	400 线
		.... x...	未加载调色板
		.... .x..	单色显示器
		.... ..x.	灰度缩放
08Ah	字节		显示组合码表(DCC)索引
08Bh	字节		软驱介质控制,仅用于 85 年 1 月 10 日
			后生产的 PC/AT/ PS/2
		xx..	最后一个软驱数据传输率
			00:500K/秒
			01:300K/秒
			10:250K/秒
			11:保留
		..xx	最后一个软驱步进速率
		.... xx..	操作开始时的数据传输率
			00:500K/秒
			01:300K/秒
			10:250K/秒
			11:保留
		xx	保留
08Ch	字节		硬盘控制器状态,仅用于 85 年 1 月 10 日
			后生产的 PC/AT/ PS/2
08Dh	字节		硬盘控制器错误状态,仅用于 85 年 1 月
			10 日后生产的 PC/AT/ PS/2
08Eh	字节		硬盘中断控制,仅用于 85 年 1 月 10 日后
			生产的 PC/AT/ PS/2
08Fh	字节		软盘控制器信息
		.x..	确定驱动器 1
		..x.	驱动器 1 是多速率的
		...x	驱动器 1 支持变化线
		.... .x..	确定驱动器 0

续表

偏移址	长度	位信息 7654 3210	意　　义
08Fh		.... ..x.	驱动器0是多速率的
		x	驱动器0支持变化线
090h	字节		软驱0介质状态,仅用于85年1月10日后生产的PC/AT/ PS/2
		xx..	软驱数据传输率
			00:500K/秒
			01:300K/秒
			10:250K/秒
			11:保留
		..x.	需要双倍步长(360K软盘/1.2M软驱)
		...x	介质已建立
		.... x...	保留
		xxx	介质/驱动器状态
			000:不能确定360K盘在360K驱动器中
			001:不能确定360K盘在1.2M驱动器中
			010:不能确定1.2M盘在1.2M驱动器中
			011:确定360K盘在360K驱动器中
			100:确定360K盘在1.2M360K驱动器中
			101:确定1.2M盘在1.2M驱动器中
			110:保留
			111:确定为720K/1.44M磁盘
091h	字节		软驱1介质状态,仅用于85年1月10日后生产的PC/AT/ PS/2
092h	字节		软盘设备0服务工作区
093h	字节		软盘设备1服务工作区
094h	字节		软盘驱动器0当前道
095h	字节		软盘驱动器1当前道
096h	字节		键盘状态及类型标志
		x...	读ID
		.x..	最后一个字符是第一个ID字符
		..x.	如果读ID和键盘,则强置Num Lock
		...x	装有101/102键键盘
		.... x...	右Alt键按下
		.... .x..	右Ctrl键按下
		.... ..x.	最后一个代码是0E0H隐藏码
		x	最后一个代码是0E1H隐藏码
097h	字节		键盘LED状态
		x...	键盘传输错误
		.x..	方式指示器更新
		..x.	重发接收标志
		...x	收到响应
		.... x...	保留(必须为0)

续表

偏移址	长度	位信息 7654 3210	意　　义
097h		.... .x..	LED 状态位:Caps Lock LED 状态
		.... ..x.	LED 状态位:Num Lock LED 状态
		x	LED 状态位:Scroll Lock LED 状态
098h	双字		用户等待完成标志指针
09Ch	双字		用户等待计数,以微秒为单位
0A0h	字节		等待激活标志,报告实时钟等待状态
		x...	等待时间已过
		.xxx x...	保留
		.... .x..	报警挂起(用于便携机)
		.... ..x.	使用 INT 4Ah 报警挂起服务(便携机)
		x	等待激活
0A1h	7 字节		网卡数据区
0A8h	双字		EGA 显示参数表指针
0ACh	44 字节		保留(单显/CGA)
0ACh	双字		EGA 动态存贮区指针
0B0h	双字		EGA 字符方式辅助字符发生器指针
0B4h	双字		EGA 图形方式辅助字符发生器指针
0B8h	双字		EGA 二级存贮区指针
0BCh	7 字节		保留,只能置为 0
0C0h	64 字节		保留
0F0h	16 字节		ICA(Intra－Application Communications Area)数据区
0:500h	字节		打印屏幕操作状态: 0:屏幕拷贝打印操作成功或无法进行 1:屏幕拷贝打印操作正在进行 0FFH:屏幕拷贝打印操作错
0:501h	3 字节		保留
0:504h	字节		单软驱逻辑驱动器号 0:逻辑驱动器 A 1:逻辑驱动器 B 0FFh:逻辑驱动器 A
0:505h	250 字节		保留

速查E　BIOS功能调用表

中断号	子功能号(AH值)/BIOS默认地址	功　　能
0h		除法溢出中断
01h		单步中断
02h		硬件中断
03h		断点中断
04h		溢出中断
05h	FFF5:4h	屏幕拷贝(按 Print Screen 键激活)
08h	FFEA:5h	时钟中断(IRQ0),每秒钟执行 18.2 次
09h		键盘中断(IRQ1)
0Bh		COM2 控制器中断(IRQ3)
0Ch		COM1 控制器中断(IRQ4)
0Dh		LPT2 控制器中断(IRQ5)
0Eh	FEF5:7h	磁盘控制器中断(IRQ6)
0Fh		LPT1 控制器中断(IRQ7)
10h		显示 I/O
	00h	置当前页显示方式 输入:AL=所置显示模式
	01h	置光标类型 输入:CH=光标起始行,CL=光标结束行
	02h	置光标位置 输入:BH=页号 CH=行号,CL=页号
	03h	读光标位置 输入:BH=页号 输出:CH/CL=光标起始/结束行 DH/DL=行/列
	04h	读光笔位置 输出:AX=光笔触发标志,1=触发,0=未触发 BX=象素列,CH=象素行 DH/DL=字符行/列
	05h	置当前显示页 输入:AL=页号
	06h	当前显示页上滚 输入:AL=上滚行数,0为初始化窗口 BH=字符填充属性 CH/CL=窗口左上角行/列坐标 DH/DL=窗口右下角行/列坐标
	07h	当前显示页下滚 输入:AL=下滚行数,0为初始化窗口 BH=字符填充属性 CH/CL=窗口左上角行/列坐标

续表

中断号	子功能号(AH 值)/BIOS 默认地址	功　　能
10h		DH/DL＝窗口右下角行/列坐标
	08h	读光标位置的字符和属性 输入:BH＝页号 输出:AH/AL＝字符/属性
	09h	在当前光标处写字符和属性(不改变光标位置) 输入:AL＝字符 BH/BL＝页号/属性 CX＝重复写字符的个数
	0Ah	在当前光标处写字符(不改变光标位置) 输入:AL＝字符 BH/BL＝页号/图形模式时字符前景色 CX＝重复写字符的个数
	0Bh	置彩色调色板 输入: BH＝0　置调色板 BL＝0　调色板 绿/红/褐 BL＝0　调色板 青/绛/白 BH＝1　置颜色 BL＝高四位/低四位为前景/背景色彩
	0Ch	写点 输入:AL＝颜色 CX＝象素列,DL＝象素行
	0Dh	读点 输入:BX＝页号 CX＝象素列,DL＝象素行 输出:AL＝颜色
	0Eh	以电传方式写字符,光标移动,解释命令字符 输入:AL＝字符 BX＝页号/图形模式时字符前景色 命令字符: 7　响铃(BEEP) 8　回格(BackSpace) 0Ah　换行(Line feed) 0Dh　回车(CR)
	0Fh	取当前显示方式 输出:AH/AL＝每行字符数/当前显示模式 BH＝当前显示页号
	13h	写字符串 输入:AL＝写方式 0　使用 BL 为属性字,光标不动 1　使用 BL 为属性字,光标移动

续表

中断号	子功能号(AH值)/BIOS默认地址	功　　能
10h		2　字符串结构为[字符,属性],光标不动 3　字符串结构为[字符,属性],光标移动 BH/BL=页号/属性 CX=字符串长度 DH/DL=起始行/列 ES:BP=字符串起始位置 命令字符: 7　响铃(BEEP) 8　回格(BackSpace) 0Ah　换行(Line feed) 0Dh　回车(CR)
11h	FF84:0Dh	取设备信息,见BIOS数据区0:410h说明
12h	FF84:1h	取内存容量 输出:AX=内存大小,单位为K
13h	FEC5:9h 00h	磁盘I/O 复位磁盘 输入:AL=物理驱动器号
	01h	取磁盘驱动器状态 输入:DL=0,1　软盘 080h,081h　硬盘 输出:AH=驱动器状态 0　无错 1　无效功能调用 2　未找到地址标记,扇区ID无效 3　磁盘写保护 4　扇区未找到 5　复位操作失败 6　驱动器中未插入磁盘 7　驱动器参数无效 8　DMA故障 9　DMA越段错 0Ah　非法扇区 0Bh　非法柱面(硬盘) 0Ch　所需介质类型未找到(软盘) 0Dh　无效的扇区号(硬盘) 0Eh　测试到控制数据地址标志(硬盘) 0Fh　DMA超范围(硬盘) 010h　读盘错(CRC或ECC错) 011h　奇偶校验错,但ECC已纠正错误 020h　控制器错 040h　查找操作失败

续表

中断号	子功能号 (AH 值)/BIOS 默认地址	功　　能
13h		080h　驱动器超时 0AAh　驱动器未准备好 0BBh　未定义错误 0CCh　写失败 0E0h　状态错 0FFh　读测试操作失败
	02h	读扇区 输入:AL＝读入扇区数 软盘仅能读同一磁道上的扇区 硬盘最多可读 128 个扇区 CH＝磁道号 CL＝低六位为起始扇区号　高二位为磁道号高位 DH/DL＝磁头号/驱动器号　ES:BX＝缓冲区指针
	03h	写扇区 输入:AL＝写入扇区数 软盘仅能写同一磁道上的扇区 硬盘最多可写 128 个扇区 CH＝磁道号 CL＝低六位为起始扇区号 高二位为磁道号高位 DH/DL＝磁头号/驱动器号 ES:BX＝缓冲区指针
	04h	检测软盘扇区 输入:AL＝检测扇区数 软盘仅能检测一磁道上的扇区 硬盘最多可检测 255 个扇区 CH＝磁道号 CL＝低六位为起始扇区号 高二位为磁道号高位 DH/DL＝磁头号/驱动器号
	05h	格式化磁道 输入:AL＝每道扇区数 CH＝磁道号 CL＝低六位为起始扇区号 高二位为磁道号高位 DH/DL＝磁头号/驱动器号 ES:BX＝扇区 ID 地址指针 ID 结构:[道号,头号,扇区号,扇区字节数]

续表

中断号	子功能号(AH值)/BIOS默认地址	功　能
13h		扇区字节数:0　128字节/扇区 1　256字节/扇区 2　512字节/扇区 3　1024字节/扇区
	08h	取当前驱动器参数(硬盘) 输入:DL=硬盘驱动器号 输出:CH=磁道号 CL=低六位为起始扇区号 高二位为磁道号高位 DH/DL=磁头号/驱动器号
	09h	初始化双驱动器
	0Ch	查找柱面(磁道) 输入:CH=磁道号 CL=低六位为起始扇区号 高二位为磁道号高位 DH/DL=磁头号/驱动器号
	0Dh	备用硬盘复位 输入:DL=硬盘驱动器号
	10h	检测硬盘驱动器准备 输入:DL=硬盘驱动器号
	11h	重新调整硬盘驱动器
		输入:DL=硬盘驱动器号
	14h	硬盘控制器内部诊断
	15h	取DASD磁盘类型 输入:DL=驱动器号 输出:AX=类型 0　磁盘驱动器不存在 1　磁盘驱动器不能检测介质变化 2　磁盘驱动器能检测介质变化 3　驱动器为硬盘,其容量为CX:DX
	16h	软磁盘变化状态 输入:DL=驱动器号 输出:AL=状态 0　软盘变化信号未激活 1　无效软盘参数 6　软盘变化信号未激活 080h　软盘驱动器未准备
	17h	置DASD格式类型

续表

中断号	子功能号(AH 值)/BIOS 默认地址	功　　能
13h		输入:AL=DASD 类型 1 360K 磁盘/360K 驱动器 2 360K 磁盘/1.2M 驱动器 3 1.2M 磁盘/1.2M 驱动器 4 720K 磁盘/720K 驱动器 DL=驱动器
	18h	置软盘介质类型 输入:CH=磁道号 CL=低六位为起始扇区号 高二位为磁道号高位 DL=驱动器号 输出:ES:DI=介质参数表(见 INT 1Eh)
	19h	磁头复位 输入:DL=驱动器号
14h	FE73:9h	RS-232 串行口 I/O
	00h	初始化串行口 输入:AL=初始化参数 DX=串行口号 输出:AH/AL=通信口状态/调制解调器状态 通信口参数/初始化参数 xxx. .... 波特率 000=110 波特,001=150 波特 010=300 波特,011=600 波特 100=1200 波特,101=2400 波特 110=4800 波特,111=9600 波特 ...x x... 奇偶校验 00=无奇偶校验,01=奇校验 10=无奇偶校验,11=偶校验 .... .x.. 终止位,0=一位,1=两位 .... ..xx 字符长度,10=7 位,11=8 位 调制解调器状态 x... .... 检测到接收线信号 .x.. .... 铃指示 ..x. .... 数据发送准备好 ...x .... 清除发送 .... x... 检测的接收线信号改变 .... .x.. 后沿铃标志 .... ..x. 数据发送准备好,状态改变 ....x 清除发送,状态改变

续表

中断号	子功能号(AH值)/BIOS默认地址	功　　能
		通信状态 x... ....　超时 .x.. ....　传送移位寄存器空 ..x. ....　传送保存寄存器空 ...x ....　检测到BREAK .... x...　帧检验错 .... .x..　奇偶校验错 .... ..x.　溢出 ....x　数据准备好
	01h	发送一字符 输入:AL=字符 DX=串行口号 输出:AH=通信口状态
	02h	接收一字符 输入:DX=串行口号 输出:AH/AL=通信口状态/字符
	03h	取串行口状态 输入:DX=串行口号 输出:AH/AL=通信口状态/调制解调器状态
	04h	初始化扩展串行口(PS/2)
	05h	扩展串行口控制
15h	FF85+9h	磁带I/O
	00h	启动盒式磁带机
	01h	停止盒式磁带机
	02h	读磁带数据块
	03h	写磁带数据块 输入:AL=扫描码 输出:AL=扫描码
	80h	打开设备 输入:BX=设备ID 进程ID
	81h	关闭设备 输入:BX=设备ID 进程ID
	82h	程序终止 输入:BX=设备ID
	83h	事件等待

续表

中断号	子功能号(AH 值)/BIOS 默认地址	功　　能
		输入:AL=0　设置间隔 1　取消间隔 CX:DX=时间间隔值(微秒) ES:BX=字节指针
	84h	游戏控制杆支持 输入:DX=0　取开关当前置位情况 1　取当前操纵杆的 X,Y 值 输出:AL=取开关当前置位情况(DX=0 时) AX=A 控制杆 X 位置 BX=A 控制杆 Y 位置 CX=B 控制杆 X 位置 DX=B 控制杆 Y 位置
	85h	系统请求击键等待 输入:AL=0 Sys Rq 键按下 1 Sys Rq 键松开
	86h	SYS Req 键支持,设备等待 输入:CX:DX=等待时间(微秒)
	87h	成块传送 输入:CX=传送字节数 ES:SI=GDT 指针
	88h	取扩展内存容量 输出:AX=内存数,单位为 K
	89h	将内存转换为保护模式 输入:BH=IDT 中前八个硬盘中断的偏移字节 BL=IDT 中前八个硬盘中断的偏移字节 ES:SI=GDT 指针
	90h	设备忙循环(等待) 输入:AL=类型码 00　硬盘(超时) 01　软盘(超时) 02　键盘(未超时) 03　点设备(超时) 080h　网络(未超时) 0FDh　软驱电机开(超时) 0FEh　打印机(超时) ES:BX=网络控制块指针
	91h	建立中断结束标志(POST)

续表

中断号	子功能号(AH值)/BIOS默认地址	功　　能
15h		输入:AL=类型码 00　硬盘(超时) 01　软盘(超时) 02　键盘(未超时) 03　点设备(超时) 080h　网络(未超时) 0FDh　软驱电机开(超时) 0FEh　打印机(超时)
	C0h	返回系统参数 输出:ES:BX=系统描述表指针 系统描述表 偏移　长度　Bits　意　义 0　字　系统描述表字节数 2　字节　机型字节 3　字节　子模型字节 4　字节　BIOS修正级(0=先释放) 5　字节　功能信息 x... ....　硬盘BIOS使用DMA3 .x.. ....　有二级中断 ..x. ....　有实时钟 ...x ....　键盘截取 .... x...　等待外部事件 .... .x..　扩展BIOS区已分配 .... ..x.　PS/2型I/O通道 ....x　保留
16h	FE82:Eh	键盘I/O
	00h	读下一键盘字符,不清除缓冲区 输出:AH/AL=键盘扫描码/ASCII字符
	01h	检测字符是否准备好,不改变缓冲区 输出:AH/AL=键盘扫描码/ASCII字符
	02h	读当前转换键状态 输出:AL=状态,见低地址0:417h的说明
	03h	键盘拍发速率与延迟 输入:AL=0　恢复默认键盘拍发速率与延迟 1　增加延迟时间 2　减慢拍发速率1/2 3　增加延迟时间,减慢拍发速率1/2 4　关闭重发字符 5　设置拍发速率与延迟 BH/BL=延迟时间/拍发速率

续表

中断号	子功能号(AH 值)/BIOS 默认地址	功　　能
16h	05h	键盘写,将字符放回缓冲区 输入:AH/AL=键盘扫描码/ASCII 字符
	10h	扩展键盘读,清除缓冲区 输出:AH/AL=键盘扫描码/ASCII 字符
	11h	读扩展键盘的击键状态,不清除缓冲区 输出:AH/AL=键盘扫描码/ASCII 字符
	12h	取扩展键盘的移位状态 输出:AH/AL=扩展的移位状态/移位状态 即低地址 0:418h/0:417h,但 0:418h 的 第七位(080h)表示 SysRq 键按下
17h	FEFD:2h	打印机 I/O
	00h	打印一字符 输入:AL=字符 DX=打印机号 输出:AH=状态 Bits　　意　义 x... ....　打印机准备好 .x.. ....　确认 ..x. ....　无纸 ...x ....　选择打印机 .... x...　I/O 错 .... .xx.　保留 ....x　超时
	01h	初始化打印口 输入:DX=打印机号 输出:AH=状态
	02h	取打印机状态 输入:DX=打印机号 输出:AH=状态
18h		ROM BASIC 语言
19h		引导装入程序
1Ah	FFE6:Eh	实时时钟
	00h	读当前时钟值 输出:CX=计时器高位 DX=计时器低位
	01h	置当前时钟值 输入:CX=计时器高位 DX=计时器低位
	02h	读电池供电时钟时间

续表

中断号	子功能号(AH值)/BIOS默认地址	功　　能
1Ah	02h	输出:CH/CL=BCD码小时数/BCD码分钟数 DH=BCD码秒数 DL=保存时间选项,1=可操作 0=未使用
	03h	置电池供电时钟时间 输入:CH/CL=BCD码小时数/BCD码分钟数 DH=BCD码秒数 DL=保存时间选项,1=可操作 0=未使用
	04h	读电池供电时钟日期 输出:CH/CL=BCD码世纪数/BCD码年数 DH/DL=BCD码月份/BCD码日期
	05h	置电池供电时钟日期 输入:CH/CL=BCD码世纪数/BCD码年数 DH/DL=BCD码月份/BCD码日期
	06h	置闹钟,到指定时间后执行 4Ah 中断 输入:CH/CL=BCD码小时数/BCD码分钟数 DH=BCD码秒数
	07h	复位闹钟
	08h	读当前时钟日计数(仅用于便携机)
	09h	置当前时钟日计数(仅用于便携机)
	0Ah	读系统时钟时间(仅用于 XT)
	0Bh	置系统时钟时间(仅用于 XT)
1Bh	FFF53h	键盘终止地址(CTRL-BREAK)
1Ch	FFF53h	定时器信号
1Dh	FF0A:4h	视频参数表
1Eh	0:522h FEFC:7h	软盘参数表 重要参数结构: 字节　长度　　说　明 3　字节　软盘扇区长度 0=128 字节,1=256 字节 2=512 字节,3=1024 字节 4　字节　每道扇区数 8　字节　格式化填充数据,默认值为 F6h
1Fh		图形字符扩展码
40h	FEC5:9h	软盘 BIOS
41h		硬盘 0 参数表
		参数表结构: 字节　长度　说　明 0　字　硬盘柱面数 2　字节　硬盘磁头数

续表

中断号	子功能号 (AH 值)/BIOS 默认地址	功　　能
41h		0Eh　字节　每道扇区数
46h		硬盘 1 参数表 参数表结构： 字节　长度　说　明 0　字　硬盘柱面数 2　字节　硬盘磁头数 0Eh　字节　每道扇区数
49h		指向键盘增强服务变换表
4Ah		用户闹钟
70h		实时钟(IRQ8)
71h		IRQ9 中断
72h		IRQ10 中断
73h		IRQ11 中断
74h		IRQ12 中断
75h		IRQ13 中断
76h		IRQ14 中断
77h		IRQ15 中断

速查 F　系统功能调用表

中断	功　　能
20h	终止程序，执行时 CS 应指向 PSP 所在段，用于 COM 程序
21h	DOS 系统功能调用
22h	程序终止时 DOS 返回地址，用户不能直接调用
23h	Ctrl-Break 处理地址，用户不能直接调用
24h	严重错误处理地址，用户不能直接调用 输出： AH　错误信息 磁盘错误时 AH 寄存器用法 Bits　含　义 x... ....　0 表示磁盘错误 .x.. ....　保留 ..x. ....　0＝不允许忽略错误 1＝允许忽略错误 ...x ....　0＝不允许重试 1＝允许重试 .... x...　0＝不允许功能调用失败 1＝允许功能调用失败 .... .xx.　出错的磁盘区域 00＝MS－DOS 区域

续表

中断	功　能
24h	01=文件分配表(FAT) 10=根目录 11=文件区域 ....x　引起错误类型 0=读盘,1=写盘 AL　对策 00H　忽略错误 01H　重试操作 02H　终止程序 03H　系统调用失败(V3及以上版本) DI　错误代码 00H　写保护错误 01H　未知的单元 02H　驱动器未准备好 03H　未知的命令 04H　CRC错 05H　命令结构长度错误 06H　查找错误 07H　未知介质类型 08H　扇区未找到 09H　打印机缺纸 0AH　写错误 0BH　读错误 0CH　一般错误 0FH　非法更换磁盘 BP：SI　指向设备驱动程序首部的指针 STACK　设置如下： 栈底 Flags　原始调用程序的FLAGS寄存器 CS　原始调用程序的CS寄存器 IP　原始调用程序的IP寄存器 ES　原始调用程序的ES寄存器 DS　原始调用程序的DS寄存器 BP　原始调用程序的BP寄存器 DI　原始调用程序的DI寄存器 SI　原始调用程序的SI寄存器 DX　原始调用程序的DX寄存器 CX　原始调用程序的CX寄存器 BX　原始调用程序的BX寄存器 AX　原始调用程序的AX寄存器 Flags　DOS的标志寄存器 CS　DOS的CS寄存器

续表

中断	功 能
24h	IP DOS 的 IP 寄存器 栈顶
25h	绝对磁盘读 输入： 所读磁盘分区<32M： AL＝驱动器号：0＝A，1＝B，2＝C.. CX＝读扇区数 DX＝起始逻辑扇区号 DS：BX＝磁盘数据传送地址 所读磁盘分区>32M：(DOS4，DOS5，COMPAQ3.31) AL＝驱动器号：0＝A，1＝B，2＝C.. CX＝－1(即 0FFFFh) DS：BX＝参数表 参数表格式 双字 起始扇区号 字 读扇区数 双字 缓冲区地址 输出：AH/AL＝错误代码，其中 AL 为 INT 24H 所返回的错误代码 AH 错误代码 80H 磁盘控制器没有响应 40H 寻找操作错误 20H 控制器出错 10H 数据错误(CRC 错误) 08H DMA 失败 04H 未找到请求的扇区 03H 写保护错误 02H 地址标记错误 01H 命令错误 注：返回时堆栈不正常，需 POP 一次
26h	绝对磁盘写 输入： 所写磁盘分区<32M： AL＝驱动器号：0＝A，1＝B，2＝C.. CX＝写扇区数 DX＝起始逻辑扇区号 DS：BX＝磁盘数据传送地址 所写磁盘>32M：(DOS4，DOS5，COMPAQ3.31) AL＝驱动器号：0＝A，1＝B，2＝C.. CX＝－1 DS：BX＝参数表 参数表格式 双字 起始扇区号

续表

中断	功　　能
26h	字　　写扇区数 双字　　缓冲区地址 输出，AH/AL＝错误代码，其中 AL 为 INT 24H 所返回的错误代码 AH 错误代码 80H　磁盘控制器没有响应 40H　寻找操作错误 20H　控制器出错 10H　数据错误(CRC 错误) 08H　DMA 失败 04H　未找到请求的扇区 03H　写保护错误 02H　地址标记错误 01H　命令错误 注，返回时堆栈不正常，需弹出一个字
27h	终止并驻留程序，仅对 COM 程序有效 输入，CS＝指向 PSP 所在段 DX＝程序驻留部分末字节＋1
28h	DOS 安全使用
29h	快速写字符
2Ah	Microsoft 网络接口
2Eh	基本 SHELL 程序装入
2Fh	多路服务中断 01　00　打印安装检查 01　把文件交给假脱机打印 02　打印打印队列中的文件 03　取消打印队列全部文件 04　暂停打印任务 05　取消打印暂停状态 06　获取打印设备 05　获取外部严重错误处理程序安装状态 06　00　获取 ASSIGN.COM/ASSIGN.EXE 安装状态 08　00　获取 DRIVER.SYS 安装状态 10h　00　获取 SHARE.EXE 安装状态 11h　00　获取网络重定向安装状态 12h　00　获取 DOS 安装状态 01　刷新文件 02　获取中断向量地址 03　获取 DOS 数据段 04　路径分隔符规范化 05　输出一个字符 06　激发严重错误 07　移动磁盘缓冲区

续表

中断			功 能
2Fh		08	减少用户计数
		0Ch	打开 IOCTL 由 DOS 使用
		0Dh	获取关闭文件的日期和时间
		0Eh	搜索缓冲区链
		10h	发现修改的缓冲区,时间延迟
		11h	ASCIIZ 文件名规范化
		12h	发现 ASCIIZ 字符串长度
		13h	大小写和国别变换
		14h	比较 32 位数字
		16h	获取 DCB 地址
		17h	获取 LDT 地址
		18h	获取用户堆栈地址
		19h	设置 LDT 指针
		1Ah	从路径名取驱动码
		1Bh	校对闰年
		1Ch	从月初计算天数
		1Dh	计算日期
		1Eh	比较字符串
		1Fh	LDT 初始化
		20h	获取 DCB 号
		21h	扩展 ASCIIZ 路径名
		22h	翻译扩充错误代码
		24h	执行延时
		25h	取 ASCIIZ 字符串长度
		26h	打开文件
		27h	关闭文件
		28h	定位文件指针
		29h	读文件
		2Bh	IOCTL 接口
		2Dh	获取扩充错误码
		2Fh	贮存 DX
	14h	00	获取 NLSFUNC 安装状态
	15h		CDROM 接口
	16h	80h	MS－DOS 无效调用
	1Ah	00	获取 ANSI.SYS 安装的状态
	43h	00	获取 XMS 驱动器安装的状态
		10h	获取 XMS 驱动器入口指针地址
	48h	00	获取 DOSKEY.COM 安装的状态
		10h	读命令行
	4Bh	01	建立消息链
		02	检测切换程序
		03	分配切换程序 ID

续表

中断	功能		
2Fh		04	释放切换程序ID
		05	识别立即数
	ADh	80h	取KEYB.COM版本号
		81h	取KEYB.COM活动代码页
		82h	设置KEYB.COM国别标志
		83h	获取KEYB.COM国别标志
	B0h	00	获取GRAPHTABL.COM安装的状态
	B7h	00	检测APPEND安装
		02	获取APPEND版本
		04	获取APPEND路径指针
		06	获取APPEND功能状态
		07	设置APPEND功能状态
		11h	设置返回找到文件名状态

速查G I/O端口功能表

I/O地址	功能
0	DMA通道0,内存地址寄存器
1	DMA通道0,传输计数寄存器
2	DMA通道1,内存地址寄存器
3	DMA通道1,传输计数寄存器
4	DMA通道2,内存地址寄存器
5	DMA通道2,传输计数寄存器
6	DMA通道3,内存地址寄存器
7	DMA通道3,传输计数寄存器
8	DMA通道0—3的状态寄存器
0Ah	DMA通道0—3的屏蔽寄存器
0Bh	DMA通道0—3的方式寄存器
0Ch	DMA清除字节指针
0Dh	DMA主清除字节
0Eh	DMA通道0—3的清屏蔽寄存器
0Fh	DMA通道0—3的写屏蔽寄存器
19h	DMA起始寄存器
20h—3Fh	可编程中断控制器使用
40h	可编程中断计时器使用,读/写计数器0
41h	可编程中断计时器寄存器
42h	可编程中断计时器杂项寄存器

续表

I/O 地址	功　能
43h	可编程中断计时器,控制字寄存器
44h	可编程中断计时器,杂项寄存器(AT)
47h	可编程中断计时器,计数器 0 的控制字寄存器
48h－5Fh	可编程中断计时器使用
60h－61h	键盘输入数据缓冲区
61h	AT,8042 控制寄存器
	XT,8255 输出寄存器
62h	8255 输入寄存器
63h	8255 命令方式寄存器
64h	8042 键盘输入缓冲区/8042 状态
65－6Fh	8255/8042 专用
70h	CMOS RAM 地址寄存器
71h	CMOS RAM 数据寄存器
80h	生产测试端口
81h	DMA 通道 2,页表地址寄存器
82h	DMA 通道 3,页表地址寄存器
83h	DMA 通道 1,页表地址寄存器
87h	DMA 通道 0,页表地址寄存器
89h	DMA 通道 6,页表地址寄存器
8Ah	DMA 通道 7,页表地址寄存器
8Bh	DMA 通道 5,页表地址寄存器
8Fh	DMA 通道 4,页表地址寄存器
93－9Fh	DMA 控制器专用
A0h	NMI 屏蔽寄存器/可编程中断控制器 2
A1h	可编程中断控制器 2 屏蔽
C0h	DMA 通道 0,内存地址寄存器
C2h	DMA 通道 0,传输地址寄存器
C4h	DMA 通道 1,内存地址寄存器
C6h	DMA 通道 1,传输地址寄存器
C8h	DMA 通道 2,内存地址寄存器
CAh	DMA 通道 2,传输地址寄存器
CCh	DMA 通道 3,内存地址寄存器
CEh	DMA 通道 3,传输地址寄存器
D0h	DMA 状态寄存器

续表

I/O地址	功　能
D2h	DMA写请求寄存器
D4h	DMA屏蔽寄存器
D6h	DMA方式寄存器
D8h	DMA清除字节指针
DAh	DMA主清
DCh	DMA清屏蔽寄存器
DEh	DMA写屏蔽寄存器
DF－EFh	保留
F0－FFh	协处理器使用
100－16Fh	保留
0170h	1号硬盘数据寄存器
0171h	1号硬盘错误寄存器
0172h	1号硬盘数据扇区计数
0173h	1号硬盘扇区数
0174h	1号硬盘柱面(低字节)
0175h	1号硬盘柱面(高字节)
0176h	1号硬盘驱动器/磁头寄存器
0177h	1号硬盘状态寄存器
01F0h	0号硬盘数据寄存器
01F1h	0号硬盘错误寄存器
01F2h	0号硬盘数据扇区计数
01F3h	0号硬盘扇区数
01F4h	0号硬盘柱面(低字节)
01F5h	0号硬盘柱面(高字节)
01F6h	0号硬盘驱动器/磁头寄存器
01F7h	0号硬盘状态寄存器
1F9－1FFh	保留
200－20Fh	游戏控制端口
210－21Fh	扩展单元
0278h	3号并行口,数据端口
0279h	3号并行口,状态端口
027Ah	3号并行口,控制端口
2B0－2DFh	保留
02E0h	EGA/VGA使用

续表

I/O 地址	功　能
02E1h	GPIP(0 号适配器)
02E2h	数据获取(0 号适配器)
02E3h	数据获取(1 号适配器)
2E4－2F7h	保留
02F8h	2 号串行口,发送/保持寄存器
02F9h	2 号串行口,中断有效寄存器
02FAh	2 号串行口,中断 ID 寄存器
02FBh	2 号串行口,线控制寄存器
02FCh	2 号串行口,调制解调控制寄存器
02FDh	2 号串行口,线状态寄存器
02FEh	2 号串行口,调制解调状态寄存器
02FFh	保留
300－31Fh	原型卡
0320h	硬盘适配器寄存器
0322h	硬盘适配器控制/状态寄存器
0324h	硬盘适配器提示/中断状态寄存器
325－347h	保留
348－357h	DCA3278
366－36Fh	PC 网络
0372h	软盘适配器数据输出/状态寄存器
375－376h	软盘适配器数据寄存器
0377h	软盘适配器数据输入寄存器
0378h	2 号并行口,数据端口
0379h	2 号并行口,状态端口
037Ah	2 号并行口,控制端口
380－38Fh	SDLC 及 BSC 通讯
390－393h	Cluster 适配器 0
3A0－3AFh	BSC 通讯
3B0－3BFh	MDA 视频寄存器
03BCh	1 号并行口,数据端口
03BDh	1 号并行口,状态端口
03BEh	1 号并行口,控制端口
03C0－3CFh	EGA/VGA 视频寄存器
03D0－3D7h	CGA 视频寄存器

续表

I/O 地址	功　能
03F0—03F7h	软盘控制器寄存器
03F8h	1号串行口,发送/保持寄存器
03F9h	1号串行口,中断有效寄存器
03FAh	1号串行口,中断ID寄存器
03FBh	1号串行口,线控制寄存器
03FCh	1号串行口,调制解调控制寄存器
03FDh	1号串行口,线状态寄存器
03FEh	1号串行口,调制解调状态寄存器
03FFh	保留

附录H　CMOS数据格式

偏　移	长　度	Bits 7654　3210	说　明
0h	字节		秒数
1h	字节		闹钟秒数
2h	字节		分数
3h	字节		闹钟分数
4h	字节		小时数
5h	字节		闹钟小时数
6h	字节		星期数
7h	字节		日期
8h	字节		月份
9h	字节		年份
0Ah	字节		状态寄存器A
0Bh	字节		状态寄存器B
0Ch	字节		状态寄存器C
0Dh	字节		状态寄存器D
0Eh	字节		诊断状态
		x...	实时时钟已掉电
		.x..	CMOS检查和出错
		..x.	错误的配制信息
		...x	CMOS记录内存字节出错
		.... x...	硬盘错

续表

偏 移	长 度	Bits 7654 3210	说 明
0Eh		.... .x..	CMOS 时间错
		xx	保留
0Fh	字节		停机状态
			00h＝加电或软复位
			01h＝内存容量通过测试
			02h＝内存测试通过
			03h＝内存测试失败
			04h＝POST 结束,引导系统
			05h＝带有 EOI 的 JMP 双字指针
			06h＝保护性测试通过
			07h＝保护性测试失败
			09h＝INT15h 块放移动
			0Ah＝没有 EOI 的 JMP 双字指针
			0Bh＝80286 使用
010h	字节		软盘驱动器的类型
		xxxx	软盘驱动器 0 的类型
		 xxxx	软盘驱动器 1 的类型
			0000＝无驱动器
			0001＝360K 驱动器
			0010＝1.2M 驱动器
			0011＝720K 驱动器
			0100＝1.44M 驱动器
011h	字节		保留
012h	字节		硬盘驱动器的类型
		xxxx	硬盘驱动器 0 的类型
		 xxxx	硬盘驱动器 1 的类型
			0000＝无驱动器
013h	字节		保留
014h	字节		所安装设备的类型
		xx..	驱动器的个数
			00＝一个驱动器

续表

偏 移	长 度	Bits 7654 3210	说 明
014h			11=两个驱动器
		..xx	当前显示类型
			00=自带显示BIOS
			01=40列CGA
			10=80列CGA
			11=MDA
		.... xx..	保留
		.... ..x.	安装了80x87协处理器
		x	安装了软盘驱动器
015h	字		基本内存容量,单位:K
017h	字		扩充内存容量,单位:K
019h	字节		硬盘驱动器0的类型
01Ah	字节		硬盘驱动器1的类型
02Eh	字		10h—20h的检查和
030h	字		扩展内存容量,单位:K
032h	字节		BCD码的世纪值
033h	字节		信息标志
		x...	安装高128K
		.x..	第一个用户信息

速查I ROM信息

地 址	长 度	意 义
F000:E000	5字节	系统启动自检(POST)地址
F000:FFF5	8字节	BIOS生产日期,格式为MM/DD/YY 新型COMPAQ的BIOS生产日期地址 在F000:FFF6
F000:FFFE	1字节	系统机型代码字节
F000:FFFF	1字节	系统机型子代码字节

机型代码表

机型	代码	子代码	修订型号	生产日期
PC	FFh	--	--	--
PC/XT	FEh	--	--	11/08/82
COMPAQ DeskPro	FEh	--	--	--
PCjr	FDh	--	--	--
PC/AT	FCh	--	--	01/10/84
PC/AT	FCh	00h	01h	06/10/85
PC/AT	FCh	01h	00h	11/15/85
PC/XT-286	FCh	02h	00h	--
PS/2 Model 50[Z]	FCh	04h	00h	最初版本
PS/2 Model 60	FCh	05h	00h	最初版本
PS/2 Model 30/286	FCh	09h	--	--
PC/XT	FBh	00h	01h	01/10/86
PC/XT	FBh	00h	02h	05/09/86
PS/2 Model 30	FAh	00h	00h	09/02/86
PS/2 Model 25	FAh	01h	--	09/02/86
PC 便携机	F9h	00h	00h	09/13/85
PS/2 Model 80(16MHz)	F8h	00h	--	最初版本
PS/2 Model 80(20MHz)	F8h	01h	--	最初版本
PS/2 Model 70(20MHz)	F8h	04h	--	--
PS/2 Model 70(16MHz)	F8h	09h	--	--
PS/2 Model 55	F8h	0Ch	--	--
PS/2 Model 70(25MHz)	F8h	0Dh	--	--
HP110 Portyable	B6h	--	--	--
COMPAQ Portable Plus	9Ah	--	--	--
MegaBIOS ROM	4Bh	--	--	--
COMPAQ Portable	2Dh	--	--	--

速查J　显示器标准显示方式速查式

模式号	模式类型	颜色种类	分辨率	起始缓冲区	显示卡				
					MDA	CGA	EGA	VGA	MCGA
0	字符	16	40×25	B800H	X	X	X	X	X
1	字符	16	40×25	B800H	X	X	X	X	X
2	字符	16	80×25	B800H	X	X	X	X	X
3	字符	16	80×25	B800H	X	X	X	X	X
4	图形	4	320×200	B800H	X	X	X	X	X
5	图形	4	320×200	B800H	X	X	X	X	X
6	图形	2	640×200	B800H	X	X	X	X	X
7	字符	单色	80×25	B000H	X	X	X	X	X
8	图形	16	160×200	B000H					X
9	图形	16	320×200	B000H					X
0Ah	图形	4	640×200	B000H					X
0Dh	图形	16	320×200	A000H			X	X	
0Eh	图形	16	640×200	A000H			X	X	
0Fh	图形	单色	640×350	A000H			X	X	
10h	图形	16	640×350	A000H			X	X	
11h	图形	2	640×480	A000H				X	
12h	图形	16	640×480	A000H				X	
13h	图形	256	320×200	A000H				X	

速查 K 显示器字符属性/颜色代码

单色显示器字符属性

Bits 7654 3210	功能	含义
x...	字符闪烁	0:普通字符 1:闪烁字符
.xxx	背景颜色	000:黑色背景 111:白色背
.... x...	字符亮度	0:普通亮度 1:高亮度
.... .xxx	前景颜色	000:黑色字符 001:下划线 111:白色字符

彩色显示器字符属性

Bits 7654 3210	功能	含义
x...	字符闪烁	0:普通字符 1:闪烁字符
.xxx	背景颜色	见颜色代码表
.... xxxx	前景颜色	见颜色代码表

字符颜色代码表

速查L 磁盘分区表

	主引导分区	
1BEH	分区信息1(10H字节)	
1CEH	分区信息2(10H字节)	
1DEH	分区信息3(10H字节)	
1EEH	分区信息4(10H字节)	
1FEH	055h	0AAh

分区信息

偏移	长度	意义
0	字节	分区状态： 0：非活动分区 080h：活动分区(可引导)
1	字节	分区起始头
2	字	分区起始扇区和起始柱
4	字节	分区类型
5	字节	分区终止头
6	字	分区终止扇区和终止柱
8	双字	分区起始绝对扇区
0Ch	双字	分区扇区数

注：起始/终止扇区和柱信息

字节 2/6

```
  7654  3210
  xx..  ....      柱面号高二位
  ..xx  xxxx      扇区号
字节 3/7          柱面号低八位
```

分区类型信息

类型号	分区类型	类型号	分区类型
0	未用	1	12 位 DOS 分区
2	XENIX 分区	4	16 位 DOS 分区
5	DOS 扩展分区	6	BIGDOS 分区
7	HPFS 分区	8	SPLIT 分区
050h	DM 分区	056h	GB 分区
061h	SPEED 分区	063h	386/ix 分区
064h	NOVELL286 分区	065h	NOVELL386 分区
075h	PC/IX 分区	0DBh	CP/M 分区
0FFh	BBT 分区		

速查 M　磁盘目录项格式

偏移	长度(字节)	意义
0	8	文件名
8	3	扩展文件名
0Bh	1	文件属性字节
0Ch	10	保留
16h	2	文件修改时间
18h	2	文件修改日期
1Ah	2	起始簇号
1Ch	4	文件长度

目录项文件名首字节含义

首字节	意义
0	自此目录项以下目录项未使用
05h	文件首字符为 0E5h
2Eh	目录为一子目录,如下一字节也为 2Eh,则其簇号为父目录的簇号
E5h	文件已经删除

速查 N　DOS 内存控制块(MCB)结构表

偏移	长度	意义
0	字节	如 MCB 为最后一块则为 5Ah,否则为 4Dh
1	字	PSP 段地址,如为 0 则该内存块未使用
3	字	当前块的段数
5	11 字节	保留

速查 O　EXE 文件头信息

偏移	长度	意义
00H－01H	字	一般为 'MZ'(04DH,05AH)或 'ZM'(05AH,04DH),EXE 文件标志信息,由 LINK 程序产生
02H－03H	字	文件长度除 512 的余数,即文件长度 MOD512
04H－05H	字	文件长度以 512 字节为一页的页数,即文件长度除 512 的商
06H－07H	字	重定位项的个数
08H－09H	字	文件头的节数,即文件头长度除 16 的商
0AH－0BH	字	程序运行所需最小段数
0CH－0DH	字	程序运行所需最大段数
0EH－0FH	字	堆栈段的偏移值
10H－11H	字	SP 寄存器初始值
12H－13H	字	文件校验和,为 2 的补码
14H－15H	字	IP 寄存器初始值
16H－17H	字	代码段的偏移值
18H－19H	字	文件第一个重定位项的偏移量
1AH－1BH	字	覆盖号,如果程序驻留则为 0
1CH	变化	保留空间
变化	变化	重定位表
变化	变化	保留空间
变化	变化	程序段和数据段
变化	变化	堆栈段

速查 P FCB 结构表

偏移	长度(字节)	意义
−7	1	扩展 FCB 标志(0FFh)
−6	5	保留
−1	1	文件属性字节
标准 FCB		
0	1	驱动器号:0 为当前驱动器,1 为 A,2 为 B...
1	8	文件名
9	3	文件扩展名
0Ch	2	当前块
0Eh	2	文件记录长度,默认值为 128
10h	4	文件长度
14h	2	文件建立日期
16h	2	文件建立时间
18h	8	保留
20h	1	当前记录号
21h	4	随机记录号

速查 Q 文件属性字节

Bits	十六进制	意义
76543210		
.......x	1	只读属性
......x.	2	隐含属性
.....x..	4	系统属性
....x...	8	卷标属性
...x....	10h	目录属性
..x.....	20h	归档属性

速查 R　程序段前缀

偏移	长度	默认值	说明
0	字	CDH,20H	INT 20H 终止地址,COM 程序可用 RET 命令调用
2	字		程序最大可用内存
4	字节	00	保留
5	5 字节		INT 21H 远调用入口
0Ah	双字		INT 22H 中断地址
0Eh	双字		INT 23H 中断地址
12h	双字		INT 24H 中断地址
16h	字		父进程 PSP 段地址
18h	20 字节	01,01,01 00,02,FF FF...	文件句柄索引表,FF 为未用句柄
2Ch	字		环境段地址
2Eh	双字	0,0,0,0	SS:SP
32h	字	14,00	最大文件句柄数
34h	双字		文件句柄表指针
38h	双字		SHARE 程序的 PSP 段地址
50h	字	CDH,21H	INT 21H 调用
52h	字节	CBH	远返回
5Ch	36 字节		未打开的 FCB1
6Ch	20 字节		未打开的 FCB2
80h	字节		命令行参数长度,也是缺省 DTA 的起始地址
81h	127 字节		命令行参数,以字符 0Dh 结束

速查 S 设备驱动程序属性字

位		属性
FEDCBA98	76543210	
.....	1	是控制台输入设备
.....	0	不是控制台输入设备
.....	.....1.	对字符设备：是控制台输出设备 对块设备：支持 32 位扇区数，即支持磁盘分区的扇区数超过 65536
.....	.....0.	对字符设备：不是控制台输出设备 对块设备：不支持 32 位扇区数
.....	....1..	当前设备为 NUL 设备
.....	....0..	当前设备非 NUL 设备
.....	...1...	当前设备为 CLOCK 设备
.....	...0...	当前设备非 CLOCK 设备
.....	..1....	设备支持 INT29H(快速字符输出)
.....000	000.....	在 DOS3.2 前保留
.....	..0.....	保留(3.2 版以后)
.....	.1.....	逻辑设备支持类属 IOCTL(3.2 版以后)
.....	.0.....	逻辑设备不支持类属 IOCTL(3.2 版以后)
.....000	0.....	保留(3.2 版以后)
....1...	.....	支持对设备打开/关闭/介质检查(3.0 版以后)
....0...	.....	不支持对设备打开/关闭/介质检查(3.0 版以后)
...0....	.....	保留
..1.....	.....	字符设备：设备支持输出直到设备忙接口 块设备：设备支持 IBM 格式(BPB)
..0.....	.....	字符设备：设备不支持输出直到设备忙接口 块设备：设备不支持 IBM 格式(BPB)
.1.....	.....	设备支持 IOCTL
.0.....	.....	设备不支持 IOCTL
1.....	.....	字符设备
0.....	.....	块设备