

# 第一章 引言

## § 1.1 关于本书

对一个程序员而言，最令人扫兴的事情，莫过于当其正在聚精会神地编写一段复杂程序时，忽然记不起一个必须使用的系统功能调用的规则和用法了。IBM PC 机的用户遇到此类问题时，必须仔细查阅 IBM PC 机的有关手册，有时需要连续查找多本手册和资料，甚至需要反复考察晦涩难懂的汇编语言源程序，才能获得有关 IBM PC、XT、AT 及兼容机系统功能的基本信息。即便如此，他们可能仍然需要经过反复试验并对结果进行反复比较，才能掌握和运用一个特定的系统功能调用。

本书正是为这些程序员而编写的，书中包括 IBM PC 机的三个基本编程服务：DOS 中断、DOS 功能调用和 BIOS 服务程序。书中对这些系统服务程序进行了完整的描述并使这些描述不仅具有一致性而且还可以迅速查找到有关的内容。

此外，对于愿意阅读程序而不愿阅读文字描述的读者，书中提供了 200 多个使用 DOS 和 BIOS 调用的程序实例。这些实例程序是使用汇编语言、Microsoft C 语言和 Turbo Pascal 语言这三种最流行的语言编写的。这些实例每一个都是独立和完整的程序，旨在说明如何准确使用 DOS 和 BIOS 所提供的服务功能。

编写本书时，我们设想一个程序员正在编写一个大型复杂的程序，他最需要的并不是程序编制的基本原则和一些不重要的细节，而是能够迅速查找到他需要了解的信息。

因此，本书在编排上力求使读者不必翻阅无关的章节就可以迅速找到所需的内容。我们希望本书能成为 IBM PC 机编程人员的一本快速查询手册。

## § 1.2 如何使用本书

正如读者不会逐页阅读一本字典一样，他们或许不会从本书的首页开始一直阅读到最后（除非读者认为必要）。本书编写的目的是为读者提供一本编程参考手册。正确的使用方法应该是仅仅翻阅当前感兴趣的部分，或者把书放在方便的地方，以便以后遇到不清楚的问题时可以迅速查找相关的内容。尽管本书篇幅较多，但仅包含五个章节和一个附录，具体编排结构如下：

第一章 引言。这就是读者目前正在阅读的这一章。该章简介全书内容并说明如何使用本书。

第二章 BIOS 和 DOS 调用：它们的功能和如何调用它们。对于以前编程时几乎从未引用过 BIOS 和 DOS 服务程序的人员来说，第二章是一个入门简介。本章讨论了 IBM PC 计算机不同的编程方法，给出中断向量表的概貌，说明了调用 DOS 和 BIOS 服务程序的一般方法。

第三章 DOS 中断调用。这一章详细讨论了所有的 DOS 中断（除中断 21H 将在第四章讨论外）。引言部分还讨论了程序段前缀（PSP）的有关问题。

**第四章 DOS 功能调用.** 这一章详细讨论了 DOS 功能调用, DOS 提供的这个编程服务是通过中断 21H 调用的. 引言部分介绍了 DOS2.0 和 DOS1.X 文件管理技术的差异, 还讨论了文件控制块 (FCB) 和标准错误代码.

**第五章 BIOS 中断和功能调用.** 这一章详细讨论了 BIOS 所提供的编程服务, 这些服务由驻留在 IBM PC 机的只读存储器 (ROM) 中的程序所提供. 此外, 这章的引言部分还讨论了 BIOS 数据区.

**附录 编程服务参照表.** 这个附录列出本书提供的全部编程服务的参照表. 内容是按功能而不是按功能调用号排列的. 所以, 读者如果完成某种功能 (如显示器 I/O 调用), 但又无法肯定 DOS 或 BIOS 是否提供这种功能时, 可查阅这个附录.

因为本书是一本参考手册, 所以没有必要从第一页阅读. 如果仅仅准备了解 BIOS 调用而不关心 DOS 提供的功能, 可以略过前四章, 直接阅读第五章. 本书内容编排的特点之一是: 在讨论某个功能调用的那一节中, 将包含有关此调用的全部信息.

为了达到这个目的, 有些章节在内容上可能有些重复和交叉, 某些问题可能在不同的地方均进行了讨论, 这样可避免为了解决一个问题而要查找许多章节, 当需要快速查找某些信息时, 在一处便可查到所需的全部信息. 因为这个特点, 如果偶然逐页阅读本书时, 读者也许会感到枯燥乏味.

本书的核心章节 (第三章至第五章) 在内容编排上, 还注意为读者查找任何有关功能调用及中断的信息提供方便.

首先, 功能调用或中断的名字及其功能调用号总是安排在顶角处. 这样, 如需查找某个功能调用及中断, 只要翻看书的顶角处便可找到. 编排次序是按中断号的大小顺序排列的, 如果一个中断可以提供多种服务, 这些服务按功能调用号大小顺序排列.

此外, 每个功能调用和中断的说明都采用统一的标准格式, 有概要说明、输入参数、输出参数三个部分. 输入参数和输出参数的说明简单清晰, 一目了然. 如果有其它要求或出错情况, 也都安排在标准位置处. 一旦了解了一个中断或功能调用的内容是如何安排的, 就会确切知道应该到哪儿查阅所需的任何中断或功能调用的相应内容.

## 第二章 BIOS 和 DOS 调用 ——作用和访问方式

### § 2.1 引言

BIOS 和 DOS 是两组系统服务软件的集合，它们使程序能够访问和使用构成 IBM PC 机的硬件。其中包括从键盘读取字符、在显示器上显示信息、读写磁盘、主机向打印机传递信息和许多其它服务。在未详细阐述 BIOS 和 DOS 可以提供什么服务之前先看一看为什么要使用它们以及如何使用它们。

一般来说，程序可以通过四种方式控制 PC 机的硬件（如图 2-1 所示），即直接访问硬件，使用 BIOS 提供的功能，使用 DOS 提供的功能和使用高级语言提供的功能。这四种方式在编程的复杂性和程序的可移植性方面各有利弊。

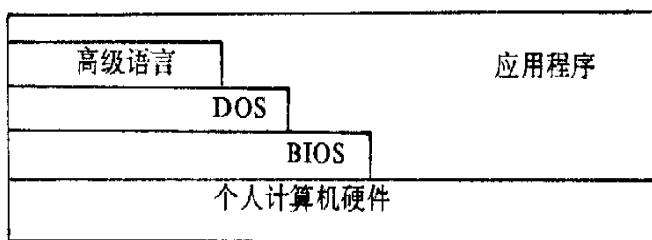


图 2-1 程序控制 PC 机硬件的四种方式

### § 2.2 直接访问硬件

使用 IBM PC 机的最基本方式是利用硬件本身的能力。例如，微处理器（8088 或 80286）通过 I/O 端口与磁盘驱动器、键盘和打印机等其它设备通讯就是如此。I/O 端口是 8 位通道，通过它可以发送和接收数据。每种设备都可以被分配一个或多个 I/O 端口地址，这样每个设备都可单独和微机处理器通讯。一个程序可以通过执行 OUT 指令从 I/O 端口向外发送信息，也可以通过执行 IN 指令从 I/O 端口接收外部信息。

一个使用 IN 和 OUT 指令的程序肯定可以充分利用设备的能力。程序可以通过外设的 I/O 端口向外发送指令，使外设完成指定的操作。然而，这种程序的编写相当繁杂，程序员必须对设备非常熟悉。例如，当访问软盘驱动器时，需要接通驱动电机、移动磁头到指定位置，对一个扇区进行读写等操作，每个操作至少需要一条复杂的设备驱动指令。

一般来说，如果不是为了得到更高的执行效率和获得 DOS 和 BIOS 不支持的功能，程序不应直接和硬件打交道，直接访问硬件的程序可移植性相当差。在一个厂商生产的机器上调试的程序，在其它厂商生产的兼容机上可能根本无法运行，甚至在同一厂家生产的不同型号的机器上也无法运行。例如在早期的 PC 机上开发的程序可能在 AT 机上根本无

法运行。

### § 2.3 使用 BIOS

BIOS 是一组低级软件程序，这组程序是计算机硬件和其它程序之间的一个缓冲。BIOS 程序本身直接和外设通讯，它为编程人员提供一个简单的接口，避免了和外设打交道而必须编制复杂程序。当计算机加电时，BIOS 服务程序对计算机进行测试以保证硬件工作正常，同时将 DOS 系统从盘上装入内存。

由于 BIOS 在程序和硬件之间形成缓冲，IBM 公司可以引进先进的硬件技术而保持用户软件兼容。这样，即使 PC 机和 AT 机端口地址和外设控制指令不同，一个在 PC 机上运行的程序同样可以在 AT 机上运行。为了在硬件不同时保持软件的兼容性，BIOS 程序必须做相应改动，以保证编程人员看到的接口完全一样。这样，用户程序就不必做任何改动了。

使用 BIOS 而不直接访问硬件将使程序更加简炼。如果用户程序包括大量控制外设的代码，程序所占空间可能超过机器的接收能力。但是，无论是否使用 BIOS，它总是驻留在机器中，仅用几条指令就可以调用用户需要的 BIOS 程序。

BIOS 驻留在系统板上的只读存储器 (ROM) 中，计算机加电后，可以随时调用 BIOS 程序。因为 BIOS 提供基本的低层服务，所以调用 BIOS 而不是相应的 DOS 程序可以提高程序的执行效率。

IBM PC 机的 BIOS 已经获得软件版权，这意味着除非得到 IBM 公司的允许，其它厂商在制造自己的计算机时完全复制 IBM 公司的 BIOS 程序代码将是非法的。这样，即使生产兼容机的厂商标榜自己的机器与 IBM PC 兼容，一个直调用 BIOS 语句的程序在兼容机上仍有可能无法运行。

尽管存在在兼容机上无法运行的风险，许多商业上取得成功的软件仍然采用 BIOS 功能调用。这主要是为了得到较高的运行效率，或者需要利用 DOS 不具备的某些功能。这些软件最常用的是 BIOS 提供的显示器控制程序，这些 BIOS 程序允许用户程序改变显示器的显示方式、设置光标和使用其它的 DOS 无法提供的功能。

当程序员编程时，他必须权衡为了使用 BIOS 而获得的较高运行效率和能力，从而冒与其它计算机或运行环境（如 Top View 或 Windows）不兼容的风险是否值得。对用户来说，除非运行效率或使用 DOS 无法提供的功能是至关重要的，否则就应该使用 DOS 调用而不是 BIOS 调用。

### § 2.4 使用 DOS

DOS 是 PC 机上应用最广泛的操作系统。除了能执行从键盘上输入的合法命令外，DOS 还为编程人员提供了丰富的服务程序。

DOS 在更高的层次上提供了与 BIOS 同样的功能。例如，既可以通过调用 BIOS，也可以通过调用 DOS 实现磁盘读写。调用 BIOS 时，必须准确说明读写位置（磁头、磁道和扇区号），才能正确读写信息。但是，在 DOS 系统中已经有文件、目录、子目录等概

念，因此调用 DOS 程序时，不必指出读写信息在盘上的物理位置，只需打开文件，在文件内找到指定位置，就可以进行读写操作。

一般来说，使用 DOS 提供的功能比使用 BIOS 提供的功能更容易。因为使用 DOS 提供的功能时，编程人员不需要对硬件有更多的了解。通过调用 DOS，还可以充分利用操作系统提供的所有功能，如分级文件系统、装载和执行程序、分配内存等。但是，并非 BIOS 的所有功能都能通过调用 DOS 来完成。由于在编写 DOS 时，对可能调用它的情况做了某些假设，因此，有些 BIOS 所提供的功能无法用 DOS 实现。

大多数情况下，并不需要利用 BIOS 调用直接访问硬件，因为用户希望外设工作的方式恰恰是 DOS 调用所提供的方式。磁盘的 I/O 操作就是一个很好的例子。大多数程序进行 I/O 操作时，都希望以文件方式读写数据。如果不使用 DOS 而使用 BIOS 调用进行文件读写，既复杂又容易出错。虽然可以将信息写在指定的位置，但往磁盘上写之前，必须对盘区格式信息非常清楚，才能既保证不覆盖已存在的文件，同时也防止刚刚写入的文件以后被别的程序覆盖。使用 DOS 时，这些工作都由 DOS 系统自动完成。

调用 DOS 功能除接口简单外，程序的可移植性也比使用 BIOS 好，仅仅使用 DOS 功能的程序可以在任何支持 PC-DOS 或 MS-DOS 的计算机上运行。因为许多兼容机厂商生产的 PC 机和 IBM PC 并不是百分之百兼容。如果开发的程序准备作为商品出售，可移植性将十分重要。

但是 DOS 程序的执行效率比 BIOS 低，提供的功能也没有 BIOS 丰富，显示器的 I/O 操作就是一例。DOS 仅仅提供简单的字符和字符串输出能力，如果使用图形方式、设置字符的属性和颜色，或者需要提高输出显示的速度，就必须使用 BIOS 或者直接访问硬件。

在可能的情况下，尽量使用 DOS 而不是 BIOS，这样程序将具有良好的可移植性。由于 DOS 对复杂操作提供了一个简单接口，程序既容易编写，又方便调试。

## § 2.5 使用高级语言提供的语句

如果用 Pascal 或 C 等高级语言编程，则可以充分利用嵌入在高级语言中的许多操作系统的功能。例如，可以直接使用高级语言提供的语句从键盘接收信息，向显示器写数据和读写磁盘文件。如果高级语言提供的功能完全能满足编程的需要，就没有必要直接使用 DOS 和 BIOS 功能调用。这样程序将具有良好的可移植性（只要把源程序在有编译程序的机器上重新编译一次即可），而且接口比较简单。

但是高级语言提供的 I/O 功能比 DOS 要少，所以有些操作仅仅使用高级语言提供的语句将无法完成。此外，使用高级语言做 I/O 操作将明显增加程序代码长度，例如，即使一个仅使用 REAKLN 和 WRITELN 语句从键盘读入和往显示显示器写出信息的简单 Pascal 程序，经过编译和连接后产生的程序代码也比调用 DOS 功能从键盘输入和显示器 I/O 操作的程序大 20KB 左右。

## § 2.6 应该使用哪种方法

泛泛地谈论应该使用哪种方法是没有意义的。应该根据编程人员的素质和所编程序的使用要求来考虑这个问题。一个程序员需权衡程序的可移植性、编程的复杂性和目标代码的大小后做出决定。

如果决定使用 BIOS 或 DOS 编程，本书将是一个十分有用的工具。书中详细讨论了 DOS 和 BIOS 功能调用的有关问题，并且对每个 DOS 功能调用都给出了三种语言编写的程序实例。本书的绝大部分内容与 IBM 和 Microsoft 的技术手册完全一致，但是，我们把分散在各种手册中的信息综合在一起，并且补充了实际使用中的大量实例和体会。在内容安排上尽量做到信息编排合理，以便读者能迅速、方便地查找有关信息。

利用高级语言提供的语句编程不在本书讨论的范围内，应该查阅有关高级语言的编译参考手册的书籍。但是在实际编程中，如果读者发现某些功能仅通过高级语言无法实现时，可以浏览本书，看看 DOS 和 BIOS 提供的功能调用是否有助于解决这些问题，最好按先 DOS 后 BIOS 的顺序查阅。

如果 DOS 和 BIOS 提供的功能调用还无法满足编程需要，就只好直接访问硬件。这种情况也不属于本书讨论的范围，应查阅《IBM 技术参考手册》。如果需要更详细的资料，可能还要查阅有关的硬件参考手册。但是在决定直接访问硬件之前，应先从 DOS 功能调用开始，认真查阅本书，也许会找到解决问题的办法。

## § 2.7 调用 BIOS 和 DOS

如果一个程序是由主程序和若干子程序组成，主程序可以通过调用语句调用子程序。但是，如果子程序是分模块编译的，则要把包含各个子程序的目标文件和主程序连接在一起，最后形成一个包含主程序和所有子程序的可执行文件。

BIOS 和 DOS 服务程序的作用与子程序类似，但实现调用时不能用它们的名字，主程序也不必与 BIOS 或 DOS 程序代码连接（见图 2-2），所以形成的代码比较紧凑。另外，调用它们的程序并不需要知道 BIOS 和 DOS 代码实际存储空间的地址。

编程时，调用 BIOS 和 DOS 程序的方法是采用处理器中断。中断是停止 CPU 目前正在处理的工作，而非运行其它程序的信号。中断可以由微处理器本身产生（例如，当试图用 0 作除数时），或者由硬件产生（键盘、磁盘驱动器、系统时钟都可产生中断），也可以由软件产生（当程序执行一条 INT 指令时，就会产生一个中断）。

无论什么时候，产生中断时处理器将做下列工作：

1.保护现场：CPU 将段寄存器（CS）、指令指示器（IP）和标志寄存器压入堆栈保护起来，以便中断处理程序运行结束后恢复中断前的现场，保证程序继续正确执行。

2.关中断：如果中断源是硬件，CPU 将清除中断标志位（IF）。这时，CPU 在未结束当前的中断服务程序前，对其它的中断请求信号不予响应。中断服务程序完成后，通过重新设置中断标志位，交出中断控制权。

3.运行中断程序：处理器将控制权交给中断服务程序。该服务程序的地址在中断向量

表中。向量表包含 256 个中断向量，首地址是 RAM 的 0H 字节。开机时，CPU 负责填写这些单元的内容（当 80286 处理器工作在保护方式时，情况则不同，包括中断向量表的地址都将发生变化，但 CPU 工作在保护方式的情况不在本书讨论的范围内）。

中断向量表中每项（又称中断向量）由四个字节组成，高两位字节是每个中断服务程序的段地址（CS），低两位字节是段内偏移量（逻辑地址）。其中段地址是 16 位，实际上代表 20 位的地址码，不过最低 4 位的值始终是“0”而已，这个段地址指向一个段的首地址。偏移量表示相对于段地址偏移的字节数。由于中断表中共有 256 个中断向量，每个中断向量占四个字节，整个中断向量占用 1KB 的空间，从内存的 0H 单元到 3FFH 单元。

产生中断时，根据中断性质和数值选择相应的中断向量。例如硬件中断来自连接 8259 中断控制器的信号线，这些信号线自动映射为中断向量表的特定项。软件中断是通过中断调用号选择向量表的特定项，如果程序中有 INT20H 指令，CPU 将调用地址在中断向量表 20H 处的中断服务程序。

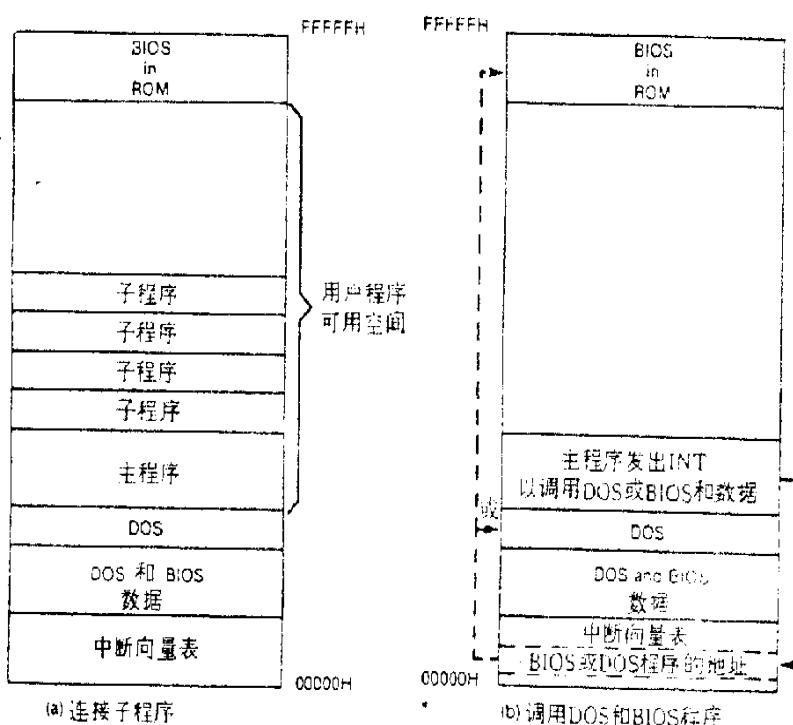


图 2-2 主程序与子程序连接完成某些 OS 服务

图 2-2 通过主程序与特定子程序连接完成某些操作系统的服务将增加程序代码的空间。但是 DOS 和 BIOS 代码已经存储在内存中，用户程序无需把它们再复制到自己的程

序中，所以调用 DOS 或 BIOS 子程序不会增加用户程序空间。

当地址在中断向量表中的程序取得控制权时（这些程序有时称作中断处理程序），通常立即执行相应的操作，或者一旦开始运行即迅速打开中断，以便必要时 CPU 能够响应更高级的中断。

中断服务程序完成以后，执行中断返回指令 IRET，该指令恢复 CS、IP 及标志寄存器，CPU 继续执行原来被中断的程序。

计算机加电时，中断关闭，CPU 执行驻留在 ROM 中的 BIOS 程序。BIOS 程序做的第一件事是填写中断向量表，把 BIOS 中断服务程序的地址写入中断向量表中。这些中断包括处理器中断（如被 0 除）、硬件中断（硬盘、键盘和时钟中断）以及 BIOS 本身特定软件中断（包括本书所讲座的 BIOS 功能调用）。一旦这些中断发生，便立即转入相应的中断处理程序。

BIOS 仅仅负责填写中断向量表的部分单元。当 BIOS 运行完启动程序（开机、自测程序等）后，将 DOS 程序装入内存。DOS 程序做的第一件事是在中断向量表的剩余单元中填入 DOS 中断处理程序的人口地址。这样，填写中断向量表的工作就完成了。但仍保留一些未用的中断向量（从 78H 到 7FH），留给编程人员建立自己的中断处理程序（Interrupt handler——又称中断句柄）。

BIOS 提供的某些中断的服务程序是计算机运行的最基本软件，如果没有这些软件，计算机根本无法运行。但是，绝大多数的 BIOS 和 DOS 中断向量是指向 BIOS 和 DOS 常用软件服务程序的人口地址。通过中断向量表，就可以调用这些服务程序，而不必知道它们的实际驻留地址，也不必把它们与用户程序连接，这样就减少了用户程序的代码长度。调用中断服务程序时，首先在某些寄存器中填写相应数值，然后执行一条 INT n 指令，其中 n 是中断号。当调用的中断处理程序（BIOS 或 DOS 功能调用）运行结束后，说明运行情况的结果存放在某些寄存器中，并交回控制权。

表 2.1 列出了 BIOS 和 DOS 建立的中断向量，这些中断的绝大部分将在以下三章中详细讨论。许多中断调用具有多重功能（例如，全部 DOS 功能调用都是通过中断 21H 实现的）。

绝对地址	中断向量		说 明
	16 进制	10 进制	
00H	0H	0	0 做除数时处理器发出的中断
04H	1H	1	单步调试时处理器发出的中断
08H	2H	2	非屏蔽中断
0CH	3H	3	调试程序设置断点时处理器发出的中断
10H	4H	4	发生算术溢出时处理器发出的中断
14H	5H	5	调用 BIOS 的屏幕拷贝操作
18-1FH	6-7H	6-7	保留单元
20H	8H	8	每 1 / 18.2 秒定时器发出的中断
24H	9H	9	按压或释放键盘时产生的中断
28H	0AH	10	保留单元

绝对地址	中断向量		说 明
	16 进制	10 进制	
2CH	0BH	11	通讯设备使用的硬件中断
30H	0CH	12	通讯设备使用的硬件中断
34H	0DH	13	交替打印时硬件产生的中断
38H	0EH	14	软盘驱动器操作结束时产生的硬件中断
3CH	0FH	15	打印机发出警告信号时产生的硬件中断
40H	10H	16	BIOS 的显示 I/O 功能调用
44H	11H	17	BIOS 设备确认调用
48H	12H	18	BIOS 确定内存空间大小的功能调用
4CH	13H	19	BIOS 的磁盘 I/O 功能调用
50H	14H	20	BIOS 的 RS-232 串行 I/O 功能调用
54H	15H	21	在 PC 和 XT 机上是 BIOS 磁带 I/O 功能调用。在 AT 机上，是 AT 扩充服务功能调用。
58H	16H	22	BIOS 的键盘 I/O 功能调用
5CH	17H	23	BIOS 的打印机 I/O 功能调用
60H	18H	24	ROM 的 BASIC 解释程序功能调用
64H	19H	25	BIOS 的装载引导服务功能调用
68H	1AH	26	BIOS 的日期时钟功能调用
6CH	1BH	27	Ctrl-Break 处理程序功能调用当键入 Ctrl-Break 键时指向可执行的程序入口，初始化时 BIOS 使此向量指向一条 IRET 指令。用户可修改此向量指向自己的程序。
70H	1CH	28	指向每 1 / 18.2 秒时可执行的服务程序的入口，初始化时此向量指向一条 IRET 指令。用户可修改此向量，使其指向自己的 Ctrl-Break 处理程序。
74H	1DH	29	指向显示控制器初始化参数表。BIOS 使这个向量指向 ROM 驻留表。

绝对地址	中断向量		说 明
	16 进制	10 进制	
78H	1EH	30	指向软盘参数表。BIOS 使这个向量指向 ROM 驻留表，但是 DOS 把它改为指向 DOS 的 RAM 驻留表。
7CH	1FH	31	指向一点阵表。在这个表中，BIOS 可以找到字符集后 128 个字符的点阵(前 128 个字符的点阵在 ROM BIOS 中)。仅在图形方式下才使用这些点阵。在字符方式下，显示适配器的字符发生器产生所有 256 个字符，BIOS 初始化这个向量为 0：0。因此，在图形方式下，用户必须建立一个表，并根据显示后 128 个字符来修改这个向量。
80H	20H	32	终止程序的 DOS 功能调用。
84H	21H	33	任何种 DOS 功能调用
88H	22H	34	指向 DOS 的结束地址
8CH	23H	35	指向 DOS 的 Ctrl-Break 处理程序
90H	24H	36	指向 DOS 的严重错误处理程序
94H	25H	37	DOS 绝对磁盘读调用
98H	26H	38	DOS 绝对磁盘写调用
9CH	27H	39	程序终止，但仍驻留在内存的 DOS 功能调用
A0~FFH	38~3FH	40~63	为 DOS 保留的单元
100H	40H	64	保留单元
104H	41H	65	指向硬盘 0 参数表，BIOS 使这个向量指向 ROM 驻留表。
108~10FH	42~43H	66~67	保留单元
110H	44H	68	PCjr 机使用，用于指向低分辨率图形字符参数表。
114H	45	69	保留单元

绝对地址	中断向量		说 明
	16进制	10进制	
118H	46H	70	指向硬盘1的参数表, BIOS使这个向量指向ROM驻留表。
11CH	47H	71	保留单元
120H	48H	72	PCjr机使用, 用于把PCjr的键盘代码变换为标准的键盘代码。
124H	49H	73	指向键盘增强服务变换表
128-17FH	4A-5FH	74-95	保留单元
180-19FH	60-67H	96-103	为用户程序保留的单元
1A0-1BFH	68-6FH	104-111	未使用
1G0H	70H	112	硬件中断(IRQ-interrupt request) 8——实时钟中断
1C4H	71H	113	硬件中断9
1C8H	72H	114	硬件中断10
1CCH	73H	115	硬件中断11
1D0H	74H	116	硬件中断12
1D4H	75H	117	硬件中断13——BIOS把这个中断向量重定向为非屏蔽中断(NMI)
1D8H	76H	118	硬件中断14
1DCH	77H	119	硬件中断15
1E0-1FFH	78-7FH	120-127	未使用
200-217H	80-85H	128-133	为BASIC保留
218-3C3H	86-F0H	134-240	BASIC程序运行时, 提供给BASIC解释程序使用
3C4-3FFH	F1-FFH	241-255	未使用

表 2.1 中断向量表

## § 2.8 用户程序如何调用 BIOS 和 DOS 服务程序

后续三章将分别讨论用户程序可调用的 DOS 中断、DOS 功能调用和 BIOS 功能调用。在每章详细讨论前，都有一节阐述在调用 DOS 和 BIOS 服务程序前需做的工作。

## 第三章 DOS 中断

### § 3.1 引言

本章和下一章将详细讨论在 PC 机上运行的 DOS 所提供的服务程序。因为调用它们的方法不同，所以分为两章讨论。

除中断 2FH 外，这一章的中断服务程序都是通过中断号调用的，而中断 2FH 是多功能中断调用。

第四章的服务程序则都是通过同一个中断（中断 21H）调用的，根据在 AH 寄存器中设置不同的数值，选择不同的服务程序。

除了调用 DOS 服务程序的方法不同以外，第四章和第三章所讨论的 DOS 中断调用几乎没有什么不同。那么，为什么这些中断服务程序中有些有自己的中断调用号，而另外一些则都采用中断 21H 呢？这主要是由于 IBM 公司试图与以前的系统（特别是 CP / M 操作系统）保持兼容的缘故。

这一点今天看起来可能令人难以置信，但在 IBM 公司刚刚开始生产 PC 机时，不可能预料到它会成为世界上使用最广泛的计算机系统。当时，DOS 操作系统刚刚推出，能在 DOS 环境下运行的应用软件还相当少。因此，DOS 的设计者必须保证它们某些功能调用与在 CP / M 中的相同。只有做到这点，用户在 CP / M 操作系统下编制的程序才能很容易地移植到 DOS 系统中。现在，DOS 成为个人计算机的主流操作系统，只有在 DOS 环境下能够运行的软件才可能有良好的市场销售前景。当时，DOS 的设计者根本没有预料到这一点，所以他们使 DOS 与 CP / M 操作系统保持部分兼容的设计思想至今仍有影响。

### § 3.2 如何使用本章

本章详细讨论了每个 DOS 中断调用。为了使读者能迅速查找到所需资料，每个中断都从新的一页开始。如果不准备自讨苦吃，就没有必要逐页阅读本章内容。当需要查阅有关的中断调用时，可以一边翻书，一边注意每页上顶角处的中断调用的编号和名字，就象查阅字典和电话号码簿一样。这样，可以迅速找到感兴趣的内容。

每个中断的内容安排都按照统一的形式，这样可以在每个中断调用的相同位置处找到同样的内容。每个 DOS 中断调用的说明中都包括支持的 DOS 版本号。如果不同的 DOS 版本执行某一功能时有差异，则予以说明。

在没有讨论 DOS 中断本身以前，本章将介绍某些重要的基础知识。首先，将介绍程序段前缀（PSP）。程序段前缀是 DOS 为在其下运行的每个程序建立的控制块，了解和掌握 PSP 对任何使用 DOS 中断（或者是第四章讨论的 DOS 功能调用）的编程人员都非常重要。在讨论 PSP 以后，还将解释如何从用户程序中调用 DOS 中断。

### § 3.3 程序段前缀 (PSP)

当 DOS 运行一个程序时，首先为暂驻程序留出足够的空间，该空间称为程序段，用于存放从磁盘装入的程序。在 DOS 装载程序前，首先在程序段的前 256 个字节建立一个

字节偏移量

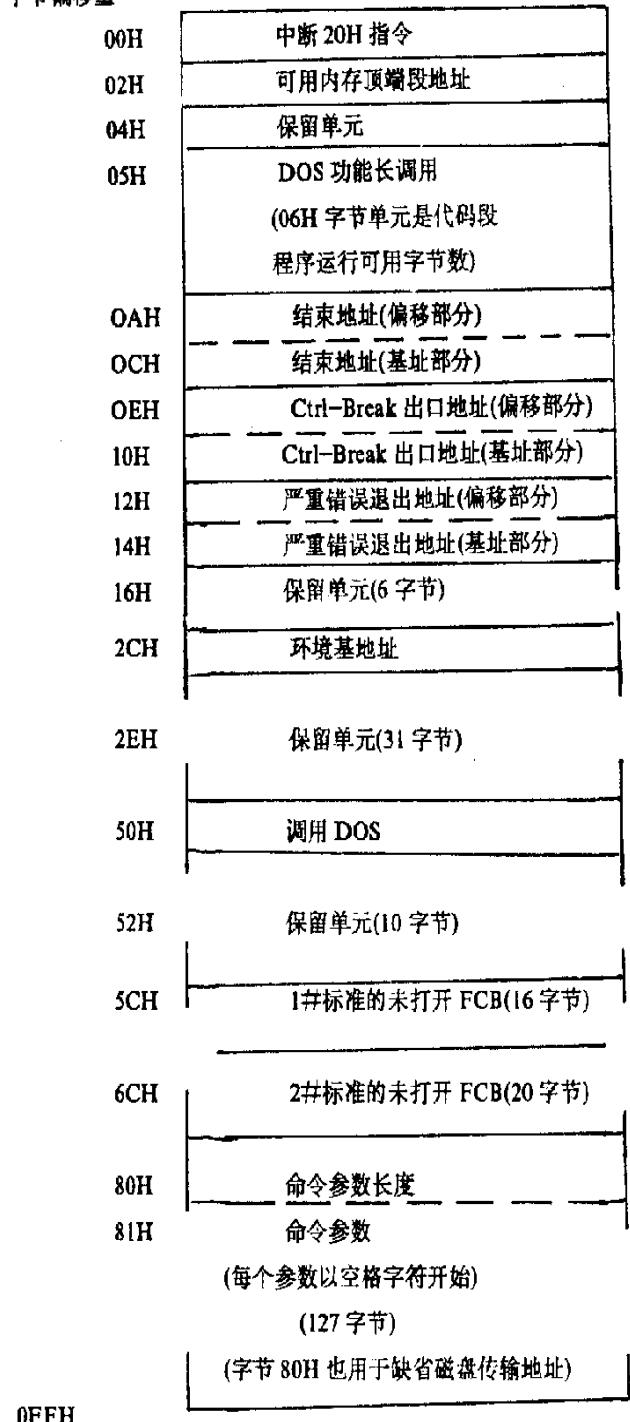


图 3-1 程序段前缀结构图

控制块——程序段前缀 (PSP)。程序段前缀包含执行程序的各种信息。这些信息可分为两部分：一部分是装入程序的有关信息（如命令行参数和内存大小等）；另一部分是程序调入前 DOS 有关的环境信息。图 3-1 是程序段前缀的结构图。下面将分别讨论每个区域的内容。

偏移量	说 明
00H-01H	<b>中断 20H 指令</b> 这两个字节的内存是中断 20H 指令（机器码 0CD20H）。正如本章后面将提到的，中断 20H 是中断当前程序的 DOS 中断。PSP 中这条指令的作用是通过转移到程序段 0 单元结束程序的运行。然而，结束程序的更好的方法是调用 DOS 功能请求 00H、31H 或 4BH，这将在第四章进行讨论。
02H-03H	<b>可用内存顶端段地址</b> 这个基地址表示程序可用的最大内存，以 16 字节的段形式表示。例如，4000H 表示 4000H 个 16 字节的段，或 4000H 字节 (256K)。这时，程序最大文章地址可达 256K，这意味着计算机仅有 256K 字节内存，或者 RAM 驻留程序（如假脱机程序、RAM 盘管理程序的实用程序）使用更高的地址。
05H-09H	<b>DOS 功能的长调用</b> 这五个字节包含对 DOS 功能的长调用，这个调用通过中断 21H 实现（第四章将要详细讨论）。程序应当使用中断 21H 与 DOS 功能调用的接口，而不应该直接转移到这些单元。06 字节单元是代码段中程序运行可用的字节数，这对于程序的运行相当重要。
0AH-0DH	<b>结束地址</b> 这 4 个字节是程序退出返回的段地址。如果程序是由命令行调用的，该地址则是 DOS 命令处理程序地址。有关结束地址的更详细内容，请参看本章有关 DOS 中断 22H 的讨论。
0EH-011H	<b>Ctrl-Break 出口地址</b> 这 4 个字节是程序运行时 Ctrl-Break 处理程序的段地址。当程序运行时，如果按下 Ctrl-Break（或者 Ctrl-C）键，Ctrl-Break 处理程序就开始运行。通过调用 DOS 中断 23H 把一个新的地址写入中断向量表的相应位置，可以改变 Ctrl-Break 中断处理程序。有关中断 23H 的调用，本章还要详细讨论。程序退出时，DOS 把 Ctrl-Break 处理程序的有关信息从 PSP 复制至中断向量表，从而恢复 Ctrl-Break 处理程序以前的状态。
12H-15H	<b>严重错误退出地址</b> 这 4 个字节是程序运行时严重错误处理程序，但是当程序退出时，DOS 将用 PSP 中的地址恢复原来的严重错误处理程序。
2CH-2DH	<b>环境参数地址</b> 这 2 个字节适用于 DOS2.0 以上版本。其中包含传送给用户程序的环境参数的基本（或段）地址。环境参数是从本段的 0 偏移地址开始。环境包含一组字符串说明与程序有关的配置信息字符串的形式是： 参数=数值 字符串是 ASCII 串，每个字符中以一个字节的 0 结尾。例如，一个字符串可以是 VERIFY=ON。从键盘键入 DOS 的 SET 指令可以改变目前的环境参数。一个传递程序的环境参数至少应有通过 COMSPEC= 字符中指明 DOS 寻找 COMMAND.COM 的路径。如果程序装入或调用的程序，有关更详细的内容可参阅第四章 DOS 功能调用 4BH。

<b>50H-51H</b>	<b>调用 DOS</b> 这 2 个字节的内容适用于 DOS2.1 以上的版本。它们包含 DOS 功能调用处理程序，并提供了请求 DOS 功能调用的其它方法。虽然 PSP 的这部分是 DOS2.1 以后版本新增加的，但是，并不提倡通过转入这个单元调用 DOS 功能，应该通过第四章将要讨论的中断 21H。
<b>5CH-6BH</b>	<b>1#标准的未打开 PCB</b>
<b>5CH-7FH</b>	<b>2#标准的未打开 PCB</b> 有些程序是通过文件名参数调用的，这两个域为这些程序访问自己的文件提供了一种途径。然而，这个机制使用的是传统（或者是 DOS1.X）的文件访问方法。正如第四章将要详细讨论的，DOS1.X 方法并不支持树形目录结构，所以对 DOS2.0 以上的系统，这种机制便失去作用。 这两个域是一个未打开文件的控制块（FCB）。DOS1.X 程序访问时，必须用 FCB。FCB 的结构及详细信息请参见 60 页。 调用程序需传递有关参数时，DOS 分析前两个参数，并把它认为是文件名，同时在 PSP 中建立这两个域作为控制块，因为 PSP 中这两域的空间无法存贮一个打开文件的 FCB（仅存贮驱动器名、文件名、扩展名），用户程序在打开文件前应把 FCB 拷贝到一个足够大的区域中。
<b>80H</b>	<b>命令参数长度</b> 如果通过参数调用程序，DOS 将从 PSP 的偏移量 81H 处开始放置这些参数，同时在偏移量 80H 的字节放置参数表所占的字节数（从 0 至 127）
<b>81H-0FFH</b>	<b>命令参数</b> DOS 把与程序有关的所有参数设置在这 127 字节的区域。如果程序是通过键盘指令调用，这些参数将不包括调用程序的命令名。它们是以命令名后的字符开始，通常紧接的是 ASCII 空格。设置这些参数时，并不移去或压缩其中的空格、逗号等分隔符。在 DOS2.0 以后的版本的系统中，如果命令行中包括重定向（> 输出文件名）参数，DOS 将修改有关参数行，删除它们。字节 80H 用于缺省的磁盘传输地址（DTA），DTA 是 DOS 用于读盘和写盘的内存缓冲起始单元地址，PSP 的偏移地址 0080H 到 00FFH 的 128 个字节可用于磁盘缓冲区。磁盘传送地址的详细内容可参阅第四章。 因为缺省磁盘传输地址覆盖命令参数，用户程序在执行 I/O 前，应从 PSP 中取出有关参数，另外建立一个 PSP；或者避免使用 DTA 的 DOS 功能调用。有关改变 DTA 的方法，请看 DOS 功能调用 IAH。

如前所述，DOS 装入用户程序时，在程序段偏移量为 0 地址处建立 PSP，并把用户程序装入 PSP 后的地址单元。程序开始运行后，某些寄存器将指向 PSP。

如果用户程序是.COM 类程序，DOS 将把这个程序装入程序段偏移量 100H 处，并把所有的段寄存器（CS, DS, SS 和 ES）指向 PSP 的起始地址。所以，用户程序可以很方便地访问 PSP。

如果用户程序是.EXE 类程序，程序的定位和段寄存器值的设置就稍微复杂一些。一个.EXE 程序含有由连接程序建立的程序重定位表，装入程序可以根据这个表确定程序的某些部分的定位地址。因此，可能某些程序的起始地址并不在偏移量 100H 处，但是，当用户程序开始运行时，ES 和 DS 寄存器总是指向 PSP 的起始地址。因此，用户程序仍然

可以访问 PSP 的有关信息。

### § 3.4 调用 DOS 中断

用户程序不能调用所有的 DOS 中断提供的功能。某些功能只有在发生特定的外部事件时才起作用（例如用户同时按 Ctrl-Break 键，或者是程序发生严重错误时）。然而，某些中断确实提供了十分有用的程序服务。

为了调用这些服务程序，首先必须给某些寄存器设置数值。下面将详细讨论调用时应该使用哪些寄存器以及在这些寄存器中应设置的数值。

寄存器设置后，通过执行 INT n (n 是中断调用的编号) 指令，就可以调用相应的中断处理程序。

有些中断处理程序会返回有关信息，当中断处理程序运行结束后，可以从寄存器或数据区取出这些信息。在每个中断的详细讨论中将列出中断所返回的信息。

下面是汇编语言调用 DOS 中断的例子。程序仅通过 DOS 功能调用 09H (第四章将详细讨论) 在显示器上显示信息的调用 DOS 中断 20H 终止程序。

```
; ;PROGRAM TERMINATE (INTERRUPT20H)
;
code segment public
assume cs:code,ds:code
org 100h
start:jmp bbegin
msg db 'Terminating via Interrupt 20h'
/
begin:mov ax,cs    ; 置 ds
      mov dx,ax
      mov dx,offset msg ; 传送信息地址
      mov ah,09h          ; 显示字串功能调用
      int 21h             ; DOS 调用
      int 20h             ; 结束程序
code ends
end start           ; start 为入口
```

DOS 中断完成的是低级操作，因此 C 或 Pascal 等高级语言几乎无法调用这些中断处理程序。

## § 3.5 DOS 中断目录表

表 3.1 列出了所有 DOS 中断，在本章中能找到有关它们的详细信息。

中 断 号		功 能
16 进制	10 进制	
20H	32	终止程序
21H	33	调用 DOS 功能调用. 见第四章
22H	34	终止地址
23H	35	Ctrl-Break 地址
24H	36	严重错误处理程序地址
25H	37	绝对读盘
26H	38	绝对写盘
27H	39	终止但驻留内存
28H-2FH	40-64	专供 DOS 使用

表 3.1 DOS 中断表

### DOS 中断 20H

终止程序

中断:

20H 终止程序

DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明:

此功能调用与 DOS 功能调用 00H (在第四章中讲述完成的操作相同)。它使当前的进程终止，并将控制权返回到它的父进程。通常，这个恢复控制权的程序是 DOS 命令处理程序 (COMMAND.COM)。但是，如果这个程序是由其它的程序启动的，那么该程序将恢复控制。

当程序退出时，占用的内存释放给 DOS，DOS 可以用这块内存装入和执行其它的程序。

此功能调用还把结束、Ctrl-Break 和严重错误的出口地址恢复成程序开始执行时的数值。DOS 从程序段前缀 (PSP) 中取回以前的数值，这些数值是在程序启动时被保留在这里的。

此功能调用关闭所有的文件描述字，并清除全部文件缓冲区，如果需要，把部分填充的缓冲区写到磁盘上。如果已经修改过某文件的长度，那么在发出这个中断之前，必须关

闭该文件（用功能调用 01H 或 3EH），否则，这个文件的长度、修改时的日期和时间将不能正确记录在目录中。

通过调用这个中断使一个程序退出时，不能把一个结束码或错误码传递给调用它的程序。如果程序需要传递这些代码，退出时应该使用功能调用 31H 或 4CH。有关内容参见第四章。

程序段前缀的偏移地址 0 中存有一条 Int 20 指令，因此，调用这个中断的另一种方法是跳转到 PSP 的偏移地址 0 处。

入口参数：

调用前，需设置寄存器：

CS 程序的 PSP 的段地址

DOS 中断 20H

终止程序

出口参数：

程序运行结束时，控制转移到程序段前缀中标出的结束地址，无返回数据。

其它要求：

在网络环境中，如果已经用 DOS 功能调用 5EH 封锁了某个文件，则必须在程序结束执行前取消这个封锁，否则，其它的程序将不能访问这些文件。

参见：

DOS 功能调用 31H—结束并保持驻留 (KEEP)

DOS 功能调用 4CH—终止进程 (EXIT)

## DOS 中断 21H

调用 DOS 功能调用

中断：

21H 调用 DOS 功能调用

DOS 版本：

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明：

此中断可调用所有标准的 DOS 功能。第四章将详细说明每个功能调用，并阐明调用前如何正确设置寄存器的值。

入口参数：

调用前，需设置寄存器

All DOS 功能调用号。

有关其它寄存器的设置可参见第四章。

出口参数：

返回后，有关寄存器的设置情况可参见第四章。

## DOS 中断 22H

终止地址

中断:

22H 终止地址

DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明:

用户程序不应直接调用此功能, DOS 使用与这个中断向量机对应的地址保存信息, 这个信息是程序结束时返回的地址。

装入一个程序时, DOS 从中断向量表中拷贝程序终止地址并把它放入程序段前缀 (PSP) 中。程序结束时, DOS 检查 PSP 中的结束地址, 看应当恢复执行哪一个程序, PSP 的划分可参见图 3-1。

如果在 DOS 提示符下通过键入程序名调用某程序, 中断向量表中的结束地址将是 DOS 命令处理程序 COMMAND.COM 的地址。因此, 当该程序结束执行时, 控制返回到 COMMAND.COM, 并在屏幕上显示 DOS 提示符。此时, 可以调用其它的程序。

但是, 如果执行一个调用子进程的程序, 子进程应该把控制返回它的父进程, 而不是 DOS。当功能调用 4BH 引进子进程时, 功能调用自动设置结束地址, 它指明控制返回到程序而不是 DOS, 因此, 当子进程退出时, 将自动返回控制到父进程。关于功能调用 4BH 的详细内容参见第四章。

对于 DOS shell 下的程序调用, 如 Microsoft Windows 或 Top View, 终止地址将指向返回 DOS shell。

## DOS 中断 23H

Ctrl-Break 地址

中断:

23H Ctrl-Break 地址

DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2

说明:

象功能调用 22H 一样, 用户程序不应直接调用此功能调用。当操作员按中断序列键 (Ctrl-Break 或 Ctrl-C) 时, 才发出这个中断。

通常, 仅有少数 DOS 功能调用检查操作员是否已经按了中断序列键, 这些功能调用是执行标准输入、标准输出、标准打印操作和标准异步通信操作的。但是, 如果用 DOS 功能调用 33H 或 DOS 命令 BREAK 打开 Ctrl-Break 检查程序, 则大多数 DOS 功能调用 (除了功能调用 06H 和 07H 之外) 将检验这种情况。如果这些功能调用检测出操作员已经按了中断序列键, 操作系统将按有关要求设置寄存器值, 并发出中断 23H。中断

23H 执行特定的 Ctrl-Break 处理程序，程序的地址在中断向量表的相应入口中列出。DOS 提供了一个缺省的 Ctrl-Break 处理程序，处理它检测到的任意一个缺省的 Ctrl-Break 处理程序，处理它检测到的任意一个中断序列。缺省 Ctrl-Break 处理程序仅终止执行当前的程序，但用户可以用自己的 Ctrl-Break 处理取代缺省的 Ctrl-Break 处理程序，使它执行自己所希望的任何操作。Ctrl-Break 处理程序执行时，一般终止中断序列功能（过程没作任何事，仅仅返回），从重复执行的操作退出，而不终止整个程序（EDLIN 程序这样做就退出插入方式）。

Ctrl-Break 处理程序能够完成多种操作，它可以使用任何一个 DOS 功能调用。但是，如果用户处理程序改变任何寄存器，首先应该保存这些寄存器中的数值并且在退出之前恢复它们。

有两种退出 Ctrl-Break 处理程序的方法。常用的方法是借助一条中断返回（IRET）指令，这条指令恢复被中断的程序使它继续执行。第二种退出 Ctrl-Break 处理程序的方法是借助一个长返回。在这种情况下，DOS 通过检查进位标志（CF）确定是否继续执行原来的程序。反之，则取消原来的程序。

如果由功能调用 09H（输出字符中）或 0AH（键盘缓冲输入）检测到这个中断序列，将显示字符：^C，后而跟一个回车并在屏幕上换行，这个应答表示 DOS 已接收中断序列。

当一个程序开始执行时，DOS 在中断向量表中为它设置 Ctrl-Break 地址。例如，如果用户写一个程序，调用 DOS 功能调用 4BH 装入并执行另一个程序，那么 DOS 为第二个程序设置 Ctrl-Break 地址。因此，当使用中断序列键时，控制返回到第一个程序功能调用 4BH 指令后的那和指令。

当 DOS 为新的程序设置 Ctrl-Break 地址时，它把旧的 Ctrl-Break 地址存入新程序的 PSP 中。即使这个新的程序设置了一个不同的 Ctrl-Break 程序，新程序执行结束时，DOS 可以从它的 PSP 恢复原来程序的 Ctrl-Break 处理程序。

参见：

DOS 功能调用 33H——获取或设置 Ctrl-Break 标志。

DOS 功能调用 4BH——装入或执行程序（EXEC）。

## DOS 中断 24H

### 严重错误处理地址

中断：

24H 严重错误处理地址

DOS 版本：

1.0, 1.1, 2.0, 3.0, 3.1, 3.2, 3.3

说明：

这个中断也是一个不能由用户程序直接调用的中断。在功能调用中遇到错误时，如果 DOS 认为是严重错误，就发出这个中断。通常，严重错误是一些磁盘错误（诸如写保护或试图访问一个不存在的软盘驱动器，驱动器把手未关等）。但其它的问题，如坏的存贮

器或打印机无纸也可以引起严重错误。

DOS 仅在通过中断 21H 调用系统服务程序时才产生严重错误中断。在纸对磁盘读(中断 25H) 和绝对磁盘写(中断 26H) 操作期间，不产生严重错误中断。

在产生严重错误中断之前，DOS 将引起严重错误的操作重复三遍。

当产生严重错误中断时，DOS 提供了一个缺省的处理程序。这个缺省处理程序让用户根据显示的信息：

Abort, retry, or ignore?

决定如何工作。

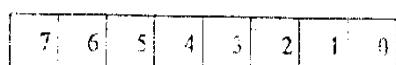
然而用户可以用自己的程序代替缺省的处理程序，完成更复杂的处理。

当严重错误处理程序开始执行时，堆栈和几个寄存器的内容将是关于错误类型的信息。这些信息有：

D1 低位字节是一个描述错误类型的代码，高位字节无定义。可能出现的错误代码如下：

代码	说 明
00H	试图写一个有写保护的磁盘。
01H	部件未定义(例如，没有 D 驱动器时，请求与其有关的操作)。
02H	驱动器未准备好(可能驱动器门未关闭)。
03H	未知命令。试图执行一个设备无法完成的操作。
04H	CRC 数据错。可能是硬件问题。
05H	命令结构长度错。错误码指出所要求的操作类型与发送到设备驱动器的数据量不符。
06H	查找错误。
07H	设备中使用的存储介质类型未知。
08H	设备驱动程序未找到指定的扇区。
09H	打印机无纸。
0AH	一般写错误。
0BH	一般读错误。
0CH	一般错误。

AH 指出错误是否是磁盘错误，如果是，则指出是哪一类磁盘错误。各位设置如下：



位	说 明
0	指出引起错误操作的类型如下: 值 操作 0 读操作 1 写操作
1-2	这二位指出错误的磁盘区域如下: 位 位 磁盘发生错误的区域 0 1 DOS 区域 0 1 文件分配表 (FAT) 1 0 目录区 1 1 数据区
3-5	检测出严重错误后，可以命令 DOS 做四种操作：忽略错误、重新操作、取消引起问题的系统调用和结束整个程序，用户的严重错误处理程序可以通过给 AL 置合适的值选择相应操作，并执行一个 IRET 指令（这在后面说明），这 3 位指出现严重错误时，应该选择的操作。 位 3=1 使引起问题的系统调用失效 =0 作废系统调用无效 位 4=1 重作原来的操作 =0 重作无效 位 5=1 忽略错误并继续 =0 忽略错误无效

如果第 7 位置“1”，错误是一个其它类型的错误。但有一个例外，这个例外也是一个与磁盘有关的错误，与 DOS 保留在内存中的文件分配表 (FAT) 的拷贝有关。如果这个 FAT 不能被访问，将通过这位置“1”返回严重错误信息。检查由 BP: SI 指向的设备标题确定这个错误是否与磁盘有关。如果 AH 的第 7 位置“1”，且设备标题指出的设备是一个块设备，这个错误与内存中 FAT 的拷贝有关系。如果设备标题指出设备是一个字符设备，那么不是磁盘错，应检查 DI 寄存器查看错误的原因。

AL 如果是磁盘错（即AH寄存器的第7位置“0”），这个寄存器列出引起问题的驱动器号。

BP: SI 对于DOS 2.0以上版本和较大的系统，这个寄存器对指向一个设备标题控制块，该控制块包括与引起错误的设备有关的信息，DOS1.X 不返回这个信息。设备标题控制块是一个由设备驱动程序提供的数据结构，通常 DOS 使用它与设备通讯。设备标题控制块可以提供有关设备的有用信息，用户的严重错误处理程序使用它确认所发生的错误，但不应改变它的内容。设备标题控制块的格式如下图所示：

01H	
02H	指向下一个设备(4字节)
03H	
04H	
05H	属性(2字节)
06H	
07H	对策子程序偏移地址(2字节)
08H	
09H	中断子程序偏移地址(2字节)
0AH	
0BH	名称和部件号(8字节)
12H	

设备标题控制块中的有关部分如下：

**指向下一个设备** 这是一个指针，它指向下一个设备的控制块。如果这是被初始化的最后一个设备，这个值将是 FFFFH。

**属性** 这个字描述设备的属性。与严重错误处理程序有关的位是：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x										x	x	x	x	

位	说 明
15	如果设备是字符设备，这位置“1”，块设备（如磁盘）则置“0”。 如果设备是字符设备（第 15 位置“1”），第 0 到第 3 位也有意义。否则，这四位无定义。
14	这是 IOCTL 位。如果这位置“1”，这个设备可以接受 DOS 功能调用 44H 发送给它的控制字符串。如果这位置“0”，设备拒绝接受控制字符串。
3	如果设备是字符设备并且是当前的 CLOCK 设备，这位置“1”。
2	如果设备是字符设备并且是当前的 NULL 设备，这位置“1”。
1	如果设备是字符设备并且是当前标准输出（stdout）设备，这位置“1”。
0	如果设备是字符设备并且是当前标准输入（stdin）设备，这位置“1”。

**对策子程序的偏移** 这是设备驱动程序对策子程序首址的偏移量。这个子程序的设备标题控制块存在同一段中，因此它的地址的基址部分存放在 BP 寄存器中。

**中断程序的偏移** 这是设备驱动器中断程序首址的偏移量。这个程序和设备标题控制块存在同一段中，因此它的地址的基址部分存放在 BP 寄存器中。

**名称和部件号** 这 8 个字节包括设备或部件名称，对于块设备还饮食部件号。如果是字符设备，名称由 ASCII 字符组成（向左对齐，后面用空格补齐）。块设备使用相同的格式，但知一个字节标识设备中的部件号。

**堆栈** 当控制转给严重错误处理程序时，这个堆栈装有严重错误出现时正在运行的那处程序的寄存器组。从栈顶到栈底的内容为：

IP	这 3 个值是在 DOS 功能调用传递
CS	控制给严重错误处理程序时
FLAGS	CS:IP 和 FLAGS 的值
AX	用户程序调用 DOS 功能调用产生严重
BX	错误时，各个寄存器的值。
CX	
DX	
SI	
DI	
BP	
DS	
ES	
IP	
CS	

当用户的错误处理程序接收控制时，它应该避免使用大多数 DOS 功能调用，但可以使用或能调用 00H 到 12H 与用户通信。如果调用其它功能调用将破坏堆栈的内容，这样将无法返回到原来的程序。

用户提供的错误处理程序可以采取几种措施，最容易的是让 DOS 处理错误。可以在 AL 寄存器中放入适当的数值，告诉 DOS 做什么，并执行一个 IRET 指令。AL 寄存器的数值可以是：

AL 值	说 明
0	DOS 不管这个错误，程序继续执行。使用这种选择应注意，因为这时 DOS 认为以前的功能调用（引起严重错误的那个）成功地完成了。 对于 DOS3.X，如果 AH 寄存器的第 5 位置“1”，表明不允许忽略那个错误，DOS 使这个忽略请求失效 (AL=3)。对于 DOS3.X，如果访问的扇区是 FAT 的一部分或者目录，则产生磁盘错，DOS 使个忽略请求失效 (AL=3)。 对于 DOS3.1 和以后的版本，如果出现一个严重网络错误 (32H-4FH)，DOS 使这个忽略请求失效 (AL=3)。
1	DOS 重新操作。 对于 DOS3.X，如果 AH 寄存器的第 4 位被置“1”，表明不允许重新操作，DOS 使重作请求失效 (AL=3)。
2	DOS 发出 Ctrl-Break 中断 (23H)，终止执行程序。
3	DOS 使引起错误的 DOS 功能调用失效，并继续执行程序。对于 DOS3.X，如果 AH 寄存器的第 3 位被置“1”，表明不允许这个功能调用失效，DOS 将把这个失效功能调用变为终止程序功能调用 (AL=2)。

用户严重错误处理程序的另一种选择是自己处理错误。这时用户程序负责处理错误，并从堆栈恢复原程序中寄存器数值，清除栈里的除最后 3 个字（原来程序的 IP，CS 和 FLAGS 寄存器值）外的所有值，并且发出 IRET 指令，处理机将执行原程序中产生严重错误的下一条指令。

如果自己处理错误，DOS 的运行情况难以预测，只有原来程序发出功能号大于 12H 的 DOS 功能调用，这种情况才能终止。DOS 技术参考手册警告说，这时 DOS 将处于一种不稳定的状态。

## DOS 中断 25H

### 绝对读盘

中断：

25H 绝对读盘

DOS 版本：

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明：

该中断读取磁盘扇区，与磁盘的文件结构无关。这个操作与 ROM BIOS 中的一个操

作相似，但读取时识别数据的方法不同。

用户使用大多数 DOS 功能调用时，只需说明读取信息的文件，而使用 ROM BIOS 功能调用，则应该通过给出磁头、磁道和扇区号说明信息所在扇区。用此中断时，可以通过给出要读的第一扇区的逻辑扇区号指定扇区。DOS 按如下原则给一个磁盘分配逻辑扇区号：

第 1 逻辑扇区（逻辑扇区 0）定位在 0 头、0 道、1 扇区。

其余的逻辑扇区号依次在相同的磁道磁头中，即逻辑扇区 1 是定位在 0 头、0 道、2 扇区；逻辑扇区 2 是在 0 头、0 道、3 扇区。（用 DOS1.X 格式化的软盘每道 8 个扇区；用 DOS2.0 和以后的版本格式化的软盘每道 3 个或 9 个扇区）。

0 道的最后一个扇区之后，逻辑扇区号从 0 头、1 道、1 扇区开始继续定位，直到 0 面（头）上的所有磁道都用完。

如果是双面盘，从第二面（1 头）开始以同样的方法继续定位。

按照这个编号方案，连续的逻辑扇区在磁盘上可能不对应连续的物理扇区，这取决于格式化磁盘时所用的交替因子，这个交替因子在两个逻辑扇区之间规定物理扇区号。

如果出 I/O 错，磁盘读操作不能重作，DOS 也不调用严重错误处理程序（中断 24H）。

#### 入口参数：

调用前，设置寄存器内容：

AL 指定读取数据的驱动器：

0 驱动器 A

1 驱动器 B

2 驱动器 C

等等

CX 所读的连续逻辑扇区数。

DX 所读的第一个扇区的逻辑扇区号，用原来设置的信息把磁盘物理位置转换成逻辑扇区号。

DS: BX 指出磁盘传送区域在内存中的地址，DOS 将所读的信息放在这个区域，用户应保留足够的内存区域容纳所要读的扇区数。

#### 出口参数：

返回后，设置如下：

DS: BX 指向传送区域的首地址，DOS 在这里存入所读的信息。

堆栈 返回时，栈中原来的标志寄存器是有效的（其它的 DOS 调用通常在控制返回前弹出标志寄存器值）。此调用把标志寄存器值放在栈中，因为它要修改标志寄存器的内容，指出是否有错误出现。返回时，必须从栈中清除这个值，以避免堆栈溢出，并保证后续的退栈操作（如 RET 指令）可以得到正确的值。

#### 其它的要求：

调用前，应保存每个重要寄存器的内容。因为这个中断破坏除段寄存器（CS, DS, SS 和 ES）之外的所有寄存器的内容。

**错误码:**

如果操作成功，此调用将进位标志 (CF) 设为“0”；若出错，则设置为“1”。下列寄存器指出错误：

AL      这个寄存器的内容与中断24H返回到D1寄存器中的错误代码一样。这些代码如下所示：

代 码	说 明
00H	试图写一个有写保护的磁盘。
01H	部件未定义（例如，可能在没有 D 驱动器时，请求与其有关的操作）。
02H	驱动器未准备好（可能驱动器未关闭）。
03H	未知命令。试图执行一个设备无法完成的操作。
04H	CRC 数据错。可能是硬件问题。
05H	命令结构长度错。错误代码指出所要求的操作类型与发送到设备驱动器的数据量不符。
06H	查找错误。
07H	设备中使用的存储介质类型未知。
08H	设备驱动程序未定位在指定的扇区。
09H	打印机无纸。
0AH	一般写错误。
0BH	一般读错误。
0CH	一般错误。

AH      包括下述错误代码之一：

代 码	说 明
80H	DOS 连接设备失败。
40H	寻找操作失败。
08H	软盘读操作时 CRC（循环冗余码校验）错。
04H	扇区未找到。
03H	写保护。
02H	其它错误。

## DOS 中断 26H

### 绝对写盘

**中断:**

26H 绝对写盘

**DOS版本:**

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

该中断和中断25H相似，只不过是写入而不是读取磁盘扇区，输入参数和给定逻辑扇区号的方法和中断25H中所描述的一样。

**入口参数:**

调用前，需设置：

**AL** 指定写取数据的驱动器

0驱动器A

1驱动器B

2驱动器C

等等。

**CX** 所写的连续逻辑扇区数。

**DX** 所写的第1个扇区的逻辑扇区号，用中断25H中说明的逻辑扇区号分配原则把磁盘物理位置转换成逻辑扇区号。

**DS: BX** 磁盘传送区域在内存中的地址，DOS将所写的信息放在这个区域，用户应保留足够的内存区域容纳所要写的扇区数。

**出口参数:**

返回后，设置如下：

**堆栈** 返回时，堆栈中原来的标志寄存器是有效的（其它的DOS调用通常在控制返回前弹出标志寄存器值）。此调用把标志寄存器值放在堆栈中，因为它要修改标志寄存器的内容，指出是否有错误出现。返回时，必须从堆栈中清除这个值，以避免堆栈溢出，并保证后续的退栈操作（如RET指令）可以得到正确的值。

**其它要求:**

调用前，应保存每个重要寄存器的内容。因为这个中断破坏除段寄存器（CS, DS, SS 和 ES）之外的所有寄存器的内容。

**错误条件:**

如果操作成功，此调用设置进位标志（CF）为“0”；若出错，则置“1”，下列寄存器指出错误：

**AL** 这个寄存器的内容与中断24H返回到DI寄存器中的错误代码一样。这些代码如下所示：

代 码	说 明
00H	试图写一个有写保护的磁盘。
01H	部件未定义（例如，可能在没有 D 驱动器时，请求与其有关的操作）。
02H	驱动器未准备好（可能驱动器未关闭）。
03H	未知命令，试图执行一个设备无法完成的操作。
04H	CRC 数据错，可能是硬件问题。

代码	说 明
05H	命令结构长度错，错误代码指出所要求的操作类型与发送到设备驱动器的数据量不符。
06H	查找错误。
07H	设备中使用的存储介质类型未知。
08H	设备驱动程序未定位在指定的扇区。
09H	打印机无纸。
0AH	一般写错误。
0BH	一般读错误。

AH 包括下述错误代码之一：

代码	说 明
80H	DOS 连接设备失败。
40H	寻找操作失败。
08H	软盘读操作时 CRC（循环冗余码校验）错。
04H	扇区未找到。
03H	写保护。
02H	其它错误。

## DOS 中断 27H

结束但驻留内存

中断：

27H 结束但驻留内存

DOS 版本：

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明：

此中断使 DOS 终止当前的程序，但使其仍驻留在内存，并且不被装入的其它程序覆盖，这个特性与 Prokey 和 Sidekick 相似。

DOS 用多种方法使程序驻留在内存中，这个中断是 DOS1.0 提供的，为了兼容性保留了下来，仅对.COM 程序有效。如果使用 DOS2.0 或更高版本，则应使用 DOS 功能调用 31H 提供的同样功能，所以建议使用功能调用 31H，是因为它允许终止程序传递一个结束码，其它程序（经功能调用 4DID 和 DOS 批处理的 ERRORLEVEL 特性可以检验这个代码。此外，功能调用 31H 允许在内存中驻留的程序大于 64K 字节（一个.EXE 文

件), 而中断 27H 则不允许。

在调用此中断使程序驻留内存前, 应首先读取 DX 寄存器, 指出必须在内存中驻留程序的长度。其它程序装入时, DOS 把 DX 寄存器中的数值作为该程序可以使用的第一字节, 从程序的首址到 DX 寄存器规定的偏移量的前一个字节, 供内存驻留程序使用, 其余所有内存被释放。

因为程序使用 PSP 中的偏移量作为新文件的起始单元, 所以内存驻留的代码数量不能超过 64K 字节。

此中断不释放该程序用 DOS 功能调用 48H (分配内存) 或 4AH (修改分配内存块) 所分配的内存, 只有通过调用 DOS 功能调用 49H (释放已分配的内存) 才释放它们。通过重新定位与程序有关的环境, 可以将驻留程序占用的内存量压缩到最低限度。环境的段地址保存在距 PSP 首址偏移 2CH 处开始的字中 (关于 PSP 的结构见第三章 PSP 的说明部分)。重新分配环境时, 把这个字装入 ES 寄存器中, 并调用功能调用 49H (释放已分配的内存)。

象中断 20H 一样, 该中断把结束、Ctrl-Break 和严重错误出口地址设置成调用前的值。DOS 从程序段前缀台取回以前的值, 这些值是调用此功能时保存在那里的。

当一个程序调用此中断终止执行时, 它的已打开的文件都不能自动关闭。因此, 在结束执行前, 该程序必须关闭所有已打开的文件。

#### 入口参数:

调用前, 需设置寄存器:

CS 程序PSP的首址, 即程序被启动时DS和ES的值。

DX 程序驻留部分的字节数加1, 是相对于程序的初始CS值的偏移量。

#### 出口参数:

无

#### 参见:

DOS 功能调用 31H——结束并驻留 (KEEP)

DOS 功能调用 48H——分配内存

DOS 功能调用 49H——释放已分配的内容

DOS 功能调用 4AH——修改已分配的内存块 (SET\_BLOCK)

DOS 功能调用 4BH——装入或执行程序 (EXEC)

DOS 功能调用 4CH——终止进程 (EXIT)

## DOS 中断 28H-2FH

#### 保留:

#### 中断:

28H-2FH 保留

这些中断均为 DOS 保留用, 程序不应调用。

## 第四章 DOS 功能调用

DOS 功能调用象第三章描述的 DOS 中断一样，是 DOS 提供的使程序直接访问操作系统的服务。程序调用功能调用的方法类似于调用其它 DOS 中断：对某些寄存器赋值并执行软中断。可是与 DOS 中断不一样的是所有功能调用均使用相同的软中断(INT21H)，或用将要描述的其它两种方法。

DOS 功能调用依照它们提供的服务，可分成若干类。这些服务包括：

- 程序执行和结束（加载并执行程序，结束程序和结束进程）
- 字符设备的 I/O 功能（如键盘、显示器、打印机）
- 文件管理功能（发选盘，创建或删除文件，打开或关闭文件，读或写文件）
- 内存管理功（如分配和释放内存）
- 日期和时间功能
- 网络功能（如文件加锁、解锁，取局域计算机名和设置网络打印机）
- 其它各种功能

### § 4.1 如何使用本章

本章的基本目的是要以标准和易于使用的方式详细地描述所有 DOS 功能调用。每一个功能调用都另起一页，并且在该页顶端给出此功能调用的名字和功能调用号，几个标准标题下给出了有关信息，这样在相同的标题下可以找到同类信息。另外，对大多数功能调用将分别给出三个例子：一个用汇编语言编写（适合于 IBM 或 Microsoft 汇编语言），一个用 Turbo Pascal 编写，另一个用 Microsoft C 语言编写。它们不是程序中的一段代码，而是能够输入并执行的完整程序。用户也可以根据自己程序的需要对它们进行修改。

在详细叙述功能调用之前，本章提供了一些叙述所需要的背景材料，它讨论了两种不同的文件管理方法（DOS1.X 方法）、许多功能调用的方法和通用数据结构以及返回的标准错误信息。

### § 4.2 文件管理：DOS1.X 以上各版本的比较

DOS1.X 中功能调用所提供的用于文件进行 I/O 操作的方法比较复杂。在 DOS2.X 中，提供了另一组功能调用来替代旧的功能调用，许多 DOS2.X 功能调用完成与 DOS1.X 相同的操作，但使用的方法不同。

DOS2.X 新增加的一组功能调用并不取代旧的功能调用，只是增加了可用的功能调用号，后续的 3.X 版本也是如此，这样在写程序时，经常要在完成同样的一个功能时对功能调用进行选择。

这种选择对大多数人来说都很容易，在后期的版本中保留了 DOS1.X 文件管理功能调用以便与早期系统兼容。使用这些功能调用的程序可以在新系统上运行，但是不能享受

DOS2.X 和 DOS3.X 所引入新特色的好处，DOS2.X 引入的新的文件管理技术更易于使用，并且它支持 DOS2.0 及以后版本的层次路径名结构。

本章下两节描述文件管理的两种方法并详细描述所使用的数据结构，先描述 DOS1.X 方法（IBM 手册中所说的“传统文件管理”）再描述 DOS2.X 方法（“扩展文件管理”）。

先描述 DOS1.X 方法并不是为了推荐此方法，仅仅因为它是最先引入的概念并且相应功能调用号比较低，除非特别希望保持与 DOS1.X 的兼容性，否则应当使用 DOS2.X 的文件管理方法。

### § 4.3 DOS1.X 文件管理方法

传统文件管理方法包括调用号从 0FH 到 24H 的功能调用，这些功能调用允许创建、删除文件；打开、关闭文件；读写和完成其它类型的标准文件操作。使用此组功能调用的关键是考虑文件控制块（FCB）的数据结构。

文件控制块是一个用来向 DOS 传递与要操作文件有关的 44 字节信息的区域，DOS 使用 FCB 向程序返回信息，图 4-1 给出 FCB 的结构，对于每个已打开的文件必须分别提供 FCB。

例如，假设希望使用 DOS1.X 文件管理功能调用来打开一个文件，首先必须建立一个与 FCB 结构相应的数据结构，然后填写 FCB 中的字段来指定驱动器号、文件名和扩展名，然后使用打开文件功能调用（0FH），在 DS: DX 中放置指向刚建立的 FCB 的指针。

当 DOS 找到所调用的文件时，进一步填写描述该文件的 FCB 的其它字段，例如，它将最后一次文件存取的日期时间和文件的尺寸填写到 FCB 中相应的字段。

执行其它操作如读和写，在设置 FCB 中相应字段来指明读或写的部分之后，要再次向 DOS 传送 FCB 的指针。

在 DOS2.0 和以后的系统中此文件管理方法的主要不足是 FCB 不能支持子目录，文件名字段只有 8 个字符长并且使用 FCB 的功能调用不认路径名分隔符，因此，FCB 只能用于指定驱动器下现行目录中的文件。

使用功能调用时操作 FCB 的细节在相应的各个功能调用中进行说明，但是 FCB 的格式和各个字段的描述将在这里进行说明，当使用功能调用时再参阅本章的这一部分来找 FCB 的相应字段。

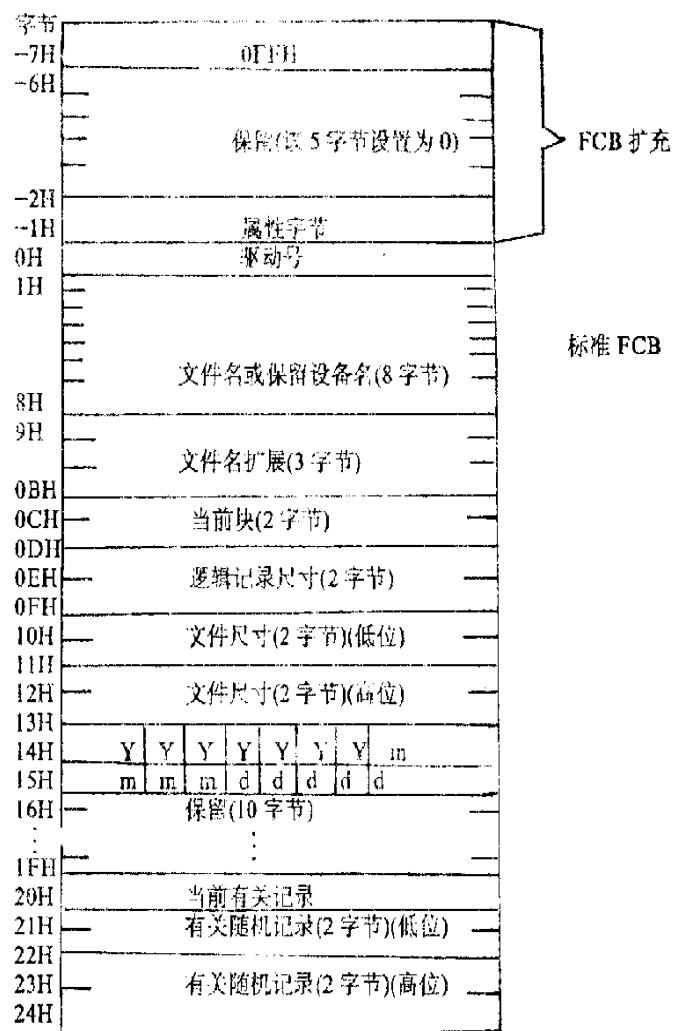


图 4-1 文件控制块 (FCB) 结构

#### § 4.4 文件控制块格式

图 4-1 给出文件控制块的格式。如图所示，FCB 分成两部分。标准 FCB 和扩展 FCB。标准 FCB 存贮有关文件的信息，如名字、尺寸、最后存取的日期和文件指针的现

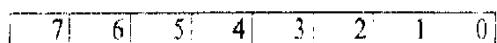
行位置。扩展 FCB 指出文件是否具有一些特殊的属性（如隐蔽文件、系统文件或其它特殊类型的文件）。从理论上讲，仅当所使用的文件具有特殊属性时才用扩展 FCB，实际上应当经常允许扩展 FCB。

当该功能调用提供 FCB 的单元时，第一个字节的值即决定了该 FCB 是标准的还是扩展的 FCB，第一个字节中的 FFH 指明是扩展的 FCB，含有图 4-1 中所有的字段区域。如第一个字节中的值不是 FFH，该 FCB 是一个仅含有图 4-1 中字节 0 到最后字段的标准 FCB。

字节 7 到 15 和字节 32 到 36 是在打开文件前应适当填写的字段，字节 16 到 31 是当 DOS 打开文件做其它 I/O 操作要填写的字段。

FCB 的字段说明如下：

字 节	说 明
-7H	扩展 FCB 的第一个字节，设这一字节为 0FFH 指明 FCB 的扩展部分无效。
-6H-2H	这些是保留字节，应设为 0。
-1H	属性字节。此字节是该文件特殊属性的一个编码指示器，此字节按如下编码（0 位是最右或最低比特位）。



- |                     |   |
|---------------------|---|
| 0位                  | 如该位被置 0，则该文件被标识为只读，对于 DOS 2.0 及以后版本，如企图写该文件，则返回一个错误代码。DOS1.X 此位无效。                |
| 1位                  | 如此位被置 1，该文件为隐蔽文件，该文件对正常的列目录不显示并且在目录搜索中找不到。  |
| 2位                  | 如此位被置 1，该文件为系统文件，它被排除于正常的目录搜索之外，系统文件没有区别于其它类型文件的其它特性，DOS 只是将它列为相应的一类文件。           |
| 3位                  | 如此位被置 1，该文件含有该盘的卷标，此卷标是在格式化磁盘时设置的，仅当此文件驻留在某卷的根目录上时 DOS 才识别此文件为卷标文件。               |
| 4位                  | 如此被置 1，该文件实际上是一个子目录而不是一个数据文件，该文件被排除于正常的目录搜索之外。DOS1.X 此位无效。                        |
| 5位                  | 此位是档案位，BACKUP 和 RESTORE 命令使用这一位来决定什么时候文件需要备份。每当文件被改变，DOS 置起此位，BACKUP 在它备份该文件时清此位。 |
| 6-7位                | 这些位被保留，必须置 0。   |
| 对正常文件，属性字节的所有位应置 0。 |   |
| 0H                  | 驱动器号。这个字节的值指出含有该文件的驱动器。在打开文件前设这个值时，该值具有下列含意：                                      |

- 0 文件在缺省驱动器
- 1 A驱动器
- 2 B驱动器
- ..... 依次类推

如这个字段置0,那么DOS在打开文件时,将改变该值指向含有该文件的实际驱动器,一旦文件被打开,这些字段的值有下列含意:

- 0 A驱动器
- 1 A驱动器
- 2 B驱动器
- 3 C驱动器
- ..... 依次类推

1H到8H 文件名。这8位含有表示该文件名字的8个ASCII字符,如名字少于8个字符,则应向左调整并且填空格字符(ASCII20H)补齐。

也可以指定下列保留名字:

CON	LPT1
AUX	LPT2
COM1	PRN
COM2	NUL

如使用这些保留名,不要在名字后跟一个冒号(用LPT1,不用LPT2:)。DOS认为指定的文件存在于当前驱动器的现行目录下,使用FCB无法指向存在于子目录下的文件。

9H到0BH 文件扩展名。这3个字节含有表示该文件扩展名的3个ASCII字符,如该扩展名少于3个字符,应向左调整并补空(ASCII20H)。

0CH到0DH 现行块。这是相对于文件起始处的现行块号,一块定义为128个记录(记录的大小定义在逻辑记录大小字段中)。这些字段和现行记录字段用来指定顺序读写文件时的位置。文件的第一块是0块,DOS在打开该文件时设块号为0。

0EH到0FH 逻辑记录尺寸。此字段中的数值表示以字节为单位的逻辑记录的大小,在用功能调用读写记录时,DOS按照这里指定的字节个数进行读写。

只要实际上处理的不是一个定长记录,就可以对一个文件使用许多不同的记录长度而不影响该文件。当DOS打开文件时,它将记录长度字段设置为80H(128),一旦文件被打开,就可以将置记录长度设置为其它数值。通常,程序设置记录长度为较小的数值以保留缓冲区空间,但是可以用较大的记录尺寸来改善性能,因为不需要执行这么多次功能调用来自传送同样量的数据。

10H到13H 文件尺寸。这两个字组成32位的整数指出以字节为单位的文件的大小。第一个字为低16位,第二个字为高16位(当然,对于任何字变量,低字节经常是第一个字节,而不是象所希望的那样是后一个字节)。例如,如果文件的大小是12345678H,这个大小以下列顺序

存在 FCB 的 10H 到 13H 中:

78H

56H

34H

12H

当 DOS 打开文件时，它从该文件的目录项中获取文件尺寸，并将该值放在此字段中。如果文件用作输出被打开，则 DOS 依照文件的大小改变而改变文件的这个字段；当文件关闭时，DOS 以这个字段中所给出的大小来更新该文件的目录项。

DOS 文档指出程序不应修改文件尺寸字段，但也许在特定的场合下，用户希望这么做。例如，假如希望象数据文件一样测试目录。打开目录前，文件尺寸被设置为 0。为了访问该目录中的数据，就必须首先将文件尺寸字段，设置为大的一些数目。

14H 到 15H 日期。这两个字节指出该文件建立或最后更新的日期。DOS 打开文件时，它从文件目录项中得到日期并将它拷贝到这里。

日期的编码如下：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
y	y	y	y	y	y	m	m	m	m	d	d	d	d	d	d

yyyyyy 指示现在年份，它的取值范围是 0 到 119（代表 1980 年 2009）。

mmmm 指示现在月份，它的取值范围是 0 到 12。

ddddd 指示现在日期，范围是 0 到 31。

16 到 1FH 保留。这 10 个字节为 DOS 保留使用。

21H 现行相对记录。这是现行记录（0---12）相对于现行块起始处的记录数，此字段和现行块字段用来指定顺序读写文件时的位置，DOS 在打开文件时不初始化该字段。

21H 到 24H 相对随机记录。这 4 个字节仅用于随机读写操作。它们指定相对于文件（不象现行记录那样相对于现行块）起始处的记录个数。DOS 在打开文件时不初始化该域。

如果指定一个记录尺寸为 64 字节或者更大，DOS 以为此字段只有前三个字节有效，如记录小于 64 字节，则 4 个字节均可使用，使用程序段中 5CH 处的 FCB 时，应对随机读写加以小心，此 FCB 中相对随机记录的最后一个字节覆盖未格式化参数区的第一个字节。

## § 4.5 磁盘传送地址

传统的文件管理功能调用也使用磁盘传送地址（DTA——disk transfer address）概念。DTA 是从磁盘读信息或向它写信息功能调用所用的内存缓冲区的起始位置，例如，要写信息到文件，应先将信息放置在起始于 DTA 的内存中，然后使用写序功能调用（15H）。若要从文件中读取信息，则使用顺序读功能调用（14H）。此功能调用自动获

取信息并将它放在起始于 DTA 的内存中。

在任何时候只有一个 DTA 是工作的，当程序开始运行时，有一个缺省的 DTA，从该程序的程序段前缀（PSP）中偏移量 80H 处开始，此缺省 DTA 有 128 个字节（PSP 的剩余部分）。如果要传送的缓冲区信息超过 128 字节，则该数据将覆盖程序。

程序可以通过使用设置磁盘传送地址功能调用（1AH）来改变 DTA 的位置，也可以通过功能调用 2FH（取磁盘传送地址）来取得现行 DTA 的地址。如果改变了 DTA，则可以使用大于 128 字节的缓冲区，而不会覆盖重要的信息。所有使用 FCB 的功能调用认为往返于文件的信息都是通过 DTA 的。

使用 DOS2.X 文件管理方法的功能调用不使用 DTA，取而代之的是，每次程序使用功能调用时，必须指定用于 I/O 操作的内存缓冲区。

## § 4.6 DOS2.X 文件管理方法

扩展的文件管理方法包括功能号从 39H 到 62H 的功能调用。这些功能调用以及一些其它的功能是从 DOS2.0 开始增加的。这些功能调用完成同传统的功能调用一样的操作，可是它们更易于使用并支持所有新的 DOS 特性，如层次目录和文件共享。

扩展的文件管理功能调用不使用 FCB，取而代之的是这些功能调用靠叫做文件描述字的 16 位数值进行工作。

打开文件时程序要提供一由 ASCII 字符串表示的完整路径名来标识文件，ASCII 0 串可以简单地用该文件路径名的 ASCII 字符串加一个用来终止该名字的字节 0 来构成。例如，下面是一个 ASCII 字符串的汇编语言定义：

```
DB"C:\utils\dp\facts., dat", 0
```

驱动器字母和路径标识（本例中的 C:\utils\dp\ 可以省略，这时 DOS 认为在缺省驱动器和缺省目录、路径分隔符既可是反斜杠\也可是正斜杠/，正斜杠与 Unix 路径结构的形式兼容。

终止字节 0 是必需的，因为带路径名的文件名长度可以有相当大的变化，该字节 0 为 DOS 指出路径名的结束。

打开文件的程序也可提供有关要打开的文件的一些附加信息，比如，它能提供属性字节（对传统文件管理，保存在扩展 FCB 中），再比如，程序可以定义它要如何操纵该文件（读，写或可读可写）和它要怎样同可能并发运行的其它进程共享该文件。

当 DOS 打开文件时，返回该文件的 16 位的描述字，要用文件完成其它 I/O 操作时（如读写），可以简单地为功能调用提供文件描述作为输入，此描述字取代 FCB 为 DOS 标识文件，因此，打开该文件后，程序不需要保存整个的 FCB，并且它甚至不需要保存 ASCII 0 串，该文件描述字就是所需要的一切。

有五种对所有程序有用的描述字，它们由 DOS 事先定义，这样，程序不需要打开任何文件就能使用它们（尽管程序可以关闭这些描述字）。这些描述字是：

描述字	缺省设定	说 明
000H	CON	标准输入设备 (stdin)
001H	CON	标准输出设备 (stdout)

描述字	缺省设定	说 明
002H	CON	标准错误输出设备 (xtderr) (不能被重定向)
003H	AUX	标准附加设备 (stdaux)
004H	PRN	标准打印设备 (stdprn)

经常把 stdin 作为只读文件，把 stdout 和 stderr 作为只写文件。stdin 和 stdout 可以重定向，这样程序能够和读写控制台的方式一样读写文件；stderr 不能重定向，这对于经常希望在屏幕显示出错信息而不必担心重定向的程序是非常有用的。

可以使用 stdaux 和 stdprn 来进行读写操作。stdaux 一般说是串行设备，而 stdprn 是并行设备。

## § 4.7 标准错误代码

错误报告是面向 DOS2.0 以上版本的功能调用进行改进的又一个方面。

DOS1.X 功能调用依据 AL 寄存器中返回的一个数值 (0FFH) 来指出错误原因，而后来的功能调用根据设置进位 (CF) 和在 AX 寄存器返回标准错误代码集中某个内容来指出错误原因。

表 4.1 列出了 DOS 功能调用和严重错误处理程序中断 24H 能够返回的错误代码，DOS2.X 只能在 AX 中返回 0 到 18 的十进制错误代码，DOS3.X 增加了 32 到 88 的错误代码，只有 DOS3.X 功能调用能直接在 AX 中返回这些错误代码。19 到 31 的十进制错误代码由严重错误处理程序所返回，IBM 保持一张含所有错误代码的表，表中的代码是唯一的，如下表所示。但在严重错误处理程序实际返回这些原因时，使用 0 到 12 的十进制代码表示同样的原因。

16 进制码	10 进制码	意 义
1H	1	无效功能调用号
2H	2	文件没找到
3H	3	路径没找到
4H	4	无可用的描述字 (打开文件过多)
5H	5	访问拒绝 (如企图写只读文件)
6H	6	无效描述字
7H	7	内存控制块被破坏
8H	8	无足够内存
9H	9	内存块地址无效
0AH	10	无效环境 (SET,PROMPT,PATH 命令)
0BH	11	无效格式
0CH	12	无效存取代码
0DH	13	无效数据
0EH	14	保留

16进制码	10进制码	意义
0FH	15	无效驱动器标识
10H	16	企图删去现行目录
11H	17	不是同一设备
12H	18	无可用文件
13H	19	磁盘写保护，写企图失效
14H	20	没有指定部件
15H	21	驱动器未准备好
16H	22	命令未定义
17H	23	CRC 校验错
18H	24	命令顺序长度不当
19H	25	搜索错误
1AH	26	未知磁盘类型
1BH	27	指定扇区未找到
1CH	28	打印机无纸
1DH	29	写错误
1EH	30	读错误
1FH	31	一般错误
20H	32	非法文件共享
21H	33	非法文件加锁
22H	34	无效磁盘改变
23H	35	文件控制块无效
24H	36	共享缓冲区溢出
25-31H	37-49	保留
32H	50	网络调用无效
33H	51	远程计算机不响应
34H	52	网络名重名
35H	53	网络名未找到
36H	54	网络忙
47H	55	网络设备取消

16进制码	10进制码	意    义
38H	56	网络 BIOS 命令超界
39H	57	网络适配器硬件错误
3AH	58	网络响应不正确
3BH	59	未预料的网络错误
3CH	60	远程适配器不兼容
3DH	61	打印队列满
3EH	62	打印文件空间不够
3FH	63	打印文件被删除
40H	64	网络名被删除
41H	65	访问拒绝
42H	66	网络设备类型不正确
43H	67	指定网络名未找到
44H	68	网络名超界
45H	69	网络 BIOS 部分超界
46H	70	临时网络暂停
47H	71	网络不能接受的调用
48H	72	网络打印或磁盘重定向设备暂停
49H-4FH	73-79	保留
50H	80	文件已经存在
51H	81	保留
52H	82	不能创建目录项
53H	83	严重错误处理程序(24H)在功能调用中失败
54H	84	重定向过多
55H	85	重复重定向
56H	86	口令无效
57H	87	参数失效
58H	88	网络设备失效

表 4.1 错误代码表

编译器

参阅第三章中 DOS 中断 24H 的描述并将列在那里的错误代码和表 4.1 中 19 到 31 的十进制错误代码相比较。

DOS3.X 还增加了一个功能调用 (59H 得到扩展错误) 来获得出错的其它信息。此功能调用可以返回由设置进位标志来指出出错的任何功能调用的错误信息，包括 DOS1.X 功能调用在内。该信息包含一个错误代码 (如表 4.1 中所示)，表 4.2 中列出的错误类别 (提供了错误类型的更多信息)，表 4.3 中列出的建议动作，表 4.4 列出的地点 (该给出错误是由系统某部分引起的提示)。这个错误报告机制能够提供更易理解的错误信息，甚至对那些不直接报告个别错误的老功能请求也是如此。

16 进制	10 进制	意    义
1H	1	资源溢出，无剩余内存或无多余通道。
2H	2	临时问题。操作现在不能完成。但问题是暂时性的 (如文件共享)。
3H	3	权力问题。用户没有得到正式的允许。
4H	4	内部错误。这是一个 DOS 内部问题，与用户程序无关。
5H	5	硬件失效
6H	6	系统失效。说明系统严重问题，但不一定是 DOS 错误，如 CONFIG.SYS 文件错误。
7H	7	应用程序错误。用户程序进行不正确或不一致的请求调用。
8H	8	项目未找到。DOS 希望一个文件或其它项目，但用户程序未指定该项目。
9H	9	错误格式。DOS 期望的文件或其它项目格式不正确。
0AH	10	上锁。该文件 (或部分文件) 被锁且因此无效。
0BH	11	介质问题。磁盘坏或磁盘错误，CRC 错或其它有关磁盘问题。
0CH	12	已经存在。指定的文件或名字是已存在的同类型项的名字。
0DH	13	未知错误类别。上述 12 个错误类别以外的或无合适错误类别的错误。

表 4.2 错误类别

## § 4.8 使用功能调用

在使用任何功能调用之前，必须设置寄存器和可能的数据缓冲区及希望传送给功能调用的信息。在每一功能调用中应说明需设置的寄存器和数值。

当 DOS 在功能调用期间进行控制时，它转向自己的内部堆栈。调用程序必须有足够的大的堆栈，以适应中断系统，即必须允许超出程序自身需要的额外 128 字节。

怎样设置寄存器和怎样实际使用功能调用取决于所用的编程语言，本章提供三种语言的编程实例：汇编语言、Microsoft C 和 Turbo Pascal，因此下面几段将从这几种语言进行描述。

## § 4.9 汇编语言使用功能调用

用汇编语言使用 DOS 功能调用是最直接的，可简单地按照每个功能调用的描述设置寄存器并用下列方法使用功能调用：

将功能调用号置入 AH 寄存器，发中断 21H，建议选用这种方法进行功能调用。

将功能调用号置入 AH 寄存器并对程序段前缀偏移 50H 做一段间长调用。尽管这种方法是 DOS 的一种比较新的特色，但并不建议使用。该方法对于不能发中断的程序也许能增加自动防止故障的机制。

这种方法仅适用于功能调用 00 到 24H。提供它是为了保持与 DOS1.X 的兼容性，在新的系统中应避免使用。使用此方法要将功能调用号置入 CL 寄存器并对现行代码段偏移 5 做一段间调用。偏移 5 处放有对 DOS 功能调用程序的段间长调用。使用这种方法时 AX 的内容将被破坏。

## § 4.10 Microsoft C 使用功能调用

在 C 语言中，程序员一直访问寄存器。但是，Microsoft C 提供可以使 C 程序员使用 DOS 功能调用的接口。这个接口包括一个叫做 REGS 的联合类型，一个叫 SREGX 的结构类型和函数 intdos intdosx 和 segread。

REGS 联合类型定义处理器寄存器 AX, BX, CX, DX, SI, DI 和 CFLAG (进位标志)。它设置了两个结构的联合，这样既可以把它们作为 16 位的寄存器 (AX) 访问，又可以作为 3 位的部件访问。定义 16 位的寄存器的结构叫做 X，8 位的寄存器结构叫做 h。

因此，要设置自己的 REGS 类型结构 (假设它为 inregs)，可以用下列语法指定一 16 位寄存器：

inregs.x.ax (或 bx, cx 等)。

可以用下列语法指定一 8 位寄存器：

inregs.h.ah (或 al, bh, bl 等)。

结构类型 SREGS 类似于 REGS，但它指定的是段寄存器 ES, CS, SS 和 DS。因此，如定义一个 SREGS 类型的叫 segreggs 的结构，能用下列语法指定段寄存器：

segreggs.ds (或 es, ss, cs, 等)

这些类型被定义在与 Microsoft C 编译程序共存的 DOS.H 文件中。要对这些定义进行访问，只需在程序的编译过程中设有 DOS.H 文件，如下：

```
#include <dos.h>
```

这些联合的结构类型提供了需要设置和接收寄存器数值的数据结构。函数 intdos, intdosx segread 使用这些数据结构来使用 DOS 功能调用。

intdos 和 intdosx 都是靠发出中断 21H 来使用 DOS 功能调用的。当要使用的 DOS 功能调用不要求任何段寄存器作为输入参数时，使用 intdos。它从一个 REGS 联合中得到入口寄存器的数值，返回到另一个联合的出口寄存器之中。当要使用的 DOS 功能调用

要求 DS 或 ES 寄存器中一个变量时，则应使用 intdosx。它从 REGS 和 SREGS 中得到入口参数。

segread 函数不使用任何 DOS 功能，但它能在使用其它功能调用前提供需要的信息。它在 SREGS 结构中返回段寄存器的现行数值，否则在 C 中很难完成功能调用。

下面是一个使用 DOS 功能调用 02H 来显示字符的简单例子：

```
#include <dos.h>
union REGS inregs,outregs;
main( )
{
    inregs.h.ah.=0x02
    inregs.h.dl=m';
    intdos(%inregs,outregs);
    intdos(inregs,outregs);
}
```

在该例子中，功能调用号在 (0×02) 在 AH 寄存器中，要显示的字符在 DL 中。在功能调用中，要显示的字符在 DL 中。如果功能调用返回什么数值的话（本例没有），它们将在 outregs 结构中。

本章中所有 C 程序均使用这种方法使用 DOS 功能调用。如还不明白怎样使用 C 功能和怎样设置数据结构，请参看几个程序例子。Microsoft C 的文档也对此进行了详细描述。

## § 4.11 Turbo Pascal 使用功能调用

象 C 一样，Pascal 也不能直接访问寄存器或直接使用中断。可是 Turbo Pascal 为使用 DOS 功能调用提供了一个过程（叫 msdos）。象相应的 C 函数一样，msdos 发出一个中断 21H 来使用 DOS 功能调用。

msdos 过程要求设置记录作为入口参数。这个记录指定处理所有寄存器的格式如下：

```
RECORD
  CASE integer OF
    1 : (ax,bx,cx,dx,bp,si,ds,es,flags:integer);
    2 : (al,ah,bl,bh,cl,dl,dh:byte);
  END;
```

在使用 msdos 过程前，必须设置此记录中的域来指定入口寄存器中的数值，当过程结束时，该记录中的域将按照功能调用出口寄存器中的数值进行设置。

同样调用 DOS 功能 02H 显示一个字符，用 Turbo Pascal 写时如下所示：

```

PROGRAM display a-character;
TYPE
  regpack = RECORD
    CASE integer OF
      1 : (ax,bx,cx,dx,dp,si,di,ds,es,flags:integer);
      2 : (alcah,bl,bh,cl,ch,dl,dh:byte);
    END;
VAR
  regs:regpack;
BEGIN
  regs.ah:= $02;
  regsd1:= prd( m' );
  msdos(regs);
END.

```

象 C 的例子一样，功能调用号（\$02）在 AH 中，要显示的字符在 DL 中。本章中所有 Turbo Pascal 例子均应用此方法使用 DOS 功能调用。如还不明白，请看 Turbo Pascal 手册。

## § 4.12 考察和运行程序实例

如上所述，本章给出的例子将具体地指导读者编写含有 DOS 功能调用的程序。

因为大多数专用程序是用汇编语言、Pascal 或 C 编写的，所以每一个程序实例将以三种语言给出。每一种语言都必须是事先熟悉的，这样，可以集中精力学习怎样使用 DOS，而不是琢磨不会使用的编程语言。

每一个例子都是可运行的完整程序。汇编语言程序是用 Microsoft 汇编语言写的，Pascal 是用 Turbo Pascal 写的。如想汇编或编译用其它语言产品编写的程序，要首先作一些小修改。

汇编和连接汇编语言的一般方法对本章中所有程序实例都是一样的。例中，下述一组命令汇编程序 DOS04.ASM 并且产生一个可运行的.COM 文件。（DOS04.COM 是功能调用 04H 的程序实例）。

masm dos04;	(执行汇编程序)
link dos04;	(执行连接程序)
exec2bin dog04;	(产生.COM类型文件)
rename dos04.bin dos04.com	(增加.COM类型文件)

用来对典型 Turbo Pascal 程序产生可执行码的命令如下所示：

turbo (按回车)	(运行Turbo Pascal)
Y	(Yes, 包括错误信息)

O (选择, 选择菜单)  
 C (命令文件, 产生.COM文件)  
 dos04 (要编译的程序名)  
 Q (退出, 脱离系统)

用来对典型 C 例程编译连接的必须命令如下:

msc dos04, dos04 / Zp; (运行 Microsoft C 编译程序)  
 link dos04, dos04; (运行连接程序)

参阅有关汇编程序和连接程序的文档以便得到更多可用的汇编连接控制信息。利用这些信息, 可以使用程序编辑程序或字处理程序来键入本章的程序实例, 然后遵照上述的指令产生可执行代码。

### § 4.13 DOS 功能调用目录

下表给出了所有 DOS 功能调用和详细信息。

功能号		功 能
16 进制	10 进制	DOS1.X 功能调用
0H	0	终止程序
1H	1	读键盘字符并回显
2H	2	显示字符
3H	3	从辅助设备读一字符
4H	4	向设备交送一字符
5H	5	向标准打印设备送一字符
6H	6	执行直接控制台 I/O
7H	7	执行直接控制台输入但不显示
8H	8	读键盘字符但不显示
9H	9	标准输出设备上显示字符串
0AH	10	键盘字符输入缓冲区
0BH	11	检查输入设备状态
0CH	12	清除输入缓冲区并调用 I/O 操作

功能号		功 能
16 进制	10 进制	DOS1.X 功能调用
0DH	13	复位磁盘
0EH	14	选择磁盘驱动器
0FH	15	打开文件
10H	16	关闭文件
11H	17	搜索第一个匹配文件
12H	18	搜索下一个匹配文件
13H	19	删除匹配文件
14H	20	读下一顺序文件记录
15H	21	写下一个顺序文件记录
16H	22	创建文件
17H	23	文件重命名
18H	24	DOS 内部使用
19H	25	取当前盘号
1AH	26	设盘传送地址
1BH	27	取当前驱动器的 FAT 信息
1CH	28	取指定驱动器的 FAT 信息 (DOS2.0)
1DH-20H	29-32	DOS 内部使用
21H	33	读一随机记录
22H	34	写一随机记录
23H	35	取文件尺寸
24H	36	置随机记录字段
25H	37	置中断向量
26H	38	创建新程序段
27H	39	读多个随机记录
28H	40	写多个随机记录
29H	41	分析文件名
2AH	42	取日期
2BH	43	置日期
2CH	44	取时间
2DH	45	置时间
2EH	46	设置或关闭校验开关
DOS.2X 功能调用		
1CH	28	取指定驱动器的 FAT 信息
2FH	47	取盘传送地址
30H	48	取 DOS 版本号
31H	49	KEEP 终止进程保持驻留

功能号		能
16进制	10进制	DOS2X 功能调用
32H	50	DOS 内部保留
33H	51	取或置 Ctrl-Break
34H	52	DOS 内部保留
35H	53	读软盘的扇区块
36H	54	取磁盘未用空间
37H	55	内部保留
38H	56	取或设置国度信息
39H	57	MKDIR——创建子目录
3AH	58	RMDIR——删除目录
3BH	59	CHDIR——改变当前目录
3CH	60	CREAT——创建文件
3DH	61	打开文件
3EH	62	关闭一文件描述字(文件句柄)
3FH	63	读文件或设备
40H	64	写入文件或设备
41H	65	UNLINK——删除文件
42H	66	LSEEK——移动读/写指针
43H	67	CNMOD——取或置文件属性
44H	68	设备的 I/O 控制
45H	69	复制文件句柄(DUP)
46H	70	强行复制文件句柄
47H	71	取当前目录
48H	72	分配内存
49H	73	释放已分配的内存
4AH	74	SET BLOCK——修改分配的内存块
4BH	75	EXEC——装入或执行程序
4CH	76	EXIT——终止进程
4DH	77	WAIT——取子进程的返回代码
4EH	78	FIND FIRST——搜索第一个匹配文件
4FH	79	FIND NEXT——搜索下一个匹配文件
50H-53H	80-83	DOS 内部保留
54H	84	取校验开关状态
55H	85	DOS 内部保留
56H	86	改文件名
57H	87	取或置文件的日期时间

功能号		功 能
16进制	10进制	DOS3.X 功能调用
44H	68	设备的控制 I/O (新增加的子功能)
58H	88	取或置内存分配对策
59H	89	取扩展错误代码
5AH	90	创建唯一文件
5BH	91	创建新文件
5CH	92	封锁或开锁文件访问
5DH	93	DOS 保留
5EH	94	建立网络参数
5FH	95	网络重定向
60H-61H	96-97	DOS 保留
62H	98	取程序段前缀地址
44H	68	设备的控制 I/O (新增加的子功能)
58H	88	取或置内存分配对策
59H	89	取扩展错误代码
5AH	90	创建唯一文件
5BH	91	创建新文件
5CH	92	封锁或开锁文件访问
5DH	93	DOS 保留
5EH	94	建立网络参数
5FH	95	网络重定向
60H-61H	96-97	DOS 保留
62H	98	取程序段前缀地址

## DOS 功能调用 00H

终止程序

功能调用:

00H 终止程序

DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明:

此功能调用结束程序并将控制转给启动它的程序。通常，得到控制的程序是 DOS 命令处理程序 (COMMAND.COM)。但如果该程序是由另一个程序所派生的，则那个程序将得到控制。

当程序退出时，该程序使用的内存退给 DOS，这样这块内存就可用来加载和执行其

它程序。

此功能调用也将把结束、Ctrl-Break 和严重错误地址设置为该程序被启动时的数值。DOS 从程序段前缀 PSP 中查找上述在程序启动时保存在这里的数值。参见前面关于 PSP 的说明。

此功能调用关闭所有文件句柄并刷新所有文件缓冲区，如有必要，可将已写入缓冲区的部分内容记盘。但如果已改变了文件的长度则必须在使用此功能调用前关闭该文件（用 10H 功能调用或 3EH）。否则该文件的长度、日期和时间将不能正确地记录在目录中。

此功能调用与 INT20H 完成的操作相同。

入口参数：

调用前，需设置寄存器：

AH=00H 指出功能调用号。

GS 必须设置成程序PSP的段地址。

出口参数：

程序结束，控制转给 PSP 中列出的结束地址无返回数据。

参见：

31H 终止并驻留 (KEEP)

4CH 终止进程 (EXIT)

程序实例：

下列汇编语言例子在屏幕上显示信息，然后使用功能调用 00H 结束程序。

这里没有列出相应的 Pascal 和 C 程序，因为这些编译程序自动地产生 DOS 结束功能调用其作为出口代码的一部分。因此，Pascal 和 C 程序从不需要直接使用此功能调用。

汇编语言实例：

```
; ;TERMINATE A PROGRAM(00H)
;
code segment public
assume cs:code,ds:code
        org      100h
start:  jmp      begin
msg     db       'Terminating via DOSfunction request 00',  '$'
begin:  mov ax,cs           ;set up ds
        mov ds,ax
        mov dx,offset msg      ;address of message
        mov ah,09h              ;display string function request
        int 21h                ;call DOS
        mov ah,00h              ;terminate a pgm function request
        int 21h                ;call DOS
```

```
code ends  
end start ;start is the entry point
```

## DOS 功能调用 01H

读键盘字符并回显

功能调用:

01H 从键盘读一字符并在屏幕显示

DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明:

此功能调用从标准输入设备上获得一个字符（一般是键盘）或等待接收一个有效字符，并在标准输入设备上显示该字符（一般是显示器）。

DOS2.0（以上版本）允许重新定向标准输入输出设备。这个重新定向使应用此功能调用的程序能从数据文件读取数据，而不从键盘读取数据。显示输出可以重新定向于其它设备，如数据文件或打印机。

此功能调用检查输入字符看是否是 Ctrl-Break；如是，该功能调用使用中断 23H 来启动 Ctrl-Break 处理程序（Ctrl-Break 句柄）。

如在使用此功能调用时无有效输入字符，则该功能调用等待直到有一个有效输入字符（直到操作员按一键）。如果不希望程序无限等待，应先使用功能调用 0BH（检查标准输入状态）来看是否有有效字符。

入口参数:

调用前，需设置寄存器:

AH = 01H 指出功能调用号。

出口参数:

返回后，设置寄存器为:

AL = 读到的键盘字符。

其它要求:

如果从功能调用返回的值是 00，则该字符是一扩展 ASCII 字符（如功能键、ALT 键或光标键）。必须再次使用此功能调用来得到扩展的 ASCII 字符。第二次返回的是该字符的扩展部分。附录 A 列出了普通和扩展的 ASCII 字符。

参见:

06H 直接控制台 I/O。

07H 直接控制输入且不显示。

08H 从键盘读一字符且不显示。

09H 检查输入设备状态。

程序实例:

下列的三个例子应用功能调用 01H 从键盘读取字符并在显示器显示。该字符显示两次，一是 DOS，另一次是由程序实例，并显示相应的信息来指明输入的字符是普通的还

是扩展的 ASCII 字符。程序实例用 Ctrl-Z 终止。

### Assembly Language Usage Example:

```
; :KEYC? INPUT WITH CS, DS, ES
;
;CODE SECTION: CS
ASSUME CS:CODE, DS:DATA
        DB      0dh,0ah, '$'
begin: JMP    begin
msg1     DB      'Press any key to test,Ctrl-Z to end.',$0dh,$0ah,'$'
msg2     DB      'Normal Ascii Character',0dh,0ah,'$'
msg3     DB      'Extended Ascii Character',0dh,0ah,'$'
begin: MOV    AX,CS ;set up ds
        MOV    DX,AX
        MOV    DX,OFFSET msg1 ; set up to display message
        MOV    AH,09H ; display string function request
        INT    21H ; call DOS
next:MOV AH,01H ; read keyboard and echo funct req
        INT 21H ; call DOS
        MOV CX,OFFSET msg2 ; set up to display message
        CMP AL,0 ; check if extended ascii char
        JNE disp ; no,take jump
        MOV AH,01H ; read keyboard and echo funct req
        INT 21H ; call DOS
        MOV CX,OFFSET msg3 ; set up to display message
disp:    JE done ; quit if control-z
        MOV DL,AL ; return char in dl for next funct
        MOV AH,02H ; display character funct request
        INT 21H ; call DOS
        MOV DX,CX ; get proper message
        MOV AH,09H ; display string function request
        INT 21H ; call DOS
        JMP next ; get next charzcter
done:   MOV AH,00H ; terminate program funct request
        INT 21H ; call DOS
```

```
code ends;           end of code segment
      end start;       start is the entry point
```

**Pascal Usage Example:**

```
{Keyboard Input With Echo ($01)}
```

```
PROGRAM get_key_echo;

CONST
  dos_get_key_echo = $01;
  control_z = $1a;

TYPE
  regpack = RECORD
    CASE integer OF
      1: (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
      2: (al,ah,b1,bh,c1,ch,d1:byte);
    END;

  VAR
    regs: regpack;
BEGIN
  WITH regs DO
    BEGIN
      al := 0;
      writeln('Press any key to test,Ctrl-Z to exit.');
      WHILE (al < > control_z) DO
        BEGIN
          ah := dos_get_key_echo;
          msdos(regs);
          IF al < > control_z THEN
            BEGIN
              IF al < > 0 THEN
                writeln(chr(al),'Normal Ascii Character')
              ELSE
                BEGIN
                  ah := dos_get_key_echo;
                  msdos(regs);
                END;
            END;
        END;
    END;
END.
```

```

        writeln(chr(al),'Extended Ascii Character')
    END;
END;
END;
END;
END.

```

### C Usage Example:

```

/*
 * Keyboard Input with Echo(0x01)
 */

#include <dos.h>

union REGS inregs,outregs;

main()
{
    printf("Press any key to test,Ctrl-z to end.\n");
    for (;;) {
        inregs.h.ah = 0x01;
        intdos(&inregs ,&outregs);
        if (outregs.h.al == 0xal)
            break;
        if (outregs.h.al != 0)
            printf("% c Normal Ascii Character\n",outregs.h.al);
        else {
            inregs.h.al = 0x01;
            intdos(&inregs,&outregs);
            printf("%C Extended Ascii Character\n",outregs.h.al);
        }
    }
}

```

### DOS 功能调用 02H

显示字符

功能调用:

**02H 在标准输出设备上显示一个字符。**

**DOS 版本:**

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

此功能调用在标准输出设备（一般是显示器）上显示一个字符。它在现行光标位置上显示一个字符，然后光标前进一个字符位置。

如果显示的字符是退格字符，此功能调用使光标向左移动一个字符的位置，但是并不抹掉那个字符。

当此功能调用在行末（屏幕右边缘）显示字符时，光标移动到下一行的左边缘。如果光标是在屏幕右下角，则光标移动时，屏幕向上滚动一行。

此功能调用在显示一个字符后检查它是否是 Ctrl-Break。如操作员键入 Ctrl-Break，则功能调用中断 23H，启动相应的 Ctrl-Break 处理程序。

在 DOS2.0（以上版本）中，可以重新定向标准输出设备。重新定向可使应用此功能调用的程序不仅可以向屏幕写出，也可以在打印机和磁盘文件上写出。

**入口参数:**

调用前需设置寄存器:

AH = 02H 指定功能调用号。

DL = 显示的 ASCII 字符代码。

**出口参数:**

无。

**参见:**

09H 在标准输出设备上显示一字符串。

**程序实例:**

下列三个程序实例使用功能调用 02H 显示一串信息。一次显示一个字符。

#### **Assembly Language Usage Example:**

```
; ; DISPLAY A CHARACTER (02H)
;
code segment public
assume cs:code,ds:code
    org      100h
start: jmp      begin
msg       db      'This message was displayed with DOS function 02h.'
msglen  equ      $-msg
begin:  mov      ax,cs ;set up ds
        mov      dx,ax
        mov      cx,msglen;   length of message in characters
```

```

    mov     si,0;           start with first character
    mov     02h;            display character function reguess
next:int 21h;          call DOS
    inc    si;             get the next character
    loop next;            continue until no more characters
    mov ah,00h ;           terminate program funct request
    int 21h ;              call DOS
code    ends ;           end of code segment
    end start;           start is the entry point

```

**Pascal Usage Example:**

{Keyboard Input With Echo (\$02) }

PROGRAM display\_character;

CONST

```

dos_display_character = $02;
control_z = $1a;
```

TYPE

```

regpack = RECORD
  CASE integer OF
    1: (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
    2: (al,ah,bl,bh,cl,ch,dl:byte);
  END;
```

VAR

```

regs : regpack ;
i : integer ;
msg : string [80]
```

BEGIN

```
msg := 'This message was displayed with DOS function 02h.';
```

WITH regs DO

```

FOR i := 1 TO length (msg) DO
  ah := dos_get_key_echo;
  dl := ord(msg[i]);           msdos(regs);
END;
```

END.

### C Usage Example:

```
/*
 * Keyboard Input with Echo(0x01)
 */

#ifndef include <dos.h>

union REGS inregs,outregs;

main()
{
    int i;

    for (i = 0; msg[i] != 0; ++i) {
        inregs.h.ah = 0x02;
        inregs.h.dl = msg(i);
        intdos (&inregs ,&outregs);
    }
}
```

### DOS 功能调用 03H

从辅助设备读入一字符

#### 功能调用:

03H 从辅助输入设备读一字符

#### DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

#### 说明:

此功能调用从标准辅助输入设备读一字符 (AUX 或缺省为 COM1)。DOS2.0 以上版本允许重定向标准辅助设备为其它设备如 COM2。

此功能调用不提供缓冲区且不是中断驱动。因此，当程序使用此功能调用时，在进行其它操作前必须等待直到返回一个有效值。然后它必须马上处理这个字符并发出下一个字符的功能调用。依赖于字符到达的速率，程序处理这个字符的有效时间将是很有限的。

在启动系统时，DOS 设置第一个辅助口 COM1 为 2400 波特，无校验，一个停止位和 8 个数据位。可以使用 DOS 的 MODE 命令来改变此设定。

#### 入口参数:

调用前，需设置寄存器:

AH = 03 指定功能调用号。

出口参数：

返回后，设置寄存器为：

AL = 从辅助设备读的字符。

错误条件：

此功能调用不返回错误码。使用 BIOS 中断 14H 可获得关于错误条件更多信息。

参见：

BIOS 中断 14H-RS-232 串行口 I/O。

程序实例：

下述三个程序实例用功能调用 03H 从辅助设备（COM1）读一字符。在屏幕上显示该字符。Ctrl-Z 终止程序。

#### Assembly Language Usage Example:

```
; ; DISPLAY A CHARACTER (02H)
;
code segment public
assume cs:code,ds:code
    org    100h
start: mov     ax,cs           ;set up ds
        mov     ds,ax
next:  mov     ah,03h          ;ayxukuart uboyt sybctuib request
        int    21h           ;call DOS
        inc    si             ;get the next character
        cmp    al,1ah          ;is char an end of file marker?
        je     done            ;then all done
        mov    ah,02h          ;display character on std output
        mov    dl,al            ;character that has been input
        int    21h           ;call DOS
        jmp    next            ;get next character
done:   mov    ah,00h          ;terminate program funct request
        int    21h           ;call DOS
        end start;           ;ends; end of code segment
                                ;start is the entry point
```

#### Pascal Usage Example:

```
{Read a Character From Auxiliary Device ($ 03) }
```

```

PROGRAM read_arx;

CONST
  dos_read_aux = $03;
  control_z = $1a;

TYPE
  regpack = RECORD
    CASE integer OF
      1: (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
      2: (al,ah,bl,bh,cl,ch,dl:byte);
    END;

VAR
  regs: regpack;

BEGIN
  WITH regs DO
    BENIN
    al := 0;
    WHILE (al < > control_z) DO
      BENIN
      dl := dos_read_aux;           msdos(regs);
      IF al < > control_z THEN
        write(al);
      END;          END;
  END.

```

#### C Usage Example:

```

/*
 * Keyboard Input with Echo(0x01)
 */

```

```

#include <dos.h>

union REGS inregs,outregs;

main()
{

```

```

for (;;) {
    intregs.h.ah = 0x03;
    intdos (&inregs, &outregs);
    if (outregs.h.al == -0x1a) break;
    else printf("%C", outregs.h.al);
}
}

```

## DOS 功能调用 04H

向辅助设备发送一字符

功能调用:

04H 向辅助输出设备发送一字符

DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明:

此功能调用向标准辅助设备 (AUX 或缺省为 COM1) 写一个字符。DOS2.0 (及以上版本) 允许重定向标准辅助设备为其它设备, 如 COM2。

类似于功能调用 03H, 此功能调用不提供缓冲区且不是中断驱动。

在启动系统时, DOS 设置第一个辅助口为 2400 波特, 无校验, 一个停止位和 8 个数据位。用 DOS 的 MODE 命令可以改变这个设定。

入口参数:

调用前, 需设置寄存器:

AH = 04H 指定功能调用号。

DL = 在此寄存器中放置要向辅助设备发送的ASCII字符。

出口参数:

无。

错误条件:

此功能调用不返回错误码。用 BIOS 中断 14H 可获得有错误条件的更多信息。

参见:

BIOS 中断 14H-RS-232 串行口 I/O。

程序实例:

下列三个实例用功能调用 04H 向辅助设备 (COM1) 发送从键盘输入的字符。  
Ctrl-Z 终止程序。

### Assembly Language Usage Example:

```

;
; SEND CHARACTER TO AUXILIARY DEVICE (04H)

```

```

;

code segment public
assume cs:code,ds:code

        org      100h
start: jmp      begin
msg     db      'Press any key to test,Ctrl-Z to end.'
        db      0dh,0ah,'$'
begin: mov      ax,cs           ;set up ds
        mov      ds,ax
        mov      dx,offset msg   ;address of message
        mov      ah,09h           ;display string function request
        int      21h             ;call DOS
next:  mov      ah,01h           ;read keyboard and echo funct req
        int      21h             ;call DOS
        cmp      al,0             ;check if extended ascii character
        jne      send             ;no,jump to send the char off
        mov      ah,01h           ;read keyboard and echo funct req
        int      21h             ;call DOSsend:  cmp al,1ah ;check for control-z
        je      done              ;qui if control-z
        mov      dl,al             ;return char in dl for next req
        mov      ah,04h           ;send char to aux device funct req
        int      21h             ;call DOS
        jmp      next              ;get next character
done:  mov      ah,00h           ;terminate program funct request
        int      21h             ;call DOScode ends ; end of code segment
        end start;               start is the entry point

```

### Pascal Usage Example:

{Send a Character To Auxiliary Device (\$04)}

PROGRAM read\_arx;

#### CONST

```

dos_get_key_echo = $01;
dos_read_aux = $04;
control_z = $1a;

```

```

TYPE
  regpack = RECORD
    C1SE integer OF
      I: (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
      R: (al,ah,bl,bh,cl,ch,dl:byte);
  END;

VAR
  regs: regpack;
BEGIN
  WITH regs DO
    BEGIN
      al := 0;
      WHILE (al < > control_z) DO
        BEGIN
          ah := dos_get_key_echo;
          msdos(regs);
        END;
      al := dos_get_key_echo;
      msdos(regs);
    END;
    IF al < > control_z THEN
      BEGIN
        dl := al;
        ah := dos_get_key_echo;
        msdos(regs);
      END;
    END;
  END;
END.

```

C Usage Example:

```

/* */
* Send a Ctrl-Z To Auxiliary Device (0x04)
*/

```

#include <dos.h>

union REGS r,regs;

```

main()
{
    printf("Press any key to test,Ctrl-z to end.\n");
    for(;;) {
        intcgs.h.ah = 0x01;
        intdos (&inregs ,&outregs);
        if (outregs.h.al == 0x1a)
            break;
        if (outregs.h.al == 0) {
            intcgs.h.ah = 0x01;
            intdos (&inregs ,&outregs);
        }
        intcgs.h.dl = outregs.h.al;
        intcgs.h.ah = 0x04;
        intdos (&inregs ,&outregs);
    }
}

```

## DOS 功能调用 05H

向标准打印设备发送一字符

**功能调用:**

05H 向标准打印设备发送一个字符。

**DOS 版本:**

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

此功能调用向标准打印设备 (PRN 或缺省为 LPT1) 发送一字符。DOS2.0 (以上版本) 允许重定向标准打印设备, 如 LPT2。

此功能调用几乎同功能调用 02H (显示一字符) 相同, 只是该字符不进显示器而送打印机。

**入口参数:**

调用前, 需设置寄存器:

AH=05H 指出功能调用号。

DL= 要由打印机发送的字符ASCII码。

**出口参数:**

无。

**错误条件:**

DOS 自动检查诸如打印机准备就绪、纸完和打印缓冲区满等错误条件。当错误条件出现时，功能调用会把错误信息（一般是显示器）上显示相应的错误信息。

程序实例：

下列三个实例用功能调用 05H 向标准打印设备发送一固定信息。

**Assembly Language Usage Example:**

```
;  
;DISPLAY A CHARACTER (02H)  
;  
code segment public  
assume cs:code,ds:code  
    org      100h  
start:  jmp      begin  
msg     db 0db,0ah      This message was displayed with DOS function 05h.'  
msglen equ     $-msg  
begin:  mov      ax,cs;set up ds  
        mov      ds,ax  
        mov      cx,msglen ;length of message in characters  
        mov      si,0       ;start with first character  
        mov      ah,05h     ;print character function req  
next:   mov      dl,msg[si] ;next char in dl for function 05h  
        int      21h       ;call DOS  
        inc      si       ;get the next character  
        loop     next      ;continue until no more characters  
        mov      ah,00h     ;terminate program funct request  
        int      21h       ;call DOS  
code    ends      ;end of code segment  
end start    ;start is the entry point
```

**Pascal Usage Example:**

```
{ Send Character to Standard Printer Device ( $ 05 ) }
```

```
PROGRAM print_character;
```

```
CONST
```

```
dos_display_character = $05;
```

```

TYPE
  regpack = RECORD
    CASE integer OF
      1: (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
      2: (al,ah,bl,bh,cl,ch,dl,bh1,cl1);
    END;

VAR
  regs: regpack;

VAR
  msg: string [80];
  i: integer;

BEGIN
  msg := 'This message was displayed with DOS function 05h.';
  + chr ($0d) + chr ($0a);
  WITH regs DO
    FOR i := 1 TO length (msg) DO
      BEGIN
        ah := dos_get_key_echo;
        dl := ord(msg[i]);
        msdos(regs);
      END;
  END.

```

#### C Usage Example:

```

/*
 * Send a Character to Standard Printer (0x05)
 */

#include <dos.h>

union REGS inregs,outregs;

char msg[] = "This message printed with DOS function 05h.\n"
main()
{
  int l;

```

```
for (i = 0;msg[i] != 0;++i) {
    integs.h.ah = 0x02;
    integs.h.dl = msg[i];
    intdos (&inregs ,&outregs);
}
}
```

## DOS 功能调用 06H

### 执行直接控制台 I / O

#### 功能调用:

06H 执行直接控制台 I / O

#### DOS 版本:

1.0, 1.1, 2.0, 3.0, 3.2, 3.3

#### 说明:

此功能调用提供下述两种能力之一：向标准输出设备（一般是显示器）发送字符和从标准输入设备（一般是键盘）接收一个字符。DOS2.0 以上版本允许重定向标准输入和输出设备。此重定向程序使用此功能调用对其他设备（如文件）进行读或写。

此功能调用不检查字符是否是 Ctrl-Break 或 Ctrl-Prtsc 字符且对这些字符不进行特殊处理。

当此功能调用从键盘接收一字符时，它在显示器上显示该字符。

如果在使用功能调用接收字符时没有有效字符，该功能调用等待，直到有一个有效字符（直到操作员按一键）。如果不希望程序无限等待，应当先使用功能调用 0BH（检查输入设备的状态）来查看是否有一个有效字符。

#### 入口参数:

调用前，需设置存储器：

AH = 06H 指出功能调用号。

DL 置为 FFH，功能调用从键盘读一字符，若为其它数值，功能调用在控制台屏幕上显示字符。

#### 出口参数:

如从键盘读收一字符（在使用功能调用前将 DL 设成 FFH），则当返回后，设置为：

AL = 如果键盘上有一个准备好的字符，零标志被清除且此寄存器将含有该字符。

如无有效字符，零标志将被置位，这种情况下 AL 为 0。

#### 其它要求:

在读一字符时，如从此功能调用返回的值是 00H，则该字符是一个扩展 ASCII 字符（如功能键，ALT 键或光标键）。必须再次使用此功能调用以确定该扩展 ASCII 字符。第二个调用返回的是该字符的扩展部分。附录 A 列出了正常的扩展的 ASCII 字符。

#### 参见:

- 01H 读键盘字符并回显。
- 07H 直接控制台输入但不回显。
- 08H 读键盘字符但不回显。
- 0BH 检查输入设备状态。

**程序实例：**

下列三个实例用功能：调用 06H 从键盘接收输入。该程序在屏幕上边疆显示一串信息。当按除 ‘S’ 之外的任何键时，该程序在屏幕上显示.....INTERRUPTED..... ‘键入 ‘S’ 结束程序。

**Assembly Language Usage Example:**

```
;
;DIRBCT CONSOLE I/O (06H)
;
code    segment public
assume cs:code,ds:code
        org      100h
start: jmp      begin
msg1   db      0dh,0ah,
        db      'press"s"to stop,any other key to interrupt.',
        db      '$'
msg2   db      ...INTERRUPTED..., '$'
begin: mov      ax,cs           ;set up ds
        mov      ds,ax
next:  mov      dx,offset msg1 ;set up to display message
        mov      ah,09h          ;display string function request
        int      21h             ;call DOS
        mov      dl,0ffh          ;read character from keyboard
        mov      ah,06h          ;direct console I/O funct request
        int      21h             ;call DOS
        jnz      keyhit          ;zf = 0 if key has been hit
        jmp      next             ;get next key
keyhit:cmp     al, 's'          ;time to terminate?
        jc      done              ;terminate
        move    dx,offset msg2 ;set up to display message
        move    ah,09h          ;display string function request
        int      21h             ;call DOS
code    ends             ;end of code segment
        end      start            ;start is the entry point
```

**Pascal Usage Example:**

{ Direct Console I/O (\$06) }

```

PROGRAM console_io ;
CONST
    dos_console_io = $06 ;
TYPE
    regpack = RECORD
        CASE integer OF
            1: (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
            2: (al,ah,bl,bh,cl,ch,dl,dh:byte);
        END ;
VAR
    regs : regpack ;
BEGIN
    WITH regs DO
        BEGIN
            al := 0 ;
            WHILE (chr(al)<> 's') DO
                BEGIN
                    writeln ;
                    write
                        (' Press "s" to stop ,any other key to interrupt.')
                    delay(30);
                    ah:=dos console io
                    dl:=$ff;
                    msdos(regs);
                    if ((al<>0) and (chr(al)<> 's')) then
                        write(' ...interrupted... ');
                END;
            END;
        END.

```

### C Usage Example:

```

/*
 * Direct Console I/O (0X06)
 */
#include <dos.h>
union REGS inregs,out=egs;
main()
{
    for (;;) {

```

```

printf("\nPress 's' to stop,any other key to interrupt."
inregs.h.ah = 0x06;
inregs.h.d1 = 0xff;
intdos(inregs,outregs);
if (outregs.h.al == 's')

    break;
else if (outregs.h.al != 0)
    printf("...INTERRUPTED...");
}
}

```

## DOS 功能调用 07H 执行直接控制台输入但不回显

**功能调用:**

07H 执行直接控制台输入但不回显

**DOS 版本:**

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

此功能调用等待标准输入设备（一般是键盘）输入一字符。当字符输入时，它被接收且不在显示器显示。此功能调用不检查字符是否是 Ctrl-Break 或 Ctrl-PrtSc 且不对这些字符做特殊处理。带 Ctrl-Break 检查、执行相；同功能的，参见功能调用 08。

如果想从终端接收一字符但既不想回显在显示器上又不想回显在现行光标位置上，此功能调用将很有用。

DOS2.0 以后的版本允许重定向标准输入设备。该重定向使应用此功能调用的程序能不从键盘而从其它设备或文件接收字符。如在使用此功能调用时没有准备好字符，该功能调用等待直到有字符准备好为止（直到操作员按一键）。如不希望程序无限等待，应当使用功能调用 0BH（检查输入设备状态）来查看是否为有效字符。

**入口参数:**

调用前，需设置寄存器：

AH=07H 指出功能调用号

**出口参数:**

返回后，设置寄存器为：

AL= 从键盘读的字符。

**其它要求:**

如此功能调用返回的数值是 00H，该字符是一个扩展 ASCII 字符。必须再次使用本功能调用来获取扩展的 ASCII 字符。第二次调用返回字符的扩展部分。附录 A 列出普通和扩展的 ASCII 字符。

参见:

- 01H 读键盘字符并回显.
- 06H 直接控制台 I/O.
- 08H 读键盘字符但不回显.
- 0BH 检查输入设备状态.

程序实例:

下列三个实例完成控制台输入，但不回显。此程序可使用户键入长达 39 个字符的口令，只有完成这个口令之后，才显示这秘密信息。

**Assembly Language Usage Example:**

```
; ; DIRECT CONSOLE INPUT WITHOUT ECHO (07H)
;
code segment public
assume cs:code,ds:code
    org      100h
start: jmp      begin
msg1   db      'Type a secret message,followed by enter',
        db      '0dh,0ah, $'
msg2   db      'YOUR SECRET MESSAGE WAS: ', '$'
secret  db      40 dup(?)
begin: mov      ax,cs           ;set up ds
        mov      ds,ax
        mov      dx,offset msg1  ;set up to display message
        mov      ah,09h          ;display string function request
        int      21h             ;call DOS
        mov      cx,39            ;max size of secret message array
        xor      bx,bx            ;index into secret message array
next:   mov      ah,07h          ;direct con input w/o echo req
        int      21h             ;call DOS
        cmp      al,0dh            ;carriage return typed?
        je      disp              ;yes,reveal the secret message
        mov      secret[bx],al     ;no,add key to secret
        inc      bx                ;inc index into secret msg
        loop     next              ;get next keystroke
disp:   mov      secret[bx], '$' ;terminate the secret message
        mov      dx,offset msg2  ;set up to display message
        mov      ah,09h          ;display string function request
        int      21h             ;call DOS
```

```

        mov     dx,offset secret ;set up to display the secret
        mov     ah,09h           ;display string function request
        int     21h              ;call DOS
        mov     ah,00h           ;terminate program funct request
        int     21h;             call DOS
code    ends                ;end of code segment
        end start              ;start is the entry point

```

Pascal Usage Example:

{Direct Console Input Without Echo(\$07)}

```

PROGRAM console input noecho ;

CONST
  dos console input noecho = $07 ;
TYPE
  regpack = RECORD
    CASE integer OF
      1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer)
      2 : (al,ah,b1,bh,c1,ch,d1,dh:byte) ;
  END;
VAR
  regs : regpack ;
  i : integer ;
  secret :string[40] ;
BEGIN
  i := 1 ;
  secret := '' ;
  writeln( ' Type a secret message, followed by enter.' ) ;
  WITH regs DO
    BEGIN
      al := 0      WHILE (i<40) AND (al<>$0d) DO
        BEGIN          ah :=dos console input noecho ;
          msdos(regs);
          IF al <> $0d THEN
            secret :=secret + chr(al);
            i := i+1 ;
        END ;
      write ( ' YOUR WECRET MESSAGE WAS : ',secret) ;
    END
  
```

```

        END ;
END.

C Usage Example:

/*
 * direct Console Input Without Echo (0x07)
 */
#include <dos.h>
union REGS inregs,outregs;
main( )
int i;
char secret[40];
printf("Type a secret message,followed by enter.\n");
for (i=0;i<=38;++i) {
    inregs.h.ah = 0x07;
    intdos(inregs,outregs);
    if (outregs.h.al == 0xd)
        break;
    else
        secret[i] = outregs.h.al;
}
secret[i] = 0 ;
printf("YOUR SECRET MESSAGE WAS: %s",secret);
}

```

## DOS 功能调用 08H

读键盘字符但不回显

**功能调用:**

08H 从键盘读一字符但不回显

**DOS 版本:**

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

此功能调用从标准输入设备（一般是键盘）获得一字符或等待一有效字符。不像功能调用 01H，此功能调用不在标准输出设备上（一般是显示器）显示该字符。

此功能调用检查字符是否是 Ctrl-Break，如是，该功能调用发出中断 23H 启动 Ctrl-Break 处理程序。要完成同样的操作且不检查 Ctrl-Break，参见功能调用 07H。

同功能调用 07H 一样，如想从键盘接收字符而既不想回显在显示屏上又不想回显在显示屏上又不想回显在现行光标位置，则该功能调用是很有用的。

DOS2.0 以上版本允许重定向标准输入设备。重定向应用此功能调用的程序能不从键

盘，而从文件或其它设备上读字符。

如在使用该功能调用时无准备好的字符，则该功能调用等待直到有准备好的字符。如不希望程序无限等待，应当使用功能调用 0BH（检查输入设备的状态）来检查是否有准备好的字符。

#### 入口参数：

调用前，需设置寄存器：

AH=08H 指出功能调用号。

#### 出口参数：

返回后，设置寄存器为：

AL= 从键盘得到的字符。

#### 其它要求：

如功能调用返回值是 00H，该字符是一个扩展 ASCII 字符，必须再次使用功能调用来获得扩展 ASCII 字符。第二次调用返回的是该字符扩展部分。

#### 参见：

- 01H 读键盘字符并回显
- 06H 执行直接控制台 I/O。
- 07H 直接控制台输入但不回显。
- 0BH 检查输入设备状态。

#### 程序实例：

下列三个实例用功能调用 08H 从键盘读字符并且不回显。程序可从键盘读长达 39 字符的秘密信息。只有信息完整时才显示。

#### Assembly Language Usage Example:

```
; ; READ KEYBOARD CHARACTER WITHOUT RCHO (08H)
; ; code segment public
assume cs:code,ds:code
org 100h
start: jmp begin
msg1 db      Type a secret message,followed by enter.','
        db      0dh,0ah, '$'
msg2 db      YOUR SECRET MESSAGE WAS', '$'
secret db      40 dup(?)
begin: mov ax,cs           ;set up ds
       mov ds,ax
       mov cx,39           ;max size of secret message
       xor bx,bx           ;index into secret message array
next:  mov ah,08h          ;read keyboard wityout echo req
```

BASIC

```

int    21h          ;call DOS
cmp    al,0dh        ;carriage return typed?
je     disp          ;yes,reveal the secret message
mov    secret[bx],al ;no,add key to secret
inc    bx            ;increment index into secret msg
loop   next          ;get next keystroke
disp: mov   secret [bx], '$' ;terminate the secret message
      mov   dx,offset msg2 ;set up to display message
      mov   ah,09h          ;display string function request
      int   21h            ;call DOS
      mov   dx,offset secret ;set up to display the secret
      mov   ah,09h          ;display string function request
      int   21h            ;call DOS
      mov   ah,00h          ;terminate program function req
      int   21h            ;call DOS
code  ends           ;end of code segment
end   start          ;start is the entry point

```

**Pascal Usage Example:**

```

{Read Keyboark Character Without Echo ($08)}
PROGRAM keyboard input noccho ;
CONST
dos keyboard input noccho = $08 ;
TYPE
rcgpack = RECORD
  CASE integer OF
    1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer)
    2 : (al,ah,bl,bh,cl,ch,dl,dh:byte) ;
  END;
VAR
regs : rcgpack ;
i : integer ;
secret :string[40] ;
BEGIN
  i := 1 ;
  secret := '' ;
  writeln( Type a secret message, followed by enter.) ;
  WITH regs DO
  BEGIN
    al := 0 ;

```

```

        WHILE (i<40) AND (al<>$0d) DO
            BEGIN
                ah := dos keyboard input noecho ;
                msdos(regs);
                IF al <> $0d THEN
                    secret := secret + chr(al);
                    i := i+1 ;
                END ;
                write ( YOUR SECRET MESSAGE WAS : ,secret) ;
            END ;
        END.

```

#### C Usage Example:

```

/*
 * Read Keyboard Character Without Echo (0x08)
 */
#include <dos.h>
union REGS inregs,outregs;
main( )
{
    int i;
    char secret[40];
    printf("Type a secret message,followed by enter.\n");
    for (i=0;i<=38;++i) {
        inregs.h.ah = 0x08;
        intdos(inregs,outregs);
        if (outregs.h.al == 0xd)
            break;
        else
            secret[i] = outregs.h.al;
        secret[i] = 0 ;
    }
    printf("YOUR SECRET MESSAGE WAS: %s",secret);
}

```

#### 功能调用 09H

标准输出设备上显示字符串

#### 功能调用:

09H 在标准输出设备上显示字符串

DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

#### 说明:

此功能调用在标准输出设备上显示一字符串。它在屏幕现行光标位置开始显示字符，然后光标前进一个位置。为了决定字符串尾，该功能调用认为字符串最后一个字符是 ASCII '\$' (24H)，它不显示 \$。因为它是串尾的一约定，不应使用此功能调用显示含有 \$ 字符的串如商业数据。如串中有一字符是退格符，它使光标左移一个字符位。

当功能请求在行尾显示字符时，光标移到下一行左边。如光标在右下角，则光标移动时屏幕上滚。

此功能调用回显字符后检查 Ctrl-Break。如操作员键入 Ctrl-Break，该功能调用使用中断 23H 启动 Ctrl-Break 处理程序。

DOS2.0 以上版本允许重定向标准输出设备，重定向使用此功能调用的程序不只写屏幕，还可向文件或其它设备写如打印机和磁盘文件。

#### 入口参数:

调用前需设置寄存器：

AH = 09H 指定功能调用号

DS: DX 此寄存器对指向内存中字符串之首。此功能调用显示字符直到遇见终止字符串的 \$ 字符 (ASCII 24H)，\$ 字符并不显示。

#### 出口参数:

无。

#### 参见:

02 H 标准输出设备上显示字符。

#### 程序实例:

下列三个实例用功能调用 09H 在屏幕上显示从 \$ 为结尾的字符串。本书中列出的大部分汇编语言实例用此功能调用显示输出。

#### Assembly Language Usage Example:

```
; ; DISPLAY STRING (09H)
;
code segment public
assume cs:code,ds:code
    org      100h
start: jmp      begin
msg     db       'Hi! This is a dollar sign terminated string.', '$'
begin: mov      ax,cs           ;set up ds
        mov      ds,ax
        mov      dx,offset msg   ;set up to display message
        mov      ah,09h          ;display string function request
        int      21h             ;call DOS
```

```

        mov     ah,00h      ;terminate program funct request
        int     21h      ;call DOS
code    ends      ;end of code segment
        end     start      ;start is the entry point

```

**Pascal Usage Example:**

{Display String (\$09)}

PROGRAM display\_string;

CONST

dos display string = \$09;

TYPE

regpack = RECORD

CASE integer of

1: (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);

2: (al,ah,bl,bh,cl,ch,dl,dh:byte);

END;

VAR

regs : regpack;

msg : string [50];

BEGIN

WITH regs do

BEGIN

msg := 'Hi! This is a dollar sign terminated string.\$' ;

ah := dos display string;

ds := seg(msg[1]);

dx := ofs(msg[1]);

msdos(regs);

END;

END;

END.

**C Usage Example:**

/\*

\* Display String (0x09)

\*/

#include <dos.h>

union REGS inregs,outregs;

char msg[ ] = "Hi! This is a dollar sign terminated string.\$";

main( )

{

## DOS 功能调用 0AH 键盘字符存输入缓冲区

### 功能调用:

0AH 键盘输出字符送输入缓冲区.

### DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

### 说明:

此功能调用从键盘读字符在屏幕上显示并将字符送入缓冲区便于以后的处理. 将字符送缓冲区, 直到 ENTER 键入时或当键入的字符总数是缓冲区所允许的总数少 1 (最后一个字符留给 ENTER) 时为止.

如果缓冲区填到只比它允许的最大字数少 1 时, 功能调用将略去此后的字符并响铃直到接收 ENTER 键为止.

在键入 Enter 键之前, 键盘操作员可以使用任何标准 DOS 行编辑字符来修正错误或改变输入缓冲区的字符.

此功能调用应用 Ctrl-Break 字符进行检查, 如键入了 Ctrl-Break 字符, 该功能调用将使用中断 23H 来启动 Ctrl-Break 处理程序.

### 入口参数:

调用前, 需设置寄存器:

DS: DX 此寄存器对必须指向从键盘接收字符的输入缓冲区首址. 此缓冲区必须比接收的字符数多两个字节, 因为前两个字节存储缓冲区信息.

缓冲区第一个字节必须非零并必须指定缓冲区能接收的字符数 (缓冲区总长减 2).

缓冲区的第二个字节由该功能调用填写, 指出实际读入到缓冲区中的字符字节数.

缓冲区其余字节为键盘键入的字符所保留.

### 出口参数:

返回后, 输入缓冲区的第二个字节设置为接收的字符个数. 此数不包括一般用作接收的最终字符 Enter (ASCII 码 0DH).

缓冲区中后续字数 (长度由第二个字节所指定) 含有键盘输入的字符. 跟着这些字符的是 Enter 字符 (ASCII 码 0DH).

### 参见:

01H 读键盘字符并回显.

06H 直接控制台 I/O.

07H 直接控制台输入但不回显.

08H 读键盘字符但不回显.

### 程序实例:

下面三个实例使用功能调用 0AH 从键盘收集最多达 40 字符的字符串并将它们送入一输入缓冲区. 在 Enter 按键后, 程序即显示这些字符.

#### Assembly Language Usage Example:

```

; BUFFERED KEYBOARD INPUT (0AH)
;

code segment public
assume cs:code,ds:code
    org      100h
start : jmp     bbegin
buffer   db      40          ;number of chars in the buffer
numchar  db      0           ;number of chars actually read
chars    db      41 dup (?) ;character part of buffer
msg1     db      Type anything, followed by enter',0dh,0ah,  '$'
msg2     db      0dh,adh,  Contents of the kbd input buffer: ',  

         db      0dh,0ah,  '$'
begin:  mov     ax,cs        ;set up ds
        mov     ds,ax        ;to same segment as cs
        mov     dx,offset msg1 ;set up to display message
        mov     ah,09h        ;display string function request
        int     21h          ;call DOS
        mov     dx,offset buffer ;address of buffer
        mov     dx,offset buffer ;address of buffer
        mov     ah,0ah        ;buffered keyboard input funct
        int     21h          ;call DOS
        mov     dx,offset msg2 ;set up to display message
        mov     ah,09h        ;display string function request
        int     21h          ;call DOS
        xor     bx,bx        ;clear bx
        mov     bl,numchar    ;number of chars read into buffer
        mov     chars[bx], '$' ;add terminator for char string
        mov     dx,offset chars ;set up to display the buffer
        mov     ah,09h        ;display string function request
        int     21h          ;call DOS
        mov     ah,00h        ;terminate program funct request
        int     21h          ;call DOS
code    ends        ;end of code segment
end     start        ;start is the entry point

```

**Pascal Usage Example:**

```

{Buffered Keyboard Input ($0a) }
PROGRAM buffered_keyboard_input ;
  CONST

```

```

dos buffered keyboard input = $0a ;

TYPE
  regpack = RECORD
    CASE integer OF
      1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer)
      2 : (al,ah,b1,bh,c1,ch,d1,dh:byte) ;
    END;
  buffer = RECORD
    max size : byte ;
    characters : string[42] ;
  ND ;
VAR
  regs : regpack ;
  buf : buffer ;
BEGIN
  WITH regs DO
    BEGIN
      buf.max size := 40 ;
      writeln( Type anything,followed by enter.) ;
      ah := dos buffered keyboard input ;
      ds := seg(buf) ;
      dx := ofs(buf) ;
      msdos(regs) ;
      writeln ;
      writeln( Contents of the keyboard input buffer:) ;
      write(buf.characters) ;
    END ;
  END.

```

### **C Usage Example:**

```

/*
 * Buffered Keyboard Input (0X)
 */
#include <dos.h>
union REGS inregs,outregs;
struct buffer {
  char max size;
  char max read;
  char characters[41];
};

```

```

struct buffer buf = {40,0};
main( )
{
    printf("Type anything,followed by enter.\n");
    inregs.h.ah = 0x0a;
    inregs.x.dx = (int) buf;
    intdos(inregs,outrcgs);
    printf("\ncontents of the keyboard input buffer:\n");
    buf.characters[buf.max read] = 0;
    printf("%s",buf.characters);
}

```

## 功能调用 0BH

### 检查输入设备状态

#### 功能调用:

0BH 检查输入设备状态

#### DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

#### 说明:

此功能调用检查标准输入设备来看是否有一字符等待接收，如操作员键入 Ctrl-Break 键，该功能调用使用中断 23H 来启动 Ctrl-Break 处理程序。

如果不希望程序无限期地等待键盘输入字符，则此功能调用是非常有用的。在希望输入时不发诸如 01H, 06H, 07H, 08H 等键盘输入功能调用，而可以先使用此功能调用来看是否有有效字符。如果操作员从上次检查以来一直未按任何键，程序可以继续其它操作。

#### 入口参数:

调用前，需设置寄存器：

AH=0BH 指出功能调用号。

#### 出口参数:

返回后，设置寄存器为：

AL 此寄存器的值数指出该功能调用的状态。

00H 无输入字符等待接收。

FFH 有一有效字符。

#### 参见:

01H 读键盘字符并回显。

06H 直接控制台 I/O。

07H 直接控制台输入但不回显。

08H 读键盘字符但不回显。

### 程序实例:

下列三个实例使用功能调用 0BH 来查看是否有一字符从键盘输入。在等待字符时，程序打印出一个标准信息。当字符有效时（一键被按下），程序打印信息并结束。

#### Assembly Language Usage Example:

```
; ; CHECK INPUT STATUS (0BH)
;
code segment public
assume cs:code,ds:code
    org      100h
start: jmp      begin
msg1   db      'Nope, no characters in type-ahead buffer.',0dh,0ah, '$'
msg2   db      'yes,there is a character in type-ahead buffer.',0dh,0ah, '$'
begin: mov      ax,cs           ;set up ds
        mov      ds,ax           ;to same as cs
next:  mov      ah,0bh          ;check input status funct request
        int      21h             ;call DOS
        cmp      al,0             ;is character available?
        jne      done             ;yes,all done
        mov      dx,offset msg1  ;address of no message
        mov      ah,09h            ;display string function request
        int      21h             ;call DOS
        jmp      next             ;try again until Ctrl-Break
done:   mov      dx,offset msg2  ;yes,setup yes message
        mov      ah,09h            ;display string function request
        int      21h             ;call DOS
        mov      ah,00h            ;terminate program funct request
        int      21h             ;call DOS
code   ends             ;end of code segment
end     start             ;start is the entry point
```

#### Pascal Usage Example:

```
{ Check Input Status ($0b) }
PROGRAM check_input_status ;
CONST
    dos_check_input_status = $0b;
TYPE
```

```

regpack = RECORD
  CASE integer OF
    1 :(ax,bx,cx,dx,bp,si di ds,es,flags:integer) ;
    2 :(al,ah,bl,bh,cl,ch,dl,ch:byte);
  END;
VAR
  regs : regpack ;
  keyhit : boolean ;
BEGIN
  Keyhit := false ;
  with regs do
    WHILE keyhit = false DO
      BEGIN
        delay(10);
        ah := dos check input status ;
        msdos(regs) ;
        IF al = 0 THEN
          writeln(  Nope,no characters in type-ahead buffer.)
        ELSE
          BEGIN
            writeln
            ( Yes,there is a character in type-ahead buffer. );
            keyhit := true ;
          END ;
      END ;
END ;

```

**C usage Example:**

```

/*
 * Check Input Status (0x0b)
 */
#include <dos.h>
union REGS inregs,outregs;
main( )
{
  for (;;) {
    inregs.h.ah = 0x0b;
    intdos(inregs,outregs);
    if (outregs.h.al == 0)
      printf("Nope,no characte , in type-ahead buffer.\n");
}

```

```

    else {
        printf
        ("Yes,there is a character in type-ahead buffer.\n");
        break;
    }
}
}

```

## DOS 功能调用 0CH 清除输入缓冲区并调用 I/O 操作

### 功能调用:

0CH 清除输入缓冲区并调用 I/O 操作。

### DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

### 说明:

此功能调用清除标准输入缓冲区中所有字符并使用下列功能调用之一:

01H 读键盘字符并回显。

06H 直接控制台 I/O。

08H 读键盘字符但不回显。

07H 直接控制台输入但不回显。

0AH 键盘字符存输入缓冲区。

此功能调用允许废弃操作员的预操作，放弃缓冲区中预键入的所有字符。然后它使用另一个功能调用来读（或写）控制台中字符。如这新调用是读，则操作员必须键入新的字符作为应答。

### 入口参数:

调用前，需设置寄存器:

AH=0CH 指出功能调用号。

AL 在此寄存器中，必须在清除输入缓冲区后放置要使用的功能调用号。允许的数值为:

01H 读键盘字符并回显。

06H 直接控制台 I/O。

07H 直接控制台输入但不回显。

08H 读键盘字符但不回显。

0AH 键盘字符存输入缓冲区。

### 出口参数:

此功能调用的出口参数领事于清除标准输入缓冲区后使用的功能调用，参见那些功能调用的描述。

### 参见:

- 01H 读键盘字符并回显。
- 06H 直接控制台 I/O。
- 07H 直接控制台输入但不回显。
- 08H 读键盘字符但不回显。
- 0AH 键盘字符存输入缓冲区。

**程序实例：**

下列三个实例使用功能调用 0CH 来清除输入缓冲区并调用一 I/O 操作。该程序从键盘读字符直到按键 Ctrl-Z。

**Assembly Language Usage Example:**

```
;
; CLEAR INPUT BUFFER AND INVOKE FUNCTION(0CH)
;

code segment public
assume cs:code,ds:code
        org      100h
start: jmp      obegin
msg     db       'Press any key to test,Ctrl-Z to end.', '$'
begin: mov      ax,cs           ;set up ds
        mov      ds,ax           ;to same as cs
        mov      dx,offset msg   ;address of message
        mov      ah,09h          ;display string function request
        int      21h             ;call DOS
next:  mov      al,01h          ;keyboard input with echo
        mov      ah,0ch           ;but flush input buffer first
        int      21h             ;call DOS
        cmp      al,1ah           ;check for control-z
        jne      next             ;quit if control-z
        mov      ah,00h           ;terminate program funct request
        int      21h             ;call DOS
code   ends             ;end of code segment
        end      start           ;start is the entry point
```

**Pascal Usage Example:**

```
{Clear Input Buffer and Invoke Function ($0c) }
PROGRAM clear input buffer ;
CONST
  dos get key echo = $01 ;
  dos clear input buffer = $0c :
```

```

control z = $1a ;
TYPE
  regpack = RECORD
    CASE integer OF
      1: (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer) ;
      2: (al,ah,bl,bh,cl,ch,dl,dh:byte) ;
    END ;
VAR
  regs : regpack ;
BEGIN
  writeln( 'Press any key to test ,Ctrl-z to end.' );
  with regs do
    BEGIN
      al := 0
      WHILE (AL< > control-z ) DO
        BEGIN
          regs.ah := dos clear input buffer ;
          regs.al := dos get key echo ;
          msdos(regs) ;
        END;
    END;
END;

```

### C Usage Example:

```

/*
 * Clear Input Buffer and Invoke Function (0x0c)
 */
#include <dos.h>
union REGS inregs,outregs;
main( )
{
  printf("Press any key to test,Ctrl-z to end.\n");
  outregs.h.al = 0;
  while (outregs.h.al != 0xa) {
    inregs.h.al = 0x01;
    inregs.h.ah = 0x0c;
    intdos(inregs,outregs);
  }
}

```

## DOS 功能调用 0DH

### 复位磁盘

功能调用:

0DH 磁盘复位

DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明:

在磁盘 I/O 过程中 DOS 维护内存缓冲区（每个同磁盘扇区大小）来暂时保存读写磁盘的信息。靠维护这些缓冲区，DOS 减少实际访问磁盘的次数（盘操作相当慢）。可以一次完成对一个扇区的读写操作。当向磁盘写信息时，DOS 简单地将信息存入内存直到缓冲区满，然后以一次操作将整个缓冲写到盘上。

此功能调用强制清除所有文件缓冲区数据并将所有写到缓冲区中信息写到盘上希望肯定所有信息确实被写盘时，此调用是很有用的。例如，Ctrl-Break 中断处理程序中使用此调用是个好主意。

但此功能调用不能代替关闭文件。如果此调用或任何其它 I/O 操作改变了文件大小，记录在盘目录中的文件大小将不修正。使用目录中含有更新后的文件尺寸信息的唯一办法是用功能调用 10H 关闭该文件。

入口参数:

调用前，需设置寄存器:

AH=0DH 指出功能调用号。

出口参数:

无。

程序实例:

下列三个实例使用功能调用 0DH 来复位磁盘，文件缓冲区被写入盘。

#### Assembly Language Usage Example:

```
; ; DISK RESET (0DH)
;
code segment public
assume cs:code,ds:code
org 100h
start: mov ax,cs      ;set up ds
       mov ds,ax    ;to same as cs
       mov ah,0dh    ;disk reset function request
       int 21h      ;call DOS
       mov ah,00h    ;terminate program funct request
       int 21h      ;call DOS
```

```
code    ends          ;end of code segment
      end      start      ;start is the entry point
```

**Pascal Usage Example:**

```
{ Disk Reset ($0d) }
PROGRAM disk reset;
CONST
  dos disk reset = $0d ;
TYPE
  regpack = RECORD
    CASE integer OF
      1: (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer) ;
      2: (al,ah,bl,bh,cl,ch,dl,dh:byte);
    END ;
VAR
  regs : regpack ;
BEGIN
  regs.ah := dos disk reset ;
  msdos(regs) ;
END.
```

**C Usage Example:**

```
/*
 * Disk Reset (0x0d)
 */
#include <dos.h>
union REGS inregs,outregs;
main( )
{
  inregs.h.ah = 0x0d;
  intdos(inregs,outregs);
}
```

## DOS 功能调用 0EH

选择磁盘驱动器

功能调用:

0EH 选择磁盘驱动器

DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

### 说明:

此功能调用选择一个磁盘驱动器作为缺省驱动器。它也可以返回系统中所有软盘和硬盘驱动器的总数。

### 入口参数:

调用前, 需设置寄存器:

AH = 0EH 指出功能调用号

DL 指定缺省驱动器的数值。可能的数值是:

0A 驱动器

1B 驱动器

### 出口参数:

返回后, 设置寄存器为:

AL 此寄存器指出驱动器总数(软盘和硬盘)。如系统只含有一个驱动器, 此功能调用在 AL 寄存器中返回 2。这个 2 使程序认为系统有两个逻辑驱动器, 尽管只有一个物理驱动器。

在 DOS3.0 以后版本中, 最小返回数值是 5。

要决定物理驱动器个数, 应使用 BIOS 中断 11H。

### 其它要求:

为保持和 DOS 将来版本兼容性, 不能根据此功能调用返回的信息确认哪一个驱动器字母是有效的。即如 AL 返回数值是 6, 不应当认为驱动器 A 到 F 都有效设备。

### 参见:

BIOS 中断 11H—确定设备。

### 程序实例:

下列三个实例使用功能调用 0EH 来选择 A 盘作为现行驱动器。

#### Assembly Language Usage Example:

```
; ;SELECT DISK (0EH)
;
code segment public
assume cs:code,ds:code
org 1,00h
strar: mov ax,cs           ;set up ds
       mov ds,ax           ;to same as cs
       mov d1,0            ;drive A
       mov ah,0eh           ;select drive function request
       int 21h              ;call DOS
       mov ah,00h           ;terminate program function reg
       int 21h              ;call DOS
```

```
code    ends          ;end of code segment
       end start      ;start is the entry point
```

#### Pascal Usage Example:

```
{ Select disk ($0Eh) }
PROGRAM select disk ;
CONST
dos select disk = $0e;
drive a = 0;

TYPE

regpack = RECORD
  CASE integer OF
    1: (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
    2: (al,ah,bl,bh,cl,ch,dl,dh:byte);
END;
```

```
VAR
regs : regpack ;
BEGIN
  regs.ah := dos select disk ;    regs.dl := drive a ;
  msdos(regs) ;
END.
```

#### C Usage Example:

```
/*
 * Select Disk (0x0e)
 */
#include <dos.h>
#define DRIVE A 0x00

union REGS inregs,outregs;
main()
{
  inregs.h.ah = 0x0e;
  inregs.h.dl = DRIVE A;
```

```
intdos(&inregs,&outregs);
}
```

## DOS 功能调用 0FH

### 打开文件

#### 功能调用:

0FH 打开文件

#### DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

#### 说明:

此功能调用打开文件以备读写。

此功能调用的入口参数是一个指向未打开文件 FCB 的首址的指针。此功能调用设置 FCB 中若干字段。参见第 60 页 FCB 说明。

如果 FCB 中驱动器码 0 (指向缺省驱动器) 该功能调用将此码变为实际驱动 (1 是 A, 2 是 B 等)。它使程序改变缺省驱动器而无须干扰该文件以后的操作。

此功能调用设置 FCB 中的现行块字段 (偏移 0CH) 为 0。它也设置偏移 0EH (逻辑记录大小) 为系统缺省值 80H (十进制 128)。它根据从该文件目录中获得的信息设置文件大小及文件最后写入的日期时间。

如认为缺省的记录尺寸太小，则应修改 FCB 偏移 0EH 和 0FH 处的值。如想把文件用作顺序读写，必须设置现行相对记录字段 (偏移 20H)。如想把文件用作随机读写，则必须设置相对随机记录字段 (偏移 21H 到 24H 处)。必须在批开文件之后，开始任何 I/O 操作之前进行 FCB 修改。

如在 DOS3.0 以上版本下用这个功能调用打开文件，该文件以兼容模式打开。这是一个网络访问时的重要的共享模式。它允许多个程序以它们希望的任何操作访问同一文件。如果它们均以兼容模式打开该文件。参见功能调用 3DH (打开文件) 的描述可了解不同共享模式的更多信息。

#### 入口参数:

调用前，需设置寄存器:

AH = 0FH 指出功能调用号。

DS: DX 此寄存器对指向当前未打开文件的FCB。

#### 出口参数:

返回后，设置寄存器为:

AL 此功能调用以输入给FCB中的文件名来搜索目录。设置此寄存器为下列数值来指出是否找到文件:

00H 该文件有效

FFH 该文件无效

#### 参见:

01H 关闭文件

- 16H 创建文件
- 3DH 打开文件
- 3EH 关闭文件描述字 (文件句柄)

**程序实例:**

下列三个实例使用功能调用 0FH 在缺省驱动器中现行目录下打开一名为 TESTFILE.ASC 的文件。

**Assembly Language Usage Example:**

```

;
; OPEN FILE (0FH)
;

code segment public
assume cs:code,ds:code

        org 100h
start:  jmp begin
        db 0, testfileasc',26 dup(?)
msg1:   db 'Opening testfile.asc.',0dh,0ah,  '$'
msg2:   db 'testfile.asc NOT FOUND !',0dh ,0ah,  '$'
msg3:   db 'testfile.asc FOUND !',0dh,0ah,1 '$'
msg4:   db 'Closing testfile.asc.',0dh,0ah,  '$'

begin:  mov ax,cs
        mov ds,ax
        mov dx,offset msg1
        mov ah,09h
        int 21h
        mov dx,offset fcb
        mov ah,0fh
        int 21h
        cmp al,0
        jc found
        mov dx,offset msg2
        mov ah,09h
        int 21h
        jmp done
found:  mov dx,offset msg3
        mov ah,09h
        int 21h
        mov dx,offset msg4
;
```

```

        mov ah,09h      ;display string function request
        int 21,h        ;call DOS
        mov dx,offset fcb ;address of file control block
        mov ah,10h      ;close file function request
        int 21,h        ;call DOS
done:   mov ah,00h      ;terminate program funct request
        int 21,h        ;call DOS
code    ends          ;end of code segment
        end start;     start is the entry point

```

**Pascal Usage Example:**

{ Open File (\$0F) }

PROGRAM open\_file,

CONST

```

dos_open_file = $0F;
dos_close_file = $10;
default_drive = 0;

```

TYPE

regpack = RECORD

CASE integer OF

```

1: (ax,bx,cx,bp,si,di,ds,es,flags:integer);
2: (al,ah,bl,bh,cl,ch,dl,dh:byte);

```

END;

fcb = RECORD

```

drive number : byte;
file name : ARRAY [1..11] OF char;
misc : ARRAY [1..26] OF byte;

```

END;

VAR

regs : regpack;

this\_file fcb :fcb;

BEGIN

```

WITH regs DO
  BEGIN
    this file fcb.drive number := default drive ;
    ghis file fcb.file name := testfileasc' ;
    writeln( Opening testfile.asc.' );
    ah := dos open file ;
    ds := seg(this file fcb) ;
    dx := ofs(this file fcb) ;
    msdos(regs) ;
    IF al < > 0 THEN
      writeln( testfile.asc NOT FOUND ! )
    ELSE

      BEGIN
        writeln( testfile.asc FOUND ! ),
        writeln( closing testfile.asc.' );

        ah := dos close file ;
        ds := seg(this file fcb) ;
        dx := ofs(this file fcb) ;
        msdos(regs) ;
      END ;
    END ;
  END.

```

### C Usage Example:

```

/*
 * Open File (0x0f)
 */

#ifndef include <dos.h>
union REGS inregs,outregs;
struct fcb {
  char drive num ;
  char filename[11];
  char misc[26];
};
struct fcb this file fcb = {0 "testfileasc"};

```

```

main()
{
    printf("Opening testfile.asc.\n");
    inregs.h.ah = 0x0f;
    inregs.x.dx = (int)&this file fcb;
    intdos (&inregs,&outregs);
    if (outregs.h.al != 0)
        printf("testfile.asc NOT FOUND !\n");
    printf("testfile.asc FOUND !\n");
    printf("Closing testfile.asc.\n");
    inregs.h.ah = 0x10;
    inregs.x.dx = (int) &this file fcb;
    intdos (&inregs,&outregs);
}
}

```

## DOS 功能调用 10H

### 关闭文件

**功能调用:**

10H 关闭文件

**DOS 版本:**

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

功能调用关闭文件。确保文件修改后相应的文件目录项都已更新。

最后一次写文件后，必须使用此功能调用。

关闭文件以前，DOS 检查当前目录以确保文件资料在目录中的位置与它在被打开时的目录中的位置相同。利用这一方法，通过 DOS 可检查磁盘是否更换，如果目录不存在，或存在但所处位置不同，则 DOS 认为磁盘已被更换，并且不更新文件。否则更新文件的目录项，并刷新此文件有关的缓冲区。

**入口参数:**

调用前，需设置寄存器：

AH = 10H 指出功能调用号

DS: DX 该寄存器对指向已打开文件的文件控制块 (FCB)。参见第60页 FCB 的格式。

**出口参数:**

返回时，设置寄存器为：

AL 使用此功能调用时，功能调用搜索当前的磁盘目录，寻找与人口时指定的 FCB 中的文件名相同的文件。如果找到了，与 FCB 中所放的位

置做比较。搜索的结果在 AL 中置成以下二值之一：

00H 在与 FCB 所记录的相同位置找到了文件。在这种情况下，功能调用根据 FCB 的状态更新目录。

FFH 文件没找到，或找到了但与 FCB 所记录的位置不同。在这种情况下，功能调用认为磁盘已被更换。当程序接收到此码时，应当进行一定的处理，例如给一请用户插入正确磁盘的信息，然后程序再做此功能调用。

参见：

3EH..—关闭文件

程序实例：

下面三个实例都是用功能调用 10H 关闭在缺省盘的当前目录中名为 TESTFILE.ASC 的文件。

#### Assembly Language Usage Example:

```
;  
;OPEN FILE (10H)  
;  
code segment public  
assume cs:code,ds:code  
  
org 100h  
start: jmp begin  
fcb db 0, testfileasc',26 dup(?)  
msg1 db 'Opening testfile.asc.',0dh,0ah, '$'  
msg2 db 'testfile.asc NOT FOUND !',0dh,0ah, '$'  
msg3 db 'testfile.asc FOUND !',0dh,0ah, '$'  
msg4 db 'Closing testfile.asc.',0dh,0ah, '$'  
begin: mov ax,cs  
       mov ds,ax  
       mov dx,offset msg1  
       mov ah,09h  
       int 21h  
       mov dx,offset fcb  
       mov ah,0fh  
       int 21h  
       cmp al,0  
       jc found  
       mov dx,offset msg2  
       mov ah,09h  
       int 21h
```

```
;set up ds  
;to same as cs  
;address of opening message  
;display string function request  
;call DOS  
;address of file control block  
;open file function request  
;call DOS  
;does file exist?  
;yes,take jump  
;address of not found message  
;display string function request  
;call DOS
```

	<b>jmp done</b>	;wrap it up
opened:	<b>mov dx,offset msg3</b>	;address of found message
	<b>mov ah,09h</b>	;display string function request
	<b>int 21,h</b>	;call DOS
	<b>mov dx,offset msg4</b>	;address of closed message
	<b>mov ah,09h</b>	;display string function request
	<b>int 21,h</b>	;call DOS
	<b>mov dx,offset fcb</b>	;address of file control block
	<b>mov ah,10h</b>	;close file function request
	<b>int 21,h</b>	;call DOS
	<b>cmp al,0</b>	;was close successful?
	<b>je close</b>	;yes,take jump
	<b>mov dx,offset msg2</b>	;address of not found message
	<b>mov ah,09h</b>	;display string function request
	<b>int 21,h</b>	;call DOS
	<b>jmp done</b>	;wrap it up
opened:	<b>mov dx,offset msg5</b>	;address of found message
	<b>mov ah,09h</b>	;display string function request
	<b>int,21h</b>	;call DOS
done:	<b>mov ah,00h</b>	;terminate program funct request
	<b>int 21,h</b>	;call DOS
code	<b>ends</b>	;end of code segment
	<b>end start;</b>	start is the entry point

#### Pascal Usage Example:

```
{ Close File ($10h) }
```

```
PROGRAM open_file;
```

#### CONST

```
dos__open__file = $0f;
dos__close__file = $10;
default__drive = 0;
```

#### TYPE

```
regpack = RECORD
  CASE integer OF
```

```

    1: (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer) ;
    2: (al,ah,bl,bh,cl,ch,dl,dh:byte);
END ;

fcb = RECORD
  drive number : byte ;
  file name : ARRAY [1..11] OF char ;
  misc : ARRAY [1..26] OF byte ;
END ;

VAR
  regs : regpack ;
  this file fcb :fcb ;

BEGIN
  WITH regs DO
    BEGIN
      this file fcb.drive number := default drive ;
      ghis file fcb.file name :=  'testfileasc' ;
      writeln(  'Opening testfile.asc.' );
      ah := dos open file ;
      ds := seg(this file fcb) ;
      dx := ofs(this file fcb) ;
      msdos(regs) ;
      IF al < > 0 THEN
        writeln(  'testfile.asc NOT FOUND !' )
      ELSE
        BEGIN
          writeln(  'testfile.asc FOUND !' );
          writeln(  'closing testfile.asc.' );

          ah := dos close file ;
          ds := seg(this file fcb) ;
          dx := ofs(this file fcb) ;
          msdos(regs) ;
        END ;
    END ;

```

#### C Usage Example:

```

/*
 * Open File (0x10)
 */

#include <dos.h>
union REGS inregs,outregs;
struct fcb {
    char drive num ;
    char filename[11];
    char misc[26];
};
struct fcb this file fcb = {0 "testfileasc"};

main()
{
    printf("Opening testfile.asc.\n");
    inregs.h.ah = 0x0f;
    inregs.x.dx = (int) &this file fcb;
    intdos (&inregs,&outregs);
    if (outregs.h.al != 0)
        printf("testfile.asc NOT OPENED !\n");
    else {
        printf("testfile.asc OPENED !\n");
        printf("Closing testfile.asc.\n");
        inregs.h.ah = 0x10;
        inregs.x.dx = (int) &this file fcb;
        intdos (&inregs,&outregs);
        if (outregs.h.al != 0)
            print("testfile.asc NOT CLOSED !\n");
        else
            printf("testfile.asc CLOSED !\n");
    }
}

```

## DOS 功能调用 11H 搜索第一个匹配文件

功能调用:

## 11H 搜索第一个匹配文件

DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明:

此功能搜索当前目录，找与给定 FCB 中列出的文件名相匹配的第一文件。FCB 中所列的文件名可以含通配符“？”，它表明可能有几个文件与此名相匹配。参见第 60 页 FCB 的格式。

如果此功能调用入口时给定的 FCB 是扩充 FCB，则根据文件的属性按下列说明搜索文件：

- 如果设置 FCB 的属性字节为 0，则功能调用只搜索普通文件。不返回卷标、子目录、隐含或系统文件等项目。
- 如果设置 FCB 的属性字节为隐含文件、系统文件或子目录，则功能调用搜索所有普通文件和指定的特文件以获得相匹配的文件。
- 属性字节的三位都设置，则功能调用搜索所有普通文件、隐含文件、系统文件和子目录。
- 如果设置属性字节为卷标，则功能调用只搜索卷标，不再搜索普通文件。

参见第 60 页属性字节的描述。使用 DOS2.1 以前版本，则不能搜索子目录和卷标属性的文件。

如果搜索成功，功能调用返回第一个匹配文件的有效 FCB 到磁盘传送地址 (DTA) 的起始位置。可以使用此 FCB 打开和调用文件。

功能调用修改入口时给定的 FCB，设置部分保留信息，以便功能调用 12H 搜索下一个匹配文件。如果打算搜索另外的匹配文件，那么在使用功能调用 12H 寻找另一个匹配文件以前不能用此 FCB 做磁盘操作。

入口参数:

调用前，需设置寄存器:

AH = 11H 指出功能调用号。

DS: DX 指向未打开的文件控制块 (FCB)。

出口参数:

返回后，设置为:

AL 指示搜索结果，表示如下:

00H 功能调用找到了匹配文件，在 DTA 设置了表示此文件的 FCB。

FFH 匹配文件没找到。功能调用没设 FCB。

DTA 如果功能调用找到了匹配文件，则为此文件构造一未打开的有效 FCB，返回到盘传送地址的起始处。功能调用在 FCB 的文件名字段放入文件名和在驱动器号域放入文件的实际驱动器号 (1 = 驱动器 A, 2 = 驱动器 B, 等)。所产生的未打开的 FCB 为 32 字节长，与文件的 32 字节目录项相同。

如果功能调用入口给定的 FCB 为扩充 FCB，则返回的 FCB 也是扩充 FCB。如果给定的是标准 FCB，则功能调用返回标准 FCB。

DS: DX 此寄存器对指向功能调用入中给定的FCB。功能调用修改FCB使得功能调用 12H 能搜索下一个匹配文件。

参见：

- 12H.. 搜索下一个匹配文件。
- 4EH.. 找第一个匹配文件 (FIND FIRST)。
- 4FH.. 找下一个匹配文件 (FIND NEXT)。

程序实例：

下面三个实例都是用功能调用 11H 在缺省驱动器的当前目录中搜索名为 TESTFILE.ASC 的文件。

Assembly Language Usage Example:

```
; ;SEARCH FOR FIRST MATCHING FILE (11H)
;
code segment public
assume cs:code,ds:code
org 100h
start: jmp begin
fcb db 0, testfileasc',26 dup (?)
msg1 db 'Searching for testfile.asc.',0dh,0ah, '$'
msg2 db 'testfile.asc NOT FOUND !,0dh,0ah, '$
msg3 db 'testfile.asc FOUND !,0dh,0ah, '$
begin: mov ax,cs           ;set up ds
       mov ds,ax           ;to same as cs
       mov dx,offset msg1 ;address of searching message
       mov ah,09h           ;display string function request
       int 21h              ;call DOS
       mov dx,offset fcb   ;address of file control block
       mov ah,11h           ;search first file function req
       int 21h              ;call DOS
       cmp al,0             ;does file exist?
       je found             ;yes,take jump
       mov dx,offset msg2   ;address of not found message
       mov ah,09h           ;display string function request
       int 21h              ;call DOS
       jmp done              ;wrap it up
found:  mov ax,offset msg3 ;address of found message
       mov ah,09h           ;display string function request
```

```

        int 21h           ;call DOS
code    ends           ;end of code segment
end      start          ;start is the entry point

```

**Pascal Usage Example:**

```

{ Search For First Matching File ($ 11) }

PROGRAM search first;
CONST
  dos search first = $ 11;
  default drive = 0;
TYPE
  fcb = RECORD
    drive number : byte;
    file name : ARRAY [1..11] OF char;
    misc : ARRAY [1..26] OF byte;
  END;
  regpack = RECORD
    CASE integer OF
      1: (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
      2: (al,ah,bl,bh,cl,ch,dl,dh:byte);
    END;
  VAR
    regs : regpack;
    this file fcb :fcb;
  BEGIN
    WITH regs DO
      BEGIN
        this file fcb.drive number := default drive;
        this file fcb.file name := testfileasc';
        writeln( Searching for testfile.asc.');
        ah := dos search first;
        ds := seg(this file fcb);
        dx := ofs(this file fcb);
        msdos(regs);
        IF al < > 0 THEN

```

```

        writeln( testfile.asc NOT FOUND !)
ELSE
        writeln( testfile.asc FOUND! );
END;
END.

```

### C Usage Example:

```

/*
 * Search For First Matching File (0x11)
 */

#include <dos.h>

union REGS inregs,outregs;
struct fcb {
    char drive num;
    char filename[11];
    char misc[26];
};
struct fcb this file fcb = {0,"testfileasc"},

main()
{
    printf("Searching for testfile.asc.\n");
    inregs.h.ah = 0x11;
    inregs.x.dx = (int) &this file fcb;
    intdos(&inregs,&outregs);
    if (outregs.h.al != 0)
        printf("testfile.asc NOT FOUND!\n");
    else
        printf("testfile.asc FOUND!\n");
}

```

### DOS 功能调用 12H

搜索下一个匹配文件

#### 功能调用:

12H 搜索下一个匹配文件

**DOS 版本:**

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

此功能调用认为已经使用功能调用 11H 找到了第一个匹配文件，而在当前目录中搜索下一个与 FCB 所列文件名匹配的文件。参见第 60 页 FCB 的格式。匹配标准与功能调用 11H 相同。

使用功能调用 11H 搜索第一个文件时，功能调用把信息放入了给定 FCB 的保留区。允许功能调用 12H 能继续搜索。因此在使用功能调用 11H 和功能调用 12H 之间不得使用有关此 FCB 的磁盘操作。

**入口参数:**

调用前，需设置寄存器：

AH=12H 指出功能调用号

DS: DX 指向未打开文件的文件控制块 (FCB)。入口指定的 FCB 必须被功能调用 11H 使用过。

**出口参数:**

返回后，设置寄存器为：

AL 指出搜索结果，表示如下：

00H 功能调用找到了匹配文件，设置 FCB 指示此文件。

FFH 匹配文件没找到，功能调用没设 FCB。

DTA 如果功能调用找到了匹配文件，则为此文件构造一未打开的有效 FCB，返回到盘传送地址的起始处。功能调用在 FCB 的文件名字段放入文件名和在驱动器号域放入文件的实际驱动器号 (1=驱动器 A, 2=驱动器 B, 等)。

如果功能调用入口给定的 FCB 为扩充 FCB，则返回的 FCB 也是扩充 FCB。如果给定的是标准 FCB，则功能调用返回标准 FCB。

DS: DX 此寄存器对指向功能调用入口给定的 FCB。功能调用修改 FCB 使得功能调用 12H 能搜索下一个匹配文件。

**参见:**

11H.. 搜索第一个匹配文件。

4EH.. 找第一个匹配文件 (FIND FIRST)。

4FH.. 找下一个匹配文件 (FIND NEXT)。

**程序实例:**

下面三个实例都是用功能调用 12H 在缺省驱动器的当前目录中搜索名为 TESTFILE.??? 的文件。由于文件名中含有通配符，在找到所有匹配文件前程序可继续使用功能调用 12H。

**Assembly Language Usage Example:**

;

```

; SEARCH FOR NEXT MATCHING FILE (12H)
;
code segment public
assume cs: code,ds:code
    org    100h
start:   jmp    begin
fcb      db     0, testfileasc',26 dup (?)
msg1     db     Searching for files matching testfile.???.'
msg2     db     Match found!',0dh,0ah, $'
begin:   mov ax,cs           ;set up ds
         mov ds,ax          ;to same as cs
         mov dx,offset msg1 ;address of searching message
         mov ah,09h          ;display string function request
         int 21h             ;call DOS
         mov dx,offset fcb  ;address of file control block
         mov ah,11h          ;search first file function request
         int 21h             ;call DOS
         cmp al,0            ;does file exist?
         je found            ;no,take jump
         mov dx,offset msg2 ;address of match found message
         mov ah,09h          ;display string function request
         int 21h             ;call DOS
next:    mov dx,offset fcb  ;address of file control block
         mov ah,12h          ;search next function request
         int 21h             ;call DOS
         cmp al,0            ;does file exist?
         jne done             ;no,take jump
         mov dx,offset msg2 ;address of match found message
         mov ah,09h          ;display string function request
         int 21h             ;call DOS
         jmp next            ;see if any more matches
done:   mov ah,00h          ;terminate program funct request
         int 21h             ;call DOS
code    ends               ;end of code segment
end start          ;start is the entry point

```

#### Pascal Usage Example:

{ Search For First Matching File (\$ 12) }

```

PROGRAM search first ;
CONST
  dos search first = $11 ;
  dos search next = $12 ;
  default drive = 0 ;
TYPE
  regpack = RECORD
    CASE integer OF
      1: (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer) ;
      2: (al,ah,bl,bh,cl,ch,dl,dh:byte);
    END ;
  fcb = RECORD
    drive number : byte;
    file name : ARRAY [1..11] OF char ;
    mis : ARRAY [1..26] OF byte ;
  END ;
VAR
  regs : regpack ;
  this file fcb :fcb ;

BEGIN
  WITH regs DO
    BEGIN
      this file fcb.drive number := default drive ;
      this file fcb.file name := testfile???' ;
      writeln( Searching for files matching testfile.???' );
      ah := dos search first ;
      ds := seg(this file fcb) ;
      dx := ofs(this file fcb) ;
      msdos(regs);
      WHILE (al = 0) DO
        BEGIN
          writeln( Match found ! )
          ah := dos search next ;
          ds := seg(this file fcb) ;
          dx := ofs(this file fcb) ;
          msdos(regs),
        END;
    END;

```

```
    END;  
END.
```

#### C Usage Example:

```
/*  
 * Search For Next Matching File (0x11)  
 */  
  
#include <dos.h>  
  
union REGS inregs,outregs;  
struct fcb {  
    char drive num;  
    char filename[11];  
    char misc[26];  
};  
struct fcb this file fcb = {0,"testfile???"};  
  
main()  
{  
    printf("Searching for files matching testfile.???.\n");  
    inregs.h.ah = 0x11;  
    inregs.x.dx = (int) &this file fcb;  
    intdos(&inregs,&outregs);  
    while (outregs.h.al == 0) {  
        printf("Match found !\n");  
        inregs.h.ah = 0x12;  
        inregs.x.dx = (int) &this file fcb;  
        intdos(&inregs,&outregs);  
    }  
}
```

#### DOS 功能调用 13H 删除匹配文件

功能调用:  
13H 删除匹配文件

DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

该功能调用删除与给定 FCB 中所列文件名匹配的所有文件。FCB 中所列的文件名可以含通配符“？”，它表明可能有几个文件与此名相匹配。

**入口参数:**

调用前，需设置寄存器：

AH=13H 指出功能调用号。

DS: DX 指向一未打开的文件控制块，参见第60页FCB的格式。

**出口参数:**

返回后，设置寄存器为：

AL 指示删除操作的结果，表示如下：

00H 功能调用找到了一个以上匹配文件，并将匹配文件删除。

FFH 匹配文件没找到。

**其它要求:**

在 DOS3.1 以上版本的网络环境中使用功能调用，必须对存放匹配文件的目录有建立访问权。

**参见:**

41H.. 删除一个文件 (UNLINK)。

**程序实例:**

下面三个实例都是使用功能调用 13H 删除在缺省驱动器的当前目录中所有以 TMP 为扩展名的文件。

**Assembly Language Usage Example:**

```
;  
;DELETE MATCHING FILE (13H)  
;  
code segment public  
assume cs:code,ds:code  
org 100h  
start:jmp begin  
    fcb db 0, ??????????tmp',26 dup (?)  
    msg1 db Delete all files with the extension tpm',  
          db 0dh,0ah, '$'  
    msg2 db File(s) deleted! ,0dh,0ah, '$'  
    msg3 db No file(s) deleted,noth found.',0dh,0ah, '$'  
begin:mov ax,cs           ;set up ds  
    mov ds,ax             ;to same as cs  
    mov dx,offset msg1   ;address of delete message
```

```

        mov ah,09h      ;display string function request
        int 21h         ;call DOS
        mov dx,offset fcb ;address of file control block
        mov ah,13h      ;delete file function request
        int 21h         ;call DOS
        cmp al,0        ;does file exist?
        je no           ;no,take jump
        mov dx,offset msg2 ;address of file delete message
        mov ah,09h      ;display string function request
        int 21h         ;call DOS
        jmp done        ;all done
no:   mov dx,offset msg3 ;address of mo files deleted msg
        mov ah,09h      ;display string function request
        int 21h         ;call DOS
done: mov ah,00h      ;terminate program funct request
        int 21h         ;call DOS
code: ends            ;end of code segment
      end start       ;start is the entry point

```

**Pascal Usage Example:**

```
{ Delete matching File (
13) }
```

```
PROGRAM delete match;
```

```

CONST
  dos delete match = $13;
  default drive = 0;

TYPE
  fcb = RECORD
    drive number : byte;
    file name : ARRAY [1..11] OF char;
    mis : ARRAY [1..26] OF byte;
  END;
  regpack = RECORD
    CASE integer OF
      1: (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
      2: (al,ah,b!,bh,cl,ch,dl,dh:byte);
    END;

```

**VAR**

```
regs : regpack ;  
this file fcb :fcb ;
```

**BEGIN**

```
WITH regs DO
```

**BEGIN**

```
this file fcb.drive number := default drive ;  
this file fcb.file name :=      ???????tmp' ;  
writeln(  Delete all fileswith the extension tmp.' );  
ah := dos delete match ;  
ds := seg(this file fcb) ;  
dx := ofs(this file fcb) :  
msdos(regs);  
IF (al = 0) THEN  
    writeln(  File(s) deleted!)  
ELSE  
    writeln(  No file(s) deleted,match not found.) ;  
END;
```

```
END.
```

**C Usage Example:**

```
/*  
 * Delete Matching File (0x13)  
 */  
  
#include <dos.h>  
  
union REGS inregs,outregs;  
  
struct fcb {  
    char drive num;  
    char filename[11];  
    char misc[26];  
};  
struct fcb this file fcb = {0,"??????tmp"};
```

```

main()
{
    printf("Delete all files with the extension tmp.\n");
    inregs.h.ah = 0x13;
    inregs.x.dx = (int) this file fcb;
    intdos(inregs,outregs);
    if(outregs.h.al == 0)

```

## DOS 功能调用 14H

读下一顺序文件记录

**功能调用:**

14H 读下一顺序文件记录

**DOS 版本:**

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

使用给定 FCB，功能调用读由 FCB 的当前块字段（字节 0CH 和 0DH）和当前记录字段（字节 IF）指定的记录。读入的数据量（记录大小）由 FCB 的记录尺寸字段确定。参见第 60 页 FCB 的格式。当文件打开时，记录尺寸自动设为 80H (128) 个字节。如果每次读文件时都想读入不同的字节数，那么在使用功能调用前应改变记录尺寸。

功能调用将读入的数据放在当前盘传送地址处，并增加 FCB 的当前块字段和当前记录字段，使它指向下一记录。随着这些字段的增加，可以再次使用此功能调用读下一顺序记录。

**入口参数:**

调用前，需设置寄存器：

AH = 14H 指出功能调用号。

DS: DX 指向一打开文件的文件控制块 (FCB)。在使用功能调用前应确保当前块字段，当前记录字段和记录尺寸字段已正确设置。

**出口参数:**

返回后，设置寄存器为：

AL 指出读的结果，表示如下：

00 读出完全成功。

01 记录中无数据 (遇到文件结束)。

02 盘传送区无足够空间存放整个记录。

03 读一部分记录并遇文件结束。此记录的剩余部分填零。

DTA 盘传送地址 (DTA) 识别从磁盘读入数据的位置。

**其它要求:**

使用功能调用 0FH 打开文件时，FCB 的当前块字段自动设置为 0 块，表示在文件的第一块。然而打开文件时，DOS 并不设置当前记录字段，因此对已打开文件进行第一次

I/O 操作时，必须正确设置当前记录字段。为了从文件起始处开始读，应设当前块字段和当前记录字段均为 0。

为了在 DOS3.1 以上版本的网络环境中使用此功能调用，必须对含此文件的目录有读取权。

参见：

- 1AH.. 设盘传送地址。
- 21H.. 读一随机记录。
- 3FH.. 从文件或设备读出。

程序实例：

下面三个实例都是功能调用 14H 顺序读 TESTFILE.ASC 文件。程序在屏幕显示文件的内容。

#### Assembly Language Usage Example:

```
; ; READ NEXT SEQUENTIAL FILE RECORD (14H)
;
code segment public
assume cs: code,ds:code
org 100h
start: jmp begin
fcb db 0, testfileasc',26 dup (0)
msg1 db 'File not found.',0dh,adh, '$'
msg2 db 'No space in DTA !',0dh,adh, '$'
begin: mov ax,cs           ;set up ds
       mov ds,ax           ;to same as cs
       mov dx,offset fcb   ;address of file control block
       mov ah,0fh           ;open file function request
       int 21h              ;call DOS
       cmp al,0              ;does file exist?
       je read               ;yes,start reading file
       mov dx,offset msg1   ;address of not found message
       mov ah,09h             ;display string function request
       int 21h              ;call DOS
       jmp done               ;wrap it up
read:  mov dx,offset fcb   ;address of file control block
       mov ah,14h             ;read sequential function request
       int 21h              ;call DOS
       cmp al,01h             ;end of file encountered
```

	jc close	;close file and wrap it up
	cmp al,02h	;no space for read in DTA
	jne cont	;continue
	mov dx,offset msg2	;address of no space in DTA msg
	mov ah,09h	;display string function request
	int 21h	;call DOS
	jmp close	;close file and wrap it up
cont:	mov cx,128	;display all 128 characters in DTA
	mov si,80h	;default DTA is at 80h in this seg
disp:	mov dl,[si]	;get next character in DTA
	cmp dl,26	;is it end of file character ?
	je close	;yes ,close file and wrap it up
	mov ah,02h	;display character funct request
	int 21h	;call DOS
	inc si	;increment to next character
	loop disp	;do again ,until all DTA displayed

	jmp read	
close:	mov dx,offset fcb	;address of file control block
	mov ah,10h	;close file function request
	int 21h	;call DOS
done:	mov ah,00h	;terminate program funct request
	int 21h	;call DOS
	code ends	;end of code segment
	end start	;start is the entry point

### Pascal Usage Example:

{ Read Next sequential File (\$14) }

PROGRAM sequential read;

#### CONST

```

dos open file = $0f;
dos close file = $10;
dos set dta = $1a;
dos sequential read = $14;
default drive = 0;
eof = 26;

```

**TYPE**

```
fcb = RECORD
    drive number : byte ;
    file name : ARRAY [1..11] OF char ;
    fill1 : ARRAY [1..20] OF byte ;
    current record : byte ;
    fill2 : ARRAY [1..4] OF byte ;
  END;
regpack = RECORD
  CASE integer OF
    1: (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer) ;
    2: (al,ah,bl,bh,cl,ch,dl,dh:byte);
  END ;
```

**VAR**

```
regs : regpack ;
this file fcb : fcb ;
dta : ARRAY [1..11] OF char ;
l : integer ;
break : boolean ;
BEGIN
  WITH regs DO
    BEGIN
      al := 0 ;
      this file fcb.current record := 0 ;
      this file fcb.drive number := default drive ;
      this file fcb.file name := testfileasc' ;
      ah := dos set dta ;
      ds := seg(dta) ;
      dx := ofs(dta) ;
      msdos(regs) ;
      ah := dos open file ;
      ds := seg(this file fcb) ;
      dx := ofs(this file fcb) ;
      msdos (regs) ;
      ah := dos open file ;
      ds := seg(this file fcb) ;
      dx := ofs(this file fcb) ;
```

```

msdos(regs);
IF al < > 0 THEN
  writeln(' File not found.')
ELSE
BEGIN
  break := false;
  WHILE break = false DO
    BEGIN
      ah := dos sequential read;
      ds := seg(this file fcb);
      dx := ofs(this file fcb);
      msdos(regs);
      IF al = 1 THEN
        break := true
      ELSE
        BEGIN
          IF al = 2 THEN
            BEGIN
              writeln(' No space in DTA !');
              break := true;
            END
          ELSE
            BEGIN
              FOR I:= 1 TO 128 DO
                BEGIN
                  IF dta[i] = chr(eof) THEN
                    break := true
                  ELSE
                    write(dta[[i]]);
                END;
              END;
            END;
          END;
        END;
      ah := dos close file;
      ds := seg(this file fcb);
      dx := ofs(this file fcb);
      msdos(regs);
    END;
  END;

```

END.

**C Usage Example:**

```
/*  
 * Read Next Sequential File Record (0x14) */  
  
#include <dos.h>  
  
union REGS inregs,outregs;  
  
struct fcb {  
    char drive num;  
    char filename[11];  
    char fill1[20];  
    char current record;  
    char fill2[4];  
};  
struct fcb this file fcb = {0,"testfileasc"};  
  
main( )  
{  
    char dta [128];  
    this file fcb.current record = 0;  
    inregs.x.dx = (int) &dta[0];  
    inregs.h.ah = 0x1a;  
    intdos(&inregs,&outregs);  
    inregs.x.dx = (int) &this file fcb;  
    inregs.h.ah = 0x0f;  
    intdos(&inregs,&outregs);  
    if (outregs.h.al != 0)  
        printf("File not found.\n");  
    else {  
        for (;;) {  
            inregs.x.dx = (int) &this file fcb;  
            inregs.h.ah = 0x14;  
            intdos (&inregs,&outregs);  
            if (outregs.h.al == 1)  
                break;  
        }  
    }  
}
```

```

        if (outregs.h.al == 2) {
            printf("No space in DTA! \n");
            break;
        }
        printf("%s",dta);
    }
inregs.x.dx = (int) &this file fcb;
inregs.h.ah = 0x10;
intdos(&inregs,&outregs);
}
}

```

## DOS 功能调用 15H

### 写下一顺序文件记录

**功能调用:**

15H 写下一顺序文件记录

**DOS 版本:**

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

此功能调用写一数据记录，将磁盘传送地址开始的一个数据记录写入由给定 FCB 的当前记录字段（字节 1FH）所指定的位置。参见第 60 页 FCB 的格式。文件打开时，记录尺寸自动设为 80H (128) 个字节。如果每次要写入不同的字节数，则在使用此功能调用前应改变记录尺寸。

此功能调用也能自动增加当前块字段和当前记录字段，指向下一记录。随着这些字段的增加，就能继续使用此功能调用写下一顺序记录。

如果写的记录不是完整的一个磁盘扇区，DOS 就将数据存放在内存缓冲区中，直到追加数据满一扇区，才写入整个扇区。因此在关闭文件以前，内存中的数据有可能还没有写入文件。所以如果在关闭文件以前程序不正常结束，可能产生文件的完整性问题。

**入口参数:**

调用前，需设置寄存器：

AH=15H 指出功能调用号。

DS: DX 指向一已打开文件的文件控制块，在使用此功能调用前应确保当前块字段，当前记录字段和记录尺寸段都已正确设置。

DTA 盘传送地址 (DTA) 确定要写入文件数据的起始地址。

**出口参数:**

返回后，设置为：

AL 指出写的结果，表示如下：

00 写入完全成功。

01 无磁盘空间，写失败。

02 DOS 内部磁盘传送区足够空间写一记录，数据无法传送。

其它要求：

- 使用功能调用 0FH 打开文件时，FCB 的当前块字段自动设为 0 块，表示在文件的第一块。然而，在打开文件时，DOS 并不设当前记录字段。因此在对已打开文件进行第一次 I/O 操作时，必须正确设置当前记录字段。为了写文件的第一个记录，应设当前块字段和当前记录字段为 0。

对于任何只读文件，此功能调用均不能写。

为了在 DOS3.1 以上版本的网络环境中使用此功能调用，必须对含有此文件的目录有写入权。

参见：

1AH.. 设盘传送地址。

40H.. 写文件或设备。

程序实例：

下面三个实例都是使用功能调用 15H 写顺序记录。程序读名为 TESTFILE.IN 的文件内容，并写到名为 TESTFILE.OUT 中。

#### Assembly Language Usage Example:

```
;  
; WRITE NEXT SEQUENTIAL FILE RECORD (15H)  
;  
code segment public  
assume cs:code, ds:code  
  
    org      100h  
start: jmp      begin  
infcb   db      0,'testfilein', 26 dup (0)  
outfcb  db      0,'testfileout', 26 dup (0)  
msg1    db      'File not found.', 0dh, 0ah, '$'  
msg2    db      'No space in DTA!', 0dh, -ah, '$'  
begin: mov      ax,cs           ;set up ds  
        mov      ds,ax           ;to same as cs  
        mov      dx, offset infcb ;addr of input file control block  
        mov      ah,0fh           ;open file function request  
        int      21h             ;call DOS  
        cmp      al,0             ;does file exist?  
        je      create           ;yes, create output file  
        mov      dx,offset msg1  ;address of not found message  
        mov      ah,09h           ;display string function request  
        int      21h             ;call DOS
```

	jmp	done	;wrap it up
create:	mov	dx,offset outfcb	;address of input fcb
	mov	ah,16h	;create file function request
	int	21h	;call DOS
read:	mov	dx,offset infcb	;address of input fcb
	mov	ah,14h	;read sequential function request
	int	21h	;call DOS
	cmp	al,01h	;end of file encountered
	je	close	;close file and wrap it up
	cmp	al,01h	;no space for read in DTA
	jnc	write	;continue
	mov	dx,offset msg2	;address of no space in DTA msg
	mov	ah,09h	;display string function request
	int	21h	;call DOS
	jmp	close	;close file and wrap it up
write:	mov	dx,offset outfcb	;address of output fcb
	mov	ah,15h	;sequential write funct request
	int	21h	;call DOS
	cmp	al,0	;check if write successful
	je	read	;read another record from file
close:	mov	dx,offset infcb	;address of input fcb
	mov	ah,10h	;close file function request
	int	21h	;call DOS
	mov	dx,offset outfcb	;address of output fcb
	mov	ah,10h	;close file function request
	int	21h	;call DOS
done:	mov	ah,00h	;terminate program funct request
	int	21h	;call DOS
code	ends		;end of code segment
	end	start	lstart is the entry point

**Pascal Usage Example:**

{Write Next Sequential File < \$15>}

PROGRAM sequential\_write;

CONST

```
dos_open_file = $0f;
dos_close_file = $10;
dos_create_file = $16;
dos_set_dta = $1a;
dos_sequential_read = $14;
dos_sequential_write = $15;
default_drive = 0;
```

TYPE

fcb = RECORD

```
    drive_number: byte;
    file_name: ARRAY [1..11] OF char;
    fill1 : ARRAY [1..20] OF byte;
    current_record: byte;
    fill1 : ARRAY [1..4] OF byte;
END;
```

regpack = RECORD

```
    CASE integer OF
        1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
        2 : (al,ah,bl,bh,cl,ch,dl,dh:byte);
    END;
```

VAR

```
regs: regpack;
input_fcb: fcb;
output_fcb: fcb;
dta: ARRAY [1..128] OF char;
break: boolean;
```

BEGIN

WITH regs DO

```

BEGIN
    input_fcb.current_record := 0;
    input_fcb.drive_number := default_drive;
    input_fcb.file_name := 'testfilein';
    output_fcb.current_record := 0;
    output_fcb.drive_number := default_drive;
    output_fcb.file_name := 'testfileout';
    ah := dos_set_dta;
    ds := seg(dta);
    dx := ofs(dta);
    msdos(regs);
    ah := dos_open_file;
    as := seg(input_fcb);
    dx := ofs(input_fcb);
    msdos(regs);
    IF al < > 0 THEN
        writeln('File not found.')
    ELSE

        BEGIN
            ah := dos_create_file;
            ds := seg(output_fcb);
            dx := ofs(output_fcb);
            msdos(regs);
            break := false;
        WHILE break = false DO
            BEGIN
                ah := dos_sequential_read;
                ds := seg(input_fcb);
                dx := ofs(input_fcb);
                msdos(regs);
                IF al = 1 THEN
                    break := true
                ELSE

```

```
BEGIN
    IF AL = 2 THEN
        BEGIN
            writeln('No space in DTA!') ;
            break := true ;
        END
    ELSE
        BEGIN
            ah := dos_sequential_write ;
            ds := seg(output_fcb) ;
            dx := ofs(output_fcb) ;
            msdos(regs) ;
            IF al < > 0 THEN
                break := true ;
            END ;
        END ;
    END ;
    ah := dos_close_file ;
    ds := seg(input_fcb) ;
    dx := ofs(input_fcb) ;
    msdos(regs) ;
    ah := dos_close_file ;
    ds := seg(input_fcb) ;
    dx := ofs(input_fcb) ;
    msdos(regs) ;
    END ;
END ;
```

### C Usage Example:

```
/*  
 * Write Next Sequential File Record (0X15)  
 */
```

```
#include <dos.h>  
  
union REGS inregs, outregs;  
struct fcb {  
    char drive_num;  
    char filename[11];  
    char fill1[20];  
    char current_record;  
    char fill2[4];  
};  
struct fcb input_fcb = {0, "testfilein -"};  
struct fcb output_fcb = {0, "testfileout"};  
  
main()  
{  
    char dta[128];  
  
    input_fcb.current_record = 0;  
    output_fcb.current_record = 0;  
    inregs.x.dx = (int) &dta[0];  
    inregs.h.ah = 0x1a;  
    intdos(&inregs, &outregs);  
    inregs.x.dx = (int) &input_fcb;  
    inregs.h.ah = 0x0f;  
    intdos(&inregs, &outregs);  
    if (outregs.h.al != 0)  
        printf("File not found.\n");  
    else {  
        inregs.x.dx = (int) &output_fcb;
```

```

inregs.h.ah = 0x16;
intdos(&inregs,&outregs);
for (;;) {
    inregs.x.dx = (int) &output_fcb;
    inregs.h.ah = 0x16;
    intdos(&inregs,&outregs);
    if (outregs.h.al == 1)
        break;
    if (outregs.h.al == 2) {
        printf("No space in DTA!\n");
        break;
    }
    inregs.x.dx = (int) &output_fcb;
    inregs.h.ah = 0x15;
    intdos(&inregs,&outregs);
    if (outregs.h.al != 0)
        break;
}
inregs.x.dx = (int) &input_fcb;
intdos(&inregs,&outregs);
inregs.x.dx = (int) &output_fcb;
inregs.h.ah = 0x10;
intdos (&inregs,&outregs);
}
}

```

## DOS 功能调用 16H

### 创建文件

#### 功能调用:

16H 创建文件

#### DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

#### 说明:

此功能调用为给定 FCB 的新文件创建目录项，参见前面所述的 FCB 格式。功能调用在当前目录中搜索与给定文件名匹配的文件。如果找到，就使用此目录，并破坏文件的当前内容，否则使用空项。

功能调用获得目录项后，初始化此项为零长度文件并打开此文件。

如果要将文件设成专用属性（例如系统或隐含文件），则可使用扩充 FCB 并设置适

当状态位给指定的隐含文件。

**入口参数:**

调用前, 需设置寄存器:

AH=16H 指出功能调用号。

DS: DX 指向一未打开文件的文件控制块 (FCB)。对此功能调用, FCB中只有文件名字段需填入。

**出口参数:**

返回后, 设置寄存器为:

AL 指出创建结果, 表示如下:

00H 文件创建成功。

FFH 文件创建失败。无可用目录项或磁盘满并且磁盘中无与给定文件名匹配的文件。

**其它要求:**

为了在 DOS3.1 版本以上的网络环境中使用此功能调用, 必须对当前目录有创建访问权。

**参见:**

0FH.. 打开文件。

0CH.. 创建文件 (CREAT)。

0DH.. 打开文件。

**程序实例:**

下面三个实例, 都是使用功能调用 16H, 创建名为 ZZZZZZZZ.ZZZ 的文件。

**Assembly Language Usage Example:**

```
; ;CREATE FILE (16H)
;
code    segment public
assume cs:code,ds:code
        org      100h
start: jmp     begin
fcb     db      0,'ZZZZZZZZZZZZ',26 dup (?)
msg1   db      'ZZZZZZZZ.ZZZ file created',0dh,0ah,'$'
msg2   db      'File not created',0dh,0ah,'$'
begin: mov     ax,cs          ;set up ds
        mov     ds,ax          ;to same as cs
        mov     dx,offset fcb  ;address of file control block
        mov     ah,16h          ;create file function request
        int     21h             ;call DOS
        cmp     cl,0            ;was file created?
```

```

jne    error      ;nope, no dir entries available
mov    dx,offset msg1 ;address of created message
mov    ah,09h        ;display string function request
int    21h         ;call DOS
jmp    close       ;close file and wrap it up
error: mov   dx,offset msg2 ;address of not created message
       mov   ah,09h        ;display string function request
       int   21h         ;call DOS
       jmp   done        ;wrap it up
close: mov   dx,offset fcb  ;address of file control block
       mov   ah,10h        ;close file function request
       int   21h         ;call DOS
done:  mov   ah,00h        ;terminate program funct request
       int   21h         ;call DOS
code:  ends          ;end of code segment
end   start        ;start is the entry point

```

**Pascal Usage Example:**

{ Create File (\$ 16) }

PROGRAM create\_file;

CONST

```

dos_close_file = $10;
dos_create_file = $16;
default_drive = 0;

```

TYPE

```

regpack = RECORD
  CASE integer OF
    1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
    2 : (al,ah,bl,bh,cl,ch,dl,dh:byte);
  END;

```

```

fcb = RECORD
  drive_number : byte;
  file_name : ARRAY [1..11] OF char;
  misc : ARRAY [1..26] OF byte;
END;

```

VAR

```

regs : regpack ;
new_file_fcb : fcb ;
BEGIN
  WITH regs DO
    BEGIN
      new_file_fcb.drive_number := default_drive ;
      new_file_fcb.file_name := 'ZZZZZZZZZZZZ' ;
      ah := dos_create_file ;
      ds := seg(new_file_fcb) ;
      dx := ofs(new_file_fcb) ;
      msdos(regs) ;
      IF al < > 0 THEN
        writeln('File not created')
      ELSE
        BEGIN
          writeln('ZZZZZZZZ.ZZZ file created') ;
          ah := dos_close_file ;
          ds := scg(new_file_fcb) ;
          dx := ofs(new_file_fcb) ;
          msdos(regs) ;
        END ;
      END ;
    END.

```

### C Usage Example:

```

/*
 * Create File (0x16)
 */
#include <dos.h>

union REGS inregs, outregs;
struct fcb {
  char drive_num;
  char filename[11];
  char misc[26];
};
struct fcb new_file_fcb = {0,"ZZZZZZZZZZZZ"};

```

```

main()
{
    inregs.x.dx = (int) &new_file_fcb;
    inregs.h.ah = 0x16;
    intdos(&inregs,&outregs);
    if (outregs.h.al != 0)
        printf("File not created\n");
    else {
        printf("ZZZZZZZZ.ZZZ file created\n");
        inregs.x.dx = (int) &new_file_fcb;
        inregs.h.ah = 0x10;
        intdos(&inregs,&outregs);
    }
}

```

## DOS 功能调用 17H

### 文件重命名

#### 功能调用:

17H 文件重命名

#### DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

#### 说明:

此功能调用以修改的 FCB 给定的信息为基础，对当前目录中的文件重新命名。参见前面的 FCB 的格式。

FCB 在正常位置必须有驱动器码和文件名。这些字段确定将被改名的文件。文件名可以含字符“？”，此时功能调用将改变所有匹配文件的名字。

必须在 FCB 中第一文件的扩展名六个字节为起始处 (DS: DX+11H 开始放新的文件名。此区通常用于文件尺寸、日期和 DOS 备份区。

功能调用在当前目录中搜索与第一个文件名匹配的文件，并将这些文件的名字改成与第二个文件名相匹配。如果第二个文件名含有通配符“？”，则相应的字符保持不变。

#### 入口参数:

调用前，需设置寄存器：

AH = 17H 指出功能调用号。

DS: DX 指向已按上述说明修改的文件控制块 (FCB)。

#### 出口参数:

返回后，设置寄存器为：

AL 指出重命名结果，表示如下：

00H 文件重命名成功。

FFH 由于匹配文件没找到或重命名后的文件已有同名文件存在，文件重命名失败。

**其它要求：**

被改名的文件必须未设只读属性，否则功能调用不能对它重命名。

为了在 DOS3.1 以后的网络环境中使用此功能调用，必须对含原文件的目录有创建访问权。

**参见：**

56H.. 文件重命名。

**程序实例：**

下面三个实例都是使用功能调用 17H 将文件 ZZZZZZZZ.ZZZ 重命名为 YYYYYYYY.YYY。

**Assembly Language Usage Example:**

```
; ; RENAME FILE (17H)
;
code      segment public
assume cs:code,ds:code
org      100h
start: jmp      begin
fcb      db      0,'ZZZZZZZZZZ'
          db      5 dup (?)
          db      'YYYYYYYYYYYY'
          db      10 dup (?)
msg1     db      'ZZZZZZZZ.ZZZ renamed to YYYYYYYY.YYY',0dh,0ah,"$"
msg2     db      'File not renamed;',0dh,0ah,'$'
begin: mov      ax,cs           ;set up ds
        mov      ds,ax           ;to same as cs
        mov      dx,offset fcb  ;address of file control block
        mov      ah,17h          ;rename file function, request
        int      21h             ;call DOS
        cmp      al,0             ;was file renamed?
        jnc      error           ;nope
        mov      dx,offset msg1 ;address of renamed message
        mov      ah,09h          ;display string function request
        int      21h             ;call DOS
        jmp      done             ;wrap it up
error:  mov      dx,offset msg2 ;address of not renamed message
        mov      ah,09h          ;display string function request
```

```

        int      21h      ;call DOS
done:  mov      ah,00h    ;terminate program funct request
        int      21h      ;call DOS
code   ends                ;end of code segment
        end      start     ;start is the entry point

```

**Pascal Usage Example:**

```

Rename File ($ 17) }

PROGRAM rename_file;
CONST
  dos_rename_file = $ 17;
  default_drive = 0;

TYPE
  fcb = RECORD
    drive_number : byte;
    file_name : ARRAY [1..11] OF char;
    misc1 : ARRAY [1..5] OF byte;
    new_file_name : ARRAY [1..11] OF char;
    misc2 : ARRAY [1..10] OF byte;
  END;
  regpack = RECORD
    CASE integer OF
      1 :(ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
      2 :(al,ah,bl,bh,cl,ch,dl,ch:byte);
    END;
  VAR
    file_fcb : fcb;
    regs : regpack;

BEGIN
  WITH regs DO
    BEGIN
      file_fcb.drive_number := default_drive;
      file_fcb.file_name := 'ZZZZZZZZZZZ';
      file_fcb.new_file_name := 'YYYYYYYYYYY';
      ah := dos_rename_file;
      ds := seg(file_fcb);
    END;

```

```

dx := ofs(file_fcb);
msdos(regs);
IF al < > 0 THEN
    writeln('File not renamed')
ELSE
    writeln('ZZZZZZZZ.ZZZ renamed to YYYYYYYY.YYY');
END;
END.

```

**C Usage Example:**

```

/* *
 * Rename File (0x17)
 */
#include <dos.h>

union REGS inregs, outregs;
struct fcb {
    char drive_num;
    char filename[11];
    char misc1[5];
    char newfilename[11];
    char misc2[10];
};
struct fcb file_fcb = {0,"ZZZZZZZZZZZ",0,0,0,0,"YYYYYYYYYYY"},

main()
{
    inregs.x.dx = (int) &file_fcb;
    inregs.h.ah = 0x17;
    intdos(&inregs,&outregs);
    if (outregs.h.al != 0)
        printf("File not renamed\n");
    else
        printf("ZZZZZZZZ.ZZZ renamed to YYYYYYYY.YYY\n");
}

```

## DOS 功能调用 18H

### DOS 内部使用

#### 功能调用:

18H 此功能调用保留为 DOS 内部使用，用户程序不能使用。

## DOS 功能调用 19H

### 获取当前盘号

#### 功能调用:

19H 获取当前盘号 DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

#### 说明:

此功能调用返回当前缺省驱动器的代码。

#### 入口参数:

调用前，需设置寄存器:

AH = 19H 指出功能调用号。

#### 出口参数:

返回后，设置寄存器为:

AL 寄存器的数值表示当前缺省驱动器。0 表示 A 驱动器，1 表示 B 驱动器等。

#### 程序实例:

下面三个实例都是使用功能调用 19H 取当前盘号。程序将被选的当前磁盘驱动器的驱动器字母显示在屏幕上。

#### Assembly Language Usage Example:

```
; ; CURRENT DISK (19H)
;
code segment public
assume cs:code,ds:code
    org 100h
start: jmp begin
msg1    db 'The currently selected drive is ''$'
msg2    db 0dh,0ah,'$'
begin: mov ax,cs           ;set up ds
       mov ds,ax          ;to same as cs
       mov dx,offset msg1 ;address of message
       mov ah,09h          ;display string function request
```

```

int    21h      ;call DOS
mov    dl,'A'   ;set up for drive a
int    da,al    ;add drive number
mov    ah,02h   ;display character funct request
add    21h      ;call DOS
mov    dx,offset msg2 ;address of message with crlf
int    ah,09h   ;display string function request
mov    21h      ;call DOS
mov    ah,00h   ;terminate program funct request
int    21h      ;call DOS
code   ends     ;end of code segment
end    start    ;start is the entry point

```

**Pascal Usage Example:**

{ Current Disk (\$19) }

```

PROGRAM current_disk ;
  COMST
    dos_current_disk = $19;

  TYPE
    regpack = RECORD
      CASE integer OF
        1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
        2 : (al,ah,bl,bh,cl,ch,dl,dh:byte);
      END;
  VAR
    regs : regpack;
  BEGIN
    WITH regs DO
      BEGIN
        ah := dos_current_disk;
        msdos(regs);
        writeln('The currently selected drive is',
               chr(al + ord('A')));
      END;
  END.

```

**C Usage Example:**

```

/*
 * Current Disk (0x19)
 */

#include <dos.h>

union REGS inregs, outregs;

main()
{
    inregs.h.ah = 0x19;
    intdos(&inregs,&outregs);
    printf("The currently selected drive is %c\n",
        outregs.h.al + 'A');
}

```

## DOS 功能调用 1AH

### 设盘传送地址

#### 功能调用:

1AH 设盘传送地址

#### DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

#### 说明:

此功能调用设置盘传送地址 (DTA)。DOS 在执行文件 I/O 操作时使用此地址。当写盘时，DOS 认为被写信息从该地址开始。同样当读盘时，DOS 从该地址开始存放读入的信息。

无论何时，要改变 DTA 时，均应使用此功能调用。例如，程序为了对多个文件进行 I/O 操作，每个文件都要使用不同的传送区。

由于 DOS 不允许盘传送复盖住段的起始部分或溢出到下一段，所以在段尾和 DTA 之间应有足够的空间，保证可能的最大盘传送区。

如果不使用此功能调用，缺省盘传送址即在 PSP 的偏移量为 80H 处。对此缺省 DTA，盘传送区只有 128 字节长度。长度大于 128 字节的数据缓冲区就要扩充到程序段，并破坏程序。参见前面所述的 PSP 的格式。

#### 入口参数:

调用前，需设置寄存器:

AH = 1AH 指出功能调用号。

DS: DX 放盘传送地址。

#### 出口参数:

无。

参见:

2FH.. 获取盘传送地址

程序实例:

下面三个实例都是使用功能调用 1AH 设置 DTA。程序把名为 TESTFILE.ASC 的文件的内容读入传送区，然后在屏上显示传送区的内容。

#### Assembly Language Usage Example:

```
; ;SET DISK TRANSFER AREA (1AH)
;
code    segment public
assume cs:code,ds:code
        org      100h
start: jmp      begin
fcb     db      0,'testfileasc',26 dup (?)
dta     db      128 dup (?)
msg1   db      'File not found.',0dh,0ah,'$'
msg2   db      'No space in DTA!',0dh,0ah,'$'
begin: mov      ax,cs          ;set up ds
        mov      ds,ax          ;to same as cs
        mov      dx,offset dta  ;address of disk transfer area
        mov      ah,1ah          ;set disk transfer area funct req
        int      21h            ;call DOS
        mov      dx,offset fcb  ;address of file control block
        mov      ah,0fh          ;open file function request
        int      21h            ;call DOS
        cmp      al,0            ;does file exist?
        je      read             ;yes,start reading file
        mov      dx,offset msg1 ;address of not found message
        mov      ah,09h          ;display string function request
        int      21h            ;call DOS
        jmp      done             ;wrap it up
read:  mov      dx,offset fcb  ;address of file control block
        mov      ah,14h          ;read sequential function request
        int      21h            ;call DOS
        cmp      al,01h          ;end of file encountered
        je      close             ;close file and wrap it up
        cmp      al,02h          ;no space for read in DTA
        jne      cont             ;continue
```

```

        mov     dx,offset msg2 ;addr of no space in DTA message
        mov     ah,09h          ;display string function request
        int     21h             ;call DOS
        jmp     close            ;close file and wrap it up
cont:  mov     cs,128           ;display all 128 characters in DTA
        mov     si,offset dta   ;address of disk transfer area
disp:  mov     dl,[si]          ;get next character in DTA
        cmp     dl,26            ;is it end of file character?
        je      close            ;yes, close file and warp it up
        mov     ah,02h           ;display character funct request

        int     21h             ;call DOS
        inc     si               ;increment to next character
        loop    disp              ;do again, until all of DTA disp
        jmp     read              ;read another record from file
close: mov     dx,offset fcb   ;address of file control block
        mov     ah,10h           ;close file function request
        int     21h             ;call DOS
done:  mov     ah,00h           ;terminate program funct request
        int     21h             ;call DOS
code:  ends
end    start             ;start is the entry point

```

#### Pascal Usage Example:

{ Set Disk Transfer Area (\$ 1A) }

PROGRAM sct\_dta;

#### CONST

```

Dos_open_file = $0f;
dos_close_file = $10;
dos_set_dta = $1a;
dos_sequential_read = $14;
default_drive = 0;
eof = 26;

```

#### TYPE

```

fcb = RECORD
  drive_number : byte;
  file_name : ARRAY [1..11] OF char;

```

```

    fill1 : ARRAY [1..20] OF byte ;
    current_record : byte ;
    fill2 : ARRAY [1..4] OF byte ;
  END ;
regpack = RECORD
  CASE integer OF
    1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer) ;
    2 : (al,ah,bl,bh,cl,ch,dl,dh:byte) ;
  END ;
VAR
  regs : regpack ;
  this_file_fcb : fcb ;
  dta : ARRAY [1..128] OF char ;
  i : integer ;
  break : boolean ;
BEGIN
  WITH regs DO
    BEGIN
      al := 0 ;
      this_file_fcb.current_record := 0 ;
      this_file_fcb.drive_number := default_drive ;
      this_file_fcb.file_name := 'testfileasc' ;
      ah := dos_set_dta ;
      ds := seg(dta) ;
      dx := ofs(dta) ;
      msdos(regs) ;
      ah := dos_open_fle ;
      ds := seg(this_file_fcb) ;
      dx := ofs(this_file_fcb) ;
      msdos(regs) ;
    IF al < > 0 THEN
      writeln ('File not found.')
    ELSE
      BEGIN
        break := false ;
        WHILE break = false DOS
          BEGIN
            IF al = 2 THEN
              BEGIN

```

```

        writeln('No space in DTA!');

        break := true;
    END
ELSE
BEGIN
FOR i := 1 TO 128 DO
BEGIN
IF dta[i] = chr(cof) THEN
    break := true
ELSE

        write(dta[i]);
    END;
END;
END;
ah := dos_close_file;
ds := seg(this_file_fcb);
dx := ofs(this_file_fcb);
msdos(regs);
END;
END;
END.

```

#### C Usage Example:

```

/*
 * set Disk Transfer Area (0x1a)
 */

#include <dos.h>

union REGS inregs, outregs;
struct fcb {
    char drive_num
    char filename[11];
    char misc[26];
};
struct fcb this_file_fcb = {0,"testfileasc"};

main()

```

```

}

char dta [128];

inregs.x.dx = (int) &dta[0];
inregs.h.ah = 0x1a;
intdos (&inregs,&outregs);
inregs.x.dx = (int) &this_file_fcb;
inregs.h.ah = 0x0f;
intdos(&inregs,&outregs);
if (outregs.h.al != 0)
    printf("File not found.\n");
else {
    for (;;) {
        inregs.x.dx = (int) &this_file_fcb;
        inregs.h.ah = 0x14;

        intdos(&inregs,&outregs);
        if (outregs.h.al == 1)
            break;
        if (outregs.h.al == 2) {
            printf("No space in DTA!\n");
            break;
        }
        printf("%S",dta);
    }
    inregs.x.dx = (int) &this_file_fcb;
    inregs.h.ah = 0x10;
    intdos(&inregs,&outregs);
}
}

```

## DOS 功能调用 1BH 获取当前驱动器的 FAT 信息

**功能调用:**

1BH 获得当前驱动器的 FAT 信息

**DOS 版本:**

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

此功能调用返回当前驱动器中有关磁盘文件分配表 (FAT) 的信息。此功能调用返回 FAT 标识字节的指针、磁盘的分配单位 (簇) 数、每簇的扇区数和物理扇区的尺寸。

更新的功能调用 (36H.. 取磁盘自由空间) 完成的操作与此类似。

#### 入口参数:

调用前，需设置寄存器：

AH = 1BH 指出功能调用号。

#### 出口参数:

返回后，设置寄存器为：

DS: DX 指向当前驱动器的FAT标识字节。在DOS2.0以前的版本中，可以认为该字节是整个FAT的第一个字节。然而从2.0版本开始，DOS不再将整个FAT放入内容。为了安全，只能认为此寄存器对指向标识字节。

标识字节放有驱动器的介质类型信息。

IBM 支持的标准介质的数值规定如下：

F8H 硬盘。

F9H 双面软盘，每道15扇区，每面40道。

FCH 单面软盘，每道9扇区，每面40道。

FDH 双面软盘，每道9扇区，每面40道。

FEH 单面软盘，每道8扇区，每面40道。

FFH 双面软盘，每道8扇区，每面40道。

用户设备驱动器（如那些支持磁带的驱动器）可以对FAT标识字节赋以不同的数值来定义那些用户设备。

FAT标识字节所提供的信息，还不能完全确定特定设备的识别，尤其是在可以支持逻辑设备的 DOS3.2 情况下，更是如此。

标识字节不在程序折数据段中，因此功能调用要改变DS的数值。为了确保正常运行，在使用此功能调用前程序应保存 DS 的数值，返回后恢复原来的 DS 数值。

DX 指出设备的分配单位 (簇) 数。

AL 指出每个分配单位的扇区数。通常，单面软盘是1，双面软盘是2。

CX 指出物理磁盘扇区字节数的大小，对于多数PC设备，此数值都是512。

#### 参见:

1CH.. 获取指定驱动器的 FAT 信息。36H.. 获取磁盘自由空间。

#### 程序实例:

下面三个实例都是使用功能调用 1BH 检索当前驱动器的 FAT 信息，并将信息显示在屏幕上。

#### Assembly Language Usage Example:

;

```

; GET FAT INFORMATION (1,BH)

;

c1de segment public
assume cs:code,ds:code

    org      100h

start: jmp      begin

msg1   db      'FAT id byte = ','$'
mag2   db      'Number of clusters = ','$'
mag3   db      'Sectors per cluster = ','$'
mag4   db      'Sector size =      ','$'
string db      6 dup (?),0dh,0ah,'$'

fatid  db      0
clust   dw      0
sect    db      0
secsize dw      0

;

;convert number to ascii

;

numasc proc          ;number to convert is in ax
    mov      cx,6          ;string is 6 bytes long
    mov      bx,offset string;get address of string
blank:  mov      byte ptr [bx],';put blanks in this byte
        inc      bx          ;increment to next byte;
loop:   loop     blank          ;do until all 6 bytes are blanked
        mov      si,10         ;use si to divide by 10
next:  xor      dx,dx         ;clear dx
        div      si          ;divide ax by 10 (in si)
        add      dx,'0'        ;convert remainder to ascii number
        dec      bx          ;goto next char pos in string
        mov      [bx],d1        ;store character in the string
        or       ax,ax         ;any more digits to convert?
        jnz     next          ;yes,get next digit
        ret                  ;no,return
numasc endp          ;end of proceder

;

; main program

;

begin:  mov      ax,cs          ;set up ds
        mov      ds,ax          ;to same as cs

```

```

push    ds      ;save ds
mov     ah,1bh   ;get fat info function request

int     21h     ;call DOS
mov     ah,[bx]  ;save fat id byte
pop     ds      ;restore original ds
mov     byte ptr fatid,ah;save fat id byte
mov     word ptr clust,dx;save number of clusters
mov     byte ptr sect,al;save number sectors per cluster
mov     word ptr secsize,cx;save sector size
mov     dx,offset msg1 ;address of FAT id message
mov     ah,09h   ;display string function request
int     21h     ;call DOS
xor    ax,ax    ;clear ax
mov     al,byte ptr fatid;FAT id
call    numasc   ;convert binary number to ascii
mov     dx,offset string ;address of converted number
mov     ah,09h   ;display string function request
int     21h     ;call DOS
mov     dx,offset msg2 ;address of num of clusters msg
mov     ah,09h   ;display string function request
int     21h     ;call DOS
xor    ax,ax    ;clear DOS
mov     ax,word ptr clust;number of cluster
call    numasc   ;convert binary number to ascii
mov     dx,offset string ;address of converted number
mov     ah,09h   ;display string function request
int     21h     ;call DOS
mov     dx,offset msg3 ;address of number of sect per clus msg
mov     ah,09h   ;Display string function request
int     21h     ;call DOS
xor    ax,ax    ;clear ax
mov     al,byte ptr sect ;sectors per cluster
call    numasc   ;convert binary number to ascii
mov     dx,offset string ;address of converted number
mov     ah,09h   ;display string function request
int     21h     ;call DOS
mov     dx,offset msg4 ;address of sector size message
mov     ah,09h   ;display string function request

```

```

int      21h          ;call DOS
xor      ax,ax         ;clear ax
mov      ax,word ptr secsize;sector size
call     numasc        ;convert binary number to ascii
mov      dx,offset string ;address of converted number
mov      ah,09h        ;display string function request
int      21h          ;call DOS

done: move   ah,00b      ;terminate program funct request
      int    21h          ;call DOS
code  ends            ;end of code segment
      end    start        ;start is the entry point

```

### Pascal Usage Example

```
{ Get FAT Information ($ 1B) }
```

```

PROGRAM get_fat_info;
CONST
  dos_get_fat_info = $ 1B;

TYPE
  regpack = RECORD
    CASE integer OF
      1:(ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
      2:(al,ah,bl,bh,cl,ch,dl,dh:byte);
    END;
VAR
  regs : regpack

BEGIN
  WITH regs DO
  BEGIN
    ah:=dos_get_fat_info;
    msdos(regs);
    writeln('Number of cluster = ',dx);
    writeln('Sectors per cluster = ',a1);
    writeln('Sector size = ',cx);
  END;
END.

```

### C Usage Example :

```
/*
 * Get FAT Information (0x1b)
 */

#include <dos.h>

union REGS inregs, outregs;

struct SREGS segregs;

main()
{
    inregs.h.ah = 0x1b;
    intdosx(&inregs,&outregs,&segregs);
    printf("Number of cluster = %d\n",outsegs.x.dx);
    printf("Sectors per cluster = %d\n",outsegs.h.al);
    printf("Sectors size = %d\n",outregs.x.cx);
}
```

## DOS 功能调用 1CH

### 获取指定驱动器的 FAT 信息

#### 功能调用:

1CH 获取指定驱动器的 FAT 信息

#### DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

#### 说明:

此功能调用除它可用于任何驱动器，而不只是当前驱动器外，几乎与前一功能调用(1BH)相同。功能调用返回与指定驱动器有关的文件分配表(FAT)的信息。该功能调用返回FAT标识字节的指针、磁盘的分配单位(簇)数、每簇的扇区数和物理扇区的尺寸。

较新的功能调用(36H..取磁盘自由空间)完成的操作与此类似。

#### 入口参数:

调用前，需设置寄存器:

AH . 1CH 指出功能调用号。

DL 要获取信息的驱动器号。0表示缺省驱动器，1表示A驱动器，2表示B

驱动器，等等。

**出口参数：**

返回后，设置寄存器为：

AL 指出，表示如下：

DS: BX 该寄存器对指向此驱动器的FAT标识字节。在DOS2.0以前的版本中，可以认为此字节为整个FAT的第一个字节。然而从2.0版本开始，DOS不再将整个FAT放入内存，为了安全，只能认为此寄存器对指向标识字节。标识字节放有驱动器的介质类型信息IBM支持的标准介值数值定义如下：

F8H 硬盘。

G9H 双面软盘，每道15扇区，每面40道。

FCH 单面软盘，每道9扇区，每面40道。

FDH 双面软盘，每道8扇区，每面40道。

FEH 单面软盘，每道8扇区，每面40道。

FFH 双面软盘，每道8扇区，每面40道。

用户设备驱动器（如那些支持磁带的驱动器）可以对FAT标识字节赋予不同的数值来定义那些用户设备。

FAT标识字节所提供的信息，还不能完全确定特定设备的识别，尤其在可以支持逻辑设备的DOS3.2的情况下，更是如此。

标识字节不在程序的数据段中，因此功能调用要改变DS的数值。为了确保正常运行，在使用此功能调用前程序应保存DS的数值，返回后恢复原来的DS数值。

DX 指出设备的分配单位（簇）数。

AL 指出每个分配单位的扇区数。通常单面软盘是1，双面软盘是2。

CX 指出物理磁盘扇区字节数的大小，对于多数PC设备，此数值都是512。

**参见：**

1BH.. 获取当前驱动器的FAT信息。

36H.. 获取磁盘自由空间。

**程序实例：**

下面三个实例都是使用功能调用1CH检索A驱动器的FAT信息，并将信息显示在屏幕上。

**Assembly Language Usage Example;**

```
;  
; GET FAT INFORMATION FOR ANY DRIVE(1CH)  
;  
code segment public  
assume cs:code,ds:code
```

```

        org    100h
start: jmp    begin
msg1     db     'FAT id byte=' '$'
msg2     db     'Number of clusters =' '$'
msg3     db     'Sectors per clusters =' '$'
msg4     db     'Sector size=' '$'
string   db     6 dup(?),0dh,0ah,'$'
fatid   db     0
clust   dw     0
sect    db     0
secsize dw     0
;
; convert number to ascii
;
numasc proc           ;number to convert is in ax
        mov cx,6          ; string is 6 bytes long
        mov bx,offset string ;get address of string
blank: mov byte ptr[bx],'' ;put blanks in this byte
        inc bx            ;increment to next type
        loop blank         ;do until all 6 bytes are blanked
        mov si,10           ;use si to divide by 10
next:  xor dx,dx          ;clear dx
        div si             ;divide ax by 10(in si)
        add dx,'0'          ;convert remainder to an ascii num
        dec bx             ;go to next char pos in string
        mov [bx],dl          ;store character in thd string
        or ax,ax            ;any more digits to convert?
        jnz next             ;yes,get next digit
        ret                 ;no,return
numasc endp           ;end of procedure
;
;main program
;
begin: mov ax,cs           ;set up ds
        mov ds,ax           ;to same as cs
        push ds             ;save ds
        mov dl,01h           ;get fat info for drive a
        mov ah,1ch           ;get fat info for drive funct reg
        int 21h              ;call DOS

```

```

    mov ah,[bx]           ;save fat id byte
    pop ds                ;restore origginal ds
    mov byte ptr fatid,ah ;save fat id byte
    mov word ptr clust,dx ;save number sectors per cluster
    mov word ptr secsize,cx ;save sector size
    mov dx,offset msg1   ;address of FAT id message
    mov ah,09h             ;display string function request
    int 21h               ;call DOS
    xor ax,ax              ;clear ax
    mov al,byte ptr fatid ;FAT id
    call numasc            ;convert binary number to ascii
    mov dx,offset string   ;address of converted number
    mov ah,09h             ;displaya string function reques;
    int 21h               ;call DOS
    mov dx,offset msg2   ;addr of num of clust message
    mov ah,09h             ;display string function reuqes
    int 21h               ;call DOS
    xor ax,ax              ;clear ax
    mov ax,word ptr clust ; number of clusters
    call numasc            ;convert binary number to ascii
    mov dx,offset string   ;address of converted number
    mov ah,09               ;display string function reques
    int 21h               ;call DOS
    mov ah,09h             ;display string function request
    int 21h               ;call dos
    xor ax,ax              ;clear ax
    mov al,byte ptr fatid ;PAT id
    call numasc            ;convert binary number to ascii
    mov dx,offset string   ;address of converted number
    mov ah,09h             ;display string function request
    int 21h               ;call DOS
    mov dx,offset msg2   ;aaddr of num of clust message
    mov ah,09h             ;displaya string function request
    int 21h               ;call DOS
    xor ax,ax              ;clear ax
    mov ax,word ptr clust ;number of clusters
    call numasc            ;convert binary number to ascii
    mov dx,offset string   ;address of converted number
    mov ah,09h             ;display stringfunction request

```

```

int 21h           ;call DOS
mov dx,offset msg3 ;num of sectors per cluster msg
mov ah,09h         ;display string function request
int 21h           ;call DOS
xor ax,ax         ;clear ax
mov al,byte ptr sect ;sectors per cluster
call numasc        ;convert binary number to ascii
mov dx,offset string ;address of converted number
mov ah,09h         ;display string function request
int 21h           ;call DOS
mov dx,offset msg4 ;address of sector size message
mov ah,09h         ;display string function request
int 21h           ;call DOS
xor ax,ax         ;clear ax
mov ax,word ptr secsize ;sector size
call numasc        ;convert binary number to ascii
mov dx,offset string ;address of converted number
mov ah,09h         ;display string function request
int 21h           ;call DOS
done: mov ah,00h    ;terminate program funct request
      int 21h           ;call DOS
code ends          ;end of code segment
      end start         ;start is the entry point

```

#### Pascal Usage Example:

{Get FAT Information For Specific Drive (\$1C)}

PROGRAM get\_fat\_info;

#### CONST

```

dos_get_fat_info_specific = $1C;
drive_a = 1;

```

#### TYPE

```

regpack = RECORD
  CASE integer OF
    1:(ax,bx,cx,dx,bp,si,di,ds,es,flags:integer)
    2:(al,ah,bl,bh,cl,ch,dl,dh:byte)
  END;

```

```

VAR
  regs:regpack;

BEGIN
  WITH regs DO
    BEGIN
      ah:=dos get fat info specific;
      dl:=drive a;
      msdos(regs);
      writeln('Number of clusters = ',dx);
      writeln('Sectors per cluster = ',al);
      writeln('Sector size = ',cx);
    END;
  END.

```

#### C Usage Example:

```

/*
 * Get FAT Information For Specific Drive (0x1c)
 */
#include <dos.h>

union REGS inregs,outregs;
struct SREGS segreggs;

main()
{
    inregs.h.dl=0x01;
    inregs.h.ah=0x1c;
    intdosx(&inregs,&outregs,&wegregs);
    printf("Number of clusters = %d\n",outregs.x.dx);
    printf("Sectors per cluster = %d\n",outregs.h.al);
    printf("Sector size = %d\n",outregs.x.cx);
}

```

DOS 功能调用 1DH, 1EH, 1FH 和 20H  
DOS 内部使用

功能调用:

1DH, 1EH, 1FH 和 20H 这些功能调用保留为 DOS 内部使用。用户程序不能使

用。

## DOS 功能调用 21H

### 读一随机记录

#### 功能调用:

21 H 从文件中读一随机记录

#### DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

#### 说明:

此功能调用从已打开文件读一随机记录。使用此功能调用前，必须设置此文件 FCB 的随机记录字段为要读的记录。例如，设随机记录字段为 0，则功能调用读此文件的第一个记录。设随机记录字段为 5，则读第 6 个记录；设为 9 读第 10 个记录，等等。参见前面所述的 FCB 格式。

此功能调用将读出的数据放在盘传送地址为起始的内存中。

#### 入口参数:

调用前，需设置寄存器：

AH = 21H 指出功能调用号。

DS: DX 指向已打开文件的文件控制块 (FCB)。必须将随机记录字段设为要读的记录，在使用功能调用前同时还应确保记录尺寸字段已正确设置。

#### 出口参数:

返回后，设置寄存器为：

FCB 功能调用设置 FCB (文件控制块) 的当前块字段和当前记录字段与随机记录字段一致。

AL 指出读的结果，表示如下：

00 读出完全成功。

01 记录中无数据 (遇到了文件结束)

02 传送完成前，已到达含有盘传送地址段的段尾。取消读出。

03 已读出部分记录后遇文件结束。此记录的剩余字节被填为零。

DTA 盘传送地址 (DTA) 指出从磁盘中读出数据的起始处。

#### 其它要求:

记录的大小和在文件中所处的位置，取决于 FCB 中的随机记录号和记录尺寸。打开文件时，DOS 自动将 FCB 中的记录尺寸字段设为 128。用此记录尺寸时，第三个记录 (随机记录号为 2) 的随机读取，将读 256 字节开始的 128 个字节数据。如果将记录尺寸改为 512 字节的数据。其差别与记录尺寸的不同有关，因此在使用此功能调用前，应确保 FCB 的记录大小字段已正确设置。

为了在 DOS3.1 以上的网络环境中使用此功能调用，必须对含有此文件的目录有读取权。

参见:

- 14H.. 读下一个顺序文件记录.
- 1AH.. 设置磁盘传送地址.
- 24H.. 设置随机记录字段.
- 27H.. 读取多个随机记录.
- 3FH.. 从设备或文件中读取.

程序实例:

下面三个实例都是使用功能调用 21H 读名为 TESTFILE.ASC 文件的第 25 个记录。  
每一记录长度均为 66 字节。

在使用 C 语言的例子中，注意 Microsoft C 不会自动将结构装入内存。确切地说，它的操作限于偶数字节，而文件控制块（FCB 结构）含有奇数字节的字段，因此为了装入这些字段，即在字段之间不能有额外的字节，要在使用 Microsoft C 编译器时，使用 /Zp 开关。这样就能装入结构了。

#### Assembly Language Usage Example:

```
; ;READ RANDOM RECORD(21H)
;
code segment public
assume cs:code,ds:code
        org      100h
start:  jmp      begin
        db       0,'testfileeasc'
curblk  dw       ?
recsize dw       ?
fill    db       17 dup(?)
ranrecl dw       0
ranrech dw       0
dta     db       66 dup(?),'$'
msg1   db       'File not found.',0dh,0ah,'$'
msg2   db       'Error on random record read',0dh,0ah,'$'
begin: mov ax,cs           ;set up ds
        mov ds,ax           ;to same as cs
        mov ddx,offset dta  ;address of disk transfer area
        mov ah,1ah           ;set dta function request
        int 21h              ;call DOS
        mov dx,offset fcb  ;address of file control block
        mov ah,0fh           ;open file function request
        int 21h              ;call DOS
```

	kcmp al,0	;does fileexist?
	je read	;yes,read a random record
	mov dx,offset msg1	;address of not found message
	mov ah,09h	;display string function request
	int 21h	;call DOS
	jmp done	;wrap it up
read:	mov recsize,66	;set record size in fcb to 66
	mov ranrec1,24	;use 24 as ofst to rec 25 in file
	mov dx,offset scb	;address of file control block
	mov ah,21h	;read random record funct request
	int 21h	;call DOS
	cmp al,0	;was random read successful?
	jne error	;no,display error message
	mov dx,offset dta	;address disk transfer area
	mov ah,09h	;display string function request
	int 21h	;call DOS
	jmp close	;jump to close file
error:	mov dx,offset msg2	;address error message
	mov ah,09h	;display string function request
	int 21h	;call DOS
close:	mov dx,offset scb	;address of file control block
	mov ah,10h	;close file functioon request
	int 21h	;call DOS
done:	mov ah,00h	;terminate program funct request
	int 21h	;call DOS
code	ends	;end of code segment
	end start	;start is the entry point

#### Pascal Usage Example:

{Read Random Record( \$ 21)}

PROGRAM read\_random;

#### CONST

```

dos open file $ 0f;
dos close file = $ 10;
dos set dta = $ 1a;
dos random read = $ 21;
default drive= 0;

```

```

TYPE
  regpack = RECORD
    CASE integer OF
      1:(ax,bx,cx,dxx,bp,si,di,ds,es,flags:integer);
      2:(al,ah,bl,bh,cl,ch,dl,dh:byte);
    END;
  fcb = RECORD
    drive number:byte;
    file name:ARRAY[1..11]OF char;
    current block:integer;
    record size:integer;
    fill:ARRAY[1..17]OF byte;
    random record low:integer;
    random record high:integer;
  END;
VAR
  regs:regpack;
  this file fcb:fcb;
  dta:ARRAY[1..66]OF char;
  l:integer;

BEGIN
  WITH regs DO
    BEGIN
      this file fcb.drive number:= default drive;
      this file fcb.file name:= '/testfileeasc';
      ah:=dos set dta;
      ds:=seg(dta);
      dx:=ofs(dta);
      msdos(regs);
      ah:=dos open file;
      ds:=seg(this file fcb);
      dx:=ofs(this file fcb);
      msdos(regs);
      If al<>0THEN
        writeln("File not found.")
      ELSE
        BEGIN
          this file fcb.record size:= 66;

```

```

    this file fcb.random record high:=0;
    this file fcb.random record low:=24;
    ah:=dos random read;
    ds:=seg(this file fcb);
    dx:=ofs(this file fcb);
    msdos(regs);
    IF al<>0 THEN
        writeln('Error on random record read')
    ELSE
        FOR 1:=1 TO 66 DO
            write(dta[1]);
        ah:=dos close file;
        ds:=seg(this file fcb);
        dx:=ofs(this file fcb);
        msdos(regs);
    END;
    END;
END.

```

**C Usage Example:**

```

/* *
 * Read Random Record (0x21)
 */
#include <dos.h>

union REGS inregs,outregs;
struct fcb {
    char drive num;
    char filename[11];
    unsigned curblk;
    unsigned recsize;
    char fill[17];
    unsigned long ranrec;
};
struct fcb this file fcb = {0,"testfileasc"};
main()
{
    char dta [66];

```

```

inregs.x.dx = (int)&dta[0];
inregs.h.ah = 0x1a;
intdos(&inregs,&outregs);
inregs.x.dx = (int)&this file fcb;
inregs.h.ah = 0x0f;
intdos(&inregs,&outregs);
if(outregs.h.al != 0)
    printf("File not found.\n");
else{
    this file fccb.recsize = 66;
    this file fcb.ranrec = 24;
    inregs.x.dx = (int)&this file fcb;
    inregs.h.ah = 0x21;
    intdos(&inregs,&outregs);
    if(outregs.h.al != 0)
        printf("Error on random record read\n");
    else
        printf(dta);
    inregs.x.dx = (int)&this file fcb;
    inregs.h.ah = 0x10;
    intdos(&inregs,&outregs);
}
}

```

## DOS 功能调用 22H

### 写一随机记录

**功能调用:**

22H 写一随机记录到文件

**DOS 版本:**

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

此功能调用在已打开文件的随机位置写记录。使用此功能调用前，必须将文件 FCB 的随机记录字段设为写操作指定的位置。例如，设随机记录字段为 0，功能调用将写入文件的第一个记录。设随机记录字段为 9，则写第 10 个记录等等。参见第 60 页 FCB 的格式。

被写信息必须放在以盘传送地址起始的内存中。

如果正在写入的记录大小与设备的扇区大小不同，DOS 将把部分或全部信息保留在

内存缓冲区中，直到足够的数据累加到一完整扇区时才写或者关闭文件。

#### 入口参数：

调用前，需设置寄存器：

AH = 22H 指出功能调用号。

DS: DX 指向已打开文件的文件控制块 (FCB)。必须将随机记录字段设为要写的位置。在使用此功能调用前应确保记录尺寸字段已正确设置。

DTA 盘传送地址 (DTA) 指出要写入文件的数据的起始处。

#### 出口参数：

返回后，设置寄存器为：

FCB 功能调用将FCB (文件控制块) 的当前块字段和当前记录字段设成与随机记录字段一致。

AL 指出写的结果，表示如下：

00 写入完全成功。

01 磁盘已满，写失败。

02 功能调用在从盘传送区复制信息时，未传送整个记录就遇段结束。写失败。

#### 其它要求：

记录在文件中的位置和大小取决于 FCB 中的随机记录号和记录尺寸。打开文件时，DOS 自动将 FCB 的记录尺寸字段设为 128。用此记录尺寸随机写第三个记录 (随机记录号为 2) 将在 256 个字节的起始处写入 128 个字节。如果将记录尺寸改为 512，那么随机写第三个记录将在 1024 个字节的起始处写入 512 个字节。由于与不同的记录尺寸有关，因此在使用此功能调用前，应确保 FCB 的记录尺寸字段已正确设置。

被写文件不能设成只读属性，否则功能调用不能对它进行写操作。

为了在 DOS3.1 以上的网络环境中使用此功能调用，必须对含此文件的目录有写入权。

#### 参见：

15H.. 写下一个顺序文件记录

1AH.. 设盘传送地址

24H.. 设随机记录字段

28H.. 写多个随机记录

40H.. 向文件或设备写

#### 程序实例：

下面三个实例都是使用功能调用 22H 写一随机记录。程序将第 25 个新记录写入文件 TESTFILE.TMP 中。每个记录的长度为 66 字节。

#### Assembly Language Usage Example:

```
;  
;WRITE RANDOM RECORD (22H)  
;
```

```

code segment public
assume cs:code
        org      100h
start:  jmp      begin
        db      0,'testfiletmp'
curblk  dw ?
recsize dw ?
fill    db      17 dup(?)
ranrect1 dw      0
ranrech  dw      0
dta     db      'This record was written into record number 25,
              db      'of testfile.tmp.',0dh,0ah
msg1   db      'File not found.',0dh,0ah,'$'
msg2   db      'Error on random record write',0dh,0ah,'$'
begin: mov      ax,cs      ;set up ds
        mov      ds,ax      ;to same as cs
        mov      dx,offset dta ;address of disk transfer area
        mov      ah,1ah      ;set disk tansfer area funct reg
        int      21h      ;call DOS
        mov      dx,offset fcb ;address of file control block
        mov      ah,0fh      ;open file function request
        int      21h      ;call DOS
        cmp      al,0      ;does file exist?
        je      write      ;yes,write a random record
        mov      dx,offset msg1 ;address of nt found message
        mov      ah,09h      ;display string function request
        int      21h      ;call DOS
        jmp      done      ;wrap it up
write:  mov      recsize,66 ;set record size in fcb to 66
        mov      ranrectl,24 ;use 24 as ofst to rec 25 in file
        mov      dx,offset fcb ;address of file control block
        mov      ah,22h      ;write random record funct request
        int      21h      ;call DOS
        cmp      ah,0      ;was random write successful?
        je      close      ;jump to close file
        mov      dx,offset msg2 ;address error message
        mov      ah,09h      ;display srtng function request
        int      21h      ;call DOS
close:  mov      dx,offset fcb ;address of file control block

```

```

        mov      ah,10h      ;close file function request
        int      21h       ;call DOS
done:   mov      ah,00h      ;terminate program funct request
        int      21h       ;call DOS
code ends          ;end of code segment
        end      start      ;start is the entry point

```

**Pascal Usage Example:**

```
{ write Random Record ($ 22) }
```

```
PROGRAM write_random;
```

**CONST**

```

dos_open_file = $0f;
dos_closc_file = $10;
dos_set_dta = $1a;
dos_random_write = $22;
default_drive = 0;

```

**TYPE**

```

regpack = RECORD
  CASE integer OF
    1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
    2 : (al,ah,bl,bh,cl,ch,dl,dh:byte);
  END;

```

```
fcb = RECORD
```

```

  drive_number : byte;
  file_name : ARRAY [1..11] OF char;
  current_block : integer;
  record_size : integer;
  fill : ARRAY [1..17] OF byte;
  random_record_low : integer;
  random_record_high : integer;
END;

```

**VAR**

```

regs : regpack;
this_file_fcb : fcb;

```

```

dta : ARRAY [1..66] OF char ;
i : integer ;
BEGIN
  WITH regs DO
    BEGIN
      this_file_fcb.drive_number := default_drive ;
      this_file_fcb.file_name := 'testfiletmp' ;
      dta :=

      'This record was written into record number 25 of testfile.tmp.';

      dta[65] := chr($0d);
      dta[66] := chr($0a);
      ah := dos_set_dta ;
      ds := seg(dta) ;
      dx := ofs(dta) ;
      msdos(regs) ;
      ah := dos_open_file ;
      ds := seg(this_file_fcb) ;
      dx := ofs(this_file_fcb) ;
      msdos(regs) ;
      IF al < > 0 THEN
        writeln('File not found.')
      ELSE
        BEGIN
          this_file_fcb.record_size := 66;
          this_file_fcb.random_record_high := 0 ;
          this_file_fcb.random_record_low := 24 ;
          ah := dos_random_write ;
          ds := seg(this_file_fcb) ;
          dx := ofs(this_file_fcb) ;
          msdos(regs) ;
          IF al < > 0 THEN
            writeln ('Error on random record write') ;
            ah := dos_close_file ;
            ds := seg(this_file_fcb) ;
            dx := ofs(this_file_fcb) ;
            msdos(regs) ;
        END ;
      END ;

```

END.

**C Usage Example:**

```
* /
* Write Random Record (0x22)
*

#include <dos.h>

union REGS inregs,outregs;
struct fcb {
    char drive_num;
    char filename[11];
    unsigned curblk;
    unsigned recsize;
    char fill[17];
    unsigned long ranrec;
};
struct fcb this_file_fcb = {0,"testfiletmp"};
char dta [66] =
"This record was written into record number 25 of testfile.tmp.
\r\n";

main()
{
    inregs.x.dx = (int) & dta[0];
    inregs.h.ah = 0x1a;
    intdos(&inregs,&outregs);
    inregs.x.dx = (int) &this_file_fcb;
    inregs.h.ah = 0x0f;
    intdos(&inregs,&outregs);
    if (outregs.h.al != 0)
        printf("File not found.\n");
    else {
        this_file_fcb.recsize = 66;
        this_file_fcb.ranrec = 24;
        inregs.x.dx = (int) &this_file_fcb;
        inregs.h.ah = 0x22;
```

```

intdos(&inregs,&outregs);
if (outregs.h.ah != 0)
    printf("Error on random record write\n");
inregs.x.dx = (int)&this_file_fcb;
inregs.h.ah = 0x10;
intdos(&inregs,&outregs);
}
}

```

## DOS 功能调用 23H

### 取文件尺寸

**功能调用:**

23H 取文件尺寸

**DOS 版本:**

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

该功能检索文件大小 (以记录计) 并将此值放入 FCB 的随机记录字段。引用该功能调用前, 设置 FCB 记录尺寸字段, 指出每个记录的字节数。为得到文件尺寸的字节数, 置记录尺寸字段为 1 (每个记录一个字节), 参见第 60 页 FCB 格式。

**入口参数:**

调用前, 需设置寄存器:

AH = 23H 指出功能调用号。

DS: DX 指向当前打开文件的文件控制块FCB。引用该功能调用前必须置记录尺寸字段。

**出口参数:**

返回后, 设置寄存器为:

FCB 功能调用置随机记录字段指出文件的记录数, 如果文件中的字节数不能被记录尺寸除尽, 则随机记录字段的数值取整。

AL = 00H 调用成功, FCB列出文件中的记录数。

AL = FFH 当前目录没有指定FCB中所列文件的项目, FCB没被更新。

**程序实例:**

以下三个实例均是使用功能调用 23H, 以得到 TESTFILE.ASC 文件的大小。汇编语言的实例以十六进制显示文件的大小; PASCAL 及 C 语言的实例以十进制和十六进制显示文件的大小。

注意: 这些实例中首先设置记录尺寸字段为 1, 所以功能调用返回的是文件中一字节记录的数目 (即以字节计的文件尺寸)。

**Assembly Language Usage Example:**

```

;

;GET FILE SIZE (23H)
;

code segment public
assume cs:code

        org      100h
start:  jmp      begin
fcb      db       0,'testfileasc'
curblk  dw ?
recsize dw 1
fill    db       17 dup(?)
ranrec  db       0
string  db       8 dup (?),'$'
msg1    db       'File not found.',0dh,0ah,'$'
msg2    db       'Size of testfile.asc = ','$'
hexcode db       '0123456789ABCDEF'

;

; convert jhex unmberto two ascii characters
;

hexasc  proc      ;number to convert is in al
        mov      dl,10h      ;use dl to divide by 16
        xor      ah,ah      ;clear ah
        xor      bx,bx      ;clear bx
        div      dl          ;divide ax by 16 (in dl)
        mov      bl,al      ;move quotient to bl
        mov      al,hexcode[bx] ;look up hex code in table
        mov      dh,al      ;ret high digit of hex code in dh
        mov      bl,ah      ;move remainder to bl
        mov      ah,hexcode[bx] ;look up hex code in table
        mov      dl,al      ;ret low digit of hex code in dl
        ret      ;return with answer in dx
hexasc endp      ;end of procedure
;

;main program
;

begin:  mov      ax,cs      ;set up ds
        mov      ds,ax      ;to same as cs
        mov      dx,offset dta ;address of file control block
        mov      ah,23h      ;get file size function request

```

	int	21h	;call DOS
	cmp	ah,0	;dose file exist?
	je	disp	;yes,display the answer
	mov	dx,offset msg1	;address of not found message
	mov	ah,09h	;display string function request
	int	21h	;call DOS
	jmp	done	;wrap it up
disp:	mov	dx,offset msg2	;address file size message
	mov	ah,09h	;display string function request
	int	21h	;call DOS
	mov	si,3	;last byte in ranrec field of fcb
	mov	di,0	;first byte in output string
	mov	cx,1	;loop all 4 bytes of ranrec field
next:	mov	al,byte ptr ranrec[si]	;get next byte to convert
	call	hexasc	;convert from hex to ascii chars
	mov	string[di],dh	;save results in output string
	inc	di	;next byte in output string
	mov	string[di],dl	;save results in output string
	inc	di	;next byte in output string
	dec	si	;next byte in ranrec field of fcb
	loop	next	;all four bytes in ranrec field
	mov	dx,offset string	;address of converted number
	mov	ah,09h	;display string function request
	int	21h	;call DOS
done:	mov	ah,00h	;terminate program funct request
	int	21h	;call DOS
code ends			;end of code segment
	end	start	;start is the entry point

### Pascal Usage Example:

{ write Random Record (\$ 23) }

PROGRAM get\_file\_size;

#### CONST

```
dos_get_file_size = $ 23;
default_drive = 0;
```

```

TYPE
  regpack = RECORD
    CASE integer OF
      1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
      2 : (al,ah,bl,bh,cl,ch,dl,dh:byte);
    END;
  fcb = RECORD
    drive_number:byte;
    file_name:ARRAY [1..11] OF char;
    current_block:integer;
    record_size:integer;
    fill:ARRAY [1..17] OF byte;
    random_record_low:integer;
    random_record_high:integer;
  END;
VAR
  regs:regpack;
  this_file_fcb:fcb;
  dta:ARRAY [1..66] OF char;

BEGIN
  WITH regs DO
    BEGIN
      this_file_fcb.drive_number := default_drive;
      this_file_fcb.file_name := 'testfileasc';
      this_file_fcb.record_size := 1;
      ah := dos_get_file_size;
      ds := seg(this_file_fcb);
      dx := ofs(this_file_fcb);
      msdos(regs);
      IF al < > 0 THEN
        writeln('File not found.')
      ELSE
        writeln('Size of file in decimal = ',
               this_file_fcb.random_record_low);
    END;
END;

```

### C Usage Example:

```
* /
 * Write Random Record (0x23)
 */

#include <dos.h>

union REGS inregs,outregs;
struct fcb {
    char drive_num;
    char filename[11];
    unsigned curblk;
    unsigned recsize;
    char fill[17];
    unsigned long ranrec;
};
struct fcb this_file_fcb = {0,"testfileasc"};

main()
{
    this_file_fcb.recsize = 1;
    inregs.x.dx = (int) &this_file_fcb;
    inregs.h.ah = 0x23;
    intdos(&inregs,&outregs);
    if (outregs.h.al != 0)
        printf("File not found.\n");
    else {
        printf("Size of testfile.asc in hex = %X\n",
               this_file_fcb.ranrec);
        printf("Size of testfile.asc in decimal = %lu\n",
               this_file_fcb.currec);
    }
}
```

### DOS 功能调用 24H 设随机记录字段

功能调用：

## 24H 置 FCB 的随机记录字段

DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明:

该功能调用置 FCB 的随机记录字段以匹配当前块和当前记录字段。该操作顺序 I/O 改为随机 I/O 时很有用 (顺序 I/O 使用当前块和当前记录字段, 随机 I/O 使用随机记录字段)。

入口参数:

调用前, 需设置寄存器:

AH = 24H 指出功能调用号。

DS: DX 指向打开文件的文件控制块FCB, 参见52页FCB格式。

出口参数:

返回后, 设置寄存器为:

FCB 的随机记录字段置为当前块字段和记录字段所对应的文件地址。

程序实例:

下面三个实例均是使用功能调用 24H 置文件 TESTFILE.ASC 的随机记录字段。这些程序读出并显示第 25 个记录每个记录长 66 字节。

这些程序不同于功能调用 21H 中的例子, 因为它们使用随机记录字段识别要读的记录。请注意, 程序首先置当前记录字段为 24, 文件批开时, 自动置当前块字段为 0, 因此, 当清示 24H 调用后, 随机记录字段指向第 25 个记录。

### Assembly Language Usage Example:

```
; ;  
;SET RANDOM RECORD (24H)  
;  
code segment public  
assume cs:code  
    org      100h  
start:  jmp      begin  
fcb      db      0,'testfiletmp'  
curblk   dw ?  
recsize  dw ?  
fill     db      16 dup(?)  
currec   db      24  
ranrect1 dw      0  
ranrech  dw      0  
dta      db      66 dup (?),'$'  
msg1    db      'File not found.',0dh,0ah,'$'  
msg2    db      'Error on random record write',0dh,0ah,'$'
```

begin:	mov	ax,cs	;set up ds
	mov	ds,ax	;to same as cs
	mov	dx,offset dta	;address of disk transfer area
	mov	ah,1ah	;set disk transfer area funct reg
	int	21h	;call DOS
	mov	dx,offset scb	;address of file control block
	mov	ah,0fh	;open file function request
	int	21h	;call DOS
	cmp	al,0	;does file exist?
	je	write	;yes, write a random record
	mov	dx,offset msg1	;address of nt found message
	mov	ah,09h	;display string function request
	int	21h	;call DOS
	jmp	done	;wrap it up
read:	mov	recsize,66	;set record size in scb to 66
	mov	ranrecl,24	;set random record funct request
	int	21h	;address of file control block
	mov	ah,21h	;read random record funct request
	int	21h	;call DOS
	cmp	ah,0	;was random read successful?
	jnc	error	;no display error message
	mov	dx,offset dta	;address disk transfer area
	mov	ah,09h	;display string function request
	int	21h	;call DOS
	jmp	close	;jump to close file
error:	mov	dx,offset msg2	;address error message
	mov	ah,09h	;display string function request
	int	21h	;call DOS
close:	mov	dx,offset scb	;address of file control block
	mov	ah,10h	;close file function request
	int	21h	;call DOS
done:	mov	ah,00h	;terminate program funct request
	int	21h	;call DOS
code ends			;end of code segment
	end	start	;start is the entry point

#### Pascal Usage Example:

{ write Random Record (\$ 24) }

PROGRAM set\_random\_record;

CONST

```
dos_open_file = $0f;
dos_colsc_file = $10;
dos_set_dta = $1a;
dos_random_write = $22;
dos_set_random_record = $24;
default_drive = 0;
```

TYPE

regpack = RECORD

```
CASE integer OF
  1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
  2 : (al,ah,bl,bh,cl,ch,dl,dh:byte);
END;
```

fcb = RECORD

```
drive_number : byte;
file_name : ARRAY [1..11] OF char;
current_block : integer;
record_size : integer;
fill : ARRAY [1..17] OF byte;
current_record : byte;
random_record_low : integer;
random_record_high : integer;
END;
```

VAR

```
regs : regpack;
this_file_fcb : fcb;
dta : ARRAY [1..66] OF char;
i : integer;
```

BEGIN

WITH regs DO

BEGIN

```
this_file_fcb.drive_number := default_drive;
```

```

this_file_fcb.file_name := 'testfileasc';
ah := dos_set_dta;
ds := seg(dta);
dx := ofs(dta);
msdos(regs);
ah := dos_open_file;
ds := seg(this_file_fcb);
dx := ofs(this_file_fcb);
msdos(regs);
IF al < > 0 THEN
    writeln ('File not found.')
ELSE
BEGIN
    this_file_fcb.record_size := 66;
    this_file_fcb.current_record := 24;
    ah := dos_set_random_record;
    ds := seg(this_file_fcb);
    dx := ofs(this_file_fcb);
    msdos(regs);
    ah := dos_random_read;
    ds := seg(this_file_fcb);
    dx := ofs(this_file_fcb);
    msdos(regs);
    IF al < > 0 THEN
        writeln ('File not found.')
    ELSE
FOR i:=1 TO 66 DO
    (write(dta)[i]);
    ah := dos_close_file;
    ds := seg(this_file_fcb);
    dx := ofs(this_file_fcb);
    msdos(regs);
END;
END;

```

### C Usage Example:

\* /

\* Write Random Record (0x22)

\* /

```
#include <dos.h>
```

```
union REGS inregs,outregs;
```

```
struct fcb {
```

```
    char drive_num;
```

```
    char filename[11];
```

```
    unsigned curblk;
```

```
    unsigned recsize;
```

```
    char fil[16];
```

```
    char fil[16];
```

```
    char currec;
```

```
    unsigned long ranrec;
```

```
};
```

```
struct fcb this_file_fcb = {0,"testfileasc"};
```

```
main()
```

```
{
```

```
char dta [66] =
```

```
    inregs.x.dx = (int) & dta[0];
```

```
    inregs.h.ah = 0x1a;
```

```
    intdos(&inregs,&outregs);
```

```
    inregs.x.dx = (int) &this_file_fcb;
```

```
    inregs.h.ah = 0x0f;
```

```
    intdos(&inregs,&outregs);
```

```
    if (outregs.h.al != 0)
```

```
        printf("File not found.\n");
```

```
    else {
```

```
        this_file_fcb.recsize = 66;
```

```
        this_file_fcb.currec = 24;
```

```
        inregs.x.dx = (int) &this_file_fcb;
```

```
        inregs.h.ah = 0x24;
```

```
        inregs.x.dx = (int) &this_file_fcb;
```

```
        inregs.h.ah = 0x21;
```

```
        intdos(&inregs,&outregs);
```

```
        if (outregs.h.al != 0)
```

```
            printf("Error on random record read.\n");
```

```
else
    printf(dta);
inregs.x.dx = (int) &this_file_fcb;
inregs.h.ah = 0x10;
intdos(&inregs,&outregs);
```

## DOS 功能调用 25H

### 置中断向量

**功能调用:**

25H 置中断向量

**DOS 版本:**

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

将 4 个字节的地址放入中断向量表中，中断向量含中断处理程序（中断句柄）的地址。当相应于中断号的中断发生时（或程序引用 INT 指令，并指定了该中断号时），就调用该中断处理程序。

**入口参数:**

调用前，需设置寄存器：

AH = 25H 指出功能调用号。

AL = 中断号中断处理程序适用的中断号范围0~225 (FFH)。

DS: DX 中断处理程序的4字节地址，DS为基地址，DX为偏移地址。

**出口参数:**

无。

**参见:**

35H.. 取中断向量。

**程序实例:**

下述汇编语言实例使用功能调用 25H 在中断向量表中置一个入口，程序建立一个特定的 Ctrl-Break 处理程序，然后，程序继续显示信息，直到按 Ctrl-Break。该功能调用没有 Pascal 及 C 语言实例，因为这是低级操作，只能使用汇编语言来完成。

**Assembly Language Usage Example:**

```
; SET INTERRUPT VECTOR (25H)

;
code    segment public
assume  cs:code,ds:code
        org     100h
start: jmp    begin
brkflag db     0
```

```

msg1    db      Control-break was hit!',0dh,0ah, '$'
msg2    db      New control-break handler installed — — try
it.',    db      0dh,0ah, '$'
;
;control-break interrupt handler
;
break   proc far           ;start of control-break handler
        push   ax           ;save ax
        push   dx           ;save dx
        mov    dx,offset msg1 ;address of control-break message
        mov    ah,09h         ;display string function request
        int    21h           ;call DOS
        mov    brkflag,01h    ;set breakflag
        pop    dx           ;restore dx
        pop    ax           ;restore ax
        iret              ;return
break   endp               ;end of procedure
;
;main program
;
begin:  mov    ax,cs          ;set up ds
        mov    ds,ax          ;to same as cs
        mov    brkflag,00h    ;initialize break flag
        mov    al,23h          ;dos ctrl-break interrupt vector
        mov    dx,offset break ;address of break interrupt vector
        mov    ah,25h          ;set disk transfer area funct reg
        int    21h           ;call DOS
disp:   mov    dx,offset msg2 ;address of message
        mov    ah,09h         ;display string function request
        int    21h           ;call DOS
        cmp    brkflag,01h    ;was control-break hit?
        jne    disp           ;no, display message again
        mov    ah,00h          ;terminate program funct request
        int    21h           ;call DOS
code   ends                ;end of code segment
end    start               ;start is the entry point

```

## DOS 功能调用 26H

### 创建一新程序段

```

start: jmp      begin
begin: mov      ax,cs           ;set up ds
              mov      ds,ax           ;to same as cs
;
              mov      dx,segment newprog;seg addr of new program's psp
              mov      ah,26h          ;create psp function request
              int      21h             ;call DOS
;

;Here is where you load the new program and jump to its start
;

        mov      ah,0           ;terminate program funct request
        int      21h             ;call DOS
code  ends            ;end of code segment
        end      start           ;start is the entry point

```

## DOS 功能调用 27H

读多个随机记录

功能调用:

27H 从文件中读多个随机记录

DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明:

该功能调用除类似功能调用 21H (读一个随机记录)、还可从一个文件中的随机位置读多个记录。请求该调用前，必须置文件的 FCB 的随机记录字段以指明起始记录、如果 FCB 中记录尺寸字段未置好，还应重新设置。

入口参数:

调用前，需设置寄存器:

AH = 27H 指出功能调用号。

DS: DX 指向已打开文件的文件控制块FCB，必须置随机记录字段指明要读的第一个记录，还应保证记录尺寸字段置的正确，参见第 60 页 FCB 格式。

CX 指出要读的记录数，此值一定不能为0，功能调用用以 FCB 随机记录字段中指定的记录开始读记录。

出口参数:

返回后，设置寄存器为:

FCB 给刚读过的最后一个记录后的那个记录设置 FCB 的当前块字段和当前记录字段。

AL = 00; 读出完全成功。

AL = 01; 遇到文件结束而且最后一个记录是完整的。

**功能调用:**

26H 创建一新程序段

**DOS 版本:**

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

对于 DOS2.0 版或更高版本, 应避免使用该功能调用, 因为它不允许装入.EXE 文件。这时, 可使用 EXEC (4BH) 功能调用。

该功能调用在装入一个子程序或覆盖模块到内存, 为创建一具新的程序段时使用, 它拷贝当前程序的程序段中前 256 个字节 (十六进制 100 字节) (即程序段前缀或 PSP) 到指定的新位置。参见第 20 页 PSP 的完整格式。

当创建 PSP 时, 此功能调用更新 PSP 中 06H 处的内存大小的信息, 并在以 0AH 开始的新段中保存用于中断 22h, 23h, 24h 的当前结束处理、Ctrl-Break 退出和严重错误的地址。当新程序退出时, DOS 将从新程序段前缀恢复这些值。

一旦建立新程序段前缀, 便可将程序代码 (.COM 文件) 立即装入新程序段前缀后的区域里。该功能调用不能装入.EXE 文件, 因为这种文件的格式结构更复杂。若要安装 .EXE 文件, 见功能调用 4BH。

**入口参数:**

调用前, 需设置寄存器:

AH = 26H 指出功能调用号。

DX 新程序段的起始位置。

**出口参数:**

无。

**参见:**

4BH.. 装入或执行程序 (EXEC)。

**程序实例:**

以下汇编语言实例使用功能调用 26H 创建一个新的程序段。下面列出的不是一个完整例子, 为使其完整, 在请求 26H 功能调用后, 需装入一个程序 (.COM), 并用 far jump 指令转移到此程序。

该功能调用没有 Pascal 及 C 语言实例, 因为创建一个新的程序段是低级操作, 只使用汇编语言。

#### Assembly Language Usage Example:

```
; ;CREATE PRCGRAM SEGMENT PREFIX (26H)
;
code    segment public
assume cs:code,ds:code
org      100h
```

```

start: jmp      begin
begin: mov      ax,cs           ;set up ds
              mov      ds,ax           ;to same as cs
;       mov      dx,segment newprog;seg addr of new program's psp
              mov      ah,26h          ;create psp function request
              int      21h             ;call DOS
;
;Here is where you load the new program and jump to its start
;
        mov      ah,0           ;terminate program funct request
        int      21h             ;call DOS
code   ends
end     start           ;start is the entry point

```

## DOS 功能调用 27H

### 读多个随机记录

#### 功能调用:

27H 从文件中读多个随机记录

#### DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

#### 说明:

该功能调用除类似功能调用 21H (读一个随机记录), 还可从一个文件中的随机位置读多个记录。请求该调用前, 必须置文件的 FCB 的随机记录字段以指明起始记录, 如果 FCB 中记录尺寸字段未置好, 还应重新设置。

#### 入口参数:

调用前, 需设置寄存器:

AH = 27H 指出功能调用号,

DS: DX 指向已打开文件的文件控制块FCB。必须置随机记录字段指明要读的第一个记录, 还应保证记录尺寸字段置的正确, 参见第 60 页 FCB 格式。

CX 指出要读的记录数, 此值一定不能为0, 功能调用用以FCB随机记录字段中指定的记录开始读记录。

#### 出口参数:

返回后, 设置寄存器为:

FCB 给刚读过的最后一个记录的那个记录设置FCB的当前块字段和当前记录字段。

AL = 00; 读出完全成功。

AL = 01; 遇到文件结束而且最后一个记录是完整的。

```

dta    db     330 dup (?), '$'
msg1   db     'File ot found.', 0dh, 0ah, '$'
msg2   db     'Error on random record read', 0dh, 0ah, '$'
begin: mov ax,cs           ;set up ds
        mov ds, ax          ;to same as cs
        mov dx, offset dta  ;address of disk transfer area
        mov ah, lah          ;set dta function request
        int 2Lh              ;set dta function request
        int 2Lh              ;call DOS
        mov dx, offset fcb  ;address of file control block
        mov ah, 0fh            ;open file function request
        int 2Lh              ;call DOS
        cmp al,0              ;does file exist?
        je read               ; yes, read random records
        mov dx, offset msg1  ;address of not found message
        mov ah, 09h            ;display string function request
        int 2Lh              ;call DOS
        jmp done               ;wrap it up
read:   mov recsize, 66      ;set record size in fcb to 66
        mov ranrec1, 24       ;use 24 as ofst to rec 25 in file
        mov cx,5              ;read 5 records
        mov dx, offset fcb  ;address of file control block
        mov ah, 27h            ;read mult ran recs funct request
        int 2Lh              ;call DOS
        cmp al,0              ;was random read successful?
        jne error              ;no, display error message
        mov dx, offset dta  ;address disk transfer area
        mov ah, 09h            ;display string function request
        int 2Lh              ;call DOS
        jmp close              ;jump to close file
error:  mov dx, offset msg2 ;address error message
        mov ah, 09h            ;display string function request
        int 2Lh              ;call DOS
close:  mov dx, offset fcb ;address of file control block
        mov ah, 10h            ;close file function request
        int 2Lh              ;call DOS
done:   mov ah, 00h          ;terminate program funct request
        int 2Lh              ;call DOS
Code ends             ;end of code segment

```

end start ;  
;start is the entry point

**Pascal Usage Example:**

{Read Multiple Random Records (\$ 27)}

PROGRAM read multiple random;

CONST

```
dos open file = $0f;  
dos close file = $10;  
dos set dta = $1a;  
dos multiple random read = $27;  
default drive = 0;
```

TYPE

```
regpack = RECORD  
  CASE integer OF  
    1 : (AX,BX,CX,DX,BP,SI,DI,DS,ES,FLAGS:INTEGER);  
    2 : (AL,AH,BL,BH,CL,CH,DL,DH:BYTE);  
  END;
```

fcb = RECORD

```
  dive number: byte;  
  ✓  file-name: ARRAY [1..11] OF char;  
  current block: integer;  
  record size: integer;  
  fill: ARRAY [1..17] OF byte;  
  random record low: integer;  
  random record-High: integer;
```

END;

VAR

```
  regs: regpack;  
  this file fcb: fcb;  
  dta: ARRAY [1..330] OF char;  
  i: integer;
```

BEGIN

WITH regs DO

BEGIN

```
      this file-fcb.drive-number := default-drive;
```

```

this file fcb.file name := 'testfileasc';
ah := dos set dat;
ds := seg(dta);
dx := ofs(dta);
msdos(regs);
ah := dos open file;
ds := seg(this file fcb);
dx := ofs(this file fcb);
msdos(regs);
IF al < > 0 THEN
  tgfwrutekb('File not found.')
ELSE
BEGIN
  this file fcb.record size := 66;
  this file fcb.random record-High := 0 ;
  this file fcb.random record low := 24;
  cx := 5 ;
  ah := dos multiple random read;
  as := seg(this file-fcb);
  dx := ofs(this file fc);
  msdos(regs);
  IF al < > 0 THEN
    writeln('Error on random record read')
  ELSE
    FOR I:= 1 to 330 DO
      write(dta[i]);
      ah := dos close file;
      ds := seg(this file fcb);
      dx := ofs(tis file fcb);
      msdos(regs);
    END;
  END;
END;

```

#### C Usage Example:

```

*
* Read Multiple Random Records (0x27)
*/
#include <dos.h>
```

```

union REGS inregs, outregs;

struct fcb {
    char drive num;
    char filename[11];
    unsigned curblk;
    unsigned recsize;
    char fill[17];
    unsigned long ranrec;
};

struct fcb tis file fcb + {0,"testfilcase"};

main()
{
    char dta [330];

    inregs.x.dx = (int) &dta[0];
    inregs.h.ah = 0x1a;
    intdos(&inregs,&outregs);
    inregs.x.dx = (int) &tis file fcb;
    inregs.h.ah = 0x0f;
    intdos(&inregs,&outregs);
    if (outregs.h.al != 0)
        printf("File not found.\n");
    else {
        this file fcb.recsize = 66;
        this file fcb.ranrec = 24;
        inregs.x.cx = 5;
        inregs.x.dx = (int) &this file fcb;
        inregs.h.ah = 0x27;
        intdos (&inregs,&outregs);
        if (outregs.h.al != 0)
            printf ("Error on random record read\n");
        else
            printf(dta);
        inregs.x.dx = (int) &this file fcb;
        inregs.h.ah = 0x10;
        intdos(&inregs,&outregs);
    }
}

```

```
}
```

## DOS 功能调用 28H

写多个随机记录

功能调用:

28H 写多个随机记录

DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明:

该功能调用除类似功能调用 22H (写一个随机记录), 还可在文件任一随机位置写多个记录。在请求该功能调用之前, 必须置文件的 FCB 的随机记录字段以指明写操作的位置, 加外, 还必须置 CX 指明要写的记录数。

如果 CX 为 0, 该功能调用不写任何记录, 但它调整文件的长度, 使得在 FCB 的随机记录字段中指定的记录为文件的最后一个记录, DOS 相应地延长或截断文件。

在将信息实际写入磁盘前, 一般 DOS 用一个内存缓冲区积累数据, 该缓冲区的大小等于设备的扇区大小。缓冲区满后, DOS 将写满整个磁盘扇区。因此, 如果所写的记录大小与设备扇区不等时, DOS 在一个内部缓冲区中保留部分信息数据下溢, 直到累积数据足以写满另一完整扇区, 或者关闭文件。

入口参数:

调用前, 需设置寄存器:

AH = 28H 指出功能调用号。

DS: DX 指向已打开文件的文件控制块FCB, 必须置随机记录字段以指明写的位置 (如 CX = 0, 则为文件的最后一个记录)。在请求功能调用之前, 应确保正确设置记录尺寸字段, 参见 60 页 FCB 的格式。

CX 指出要写的记录数, 该功能调用从FCB随机记录字段指定的位置开始写记录。

以该寄存器内容为 0 时来核对文件长度所以在 FCB 随机记录字段中规定的记录处文件就结束。

DTA 指明要写入文件的数据起始位置。

出口参数:

返回后, 设置为:

FCB 置 FCB 当前字段和当前记录字段为所写最后一个记录之后的那个记录。

AL = 00 写入完全成功。

AL = 01 磁盘没有足够的空间存放所有记录, 因此没有写任何信息。

AL = 02 功能调用还没有从盘传送区传送所有要求的数据便遇到了段结束, 写和即被终止。

CX 实际被写入的记录数。

其它要求：

记录的大小及它们在文件中的位置，不仅取决于 FCB 中的随机记录数，还决定于记录尺寸。打开文件时，DOS 自动置 FCB 的记录尺寸字段为 128。请求系统调用前，必须正确设置 FCB 的记录尺寸字段。

在 DOS3.1 以上版本以的网络环境中使用此功能调用时，必须对含此文件的目录有写入权。

参见：

- 15H.. 写下一顺序文件
  - 1AH.. 设盘传送地址
  - 22H.. 写一随机记录
  - 24H.. 设随机记录字段
  - 28H.. 写多个随机记录字段
  - 40H.. 写至文件或设备

程序案例：

以下三个实例均使用功能调用 28H 功能调用在 TESTFILE.TMP 文件中写入两个新记录。

### Assembly Language Usage Example:

```

; WRITE MULTIPLE RANDOM RECORDS (28H)

code segment public
assume cs:code, ds:code

org 100h

start: jmp begin

fcb db 0,'testfiletmp'
curblk dw ?
recsize dw ?
fill db 17 dup (?)
ranrecl dw 0
ranrech dw 0
dta db 'This record was written into record number 25'
      db 'of testfile.tmp.',0dh,0ah
      db 'This record was written into record number 26'
      db 'of testfile.temp.',0dh,0ah
msg1 db 'File not found.',0dh,0ah,'$'
msg2 db 'Error on random record write',0dh,0a,'$'

begin: mov ax,cs          ;set up ds

```

```

        mov ds,ax          ;to same as cs
        mov dx,offset dta ;address of disk transfer area
        mov ah,1ah         ;set dta function request
        int 21h            ;call DOS
        mov dx,offset fcb ;address of file control block
        mov ah,0fh          ;open file function request
        je write            ;yes, write random records
        mov dx,offset msg1 ;address of not found message
        mov ah,09h          ;display string function request
        int 21h            ;call DOS
        jmp done            ;wrap it up

write:  mov recsize,66   ;set record size in fcb to 66
        mov ranrecl,24    ;use 24 as ofst to rec 25 in file
        mov cx,2           ;write 2 records
        mov dx,offset fcb ;address of file control block
        mov ah,28h          ;write multiple ran recs funct req
        int 21h            ;call DOS
        cmp al,0           ;was mult ran write successful?
        je close             ;jump to close file
        mov dx,offset msa2 ;address error message
        mov ah,09h          ;display string function request
        int 21h            ;call DOS

close:  mov dx,offset fcb ;address of file control block
        mov ah,10h          ;close file function request
        int 21h            ;call DOS

done:   mov ah,00h          ;terminate program funct request
        int 21h            ;call DOS

code ends           ;end of code segment
        end start           ;start is the entry point

```

#### Pascal Usage Example:

{ Write Multiple Random Records (\$ 28) }

PROGRAM write\_multiple\_random;

#### CONST

```

dos open file + $0f;
dos close file = $10;
dos set dta = $1a;
dos multiple random write = $28;

```

```

default drive = 0;

TYPE
  regpack = RECORD
    CASE integer OF
      1 : (ax,bx,cx,dx,bp,si,di,ds/es,flags:integer);
      2 : (al,ah,bl,bh,cl,ch,dl,dh:byte);
    END;
  fcb = RECORD
    drive number : byte;
    file name : ARRAY [1..11] OF char,
    current-block : integer;
    record size : integer;
    fill : ARRAY [1..17] OF byte;
    random record low : integer;
    random record high : integer;
  END;
VAR
  regs : regpack;
  this file-fcb : fcb;
  dta : ARRAY [1..2,1..66] OF char;
  i : integer;
BEGIN
  WITH regs DO
    BEGIN
      this file fcb.drive number := default drive;
      this file fcb.file name := 'testfiletmp';
      dta[1]:=
```

'This record was written into record number 25 of testfile.temp.';

```

      dta[1..65] := chr($0d);
      dta[1..66] := chr($0a);
      dta[2]:=
```

'This record was written into record number 26 of testfile.temp.t';

```

      dta[2..65] := chr($0d);
      dta[2..66] := chr($0a);
      ah := dos set dat;
      ds := scg(dta);
      dx := ofs(dta);
      msdos(regs);
      ah := dos open file;
```

```

        ds := seg(this file fcb);
        dx := ofs(this file fcb);
        msdos(regs);
        IF al < > 0 THEN
            writeln('File not found.')
        ELSE
            BEGIN
                this file fcb.record size := 66;
                this file fcb.random-record-high := 0 ;
                this file fcb.random record low := 24 ;
                cx := 2 ;
                ah := dos multiple random write;
                ds := seg(this file fcb);
                dx := ofs(this file fcb);
                msdos(regs);
                IF al < > 0 THEN
                    writeln('Error on random record write');
                    ah := dos close file;
                    ds := seg(this file fcb);
                    dx := ofs(this file fcb);
                    msdos(regs);
                END ;
            END ;
        END.

```

#### C Usage Example:

```

/*
 * Write Multiple Random Records (0x28)
 */

```

```
#include <dos.h>
```

```

union REGS inregs, outregs;
struct fcb {
    char drive num;
    char filename[11];
    unsigned curblk;
    unsigned recsize;
    char fill[17];

```

```

        unsigned long ranrec;
    };

    struct fcb tis file fcb = {0,"testfiletmp"};
    char dta {2} [66] =
    {
        "This record was written into record number 25 of testfile.temp.\n",
        "\r\n",
        "This record was written into record number 26 of testfile.tmp \n",
        "\r\n"};
    }

main()
{
    inregs.x.dx = (int) &dta[0][0]
    inregs.h.ah = 0x1a;
    intdos(&inregs,&outregs);
    inregs.x.dx = (int) &this file fcb;
    inregs.h.ah = 0x0f;
    intdos(&inregs,&outregs);
    if (outregs.h.al != 0)
        printf("File not found.\n");
    else {
        this file fcb.recsiz = 66;
        this file fcb.ranrec = 24;
        inregs.x.cx = 2;
        inregs.x.dx = (int) &tis file fcb;
        inregs.h.ah = 0x28;
        intdos(&inregs,&outregs);
        if (outregs.h.al != 0)
            printf("Error on random record writ\n");
        inregs.x.dx = (int) &this file fcb;
        inregs.h.ah = 0x10;
        intdos(&inregs,&outregs);
    }
}

```

## DOS 功能调用 29H

分析文件名

功能调用:

## 29H 分析文件名

DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明:

该功能调用分析含有文件名的命令行 (由 DS: SI 指出), 并为要分析的文件名创建一个 FCB。它用 AL 的内容决定如何分析文件名。

该功能调用认为下列字符为文件名的分隔符:

..;, =+TABSPACE.

文件名的终止符包括: / < >; “□及任何控制字符。该功能调用认为文件名的格式为

d: filename.ext

由于此功能调用在 DOS2.0 以前的版本可行, 因此, 它不能处理具有路径名的文件名 (如 d: directory\name.ext)。

如果功能调用找到一个文件名, 便创建一个未打开的 FCB。如果文件名不包含指定的驱动器, 便认为是缺省驱动器。如果文件名忽略扩展名, 则认为是空格。如果“\*”出现在文件名或扩展名中, 功能调用置那部分文件名中所有剩余字符为?。

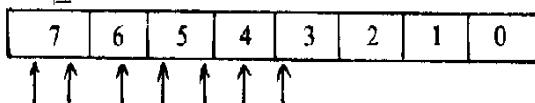
入口参数:

调用前, 需设置:

AH = 29H 指出功能调用号。

AL 第0至3位决定如何分析

位



第0位: = 1 则跳过命令行开始的所有分隔符。

= 0 则认为文件名开始于命令行的第一字节。

第1位: = 1 仅当命令行中指定驱动器时, 才在新的FCB中设置驱动器ID字节, 即让程序建立有其自己省缺驱动器的 FCB。

= 0 不改变驱动器ID字节。

第2位: = 1 仅当命令行包含文件名时, 才改变新FCB中的文件名。这样可允许程序在请功能调用没有指定文件名时, 建立缺省文件名。

= 0 如果命令行中不包含文件名, 则FCB中的文件名置成8个空格。

第3位: = 1 仅当命令行中包含扩展名时, 才改变FCB中的文件扩展名。

= 0 根据命令行中文件名, 在FCB中置文件扩展名 (如果命令行未指定扩展名, FCB 中的扩展名便置为空格)。

DS: SI 指向要分析的命令行。

ES: DI 指向要分析的命令行。

**出口参数:**

调用前, 需设置:

ES: DI 指向已格式化的FCB的首字节, 如命令行中不含有效文件名, 则  
ES: DI+1 包括一个空格 (ASCII 码 20H), 参见 FCB 的格式。

DS: SI 指向已分析的文件名之后命令行中的第一个字符。

AL = 00 操作成功, 已分析一文件名。

= 01 操作成功, 在文件名或扩展名中有“\*”或“?”

= FFH 由于指定驱动器 (A; akB; 等等) 无效, 操作失败。

**程序实例:**

以面三个实例均使用功能调用 29H 分析键盘键入的文件名, 程序还可报告是否成功地分析了一个文件名; 是否文件名中包括通配符“\*”或“?”; 是否由于指定驱动器无效而未分析文件名。

**Assembly Language usage Example: \$ \$**

```
;  
; GET CURRENT DATE (2AH)  
;  
code segment public  
assume cs:code, ds:code  
    org      100h  
start: jmp      begin  
string1 db      'The date is'  
string2 db      2 dup (0),//  
string3 bd      2 dup (0),//  
string4 db      4 dup (0),0dh,0ah,'$'  
month   db      0  
day     db      0  
year    dw      0  
;  
; convert number to ascii  
; numasc proc      ;number to convert is in ax,  
;                   ;ofst to end of string is in bx  
        mov si,10      ;use si to divide by 10  
next:  xor dx,dx      ;clear dx  
        div si      ;divide ax by 10 (in si)  
        add dx,'0'      ;convert remainder to ascii number  
        dec bx      ;go to next char position
```

```

        mov [bx],d!      ;store character in the string
        or ax,ax       ;any more digits to convert?
        jnz next      ;yes, get next digit
        ret          ;no, return
        numasc endp   ;end of procedure

;

; main program

;

begin;    mov ax,cs      ;set up ds
        mov ds,ax      ;to same as cs
        mov ah,2ah     ;get date function request
        int 21h        ;call DOS
        mov byte ptr month,dh  ;save month
        mov byte ptr day,al   ;save day
        mov bx,offset string2 + 2 ;get address of end of string
        xor ax,ax      ;clear ax
        mov al,byte ptr month  ;month
        call numasc    ;convert binary number to ascii
        mov bx,offset string3 + 2 ;get address of end of string
        xor ax,ax      ;clear ax
        mov al,byte ptr day   ;day
        call numasc    ;convert binary number to ascii
        mov bx,offset string4 + 4 ;get address of end of string
        mov ax,word ptr year  ;year
        call numasc    ;convert binary number to ascii
        mov dx,offset string1  ;start of complete message
        mov ah,09h     ;display string function request
        int 21h        ;call DOS
        mov ah,00h     ;terminate program funct request
        int 21h        ;call DOS
code ends           ;end of code segment
end start          ;start is the entry point

```

#### Pascal Usage Examples:

{ Get Current Date (B7e) }

PROGRAM get\_date;

```

CONST
  dos get date = $2a;
TYPE
  regpack = Record
    CASE integer OF
      1 : (ax,bx,cx,dx,bp,si,di,ds,cs,flags:integer) ;
      2 : (al,ah,bl,bh,cl,ch,dl,dh:byte) ;
  END;

VAR
  regs : regpack ;
BEGIN
  WITH regs DO
    BEGIN
      ah := dos get date ;
      msdos(regs) ;
      writeln('The date is ',dh,' / ',dl,' / ',cx);
    END;
END.

```

#### C Usage Example:

```

/*
 * Get Current Date (0x2a)
 */

```

```

#include <dos.h>

union REGS inregs, outregs;

main()
{
    inregs.h.ah = 0x2a;
    intdos(&inregs,&outregs);
    printf("The date is %d / %d\n",outregs.h.dh,outregs.h.dl,
          outregs.x.cx);
}

```

#### DOS 功能调用 2AH 取日期

**功能调用:**

2AH 取日期

**DOS版本:**

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

该功能调用返回由 DOS 保持的年、月、日及星期 (DOS1.1 版以上)。

DOS 日期以机器打开电源或重启动时用户输入的值为基准 (或者取时钟 / 日历)，计算机运行期间，即使年、月、日变化，DOS 也保持正确的日期，包括闰年。

**入口参数:**

调用前，需设置：

AH = 2AH 指出功能调用号。

**出口参数:**

返回后，调置为：

CX = 年份 (1980~2099)。

DH = 月份 (1~12)。

DL = 日 (1~31)。

AL = 星期 (0~6, 0 代表星期天, 1 代表星期一等) 用于 DOS1.1 版本以上。

**参见:**

2BH.. 置日期。

2CH.. 取时间。

2DH.. 置时间。

**程序实例:**

以下三个实例使用功能调用 2AH 取当前日期，并在屏幕上显示日期。

#### Assembly Language usage Example: \$ \$

```
; ; GET CURRENT DATE (2AH)
;
code segment public
assume cs:code, ds:code
    org      100h
start: jmp      begin
string1 db      'The date is'
string2 db      2 dup (0), '/'
string3 bd      2 dup (0), '/'
string4 db      4 dup (0), 0dh, 0ah, '$'
month    db      0
day     db      0
```

```

year dw 0
;
; convert number to ascii
; numasc proc      ;number to convert is in ax,
;                   ;ofst to end of string is in bx
    mov si,10      ;use si to divide by 10
next;  xor dx,dx  ;clear dx
    div si      ;divide ax by 10 (in si)
    add dx,'0'    ;convert remainder to ascii number
    dec bx      ;go to next char position
    mov [bx],dl  ;store character in the string
    or ax,ax    ;any more digits to convert?
    jnz next    ;yes, get next digit
    ret        ;no, return
numasc endp  ;end of procedure
;
; main program
;
begin;  mov ax,cs      ;set up ds
        mov ds,ax      ;to same as cs
        mov ah,2ah     ;get date function request
        int 21h       ;call DOS
        mov byte ptr month,dh  ;save month
        mov byte ptr day,al   ;save day
        mov bx,offset string2+2  ;get address of end of string
        xor ax,ax      ;clear ax
        mov al,byte ptr month  ;month
        call numasc    ;convert binary number to ascii
        mov bx,offset string3+2  ;get address of end of string
        xor ax,ax      ;clear ax
        mov al,byte ptr day  ;day
        call numasc    ;convert binary number to ascii
        mov bx,offset string4+4  ;get address of end of string
        mov ax,word ptr year  ;year
        call numasc    ;convert binary number to ascii
        mov dx,offset string1  ;start of complete message
        mov ah,09h     ;display string function request
        int 21h       ;call DOS
        mov ah,00h     ;terminate program funct request

```

```
int 21h      ;call DOS
code ends    ;end of code segment
end start    ;start is the entry point
```

Pascal Usage Example:

```
{ Get Current Date ($ 2a) }
```

```
PROGRAM get_date;
```

```
CONST
```

```
dos_get_date = $2a;
```

```
TYPE
```

```
regpack = Record
```

```
CASE integer OF
```

```
1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
```

```
2 : (al,ah,bl,bh,cl,ch,dl,dh:byte);
```

```
END;
```

```
VAR
```

```
regs: regpack;
```

```
BEGIN
```

```
WITH regs DO
```

```
BEGIN
```

```
ah := dos_get_date;
```

```
msdos(regs);
```

```
writeln('The date is ', dh, '/', dl, '/', cx);
```

```
END;
```

```
END.
```

C Usage Example:

```
/*
 * Get Current Date (0x2a)
 */
```

```
#include <dos.h>
```

```
union REGS inregs, outregs;
```

```

main()
{
    intregs.h.ah = 0x2a;
    intdos(&inregs,&outregs);
    printf("The date is %d / %d \n",outregs.h.dh,outregs.h.dl,
    outregs.x.ex);
}

```

## DOS 功能调用 2BH

### 置日期

**功能调用:**

2BH 置日期

**DOS 版本:**

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

该功能调用类似于功能调用 2AH, 它置 DOS 日期。

功能调用对于使用电池供电的时钟 / 日历工作的程序很有用。程序可从时钟 / 日历取到日期和时间，也可使用该功能调用（及功能调用 2DH）置 DOS 的日期和时间。若 AUTOEXEC.BAT 文件引用该程序，用户无须手工输入日期和时间。

**入口参数:**

调用前，需设置：

AH = 2BH 指出功能调用号。

CX = 要置的年份 (1988-2099)。

DH = 要置的月份 (1-12)。

DL = 要置的日 (1-28, 29, 30, 31, 不同的月份取不同的值)。

**出口参数:**

返回后，设置为：

AL 指明操作状态：

AL = 00H 置入的日期是有效的，并已相应地设置 DOS 的日期。

AL = FFH：置入的日期无效（如1985年2月30日）DOS没置日期。

**参见:**

2AH 取日期

2CH 取时间

2DH 置时间

**程序实例:**

以下三个实例均使用功能调用 2BH 设置日期：1986, 7, 4

**Assembly Language usage Example:**

```

;
; SET DATE (2BH)
;

code segment public
assume cs:code,ds:code
    org      100h
start: jmp      bbegin
msg     db       'Date set to 7 / 4 / 1990',0dh,0ah,'$'
begin: mov ax,cs           ;set up ds
       mov ds,ax           ;to same as cs
       mov ah,2bh          ;set date function request
       mov dh,7             ;month
       mov dl,4             ;day
       mov cx,1990          ;year
       int 21h              ;call DOS
       mov dx,offset msg   ;address of message
       mov ah,09h          ;display string function request
       int 21h              ;call DOS
       mov ah,00h          ;terminate program funct request
       int 21h              ;end of code segment
code ends            ;end of code segment
end start            ;start is the entry point

```

#### Pascal Usage Example:

```
{set Date ($ 2b) }
```

```
PROGRAM set-date;
```

#### CONST

```
dos set date = $ 2b;
```

#### TYPE

```
regpack = RECORD
```

```
    CASE integer OF
        1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
        2 : (al,ah,bl,bh,cl,ch,dl,dh:byte);
    END;
```

#### VAR

```
regs:regpack;
```

```

BEGIN
  WITH regs DO
BEGIN
  ah := dos.set.date;
  dh := 7;
  dl := 7;
  dl := 4;
  cx := 1990;
  msdos(regs);
  writeln('Date set to 7 / 4 / 1990');
END;
END.

```

#### C Usage Example:

```

/*
 * Set Date (0x2b)
 */

#include <dos.h>

union REGS inregs, outregs;

main()
{
    inregs.h.ah = 0x2b;
    inregs.h.dh = 7;
    inregs.h.dl = 4;
    inregs.x.cx = 1990;
    intdos(&inregs, &outregs);
    printf("%7F%7FDDate set to 7 / 4 / 1990\n");
}

```

### DOS 功能调用 2CH 取时间

功能调用:  
2CH 取时间  
DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

该功能调用返回 DOS 保持的时间 (小时、分钟及百分秒)。打开机器或重新启动后，DOS 时间以用户输入的值为基准 (或取过的时钟 / 日历)。

**入口参数:**

调用前，需设置：

AH=2CH 指出功能调用号。

**出口参数:**

返回后，设置为：

CH=小时 (0-23)。

CL=分钟 (0-59)。

DH=秒 (0-59)。

DL=1 / 100 秒 (0-99)。

**参见:**

2AH 取日期。

2BH 置日期。

2DH 置时间。

**程序实例:**

以下三个实例使用功能调用 2CH 取当前时间，并在屏幕上显示时间。

**Assembly Language Usage Example:**

```
; ;GET CURRENT TIME (2CH)
;
code segment public
assume cs:code,ds:code
        org      100h
start:  jmp      begin
string1 db      'The time is '
string2 db      2 dup (0), '/'
string3 db      2 dup (0), '/'
string4 db      2 dup (0), '/'
string5 db      2 dup (0), 0dh,0ah,'$'
hours    db      0
minutes  db      0
seconds  db      0
hundred db      0
;
; convert number to ascii
```

```

; numasc proc      ;number to convert is in ax,
                    ;offset to end of string is in bx
    mov si,10      ;use si to divide by 10
next: xor dx,dx    ;clear dx
    div si        ;divide ax by 10 (in si)
    add dx,'0'    ;convert remainder to ascii number
    dec bx        ;go to next char position
    mov [bx],dl    ;store character in the string
    or ax,ax      ;any more digits to convert?
    jnz next      ;yes, get next digit
    ret           ;no, return e
numasc endp      ;end of procedure

;

; main program
;

begin: mov ax,cs      ;set up ds
    mov ds,ax      ;to same as cs
    mov ah,2ch     ;get time function request
    int 21h        ;call DOS
    mov byte ptr hours,cl  ;save hours
    mov byte ptr minutes,cl ;save minutes
    mov byte ptr seconds,dh  ;save seconds
    mov byte ptr undred,dl   ;save hundredths
    mov bx,offset string2 + 2 ;get address of end of string
    xor ax,ax      ;clear ax
    mov al,byte ptr hours  ;hours
    call numasc    ;convert binary number to ascii
    mov bx,offset string3 + 2 ;get address of end of string
    xor ax,ax      ;clear ax
    mov al,byte ptr minutes ;minutes
    call numasc    ;convert binary number to ascii
    mov bx,offset string4 + 2 ;get address of end of string
    xor ax,ax      ;clear ax
    mov al,byte ptr seconds ;seconds
    call numasc    ;convert binary number to ascii
    mov bx,offset string5 + 2 ;get address of end of string
    xor ax,ax      ;clear ax
    mov al,byte ptr hundred ;hundredths
    call numasc    ;convert binary number to ascii

```

```

        mov dx,offset string1 ;start of complete message
        mov ah,09h  ;display string function request
        int 21h      ;call DOS
        mov a,00h    ;terminate program funct request
        int 21h      ;call DOS
code ends           ;end of code segment
end start          ;start is the entry point

```

**Pascal Usage Example:**

```

{ Get Current Time ($2c) }

PROGRAM get_time;

CONST
  dos_get_time = $2c;
TYPE
  regpack = RECORD
    CASE integer OF
      1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
      2 : (al,ah,b1,bh,c1,ch,d1,dh:byte);
    End;
VAR
  regs: regpack;
BEGIN
  WITH regs DO
  BEGIN
    ah := dos_get_time;
    msdos(regs);
    writeln('The time is ',c,'.',cl,'.',dh,'.',dl);
  END;
END.

```

**C Usage Example:**

```

/*
 * Get Current Time (0x2c)
 */

```

```
#include <dos.h>
```

```

union REGS inregs, outregs;

main()
{
    inregs.h.ah = 0x2c;
    intdos(&inregs.&outregs);
    prints("%7F%7FThe time is %d:%d\n", outregs.h.ch, outregs.h.cl,
          outregs.h.dh, outregs.h.dl);
}

```

## DOS 功能调用 2DH

### 置时间

功能调用:

2DH 置时间

DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明:

该功能调用类似功能调用 2CH, 它置 DOS 时间。

该功能调用对于使用电池供电的时钟 / 日历工作的程序很有用。程序能够从时钟 / 日历中取到日期和时间并能使用该功能调用 (与功能调用 2BH) 相应地置 DOS 日期和时间, 如果 AUTOEXEC.BAT 文件中使用此程序, 用户无须人工输入日期或时间。

程序也可使用功能调用预置时间为 0, 这时可使用 DOS 时钟作为计时器。

入口参数:

调用前, 需设置:

AH=2DH 指出功能调用号。

CH= 小时 (0-23)。

CL= 分钟 (0-59)。

DH= 秒 (0-59)。

DL= 1 / 100 秒 (0-99)。

出口参数:

返回后, 设置寄存器为:

AL 指出操作状态:

AL=00H 置入的时间有效, 并 DOS 相应地置过时间。

AL=FFH 置入的时间中至少有一项无效 (如25小时或者63秒), 功能调用未置时间。

参见:

2AH 取日期。

2BH 置日期,

2CH 置时间。

**程序实例:**

以下三个实例均使用功能调用 2DH 设置当前时间 00: 00: 00: 0.

**Assembly Language Usage Example:**

```
; ; SET TIME ( DH)
;
code segment public
assume cs:code,ds:code
        org      100h
start: jmp      begin
msg     db       'Time set to 00:00:00.00',0dh,0ah,'$'
begin: mov ax,cs    ;set up ds
        mov ds,ax    ;to same as cs
        mov ah,2dh   ;set time function request
        mov ch,0     ;hours
        mov cl,0     ;minutes
        mov dh,0     ;seconds
        mov dl,0     ;hundredths
        int 21h     ;call DOS
        mov dx,offset msg  ;address of message
        mov ah,09h   ;display string function request
        int 21h     ;call DOS
        mov ah,00h   ;terminate program funct request
        int 21h     ;call DOS
codec  ends      ;end of code segment
end start   ;start is the entrypoint
```

**Pascal Usage Example:**

```
{ Set Time ($ 2d) }
PROGRAM set_time;
CONST
  dos_set_time = $ 2d;
TYPE
  regpack = RECORD
  CASE integer OF
```

```

1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
2 : (al,ah,bl,bh,cl,ch,dl,dh:byte);
END;

VAR
  regs : regpack;
BEGIN
  WITH regs DO
    BEGIN
      ah := dos.set.time;
      ch := 0;
      cl := 0;
      cl := 0;
      dh := 0;
      dl := 0;
      msdos(regs);
      writeln('Time set to 00:00:00.00'0);
    END;
  END.

```

### C. Usage Example:

```

/*
 * Set Time (0x2d)
 */
#ifndef include <dos.h>
union REGS inregs, outregs;
main()
{
  inregs.h.ah = 0x2d;
  inregs..ch = 0;
  inregs.h.cl = 0;
  inregs.h.dh = 0;
  inregs.h.dl = 0;
  intdos(&inregs,&outregs);
  printf("Time set to 00:00:00.00\n"0);
}

```

DOS 功能调用 2EH  
设置或关闭校验开关

**功能调用:**

2EH 设置(置1)或关闭(置0)校验开关

**DOS版本:**

1.0, 1.1, 1.2, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

该功能调用检测磁盘在每次写盘时是否做写校验。写校验由检验 CRC 位构成以确保写时不发生奇偶校验错误。一个奇偶校验错误说明至少有一位没有正确传输。

写错误相对地很少见,因此,没必要对要写的所有数据都打开写校验。但可校验要正确写入磁盘的重要信息。写校验时的操作比没有校验时慢一倍。

DOS 功能调用 54H (校验设置) 能决定校验开关的当前状态。

**入口参数:**

调用前,需设置:

AH=2EH 指出功能调用号。

AL 背入其中的数值说明校验开关的设置:

AL=00H 关闭(置0)校验开关, DOS将不校验写操作

=01H 打开(置1)校验开关, DOS将校验所有写操作

**出口参数:**

无。

**参见:**

54H 取校验设置

**程序实例:**

以下三个实例均使用功能调用 2EH 打开(置1)校验开关。

#### Assembly Language Usage Example:

```
; ; SET VERIFY SWITC (2EH)
;
code segment public
assume cs:code,ds,code
org 100h
start: jmp begin
        msg db      'Verify switc is ON.',0dh,0ah,'$'
begin:  mov ax,cs      ;set up ds
        mov ds,ax    ;to same as cs
        mov ah,2h      ;set verify switch funct request
        mov al,01h    ;turn on verify switc
        int 21h      ;call DOS
```

```

        mov dx,offset msg ;address of message
        mov ah,09h      ;display string function request
        int 21          ;call DOS
        mov ah,00h      ;terminate program funct request
        int 21h         ;call DOS
code    ends           ;end of code segment
end start        ;start is the entry point

```

**Pascal Usage Example:**

```

{ Set Verify Switch ($ 2e) }
PROGRAM set_verify;
CONST
  dos set-verify + $ 2e;
  on + 1;
TYPE
  regpack = RECORD
  CASE integer OF
    1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
    2 : (al,ah,bl,bh,cl,ch,dl,dh:byte);
  END;
VAR
  regs:regpack;
BEGIN
  WITH regs DO
  BEGIN
    ah := dos set verify;
    al := on;
    msdos(regs);
    writeln('Verify switch is ON.');
  END;
END.

```

**C Usage Example:**

```

/*
 * Set Verify Switch (0x2e)
 */
#include <dos.h>

```

```

union REGS inregs, outregs;
main()
{
    [inregs.h.ah = 0x2e;
    inregs.h.al = 1;
    intdos(&inregs,&outregs);
    printf("Verify switch is ON.\n");
}

```

## DOS 功能调用 2FH

### 取盘传送地址 (DTA)

#### 功能调用:

2FH 取盘传送地址 (DTA)

#### DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

#### 说明:

该功能调用返回 DOS 当前使用的盘传送地址 (DTA), DTA 是程序存放要写入磁盘的信息存储单元及存放 DOS 从磁盘读出的信息的存储单元。

该功能调用将 DTA 返回至 ES: BX, 因为寄存器 ES 存放地址, 而不是 DS, 所以调用程序不需复位寄存器 DS 以恢复程序的正常操作。

#### 入口参数:

调用前, 需设置:

AH = 2FH 指出功能调用号。

#### 出口参数:

返回后, 设置为:

ES: BX = 盘传送地址。

#### 参见:

1AH 取盘传送地址。

#### 程序实例:

以下三个实例均使用功能调用 2FH 取 DTA, 并在屏幕上显示该地址。

#### Assembly Language Usage Example:

```

;
; GET DISK TRANSFER AREA (2FH)
;
code segment public
assume cs:code,ds:code

```

```

org 100h
start: jmp begin
msg db 'The dTA is'
seg db 4 dup (0),''
ofs db 4 dup (0),0dh,0ah,'$'
;
; convert hex nibble to ascii
;
hexasc proc ;number to convert is in al
    add al,'0' ;turn into ascii code
    cmp al,'9' ;is it a letter?
    jle done ;no, all done
    add al,7 ;yes, add fudge factor
done: ret ;return with answer in al
hexasc endp ;end of procedure
;
; main program
;
begin: mov ax,cs ;set up ds
        mov ds,ax ;to same as cs
        mov ah,2fh ;get DTA function request
        int 21h ;call DOS
        mov cl,4 ;shift count
        mov al,b1 ;offset of DTA, low byte
        shr al,cl ;high nibble of low byte
        call hexasc ;convert hex to ascii
        mov ofs[2],al ;store in offset part of message
        mov al,b1 ;offset of DTA, low byte
        and al,0fh ;low nibble of low byte
        call hexasc ;convert hex to ascii
        mov ofs[3],al ;store in offset part of message
        mov al,bh ;offset of DTA, high byte
        shr al,cl ;high nibble of high byte
        call hexadc ;convert hex to ascii
        mov ofs,al ;store in offset part of message
        mov al,bh ;offset of DTA, high byte
        and al,0fh ;low nibble of high byte
        call hexasc ;convert hex to ascii
        mov ofs[1],al ;store in offset part of message

```

```

    mov ax,es    ;segment of DTA
    shr al,cal   ;high nibble of low byte
    call hexasc  ;convert hex number to ascii
    mov seg[2],al ;store in segment part of message
    mov ax,es    ;segment of DTA
    and al,0fh    ;low nibble of low byte
    call hexasc  ;convert hex to ascii
    mov seg[3],al ;store in segment part of message
    mov ax,es    ;segment of DTA
    mov al,ah    ;high byte
    shr al,cal   ;high nibble of high byte
    call hexasc  ;convert hex to ascii
    mov seg,al   ;store in segment part of message
    mov al,ah    ;high byte
    and al,0fh    ;low nibble of high byte
    call hexasc  ;convert hex to ascii
    mov seg[1],al ;store in segment part of message
    mov dx,offset msg ;start of complete mesage
    mov ah,09h    ;display string function request
    int 21h      ;call DOS
    mov ah,4ch    ;terminate process function request
    int 21h      ;call DOS
code    ends      ;end of code segment
start    ;start is the entry point

```

#### Pascal Usage Example:

```

{ Get Disk Transfer Area ($ 2f) }

PROGRAM get DAT;
CONST
  dos get DAT = $ 2f;
  on = 1;
TYPEe
  regpack = RECORD
    CASE integer OF
      1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
      2 : (al,ah,bl,bh,cl,ch,dl,dh:byte);
    END;
  VAR

```

```

regs : regpack ;
BEGIN
  WITH regs DO
    BEGIN
      ah := dos get DTA ;
      msdos(regs) ;
      writeln('The DTA is ',es,'.',bx) ;
    END ;
  END.

```

#### C Usage Example:

```

/*
 * Get Disk Transfer Area (0x2f)
 */
#include <dos.h>
union REGS inregs, outregs;
struct SREGS segreg;
main()
{
  inregs.h.ah = 0x2f;
  intdos(&inregs,&outregs);
  segread(&segregs);
  printf("The DTA is %X.%X\n",segregs.es,outregs.x.bx);
}

```

DOS 功能调用 30H

取 DOS 版本号

功能调用:

30H 取 DOS 版本号

DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明:

该功能调用返回 DOS 版本号。版本号分为两部分：主版本号和子版本号（如 3.1 版本，则 3 为主版本号，1 为子版本号）。

该功能调用对 DOS2.0 版以上有效，如果在更早的版本下使用它，则主版本号返回值为 0，子版本号将不一致。

此功能调用对于只工作在某种 DOS 版本下的程序是有用的，程序可请求此功能调用

确定版本号，如果版本号错，可给出解释信息并退出。

某些 MS-DOS 版本，功能调用返回的子版本号与在 DOS 下显示的子版本号信息是不一致的。

入口参数：

调用前，需设置：

AH = 30H 指出功能调用号。

出口参数：

返回后，设置为：

AL = 主版本号（如版本号为2.0，则AL=2）；若低于2.0，则AL=0。

AH = 子版本号（如版本号为2.0，则AH=0）

BC与CX 请求功能调用前，要保存这些寄存器中的数值，该功能调用在操作时，要使用它们做完后将它们置为0。

程序实例：

以下三个实例使用功能调用 30H 以显示 DOS 版本号。

#### Assembly Language Usage Example:

```
; ; GET DOS VERSION (30H)
;
code segment public
assume cs:code,ds:code
org 100h

start: jmp begin

string1 db 'DOS Version is'
string2 db 2 dup(0), '/'
string3 db 2 dup(0),0dh,0ah,'$'

major db 0
minor db 0

;
; convert number to ascii
;

numasc proc      ;number to convert is in ax,
                  ;offset to end of string is in bx
    mov si,10      ;use si to divide by 10
next: xor dx,dx    ;clear dx
    div si        ;divide ax by 10 (in si)
    add dx,'0'     ;convert remainder to ascii number
    dec bx        ;go to next char pos in string
```

```

    mov [bx],dl      ;store character in the string
    or ax,ax        ;any more digits to convert?
    jnz next       ;yes, get next digit
    ret            ;no, return
    numasc endp    ;end of procedure
;

; main program
;
begin;   mov ax,cs      ;set up ds
         mov ds,ax      ;to same as cs
         mov ah,30h     ;get DOS version function request
         int 21h        ;call DOS
         mov byte ptr major,al ;save major version number
         mov byte ptr minor,ah ;save minor version number
         mov bx,offset string2 + 2 ;get address of end of string
         xor ax,ax      ;clear ax
         mov al,byte ptr major ;major version number
         call numasc     ;convert binary number to ascii
         mov bx,offset string3 + 2 ;get address of end of string
         xor ax,ax      ;clear ax
         mov al,byte ptr minor ;minor version number
         call numasc     ;convert binary number to ascii
         mov dx,offset string1 ;start of complete message
         mov ah,09h     ;display string function request
         int 21h        ;call DOS
         ov ah,4ch      ;terminate process funct reguest
         int 21h        ;call DOS
code    ends          ;end of code segment
end start        ;start is the entry point

```

#### Pascal Usage Example:

```

{ Get DOS Version ($ 30) }
PROGRAM got_dos_version;
CONST
  dos_get_dos_version = $ 30;
TYPE
  regpack = RECORD
    CASE integer OF

```

```

1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
2 : (al,ah,bl,bh,cl,ch,dl,dh:byte);
END;

VAR
  regs: regpack;
BEGIN
  WITH regs DO
    BEGIN
      a := dos get dos version;
      msdos(regs);
      writeln('DOS Version is ',al,'.',ah);
    END;
  END.

```

#### C Usage Example:

```

/*
 * Get DOS Version (0x30)
 */
#include <dos.h>
union REGS inregs, outregs;
main()
{
  inregs.h.ah = 0x30;
  intdos(&inregs,&outregs);
  printf("DOS Version is %d.%d\n",outregs.h.al,outregs.h.ah);
}

```

### DOS 功能调用 31H

终止进程并保持驻留

功能调用:

31H 终止进程并保持驻留

DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明:

该功能调用结束当前程序，但并不释放内存使 DOS 可分配给其它程序，而是由 DOS 保留内存并允许程序驻留于内存。该功能调用除完成相当于 DOS 中断 27H 的功能外，还使终止程序在终止时向 AL 寄存器传送一个代码。其它程序可使用 WAIT 功能调

用 (4DH) 检查终止码，也可通过批处理文件的 ERRORLEVEL 特性检查该码。如果程序在其终止时需要把信息传给另一个程序，就应使用该功能调用，而不能用 DOS 中断 27H。程序可使用该功能调用建立驻留内存应用程序，如脱机打印程序或堆栈弹出程序。这些程序可以一直保留在内存中并可由中断启动工作。

结束程序时，必须在 DX 寄存器中说明其应驻留在内存的大小及 DOS 可释放供其它程序使用的内存大小。

例如，程序可包括为其建立中断向量的初始化程序（使其可响应中断），此初始化程序只需执行一次，则该程序所占内存程序退出时便可释放。

该功能调用不能释放任何由功能调用 48H（分配内存）或 4AH（更改已分配的内存块）分配的内存。在请求功能调用 49H（释放已分配的内存）之前，不释放额外的内存。尤其可通过重新分配与程序有关的环境，使程序驻留内存部分最小。环境的段地址放在程序 PSP 偏移量 2CH 处的起始字中（见第 20 页 PSP 结构），如要重新分配环境，将此段地址装入 ES 寄存器中，并请求功能调用 49H（释放已分配的内存）。

请求该功能调用后，原来打开的文件仍是打开的。

#### 入口参数：

调用前，需设置：

AH = 31H 指出功能调用号。

AL = 可由程序放置的结束代码，为其它程序指出此程序的结束条件，此处无标准值，各程序必须自己定义该值。

DX = 该寄存器需指出 DOS 为该程序保留的内存段数（十六字节为一段，从 PSP 结尾处开始）。DOS 置内存的初始分配块数为该值，并释放赋予该程序的其余内存空间。

#### 出口参数：

返回后，设置为：

在程序请求该功能调用后，控制不返回至该程序。但下列寄存器的设置不变，同时，其它程序可检索其值。

AL = 终止码，该码由终止程序置入。

#### 参见：

48H 分配内存

49H 释放已分配的内存。

4AH 修改一已分配的内存块 (SETBLOCK)

4BH 装入或执行程序 (EXEC)

4CH 终止进程 (EXIT)

4DH 取子进程的返回码 (WAIT)

DOS 中断 27H 终止但驻留内存

## DOS 功能调用 32H

#### DOS 内部使用：

**功能调用:**

32H DOS 内部使用，程序不能使用该功能调用。

**DOS 功能调用 33H**

**取或设置 Ctrl-Break**

**功能调用:**

33H 取或设置 Ctrl-Break 处理的状态

**DOS 版本:**

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

该功能调用 33H 取或设置 Ctrl-Break 处理的状态，可检查 Ctrl-Break 处理是否有效，也可打开或关闭该处理。

Ctrl-Break 处理打开时，用户可按 Ctrl-Break（或 Ctrl-C）结束不运行或失控了的程序。在 DOS1.X 版中，大多数（不闻全部）功能调用均可自动识别和响应在命令行上输入的 Ctrl-Break 和 Ctrl-C。从 DOS2.0 版开始引入的该功能调用可扩展该特性至所有功能调用或从扩展集中取消该特性。

为检查 Ctrl-Break 处理的状态，请求该功能调用时置 AL 寄存器为 0。返回时，DL 为寄存器状态标志。

若要置 Ctrl-Break 处理状态，需将 AL 置为 1，DL 置成合适的状态。

还有其它两种方法置 Ctrl-Break 处理，即使用 DOSBREAK 命令或使用 CONFIG.SYS 文件。

**入口参数:**

调用前，需设置：

AH=33H 指出功能调用号。

AL 其中的值指明是设置还是检查 Ctrl-Break 处理的状态。

AL=00H 检查 Ctrl-Break 处理的当前状态。这时，功能调用返回的状态放在 DL 中。

AL=01H 设置 Ctrl-Break 处理当前状态，这时，程序中必须在 DL 中设置所需的状态。

DL 如果设置 Ctrl-Break 处理，必须设置此寄存器以指明状态。

DL=00H 关闭 Ctrl-Break 处理

DL=01H 打开 Ctrl-Break 处理

检查 Ctrl-Break 处理状态的程序不应设置此寄存器。

**出口参数:**

返回后，设置为：

DL 如果检查 Ctrl-Break 处理状态，该寄存器数值指明当前状态。

DL=00H Ctrl-Break 是关闭状态。

DL=01H Ctrl-Break 是打开状态。

如果设置 Ctrl-Break 处理状态，则该寄存器内容无意义。

**程序实例：**

以下三个实例均使用 33H 决定 Ctrl-Break 处理是打开的还是关闭的，并在屏幕上显示结果。

**Assembly Language Usage Example:**

```
; CEECK CONTROL-BREAK (33h)
;
code segment public
    ssure cs:code,ds:code
        org 100h
start: jmp begin
msg1    db 'Break is ON',0dh,0ah,'$'
msg2    db 'Break is OFF',0dh,0ah,'$'
begin:  mov ax,cs           ;set up ds
        mov ds,ax           ;to same as cs
        mov ah,33h          ;check ctrl-break function request
        mov al,0             ;check ctrl-break
        int 21h              ;call DOS
        cmp dl,0             ;is break off?
        jnc on               ;jump if break on
        mov dx,offset msg2  ;address of off message
        jmp disp              ;display the message
on:      mov dx,offset msg1  ;address of on message
disp:   mov ah,09h          ;display string function request
        int 21h              ;call DOS
        mov ah,4ch            ;terminate process funct request
        int 21                ;call DOS
code:   ends
        end start            ;start is the entry point
```

**Pascal Usage Example:**

```
{ check Control-Break ($ 33) }
PROGRAM check control break ;
CONST
    dos check control break = $ 33 ;
    check = 0 ;
```

```

TYPE
  regpack = RECORD
    CASE integer OF
      1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
      2 : (al,ah,bl,bh,cl,ch,dl,dh:byte);
    END;
VAR
  regs : regpack ;
BEGIN
  WITH regs DO
  BEGIN
    ah := dos check control break ;
    al := check;
    msdos(regs) ;
    IF dl = 0 THEN
      writeln('Break is OFF')
    ELSE
      writeln('Break is ON');
  END ;
END.

```

### C Usage Example:

```

/*
 * Check Control-Break (0x33)
 */
#include <dos.h>
union REGS inregs, outregs;
main()
{
  inregs.h.ah = 0x33;
  inregs.h.al = 0;
  intdos(&inregs,&outregs);
  if (outregs.h.dl == 0)
    printf("Break is OFF\n");
  else
    printf("Break is ON\n");
}

```

## DOS 功能调用 34H

### DOS 内部使用

功能调用:

34H DOS 内部使用, 程序不应使用该功能调用。

## DOS 功能调用 35H

### 取中断向量地址

功能调用:

35H 取中断向量地址

DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明:

该功能调用返回某一指定中断号的中断向量, 该向量是中断处理程序 (或中断处理程序表) 的地址。

可使用该功能调用在设置中断向量为其它值以前将它保存, 也可使用该功能调用查看缺省向量是否已改变, 或查看中断处理程序 (句柄) 是否确实已分配给中断。在寄存器 ES 和 BX 中返回零值则表明没有中断处理程序 (句柄) 分配给中断。

入口参数:

调用前, 需设置:

AH = 35H 指出功能调用号。

AL = 所要中断向量的16进制的中断等级。

出口参数:

返回后, 设置为:

ES: BX 中断处理程序的地址 (CS: IP值)

参见:

25H 置中断向量

程序实例:

以下三个实例均使用功能调用 35H 检查中断向量表, 并找出严重错误处理程序的入口, 程序还在屏幕上显示严重错误处理程序的地址。

#### Assembly Language Usage Example:

```
; ; GET VECTOR (35H)
; code segment public
assume cs:code,ds:code
```

```

        org 100h
start: jmp begin
msg     db 'Critical error interrupt handler located at '
seg     db 4 dup (0), '/'
ofs     db 4 dup (0),0dh.0ah,'$'
;
; convert hex nibble to ascii
;
hexasc  proc           ;number to convert is in al
        add al,'0'          ;turn into ascii code
        cmp al,'9'          ;is it a letter?
        jle done             ;no, all done
        add al,7             ;yes, add fudge factor
done:   ret              ;return with answer in al
hexasc  endp            ;end of procedure
;
; main program
;
begin: mov ax,cs          ;set up ds
        mov ds,ax          ;to same as cs
        mov al,24h          ;critical error handler interrupt
        int 24h             ;call DOS
        mov cl,4             ;shift count
        mov al,bl            ;offset of DTA, low byte
        shr al,cl            ;high nibble of low byte
        call hexasc          ;convert hex to ascii
        mov ofs[2],al         ;store in offset part of message
        mov al,bl            ;offset of DTA, low byte
        and al,0fh            ;low nibble of low byte
        call hexasc          ;convert hex to ascii
        mov ofs[3],al         ;store in offset part of message
        mov al,bh            ;offset of DTa, high byte
        shr al,cl            ;high nibble of high byte
        call hexasc          ;convert hex to ascii
        mov ofs$,al           ;store in offset part of message
        mov al,bh            ;offset of DTA, high byte
        and al,0fh            ;low nibble of high byte
        call hexasc          ;convert hex to ascii
        mov ofs[1],al         ;store in offset part of message

```

mov ax,cs	;segment of DTA
shr al,cl	;high nibble of low byte
call hexasc	;convert hex number to ascii
mov seg[2],al	;store in segment part of message
mov ax,es	;segment of DTA
and al,0fh	;low nibble of low byte
call hexasc	;convert hex to ascii
mov seg[3],al	;store in segment part of message
mov ax,es	;segment of DTa
mov al,ah	;high byte
shir al,cl	;high nibble of high byte
call hexasc	;convert hex to ascii
mov seg,al	;store in segment part of message
mov al,ah	;high byte
and al,0fh	;low nibble of high byte
call hexasc	;convert hex to ascii
mov seg[1],al	;store in segment part of message
mov dx,offset msg	;start of complete message
mov ah,09h	;display string function request
int 21h	;call DOS
mov ah,4ch	;terminate process funct request
int 21h	;call DOS
Code ends	;end of code segment
end start	;start is the entry point

#### Pascal Usage Example:

```

{ Get Vector ($35) }
PROGRAM get_vector;
CONST
  dos_get_vector = $35;
  critical_error_intr + $24;
TYPE
  regpack = RECORD
    CASE integer OF
      1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
      2 : (al,ah,bl,bh,cl,ch,dl,dh:byte);
    END;
  VAR

```

```

regs: regpack;
BEGIN
  WITH regs DO
    BEGIN
      ah := dos get vector;
      al := critical error intr;
      msdos(regs);
      writeln('Critical error interrupt handler located at',
             es,'.',bx);
    END;
  END.

```

#### C Usage Example:

```

/*
 * Get Vector (0x35)<
 */
#include <dos.h>
union REGS inregs, outregs;
struct SREGS segreg;
main()
{
  inregs.h.ah = 0x35;
  inregs.h.al = 0x24;
  intdos(&inregs, &outregs);
  segread(&segregs);
  printf("Critical error interrupt handler located at %X:%X\n",
         segreg.es, outregs.x.bx);
}

```

#### DOS 功能调用 36H

取磁盘未用空间

功能调用:

36H 取磁盘未用空间

DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明:

该功能调用所返回的是有关取磁盘未用空间数量的信息，它返回磁盘可用分配单元的

数量（它们尚未分配给任何文件）。此外，还返回磁盘簇的总数量、每簇扇区数、每扇区字节数。这些返回的附加信息类似于功能调用 1BH（取当前驱动器 FAT 信息）和 1CH（取任意驱动器 FAT 信息）。

#### 入口参数：

调用前，需设置：

AH = 36H 指出功能调用号。

DL = 指驱动器号，0 表示缺省，1 表示驱动器 A，2 表示驱动器 B 等。

#### 出口参数：

返回后，设置为：

AX 如果未发生错误，其内容为每个分配单元（簇）扇区数，否则为错误代码号（见错误信息部分）。

BX 磁盘可用簇数，这些簇尚未分配给任何文件。

CX 磁盘每个扇区的字节数。

DX 磁盘总簇数。

#### 错误信息：

如果出现错误，有以下参数返回：

AX = FFFFH，表示所选择的驱动器无效。

#### 参见：

1BH.. 取当前驱动器 FAT 信息

1CH.. 取任意驱动器 FAT 信息

#### 程序实例：

下述三个实例均使用 36H 功能调用以获得缺省驱动器的有关信息，从而计算磁盘未用的剩余空间，并将这些信息显示在屏幕上。

#### Assembly Language Usage Example:

```
;  
; GET DISK FREE SPACE (36H)  
;  
code segment public  
assume cs:code,ds:code  
    org 100h  
start: jmp begin  
msg1    db 'Total number of clusters = ','$'  
msg2    db 'Number of available clusters = ','$'  
msg3    db 'Sectors per cluster = ','$'  
msg4    db 'Bytes per sector = ','$'  
string  db 6 dup (?),0dh,0ah,'$'  
total    dw 0
```

```

avail    dw 0
sect    dw 0
secsize dw 0
;
; convert number to ascii
;
numasc  proc      ;number to convert is in ax
        mov cx,6      ;string is 6 bytes long
        mov bx,offset string ;get address of string
blank:  mov byte ptr [bx],'' ;put blanks in this byte
        inc bx      ;increment to next byte
        loop blank   ;do until all 6 bytes are blanked
        mov si,10    ;use si to divide by 10
next:   xor dx,dx    ;clear dx
        div si      ;divide ax by 10 (in si)
        add dx,'0'   ;convert remainder to ascii number
        dec bx      ;go to next char pos in string
        mov [bx],dl   ;store character in the string
        or ax,ax    ;any more digits to convert?
        jnz next    ;yes, getnext digit
        ret       ;no, return
numasc endp  ;end of procedure
;
; main program
;
begin:  mov ax,cs    ;set up ds
        mov ds,ax    ;to same as cs
        mov dl,0h    ;get info for default drive
        mov ah,36h   ;get disk free space funct req
        int 21h     ;call DOS
        mov word ptr sect,ax ;save sectors per cluster
        mov word ptr avail,bx ;save number of free clusters
        mov word ptr total,dx ;save total number of clusters
        mov word ptr secsize,cx ;save sector size in bytes
        mov dx,offset msg1 ;address of total clusters msg
        mov ah,09h    ;display string function request
        int 21h     ;call DOS
        mov ax,word ptr total ;total number of clusters
        call numasc   ;convert binary number to ascii

```

```

        mov dx,offset string ;address of converted number
        mov ah,09h    ;display string function request
        int 21h      ;call DOS
        mov dx,offset msg2 ;address of num of clust message
        mov ah,09h    ;display string function request
        int 21h      ;call DOS
        mov ax,word ptr avail ;number of free clusters
        call numasc   ;convert binary number to ascii
        mov dx,offset string ;address of converted number
        mov ah,09h    ;display string function request
        int 21h      ;call DOS
        mov dx,offset msg3 ;address of num of sect per clause
        mov ah,09h    ;display string function request
        int 21h      ;call DOS
        mov ax,word ptr sect ;sectors per cluster
        call numasc   ;convert binary number to ascii
        mov dx,offset string ;address of converted number
        mov ah,09h    ;display string function request
        int 21h      ;call DOS
        mov dx,offset msg4 ;address of sector size message
        mov ah,09h    ;display string function request
        int 21h      ;call DOS
        mov ax,word ptr secsize ;sector size
        call numasc   ;convert binary number to ascii
        mov dx,offset string ;address of converted number
        mov ah,09h    ;display string function request
        int 21h      ;call DOS
done:  mov ah,4ch    ;terminate process funct request
        int 21h      ;call DOS
code   ends      ;end of code segment
start  end start  ;start is the entry point

```

#### Pascal Usage Example:

```

{ Get Disk Free Space ($ 36) }
PROGRAM get_disk_free_space;
CONST
  dos get disk free space;
  default drive = 0;

```

```

TYPE
  regpack = RECORD
    CASE integer OF
      1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
      2 : (al, ah,bl,bh,cl,ch,dl,dh:byte);
    END ;
VAR
  regs : regpack ;
BEGIN
  WITH regs DO
    BEGIN
      ah := dos_get_disk_free_space ;
      dl := default_drive;
      msdos(regs);
      writeln("Total number of clusters = ',dx);
      writeln("Number of available clusters = ',bx);
      writeln('Sectors per cluster = ',ax);
      writeln('Bytes per sector = ',cx);
    END ;
  END.

```

### C Usage Example:

```

/*
 * Get Disk Fre Space (0x36)
 */
#include <dos.h>
union REGS inregs, outregs;
main()
{
  inregs.h.dl + 0x00;
  inregs.h.ah = 0x36;
  intdos(&inregs,&outregs);
  printf("Total number of clusters = %d\n",outregs.x.dx);
  printf("Number of available clusters = %d\n",outregs.x.bx);
  printf("Sectors per cluster = %d\n",outregs.x.ax);
  printf("Bytes per sector = %d\n",outregs.x.cx);
}

```

## DOS 功能调用 37H

DOS 内部使用

功能调用：

37H DOS 内部使用，程序不能请求此功能调用。

## DOS 功能调用 38H

获取或设置国度信息

功能调用：

38H 取出或设置国度信息

DOS 版本：

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明：

对于 DOS2.X 版本，此功能调用的返回信息为不同国别的日期和时间的格式以及货币符号；对于 DOS3.X 版本，根据给出的国别代码此功能调用既可检索又可改变当前信息集。

对于 DOS3.X 系统，KEYBXX 命令设置当前键盘服务程序才起作用，键盘服务程序截取键入的字符并将键盘扫描码转换为相应 ASCII 字符，通过改变键盘程序可以改变各键的含义。例如，可改变一个外国语言字符集或 Dvorak 键盘布局，当然这要有一个实现这些目标的键盘服务程序可供利用。

每一个键盘服务程序都有一个国别代码和一组相应的国度信息，一旦改变了键盘服务程序，与新键盘服务程序的国别代码相对应的国度信息开始起作用。对于 DOS3.X 版本，此功能调用允许检查当前国度信息以及其它各组国度信息。

缺省键盘服务程序是在 ROM BIOS 里的 USA 英文键盘服务程序，它的国别代码为 0。当利用 KETBXX 命令去改变键盘服务程序时，DOS 就将新的键盘服务程序装在用户可用内存的最低区，同时也改变了 BIOS 中服务于键盘的中断向量，使它指向新的键盘服务程序。一个键盘程序大约占用 RAM1.6K 字节的空间。

一旦改变了键盘服务程序，那么键盘中断就由装入 RAM 的新程序来处理。可以通过按 Ctrl-Alt-F1 键回到驻存在 ROM 里的键盘服务程序，然而这并不改变中断向量中给出的地址。更换后的中断服务程序（装在 RAM 里）仍处理键盘中断，但它用 ROM 查询表代替它自己的查询表，将键盘扫描码转换为 ASCII 码。按 Ctrl-Alt-F2 键使键盘服务程序使用驻留在 RAM 里的查询表。

当获取或设置国度信息时，该信息保存在 32 个字节大小的内存块中，对于 DOS2.X 和 DOS3.X 系统，该内存块的格式是不同的。

DOS2.X 系统的块格式：

偏移量	
00H	日期 / 时间格式 (字)
02H	货币符 (以 00 字节结束的 ASCIIZ 字串)
04H	千位分隔符 (以 00 字节结束的 ASCIIZ 字串)
06H	小数分隔符 (以 00 字节结束的 ASCIIZ 字串)
08H	
01FH	24 个保留字节

DOS.3X 系统的块格式:

偏移量	
00H	日期 / 时间格式 (字)
02H	货币符号 (以 00 字节结束的 4 字节的 ASCIIZ 字串)
07H	千位分隔符 (以 00 字节结束 ASCIIZ 字串)
09H	小数分隔符 (以 00 字节结束 ASCIIZ 字串)
0BH	日期分隔符 (以 00 字节结束的 ASCIIZ 字串)
0DH	时间分隔符 (以 00 字节结束的 ASCIIZ 字串)
0FH	货币符号定位 (字节)
10H	货币小数位 (字节)
11H	时间格式 (24 或 12 小时) (字节)

上述各字段说明如下:

日期和时间代码:

该字表明显示的日期和时间格式，可取值如下：

- |          |         |           |
|----------|---------|-----------|
| 0:U.S.标准 | h: m: s | m / d / y |
| 1:欧洲标准   | h: m: s | d / m / y |
| 2:日本标准   | h: m: s | y: m: d   |

对于 DOS3.X，分隔符（这里用冒号或斜线表示）由日期分隔符和时间分隔符的字段来确定。

**货币符号：**

该字段确定货币符号，（\$为美元符号）对于 DOS2.X 系统，货币符号在第一字节，对于 DOS3.X 系统，货币符号可置四个 ASCII 字符长，最后一个字节为 0。

**千数分隔符：**

该字段确定用于千位分隔的符号（以便从几千的数目字中分隔出几百的数字）在美国，该符号为逗号（,），最后一个字节为 0.

**小数分隔符：**

该字段确定小数点符号，在美国该符号为圆点，其它国家用逗号，最后一个字节为 0.

**日期分隔符：**

该字段确定用来分隔日期各部分的符号，如 10 / 12 / 86 中的斜线（/），最后一个字节为 0.

**时间分隔符：**

该字段确定用来分隔时间单位的符号，如 11:45 中的冒号。

**货币符号定位：**

该字节描述货币符号在何处出现，该字节的 0 位和 1 位控制定位如下：

位

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

0 位	说明
0	货币符在数值前出现如(\$ 5.00)
1	货币符在数值后出现
1 位	
0	货币符和数值间无空格
1	货币符和数值间有一空格

**货币小数位数：**

该字节是一个整数，它表明货币中的小数位数，对美国，该值为 2.

### 时间格式:

该字节表明国别使用 12 还是 24 小时时间格式, 如下:

0: 12 小时时钟

1: 24 小时时钟

### 字盘映象过程地址:

该指针编出字盘映射子程序的远程地址, 该子程序将 256 字符集的高位 128 字符映射为它们相应的数值。DOS 服务无论何时接收一个输入字符 (其 ASCII 码在 80H 到 FFH 之间), 都将 ASCII 码置入 AL 寄存器中, 并调用字盘映射过程, 该过程必须将字符转换成相应字符, 并将新的 ASCII 码存放 AL 中返回, 如果无转换要求, 则 AL 值不变。

### 列表分隔符:

该字段确定表中分隔各项的字符, 在美国, 该符号是逗号 (,), 最后一个字节为 0.

### 入口参数:

调用前, 需设置:

AH = 38H 指出功能调用号。

AL 需置下述数值, 表明要参照的国度信息:

- 在 DOS2.X 版本中, 置为 0.
- 在 DOS3.X 版本中, 取当前别有关信息, 置 00H.
- 在 DOS3.X 版本中, 置当前国别有关信息或取或置任何指定国别的有关信息时, 该寄存器置为所要国度信息对应的国别代码, 码值要小于 255 (FFH)。该代码与用在国际电话中用于标识不同国别的前三位码机同。注意, 设置当前国家信息不同于获取当前国家信息, 需要指定国别代码。
- 在 DOS3.X 版本中, 如果国别代码大于或等于 255 (FFH), 在这种情况下, 该寄存器置为 FFH, BX 置实际的国别代码。

BX 在 DOS3.X 版本中, 如果国别代码大于或等于 255 (FFH), 置该寄存器为实际的国别代码。

DS: DX 如果需要获取国度有关信息, 该寄存器对必须指向 32 个字节的内存块, 功能调用将有关国度信息置于该内存块后返回, 如果是设置国度信息 (仅对 DOS3.X 而言), 置 DX 寄存器为 FFFFH。

### 出口参数:

返回后, 设置为:

DS: DX 如果是获取国度信息, 象前面描述的那样, 指向 32 字节的内存区。

BX 对 DOS3.X 版本而言, 如果是获取国度信息, 该寄存器的内容是国别代码, 如果不知道当前国家的国别代码, 在置 AL 为 0 以后, 进行此功能调用, 检查此寄存器即可。

### 错误信息:

如果出现错误, 进位标志 CF 置位, 且 AX 寄存器内容为标准错误代码。

### 程序实例:

以下三个实例均使用 38H 功能调用, 检索有关国度信息, 并显示当前选定国家的货

币符号。

**Assembly Language Usage Example:**

```
;  
; GET OR SET COUNTRY INFO (38H)  
;  
code segment public  
assume cs:code,ds:code  
org 100h  
start: jmp begin  
msg db 'Currency symbol for this country = ','$'  
info db 32 dup (?)  
begin: mov ax,cs ;set up ds  
       mov ds,ax ;to same as cs  
       mov dx,offset info ;set offset for return data area  
       mov al,0 ;get country information  
       mov ah,38h ;get / set country info funct reg  
       int 21h ;call DOS  
       mov dx,offset msg ;address of currency symbol msg  
       mov ah,09h ;display string function request  
       int 21h ;call DOS  
       mov dl,info(02h) ;currency symbol returned  
       mov ah,02h ;display character funct request  
       int 21h ;call DOS  
done: mov ah,4ch ;terminate process funct request  
      int 21h ;call DOS  
code ends ;end of code segment  
end start ;start is the entry point
```

**Pascal Usage Example:**

```
{ Get or Set Country Info ($ 38) }
```

```
PROGRAM get set country info;
```

```
CONST
```

```
dos get set country-Info = $ 38 ;  
get country info = 0 ;
```

TYPE

```
regpack = RECORD
    CASE integer OF
        1 : (ax,bx,cx,dx,bp,si,di,ds,cs,flags:integer) :
        2 : (al,ah,b1,bh,cl,ch,dl,flh:byte) ;
    END ;
country_info = RECORD
    date_time_format:integer;
    currency_symbol:ARRAY [1..5] OF char;
    thousands_separator:ARRAY [1..2] OF char;
    decimal_separator:ARRAY [1..2] OF char;
    date_separator:ARRAY [1..2] OF char;
    time_separator:ARRAY [1..2] OF char;
    currency_symbol_location:byte;
    currency_decimal_places:byte;
    time_format:byte;
    case_mapping_cell_low:integer;
    case_mapping_cell_high:integer;
    list_separator:ARRAY [1..2] OF char;
    reserved:ARRAY [1..8] OF byte; \
END ;
```

VAR

```
regs: regpack;
info: country_info;
```

BEGIN

WITH regs DO

BEGIN

```
    ah := dos_get_set_country_info;
    al := get_country_info;
    ds := seg(info);
    dx := ofs(info);
    msdos(regs);
    writeln('Currency symbol for this country = '
           info.currency_symbol);
```

END ;

END.

C Usage Example:

```

/*
 * Get or Set Country Info (0x38)
 */

#include <dos.h>

union REGS inregs, outregs;

struct country_info {
    unsigned date_time_fmt;
    char curr_sym[5];
    char thou_sep[2];
    char dec_sep[2];
    char date_sep[2];
    char time_sep[2];
    char curr_sym_loc;
    char curr_dec_places;
    char time_fmt;
    unsigned long case_map_ccll;
    char list_sep[2];
    char reserved[8];
};

struct country_info info;

main()
{
    inregs.x.dx = (int) &info;
    inregs.h.al = 0x00;
    inregs.h.ah = 0x38;
    intdos(&inregs, &outregs);
    printf("Currency symbol for this country = %s\n",
           info.curr_sym);
}

```

## DOS 功能调用 39H MKDIR.. 创建子目录

功能调用:

### 39H MKDIR.. 创建一个子目录

DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明:

该功能调用创建一个新的子目录，正如 MKDIR 命令一样。如果指定的路径名不存在，除非是新的子目录，该功能调用不创建目录。

入口参数:

调用前，需设置寄存器:

AH = 39H 指出功能调用号。

DS: DX 指向含有驱动器号和目录路径名的 ASCIIZ 字串，字串需以 0 字节结尾。

出口参数:

无。

其它要求:

在使用 DOS3.1 以上版本网络环境下调用此功能调用，需具有创建和访问含有子目录之目录的权利。

错误信息:

如果出现错误，进位标志 CF 被置位，且 AX 将含以下之一数值。

03H 指定的路径名不存在。05H 该数值表明几种情况：对路径中的至少一个目录不具有存取权利；根目录已满；子目录名已存在或指定驱动器的路径已存在。

参见:

59H.. 取扩展错误码。

程序实例:

以下三个实例均使用 39H 功能调用，在当前目录中建立一名为 TEMP 的子目录。

#### Assembly Language Usage Example:

```
; ; CREATE SUBDIRECTORY (39H)
;
code segment public
assume cs:code,ds:code
        org      100h
start:  jmp      begin
newdir: db      'temp',0
msg1:   db      'Directory TEMP created in current directory.', '$'
msg2:   db      'Error, directory not created.', '$'
begin:  mov ax,cs           ;set up ds
        mov ds,ax           ;to same as cs
        mov dx,offset newdir ;set offset for path of new dir
```

```

mov ah,39h           ;create subdirectory funct request
int 21h              ;call DOS
jc error             ;jump if error occurred
mov dx,offset msg1  ;address of dir created message
mov a,09h             ;display string function request
int 21h              ;call DOS
jmp done              ;all done
error; mov dx,offset msg2  ;error creating directory message
                      ;display string function request
                      ;call DOS
done: mov ah,4ch        ;terminate process funct request
      int 21h            ;call DOS
code ends             ;end of code segment
end start             ;start is the entry point

```

**Pascal Usage Example:**

{ Create Subdirectory (\$ 39) }

PROGRAM create subdirectory;

CONST

dos create subdirectory + \$ 39 ;  
carry flag = 1 ;

TYPE

regpack + RECORD

CASE integer OF  
1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);  
2 : (al,ah,bl,bh,cl,ch,dl,dh:byte);

END ;

VAR

regs : regpack ;  
newdir : string [50] ;

BEGIN

WITH regs DO

BEGIN

newdir := 'temp' + chr(0) ;  
ah := dos create subdirectory ;  
ds := seg(newdir[1]) ;

```

dx := ofs(newdir[1]);
msdos(regs);
If ((flags AND carry flag) = 1) THEN
    writeln("Error, directory not created.");
ELSE
    writeln("Directory TEMP created in current directory");
END;
END.

```

#### C Usage Example:

```

/*
 * Create Subdirectory (0x39)
 *
#include <dos.h>

union REGS inregs, outregs;
char newdir[] = "temp";

main()
{
    inregs.x.dx = (int) &newdir[0];
    inregs.h.ah = 0x39;
    intdos(&inregs, &outregs);
    if (outregs.x.cflag != 0)
        printf("Error, directory not created.");
    else printf("Directory TEMP created in current directory.");
}

```

#### DOS 功能调用 3AH

#### RMDIR.. 删除一目录

##### 功能调用:

3AH RMDIR.. 删除一个目录

##### DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

##### 说明:

此功能调用删除一个已登记的目录项，正如 RMDIR 俱一样，但不能用此功能调用

去删除当前目录或含有文件的目录。

**入口参数:**

调用前，需设置寄存器：

AH = 3AH 指出功能调用号。

DS: DX 指向含有驱动器号和目录路径名的 ASCII 字串，字串以 0 字节结尾。

**出口参数:**

无。

**其它要求:**

在使用 DOS3.1 以上版本网络环境下，调用此功能调用需具有创建和访问含有子目录之目录的权利。

**错误信息:**

如果出现错误，进位标志 CF 置位，且 AX 含以下数值之一。

03H 指定的路径名不存在。

05H 没有访问权利，意思是指欲删除的目录是根目录或目录中含有文件。

0FH 指定的驱动器非法。

10H 试图删除当前目录。

**参见:**

59H.. 取扩展错误码。

**程序实例:**

以下三个实例均使用 3AH 功能调用，从当前目录中删去名为 TEMP 的子目录。

**Assembly Language Usage Example:**

```
; ; REMOVE DIRECTORY (3AH)
;
code segment public
assume cs:code,ds:code
    org      100h
start:  jmp      begin
olddir   db       'temp',0
msg1     db       'Directory TEMP removed from current directory.',0
        db       '$'
msg2     db       'Error, directory not removed./$'
begin:   mov ax,cs           ;set up ds
        mov ds,ax           ;to same as cs
        mov dx,offset olddir ;set offset for path of old dir
        mov ah,3ah           ;remove directory function request
        int 21h              ;call DOS
```

```

jc error           ;jump if error occurred
mov dx,offset msg1 ;address of dir removed message
mov ah,09h          ;display string function request
int 21h             ;call DOS
jmp done            ;all done
error: mov dx,offset msg2 ;error removing directory message
mov ah,09h          ;display string function request
int 21h             ;call DOS
done: mov ah,4ch    ;terminate process funct request
int 21h             ;call DOS
code ends           ;end of code segment
end start           ;start is the entry point

```

**Pascal Usage Example:**

```

Remove Directory ( $ 3a ) }

PROGRAM remove_directory;
CONST
  dos_remove_directory = $ 3a ;
  carry_flag = 1 ;
TYPE
  regpack = RECORD
    CASE integer OF
      1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer) ;
      2 : (al,ah,bl,bh,cl,ch,dl,dh:byte) ;
    END ;
VAR
  regs : regpack ;
  olddir : string [50] ;
BEGIN
  WITH regs DO
    BEGIN
      olddir := 'temp' + chr(0) ;
      ah := dos_remove_directory ;
      ds := seg(olddir[1]) ;
      dx := ofs(olddir[1]) ;
      msdos(regs) ;
      If ((flags AND carry_flag) = 1) THEN
        writeln('Error, directory not removed.')
      ELSE
    END
  END

```

```
writeln('Directory TEMP removed from current directory'0;
END ;
END.
```

#### C Usage Example:

```
/*
 * Remove Directory (0x3a)
 */
#include <dos.h>

union REGS inregs, outregs;
char olddir[] = "temp";

main()
{
    inregs.x.dx = (int) &olddir[0];
    inregs.h.ah = 0x3a;
    intdos(&inregs,&outregs);
    if (outregs.x.cflag != 0)
        printf("Error, directory not removed.");
    else printf("Directory TEMP removed from current directory.");
}
```

### DOS 功能调用 3BH

#### CHDIR.. 改变当前目录

##### 功能调用:

3BH CHDIR.. 改变当前目录

##### DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

##### 说明:

本功能调用请求改变当前目录。正如 CHDIR 命令一样，如果指定目录路径不存在，则功能调用不改变当前目录。

##### 入口参数:

调用前，需设置寄存器：

AH=3BH 指出功能调用号。

DS: DX 指向含有驱动器号和目录路径名的ASCIIZ字串，表示新的当前目录，字串以 0 字节结尾。驱动器号和路径名总长度不能超过 64 字符，另外，对于 DOS3.1 以上版本，路径名不能包括网络路径。

**出口参数:**

无。

**错误信息:**

如果出现错误，进位标志 CF 置位且 AX 取下列值：

03H 指定的路径名不存在或指定了一个数据文件（不是目录）

**参见:**

59H.. 取扩展错误

**程序实例:**

以下三个实例均使用 3BH 功能调用，改变当前名为 TEMP 的目录。

**Assembly Language Usage Example:**

```
; ; CHANGE DIRECTORY (3BH)
;
code segment public
assume cs:code,ds:code
        org      100h
start:  jmp      begin
newdir  bd       'tmp',0
msg1    db       'Current directory changed to TEMP!,$'
msg2    db       'Error, directory not changed!,$'
begin:  mov ax,cs           ;set up ds
        mov ds,ax ;to same as cs
        mov dx,offset newdir;set offset for path of new dir
        mov ah,3bh ;change directory function request
        int 21h ;call DOS
        jc error jjump if error occurred
        mov dx,offset msg1 ;directory changed message
        mov ah,90h ;display string function request
        int 21h ;call DOS
        jmp done ;all done
error: mov dx,offset msg2 ;error changing directory message
        mov ah,09h ;display string function request
        int 21h ;call DOS
done:  mov ah,4ch ;terminate process funct request
        int 21h ;call DOS
code ends ;end of code segment
        end start ;start is the entry point
```

**Pascal Usage Example:**

```
{ Change Directory $ (3b) }

PROGRAM change_directory;

CONST
  dos_change_directory = $3b;
  carry_flag = 1;

TYPE
  regpack = RECORD
    CASE integer OF
      1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
      2 : (al,ah,bl,bh,cl,ch,dl,dh:byte);
  END;
  VAR
    regs : regpack;
    newdir : string[50];
BEGIN
  WITH regs DO
  BEGIN
    newdir := 'temp' + chr(0);
    ah := dos_change_directory;
    ds := seg(newdir[1]);
    dx := ofs(newdir[1]);
    msdos(regs);
    IF ((flags AND carry_flag) = 1) THEN
      writeln("Error, directory not changed.")
    ELSE
      writeln('Current directory changed to TEMP.');
  END;
END.
```

**C Usage Example:**

```
/*
 * Create Directory (0x3b)
 */
```

```
#include <dos.h>
```

```

union REGS inregs, outregs;
char newdir[] = "temp";

main()
{
    inregs.x.dx = (int) &newdir[0];
    inregs.h.ah = 0x3b;
    intdos(&inregs,&outregs);
    if (outregs.x.cflag != 0)
        printf("Error, directory not changed.");
    else printf("Current directory changed to TEMP.");
}

```

## DOS 功能调用 3CH CREAT.. 创建一文件

功能调用:

3CH 创建一个文件

DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明:

此功能调用创建并打开一个新文件或打开一个已存在的文件长度被截为 0 的旧文件。  
若文件在打开前不存在，则在指定目录创建该文件，并打开文件可做读写访问。

与功能调用 16H (创建一个文件) 相比，该功能调用可在任何子目录下创建文件。

入口参数:

调用前，需设置寄存器:

AH = 3CH 指出功能调用号。

DS: DX 指向含有驱动器号、路径名和文件名的ASCIIZ字串，表明新的文件名，ASCIIZ 字串以 0 字节结尾。

CX 此寄存器的低位字节含新文件的属性如下:

位

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

为了给出文件的属性，需置相应位为 1

位	说明
0	只读文件
1	隐含文件
2	系统文件
3	卷标
4	子目录
5	归档(Archive)
6-7	未用

#### 出口参数:

返回后, 设置寄存器为:

AX           如果未发生错误, 其内容为新文件的文件描述字 (文件句柄), 如果  
          出现了错误, 寄存器内容为错误代码。

#### 其它要求:

在使用 DOS3.1 以上版本的网络环境下调用此功能调用, 需具有创建和存取目录及其  
文件的权利。

#### 错误信息:

如果出现错误, 进位标志 CF 置位且 AX 为以下数值之一:

03H   指定的目录路径名至少一个不存在。

04H   无可用的文件描述字 (文件被打开太多)。

05H   目录已满或指定的文件已经存在并标志为只读以防止被截取。

#### 参见:

43H.. 取或置文件属性 (CHMOD)

59H.. 取扩展错误码

5BH.. 创建一个新文件

#### 程序实例:

以下三个实例均用 3CH 功能调用, 在根目录下创建名为 NEWFILE 的文件。

#### Assembly Language Usage Example:

```
;
;CREATE FILE (3CH)
;
code segment public
assume cs:code,ds:code
org      100h
```

```

start: jmp begin
newfile db '\newfile',0
handle dw ?
msg1 db 'File created in root directory of default drive.'
        db 0dh,0ah,'$'
msg2 db 'File not created',0dh,0ah,'$'
begin: mov ax,cs           ;set up ds
        mov ds,ax          ;to same as cs
        mov dx,offset newfile ;address of path for new file
        mov cx,0            ;file attributes for new file
        mov ah,3ch          ;create file function request
        int 21h             ;call DOS
        jc error            ;check if error
        mov handle,ax       ;save file handle
        mov dx,offset msg1  ;address of created message
        mov ah,09h          ;display string function request
        int 21h             ;call DOS
        mov bx,handle       ;file handle for file to close
        mov ah,3eh          ;close file function request
        int 21h             ;call DOS
        jmp done             ;wrap it up
error:  mov dx,offset msg2  ;address of not created message
        mov ah,09h          ;display string function request
        int 21h             ;call DOS
done:   mov ah,4ch          ;terminate process function request
        int 21h             ;call DOS
code    ends
end start           ;start is the entry point

```

#### **Pascal Usage Example:**

{ Create File (\$3c) }

PROGRAM create\_file;

#### **CONST**

```

dos create file = $3c;
dos close file = $3e;
normal attrib = 0;

```

```

carry flag = 1;
TYPE
  regpack = RECORD
    CASE integer OF
      1 : (ax,bx,cx,dx,bp,si,di,ds,cs,flags:integer);
      2 : (al,ah,bl,bh,cl,ch,dl,dh:byte);
    END;

VAR
  regs : regpack ;
  new_file : string [50];
BEGIN
  WITH regs DO
    BEGIN
      new_file := '\newfile' + chr(0);
      ah := dos create file;
      ds := seg(new_file[1]);
      dx := ofs(new_file[1]);
      cx := normal atrib;
      msdos(regs);
      If ((flags AND carry flag) = 1) THEN
        writeln('File not created.')
      ELSE
        BEGIN
          writeln
            ('file created in root directory of default drive.');
          bx := ax;
          ah := dos close file;
          msdos(rcgs);
        END;
    END;
END;

```

#### C Usage Example:

```

/* *
 * Create File (0x3c)
 */

```

```
#include <dos.h>
```

```

union REGS inregs, outregs;
char newfile[] = "\\\\";

main()
{
    inregs.x.dx = (int) &newfile[0];
    inregs.x.cx = 0;
    inregs.h.ah = 0x3c;
    intdos (&inregs,&outregs);
    if (outregs.x.cflag != 0);
    else {
        printf("File created in root directory of default drive.");
        inregs.x.bx = outregs.x.ax;
        inregs.h.ah = 0x3e;
        intdos(&inregs,&outregs);
    }
}

```

## DOS 功能调用 3DH

打开文件

功能调用:

3DH 打开一个文件

DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明:

此功能调用打开一个已存在的文件以准备进行 I/O 操作，需指定文件路径名和希望具有的存取方式，功能调用返回一个文件描述字（文件句柄），当读或写该文件（3FH 和 40H 功能调用）时，可以使用此文件描述字。

对于 DOS2.X 版本，存取文件的方式包含一程序打算如何使用或访问文件，在功能调用前，通过设置 AL 寄存器指定存取方式。存取方式有三种：读访问（程序打算读但不写该文件）、写访问（程序打算写但不读文件）以及读/写访问。

对于 DOS3.X 版本，增加了两种程序必须规定的存取方式。首先是继承标志（inheritance flag），它表示程序建立的任何子进程是否也能访问该文件，子进程是程序引用的子程序。一旦被引用，这些子进程的运行便独立于调用它们的程序。

对于 DOS3.X 增加的第二种存取方式是共享方式字段。该字段表明仍然正在访问文件的程序如何与其它试图打开文件的程序共享文件。该方式仅在使用 DOSSHARE 命令时有效。

共享方式共有五种：拒绝读／写方式、拒绝写方式、拒绝读方式、无拒绝方式和兼容方式。拒绝读／写方式意味着不与任何其它程序共享文件；拒绝写方式意味着可与其它要读该文件的程序共享文件，但不与其它要写该文件的程序共可与任何程序以任意方式共享文件；兼容方式与无拒绝方式类似，允许网络程序共享相同文件。当文件共享方式被置为兼容方式，其它程序只要也被置于兼容方式就能以任何访问方式打开文件。如果以兼容方式打开一个文件，另一个程序试图以另外的共享方式打开存取方式（读、写、读／写）是否对应其共享方式。

例如：如果以兼容方式打开文件用于写访问，另一个程序试图用拒绝写共享方式打开相同的文件用以读访问，则该程序不能打开此文件，因为它拒绝其它程序对文件的写访问，但你却已可以写访问了。

兼容方式与无拒绝方式的区别在于：如果第一个程序打开一个文件，设置为某种共享方式而不是兼容方式，那么，其它程序如果指定了兼容方式也不能访问该文件（当然直到第一个程序关闭该文件为止），否则将引起中断24H，控制转到严重错误处理程序，该处理程序将指出“驱动器没准备好”的错误，并带有“破坏共享”的扩展错。

通常，一个程序访问文件的能力取决于该文件的属性（是否只读）以及文件是否已打开。如果文件未被打开，则程序可以打开文件用于与文件的只读属性不相冲突的任何访问，如果后继程序的访问方式与文件的只读属性不冲突，或者与第一个程序的共享方式不冲突，并且其共享方式与第一个程序的存取方式不冲突则其可以打开该文件。

下表指明打开一个已打列文件的存取和共享方式的可能性。

第一次打开文件的存取和共享方式	只读访问	只写访问	读／写访问
兼容共享方式	后继打开： 任何访问 共享方式： 兼容 拒绝写 无拒绝	后继打开： 任何访问 共享方式： 兼容 拒绝读 无拒绝	后继打开： 任何访问 共享方式： 兼容 无拒绝
无拒绝共享方式	后继打开： 任何访问 共享方式： 拒绝写 无拒绝	后继打开： 任何访问 共享方式： 拒绝读 无拒绝	后继打开： 任何访问 共享方式： 无拒绝
拒绝读共享方式	后继打开： 只写访问 共享方式： 拒绝写 无拒绝	后继打开： 只写访问 共享方式： 拒绝读 无拒绝	后继打开： 只写访问 共享方式： 无拒绝
拒绝写共享方式	后继打开： 只读访问 共享方式： 拒绝写 无拒绝	后继打开： 只读访问 共享方式： 拒绝读 无拒绝	后继打开： 只读访问 共享方式： 无拒绝
拒绝读写共享方式	无其它存取方式	无其它存取方式	无其它存取方式

如果打开的文件在网络盘上，当设置共享方式为兼容、拒绝读/写和拒绝写方式时，DOS 仅完成本地缓冲操作。若指定其它的共享方式，则不做本地缓冲操作。

当打开一个文件，功能调用设置读/写指针指向文件的第一个字节，并设置记录尺寸为一个字节时，可以通过功能调用 42H 改变读/写指针的位置。

请求功能调用时，不要指定文件（如隐含、系统等）的属性，请求功能调用可以打开任何正式或隐含文件，但是不能打开任何以冒号结尾文件的文件（如 COM1:）。

入口参数：

调用前，需设置寄存器：

AH = 3DH 指出功能调用号。

AL = 存取方式代码，该字节的各位说明如下：

位

7	6	5	4	3	2	1	0
i	s	s	s	r	a	a	a

对于 DOS3.X 系统，所有数位都有其各自的含义。对 DOS2.X 系统，仅提供 0-2 位 (aaa 位)。在 DOS2.X 系统下运行的程序应设置其它位为 0。

方式位说明如下：

i 继承位。如果置为 0，由该程序创建的任何子进程也可以这里说明的同样方式存取文件；如果置为 1，子进程不能自动存取该文件，但子进程能打开文件本身。

此位仅用于 DOS3.X 系统。

sss 共享方式，这些位表明程序如何与试图同时打开该文件的其它程序共享文件。置各位以下二进制数值：

000 兼容方式

001 拒绝读/写方式，不与试图读/写该文件的程序共享文件

010 拒绝写方式，不与试图写该文件的程序共享文件

011 拒绝读方式，不与试图读该文件的程序共享文件

100 无拒绝方式，与任何其它程序共享文件这些位仅用于 DOS3.X 系统。

r 此位保留且置为 0

aaa 存取码，这些位表明程序如何使用文件。DOS2.X 和 DOS3.X 程序将这些位位置以下二进制数值：

000 只对读访问打开文件

001 只对写访问打开文件

010 对读、写访问打开文件

DS：DX 通过指向含有驱动器、目录路径和文件名的 ASCIIZ 字串来表明文件、字串以 0 字节结尾。

出口参数：

返回后，设置寄存器为：

AX 除非出现错误，否则其内容为被打开的文件描述字（文件句柄）。

**错误信息：**

如果出现错误，进位标志 CF 置位且 AX 有以下数值之一。

02H DOS 没有找到指定的文件。

03H 指定的路径名非法，可能的原因之一是指定的子目录不存在。

04H 无可用的文件描述字，也就是说已打开的文件太多。

05H 拒绝访问文件，可能因为已打开的文件存取方式与所需存取文件的方式相矛盾。如果指定的文件名是一卷标或目录也返回此值。

0CH 指定了非法的访问代码。

**参见：**

3EH.. 关闭文件描述字（文件句柄）

3FH.. 读文件或设备

40H.. 写入文件或设备

42H.. 移动读 / 写指针 (LSEEK)

43H.. 取出或设置文件属性

57H.. 取出或设置文件的日期和时间

59H.. 取扩展错误代码

**程序实例：**

以下三个实例均应用功能调用 3DH，打开名为 TESTFILE.ASC 的文件，并关闭文件。

**Assembly Language Usage Example:**

```
; ; OPEN FILE (3DH)
;
code segment public
assume cs:code,ds:code
    org      100h
start:  jmp      begin
file     db       'testfile.asc',0
handle   dw       ?
msa1    db       'TESTFILE.ASC opened.',0dh,0ah,'$'
msg2    db       'TESTFILE.ASC not opened.',0dh,0ah,'$'
begin:  mov ax,cs           ;set up ds
        mov ds,ax           ;to same as cs
        mov dx,offset file  ;address of file to open
        mov al,0              ;access mode
        mov ah,3dh            ;open file function request
```

```

int 21h           ;call DOS c
jc error          ;check if error
mov handle,ax     ;save file handle
mov dx,offset msg1 ;address of opened message
mov a,09h          ;display string function request
int 21             ;call DOS
mov bx,handle      ;file handle for file to close
mov ah,3eh          ;close file function request
int 21h            ;call DOS
error: mov dx,offset msg2 ;address of not opened message
        mov a,09h          ;display string function request
        int 21             ;call DOS
        int 21h            ;call DOS
code ends          ;end of code segment
end start          ;start is te entry point

```

**Pascal Usage Example:**

```
{ Open File ( $ 3d ) }
```

```
PROGRAM open_file;
```

```
CONST
```

```
dos_open_file = $ 3d ;
dos_close_file = $ 3e ;
read_access = 0 ;
carry_flag = 1 ;
```

```
TYPE
```

```
regpack = RECORD
  CASE integer OF
    1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer) ;
    2 : (al,a,bl,bh,cl,ch,dl,dh:byte);
  END ;
```

```
VAR
```

```
regs: regpack ;
file_to_open: string [50] ;
```

```
BEGIN
```

```
WITH regs DO
```

```
BEGIN
```

```
  file_to_open := 'testfile.asc' + chr(0) ;
```

```

        ah := dos__open__file ;
        ds := seg(file__to__open[1]) ;
        dx := ofs(file__to+__open[1]) ;
        al := read__access ;
        msdos(regs) ;
        IF ((flags AND carry_flag) = 1) THEN
            writeln('TESTFILE.ASC not opened.')
        ELSE
            BEGIN
                writeln('TESTFILE.ASC opened.') ;
                bx := dx;
                ah := dos__close__file ;
                sdos(regs); \
            END ;
        END ;
    END.

```

### C Usage Example:

```

/*
 * Open file (0x3d)
 */
#include <dos.h>

union REGS inregs,outregs;
char filename] = %7F%7Ftestfile.asc",

main()
{
    inregs.x.dx = (int) &filename[0];
    inregs.h.al = 0;
    inregs.h.ah = 0x3d;
    intdos(&inregs,&outregs);
    if (outregs.x.cflag != 0)
        printf("TESTFILE.ASC not opened.");
    else {
        printf("TESTFILE.ASC opened.");
        inregs.x.bx = outregs.x.ax;
    }
}

```

```
inregs.ah = 0x3e;
intdos(&inregs.&outregs);
}
}
```

## DOS 功能调用 3EH

关闭一文件描述字 (文件句柄)

**功能调用:**

3EH 关闭文件描述字 (文件句柄)

**DOS 版本:**

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

此功能调用用于关闭以 3CH 或 3DH 功能调用打开的文件。结束文件使用时，可应用此功能调用；还可用于修改文件的目录和清除所有与该文件有关的内部缓冲区，清除的缓冲区包括内存中写文件的局部缓冲区。

**入口参数:**

调用前，需设置寄存器：

AH = 3EH 指出功能调用号。

BX = 通过创建一个文件 (3CH) 或打开一个文件 (3DH) 的调用而返回的文件描述字。

**出口参数:**

无。

**错误信息:**

如果出现错误，进位标志 CF 置位且 AX 为以下数值：

06H 提供的文件描述字不合法。

**参见:**

3CH.. 创建一文件 (CREAT)

3DH.. 打开一文件

3FH.. 读文件和设备

40H.. 写入文件和设备

59H.. 取扩展错误代码

**程序实例:**

以下三个程序实例均使用 3EH 功能调用关闭一个文件，首先打开，然后关闭。

**Assembly Language Usage Example:**

```
; ; CLOSE FILE (3EH)
;
```

```

cods segment public
assume cs:code,ds:code
    org      100h
start: jmp      begin
file   db       'testfile.asc',0
handle  dw      ?
msg1   db       'TESTFILE.ASC opened.',0dh,0ah,'$'
msg2   db       'TESTFILE.ASC not opened.',0dh,0ah,'$'
msg3   db       'TESTFILE.ASC closed.',0dh,0ah,'$'
msg4   db       'TESTFILE.ASC not closed.',0dh,0ah,'$'
begin: mov ax,cs           ;set up ds
        mov ds,ax           ;to same as cs
        mov dx,offset file  ;address of file to open
        mov al,0 'access ,pdet
        mov a,3dh            ;open file function request
        int 21h              ;call DOS
        jc operr             ;check if error
        mov handle,ax         ;save file handle
        mov dx,offset msg1   ;address of opened message
        mov a,09h              ;display string function request
        int 21h              ;call DOS
        mov bx,handle          ;file handle for file to close
        mov ah,3ch              ;close file function request
        int 21h              ;call DOS
        jc clerr             ;check if error
        mov dx,offset msg3   ;address of closed message
        mov ah,09h              ;display string function request
        int 21h              ;call DOS
        jmp done              ;wrap it up
clerr: mov dx,offset msg4   ;address of not closed message
        mov ah,09h              ;display string function request
        int 21h              ;call DOS
        jmp done              ;wrap it up
operr: mov dx,offset msg2   ;address of not opened message
        mov a,09h              ;display string function request
        int 21h              ;call DOS
done:  mov ah,4ch            ;terminate process funct request
        int 21                ;call DOS
code ends           ;end of code segment

```

end start ;  
;start is the entrypoint

**Pascal Usage Example:**

{ Close File (\$3e) }

PROGRAM close\_file;

CONST

dos\_open\_file = \$3d;  
dos\_close\_file = \$3e;  
read\_access = 0;  
carry\_flag = 1;

TYPE

regpac = RECORD  
CASE integer OF  
1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);  
2 : (al,ah,bl,bh,cl,ch,dl,dh:byte);  
END;

VAR

regs : regpack;  
file\_to\_test : string[50];

BEGIN

WITH regs DO

BEGIN

file\_to\_test := 'testfile.asc' + chr(0);  
ah := dos\_open\_file;  
ds := seg(file\_to\_test[1]);  
dx := ofs(file\_to\_test[1]);  
al := read\_access;  
msdos(regs);  
IF ((flags AND carry\_flag) = 1) THEN  
writeln('TESTFILE.ASC not opened.')

ELSE

BEGIN

writeln('TESTFILE.ASC opened.');//  
bx := ax;  
ah := dos\_close\_file;  
msdos(regs);

```

        IF ((flags AND carry_flag) = 1) THEN
            writeln('TESTFILE.ASC not closed.')
        ELSE
            writeln('TESTFILE.ASC CLOSED.');
        END;
    END;
END.

```

### C Usage Example:

```

/*
 * Close File (0x3e)
 */

#include <dos.h>

union REGS inregs, outregs;
char filename[] = "testfile.asc";

main()
{
    inregs.x.dx = (int) &filename[0];
    inregs.h.al = 0;
    inregs.h.ah = 0x3d;
    intdos(&inregs,&outregs);
    if (outregs.x.cflag != 0)
        printf("TESTFILE.ASC not opened.");
    else {
        printf("TESTFILE.ASC opened.\n");
        inregs.x.bx = outregs.x.ax;
        inregs.h.ah = 0x3e;
        intdos(&inregs,&outregs);
        if (outregs.x.cflag != 0)
            printf("TESTFILE.ASC not closed.\n");
        else printf("TESTFILE.ASC closed.\n");
    }
}

```

### DOS 功能调用 3FH

## 读文件或设备

功能调用：

3FH 读文件或设备

DOS 版本：

1.0, 1.1, 2.0, 2.1, 3.0; 3.1, 3.2, 3.3

说明：

此功能调用对文件或设备进行读操作。该文件或设备已由创建一个文件 (3CH) 或打开一个文件 (3DH) 功能调用打开。该功能调用代替 DOS1.X 版本的功能调用：14H (读下一顺序文件记录)、21H (读出一随机记录) 及 27H (读出多个随机记录)。

信息的读取是从读 / 写指针的当前位置开始的。(该指针通过功能调用 42H 设置) 读取的字节数由设置 CX 寄存器而定。

此功能调用并不保证读取的所有字节都是所需要的。例如，如果从键盘读取，则功能调用返回一个文本一行中字节数的最大值 (直到按回车键)。

不象早期 DOS1.X 版本的功能调用从文件读取信息 (如 14H)。此功能调用不能自动将信息传送到 DTA，而是通过设置 DS: DX 地址来指定接收信息的内存区。

如果消用此功能调用从标准输入设备读取信息，可以重定向输入。

入口参数：

调用前，需设置寄存器：

AH = 3FH 指出功能调用号。

BX 其内容应是创建一个文件 (3CH) 或打开一个文件 (3DH) 之功能调用返回的文件描述字。文件必须为读操作打开。CX 规定所要读的字节数。

DS: DX 指向功能调用中存放读取信息的内存块。

出口参数：

返回后，设置寄存器为：

AX 除非出现错误，否则其内容为实际读出的字节数；如果为 0，说明程序是从文件尾读取的。

DS: DX 指向内存块，该块存放从文件或设备读取的信息。

其它要求：

在调用前，需应用创建一个文件 (3CH) 或打开一个文件 (3DH) 功能调用打开该文件。用 3DH 打开文件时，需请求读或读 / 写访问该文件。

在用 DOS3.1 以上版本的网络环境下调用此功能，必须具有对目录及其文件进行读访问的权利。

错误信息：

如果出现错误，进位标志 CF 被置位且 AX 为下列数值之一：

05H 拒绝访问文件，可能的原因是没有为读访问打开的文件。

06H 提供的文件描述字 (文件句柄) 非法。

参见：

3DH.. 打开一个文件

- 3EH.. 关闭文件描述字 (文件句柄)
- 40H.. 写入文件或设备
- 42H.. 移动读 / 写指针 (LSEEK)
- 43H.. 取或置文件属性 (CHMOD)
- 57H.. 取或置文件的日期和时间
- 59H.. 取扩展错误代码

**程序实例：**

以下三个程序实例均使用 3FH 功能调用，读取名为 TESTFILE.ASC 的文件且在屏幕上显示文件内容。

**Assembly Lanugage Usage Example;**

```

;
; READ FORY OR DEVICE (3FH)
;

code segment public
assume cs:code,ds:code
org 100h

start: jmp begin

file    db 'testfile.asc',0
handle   dw ?
buffer  db 513 dup (?)
msg1    db 'Unable to open TESTFILE.ASC.',0dh,0ah,'$'
msg2    db 'Unable to read TESTFILE.ASC.',0dh,0ah,'$'

begin: mov ax,cs           ;set up ds
       mov ds,ax          ;to same as cs
       mov dx,offset file ;address of file to open
       mov al,0            ;access mode
       mov ah,3dh          ;open file function request
       int 21h             ;call DOS
       jc operr            ;check if error
       mov handle,ax        ;save file handle

read:  mov bx,handle        ;file handle for file to read
       mov dx,offset buffer ;address of read buffer
       mov cx,412            ;number of bytes to read
       mov ah,3fh             ;read file or device funct request
       int 21h               ;call DOS
       jc readerr            ;check if error
       cmp ax,0              ;reached end of file yet?
       je close              ;close file and wrap it up

```

```

        mov bx,ax          ;number of bytes read
        mov buffer[bx],'$' ;end of string
        mov dx,offset buffer ;address of buffer
        int 21h             ;call dOS
        jmp read            ;read next buffer full
oper:  mov dx,offset msg1 ;address of open error message
        mov ah,09h           ;display string function request
        int 21h             ;call DOS
        jmp done            ;wrap it up
readerr: mov dx,offset msg2 ;address of read error message
        mov ah,09h           ;display string function request
        int 21h             ;call DOS
close:  mov bx,handle      ;file handle for file to close
        mov ah,3eh            ;close file function request
        int 21h             ;call DOS
done:   mov ah,4ch            ;terminate process function request
        int 21h             ;call DOS
code:   ends                ;end of code segment
        end start           ;start is the entry point

```

### Pascal Usage Example:

```

{ Read From File or Device ($3F) }

PROGRAM read_file;

CONST
  dos_open_file = $3d;
  dos_close_file = $3e;
  dos_read_file = $3f;
  read_access = 0;
  carry_flag = 1;

TYPE
  regpac = RECORD
    CASE integer OF
      1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags,integer),
      2 : (ai,ah,bl,bh,cl,ch,di,dh,by,integer);
    END;
  VAR
    regs : regpack;
    filename : string [50];

```

```

handle : integer ;
buffer : ARRAY [1..512] OF char ;
i : integer ;
BEGIN
  WITH regs DO
    BEGIN
      filename := 'testfile.asc' + ch(0) ;
      ah := dos_open_file ;
      ds := seg(filename[1]) ;
      dx := ofs(filename[1]) ;
      al := read_access ;
      msdos(regs) ;
      IF ((flags AND carry_flag) = 1) THEN
        writeln('Unable to open TESTFILE.ASC.')
      else
        BEGIN
          handle := ax ;
          WHILE ax <= 3 DO
            BEGIN
              bx := handle ;
              ds := seg(buffer) ;
              dx := ofs(buffer) ;
              cx := $12 ;
              ah := dos_read_file ;
              msdos(regs) ;
              IF ((flags AND carry_flag) = 1) THEN
                BEGIN
                  writeln('Unable to read TESTFILE.ASC.') ;
                  ax := 0 ;
                END
              ELSE IF ax <= 0 THEN
                FOR i := 1 TO ax DO
                  write(buffer[i]) ;
            END ;
            bx := handle ;
            ah := dos_close_file ;
            msdos(regs) ;
        END ,
      END ;

```

END.

**C Usage Example:**

```
/*
 * Read From File or Device (0x3f)
 */
#include<dos.h>
union REGS inregs, outregs;
char filename[] = "testfile.asc";
main()
{
    char buf[512];
    inregs.x.dx = (int)&filename[0];
    inregs.h.al = 0;
    inregs.a.ah = 0x3d;
    intdos(&inregs,&outregs);
    if (outregs.x.cflag != 0)
        printf("Unable to open TESTFILE.ASC.");
    else {
        inregs.x.bx = outregs.x.ax;
        inregs.x.dx = (int) &buf[0];
        inregs.x.cx = 512;
        inregs.a.ah = 0x3f;
        for (;;) {
            intdos(&inregs,&outregs);
            if (outregs.x.cflag != 0) {
                printf("Unable to read TESTFILE.ASC");
                break;
            }
            if (outregs.x.ax == 0) break;
            else {
                buf[outregs.x.ax] = 0;
                printf("%s",buf);
            }
            inregs.h.ax = 0x3e;
            intdos(&inregs,&outregs);
        }
    }
}
```

## DOS 功能调用

### 40H 写入文件或设备

功能调用:

40H 写入文件或设备

DOS 版本:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明:

此功能调用对文件或设备进行写操作。该文件或设备需已用打开一文件 (3DH) 或创建一文件 (3CH) 的功能调用为写操作打开。该功能调用代替 DOS1.X 版本的功能调用: 15H (写一顺序文件记录)、22H (写单个随机记录) 和 28H (写多个随机记录)。

信息的写入从读写指针的当前位置开始 (指针由功能调用 42H 设置), 通过设置 CX 寄存器指定写入的字节数。

如果指定写入的字节数为 0, 则功能调用在读 / 写指针的当前位置截止文件。在做此操作前, 可以通过 42H 功能调用 (移动文件读 / 写指针) 移动读 / 写指针到正确位置。

不像早期 DOS1.X 版本写入文件功能调用 (如 15H), 此功能调用不设 DTA 识别信息的起点, 而是设置 DS: DX 指向存放写入信息的内存区。

如果用此功能调用写到标准输出设备可以重定向输出。

入口参数:

调用前, 需设置寄存器:

AH = 40H 指出功能调用号。

BX 由 3CH (创建一个文件) 或 3DH (打开一个文件) 功能调用返回的文件描述字 (文件句柄), 文件必须为写操作打开。

CX 规定写字节数, 若置为 0, 则在读 / 写指针的当前位置截止文件。

DS: DX 指向存放写入信息的内存块。

出口参数:

返回后, 设置寄存器为:

AX 除非出现错误, 否则表示实际写入的字节数。

其它要求:

调用前, 需通过创建一个文件 (3CH) 或打开一个文件 (3DH) 的功能调用打开该文件, 当用 3DH 打开文件时, 必须要求写或读 / 写操作。

在用 DOS3.X 以上版本的网络环境下使用此功能调用时, 必须具有对目录用其文件进行写访问的权利。

错误信息:

如果实际写入的字节数 (返回 AX 寄存器中的数值) 不等于设置的字节数 (CX 寄存器所设置的数值), 将会出现错误, 此功能调用不能返回另外信息来表明此错误, 所以程序要自行检查这种情况。

如果出现错误, 进位标志 CF 置位且 AX 为以下数值之一:

05H 拒绝文件存取，可能因为没有为写操作打开文件。

06H 提供的文件描述字（文件句柄）不合法。

参见：

- 3DH.. 打开一个文件
- 3EH.. 关闭文件描述字（文件句柄）
- 3FH.. 读取文件或设备
- 42H.. 移动读／写指针 (LSEEK)
- 43H.. 取或置文件属性 (CHMOD)
- 57H.. 取或置文件日期和时间
- 59H.. 取扩展错误

程序实例：

以下三个实例均使用 40H 功能调用写入一个文件，程序将名为 TESTFILE.IN 文件内容拷贝到名为 TESTFILE.OUT 文件中。

#### Assembly Language usage Example;

```
; ; WRITE TO FILE OR DEVICE 94OH)
;
code segment public
assume CS:code,ds:code
    org 100h
start: jmp begin
infile db 'testfile.in',0
outfile db 'testfile.out',0
inhdl dw ?
outhdl dw ?
buffer db 512 dup (?)
msg    db 'TESTFILE.IN not found.',0dh,0ah,'$'
begin: mov ax,cs           ;set up ds
       mov ds,ax          ;to same as cs
       mov dx,offset infile ;address of input file
       mov al,0             ;access mode
       mov ah,3dh            ;open file function required
       int 21h              ;call DOS
       jc operr             ;check if error
       mov inhdl,ax          ;save file handle
       mov dx,offset outfile ;address of output file
       mov cx,0              ;file attributes
```

	mov ah,3ch	;create file function request
	int 21h	;call DOS
	mov outhdl,ax	;save file handle
	mov bx,inhdl	;file handle for file to read
	mov dx,offset buffer	;address of read buffer
	mov cx,512	;number of bytes to read
	mov ah,3fh	;read file or device funct request
	int 21h	;call DOS
	cmp ax,0	;reached end of file yet?
	je close	;close file and wrap it up
	mov ex,ax	;number of bytes read and to write
	mov bx,outhdl	;file handle for output file
	mov dx,offset buffer	;address of buffer
	mov ah,40h	;write file to device funct req
	int 21h	;call DOS
	jmp read	;read next buffer full
open:	mov dx,offset msg	;address of open error message
	mov ah,09h	;display string function request
	int 21h	;call DOS
	jmp done	;wrap it up
close:	mov bx,inhdl	;file handle for inp file to close
	mov ah,3eh	;close file function request
	int 21h	;call DOS
	mov bx,outhdl	;file handle for outp file to close
	mov ah,3eh	;close file function request
	int 21h	;call DOS
done:	mov ah,4ch	;terminate process funct request
	int 21h	;call DOS
endcode	end	;end of code segment
	start	;start is the entry point

#### Pascal Usage Example:

{ Write To File or Device (\$ 40) }

PROGRAM write\_file;

CONST

```

dos__open__file = $3d;
dos__create__file = $3c;
dos__close__file = $3e;

```

```

dos_read_file = $3F;
dos_write_file = $40;
read_access = 0;
carry_flag = 1;

TYPE
  regpack = RECORD
    CASE integer OF
      1 : (ax,bx,cx,dx,dp,si,di,ds,es,flags:integer);
      2 : (ai,ah,bl,dl,ch,cl,di,dh:byte);
    END;
  VAR
    regs : regpack;
    infile : string [50];
    outfile : string [50];
    inhandle : integer;
    outhandle : integer;
    buffer : ARRAY [1..512] OF char;
  BEGIN
    WITH regs DO
      BEGIN
        infile := 'testfile.in' + chr(0);
        ah := dos_open_file;
        ds := seg(infile[1]);
        dx := ofs(infile[1]);
        al := read_access;
        msdos(regs);
        IF ((flags AND carry_flag) = 1) THEN
          writeln('TESTFILE.in not found.')
        ELSE
          BEGIN
            inhandle := ax;
            outfile := 'testfile.out' + chr(0);
            ah := dos_create_file;
            ds := seg(outfile[1]);
            dx := ofs(outfile[1]);
            cx := normal_attrib;
            msdos(regs);
            outhandle := ax;
            WHILE ax < > 0 DO

```

```

BEGIN
    bx := inhandle;
    ds := seg(buffer);
    dx := ofs(buffer);
    cx := 512;
    ah := dos_read_file;
    msdos(regs);
    IF ax < > 0 THEN
        BEGIN
            cx := ax;
            bx := outhandle;
            ds := scg(buffer);
            dx := ofs(buffer);
            ah := dos(buffer);
            msdos(regs);
        END ;
    END ;
    bx := inhandle;
    ah := dos_close_file;
    msdos(regs);
    bx := outhandle;
    ah := dos close file;
    msdos(regs);
END ;
END ;
END.

```

### C Usage Example:

```

/*
 * Write To File of Device (0x40)
 */
#includec <dos.h>
union REGS inregs, outregs;
char infile[] = "testfile.in";
char outfile[] = "testfile.out";
main()
{
    int inhandle;

```

```

int outhandle;
char buf[512];
inregs.x.dx = (int) &infile[0];
inregs.h.al = 0;
inregs.h.ah = 0x3d;
intdos(&inregs,&outregs);
if (outregs.x.cflag != 0)
    ptitle("TESTFILE.IN not found\n");
else {
    inhandle = outregs.x.ax;
    inregs.x.dx = (int) &outfile[0];
    inregs.x.cx = 0;
    inregs.h.ah = 0x3c;
    intdos(&inregs,&outregs);
    outhandle = outregs.c.ax;
    for (;;) {
        inregs.x.bx = inhandle;
        inregs.x.ds = (int) &buf[0];
        inregs.x.cx = 512;
        inregs.a.ah = 0x3f;
        intdos(&inregs,&outregs);
        if (outregs.x.ax == 0) break;
        else {
            inregs.x.cx = outregs.x.ax;
            inregs.x.bx = outhandle;
            inregs.x.dx = (int) &buf[0];
            inregs.h.ah = 0x40;
            intdos(&inregs,&outregs);
        }
    }
    inregs.x.bx = inhandle;
    inregs.h.ah = 0x3e;
    intdos(&inregs,&outregs);
    inregs.x.bx = outhandle;
    intdos(&inregs,&outregs);
}
}

```

## DOS 功能调用 41H

## **UNLINK.. 删除一文件**

**功能调用:**

41H UNLINK.. 删除一文件

**DOS版本:**

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

从指定的目录里删除一个文件(实际上它消除文件在目录中的登录)。

用此功能调用不能删除只读文件，要删除只读文件必须首先用取或设置文件的属性的功能调用43H，改变文件的属性为非只读属性。

**入口参数:**

调用前，需设置寄存器：

AH=41H 指出功能调用号。

DS: DX 指向含有被删除文件的驱动器和目录路径名的 ASCIIZ 字串，字串以 0 结尾。

文件通过配符（\* 和?）不允许在字串的任何位置出现。

**出口参数:**

无。

**其它要求:**

删除只读文件前，需通过取出或设置文件属性功能调用(43H)改变属性为非只读属性。在使用DOS3.1以上版本的网络环境下调用此功能时必须蛤有创建访问目录及其文件的权利。

**错误信息:**

如果出现错误，进位标志 CF 置位且 AX 以下数值之一。

02H DOS 没有找到指定文件。

03H 指定的路径名非法，可能的原因之一是指定的子目录不存在。

05H 拒绝访问文件，可能的原因是文件为只读文件。

**参见:**

3CH.. 创建一个文件 (CREAT)

3EH.. 关闭文件描述字 (文件句柄)

3FH.. 读取文件或设备

40H.. 写入文件或设备

42H.. 移动读 / 写指针 (LSEEK)

43H.. 取或置文件属性 (CHMOD)

57H.. 取或置文件日期和时间

59H.. 取扩展错误

**程序实例:**

以下三个实例均使用41H功能调用删除名为 TESTFILE.TMP 的文件。

### Assembly Language Usage EXample:

```
;  
; DELETE FILE (41H)  
;  
code segment public  
assume cs:code,ds:code  
    org      100h  
start:  jmp      begin  
file     db       'testfile.tmp',0  
msg      db       'TESTFILE.TMP not found or access denied.',  
        db       '0dh,0ah,'$'  
begin:  mov ax,cs           ;set up ds  
        mov ds,ax           ;to same as cs  
        mov dx,offset file ;addr of path for file to dele  
        mov ah,41h          ;delete file function request  
        int 21h             ;call DOS  
        jnc done             ;check if error  
        mov dx,offset msg  ;address of error message  
        mov ah,09h          ;display string function reques  
        int 21h             ;call DOS  
done:   mov ah,4ch          ;terminate process funct request  
        int 21h             ;call DOS  
code:   ends               end of code segment  
        end start            ;start is the entry point.
```

### Pascal Usage EXample:

```
{ Delete File ( $ 41) }
```

```
PROGRAM delete_file;
```

#### CONST

```
dos_delete_file = $ 41;  
carry_flag = 1;
```

#### TYPE

```
regpack = PECORD  
CASE integer OF  
1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
```

```

        2 : (al,ah,b1,bh,c1,ch,d1,dh :byte) ;
    END ;

VAR
  recs : regpack

filename : string [50] ;

BEGIN
  WITH regs DO
    BEGIN
      filename := 'testfile.tmp' + chr(0) ;
      ah := dos_delete_file ;
      ds := seg(filename[1]) ;
      dx := ofs(filename[1]) ;
      msdos(regs) ;
      IF ((flags AND carry_flag) = 1) THEN
        writeln('TESTFILE.TMP not found or access denied.')
    END ;END.

```

### C Usage Example

```

/* * * Delete File (Dx4l)
 */
#include <dos.h>
union REGS inregs,outregs;
main( )
{
  inregs.x.dx = (int) &file[ 0 ];
  inregs.h.ah = Dx4l;
  intdos(&inregs,&outregs);
  if (outregs.x.flag != D)
    printf("TESTFILE.TMP not found or access denied.");
}

```

### DOS 功能调用 42H

Lseek--移动读 / 写指针

功能调用:

## 42H Lseek—移动读 / 写指针

DOS 版本:

2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明:

改变一个已打开文件的读 / 写指针的位置。读 / 写指针标识文件读（功能调用 3FH）或写（功能调用 40H）的地点。DOS2.X 和 DOS3.X 为完成随机输入 / 输出以此功能调用改变读 / 写指针。

为指出新的指针位置，需提供一个方法代码来表明指针的起始位置，同时还要提供一个 32 位数值（在 CX:DX 里）表明从起始位置移动指针的偏移量。

如果对网络盘里的文件应用此功能调用，且在本地打开文件时程序设置共享方式为拒绝读或拒绝写，则 DOS 就要调整实际含有该文件的机器的指针信息，如果用其它不同的共享方式打开文件，则 DOS 只调整本地计算机上的指针信息。

入口参数:

调用前，需设置:

AH = 42H 指出功能调用号。

AL 必须置以下方法代码:

0: 从文件开始移动指针的偏移量 (CX: )。

1: 从文件指针的当前位置开始，按 CX: DX 所确定的偏移量向前移动指针。

2: 从文件尾部开始，按 CX: DX 确定的偏移量向前移动指针。此方法代码在 CX: DX 中内容为 0 时，可决定当前文件的大小。

BX 由创建一文件的 3CH 或打开一文件的 3DH 功能调用返回的文件描述（文件句柄）。

CX: DX 其内容为 32 位无符号长整型数，该数表明从 AL 指定的位置起文件指针移动多少字，DX 为整数的低位部分，CX 为高位部分。

出口参数:

返回后，设置为:

DX: AX 其内容为 32 位长整型数，表明新的读 / 写指针位置（作为文件起始的偏移量）。AX 为整数低数，DX 为高位。

其它要求:

调用前，用创建文件的 3CH 或打开一文件的 3DH 来打开一个文件。

错误信息:

如果出现错误，进位标志 CF 置位且 AX 有以下数值之一。

01H 非法方法代码

06H 文件描述字（文件句柄）不合法。

参见:

3CH—创建一文件 (CREATE)

3DH—打开一文件

3EH—关闭文件描述字（文件句柄）

40H--写入文件或设置

43H--取或置文件属性 (CHMOD)

57H--取或置文件的日期和时间

59H--取扩展错误代码

#### 程序实例::

以下三个实例均使用 42H 功能调用，寻长至 TESTFILE.ASC 文件的中部，并在屏幕显示文件剩余部分的内容。

#### Assembly Language Usage Example:

```
;  
;MOUVE FILE READ / WRITE POINTER(42H)  
;  
code segment public  
assume cs:code,ds:code  
    org      LOOh  
start: jmp      begin  
file  db      'testfilc.asc',  
handle dw      ?  
buffer db      6s dup(?)  
msg1  db      'Unable to openTESTFILE.ASC.',Odh,Oah,'$'  
msg2  db      'Unable to seek TESTFILE.ASC.',Odh,Oah,'$'  
msg3  db      'Unable to read TESTFILE.ASC.',Odh,Oah,'$'  
begin: mov      ax,cs          ;set up ds  
            mov      ds,ax          ;to same as cs  
            mov      dx,offset file ;address of file to open  
            mov      al,0           ;access mode  
            mov      ah,3dh         ;open file function request  
            int      21h             ;call DOS  
            jc      operr          ;check if error  
            mov      handle,ax       ;save file handle  
            mov      bx,handle       ;file hndl for file to seek into  
            mov      cx,0           ;high part of offset into file  
            mov      dx,.650          ;low part of offset into file  
            mov      al,0           ;seek from start of file  
            mov      ah,42h         ;move read / write ptr funct req  
            int      21h             ;call DOS  
            jc      seeker          ;check if error  
read:  mov      bx,handle       ;file handle for file to read  
            mov      dx,offset buffer ;address of read buffer
```

```

        mov    cx,64      ;number of bytes to read
        mov    ah,3fh     ;read file or device funct req
        int    2Lh       ;call DOS
        jc    readerr   ;check if error
        cmp    ax,0       ;reached end of file yet?
        jc    close      ;close file and wrap it up
        mov    bx,ax     ;number of bytes read
        mov    buffer[bx],'$' ;end of string
        mov    dx,offset buffer;address of buffer
        mov    ah,0qh     ;display string function request
        int    2Lh       ;call DOS
        jmp    read      ;read next buffer full
operr: mov    dx,offset msg1 ;address of open error message
        mov    ah,0qh     ;display string function request
        int    2Lh       ;call DOS
        jmp    done      ;wrap it up
seekerr:mov   dx,offset msg2 ;address of seek error message
        mov    ah,09h     ;display string function request
        int    2Lh       ;call DOS
readerr:mov   dx,offset msg3 ;address of read error message
        mov    ah,09h     ;display string function request
        int    2Lh       ;call DOS
close:  mov    bx,handle   ;file handle for file to close
        mov    ah,3eh     ;close file function request
        int    2Lh       ;call DOS
done:   mov    ah,4ch     ;terminate process funct request
        int    2Lh       ;call DOS
code:   ends
end    start      ;start is the entry point

```

#### Pascal Usage Example:

```
{ Move File Read / Write Pointer( $42)}
```

```
PROGRAM move-pointer;
```

#### CONST

```
dos-open-file = $3d;
dos-close-file = $3e;
```

```

dos-read-file = $31;
dos-move-pointer = $42;
read-access = 0;
from-start = 0;
carry-flag = L;

TYPE
  regpack = RECORD
    CASE integer OF
      1:(ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
      2:(al,ah,bl,bh,cl,ch,dl,dh:byte);
    END;
VAR
  regs:regpack;
  filename:string[50];
  handle:integer;
  buffer:ARRAY [1..64] OF char;
BEGIN
  WITH regs DO
    begin
      filename:='testfile.asc'+chr(0);
      ah:=dos-open-file;
      ds:=scg(filename[1]);
      dx:=ofs(filename[1]);
      al:=read-access;
      msdos(regs);
      IF((flags AND carry-flag)=1) THEN
        writeln('Unable to open TESTFILE.ASC.')
      ELSE
        BEGIN
          handle:=ax;
          ah:=dos-move-pointer;
          bx:=handle;
          cx:=0;
          dx:=1650;
          al:=from-start;
          msdos(regs);
          IF ((flags AND carry-flag)=1)THEN
            writeln('Unable to seek TESTFILE.ASC.')
        end;
    end;
end;

```

```

ELSE
  WHILE ax < > 0 DO
    BEGIN
      bx:= handle;
      ds:= seg(buffer);
      dx:= ofs(buffer);
      cx:= 64;
      ah:= dos-read-file;
      msdos(regs);
      IF ((flags AND carry-flag)= 1) THEN
        BEGIN
          writeln('Unable to read TESTFILE.ASC.')
          ax:= 0;
        END
      ELSE IF ax < > 0 THEN
        FOR i:= 1TO ax DO
          write(buffer[i]);
      END;
      bx:= handle;
      ah:= dos-close-file;
      msdos(regs);
    END;
    bx:= handle;
    ah:= dos-close-file;
    msdos(regs);
  END;
END.

```

#### C Usage Example:

```

/*
 * Mov File Read / Write Pointer(0x42)
 */

#include <dos.h>

union REGS inregs,outregs;
char filename[] = "testfile.asc";
main()

```

```

{
    char buf[64];
    int handle;

    inregs.x.dx = (int) &filename[0];
    inregs.h.al = 0;
    intrgs.h.ah = 0x3d;
    intdos(&inregs,&outregs);
    if (outregs.x.cflag != 0)
        printf("Unable to open TESTFILE.ASC.");
    else {
        handle = outregs.x.ax;
        intrgs.x. cx = 0;
        inregs.x.dx = 1650;
        inregs.h.al = 0;
        inregs.h.ah = 0x42;
        intdos( * inregs.&outregs);
        if (outregs.x.cflag != 0)
            printf("Unable to seek TESTFILE.ASC.");
        else{
            inregs.x.bx = handle;
            inregs.x.dx = (int) &buf[0];
            inregs.h.ah = 0x3f;
            for (;;) {
                intdos (&inregs,&outregs);
                if (outregs.x.cflag != 0) {
                    printf("Unable to read TESTFILE.ASC");
                    break;
                }
                if (outregs.x.ax == 0) break;
                else {
                    buf[outregs.x.ax] = 0;
                    printf("%s",buf);
                }
            }
            inregs.x.bx = handle;
            inregs.h.ah = 0x3e;
            intdos(&inregs,&outregs);
        }
    }
}

```

```
}
```

## DOS 功能调用 43H

### CHMOD—获取或设置文件属性

#### 功能调用:

43H CHMOD—获取或设置文件属性

#### DOS 版本

2.0, 2.1, 3.0, 3.1, 3.2, 3.3

#### 说明:

获取或设置文件属性。调用前 AL 寄存器置 1，且在 CX 中指定新的属性。

#### 入口参数:

调用前，需设置:

AH = 43H 指出功能调用号。

AL 表明是置还是取文件属性，规定数值如下:

0: 获取当前文件属性。

1 设置文件属性，在 CX 中规定其数值。

CX 设置文件属性时，低位字节应含新文件的文件属性，如下:

位

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

给出文件某属性，置相应位为 1；不删除文件属，置为 0。

对于网络文件，无需改变文件的归档属性。为了改变任何其它属性需要具有创建访问权利。

位

说明

0 只读文件。

1 隐含文件。

2 系统文件。

3 卷标，此功能调用不能改变卷标，否则出现错误信息。

4 子目录，此功能调用不能改变子目录属性，否则出现错误信息。

5 归档。

6~7 未用。

DS: DX 指向含驱动器、目录路径名和要改变属性的文件名的ASCII 字串，字串以 0 字节结尾。

#### 出口参数:

返回后，设置为:

CX 指出新的文件属性。该数值的格式同上述人口参数中 CX 寄存器的数值相同。

**错误信息：**

如果出现错误，进位标志 CF 置位，且 AX 取以下数值之一：

01H 功能调用非法，原因是 AL 数值是非 0 或非 1 的其它数值。

02H 指定的文件没找到。

03H 至少有一个目录路径名不存在。

05H 拒绝访问，属性不能改变，因为文件是一个目录或卷标。

**参见：**

59H--取扩展错误代码

**程序实例：**

以下在个程序实例均应用 43H 功能调用，取出 IBMDOS.COM 文件的属性，并在屏幕上显示属性。

**ASSEMBLY Language Usage Example:**

```
; ; change mode(43H)
;
code segment public
assume cs:code,ds:code
    org      100h
start: jmp      begin
file    db      '\ibmdos.com',0
msg1   db      'Unable to find IBMDOS.COM.',0dh,0ah, '$'
msg2   db      'File attributes of IBMDOS.COM are : ', '$'
msg3   db      'read-only ', '$'
msg4   db      'hidden ', '$'
msg5   db      'system ', '$'
msg6   db      'volume label ', '$'
msg7   db      'subdirectory ', '$'
msg8   db      'archive ', '$'
msg9   db      'normal ', '$'
attrib dw      ?
begin: mov     ax,cs          ;set up ds
        mov     ds,ax          ;to same as cs
        mov     dx,offset file ;address of file to open
        mov     al,0            ;get file attributes
        mov     ah,43h          ;change mode function request
        int     21h            ;call DOS
```

```

jc    error      ;check if error
mov   attrib,cx  ;save file attributes
mov   dx,offset msg2 ;address of file attrib message
mov   ah,09h      ;display string function request
int   21h        ;call DOS
cmp   cx,0       ;is file normal?
je    normal     ;yes, display normal message
mov   cx,6       ;loop through bits 0 thru 5
cont: test  attrib,bx  ;start with attribute bit 0
      jz    next      ;no, go to the next bit
      cmp   bx,1       ;is it a read-only file?
      jne  hidden     ;no, check hidden
      mov   dx,offset msg3 ;yes, set up addr of read-only msg
      jmp  disp       ;display message
hidden: cmp   bx,2       ;is it a hidden file?
      jne  system     ;no, check system
      mov   dx,offset msg4 ;yes, set up address of hidden msg
      jmp  disp       ;display message
system: cmp   bx,4       ;is it a volume label?
      jne  dir        ;no, check directory
      mov   dx,offset msg5 ;yes, set up address of system msg
      jmp  disp       ;display message
volume: cmp   bx,8       ;is it a volume label?
      jne  dir        ;no, check directory
      mov   dx,offset msg6 ;yes, set up address of volume msg
      jmp  disp       ;display message
dir :  cmp   bx,16      ;is it a directory
      jne  archive   ;no, check archive
      jmp  disp       ;display message
archive:cmp  bx,32      ;is it an archive?
      jne  next      ;no, fall out the bottom
      mov   dx,offset msg8 ;yes, set up archive message
      jmp  disp       ;display message
normal: mov   dx,offset msg9 ;set up normal message
disp:  mov   ah,09h      ;display string function request
      int   21h        ;call DOS
next:  shl   bx,1       ;get next attribute bit
      loop  cont      ;loop until all 6bits examined
      jmp   done       ;wrap it up

```

```

error: mov dx,offset msg1 ;address of not found message
       mov ah,09h      ;display string function request
       int 21h        ;call DOS
done:  mov ah,4ch      ;terminate process funct request
       int 21h        ;call DOS
code:  ends          ;end of code segment
end:   start         ;start is the entry point

```

Pascal Usage Example:

```
{ Change Mode ($43) }
```

```
PROGRAM change mode;
```

CONST

```

dos change mode = $43;
get attributes = 0 ;
carry flag = 1 ;

```

TYPE

```
REGPACK = RECORD
```

```

CASE integer OF
  1 : (ax,bx,cx,dx,bp,xi,di,ds,es,flags:integer);
  2 : (al,ah,bl,bh,cl,ch,dl,dh:byte);
END ;

```

VAR

```

regs : regpack ;
file to check :string [50];
attributes : integer ;
mask : integer ;
i : integer ;

```

BEGIN

```
WITH regs DO
```

BEGIN

```

file to check := '\ibmdos.com' + chr(0) ;
ah := dos change mode ;
ds := seg (file to check[1]) ;

```

```

dx := ofs (file to check[1]) ;
al := get attributes ;
msdos(regs) ;
IF ((flags AND carry flag) = 1) THEN
  writeln( 'Unable to find IBMDOS.COM.' )
ELSE
BEGIN
  attributes := cx ;
  write( ' File attributes of IBMDOS.COM are: ' ) ;
  IF attributes = 0 THEN
    write( ' normal ' )
  ELSE
BEGIN
  MASK := 1 ;
  FOR i := 1 TO 6 DO
BEGIN
  cx := attributes AND mask ;
CASE CX OF
  1 : write( ' read-only ' ) ;
  2 : write( ' hidden ' ) ;
  4 : write( ' system ' ) ;
  8 : write( ' volume label ' ) ;
  16 : write( ' subdirectory ' ) ;
  32 : write( ' archive ' ) ;
END ;
mask := mask * 2 ;
END ;
END ;
END ;
END ;

```

#### C Usage Example:

```

/* 
 * Change Mode (0x43)
 */

```

```
#include <dos.h>
```

```

union REGS inregs,outregs;
char filename[] = "\\\ibmdos.com";
main()
{
    int mask;
    int i;

    inregs.x.dx = (int) &filename[0];
    inregs.h.al = 0;
    inregs.h.ah = 0x43;
    intdos(&inregs,&outregs);
    if (outregs.x.cflag != 0)
        printf("Unable to find IBMDOS.COM.\n");
    else {
        printf("File attributes of IBMDOS.COM ARE: ");
        if (outregs.x.cx == 0) printf("normal ");
        else {
            mask = 1;
            for (i=1;i<=6;++i) {
                switch (outregs.x.cx & mask) {
                    case 1:
                        printf("read-only ");
                        break;
                    case 2:
                        printf("hidden ");
                        break;
                    case 4:
                        printf("system ");
                        break;
                    case 8:
                        printf("subdirectory ");
                        break;
                    case 32:
                        printf("archive ");
                        break;
                }
                mask = (mask << 1);
            }
        }
    }
}

```

```
}
```

## DOS 功能调用 44H 设备的 I/O 控制

### 功能调用:

44H 设备的 I/O 控制

### DOS 版本:

2.0, 2.1, 3.0, 3.1, 3.2, 3.3

### 说明

该功能调用允许完成一系列设备的 I/O 控制操作。这些操作包括取和置设备信息、读或写设备控制通道、检查设备是否已准备好 I/O 操作、检测设备是否有可移动介质、返回访问冲突信息前的 DOS 的置共享重试次数、控制通用的块和字符设备以及取或置逻辑设备映象。

该功能调用提供的操作主要用于设备驱动程序或与设备驱动程序直接通信的程序。除非写自己的设备驱动程序，否则可基本不用该功能调用。

执行的制(或子功能)由设置在 AL 中的数值来决定。因为该功能调用被划分成几个似乎毫不相关的子功能，因此本说明也分成下列各个独立的分段。

AL	分段
00H, 01H	取和置设备信息
02H—05H	读写设备控制通道
06H, 07H	取输入或输出状态
08H	检测设备是否有可移动介质
09H, 0AH	测试网络设备和句柄(即处理程序)
0BH	置共享重试次数
0CH	字符设备的通用 I/O 控制
0DH	块设备的通用 I/O 控制
0EH, 0FH	取和置逻辑驱动器映象

控制通用块设备 (0DH) 子功能提供若干种制，因此说明该子功能的分段仍被进一步划分以包括每个独立的操作。

DOS 功能调用 44H

### 设备的 I/O 控制

#### 获取或设置设备信息

子功能：

#### 获取或设置设备信息

DOS 版本：

2.0, 2.1, 3.0, 3.1, 3.2, 3.3

#### 说明：

这些子功能恢复或设置有关打开文件或设备句柄的信息。这些信息取决于句柄是否代表一个设备或一个磁盘文件。

所取的设备信息不支持网络设备，而设置该设备信息时需要装入 DOS SHARE 命令

### 入口参数·

### 调用前，需设置：

AH=44H 指出功能调用号

AL=子功能号

## 0 获取设备信息

## 1 设置设备信息

BX == 创建文件 (3CH) 或打开文件 (3DH) 功能调用返回的文件句柄。该文件句柄指定取或置设备信息的设备。

**DX** 若设置设备信息，则该寄存器折编码值指出新设备信息，该编码值取决于  
**RX** 中指定的文件句柄是否代表一个设备或一个磁立文件

对于一个设备其编码加上

17

E E D C B A98 7 6 5 4 3 2 1

Resrvd	ISCLN
Resrvd	ISCOT
Resrvd	ISNUL
Resrvd	ISCLK
Resrvd	BINARY
EOF	EOF
ISDEV	ISDEV
Resrvd	Resrvd
OPN~CL	OPN~CL
NETWRK	NETWRK
OUN~FAT	OUN~FAT
CTRL	CTRL
Resrvd	Resrvd

0FH, 0AH-08H, 04H 位为系统保留位, 应置为0。

其它各位含义如下：

**ISCIN = 1,** 若设备是控制台(键盘)输入; 否则置 0。

$ISCOT = 1$ , 若设备是控制台(显示屏)输出; 否则置0。

ISNUL = 1, 若设备是控设备 (用于丢弃输入或输出的逻辑设备); 否则置 0.

ISCLK = 1, 若设备是时钟设备; 否则置 0.

BINARY = 1, 若设备以二进制模式制 (DOS 不检查 Ctrl-Z 作为文件尾字符).  
= 0, 若设备以 ASCII 模式操作 (检查 Ctrl-Z 作为文件尾字符).

EOF = 0, 若输入时遇文件尾.

ISDEV = 1, 若通道是一个设备.  
= 0, 若通道是一个磁盘文件, 0-5 位将在以后说明.

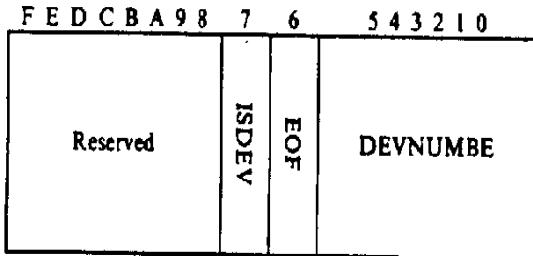
OPN / CL = 1, 若设备支持 DOS3.0 以上版本提供的设备驱动程序功能, 象打开设备、关闭设备、通知可移动介质等.  
该位由设备驱动程序设置, 其它程序不能改变它.

NETWRK = 1, 若设备是远程设备.  
= 0, 若设备是局部设备.  
该位由网络软件设置, 其它程序不能改变它.

OUN / FAT 该位仅适用于 DOS3.2 以上版本, 它对字符设备和块设备有不同的含义. 对于字符设备 (象打印机), 当  
该位为 1 时, 该位设备支持 Print-Until-Busy 功能, 当存储缓冲区象假脱机打印被使用时,  
该功能加速设备执行.  
对于块设备 (象磁盘驱动器), 当该位为 1 时, 该位意指设备驱动程序使用介质描述符字节而不是 FAT 描述符字节来决定设备包含有什么类型的; 当该位为 0 时,  
FAT 标识字节 (FAT 的第一个字节) 被使用且在设备支持的各种介质中被固定在相同的位置.  
该位只能由设备驱动程序设置, 其它程序不能改变它.

CTRL = 1 若设备能够调用子功能 AL = 02H, AL = 03H 处理控制串.  
= 0 若设备不能处理控制串, 设备驱动程序在初始化时置该位, 其它程序不能通过置该位来改变设备性能以便处是控制串.

对一个磁盘文件的编码如下:



OFH-08H 位是系统保留，应置 0。

其它位的含义如下：

**DEVNUMBER** 这 5 位指出包含有磁盘文件设备的块设备码。

0=设备A, 1=设备B等。

EOF 如果由 DEVNUMBER 指明的通道已被写则置该位为0。

SDEV 对于磁盘文件，置该位为 0，（对于设备置该位为 1 且 0-5 位按前面说明的设置）。

#### 出口参数:

返回后，设置为：

**DX** 若取设备信息，则 DOS 在该寄存器返回一个编码值以指出设备信息，该编码值取决于 BX 中规定的文件句柄是否为代表一个设备或一个磁盘文件，设备和磁盘文件的编码与输入参数段相同。

**错误信息：**

若出现错误则置进位标志，AX 寄存器返回出错代码。

01H 原在AL中放置的无效子功能码。

05H 拒绝存取 无效驱动器

06H 原放在 BX 中的地效句柄。

9DH 原放在 DX 中的无效数据。

### 参见 ·

59H--取扩充错误代码。

DOS 功能调用 44H

### 设备的I/O控制

## 读和写设备控制通

子功能：

#### 读和写设备控制通道

DOS 版本：

2.0, 2.1, 3.0, 3.1, 3.2, 3.3

## 说明·

这些子功能允许送控制串到一个设备或从一个设备接收控制串。这些控制是设备驱动程序的控制或应答命令。它们不能被写到设备的文件上或从文件中读出。

这些控制串经常能被用来启动其它 DOS 功能调用不特别支持的制。例如，若系统含有

一个磁带驱动的设备驱动程序，可以使用这些控制串来执行重绕、保持、快进制控制串信息的格式和它的含义由设备驱动程序定义。

对于子功能 2 和 3，设备由 BX 中的文件句柄指出，该设备必须是字符设备（象打印机串行口）；对于子功能 4 和 5，可以通过在 BL 中设置设备码来指定设备，该设备必须是块设备（象磁盘驱动器）。

入口参数：

调用前，需设置：

AX = 44H 指出功能调用号。

AL = 子功能号：

2 从字符设备的控制通道中读设备控制信息，该设备句柄指定的 BX 中。

3 写设备控制信息到字符设备的控制通道中，该设备句柄指定在 BX 中。

4 从块设备的控制通道中读设备控制信息，该块设备驱动号指定在 BL 中。

5 写设备控制信息到块设备的控制通道中，该块设备驱动号指定在 BL 中。

BX 对于子功能 2 和 3，该寄存器包含一个字符设备句柄，该句柄指定取或置设备信息的设备。

对于子功能 4 和 5，该寄存器的低 8 位 (BL) 包含设备的驱动器号：0=默认驱动器，1=A 驱动器、2=B 驱动器等。

DS: DX = 指向一个数据区，对于子功能 3 和 5，该数据区包含 DOS 发送给设备控制通道的信息，对于子功能 2 和 4，指定的该数据区是用来接收设备发出的控制串。

CX 对于子功能 3 和 5，该寄存器指出要发送给控制通道的字节数。

出口参数：

返回后，设置为：

CX 对于子功能 2 和 4，该寄存器指出从控制通道发送给 DS: DX 所指数据区的字节数。

其它要求：

这些子功能仅对能处理控制串的设备是有效的，子功能 0 (取设备信息) 指出设备是否能够处理控制串，返回到 DX 寄存器中态字的第 14 位 (CTRL 位) 指出这种能力。

错误信息：

若出现错误则置进位标志，且 AX 返回如下一种出错代码：

01H AL 中放了无效的子功能码。

05H 拒绝存取。无效的驱动器号。

06H 非法的文件句柄。

0DH 无效的设备控制串。

参见：

59H 取扩展错误代码。

DOS 功能调用 44H

设备的 I/O 控制

取输入和输出状态

**子功能:**

取输入和输出状态

**DOS 版本:**

2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

这些子功能允许检查文件句柄是否已准备好输入(子功能 6)或输出(子功能 7)。

**入口参数:**

调用前, 需设置:

AH = 44H 指出功能调用号。

AL = 子功能号, 如下:

6 检查文件句柄是否已准备好输入。

7 检查文件句柄是否已准备好输出。

BX = 打开文件功能调用(3DH)返回的文件句柄。该文件句柄指定正在检查的文件或设备。

**出口参数:**

AL 指出输入或输出状态:

00H 文件或设备尚未准备好输入或输出, 对于文件, 该值指出读写指针被定在文件尾。

FFH 文件或设备已准备好输入或输出。

**错误信息:**

若出现错误, 则置进位标志, 且 AX 返回出错代码如下:

01H AL 中放了无效的子功能代码。

05H 拒绝存取。

06H 非法的文件句柄。

**参见:**

59H 取扩展错误代码。

**DOS 功能调用 44H**

**设备的 I/O 控制**

**检查可移动介质**

**子功能:**

测试是否是可移动介质

**DOS 版本:**

3.0, 3.1, 3.2, 3.3

**说明:**

这些子功能仅适用 DOS3.0 以上版本, 它检查驱动器是否包含可移动介质(如软盘)或非移动介质(如硬盘)。对于含有可移动介质的设备程序必须检查介质是否已被改变。

对含有非移动介质的设备，程序可假定介质总是相同的。

入口参数：

调用前，需设置：

AH = 44H 指出功能调用号。

AL = 08H 指出子功能号。

BL = 设备的驱动器号 (0 = 默认驱动器, 1 = A 驱动器, 2 = B 驱动器等)。

出口参数：

返回后，设置为：

AX = 状态值如下：

0 设备含有可移动介质 (如软盘)。

1 设备含有非移动介质 (如硬盘)。

错误信息：

若出现错误，则置进位标志，且 AX 返回出错代码如下：

01H 无效的子功能码。

0FH 无效的驱动器号。

参见：

59H 取扩展错误代码。

## DOS 功能调用 44H

### 设备的 I/O 控制

#### 检查网络设备和句柄

子功能：

检查网络设备和句柄

DOS 版本：

3.1, 3.2, 3.3

说明：

这些子功能仅适用于 DOS3.1 以上版本，它们测试一个逻辑设备 (子功能 09H) 或一个句柄 (子功能 0AH) 是否是远程的还是局部的。

入口参数：

调用前，需设备：

AH = 44H 指定功能调用号。

AL = 子功能号如下：

09H 测逻辑设备是否与网络目录相连。

0AH 测试句柄是代表局部还是远程设备。

BL 对于子功能 9，该寄存器为块设备的驱动器号 (0 = 默认驱动器, 1 = A 驱动器, 2 = B 驱动器等)。

BX 对于子功能 0AH，该寄存器为打开文件 (3DH) 功能调用返回的文件名柄，该句柄识别正在检查的文件。

#### **出口参数:**

返回后, 设置为:

**DX=DOS** 返回的设备或句柄的状态, 从设备标题返回的数值代表属性字。对于子功能 9, 若设备是远程设备则置第 12 位 (1000H); 对于子功能 0AH, 若句柄是代表远程设备则置第 15 位 (8000H)

如果挂起磁盘重定向, 则对于局部设备, 子功能 9 返回的属性字并不置 12 位; 因此, 程序不能依靠第 12 位来判定设备是局部还是远程的。

#### **错误信息:**

如果出现错误, 则置进位标志, 且 AX 返回下面一种出错代码如下:

01H AL 中放了无效的子功能码。

06H BX 中放了无效的句柄。

0FH BL 中放了无效的驱动器号。

#### **参见:**

59H 取扩展错误代码。

### **DOS 功能调用 44H**

#### **设备的 I/O 控制**

#### **设置共享重试次数**

#### **子功能:**

置共享重试次数

#### **DOS 版本:**

3.03.13.23.3

#### **说明:**

该子功能仅适用于 DOS3.0 以上版本, 若一个共享文件在第一次存取时发生冲突, DOS 设备共离重试次数和每一重试之间等待的时间。在重试多次失败后, DOS 即返回一个错误信息。

如果不使用该功能调用, 对于所有操作 DOS 试图重试三次, 每一重试之间延迟 65536 个机器周期。

#### **入口参数:**

AH=44H 指出功能调用号。

AL=0BH 指出子功能号。

DX=DOS 重试一次 I/O 操作的次数。

CX=重试之间的延迟循环数, 每一循环延迟 65536 个机器周期。

#### **出口参数:**

无。

#### **错误信息:**

若出现错误, 则置进位标志, 且 AX 返回下面一种出错代码:

01H AL 中放了无效的子功能码, 或没有装入 IBM 网络程序。

**参见:**

59H 取扩展错误代码

**DOS 功能调用 44H**

设备的 I/O 控制

通用字符设备的 I/O 控制

**子功能:**

通用字符设备的 I/O 控制

**DOS 版本:**

3.2, 3.3

**说明:**

该子功能仅适用 DOS3.2 以上版本，它和下一个子功能提供一个与外部设备（象打印机、磁盘驱动器和磁带驱动器）能信的标准接口。这些外部设备 DOS 不提供直接的支持。

该子功能 (0CH) 适用于字符设备，象打印机和串行口设备。它为许多服务建立保护，程序可通过在 CH 和 CL 中设置的数值来选择指定的 I/O 控制服务。CH 值指定通用字符设备的主要类型，CL 值指定操作类型。

采用该子功能和下一个子功能，选择实际招待的操作时，饮食几个分级。每一分级通过寄存器中的数值来选择，有必要在此再来研究一下整个分级（层次）情况。

对于所有的功能调用，可通过置 AH 的数值来选择（就所有的 IOCTL 功能调用而言，AH = 44H）功能调用，可以通过置 AL 中的数值来选择指定的子功能（就通用字符设备的 I/O 控制而言，AL = 0CH），也可以通过置 CH 的数值来选择字符设备的类型。最后，可通过置 CL 的数值来指定操作类型。

虽然该特殊子功能可为许多通用字符设备提供服务，但在 DOS3.2 中该子功能仅提供两种制。两种都用于打印机设备，且它们能取或置打印机重接次数 (iteration count)。

该重接次数用来细调 DOS3.2 的一个功能 (Print Until Busy 忙完了打印)，该功能帮助假脱机打印更有效地完成它们的工作。Print Until Busy (PUB) 功能允许假脱机打印向打印机发送一串猝发的字符而不用等待收到每个字符的准备好指示。由于大多数打印机有一小块存储缓冲区（一行或两行）来存储将要打印的输入字符，因此该猝发方式是可能的。通过发送猝发的字符串，假脱机能立即填满整个打印机缓冲区，然后在发送下一猝发的字符串之前在后台等待缓冲区变为空。

设备驱动程序通过存储一定量从打印机响应的准备好信号来完成 PUB 功能，而不用发送的相应的设备忙信号到正发送字符的程序。由该功能调用设置（或接收）的重接次数就是在向打印程序指出打印机是忙的之前驱动程序存储的准备好响应的次数。该次数通常是打印机缓冲区中的字符数。

并非所有的字符设备都支持 PUB 功能。为了弄清一个指定的设备是否支持该功能，应请求功能调用 44H 的子功能 00H (取设备信息)。

**入口参数:**

调用前，需设置：

AH=44H 指出功能调用号。

AL=0CH 指出通用字符设备的 I/O 控制子功能。

CH= 该子功能所用的字功能所用的字符设备的类型，在DOS3.2中，唯一的合法值是05H，表示打印机设备。

CL=要执行的操作类型

45H 置重接次数

65H 取重接次数

BX=取或置重接次数字符设备的句柄。

DX: DX=指向一个用来取或置重接次数的字，若是重接次数，则在该字中放重接次数。若是取重接次数，则 DOS 在该寄存器对代表的字中放入重接的次数。

出口参数：

返回时，设置为：

DS: DX 若是取重接次数，则返回时 DOS 在该字中放入取来的次数。

错误信息：

若出现错误，则置进位，且 AX 返回一种出错码如下：

01H AL 中放了无效的子功能码。

参见：

44H 设备的 I/O 控制子功能 00H-- 动设备信息。

59H 获取扩展错误代码。

## DOS 功能调用 44H

设备的 I/O 控制

通用块设备的 I/O 控制

子功能：

通用块设备的 I/O 控制

DOS 版本：

3.2, 3.3

说明：

该子功能仅适用于 DOS3.2 以上版本，它和上述的子功能提供一个与外部设备（象打印机、磁盘驱动器和磁带驱动器）通信的标准接口，这些外部设备 DOS 不提供直接的支持。

该子功能（0DH）适用于块设备，象磁盘驱动器。它提供一个低级的设备接口，这个设备接口类似于 BIOS 提供的设备接口。该接口使新设备（象 3.5 英寸磁盘驱动器或其它小型数字磁盘）能被 DOS 支持。它也为能够在相同驱动器上处理不同介质（即 1.2M 软盘和 360K 软盘）支持的设备（象 1.2M 字节软 盘驱动器）提供支持。

有了该低级的支持，一个非标准设备的驱动程序能调用标准的 DOS 功能调用而不是机器特有的 BIOS 功能调用。这使得设备驱动能够在所有的以 DOS 为基础的系统上运

行，而不仅在特定的机器上运行。

类似于子功能 0CH，该子功能也在单个子功能保护下提供几个不同服务，可能通过 CH 和 CL 的数值来选择 I/O 控制服务。CH 数值指定通用块设备的主要类型，而 CL 数值则指定执行的操作类型。

该子功能提供几个不同种类的操作，这些操作包括取和置设备参数，在逻辑设备上读和写一个磁道，在逻辑设备上格式化和检查一条磁道。为了详细地描述每一种操作，该子功能被划分成上列独立的条目：

CL	条目
60H, 40H	取和置设备参数
61H, 41H	在逻辑设备上读和写一条磁道
62H, 42H	在逻辑设备上格式化和检查一条磁道

下面内容所描述的操作，通过允许程序询问特定设备信息来使用设备，这种特定设备中有些信息能够被显示在它自己的介质上（在引导扇区）。当程序格式化一条磁道或当它读或写数据时它能使用这种信息，询问特定设备信息的功能使程序可在不直接用 BIOS 的情况下支持新设备。

当使用该子功能来进行读、写、格式化或检查操作时，程序应招待下列过程：

1. 使用取设备参数操作来查找当前的设备参数；保存这些参数以待后面恢复。
2. 使用置设备参数为执许的操作准备设备。
3. 招待 I/O 操作（读、写、格式化或检查）。
4. 最后，通过置设备参数恢复原来的设备参数。

## DOS 功能调用 44H

### 设备的 I/O 控制

#### 通用块设备的 I/O 控制

#### 获取或设置设备能数

#### 子功能：

通用块设备的 I/O 控制

#### 操作：

获取或设置设备参数

#### DOS 版本：

3.2, 3.3

#### 说明：

该操作设置或恢复有关块设备特有和详细的信息。该设备信息包括设备类型、柱面数、介质密度、BIOS 参数区和设备磁道格式等项目。通过取一个信息，程序就可测定设备包含什么类型的介质和如何进行读写信息，也就是说，如果设备支持不同种类的介质（或以不同方式格式化的介质），程序能把置设备参数作为格式化的准备。

#### 入口参数：

调用前，需设置：

AH=44H 指出功能调用号。

AL=0DH 指出通用块设备子功能的I/O控制。

CH=该子功能所用的块设备类型；在DOS3.2中，唯一的合法数值是08H。

CL=执行的操作类型

40H 置设备参数

60H 取设备参数

BL=块设备的驱动器号(0=默认驱动器, 1=A驱动器, 2=B驱动器等)

DS: DX=参数区的前一个字节地址，当置设备参数时填该参数区。当取设备参数时，功能调用返回在该参数区中的信息。

参数区的格式如下：

DS: DX 偏移

01H	特殊功能 (字节)
02H	设备类型 (字型)
03H	设备属性 (字)
04H	
05H	柱面数 (字)
06H	
07H	介质类型 (字节)
08H	BIOS 参数区 (13字节)
15H	磁道格式 (n字节)

参数区各字段的含义如下：

特殊功能：

该字节所用特殊功能信息的编码如下：

位

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

位	说明
0	当该位为0时, BIOS参数区(BPB)字段包含一个新的在对设备进行I/O操作时使用的默认BPB。当该位为1时,通知设备驱动程序返回到当前BPB(与默认的那个相对而言)准备下次取用该设备参数。
1	当该位为0时,设备驱动程序应在参数区建立使用所有信息分段的设备。当该位为1时,仅磁道格式字段是有意义的。
2	当该位为0时,一条磁道上的单个扇区能具有可变尺寸。当该位为1时,一条磁道上的所有扇区的大小都是相同的,该位应总置为1。
3-7	这些位是系统保留位,必须置为0。

**设备类型:** 置该字段以指出设备类型如下:

数值	设备类型
0	320K / 360K, 5.25英寸软盘驱动器
1	1.2M、5.25英寸软盘驱动器
2	720K、5.25英寸软盘驱动器
3	单密度、8英寸软盘驱动器
4	双密度、8英寸软盘驱动器
5	硬盘
6	磁带驱动器
7	其它设备

**设备属性:**

该字的编码用来指出设备的任何特殊属性,该字各数位的含义如下:

位

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

位	说明
0	若该位为 0，则设备是可移动介质（象软盘驱动器），若为 1，则介质是非移动介质（硬盘）。
1	若该位为 0，则驱动器不能检测驱动器的门是否已被打开（也就是说，介质是否已被改变）。若该位为 1，驱动器能检测介质是否已被改变。
2-F	这些位是系统保留位，应置 0。

#### 柱面数：

该字段指出设备的最大柱面数。不必置该字段，当取设备信息时由设备驱动程序置该字段。

#### 介质类型：

对于支持不同种类介质的设备（如 AT 机上的 1.2M 软盘驱动器），该字段指出驱动器要求的介质类型。对于不同的设备驱动程序，该字段有不同的含义。对于 1.2M 软盘驱动器，定义下面的数值为：

数值	说明
0	介质是单密度 1.2M 软盘，这是默认的介质类型。
1	介质是倍密度 320K / 360K 软盘。

#### BIOS 参数区：

BPB 是一个包含有关介质格式信息的 13 个字节信息字段。当特殊功能字段的 0 位为 0 时，该 BPB 被用作为设备新的 BPB，通常作为格式化的准备。若特殊功能字段的 0 位为 1，则设备驱动程序使用该 BPB 与介质引导记录中的版本进行比较，这种比较可检测介质是否已被改变。

BPB 的格式如下：

每个扇区的字节数 (字)
每一簇的扇区数
必须 2 的幂 (字节)
保留的扇区数
从逻辑扇区 0 开始 (字)
介质上的文件分配表
(FAT) 数 (字节)
允许的数据项及项目数 (字)
介质上的扇区总数，包括
数据、目录和引导扇区 (字)
介质描述符 (字节)
每 FAT 的扇区数 (字)

除了介质描述符外，BPB 的字段是自解释的。介质描述符标识一种特殊类型的介质（例如，单面 8 扇区盘）。对于某些设备，介质描述符与功能调用 1BH 和 1CH（取 FAT 信息）返回的 FAT 标识字节有相同的数值。该介质描述符字节是 DOS 版本 2 和 3 唯一地标识介质类型，（DOS 版本 1 驱动程序用的是 FAT 中 ID 字节）的机制。可是介质描述符可接收从 00H 到 FFH 中的任何数值，介质描述符中数值如何与实际的介质类型对应，完全取决于设备驱动程序。

#### 磁道格式：

这是一张长度可变的表，它对一特定磁道描述扇区个数和大小。该表的格式如下：

每道的扇区数 (字)
第一个扇区的区号 (字)
第一个扇区的大小 (字)
第二个扇区的扇区号 (字)
最后一个扇区的扇区号 (字)
最后一个扇区的大小 (字)

若特殊功能字段的第 2 位为 0，则该表中的扇区大小是可变的。可是正常情况下，每一磁道上所有扇区的大小都是相同的。

#### 出口参数：

返回后，设置为：

DS: DX 若是取设备参数，则该寄存器对指向包含这些参数的参数区。该参数区与输入参数段中描述的相同，但下列字段除外：

特殊功能域：当取设备信息时，仅该字段的 0 位是有意义的。当该位为 0 时，BISO 参数区字段包含设备默认的 BPB。当该位为 1 时，它包含从当前介质获得的 BPB。

该字段的各位均置为 0。

磁道格式：当取设备参数时，该字段无返回内容。

#### 错误信息：

若出现错误，则置进位标志，且 AX 返回如下出错代码：

01H AL 中放了无效的子功能码。

05H 拒绝存取。无效的驱动器号。

#### 参见：

44H 设备的 I/O 控制子功能 00H—取设备信息

59H 取扩充错误代码

## DOS 功能调用 44H

## 设备的 I/O 控制

### 通用块设备的 I/O 控制

在逻辑设备上读和写一磁道

子功能:

通用块设备的 I/O 控制

操作:

在逻辑设备上读和写一磁道

DOS 版本:

3.2, 3.3

说明:

这里描述的两个操作允许程序从块设备读和向块设备写。这些操作连同前面描述的置设备参数操作一起工作。在从块设备读和写之前，必须用 01H 操作置设备参数。

这里描述的读和写操作是相当低级的，它们不涉及象文件或目录等 DOS 概念。相反，必须指定扇区和磁道来进行读或写。

这些操作相当于 BIOS 的读和写操作，在 DOS 中它们可用来标准化低级接口。对于与 IBM BIOS 不兼容但建立在 DOS 基础之上的机器来说，该功能允许非标准设备在 DOS 级水平上得到支持。

入口参数:

调用前，需设置:

AH=44H 指出功能调用号。

AL=0DH 指出通用块设备的 I/O 控制子功能。

CH=该子功能所用的块设备类型，在DOS3.2中，唯一的合法数值是08H。

CL=要招待的操作类型如下:

41H 在逻辑设备上写磁道。

61H 在逻辑设备上读磁道。

BL=块设备的驱动器号 (0=默认驱动器, 1=A驱动器, 2=B驱动器等)

DS: DX=参数区前一字节的地址。该参数区说明读或写的位置，列出读或写的扇区数，且指向一内存缓冲区。

参数区的格式如下:

DS: DX 偏移

01H	特殊功能 (字节)
02H	
03H	磁头号 (字)
04H	柱面号 (字)
05H	

06H	第一个扇区(字)
07H	
08H	扇区数(字)
09H	
0AH	传送区偏移部分的地址
0BH	(双字)
0CH	基本操作
0DH	

特殊功能: 将该字节为0.

磁头号: 该字指出块读或写的磁头号。对于软盘，磁头号就是盘面号(0或1)。

柱面号: 该字指出倍待读或写的柱面号。对于软盘，柱面叫与磁道号相同。

第一扇区: 该字指出读或写的第一扇区的扇区号，该扇区号即磁头和柱面号一起所选的起始磁道号。不象 BIOS，该操作假定扇区从0开始编号，因此第三扇区是扇区2。

扇区数: 该字指出从第一扇区号开始读或写的扇面数。

传送区地址: 这两个字包括内存缓冲区的起始段地址。当向设备写数据时，该数据区保存要写的数据；当从设备读数据时，该数据区用来接收数据。应保证所设置的内存缓冲区大小足以接收任何需要的数据。

#### 出口参数:

若是从设备读取数据，则参数区中指定的传送区包含从逻辑设备读取的信息。

#### 错误信息:

若出现错误，则置进位标志，且AX返回如下错代码：

01H AL存放无效的子功能号。

05H 拒绝存取。无效的驱动器号。

#### 参见:

44H 设备的I/O控制子功能01H—取设备信息。

59H 取扩充错误代码。

BISO 中断13H，功能调用02H—读磁盘扇区。

BISO 中断13H，功能调用03H—写磁盘扇区。

### DOS 功能调用 44H

#### 设备的I/O控制

#### 通用块设备的I/O控制

#### 在逻辑设备上格式化和检查一磁道

#### 子功能:

通用块设备的I/O控制

#### 操作:

在逻辑设备上格式化 检查一磁道

DOS 版本:

3.2, 3.3

说明:

这两个操作允许程序格式化和检查逻辑设备磁道的内容，在执行这些操作任一个之前，必须用 01H 操作置设备参数。

这里描述的操作是低级操作，这些操作相当于 BIOS 中通用的那些操作。在 DOS 中，它们可用为标准化低级接口，对于与 IBM BIOS 级不兼容，但对于建立在 DOS 基础之上的机器来说，这就允许非标准设备在 DOS 水平上得到支持。

入口参数:

调用前，需设置:

AH = 4H 指出功能调用号。

AL = 0DH 指出通用块设备的 I/O 控制子功能。

CH = 该子功能所用的块设备类型，DOS3.2 中唯一合法的数值是 08H。

CL = 执行的操作类型如下:

42H 在逻辑设备上格式化和检查一磁道。

62H 在逻辑设备上检查一磁道。

BL = 块设备的驱动器号 (0 = 默认驱动器, 1 = A 驱动器, 2 = B 驱动器等)

DS: DX = 参数区前一个字节的地址，该参数区说明格式化或检查的位置。

参数区的格式如下:

DS: DX 偏移

01H	特殊功能 (字节)
02H	磁头号 (字)
03H	
04H	柱面号 (字)
05H	

参数区各字段的含义如下:

特殊功能: 置该字节为 0。

磁头号: 该字指定磁头号，磁头号和柱面号的组合指定要格式化和检查的磁道。对于软盘，磁头号就是盘面号 (0 或 1)。

柱面号: 该字指出要格式化和检查的柱面号，对于软盘，柱面号与磁道相同。

出口参数:

若正在格式化磁道，所选磁道即被格式化。

错误信息:

若出现错误，则置进位标志，并 AX 返回如下出错代码:

01H AL 中放了无效的子功能码。

**05H 拒绝存取。无交的驱动器号。**

**参见:**

44H---设备的 I/O 控制 子功能 01H---置设备信息。

59H---取扩充错误代码

BIOS 中断 13H, 功能调用 04H---验证盘扇区

BIOS 中断 13H, 功能调用 05H---格式化磁道

## **DOS 功能调用 44H**

**设备的 I/O 控制**

**获取和设置逻辑驱动器映象**

**子功能:**

获取和设置逻辑驱动器映象

**DOS 版本:**

3.2, 3.3

**说明:**

这些子功能仅适用于 DOS3.2 以上版本，一种子功能允许对相同一块设备分配两个驱动器字母。另外一种子功能可以分清所确定的是哪一个逻辑设备。作为使用这些子功能的一个例子，可把 B 驱动器分配给通常称为 A 驱动器的软盘驱动器。以后无论程序涉及的是 A 还是 B 驱动器，总对同一驱动器进行存取。在这种情况下，DOS 将不会发出要求在驱动器中应放置正确的盘的信息。

如果尚未分配逻辑驱动器，就试图取出逻辑驱动器象，这时就会出现一个错误信息。

**入口参数:**

调用前，需设置：

AH = 44H 指出功能调用号。

AL = 指出要执行的子功能如下：

0 FH 获取逻辑驱动器映象。

0 FH 设置逻辑驱动器映象。

BL = 物理驱动器。可能的数值包括：0 = 默认驱动器，1 = A 驱动器，2 = B 驱动器等。

BH = 映象到 DL 中指定的物理驱动器的逻辑驱动器。若是置逻辑驱动器映象，由于指定默认驱动器对于该参数是不恰当的。因此驱动器号与 DL 中指定的驱动器事情略有不同，编号如下：

DL 中指定的驱动器号略有不同，编号如下：

0 A 驱动器。

1 B 驱动器。

等等。

若是取逻辑驱动器映象，不要置该寄存器。

**出口参数:**

返回后，设置为：

AX = 映象到物理驱动器的逻辑驱动器号，如果是逻辑驱动器映象且没有出现错误（即进位标志没被设置）。

错误信息：

若出现错误，置进位标志，且 AX 返回如下出错代码：

01H AL 中施了无效的子功能码。

05H 拒绝存取，无效的驱动器号。

参见：

59H—取扩展错误代码。

## DOS 功能调用 45H

### 复制文件句柄 (DUP)

功能调用：

45H DUP—复制文件句柄

DOS 版本：

2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明：

该功能调用接收一个现有的文件句柄且返回一个表示相同文件的新文件句柄。新文件句柄的读 / 写指针与原文件句柄的读 / 写指针在文件中处于相同位置。

由该功能调用产生的复制文件句柄是原先文件句柄的一个映象。若移动任一文件句柄的读 / 写指针，另一个句柄的读 / 写指针也相应地移动。

该功能调用对于 I/O 重定向是有用的，有了它，就能够保存一个复制文件句柄，留待后用。然后可用功能调用 46H（强迫复制文件句柄）重定向 I/O。为了返回 I/O 到原来的状态，包括读 / 写指针的位置，再次使用功能调用 46H 以恢复用功能调用 45H 复制的原先的文件句柄。

入口参数：

调用前，需设置寄存器：

AH = 45H 指出功能调用号。

BX = 建立文件或 (3CH) 或打开文件 (3DH) 功能调用所返回的文件句柄，即所要复制的文件句柄。

出口参数：

返回后，设置寄存器为：

AX = 指向相同文件的新文件句柄。

错误信息：

若出现错误，则置进位标志，且 AX 返回如下出错代码：

04H 无多余文件句柄可用。

06H 非法的文件句柄。

参见：

46H---强行复制文件句柄。

59H---取扩展错误代码。

程序实例：

下面三个实例均使用功能调用 45H 为一名为 TEXTFILE.ASC 的文件建立一复制文件句柄。

Assembly Language Usage Example:

```
; ; DUPLICATE FILE HANDLE(45h)
;
code segment public
assume cs:code,ds:code
    org 100h
start: jmp begin
file db testfile.asc',0
handle1 dw ?
handle2 dw ?
msg1 db TESTFILE.ASC opened.',0dh,0ah, $
msg2 db TESTFILE.ASC not opened.',0dh,0ah, $
msg3 db TESTFILE.ASC handle duplicated.',0dh,0ah, $
msg4 db TESTFILE.ASC handle not duplicated.',0dh,0ah, $
begin: mov ax,cs      ;set up ds
       mov ds,ax      ;to same as cs
       mov dx,offset file ;address of file to open
       mov al,0        ;access mode
       mov ah,3dh      ;open file function request
       int 21h         ;call DOS
       jc operr        ;check if error
       mov handle1,ax  ;save file handle
       mov dx,offset msg1 ;address of opened message
       mov ah,09h      ;display string function request
       int 21h         ;call DOS
       mov bx,handle1  ;handle to duplicate
       mov ah,45h      ;duplicate file handle funct req
       int 21h         ;call DOS
       jc duperr       ;check if error
       mov handle2,ax  ;save duplicate file handle
       mov dx,offset msg3 ;address of duplicated message
       mov ah,09h      ;display string function request
```

```

        int    21h          ;call DOS
        jmp    done          ;wrap it up
duperr: mov   dx,offset msg4 ;address of not duplicated message
        mov   ah,09h         ;display string function request
        int    21h          ;call DOS
        jmp    done          ;wrap it up
operr:  mov   dx,offset msg2 ;address of not opened message

        mov   ah,09h         ;display string function request
        int    21h          ;call DOS
done:   mov   ah,4cb         ;terminate process funct request
        int    21h          ;call DOS
code   ends             ;end of code segment
        end    start         ;start is the entry point

```

#### Pascal Usage Example:

```
{ Duplicate File Handle ($45)}
```

```
PROGRAM duplicate file handle;
```

#### CONST

```

dos open file = $3d;
dos close file = $3e;
dos dup file handle = $45;
read access = 0;
carry flag = 1;

```

#### TYPE

```

regpack = RECORD
  CASE integer OF
    1:(ax,bx,cx,dx,dp,si,di,ds,es,flags:integer)
    2:(al,ah,b1,bh,cl,ch,dl,dh:byte);
END ;

```

#### VAR

```

regs : regpack ;
file to open : string[50];
handle1 : integer ;

```

```

handle2 : integer ;

BEGIN
  WITH regs DO
    BEGIN
      file to open := testfile.asc' + chr(0);
      ah := dos open file;
      ds := seg(file to open[1]);
      dx := seg(file to open[1]);
      al := read access;
      msdos(regs);
      IF((flags AND carry flag)=1)THEN
        writeln(' TESTFILE.ASC not opened.')
      ELSE
        BEGIN
          writeln(' TESFILE.ASC not opened.')
          handle1 := ax ;
          bx := handle1 ;
          ah := dos dup file handle ;
          handle2 := ax ;
          msdos(regs);
          IF((flags AND carry flag)=1) THEN
            writeln(' TESTFILE.ASC handle not duplicated.')
          ELSE
            writeln(' TESTFILE.ASC handle duplicated.');
          bx := handle1;
          ah := dos close file ;
          msdos(regs);
        END ;
      END ;
    END.

```

#### C Usage Example:

```

/* *
 * Duplicate File Handle (0x45)
 */

```

```
#include <dos.h>
```

```

union REGS inregs,outregs;
char filename[]="testfile.asc";

main()
{
    int handle1,handle2;

    inregs.x.dx = (int) &filename[0];
    inregs.h.al = 0;
    inregs.h.ah = 0x3d;
    intdos(&inregs,&outregs);
    if (outregs.x.cflag != 0)
        printf("TESTFILE.ASC not opened.");
    else {
        printf("TESTFILE.ASC not opened.");
        handle1 = outregs.x.ax;
        inregs.x.bx = handle1;
        inregs.h.ah = 0x45;
        intdos(&inregs,&outregs);

        handle2 = outregs.x.ax;
        if (outregs.x.cflag != 0)
            printf("TESTFILE.ASC handle not duplicated.");
        else printf("TESTFILE.ASC handle duplicated.");
        inregs.x.bx = handle1;
        inregs.h.ah = 0x3e;
        intdos(&inregs,&outregs);
    }
}

```

## DOS 功能调用 46H

强行复制文件句柄

**功能调用：**

46H 强行复制文件句柄

**DOS 版本：**

2.0, 2.1, 3.0, 3.1, 32, 3.3

**说明：**

该功能调用类似于功能调用 45H，除了不返回一新文件句柄外，该功能调用还接收

第二个当前存在的文件句柄，并把它作为第一个文件句柄的拷贝。

当该功能调用改变第二个文件句柄时，它抛弃文件和句柄之间原有的关系。因此，如果第二个文件句柄涉及一个已打开的文件，则在复制之前需先关闭该文件。

当该功能调用完成时，第二个文件句柄与第一个文件句柄涉及同一个文件。它的读／写指针与第一个文件句柄的读／写指针在文件中处于相同的位置。第二个文件句柄现在是第一个文件句柄的一个映象，如果移动两个中任一文件句柄的读／写指针，则另一个文件句柄的读／写指针也相应地移动。

该功能调用对于从标准设备（象标准输入、标准输出或标准打印机）到文件的 I/O 重定向是有用的。有 I/O 重定向，即一个程序就可不必确切知道从哪里输入或向哪里输出。它能简单地从标准设备读或向一个标准输出设备写。这些标准设备可适当重定向以便建立从文件读或向文件写的程序。

为了重定向 I/O，必须使用该功能调用和功能调用 45H（复制文件句柄）且按以下步骤进行：

1、先使用功能调用 45H 复制一个标准设备的文件句柄到另一文件句柄。当需要恢复标准设备到原来状态时，则保存原来的句柄留待以后用。

2、然后在 CX 中设置标准设备的文件句柄，在 BX 设置要重定向 I/O 的文件句柄，请求功能调用 46H。改变标准设备的文件句柄指向的所定的文件。用标准设备执行 I/O 操作的程序现在将它的 I/O 重定向到所定的文件或由所指定的文件重定向 I/O。

3、为了使 I/O 返回到标准设备，只需简单地再次使用功能调用 46H 将所保存的标准设备句柄的版本复制到原来的标准设备句柄。

#### 入口参数：

调用前，需设置寄存器：

AH=46H 指出功能调用号。

BX = 建立文件 (3CH) 或打开文件 (3DH) 功能调用返回的文件句柄，这是要复制的文件句柄。

CX = 第二个文件句柄。由于该功能调用返回后，即使 CX 中的文件句柄与 BX 中的文件句柄指向同一个文件。

#### 出口参数：

无。

#### 错误信息：

若出现错误，则置进位标志，且 AX 返回如下出错代码：

04H 无可用文件句柄。

06H 非法的文件句柄。

#### 参见：

45H—复制文件句柄 (DUP)

59H—取扩展错误代码。

#### 程序实例：

下面三个实例使用功能调用 46H，为一文件名为 TESTFILE.ASC 的文件复制文件句柄。

**Assembly Language Usage Example:**

```
;  
; FORCE DUPLICATE FILE HANDLE(46H)  
;  
code    segment public  
assume cs:code,ds:code  
        org    100h  
start: jmp    begin  
file  db     'testfile.asc',0  
handle dw     ?  
msg1  db     'TESTFILE.ASC opened.',0dh,0ah,'$'  
msg2  db     'TESTFILE.ASC not opened.',0dh,'$'  
msg3  db     'TESTFILE.ASC handle duplicated as the printer.',  
       db     0dh,0ah,'$'  
msg4  db     'TESTFILE.ASC handle not duplicated.',0dh,0ah,'$':  
begin: mov    ax,cs           ;set up ds  
        mov    ds,ax           ;to same as cs  
        mov    dx,offset file  ;address of file to open  
        mov    al,0             ;access mode  
        mov    ah,3dh            ;open file function request  
        int    21h              ;call DOS  
        jc    operr             ;check if error  
        mov    handle,ax         ;save file handle  
        mov    dx,offset msg1   ;address of opened message  
        mov    ah,0qh              ;display string function request  
        int    21h              ;call DOS  
        mov    bx,handle          ;handle to duplicate  
        mov    cx,04h              ;standard printer file handle  
        mov    ah,46h              ;force dup file handle funct req  
        int    21h              ;call DOS  
        jc    duperr             ;check if error  
        mov    dx,offset msg3   ;address of duplicated message  
        mov    ah,09h              ;display string function request  
        int    21h              ;call DOS  
        mov    bx,handle          ;file handle for file to close  
        mov    ah,3ch              ;close file function request  
        int    21h              ;call DOS  
        jmp    done               ;wrap it up
```

```

duperr:mov    dx,offset msq4 ;address of not duplicated msg
              mov     ah,0qh      ;display string function request
              int     21h        ;call DOS
              jmp     done       ;wrap it up
operr:mov    dx,offset msq2 ;address of not opened message
              mov     ah,0qh      ;display string function request
              int     21h        ;call DOS
done:  mov    ah,4ch      ;terminate process funct request
              int     21h        ;call DOS
code   ends          ;end of code segment
end    start         ;start is the entry point

```

**Pascal Usage Example:**

{ Force Duplicate File Handle (\$46)}

PROGRAM force duplicate file handle;

**CONST**

```

dos open file = $3d;
dos close file handle = $46;
read access = 0;
carry flag = 1;

```

**TYPE**

```

reqpack = RECORD
  CASE integer OF
    1:(ax,bx,cx,dx,bp,si,di,ds,es,flqs:integer);
    2:(al,ah,bl,bh,cl,ch,dl,dh:byte);
END;

```

**VAR**

```

reqs:reqpack;
file to open:string[50];
handle:integer;

```

**BEGIN**

```

WITH reqs DO
BEGIN
  file to open:='testfile.asc'+chr(0);

```

```

ah:=dos open file;
ds:=seqfile to open[1];
dx:=ofs(file to open[1];
al:=read access;
msdos(reqs);
IF ((flags AND carry flag)=1) THEN
  writeln(TESTFILE.ASC not opened.)
ELSE
  BEGIN
    writeln('TESTFILE.ASC opened.');
    handle:=ax;
    bx:=handle;
    cx:=4;
    ah:=dos force dup file handle;
    msdos(regs);
    IF ((flags AND carry flag)=1) THEN
      writeln('TESTFILE.ASC handle not duplicated.')
    ELSE
      writeln
        ('TESTFILE.ASC handle duplicated as printer.');
    bx:=handle;
    ah:=dos close file;
    msdos(reqs);
  END;
END;
END.

```

### C Usage Example:

```

/*
 * Force Duplicate File Handle (046)
 */

```

```
#include <dos.h>
```

```
union REGS inreqs,outreqs;
char filename[]="testfile.asc";
```

```
main()
{
```

```

int handle;

inregs.x.dx = (int)&filename[0];
inregs.h.al = 0;
inregs.h.ah = 0x3d;
intdos(&inregs,&outregs);
if (outregs.x.cflag != 0)
    printf("TESTFILE.ASC not opened.");
else {
    printf("TESTFILE.ASC opened.\n");
    handle = outregs.x.ax;
    inregs.x.bx = handle;
    inregs.x.cx = 4;
    inregs.h.ah = 0x46;
    intdos(&inregs,&outregs);
    if (outregs.x.cflag != 0)
        printf("TESTFILE.ASC handle not duplicated.");
    else printf("TESTFILE.ASC handle duplicated as printer.");
    inregs.x.bx = handle;
    inregs.h.ah = 0x3e;
    intdos(&inregs,&outregs);
}
}

```

## DOS 功能调用 47H

取当前目录

**功能调用:**

47H 取当前目录

**DOS 版本:**

2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

该功能调用返回一指定驱动器当前目录的 ASC II Z 字串。驱动器字母（如 A）不在返回字符串的部分，字符串也不以空格作为开头。目录名以 00H 字节作为 ASC II Z 字串结束。

**入口参数:**

调用前，需设置寄存器：

AH = 47H 指出功能调用号。

DL = 当前目录所在驱动器号 (0 = 缺省驱动器, 1 = A 驱动器等)。

**DS: SI =** 指向64字节的数据区域的指针。在该数据区域中DOS返回的名为当前目录的ASCII字串名。

**出口参数:**

返回后，设置寄存器为：

**DS: SI** 所指的数据区：该数据区存放DL所指定的驱动器中当前目录的ASCII字串。若当前目录是根目录，则该数据区将是个空串。

**错误信息:**

若出现错误，则置进位标志，且AX返回如下出错代码：

OFH DL中放了无效的驱动器号。

**参见:**

59H—取扩展错误代码。

**程序实例:**

下面三个实例均使用功能调用47H，来恢复当前目录名，程序在屏幕上显示该名字。

**Assembly Language Usage Example:**

```
;  
;GET CURRENT DIRECTORY (47H)  
;  
code    segment public  
assume cs:code,ds:code  
        org     100h  
start: jmp      begin  
msg1   db      'Drive number is invalid.',0dh,0ah,'$'  
msg2   db      'Current directory is ','$'  
buffer db      64 dup (?)           ;place to hold directory name  
begin: mov      ax,cs             ;set up ds  
        mov      ds,ax             ;to same as cs  
        mov      dl,0              ;use default drive  
        mov      si,offset buffer ;put dir name in buffer  
        mov      ah,47H            ;get current dir funct request  
        int      21h              ;call DOS  
        jc       error             ;jump if error encountered  
        mov      bx,offset buffer ;set up loop index  
loop:  cmp      bx,ptr [bx],0      ;is this byte a zero?  
        jc       found             ;found a zero  
        inc      bx                ;next byte  
        jmp      short loop  
found: mov      dx,offset msg2    ;address of current dir msg
```

```

    mov     ah,09h      ;display string funct request
    int     21h         ;call DOS
    mov     byte ptr [bx],$/';add '$' at end of string
    mov     dx,offset buffer ;address of directory name
    mov     ah,09h      ;display string function request
    int     21h         ;call DOS
    jmp     done        ;wrap it up
error: mov    dx,offset msg1 ;address of error message
       mov    ah,09h      ;display string function request
       int    21h         ;call DOS
done:  mov    al,0        ;normal exit
       mov    ah,4ch      ;terminate process funct request
       int    21h         ;call DOS
code   ends           ;end of code segment
end    start          ;start is the entry point

```

**Pascal Usage Example:**

```
{ Get Current Directory ($47)}
```

```
PROGRAM tcb; current directory;
```

**CONST**

```
dos get dir = $47;
default = 0;
carry flag = 1;
```

**TYPE**

```
regpack = RECORD
  CASE integer OF
    1: (ax,bx,cx,dx,bp,si,di,ds,es,flgs:integer);
    2: (al,ah,bl,bh,cl,ch,dl,dh:byte);
  END;
```

**VAR**

```
regs:regpack;
buffer:string [64];
len:integer;
```

**BEGIN**

```
WITH regs DO
```

```
  BEGIN
```

```

buffer[0]:=chr(64);
ah:=dos get dir;
si:=ofs(buffer[1]);
dl:=default;
msdos(regs);
; IF ((flags AND carry flag)=1) THEN
    WRITELN('Drive number invalid')
ELSE
BEGIN
    len:=pos(chr(0),buffer);
    buffer[0]:=chr(len-1);
    writeln('current idirectory is ',buffer);
END;
END;
END.

```

### C Usage Example:

```

/*
 * Get Current Directory (0x47)
 */
#include <dos.h>
union REGS inregs,outregs;
main()
{
char buffer[64];
inregs.h.ah = 0x47;
inregs.x.si = (int)&buffer[0];
inregs.h.dl = 0;
intdos(&inregs,&outregs);
if (outregs.x.cflag != 0)
    printf("Drive number is invalid \n");
else printf("current directory is %s\n",buffer);

```

## DOS 功能调用 48H

### 分配内存

功能调用：

48H 分配内存

DOS 版本：

2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

该功能调用动态地把内存分配给需要的程序。

当一个程序在运行时, 它起初占用系统中可用的全部内存。然而若程序本身再起动其它独立程序, 则它必须释放某些内存, 该功能调用对回收释放的内存是有用的。

**入口参数:**

调用前, 需设置寄存器:

AH = 48H 指出功能调用号。

BX = 程序需要的 16 字节的内存段落数。

**出口参数:**

返回后, 设置寄存器为:

AX: 0 指向被分配的内存块起点。

BX 仅当没有足够大的内存可供分配时才设置该寄存器。在那种情况下, 该寄存器给出内存最大可用块中 16 字节的段数。

**错误信息:**

若出现错误, 则置进位标志, 且 AX 返回如下出错代码:

07H 存贮控制块被破坏, 因为程序改变了未分配给它的内存。

08H 没有足够的内存块满足要求。

**参见:**

49H--释放已分配的内存。

4AH--修改分本的内存块 (SETBLOCK)。

58H--取或置内存分配对策。

59H--取出扩展错误代码。

**程序实例:**

下面汇编语方实例使用功能调用 48H 从 DOS 分配 20 段内存。程序首先调用功能调用 4AH 改变它所用的内存大小。然后分配内存, 最后, 释放所分配的内存。

该功能调用没有 PASCAL 和 C 语言 R 实例是因为分配内存是一种低级操作, 几乎总是用汇编语言实现。

**Assembly Language Usage Example:**

```
; ; ALLOCATE MEMORY (48H)
;
code segment public
assume cs:code,ds:code
    org      100h
start: jmp      begin
block  dw      ?
msg1   db      'set block completed ok.',0dh,0ah,'$'
;
```

```

msg2 db 'Allocate memory completed ok.',0dh,0ah,'$'
msg3 db 'Free memory completed ok !',0dh,0ah,'$'
msg4 db 'Set block failed.',0dh,0ah,'$'
msg5 db 'Allocate memory failed.',0dh,0ah,'$'
msg6 db 'Free memory failed.',0dhh,0dh,'$'

begin: mov ax,cs           ;set up ds
       mov ds,ax          ;to same as cs
       mov es,ax          ;ditto for es
       mov bx,offset pgmend ;address of last byte of program
       mov cl,4            ;shift count
       shr bx,cl          ;divide by 16
       inc bx             ;round up
       mov ah,4ah          ;set block function request
       int 21h             ;call DOS
       jc seterr           ;jump if error encountered
       mov dx,offset msg1  ;address of set message
       mov ah,0gh          ;display string function request
       int 21h             ;call DOS
       mov bx,20            ;allocate 20 paragraphs
       mov ah,48h          ;allocate memory funct request
       int 21h             ;call DOS
       jc allerr           ;jump if error encountered
       mov block,ax         ;seg addr of alloc mem block
       mov dx,offset msg2  ;address of allocate message
       mov ah,09h          ;display string function request
       int 21h             ;call DOS
       jc freerr           ;free the whole allocated block
       mov dx,offset msg3  ;free memory function request
       mov ah,09h          ;display string function request
       int 21h             ;call DOS
       jmp done             ;wrap it up

seterr: mov dx,offset msg4 ;addr of set block error message
       jmp disp             ;display error message
allerr: mov dx,offset msg5 ;address of allocate error msg
       jmp disp             ;display error message
freerr: mov dx,offset msg6 ;address of free error message
disp:   mov ah,09h          ;display string function request
       int 21h             ;call DOS
done:  mov ah,4ch          ;terminate process funct request

```

```

int      21h          ;call DOS
pgmend db   ?          ;last byte of code in the pgm
code    ends           ;end of code segment
end     start          ;start is the entry point

```

## DOS 功能调用 49H

释放已分配的内存

**功能调用:**

49H 释放已分配的内存

**DOS 版本:**

2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

该功能调用可释放功能调用 48H 分配的内存，将这些内存交予 DOS 管理系统。

**入口参数:**

调用前，需设置寄存器：

AH = 49H 指出功能调用号。

ES = 要交还功能调用 48H 所分配的内存块的基本段地址。

**出口参数:**

无。

**错误信息:**

若出现错误，则置进位标志，且 AX 返回如下出错代码：

07H 存贮控制块被破坏，因为程序改变了未分配给它的内存。

09H 由 48H 功能调用规定的内存块无一有效。

**参见:**

48H—分配内存。

4AH—修改已分配的内存块 (SETBLOCK)。

59H—取扩展错误代码。

**程序实例:**

下面汇编语言实例使用功能调用 19H 释放 20 段内存，程序首先调用功能调用 4AH 改变它所用的内存大小，然后分配内存，最后释放所分配的内存。

该功能调用没有 PASCAL 和 C 语言例子是因为释放内存是一个低级操作，几乎总是用汇编语言实现。

**Assembly Language Usage Example:**

```

;
;FREE ALLOCATED MEMORY (49H)
;
code  segment public

```

```

assume cs:code,ds:code
org 100h
start: jmp bgin
block dw ?
msg1 db 'set block completed ok.',0dh,0ah,'$'
msg2 db 'Allocate memory completed ok.',0dh,0ah,'$'
msg3 db 'Free memory completed ok.',0dh,0ah,'$'
msg4 db 'Set block failed.',0dh,0ah,'$'
msg5 db 'Allocate memory failed.',0dh,0ah,'$'
msg6 db 'Free memory failed.',0dh,0ah,'$'
begin: mov ax,cs           ;set up ds
       mov ds,ax          ;to same as cs
       mov es,ax          ;ditto for es
       mov bx,offset pgmend ;address of last byte of program
       mov cl,4            ;shift count
       shr bx,cl          ;divide by 16
       inc bx             ;round up
       mov ah,4ah          ;set block function request
       int 21h             ;call DOS
       jc seterr          ;jump if error encountered
       mov dx,offset msg1 ;address of set message
       mov ah,09h          ;display string function request
       int 21h             ;call DOS
       mov bx,20            ;allocate 20 paragraphs
       mov ah,48h          ;allocate memory funct request
       int 21h             ;call DOS
       jc allerr          ;jump if error encountered
       mov block,ax         ;seg addr of allocated mem block
       mov dx,offset msg2 ;address of allocate message
       mov ah,09h          ;display string function reguset
       int 21h             ;call DOS
       mov es,block         ;free the whole allocated block
       mov ah,49h          ;free memoify function request
       int 21h             ;call DOS
       jc freerr          ;jump if error encountered
       mov dx,offset msg3 ;address of free message
       mov ah,09h          ;display string function reguset
       int 21h             ;call DOS
       jmp done            ;wrap it up

```

```

seterr:mov    dx,offset msg4    ;addr of set block error message
    jmp      disp           ;display error message
aller:mov    dx,offset msg5    ;addr of allocate error message
    jmp      disp           ;display error message
freerr:mov    dx,offset msg6    ;address of free error message
disp:   mov     ah,09h          ;display string function request
        int     21h            ;call DOS
done:  mov     ah,4ch          ;terminate process funct request
        int     21h            ;call DOS
pgmend db     ?              ;last byte of code segment
code   ncds          ;end of code segment
end    start          ;start is the entry point

```

## DOS 功能调用 4AH

### SETBLOCK--修改分配的内存块

**功能调用:**

4AH--修改分配的内存块

**DOS 版本:**

2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

该功能调用修改原先用功能调用 48H 所分配的内存块的大小。可用该功能调用增加或减少内存块的大小。该功能调用可以增加或撤消块尾的内存。

**入口参数:**

调用前，需设置寄存器：

AH = 4AH 指出功能调用号。

ES = 功能调用 48H 分配的内存块的段地址。

BX = 已分配内存块所应包括的 16 字节的段落数。

**出口参数:**

返回后，设置寄存器为：

BL 仅当块不能增加到所希望的大小时，才设置该寄存器。在那种情况下，该寄存器内容指出最大可用内存中 16 字节的段落数。

**错误信息:**

若出现错误，则置进位标志，且 AX 反回如下出错代码：

07H 存贮控制块被破坏，因为程序发言将未分配给它的内存。

08H 没有足够的内存块满足要求。

09H 无效的内存块地址不能调功能调用 4AH 分配指定内存块。

**参出:**

48H--分配内存

49H—释放已分配的内存

58H—取或置内存分配对策

59H—取扩展错误代码

**程序实例：**

下面汇编语言实例使用功能调用 4AH 把程序分配的内存大小改变到程序和数据所需的内存大小。

该功能调用没有 PASCAL 和 C 语言例子是因为修改分配的内存块是一个低级操作，几乎总是用汇编语言实现。

**Assembly Language Usage Example:**

```
;  
;SET BLOCK (4AH)  
;  
code segment public  
assume cs:code,ds:code  
    org 100h  
start: jmp begin  
block dw ?  
msg1 db 'Set block completed ok.',0dh,0ah, '$'  
msg2 db 'Allocate memory completed ok.',0dh,0ah, '$'  
msg3 db 'Free memory completed ok.',0dh,0ah, '$'  
msg4 db 'Set block failed.',0dh,0ah, '$'  
msg5 db 'Allocate memory failed.',0dh,0ah, '$'  
msg6 db 'Free memory failed.',0dh,0ah, '$'  
begin: mov ax,cs           ;set up ds  
       mov ds,ax           ;to same as cs  
       mov es,ax           ;ditto for es  
       mov bx,offset pgmend ;address of last byte of program  
       mov cl,4             ;shift count  
       shr bx,cl           ;divide by 16  
       inc bx              ;round up  
       mov ah,4ah            ;set block function request  
       int 21h              ;call DOS  
       jc seterr            ;jump if error encountered  
       mov dx,offset msg1   ;address of set message  
       mov ah,09h             ;display string function request  
       int 21h              ;call DOS  
       mov bx,20              ;allocate 20 paragraphs  
       mov ah,48h             ;allocate memory funct request
```

```

int    21h          ;call DOS
jc     allerr       ;jump if error encountered
mov    block,ax      ;seg addr of allocated mem block
mov    dx,offset msg2 ;address of allocate message
mov    ah,09h        ;display string function request
int    21h          ;call DOS
mov    es,block      ;free the whole allocated block
mov    ah,49h        ;free memory function request
int    21h          ;call DOS
jc     freerr       ;jump if error encountered
mov    dx,offset msg3 ;address of free message
mov    ah,09h        ;display string function request
int    21h          ;call DOS
jmp    done         ;wrap it up

seterr: mov   dx,offset msg4 ;addr of set block error message
        jmp   disp      ;display error message
allerr: mov   dx,offset msg5 ;addr of allocate error message
        jmp   disp      ;display error message
freerr: mov   dx,offset msg6 ;address of free error message
disp:   mov   ah,09h        ;display string function request
        int   21h          ;call DOS
done:   mov   ah,4ch        ;terminate process funct request
        int   21h          ;call DOS
pgmend db   ?           ;last byte of code in program
code   ends          ;end of code segment
end    start         ;start is the entry point

```

## DOS 功能调用 4BH

**EXEC—装入或执行程序**

**功能调用:**

**EXEC—装入或执行程序**

**DOS 版本:**

2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

该功能调用允许一个程序将另一个程序装入内存并可选择地开始执行那个程序。

当使用该功能调用装入一个子程序时，除了些继承位为 1 的以外（想更多了解有关继

承位的信息,请参见打开文件功能调用 3DH),程序所有打开的文件对子程序堵阳可用的。这意味着借助于功能调用 46H(强迫复制文件句柄)程序能够控制子程序使用的标准输入、标准输出、辅助设备和打印机的含义。通过相应地调整这些设备,程序能够把标准输入和输出设备置成两个文件,且请求一个子程序处理标准输入的数据(可能它分类)并写结果到标准输出。

用该功能调用装入一个程序也留给新程序送去一个环境。该环境是一字符串构成的内存块(小于 32K 字节),包括与程序有关的配置参数。每个字符串都是 ASCII Z 字串必须另外以 00H 字节结束。典型的每段 ASCII Z 字串包括环境字串的形式是:

参数 = 数值

如 VERIFY = ON, 程序就能够把它自己环境的一个副本或程序所建立的新环境传递给已装入的程序。

入口参数:

调用前,需设置寄存器:

AH = 4BH 指出功能调用号。

AL = 指出程序是否仅装入内存,或程序装入且招待。其数值是:

0 装入且执行程序。在这种情况下,DOS应为文件建立程序段前缀(PSP, EXEC 功能调用之后,要给指令置结束地址和 Ctrl-Break 地址。这意味着当程序退出时,或当操作员键入 Ctrl-Break 时,原先的程序将恰好在它请求子程序的点处重新控制返回时,所有寄存器包括栈寄存器内容可能都变了。继续执行前,原程序必须恢复任何所需的寄存器。PSP 的格式见第 20 页。

3 装入但不执行程序。在这种情况下,DOS不为文件建立程序段前缀(PSP)。

该选择对装入主程序覆盖和装入数据是有用的。

DS: DX 指向有ASCII Z 字符串的存贮区,包括要装入驱动器、路径和文件名。

ES: BX 指向参数区,参数区的内容变化取决于装入的程序是否执行。

若执行装入程序(AL=0),则参数区的格式如下:

偏移	内容
----	----

00H-01H 单字,给被装入程序传送的环境字符串的段地址。若该字为 0,则传送当前程序的环境字符串。否则,环境字符串以该段的 0 偏移为起始地址(即在段边界)。

02H-05H 双字,指向被装入程序命令行的指针,在该地址,至多可设置 128 个字节的信息,此时被装入程序仿佛是处理在命令行上打入信息。这 128 个字节被放在装入程序的 PSP 偏移 80H 后位为首址的地方。命令行文本应包括回车(0DH),以便被装入程序能够判断命令行是否结束。PSP 的格式见第 20 页。

06H-09H 双字,指向传送给装入程序的 2 个默认 FCB 中的第一个。从 DOS1.X 开始,这些 FCB 就作为保留机制,使一个程序能够识别装入程序应该进行处理的 2 个文件。第一个 FCB 指针在装入程序 PSP+5CH 处。PSP 的格式见第 20 页。对于 DOS2.X 以上程序继承其父进程的文件句柄。因此,父进程可重定向标准输入和输出以建立装入程序应该进行处理的文件。

**0AH** 双字，指向装入程序的第二个默认 FCB.

**11H** 含有不连续信息的 EXE 文件。

若不执行装入程序 (AL = 3)，则参数区的格式如下：

偏移              内容

**00H-01H** 单字，表示程序应被装入的段地址。

**02H-03H** 单字，表示用于代码映象的重分配因子。这仅于.EXE 文件且实质上是从代码段起点计算的映象偏移。装入程序使用该信息可修改代码某些地址（称重分配项），以便映象代码能正常执行。

对.COM 文件该数值是不必要的，因为.COM 文件不必重分

配。

**出口参数：**

无。

**其它要求：**

当控制返回到原程序时，所有寄存器和堆栈可能都变了。因此，在请求功能调用前必须保护 SS、SP 及任何其它必要的寄存器内容。

当原程序接受控制时，所有可用的内存均被分配给它。在请求该功能调用前，应保证给新程序有足够的可用的内存。若必要的话，使用功能调用 49H 或 4AH 释放内存。

该功能调用使用 DOS 装入程序来装入程序。对于 DOS2.X，若程序已分配很内存以至它覆盖了 COM-MAND.COM 文件，这时在继续处理前该功能调用将重新加载装入程序。若装入程序没有足够的内存可用，就会收到错误信息。在 DOS3.X 中，装入程序驻留在 IBMDOS.COM 文件内而在 COMMAND.COM 文件之中，因此将不会被覆盖。

**错误信息：**

若出现错误，则进位标志被设置，且 AX 返回如下出错代码：

**01H** AL 中设置了不是 0 或 3 的无效的功能码。

**02H** DOS 没有找到由 DS: DX 中规定了名字的文件。

**03H** 至少有一目录路径名未找到。

**05H** 不能访问 DS: DX 中规定了名字的文件。

**08H** 装入应用程序或加载程序使用的内存不够。

**0AH** 至少有一个无效的环境字串。

**参见：**

58H—取或置内存分配对策。

59H—取扩展错误代码。

**程序实例：**

下面汇编语言实例使用功能调用 4BH 来装入并执行名为 DOS47.COM 的程序。

该功能调用没有 PASCAL 和 C 语言实例是因为包含的操作是低级操作，几乎总是用汇编语言来实现。

#### Assembly Language Usage Example:

;

```

; EXEC (4BH)
;
code    segment public
assume cs:code,ds:code
        org      100h
start: jmp      begin
file   db       dos47.com',0
parms   db       14 dup(?)
cmd    db       0dh
msg1   db       Set block failed.',0dh,0ah,  $
msg2   db       Exec failed.',0dh,0ah,  $
begin: mov      ax,cs           ;set up ds
        mov      ds,ax           ;to same as cs
        mov      es,ax           ;ditto for es
        mov      bx,offset pgmend ;address of last byte of program
        mov      cl,4             ;shift count
        shr      bx,cl           ;divide by 16
        inc      bx             ;round up
        mov      ah,4ah           ;set block function request
        int      21h              ;call DOS
        jc      seterr            ;jump if error encountered
        mov      dx,offset file   ;path for the file to be loaded
        mov      bx ,offset parms ;address of parameter block
        mov      word ptr parms[00h],0 ;pass existing environment
        mov      word ptr parms[02h],offset cmd  ;ofst of comd line
        mov      word ptr parms[04h],cs  ;segment of command line
        mov      word ptr parms[06h],5ch ;offset of first FCB
        mov      word ptr parms[08h],es  ;segment of first FCB
        mov      word ptr parms[0ah],6ch ;offset of second FCB
        mov      word ptr parms[0ch],es  ;segment of second FCB
        mov      al,0               ;load and esecute the file
        mov      ah,4bh             ;exec function request
        int      21h              ;call DOS
        jnc      done              ;no errors
        mov      dx,offset msg2    ;address of error message
        mov      ah,09h             ;display string function request
        int      21h              ;call dos
        jmp      done              ;wrap it up
done:  mov      dx,offset msg1    ;address of error message
seterr: mov

```

```

        mov     ah,09h      ;display string function request
        int     21h          ;call dos
done:   mov     ah,4ch      ;terminate process funct request
        int     21h          ;call DOS
pgmend db    ?           ;last byte of code in program
code    ends            ;end of code segment
        end     start       ;start is the entry point

```

## DOS 功能调用 4CH

EXIT—终止进程

功能调用:

4CH EXIT—终止进程

DOS 版本:

2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明:

该功能调用终止当前程序并且把控制转给调用它的程序。由被终止程序打开的全部文件都被关闭。在终止时，程序用的内存还给 DOS 并可重新分配给其它程序。

该功能调用允许被终止程序传送一返回代码给调用它的程序。该返回码可指出终止的状态。这取决于程序是如何设计。不管是否有错或什么错，在终止程序退出后重新获得控制的程序能检查该代码以确定终止的状态。

若被终止程序是用 EXEC 功能调用 (4BH) 调入的，调用的程序可请求功能调用 4DH (取子进程的返回代码) 来恢复返回代码。若被终止程序是 DOS 命令调入的，则返回代码可用 DOS 的 IF 和 ERRORLEVEL 批命令确定。

入口参数:

调用前，需设置寄存器:

AH=4CH 指出功能调用号。

AL= 终止代码，向其它程序指出该程序终止的条件，该代码没有标准的数值可由程序员自定。

出口参数:

无。

其它要求:

若使用功能调用 5CH 来封锁文件或文件的一部分，在终止进程前，必须解锁那些区域，否则会引起不可预测的错误。

参见:

31H—终止进程保留常驻 (KEEP)

4DH--取子进程的返回代码 (WAIT)

59H—取扩展错误代码

程序实例:

下面汇编语言实例使用功能调用 4CH 来终止它自己。  
没有 PASCAL 和 C 语言实例是因为那些编译程序自动产生 DOS 终止功能调用成为它们退出代码的一部分。

**Assembly Language Usage Example:**

```
; ; TERMINATE PROCESS (4CH)
;
code    segment public
assume cs:code,ds:code
        org      100h
start: jmp     begin
msg     db      'Terminating via DOS function request 4CH', '$'
begin: mov     ax,cs           ;set up ds
        mov     ds,ax
        mov     dx,offset msg   ;set up to print
        mov     ah,09h          ;display string function request
        int     21h             ;call DOS
        mov     al,0             ;return code
        mov     ah,4ch            ;terminate process funct request
        int     21h             ;call DOS
code    ends
end     start             ;start is the entry point
```

**DOS 功能调用 4DH**

**WAIT—取子进程的返回代码**

**功能调用：**

4DH WAIT—获取子进程的返回代码

**DOS 版本：**

2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明：**

该功能调用将收回由另一个进程结束时利用功能调用 31H (终止进程保留常驻) 和 4CH (终止进程) 所发送的返回码。此外，该功能调用也可收回 DOS 提供的，指出进程如何终止的信息。

**入口参数：**

调用前，需设置寄存器：

AH=4DH 指出功能调用号。

**出口参数：**

返回后，设置寄存器为：

AL 包含终止进程提供的终止代码。该代码由功能调用 3H 和或 4CH 给出。

AH 指出终止进程是如何结束的，其数值如下：

- 00 程序正常终止（经调用功能调用 4CH）。
- 01 用 Ctrl-Break 键结束程序。
- 02 严重设备错导致的结束程序。
- 03 调用功能 31H 正常结束程序，但保留常驻。

参见：

31H—终止进程保留常驻（KEEP）。

4CH—终止进程（EXIT）。

59H—取扩展错误代码。

程序实例：

下面汇编语言实例使用功能调用 4DH 执行名为 DOS47.COM 的程序文件。该例显示程序文件是否正常终止。

访问能调用没有 PASCAL 和 C 语言实例是因为包含的操作是低级操作，几乎总是用汇编语言来实现。

#### Assembly Language Usage Example:

```
; ; GET EXIT CODE (4DH)
;
code segment public
assume cs:code,ds:code
    org 100h
start: jmp begin
file db dos47.com',0
parms db 14 dup (?)
cmd db 0dh
msg1 db 'Exec failed.',0dh,0ah, '$'
msg2 db '0dh,0ah, Normal termination.', '$'
msg3 db '0dh,0ah, Terminated abnormally.', '$'
msg4 db 'Set block failed.',0dh,0ah, '$'
begin: mov ax,cs           ;set up ds
       mov ds,ax          ;to same as cs
       mov es,ax          ;ditto for es
       mov bx,offset pgmend ;address of last byte of program
       mov cl,4            ;shift count
       shr bx,cl          ;divide by 16
       inc bx             ;round up
```

```

        mov     ah,4ah      ;set block function request
        int     21h         ;call DOS
        jc      seterr       ;jump if error encountered
        mov     dx,offset file ;path for the file to be loaded
        mov     bx,offset parms ;address of parameter block
        mov     word ptr parms[00h],0 ;pass existing environment
        mov     word ptr parms[02h],offset cmd ;command line
        mov     word ptr parms[04h],cs ;segment of command line
        mov     word ptr parms[06h],5ch ;offset of first FCB
        mov     word ptr parms[08h],es ;segment of first FCR
        mov     word ptr parms[0ah],6ch ;offset of second FCB
        mov     word ptr parms[0ch],es ;segment of second FCB
        mov     al,0          ;load and execute the file
        mov     ah,4bh        ;exec function request
        int     21h         ;call DOS
        jnc     exit         ;no errors
        mov     dx,offset msg1 ;address of error message
        mov     ah,09h        ;display string function request
        int     21h         ;call dos
exit:   mov     ah,4dh        ;get exit code function request
        int     21h         ;call dos

        cmp     ax,0          ;normal termination?
        jne     abnorm        ;jump if abnormal
        mov     dx,offset msg2 ;address of normal message
        mov     ah,09h        ;display string function request
        int     21h         ;call dos
        jmp     term          ;terminate
abnorm: mov     dx,offset msg3 ;address of abnormal message
        mov     ah,09h        ;display string function request
        int     21h         ;call dos
        jmp     term          ;terminate
seterr: mov     dx,offset msg4 ;address of error message
        mov     ah,09h        ;display string function request
        int     21h         ;call dos
term:   mov     ah,4ch        ;terminate process funct request
        int     21h         ;call DOS
pgmend db      ?           ;last byte of code in program
code    ends          ;end of code segment

```

end start ;start is the entry point

## DOS 功能调用 4EH

FIND FIRST--搜索第一个匹配文件

功能调用:

4EH FIND FIRST--搜索第一个匹配文件

DOS 版本:

2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明:

该功能调用查找与指定文件名相匹配的第一个文件的目录。该文件名包含通配符“?”和“\*”，找出与该名相匹配的几个文件。

该功能调用是 DOS2.X 功能调用 11H 的变型。这两个功能调用执行相同的操作，但功能调用 4EH 接受完整路径名作为输入，可查找任何目录（能调用 11H 仅能查找当前目录），与功能调用 11H 不同，功能调用 4EH 返回足够的信息为文件形成一个完整的目录项；但作为目录项中一个字段的文件起始簇并不返回。

当指定 CX 中的属性字节时，用此功能调用则可依据它们的属性来查找文件，如下：

- 若置属性字节为 0，则功能调用仅查找一般文件，不查找卷标号、子目录、隐含文件或系统文件。

- 若置属性字节为隐含文件、系统文件和 / 或子目录，则该功能调用通过查找一般文件和所选的特定文件即可得到匹配文件。全部三位都设置时，此功能调用可以找一般文件、隐含文件、系统文件和目录。

- 若置属性字节为卷 标号，该功能调用仅查找卷标号，不查找一般文件。

- 归档位和只读位对搜索没有影响。

当该功能调用找到第一个匹配文件时，它就提供有关该文件在以 DTA 为起始地址的内存中所存的信息。可使用该信息去打开文件，当搜索指定匹配的下一个文件时，功能调用 4FH 也使用 DTA 中起始的信息。

入口参数:

调用前，需设置:

AH=4EH 指出功能调用号。

DS: DX = 指向 ASCII 字符串的指针，字符串包含要查找的文件的驱动器、目录、路径和文件名。ASCII 字符串必须以 00H 为结束符，文件名部分可以使用文件通配字符：“?”和“\*”。

CX: 低位字节为查找文件使用的属性，如下。

位
7 6 5 4 3 2 1 0

为了在查找过程中使用该属性，相应的位应置为 1。

位 说明

0 只读文件

- 1 隐含文件
- 2 系统文件
- 3 卷标号
- 4 子目录
- 5 档案(归档)
- 6-7 未用

#### 出口参数:

返回后, 设置为:

**DTA** 盘传送地址应指描述第一个匹配文件信息中 43 个字节区域的首址。该信息的格式如下:

偏移

00H 21 个字节, 为系统保留位后面的 FINDEXT (04FH) 功能调用将始  
用这 些位

15H

16H 1 个字节, 文件属性

17H 2 个字节, 文件的时间标记

18H

19H 2 个字节, 文件的日期标记

1AH

1BH 2 个字节, 文件尺寸(低位字)

1CH

1DH 2 个字节, 文件尺寸(高位字)

1EH

1FH 13 个字节, 文件的名字和扩展名

2BH

该信息的各个字段如下:

系统保留位: 前 21 字节为 DOS 保留用。这里存贮的信息允许请求功能调用 4FH 来寻找下一个匹配文件。

文件属性: 该字节是文件属性字节。当请求该功能调用时, 其编码方法与 ^X 的编码方法相同。

文件时间标记: 这两字节指出文件上次建立或修改的时间, 记录和时间, 如下:

hhhhh mmmmm sssss

0--4 位是被 2 除的秒, 5--10 位是分钟, 11--15 位是小时。

文件的日期标记: 这两字节数值指出文件上次建立或修改日期, 记录的日期如下:

yyyyy mmmmm dddd

0--4 位为日期, 5--8 位为月份 (0=一月, 1=二月等)。

9--15 位为年, (其值为年份减去年 980)。

文件尺寸: 这四个字节的数值指出文件尺寸(以字节为单位)。

文件名和扩展名: 这十三个字节包含标识第一个匹配文件的 ASCII 字符

串，该文件名不包含起始的空格和分格符。若文件名包含一个扩展名，则在名字和扩展名之间有一圆点，ASC II Z 字符串以 00H 结束符。

**其它要求：**

若使用功能调用 4FH 查找其它匹配文件，不要改变以 DTA 为首址时返回的信息。DOS 可使用该信息继续查找文件。

**错误信息：**

若出现错误，则置进位标志，且 AX 返回如下出错代码：

02H 文件未找到。

03H 至少一个指定的目录路径名未找到。

12H 目录中无匹配文件。

**程序实例：**

下面三个实例均使用功能调用 4EH，在当前目录中查找文件名为 TESTFILE.ASC 的文件。

**Assembly Language Usage Example:**

```
; ; FIND FIRST MATCHING FILE (4EH)
;
code    segment public
assume cs:code,ds:code
org      100h
start: jmp begin
file    db     'testfile.asc',0
msg1   db     'searching for testfile.asc.',0dh,0ah,'$'
msg2   db     'testfile.asc NOT FOUND!',0dh,0ah,'$'
msg3   db     'testfile.asc FOUND!,0DH,0AH,'$
begin:mov ax,cs           ;set up ds
      mov ds,ax          ;to same as cs
      mov dx,offset msg1 ;address of searching message
      mov ah,09h          ;display string function request
      int 21h             ;call DOS
      mov dx,offset file ;address of file name
      mov cx,0             ;attributes
      mov ah,4eh           ;find first matching file funct
      int 21h             ;call DOS
      jnc found           ;jump if file found
      mov dx,offset msg2 ;address of not found message
      mov ah,09h           ;display string function request
      int 21h             ;call DOS
```

```

jmp      done          ;wrap it up
found:mov  dx,offset msg3 ;address of found message
        mov   ah,09h       ;display string function request
        int   21h          ;call DOS
done:  mov   ah,4ch       ;terminate process funct request
        int   21h          ;call DOS
code   ends          ;end of code segment
end    start         ;start is the entry point

```

**Pascal Usage Example:**

{ Find First Matching File (\$4e) }

PROGRAM find first;

CONST

```

dos find first = $4e;
attributes = 0 ;
carry flag = 1 ;

```

TYPE

```

regpack = RECORD
  CASE integer OF
    1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
    2 : (al,ah,bl,bh,cl,ch,dl,dh:byte);
  END ;

```

VAR

```

regs : regpack ;
filename : string [50] ;

```

BEGIN

WITH regs DO

BEGIN

```

filename := testfile.asc' + chr(0) ;
writeln( Searching for testfile.asc.' );
ah := dos find first ;
ds := seg(filename[1]);
dx := ofs(filename[1];
cx := attributes ;
msdos(regs);

```

```

        IF((flags AND carry flag) = 1) THEN
            writeln( testfile.asc NOT FOUND!)
        ELSE
            writeln( testfile.asc FOUND!);
        END ;
    END .

```

**C Usage Example:**

```

/*
 * Find First Matching File (0x4e)
 */

#include <dos.h>

union REGS inregs,outregs;
char filename{} = "testfile.asc";

main()
{
    printf("Searching for testfile.asc.\n");
    inregs.h.ah = 0x4e;
    inregs.x.dx = (int) &filename[0];
    inregs.x.cx = 0;
    intdos(&inregs,&outregs);
    if(outregx.x.cflag != 0)
        printf("testile.asc NOT FOUND!\n");
    else
        printf("testfile.asc FOUND!\n");
}

```

**DOS 功能调用 4FH**

**FIND NEXT**--搜索下一个匹配文件

功能调用:

4FH FIND NEXT--搜索下一个匹配文件

DOS 版本:

2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明:

本功能调用是查找下一个目录项，其名称应与前面 FIND FIRST (4EH) 或 FIND NEXT (\$FH) 功能调用所指定的名称匹配。其匹配准则与功能调用 4EH 所

列内容相同。

当此功能调用找到下一个匹配文件时，更新 DTA 起始处描述该文件的信息，用此信息作为输入，可继续引用功能调用 4FH，直到把所有匹配文件都找到为止。

**入口参数：**

调用前，需设置：

AH = 4FH 指出功能调用号。

DTA 磁盘传送地址必须置成由前面 FIND FIRST (4EH) 或 FIND NEXT (4FH) 功能调用应返回的信息起始地址。

**出口参数：**

返回时，设置为：

DTA DTA 盘传送地址应标记有关找到匹配文件所用 43 个字节的信息的首址。信息格式与功能调用 FIND FIRST (4EH) 的说明相同。

**其他要求：**

如果希望使用功能调用 4FH 来查找其他匹配文件，不应改变 DTA 处起始的返回信息，DOS 可使用此信息继续查找所需的匹配文件。

**错误信息：**

如果出现差错，则设置进位标志，且将寄存器 AX 置成如下：AX = 12H 不存在匹配文件

**参见：**

4EH--搜索 FIND FIRST 第一个匹配文件。

59H--取扩充错误代码。

**程序实例：**

下述三个实例是用功能调用 4FH 在当前目录中查找与名“TESTFILE???”相匹配的文件。由于这个文件含有通配字符，因此程序将连续调用 4FH，直到把所有匹配文件找到为止。

**Assembly Language Usage Example:**

```
; ; FIND NEXT MATCHING FILE (4FH)
;
code segment public
assume cs:code,ds:code
org 100h
start: jmp begin
file db 'testfile.???.0
msg1 db 'searching for files matching testfile.???.',
      db 0dh,0ah,'$'
msg2 db 'match found!',0dh,0ah,'$'
begin: mov ax,cs           ;set up ds
```

```

        mov     ds,ax          ;to same as cs
        mov     dx,offset msg1 ;address of searching message
        mov     ah,09h          ;display string function request
        int     21h             ;call DOS
        mov     dx,offset file ;address of file name
        mov     cs,0              ;attributes
        mov     ah,4eh            ;find first matching file funct
        int     21h             ;call DOS
        jc    done              ;jump if on files found
nest:  mov     dx,offset msg2 ;address of found message
        mov     ah,09h          ;display string function request
        int     21h             ;call DOS
        mov     dx,offset file ;address of file name
        mov     ah,4fh            ;find next matching file funct
        int     21h             ;call DOS
        jnc    next              ;look for more if carry not set
done:  mov     ah,4ch            ;terminate process funct request
        int     21h             ;call DOS
code   ends              ;end of code segment
        end     start             ;start is the entry point

```

**Pascal Usage Example:**  
**{Find Next Matching File(\$ 4f)}**

**PROGRAM** find nest;

**CONST**

```

dos find first = $ 4c;
dos find nest = $ 4f;
attributes = 0;
carry flag = 1;

```

**TYPE**

```

regpack = RECORD
  CASE integer OF
    1:(ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
    2:(al,ah,b1,bh,c1,ch,di,dh:byte);
  END;

```

**VAR**

```
regs:regpack;
```

```

filename:string[50];
BEGIN
  WITH regs DO
    BEGIN
      filename := 'testfile.???' + chr(0);
      wrhteln('searching for files matching testfile.???.');
      ah:=dos find first;
      ds:=seg(filename[1]);
      dx:=ofs(filename[1]);
      cx:=attributes;
      msdos(regs);
      WHILE((flags AND carry tflag<>1)DO
        BEGINB
          writeln('match found!');
          ah:=dos find enxt;
          ds:=seg(filename[1]);
          dx:=ofs(filename[1]);
          msdos(regs);
        END;
      END;
    END;

```

### C Usage Example:

```

/*
 * Find Next Matching File(0x4f)
 */

#include<dos.h>

union REGS inregs,outregs;

charfilename[]="testfile.???" ;
main()
{
  printf("searching for files matching testfile.???.\n");
  inregs.h.ah=0x4e;
  inregs.x.dx=(int)&filename[0];
  inregs.x.cx=0;
  intdos(&inregs,&outregs);
}

```

```
while(outregs.x.cflag == 0) {
    printf("match found!\n");
    inregs.h.ah = 0x4f;
    inregs.x.dx = (int)&filename[0];
    intdos(&inregs,&outregs);
}
}
```

## DOS 功能调用 50H, 51H, 52H, 53H

DOS 内部使用

功能调用:

50H, 51H, 52H, 53H

这些功能调用保留给 DOS 内部使用，用户程序不应调用。

## DOS 功能调用 54H

取校验开关状态

功能调用:

54H 取校验开关状态。

DOS 版本:

2.0, 2.1, 3.0, 3.1, 3.2, 3.3

说明:

本功能调用检查校验开关的当前状态，此开关用来说明向磁盘写入信息时，DOS 是否要进行纠错校验（在磁盘写入信息后，立即读此信息），可用功能调用 2EH 来设置或恢复校验开关。

入口参数:

调用前，需设置:

AH = 54H 指出功能调用

出口参数:

控制返回后设置:

AL 给出校验开关当前的设置情况:

00: 校验开关关闭，表示 DOS 对写入磁盘的信息不做校验。

01: 校验开关打开，每当将信息写入磁盘后，DOS 立即读取此信息做校验。

参见:

2EH—设置校验开关

59H—取扩充错误代码

程序实例:

以下三个程序实例是用功能调用 54H 来取出并在屏幕上显示校验开关当前状态。

**Assembly Language Usage Example:**

```
; ; GET VERIFY STATE(54h)
;
code    segment public
assume cs:code,ds:code
        org      100h
start: jmp     begin
msg1   db      'Verify is on',0dh,0ah,  '$'
msg2   db      'Verify is off',0dh,0ah,  '$'
begin: mov     ax,cx          ;set up ds
        mov     ds,ax          ;to same as cs
        mov     ah,54h          ;get verify state function request
        int     21h             ;call DOS
        cmp     al,0             ;is verify on?
        jc     off              ;jump if it is off
        mov     dx,offset msg1  ;address of on message
        mov     ah,09h          ;display string function request
        int     21h             ;call DOS
        jmp     done             ;wrap it up
off :  mov     dx,offset msg2  ;address of off message
        mov     ah,09h          ;display string function request
        int     21h             ;call DOS
done:  mov     ah,4ch          ;terminate process funct request
        int     21h             ;call DOS
code   ends             ;end of code segment
end    start            ;start is the entry point
```

**Pascal Usage Example:**

```
{ Get Verify State ($54) }
PROGRAM get_verify_state;
```

**CONST**

```
dos_get_verify = $54;
```

**TYPE**

```
regpack = RECORD
```

```

CASE integer OF
  1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
  2 : (al,ah,bl,bh,cl,ch,dl,dh:byte);
END ;

VAR
  regs : regpack;

BEGIN
  WITH regs DO
    BEGIN
      ah := dos get verify ;
      msdos(regs);
      IF (al = 0) THEN
        writeln( ' Verify is off.' )
      ELSE
        writeln( ' Verify is on.' );
    END ;
  END.

```

#### C Usage Example:

```

/*
 * Get Verify State (0x54)
 */

#include <dos.h>

union REGS inregs,outregs;

main()
{
  intrgs.h.ah = 0x54;
  intdos(&inregs,&outregs);
  if((outregs.h.al)==0)printf("Verify is off.\n");
  else printf("Verify is on.\n");
}

```

DOS 或能调用 55H

DOS 内部使用的命令

**功能调用:**

55H 本功能调用是留作 DOS 内部使用的，用户程序不应调用。

**DOS 功能调用 56H**

改文件名

**功能调用:**

56H 改文件名

**DOS 版本:**

2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

本功能调用用于改变一个文件名，是 DOS2.X 中功能调用 17H 的变型。与 DOS2.X 版本的功能调用 17H 不同的是，它可以改变任一子目录中任一文件的名称。

本功能调用类似于 DOS RENAME 程序，但增加另外性能功能更强。它不仅能改变文件名，而且还能把文件从一个子目录移到另一个子目录中去。本功能调用并不移动文件中的数据，而只是移动目录项。目录之间的任何移动，只能在同一个驱动器中的目录进行。

本功能调用只适用于单个文件，因此，在文件名中不得出现通配文件名字符。

用户不能使用本功能调用对已打开的文件、隐含文件、系统文件或子目录重新命名。虽然这些操作可能会成功，但重命名这些文件有可能破坏磁盘中文件的结构。

**入口参数:**

调用前，需设置：

AH = 56H 指出功能调用号。

DS: DX 指向标识被重命名的文件的 ASCII 字符串的指针。该字符串可由驱动器、路径和文件名组成，但不允许出现通配文件名字符“?”和“\*”

ES: DI 指向标识一个新的文件名的 ASCII 字符串的指针。虽然路径和文件名可与原来文件不同，但驱动器名（如果出现）必须相同。

**出口参数:**

无

**其他要求:**

为了在 DOS3.1 或以后版本的网络环境下使用本功能调用，程序必须具有对目录及其文件的创建访问权。

**错误信息:**

如果出现错误，则设置进位标志，且 AX 寄存器为以下数值之一：AX=02H 表示找不到指定文件。

AX=03H 指定的路径名之一不存在。

AX=05H 拒绝访问。第一路径名（由 DS: DX 指定）规定了一个现有文件，或该文件未打开。

AX=11 表示在不同的驱动器上重新命名文件。

**参见:**

**59H 取扩充错误代码。**

**程序实例：**

下面三个程序实例是使用功能调用 56H，将文件 ZZZZZZZZ.ZZZ 重新命名为文件 YYYYYYYY.YYY.

**Assembly Language Usage Example:**

```
; ; RENAME FILE (56H)
;
code segment public
assume cs:code,ds:code
    org 100h
start: jmp begin
old    db      ZZZZZZZZ.ZZZ',0
new    db      YYYYYYYY.YYY',0
msg1   db      ZZZZZZZZ.ZZZ renamed to YYYYYYYY.YYY',0dh,0ah,
$'
msg2   db      File not renamed',0dh,0ah, $'
begin: mov ax,cs          ;set up ds
       mov ds,ax        ;to same as cs
       mov dx,offset old ;address of old file pathname
       mov es,ax          ;set up es same as ds
       mov di,offset new ;address of new file pathname
       mov ah,56h          ;rename file function request
       int 21h            ;call DOS
       jc error           ;error
       mov dx,offset msg1 ;address of renamed message
       mov ah,09h          ;display string function request
       int 21h            ;call DOS
done:  mov ah,4ch          ;terminate process funct request
       int 21h            ;call DOS
code   ends             ;end of code segment
end    start            ;start is the entry point
```

**Pascal Usage Example:**

```
{ Rename File ($56) }
```

```
PROGRAM rename_file;
```

```

CONST
  dos rename file = $56 ;
  carry flag = 1 ;

TYPE
  regpack = RECORD
    CASE integer OF
      1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer) ;
      2 : (al,ah,bl,bh,cl,ch,dl,dh:byte) ;
    END ;
VAR
  regs : regpack ;
  old : string [50] ;
  new : string [50] ;
BEGIN
  WITH regs DO
    REGIN
      old := ZZZZZZZZ.ZZZ + chr(0) ;
      new := YYYYYYYY.YYY + chr(0) ;
      ah := dos rename file ;
      ds := seg(old[1]) ;
      dx := ofs(old[1]) ;
      es := seg(new[1]) ;
      di := ofs(new[1]) ;
      msdos(rcgs),
      IF((flags AND carry flag)=1)THEN
        writeln(' File not renamed')
      ELSE
        writeln(' ZZZZZZZZ.ZZZ renamed to YYYYYYYY.YYY') ;
    END ;
  END.

```

### C Usage Example:

```

/*
 * Rename File (0x56)
 */

```

```
#include <dos.h>
```

```

union REGS inregs,outregs;
char old[] = "ZZZZZZZZ.ZZZ";
char new[] = "YYYYYYYY.YYY";

main()
{
    inregs.x.dx = (int) &old[0];
    inregs.x.di = (int) &new[0];
    inregs.h.ah = 0x56;
    intdos(&inregs,&outregs);
    if (outregs.x.cflag == 0)
        printf("ZZZZZZZZ.ZZZ renamed to YYYYYYYY.YYY\n");
    else
        printf("File not renamed\n");
}

```

## DOS 功能调用 57H

取或 / 置文件的日期和时间

**功能调用:**

57H 取或置文件的日期和时间。

**DOS 版本:**

2.0, 2.1, 3.0, 3.1, 3.2, 3.3

**说明:**

本功能调用允许用户获取或设置文件的日期和时间，该日期和时间被作为文件目录项的一部份，表示文件建立的时间或最后一次修改的日期和时间。

**入口参数:**

调用前，需设置：

AH=57H 指出功能调用号。

AL 表示所要完成的操作类型，如下

00 取日期和时间信息；

01 设置日期和时间信息。

BX： 本寄存器需保存由建立文件(3CH)、建立新文件(5BH)或打开文件(3DH)功能调用所返回的文件句柄。

CX 若设置文件的日期和时间置 AL 为 1，则此寄存器应保存新时间值，寄存器各位设置如下：

hhhh mmssss

0-4 位存秒（秒的值除 2），5-10 位存分，11-15 位存小时。DX 若设置文件的日期和时间 (AL 置 1) 则此寄存器保存新的日期，寄存器各位设置如下：

yyyyyy mmmm dddd

0-4位存日期；5-8位存月份（0等于1月，1等于2月，…）；9-15位存年代码（“当年-1980”所得的值）。

**出口参数:**

控制返回后，设置如下：

CX 若用户在取文件的日期和时间，则此寄存器含时间信息，其格式与人口参数相同。

**DX** 若用户在取文件的日期和时间，则此寄存器含日期信息，其格式与人口参数相同。

其它要求：

在获取或设置文件的日期之前，用户必须通过建立文件（功能调用 3CH）或打开文件（功能调用 3DH）取得该文件和句柄。在所改动的日期和时间生效之前，必须关闭该文件（功能调用 3EH）。

**错误信息：**

如果出错，则设置进位标志，且将 AX 寄存器设为如下数值之一：

$AX=01H$  在 AL 寄存器中送入了 0 或 1 以外的数值。

**AX=06H** 所采用的文件句柄无效。

### 参见：

59H---取扩充错误代码.

### 程序实例：

下述三个程序实例使用功能调用 57H 以取得“TESTFILE.ASC”文件的最后一次写入的日期和时间。程序不显示日期和时间，但用户需要时，可使用此信息。

## Assembly Language Usage Example:

```
; ; GET / SET FILE DATE OR TIME (57H)
;
code    segment public
assume cs:code,ds:code
        org      100h
start: jmp     begin
file    db      testfile.asc',0
handle  dw      ?
date    dw      ?
time    dw      ?
msg1   db      'Got date and time TESTFILE.ASC last written.',0
       db      0dh,0ah,  '$'
msg2   db      'Unable to get date / time for TESTFILE.ASC.',0
       db      0dh,0ah,  '$'
begin: mov     ax,cs           ;set up ds
```

```

        mov    ds,ax          ;to same as cs
        mov    dx,offset file ;address of file to open
        mov    al,0           ;attributes
        mov    ah,3dh         ;open file function request
        int    21h           ;call DOS
        jc    error          ;check if error
        mov    handle,ax      ;save file handle
        mov    bx,handle      ;handle for file to get date / time
        mov    al,0           ;get date and time
        mov    ah,57h         ;get / set file date / time funct req
        int    21h           ;call DOS
        jc    error          ;check for error
        mov    time,cx        ;time last written
        mov    date,dx        ;date last written
        mov    dx,offset msg1 ;address of date message
        mov    ah,09H          ;display string function request
        int    21h           ;call DOS
        mov    bx,handle      ;file handle for file to close
        mov    ah,3eh          ;close file function request
        int    21h           ;call DOS
        jmp    done           ;wrap it up
error:  mov    dx,offset msg2 ;address of error message
        mov    ah,09h          ;display string function request
        int    21h           ;call DOS
done:   mov    ah,4ch          ;terminate process funct request
        int    21h           ;call DOS
code:   ends
end    start          ;start is the entry point

```

#### Pascal Usage Example:

```
{ Get / Set File Date or Time < $57 } 
```

PROGRAM open file;

#### CONST

```

dos open file = $3d ;
dos close file = $3e ;
dos file date time = $57 ;
read access = 0 ;

```

```

carry flag = 1 ;
get time = 0 ;

TYPE
  regpack = RECORD
    CASE INTEGER of
      1 : (ax,bx,cx,dx,dp,si,di,ds,es,flags:integer);
      2 : (al,ah,bl,bh,cl,ch,dl,dh:byte) ;
    END ;

VAR
  regs : regpack ;
  file to open : string [50] ;
  handle : integer ;
  date,time : integer ;

BEGIN
  WITH regs DO
    BEGIN
      file to open := testfile.asc' + chr(0);
      ah := dos open file ;
      ds := seg(file to open[1]) ;
      dx := ofs(file to open[1]) ;
      al := read access ;
      msdos(regs);
      IF((flags AND carry flag) = 1)THEN
        writeln
        ( Unable to get date / time TESTFILE.ASC last written.)
      ELSE
        BEGIN
          date := dx ;
          time := cx ;
          writeln
          ( Got date and time TESTFILE.ASC last written.) ;
        END ;
        bx := handle ;
        ah := dos close file ;
        msdos(regs);
    END ;

```

```
    END ;  
END.
```

#### C Usage Example:

```
/*  
 * Get / Set File Date / Time (0x57)  
 */  
  
#include <dos.h>  
  
union REGS inregs, outregs;  
char filename[] = "testfile.asc";  
main()  
{  
    int handle,date,time;  
  
    inregs.x.dx = (int) &filename[0];  
    inregs.h.al = 0;  
    inregs.h.ah = 0x57;  
    intdos(&inregs,&outregs);  
    if (outregs.x.cflag != 0)  
        printf  
        ("Unable to get date / time TESTFILE.ASC last written.");  
    else {  
        date = outregs.x.dx;  
        time = outregs.x.cx;  
        printf  
        ("Got date and time TESTFILE.ASC last written.");  
    }  
    inregs.x.bx = handle;  
    inregs.h.ah = 0x3e;  
    intdos(&inregs,&outregs);  
}
```

## DOS 功能调用 58H 取或置内存分配对策

功能调用：

## 58H 取或置内存分配对策

DOS 版本:

3.2, 3.3

说明:

本功能调用允许用户查询或设置 DOS 为程序申请 (如通过功能调用 48H 或 4AH) 分配存贮空间和方法。

不用此功能调用时, DOS 从存贮空间低端开始, 将所找到的最大满足程序需要的存贮块分配给程序; 用此功能调用时, 可让 DOS 从存贮空间的高端开始搜索, 或通知 DOS 搜索全部可用空间, 从中挑出一个满足需要的最小存贮块分配给程序。

入口参数:

调用前, 需设置:

AH = 58H 指出本功能调用号。

AL 按以下数值设置本寄存器, 以表明所要完成的操作类型:

001H 获取内存分配:

01H 设置内存分配方案。

BX 若为设置内存分配方案 (AL=01H), 则按以下数值设置此寄存器, 以指明要求采用的内存分配对策:

0 首次适配: DOS 将从内存低端开始查找, 并指定第一个足够大的可用存贮块。当不使用本功能调用改变分配策略时, 这是系统自动设置的方案 (又称缺省对策)。

1 最佳适配: DOS 将查找所有可用内存块, 分配满足程序要求的最小可用块。

2 最后适配: DOS 将从内存高端开始查找, 分配第一个满足要求的可用内存块。若用户只需获取了解内存分配方案 (AL=00H), 则不需设置此寄存器。

出口参数:

控制返回后, 设置如下:

AX 若为获取内存分配方案, 且不出错时, 则此寄存器应设以下数值, 以表明当前 DOS 的分配策略。

0 首次适配

1 最佳适配

2 最后适配

若为设置内存分配方案, 且不出错时, 此寄存器信息无意义。

错误信息:

如果出现错误, 则设置进位标志, 且 AX 寄存器数值如下:

01H AL 寄存器中放入了 0 或 1 之外的其它值, 或在 BX 寄存器中设置了 0、1 或 2 这样的值。

参见:

48H—分配内存。

4AH—修改所分配的内存块 (SETBLOCK)。

4BH—装入或执行程序 (EXEC)。

**59H—取扩充错误代码.**

## **DOS 功能调用 59H**

**取扩充错误代码**

### **功能调用:**

**59H 取扩充错误代码**

### **DOS 版本:**

**3.0, 3.1, 3.2, 3.3**

### **说明:**

本功能调用返回一个在执行 DOS 功能调用的中断 24H 错误处理程序时所出现的附加错误信息。可在下述任何一种状态下调用本功能：

- 当 DOS 返回一个错误代码时，可从中断 24H 错误处理程序内。
- 当通过中断 21H 调用的 DOS2.X 和 3.X 的功能调用，且通过这些功能调用设置的进位位表明出错信息时。
- 当使用 DOS1.X 功能调用，且通过这些功能调用设置的进位位表明出错信息。
- 当使用 DOS1.X 功能调用，且这些功能和调用且这些功能调用通过在 AX 寄存器中返回 FFH 来表明出错信息时。

当出现错误后，可立即用功能调用 59H，并返回刚出现的有关错误的四种信息，即错误代码、错误类别（它提供上述错误类型的更详细信息）、建议的行动和出错位置（系统出错部分的大概位置）。

对于 DOS2.X 来说，只有通过置进位标志错误时，才能利用本功能调用的有关信息。对于 DOS2.X 的其它功能调用，仅在 AX 寄存器中返回出错信息。对于置进位标志的功能调用，应主要使用本功能调用返回错误信息，而不是使用 AX 寄存器中的错误代码。

由于本功能调用返回时，破坏了寄存器 DX、SI、DI、ES、CL 和 DS 的内容，因此，在调用本功能前，应将上述所用寄存器中的数值保存起来。

### **入口参数:**

调用前，需设置：

**AH = 59H 指出功能调用号**

**BX 设置此寄存器以表明用户所需的错误信息的版本，对于 DOS3.X 而言，错误信息只有一种版本，此时寄存器应置为 0。**

### **出口参数**

控制返回后，设置：

**AX 为扩充错误代码，表 4-1（见第四章）列出了可返回的各种错误代码。**

**BH 为扩充错误类型的代码，它给出有关错误的更多信息。表 4-2 列出了各种错误类型。**

**BL 为建议行动的代码，表 4-3 列出了各种可能的建议行动。**

**CH 为位置代码，它指明系统出错的有关部位，表 4-4 列出可能的位置。**

### 程序实例：

下述三个程序实例均用以打开文件 CLORP.YUK。当无法打开此文件时，使用功能调用 59H 接收扩充错误信息。虽然此例不显示出错信息，但用户程序可将这些扩充错误代码转换成文字说明，或根据返回的数值采取相应的动作。

#### Assembly Language Usage Example:

```
;  
; GET EXTENDED ERROR (59H)  
;  
code    segment public  
assume cs:code,ds:code  
        org      100h  
start: jmp     begin  
file    db      glorp.yuk',0  
handle  dw      ?  
msg1   db      'GLORP.YUK opened.',0dh,0ah, '$'  
msg2   db      'GLOPR.YUK not opened.',0dh,0ah,  
           db      'Extended error information returned.',0dh,0ah, '$'  
exerror dw      ?  
class   db      ?  
action  db      ?  
locus   db      ?  
begin: mov     ax,cs          ;set up ds  
        mov     ds,ax          ;to same as cs  
        mov     dx,offset file ;address of file to open  
        mov     al,0            ;access mode  
        mov     ah,3dh          ;open file function request  
        int     21h            ;call DOS  
        jc     error           ;check if error  
        mov     handle,ax       ;save file handle  
        mov     dx,offset msg1 ;address of opened message  
        mov     ah,09h          ;display string function request  
        int     21h            ;call DOS  
        mov     bx,handle       ;file handle for file to close  
        mov     ah,3eh          ;close file function request  
        int     21h            ;call DOS  
        jmp     done             ;wrap it up  
error:  mov     ah,59h          ;get extended error funct request  
        mov     bx,0            ;zero for dos 3.0 and 3.10
```

```

int    21h          ;call DOS
mov    exerror,ax   ;extended error code
mov    class,bh     ;error class code
mov    action,bl    ;suggested action
mov    locus,ch     ;locus
mov    dx,offset msg2 ;address of error message
mov    ah,09h        ;display string function request
int    21h          ;call DOS
done: mov    ah,4ch    ;terminate process funct request
      int    21h          ;call DOS
code  ends          ;end of code segment
end    start         ;start is the entry point

```

**Pascal Usage Example:**

```
{ Get Extended Error {$59} }
```

```

PROGRAM get_extended_error;
CONST
  dos_open_file = $3d;
  dos_close_file = $3e;
  dos_get_error = $59;
  read_access = 0;
  carry_flag = 1;
  version = 0;

TYPE
  regpack = RECORD
    CASE integer OF
      1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
      2 : (al,ah,bl,bh,cl,ch,dl,dh:byte);
    END;
VAR
  regs : regpack;
  file_to_open : string [50];

```

```

BEGIN
  WITH regs DO
    BEGIN
      file_to_open := 'glorp.yuk' + chr(0);

```

```

ah := dos open file ;
ds := seg(file to open[1] ;
dx := ofs(file to open[1] ;
al := read access ;
msdos(regs);
IF ((flags AND carry flag) = 1) THEN
BEGIN
writeln( GLORP.YUK not opened.) ;
writeln( Extended error information returned.) ;
ah := dos get error ;
bx := version ;
msdos(regs);
writeln( Extened error code = ',ax);
writeln( Error class = ',bh);
writeln( Suggested action =',bl);
writeln( Locus= ,ch);

END
ELSE
BEGIN
writeln( GLORP.YUK opened.);
bx := ax ;
ah := dos close file ;
msdos(regs) ;
END ;
END ;

```

### C Usage Example:

```

/* *
 * Get Extended Error (0X59)
 */

```

```
#include <dos.h>
```

```
union REGS inregs,outregs;
char filename[] = "glorp.yuk";
```

```
main()
```

```

{
    inregs.x.dx = (int) &filename[0];
    inregs.h.al = 0;
    inregs.h.ah = 0x3d;
    intdos(&inregs,&outregs);
    if (outregs.x.cflag != 0){
        printf
        ("GLORP.YUK not opened.\nExtended error info returned.\n");
        inregs.x.bx = 0;
        inregs.h.ah = 0x59;
        intdos(&inregs,&outregs);
        printf("Extended error code = %d\n",outregs.x.ax);
        printf("Error class = %d\n",outregs.h.bh);
        printf("Suggested action = %d\n",outregs.h.bl);
        printf("Locus = %d\n",outregs.h.ch);
    }
    else {
        printf("GLORP.YUK opened.");
        inregs.x.bx = outregs.x.ax;
        inregs.h.ah = 0X3e;
        intdos(&inregs,&outregs);
    }
}

```

## DOS 功能调用 5AH

创建唯一文件

**功能调用:**

5AH 创建唯一文件

**DOS 版本:**

3.0, 3.1, 3.2, 3.3

**说明:**

本功能调用检查用户指定的文件目录，并在那个目录中创建一个不重名的新文件，这对于需要临时文件的程序是非常有用的。使用本功能调用可建立一个临时文件，并确保所产生的文件不与现有文件重名。因而不必检查每个文件名本身，或者在不检查（和尽可能不破坏现有文件）的情况下，使用一个独特的文件名。

为了使用本功能调用，调用程序对将要建立的新文件目录具有创建访问权。

**入口参数:**

调用前，需设置：

AH=5AH 指出功能调用号。

DS: DX 本寄存器对需指明建立新文件的目录，该目录将指向包含驱动器和目录路径的 ASCII Z 字符串。路径的最一个字符必须是斜线 (\)。ASCII Z 字符串本身，需以 0 字节结束。

CX 用户所需的新文件的文件属性寄存器，其低位字节编码如下：

位	说明
7	只读文件
6	隐含文件
5	系统文件
4	卷标
3	子目录
2	归档
1	未用
0	未用

为了表示属性，寄存器的相应位需置为 1。

位	说明
0	只读文件
1	隐含文件
2	系统文件
3	卷标
4	子目录
5	归档
6-7	未用

出口参数：

控制返回后，设置为：

DS: DX 此寄存器对指向一个有驱动器、路径和文件名的 ASCII Z 字符串，此字符串含有新文件的 ASCII Z 字符串，需以字节 0 结束。

其它要求：

本功能调用只创建新文件，但不打开它。在使用文件之前，必需用功能调用 3DH 打开文件。在完成文件的使用之后，应该关闭该文件。如果程序用新文件作为临时文件使用，在结束文件使用之后，必须删除此文件。当该程序还存在时，DOS 不要删除此文件。

为了在 DOS3.1 或以后版本的网络环境下使用本功能调用，用户必须有创建目录及其文件的访问权。

错误信息：

如果出错，则置进位标志，且 AX 寄存器为以下数值之一：

03H—至少一目录路径名不存在。

04H—非法文件句柄（打开文件太多）。

05H—在根目录创建文件时根目录已满。

参见：

16H—创建文件

3CH—创建文件 (CREATE)。

58H—取扩充错误代码。

程序实例：

下述三个程序实例是使用功能调用 5AH 在缺省驱动器的当前目录中，创建一个唯一性文件。

### **Assembly Language Usage Example:**

```
;  
; CREATE UNIQUE FILE (5ah)  
;  
code segment public  
assume cs:code,ds:code  
    org 100h  
start: jmp begin  
handle dw ?  
msg1 db "File created in current directory of default"  
      db ".drive.",0dh,0ah,  
path db ".\\",0  
file db ', $'  
msg2 db "File not created',0dh,0ah, '$'  
begin: mov ax,cs           ;set up ds  
       mov ds,ax           ;to same as cs  
       mov dx,offset path ;address of path for new file  
       mov cx,0             ;file attributes for new file  
       mov ah,5Ah            ;create unique file funct request  
       int 21h              ;call DOS  
       jc error             ;check if error  
       mov handle,ax         ;save file handle  
       mov dx,offset msg1   ;address of created message  
       mov ah,09h            ;display string function request  
       int 21h              ;call DOS  
       mov bx,handle          ;file handle for file to close  
       mov ah,3ed             ;close file function request  
       int 21h              ;call DOS  
       jmp done              ;wrap it up  
error: mov dx,offset msg2 ;address of not created message  
       mov ah,09h            ;display string function request  
       int 21h              ;call DOS  
done:  mov ah,4ch            ;terminate process funct request  
       int 21h              ;call DOS  
code ends               ;end of code segment  
end start                ;start is the entry point  
error: mov dx,offset msg2 ;address of not created message  
       mov ah,09h            ;display string function request
```

```

done:  mov      ah,4ch          ;call DOS
        int      21h          ;call DOS
code   ends           ;end of code segment
        end      start         ;start is the entry point

```

**Pascal Usage Example:**

{ Create Unique File (\$ 5a) }

PROGRAM create unique file ;

CONST

```

dos create unique file = $ 5a ;
dos close file = $ 3e ;
normal attrib = 0 ;
carry flag = 1 ;

```

TYPE

```

regpack = RECORD
  CASE integer OF
    1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
    2 : (al,ah,b1,bh,c1,ch,d1,dh:byte);
  END ;

```

VAR

```

regs : regpack ;
new file : string [50] ;

```

BEGIN

WITH regs DO

BEGIN

```

  new file := .\ ' + chr(0) + '
  ah := dos create unique file ;
  ds := seg(new file[1]) ;
  dx := ofs(new file[1]) ;
  cx := normal attrib ;
  msdos(regs) ;

```

IF ((flags AND carry flag) = 1) THEN

writeln(' File not created.')

ELSE

```

BEGIN
    writeln
        (' File created in current directory of default drive.');
    writeln(new_file);
    bx := ax ;
    ah := dos close file ;
    msdos(regs);
END ;
END ;
END.

```

### C Usae Example:

```

/*
 * Create Unique File (0x5a)
 */

#include <dos.h>

union REGS inregs,outregs;
char path[] = "/>0"; . . . . .;

main()
{
    inregs.x.dx = (int)&path[0];
    inregs.x.cx = 0;
    inregs.h.ah = 0x5a;
    intdos(&inregs,&outregs);
    if (outregs.x.cflag != 0)
        printf("File not created.");
    else {
        printf
            ("File created in current directory of default drive.\n");
        printf("%s",path);
        inregs.x.bx = outregs.x.ax;
        inregs.h.ah = 0x3c;
        intdos(&inregs,&outregs);
    }
}

```

## DOS 功能调用 5BH

### 创建一新文件

功能调用:

5BH 创建一新文件

DOS 版本:

3.0, 3.1, 3.2, 3.3

说明:

本功能调用除只创建新文件外，其它功能与功能调用 3CH (创建一文件) 相同。如果要创建的文件已经存在，该功能调用返回出错信息。

为使用本功能，调用程序必须建立对目录及其新文件的访问权。已创建的文件将被打开，可供兼容方式读写使用。

入口参数:

调用前，需设置:

AH = 5BH 指出本功能调用号。

DS: DX 该寄存器对使用有驱动器、跟着 线和文件名 ASCII 字符串来识别新文件。ASCII 字符串必须以字节 0 结尾。

CX 置该寄存器的低位字节为新文件属性编码，其格式为:

位

7 6 5 4 3 2 1 0

为了表示文件属性，寄存器的相应位需置为 1，各位含义为下:

位	说明
0	只读文件
1	隐含文件
2	系统文件
3	卷标
4	子目录
5	归档
6-7	未用

出口参数:

控制返回后:

AX 若未出错，此寄存器保存新文件句柄；若出错，则给出错误代码。

其它要求:

为了在 DOS3.1 或以后版本的网络环境下使用本功能，需创建对目录的访问权。

错误信息:

如果出错，则置进位标志，且 AX 寄存器为以下数值之一:

03H--至少一目录路径名不存在。

04H--非法文件句柄 (打开文件太多)。

05H--在根目录中创建文件时根目录已满。

50H——该文件已存在.

**参见:**

3CH——创建一文件 (CREAT).  
43H——取或置文件属性 (CHMID).  
59H——取库充错误代码.

**程序实例:**

下述三个程序实例是使用功能调用 5BH 在缺省驱动器的根目录中创建一个新文件，  
文件名是 NEWFILE.

**Assembly Language Usage Example:**

```
; ; CREATE NEW FILE (5BH)
;
code    segment public
assume cs:code,ds:code
        org      100h
start: jmp     begin
newfile db      '\newfile',0
handle  dw      ?
msg1   db      'File created in root directory of default'
        db      'drive.',0dh,0ah,  '$'
msg2   db      'File not created',0dh,0ah,  '$'
begin: mov     ax,cs           ;set up ds
        mov     ds,ax           ;to same as cs
        mov     dx,offset newfile;address of path for new file
        mov     cx,0             ;file attributes for new file
        mov     ah,5bh            ;create new file function request
        int     21h              ;call DOS
        jc     error            ;check if error
        mov     handle,ax         ;save file handle
        mov     dx,offset msg1  ;address of created message
        mov     ah,09h            ;display string function request
        int     21h              ;call DOS
        mov     bx,handle          ;file handle for file to close
        mov     ah,3eh            ;close file function request
        int     21h              ;call DOS
        jmp     done              ;jmp to it up
error:  mov     dx,offset msg2  ;address of not created message
        mov     ah,09h            ;display string function request
```

```

        int 21h          ;call DOS
done:  mov ah,4ch      ;terminate process funct request
        int 21h          ;call DOS
code   ends           ;end of code segment
        end start       ;start is the entry point

```

**Pascal Usage Example:**

```

{ Create New File($5b) }
PROGRAM create new file ;

```

**CONST**

```

dos create new file = $5b ;
dos close file = $3e ;
normal attrib = 0 ;
carry flag = 1 ;

```

**TYPE**

```

regpack = RECORD
    CASE integer OF
        1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
        2 : (al,ah,bl,bh,cl,ch,dl,dh:byte) ;
    END ;

```

**VAR**

```

regs : regpack ;
nwe file : string[50] ;

```

**BEGIN**

```

WITH regs DO

```

**BEGIN**

```

    nwe file := '\nwefile' + chr(0) ;
    ah := dos create new file ;
    ds := seg(new file[1]) ;
    dx := ofs(new file[1]) ;
    cx := normal attrib ;
    msdos(regs);

```

```

IF ((flags AND carry flag) = 1) THEN
    writeln(' File not created.')

```

**ELSE**

**BEGIN**

```

        writeln
        ( File created in root directory of default drive. );
        bx := ax ;
        ah := dos close file ;
        msdos(regs);
    END ;
END ;

```

### C Usage Example:

```

/*
 * Create New File (0x5b)
 */

#include <dos.h>
union REGS inregs,outregs;
char newfile[] = "\\\newfile";

main()
{
    inregs.x.dx = (int)&newfile[0];
    inregs.x.cx = 0;
    inregs.h.ah = 0x5b;
    intdos(&inregs,&&outregs);
    if (outregs.x.cflag != 0)
        printf("File not created.");
    else {
        printf("File created in root directory of default drive.");
        inregs.x.bx = outregs.x.ax;
        inregs.h.ah = 0x3e;
        intdos(&inregs,&outregs);
    }
}

```

## DOS 功能调用 5CH

封锁 / 开锁文件访问

### 功能调用:

5CH 封锁 / 开锁文件访问

DOS 版本:

3.0, 3.1, 3.2, 3.3

说明:

本功能调用在几个程序想共同访问同一文件的共享文件环境下是非常有用的。它可使程序封锁文件中的某些字节，以防止任何其它程序读或写该范围内的这些字节，直到第一个程序将这些字节解锁为止。这个被封锁的字节范围称为一个区域，通过本功能调用可规定这个区域的起始位置区域的起始位置和区域的长度。

当一个程序需要更新一个文件时，必须保证在未更新完之前，不得有其它程序来访问这些信息。此时，本功能调用非常有用。

本功能调用只能对下述一些文件实施封锁或开锁，即以拒读或拒非共享方式打开的文件，或者以读 / 写或只写访问以及拒写共享方式打开的文件。在这些方式中，DOS 不在远程磁盘提供本地的文件缓冲区。

如果用户使用本功能调用封锁一部分文件，其它程序试图访问已封锁的部分时，其 DOS 的 I/O 操作将重试一定次数（其次数由功能调用 44H 来确定），如果重试多次后，此部分仍处于封锁状态，则其它程序的 DOS 功能调用便返回出错代码，这时程序应使用功能 59H 以获取更多的错误信息。

通过 I/O 操作的检查错误状态，确定封锁的位置是不可取的。相反的应设置自己的封锁，如果出错，则说明所指定的区域已被封锁。

如果封锁一个文件的某个区域，并复制了该文件句柄（通过功能调用 45H 或 46H），那末任何使用已复制的文件句柄访问该文件的程序也就访问了该封锁区域。然而，如果使用 EXEC 功能调用（4BH）开始一个子进程，则此子进程不访问该封锁区域。

入口参数:

调用前，需设置：

AH = 5CH 指出功能调用号。

AL = 设置文件区封锁或开锁的功能码：

00H 封锁

01H 开锁

BX 本寄存器保存由功能调用 3CH（创建文件）、5BH（创建新文件）和 3DH（打开文件）返回的文件句柄，它标识所需封锁和开锁的文件。

CX, DX 由这两个寄存器组合形成一个四字节的整数，它指明部份文件封锁区域的起始位置，该四字节整数是起始位置相对于文件首地址偏移量。其中 DX 存器存放低位的两字节，CX 寄存器存放高位的两字节。

如果是开锁区域，这四字节整数必须和封锁区域时所使用的四字节整数值相同。

SI, DI 这两个寄存器组成一个四字节整数，表示封锁区域的字节长度，其中低位的两个字节在 DI 寄存器中，高位两个字节在 SI 寄存器中。如果是开锁区域，这四字节整数必须和封锁区域时所使用的四字节整数值相同。

出口参数:

无

### **其它要求:**

关闭一个文件并不会自动地撤除放在文件中的任何已封锁的区域。相反，关闭的文件所含有的封锁区域将产生不确定的结果。因此，在关闭文件之前，必须先将已封锁的区域开锁。

### **错误信息:**

如果出现错误，则设置进位，且在 AX 寄存器中保存下述错误代码之一，应使用功能调用 59H 查看有关错误类型、建议行动和出错位置等更多的信息。

01H 为了使用本功能调用，必须装载文件共享。

06HBX 寄存器存放的文件句柄无效。

21H 所指定的全部或部分区域已被封锁，或试图开锁的区域与封锁区域不一致。

24H DOS 用以保存封锁区域磁道的缓冲区已满。

### **参见:**

44H--控制 I/O 设备 (IOCTL)。

59H--取扩充错误代码。

## **DOS 功能调用 5DH**

### **DOS 内部使用功能**

### **功能调用:**

5DH

本功能调用留作 DOS 内部使用，用户程序不能调用。

## **DOS 功能调用 5EH**

### **建立网络参数**

### **功能调用:**

5EH 建立网络参数

### **DOS 版本:**

3.1, 3.2

### **说明:**

本功能调用完成有关设置 IBM PC 网络方面的三种操作，即获取、设置打印机参数串以及获取机器名。所有这些操作均要求装载 IBM PC 网络程序。

通过设置 AL 寄存器，选择所需完成的操作（或子功能），由于具有三个独立的子功能，因此这里也分成以下三节来分别说明：

AL 三节内容

00H: 获取机器名

02H: 设置打印机参数

03H: 获取打印机参数

## DOS 功能调用 5EH

建立网络参数

取机器名

子功能:

    获取机器名

DOS 版本:

    3.1, 3.2, 3.3

说明:

    本子功能调用要求必须装载 IBM PC 网络程序。此时，返回本地计算机的名称和由 NETBIOS 分配给它的名称。

入口参数:

    调用前，需设置:

        AH = 5EH 指出功能调用号。

        AL = 00H 表示获取机器名的子功能。

        DS: DX 本寄存器对指向返回计算机名用的一个内存缓冲区，该缓冲区的长度应足够容纳一个完整的名称和一个全零字节的结束标志。

出口参数:

    控制返回后，设置为:

        DS: DX 本寄存器对指向含计算机名的一个 ASCII 字符串，该字符串必须以一个全零字节结束。

        CH 本寄存器指明该计算机名称的有效性和由此子功能返回的数值如下:

            0 该计算机名未定义。CL 寄存器和由 DS: DX 所指示的缓冲区中保存的信息无效。

            非 0 CL 寄存器和由 DS: DX 所指示的名有效。

        CL 保存给已返回的计算机名所分配的 NETBIOS 号。

错误信息:

    如果出错，则设置进位标志，且 AX 寄存器存放以下错误代码。应使用功能调用 59H 查看有关错误类型、建议的行动和错误位置等更多的信息。

    01H IBM PC 网络程序未运行。

## DOS 功能调用 5EH

建立网络参数

设置打印机参数

子功能:

    设置打印机参数

DOS 版本:

### **3.1, 3.2, 3.3**

#### **说明:**

本子功能允许用户设置打印机参数串，DOS 把此参数串设置在发送给打印机的文件的前边，其长度可达 64 (十进制)个字节，并包含用户所需的任何信息，通常根据不同的打印机来设置。典型的参数串包含建立字体、字符尺寸、页边空白等控制代码或有关打印机的其它信息。

本功能调用提供给网络环境，且只有装载了 IBM PC 网络程序后才能工作。使用本功能调用，可使多个用户同时发送信息给一部网络打印机，而每个用户同时发送信息给一部网络打印机，而每个用户可根据各自的需要设置该打印机。

#### **入口参数:**

调用前，需设置：

AH = 5EH 指出功能调用号。

AL = 02H 设置打印机参数子功能

BX 此寄存器保存打印机重定向表索引，该索引是由网络重定向功能调用(5FH)中的设备重定向子功能建立的，它标识此参数串所适用的网络打印机。

CX 设置此寄存器以指明用户所提供的参数串的字节长度，由于此参数串不同于其它字符串，它并不是一个 ASCII 字符串，因此用户必须提供这些信息。

DS: SI 此寄存器对指向存放打印机参数串的内存缓冲区。

#### **出口参数:**

无

#### **错误信息:**

如果出错，则设置进位标志，且在 AX 寄存器中保存以下出错代码，应使用功能调用 59H 以查看有关错误类型、建议的行动和出错位置等更多的信息。

#### **参见:**

59H—取扩充错误代码。

5EH—网络参数 (子功能 03H，取打印机参数)。

5FH—网络重定向 (子功能 03H，设备重定向)。

## **DOS 功能调用 5EH**

建立网络参数

取打印机参数

#### **子功能:**

取打印机参数

#### **DOS 版本:**

3.1, 3.2, 3.3

#### **说明:**

本子功能实际上是对指定的打印机返回当前打印机参数串的有关信息。该打印机参数串由设置打印机参数串子功能 (子功能 02H) 建立。

此功能调用仅当 IBM PC 网络程序被装载后才能使用。

**入口参数:**

调用前, 需设置:

AH = 5EH 指出功能调用号。

AL = 03H 表示取打印机参数串子功能。

DX 此寄存器保存打印机重定向表索引, 该索引是由网络重定向功能调用 (5FH) 的设备重定向 (03H) 子功能建立的, 它标识此参数串所适用的网络打印机。

ES: SI 此寄存器对指向子功能返回的打印机参数串所在的内存缓冲区。

**出口参数:**

控制返回后, 需设置:

CX 指明用户所提供的打印机参数串的字节长度, 由于该参数串不同于其它字符串, 它不是一个 ASCII 字符串, 因此这个长度非常重要。

ES: SI 此寄存器对指向打印机参数串用的内存缓冲区, 所返回的长度保存在 CX 中。

**错误信息:**

如果出错, 则设置进位标志, 且在 AX 寄存器中保存以下错误代码。应使用功能调用 59H 以接收有关出错类型、建议的行动和出错位置等更多的信息。

01H IBM PC 网络程序没被装载。

**参见**

59H—取扩充错误代码。

5EH—网络参数 (子功能 02H, 设置打印机参数)。

5FH—网络重定向 (子功能 03H, 设备重定向)。

## DOS 功能调用 5FH

### 网络重定向

**功能调用:**

5FH 网络重定向

**DOS 版本:**

3.1, 3.2, 3.3

**说明:**

本功能调用完成有关网络设备重定向的三个操作, 使这些设备能够本地使用。这些操作包括获取重定向表、设备重定向和清除重定向。所有这些操作均需装 IBM PC 网络程序。

所需完成的操作 (或子功能) 由 AL 寄存器存放的数值确定。由于是三个独立的子功能, 故分为下述三节分别说明:

AL 分节内容

02H 获取重定向表入口。

03H 设备重定向。

04H 清除重定向。

## DOS 功能调用 5FH

网络重定向

取重定向表入口

子功能：

取重定向表入口

DOS 版本：

3.1, 3.2, 3.3

说明：

本子功能调用返回已用设备重定向 (03H) 子功能重新定向的设备的有关信息，包括设备的类型、设备的本地名称和设备的网络名称。

此功能的每次调用将返回单个设备（即用户在 BX 寄存器中所指明的那些重定向索引之一）的有关信息。除了获取所有网络设备的信息，每调用本子功能一次则 BX 寄存器的值加 1。然而，重定向表中的内容在两次调用之间可能发生变化。

由于此子功能的返回，DX 和 BP 寄存器中的内容将被破坏，故在调用此子功能之前，程序需将这些寄存器中的信息保存起来。

当 IBM PC 网络程序未装载时，本子功能调用无效。即使磁盘或打印机重定向已中止（将引起这些网络连接设备的短暂中断），此子功能调用仍返回有效信息。

入口参数：

调用前，需设置：

AH = 5FH 指出功能调用号。

AL = 02H 不示获取重定向表入口的子功能调用。

BX 此寄存器保存用户所需了解的有关设备的重定向表索引。

0： 可用的网络设备。

1： 不可用设备，且 BL 中的返回信息、DS: SI 和 ES: DI 缓冲区中的返回信息不能使用。

BL 此寄存器表明网络设备的类型，内容如下：

03H 为打印机设备。

04H 为文件设备。

CX 此寄存器专为 IBM PC 网络程序而设置，当使用其它程序时，此寄存器可能被省略。

DS: SI 此寄存器对指向一个 128 字节内存缓冲区，缓冲区存放标识该设备本地名的 ASC II Z 字符串。字符串以一个全零字节结束。

ES: DI 此寄存器对指向一个 128 字节缓冲区，该缓冲区存放一个标识该设备全局名的 ASC II Z 字符串。字符串以一个全零字节结束。

错误信息：

如果出错，则设置进位标志，且 AX 寄存器中保存以下错误代码。 使用功能调用

**59H** 查看有关出错类型、建议的行动和出错位置等更多的信息。

01H IBM PC 网络程序没被装载。

12H 重定向表中无多余设备或在 BX 指定中索引大于重定向表中的人口数目。

参见

**59H—取消或清除错误代码。**

**5FH—网络重定向 (子功能 03H, 设备重定向)。**

## DOS 功能调用 **5FH**

网络重定向

设备重定向

子功能:

设备重定向

DOS 版本:

3.1, 3.2, 3.3

说明:

本子功能使本地计算机与使用网络打印机或网络文件建立起连接，它可将打印机名 (PRN, LPT1, LPT2 或 LPT3) 与一个网络打印机联结起来。因此，任何时候用户只要发送含局部打印机名的文件，则该文件便可在已连接的相应网络打印机上准确地打印输出。DOS 在中断 17H 级上重定向该打印机。

此子功能还可使一个驱动器字母与网络文件服务器的一个目录联结起来。通过使用驱动器标识符，用户可对网络服务器中的文件进行存取。

若用户使用此子功能重定向设备，则当用户键入“NETUSE”命令时，不显示这些设备。

未装载 IBM PC 网络程序时，不能使用此子功能。若试图对不允许重定向，则子功能返回错误信息。

入口参数:

调用前，需设置:

AH = 5FH 指出功能调用号。

AL = 03H 表示设备重定向子功能。

BL 按以下数值设置此寄存器，以指明用户需重定向的网络设备类型:

03H 打印机设备

04H 文件设备

CX 为了与 IBM PC 网络程序兼容，此寄存器应设置为 0。

DS: SI 此寄存器对指向一个 ASCII 字符串，该字符串标识所需定向的设备的本地名，且以一个全零字节结束。

打印机设备的名称可为 PRN, LPT1, LPT2 或 LPT3。

文件设备的名称可为一个驱动器字母（例如：A:、B:、C: 等等）。若字符串是一个空串，则该子功能允许访问网络路径，而无需任何重定向操作。无重定向

的访问意味着用户可以访问网络文件，但只能使用该网络路径名访问。

**EX: DI** 此寄存器对指向两个 ASCⅡ Z 字符串。其中一个字符串网络名或路径，另一个字符串指明访问该网络设备的口令，这两个 ASCⅡ Z 字符串均以一个全零字节结束。

打印机设备名的字符串必须是一个按上述格式表示的网络名字符串：

\计算机名\{缩写名：打印设备}

文件设备名和字符串必须是一个网络路径。

为了访问网络设备，在网络名之后必须紧跟一个访问该设备的 ASCⅡ Z 口令字符串（由 0-8 个字符组成）。该口令字为取得网络设备访问权所必需。

**出口参数：**

无。

**错误信息：**

如果出错，则置进位标志，且 AX 寄存器将保存以下错误代码。应使用功能调用 59H 查看有关错误类型、建议的行动和出错位置等更多的信息。

01H 此代码可表示以下任何一种错误：IBM PC 网络程序未装载，BL 寄存器数值不是 3 或 4，本地或网络字符串格式错以及本地设备已经被重定向。

03H 网络路径无效或不存在。

05H 网络目录各口令的组合无效。

08H 设备重定向所需信息贮存的内存不够。

**参见：**

59H—取扩充错误代码。

5FH—网络重定向（子功能 02H，取重定向表入口）。

5FH—网络重定向（子功能 04H，清除重定向）。

## DOS 功能调用 5FH

**网络重定向**

**清除重定向**

**子功能：**

清除定向

**DOS 版本：**

3.1, 3.2, 3.3

**说明：**

本子功能清除本地计算机与网络打印机或文件设备之间的连接。此连接是由设备重定向（03H）子功能原先建立的。

例如，假定用户已使用子功能 03H 建立起 LPT1 名称与网络打印机之间的连接，则无论何时，只要这种连接存在，便可将文件拷贝到 LPT1: 上，而这时的 LPT1: 就是网络打印机。当调用清除重定向子功能以中止连接时，任何发给 LPT1: 的文件便送到接到用户计算机上第一个并行口的打印机去。

未装载 IBM PC 网络程序时，不能使用此子功能，如果要对已中断重定向的设备清除重定向，则子功能返回一个出错信息。

**入口参数：**

调用前，需设置：

AH=5FH 指出功能调用号。

AL=04H 表示清除重定向子功能。

DS: SI 此寄存器对指向一个 ASCⅡ Z 字符串，该字符串标识必须清除的连接并以一个全零字节结束。

为了清除与网络打印机的连接，在该字符串中应设置打印机的本地名 (PRN、LPT1、LPT2 或 LPT3)。

为清除与网络文件设备的连接，则在此字符串中应设置该设备的本地名（如 A:、B:、等）。如果在重定向时，未建立本地名与网络文件设备之间的联系，则可在此字符串中设置该网络名，以清除本地计算机与该网络设备之间的这种联系关系。

**出口参数：**

无。

**错误信息：**

如果出错，则设置进位标志且 AX 寄存器保存以下错误代码之一，应使用功能调用 59H 查看有关出错类型、建议行动和出错位置等更多的信息。

01H IBM PC 网络程序未装载。

0FH 磁盘和打印机的重定向功能当前无法实现（已中断）。

**参看：**

59H—取扩充错误代码。

5FH—网络重定向（子功能 02H，取重定向表入口）。

5FH—网络重定向（子功能 03H，设备重定向）。

## DOS 功能调用 60H 和 61H

### DOS 内部使用

**功能调用：**

60H 和 61H

这两个功能调用留作 DOS 内部使用，用户程序不得调用。

## DOS 功能调用 62H

### 取程序段前缀地址

**功能调用：**

62H 取程序段前缀地址

**DOS 版本：**

3.0, 3.1, 3.2, 3.3

### 说明:

此功能调用可收回程序的程序段前缀 (PSP) 地址，这对于含有多个代码段 (.EXE 文件) 的程序和需要访问 PSP 的中断句柄是非常有用的。

### 入口参数:

调用前，需设置：

AH=62H 指出功能调用号。

### 出口参数:

控制返回后，设置：

BX 此寄存器保存程序的 PSP 的段地址 (基地址部分)，PSP 是从该程序段的头部开始，而 PSP 地址的偏移量部分为 0，参看第 20 页 PSP 的格式说明。

### 错误信息:

如果出错，则设置进位标志，且 AX 寄存器存放错误代码。应使用功能调用 59H，查看有关出错类型、建议的行动和出错位置等更多的信息。

### 参见:

59H—取扩充错误代码。

### 程序实例:

下述三个程序实例使用功能调用 62H 以接收程序段前缀地址，该程序将在屏幕上显示此地址。

#### Assembly Language Usage Example:

```
; ; GET PSP ADDRESS (62h)
;
code segment public
assume cs:code,ds:code
    org      100h
start: jmp      begin
msg     db      'PSP address = '
psp     db      4 dup (0),odh,0ah, '$'
;
; convert hex nibble to ascii
;
hexasc proc          ;number to convert is in al
    add     al, 0'           ;turn into ascii code
    cmp     al, 9'           ;is it a letter?
    jle     done             ;no,all done
    add     al,7              ;yes,add fudge factor
done:  ret               ;return with answer in al
```

```

hexasc endp ;end of procedure
;
; main program
;
begin: mov ax,cx ;set up ds
        mov ds,ax ;to same as cs
        mov ah,62h ;get PSP address function request
        int 21h ;call DOS
        mov cl,4 ;shift count
        mov al,bl ;PSP segment ,low byte
        shr al,cl ;high nibble of low byte
        call hexasc ;convert hex to ascii
        mov psp[2],al ;store in message
        mov al,bl ;PSP segment,low byte
        and al,0fh ;low nibble of low byte
        call hexasc ;convert hex to ascii
        mov psp[3],al ;store in message
        mov al,bh ;psp segment,high byte
        shr al,cl ;high nibble of high byte
        call hexasc ;convert hex to ascii
        mov psp,al ;store in message
        mov al,bh ;psp segment, high byte
        and al,0fh ;low nibble of high byte
        call hexasc ;convert hex to ascii
        mov psp[1],al ;store in message
        mov dx,offset msg ;start of complete message
        mov ah,09h ;display string function request
        int 21h ;call DOS
        mov ah,4ch ;terminate process funct request
        int 21h ;call DOS
code ends ;end of code segment
end start ;start is the entry point

```

**Pascal Usage Example:**  
{ Get PSP Address (\$ 62) }

PROGRAM get PSP address ;

CONST

```

dos get PSP address= $ 62 ;

TYPE
  regpack = RECORD
    CASE integer OF
      1 : (ax,cx,dx,dp,si,di,ds,es,flags:integer);
      2 : (al,ah,b1,bh,cl,ch,d1,dh:byte);
    END;
VAR
  regs : regpack ;
BEGIN
  WITH regs DO
    BEGIN
      ah := dos get PSP address ;
      msdos(jegs);
      writeln(  PSP address = ',bx),
    END;
END

```

### C Usage Example:

```

/*
 * Get PSP Address (0x62)
 */

#include <dos.h>
union REGS inregs,outregs;
main()
{
  inregs.h.ah = 0x62;
  intdos(&inregs,&outregs);
  printf("PSP address = %X\n",outregs.x.bx);
}

```

## 第五章 BIOS 中断和功能调用

本章详细叙述基本输入 / 输出系统 (BIOS) 所具有的各种服务。这些服务程序常驻 IBM 个人计算机的 ROM 中，只要计算机开始运行，便可以调用它们。BIOS 程序独立于任何操作系统，因此无论计算机是运行 DOS、CP/M-86、PC IX Unix 派生系统，还是其它任何操作系统，用户均可调用这些服务。（有一例外，即使用 PC AT 时，如果使用开关使 80286 操作系统进入保护方式时，不能再访问这些 BIOS 功能调用。）

BIOS 服务比第三章和第四章所述的 DOS 服务更为基础。事实上，DOS 服务通过调用 BIOS 以完成某些操作。因此，当使用 BIOS 服务时，用户自己需要进一步编程，并应当非常熟悉计算机系统所配置的有关设备的操作。

通常，用户不必使用 BIOS 服务，除非感到同样的 DOS 中断或功能调用速度不够快或不能完成用户所需的工作时才这样做。一旦选择了使用 BIOS，就必须认识到用户所编写的程序可能与其它以 DOS 为基础的计算机不兼容。

本章所述的 BIOS 服务调用，如同在第三章和第四章所介绍的 DOS 服务一样，使用软件中断，有些服务有其本身的中断，还有一些是组合一起通过同一个中断来调用的。

### § 5.1 如何使用本章

和上述两章一样，本章是编程的参考资料，不是一般读物。引言之后，将采用标准的格式来说明 BIOS 服务，或功能调用。这些功能调用按数值大小为序来列表（按中断号的大小排列，如果中断号相同，则以功能调用大小排列）。每页顶上的标题有助于用户迅速查找有关内容，而不必去翻阅无关的章节。

为了方便用户查找某一标题下的有关内容，本章有些内容是重复的。但如果用户在使用时，通过查阅页顶标题或目录总表中相应的功能调用，则通常可以很快查到某一标题的所有相关内容，很少需要参阅本书的其它章节。然而，如果其它功能调用提供了类似的服务，则可参看那些功能调用，以便了解其它功能调用是否可以更好地完成用户所需的操作。

每种功能调用的说明均按标准格式给出，以便用户始终在相同的位置上查到同一类型的信息。说明中还列出了支持该功能的各种计算机。当同计算机的 BIOS 实现该功能的方法有所不同时，将指出这些差异。

本章还提供了部分 BIOS 服务的编程实例，这些程序都是可由用户自己键入并运行的完整程序。每个实例均使用 Turbo Pascal 和 Microsoft C 两种语言编制。

### § 5.2 BIOS 数据区

BIOS 除驻留在 ROM 中之外，还使用 256 个字节 RAM 作为数据区。该数据区位于内存绝对地址为 400H-4FFH 的范围内，它包含计算机相应部分如何工作的有关信息。BIOS 在初始化时建立此数据区。

IBM 的有关资料建议用户不要改变 BIOS 数据区中的任何信息。然而，用户程序可以从数据区中读取有用的信息。因此，下面将逐个介绍该数据区中每一字段的内容，若改变其中任何一种信息，都可能产生无法预料的结果。

地址	说明
400H-407H	计算机上 RS232 适配器的基址.
408H-40FH	计算机上的各并行打印机适配器的基址.
410H-411H	该字保存与计算机连接的设备编码表, BIOS 中断 11H(设备测定)可用以返回此信息.
412H	初始化标记.
413H-414H	该字给出计算机中可用 RAM 的总容量.BIOS 中断 12H(内存大小测定)可用以返回此信息.
415H-416H	暂存区域.
417H	这是第一个键盘状态字, 通过编码, 使每位均具有 特定的含义, 其格式如下:  位 

位	说明
0	表明键盘右边的换档(shift)键当前是否被按下(当该位 为 1 时, 表示该键按下, 0 表示未按下).
1	表明键盘左边的换档(shift)键当前是否按下(1 表示该 键按下, 0 表示未按下).
2	表明 Ctrl 键当前是否按下(1 表示该键按下, 0 表示未按 下).

位	说明									
3	表明 Alt 键当前是否按下(1 表示按下,0 表示未按).									
4	屏幕(Scroll)锁定开关状态(1 表示屏幕锁定为开,0 表示关).									
5	数字锁定开关状态(1 表示数字锁定处于开,0 表示关).									
6	大写字母(caps lock)开关状态(1 表示 caps lock)为开,0 表示关).									
7	插入状态,这表明 Ins 键是否已按下,以使计算机进入"插入"方式,1 表明插入状态正工作,0 表明未动作.									
418H	这是第二个键盘状态字,其格式如下: <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <th>位</th> <td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> </table>	位	7	6	5	4	3	2	1	0
位	7	6	5	4	3	2	1	0		

位	说明
0-1	目前未用
2	只用于 PCjr,如果键盘定位(click)接通,则该位为1,若断开则该位为0.
3	如果系统键(Ctrl Num Lock)按下且保持位,则该位为1,当这个系统键依次按下时, BIOS 暂停处理,直至下键按下为止.但它仍响应中断.
4	表明屏幕(Scroll)锁定键当前是否按下(1 表示该键按下,0 表示未按下).
5	表示数字锁定键当前是否按下(1 表示该键按下,0 表示未按下).

位	说明								
6	表明大写字母(Caps)锁定键当前是否按下(1 表示该键按下,0 表示未按下).								
7	表明 Ins 键当前是否按下(1 表示该键按下,0 表示未按下).								
419H	该字节为存储由备用键盘键入的字符而保留.								
41AH~41BH	该字节指向存放键盘键入字符的循环缓冲区首址.								
41CH~41DH	该字节指向存放键盘缓冲区尾址,当该值等于前一字节的值数时,说明该缓冲区满.								
41E~43DH	循环键盘缓冲区,它保存键盘键入的字符,直到程序可以接收这些字符为止.前两个字节指向此缓冲区的当前头和尾.								
43EH	该字节表示磁盘驱动器的搜索状态,0~3 位对应于驱动器 0~3.如果这些位中有一位为 0,则在搜索到磁道之前,必须重新校准相应的驱动器.								
43FH	该字节表示磁盘驱动器的马达状态,如同前一个字段,0~3 位对应驱动器 0~3.如果某位被置为 1,则相应驱动器的马达正在转动.								
440H	该字节保存一个表明驱动器马达接通多长时间的计数,每个时钟节拍,计数减 1.当计数为 0 时马达停转.								
441H	该字节表明磁盘工作状态,它被编码,通过使相应位置 1 来表示一个特定的状态,此字节的格式如下: 位 <table border="1"><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr></table>	7	6	5	4	3	2	1	0
7	6	5	4	3	2	1	0		

位	说明
0	送给磁盘控制器的是无效命令.
1	在盘上未找到地址标记.
0 和 1	试图在有写保护的盘上进行写操作.
2	所请求扇区未找到.
1 和 2	磁盘已清除.
3	DMA 错.
0 和 3	试图使 DMA 对 64KB 存储体进行存取.
4	循环冗余校验(CRC)错.
5	NEC 磁盘控制器片出现错误.
6	无效的查找操作.
7	响应任何命令的无效驱动.
442H~448H	从 NEC 磁盘控制器返回的七个字节状态信息.
449H	该字节指明当前视频方式.参看 BIOS 中断 10H 功能调用 0,用以说明当前视频方式和它的参数值.
44AH~44BH	该字节用以指明显示屏幕的当前列宽,它可以是 14H(十进制 20),28H(十进制 40)或 50H(十进制 80).
44CH~44DH	该字节指明一个显示页面的字节数,它随视频方式 的不同而变化.
44EH~44FH	该字节指明当前显示页面的地址,即显示在当前显 示屏幕上的显示页面.

位	说明
450H~45FH	八个字,每个字均表示有关显示页面内当前光标的位置,每个字的第一个字节表示列数(0~20、40 或 80 列),第二个字节表示行数(0~24 行).
460H~461H	该字节指明光标的形状,第一个字节表示光标字符点阵的最下一行的行号,第二字节表示光标字符点阵的最上一行的行号, BIOS 中断 10H 功能调用 1 设置此光标形状.
462H	该字节指明工作显示页面(一个正在显示的页面)号,由 BIOS 中断 10H 功能调用 5 设置当前工作页面.
463H~464H	该字节指明当前工作显示板的口地址.
465H	该字节指明 6845 芯片的方式寄存器的当前值.
466H	该字节表示当前面板的设置,中断 10 功能调用 0BH 可设置当前面板.
67H~46FH	在 PC 机中,这 5 个字节用以表示磁带控制的定时计数字、CRC 寄存器字和最后输入数值字节. 在 AT 机中,这些字节作为端口使用,从 467H 开始的双字长是一个指针,它指向 BIOS 开关使 80286 由保护虚地址方式转到实地址方式时控制返回的位置.
46CH~46FH	这是 BIOS 作为时钟计数器的一个双字单元,时钟每步进一次,此值增加一次.其值为 0,表示一天开始(午夜),而该值为 1800BOH 时,则表示一天结束,(下一个午夜).当此计数器达到一天结束的值时,计数器清 0,且字节 470H 置 1.中断 1AH 功能调用 0 可从此双字单元中读取一天的时间.

位	说明
470H	这是一个时钟翻转字节.当时钟计数器达到一天结束且复位时,此字节置 1 以表明新的一天开始.中断 1AH 功能调用 0 在读取这一天的时间后,将此字节复位.
471H	如果该字节的第七位置位,表明从键盘已键入 Ctrl-Break 序列.
472H-473H	该字节表明系统正在进行热启动(Ctrl---ALT-Del).热启动时,此寄存器置为 1234H.
474H-488H	这些字节供 PCjr 使用.
489H-4FFH	留作今后附加功能用.

除 BIOS 数据区外, 内存区域 500H-51DH 留给 DOS 和 BASIC 使用, 下述这些字段对于 DOS 是很重要的:

地址	说明
500H	该字节是 DOS 和 BASIC 用以存储显示(打印)屏幕操作的状态,其值为: 00H:屏幕拷贝打印操作成功或无法进行. 01H:屏幕拷贝(打印)操作正在进行. 0FFH:屏幕拷贝打印操作错.
504H	当系统仅有一个磁盘驱动器,但要当两个逻辑磁盘驱动器使用时,DOS 使用此字节,其数值指明物理驱动器何时作为驱动器 A 或 B 使用.当取值为: 00H:驱动器作为驱动器 A 使用. 01H:驱动器作为驱动器 B 使用.

### § 5.3 调用 BIOS 中断

调用 BIOS 服务与调用 DOS 服务相似。首先，用户必须设置有关寄存器的数值（参看本章有关部分的说明），然后通过调用包含当前中断号的中断指令（Int）获取该服务。当 BIOS 服务完成其处理过程时，用户可以查看 BIOS 存放返回信息的那些寄存器的数据区（参看本章参考部分所列的那些寄存器）。

例如，下面是一个用汇编语言编写的调用 BIOS 中断 10H 以清除显示屏幕的程序实例（确切地说，是将整屏信息向上滚出屏幕）。

```
;  
CLEAR SCREEN  
;  
code      segment public  
assume cs:code,ds:code  
        org      looh  
start:  jmp      begin  
begin:   mov      al,0          ;blank entire window  
        mov      ch,0          ;row of upper left corner  
        mov      cl,0          ;col of upper left corner  
        mov      dh,24         ;row of lowerright corner  
        mov      dl,7q         ;col of lower right corner  
        mov      bh,7           ;normal attributes  
        mov      ah,06h         ;scroll active page up  
        int      10h           ;BIOS video i/o interrupt  
        int      20h           ;program terminate interrupt  
code      ends  
end      start           ;start is the entry point
```

可以看出，该汇编程序在 AL、CH、CL、DH、DL 和 BX 寄存器中设有调用参数，接着将功能调用号（06H）放入 AH 并调用中断 10H。

为从 C 语言或 Pascal 语言中调用 BIOS 服务，用户可使用第四章所述的与调用 DOS 功能相似的机制。例如，下面是一个用 C 语言编写的利用 BIOS 中断 10H 清屏的上述汇编语言程序等效程序。

```
/*  
 * Clear Screen  
 */  
#include<dos.h>  
#define VIDEO_IO 0X10
```

```

union REGS inregs, outregs;
main()
{
    inregs.h.al = 0;
    inregs.h.ch = 0;
    inregs.h.cl = 0;
    inregs.h.dh = 24;
    inregs.h.dl = 79;
    inregs.h.bh = 7;
    inregs.h.ah = 6;
    int86(VIDEO_IO, &inregs, &outregs);
}

```

用 Pascal 编写的另一等效程序如下：

```

{Clear Screen}
PROGRAM clear_Screen;
CONST
    video_io = $10;
    scroll_up = $06;
    normal_attrib = $07;
TYPE
    regpack = RECORD
        CASE integer OF
            1:(ax,bx,cx,bp,si,di,
                ds,cs,flags:integer);
            2:(al,ah,b1,bh,cl,ch,
                dl,dh:byte);
        END;
    VAR
        regs:regpack;
    BEGIN
        WITH regs DO
        BEGIN
            al := 0;
            ch := 0;
            cl := 0;
            dh := 0;
            dl := 0;
            bh := normal_attrib;
            ah := SCROLL_UP
            intr(video_io,regs);
        END;
    END;

```

END

END

## § 5.4 BIOS 功能调用目录

中断调用				功能
16进制	10进制	16进制	10进制	
05H	05			屏幕拷贝(打印)
10H	16	00H	0	显示 I/O
		01H	1	置显示方式
		02H	2	置光标类型
		03H	3	置光标位置
		04H	4	读光标位置
		05H	5	读光笔位置
		06H	6	置当前显示页
		07H	7	当前显示页上滚
		08H	8	当前显示页下滚
		09H	9	读字符和属性
		0AH	10	写字符和属性
		0BH	11	写字符
		0CH	12	置彩色调色板
		0DH	13	写点
		0EH	14	读点
		0FH	15	以电传方式写字符
		13H	19	取当前显示方式
11H	17			写字符串
12H	18			确定设备
				确定内存容量

中断调用				功能
16进制	10进制	16进制	10进制	
13H	19	00H	0	磁盘I/O
		01H	1	复位磁盘
		02H	2	取磁盘状态
		03H	3	读扇区
		04H	4	写扇区
		05H	5	检测扇区
		08H	8	格式化磁道
		09H	9	取当前驱动器参数
		0AH	10	初始化双驱动器
		0BH	11	读长扇区
		0CH	12	写长扇区
		0DH	13	查找柱面(磁道)
		10H	16	备用磁盘复位
		11H	17	检测驱动器准备
		14H	20	复校驱动器
		15H	21	控制器内部诊断
		16H	22	取磁盘类型
		17H	23	改变磁盘状态
				置磁盘类型
14H	20	RS-232串行I/O		
		00H	0	初始化串行口
		01H	1	发送一字符
		02H	2	接收一字符
		03H	3	取串行口状态

中断调用				功能
16进制	10进制	16进制	10进制	
15H	21	00H	0	磁带I/O
		01H	1	启动盒式磁带机
		02H	2	停止盒式磁带机
		03H	3	读数据块
				写数据块
15H	21	83H	131	AT机扩充服务
		84H	132	事件等待
		85H	133	控制杆支持
		86H	134	SYS Req键支持
		87H	135	等待
		88H	136	成块传送
		89H	137	取扩展内存容量
		90H	144	转移到PVAM
		91H	145	设备忙循环(等待)
				建立中断结束标志 (POST)
16H	22	00H	0	键盘I/O
		01H	1	读下一键盘符
		02H	2	检测字符是否准备好
				读当前转换键状态
17H	23	00H	0	打印机I/O
		01H	1	打印一字符
		02H	2	初始化打印机口
				取打印机状态

中断调用				功能
16进制	10进制	16进制	10进制	
18H	24			ROM BASIC 语言
19H	25			引导装入程序(重新 启动系统)
1AH	236	00H	0	实时时钟
		01H	1	读当前时钟值
		02H	2	置当前时钟值
		03H	3	读电池供电时钟时间
		04H	4	置电池供电时钟时间
		05H	5	读电池供电时钟日期
		06H	6	置电池供电时钟日期
		07H	7	置闹钟
				复位闹钟
1BH	27			键盘终止地址
1CH	28			定时器信号
1EH	30			软盘参数表
1FH	31			图形字符扩展码
41H 和 46H	65 和 70			硬盘参数表

## **BIOS Int 05H**

**屏幕拷贝（打印）**

**中断：**

05H 屏幕拷贝（打印）

**计算机：**

PC, PCjr, XT, 便携机和 AT

**说明：**

本中断可调用 PC 机的屏幕拷贝服务程序，即只要用户调用本中断，在打印机设置情况下，将屏幕上的信息送往 0 端口打印机。

无论显示屏幕处于文本方式还是处于图形方式，该中断服务程序均保持当前的光标，并打印所有的可打印字符。

**入口参数：**

调用此服务程序时，用户不设置任何寄存器，而只要简单地调用中断 05H 即可。

**出口参数：**

将屏幕内容送到打印口 0.

**错误信息：**

该中断服务程序不在任何寄存器中返回状态信息，而在绝对地址 500H 处返回屏幕拷贝服务程序的状态，该字节表示的状态类型由以下数值确定：

00H 屏幕拷贝操作已完成（或没有操作请求）；

01H 屏幕拷贝操作正在进行中；

FFH 屏幕拷贝错。

**程序实例：**

下述两个实例为使用 BIOS 中断 05H 以打印屏幕信息的程序。

**Pascal Usage Example:**

```
PROGRAM clear_Screen;
  CONST
    print_screen = $05;
  TYPE
    regpack = RECORD
      CASE integer OF
        1:(ax,bx,cx,bp,si,di,
           ds,es,flags:integer);
        2:(al,ah,bl,bh,cl,ch,
           dl,dh:byte);
    END;
  VAR
    regs:regpack;
```

```
BEGIN
    intr(print_screen,regs);
END.
```

#### C Usage Example:

```
/*
 * print Screen(0x05)
 */
#include <dos.h>
union REGS inregs,outregs;
main()
{
    int86(0x05,&inregs,&outregs);
}
```

### BIOS Int 10H

#### 显示 I/O

##### 中断:

10H 显示 I/O

##### 说明:

BIOS 借助于该中断产生的功能调用控制 PC 机屏幕上的文本和图形。通过给 AH 寄存器设置适当的值选择所希望的功能调用，然后激活中断 10H。此中断可提供如下功能调用：

AH	功能调用
00H	置显示方式
01H	置光标类型
02H	置光标位置
03H	读光标位置
04H	读光笔位置
05H	选择当前显示页
06H	当前显示页上滚
07H	当前显示页下滚
08H	读字符和属性
09H	写字符和属性
0AH	写字符
0BH	置彩色调色板
0CH	写点
0DH	读点

**0EH** 以电传方式写字符

**0FH** 取当前显示方式

**13H** 写字符串

下面详细描述了这些显示功能调用。

## **BIOS Int 10H**

### **显示 I/O**

#### **功能调用 00H——置显示方式**

**中断:**

10H 显示 I/O

**功能调用:**

00H 置显示方式

**计算机:**

PC, PCjr, XT, Portable 和 AY

**说明:**

ROM BIOS 在显示屏幕上支持几种不同的显示信息方法（或方式）。这些方式控制显示适配器是以文本方式工作，还是以图形方式工作，控制显示的文本数量、屏幕的分辨率以及色彩。该功能调用设置显示方式。

就 IBM PC 而言，并不是所有的显示适配器都能使用这些显示方式。标准的彩色/图形适配器 (CGA) 支持七种不同的文本和图形方式，PCjr 除支持这七种方式外，还有它自己的三种。单色显示适配器有一专用的单独单色文本方式。而增强型图形适配器 (EGA) 除了支持基本的七种 CGA 方式，其中之一用于单色适配器所用的方式，另外三种专门用于它的方式。

除专用于单色适配器的方式外，所有方式都允许设置彩色属性，彩色方式把这些属性转换成屏幕上的实际色彩。黑白（或彩色控制）方式把彩色属性转换成灰度，除了彩色同步信号不能实现以外，它们和对应的彩色方式一样。用于单色适配器的方式不支持彩色或灰度，但能够显示字符的明暗，带下划线以及反显。

由彩色/图形适配器支持的方式可以用不同的方式划分。如前所述，有彩色方式和彩色控制方式。但是这些方式也可以根据是由文字还是图形构成屏幕来划分。

在文本方式中，光标总是显示在屏幕上。在图形方式中，不显示光标。文本方式能够显示的信息种类受到限制，只能显示标准 IBM 字符中的字符。图形方式允许控制屏幕像素，因而能生成许多可见的效果不同的字符集。

当然，文本方式有它自己的优点，因为字符出自于字符表而不是当场时点来构造的。所以对于显示文字，文本方式快得多。文本方式占用内存也少，因此用户可以在屏幕上立即建立和显示几页文件。文本方式可以产生闪烁字符，这是在图形方式中没有的特性。最后，文本方式可选择更多的色彩。

**入口参数:**

调用前，需设置：

- AH = 00H** 指出置显示方式功能调用。
- AL =** 设置此寄存器以指出所希望的显示方式，定义如下：
- 00H** 对应彩色 / 图形适配器的文本方式。这是一个中分辨率黑白方式，每行 40 个字符，每屏 25 行。
  - 01H** 对应彩色 / 图形适配器的文本方式。支持 16 色的中分辨率方式，每行 40 个字符，每屏 25 行。
  - 02H** 对应彩色 / 图形适配器的文本方式。这是高分辨率黑白方式，每行 80 个字符，每屏 25 行。
  - 03H** 对应彩色 / 图形适配器的文本方式。支持 16 色的高分辨率方式，每行 80 个字符，每屏 25 行。
  - 04H** 对应彩色 / 图形适配器的图形方式。这是中分辨率方式 ( $320 \times 200$  像素)，支持 4 种色彩。
  - 05H** 对应彩色 / 图形适配器的图形方式。这是中分辨率 ( $320 \times 200$  像素) 黑白方式。
  - 06H** 对应彩色 / 图形适配器的图形方式。这是高分辨率 ( $640 \times 200$  像素)，黑白方式。
  - 07H** 对应单色适配器的文本方式。这是高分辨率，黑白方式，每行 80 个字符，每屏 25 行。
  - 08H** 对应 PCjr 的图形方式。这是低分辨方式 ( $160 \times 200$  像素)，支持 16 种色彩。
  - 09H** 对应 PCjr 的图形方式。这是中分辨方式 ( $320 \times 200$  像素)，支持 16 种色彩。
  - 0AH** 对应 PCjr 的图形方式。这是高分辨方式 ( $640 \times 200$  像素)，在 PCjr 上支持 4 种色彩。
  - 0DH** 对应增强型图形适配器的图形方式。这是中分辨方式 ( $320 \times 200$  像素)，支持 16 种色彩。
  - 0EH** 对应增强型图形适配器的图形方式。这是高分辨方式 ( $640 \times 200$  像素)，支持 16 种色彩。
  - 0FH** 对应增强型图形适配器的图形方式。这是超高分辨方式 ( $640 \times 350$  像素)，支持 4 种色彩。

**出口参数：**

无。

**参见：**

中断 10H，功能调用 0FH——取当前显示方式。

**BIOS Int 10H**

**显示 I/O**

**功能调用 01H——设置光标类型**

**中断:**

10H 显示 I/O

**功能调用:**

01H 设置光标类型

**计算机:**

PC, PCjr, XT, Portable 和 AT

**说明:**

在所有文本方式中，硬件（显示器适配器卡）显示一个闪烁的光标，程序员无法停止这个光标的闪烁，但可以使用该功能调用改变光标的形状。

在图形方式中，没有常见的光标，如果想要光标，必须仿造一个。

所有在文本方式中显示的字符由点排列组成，彩色 / 图形适配器生成的每个字符使用 8 行点（顶部是 0 行，底部是 7）；单色适配器用 14 行（0 行到 13 行）。

按缺省所设置，彩色 / 图形适配器显示的光标由第 6 和第 7 行的所有点组成。由单色适配器显示的缺省光标使用第 12 行和 13 行。可以用这个功能调用改变组成光标点的行数。

用此功能调用，依次在 CH 和 CL 寄存器中指定光标的开始和结束行，就可以使用全部或大部分行生成一个较大的光标。可以使一个水平行上的光标移动到字符的中间或顶部位置，或者指定光标的开始行大于结束行使光标环绕于字符上下，形成双光标。

**入口参数:**

调用前，需设置：

AH = 01H 指出设置光标形状功能调用。

CH 设置该寄存器的第 0 位到第 4 位，指明光标的开始行，对于彩色 / 图形适配器，允许值为 0 到 7，对于单色适配器允许值是 0 到 13。对于彩色 / 图形适配器，缺省值是 6，对于单色适配器，缺省值是 12。

IBM 文件告戒应设置此寄存器的第 5 位，或者使光标反复闪烁，或者使光标消失。实际上，如果把 CH 寄存器设置为 20H（第 5 位置“1”，其余的位都置“0”），光标将消失。

CL 设置此寄存器的第 0 位到第 4 位，指明光标的结束行。对于彩色 / 图表适配器允许值为 0 到 7，对于单色适配器允许值是 0 到 13。对于彩色 / 图形适配器，缺省值是 7；对于单色适配器，缺省值是 13。

**出口参数:**

无。

**参见:**

中断 10H，功能调用 03H——读取光标位置。

**BIOS Int 10H**

**显示 I/O**

**功能调用 02H——设置光标位置**

**中断:**

10H 显示 I/O

**功能调用:**

02H 设置光标位置

**计算机:**

PC, PCjr, XT, Portable 和 AT

**说明:**

该功能调用能够设置显示光标的位置。该功能调用以文本和图形两种方式工作，尽管在图形方式中没有可见的光标。图形方式光标（虽然不可见）将自动移到屏幕的下一个字符显示的位置。

在 DH 和 DL 寄存器中规定使用字符坐标（行和列）来设置光标位置。在屏幕左上角的字符位置是 0, 0 (0 行, 0 列)，此字符坐标系统用于文本方式和图形方式，但不能在图形方式中用于请求像素坐标。屏幕上的行数和列数取决于当前的显示方式。

对于文本方式中的多个显示项，每页都存在着一个与此相关的当前光标位置，因此，需要指定页数（40 列方式为 0 到 7，80 列方式为 0 到 3）。在图形方式中，仅有一页（0 页）。

**入口参数:**

调用前，需设置：

AH = 02H 指出设置光标位置功能调用。

DH 该寄存器指明光标的行位置（行数），0 行是字符的顶行，24 行是底行。

DL 设置此寄存器指明光标的列位置（列数），0 列是字符的最左边一列，可用列的数目取决于显示的方式。

BH 该寄存器指明正在设置光标的显示页数，在 40 列方式中，可用 0 页到 7 页，在 80 列方式中，可用 0 页到 3 页。在图形方式中，总是把该寄存器设置成 0。

**出口参数:**

无。

**参见:**

中断 10H，功能调用 03H——读取光标位置。

**程序实例:**

下述每个程序均使用了功能调用 02H，在当前显示页上移动光标。光标被随机地移动了 100 次位置时，在屏幕上写出一个“X”。

**Pascal Usage Example:**

```

{set Cursor Position ($10)}
PROGRAM set_cursor_position;

CONST
  video_io = $10;
  cursor_position = 2;

TYPE
  regpack = RECORD
    CASE integer OF
      1:(ax,bx,cx,dx,bp,si,di,ds,es,flags:integer0);
      2:(al,ah,bl,bh,cl,ch,dl,dh:byte);
    END;
  VAR
    regs:regpack;
    i:integer;
  PROCEDURE set_cur_pos(row,col:integer);
  BEGIN
    WITH regs DO
      BEGIN
        dh := row;
        dl := col;
        bh := 0;
        ah := cursor_pos;
        intr(video_io,regs);
      END;
    END;

BEGIN
  FOR i:=0 TO 100 DO
    BEGIN
      set_cur_pos(random(24),random(80));
      write(' X ');
    END;
  set_cur_pos(23,0);
END.

```

#### C Usage Example:

```
/ *
```

```

* Set Cursor Position(0x10)
* /

#include <dos.h>
#include <stdio.h>
union REGS inregs,outregs;

double drand()
{
    return (double)rand() / 32767;
}

set_cur_pos(row,col)
int row;
eint col;
{
    integs.h.dh = row;
    integs.h.dl = col;
    integs.h.bh = 0;
    integs.h.ah = 2;
    int86(0x10,&inregs,&outregs);
}

main()
{
    int i;

    for(i = 0;i <= 1000; ++i)
        set_cur_pos((short)(drand() * 23),(short)(drand() * 79));
        printf(" X ");
    }
    set_cur_pos(23,0);
}

```

**BIOS Int 10H  
显示 I/O  
功能调用 03H——读取光标位置**

**中断:**

## 10H 显示 I/O

功能调用:

03H 读光标位置

计算机:

PC, PCjr, XT, Portable 和 AT

说明:

功能调用 01H 和 02H 设置光标的类型和位置，本功能调用取光标的当前位置和类型。象其它两个功能调用一样，必须规定所要相找的显示页数。

入口参数:

调用前，需设置:

AH = 03H 指出“读光标位置”功能调用。

BH 此寄存器指明正在获取信息的显示页。在40列方式中，可用0到7页，在80列方式中，可用0页到3页，在图形方式中，这个寄存器应当总是设置成“0”。

出口参数:

返回后，设置为:

DH 这个寄存器包含光标的当前行数，0行是字符的顶行，24行是底行。

DL 该寄存器包含光标的当前列数，0列是屏幕上字符的最左列，可用列的数目取决于显示的方式。

CH 该寄存器指明光标位置处光标图形的开始行。该寄存器中的数目是光标开始所在的点的行数。（彩色/图形适配器为0到7，单色适配器为0到13。）

CL 该寄存器指明光标位置处光标图形的结束行。该寄存器中的数目是光标结束所在的点的行数。

参见:

中断 10H，功能调用 01H——设置光标类型。

中断 10H，功能调用 02H——设置光标类型。

## BIOS Int 10H

显示 I/O

功能调用 04H——读取光笔位置

中断:

10H 显示 I/O

功能调用:

04H 读取光笔位置

计算机:

PC, PCjr, XT, Portable 和 AT

说明:

该功能调用检查光笔是否已在屏幕上选择了位置。如果已经使用了光笔，则该功能调用返回光笔选择的坐标。在文本方式（选择字符的行与列）和图形方式（选择点的光栅行和象素的列）均返回坐标。在图形方式中，返回的坐标反映当前的图形方式。

此功能调用还可以复位光笔，因此它可以反复用来选择显示屏幕上的点。

#### 入口参数：

调用之前，需设置：

AH=04H 指出“置光笔位置”功能调用。

#### 出口参数：

返回后，设置为：

AH 该寄存器中的数值指明光笔是否已在显示屏幕上选择了一个点。

0 光笔还没有被触发。BX，CX和DX中所列的坐标值是无效的。

1 光笔已被触发，坐标在BX，CX和DX中列出。

DH 如果在文本方式中，该寄存器列出被光笔选择的字符行数。行的数值从0（顶行）到24（底行）。

DL 如果在文本方式中，这个寄存器列出由光笔选择的字符列数。最左边的列是0列；根据所选择的文本方式，最右边的列是39或79列。

CH 如果在图形方式中，该寄存器存放被选择点对应的光栅行。光栅行可以返回0（顶行）到199。

BX 如果在图形方式中，则该寄存器存放被选择点的象素列。列的数目随图形方式而变化。

#### 其它要求：

实际上该功能在AH寄存器返回“0”，指出光笔还未使用，BX，CX和DX寄存器中的内容无效。如果这些寄存器包含重要的信息，则在调用这个功能调用之前，应该保存这些内容。

### BIOS Int 10H

#### 显示I/O

#### 功能调用 05H——设置当前显示页

#### 中断：

10H 显示I/O

#### 功能调用：

05H 设置当前显示页

#### 计算机：

PC, PCjr, XT, Portable 和 AT

#### 说明：

在生成屏幕图象时，彩色/图形适配器包括16K字节可用的随机访问存储器(RAM)。在图形方式中，该存储器使屏幕具有 $640 \times 200$ 象素的分辨率。(每个象素要占存储器的一位，表明该象素用了还是没用。)然而在文本方式中，彩色/图形适配器需要

的内存少，每个字符位置仅仅对应 2 个字节，一个字节对应字符，另一个对应字符属性。(对于中分辨率方式，需要 2K 字节，40 字符 × 25 行，高分辨率方式要 4K 字节，80 字符 × 25 行。)

因为在文本方式中显示一屏并不需要用彩色 / 图形适配器的全部存储器，所以适配器把它的存储器分解成独立的页，因而可以在往一页写信息的同时，使另一页在屏幕上显示出来。通过转换这些页，可以瞬时地更新屏幕上的信息。

在 40 列文本方式中（方式 0 和 1），有 8 个显示页，0 到 7 页。在 80 列文本方式中（方式 2 和 3），有 4 页，0 到 3。此功能调用允许选择当前显示页，即将这一页的信息显示在屏幕上。

在 40 列和 80 列两种方式中，显示页 0 在彩色 / 图形适配器 RAM 的开头，根据方式不同，其余的页随在 2K 或 4K 字节区间中，缺省页总是 0。

#### 入口参数：

调用之前，需设置：

AH = 05H 指出“设置当前显示页”功能调用。

AL 把该寄存器设置成所希望的当前显示页（显示在屏幕上）。在 40 列方式中（0 和 1），可以用 0 页到 7 页。在 80 列方式中（方式 2 和 3），可以用 0 页到 3 页。

#### 出口参数：

在屏幕上显示新的显示页中字符。

#### 参见：

中断 10H，功能调用 00H——设置显示方式。

### BIOS Int 10H

#### 显示 I/O

#### 功能调用 06H——当前显示页上滚

#### 中断：

10H 显示 I/O

#### 功能调用：

06H 当前显示页上滚

#### 计算机：

PC, PCjr, XT, Portable 和 AT

#### 说明：

该功能调用允许在屏幕上定义一个窗口（当前显示页），并且使窗口中的信息不产生滚动。对应的功能调用（07H）允许向下滚动。

通过规定窗口的左上角和右下角的字符坐标，指明窗口的大小和位置。字符位置在屏幕左上角以 0 行 0 列为开始。在 40 列方式中，屏幕上的最后一个字符位置是 24 行 39 列。而在 80 列方式中，是 24 行 79 列。在图形方式中，如果适配器是在 80 列文本方式，则坐标被解释为字符坐标（即它们被解释成字符坐标，不是象素坐标）。

通过在窗口底部插入空行使信息向上滚动，并相应地向上移动其余的行，滚出窗口顶

部的信息就消失了。规定空行的属性（例如，在彩色方式中规定色彩，或者在文本方式中规定是否使用反相显示）作为输入参数发送给该功能调用。

#### 入口参数：

调用之前，需设置：

AH=06H 指出“当前显示页上滚”功能调用。

AL 设置此寄存器指明滚动的文本行数。如果该寄存器中放置“0”，则整个窗口为空白。

CH 设置此寄存器指明窗口左上角的行位置（从0到24）。

CL 该寄存器指明窗口左上角的列位置（在40列文本方式中，从0到39，在80列文本或图形方式中，从0到79）。

DH 该寄存器指明窗口右下角的行位置（从0到24）。

DL 该寄存器指明窗口右下角的列位置（在40列文本方式中，从0到39，在80列文本或图形方式中，从0到79）。

BH 此寄存器指明加到窗口底部的空行显示属性。

对于彩色/图形适配器，属性字节如下所示：

位
7 6 5 4 3 2 1 0

位	说明
7	该位置“1”用于字符的闪烁，置“0”时不闪烁。
6-4	背景色，可用的是：
6 5 4	颜色
0 0 0	黑
0 0 1	蓝
0 1 0	绿
0 1 1	青
1 0 0	红
1 0 1	绛
1 1 0	褐
1 1 1	浅灰
3	前景字符的亮度，这位置1，前景字符为高亮度，置0，前景字符为正常亮度。
2-0	前景色，可使彩色与背景色不同。

对于单色适配器，属性字节的格式与通常的一致，即第7位是闪烁位，第3位是亮度位。单色适配器不支持彩色，但是适当地设置彩色位，也可以支持下列单色属性：

正常的字符 设置背景色为黑(000), 前景色为白(111)产生。  
字符反显 设置背景色为白(111), 前景色为黑(000)产生。  
字符不可见 把前景色和背景色都设置为黑(000)产生。  
字符带下划线 设置背景色为黑(000), 前景色为蓝(001)产生。

#### 出口参数:

屏幕上的窗口适当地向上滚动。

#### 参见:

中断 10H, 功能调用 07H——当前显示页下滚。

BIOS Int 10H

显示 I/O

功能调用 07H——当前显示页下滚

#### 中断:

10H 显示 I/O

#### 功能调用:

07H 读取光笔位置

#### 计算机:

PC, PCjr, XT, Portable 和 AT

#### 说明:

该功能调用除了在窗口中的信息向下滚动和在窗口顶部插入空行之外, 其它与功能调用 06H 一样。

#### 入口参数:

调用之前, 需设置:

AH = 07H 指出“当前显示页下滚”功能调用。

AL 此寄存器指明滚动的文本行数。如果该寄存器置“0”, 则整个窗口为空白。

CH 该寄存器指明窗口左上角的行位置 (从0到24)。

CL 该寄存器指明窗口左上角的列位置 (在40列文本方式中, 从0到39, 在80列文本或图形方式中, 从0到79)。

DH 该寄存器指明窗口右下角的行位置 (从0到24)。

DL 该寄存器指明窗口右下角的列位置 (在40列文本方式中, 从0到39, 在80列文本或图形方式中, 从0到79)。

BH 该寄存器指明加到窗口部的空行的显示属性。对于彩色 / 图形适配器, 这个属性字节如下所示:

位	7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---	---

位	说明
7	这位置“1”用于字符的闪烁,置“0”字符不闪烁.
6-4	背景色可用的是:
6 5 4	颜色
0 0 0	黑
0 0 1	蓝
0 1 0	绿
0 1 1	青
1 0 0	红
1 0 1	绛
1 1 0	褐
1 1 1	浅灰
3	前景字符的亮度.这位置 1,前景字符为高亮度,置 0,,前景字符为正常亮度.
2-0	前景色.可使彩色与背景色相同.

对于单色适配器,属性字节的格式与通常的一致,即第 7 位是闪烁位,第 3 位是灰度位。单色适配器不支持彩色,但是适当地设置彩色位,也可以支持下列单色属性:

正常的字符 设置背景色为黑 (000), 前景色为白 (111) 产生。

字符反显 设置背景色为白 (111), 前景色为黑 (000) 产生。

字符不可见 把前景色和背景色都设置为黑 (000) 产生。

字符带下划线 设置背景色为黑 (000), 前景色为蓝 (001) 产生。

其余的色彩组合对于单色适配器是无效的。

#### 出口参数:

屏幕上的窗口适当地下滚。

#### 参见:

中断 10H, 功能调用 06H——当前显示页上滚。

## **BIOS Int 10H**

**显示 I/O**

**功能调用 08H——读取字符和属性**

**中断:**

10H 显示 I/O

**功能调用:**

08H 读取字符和属性

**计算机:**

PC, PCjr, XT, Portable 和 AT

**说明:**

该功能调用从屏幕（或从另一个显示页）当前光标位置读取字符。该功能调用和功能调用 02H（设置光标位置）可以一起使用，用来读屏幕上任何位置处的信息。

此功能调用对于文本的图形方式均适用。在图形方式中，该功能调用把光标处的字符与产生字符的 ROM 表相比较，然而在该表中仅仅保存开始的 128 个字符。功能调用要取对应于第二个 128 个字符的代码，则必须预置中断向量 1FH，使之指向 1K 字节表的存储单元，该表列出了那些代码（BIOS 没有定义缺省表）。如果此功能调用在这两个表之中找到字符，则返回对应于该字符的 ASCII 码。如果没有与该字符对应的 ASCII 码，则该功能调用返回值 0。

在文本方式中，可以读 256 个字符中的任意一个。必须指定（想要的信息）在哪一显示页（每个显示页都有它自己的光标）。功能调用返回对应于字符的 ASCII 码和属性字节。

**入口参数:**

调用之前，需设置：

AH = 08H 指出“读取字符和属性”功能调用。

BH 在文本方式中，设置此寄存器指明正在读的显示页，在 40 列方式中，要用 0 到 7 页，在 80 列方式中，可以用 0 到 3。在图形方式中，该寄存器总是为 0。

**出口参数:**

返回后，设置为：

AL 此寄存器存放对应于现行光标位置字符的 ASCII 码。如果没有对应于字符的 ASCII 码，则该寄存器将被设置为 0。

AH 在图形方式中，此寄存器包含的信息无意义。在文本方式中，此寄存器包含对于正在读取的字符的属性字节。

对于彩色 / 图形适配器，属性字节具有下列格式：

位

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

位	说明
7	这位置“1”用于字符的闪烁,置“0”字符不闪烁.
6~4	背景色.可用的是:
6 5 4	颜色
0 0 0	黑
0 0 1	蓝
0 1 0	绿
0 1 1	青
1 0 0	红
1 0 1	紫
1 1 0	褐
1 1 1	浅灰
3	前景字符的亮度.这位置 1,前景字符为高亮度,置 0,,前景字符为正常亮度.
2~0	前景色.可使彩色与背景色相同.

对于单色适配器,属性字节的格式与通常的一致,即第 7 位是闪烁位,第 3 位是灰度位。单色适配器不支持彩色,但是适当地设置彩色位,也可以支持下列单色属性:

正常的字符 背景色的黑色 (000), 前景色的白色 (111).

字符反显 背景色为白 (111), 前景色为黑 (000).

字符不可见 前景色和背景色都是黑色 (000).

字符带下划线 背景色为黑色 (000), 前景色为蓝色 (001).

参见:

中断 10H, 功能调用 02H——设置光标位置。

中断 10H, 功能调用 09H——写字符和属性。

**BIOS Int 10H**

**显示 I/O**

**功能调用 09H——写字符和属性**

**中断:**

10H 显示 I/O

### 功能调用:

09H 写字符和属性

### 计算机:

PC, PCjr, XT, Portable 和 AT

### 说明:

该功能调用从当前光标位置开始把一个或多个字符写到屏幕上（或其它显示页）。在文本方式中，可以设置被写字符的属性位。在图形方式中，可以设置字符的色彩。

用此功能调用，可以把某字符的多个拷贝写到屏幕上，这与功能调用 0EH 不同。在文本方式中，如果需要，这些字符可以移到下一行。在图形方式中，没有这样的特性。在文本和图形两种方式中，无论写多少字符光标均不移动。

与功能调用 08H 的情况一样，在图形方式中，该功能调用 ROM 字符表产生开始的 128 个字符。要显示超过 128 个字符的内容，必须定义 1K 字节的表，这个表列出了这些字符和它们的代码，然后预置中断向量 1FH 使之指向该表（BIOS 没有定义缺省表）。在文本方式中，可以显示任意的 256 个字符。

### 入口参数:

调用之前，需设置：

AH = 09H 指出“写字符和属性”功能调用。

BH 在文本方式中，该寄存器指明正在写的显示页。在 40 列方式中，可以用 0 到 7 页，在 80 列方式中，可以用 0 到 3 页。在图形方式中，该寄存器总是 0。

AL 此寄存器放要写字符对应的 ASCII 码。

BL 在文本方式中，此寄存器为写字符的属性。属性位格式如下：

位	说明
7	这位置“1”用于字符的闪烁，置“0”字符不闪烁。
6-4	背景色 可用的是：
6 5 4	颜色
0 0 0	黑
0 0 1	蓝
0 1 0	绿
0 1 1	青
1 0 0	红
1 0 1	绛
1 1 0	褐
1 1 1	浅灰
3	前景字符的亮度。这位置 1，前景字符为高亮度，置 0，前景字符为正常亮度。
2-0	前景色。可使彩色与背景色相同。

位

说明

7 这位置“1”用于字符的闪烁，置“0”字符不闪烁。

6-4 背景色 可用的是：

6 5 4 颜色

0 0 0 黑

0 0 1 蓝

0 1 0 绿

0 1 1 青

1 0 0 红

1 0 1 绛

1 1 0 褐

1 1 1 浅灰

3 前景字符的亮度。这位置 1，前景字符为高亮度，置 0，前景字符为正常亮度。

2-0 前景色。可使彩色与背景色相同。

对于单色适配器，属性字节的格式与通常的一致，即第 7 位是闪烁位，第 3 位是灰度位。单色适配器不支持彩色，但是适当地设置彩色位，也可以支持下列单色属性：

正常的字符 设置背景色为黑 (000)，前景色为白 (111) 产生。

字符反显 设置背景色为白 (111)，前景色为黑 (000) 产生。

字符不可见 把前景色和背景色都设置为黑 (000) 产生。

字符带下划线 设置背景色为黑 (000)，前景色为蓝 (001) 产生。

其余的色彩组合对于单色适配器都是无效的。

在图形方式中，在该寄存器中内容规定字符的前景色，对于彩色 / 图形适配器上的中分辨率图形方式（方式 4 和 5），用 1 到 3 中的数值在当前调色板中选择三个前景色之一。（参照功能调用 0BH 的说明，如何设置彩色调色板。）如果把该寄存器的第 7 位也设置成“1”，功能调用对这里规定的彩色位和在显示页上的当前景色位执行按位加 (XOR) 操作，这就保证了所得到的字符与原来的色彩不同。

在彩色 / 图形适配器的高分辨率方式中（方式 6），把此寄存器设置成“1”，写出的是白色字符，设置成“0”，写出的是黑色字符。

**CX** 此寄存器指明将字符和属性写到屏幕上的次数。在图形方式中，字符只能写在当前行内。在文本方式中，字符可以换行连续写。因此，可写的字符数目限制在一个显示页的字符数目之内（文本方式时），或者限制在一行字符数目内（图形方式时）。

即使把多个字符写到屏幕上，光标还是停留在它原来的位置。如果把该寄存器设置成“0”，该功能调用连续地写字符。

#### 出口参数：

一个字符或多个字符显示在屏幕上（或者写到显示页中）。

#### 参见：

中断 10H，功能调用 02H——设置光标位置。

中断 10H，功能调用 08H——读取字符和属性。

中断 10H，功能调用 0AH——写字符。

中断 10H，功能调用 0EH——以电传方式写字符。

#### BIOS Int 10H

#### 显示 I/O

#### 功能调用 0AH——写字符

#### 中断：

10H 显示 I/O

#### 功能调用：

0AH 写字符

#### 计算机：

PC, PCjr, XT, Portable 和 AT

#### 说明：

该功能调用除了不允许在文本方式时置属性字节外，其它同功能调用 09H 写字符和属性一样，但当前属性字节保持不变。

在图形方式中，同功能调用 09H 一样，也可提供字符的色彩。

#### 入口参数：

调用之前，需设置：

AH=0AH 指出“写字符”功能调用。

AL 该寄存器放要写字符对应的ASCII码。

BH 在文本方式中，该寄存器指明正在写的显示页。在40列方式中，可以用0到7页。在80列方式中，可以用0到3页。在图形方式和方式7（对应于单色显示适配器的文本方式）中，可以忽略该寄存器。

BL 在图形方式中，用该寄存器规定字符的前景色。对于彩色／图形适配器上的中分辨率图形（方式4和方式5），用1到3中间的一个数值在当前调色板中选择三个前景色之一（参照功能调用0BH的描述，看如何设置彩色调色板）。

如果把该寄存器第7位也置“1”，该功能调用对选择的色彩的象素彩色位和该显示页上的前景色进行按位加（XOR，这就保证了新得到的字符与原来的色彩不同）。

在彩色／图形适配器的高分辨率方式中（方式6），此寄存器置“1”是以白色写字符，置“0”，则是以黑色写字符。

在文本方式中，可以忽略该寄存器。

CX 此寄存器指明把字符和属性写到屏幕上的次数。在图形方式中，写字符时不能自动换行，在文本方式中，则可以。因此，能写的字符个数限制在一个显示页的字符数目内（文本方式），或者限制在一行的字符数目内（图形方式）。

即使把多个字符写到屏幕上，光标还是停留在原来的位置。如果把此寄存器置“0”，该功能调用将连续不断地写字符。

#### 出口参数：

一个字符或多个字符显示在屏幕上（或者写到显示页中）。

#### 参见：

中断10H，功能调用02H——设置光标位置。

中断10H，功能调用08H——读取字符和属性。

中断10H，功能调用09H——写字符和属性。

中断10H，功能调用0EH——以电传方式写字符。

**BIOS Int 10H**

**显示 I/O**

**功能调用 0BH——设置彩色调色板**

**中断:**

10H 显示 I/O

**功能调用:**

0BH 设置彩色调色板

**计算机:**

PC, PCjr, XT, Portable 和 AT

**说明:**

彩色 / 图形适配器处于中分辨率彩色图形方式 (方式 4) 时, 能显示 8 种不同的色彩, 但在同一时刻只能用 4 种色彩。每一 4 种色彩的组合称为一个调色板。

有两种与彩色 / 图形适配器有关的标准调色板。0 号调色板由这些色彩组成:

色彩值	颜色
0	与背景色相同(缺省为黑色)
1	绿(2)
2	红(4)
3	褐(6)

1 号调色板由这些色彩组成:

色彩值	颜色
0	与背景色相同(缺省为黑色)
1	青(3)
2	绛(5)
3	浅灰(7)

在同一时间内, 只有一个调色板有效。

该功能调用用来选择屏幕颜色的调色板，以设置背景色。背景色可以是 16 种色彩中的任意一种，而在文本方式中，它设置边框色彩。

对应每个调色板数目中所列的色彩值即是用功能调用 0CH（写点）把象素写到屏幕上时所用的数字。使用功能调用 0DH（读点）时，即返回这些数字。所以认为这些数字是调色板中的位置参照点，而不是唯一作为标识色彩的数字。

例如，当 0 号调色板有效时，写一个点，其色彩值是 1，实际上是写一个绿色的点。当调色板有效时，同样的操作写了一个青色的点。

在色彩右侧括号里的数字都是象素色彩代码。这些数字分别标识与彩色 / 图形适配器对应的 16 种标准色彩（8 种正常色彩和 8 种增强色彩）。当选择对应调色板的背景色（或在文本方式中设置字符属性时，使用这些象素色彩代码规定背景色和前景色。象素色彩代码如下所示：

代码	颜色
00H	黑
01H	蓝
02H	绿
03H	青
04H	红
05H	绛
06H	褐
07H	浅灰
09H	浅蓝
0AH	浅绿
0BH	浅青
0CH	浅红
0DH	浅绛
0EH	黄
0FH	白

在图形方式中，选择的前景色在写文本字符时，总是有效的。例如，在图形方式中，

如果使用功能调用 0AH (写字符) 或功能调用 0EH (以电传方式写字符) 只能规定字符的色彩 (前景色), 背景色则被假定为正在使用的调色板所对应的背景色。

#### 入口参数:

调用之前, 需设置:

AH=0BH 指出“设置彩色调色板”功能调用。

BH 调色板色彩ID。此寄存器中的数值指明要执行的功能调用操作。对于彩色 / 图形适配器 (CGA), 该寄存器中的数值可以是:

0 在图形方式中, 该功能调用将设置对应于当前调色板的前景色。在文本方式中, 该功能调用将设置对应于当前调色板的背景色。在文本方式中, 该功能调用设置边框色。必须在 BL 寄存器中放置对应于背景色或边框色的代码。

1 此功能调用将选择一调色板。必须在 BL 中放置调色板号。此操作仅用于图形方式。

BL 如果正在选择背景色或边框色 (BH=0), 则在该寄存器中设置色彩像素代码 (从 0 到 0FH)。如果正在选择调色板, 在此寄存器中放置调色板号。

#### 出口参数:

无。

**BIOS Int 10H**

**显示 I/O**

**功能调用 0CH——写点**

#### 中断:

10H 显示 I/O

#### 功能调用:

0CH 写点

#### 计算机:

PC, PCjr, XT, Portable 和 AT

#### 说明:

此功能调用仅用于图形方式。它把图形显示的最小单位 (点或象素) 写到屏幕上。因此该功能调用规定用于图形方式操作, 所以必须指明由光栅行和象素列规定的那个点的位置。

#### 入口参数:

调用之前, 需设置:

AH=0CH 指出“写点”功能调用。

AL 此寄存器指明点的调色板色彩代码。在彩色 / 图形适配器的中分辨率方式中 (方式 4), 可以规定数值 1 到 3, 用以选择与当前调色板相应的三种前景色之一。在彩色 / 图形适配器的高分辨率方式中 (方式 6), 可以选

择 0 或 1，用来指明相应的黑色和白色。

对可用于PCjr和增强型图形适配器的16色图形方式，可以规定16种象素色彩代码之一。

当设置色彩时，如果寄存器的第7位置“1”，则该功能调用对选择的色彩的象素彩色位和当前点的彩色位执行按位加（XOR）操作，这就保证了该点的色彩与原来不同。

**DX** 此寄存器用以指明要写点的行（光栅行）。可用的数值为0（顶行）到199。

**CX** 该寄存器用以指明要写点的列。屏幕上的列数取决于所用的图形分辨率，在中分辨率中（方式4，5，9和13），列数从0（最左边的列）到319。在高分辨率方式中（方式7，10和14），列数从0到639。

#### 出口参数：

点显示在屏幕上。

#### 参见：

中断10H，功能调用0BH——设置调色板。

中断10H，功能调用0DH——读点。

**BIOS Int 10H**

**显示I/O**

**功能调用0DH——读点**

#### 中断：

10H 显示I/O

#### 功能调用：

0DH 读点

#### 计算机：

PC，PCjr，XT，Portable 和 AT

#### 说明：

该功能调用类似于功能调用0CH（写点），本功能调用以读点代之写点。规定一个点的位置，该功能调用即返回此点的当前色。象功能调用0CH一样，该功能调用操作仅用于图形方式。

#### 入口参数：

调用之前，需设置：

AH=0DH 指出“读点”功能调用。

**DX** 该寄存器指明要读点的行（光栅行）。可用数值为0（顶行）到199。

**CX** 该寄存器指明要读点的列。屏幕上的列数取决于所用的图形分辨率。在中分辨率方式中（方式4，5，9和13），列数从0（最左边的列）到319。在高分辨率方式中（方式7，10和14），列数从0到639。

#### 出口参数：

返回后，设置为：

AL 该寄存器中的数值是所读点的调色板色彩代码，在彩色 / 图形适配器的中分辨率方式中（方式 4），是 1 到 3 中间的一个值，以此指明用于当前调色板的四种色彩之一。在彩色 / 图形适配器的高分辨率方式（方式 6）中，此数值是 0 或 1，分别指明黑色和白色。对用于 PCjr 和增强型图形适配器的 16 种图形方式，是 16 种象素色彩代码中的一个。

参见：

中断 10H，功能调用 0BH——设置彩色调色板。

中断 10H，功能调用 0CH——写点。

**BIOS Int 10H**

**显示 I/O**

**功能调用 0EH——以电传方式写字符**

中断：

10H 显示 I/O

功能调用：

0EH 以电传方式写字符

计算机：

PC, PCjr, XT, Portable 和 AT

说明：

该功能调用把一个字符写到屏幕（当前显示页）当前光标位置处，将屏幕作为一般的电传机终端看待。如果写完了一行，则绕到下一行开始位置继续写字符。若是已写完了一屏最下面的一行，则整个屏幕向上滚动一行，于是又可以从这新的一行的起始位置继续写字符。

该功能调用类似于功能调用 09H（写字符和属性）与 0AH（写字符），一次调用不能写多个字符，也不能在文本方式或设置属性字节。然而，每写一个字符时，它自动地把光标移到下一个字符位置，这个特性在功能调用 09H 和 0AH 中是不具备的。

除了 4 个字符之外，使用这个功能调用写的每个字符都认为是可显示字符。下列 4 个字符被当作命令而不是可显示字符：

ASCII	字符
7	Beep(响铃)
8	Backspace(退格)
10	Line feed(换行)
13	Carriage return(回车)

在文本方式中发出该功能调用时，不能设置属性字节。此功能调用假设新的字符同以前字符具有一样的属性。在图形方式中，必须规定字符的色彩。

在图形方式中，此功能调用使用 ROM 字符表产生开始的 128 个字符，要想显示超过这 128 个的任何字符，就必须生成一个列出这些字符以及它们的编码的 1K 字节的表，来预置指向此表的中断向量 1FH。在文本方式中，显示超出这 128 个的任何字符无须生成此表。

#### 入口参数：

调用之前，需设置：

AH = 0EH 指出“以电传打字方式写字符”功能调用。

AL 该寄存器存放对应于要写字符的 ASCII 码。

BL 该寄存器规定字符的前景色。对于彩色 / 图形适配器上的中分辨率图形（方式 4 和方式 5）用 1 到 3 中间的数值在当前调色板中选择三种景色之一（参照功能调用 0BH 的说明，看如何设置彩色调色板）。如果把该寄存器的第 7 位置“1”，此功能调用对规定色彩的色彩位和在显示页上的当前前景色彩位执行按位加（XOR）操作，这样得到的字符和原来的色彩不同。在彩色 / 图形适配器的高分辨率方式中（方式 6），该寄存器置“1”是写白色字符，置“0”是写黑色字符。

在文本方式中，此寄存器置“0”。

#### 出口参数：

字符显示在屏幕上（或写到显示页中）。

#### 参见：

中断 10H，功能调用 02H——设置光标位置。

中断 10H，功能调用 08H——读取字符和属性。

中断 10H，功能调用 09H——读取字符。

中断 10H，功能调用 0AH——写字符。

### BIOS Int 10H

#### 显示 I/O

#### 功能调用 0FH——取当前显示方式

#### 中断：

10H 显示 I/O

#### 功能调用：

0FH 取当前显示方式

#### 计算机：

PC, PCjr, XT, Portable 和 AT

#### 说明：

该功能调用返回有关当前显示方式的信息。返回的信息是当前方式号，在该方式下每

行的字符数及当前显示页数。

**入口参数:**

调用之前，需设置：

AH = 0FH 指定“取当前显示方式”功能调用。

**出口参数:**

返回后，设置为：

AL 该寄存器放当前显示方式号，参见功能调用00H（设置显示方式）对可用方式及其它们号码的说明。

AH 该寄存器指明每行中文本字符数（80, 40或20）。

BH 此寄存器列出当前的显示页数（现在出现在屏幕上的页）。在40列方式中，可以用0到7页，在80列方式中，可以用0到3页，在图形方式中，该寄存器总是设置成“0”。

**参见:**

中断10H，功能调用00H——设置显示方式。

**BIOS Int 10H**

**显示 I/O**

**功能调用 13H——写字符串**

**中断:**

10H 显示 I/O

**功能调用:**

13H 写字符串

**计算机:**

AT

**说明:**

该功能调用仅能用于 AT 机。它能够把一串字符写到屏幕的指定位置上。使用该功能调用的任选项，可以把光标定位在字符串的开头或结尾处。在文本方式中，还可以规定对应于所有字符串属性字节，或者为每个字符提供单独的属性字节。在图形方式中，可以规定一个色彩字节，或者各种色彩字节。

如同功能调用0EH（以电传方式写字符）的情况一样，该功能调用把下列字符作为命令而不是作为可显示的字符：

ASCII	字符
7	Beep(响铃)
8	Backspace(退格)
10	Line feed(换行)
13	Carriage return(回车)

## 入口参数:

调用之前，需设置：

AH = 13H 指定“写字符串”功能调用。

AL 该寄存器指明放置光标的方式并规定属性，表示如下：

0 使用BL中规定的属性字节或前景色与字符串。光标留在DH, DL指定的坐标位置上。

1 使用BL中规定的属性字节或前景色。光标放置在字符串的结尾处。

2 设对应于每个字符的属性字节或者前景色紧跟在字符的后面（如〔字符，属性，字符，属性，…，字符，属性〕）写字符串。光标留在DH, DL规定的坐标位置上。

3 设对应于每个字符的属性字节或前景色紧跟在字符的后面，就象任选项2一样，写字符串。光标放置在字符串的结尾处。

ES : BP 此寄存器对指定要显示的字符串所在的存储区的起始位置。当AL被置“0”或置“1”时，该字符串只能放文本字符。当AL是2或3时，此字符串放字符（和它们属性字节）的交替序列（在图形方式中，是前景色）。关于属性字节和前景色的细节参见功能调用09H（写字符和属性）的说明。

CX 该寄存器指明字符串的字节数。

BH 在文本方式中，该寄存器指明正在写的显示页数。在40列方式中，可以用0到7页。80列方式中，可以用0到3页。在图形方式中，该寄存器应该总是设置成“0”。

BL 当AL置成“0”或“1”时，该寄存器放对应于字符串中所有字符属性字节（在图形方式中为前景色）。关于属性字节和前景色的说明，参见功能调用09H（写字符和属性）的说明。

DH 该寄存器指明显示字符串文本的起始行。0行是屏幕上字符的最顶行，24行是最底下一行。此文本坐标制适用于文本和图形两种方式中。

DL 此寄存器指明显示字符串文本的起始列。在40字符方式中，列的取值范围是0到39，在80列或图形方式中，是0到79列。

## 出口参数:

字符串显示在屏幕上（或者写到显示页中）。

## 参见:

中断10H，功能调用08H——读取字符和属性。

中断10H，功能调用09H——写字符和属性。

中断10H，功能调用0AH——写字符。

中断10H，功能调用0EH——以电传打字方式写字符。

## BIOS Int 11H

确定设备

中断:

11H 确定设备

计算机:

PC, PCjr, XT, Portable 和 AT

说明:

此中断用于查看一个连接在计算机上的标准硬件设备编码表。此表允许程序根据当前的设备类型以不同的方法操作。此中断最显著的用途是确定计算机当前是彩色 / 图形适配器还是单色适配器，从而调整程序的输出。

此中断返回的信息在系统启动时由 BIOS 确定并存放在内存十六进制地址 40:10 处。

入口参数:

调用该功能调用，发出中断 11H 即可，事先不必设置任何寄存器。

出口参数:

返回时，设置为:

AX 此寄存器放连接在计算机上的设备编码表。该设备字的意义如下:

位

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
----	----	----	----	----	----	---	---	---	---	---	---	---	---	---	---

位	说明
15-14	打印机数
13	对于 PGjr,如果有几台打印机,该位被置"1",对于其它机器,这位没有用.
12	有游戏适配器,这位置"1".对于 PGjr 这位总是置"1".
11-9	RS-232 串行端口数.
8	置"0"时,该位指出在计算机中当前是 DMA(直接存储器地址)芯片.对于 PGjr 这位总是置"1".

位	说明
7-6	<p>如果第 0 位置“1”，这几位才有意义，这两位组成的字段表示设置的软盘驱动器数。表示如下：</p> <ul style="list-style-type: none"> <li>00 1 个驱动器</li> <li>01 2 个驱动器</li> <li>02 3 个驱动器</li> <li>03 4 个驱动器</li> </ul>
5-4	<p>初始显示方式。表示如下：</p> <ul style="list-style-type: none"> <li>00 未用</li> <li>01 40 列文本方式(认为是彩色 / 图形适配器)。</li> <li>10 80 列文本方式(认为是彩色 / 图形适配器)。</li> <li>11 80 列文本方式(认为是单色适配器)。</li> </ul>
3-2	<p>早期的 IBM PC 系统板上只有 64K 字节的 RAM，这两位组成的字段表示系统板上的 RAM 量。表示如下：</p> <ul style="list-style-type: none"> <li>00 16K 字节</li> <li>01 32K 字节</li> <li>10 48K 字节</li> <li>11 64K 字节</li> </ul> <p>对于 XT，这几位无意义。</p>
1	<p>对于 PC,XT,Portable 和 PG Jr，这位无意义。对 AT 如果是用 80287 数学协处理器，则该位置“1”。</p>
0	<p>如果有软盘驱动器连接到计算机上，这位置“1”，第 7 位和第 6 位规定了驱动器数。如果这位为“0”，则没有连接任何软盘驱动器，并且第 7 位和第 6 位无意义。</p>

**参见:**

中断 12H——确定内存大小

**程序实例:**

下面每一个程序例子都使用了中断 11H 确定可用设备。此信息输出到屏幕上。

**Pascal Usage Example:**

```
{equipment Determination ($ 11)}  
PROGRAM equipment_determination;
```

**CONST**

```
equip_determination = $ 11;  
printer_mask = $ c0000;  
gamec_mask = $ 00c0;  
dma_mask = $ 1000;  
serial_mask = $ 0c00;  
dma_mask = $ 0100;  
drive_num_mask = $ 00c0;  
video_mask = $ 0030;  
mdp_mask = $ 0002;  
drive_mask = $ 0001;  
c040 = $ 0010;  
c080 = $ 0020;  
mono = $ 0030;
```

**TYPE**

```
regpack = RECORD  
  CASE integer OF  
    1:(ax,bx,cx,dx,bp,si,di,ds,cs,foags:integer);  
    2:(al,ah,bl,bh,cl,ch,dl,dh:byte);  
  END;
```

**VAR**

```
regs:regpack;
```

**BEGIN**

```
intr(equip_determination,regs);  
writeln(' Number of printers = ',  
      abs((regs.ax AND printer_mask) DIV 16384));  
IF (regs.ax AND gamec_mask) < > 0 THEN
```

```

        writeln( ' Game adapter present' )
ELSE
        writeln( ' Game adapter not present' );
writeln( ' Number of serial ports = ' ,
        (regs.ax AND serial_mask) DIV 512);
IF (regs.ax AND dma_mask)= THEN
        writeln( ' Dma present' )
ELSE
        writeln( ' Dma not present' );
IF (regs.ax AND drive_mask) < > 0 THEN
        writeln( ' Number of diskette drives installed = ' ,
        ((regs.ax AND drive_num_mask) DIV 64)+1)
ELSE
        writeln( ' No diskette drives installed' );
write( ' Initial videl mode = ' );
CASE(regs.ax AND video_mask) OF
        c040:writeln( ' CGA 40×25 B / W text' );
        c080:writeln( ' CGA 80×25 B / W text' );
        mono:writeln( ' monochrome 80×25 text' );
END;
IF (regs.ax AND ndp_mask) < > 0THEN
        writeln( ' Math coprocessor present' )
ELSE
        writeln( ' Math coprocessor not present' );
END.

```

#### C Usage Example:

```

/* *
 * Equipment Determination(0x11)
 */

```

```
#include <dos.h>
```

```
union REGS inregs,outregs;
```

```
union {
    unsigned axreg;
    struct {
        unsigned diskette:1;
```

```

        unsigned ndp :1;
        unsigned sys_ram :2;
        unsigned video_mode :2;
        unsigned dma :1;
        unsigned serial_ports :3;
        unsigned game_adapter :1;
        unsigned pcjr_printer :1;
        unsigned printer_num :2;
    } flags;
} equip;

main()
{
    int86(0x11,&inregs,&outregs);
    equip.axreg = outregs.x.ax;

    printf(" Number of printers = %d\n",
           equip.flags.printer_num);
    if (equip.flags.game_adapter != 0)
        printf(" Game adapter present\n");
    else printf(" Game adapter not present\n");
    printf(" Number of serial ports = %d\n",
           equip.flags.serial_ports);
    if (equip.flags.dma)
        printf(" DMA not present\n");
    else printf(" DMA present\n");
    if (equip.flags.diskette)
        printf(" Number of diskette drives installed = %d\n",
               equip.flags.drive_num+1);
    else printf(" No diskette drives installed\n");
    printf(" Initial video mode = ");
    switch (equip.flags.video_mode) {
        case 1: printf(" CGA 40×25 B / W text\n");
                  break;
        case 2: printf(" CGA 80×25 B / W text\n");
                  break;
        case 3: printf(" mono 80×25 text\n");
                  break;
        default:break;
    }
}

```

```
    }
    if (equip.flags.ndp)
        printf(" Math coprocessor present\n");
    else printf(" Math coprocessor not present\n");
```

## BIOS Int 12H

确定内存大小

中断:

12H 确定内存大小

计算机:

PC, PCjr, XT, Portable 和 AT

说明:

发出此中断, BIOS 将报告计算机中可用的随机存取存储器 (RAM) 空间, 可用的存储器是从地址 0 开始的连续存储器 (在存储区之间无任何间隔). 如当前最多是 640K 字节.

在 PC 中, BIOS 通过读取系统板上的开关来确定存储器空间. 如果开关设置有错, 该中断程序将返回错误数值.

在 PCjr 和 AT 中, BIOS 在加电自检 (POST—power-on self-test) 期间通过检查存储器来确定存储器的可用空间. PCjr 从这全部内存中减去 16K, 专门留作屏幕显示存储器. 当在显示方式 09H 和 0AH 中工作时, PCjr 保留额外的 16K 字节, 但在此中断工作时, BIOS 不报告这个额外的存储器损耗.

入口参数:

调用该功能请求, 发出中断 12H 即可, 预先不必设置任何寄存器.

出口参数:

返回时, 设置为:

AX 该寄存器以1K字节 (1024字节) 为单位列出可用的RAM空间.

参见:

中断 11H——确定设备.

程序实例:

下面程序是利用 BIOS 中断 12H 获取 RAM 尺寸的例子. 结果在显示器中给出.

```
Pascal Usage Example:
{ Get Memory Size ($ 12)}
PRORAM get_memory_size;

CONST
get_mem_size = $ 12;
```

```

TYPE
  regpack = RECORD
    CASE integer OF
      1:(ax,bx,cx,dx,bp,si,di,ds,es,flags:integer0;
      2:(al,ah,bl,bh,cl,ch,dl,dh:byte);
    END;
VAR
  regs:regpack;

BEGIN
  intr(get_mcm_size,regs);
  writeln(' Memory Size = ',regs.ax,' K bytes' );
END;

```

#### C Usage Example:

```

/*
 * Get memory size (0x12)
 */

#include <dos.h>

union REGS inregs,outregs;

main()
{
    int86(0x12,&inregs,&outregs);
    printf(" Memory Size = %dk bytes\n",outregs.x.ax);
}

```

### BIOS Int 13H

#### 磁盘 I/O

##### 中断:

13H 磁盘 I/O

##### 说明:

通过这个中断所引用的 BIOS 功能调用允许用户在磁盘驱动器（软盘和硬盘）上进行 I/O 操作。BIOS 通过用户提供的驱动器号来识别它所使用的驱动器，驱动器号通常放在 DL 寄存器内。数码小的驱动器号（如 0, 1, 2 等）分配给软盘驱动器，产生的控制传

送到 BIOS 软盘处理的那一部分，数码较大的驱动器号（如 80H, 81H, 82H 等）分配给硬盘，产生的控制送到 BIOS 硬盘处理部分。

利用中断端口 13H 可进行多种功能调用，前 6 个功能适用于各种磁盘驱动器，其余的功能仅适用于 AT 机的软盘驱动器和 XT、AT 的硬盘驱动器。用户选择调用功能是通过赋予 AH 寄存器置相应的数值，同时发出 13H 中断来完成的。利用中断 13H 的功能调用如下：

AH	功能调用
00H	复位磁盘
01H	取磁盘状态
02H	读扇区
03H	写扇区
04H	检测扇区
05H	格式化磁道
08H	取当前驱动器参数
09H	初始化双驱动器
0AH	读长扇区
0BH	写长扇区
0CH	查找柱面(磁道)
0DH	备用磁盘复位
10H	检测驱动器是否准备好
11H	复校驱动器
14H	控制器内部诊断
15H	取磁盘类型
16H	改变磁盘状态
17H	置磁盘类型

下面详细叙述这些磁盘功能调用。

**BIOS Int 13H**

**磁盘 I/O**

**功能调用 00H——复位磁盘**

**中断:**

13H 磁盘 I/O

**功能调用:**

00H 复位磁盘

**计算机:**

PC, PCjr, XT, Portable 和 AT

**说明:**

该功能复位磁盘控制器板和磁盘驱动器，它在磁盘控制器芯片上完成复位操作并在磁盘进行所需的操作之前做一系列用于磁盘校准的磁盘操作。

无论何时，其它磁盘 I/O 功能调用出现错误，都要调用这个复位功能，此刻复位功能将使 BIOS 象该磁盘重新插入一样检查驱动器中磁盘状态，就是说对校准所有磁头使之在应在的位置上。

复位功能调用不影响软盘或硬盘上的数据。

**入口参数:**

调用此功能前，必须设置下列寄存器：

AH = 00H 指明调用复位磁盘功能。

DL 为识别需复位驱动器，置驱动器号。0 和 1 代表软盘，80H 和 81H 代表硬盘。

**出口参数:**

无。

**错误信息:**

若错误产生，进位标志 CF = 1，错误信息放在 AH 寄存器：

AH 存放错误信息代码，参见功能调用 01H（取磁盘状态）中对错误信息的说明。

若没有错误产生，CF 和 AL 寄存器均为 0。

**参见:**

中断 13H 功能 01H——取磁盘状态。

中断 13H 功能 0DH——备用磁盘复位。

## **BIOS Int 13**

### **磁盘 I/O**

**功能调用 01H——取磁盘状态**

**中断:**

13H 磁盘 I/O

**功能调用:**

01H 取磁盘状态

**计算机:**

PC, PCjr, XT, Portable 和 AT

**说明:**

该功能调用结果报告上次硬盘或软盘操作的状态，即使这个状态信息能够从产生错误时 BIOS 功能调用的返回信息中立即得到，也应通过此功能调用获取状态信息，以便专门的错误处理程序能分别处理所有的错误。

**入口参数:**

调用此功能时，用户必须设置下列寄存器：

AH=01H 指出取磁盘状态功能调用。

DL 设置需要取出磁盘状态的磁盘驱动器号，任何小于80H数值将报告上次软盘错误，任何等于或大于 80H 数值将报告上次硬盘错误信息。

**出口参数:**

功能调用结果存于 AL 寄存器：

AL 寄存器码表示最后软盘、硬盘错误，下面是AL寄存器代码和表示的错误信息：

AL 错误状态

00H 未出错

01H 非法功能调用命令

02H 地址标记损坏，扇区标识 (ID) 无效或未找到

03H 企图对有写保护的磁盘执行写操作

04H 可能由于扇区号太大，所寻找的扇区没找到

05H 复位操作失败

06H 无介质

07H 初始化错误，驱动参数无效

08H DMA故障

09H DMA边界错误，数据未存在DMA的64K缓冲区内

0AH 检测出错误的扇区标志，该扇区已被标上不能使用的标记，I/O操作不能进行

10H 读磁盘时循环校检码 (CRC) 奇偶校验错，且纠错码 (ECC) 不能纠正

11H 读磁盘时奇偶校检出错，但纠错码 (ECC) 已纠正错误，这是一个情况

	报告信号而不是出错信号
20H	控制器故障
40H	查找操作无效
80H	超时错误，驱动器不响应
AAH	驱动器未准备好
BBH	未定义错误
CCH	被选驱动器出现写故障
FFH	读出操作失败（只对XTY）

### BIOS Int 13H

磁盘 I/O

功能调用 02H——读扇区

中断：

13H 磁盘 I/O

功能调用：

02H 读扇区

计算机：

PC, PCjr, XT, Portable 和 AT

说明：

调用这个功能将从磁盘上把一个或更多的扇区内容读进存储器。因为这是一个低级功能，在一个操作中读取的全部扇区必须在同一条磁道上（就是说，要有相同的磁头号和磁道柱面号或磁道号）。BIOS 不能自动地从一条磁道末尾切换到另一条磁道开始，因此用户必须把跨越多条磁道的读操作分为若干条单磁道读操作。

入口参数：

在调用这个功能时，用户必须设置下列寄存器：

AH = 02H 指出读扇区功能调用号。

AL 设置要读的扇区数目，不允许使用读磁道末端以外的数值，也不允许使该寄存器为 0。

DL 进行读操作的驱动器的代码，数码 0 和 1 用于识别软盘，数码 80H 和 81H 识别硬盘。

DH 所读磁盘磁头号，0 或 1 是软盘磁头，0 到 15（十进制数）代表 XT 或 AT 机的硬盘，其它的磁盘也可能有不同的磁头号。

CH 识别 10 位磁道柱面号（或软盘的磁道号）的低 8 位数。CL 寄存器的第 6 位和第 7 位存放其高 2 位数。对于 320K / 360K 软盘，磁道号范围是 0 到 39（十进制）；对于 1.2M 软盘，磁道号是 0 到 79（十进制）。对于硬盘磁道柱面号，范围是 0 到 1023（十进制）。

CL 寄存器到 5 位放入所读的第一个扇区的扇区号，对于 320K / 360K 软盘，该值范围是 1 到 8 或 9；对 1.2M 软盘是 1 到 15；对于硬盘，从 1 到

17. 需注意扇区号是从 1 而不是 0 开始。CL 寄存器的 6、7 位表示磁道柱面号的高两位。

ES: BX 置存放从磁盘上读出数据的存储器容量。这个存储器容量应该是能容纳 02H 功能调用要读的所有扇区数据。因此用户在调用这个功能之前必须了解所读扇区的大小和数量。

#### 出口参数:

在功能调用结束控制返回后，设置寄存器 ES: BX:

ES: BX 这个寄存器对是数据存储区域指针，指向从磁盘读入的数据区域的开始。如果读入若干个扇区的数据，这些扇区的数据会依次排列。

#### 错误信息:

如果产生错误，进位标志 CF = 1，错误信息放在 AH 寄存器内：

AH 内装错误代码，参见 01H (取磁盘状态) 关于错误信息的说明。

如果没有错误产生，CF 和 AL 寄存器都为 0。

#### 参见:

中断 13H，功能调用 01H——取磁盘状态。

DOS 功能调用 44H，子功能 0DH——普通块设备 I/O 控制。

### BIOS Int 13H

#### 磁盘 I/O

#### 功能调用 03H——写扇区

#### 中断:

13H 磁盘 I/O

#### 功能调用:

03H 写扇区

#### 计算机:

PC, PCjr, XT, Portable 和 AT

#### 说明:

这个功能与 02H 功能相反，它是要写一个或更多的磁盘扇区，参数和限定都和 02H 的相同。

#### 入口参数:

在调用 03H 之前，必须设置下列寄存器：

AH = 03H 指出写磁盘扇区功能调用号。

AL 在这个寄存器内，放入要写的连续扇区数目，不允许在超出磁道末端处写，也不允许将这个寄存器设置为 0。

DL 设置用于识别所要写的驱动器代码，0 和 1 表示软盘驱动器，80H 和 81H 表示硬盘驱动器。

DH 放入所要写的磁头号码，0 和 1 是软盘磁头，0 到 15 (十进制) 代表硬盘磁头。

- CH      设置10位磁道柱面号中的低8位（或软盘的磁道号），CL寄存器中位6位存放10位中的高两位。对320K / 360K 磁盘，磁道号范围0到39（十进制）；对1.2M 磁盘从0到79（十进制）。对硬盘，磁道柱面号范围0到1023（十进制）。
- CL      寄存器内0到5位用于识别所要写的第一个扇区的扇区号。对于320K / 360K 软盘，这个值可从1到8或9；对1.2M 软盘从1到15（十进制）；对硬盘是1到17（十进制）。注意扇区编号从1而不是0开始。第6位和第7位是磁道柱面号的高两位。
- ES: BX    给出放置写数据所需的存储器容量。

#### 出口参数：

除非产生错误，否则这些信息都将写入磁盘。

#### 其它要求：

在写操作进行之前，必须格式化损坏过的扇区。

#### 错误信息：

如果错误产生，进位标志CF=1，错误信息在AH寄存器中：

AH      存放错误信息代码，详细情况参见功能01H错误信息说明，若没有错误产生，CF和AL寄存器设置为0。

#### 参见：

中断13H，功能调用01H——取磁盘状态。

中断13H，功能调用05H——格式化磁道。

DOS功能调用44H子功能0DH——普通块设备I/O控制。

### BIOS Int 13H

#### 磁盘I/O

#### 功能调用04H——检测扇区

#### 中断：

13H 磁盘I/O

#### 功能调用：

04H 检测扇区

#### 计算机：

PC, PCjr, XT, Portable 和 AT

#### 说明：

这个功能检测磁盘上1个或更多个扇区。这个验证测试不是把磁盘上的数据和内存中的数据进行比较，而只是简单地确定读出的数据有无CRC错误。由于磁盘操作一般是可靠的，所以调用这个功能没有太大的必要。

#### 入口参数：

在调用这个功能之前，必须设置下列寄存器：

AH=04H 指出检测扇区功能调用号。

AL	放入被测的连续扇区数目。不允许使用超过磁道末尾的数值，也不允许把寄存器设置为 0。
DL	设置被测驱动器的代码，0和1标识软盘，80H和81H表示硬盘。
DH	设置磁头号，0或1表示软盘磁头，0到15（十进制）标识硬盘磁头。
CH	放入磁道柱面号10位中的低8位（对软盘是磁道号），CL寄存器的第6位第7位存放磁道柱面号的高两位。对320K / 360K 软盘，磁道号范围是0 到 39（十进制）；对1.2M 软盘，磁道号范围是0 到 79（十进制）；对硬盘，磁道柱面号范围则是0 到 1023（十进制）。
CL	0到5位设置测试的第一个扇区号，对于320K / 360K 软盘，其值可从1 到 8 或 9；对1.2M 软盘，从1 到 15（十进制）；对硬盘是从1 到 17（十进制）。注意扇区编号从1开始而不是0。第6位和第7位是磁道柱面号的高两位。

#### 出口参数：

若测试无错误没有输出。

#### 错误信息：

如果测试后产生错误，进位标志 CF=1，错误信息在 AH 寄存器内：

AH 存放错误代码，参见功能01H（取磁盘状态）对错误信息的说明，如果没有错误产生，CF 和 AL 寄存器都为 0。

#### 参见：

中断 13H，功能 01H——取磁盘状态。

DOS 功能调用 44H 的子功能 0DH——普通块设备 I/O 控制。

### BIOS Int 13H

#### 磁盘 I/O

#### 功能调用 05H——格式化磁道

#### 中断：

13H 磁盘 I/O

#### 功能调用：

05H 格式化磁道

#### 计算机：

PC, PCjr, XT, Portable 和 AT

#### 说明：

这个功能格式化磁盘的单条磁道。这种格式化包括识别该磁道上每个扇区的位置和长度。功能调用使用所设置的扇区号，扇区号和分界因子（扇区怎样依照它们在盘上的实际位置进行编号）。

格式化过程是在每一个扇区的第一字节里写入一个标准值以便读/写操作能找到相应的扇区。当软盘格式化时，初写入的数值由磁盘参数表来决定，表地址存放在中断向量 1EH 里。当格式化硬盘时，磁盘特性由硬盘参数表来决定。这里有两个硬盘的参数表，

它们的地址分别放在中断向量 41H 和 46H 内，参数表的格式参见中断 1EH，41H 和 46H。

### 入口参数：

调用这个功能时，必须设置下列寄存器：

AH=04H 指出格式化磁道功能调用号。

DL 设置被格式化的磁盘驱动器号，0和1表示软盘驱动器，80H和81H表示硬盘驱动器。

DH 设置在磁道上的磁头号，0或1表示软盘磁头，0到15（十进制）表示硬磁头。

CH 设置磁道柱面号的低8位（软盘是磁道号）。磁道柱面号的高2位在CL寄存器地第6位和第7位内。对于 320K / 360K 软盘，磁道号范围是 0 到 39（十进制）；1.2M 磁盘范围是 0 到 79（十进制）；对于硬盘，磁道柱面号范围从 0 到 1023（十进制）。

CL 设置第6位和第7位为磁道柱面号的高2位，0到5位不用。

ES: BX 寄存器对指被格式化磁道的地址字段集合，当格式化软盘时，每个扇区地址字段是由四个字节组成：

字节 1 磁道柱面号（用 C 来代替）。

字节 2 磁头号（用 H 来代替）。

字节 3 扇区号（用 R 来代表记录号）。

字节 4 每个扇区的字节数目（用 N 来代替）。它有四种可能的数值：

0 128 字节 扇区。

1 256 字节 扇区。

2 512 字节 扇区。

3 1024 字节 扇区。

在一定磁道里，每一个扇区必定有这样四个字节一组的信息。这些信息根据它们在存储器中出现的顺序记录在磁盘上。因此，如果要把邻近的扇区号连接起来（对软盘），用户应在地址字段连续地给出扇区号。但是如果用户提供了一个分界因子，就能重新排列地址字段，以便在逻辑上连续扇区可以不是物理上相邻。

当格式化一个硬盘时，ES: BX 应指向 512 字节区域，这个区域有两个字节与磁道中每个扇区有联系。这两个字节格式如下：

字节 1 对于好的扇区，它是 00H，对于坏的（或不能用）的扇区是 80H。

字节 2 存放扇区的逻辑扇区号。

和软盘情况一样，这些扇区在存储器内编号影响磁盘上扇区的分界。例如，下列数据将会使磁道按 2 分界格式化：

00H, 01H

00H, 0AH

00H, 02H

00H,	0BH
00H,	03H
00H,	0CH
00H,	04H
00H,	0DH
00H,	05H
00H,	0EH
00H,	06H
00H,	0FH
00H,	07H
00H,	10H
00H,	08H
00H,	11H
00H,	09H

#### 出口参数:

被选磁道格式化.

#### 其它要求:

中断向量 1EH 实际上是描述 PC 如何访问各软盘驱动器的软盘参数表地址, 中断向量 41H 和 46H 是描述计算机如何访问各硬盘参数表地址. 所有磁盘操作时都要用到这些表, 并且在磁道格式化时特别有用. 详见中断 1EH, 41H 和 46H 关于这些表内容的说明.

若要在 AT 上格式化软盘, 应首先调用功能 17H (设置磁盘类型) 来指明是普通磁盘驱动器还是高密度磁盘驱动器.

#### 错误信息:

若检测出错误, 进位标志 CF = 1, 错误信息在 AH 寄存器内:

AH            内装错误代码, 详见功能调用 01H (取磁盘状态) 中关于错误信息的说明.

若没有错误, CF 和 AL 寄存器均为 0.

#### 参见:

中断 13H, 功能调用 01H——取磁盘状态.

中断 13H, 功能调用 17H——设置磁盘类型.

中断 1EH, ——软盘参数表.

中断 41H 和 46H, ——硬盘参数表.

DOS 功能调用 44H, 子功能 0DH——普通块设备 I/O 控制.

## **BIOS Int 13H**

### **磁盘 I/O**

#### **功能调用 08H——取当前驱动器参数**

**中断:**

13H 磁盘 I/O

**功能调用:**

08H 取当前驱动器参数

**计算机:**

XT 和 AT

**说明:**

该功能提供硬盘驱动器联机的信息。由于几种不同类型的硬盘驱动器可以联机，所以这个功能可以提供格式化任意一种驱动器时所必须的信息（例如，有多个磁道柱面，磁头或含有的扇区数）。

**入口参数:**

在调用这个功能之前，下列寄存器必须被置位：

AH = 08H 指出取当前驱动器参数功能调用号。

DL 置需要信息的驱动器号80H和81H代表硬盘。

**出口参数:**

操作完成后，返回信息在下列寄存器：

DL 列出联机硬盘驱动器数目（从0到2）。

DH 给出最大磁头号。

CH 存放10位最大磁头号。

CL 0到5位存放最大可用扇区数目，第6位和第7位是磁道柱面号的高2位。

**错误信息:**

若检测出错误，进位标志 CF = 1，错误信息存放在 AH 寄存器：

AH 存放错误信息代码。详细情况见功能01H（取磁盘状态）对错误信息的说明。

若没有错误，CF和AL寄存器都为0。

**参见:**

中断 13H，功能 01H——取磁盘状态。

DOS 功能调用 44H，子功能 0DH——普通块设备 I/O 控制。

**程序实例:**

下面的程序实例都是利用 08H 功能获取各种连到系统的硬盘的信息，每个硬盘的信息显示在屏幕上。

**Pascal Usage Example:**

{ Get Hard Disk Parameters (\$ 13) }

PROGRAM get\_hard\_disk\_parameters;

CONST

disk\_io = \$ 13 ;  
get\_params = \$ 08 ;

TYPE

regpack = RECORD

CASE integer OF  
1:(ax,bx,cx,dx,bp,si,di,ds,es,flags:integer):  
2:(al,ah,bl,bh,ch,dl,dh:byte);  
END ;

VAR

regs : regpack ;  
i,num : integer;

BEGIN

WITH regs DO

BEGIN

ah := get\_params ;  
dl := \$ 80 ;  
intr(disk\_io,regs) ;  
num := dk-1 ;  
FOR i := 0 TO num DO

BEGIN

ah := get\_params ;  
dl := \$ 80 + i  
intr(disk\_io,regs) ;  
writeln(' Parameters for Drive ',i);  
writeln(' Heads = ',dh+1);  
bh := (cl AND \$ c0) SHR 6 ;  
bl := ch ;  
writeln(' Cyinders = ',bx);  
writeln(' Sectors = ',(cl AND \$ 3f));

END ;

END ;

END.

**C C sage Example:**

```
/*  
 * Get Hard Disk Parameters (0x13)  
 */  
  
#include <dos.h>  
  
union REGS inregs,outregs;  
  
main()  
{  
    int i;  
    inregs.h.ah = 0x08;  
    inregs.h.dl = 0x80;  
    int86(0x13,&inregs,&outregs);  
    for (i = 0; i < outregs.h.dl; ++i)  
    {  
        inregs.h.dl = 0x80 + i;  
        int86(0x13,&inregs,&outregs);  
        printf(" Parameters ofr Drive %d\n",i);  
        printf("      Heads = %d\n",++outregs.h.dh);  
        outregs.h.bh = (outregs.h.ch & 0xc0) >> 6;  
        outregs.h.bl = outregs.h.ch;  
        printf("      Cylinders = %d\n",outregs.x.bx);  
        printf("      Sectors = %d\n",outregs.h.cl & 0x3f);  
    }  
}
```

**BIOS Int 13H**

**磁盘 I/O**

功能调用 09H——初始化双驱动器

中断:

13H 磁盘 I/O

功能调用:

09H 初始化双驱动器

计算机:

XT 和 AT

### 说明:

这个功能就是把硬盘参数表的内容装入硬盘控制器来完成对硬盘控制器的初始化。硬盘参数表的地址是存在中断向量表中的。建立指向驱动器 0 的参数表的中断向量 41H，建立指向驱动器 1 的参数表的中断向量 46H。41H 和 46H 的说明给出了参数表的格式。

### 入口参数:

在调用本功能时，需设置位 AH:

AH = 09H 指出初始化双驱动器功能调用号。

### 出口参数:

无。

### 错误信息:

若产生错误，进位标志 CF = 1，错误信息在 AH 寄存器内：

AH 存放错误信息代码，详细情况见 01H (取磁盘状态) 关于错误信息代码说明。

若没有错误，CF 和 AL 寄存器为 0。

### 参见:

中断 13H, 01H 功能——取磁盘状态。

## BIOS Int 13H

### 磁盘 I/O

### 功能调用 0AH——读长扇区

### 中断:

13H 磁盘 I/O

### 功能调用:

0AH 读长扇区

### 计算机:

XT 和 AT

### 说明:

该功能从硬盘上读一个或多个长扇区。每个长扇区都含有一个 4 字节的用来检查所读出数据正确性的纠错码 (ECC)。除了 ECC 码，这个功能与 02H (读扇区) 十分相似。

### 入口参数:

调用此功能时，必须设置下列寄存器：

AH = 0AH 指出读长扇区功能调用号。

AL 设置连续读的扇区数，不用读超出磁道范围的数值，也不给寄存器置 0。

DL 设置所读的硬盘驱动器号，80H 和 81H 表示硬盘。

DH 设置磁头号，0 到 15。

CH 放入 10 位磁道柱面号的低 8 位，CL 寄存器的第 6、7 位存放其高 2 位，磁道柱面号的范围从 0 到 1023 (十进制)。

**CL** 使0到5位存放所读的第一个扇区的扇区号，该值范围是0到17。注意扇区号是从1开始而不是从0开始。第6位、7位是磁道柱面号的高2位。

**ES: BX** 指向存放从磁盘读出的数据存储区域。这个区域应该足以装下该功能读出的所有扇区的数据。因此用户应了解所要读的扇区的大小及数量。

#### 出口参数：

操作完成后，设置下面寄存器：

**AH** 它指向硬盘读进的数据区域的开始，若读了多个扇区，这些扇区将顺序给出。

#### 错误信息：

若有错误产生，进位标志 CF = 1，AH 寄存器内是错误信息：

**AH** 存放错误信息代码。详见功能01H（取磁盘状态）对错误信息的描述。若没有错误产生，CF 和 AL 寄存器均设置为0。

#### 参见：

中断13H，功能调用——取磁盘状态。

### BIOS Int 13H

#### 磁盘 I/O

#### 功能调用 0BH——写长扇区

#### 中断：

13H 磁盘 I/O

#### 功能调用：

0BH 写长扇区

#### 计算机：

XT 和 AT

#### 说明：

此功能向硬盘上写一个或更多的长扇区，每个扇区都含有一个4字节的验证所写数据正确性的纠错码（ECC），除了（ECC）码，这个功能和03H（写扇区）十分相似。

#### 入口参数：

调用该功能时，必须设置下列寄存器：

**AH=0AH** 指出写长扇区功能调用号。

**AL** 设置连续写入的扇区数，不能在超出磁道范围处写，也不允许将寄存器设置为0。

**DL** 设置写操作所用的硬盘驱动器号，80H和81H表示硬盘。

**DH** 设置驱动器磁头号，0到15（十进制）是XT和AT硬盘磁头号，其它盘可能有不同的磁头号。

**CH** 放入10位磁道柱面号的低8位，CL寄存器的第6、7位存放其高2位，磁

道柱面号的范围从 0 到 1023 (十进制)。

**CL** 使 0 到 5 位存放所要写的一个扇区的扇区号，对于 AT 和 XT 硬盘，该值范围是 0 到 17。注意扇区号是从 1 开始而不是从 0 开始。第 6 位、7 位是磁道柱面号的高 2 位。

**ES: BX** 该寄存器指向存放写数据的存储区域。

#### 出口参数：

若没有出错，信息写入硬盘。

#### 其它要求：

写之前，必须对损坏过的扇区格式化。

#### 错误信息：

若有错误产生，CF = 1，AH 寄存器内是错误信息：

**AH** 存放错误信息代码。详见功能 01H (取磁盘状态) 对错误信息的说明。若没有错误产生，CF 和 AL 寄存器均设置为 0。

#### 参见：

中断 13H，功能调用 01H——取磁盘状态。

中断 13H，功能调用 05H——格式化磁道。

### BIOS Int 13H

#### 磁盘 I/O

#### 功能调用 0CH——查找柱面 (磁道)

#### 中断：

13H 磁盘 I/O

#### 功能调用：

0CH 查找柱面 (磁道)

#### 计算机：

XT 和 AT

#### 说明：

此功能可将硬盘读 / 写磁头置于指定的柱面磁道上。

#### 入口参数：

在调用这个功能之前，必须设置下列寄存器：

**AH = 0CH** 指出查找柱面功能调用号。

**DL** 设置所写驱动器号，80H 和 81H 表示硬盘。

**DH** 设置驱动器磁头号，0 到 15 (十进制) 为 XT 和 AT 机硬盘磁道号，其它磁盘可能有不同的磁头号。

**CH** 放入磁头定位处 10 位磁头柱面号的低 8 位，高 2 位放入 CL 寄存器的第 6、7 位，磁道柱面号范围 0-1023 (十进制)。

**CL** 第 6、7 位放入磁道面号的高 2 位，0 到 5 位不用。

### **出口参数:**

若没有错误，磁头将移到指定的位置。

### **错误信息:**

若错误产生，进位标志 CF=1，错误信息在 AH 寄存器：

AH 存放错误信息代码，详见功能01H（取磁盘状态）对错误信息说明。

### **参见:**

中断 13H，功能调用 00H——复位磁盘。

中断 13H，功能调用 01H——取磁盘状态。

## **BIOS Int 13H**

### **磁盘 I/O**

#### **功能调用 0DH——备用磁盘复位**

### **中断:**

13H 磁盘 I/O

### **功能调用:**

0DH 备用磁盘复位

### **计算机:**

XT 和 AT

### **说明:**

此功能完成对硬盘控制器的一个复位操作，它的过程和 00H（磁盘复位）一样，但这个复位仅对硬盘控制器而不对软盘控制器起作用。

### **入口参数:**

在调用这种功能之前，必须设置下列寄存器：

AH = 0DH 指出备用磁盘复位功能调用号。

DL 设置所复位的驱动器号，80H 和 81H 表明硬盘。

### **出口参数:**

无。

### **错误信息:**

若产生错误，进位标志 CF=1，错误信息见 AH 寄存器：

AH 存放错误信息代码，详见功能01H（取磁盘状态）对错误信息说明。

### **参见:**

中断 13H，功能调用 00H——复位磁盘。

中断 13H，功能调用 01H——取磁盘状态。

## **BIOS Int 13H**

### **磁盘 I/O**

#### **功能调用 10H——检测驱动器准备**

**中断:**

13H 磁盘 I/O

**功能调用:**

10H 检测驱动器准备

**计算机:**

XT 和 AT

**说明:**

这个功能测试驱动器准备好信号来查看硬盘驱动器是否可用。

**入口参数:**

调用这个功能之前, 设置下列寄存器:

AH = 10H 指出测试驱动器准备好功能调用号。

DL 设置被测驱动器号, 80H和81H代表硬盘。

**出口参数:**

若驱动器准备好, AH 寄存器为 0, 否则 AH 内放入机应错误代码。

**错误信息:**

若错误产生, 进位标志 CF=1, 错误信息在 AH 寄存器:

AH 存放错误信息代码, 详见功能01H (取磁盘状态) 对错误信息的说明。

**参见:**

中断 13H, 功能调用 01H——取磁盘状态。

## **BIOS Int 13H**

### **磁盘 I/O**

#### **功能调用 11H——复校驱动器**

**中断:**

13H 磁盘 I/O

**功能调用:**

11H 复校驱动器

**计算机:**

XT 和 AT

**说明:**

这个功能重新校准指定的硬盘驱动器。

**入口参数:**

调用此功能前，必须设置下列寄存器：

AH = 11H 指出复校驱动器功能调用号。

DL 设置需校准的驱动器，80H和81H代表硬盘。

出口参数：

无。

错误信息：

若错误产生，进位标志 CF=1，错误信息在 AH 寄存器：

AH 存放错误信息代码。详见功能调用01H（取磁盘状态）的错误信息的说明。

若没有错误，CF和AL寄存器均为0。

参见：

中断 13H，功能调用 01H——取磁盘状态。

**BIOS Int 13H** 的功能和空闲一样，它能读取磁盘驱动器的状态，从而完成磁盘 I/O。如果磁盘驱动器知道你要读取的数据，它将返回数据；否则将返回功能调用 14H——控制器内部诊断。

中断：

13H 磁盘 I/O

功能调用：14H 控制器内部诊断

计算机：

XT 和 AT

说明：

该功能指示硬盘控制器进行自检诊断，结果返回到 AH 寄存器。

入口参数：无。即不用设置入口参数，直接调用即可。

调用此功能前，需设置 AH 寄存器：14H 指出控制器内部诊断功能调用号。

出口参数：无。即不用设置出口参数，直接调用即可。

控制器通过诊断，AH 为 0，反之存放相应的错误代码。

错误信息：

若产生错误，进位标志 AF=1，错误信息在 AH：

AH 存放错误代码，详见功能01H（取磁盘状态）对错误信息的说明。

若没有错误，CF和AL寄存器均为0。

参见：

中断 13H，功能调用 01H——取磁盘状态。

## **BIOS Int 13H**

### **磁盘 I/O**

#### **功能调用 15H——取磁盘类型**

**中断:**

13H 磁盘 I/O

**功能调用:**

15H 取磁盘类型

**计算机:**

AT

**说明:**

这个功能调用结果给出用户所要使用的磁盘驱动器类型。这对 AT 机是很重要的，因为 AT 机有增加高密度磁盘驱动器 (1.2M) 的能力，即能对软盘产生的变化进行检测。在明确了当前用于处理的磁盘驱动器后，用户就可以决定其程序是否利用这个高容量的先进特性。

**入口参数:**

调用此功能之前，必须设置下列寄存器：

AH = 15H 指出取磁盘类型功能调用号。

DL 设置被检测的驱动器号，0和1表示软盘驱动器，80H和81H表示硬盘驱动器。

**出口参数:**

功能执行完毕，信息返回到下列寄存器：

AH 给出磁盘驱动器的类型：

0 给定的磁盘驱动器不存在。

1 给定的磁盘驱动器是不能检测出软盘发生的变化，标准的 320K / 360K 软盘驱动器属于这种类型。

2 指定的磁盘驱动器当软盘容量发生变化时检测的出来，标准的 1.2M 软盘驱动器属于这种类型。

3 给定的磁盘驱动器是硬盘驱动器，其容量大小由寄存器 CX: DX 给出。

CX: DX 若给定的是一个硬盘驱动器，这个寄存器对将给出一个4字节的整数，它是当前硬盘上可用的扇区总数。

**错误信息:**

若错误信息产生，进位标志 CF = 1，错误信息在 AH 寄存器：

AH 存放错误信息代码，详见功能 01H (取磁盘状态) 中对错误信息的说明。

若没有错误产生，CF 和 AL 寄存器都为 0。

**参见:**

中断 13H，功能调用 01H——取磁盘状态。  
中断 13H，功能调用 16H——改变磁盘状态。

**BIOS Int 13H**  
**磁盘 I/O**  
**功能调用 16H——改变磁盘状态**

**中断：**

13H 磁盘 I/O

**功能调用：**

16H 改变磁盘状态

**计算机：**

AT

**说明：**

这个功能对于含有可检测出软盘容量变化的磁盘驱动器的系统是有用的（例如 AT 含有 1.2M 字节软盘驱动器）。对于这类驱动器，该功能在其被调用之时即查看此时是否磁盘容量发生变化。这个检测信息可帮助程序判断是否 FAT 和其它驻留存储器的磁盘信息仍然是正确的，是否需要让程序跳出，在重新读入磁盘状态信息之后运行程序。

**入口参数：**

FS 调用此功能之前，必须设置下列寄存器：

AH = 16H 指出检测软盘容量改变功能调用号。

DL 设置被检测驱动器号，0 和 1 代表软盘驱动器。

**出口参数：**

功能调用完毕，结果将留在下述寄存器：

AH 它给出软盘容量是否发生变化的信息：

0 表示在此功能调用之时，软盘容量还没有发生变化。

6 在这个驱动器内的软盘容量改变了，进位寄存器 CF 也已置位。

**错误信息：**

若错误产生，进位标志 CF = 1，错误信息在 AH 寄存器：

AH 存放错误信息代码，详见功能调用 01H（取磁盘状态）对错误信息的说明，若没有错误，CF 和 AL 寄存器均为 0。

**参见：**

中断 13H，功能调用 01H——取磁盘状态。

中断 13H，功能调用 15H——取磁盘类型。

## **BIOS Int 13H**

**磁盘 I/O**

**功能调用 17H——设置磁盘类型**

**中断:**

13H 磁盘 I/O

**功能调用:**

17H 设置磁盘类型

**计算机:**

AT

**说明:**

此功能用于指定软盘或硬盘驱动器的类型，以便进行格式化操作（功能调用 05H）。

**入口参数:**

调用此功能之前，必须设置下列寄存器：

AH = 17H 指出设置磁盘驱动器类型功能调用号。

AL 指定软盘或硬盘驱动器类型，可能的数值如下：

1 驱动器是320K / 360K，软盘是320K / 360K

2 驱动器是1.2M，硬盘是320K / 360K

3 驱动器是1.2M，软盘是1.2M

DL 设置驱动器号，0和1代表软盘驱动器号。

**出口参数:**

无

**错误信息:**

若错误产生，CF = 1，错误信息在AH寄存器：

AH 存放错误信息代码，详见功能01H（取磁盘状态）对错误信息的说明。

若没有错误产生，CF和AH寄存器均为0。

**参见:**

中断13H，功能调用01H——取磁盘状态。

中断13H，功能调用05H——格式化磁道。

中断13H，功能调用15H——取磁盘类型。

## **BIOS Int 14H**

**RS-232 串行 I/O**

**中断:**

14H RS-232 串行 I/O 接口

**说明:**

BIOS 功能调用通过这个中断来控制 RS-232 串行通信口的操作。这些串行口实际上是允许程序与调制解调器、串行打印机以及其它计算机进行通信的标准通讯路径。

这个中断涉及四个功能调用，选择所需的功能通过向 AH 寄存器置适当数值和发 14H 中断来实现。14H 中断可调用以下四个功能：

- AH 功能调用
- 00H 初始化串行通信口。
- 01H 向通信口写一字符（发送）。
- 02H 从通信口读一字符（接收）。
- 03H 取串行口状态。

下面详细说明这些 RS-232 串行口功能的调用。

## BIOS Int 14H

### RS-232 串行 I/O

#### 功能调用 00H——初始化串行口

##### 中断：

14H RS232 串行接口

##### 功能调用：

00 初始化串行通信口

##### 计算机：

PC, PCjr, XT, Portable 和 AT

##### 说明：

该功能调用为信息的发送或接收设置串行接口。初始化结果将设定所希望的波特率、奇偶性、终止位数和字长。

##### 入口参数：

调用此功能前，必须设置下列寄存器：

AH = 00H 指出初始化串行通信口功能调用号。

AL 设置初始化参数代码。参数代码定义如下：

位

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

7-5 波特率。这三位给出每秒应发送的字符数，可能值如下：

数值	波特率
000	110
001	150
010	300
011	600
100	1200
101	2400
110	4800
111	9600

对于 PCjr，最高只能支持 4800 波特，置高于 4800 的波特率，实际上只能是 4800。  
**4-3 奇偶性。**这两位对串行口所用的方式进行奇偶检验，确认在发送过程中不丢失数据，可能数值如下：

数值	奇偶检验
00	无奇偶检验
01	奇检验
10	无奇偶检验
11	偶检验

**2 终止位数。**这一位指定用以表示字符发送完毕后发出的信号的位数，可能值如下：

数值	终止位数
0	只用一个停止位
1	用两个停止位

**1-0 字符长度。**由于信息发送是串行发送，所以处理机必须知道每个字符是多少位，用以下两个数值之一设置这两位：

数值	字符尺寸
10	七位字符值(标准的 ASCII)
11	八位字符

DX           设置所要初始化的串行接口代码，代码如下：

0           第一个（或唯一的）串行口。

1           第二个串行口。

#### 出口参数：

串行通信接口按用户要求建立了；且在下面的寄存器中存放状态信息。状态信息和功能调用 03H（取串行口状态）返回的结果相同。

AH           此寄存器为传送结状态代码，相应位设置成1代表相应状态。

位

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

位	说明
7	超时
6	传送移位寄存器
5	传送保持寄存器空
4	断开
3	帧检验错
2	奇偶检验错
1	溢出
0	数据准备好

AL           存放调制解调器状态代码，相应位的设置代表相应状态信息。

位

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

位	说明
7	检测接收线信号(检测到载波信号)
6	检测到呼叫指示器信号
5	数据发送准备好 DSR
4	清除发送(CTS)
3	接收线信号监测的改变(载波信号)
2	后沿呼叫检测器信号
1	数据装备准备好(DSR)信号变化
0	清除发送(CTS)信号变化

参见:

中断 14H, 功能调用 03H——取串行口状态。

**BIOS Int 14H**

**RS-232 串行 I/O**

功能调用 01H——发送一字符

**中断:**

14H RS-232 串行 I/O 接口

**功能调用:**

01H 向通信端口写一字符

**计算机:**

PC, PCjr, XT, Portable 和 AT

**说明:**

该功能通过 RS-232 串行口发送一字符。

**入口参数:**

在调用此功能前, 必须设置下列寄存器:

AH = 01H 指出向通信口写一字符功能调用号。

AL 存放所要发送的字符的ASCII码。

DX 给出要发送字符的串行口代码:

0 第一个(或唯一的)串行口

1 第二个串行口

**出口参数:**

若没有错误发生, 字符送到串行端口。

## 错误信息:

寄存器 AH 内的信息指出此功能调用是否成功:

AH 若寄存器为0, 没有错误发生; 但若第7位是1, 则表明有错误产生.

余下的 6 位编码表示传输线状态, 如下表所示:

位

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

位	说明
6	传送移位寄存器空
5	传送保持寄存器空
4	断点检测错
3	帧检验错
2	奇偶检验错
1	溢出
0	数据准备好

该寄存器所存错误情况是功能调用 03H (取串行端口状态) 的子集, 它不能报告的错误状态是由第 7 位表示的超时错误. 本功能调用 01H 是把第 7 位看作所有错误的错误标志, 而不是某一专指的错误. 为了保证得到一个完整的错误信息报告, 可以检查第 7 位, 若第 7 位设置为 1, 则调用功能 03H 以获得全部错误状态信息.

参见:

中断 14H, 功能调用 03H——取串行端口状态.

**BIOS Int 14H**

**RS-232 串行 I/O**

**功能调用 02H——接收一字符**

中断:

14H RS-232 串行 I/O 接口

功能调用:

02H 从通信端口读一字符

计算机:

PC, PCjr, XT, Portable 和 AT

说明:

此功能从串行端口读一个字符, 调用该功能后, 计算机就处于等待状态, 直到从串行口读到一个字符或者超时信号产生为止. 因此, 如不想被强迫等待, 就应该先调用 03H

(取串行端口状态) 决定被接受的数据是否准备好。

#### 入口参数:

调用该功能之前，必须设置下列寄存器：

AH = 02H 指出从串行口接收字符功能调用号。

DX 设置要接收字符的串行端口代码，0代表第一个或唯一的串行端口，1代表第二个串行端口。

#### 出口参数:

若没有错误产生，功能调用完毕，结果存在 AL 寄存器中：

AL 存放从串行端口接收到的字符的ASCII码。

#### 错误信息:

AH 寄存器的信息表示功能调用成功与否：

AH 若寄存器为0，没有错误产生；第7位置1，某类错误产生。余下的6位代码表示传送线路状态，如下所示：

位	
7	
6	
5	
4	
3	
2	
1	
0	

位	说明
6	传送移位寄存器空
5	传送保持寄存器空
4	断点检测错
3	帧检验错
2	奇偶检验错
1	溢出
0	数据准备好

寄存器里的错误信息是功能调用 03H (取串行端口状态) 调用结果的子集。这里唯有超时错误不能表示，因为超时错误一般由第 7 位表示，但功能调用 01H 是把第 7 位作为一般错误产生的标志，因此为保证得到一个完整的错误信息报告，可只检查第 7 位，若设置为 1，则调用 03H 取出计算机的全部错误状态。

#### 参见:

中断 14H，功能调用 03H——取串行端口状态。

**BIOS Int 14H**

**RS-232 串行 I/O**

**功能调用 03H——取串行端口状态**

**中断:**

14H RS-232 串行接口

**功能调用:**

03H 取串行口状态

**计算机:**

PC, PCjr, XT, Portable 和 AT

**说明:**

这个功能调用报告串行接口的工作状态，它有两种返回信息，即传送线路控制状态和调制解调器状态。传送控制状态信息和功能调用 01H、02H 返回的信息是一类信息，所不同的是比那两种调用返回信息更全一些，因为 03H 功能调用返回的传送状态包括那两种调用不能表示的超时错误状态。若调制解调器联机，调制解调器状态将反映它的当前工作状态和同步信息。

尽管这个功能调用提供了一些与出错有关的信息，但也为串行端口的同步提供了信息。例如：从串口接收一个字符就要调用功能 02H 然后长时间等待。03H 可以改变这种情况，首先调用 03H 检查数据准备好标志位。如果此时数据不能接收，程序可进行其它操作，稍后再检测这个标志位。当数据准备标志位最后给出数据要接收标时，再调用 02H.

**入口参数:**

调用此功能之前，必须设置下列寄存器：

AH = 03H 指出取串行端口状态功能调用号。

DX 设置所需串行端口代码：

0 第一个（或唯一的）串行接口

1 第二个串行接口

**出口参数:**

功能执行后，返回的信息在下列寄存器：

AH 存放传送线路状态代码如下所示：

位							
7	6	5	4	3	2	1	0

7	超时错误
6	传送移位寄存器空
5	传送保持寄存器空

4	断点检测错
3	帧检验错
2	奇偶检验错
1	溢出
0	数据准备好

AL 存放调制解调器状态码，如下所示：

位

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

位	说明
7	检测接收信号(检测到传送线上载频信号)
6	检测到呼叫指示器信号
5	数据准备好(DSR)
4	清除发送(CTS)
3	接收线路信号(载频信号)改变
2	后沿呼叫检测信号
1	数据准备好(DSR)
0	清除发送(CTS)变化

参见：

中断 14H，功能调用 01H——向通信口发送一字符。

中断 14H，功能调用 02H——从通信口接收一字符。

**BIOS Int 15H**

盒式磁带 I/O

**中断：**

15H 盒式磁带 I/O 接口

**说明：**

通过这个中断接口有两类 BIOS 功能可调用：一类是早期的，一类是新近的。第一类为早期 PC 机配的盒式磁带 I/O 功能（这种盒式磁带接口只在最早型号的 PC 机上有）。后面将要详细说明的第二类是为 AT 机配的扩充服务功能。

在这个中断接口上调用盒式磁带 I/O 功能可以控制一个连接在 PC 机盒式磁带接口上音频盒式磁带机。因为只有 PCjr 和早期型号的 IBM PC 机有盒式磁带接口，所以这个功能只能用在很少的机器上。IBM XT 和 AT 机都没有磁带接口。即使在有磁带接口的那些机器上，磁带功能调用也是很少用的，因为这种接口是较原始的，用途有限，用户在读/写时须手动磁带，还要手工绕带。此外，盒式磁带 I/O 功能也只能做简单操作，例如：起停磁带机和读、写数据块。

调用盒式磁带接口功能必须在 AH 寄存器内设置相应功能的数值，同时发出 15H 中断。经过这个中断可完成下面的盒式磁带功能：

AH	功能调用
00H	启动盒式磁带机
01H	停止盒式磁带机
02H	读数据块
03H	写数据块

下面详细说明 15H 中断的功能。

### BIOS Int 15H

#### 盒式磁带 I/O

功能调用 00H——启动盒式磁带机

#### 中断：

15H 盒式磁带接口

#### 功能调用：

00H 启动盒式磁带机

#### 计算机：

PC, PCjr

#### 说明：

对软盘进行 I/O 操作时，BIOS 会自动地启动驱动器马达使软盘开始旋转。而磁带操作比软盘要原始得多。对磁带进行读、写之前，必须先调用这个功能启动磁带马达。调用之后，程序也应等一会儿，给读写的马达一个启动时间。

#### 入口参数：

调用这个功能之前，必须设置 AH 寄存器：

AH = 00H 指出启动盒式磁带马达功能调用号。

#### 出口参数：

盒式磁带机马达启动。

#### 错误信息：

若产生错误，进位标志 CF = 1，错误信息在 AH 寄存器中：

AH 若盒式磁带接口不存在，其值为86H.

参见：

中断 15H，功能调用 01H——停止磁带机马达。

**BIOS Int 15H**

**盒式磁带 I/O**

**功能调用 01H——停止盒式磁带机**

中断：

15H 盒式磁带接口

功能调用：

01H 停止盒式磁带机

计算机：

PC, PCjr

说明：

该功能停止盒式磁带机的马达。这与软盘操作不一样，盒式磁带机马达不会自动停住。因此，在完成磁带 I/O 后必须调用此功能。

入口参数：

调用此功能前必须设置 AH 寄存器：

AH=01H 指出停住盒式磁带马达功能调用号。

出口参数：

磁带机马达停止转动。

错误信息：

若产生错误，进位标志 CF=1，错误信息在 AH 寄存器：

AH 若盒式磁带机接口不存在，其值为86H.

参见：

中断 15H，功能调用 00H——启动磁带机。

**BIOS Int 15H**

**盒式磁带 I/O**

**功能调用 02H——读取数据块**

中断：

15H 盒式磁带接口

功能调用：

02H 读数据块

计算机：

PC, PCjr

说明：

该功能从磁带上把数据读进存储器。数据在盒式磁带上是以 256 字节为一块存储的，到存储器也是按块传送的。因此数据缓冲区必须是 256 字节的倍数。但是也可指定所要读的数据字节数，而这个数字并不是 256 字节的倍数。这个功能将传送适当的 256 字节的块，并调整存储器指针指向所读取的最后一个数据。

#### **入口参数：**

调用此功能时，必须设置下列寄存器：

AH = 02H 指出读数据块功能调用号。

ES: BX 指向存放数据的缓冲区开始地址。

CX 设置从磁带上所要读的数据字节数。

#### **出口参数：**

若没有错误产生，数据被读进存储器，控制返回信息在下列寄存器中：

ES: BX 指向读出最后一个字节读出后的存储单元。

DX 记录传送的字节数。

CX 每个字节读完后，确定 CX 中的数值，即把开始输入的全部需读字节总数减 1。在读操作结束时，这个寄存器就是 0。

#### **错误信息：**

若有错误产生，进位标志 CF = 1，错误信息在 AH 寄存器：

AH 给出所发生错误的类型，如下所示：

01H 循环检验 (CRC) 错

02H 转换数据丢失（当有数据块到来时转换的一些数据未能读入）

03H 磁带上无数据

86H 没有盒式磁带接口

#### **参见：**

中断 15H，功能调用 03H——写数据块。

### **BIOS Int 15H**

#### **盒式磁带 I/O**

#### **功能调用 03H——写数据块**

#### **中断：**

15H 盒式磁带 I/O 接口

#### **功能调用：**

03H 写数据块

#### **计算机：**

PC, PCjr

#### **说明：**

该功能把用户提供的数据从存储器缓冲区取出写入盒式磁带。在盒式磁带上数据是 256K 字节长。因此，若提供的数据缓冲区不是 256 字节的倍数，此功能就要用数据填满成块。

### 入口参数:

调用此功能之前，需置下列寄存器：

AH = 03H 指出写数据块功能调用号

EX: BX 指针指向存放被写数据的存储器缓冲区。

CX 给出要写入盒式磁带字节总数。

### 出口参数:

若没有错误产生，数据写入磁带当前位置。控制返回后下列寄存器置位：

ES: BX 指向最后一个字节写完后的存储单元。

CX 每个字节写完之后，该功能都要使CX里存的数减1。在写操作完成之后，寄存器就设置为0。

### 错误信息:

盒式磁带机没有能力告诉计算机所发生的写错误。因此，这个功能调用不能检测任何错误。为了确认没有写错误，就必须进行人工倒带，把从磁带上读出的数据与写入的数据相比较。

若没有盒式磁带机接口，AX寄存器值为86H。

### 参见:

中断15H，功能调用02H——读取数据块。

## BIOS Int 15H

### AT机扩充服务

#### 中断:

15H AT机扩充服务

#### 说明:

在下面说明的这一类 BIOS 功能调用是给 AT 机增加的 BIOS 扩展功能，它仅在 AT 机 BIOS 上出现。这些功能调用而并不是真与 15H 有关，但的确是为了方便设立在中断 15H 之下的。

只要对 AT 寄存器设置相应的数值并发出 15H 中断，就可得到所需的功能。通过这个中断，AT 机扩充服务如下所示：

AH	功能调用
83H	事件等待
84H	控制杆支持
85H	Sys Req 键支持
86H	等待
87H	成块传送
88H	取扩展存储器容量
89H	转移到 PVAM 方式
90H	设备忙循环(WAIT)
91H	置中断完成标志(POST)

下面详细介绍 AT 机的扩展功能.

### **BIOS Int 15H**

#### **AT 机扩充服务**

##### **功能调用 83H——事件等待**

###### **中断:**

15H AT 机扩充服务

###### **功能调用:**

83H 事件等待

###### **计算机:**

AT

###### **说明:**

该功能只适用于 AT 机，它管理一个事件定时器，这个定时器一次可设置一位微秒表示的时间间隔数，利用这个定时器可产生两个子功能：一个是设置事件等待所需时间间隔，另一个是取消事件等待操作。

###### **入口参数:**

调用此功能之前，需设置下列寄存器：

AH = 83H 指出事件等待功能调用号。

AL 设置所选择的子功能号：

0 设置间隔时间，用户同时还要置下面说明的 CX, DX, 和 ES: BX。

1 取消先前置的时间操作。

CX 该功能调用采用 CX、DX 两个寄存器组成一个 32 位整数形式的寄存器来存放等待时间微秒数值。CX 寄存器存 32 位整数的高 16 位（最高有效位）并且此值只有 AL=0 才赋值。

DX 存放指定时间间隔微秒的低 16 位（最低有效位），这个值也只有 AL=0 才会赋值。

ES: BX 使此寄存器指向可表示所指定的时间间隔结束的一个字节，在时间到期结束时，此字节的高位第 7 位置 1。上述寄存器内的数值只有 AL=0 时才显得重要。

###### **出口参数:**

此功能调用的返回信息不等所设置等待时间结束就返回，这样就允许同时进行其它操作。当等待时间确实结束时，置位下述寄存器：

ES: BX 若设置时间间隔计数器 (AL=0)，当等待时间结束时刻，这个寄存器指向一个字节，此字节的最高位（第 7 位）就会被设置为 1。

###### **错误信息:**

事件时间计数器一次只能给一个进程用。如果有另外一个进程正在使用事件时间计数器，进位标志 CF=1，若时间计数器可被使用，则 CF=0。

###### **参见:**

**中断 15H，功能调用 86H——等待。**

**BIOS Int 15H**

**AT 机扩充服务**

**功能调用 84H——控制杆支持**

**中断：**

15H AT 机扩充服务

**功能调用：**

84H 控制杆支持

**计算机：**

AT

**说明：**

此功能仅适用于 AT 机。它从 AT 机的控制杆接口检索信息来完成两个子功能：一个是读取当前开关的置位情况，另一个是获得两个控制杆的最新 X、Y 值。

**入口参数：**

若选择取当前开关位置 (DX = 0)，返回信息在 AL 寄存器中：

AL 7-4位，存放控制杆接口开关当前置位情况，此数值取决于选择的控制杆类型。

若选择取当前控制杆位置 (DX = 1)，返回信息在下列寄存器中：

AX 存放A控制杆X位置

BX 存放A控制杆Y位置

CX 存放B控制杆X位置

DX 存放B控制杆Y位置

**错误信息：**

若计算机没有控制杆接口，给出非法输入参数，进位标志 CF = 1。

**BIOS Int 15H**

**AT 机扩充服务**

**功能调用 85H——Sys Req 键支持**

**中断：**

15H AT 机扩充服务

**功能调用：**

85H Sys Req 键支持

**计算机：**

AT

**说明：**

该功能只适用于 AT 机，不管什么时候 Sys Req 键被按下或断开又按下，BIOS 都

可发出这个功能调用，但因为此功能存在于 BIOS 之中，它只是简单地执行 IRET 指令以后便返回调用程序，所以功能代码没起作用。然而正是因为 BIOS 提供了这个功能的结构，在这里用户才有可能使用键 Sys Req 的支持。

为了扩展对 Sys Req 键的支持，必须增加对功能调用 85H 响应的中断处理程序。为了做到这一点，用户编写的中断处理程序初始化部分必须进行 IBM 文件称之为挂钩的操作，来和中断 15H 挂钩。此功能调用的步骤如下：

- 1、中断处理程序必须读和保存中断 15H 的中断向量。
- 2、然后程序必须为自己选择相应地址建立中断向量。
- 3、当中断 15H 产生时，用户建立的中断处理程序（中断句柄）在控制。它必须测试 AH 寄存器判断是哪个功能被调用。若 AH 值是 85H，程序响应 85H 所对应的 Sys Req 键功能，并在响应完成之后发出 IRET 指令，使控制返回到调用程序。反之，若 AH 值不是 85H 而是别的值，程序则应立即产生 CALL 或 JMP 指令，使控制转到原先存在中断向量表中的地址上去。

#### 入口参数：

当 BIOS 调用功能 85H 时，下面信息被放入 AH, AL 寄存器：

AH	此寄存器存放 85H 表明调用 Sys Req 键支持功能。
AL	BIOS 设置此寄存器两种类型：
00	Sys Req 键按下。
01	Sys Req 键松开。

#### 出口参数：

无。

#### 其它要求：

Sys Req 中断处理程序必须保存和恢复除 AX 寄存器外的所有寄存器内容。

### BIOS Int 15H

#### AT 机扩充服务

#### 功能调用 86H——等待

#### 中断：

15H AT 机扩充服务

#### 功能调用：

86H 等待

#### 计算机：

AT

#### 说明：

该功能只适用于 AT 机，在允许用户给定等待时间间隔方面，它类似于功能 83H，与 83H 不同的是到等待时间结束控制才从中断 86H 返回，所以这个功能调用可将程序在指定时间内保持。

#### 入口参数：

在调用此功能前，必须设置下列寄存器：

AH=86H 指出等待功能调用号。

CX 此功能把CX, DX合起来作为一个32位整数寄存器，存放等待时间的  
微秒数。CX寄存器存放的是32位微秒数的高16位（高有效位）。

DX 存放32位微秒数的低16位。

#### 出口参数：

在返回控制之前，功能调用使机器等待 CX 和 DX 指定的时间。

#### 错误信息：

事件定时器一次只能给一个进程用，如果有另外一个进程正在使用事件定时器，进位  
标志 CF = 1；此时的定时器可用，CF = 0。

#### 参见：

中断 15H，功能调用 83H——事件等待。

### BIOS Int 15H

#### AT 机扩充服务

#### 功能调用 87H——成块传送

#### 中断：

15H AT 机扩充服务

#### 功能调用：

87H 成块传送

#### 计算机：

AT

#### 说明：

此功能调用只适用于 AT 机，它主要用于在标准存储器（0-640K）和扩展存储器  
(1M-16M) 之间成块传送数据。该功能使 VDISK 程序应用成为可能，而 VDISK 允许  
把超出 640K 的存储器作为 RAM 虚盘使用。

当 80286 处理器处于实地址方式（类似于 8088），此功能不能完成成块数据传送。因  
为在实地址方式 80286 只能存取 1M 字节，故 BIOS 必须首先把 80286 处理器存储方式转  
换成有存储保护的虚地址方式（PVAM），在这种方式下可以直接存取 16M 字节。一旦  
数据传送完毕，该功能将把处理器恢复成实地址。

在实地址方式和 PVAM 之间有许多不同之处，尽管以下几段将说明用户需知的运用  
本功能的方法，但详细讨论 PVAM 还是超出本书范围。关于 PVAM 的详细情况参见  
Intel 文献，特别是 IAPX286 程序员参考手册（Intel 顺序号为 210498）。

为调用此功能，必须了解处理器转移到 PVAM 后存储寻址方式的改变，下面是这种  
改变的综述。

在实地址方式下，所访问的任何存储器地址都是它的基地址加偏移，基地址标识一个  
存储器起始地址。所有段都是以偶数地址开头，以 16 字节分界，偏移数值准确定位段  
开始地址的距离，从而指出实际存储单元位置。

在 PVAM 中仍然有基地址和偏移值，但是一个基地址不再直接访问一个存储器单元，而是作为一种表格（称描述符表）的索引。这个表列出处理器能存取的所有段的信息，有关每个段的信息就称作描述符。每个描述符包括标识实际段开始位置的 24 位地址，偏移值就如在实地址方式下一样，表示距此段开始处的距离。

由于在实地址方式下，地址不能有效地标识超出 1M 字节的存储器空间，故当调用此功能时，必须通过建立描述符表来辨认存储区域的源区域和目的区域地址。这个表包括这两个区域的所有分段信息。然后，当处理器转移到 PVAM 时，就要利用描述符表在复制信息之前识别该数据段。

在 PVAM 下，中断情况也与实地址方式不同。为解决这个问题，在调用此功能时，当处理器处于 PVAM 下，BIOS 就禁止所有中断。因此日历钟在此刻不更新，任何其它依赖于中断的服务工作在这段时间也不能进行。这可能会给以较高波特率工作的数据通信程序带来麻烦。

#### 入口参数：

在调用此功能之前，需设置下列寄存器：

AH = 87H 指出成块传送数据功能调用号。

CX 给出从一个存储块传送到另一个存储块字数（16位）。

EX: SI 设置这对寄存器指向描述符表，该表描述含有信息的存储区。根据所建立的描述符表，既可向扩展存储器拷贝到扩展存储器，也可以从扩展存储器中拷贝输出信息。

下面将说明必须建立的描述符表，首先说明单个段描述符的一般格式，然后说明该功能调用如何建立数据描述符表。

单个描述符由 8 个字节组成，8 个字节分成字段如下：

相对描述符起点偏移位置（字节）

0	段界
1	
2	基地址低位字
3	
4	基地址高位字节
5	存取权
6	保留字
7	

字段含义如下：

段界 这是一个字，指定段长的字节数。段长可以是不超过 FFFFH 字节（64K）的任意长度，在 PVAM 下，硬件不允许做超出这个指定段界的存取操作。

**基地址的低位字和高位字节** 这两个字段给出段开始处完整的 24 位地址。用 24 位地址时在 16M 字节地址空间，段可从任何地方开始。第一个字段（低位字）必须装有地址的低 16 位有效数字，第二个字段（高位字节）必须是地址的高 8 位有效数字。与实地址方式下的基地址不同之处是一个 PVAM 基地址可以地址空间任何一点开始，但它最好总是在偶数字节上开始分段。

**存取权** 这个字段是一个指定存储段存取权的字节。存取权涉及到存储保护和处理器的虚存能力，且超出本手册范围。在调用本功能建立描述符时，应置存储权为 0（对伪描述符）或 93H（对所指定的源和目的段描述符）。

**保留字** 即使在 PVAM 下 80286 也不用这个字段，为了与 80386 所用的描述符兼容，该字段必须总是置 0。

为了使建立的描述符表能作为功能调用的输入，必须按下面次序建立描述符。

相对 EX：SI 位置偏移

00H	伪描述符
08H	描述符表的描述符
10H	源描述符
18H	目的描述符
20H	BIOS 代码段描述符
28H	堆栈段描述符

用下面信息填写各个字段，按上述格式建立每个描述符。

**伪描述符** 每个描述符表是以一个伪描述符开始的，初始化将该描述符所有字段清 0.

**描述符表的描述符** 描述符表象其它任何段一样，也是一个段。因此在表中必须有自己的描述符，使程序能够对它存取和修改。在功能调用 87H 选取适当的信息建立此描述符时，恰恰是在处理器转换到 PVAM 之前，因此用户对这个描述符所有字段应初始化为 0.

**源数据区描述符** 这个描述符必须对原先存放那些准备拷贝传送的信息存储区域进行说明，设置字段如下：

**段界** ——此字段是大于等于  $2 \times (\text{CX}) - 1$  数值。

**基地址** ——这三个字节存放存储区域开始 24 位的首地址，如果其值大于 FFFFFH (1M 字节) 就应从扩展存储器里取，如果其小于 A0000H，就应从标准存储器里取。

**存取权** ——将此位设置为 93H。

**目的数据区描述符** 除了指向取信息存入到存储区域不同外，其它都与源数据区描述符相同。

**BIOS 程序段描述符** 为了能在 PVAM 方式下运行，BIOS 必须有自己的程序段描述

符。该功能调用在这个描述符建立诸字段，同样要把它们初始化成 0.

**堆栈段描述符** 堆栈段也需要描述符，它由功能调用建立，也要把它的所有字段初始化为 0。

**出口参数：**

若此功能调用是成功的，零位标志 ZF = 1，AH 将置 0；数据从源区域到目的区域；若有错误发生，进位标志 CF = 1.

**错误信息：**

若产生错误 CF = 1，下述寄存器复位：

AH           给出如下工作状态：

0           成功完成调用。

1           RAM 产生奇偶错，此功能调用清除过错误。

2           其它例外的中断错误产生，表示企图进行非法的 PVAM（这种可能是对描述符设置了不合适的存取权而引起保护的错误）。

3           在此功能调用期间，BIOS 要使 20 号地址线工作，但该错误表明此线还处于待用未工作状态。

**BIOS Int 15H**

**AT 机扩充服务**

**功能调用 88H——取扩展内存容量**

**中断：**

15H AT 机扩充服务

**功能调用：**

88H 承扩展内存容量

**计算机：**

AT

**说明：**

此功能只适用于 AT 机，它的返回结果是现安装在计算机上的扩展存储器容量，也就是超过 1M 字节存储器地址的总量。

扩展存储器的容量原先是一开机由运行 POST（加电，自检）决定的。POST 检查存储器的容量并将结果存在 AT 机的 RT / CMOS 上的 RAM 芯片上，地址为 30H 和 31H 的单元内。存储器的容量存在 RT / CMOS 芯片，这样即使在软启动普通 RAM 复位，而 POST 没有运行时也可用。本功能 88H 就是取存在 RT / CMOS 芯片中的数值。

原先 POST 记录扩展存储器的容量时，就假定了扩展存储器是工作存储器，而且所有存储器从 0—640K 都是连接的。铕记录相邻的扩充存储器的数，若在扩展的存储器之间有间隙，则当前只能记录 1M 字节处开始第一个存储区域。

**入口参数：**

调用此功能之前，需设置下列寄存器：

AH = 88H 指出取扩展存储器容量功能调用号。

### **出口参数:**

控制信息返回到 AX 寄存器:

AX 是连续扩展存储块数 (每块 1K 字节, 从 1M 字节地址处开始计算).

### **BIOS Int 15H**

#### **AT 机扩充服务**

#### **功能调用 89H——转移到 PVAM**

### **中断:**

15H AT 机扩充服务

### **功能调用:**

89H 转移到 PVAM

### **计算机:**

AT

### **说明:**

该功能只适用于 AT 机, 这个功能使 80286 从 8088 方式 (实地址) 切换到 PVAM (保护) 方式, 从而 80286 可直接寻址 16M 字节存储器空间. PVAM 还提供存储保护特性. 然而, DOS 的各种版本一直到 3.X, BIOS 和 DOS 都不提供对 PVAM 的支持, 因此如果转许多到 PVAM 方式, 用户必须自己使用所有的 I/O 和自己处理所有的中断.

下面几段说明调用此功能的基础, 然而详细讨论保护方式已超出本书范围, 实际上它可能需要一本书来专题叙述. 但下面的讨论将提及一些在书中其它地方没有做详细叙述的名词和概念, 如果以前不理解这些名词, 就不要运用这个功能.

若要了解详细的 PVAM 背景信息, 参见 Intel 公司的文献, 特别是《IAPX286 介绍》(Intel 序号是 210308 和《IAPX286 程序员参考手册》(Intel 序号是 210498).

由于实地址方式和保护方式在存储器内寻址方式不同 (见功能 87H——传送数据块说明), 调用此功能之前, 必须建立一个专门表, 称为 GDT (全局描述符表). 在 PVAM 下操作时, GDT 允许处理器在 PVAM 方式存取所需要的所有存储段.

在 GDT 中, 每个单独的段描述符都由 8 个字节组成. 这 8 个字节分成下面的字段:

相对于描述符的偏移位置 (字节)

0	段界
1	
2	基地址低位字
3	
4	基地址高位字节
5	存取权
6	保留字
7	

**字段含义如下：**

**段界** 这是一个字，指定段长的字节数。段长可以是不超过 FFFFH 字节（64K）的任意长度，在 PVAM 下，硬件不允许超出这个指定段界进行存取。

**基地址的低位字和高位字节** 这两个字段给出段开始处完整的 24 位地址。用 24 位地址时在 16M 字节地址空间，段可从任何地方开始。第一个字段（低位字）必须装有地址的低 16 位有效数字，第二个字段（高位字节）必须是地址的高 8 位有效数字。与实地址方式下的基址不同之处是一个 PVAM 基址可以地址空间任何一点开始，但它最好总是在偶数字节上开始分段。

**存取权** 这个字段是一个指定该段存取权的字节。存取权涉及到存储保护和处理器的虚存能力，且超出本手册范围。参见 Intel 公司文献（为使数据字段有 0 级特权和进行读/写存取操作，设置其存取权为 93H，9BH 可建立一个程序段，它具有 0 级特权和可进行读/执行存取操作。）

**保留字** 80286 无论在实地址方式还是 PVAM 下，都不用这个字段，它是为了与 80386 所用的描述符兼容而保留的。用户应总是将此字段设置为 0。

在 PVAM 下，中断处理也与以前不同。处理器不再认为中断向量表是从 0 地址开始。此时用户必须为达到与以前制定中断向量表同样目的而要定义一个中断描述符表（IDT）。这个 IDT 必须包括所有硬件中断入口（描述符）。这些入口不涉及 80286 存储器保留区。功能 89H 将在 IDT 提供信息基础上初始化 8259 中断控制器。

IDT 中各个描述符也由 8 个字节组成，但它的格式与段描述符不同，8 字节的 IDT 描述符可分成下面的字段。

相对于描述符的位置偏移（字节）

0	中断程序相对地址偏移
1	
2	中断程序选择器
3	
4	未用字节
5	存取权
6	保留字
7	

**字段含义如下：**

**中断程序相对地址（偏移）和选择器** 这个字和选择器是进行中断处理的程序的指针。选择器是距 GDT 的偏移。程序在 GDT 中被定义。

**未用字节** 置 0 此字节。

**存取权** 这个字段是一个指定该描述符存取权的字节。存取权涉及到存储保护和处理

器的虚存能力，且超出了本手册范围，参见 Intel 公司的文献，以便详细了解。(为使中断控制享有 0 级特权，设置存取权为 96H，97H 可建立一个具有 0 级特权的陷阱控制)。

**保留字** 这个字段 80286 不用，即使用在 PVAM 下也是一样，它为了与 80386 所用的描述符兼容，用户必须总是置此字段为 0。

一旦 80286 处于 PVAM 方式，就没有简单的方法使之回到实地址方式。这是因为附加了所有 PVAM 需要的数据结构之故，使已处于 PVAM 方式下工作的处理器只有在系统重新启动时才能回到实地址方式。事实上，从有存储保护的方式回到实地址方式唯一的方法就是复位 80286。

在 AT 机，复位 80286 一般会清除存储器和寄存器，并重新装入 DOS。但是为让 RAM 磁盘使用 AT 机的扩展存储器(存储地址大于 1MB)，BIOS 提供了功能 87H(成块传送)。87H 可实际转换到 PVAM 方式，把一些信息送到扩展存储器或从扩展存储器里取信息，返回到实地址方式。BIOS 的初始化测试也是用同样方法回到实地址方式。

BIOS 提供的不破坏存储器和寄存器信息的从保护方式转换到实地址方式的方法，调用了 8042 键盘控制器和 RT / CMOS 实时时钟。8042 状态寄存器的第二位是系统标志位。当 80286 一复位，BIOS 初始化程序就测试系统标志位。若标志位为 1，BIOS 就知道复位不是加电或重新引导产生的。为弄清这些是如何进行的，BIOS 检查在 RT / CMOS 时钟关机字节(0FH)。关机字节里大多数可能的数值都是供 BIOS 内部使用的。例如，通知 BIOS 以前转换到 PVAM 方式时就是加电自检的结果，将继续进行下面的测试。不管何种情况，在关机字节总是出现两个代码，一般都“从 PVAM 返回”，这两个码将使 BIOS 在不复位寄存器和存储器的情况下继续进行系统操作。

如果关机字节所设置的数码是 05H 和 0AH，BIOS 继续做系统操作不需重新引导和复位的存储器和寄存器。此时为了继续进行处理，BIOS 将跳到一个指定地址处，此地址存放在绝对地址 467H(是 BIOS 数据区的一部分)开始处的双字里。两个关机码(05H 和 06H)之间的区别是 05H 使 BIOS 在继续工作前清除键盘缓冲区，而 0AH 在继续工作前不对键盘缓冲区作任何改动。

从 PVAM 方式到实地址方式的转换步骤可归纳如下：

1. 将控制需返回的存储单元的段地址(实地址方式)放在绝对地址 467H 开头的双字里。
2. 把 05H 或 0AH 置入 RT / CMOS 实时时钟的关机字节。
3. 把 8042 状态寄存器置 1。
4. 复位 80286。BIOS 送一条命令到 8042 触发 8042 的 0 号输出端口(它连接着系统复位线)。机器在停机指令上循环。

尽管从 PVAM 转换到实地址方式是可能的，但是这种转换和由实地址方式到 PVAM 的操作都应该，也只能由高级程序员来完成。

#### 入口参数：

在调用此功能之前，必须设置下列寄存器：

AH = 89H 指出转移到 PVAM 功能调用号。

BH 置距 IDT 中前 8 个硬件中断开始处的偏移字节数，它们响应一级中断。

BL 置距 IDT 中后 8 个硬件中断开始处的偏移(字节数)，它们响应二级中

断。

**ES: SI** 全局描述符表GDT的指针。它在80286转到PVAM时起作用，用户必须按下面的顺序在 **ES: SI** 之后建立描述符：

相对 EX: SI 位置偏移

00H	伪描述符
08H	GDT 描述符
10H	IDT 描述符
18H	DS 描述符
20H	ES 描述符
28H	SS 描述符
30H	CS 描述符
38H	暂时 BIOS 描述符

所有描述符在建立时都要按上面的格式，下面给出填写说明的字段：

**虚描述符。** 每个描述符表都是从一个虚描述符开始的，在这个描述符中的所有字段初始化时清0。

**GDT 表描述符。** 描述符表象其它任何段一样，也是一个段。因此在表中必须有自己的描述符，使程序能够对它存取和修改。建立这个描述符指向和数据段一样的描述符表。

**IDT 描述符。** 该描述符指向已经建立的 IDT，并定义 IDT 为一数据段。

**DS 描述符。** 该描述符为定义用户程序所需数据段，本处 89H 功能调用时将根据此数值初始化 DS。

**ES 描述符。** 该描述符为定义用户程序所需的额外数据段，89H 功能调用时根据此数值初始化 ES。

**SS 描述符。** 建立此描述符定义用户程序所用堆栈。本处 89H 功能调用时根据此值初始化 SS。

**CS 描述符。** 建立此描述符定义用户程序所用的程序段。本处 89H 功能调用时根据此值初始化 CS。并且当退出 PVAM 方式时使控制转到这个段。

**暂时 BIOS 描述符。** 在处理器转换到 PVAM 方式后，89H 功能需要执行少量指令。因此在 GDT 里必须有其程序的描述符。功能调用时在这个描述符里建立字段，并初始化为0。当用户程序可以接受控制时，就可以任何方式修改此描述符。

**出口参数：**

如果 89H 功能调用成功，控制将转移到描述符表中指定的程序段，AH=0。控制返回后，将禁止所有中断。

**其它要求：**

用户建立的 DT 不能覆盖地址方式下的中断向量表。  
此功能调用改变所有段寄存器的内容，AX 和 BP 也不例外。  
在功能调用期间，禁止所有中断。日历钟在这期间也停止更新，任何其它与中断有关的服务均不能进行。

任何期望在实地址方式和 PVAM 方式运行的程序代码都不能进行远程访问，包括远程调用；任何放在堆栈里的返回地址必须以同样方式形成，使返回程序在此方式下可以运行，或者必须近程返回。

#### 错误信息：

若功能调用成功 AH = 0，AH 中出现非零值就表明功能调用失败。

#### 参见：

中断 15H，功能调用 87H——成块传送。

### BIOS Int 15H

#### AT 机扩充服务

功能调用 90H——设备忙循环（等待）

#### 中断：

15H AT 机扩充服务

#### 功能调用：

90H 设备忙循环（等待）

#### 计算机：

AT

#### 说明：

该功能仅适用于 AT 机，它（和功能调用 90H 一起）使程序可按实地址方式（8088 仿真）支持多任务工作。

微处理器常常因等待设备进行所需的操作（例如磁盘驱动器、打印机或键盘）而出现空闲（有些地方称之为忙循环）。此时，其它程序（有时称任务）可以运行，以实现多任务工作。

BIOS 用功能调用 90H、91H 来指出什么时候这个空闲期出现。它发出功能 90H 通知忙循环开始工作。当忙循环结束（也就是说，当设备最终响相应信息时），BIOS 再发出功能 91H。

因此在 BIOS 里完成上述任务 90H 和 91H 不做什么工作，只简单地执行 IRET 指令使处理器返回到调用程序。但在这里不做工作的功能调用 90H 和 91H 就成为用户对系统追加自己多任务程序的挂钩。

为增加对多任务操作的支持功能，用户必须增加响应功能 90H 和 91H 的中断处理程序（中断句柄）。要做到这点，程序初始化部分必须有 IBM 称之为“挂钩”的 15H 中断。功能调用步骤如下：

- 1、程序读出保存中断 15H 向量。
- 2、然后程序必须为自己的多任务程序选用适当的地址建立中断向量。

3、在中断15H发生时，用户的中断处理程序（中断句柄）获得控制权。它必须测试AH寄存器以便了解哪个功能正被调用。若AH中数值为90H和91H，程序即应执行相应的多任务操作，并在完成之后发出IRET指令使控制返回到调用程序。但是，如果AH内的数值不是90H或91H而是其它值，程序应立即产生一个CALL或JMP指令，将控制转移到中断向量表中原先保存的地址上去。

当BIOS产生一个功能调用90H时，它表明忙循环就要开始了。用户写的支持90H的任何程序都应该保存机器的当前状态（所在寄存器内容），然后开始运行另一个任务。

当然，BIOS提供的支持多任务的任务数量是很少的。因为增加多任务程序就要不断地为各个任务申请路径，或建立优先级系统，来决定哪个任务挂起，哪个任务是下面要运行的任务，并要管理存储器。然而当BIOS调用功能90H时，将在AL寄存器内提供一些产生忙循环的设备类型的信息，了解这个信息是多个程序运行的基础。

BIOS可识别三类设备，即顺序可再用设备、可重入设备和非中断设备。

**顺序可再用设备**（设备类型代码0—7FH）是在一个时刻只能一个任务使用。因此，如果BIOS指出顺序可再用的设备正在进行忙循环，则用户多任务程序就不允许任何其它的任务存取同一设备。例如软盘、硬盘和键盘设备都是顺序可再用设备。

**可重设备**（80H—BFH）是在一个时刻可多于一个任务使用的设备。如果BIOS通知可重入正在进行忙循环，则用户的中断即可让其它任务用这同一设备。网络上的设备是可重入设备，例如硬盘驱动器，若干个用户可同时进行存取。

**非中断设备**（COH—FFH）是完成所要求操作后不产生中断的设备。通常处理器使设备开始工作，是用一条设备命令（同时BIOS调用功能90H）。在设备完成操作时，它产生硬件中断来通知处理器（同时BIOS调用功能91H）。非中断设备不产生表示操作完成中断（BIOS也从不调用功能91H）。对于这种设备，多任务程序必须选择适当的时间结束忙循环，将控制回到原来的程序。有些打印机是非中断设备，软盘驱动器马达也是非中断设备。

此外，多任务程序应知道超时条件。无限等待设备完成操作是不适当的。因此，当等待一段时间后，当前的程序应返回原来的程序并且设备CF=1指出错误信息。

等待时间的长短，一个设备与另一个设备之间各不相同。对于非标准设备，用户必须确定超时值是多少；对于标准设备，设备类型代码和超时值如下所示：

设备类型代码	设备	超时
00H	AT硬盘	6秒
01H	AT软盘	2秒
02H	AT键盘	无
80H	网络	无
FDH	软盘马达启动装置	一个写操作1秒 一个读操作625毫秒
FEH	打印机	由打印机决定

此表未包括异步通信（串行）接口，AT机的异步通信适配器能产生表明操作完成的中断。但是BIOS在那种方式下不支持串行口，所以用户必须增设自己处理多任务程序的串行口。

#### 入口参数：

当BIOS调用功能90H时，必须置位下面寄存器：

AH=90H 指出设备循环忙功能调用号。

AL BIOS设置代表产生忙循环设备类型代码，支持下述各种设备：

00H—7FH顺序可再用设备。

80H—BFH可重入设备。

C0H—FFH非中断设备。

ES: BX 该寄存器对只有在进行忙循环的设备是可重入设备时有效。在那种情况下，ES: BX指出正在使用设备的任务。利用这个信息，用户多任务软件可重新开始另一任务，而不必担心继续做的是同一个已经在等待完成的任务。

#### 出口参数：

无。

#### 其它要求：

用户有责任确保所写的多任务程序中所有设备驱动器代码是顺序方式。因为设备驱动器代码不是可重入的，否则对重入方式将会产生严重错误

#### 参见：

中断15H，功能调用91H——建立中断结束标志(POST)。

### BIOS Int 15H

#### AT机扩充服务

#### 功能调用91H——建立中断结束标志(POST)

#### 中断：

15H AT机扩充服务

#### 功能调用：

91H 建立中断结束标志(POST) 计算机：

AT

#### 说明：

该功能仅适用于AT机，它和功能90H联合使用为用户程序提供多任务程序操作环境。

类似于90H，这个功能调用如设在BIOS之中，只是执行返回调用程序没有更多的操作。这里BIOS利用91H在中断15H里规定用户自己的“挂钩”，详见中断90H功能中关于挂钩中断的说明。

BIOS用功能90H和91H指明了处理器等待设备（磁盘驱动器、打印机等）进行所

需操作时产生。在忙循环期间，用户提供的多任务程序可保存处理器的当前状态，并调用新的任务。

通常，当设备完成 I/O 操作后，设备就送一个中断至处理器。中断产生后，BIOS 置中断标志（称中断完成标志）并发出功能调用 91H，因此工能衰呼就是在忙循环期间多任务程序结束的信号。先前运行的任务现已准备好，又可再次运行了。

但不是所有的设备都能送出中断表示 I/O 操作完成，对于那些不能发中断的设备，用户的多任务程序必须计算什么时候设备操作完成并发出相应的功能调用 91H（或以其它方式处理这种情况）。

为了通知表明设备已完成它的处理的功能调用 91H 处理程序，BIOS 把设备类型码送至 AL 寄存器。普通设备类型，参见入口参数特殊设备表及其代码内容（参见 90H 功能调用中的说明）。

#### 入口参数：

若 BIOS 调用功能 90H，将设置下列寄存器：

AH = 91H 指出建立中断结束标志功能调用号。

AL BIOS 设置表明已完成操作的设备类型代码，现支持的设备码如下：

00H—7FH 顺序可再用设备

80H—BFH 可重入设备

C0H—FFH 非中断设备

ES: BX 仅当完成操作的设备是可重入设备时此寄存器时才有效。在那种情况下，ES: BX 指出正在用此设备工作的任务。用户多任务软件根据此信息可重新开始执行任务，而不必担心任务仍在等待来自该设备的信息。

#### 出口参数：

无。

#### 其它要求：

用户编写的多任务程序应确保所有设备驱动器代码都是顺序方式，设备驱动器代码是不可重入的，否则用重入方式就会出现严重错误。

#### 参见：

中断 15H，功能调用 91H——设备忙循环（等待）。

### BIOS Int 16H

#### 键盘 I/O

#### 中断：

16H 键盘 I/O

#### 说明：

通过这个中断所调用的 BIOS 功能调用将检查是否有键盘输入并接收键盘输入字符。与执行同样操作的 DOS 功能调用不同的是，BIOS 功能调用可以跳过一些 DOS 自动执行的操作处理。例如，如果键入 Ctrl-C，DOS 字符处理程序（character-handling routines）自动认为是终止程序的特殊符号，然而，BIOS 功能调用在接收到这种字符时不做

这种假定。

要通过适当的设备 AH 寄存器的数值并且发出中断 16H 选择所要的功能调用。这个中断所提供的功能调用如下：

AH 功能调用

00H 读下一个键盘字符。

01H 检测输入的字符是否准备好。

02H 得到当前转换键 (shift) 的状态。

下面对键盘 I/O 进行详细说明。

## BIOS Int 16H

### 键盘 I/O

功能调用 00H——读下一个键盘字符

中断：

16H 键盘 I/O

功能调用：

00H 读下一个键盘字符

计算机：

PC, PCjr, XT, Portable 及 AT

说明：

这个功能调用用于从键盘读取一个字符。如果字符已经键入并被放在 BIOS 的键盘缓冲区中，就立即返回这个字符。否则这个功能调用将等待直到有字符输入时为止。

这个功能调用返回所输入字符的两部分信息，即其 ASCII 码和扩展码。ASCII 码是在许多编程语言和计算机系统中所用字符的通用标准代码。而扩展码则是 BIOS 用于标识专用键及组合键的特殊代码。

同时返回 ASCII 及扩展代码有两个原因。首先，并非 IBM 键盘上的所有键都对应 ASCII 字符。对于这些字符及字符与 shift, Ctrl 或 Alt 键的组合，返器的 ASCII 码为 0，仅由扩展码区分这些字符。其次，在 IBM 键盘上有些键，如换档键 (Shift) 和星号 (\*) 以及每个数字键都是两个。它们的 ASCII 码相同，但扩展码不同。通过返回 ASCII 码和扩展码，可确切标识所敲入的键。例如，可区分接收的数字键盘扫描码是从上部还是从右侧小键盘上的数字键输入的。

在大多数情况下，由这个功能调用返回的扩展码是敲基本键的键盘扫描码。键盘扫描码是 IBM 键对应所敲键发送的特殊代码。键盘扫描码仅用于单键。没有用于区分大写 (Shifted) 字符或在按 Ctrl 及 Alt 键的同时键入的字符的代码。在大多数情况下，为了确定 Shift, Ctrl 或 Alt 键是否与其它键一起输入，就必须使用功能调用 02H (取当前转换键状态)。

图 5-1 和 5-2 是两种标准的 IBM 键盘的键盘扫描码。图 5-1 给出的键盘适用于 IBM PC, XT 和便携式计算机。图 5-2 给出的是 AT 键盘。

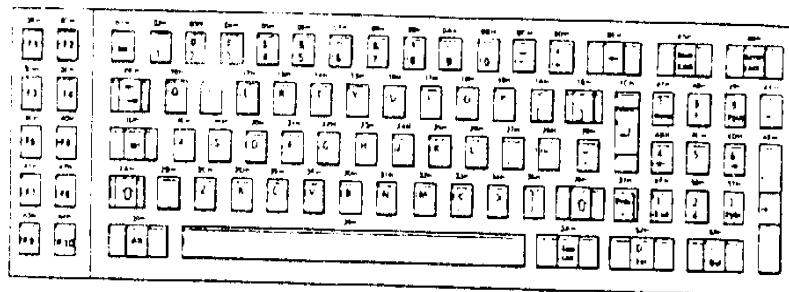


图 5-1 IBM PC, XT 和便携式键盘的扫描码

对于一些组合键，即当同时键入标准键及转换键（不管是 shift, Alt 或 Ctrl 键）时，这个功能调用返回的扩展码与标准的键盘扫描码不同。表 5-2 给出了组合键及扩展代码。

如果用户程序并不想等待键盘的输入，可引用功能请求 01H（检测输入的字符是否准备好）来差别是否有字符在等待接收。

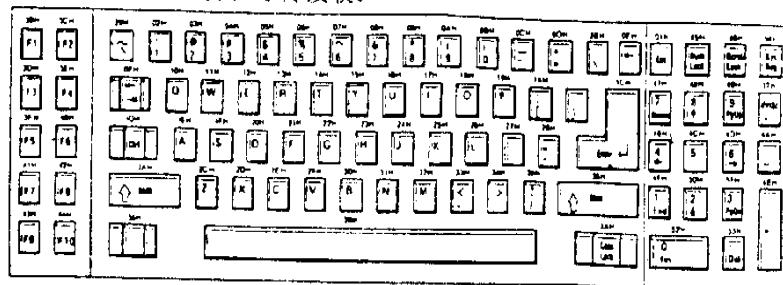


图 5-2 IBM AT 键盘扫描码

扩展代码	字符组合
03H	Nul 字符
54H-5DH	shift+f1-f10 键
5EH-67H	ctrl+f1-f10 键
68H-71H	alt+f1-f10 键
72H	ctrl+prtsc 键
73H	ctrl+左移键
74H	ctrl+右移键
75H	ctrl+End 键
76H	ctrl+PgDn 键
77H	ctrl+Home 键
78H-83H	Alt+1,2,3,4,5,6,7,8, 9,0,-, = 键
84H	ctrl+PgDp 键

表 5-2 特殊的键组合的扩展代码

### 入口参数:

调用前, 需设置:

AH = 00H 指定读下一个键盘字符功能调用。

### 出口参数:

只有当键盘有字符输入时控制才返回。当控制返回后, 设置下列寄存器的数值:

AL 存放所敲键的ASCII码, 如果键不对应256个ASCII字符中的某一个, 这个寄存器的数值为0。

AH 这个寄存器含有字符的扩展码。其值或者是如图5-1或5-2所给出的键盘扫描码, 或者是如表5-2所示的扩展码。

### 参见:

中断16H, 功能请求01H——检测输入字符是否准备好。

中断16H, 功能请求02H——取当前转换键的状态。

## BIOS Int 16H

### 键盘I/O

#### 功能调用 00H——检测字符是否准备好

### 中断:

16H 键盘I/O

### 功能调用:

01H 检测字符是否准备好

### 计算机:

PC, PCjr, XT, Portable 及 AT

### 说明:

这个功能调用检测键盘是否有键入的字符, 即 BIOS 键盘缓冲区中是否已有字符。这个功能调用立即返回结果并且可用做为引用功能调用00H(读下一个键盘字符)之前的检测。

这个功能调用零标志(ZF)指明是否有键盘输入。如果ZF等于0, 表示且有一个字符可读; 如果ZF等于1, 则表示没有键盘输入字符。虽然这些设置看起来有点落后, 但它使程序能使用判别标志是否为零的跳转指令来决定是否读键盘字符。

另外, 可用此功能预先查看键盘缓冲区中的字符。AH和AL寄存器的数值用于给出字符的ASCII码和扩展码。然而, 这个功能调用并非取走字符, 而以后的功能调用00H(读下一个键盘字符)才取走这里返回的同一个字符, 并从键盘缓冲区清除这个字符。

### 入口参数:

调用前, 需设备:

AH = 01H 指出“检测下一个字符是否准备好”功能调用号。

### 出口参数:

返回后, 设置下列数值:

- ZF** 这个ZF标志用于标识从键盘输入的字符是否准备好，其数值的意义如下：
- 0** 表示已有一个字符可接收。
  - 1** 键盘缓冲区中没有字符。
- AL** 这个寄存器含有下一个待接收字符的ASCII码。如果按键不对应256个ASCII字符中的一个，这个寄存器的数值为零。
- AH** 这个寄存器含有待接收字符的扩展码。这个值或者是图5-1或图5-2所示的键盘扫描码，或者是表5-2所示的扩展码。

#### 参见：

- 中断16H，功能调用00H——读下一个键盘字符。
- 中断16H，功能调用02H——取当前转换键的状态。

### BIOS Int 16H

#### 键盘I/O

#### 功能调用02H——取当前转换键状态

#### 中断：

16H 键盘I/O

#### 功能调用：

02H 取当前转换键状态

#### 计算机：

PC, PCjr, XT, Portable 及 AT

#### 说明：

这个功能请求用于报告在打入最后一个键时转换键（Shift, Alt, Ctrl, Numlock 和 Scroll Lock 键）的状态，因为返回的键盘扫描码以及许多字符并不由于按下这些键而改变。程序只有通过这个功能调用所报告的转换键状态才能确切得知所打入键的组合情况。

#### 入口参数：

调用前，需设置：

AH=02H 指出取当前转换键状态功能调用号。

#### 出口参数：

在控制返回之后，设备下列寄存器的数值：

**AL** 这个寄存器用于标识转换键的状态。每一位对应一种转换键。寄存器各位的编码表示如下：

位

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

位	说明
7	输入状态(Insertstate),按 Ins 键触发该位开和关。 虽然它可用于跟踪字处理程序的插入模式,但实际上用这个比特位的程序很少.
6	Caps Lock,按 Caps Lock 键触发这个位的设置.
5	Num Lock,按 Num Lock 键触发这个位的设置.
4	Scroll Lock,按 Scroll Lock 键触发这个位的设置.
3	Alt 键,如果在上一个字符输入时按这个键,此位置 1.
2	Ctl 键,如果当上一个字符输入时按这个键,此位置 1.
1	左边的 Shift 键,如果当上一个字符输入时,键盘左手旁的 Shift 键被压下,此位置 1.
0	右边的 Shift 键,如果当最后一个字输入时,键盘右手旁的 Shift 键被压下,此位置 1.

## 参见:

中断 16H, 功能调用 00H——读下一个键盘字符。

中断 16H, 功能调用 01H——检测输入的字符是否准备好。

## 程序实例:

下面的每一个程序实例都使用了功能请求 02H 来检测转换键的设备情况, 每一个键的状态被显示在屏幕上。

### Pascal Usage Example:

```
{Get Current Shift Status ($16)}
```

```
PROGRAM get_current_shift_status;
```

```
CONST
```

```
key_shift_status = $02;
```

```
TYPE
```

```

regpack = RECORD
    CASE integer OF
        1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
        2 : (al,ah,bl,bh,cl,ch,dl,dh:byte);
    END;

VAR
    regs : regpack ;

BEGIN
    WITH regs DO
        BEGIN
            ah := get_shift_status ,
            intr(keyboard_io,regs) ;
            IF (al AND $80) < > THEN
                writeln( ' Insert is on ' )
            ELSE
                writeln( ' Insert is off ' );
            IF (al AND $40) < > THEN
                writeln( ' Caps Lock is on ' )
            ELSE
                writeln( ' Caps Lock is off ' );
            IF (al AND $20) < > THEN
                writeln( ' Num Lock is on ' )
            ELSE
                writeln( ' Num Lock is off ' );
            IF (al AND $10) < > THEN
                writeln( ' Scroll Lock is on ' )
            ELSE
                writeln( ' Scroll Lock is off ' );
            IF (al AND $08) < > THEN
                writeln( ' Alt key pressed ' )
            ELSE
                writeln( ' Alt key not pressed ' );
            IF (al AND $04) < > THEN
                writeln( ' Ctrl key pressed ' )
            ELSE
                writeln( ' Ctrl key not pressed ' );
            IF (al AND $02) < > THEN

```

```

        writeln( ' Left Shift pressed' )
ELSE
        writeln( ' Left Shift not pressed' );
IF (al AND $01) <> THEN
        writeln( ' Right Shift pressed' )
ELSE
        writeln( ' Right Shift not pressed' );
END;
END.

```

### C Usage Example:

```

/* * -> Get Current Shift Status (0x16)
 */

#include <dos.h>

union REGS intregs, outregs;

main()
{
    intregs.h.ah := 0x02;
    int86(0x16,&intregs,&outregs);
    if (outregs.h.al & 0x80)
        printf( " Insert is on\n" );
    else printf( " Insert is off\n" );
    if (outregs.h.al & 0x40)
        printf( " Caps Lock is on\n" );
    else printf( " Caps Lock is off\n" );
    if (outregs.h.al & 0x20)
        printf( " Num Lock is on\n" );
    else printf( " Num Lock is off\n" );
    if (outregs.h.al & 0x10)
        printf( " Scroll Lock is on\n" );
    else printf( " Scroll Lock is off\n" );
    if (outregs.h.al & 0x08)
        printf( " Alt key pressed\n" );
    else printf( " Alt key not pressed\n" );
    if (outregs.h.al & 0x04)
        printf( " Ctrl key pressed\n" );

```

```

else printf( " Ctrl key not pressed\n" );
if (outregs.h.al & 0x02)
    printf( " Left Shift pressed\n" );
else printf( " Left Shift not pressed\n" );
if (outregs.h.al & 0x01)
    printf( " Right Shift pressed\n" );
else printf( " Right Shift not pressed\n" );
}

```

### **BIOS Int 17H**

#### **打印机 I/O**

**中断:**

17H 打印机 I/O

**说明:**

通过这个中断所引用的 BIOS 功能调用使用户能对打印机控制输出。这种服务比对应的 DOS 功能调用用途更多，因为它允许指定究竟和哪一个打印机打交道。

可通过设置适当的 AH 寄存器的数值并且发出中断 17H 而选择所希望的功能调用。这个中断所提供的功能调用如下：

AH	功能调用
00H	打印一个字符
01H	初始化打印机口
02H	取打印机状态

下面对这些打印机 I/O 功能调用进行详细描述。

### **BIOS Int 17H**

#### **打印机 I/O**

**功能调用 00H——打印一个字符**

**中断:**

17H 打印机 I/O

**功能调用:**

00H 打印一个字符

**计算机:**

PC, PCjr, XT, Portable 及 AT

### 说明:

这个功能调用发送一个字符到打印机。该功能调用将等到打印机不忙或打印机超时到期为止。

### 入口参数:

调用前, 需设置:

AH = 00H 指出打印一个字符功能调用号。

AL 将要打印字符的ASCII码放入这个寄存器。

DX 设置这个寄存器为0, 1或2, 用于指定将字符发送到哪一个并行打印机。

### 出口参数:

当控制返回时, 下列寄存器报告执行状态:

AH 用于给出操作状态代码。通过寄存器不同位设置为1来指示相应的状态条件, 各位的代码如下:

位

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

位	说明
7	打印机准备好
6	对最后一个字符的确认。
5	无纸。
4	选择打印机
3	I/O 错。
2-1	未用。
0	超时错

### 程序实例:

下面的每一个程序实例都使用功能调用 00H, 发送字符到打印机。发送字符“Message for printer”及回车、换行符到打印机。

#### Pascal Usage Example:

```
{ Print a Character ($17) }
```

```
PROGRAM print_character;
CONST
  printer_io = $17;
  printer_character = $00;
```

```

TYPE
  regpack = RECORD
    CASE integer OF
      1 : (ax,bx,cx,dx,bp,si,di,ds,es,flags:integer)
      2 : (al,ah,bl,bh,cl,ch,dl,ch:byte);
    END;

VAR
  regs : regpack;
  mag : string [80];
  i : integer;
BEGIN
  WITH regs DO
    BEGIN
      mag := ' Message for printer.' + chr($0d) + chr($0a);
      FOR i := 1 TO length(mag) DO
        BEGIN
          ah := print_character;
          al := ord(mag[i]);
          dx := 1;
          intr(printer__io,regs);
        END;
    END;
END.

```

#### C Usage Example:

```

/* *
+ * Print a Character (0x17)
 */

```

#include <dos.h>

```

union REGS inregs,outregs;

char msg[] = " Message ofr printer.\n";
main()
{
  int i;

```

```

i = 1;
while (msg[i] != 0){
    inregs.h.ah = 0x00;
    inregs.h.al = msg[i];
    inregs.x.dx = i;
    int86(0x17,&inregs,&outregs);
    ++i;
}

```

## BIOS Int 17H

打印机 I/O

功能调用 01H——初始化打印机口

**中断:**

17H 打印机 I/O

**功能调用:**

01H 初始化打印机口

**计算机:**

PC, PCjr, XT, Portable 及 AT

**说明:**

这个功能调用通过将控制码 08H 和 0CH 发送到打印机控制端口 (端口地址 2FSH) 来初始化打印机。这个功能调用将复位打印机并设置页的顶部为当前打印纸的位置。

**入口参数:**

调用前, 需设置:

AH = 01H 指出初始化打印机口功能调用号。

DX 设置这个寄存器为 0, 1 或 2 选择初始化哪一个并行打印机。

**出口参数:**

当控制返回时, 下列寄存器报告操作状态。

AH 这个寄存器用于给出操作状态, 通过寄存器的各位置为 1 指示有关的状态, 寄存器各位的编码如下:

位

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

位	说明
7	打印机准备好
6	对最后一个字符的确认.
5	无纸.
4	选择打印机
3	I/O 错.
2-1	未用.
0	超时错

## BIOS Int 17H

打印机 I/O

功能调用 02H——获取打印机状态

中断:

17H 打印机 I/O

功能调用:

02H 获取打印机状态

计算机:

PC, PCjr, XT, Portable 及 AT

说明:

这个功能调用用于获取打印机的当前状态。因为这个功能调用是立即返回的，程序可先发出这个功能调用检查打印机是否准备好接收字符。当引用功能调用 00H (打印一个字符) 时，就不必等待。

本功能调用返回的状态与功能调用 00H 和 01H 相同。

入口参数:

调用前，需设置:

AH = 02H 指出取打印机状态功能调用号。

DX 设置这个寄存器为0, 1或2, 选择检测哪一个并行打印机的状态。

出口参数:

当控制返回时，下列寄存器报告操作的状态。

AH 这个寄存器用于指出操作状态。通过寄存器各位设置为1给出有关的状态。寄存器各位的编码如下:

位

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

位	说明
7	打印机准备好
6	对最后一个字符的确认。
5	无纸。
4	选择打印机
3	I/O 错。
2-1	未用。
0	超时错

## BIOS Int 18H

ROM BASIC 语言

中断:

18H ROM BASIC

计算机:

PC, PCjr, XT, Portable 及 AT

说明:

这个中断用于启动 PC 机的 ROM BASIC 解释器。在初始化期间，指向 ROM 中 BASIC 解释器的指针填写在关于该中断的向量表中。如果计算机启动时，ROM 引导程序在驱动器 A 中没有发现磁盘或没有发现可接受的包含 DOS 的硬盘，则 ROM 中的程序发出中断 18H 进入 ROM BASIC。

将指向 ROM BASIC 解释器入口的指针放在中断向量表入口，而取代直接调用有两个原因：首先，如果需要的话，用户程序也可以发送中断 18H 而引用 ROM BASIC；其次如果想用驻留在内存的不同版本的 BASIC 取代 ROM BASIC，仅需要简单地改变中断向量表中的入口即可。然后选择所有正常访问 ROM BASIC 的程序也都能访问替换的 BASIC。

入口参数:

为了引用 ROM BASIC，仅简单地发送中断 18H 即可。

出口参数:

进入 ROM BASIC 的控制。

## **BIOS Int 19H**

**引导加载程序**

**中断:**

19H 引导加载程序

**计算机:**

PC, PCjr, XT, Portable 及 AT

**说明:**

这个中断引用 PC 机的引导加载服务程序，用于启动计算机的初始化。因而，这个中断服务所提供的功能与加电启动计算机或同时按下 Ctrl-Alt-Del 三键复位时计算机系统所提供的服务类似，但并非完全相同。

当加电启动计算机时，BIOS 执行一系列被称作 POST 加电自检的测试，用于检验机器的各个部件，有时包括耗时的存储器测试。用 Ctrl-Alt-Del 三键复位计算机时，则可避免 POST。但它复位和重新初始化计算机的存储器，包括中断向量表和 BIOS 用于存储关于计算机的信息（如存储器大小和设备清单）的 256 个字节（从 400H 至 4FFH）。直接发送中断 19H，可避免 POST 和存储器初始化。这个中断服务停止计算机所做的一切工作，并重新装入加载程序，但它假定 POST 及对存储器地址 400H 至 4FFH 的设备都已顺利完成。

这个中断对那些需要对 DOS 来隐藏资源的程序尤其有用，如 RAM 盘。先确立一个内存区域作为 RAM 盘，再向下调整存放在地址 413H 处，表示 RAM 存储区尺寸的数值，使 DOS 认为这部分内存空间不存在，然后用中断 19H 重新启动计算机。这样，DOS 认为系统仅有 RAM 存储区尺寸所说明的内存量。RAM 盘对隐藏的存储器具有专用权，不用担心 DOS 将它分配给被装载的程序。

**入口参数:**

为了调用引导加载程序，只需简单地发送中断 19H。

**出口参数:**

当前的程序终止，并且 BIOS 重新装入 DOS。

## **BIOS Int 1AH**

**实时时钟**

**中断:**

1AH 实时时钟 (Time of Day)

**说明:**

BIOS 通过 RAM 中地址 46CH 起始的四字节计数器保存实时时钟。BIOS 大约每秒发送中断 08H 18.2 次（实际上是每秒 1193180 / 65536 次）调用服务程序将计数器加 1。如果时间设置准确，这个四字节的计数器每天的 0 点为 0，午夜（24 点）为 1800B0H。

在午夜，当计数器为 1800B0H 时，BIOS 将计数器归 0，并置地址 470H 单元中内容为 1，表示从计算机加电启动以来新的一天已经开始。这个机制允许 DOS 测定当前的日

期和时间。

当 DOS 开始运行时，显示请用户设置日期和时间的信息（或取决于 AUTOEXEC.BAT 文件，它从电池供电时钟中读取日期和时间或假定日期是 1980 年 1 月 1 日午夜）。根据此信息设置从 46CH 开始的四个字节的计数器值作为起始时间，并且将单元中 470H 翻转字节（rollover byte）设置为 0。然后 BIOS 开始接管，增加四字节计数器的数值使其为当前时间。

当 DOS 想知道实时时钟时，它把从 46CH 开始的四个字节计数器数值转换为可懂的时间值，并检查翻转字节决定是否应增加日期的数值。然而 DOS 并不知道自从它最后一次从时钟读时间以来已经过了多少天。所以，如果计算机空转几天，经过多个午夜，如过一个周末，那么时钟将丢失全部的天数。

通过这个中断所引用的功能调用允许读取 / 设置由 BIOS 所保持的时间。前两个功能调用适用于所有 IBM 机器，用于读取 / 设置当前时钟计数值，后六个功能调用仅适用于 AT 机，用于读取 / 设置 AT 机的可充电电池时钟或设置闹钟。

可通过在 AH 寄存器中设置不同的数值而选择所需功能调用，然后发送中断 1AH。这个中断所提供的功能调用如下：

AH	功能调用
00H	读取当前时钟值。
01H	设置当前时钟值。
02H	从电池供电时钟中读时间。
03H	设置电池供电时钟的时间。
04H	从电池供电时钟中读日期。
05H	设置电池供电时钟的日期。
06H	设置闹钟。
07H	复位闹钟。

下面对这些功能调用进行详细说明。

### BIOS Int 1AH

实时时钟

功能调用 00H——读取当前时钟值

中断：

1AH 实时时钟

功能调用：

00H 读当前时钟值

计算机：

PC, PCjr, XT, Portable 及 AT

**说明:**

这个功能调用读取 RAM 地址 46CH 中保存的四个字节时钟数值，并检查 RAM 地址 470H 中翻转字节是否被设置。该字节指明从上一次检查钟以来是否已过了一天。如果翻转字节设置，功能调用在 AL 寄存器返回一个非零值，并且复位翻转字节为 0。

因为这个功能调用复位翻转字节，在使用这个功能调用时应该小心。因 DOS 依赖翻转字节了解日期的变化，如果用户程序调用功能调用 00H，结果复位了翻转字节，DOS 并不知道日期已经改变，DOS 记录结果就会出错。

因在翻转字节中仅一位置 1，如果最近一次检查时钟后过了好几天，计算机就无法知道准确的天数。当机器空度周末时就会发生这种情况。回来时，时钟将滞后一天。

**入口参数:**

调用前，需设置：

AH=00H 指出读当前时钟值功能调用号。

**出口参数:**

当控制返回时，设置下列寄存器的数值：

CX 存放四字节时钟值的高位部分（高有效的两个字节）。

DX 存放四字节时钟值的低位部分（低有效的两个字节）。

AL 存放翻转字节（是否是新的一天的开始）的状态，表示如下：

0 地址 470H 中的翻转字节未被设置。从时钟上一次被读以来，日期没有改变。

非零翻转字节被设置，指明当前的时间是新的一天的时间，在这种情况下，这个功能调用复位在 470H 单元中的翻转字节为 0。

**参见:**

中断 1AH，功能调用 01H——设置当前时钟值。

**BIOS Int 1AH**

实时时钟

功能调用 01H——设置当前时钟值

**中断:**

1AH 实时时钟

**功能调用:**

01H 设置当前时钟值

**计算机:**

PC, PCjr, XT, Portable 及 AT

**说明:**

这个功能调用设置 RAM 地址 46CH 开始保存的四个字节时钟计数值。但它并不调整 470H 单元中翻转字节的数值。

**入口参数:**

调用前，需设置：

AH=01H 指出置当前时钟功能调用号。

CX 设置这个寄存器为四字节时钟值的高位部分（高有效的两个字节）。

DX 设置这个寄存器为四字节时钟值的低位部分（低有效的两个字节）。

出口参数：

RAM 中地址 46CH 开始的四字节时钟值将做相应的设置。

参见：

中断 1AH，功能调用 00H——读取当前时钟值。

**BIOS Int 1AH**

**实时时钟**

**功能调用 02H——读取电池供电时钟时间**

**中断：**

1AH 实时时钟

**功能调用：**

02H 读电池供电时间

**计算机：**

AT

**说明：**

这个功能调用仅适用于 AT 机。它用于从 AT 机的电池时钟中读取时间。返回的时间并不象功能调用 00H 和 01H 那样以计数器值的形式，而是以时分秒的形式给出。

这个功能调用返回当前时间在 CH, CL 和 DH 寄存器中，但不用这个时间去设置由 BIOS 在 RAM 中所存放的时钟数值。

**入口参数：**

调用前，需设置：

AH=02H 指出读电池供电时钟时间功能调用号。

**出口参数：**

当控制返回时，设置下列寄存器的数值：

CH 从电池供电时钟读取的当前时间的小时数（二进制编码十进制数 0-23）。

CL 从电池供电时钟读取的当前时间的分钟数（二进制编码十进制数 0-59）。

DH 从电池供电时钟读取的当前时间的秒数（二进制编码十进制数 0-59）。

**错误信息：**

如果电池供电时钟没有运行，进位标志（CF）将设置为 1；其它情况下，CF 为 0。

**参见：**

中断 1AH，功能调用 03H——置电池供电时钟时间。

中断 1AH，功能调用 04H——读电池供电时钟日期。

中断 1AH, 功能调用 05H——设置电池供电时钟日期.

DOS 功能调用 2DH——置时间.

### BIOS Int 1AH

实时时钟

功能调用 03H——设置电池供电时钟时间

中断:

1AH 实时时钟

功能调用:

03H 设置电池供电时钟时间

计算机:

AT

说明:

这个功能调用仅适用于 AT 机, 它用于改变 AT 机电池供电时钟的时间. 程序可以在更换新电池时使用这个功能调用, 也可以在改变电池供电时钟的时间时使用(如移到一个新的时区).

这个功能调用也可选择是否夏令时调整电池时钟调整时间, 当选择这个功能时, 电池时钟自动在四月最后一个星期天的早晨 2 点向前调一个小时; 在十月最后一个星期天的早晨 2 点向回调一个小时. 当然, BIOS 所保存在 RAM 中的时间不会自动改变. 但这个功能调用可用于从电池供电时钟中得到新的时间.

这个功能调用不设置 BIOS 保存在 RAM 中的时钟计数值, 仅改变电池供电时钟.

入口参数:

调用前, 需设置:

AH=03H 指出设置电池供电时钟时间功能调用号.

CH 设置这个寄存器用于指明要设置时间的小时数(二进制编码十进制数 0-23).

CL 设置这个寄存器用于指明要设置时间的分钟数(二进制编码十进制数 0-25).

CH 设置这个寄存器用于指明要设置时间的秒数(二进制编码十进制数 0-59).

DL 如果调整夏令时时钟, 设置这个寄存器为1. 如果设为0则忽略夏令时方式.

出口参数:

电池时钟时间相应的设置.

参见:

中断 1AH, 功能调用 02H——读取电池供电时钟时间.

中断 1AH, 功能调用 04H——读取电池供电时钟日期.

中断 1AH, 功能调用 05H——设置电池供电时钟日期.

## **BIOS Int 1AH**

**实时时钟**

**功能调用 00H——读取当前时钟值**

**中断:**

1AH 实时时钟

**功能调用:**

04H 读取电池供电时钟日期

**计算机:**

AT

**说明:**

这个功能调用仅适用于 AT 机，用于从 AT 机的电池供电时钟中读取日期。但它并不自动使用这些数据设置由 DOS 保持的日期。

**入口参数:**

调用前，需设置：

AH=04H 指出读取电池供电时钟日期功能调用号。

**出口参数:**

当控制返回时，将设置下列寄存器的数值：

CH 这个寄存器存放从电池供电时钟读取的当前日期的世纪值（二进制编码十进制数 19 或 20）。

CH 这个寄存器存放从电池供电时钟读取的当前日期的年度值（二进制编码十进制数 00-99）。

CH 这个寄存器存放从电池供电时钟读取的当前日期的月份值（二进制编码十进制数 0-12）。

CH 这个寄存器存放从电池供电时钟读取的当前日期的日期值（二进制编码十进制数 0-28, 29, 30 或 31）。

**错误信息:**

如果电池供电时钟没有工作，进位标志 (CF) 将置为 1，其它情况下为 0。

**参见:**

中断 1AH，功能调用 02H——读取电池供电时钟时间。

中断 1AH，功能调用 03H——设置电池供电时钟时间。

中断 1AH，功能调用 04H——设置电池供电时钟日期。

DOS 功能调用 2BH——设置日期。

## **BIOS Int 1AH**

**实时时钟**

**功能调用 05H——设置电池供电时钟日期**

**中断:**

1AH 实时时钟

**功能调用:**

05H 设置电池供电时钟日期

**计算机:**

AT

**说明:**

这个功能调用仅适用于 AT 机。用于改变 AT 机电池供电时钟的日期值。程序可以在更换新电池后或者发现日期不对时使用这个功能调用。

这个功能调用并不设置由 DOS 保持的日期，它仅改变电池供电时钟。

**入口参数:**

调用前，需设置：

AH = 05H 指出置电池供电时钟日期功能调用号。

CH 存放要设置的正确日期的世纪值（二进制编码十进制数19或20）。

CL 存放要设置的正确日期的年度值（二进制编码十进制数00—99）。

CH 存放要设置的正确日期的月份值（二进制编码十进制数0—12）。

CH 存放要设置的正确日期的日期值（二进制编码十进制数0—28, 29, 30或31）。

**出口参数:**

电池供电时钟被设置成相应的值。

**参见:**

中断 1AH, 功能调用 02H——读取电池供电时钟时间。

中断 1AH, 功能调用 03H——设置电池供电时钟时间。

中断 1AH, 功能调用 04H——读取电池供电时钟日期。

## **BIOS Int 1AH**

**实时时钟**

**功能调用 06H——设置闹钟**

**中断:**

1AH 实时时钟

**功能调用:**

06H 设置闹钟

**计算机:**

AT

**说明:**

这个功能调用仅适用于 AT 机，用于设置闹钟功能。闹钟功能可从引用这个功能调用时起，23 小时 59 分和 59 秒内任何时间里导致中断 4AH 发生。

当闹钟触发时，中断 4AH 发生，导致中断向量表内中断 4AH 所在位置中所指地址的程序运行。没有缺省的闹钟程序，但可以创建任何有用的程序，将它的入口地址放入中断向量表中断 4AH 所在的相应位置上，当闹钟触发时就会运行这个程序。例如，可以写一个简单的蜂鸣程序，将计算机变为真正的闹钟，或者可以做一些更有意义的工作，如在午夜时启动一磁带备份程序。

仅能设置一个闹钟时间。为了改变闹钟时间，首先使用功能调用 07H 复位闹钟，然后引用功能调用 06H，再设置不同的时间。

**入口参数:**

调用前，需设置：

AH = 06H 指出闹钟功能调用号。

CH 将寄存器设置为从当前时间开始到触发闹钟之间的小时数（二进制编码十进制数 0—23）。

CH 将寄存器设置为从当前时间开始到触发闹钟之间的分钟数（二进制编码十进制数 0—59）。

CH 将寄存器设置为从当前时间开始到触发闹钟之间的秒数（二进制编码十进制数 0—59）。

**错误信息:**

如果电池供电时钟没有工作，进位标志（CF）将置为 1，其它情况下则设置为 0。

**参见:**

中断 1AH，功能调用 07H——复位闹钟。

**BIOS Int 1AH**

实时时钟

功能调用 07H——复位闹钟

**中断:**

1AH 实时时钟

**功能调用:**

07H 复位闹钟

**计算机:**

AT

**说明:**

这个功能调用仅适用于 AT 机。它用于关闭（复位）闹钟，如果仅想关闭闹钟或者为将闹钟设为不同的数值做准备（因闹钟在能够设置不同的新数值之前必须复位）就可以引用这个功能调用。

**入口参数:**

调用前, 需设置:

AH=07H 指出复位闹钟功能调用号.

**出口参数:**

闹钟被复位.

**参见:**

中断1AH, 功能06H——设置闹钟.

## **BIOS Int 1BH**

**键盘终止地址**

**中断:**

1BH 键盘终止地址

**计算机:**

PC, PCjr, XT, Portable 及 AT

**说明:**

每当在键盘上打入 Ctrl-Break (按下 Ctrl 键的同时按下 Break 键) 时, BIOS 就发生这个中断. 起动后, BIOS 将对应这个中断的中断向量设为指向一个 IRET 指令. 这样当中断发生时, 就能不执行任何操作将控制立即返回到以前的程序.

当 DOS 装入时, 它有自己处理 Ctrl-Break 序列的中断. 它将它自己的中断处理程序地址插入到这个中断向量中, 那个中断处理程序中断句柄只发出 DOS 中断 23H 而调用 DOS 的 Ctrl-Break 处理程序取代 BIOS 的伪处理程序. 处理程序可以执行用户需要的任何操作, 但应发出 IRET 指令或调用或转移到地址原先就列在中断向量表中的程序返回.

## **BIOS Int 1CH**

**定时器信号**

**中断:**

1CH 定时器信号 (Timer tick)

**计算机:**

PC, PCjr, XT, Portable 及 AT

**说明:**

在每次系统时钟信号 (每秒钟 18.006481 次) 出现时都执行由这个中断向量所指向的程序代码. 起初, BIOS 将这个中断向量指向 IRET 指令. 这样, 当发生中断时, 不用执行任何操作就可将控制立即返回到以前的程序.

如果用户想在每一时钟信号出现时执行其它的操作, 可以用自己的处理程序替换这个 BIOS 伪中断处理程序. 在处理程序中可执行用户需要的任何操作. 然而, 为了与其它在每个时钟信号时也需操作的程序共存, 当用户的程序与这个中断相挂时, 操作步骤正确无误, 可按如下步骤:

- 1、用户程序的初始化部分必须读取和保存与中断 1CH 有关的中断向量。
- 2、用关于程序本身的适当地址建立中断向量。
- 3、当中断1CH发生时，进入用户中断处理程序控制，执行完操作后发出CALL或JMP指令将控制转给步骤1中所保存的中断向量表的地址。

如果所有需要使用中断 1CH 的程序都按上述要求链接在一起，就不会发生明显的不兼容，所有的程序都可共享这个中断。然而，如果一个中断处理程序使用 IRET 指令取代 CALL 或 JMP 指令返回控制，那么控制将返回到主应用程序上去，而链上的其它处理程序（这些程序可能安装的比较早）将得不到控制权。

## BIOS Int 1EH

### 软盘参数表

#### 中断：

1EH 软盘参数表

#### 计算机：

PC, PCjr, XT, Portable 及 AT

#### 说明：

这个中断向量指向列出计算机如何存取软盘驱动器的表。该表最初由 BIOS 建立，但 DOS 将向量改变为指向它自己的表。软盘基本表的字段如下（这里字节 0 是表的第一个字节）：

偏移量	说明
字节 0	软盘控制器的技术说明。高四位是步进时间 (step-rate time) (磁道之间移动磁头所需时间)，取值范围为 1-6 毫秒。低四位是磁头卸载时间 (取值范围为 16-240 毫秒，增量为 16 毫秒)。
字节 1	仍是软盘控制器的技术说明。高七位是磁头加载时间 (取值范围为 2-254 毫秒，增量为 2 毫秒)。最低一位用于指定 DMA 方式 (0 表示 DMA 方式，1 表示非 DMA 方式)。
字节 2	在每一个操作完成后，磁盘驱动器马达仍保持运转的时钟信号数。这样，如果很快做另一个操作，就不用再启动马达，缺省值是 25H 个时钟信号数，大约为 2 秒钟。
字节 3	扇区长度代码 (0-128, 1-256, 2-512, 3-1024)。通常这个值为 2，但如果使用不同长度的扇区，就必须改变这个值。
字节 4	磁道上最后一个扇区的记录数。对于 320K 的软盘 (DOS1.1) 这个值是 8，对于 360K 的软盘，这个值是 9。
字节 5	在读写扇区时 BIOS 所假定的扇区间的间隔。缺省值是 2AH。
字节 6	数据传输长度。当扇区尺寸没有指定时，设置为数据传输的最大长度。

偏移量	说明
字节 7	当格式化扇区时, BIOS 所假定的扇区间的间隔尺寸, 缺省值是 50H.
字节 8	当格式化软盘时, 存在磁道上的每个字节中的数据, 缺省值是 ASCII 码 F6H (/).
字节 9	磁头定位时间, 当磁头移到一条新磁道时, 使其变为稳定所需等待的时间量, DOS2.1 以上的版本将这个值设置 0FH 毫秒.
字节 0AH	马达启动时间, 磁盘马达为了使软盘旋转达到适当的速度所花费的时间量, 这个值以 1/8 秒的时间单位计算, DOS2.1 设置这个值为 2 (1/4 秒).

参见:

中断 13H, 功能调用 05H——格化磁道.

BIOS Int 1FH

图形字符扩展码

中断:

1FH 图形字符扩展码

计算机:

PC, PCjr, XT, Portable 及 AT

说明:

当运行在中分辨率 ( $320 \times 200$ ) 或高分辨率 ( $640 \times 200$ ) 图形方式时, 字符在屏幕上的显示是通过 BIOS 将存放在存储器中字符的点阵字模传送到屏幕的方式实现. 第一组 128 个 ASCII 字符的点阵字模库是存储在 BIOS ROM 中的, 为了显示上档的 128 个字符, 程序必须在 RAM 中建立点阵字模库, 并且将这个中断的中断向量指向那个库.

当建立字符表时, 要以字符代码顺序指定点阵字模库, 其开始顺序于字符 128 (十进制). 每一个字符由  $8 \times 8$  点阵组成. 这样, 每个字符需要 8 个字节的信息, 总表大小为 1K 字节.

对某个字符的 8 个字节信息中每一个字节表示点阵的一行, 第一个字节是字符点阵图形的第一行. 当在屏幕上显示字符时, 就要把设置为 1 的各个位显示出相应的各个点.

当计算机开始运行时, BIOS 将这个中断的向量表入口初始化为 0: 0, 表示 RAM 中没有字符库存储. 因而, 如果当工作在图形方式需要显示当前的 128 个 ASCII 字符时, 就必须建立字符表, 并将它的入口地址放入中断向量表中.

参见:

中断 10H, 功能调用 08H——读取字符及属性

中断 10H, 功能调用 09H——写字符及属性

中断 10H, 功能调用 0AH——写字符

中断 10H, 功能调用 0EH——TTY 方式写字符

BIOS Int 41H 表

### 硬盘参数表

中断:

41H 和 46H 表

计算机:

XT 和 AT

说明:

这两个中断向量指向计算机如何存取硬盘驱动器的表。中断向量 41H 指向说明第一个硬盘（硬盘 0）的表，而中断向量 46H 则指向说明硬盘 1 的表。BIOS 最初将这些向量设为指向 BIOS 内部表中的一个表。可以将中断向量改为指向 BIOS 中的其它硬盘参数表，或者使其指向用户在 RAM 中建立的表。硬盘参数表中的字段说明如下：

（在这里字节 00H 是表的第一个字节）。

字节	说明								
00H-01H	字,指定驱动器中最大的柱面数.								
02H	字节,指定驱动器中最大的磁头数.								
03H-04H	字,指定驱动器开始降低写电流的柱面号 (仅适用于 XT 机).								
05H-06H	字,指定驱动器开始执行写预补偿 (write precompensation)的柱面号.								
07H	字节,指定从 LBA 到 C 数据簇长度.								
08H	控制字节,具有下列格式: 位 <table border="1"><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr></table> 位 7-6      如果这两位都为 0 时,允许读写簇. 置这两位都为 1,则不允许读写. 5-4      保留,设为 0. 3          如果磁盘上磁头多于 8 个,则应设置此位为 1. (仅用于 AT 机). 2-0      驱动器选项(仅用于 XT 机).	7	6	5	4	3	2	1	0
7	6	5	4	3	2	1	0		

字节	说明
09H	字节,指定标准的超量值 (仅用于 XT 机).
0AH	字节,格式化驱动器时的超时值 (仅用于 XT 机).
0BH	字节,检查驱动器时的超时值 (仅用于 AT 机).
0CH-0DH	字,磁头降落区所用的柱面数 (仅用于 AT 机).
0EH	字节,每条磁道的扇区数 (仅用于 AT 机)
0FH	保留字节,应置为 0.

参见:

中断 13H, 功能调用 05H——格式化磁道.

中断 13H, 功能调用 09H——初始化双驱动器.

## 附录 编程服务参照表

服务	中断或功能调用
磁盘服务部分	
确定设备	BIOS 中断 11H
初始化	
复位磁盘	DOS 功能调用 0DH
设备传送地址	DOS 功能调用 1AH
复位磁盘	BIOS 中断 13H, 功能调用 00H
备用磁盘复位	BIOS 中断 13H, 功能调用 0DH
初始化双驱动器	BIOS 中断 13H, 功能调用 09H
检测驱动器准备	BIOS 中断 13H, 功能调用 10H
复校驱动器	BIOS 中断 13H, 功能调用 11H
控制器内部诊断	BIOS 中断 13H, 功能调用 14H
磁盘及文件状态	
获取当前盘号	DOS 功能调用 19H
选择磁盘驱动器	DOS 功能调用 0EH
获取当前驱动器的 FAT 信息	DOS 功能调用 1BH
获取指定驱动器的 FAT 信息	DOS 功能调用 1CH
获取文件尺寸	DOS 功能调用 23H
获取盘传送地址	DOS 功能调用 2FH
获取磁盘未用空间	DOS 功能调用 36H
CHMOD-取或置文件属性	DOS 功能调用 43H
获取或设置文件的日期或时间	DOS 功能调用 57H
获取磁盘状态	BIOS 中断 13H, 功能调用 01H
获取当前驱动器参数	BIOS 中断 13H, 功能调用 08H
获取磁盘类型	BIOS 中断 13H, 功能调用 15H
改变磁盘状态	BIOS 中断 13H, 功能调用 16H
置磁盘类型	BIOS 中断 13H, 功能调用 17H
文件名	
分析文件名	DOS 功能调用 29H
搜索第一个匹配文件	DOS 功能调用 11H
FIND FIRST 搜索	
第一个匹配文件	DOS 功能调用 4EH

服务	中断或功能调用
搜索下一个匹配文件	DOS 功能调用 12H
FINDNEXT 搜索下一个匹配文件	DOS 功能调用 4FH
创建和删除文件	
创建文件	DOS 功能调用 16H
CREAT—创建文件	DOS 功能调用 3CH
创建—唯一文件	DOS 功能调用 5AH
创建—新文件	DOS 功能调用 5BH
删除匹配文件	DOS 功能调用 13H
UNLINK—删除文件	DOS 功能调用 41H
更名	
文件重命名	DOS 功能调用 17H
改文件名	DOS 功能调用 56H
打开和关闭	
打开文件	DOS 功能调用 0FH
打开文件	DOS 功能调用 3DH
复制文件句柄(DUP)	DOS 功能调用 45H
强行复制文件句柄	DOS 功能调用 46H
关闭文件	DOS 功能调用 10H
关闭文件描述字(文件句柄)	DOS 功能调用 3EH
读	
读下一顺序文件记录	DOS 功能调用 14H
读一随机记录	DOS 功能调用 21H
读多个随机记录	DOS 功能调用 27H
读文件或设备	DOS 功能调用 3FH
绝对读盘	DOS 中断 25H
读扇区	BIOS 中断 13H, 功能调用 02H
读长扇区	BIOS 中断 13H, 功能调用 0AH
写	
写下一顺序文件记录	DOS 功能调用 15H
写一随机记录	DOS 功能调用 22H

服务	中断或功能调用
写多个随机记录	DOS 功能调用 28H
写入文件或设备	DOS 功能调用 40H
绝对写盘	DOS 中断 26H
写扇区	BIOS 中断 13H, 功能调用 03H
写长扇区	BIOS 中断 13H, 功能调用 0BH
检索	
设随机记录字段	DOS 功能调用 24H
LSEEK-移动读 / 写指针	
查找柱面(磁道)	BIOS 中断 13H, 功能调用 0CH
目录	
MKDIR-创建子目录	DOS 功能调用 39H
RMDIR-删除目录	DOS 功能调用 3AH
CHDIR-改变当前目录	DOS 功能调用 3BH
取当前目录	DOS 功能调用 47H
校验	
设置或关闭校验开关	DOS 功能调用 2EH
取校验开关状态	DOS 功能调用 54H
检测扇区	BIOS 中断 13H, 功能调用 04H
其它	
设备的 I/O 控制	DOS 功能调用 44H
格式化磁道	BIOS 中断 13H, 功能调用 05H
软盘参数表	BIOS 中断 1EH
硬盘参数表	BIOS 中断 41H 和 46H
程序和存储器控制	
部分创建程序	
创建新程序段	DOS 功能调用 26H
EXEC-装入或执行程序	DOS 功能调用 4BH
终止程序	
终止程序	DOS 中断 20H
终止程序	DOS 功能调用 00H
终止但驻留内存	DOS 中断 27H

服务	中断或功能调用
KEEP-终止进程并	
保持驻留	DOS 功能调用 31H
EXIT-终止进程	DOS 功能调用 4CH
终止地址	DOS 中断 22H
WAIT-取子进程的	
返回代码	DOS 功能调用 4DH
Ctrl-Break 和临界错误	
处理程序	
Ctrl-Break 地址	DOS 中断 23H
取或置 Ctrl-Break	DOS 功能调用 33H
严重错误处理地址	DOS 中断 24H
存储器分配	
分配内存	DOS 功能调用 48H
释放已分配的内存	DOS 功能调用 49H
SET BLOCK-修改分配的	
内存块	DOS 功能调用 4AH
取或置内存分配对策	DOS 功能调用 58H
成块传送	BIOS 中断 15H, 功能调用 87H
取扩展内存容量	BIOS 中断 15H, 功能调用 88H
确定内存容量	BIOS 中断 12H
其它	
取程序段前缀地址	DOS 功能调用 32H
事件等待	BIOS 中断 15H, 功能调用 85H
等待	BIOS 中断 15H, 功能调用 86H
转移到 PVAM	BIOS 中断 15H, 功能调用 85H
设备忙循环(等待)	BIOS 中断 15H, 功能调用 86H
建立中断结束标志	
(POST)	BIOS 中断 15H, 功能调用 91H
键盘 I/O 部分	
读键盘字符并回显	DOS 功能调用 40H
执行直接控制台 I/O	DOS 功能调用 41H

服务	中断或功能调用
执行直接控制台输入	
但不回显	DOS 功能调用 07H
读键盘字符但不回显	DOS 功能调用 08H
键盘字符存入缓冲区	DOS 功能调用 0AH
检查输入设备状态	DOS 功能调用 0BH
清除输入缓冲区并调用	
I/O 操作	DOS 功能调用 0CH
读下一个键盘字符	BIOS 中断 16H, 功能调用 00H
检测字符是否准备就绪	BIOS 中断 16H, 功能调用 01H
取当前转移键状态	BIOS 中断 16H, 功能调用 02H
SYS Req 键支持	BIOS 中断 16H, 功能调用 85H
键盘终止地址	BIOS 中断 1BH
视频 I/O 部分	
确定设备	BIOS 中断 11H
读屏幕	
屏幕拷贝(打印)	BIOS 中断 05H
读光标位置	BIOS 中断 10H, 功能调用 03H
读光笔位置	BIOS 中断 10H, 功能调用 04H
读字符和属性	BIOS 中断 10H, 功能调用 08H
读点	BIOS 中断 10H, 功能调用 0DH
写屏幕	
读键盘字符并回显	DOS 功能调用 01H
显示字符	DOS 功能调用 02H
执行直接控制台 I/O	DOS 功能调用 06H
执行直接控制台输入	
但不回显	DOS 功能调用 07H
标准输出设备上显示	
字符串	DOS 功能调用 09H
清除输入缓冲区并调用	
I/O 操作	DOS 功能调用 0CH
写字符和属性	BIOS 中断 10H, 功能调用 09H
写字符串	BIOS 中断 10H, 功能调用 0AH

<b>服务</b>	<b>中断或功能调用</b>
写点	BIOS 中断 10H, 功能调用 0CH
以电传方式写字符	BIOS 中断 10H, 功能调用 0EH
写字符串	BIOS 中断 10H, 功能调用 13H
设置显示方式	BIOS 中断 10H, 功能调用 00H
设置光标类型	BIOS 中断 10H, 功能调用 01H
设置光标位置	BIOS 中断 10H, 功能调用 02H
设置当前显示页	BIOS 中断 10H, 功能调用 05H
当前显示页上滚	BIOS 中断 10H, 功能调用 06H
当前显示页下滚	BIOS 中断 10H, 功能调用 07H
置彩色调色板	BIOS 中断 10H, 功能调用 0BH
其它	
取当前显示方式	BIOS 中断 10H, 功能调用 0FH
图形字符扩展码	BIOS 中断 1FH
<b>RS-232 串行口部分</b>	
确定设备	BIOS 中断 11H
初始化串行口	BIOS 中断 14H, 功能调用 00H
从辅助设备读入一字符	DOS 功能调用 03H
接收一字符	BIOS 中断 14H, 功能调用 02H
向辅助设备发送一字符	DOS 功能调用 04H
发送一字符	BIOS 中断 14H, 功能调用 01H
获取串行口状态	BIOS 中断 14H, 功能调用 03H
<b>并行打印机 I/O 部分</b>	
确定设备	BIOS 中断 11H
初始化打印机口	BIOS 中断 17H, 功能调用 01H
向标准打印设备	
发送一字符	DOS 功能调用 05H
打印一字符	BIOS 中断 17H, 功能调用 00H
屏幕拷贝(打印)	BIOS 中断 05H
取打印机状态	BIOS 中断 17H, 功能调用 02H
<b>中断向量部分</b>	
设置中断向量	DOS 功能调用 25H
取中断向量地址	DOS 功能调用 35H

服务	中断或功能调用
日期和时间功能部分	
读日期和时间	
读日期	DOS 功能调用 2AH
读电池供电时钟日期	BIOS 中断 1AH, 功能调用 04H
获取时间	DOS 功能调用 2CH
读当前时钟值	BIOS 中断 1AH, 功能调用 00H
读电池供电时钟时间	BIOS 中断 1AH, 功能调用 02H
设置日期和时间	
设置日期	DOS 功能调用 2BH
设置电池供电时钟的日期	BIOS 中断 1AH, 功能调用 05H
设置时间	DOS 功能调用 2DH
设置当前时钟值	BIOS 中断 1AH, 功能调用 01H
设置电池供电时钟时间	BIOS 中断 1AH, 功能调用 03H
设置闹钟	BIOS 中断 1AH, 功能调用 06H
复位闹钟	BIOS 中断 1AH, 功能调用 07H
其它	
定时器信号	BIOS 中断 1CH
网络功能部分	
封锁或开锁文件访问	DOS 功能调用 5CH
建立网络参数	DOS 功能调用 5EH
网络重定向	DOS 功能调用 5H
磁带 I/O 部分	
启动盒式磁带机	BIOS 中断 15H, 功能调用 00H
停止盒式磁带机	BIOS 中断 15H, 功能调用 02H
读数据块	BIOS 中断 15H, 功能调用 02H
写数据块	BIOS 中断 15H, 功能调用 03H
其它服务部分	
获取 DOS 版本号	DOS 功能调用 30H
获取或设置国度信息	DOS 功能调用 38H
获取扩充错误代码	DOS 功能调用 38H
获取扩充错误代码	DOS 功能调用 59H
控制杆支持	BIOS 中断 15H, 功能调用 84H
ROM BASIC 语言	BIOS 中断 18H
引导装入程序(重新启动系统)	BIOS 中断 19H